EEE 410 – Microprocessors I

Spring 04/05 – Lecture Notes # 14

Outline of the Lecture

- **Signed Numbers and Signed Number Operations**
- **String Operations**

➢ SIGNED NUMBER DIVISION

IDIV ;(signed number division)

➢ According to Intel manual IDIV means "integer division". Note that all arithmetic instructions of 8086 are for integer numbers. For real numbers (i.e. 5.32) 8087 coprocessor is used.

Signed Division Summary:

Division	Numerator	Denominator	Quotient	Remainder
byte/byte	AL = byte CBW	register or memory	AL	AH
word/word	AX = word CWD	register or memory	AX	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 –127>AL>+127
2) Divide error interrupt if –32767>AX>+32767

➢ SIGNED NUMBER MULTIPLICATION

IMUL ;(signed number multiplication)

➢ According to Intel manual IMUL means "integer multiplication".

Signed Multiplication Summary:

Multiplication	Operand 1	Operand 2	Result
byte x byte	AL	register or memory	AX[1]
word x word	AX	register or memory	DXAX[2]
word x byte	AL = byte CBW	register or memory	DXAX[2]

Notes: 1) CF=1 and OF=1 if AH has part of the result, but if the result is not large enough to need AH, the sign bit is copied to the unused bits and CPU makes CF=0 and OF=0 to indicate that.
2) CF=1 and OF=1 if DX has part of the result, but if the result is not large enough to need DX, the sign bit is copied to the unused bits and CPU makes CF=0 and OF=0 to indicate that.

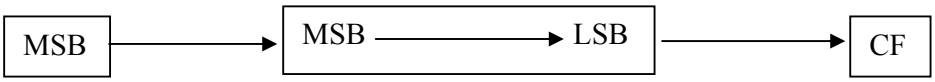
Example shown below (Program 6-1) is an application of signed number arithmetic, which computes the average of the following temperature measurements.

```
Ex:     ……..
        SIGN_DAT    DB      +13,-10,+19,+14,-18,-9,+12,-9,+16
                    ORG     0010H
        AVERAGE     DW      ?
        REMINDER    DW      ?
        ……..
                    MOV     CX,9                    ;load counter
                    SUB     BX,BX                   ;clear BX, used as an accumulator
                    MOV     SI,OFFSET SIGN_DAT      ;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
                    LOOP    BACK                    ;loop if not finished
                    MOV     AL,9                    ;move count to AL
```

```
CBW                     ;sign extend into AX
MOV  CX,AX              ;save denominator in CX
MOV  AX,BX              ;move sum to AX
CWD                     ;sign extend the sum
IDIV  CX                ;find the average
MOV  AVERAGE,AX         ;store the average (quoitent)
MOV  REMINDER,DX        ;store the reminder
…..
```

- **ARITHMETIC SHIFT**
  - There are two types of shifts: logical shift (unsigned numbers) and arithmetic shift (signed numbers).
  - The arithmetic shift is basically the same as the logical shift, except that the sign bit is copied to the shifted bits.

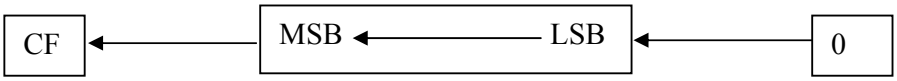
SAR destination,count ;(shift arithmetic right)



- As the bits of the destination are shifted to the right into CF, the empty bits are filled with the sign bit.
- SAR instruction can be used to divide a signed number by 2 as shown in the example below:

```
Ex:   MOV  AL,-10       ;AL=-10=F6H=1111 0110
      SAR  AL,1         ;AL is shifted right arithmetic once to get: AL=1111 1011 =FDH =-5
```

SAL destination,count ;(shift arithmetic left) the same as **SHL (shift left)**



- SAL and SHL are performing exactly the same task.

- **SIGNED NUMBER COMPARISON**

CMP destination ,source ;(compare destination with source)

destination > source **OF=SF or ZF=0**
destination = source **ZF=1**
destination < source **OF=negation of SF**

- The mnemonics used to detect the conditions above are as follows:

JG	Jump Greater	jump if OF=SF or ZF=0
JGE	Jump Greater or Equal	jump if OF=SF
JL	Jump Less	jump if OF=inverse of SF
JLE	Jump Less or Equal	jump if OF=inverse of SF or ZF=1
JE	Jump Equal	jump if ZF=1

- **STRING OPERATIONS**

  - In 8086 family of microprocessors there are a group of instructions referred to as the *string* instructions.
  - DI and SI registers are used to point the source and the destination operands. Until now because the ES(Extra segment) is not defined within the programs both the DI and SI registers were used the offset of DS(Data segment).
  - When ES is defined as:

```
…..
ASSUME      CS:CODSEG, DS:DATSEG, SS:STASEG,ES:DATSEG
MOV  AX,DATSEG
MOV  DS,AX
MOV  ES,AX
…
```

  then by default DI becomes the offset address of ES and SI becomes the offset address of DS.

  - In each of the string instructions the operands can be byte or word. This is indicated by adding B (byte) and W (word) to the end of the instruction mnemonic.

String Operation Summary:

Instruction	Mnemonic	Destination	Source	Prefix
move string byte	MOVSB	ES:DI	DS:SI	REP
move string word	MOVSW	ES:DI	DS:SI	REP
store string byte	STOSB	ES:DI	AL	REP
store string word	STOSW	ES:DI	AX	REP
load string byte	LODSB	AL	DS:SI	none
load string word	LODSW	AX	DS:SI	none
compare string byte	CMPSB	ES:DI	DS:SI	REPE/REPNE
compare string word	CMPSW	ES:DI	DS:SI	REPE/REPNE
scan string byte	SCASB	ES:DI	AL	REPE/REPNE
scan string word	SCASW	ES:DI	AX	REPE/REPNE

- **DF, the direction flag:** To process operands in consecutive memory locations requires that the pointer be incremented or decremented. DF in string operations is used to indicate if SI and DI pointers will increment or decrement automatically.
- It is the job of the programmer to specify the choice of increment or decrement by setting the direction flag high or low.

CLD ;(clear direction flag) will reset (put to zero) DF, indicating that the string instruction should increment the pointers (SI,DI) automatically. This automatic incrementation is sometimes referred as *autoincrement*.

STD ;(set direction flag) will set (put to one) DF, indicating that the string instruction should decrement the pointers (SI,DI) automatically.

REP prefix ; The REP (repeat) allows a string instruction to perform the operation repeatedly. REP assumes that CX holds the number of times that the instruction should be repeated. As the operation continues the CX register is decremented until it becomes zero.

REPE/ REPNE prefix ; REPE allows a string instruction to perform the operation repeatedly as long as CX is not zero and the comparison gives equality. REPNE allows a string instruction to perform the operation repeatedly as long as CX is not zero and the comparison gives inequality.

Ex: Using the string instructions, write a program that transfers a block of 20 bytes of data.

In the data segment:

```
DATA1   DB   'ABCDEFGHIJKLMNOPQRST'
        ORG  30H
DATA2   DB   20 DUP (?)
```

In the code segment:

```
        ASSUME    CS:CODESEG,DS:DATSEG,SS:STASEG, ES:DATSEG
        MOV  AX,DATSEG
        MOV  DS,AX                  ; initialize the data segment
        MOV  ES,AX                  ; initialize the extra segment
        CLD                         ;clear direction flag for autoincrement
        MOV  SI,OFFSET DATA1        ;load the source pointer
        MOV  DI,OFFSET DATA2        ;load the destination pointer
        MOV  CX,20                  ;load the counter
        REP  MOVSB                  ;repeat until CX becomes zero
```

- After the transfer of every byte by the MOVSB instruction, both SI and DI registers are incremented automatically once only (notice CLD).
- The REP (repeat) prefix causes the CX counter to be decremented and MOVSB is repeated until CX becomes zero.

Ex: Assuming that there is a spelling of "Europe" in an electronic dictionary and a user types in "Euorope", write a program that compares these two and displays the following message, depending on a result:

1. If they are equal, display "The spelling is correct"
2. If they are not equal, display "Wrong Spelling"

;from the data segment

```
DAT_DIC     DB   'Europe'
DAT_TYPED   DB   'Euorope'
MESSAGE1    DB   'The spelling is correct','$'
MESSAGE2    DB   'Wrong spelling','$'
```

;from code segment:

```
            ....
            CLD                         ;DF=0 for increment
            MOV  SI,OFFSET DAT_DIC      ;SI=offset of DAT_DIC
            MOV  DI,OFFSET DAT_TYPED    ;DI=offset of DAT_TYPED
            MOV  CX,06                  ;load the counter
            REPE CMPSB                  ;repeat as long as equal or until CX=0
            JE   OVER                   ;if ZF=1 then display MESSAGE1
            MOV  DX,OFFSET MESSAGE2     ;if ZF=0 then display MESSAGE2
            JMP  DISPLAY
OVER:       MOV  DX,OFFSET MESSAGE1
DISPLAY:    MOV  AH,09
            INT  21H
```