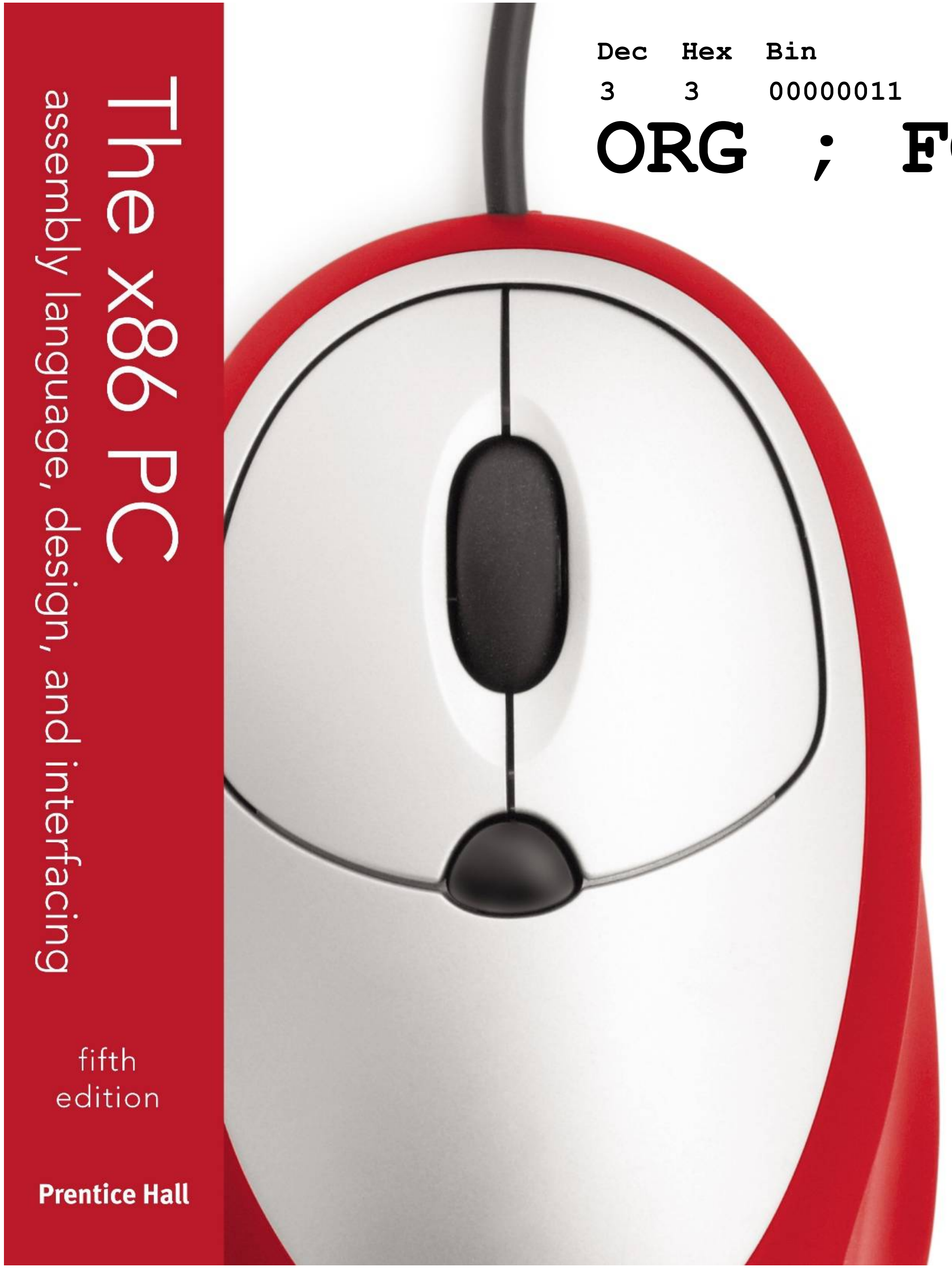
Dec	Hex	Bin
3	3	00000011

ORG ; FOUR

Arithmetic and Logic Instructions And Programs



The x86 PC

assembly language, design, and interfacing

fifth edition

MUHAMMAD ALI MAZIDI
JANICE GILLISPIE MAZIDI
DANNY CAUSEY

OBJECTIVES
this chapter enables the student to:

- Demonstrate how 8-bit and 16-bit unsigned numbers are added in the x86.
- Convert data to any of the forms:
  - ASCII,packed BCD,unpacked BCD.
- Explain the effect of unsigned arithmetic instructions on the flags.
- Code the following Assembly language unsigned arithmetic instructions:
  - Addition instructions: **ADD** and **ADC**.
  - Subtraction instructions **SUB** and **SBB**.
  - Multiplication and division instructions **MUL** and **DIV**.

OBJECTIVES (cont)

this chapter enables the student to:

- Code BCD arithmetic instructions:
  - **DAA** and **DAS**.
- Code the Assembly language logic instructions:
  - **AND**, **OR**, and **XOR**.
  - Logical shift instructions **SHR** and **SHL**.
  - The compare instruction **CMP**.
- Code bitwise rotation instructions
  - **ROR**, **ROL**, **RCR**, and **RCL**.
- Demonstrate an ability to use all of the above instructions in Assembly language programs.
- Perform bitwise manipulation using the C language.

3.0: UNSIGNED ADDITION AND SUBTRACTION

- Unsigned numbers are defined as data in which all the bits are used to represent data.
  - Applies to the ADD and SUB instructions.
  - No bits are set aside for the positive or negative sign.
    - Between 00 and FFH (0 to 255 decimal) for 8-bit data.
    - Between 0000 and FFFFH (0 to 65535 decimal) for 16-bit data.

3.1: UNSIGNED ADDITION AND SUBTRACTION
addition of unsigned numbers

- The form of the ADD instruction is:

```
ADD destination,source ;destination = destination + source
```

- ADD and ADC are used to add two operands.
  - The destination operand can be a register or in memory.
  - The source operand can be a register, in memory, or immediate.
    - Memory-to-memory operations are never allowed in x86 Assembly language.
  - The instruction could change ZF, SF, AF, CF, or PF bits of the flag register.

Arithmetic Instructions – ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	(S) + (D) ➔ (D) Carry ➔ (CF)	All
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) ➔ (D) Carry ➔ (CF)	All
INC	Increment by one	INC D	(D) + 1 ➔ (D)	All but CY

Examples

Ex. 1 ADD AX, 2
ADC AX, 2

Ex. 2 INC BX
INC word ptr [BX]

25
56
\+ ----------
7B → 81

3.1: UNSIGNED ADDITION AND SUBTRACTION
addition of unsigned numbers

Example 3-1

Show how the flag register is affected by

```
MOV   AL,0F5H
ADD   AL,0BH
```

Solution:

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

After the addition, the AL register (destination) contains 00 and the flags are as follows:
CF = 1, since there is a carry out from D7
SF = 0, the status of D7 of the result
PF = 1, the number of 1s is zero (zero is an even number)
AF = 1, there is a carry from D3 to D4
ZF = 1, the result of the action is zero (for the 8 bits)

3.1: UNSIGNED ADDITION AND SUBTRACTION
CASE1 addition of individual byte/word data

- Program 3-1a uses AH to accumulate carries as the operands are added to AL.

Write a program to calculate the total sum of 5 bytes of data. Each byte represents the daily wages of a worker. This person does not make more than $255 (FFH) a day. The decimal data is as follows: 125, 235, 197, 91, and 48.

```
TITLE       PROG3-1A (EXE) ADDING 5 BYTES
PAGE        60,132
.MODEL SMALL
.STACK 64
;---------------------------------
            .DATA
COUNT       EQU    05
DATA        DB           125,235,197,91,48
            ORG    0008H
SUM         DW     ?
;--------------------------------
     .CODE
MAIN PROC   FAR
```

See the entire program listing on page 93 of your textbook.

3.1: UNSIGNED ADDITION AND SUBTRACTION
CASE1 addition of individual byte/word data

- Numbers are converted to hex by the assembler:
  - **125=7DH 235=0EBH 197=0C5H 91=5BH 48=30H**
- Three iterations of the loop are shown below.
  - In the first, 7DH is added to AL.
    - CF = 0 and AH = 00.
    - CX = 04 and ZF = 0.
  - Second, EBH is added to AL & since a carry occurred, AH is incremented
    - AL = 68H and CF = 1.
    - CX = 03 and ZF = 0.
  - Third, C5H is added to AL, again a carry increments AH.
    - AL = 2DH, CX = 02 and ZF = 0.

- This process continues until CX = 00 and the zero flag becomes 1, causing JNZ to fall through.
  - The result will be saved in the word-sized memory set aside in the data segment.

3.1: UNSIGNED ADDITION AND SUBTRACTION
CASE1 addition of individual byte/word data

- Due to pipelining it is strongly recommended that the following lines of the program be replaced:

Replace these lines	With these lines
`BACK: ADD AL,[SI]`	`BACK: ADD AL,[SI]`
`JNC OVER`	`ADC AH,00 ;add 1 to AH if CF=1`
`INC AH`	`INC SI`
`OVER: INC SI`	

  - The "**ADC AH,00**" instruction in reality means add **00+AH+CF** and place the result in **AH**.
    - More efficient since the instruction "**JNC OVER**" has to empty the queue of pipelined instructions and fetch the instructions from the OVER target every time the carry is zero (CF = 0).
    - Program 3-1b is the same as 3-1a, rewritten for word addition. (See the program listing on page 94 of your textbook.)

3.1: UNSIGNED ADDITION AND SUBTRACTION

CASE2 addition of multiword numbers

- Assume a program to total U.S. budget for the last 100 years or mass of planets in the solar system.
  - Numbers being added could be 8 bytes wide or more.
- The programmer must write the code to break the large numbers into smaller chunks to be processed.
  - A 16-bit register & an 8 byte operand is wide would take a total of four iterations.
  - An 8-bit register with the same operands would require eight iterations.

3.1: UNSIGNED ADDITION AND SUBTRACTION

CASE2 addition of multiword numbers

- In writing program 3-2, the first decision was the directive for coding the data in the data segment.

```
TITLE       PROG3-2 (EXE) MULTIWORD ADDITION
PAGE        60,132
.MODEL SMALL
.STACK 64
;------------------------------------
      .DATA
DATA1 DQ    548FB9963CE7H
      ORG   0010H
DATA2 DQ    3FCD4FA23B8DH
      ORG   0020H
DATA3 DQ    ?
;------------------------------------
     .CODE
MAIN PROC   FAR
     MOV    AX,@DATA
     MOV    DS,AX
     CLC                          ;clear carry before first addition
     MOV    SI,OFFSET DATA1       ;     for operand1
```

DQ was chosen since it can represent data as large as 8 bytes wide.

See the entire program listing on page 95 of your textbook.

3.1: UNSIGNED ADDITION AND SUBTRACTION

CASE2 addition of multiword numbers

- In addition of multibyte (or multiword) numbers, the ADC instruction is always used, as the carry must be added to the next-higher byte (or word) in the next iteration.
  - Before executing ADC, the carry flag is cleared (CF = 0) using the CLC (clear carry) instruction.
- Three pointers have been used:
  - SI for DATA1; DI for DATA2.
  - BX for DATA3. (where the result is saved)

3.1: UNSIGNED ADDITION AND SUBTRACTION

CASE2 addition of multiword numbers

- A new instruction, "**LOOP XXXX**", replaces the often used "**DEC CX**" and "**JNZ XXXX**".

```
LOOP  xxxx  ;is equivalent to the following two instructions

DEC   CX
JNZ   xxxx
```

  - When "**LOOP xxxx**" is executed, CX decrements automatically, and if CX is not 0, the processor will jump to target address **xxxx**.
    - If CX is 0, the next instruction (below "**LOOP xxxx**") is executed.

3.1: UNSIGNED ADDITION AND SUBTRACTION
subtraction of unsigned numbers

- In subtraction, x86 processors use 2's complement.
  - Internal adder circuitry performs the subtraction command.
- x86 steps in executing the SUB instruction:
  - 1. Take the 2's complement of the subtrahend. (source operand)
  - 2. Add it to the minuend. (destination operand)
  - 3. Invert the carry.
    - The steps are performed for every SUB instruction regardless of source & destination of the operands.

```
SUB  dest,source;dest = dest - source
```

3.1: UNSIGNED ADDITION AND SUBTRACTION
subtraction of unsigned numbers

- After the execution, if CF = 0, the result is positive.
  - If CF = 1, the result is negative and the destination has the 2's complement of the result.

Example 3-2

Show the steps involved in the following:

```
        MOV   AL,3FH       ;load AL=3FH
        MOV   BH,23H       ;load BH=23H
        SUB   AL,BH        ;subtract BH from AL. Place result in AL.
```

Solution:

```
 AL    3F          0011 1111          0011 1111
-BH   -23        - 0010 0011        + 1101 1101 (2's complement)
       1C                          1 0001 1100 CF=0 (step 3)
```

The flags would be set as follows: CF = 0, ZF = 0, AF = 0, PF = 0, and SF = 0. The programmer must look at the carry flag (not the sign flag) to determine if the result is positive or negative.

3.1: UNSIGNED ADDITION AND SUBTRACTION
subtraction of unsigned numbers

- NOT performs the 1's complement of the operand.
  - The operand is incremented to get the 2's complement.

Example 3-3

Analyze the following program:

```
;from the data segment:
DATA1       DB     4CH
DATA2       DB     6EH
DATA3       DB     ?
;from the code segment:
            MOV    DH,DATA1     ;load DH with DATA1 value (4CH)
            SUB    DH,DATA2     ;subtract DATA2 (6E) from DH (4CH)
            JNC    NEXT         ;if CF=0 jump to NEXT target
            NOT    DH           ;if CF=1 then take 1's complement
            INC    DH           ;and increment to get 2's complement
NEXT:       MOV    DATA3,DH     ;save DH in DATA3
```

Solution:

Following the three steps for "SUB DH,DATA2":

```
     4C    0100 1100         0100 1100
    -6E    0110 1110      +  1001 0010  (2's complement)
    -22                     01101 1110 CF=1 (step 3)result is negative
```

3.1: UNSIGNED ADDITION AND SUBTRACTION
SBB subtract with borrow

- SBB is used for multibyte (multiword) numbers.
  - It will take care of the borrow of the lower operand.
    - If the carry flag is 0, SBB works like SUB.
    - If the carry flag is 1, SBB subtracts 1 from the result.
- The PTR (pointer) data specifier directive is widely used to specify the size of the operand when it differs from the defined size.

3.1: UNSIGNED ADDITION AND SUBTRACTION
SBB - subtract with borrow

- "**WORD PTR**" tells the assembler to use a word operand, though the data is defined as a doubleword.

Example 3-4

Analyze the following program:

```
DATA_A      DD    62562FAH
DATA_B      DD    412963BH
RESULT      DD    ?
...         ...   ...
            MOV   AX,WORD PTR DATA_A      ;AX=62FA
            SUB   AX,WORD PTR DATA_B      ;SUB 963B from AX
            MOV   WORD PTR RESULT,AX      ;save the result
            MOV   AX,WORD PTR DATA_A +2   ;AX=0625
            SBB   AX,WORD PTR DATA_B +2   ;SUB 0412 with borrow
            MOV   WORD PTR RESULT+2,AX    ;save the result
```

Solution:

After the SUB, AX = 62FA – 963B = CCBF and the carry flag is set. Since CF = 1, when SBB is executed, AX = 625 – 412 – 1 = 212. Therefore, the value stored in RESULT is 0212CCBF.

3.2: UNSIGNED MULTIPLICATION & DIVISION
multiplication of unsigned numbers

- In multiplying two numbers in the x86 processor, use of registers AX, AL, AH, and DX is necessary.
  - The function assumes the use of those registers.
- Three multiplication cases:
  - byte times byte; word times word; byte times word.

Table 3-1: Unsigned Multiplication Summary

Multiplication	Operand 1	Operand 2	Result
byte × byte	AL	register or memory	AX
word × word	AX	register or memory	DX AX
word × byte	AL = byte, AH = 0	register or memory	DX AX

3.2: UNSIGNED MULTIPLICATION & DIVISION
multiplication of unsigned numbers

- **byte × byte** - one of the operands must be in the AL register and the second can be in a register or in memory.
  - After the multiplication, the result is in AX.

```
RESULT  DW    ?           ;result is defined in the data segment
        ...
        MOV   AL,25H      ;a byte is moved to AL
        MOV   BL,65H      ;immediate data must be in a register
        MUL   BL          ;AL = 25 x 65H
        MOV   RESULT,AX   ;the result is saved
```

  - 25H is multiplied by 65H and the result is saved in word-sized memory named RESULT.
    - Register addressing mode was used.
    - Examples of other address modes appear on textbook page 98.

3.2: UNSIGNED MULTIPLICATION & DIVISION
multiplication of unsigned numbers

- **word × word** - one operand must be in AX & the second operand can be in a register or memory.
  - After multiplication, AX & DX will contain the result.
    - Since word-by-word multiplication can produce a 32-bit result, AX will hold the lower word and DX the higher word.

```
DATA3       DW    2378H
DATA4       DW    2F79H
RESULT1     DW    2 DUP(?)
...         ....
            MOV   AX,DATA3       ;load first operand into AX
            MUL   DATA4          ;multiply it by the second operand
            MOV   RESULT1,AX     ;store the lower word result
            MOV   RESULT1+2,DX   ;store the higher word result
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
multiplication of unsigned numbers

- **word × byte** - similar to word-by-word multiplication except that AL contains the byte operand and AH must be set to zero.

```
;from the data segment:
DATA5         DB     6BH
DATA6         DW     12C3H
RESULT3       DW     2 DUP(?)
;from the code segment:
              MOV    AL,DATA5      ;AL holds byte operand
              SUB    AH,AH         ;AH must be cleared
              MUL    DATA6         ;byte in AL mult.by word operand
              MOV    BX,OFFSET RESULT3  ;BX points to product
              MOV    [BX],AX            ;AX holds lower word
              MOV    [BX]+2,DX          ;DX holds higher word
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
division of unsigned numbers

- Like multiplication, division of two numbers in the x86 uses of registers AX, AL, AH, and DX.
- Four division cases:
  - byte over byte; word over word.
  - word over byte; doubleword over word.
- In divide, in cases where the CPU cannot perform the division, an interrupt is activated.
  - Referred to as an *exception,* and the PC will display a **`Divide Error`** message.
    - If the denominator is zero. (dividing any number by 00)
    - If the quotient is too large for the assigned register.

division of unsigned numbers

- **byte/byte** - the numerator must be in the AL register and AH must be set to zero.
  - The denominator cannot be immediate but can be in a register or memory, supported by the addressing modes.
    - After the DIV instruction is performed, the quotient is in AL and the remainder is in AH.

Table 3-2: Unsigned Division Summary

Division	Numerator	Denominator	Quotient	Rem.
byte/byte	AL = byte, AH = 0	register or memory	AL[1]	AH
word/word	AX = word, DX = 0	register or memory	AX[2]	DX
word/byte	AX = word	register or memory	AL[1]	AH
doubleword/word	DXAX = doubleword	register or memory	AX[2]	DX

Notes: 1. Divide error interrupt if AL > FFH. 2. Divide error interrupt if AX > FFFFH.

3.2: UNSIGNED MULTIPLICATION & DIVISION division of unsigned numbers

- Various addressing modes of the denominator.

```
QOUT1         DB    ?
REMAIN1       DB    ?
;using immediate addressing mode will give an error
              MOV   AL,DATA7          ;move data into AL
              SUB   AH,AH             ;clear AH
              DIV   10                ;immed. mode not allowed!!
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
division of unsigned numbers

- Various addressing modes of the denominator.

```
;allowable modes include:
;using direct mode
            MOV   AL,DATA7        ;AL holds numerator
            SUB   AH,AH           ;AH must be cleared
            DIV   DATA8           ;divide AX by DATA8
            MOV   QOUT1,AL        ;quotient = AL = 09
            MOV   REMAIN1,AH      ;remainder = AH = 05
;using register addressing mode
            MOV   AL,DATA7            ;AL holds numerator
            SUB   AH,AH               ;AH must be cleared
            MOV   BH,DATA8            ;move denom. to register
            DIV   BH                  ;divide AX by BH
            MOV   QOUT1,AL            ;quotient = AL = 09
            MOV   REMAIN1,AH          ;remainder = AH = 05
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
division of unsigned numbers

- Various addressing modes of the denominator.

```
;allowable modes include:
;using register indirect addressing mode
         MOV  AL,DATA7            ;AL holds numerator
         SUB  AH,AH               ;AH must be cleared
         MOV  BX,OFFSET DATA8     ;BX holds offset of DATA8
         DIV  BYTE PTR [ BX]      ;divide AX by DATA8
         MOV  QOUT2,AX
         MOV  REMAIND2,DX
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
division of unsigned numbers

- **word/word** - the numerator is in AX, and DX must be cleared.
  - The denominator can be in a register or memory.
    - After DIV, AX will have the quotient.
    - The remainder will be in DX.

```
MOV   AX,10050      ;AX holds numerator
SUB   DX,DX         ;DX must be cleared
MOV   BX,100        ;BX used for denominator
DIV   BX
MOV   QOUT2,AX      ;quotient = AX = 64H = 100
MOV   REMAIND2,DX   ;remainder = DX = 32H = 50
```

3.2: UNSIGNED MULTIPLICATION & DIVISION
division of unsigned numbers

- **word/byte** - the numerator is in AX & the denominator can be in a register or memory.
  - After DIV, AL will contain the quotient, AH the remainder.
    - The maximum quotient is FFH.
- This program divides AX = 2055 by CL = 100.
  - The quotient is AL = 14H (20 decimal)
  - The remainder is AH = 37H (55 decimal).

```
MOV   AX,2055         ;AX holds numerator
MOV   CL,100          ;CL used for denominator
DIV   CL
MOV   QUO,AL          ;AL holds quotient
MOV   REMI,AH         ;AH holds remainder
```

division of unsigned numbers

- **doubleword/word** - the numerator is in AX and DX.
  - The most significant word in DX, least significant in AX.
    - The denominator can be in a register or in memory.
  - After DIV, the quotient will be in AX, the remainder in DX.
    - The maximum quotient FFFFH.

```
;from the data segment:
DATA1       DD    105432
DATA2       DW    10000
QUOT        DW    ?
REMAIN      DW    ?
;from the code segment:
            MOV   AX,WORD PTR DATA1     ;AX holds lower word
            MOV   DX,WORD PTR DATA1+2;DX higher word of
                                      numerator
            DIV   DATA2
            MOV   QUOT,AX               ;AX holds quotient
            MOV   REMAIN,DX             ;DX holds remainder
```

Example

- Write a program that calculates the average of five temperatures and writes the result in AX

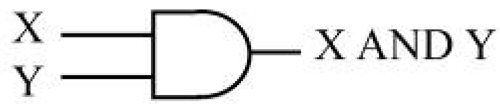
```
DATA      DB    +13,-10,+19,+14,-18          ;0d,f6,13,0e,ee
          MOV       CX,5                              ;LOAD COUNTER
          SUB       BX, BX                   ;CLEAR BX, USED AS ACCUMULATOR
          MOV       SI, OFFSET DATA          ;SET UP POINTER
BACK:     MOV       AL,[SI]                           ;MOVE BYTE INTO AL
          CBW                                         ;SIGN EXTEND INTO AX
          ADD       BX, AX                            ;ADD TO BX
          INC       SI                                ;INCREMENT POINTER
          DEC       CX                                ;DECREMENT COUNTER
          JNZ       BACK
          mov ax,bx                          ;LOOP IF NOT FINISHED
          MOV       CL,5                              ;MOVE COUNT TO AL
          DIV       CL                                ;FIND THE AVERAGE
```

3.3: LOGIC INSTRUCTIONS
AND

- **AND destination, source**
  - This instruction will perform a logical AND on the operands and place the result in the destination.
    - Destination operand can be a register or in memory.
    - Source operand can be a register, memory, or immediate.

Logical AND Function

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



- AND will automatically change the CF & OF to zero.
  - PF, ZF, and SF are set according to the result.
    - The rest of the flags are either undecided or unaffected.

3.3: LOGIC INSTRUCTIONS

AND

- AND can mask certain bits of the operand, and also to test for a zero operand:

```
        AND   DH,DH
        JZ    XXXX
        ...
XXXX:   ...
```

This code will AND DH with itself and set ZF = 1 if the result is zero.

Example 3-5

Show the results of the following:

```
        MOV   BL,35H
        AND   BL,0FH         ;AND BL with 0FH. Place the result in BL.
```

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
```

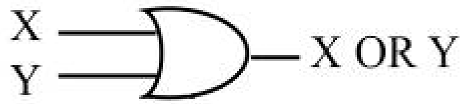
Flag settings will be: SF = 0, ZF = 0, PF = 1, CF = OF = 0.

3.3: LOGIC INSTRUCTIONS
OR

- `OR destination, source`
  - Destination/source operands are Ored, result placed in the destination.
    - Can set certain bits of an operand to 1.
    - Destination operand can be a register or in memory.
    - Source operand can be a register, in memory, or immediate.

Logical OR Function

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



- Flags are set the same as for the AND instruction.
  - CF & OF will be reset to zero.
    - SF, ZF, and PF will be set according to the result.
    - All other flags are not affected.

3.3: LOGIC INSTRUCTIONS
OR

- The OR instruction can also be used to test for a zero operand.
  - "**OR BL,0**" will OR the register BL with 0 and make ZF = 1 if BL is zero.
  - "**OR BL,BL**" will achieve the same result.

Example 3-6

Show the results of the following:

```
MOV  AX,0504          ;AX = 0504
OR   AX,0DA68H        ;AX = DF6C
```

Solution:

```
0504H   0000 0101 0000 0100
DA68H   1101 1010 0110 1000
DF6C    1101 1111 0110 1100
```

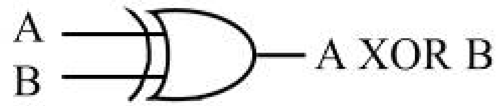
Flags will be: SF = 1 , ZF = 0, PF = 1, CF = OF = 0.
Notice that parity is checked for the lower 8 bits only.

3.3: LOGIC INSTRUCTIONS
XOR

- **`XOR dest, src`**
  - XOR will eXclusive-OR operands and place result in the destination.
    - Sets the result bits to 1 if they are not equal, otherwise, reset to 0.
    - Flags are set the same as for AND.
    - Operand rules are the same as in the AND and OR instructions.

Logical XOR Function

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



3.3: LOGIC INSTRUCTIONS
XOR

- XOR can be used to see if two registers have the same value.
  - "**XOR BX,CX**" will make ZF = 1 if both registers have the same value, and if they do, the result (0000) is saved in BX, the destination.
- A widely used application of XOR is to toggle bits of an operand.

```
XOR   AL,04H         ;XOR  AL with 0000 0100
```

  - Toggling bit 2 of register AL would cause it to change to the opposite value; all other bits remain unchanged.

3.3: LOGIC INSTRUCTIONS
XOR

Example 3-7

Show the results of the following:

```
MOV     DH,54H
XOR     DH,78H
```

Solution:

```
54H     0 1 0 1 0 1 0 0
78H     0 1 1 1 1 0 0 0
2C      0 0 1 0 1 1 0 0
```

Flag settings will be: SF = 0, ZF = 0, PF = 0, CF = OF = 0.

Example 3-8

The XOR instruction can be used to clear the contents of a register by XORing it with itself. Show how "XOR AH,AH" clears AH, assuming that AH = 45H.

Solution:

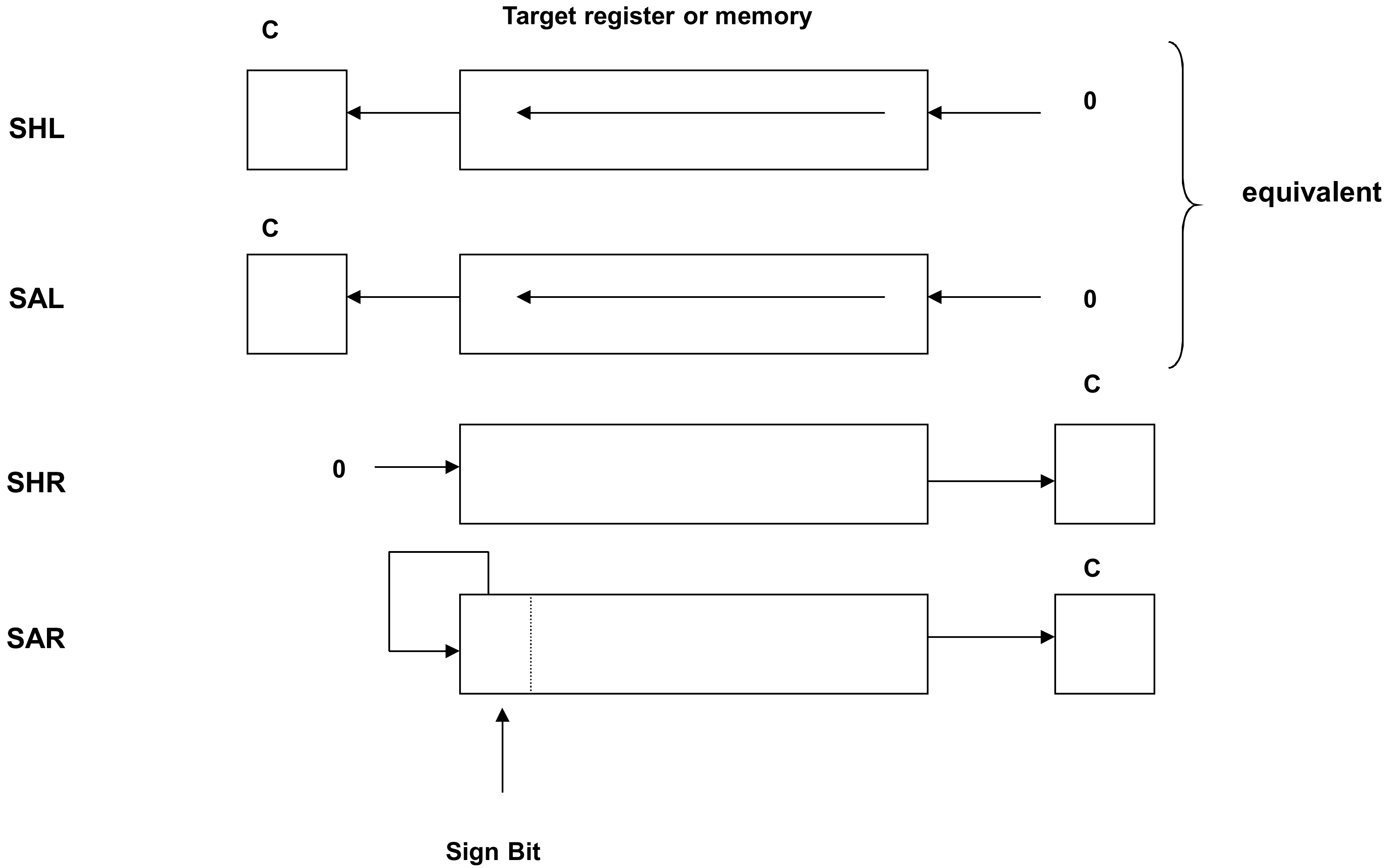
```
45H     01000101
45H     01000101
00      00000000
```

Flag settings will be: SF = 0, ZF = 1, PF =1 , CF = OF = 0.

3.3: LOGIC INSTRUCTIONS
SHIFT

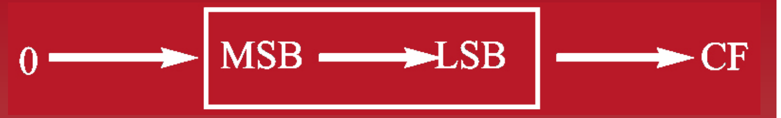
- Shifts the contents of a register or memory location right or left.
  - There are two kinds of shifts:
    - **Logical** - for unsigned operands.
    - **Arithmetic** - for signed operands.
- The number of times (or bits) the operand is shifted can be specified directly if it is *once only*.
  - Through the CL register if it is more than once.

Shift



C
Target register or memory
SHL
0
equivalent
C
SAL
0
C
SHR
0
C
SAR
Sign Bit

3.3: LOGIC INSTRUCTIONS
SHIFT RIGHT



- SHR - logical shift right.
  - Operand is shifted right bit by bit.
    - For every shift the LSB (least significant bit) will go to the carry flag. (CF)
    - The MSB (most significant bit) is filled with 0.

Example 3-9

Show the result of SHR in the following:

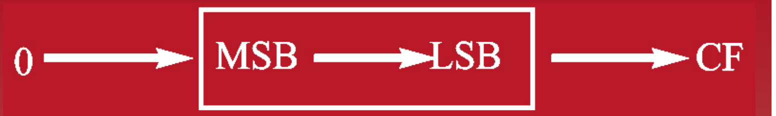
```
MOV   AL,9AH
MOV   CL,3    ;set number of times to shift
SHR   AL,CL
```

Solution:

```
9AH =     10011010
          01001101    CF = 0  (shifted once)
          00100110    CF = 1  (shifted twice)
          00010011    CF = 0  (shifted three times)
```

After shifting right three times, AL = 13H and CF = 0.

3.3: LOGIC INSTRUCTIONS
SHIFT RIGHT

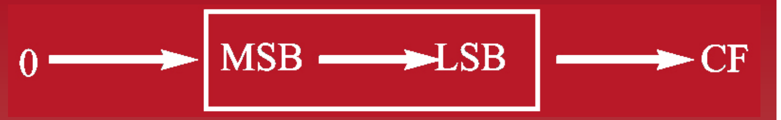


- If the operand is to be shifted once only, this is specified in the SHR instruction itself.

```
MOV    BX,0FFFFH    ;BX=FFFFH
SHR    BX,1         ;shift right BX once only
```

  – After the shift, BX = 7FFFH and CF = 1. SHIFT.

3.3: LOGIC INSTRUCTIONS
SHIFT RIGHT



- The operand to be shifted can be in a register or in memory.
  - Immediate addressing mode is not allowed for SHIFT.
    - "**SHR 25,CL**" will cause the assembler to give an error.

Example 3-10

Show the results of SHR in the following:

```
        ;from the data segment:
        DATA1        DW     7777H
        ;from the code segment:
        TIMES        EQU    4
                     MOV    CL,TIMES     ;CL=04
                     SHR    DATA1,CL     ;shift DATA1 CL times
```

Solution:

After the four shifts, the word at memory location DATA1 will contain 0777. The four LSBs are lost through the carry, one by one, and 0s fill the four MSBs.

3.3: LOGIC INSTRUCTIONS
SHIFT LEFT

- SHL - Logical shift left, the reverse of SHR.
  - After every shift, the LSB is filled with 0.
    - MSB goes to CF.
  - All rules are the same as for SHR.

Example 3-11

Show the effects of SHL in the following:

```
MOV   DH,6
MOV   CL,4
SHL   DH,CL
```

Solution:

```
                 00000110
CF=0             00001100    (shifted left once)
CF=0             00011000
CF=0             00110000
CF=0             01100000    (shifted four times)
```

After the four shifts left, the DH register has 60H and CF = 0.

3-11 can also be coded as:

```
MOV   DH,6
SHL   DH,1
SHL   DH,1
SHL   DH,1
SHL   DH,1
```

Examples

Examples

```
SHL AX,1
SAL DATA1, CL   ; shift count is a modulo-32 count
```

Ex.

```
; Multiply AX by 10
        SHL AX, 1
        MOV BX, AX
      MOV CL,2
        SHL AX,CL
        ADD AX, BX
```

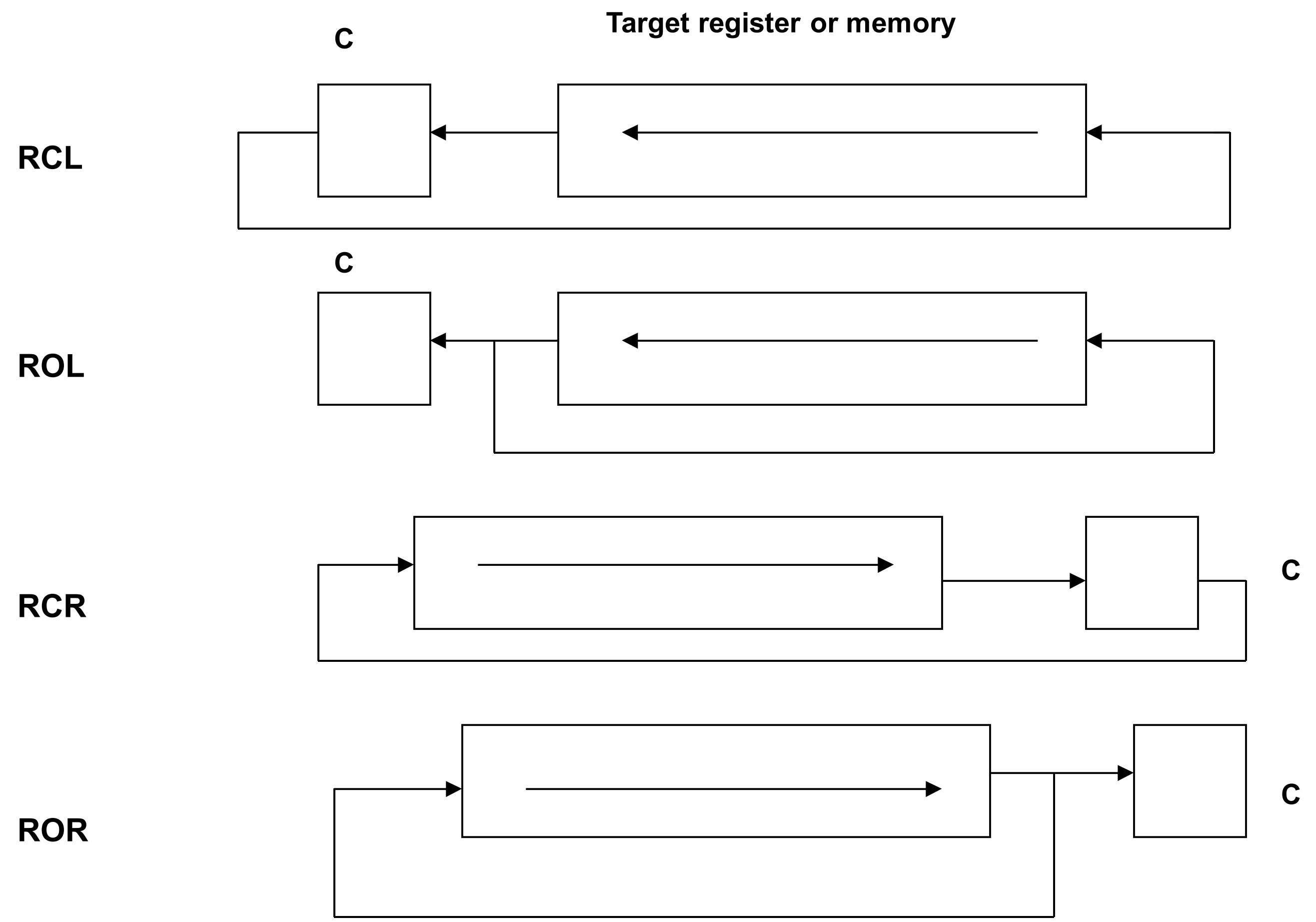
Ex. What are the results of SAR CL, 1 if CL initially contains B6H?

Ex. What are the results of SHL AL, CL if AL contains 75H and CL contains 3?

3.5: ROTATE INSTRUCTIONS

- ROR, ROL and RCR, RCL are designed specifically to perform a bitwise rotation of an operand.
  – They allow a program to rotate an operand right or left.
- Similar to shift instructions, if the number of times an operand is to be rotated is more than 1, this is indicated by CL.
  – The operand can be in a register or memory.
- There are two types of rotations.
  – Simple rotation of the bits of the operand
  – Rotation through the carry.

Rotate



What is the result of ROL byte ptr [SI], 1 if this memory location 3C020 contains 41H?

Ex.

What is the result of ROL word ptr [SI], 8 if this memory location 3C020 contains 4125H?

3.5: ROTATE INSTRUCTIONS
ROR/ROL rotate right/rotate left

- In ROR (Rotate Right), as bits are shifted from left to right, they exit from the right end (LSB) and enter the left end (MSB).
  - As each bit exits LSB, a copy is given to the carry flag.
    - In ROR the LSB is moved to the MSB, & copied to CF.
- In ROL (Rotate Left), as bits are shifted from right to left, they exit the left end (MSB) and enter the right end (LSB).
  - Every bit that leaves the MSB is copied to the carry flag.
    - In ROL the MSB is moved to the LSB and is also copied to CF

Programs 3-7 & 3-8 on page 120 show applications of rotation instructions

3.5: ROTATE INSTRUCTIONS
ROR rotate right


```
        MOV     AL,36H          ;AL=0011 0110
        ROR     AL,1            ;AL=0001 1011   CF=0
        ROR     AL,1            ;AL=1000 1101   CF=1
        ROR     AL,1            ;AL=1100 0110   CF=1
;or:
        MOV     AL,36H          ;AL=0011 0110
        MOV     CL,3            ;CL=3 number of times to rotate
        ROR     AL,CL           ;AL=1100 0110 CF=1
;the operand can be a word:
        MOV     BX,0C7E5H       ;BX=1100 0111 1110 0101
        MOV     CL,6            ;CL=6 number of times to rotate
        ROR     BX,CL           ;BX=1001 0111 0001 1111 CF=1
```

- If the operand is to be rotated once, the 1 is coded.
  - If it is to be rotated more than once, register CL is used to hold the number of times it is to be rotated.

3.5: ROTATE INSTRUCTIONS
ROL rotate left


```
        MOV    BH,72H         ;BH=0111 0010
        ROL    BH,1           ;BH=1110 0100  CF=0
        ROL    BH,1           ;BH=1100 1001  CF=1
        ROL    BH,1           ;BH=1001 0011  CF=1
        ROL    BH,1           ;BH=0010 0111  CF=1
;or:
        MOV    BH,72H         ;BH=0111 0010
        MOV    CL,4           ;CL=4 number of times to rotate
        ROL    BH,CL          ;BH=0010 0111  CF=1

;The operand can be a word:
        MOV    DX,672AH       ;DX=0110 0111 0010 1010
        MOV    CL,3           ;CL=3 number of times to rotate
        ROL    DX,CL ;DX=0011 1001 0101 0011 CF=1
```

– If the operand is to be rotated once, the 1 is coded.
  - If it is to be rotated more than once, register CL is used to hold the number of times it is to be rotated.

3.5: ROTATE INSTRUCTIONS
RCR/RCL right/left through carry

- In RCR, as bits are shifted from left to right, they exit the right end (LSB) to the carry flag, and the carry flag enters the left end (MSB).
  - The LSB is moved to CF and CF is moved to the MSB.
    - CF acts as if it is part of the operand.
- In RCL, as bits are shifted from right to left they exit the left end (MSB) and enter the carry flag, and the carry flag enters the right end (LSB).
  - The MSB is moved to CF and CF is moved to the LSB.
    - CF acts as if it is part of the operand.

3.5: ROTATE INSTRUCTIONS
RCR right through carry


```
        CLC                     ;make CF=0
        MOV    AL,26H           ;AL=0010 0110
        RCR    AL,1             ;AL=0001 0011 CF=0
        RCR    AL,1             ;AL=0000 1001 CF=1
        RCR    AL,1             ;AL=1000 0100 CF=1
or:
        CLC                     ;make CF=0
        MOV    AL,26H           ;AL=0010 0110
        MOV    CL,3             ;CL=3 number of times to rotate
        RCR    AL,CL            ;AL=1000 0100 CF=1

;the operand can be a word
        STC                     ;make CF=1
        MOV    BX,37F1H         ;BX=0011 0111 1111 0001
        MOV    CL,5             ;CL=5 number of times to rotate
        RCR    BX,CL            ;BX=0001 1001 1011 1111 CF=0
```

– If the operand is to be rotated once, the 1 is coded.CF=1
  - If more than once, register CL holds the number of rotations.

3.5: ROTATE INSTRUCTIONS
RCL left through carry

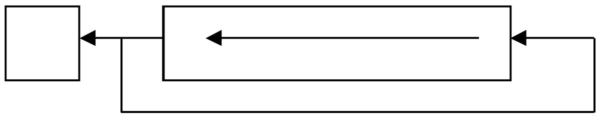

```
        STC                     ;make CF=1
        MOV     BL,15H          ;BL=0001 0101
        RCL     BL,1            ;0010 1011 CF=0
        RCL     BL,1            ;0101 0110 CF=0
or:
        STC                     ;make CF=1
        MOV     BL,15H          ;BL=0001 0101
        MOV     CL,2            ;CL=2 number of times for rotation
        RCL     BL,CL           ;BL=0101 0110 CF=0

;the operand can be a word:
        CLC                     ;make CF=0
        MOV     AX,191CH        ;AX=0001 1001 0001 1100
        MOV     CL,5            ;CL=5 number of times to rotate
        RCL     AX,CL           ;AX=0010 0011 1000 0001 CF=1
```

- If the operand is to be rotated once, the 1 is coded.
  - If more than once, register CL holds the number of rotations.

Example

Write a program that counts the number of 1's in a byte and writes it into BL

```
DATA1        DB 97                      ; 61h
             SUB   BL,BL                ;clear BL to keep the number of 1s
             MOV   DL,8 ;rotate total of 8 times
             MOV   AL,DATA1
AGAIN:  ROL   AL,1      ;rotate it once
             JNC   NEXT                 ;check for 1
             INC   BL   ;if CF=1 then add one to count
NEXT:   DEC   DL        ;go through this 8 times
             JNZ   AGAIN                ;if not finished go back
             NOP
```



3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

- **CMP destination,source**
  - Compares two operands & changes flags according to the result of the comparison, leaving the operand unchanged.
    - Destination operand can be in a register or in memory.
    - Source operand can be in a register, in memory, or immediate.
- CF, AF, SF, PF, ZF, and OF flags reflect the result.
  - Only CF and ZF are used.

Table 3-3: Flag Settings for Compare Instruction

Compare operands	CF	ZF
destination > source	0	0
destination = source	0	1
destination < source	1	0

Compare

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) − (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Unsigned Comparison		
Comp Operands	CF	ZF
Dest > source	0	0
Dest = source	0	1
Dest < source	1	0

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Signed Comparison		
Comp Operands	ZF	SF,OF
Dest > source	0	SF=OF
Dest = source	1	x
Dest < source	0	SF<>OF

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

- Compare is really a SUBtraction.
  - Except that the values of the operands do not change.
    - Flags are changed according to the execution of SUB.
    - Operands are unaffected regardless of the result.
    - Only the flags are affected.

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

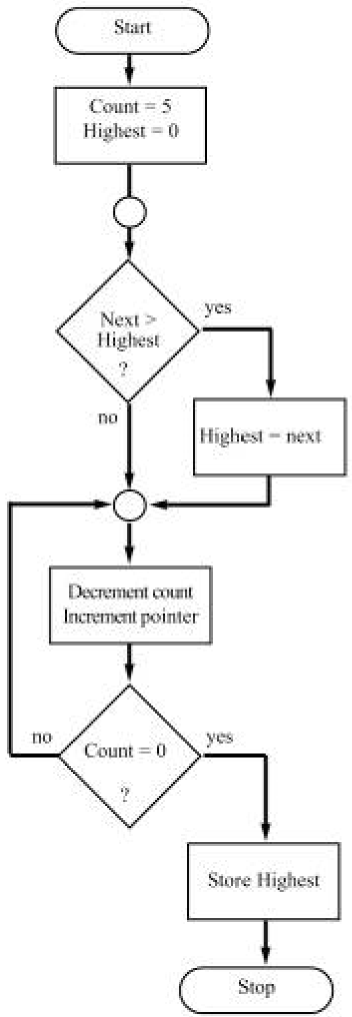
- Program 3-3 uses CMP to find the highest byte in a series of 5 bytes defined in the data segment.

Assume that there is a class of five people with the following grades: 69, 87, 96, 45, and 75. Find the highest grade.

```
TITLE       PROG3-3 (EXE) CMP EXAMPLE
PAGE        60,132
.MODEL SMALL
.STACK 64
;------------------
            .DATA
GRADES      DB     69,87,96,45,75
            ORG    0008
HIGHEST     DB     ?
;------------------
            .CODE
MAIN        PROC   FAR
            MOV    AX,@DATA
            MOV    DS,AX
```

See the entire program listing on page 107 of your textbook.

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers




```
Count = 5
Highest = 0

REPEAT
   IF (Next > Highest)
   THEN
          Highest = Next
   ENDIF
   Decrement Count
UNTIL Count = 0

Store Highest
```

- Program 3-3 searches five data items to find the highest grade, with a variable called "Highest" holding the highest grade found so far.

A REPEAT-UNTIL structure was used in the program, where grades are compared, one by one, to Highest.

If any of them is higher, that value is placed in Highest, continuing until all data items are checked.

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

- Program 3-3, coded in Assembly language, uses register AL to hold the highest grade found so far.
  - AL is given the initial value of 0.
- A loop compares each of the 5 bytes with AL.
  - If AL contains a higher value, the loop continues to check the next byte.
  - If AL is smaller than the byte checked, the contents of AL are replaced by that byte and the loop continues.

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

- There is a relationship between the pattern of lowercase/uppercase ASCII letters, as shown below for A and a:

```
A 0100 0001   41H
a 0110 0001   61H
```

The only bit that changes is **d5**.

To change from lowercase to uppercase, **d5** must be masked.

Letter	Hex	Binary	Letter	Hex	Binary
A	41	0100 0001	a	61	0110 0001
B	42	0100 0010	b	62	0110 0010
C	43	0100 0011	c	63	0110 0011
...	...	...	...	...	...
Y	59	0101 1001	y	79	0111 1001
Z	5A	0101 1010	z	7A	0111 1010

3.3: LOGIC INSTRUCTIONS
COMPARE of unsigned numbers

- Program 3-4 uses CMP to determine if an ASCII character is uppercase or lowercase.
  - It detects if the letter is in lowercase, and if it is, it is ANDed with 1101 1111B = DFH.
    - Otherwise, it is simply left alone.
  - To determine if it is a lowercase letter, it is compared with 61H and 7AH to see if it is in the range a to z.
    - Anything above or below this range should be left alone.

Compare Example

```
DATA1     DW        235Fh
…
MOV AX, CCCCH
CMP AX, DATA1
JNC OVER
SUB AX,AX
OVER: INC DATA1
```

CCCC – 235F = A96D => Z=0, CF=0 =>
CCCC > DATA1

Compare (CMP)

For ex: CMP CL,BL ; CL-BL; no modification on neither operands

Write a program to find the highest among 5 grades and write it in DL

```
DATA        DB      51, 44, 99, 88, 80          ;13h,2ch,63h,58h,50h
            MOV     CX,5                        ;set up loop counter
            MOV     BX, OFFSET DATA             ;BX points to GRADE data
        SUB     AL,AL                           ;AL holds highest grade found so far
AGAIN:      CMP     AL,[BX]                                 ;compare next grade to highest
        JA      NEXT                            ;jump if AL still highest
        MOV     AL,[BX]                         ;else AL holds new highest
NEXT:       INC     BX                          ;point to next grade
            LOOP AGAIN                                      ;continue search
        MOV  DL, AL
```

Dec Hex Bin

3 3 00000011

ORG ; ENDS

The x86 PC

assembly language, design, and interfacing

fifth edition

MUHAMMAD ALI MAZIDI
JANICE GILLISPIE MAZIDI
DANNY CAUSEY

The x86 PC
assembly language, design, and interfacing
fifth edition

Prentice Hall