**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.

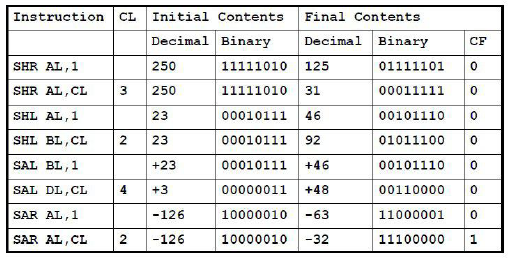
There are two different sets of shift instructions

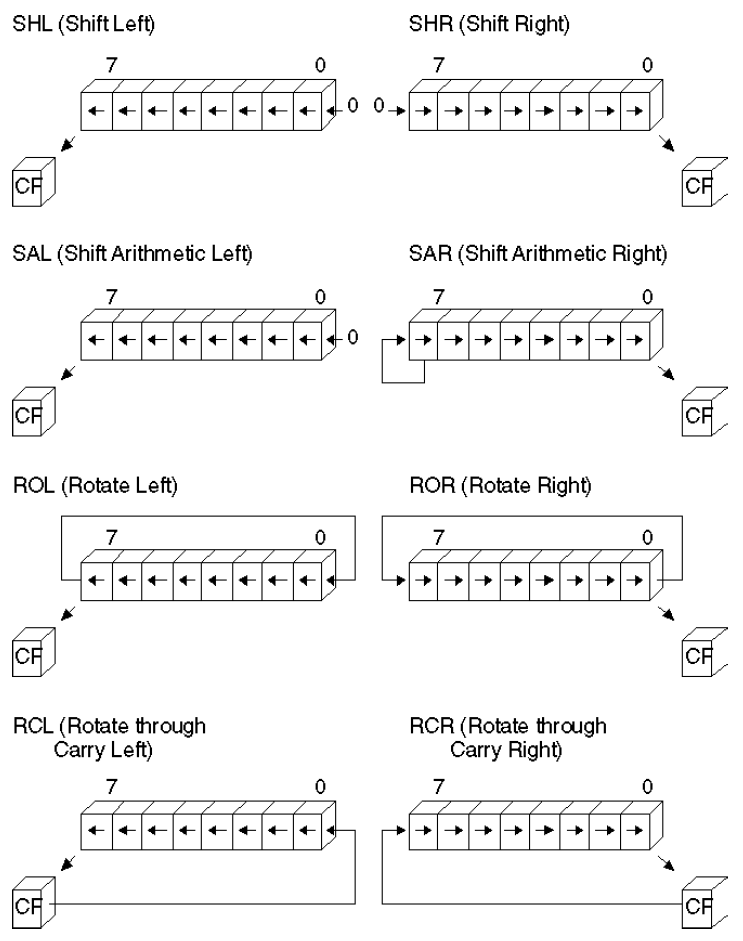
1. One set for doubling and halving unsigned binary numbers

* 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*

.code

main PROC

mov bl,0Fh ; BL=10001111b = 15

SHL bl,1 ; CF=1, BL=00011110b = 30

call dumpregs

mov al,00001000b ; AL=00001000b = 8

SHL al,2 ; CF=0, AL=00100000b = 32

call dumpregs

mov dl,5 ; DL=00000101b =5

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

call dumpregs

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

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

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

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

*SHR dl,1 ;DL=00001000b, CF=0 =8*

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

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

*SAR destination, count*

*movax,-128 ;EAX=????FF80h*

*SAL eax,16 ;EAX=FF800000h*

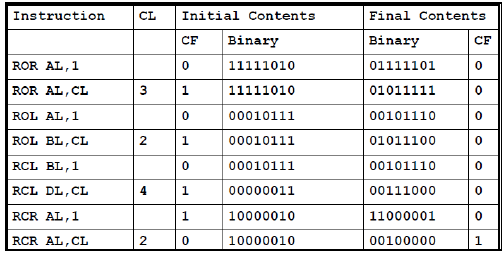
call dumpregs

*SAR eax,16 ;EAX=FFFFFF80h*

call dumpregs

***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=6A4Bh*

*ROL ax,4 ;AX=A4B6h*

*ROL ax,4 ;AX=4B6Ah*

*ROL ax,4 ;AX=B6A4h*

*ROL ax,4 ;AX=6A4B*

*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*

## Clear Carry Flag (clc)

**Operation**

0 -> CF

**Description**

Sets the carry flag to zero; affects no other flags.

## Set Carry Flag (stc)

stc

##### Operation

1 -> CF

##### Description

Sets the carry flag to 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:**

.code

MOV EAX,0

MOV EBX,0

MOV BX,9BA6H

mov ax,0AC36h

shldBX,ax,4 ;a=BA6Ah

call dumpregs

shldBX,ax,4 ;a=A6AAh

call dumpregs

shldBX,ax,4 ;a=6AAAh

call dumpregs

**EXAMPLE # 02:**

.code

MOV EAX,0

MOV EDX,0

mov ax,234Bh

mov dx,7654h

shrdax,dx,4 ;ax=4234h

CALL DUMPREGS

shrdax,dx,4 ;ax=4423h

CALL DUMPREGS

shrdax,dx,4 ;ax=4442h

CALL DUMPREGS

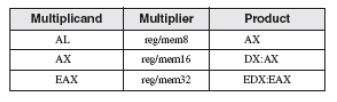


**Multiplication and Division Instruction**



***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*

*mulbl ; AX = 0050h, CF = 0*

**EXAMPLE # 02:**

*.data*

*val1 WORD 2000h*

*val2 WORD 0100h*

*.code*

*movax,val1 ; AX = 2000h*

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

**EXAMPLE # 03:**

*mov eax,12345h*

*mov ebx,1000h*

*mulebx*

*; 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*

*imulbl ;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:

*moval,-4*

*mov bl,4*

*imulbl ; 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:

*moveax,+4823424*

*movebx,-423*

*imulebx ; EDX:EAX = FFFFFFFF86635D80h, OF = 0*

The following instructions demonstrate two-operand formats:

**EXAMPLE # 01:**

.data

word1 SWORD 4

dword1 SDWORD 4

.code

movax,-16 ; AX = -16

mov bx,2 ; BX = 2

imulbx,ax ; BX = -32

imul bx,2 ; BX = -64

imulbx,word1 ; BX = -256

moveax,-16 ; EAX = -16

mov ebx,2 ; EBX = 2

imulebx,eax ; EBX = -32

imul ebx,2 ; EBX = -64

imulebx,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

imulbx,word1,-16 ; BX = -64

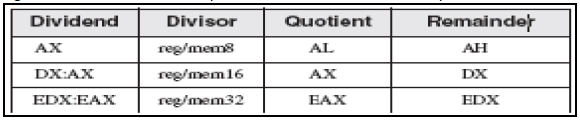
imulebx,dword1,-16 ; EBX = -64

imulebx,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

moval,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

movax,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

moveax,dwordVal

cdq ; EDX:EAX = FFFFFFFFFFFFFF9Bh

***IDIV Instructions:***

**Syntax: I***DIV source*

**EXAMPLE # 01:**

.data

byteVal SBYTE -48 ; D0 hexadecimal

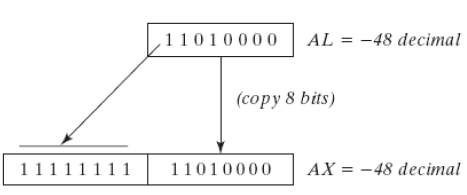
.code

moval,byteVal ; lower half of dividend

cbw ; extend AL into AH

movbl,+5 ; divisor

idivbl ; 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*

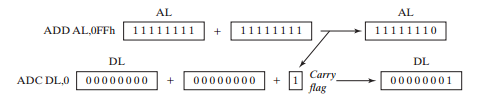
**EXAMPLE # 01:**

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*

