Assembly Arithmetic Instructions – Detailed Notes

1. MUL – Unsigned Multiply

◇ Purpose:

- Performs **unsigned multiplication** between an implicit accumulator and a specified operand.

◇ Operand Type:

- Only **one operand** is provided (source).
- The **accumulator register** is **implied**:
  - 8-bit → AL
  - 16-bit → AX
  - 32-bit → EAX

◇ Result Storage:

Operand Size	Implied Operand	Result Stored In
8-bit	AL	AX
16-bit	AX	DX:AX
32-bit	EAX	EDX:EAX

◇ Affected Flags:

- CF (Carry Flag) and OF (Overflow Flag):
  - Set if **upper half** of result ≠ 0
  - Cleared if **upper half** of result = 0

◇ Examples:

8-bit:

```
asm
CopyEdit
mov al, 5
mov bl, 6
mul bl       ; AX = 30 (5 * 6)
```

16-bit:

```
asm
CopyEdit
mov ax, 300h
mov bx, 100h
mul bx       ; DX:AX = 768 * 256 = 196608
```

32-bit:

```
asm
CopyEdit
mov eax, 10000h
mov ebx, 2
mul ebx      ; EDX:EAX = 65536 * 2 = 131072
```

2. IMUL – Signed Multiply

◇ Purpose:

- Performs **signed multiplication** and preserves **sign information**.

◇ Operand Formats:

Format	Behavior
One-Operand	Similar to `MUL`, result goes to AX/DX:AX/EDX:EAX
Two-Operand	Result is truncated and stored in first operand (a register)
Three-Operand	Multiplies second & third, stores in first operand (a register)

◇ Important:

◆ Two- and Three-Operand Forms Do NOT Use DX or EDX

- The result is **truncated** to the size of the destination register.
- Overflow (OF) and carry (CF) flags are set **if truncation occurs** (i.e., if result is too large).

◇ Result Storage:

Format	Result Location
One Operand	AX / DX:AX / EDX:EAX
Two Operand	First operand register (truncated)
Three Operand	First operand register (truncated)

◇ Examples:

One Operand (Signed 16-bit):

```
asm
CopyEdit
mov ax, -48
imul word ptr [num]   ; DX:AX = -48 * num
```

Two Operand:

```
asm
CopyEdit
mov ax, 5
imul bx, ax           ; BX = BX * AX (result stored in BX)
```

Three Operand:

```
asm
CopyEdit
imul eax, ebx, 4      ; EAX = EBX * 4
```

◇ Affected Flags:

- CF and OF set if **sign extension is not preserved** (i.e., result too large).
- Cleared if the **upper half is a proper sign extension** of lower half.

3. DIV – Unsigned Division

◇ Purpose:

- Performs **unsigned integer division**.

◇ Operand Type:

- One operand (the divisor).
- **Dividend** is always stored in **implied registers**.

◇ Operand Sizes & Registers:

Operand Size	Dividend	Quotient	Remainder
8-bit	AX	AL	AH
16-bit	DX:AX	AX	DX
32-bit	EDX:EAX	EAX	EDX

◇ Flags:

- None affected
- CPU **interrupts** on:
  - **Divide-by-zero**
  - **Quotient overflow**

◇ Examples:

8-bit:

```
asm
CopyEdit
mov ax, 50
mov bl, 5
div bl            ; AL = 10, AH = 0
```

16-bit:

```
asm
CopyEdit
mov dx, 0
mov ax, 0300h     ; 768
mov bx, 100h      ; 256
div bx            ; AX = 3, DX = 0
```

32-bit:

```asm
CopyEdit
mov edx, 0
mov eax, 10000h  ; 65536
mov ebx, 256
div ebx          ; EAX = 256, EDX = 0
```

4. IDIV – Signed Division

◇ Purpose:

- Performs **signed integer division**.

◇ Operand Type:

- One operand (divisor).
- Dividend must be **sign-extended** beforehand.

◇ Sign-Extension Instructions:

Instruction	Use Case	Purpose
`CBW`	8-bit AL → AX	Sign-extend AL into AH
`CWD`	16-bit AX → DX:AX	Sign-extend AX into DX
`CDQ`	32-bit EAX → EDX:EAX	Sign-extend EAX into EDX

◇ Operand Sizes & Registers:

Operand Size	Dividend	Quotient	Remainder
8-bit	AX	AL	AH
16-bit	DX:AX	AX	DX

32-bit	EDX:EAX	EAX	EDX

◇ Flags:

- None affected
- **Divide Overflow** or **Divide-by-Zero** causes **CPU exception**

◇ Examples:

8-bit:

```
asm
CopyEdit
mov al, -48
cbw               ; Sign-extend AL → AX
mov bl, 5
idiv bl           ; AL = -9, AH = -3
```

16-bit:

```
asm
CopyEdit
mov ax, -300
cwd               ; Sign-extend AX → DX
mov bx, 10
idiv bx           ; AX = -30, DX = 0
```

32-bit:

```
asm
CopyEdit
mov eax, -100000
cdq               ; Sign-extend EAX → EDX
mov ebx, 250
```

```
idiv ebx          ; EAX = -400, EDX = 0
```

Final Comparison Table

Instruction	Type	Operands	Signed	Implied Registers	Result Stored In
`MUL`	Multiply	1	No	AL/AX/EAX	AX / DX:AX / EDX:EAX
`IMUL`	Multiply	1	Yes	AL/AX/EAX	AX / DX:AX / EDX:EAX
`IMUL`	Multiply	2/3	Yes	None	Destination register (truncated)
`DIV`	Divide	1	No	AX/DX:AX/EDX:EAX	AL:AH / AX:DX / EAX:EDX
`IDIV`	Divide	1	Yes	AX/DX:AX/EDX:EAX	AL:AH / AX:DX / EAX:EDX