Multiplication and Division Instructions

- MUL Instruction
- IMUL Instruction
- DIV Instruction
- Signed Integer Division (IDIV Instruction)
- Implementing Arithmetic Expressions
- Decimal Input & Output

MUL Instruction

- The MUL (unsigned multiply) instruction multiplies an 8/16 bit operand by AL/AX

- The instruction formats are:

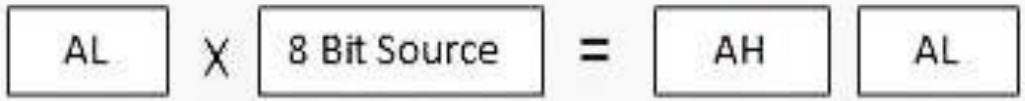
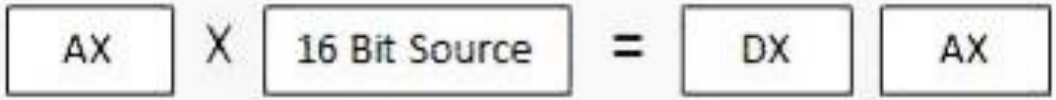
```
MUL multiplier
```

- Multiplier can be register or memory

Implied operands:

Multiplicand	Multiplier	Product
AL	*r/m8*	AX
AX	*r/m16*	DX:AX

MUL Instruction

SN	Scenarios
1	**When two bytes are multiplied** The multiplicand is in the AL register, and the multiplier is a byte in the memory or in another register. The product is in AX. High order 8 bits of the product is stored in AH and the low order 8 bits are stored in AL 
2	**When two one-word values are multiplied** The multiplicand should be in the AX register, and the multiplier is a word in memory or another register. For example, for an instruction like MUL DX, you must store the multiplier in DX and the multiplicand in AX. The resultant product is a double word, which will need two registers. The High order (leftmost) portion gets stored in DX and the lower-order (rightmost) portion gets stored in AX. 

IMUL Instruction

- IMUL (signed integer multiply) multiplies an 8/16 bit signed operand by AL/AX
- Preserves the sign of the product by sign-extending it into the upper half of the destination register
- The instruction formats are:

  `IMUL multiplier`

- Multiplier can be register or memory

MUL & IMUL Examples

2's complement

MOV AL, 80h ; AL = -128D
MOV BL, FFh ; BL = -1D

Instructions	Decimal Product	Hex Product	AH	AL
MUL BL	~~128~~	7F80	7F	80
IMUL BL	128	0080	00	80

MUL: 128X255=32640=7F80

MUL & IMUL Examples

MOV AX, 1
MOV BX, ~~FFFFh~~ 0FFFFh ; BX = 65535D/ -1D

Instructions	Decimal Product	Hex Product	DX	AX
MUL BX	65535	0000FFFF	0000	FFFF
IMUL BX	-1	FFFFFFFF	FFFF	FFFF

MUL & IMUL Examples

MOV AX,0FFFFh ; AX =65535D/-1
MOV BX,0FFFFh ; BX =65535D/-1

Instructions	Decimal Product	Hex Product	DX	AX
MUL BX	4294836225	FFFF0001	FFFF	0001
IMUL BX	1	00000001	0000	0001

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute?

```
MOV AX,0FFFh
MUL AX
IMUL AX
```

DX = ?
AX = ?

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute?

```
MOV AX,0100h
MOV CX,FFFFh
MUL CX
IMUL CX
```

MUL

DX = ? 00FF

AX = ? FF00

IMUL

DX: FFFF

AX: FF00

Your turn . . .

What will be the hexadecimal values of DX and AXg after the following instructions execute?

```
MOV AX,8760h
MOV BX,100h
MUL BX
IMUL BX
```

MUL

DX = ? 0087

AX = ? 6000

IMUL

DX: FF87

AX: 6000

Your turn . . .

- Translate the following high level language assignment into assembly Language. Let A and B are word variables.

A = 5A-12B

- Calculate factorial of N

N! =N*(N-1)*(N-2)*……*1

```
Algorithm:
factorial = 1  (initialization)
input: N
for N times do
    factorial = factorial * N
    N = N - 1 (loop instruction)
end for
```

DIV Instruction

- The DIV (unsigned divide) instruction performs 8-bit and 16-bit division on unsigned integers
- Instruction formats:
  - **DIV *divisor***
- Divisor can be register or memory operand

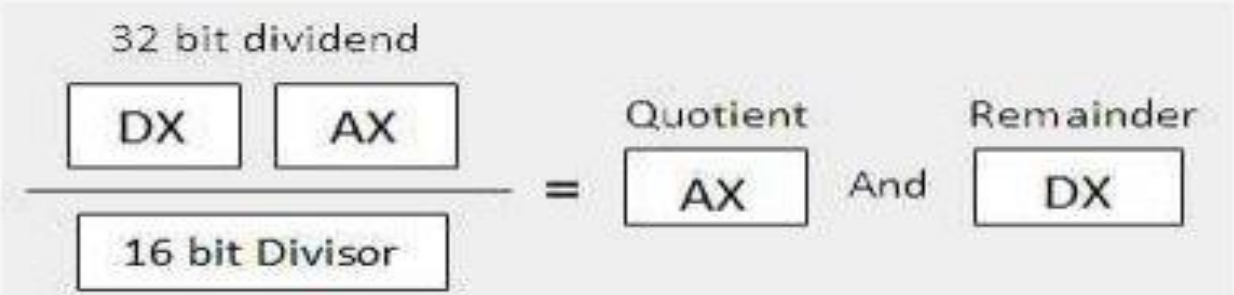
DIV *r/m16*
DIV *r/m32*

Default Operands:

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX

DIV Instruction

SN	Scenarios
1	**When the divisor is 1 byte** The dividend is assumed to be in the AX register (16 bits). After division, the quotient goes to the AL register and the remainder goes to the AH register.
2	**When the divisor is 1 word** The dividend is assumed to be 32 bits long and in the DX:AX registers. The high order 16 bits are in DX and the low order 16 bits are in AX. After division, the 16 bit quotient goes to the AX register and the 16 bit remainder goes to the DX register.



IDIV Instruction

- IDIV (signed divide) performs signed integer division
- Instruction formats:

  IDIV *divisor*

- Divisor can be register or memory operand
- Uses same operands as DIV

DIV & IDIV Examples

```
MOV DX, 0000
MOV AX, 0005
MOV BX, 0002
```

Instructions	Decimal Quotient	Decimal remainder	AX	DX
DIV BX	2	1	0002	0001
IDIV BX	2	1	0002	0001

DIV & IDIV Examples

```
MOV DX, 0000h
MOV AX, 0005h
MOV BX, 0FFFEh ; BX=65534D/-2D
```

Instructions	Decimal Quotient	Decimal remainder	AX	DX
DIV BX	0	5	0000	0005
IDIV BX	-2	1	FFFE	0001

DIV & IDIV Examples

MOV AX, 00FBh ; AX=251D
MOV BL, FFh ; BL =256D/-1D

Instructions	Decimal Quotient	Decimal remainder	AH	AL
DIV BL	0	251	FB	00
IDIV BL	-251(Too Big)	Divide Overflow		

DIV & IDIV Examples

MOV DX, FFFFh
MOV AX, FFFBh ; DX:AX=FFFFFFFB=4294967291D/-5D
MOV BX, 0002h

Instructions	Decimal Quotient	Decimal remainder	AX	DX
DIV BX	2147483646/7FFFFFFEh (Too Big)	Divide Overflow		
IDIV BX	-2	-1	FFFE	FFFF

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
MOV DX,0087H
MOV AX,6000H
MOV BX,100H
DIV BX
```

DX = ?, AX = ?

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
MOV DX,0087H
MOV AX,6002H
MOV BX,10H
DIV BX
```

?????

Signed Integer Division

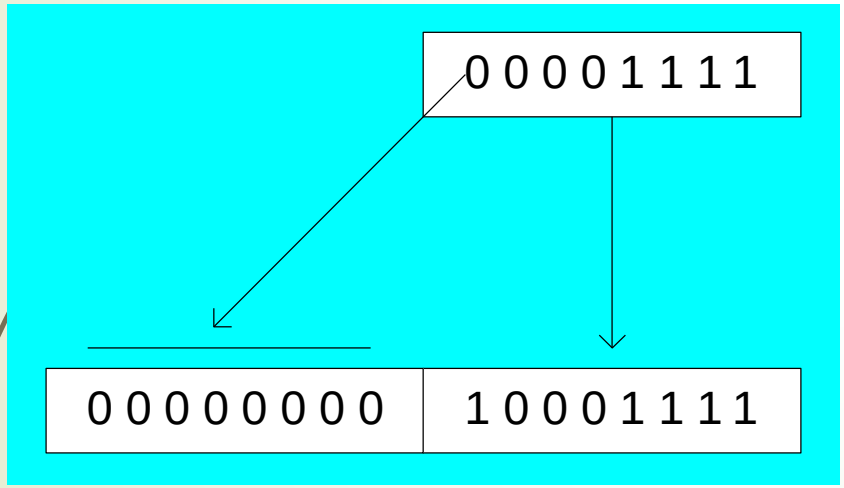
- Signed integers must be sign-extended before division takes place
  - fill high byte/word with a copy of the low byte/word's sign bit

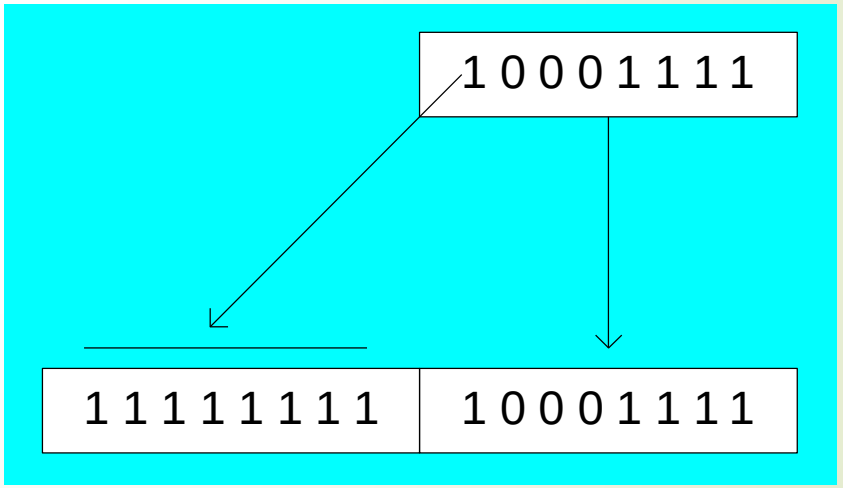
- Byte Division
  - For DIV, AH should be cleared
  - For IDIV, AH should be made the sign extension of AL. The instruction CBW (convert byte to word) will do this extension.

- Word Division
  - For DIV, DX should be cleared
  - For IDIV, DX should be made the sign extension of AX. The instruction CWD (convert word to doubleword) will do this extension.

Signed Integer Division

- For example, the high byte contains a copy of the sign bit from the low byte:





CBW, CWD Instructions

- The CBW and CWD instructions provide important sign-extension operations:
  - CBW (convert byte to word) extends sign bit of AL into AH
  - CWD (convert word to doubleword) extends sign bit of AX into DX
- For example:

```
MOV AX,0FFFBH
CWD             ; DX:AX = FFFFFFFBh
```

CBW & CWD Examples

Example 1:

MOV AL, -7
CBW
MOV BL, -7
IDIV BL

AL = 01 , AH = 00

Example 2:

MOV AX, -50
CWD
MOV BX, 7
IDIV BX

AX = FFF9h (-7) , DX = FFFFh (-1)

Implementing Arithmetic Expressions

Example: **var4 = (var1 + var2) * var3**

```
MOV AX,VAR1
ADD AX,VAR2
MUL VAR3
MOV VAR4,AX          ; save product
```

Example: **eax = (-var1 * var2) + var3**

```
MOV AX,VAR1
NEG AX
MUL VAR2
ADD AX,VAR3
```

Implementing Arithmetic Expressions

Example: **`var4 = (var1 * 5) / (var2 - 3)`**

```
MOV AX,VAR1                ; left side
MOV BX,5
MUL BX                     ; DX:AX = product
MOV BX,VAR2                ; right side
SUB BX,3
DIV BX                     ; final division
MOV VAR4,AX
```

Implementing Arithmetic Expressions

Example: `var4 = (var1 * -5) / (-var2 % var3);`

```
MOV  AX,VAR2              ; begin right side
NEG  AX
CWD                       ; sign-extend dividend
IDIV VAR3                 ; DX = remainder
MOV  BX,DX                ; BX = right side
MOV  AX,-5                ; begin left side
IMUL VAR1                 ; DX:AX = left side
IDIV BX                   ; final division
MOV  VAR4,AX              ; quotient
```

Your turn . . .

Implement the following expression. Do not modify any variables other than var3:

```
var3 = (var1 * -var2) / (var3 – ebx)
```

Decimal Input

- **Example: Input of 123**
- **number = 0**
- **Read '1'**
- Convert '1' to 1
- **number = 0*10 +1 =1**
- **Read '2'**
- Convert '2' to 2
- **number = 0*10 +2 =12**
- **Read '3'**
- Convert '3' to 3
- **number = 12*10 +3 =123**

```
Algorithm for Decimal Input

    number = 0
    while input != "\n" do
        number = number * 10
        input a charcter
        number = number + input
        count++
```

Decimal Output

```
Algorithm for Decimal Output

    for i = 1 to count do
        reminder = number % 10
        push reminder
    for i = 1 to count do
        pop reminder
        print reminder
```

Any Query?