Date

what are booleans?

boolean values are either true or false represented by 1 or 0

- and, or, not, xor

Boolean and comparison instructions:

neg, cmp, and, or, xor, not changes the flag registers.

AND OPERATION

and instruction performs a bit wise "and" operation between corresponding bits in the two operands and places the result in the first operand.

0	0	0
0	1	0
1	0	0
1	1	1

```
and reg, reg
and reg, mem
and reg, imm
and mem, reg
and mem, imm
```

reg, mem and imm can be 8, 16 or 32 bits

example:

```
  0011 1011
  0000 1111
  ---------
  0000 1011
```

cleared → ← unchanged by placing 1's in those positions

by placing 0's under the positions which we want to clear

The AND instruction can be used to clear selected bits in an operand while preserving the remaining bits. This is called masking.

```
mov al, 00111011b
and al, 00001111b        ; al = 0000 1011b
```

Date:

OR Instruction:

The "or" instruction performs a bitwise OR operation between corresponding bits in the two operands and places the result in the first operand.

```
OR   reg, reg
OR   reg, mem
OR   reg, imm
OR   mem, reg
OR   mem, imm
```

reg, mem and imm can be 8, 16 or 32 bits

example:

```
  0011 1011
  0000 1111
  ---------
  0011 1111
```

unchanged by placing 0's — set by placing 1's

"OR instruction can be used to set selected bits in an operand while preserving the remaining bits."

example:

OR can be used to convert a one-digit value into its ASCII equivalent

```
mov dl, 5        ; binary value
or  dl, 30h      ; convert to ASCII
```

Note: ORing a value with itself preserves the value but sets flag

```
ZF = 1      if AL = 0
SF = 1      if AL < 0
SF = ZF = 0 if AL > 0
or al, al   ; sets the flags.
```

Date

XOR Instruction

The XOR instruction performs a bit wise xor operation between two corresponding bits in the two operands and places the result in the first operand.

```
xor reg, reg
xor reg, mem
xor reg, imm
xor mem, reg
xor mem, imm
```

"The xor instruction can be used to reserve selected bits in an operand while preserving the remaining bits"

example:

```
  0011 1011
  0011 1111
  ---------
  0000 0100
```

unchanged by placing 0's — inverted by placing 1's

```
mov al, 00111011
xor al, 00111111        ; al = 00000100b
```

example: checking the parity flag

- parity flag indicates whether the lowest order byte of the result an arithmetic or bit wise operation has an even or odd number of 1's

- parity flag = 1 if parity is even
- parity flag = 0 if parity is odd

Q. we want to find the parity of a number without changing its value:

```
mov al, 10110101b       ; 5 bits = odd parity
xor al, 0               ; pf = 0
mov al, 11010100b       ; 4 bits = even parity
xor al, 0               ; pf = 1
```

Date

example. 16 bit parity flag
you can check the parity of a 16-bit register by performing an exclusive or between the upper and lower bytes.

```
mov ax, 64C1h      ; 0110 0100 1100 0001
xor ah, al         ; pf = 1
```

NOT instruction.
reverses / inverts all bits in an operand.

```
not reg
not mem
```

example:

```
mov al, 11110000b
not al     ; al = 0Fh
             or 0000 1111b
```

"and, or, xor affects the following flags with the result determining their actual values."
- overflow, sign, zero, parity, carry, auxiliary carry

TEST instruction
performs an implied "and" operation between corresponding bits in the two operands and sets the flag without modifying either operand.

```
TEST reg, reg
TEST reg, mem      reg, mem and immediate can be 8, 16
TEST reg, imm      or 32 bits.
TEST mem, reg
TEST mem, imm
```

Signature _____ RS No. _____

Date

example: The TEST instruction can check several bits at once
- if we wants to know either bit 0 or bit 3 is set in the al register, then:

```
test al, 00001001b      ; test bits 0 and 3
```

```
al:  00 10 0101
     0000 1001
     ---------
     00 0000 01     result: zf = 0
```

```
al:  0010 0100
     0000 1001
     ---------
     0000 0000      result: zf = 1
```

CMP Instruction: sets the flags as if it had performed subtraction on the operand.
- neither operand is changed.

```
cmp reg, reg
cmp reg, mem
cmp reg, imm
cmp mem, reg
cmp mem, imm
```

unsigned

cmp result	ZF	CF
dest. < source	0	1
dest. > source	0	0
dest = source	1	0

signed / unsigned

cmp results	Flags
dest < source	SF ≠ OF
dest. > source	SF = OF
dest = source	ZF = 1

examples:
- subtracting 5-10 requires a borrow.

```
mov ax, 5
cmp ax, 10      CF = 1
```

- subtracting 1000 from 1000 results in zero

```
mov ax, 1000
cmp ax, 1000    ; ZF = 1
```

- subtracting 0 from 105 produces a positive difference.

```
mov si, 105
cmp si, 0       ; ZF = 0 and CF = 0
```

Signature _____ RG No. _____

Date

Setting and clearing individual flags

- setting and clearing zero flag:

```
and al,0    ; sets zero flag
or al,1     ; clears zero flag
```

- setting and clearing the sign flag

```
or  al, 80h     ; sets sign flag
and al, 7Fh     ; clears sign flag
```

- setting and clearing the carry flag

```
stc     ; sets carry flag
clc     ; clears carry flag
```

- setting and clearing the overflow flag

```
mov al, 7Fh     ; al = +127
inc al          ; al = 80h ; OF = 1
or eax, 0       ; clears overflow flag.
```

CONDITIONAL STRUCTURES

example #01: comparing al to zero. Jump to L1 if the zero flag was set

```
cmp al,0
jz  L1
L1: ___
```

example #02: perform a bitwise "and" on the dl register jump to L2 if the zero flag is clear.

```
and dl, 10110000b
jnz L2

L2: ___
```

Conditional Jumps

Date _____

cond #1:

```
mov ax, 5
cmp ax, 5
je  L1          ; jmp if equal

mov ax, 6
cmp ax, 4
jg  L1          : jmp if greater

mov ax, 5
cmp ax, 6
jl  L1          ; jmp if less
```

EXAMPLES

Date

Q. Convert the character in al to upper case

Solution: use the AND instruction to clear bit 5

```
mov al, 'a'          ; al = 01100001b
and al, 11011111b    ; al = 0100 0001b
```

Q. Convert a binary decimal byte into its equivalent ascii decimal digit

solution. use the or instruction to set bits 4 and 5

```
mov al, 6            ; al = 0000 0110b
or al, 00110000b     ; al = 0011 0110b
```

Q. Jump to a label if an integer is even

solution: "and" the lowest bit with 1, if result = zero, then the number is even.

```
mov ax, wordval
and ax, 1
jz evenvalue         ; jmp if zero flag.
```

Q. Jump to a label if the value in al is not zero

Solution: "or" the byte with itself, then use jnz

```
or al, al
jnz isNotzero        ; jmp if not zero.
```

Q. Jump to a label if either bit 0 or bit 1 in al is set

```
test al, 0000 0011b
jnz valueFound
```

Q. Jump to a label if neither bit 0 or bit 1 in al is set

```
test al, 0000 0011b
jz noValueFound
```

Date

example #01:

```
mov edx, 0A523h
cmp edx, 0A523h
jne L5                ; jmp not taken
je  L1                ; jmp taken
```

example #02:

```
mov bx, 1234h
sub bx, 1234h
jne L5                ; jmp not taken
je  L1                ; jmp taken
```

example #03:

```
mov cx, 0FFFFh
inc cx
jcxz L2               ; jmp taken
```

example #04:

```
xor ecx, ecx
jecxz L1              ; jmp taken
```

example #05: all characters in an array are converted to upper case.

```
.data
array byte 50 "This sentence is in mixed case", 0
.code
    mov ecx, lengthof array
    mov esi, offset array
L1:
    and byte ptr [esi], 11011111b
    inc esi
    loop L1
```

Date

7.2 Shift and Rotate Instructions

- Shifting means to move bit right and left inside an operaud.

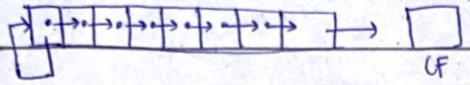
7.2.1 logical shifts and arithmetic shifts

- logical shift : fills a newly created bit position with a zero.

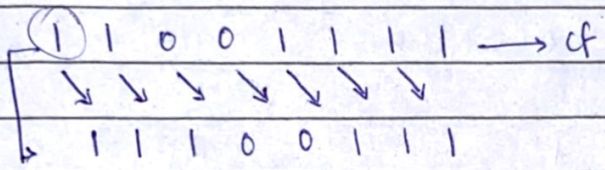
example: 11001111

0 1 1 0 0 1 1 1 CF

- arithmetic shift: the newly created bit position is filled with a copy of the original number's sign bit



example



7.2.2 SHL Instruction: performs a "logical left shift" on the destination operand, filling the lowest bit with 0.

→ The highest bit is moved into the cf, and the bit that was in the cf was discarded.

syntax: shl destination, count

```
shl reg, imm8
shl mem, imm8
shl reg, cl
shl mem, cl
```

imm8 : 0 — 255

cl : can contain the shift count.

example: cf ← 1 1 0 0 1 1 1 1 shift left by 1 bit

1 0 0 1 1 1 1 0

```
example: mov bl, 8Fh      ; bl = 10001111b
         shl bl, 1        ; cf = 1 , bl = 00011110b
```

Signature _____ RG No. _____

Date

Multiple Shifts: when a value is shifted leftward multiple-times, the carry flag contains the last bit to be shifted out of the MSB In the following example, bit 7 does not end up in the cf because it is replaced by bit 6.

```
mov al, 10000000b
shl al, 2          ; CF = 0, al = 00000000b
```

similarly when a value is shifted rightward multiple-times, the carry flag contains the last bit to be shifted out of the LSB

BITWISE MULTIPLICATION: Shl can perform multiplication by power of 2.

- shifting any operand left by n bits multiplies the operand by 2^n

example: 5: 00000101

```
mov dl, 5
shl dl, 1          ; dl = 00001010
```

operand → (5) × $2^{(1)}$ ← count = 10

example: 00001010

```
mov dl, 10
shl dl, 2          ; dl = 00101000
```

$10 \times 2^2 = 40$

7.2.3 SHR Instruction: The SHR instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0.

```
mov al, 0D0h       ; al = 11010000b
shr al, 1          ; al = 01101000b
                   CF = 0
```

Multiple shifts:

```
mov al, 00000010b
shr al, 2          ; al = 00000000  CF = 1
```

Signature _____ RS. No. _____

Date

Bitwise Division: logically shifting an unsigned integer right by n bits divides the operand by 2^n.

example: we divide 32 by 2^1, producing 16

```
mov dl, 32      ; dl = 00100000b = 32
shr dl, 1       ; dl = 00010000b = 16
```

example: (64) operand is divided by $2^{(3)}$ → count

```
mov dl, 64      ; dl = 01000000b = 64
shr dl, 3       ; dl = 00001000b = 8
```

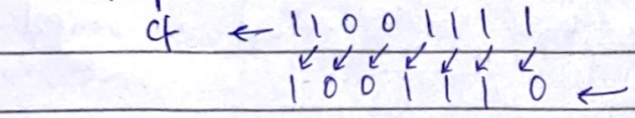
"DIVISION OF SIGNED NUMBERS BY SHIFTING IS ACCOMPLISHED USING THE SAR INSTRUCTION BECAUSE IT PRESERVES THE NUMBER'S SIGN BIT."

7.2.4 SAL and SAR Instruction

SAL: shift arithmetic left instruction works the same as the SHL instruction. for each shift count, sal shifts each bit in the destination operand to the next highest bit position.

→ lowest bit is assigned 0
→ highest bit is moved to the cf.

example:



SAR: shift arithmetic right instruction performs a right arithmetic shift on its destination operand.

- SAR duplicates the sign bit.

example:

```
mov al, 0F0h    ; al = 11110000b  (-16)
sar al, 1       ; al = 11111000b  (-8)
                                  cf = 0
```

Date

Signed Division: You can divide an operand by a power of 2, using sar.

example: -128 divided by 2^3 $\quad \frac{-128}{8} = -16$

```
mov dl, -128        ; dl = 10000000b  (-128)
sar dl, 3           ; dl = 11110000b  (-16)
```

Sign-extend ax into eax: suppose ax contains a signed integer and you want to extend its sign into eax.

Step #01: shift 16 bits to the left (eax)
Step #02: arithmetically shift 16 bits to the right

```
mov ax, -128        ; eax = ????FF80h
shl eax, 16         ; eax = FF800000h
sar eax, 16         ; eax = FFFFFF80h
```

7.2.5 rol instruction: rotate left shifts each bit to the left. The highest bit is copied into the cf and the lowest bit position.

"BIT ROTATION DOES NOT LOSE BITS."
"A BIT ROTATED OFF ONE END OF A NUMBER APPEARS AGAIN AT THE OTHER END."

note in the following examples how the high bit is copied into

```
mov al, 40h         ; al = 01000000b
rol al, 1           ; al = 10000000b  ; cf = 0
rol al, 1           ; al = 00000001b  cf = 1
rol al, 1           ; al = 00000010b  cf = 0
```

Multiple Rotations: when using a rotation count greater than 1, the cf contains the last bit rotated out of the MSB

```
mov al, 0010 0000b
rol al, 3           ; al = 00000001  cf = 1
```

Signature _____ No. _____

Date

Exchanging group of bits : you can use 'rol' to exchange the upper (bits 4-7) and lower (bits 0-3) halves of a byte.

```
mov al, 26h
rol al, 4          ; al = 62h
```

when rotating a multibyte integer by 4bits, the effect is to rotate each hexadecimal digit one position to the right or left.

```
mov ax, 6A4Bh
rol ax, 4          ; ax = A4B6h
rol ax, 4          ; ax = 4B6Ah
rol ax, 4          ; ax = B6A4h
rol ax, 4          ; ax = 6A4Bh
```

ROR Instruction : The rotates right instruction shifts each bit to the right and copies the lowest bit into the cf and the highest bit position.

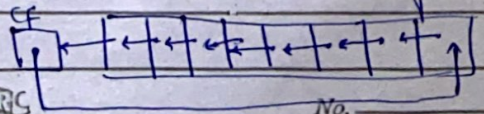
```
mov al, 01h        ; al = 00000001b
ror al, 1          ; al = 10000000b   cf=1
ror al, 1          ; al = 01000000b   cf=0
```

Multiple Rotations : when using a rotation count greater than 1, cf contains the last bit rotated out of the LSB position.

```
mov al, 00000100b
ror al, 3          ; al = 10000000   cf=1
```

7.2.7. RCL and RCR Instructions

RCL : rotate carry left instruction shifts each bit to the "LEFT", copies the carry flag to the LSB and copies the MSB into the carry flag.



Signature

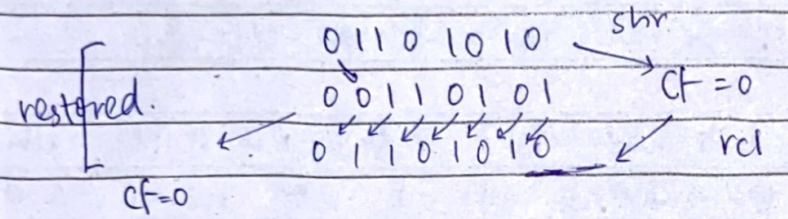
No.

Date

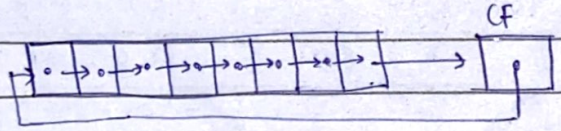
example:
```
clc                 ; cf = 0
mov bl, 88h         ; bl = 1000 1000b
rcl bl, 1           ; cf = 1, bl = 00010000 prev cf
                      msb of prev bl val.   rotated left
```

RECOVER A BIT FROM THE CARRY FLAG

```
.data
testval byte 01101010b
.code
shr testval, 1      ; shifts LSB → carry flag
jc exit             ; exit if cf = 1
rcl testval, 1      ; else restore the number
```



RCR : rotate carry right instruction shifts each bit to the right, copies the carry flag into the MSB, and copies the LSB into CF



example:
```
stc                 ; cf = 1
mov ah, 10h         ; ah = 00010000b
rcr ah, 1           ; ah = 10001000 b
                                cf = 0
```

7.2.8 SIGNED OVERFLOW

The overflow flag is set when shifting or rotating a signed integer by one bit position generates a value outside the signed integer range of the destination operand.

- Number's sign is reversed.

Date

example #01: mov al, +127 ; al = 01111111b (+127)
rol al, 1 ; al = 11111110b (-2)
OF = 1

example #02: mov al, -128 ; al = 10000000b
shr al, 1 ; al = 01000000b
OF = 1

THE VALUE OF OVERFLOW FLAG IS UNDEFINED WHEN THE SHIFT OR ROTATION IS GREATER THAN 1.

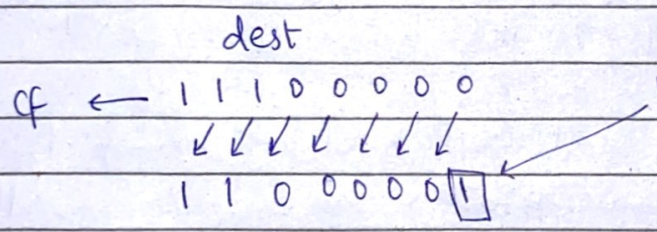
7.2.9 SHLD / SHRD Instructions

SHLD: The shift left double instruction shifts a destination operand a given number of bits to the left.

- Source operand is not affected.
- Sign, zero, parity, carry flags are affected.

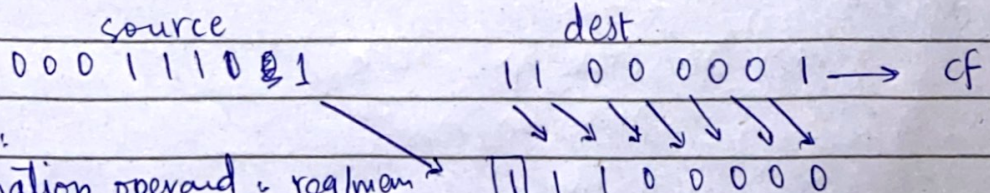
Syntax: shld dest, source, count

example: shld with a shift count of 1



SHRD: The shift right double instruction shifts a destination operand a given number of bits to the right.

shrd dest, source, count



NOTE:
destination operand: reg/mem
source operand: reg
count operand: cl reg / 8 bit imm operand.

Signature _____ No. _____

Date

7.4 Multiplication and Division

The MUL and IMUL instruction performs unsigned and signed integer multiplication respectively.

7.4.1 MUL INSTRUCTION:

```
mul reg/mem 8
mul reg/mem16
mul reg/mem32
```

- The multiplier and multiplicand must always be the same size, and the product is twice their size.
- The single operand in the MUL is the multiplier
- overflow cannot occur, because the dest. operand is twice.
- MUL sets the cf and of if the upper half of the product ≠ 0

Multiplicand	MULTIPLIER	PRODUCT
AL	reg/mem8	AX
AX	reg/mem16	DX : AX
EAX	reg/mem32	EDX : EAX

example #01: The following statement multiply al by bl, storing product in ax. CF = 0, because AH (the upper half = 0)

```
mov al, 5h
mov bl, 10h
mul bl
```

AL		BL		AX	CF
05	x	10	→	0050	0

because upper half is equals to zero.

example #02

```
mov ax, 2000h
mov bx, 0100h
mul bx
```

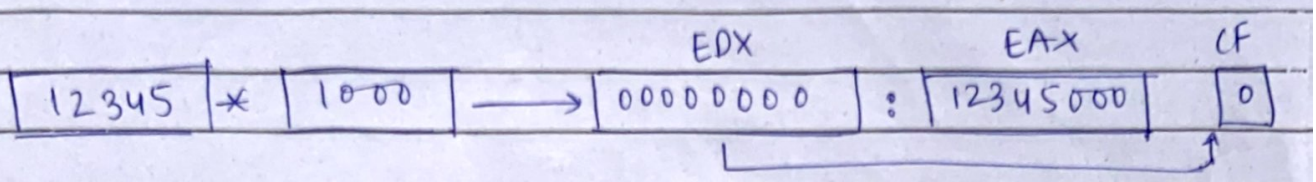
AX		BX		DX	:	AX	CF
2000h	*	0100	→	0020	:	0000	1

Signature _____ RS No. _____

Date

example #03:

```
mov eax, 12345h
mov ebx, 1000h
mul ebx
```



7.4.2 IMUL INSTRUCTION: signed multiplication
↳ preserves the sign of the product.

ONE OPERAND:

```
imul reg/mem8      ; AX = AL * reg/mem8
imul reg/mem16     ; DX:AX = AX * reg/mem16
imul reg/mem32     ; EDX:EAX = EAX * reg/mem32
```

- The carry and overflow flags are set if the upper half of the product is not a sign extension of the lower half.
↳ you can use this information to decide whether to ignore the upper half or not.

example #01:

```
mov al, 48
mov bl, 4
imul bl        ; ax = 00C0, OF=1
```

because ah is not the sign extension of al. hence of=1

example #02:

```
mov al, -4
mov bl, 4
imul bl        ; ax = FFF0h, OF=0
```

example #03: OF=0 because DX is the sign extension of AX.

```
mov ax, 48
mov bx, 4
imul bx        ; DX:AX = 000000C0h, OF=0
```

Signature _____ RG No. _____

Date

IMUL INSTRUCTION: signed multiplication

ONE OPERAND:

imul reg/mem8	ax = al * reg/mem8
imul reg/mem16	dx:ax = ax * reg/mem16
imul reg/mem32	edx:eax = eax * reg/mem32

"THE CF & OF ARE SET IF THE UPPER HALF OF THE PRODUCT IS NOT A SIGN EXTENSION OF THE LOWER HALF."

- you can use this information to decide whether to ignore the upperhalf of the product.

TWO OPERANDS: destination must be a "REGISTER."

~~imul reg16/mem16~~

imul reg16, reg/mem16
imul reg16, imm8
imul reg16, imm16

imul reg32, reg/mem32
imul reg32, imm8
imul reg32, imm32

THREE OPERANDS:

imul reg16, reg/mem16, imm8
imul reg16, reg/mem16, imm16

For 32:

imul reg32, reg/mem32, imm8
imul reg32, reg/mem32, imm32

Date

DIV Instruction : unsigned division

syntax :

```
div reg/mem8
div reg/mem16
div reg/mem32
```

Dividend	Divisor	Quotient	Remainder
AX	reg/mem8	al	ah
DX : AX	reg/mem16	ax	dx
EDX : EAX	reg/mem32	eax	edx

Div Examples :

8 bit unsigned division

```
mov ax, 0083h      ; dividend
mov bl, 2          ; divisor
div bl             ; al = 41h , ah = 01h
                   ↓ quotient   ↳ remainder
```

16 bit unsigned division

```
mov dx, 0
mov ax, 8003h
mov cx, 100h
div cx
```

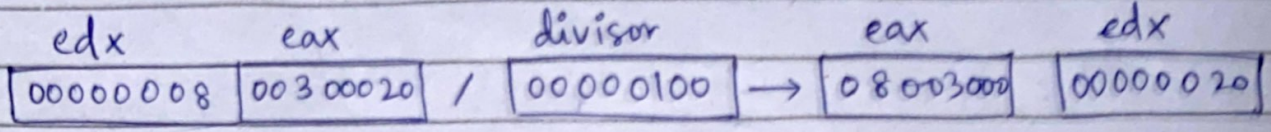
DX	AX		CX		AX	DX
0000	8003	/	0100	→	0080	0003
					quotient	remainder

32 bit unsigned division

```
.data
dividend QWORD 0000000800300020h
divisor dword 00000100h
```

Signature _____ No. _____

```
.code
mov edx, dword ptr dividend + 4
mov eax, dword ptr dividend
div divisor
```



Signed Integer Division:

almost identical to unsigned division.

- The dividend must be fully sign-extended before the division takes place.

SIGN EXTENSION INSTRUCTIONS:

- CBW : convert byte to word
- CWD : convert word to double word
- CDQ : convert double word to quad word

CBW instruction extends the sign bit of AL into AH, preserving the number's sign.

example:

```
.data
byteVal sbyte -101
.code
mov al, byteval          al = 9Bh
cbw                      ax = FF9Bh
```

CWD:

```
.data
wordval sword -101       ; FF9Bh
.code
mov ax, wordval          ; AX = FF9Bh
cwd                      ; DX:AX = FFFFFF9Bh
```

Date

CDQ.

```
.data
dwordVal sdword -101          ; FFFFFF9Bh
.code
mov eax, dwordVal             ; eax = FFFFFF9Bh
cdq                           ; edx:eax = FFFFFFFFFFFFFF9Bh
```

The IDIV instruction :

- Before executing 8-bit division, the dividend ax must be completely sign extended.
- "THE REMAINDER ALWAYS HAS THE SAME SIZE AS DIVIDENDS."

example #01 : 8 bit

```
.data
byteval sbyte -48
.code
mov al, byteval
cbw
mov bl, +5
idiv bl
```

example #02 : 16 bit

```
.data
wordval sword -5000
.code
mov ax, wordval
cwd
mov bx, +256
idiv bx
```

example #03 : 32 bit

```
.data
dwordval sdword +5000
.code
mov eax, dwordval
cdq
mov ebx, -256
idiv ebx
```

"USE A 32 BIT DIVISOR AND 64 BIT DIVIDEND TO REDUCE THE PROBABILITY OF A DIVIDE OVERFLOW CONDITION."

example:

```
mov eax, 1000h
cdq
mov ebx, 10h
div ebx
```

Date

Extended Addition and Subtraction

ADC : Add with carry

- adds both a source operand and the contents of the carry flag to a destination operand.

example :

```
mov dl, 0
mov al, 0FFh
add al, 0FFh          ; AL = FEh
adc dl, 0             ; DL/AL = 01FEh
```

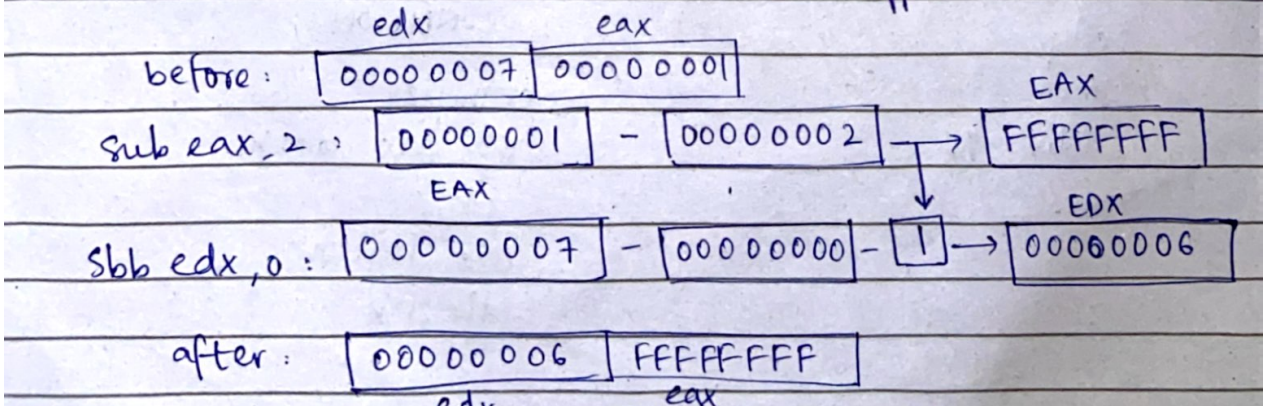
example :

```
mov edx, 0
mov eax, 0FFFFFFFFh
add eax, 0FFFFFFFFh
adc edx, 0            edx : eax = 00000001FFFFFFFEh
```

SBB : subtract with borrow

- subtracts both a source operand and the value of the carry flag from a destination operand.

```
mov edx, 7            ; upper half
mov eax, 1            ; lower half
sub eax, 2            ; subtract 2
sbb edx, 0            ; subtract upper half
```



Signature _____ RS No. _____