**Lab Session 08**

**Objective**

* **Shift and Rotate Instruction**
* **Multiplication and Division**
* **Extended Addition and Subtraction**



**Shift and Rotate Instructions**



The 8086-based processors provide a complete set of instructions for shifting and rotating bits.

***Shift Instructions:***

Shift instructions move bits a specified number of places to the right or left. The last in the direction of the shift goes into the carry flag, and the first bit is filled with 0 or with the previous value of the first bit.

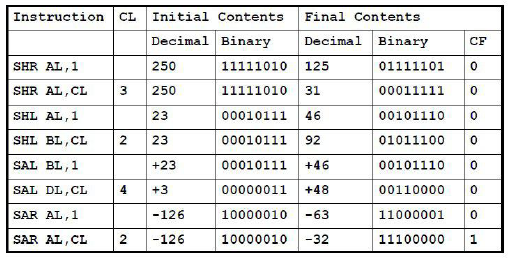
There are two different sets of shift instructions

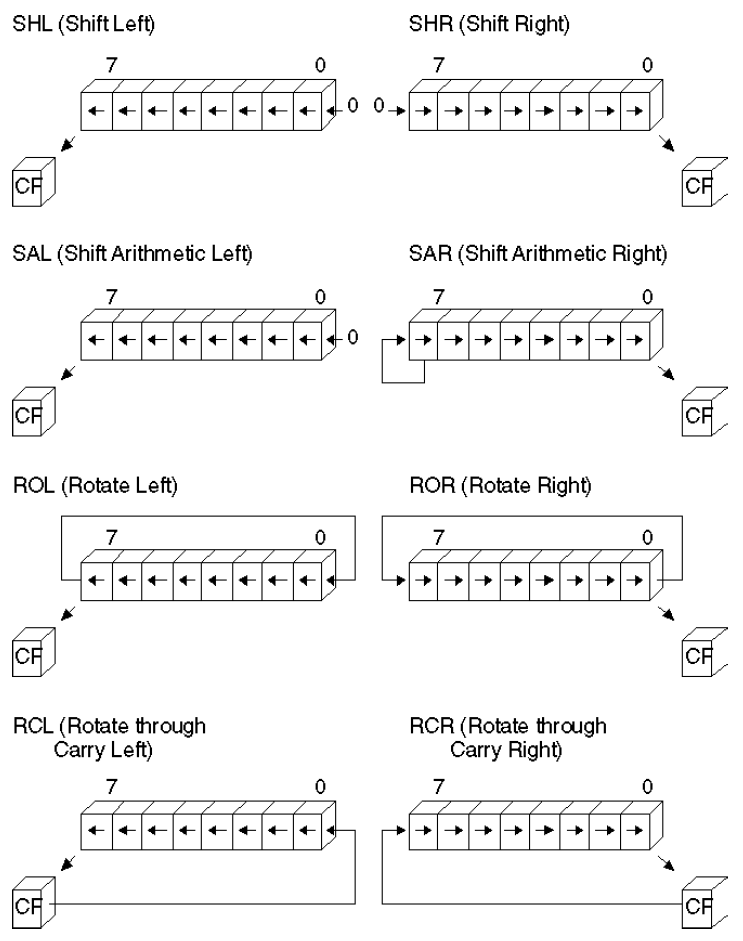
1. One set for doubling and halving unsigned binary numbers

* SHL (Shift Left)
* SHR (Shift Right)

1. The other for doubling and halving signed

* SAL (Arithmetic Shift Left)
* SAR (Arithmetic Shift Right)





**1. SHL:**(Shift Left)

**Syntax:** *SHL destination, count*

*mov bl,8Fh ;BL=10001111b*

*SHL bl,1 ;CF=1, BL=00011110b*

*mov al,10000000b ;AL=10000000b*

*SHL al,2 ;CF=0, AL=00000000b*

*mov dl,5 ;DL=00000101b =5*

*SHL dl,1 ;CF=0, DL=00001010b =10*

**2. SHR:**(Shift Right)

**Syntax:** *SHR destination, count*

*mov dl,32 ;DL=00100000b =32*

*SHR dl,1 ;DL=00010000b, CF=0 =16*

**3. SAL& SAR:**(Shift Arithmetic Left)& (Shift Arithmetic Right)

**Syntax:** *SAL destination, count*

*SAR destination, count*

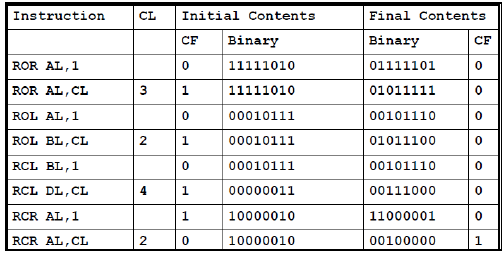
*mov ax,-128 ;EAX=????FF80h*

*SHL eax,16 ;EAX=FF800000h*

*SAR eax,16 ;EAX=FFFFFF80h*

***Rotate Instructions:***

Rotate instructions also move bits a specified number of places to the right or left. For each bit rotated the last bit in the direction of the rotate operation moves into the first bit position at the other end of the operand. With some variations, the carry bit is used as an additional bit of the operand.**RCR** (Rotate Carry Right) and **RCL** (Rotate Carry Left) instructions carry values from the first register to the second by passing the leftmost or rightmost bit through the carry flag.

****

**1. ROL:**(Rotate Left)

**Syntax:** *ROL destination, count*

*mov ax,6A4Bhh ;AX=A4B6h*

*ROL ax,4 ;AX=4B6Ah*

*ROL ax,4 ;AX=B6A4h*

*ROL ax,4 ;AX=6A4B*

*ROL ax,4*

*mov al,26h*

*ROL al,4 ;AL=62h*

**2. ROR:**(Shift Right)

**Syntax:** *ROR destination, count*

*mov al,0000100b*

*ROR al,3 ;DL=10000000b, CF=1*

**3. RCL & RCR:**(Rotate Carry Left) & (Rotate Carry Right)

**Syntax:** *RCL destination, count*

*RCR destination, count*

*CLC ; CF=0*

*mov bl,88h ; CF,BL = 0 10001000b*

*RCL bl,1 ;CF,BL = 1 00010000b*

*RCL bl,1 ;CF,BL = 0 00100001b*

*STC ;CF=1*

*mov ah,10h ;AH,CF=00010000 1*

*RCR ah,1 ;AH,CF=10001000 0*

***SHLD/SHRD Instructions:***

**Syntax:** *SHLD destination,source, count*

**EXAMPLE # 01:**

.data

a WORD 9BA6h

.code

mov ax,0AC36h

shld a,ax,4 ;a=BA6Ah

**EXAMPLE # 02:**

.code

mov ax,234Bh

mov dx,7654h

shrd ax,dx,4 ;ax=4234h

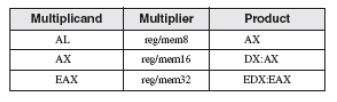


**Multiplication and Division Instruction**



***MUL Instructions:***

The **MUL** instruction is for unsigned multiplication. Operands are treated as unsigned numbers.



**Syntax:** *MUL source*

**EXAMPLE # 01:**

*mov al,5h*

*mov bl,10h*

*mul bl ; AX = 0050h, CF = 0*

**EXAMPLE # 02:**

*.data*

*val1 WORD 2000h*

*val2 WORD 0100h*

*.code*

*mov ax,val1 ; AX = 2000h*

*mul val2 ; DX:AX = 00200000h, CF = 1*

**EXAMPLE # 03:**

*mov eax,12345h*

*mov ebx,1000h*

*mul ebx ; EDX:EAX = 0000000012345000h, CF = 0*

***IMUL Instructions:***

The **IMUL** instruction is for signed multiplication. Operands are treated as signed numbers and result is positive or negative depending on the signs of the operands.

**Syntax: I***MUL source*

The following instructions multiply 48 by 4, producing -192 in AX. Although the product is correct, AH is not a sign extension of AL, so the Overflow flag is set:

*mov al,48*

*mov bl,4*

*imul bl ;AX = 00C0h, OF = 1*

The following instructions multiply -4 by 4, producing -16 in AX. AH is a sign extension of AL so the Overflow flag is clear:

*mov al,-4*

*mov bl,4*

*imul bl ; AX = FFF0h, OF = 0*

The following instructions perform 32-bit signed multiplication (4,823,424 \* -423), producing -2,040,308,352 in EDX:EAX. The Overflow flag is clear because EDX is a sign extension of EAX:

*mov eax,+4823424*

*mov ebx,-423*

*imul ebx ; EDX:EAX = FFFFFFFF86635D80h, OF = 0*

The following instructions demonstrate two-operand formats:

**EXAMPLE # 01:**

.data

word1 SWORD 4

dword1 SDWORD 4

.code

mov ax,-16 ; AX = -16

mov bx,2 ; BX = 2

imul bx,ax ; BX = -32

imul bx,2 ; BX = -64

imul bx,word1 ; BX = -256

mov eax,-16 ; EAX = -16

mov ebx,2 ; EBX = 2

imul ebx,eax ; EBX = -32

imul ebx,2 ; EBX = -64

imul ebx,dword1 ; EBX = -256

The following instructions demonstrate three-operand formats, including an example of signed overflow:

**EXAMPLE # 02:**

.data

word1 SWORD 4

dword1 SDWORD 4

.code

imul bx,word1,-16 ; BX = -64

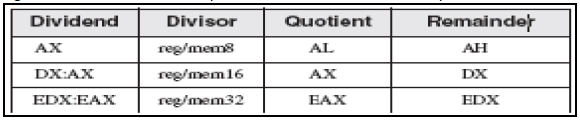
imul ebx,dword1,-16 ; EBX = -64

imul ebx,dword1,-2000000000 ; OF = 1

***DIV Instructions:***

The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit unsigned integer division. The single register or memory operand is the divisor.

The following table shows the relationship between the dividend, divisor, quotient, and remainder:



**Syntax:** *DIV source*

*mov ax,0083h ; dividend*

*mov bl,2 ; divisor*

*div bl ; AL = 41h, AH = 01h*

*mov dx,0 ; clear dividend, high*

*mov ax,8003h ; dividend, low*

*mov cx,100h ; divisor*

*div cx ; AX = 0080h, DX = 0003h*

***Sign Extension Instructions(CBW,CWD,CDQ):***

Dividends of signed integer division instructions must often be sign-extended before the division takes place. Intel provides three useful sign extension instructions: CBW, CWD, and CDQ. The CBW instruction (convert byte to word) extends the sign bit of AL into AH, preserving the number’s sign. In the next example, 9Bh (in AL) and FF9Bh (in AX) both equal −101 decimal:

**EXAMPLE # 01:**

.data

byteVal SBYTE -101 ; 9Bh

.code

mov al,byteVal ; AL = 9Bh

cbw ; AX = FF9Bh

**The CWD (convert word to doubleword) instruction extends the sign bit of AX into DX:**

.data

wordVal SWORD -101 ; FF9Bh

.code

mov ax,wordVal ; AX = FF9Bh

cwd ; DX:AX = FFFFFF9Bh

**The CDQ (convert doubleword to quadword) instruction extends the sign bit of EAX into EDX:**

.data

dwordVal SDWORD -101 ; FFFFFF9Bh

.code

mov eax,dwordVal

cdq ; EDX:EAX = FFFFFFFFFFFFFF9Bh

***IDIV Instructions:***

**Syntax: I***DIV source*

**EXAMPLE # 01:**

.data

byteVal SBYTE -48 ; D0 hexadecimal

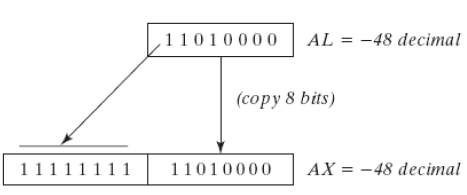
.code

mov al,byteVal ; lower half of dividend

cbw ; extend AL into AH

mov bl,+5 ; divisor

idiv bl ;AL = -9, AH = -3





**Extended Addition and Subtraction**



***ADC Instructions:***

The ADC (add with carry) instruction adds both a source operand and the contents of the Carry flag to a destination operand.

**Syntax:** *ADC Destination, source*

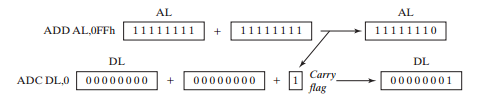
**EXAMPLE # 01:**

mov dl,0

mov al,0FFh

add al,0FFh ; AL = FEh

adc dl,0 ; DL/AL = 01FEh



***SBB Instructions:***

The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand.

**Syntax:** *SBB Destination, source*

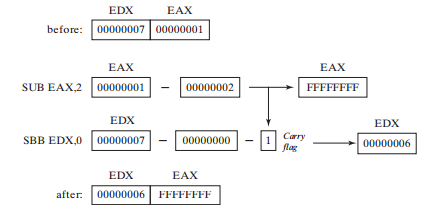
**EXAMPLE # 01:**

*mov edx,7 ; upper half*

*mov eax,1 ; lower half*

*sub eax,2 ; subtract 2*

*sbb edx,0 ; subtract upper half*



**ACTIVITIES:**

1. Implement the following C++ statement, using unsigned 32-bit integers:

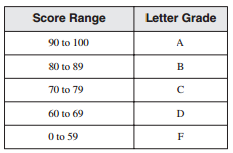
var4 = (var1 \* 5) / (var2 - 3);

1. Implement the following C++ expression in assembly language, using 32-bit signed operands:

val1 = (val2 / val3) \* (val1 % val2);

var4 = (var1 \* -5) / (-var2 % var3);

1. Write ASM instructions that calculate EAX \* 21 using binary multiplication. Hint: 21 = 24 + 22 + 20.
2. Give an assembly language program to move -128 in ax and expend eax. Using shift and rotate instruction.
3. Create a procedure **Extended\_Sbb** procedure to subtract two 64-bit (8-byte) integers.
4. Using the following table as a guide, write a program that asks the user to enter an integer test score between 0 and 100. The program should display the appropriate letter grade:



1. Write a program that performs simple encryption by rotating each plaintext byte a varying number of positions in different directions. For example, in the following array that represents the encryption key, a negative value indicates a rotation to the left and a positive value indicates a rotation to the right. The integer in each position indicates the magnitude of the rotation:

key BYTE -2, 4, 1, 0, -3, 5, 2, -4, -4, 6

Your program should loop through a plaintext message and align the key to the first 10 bytes of the message. Rotate each plaintext byte by the amount indicated by its matching key array value.

Then, align the key to the next 10 bytes of the message and repeat the process.

