



Assembly Language Programming

1- What is Assembly Language

The Assembly Language is the Bridge Between Hardware and the Code. Assembly instructions, written in form (like MOV AX, 5), are translated by the assembler into binary machine code that the CPU can directly execute. Each assembly instruction consists of two main parts: the **opcode** and the **operand**. Example, in the instruction **MOV AX, 5**, the **Opcode** is **MOV** and the **Operands** is **register AX** and **5**. The entire group of instructions that a microprocessor support is called Instruction Set.

- The 8086 Implementation:

- The instruction set of the 8086 microprocessor contains different arithmetic instructions, operations, using Signed, Unsigned decimal bytes, or **ASCII numbers**.
- The 8086-instruction set includes operations that can move between memory and registers, compare, and manipulate both data and instructions in memory.
- The instruction set allows reading and writing to any memory location, regardless of whether it holds data or code.

2- Registers: The **8086** microprocessor CPU contain 16-bit Registers that are very small, very fast storage units, used for different purposes. Registers type are:

1 General-Purpose Registers: Used for arithmetic, logic, and data movement operations. They can hold data, results, or memory addresses. The 8086 registers:

- **AX (Accumulator):** consist of (Ah, AL) used for arithmetic operations.
- **BX (Base):** consist of (BH, BL) used for addressing memory.
- **CX (Count):** consist of (CH, CL) used for loops or shift instructions.
- **DX (Data):** consist of (DH, DL) used in multiplication/division and I/O operations.

□ **Index Registers:** used for indexed addressing also for string and arithmetic.
Index Registers include SI, DI:



- SI (Source Index) used for source **addressing** in string and array operations
- DI (Destination Index) used for destination **addressing** in string and array operations

2 Segment Registers: it's a 16-bit registers used to create memory addresses. Each one points to the starting address of a 64KB memory segment, each segment register has a specific job in organizing the computer's memory:

☐ **Code Segment (CS):** Points to the memory location where the program's executable **instructions** are stored. It works with the **Instruction Pointer (IP)** to tell the CPU where to fetch the next instruction.

☐ **Data Segment (DS):** Points to the location of the program's **data**. It is used by default to access variables and other data stored in memory.

☐ **Stack Segment (SS):** Points to the starting address of the **stack**, which is a special memory area used for temporary data storage, like local variables and return addresses for function calls. It works with the **Stack Pointer (SP)**.

☐ **Extra Segment (ES):** An additional data segment that provides another memory area for data. Its most common use is as a destination segment for string and array operations.



Tip: ES, FS, and GS registers are used in 32-bit and 64-bit processors

3 Special-Purpose Registers: hold information that CPU needs to control operations.

☐ **PC / IP (Program Counter or Instruction Pointer):** Holds the address of the next instruction to be fetched from memory.

☐ **IR (Instruction Register):** Holds the current instruction being decoded and executed.

☐ **MAR (Memory Address Register):** Holds the address of memory location to read from or write to.

☐ **MDR (Memory Data Register):** Holds the data being transferred to or from memory.

☐ **Accumulator (A or AX):** Sometimes considered both general- and special-purpose, as it's heavily used by the ALU.

☐ **Stack-Related Registers:** Used to manage function calls and temporary storage.



- **SP (Stack Pointer)**: points to the top of the stack in memory.
- **BP (Base Pointer)**: used to access function parameters and local variables on the stack.

4 Status or Flag Registers: Hold flags (bits) that show the results of operations or CPU states. The 16-bit register contain two types:

□ **Control Flags (DF, IF, TF)**, directly control a specific function of the processor ex. enable/disable interrupt, it's not affected by the ALU.

- **Direction Flag (DF)**: Controls the direction of string processing.
 - If DF = 0, strings are processed from low memory to high (auto-increment SI and DI).
 - If DF = 1, strings are processed from high memory to low (auto-decrement SI and DI).
- **Interrupt Enable (IE)**: Controls the CPU's response to maskable hardware interrupts. case IE = 1 → CPU accepts interrupts, else ignores interrupts.
- **Trap Flag (TF)**: Enables single-step mode for debugging. If TF = 1, the CPU executes one instruction at a time and triggers an interrupt after each instruction, else (TF = 0), CPU executes instructions normally.

□ **Status or Condition Flags (ZF, SF, PF, CF, AF, OF)** . It holds the result's properties of an arithmetic or logical operation

- The **Zero** flag the **ZF** if a result is **Zero**- the **ZF** is set to 1.
- The **Sign-SF** set to 1 if the operation result is **negative** and 0 if it's **positive**.

Tip: Signed or Unsigned **number** depend on the program or the instruction.

- The **Parity Flag (PF)**, depends on the number of (1) bits in the lowest 8 bits. PF set to 1 if the parity is **even** and reset to 0 if it's **odd**. Its main purpose is for checking data, often used in communications.

Tip:
Bit number: 7 6 5 4 3 2 1 0
Value: 1 0 0 0 1 0 1 1
MSB **LSB**

Example: the lowest 8 bits are 0000 1011. There are **three** 1s, which is an **odd** number, so the **PF=0**.



- The **Carry (CF)**: indicates a carry or borrow out of the **most significant bit (MSB)** of the result, CF work with **Unsigned** numbers, for an 8-bit operation, a carry from bit 7 sets the CF.

$$\begin{array}{r} 1111\ 1111\ (255) + \\ 0000\ 0001\ (1) \\ \hline \end{array}$$

Example: Adding two 8-bit **binary** numbers: Results in 1 0000 0000 (0) 100000000b. The true result = 256 But the 8-bit register can

only hold 0–255 so the result = 0 for unsigned number, the Carry Flag (CF) set to 1 mean the value is overflow, means actual result was **256**.

Tip:: think of the Carry Flag as the 9th bit of the result.

- The **Auxiliary (AF)**: flag is set if a **Carry-out** or borrow-in occurs from the **lower 4 bits** (the least significant nibble) to the **upper 4 bits** of the result during an arithmetic operation. Example on **Auxiliary** state On Addition operation

$$\begin{array}{r} 0000\ \underline{0}\ 1001\ (9) + \\ 0000\ 0111\ (7) \\ \hline 0001\ 0000\ (16) \end{array}$$

- The **Overflow Flag (OF)** ⚠ work when using **Signed** number, if arithmetic operation (e.g. adding two positive numbers or two negative numbers) produced a result that is too large or too small to fit in the destination register. In example: $72 + 80 = 152 \rightarrow$ cannot represent in 8-bit. if the

$$\begin{array}{r} 0100\ 1000\ (72) + \\ 0101\ 0000\ (80) \\ \hline \underline{1}001\ 1000\ (-104) \\ \text{(in two's complement)} \end{array}$$

number is set as signed and MSB= 1, mean the value is a negative number (represented in two's complement) = (-104). **therfor The OF = 1.**

Tip:: The CPU represent the Signed Negative number in two's complement

Tip:: Signed 8-Bit produce 128 number rang (0-127), because MSB (8th bit = position 7) used for the Signed, while Unsigned number rang (0-255) where $2^8=256$ binary number.

Tip: the 32-bit registers: **EAX, EBX, ECX, EDX, ESI, EDI, ESP,EBP**

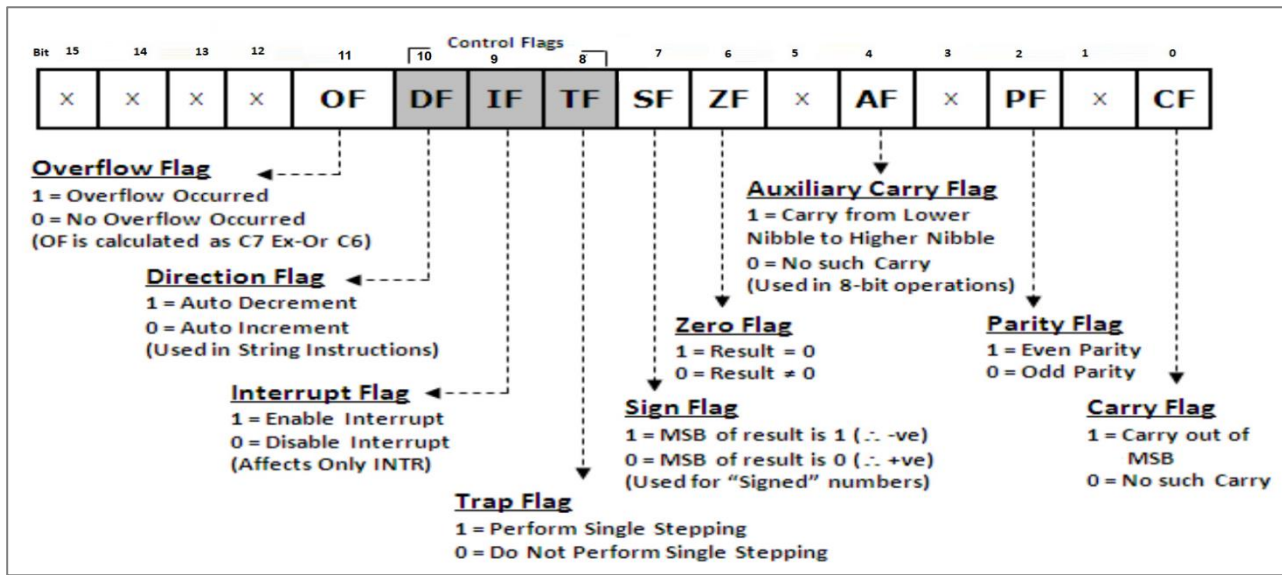
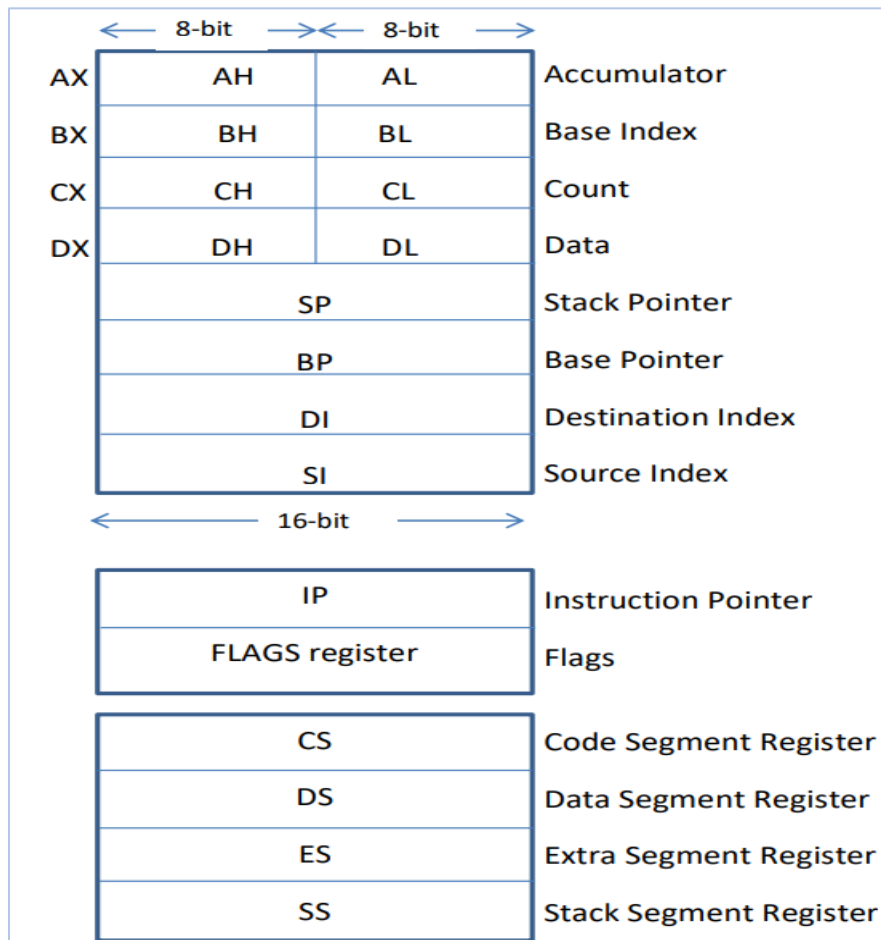


Figure of CPU general Register and Segment registers used for memory addressing





3- The general types of assembly instruction and CPU operation:

The 8086 microprocessor supports 8 types of instructions

- 1- Data Transfer Instructions
- 2- Arithmetic Instructions
- 3- Bit Manipulation Instructions
- 4- String Instructions
- 5- Branch Instructions
- 6- Processor Control Instructions
- 7- Loop & Iteration Instructions
- 8- Interrupt Instructions

1. Data Transfer Instructions:

These instructions are used for data transfer, port I/O, address loading, and flag manipulation. It includes the instructions:

A. Instructions to transfer data

- MOV: Copies data from the source to the destination.
- PUSHA / POPA: Push all general-purpose registers to the stack / pop from the stack.
- XCHG: Exchanges data between two registers or between a register and memory.
- XLAT: Translates a byte in AL using a lookup table in memory.

B. Instructions for input/output port transfer

- IN: Reads a byte or word from an input port into the accumulator (AL or AX).
- OUT: Sends a byte or word from the accumulator to an output port.

C. Instructions to transfer addresses

- LEA: Loads the effective address of the operand into a register.
- LDS: Loads the DS register and another specified register from memory.



- LES: Loads the ES register and another specified register from memory.

D. Instructions to transfer flag register

- LAHF: Loads the low byte of the flag register into AH.
 - SAHF: Stores the contents of AH into the low byte of the flag register.
 - PUSHF: Pushes the flag register onto the stack.
 - POPF: Pops a word from the stack into the flag register.
-

2. Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc. Following is the list of instructions under this group:

A. Addition instructions

- ADD: Adds two values.
- ADC: Adds two values plus the carry flag.
- INC: Increases a value by 1.
- AAA: Adjusts result after ASCII addition.
- DAA: Adjusts result after decimal (BCD) addition or subtraction.

B. Subtraction instructions

- SUB: Subtracts one value from another.
- SBB: Subtracts with borrow (including carry).
- DEC: Decreases a value by 1.
- NEG: Negates a value (two's complement).
- CMP – Used to compare 2 provided byte/word.
- AAS – Used to adjust ASCII codes after subtraction.



- DAS – Used to adjust decimal after subtraction.

C: Instruction to perform multiplication

- MUL – Used to multiply unsigned byte by byte/word by word.
- IMUL – Used to multiply signed byte by byte/word by word.
- AAM – Used to adjust ASCII codes after multiplication.

D. Division Instructions

- DIV: Divides an **unsigned** number. For example, divides a word by a byte, or a double word by a word.
- IDIV: Divides a **signed** number (handles positive and negative values).
- AAD: Adjusts the result after division of ASCII values.
- CBW: Converts byte in AL to word in AX by **sign** extension (copies sign bit of AL to AH).
- CWD: Converts word in AX to double word in DX:AX by **sign** extension (copies sign bit of AX to DX).

Example on Div Operation :

MOV AX, 1000h ; AX = 1000h (dividend)

MOV BL, 10h ; BL = 10h (divisor)

DIV BL ; ; Divide AX by BL , the Result goes to AL ; Remainder goes to AH

3. Bit Manipulation Instructions

These instructions work with individual bits in data, like logical and shift operations. It includes:

A. Logical Instructions

- NOT: Flips every bit (1 becomes 0, 0 becomes 1).
- AND: Compares bits of two values; result bit is 1 only if both bits are 1.
- OR: Compares bits of two values; result bit is 1 if either bit is 1.
- XOR: Compares bits of two values; result bit is 1 if bits are different.



- TEST: Performs AND on two values but only sets the flags; data stays unchanged.

B. Shift Instructions

- SHL / SAL: Shifts bits left; fills from right with 0.
- SHR: Shifts bits right; fills from left with 0.
- SAR: Shifts bits right; fills from left with the original leftmost bit (keeps sign for signed numbers).

C: Instructions to perform rotate operations

- ROL (Rotate Left): Moves bits left. The leftmost bit (MSB) moves to the rightmost bit (LSB) and also to the Carry Flag (CF).
- ROR (Rotate Right): Moves bits right. The rightmost bit (LSB) moves to the leftmost bit (MSB) and also to CF.
- RCL (Rotate through Carry Left): Like ROL but includes the Carry Flag in rotation, shifting CF into LSB.
- RCR (Rotate through Carry Right): Like ROR but includes the Carry Flag, shifting CF into MSB.

Example of ROL (Rotate Left)

MOV AL, 10010011b ;AL = 93h (binary)

ROL AL, 1 ; Rotate bits left by 1

Result: ; AL = 00100111b (27h) , CF = 1 (bit shifted out from MSB)

4. String Instructions

String is a group of bytes/words and allocated in memory in a sequential order,

Common instructions:

- REP: Repeat the next instruction while CX $\neq$ 0.
- REPE/REPZ: Repeat while CX $\neq$ 0 and zero flag (ZF) = 1.
- REPNE/REPNZ: Repeat while CX $\neq$ 0 and zero flag (ZF) = 0.
- MOVS/MOVSb/MOVSsw: Copy a byte/word from one string to another.
- CMPS/CMPSb/CMPSsw: Compare bytes/words from two strings.



- INS/INSB/INSW: Input byte/word from an I/O port to memory.
- OUTS/OUTSB/OUTSW: Output byte/word from memory to an I/O port.
- SCAS/SCASB/SCASW: Scan a string comparing each byte/word with AL/AX.
- LODS/LODSB/LODSW: Load byte/word from string into AL/AX.

Example: Using REP with MOVSB, to copies 10 bytes from the SOURCE string to the DEST string

MOV CX, 10	; Number of bytes to copy
MOV SI, SOURCE	; Source string offset
MOV DI, DEST	; Destination string offset
CLD	; Clear direction flag (increment registers)
REP MOVSB	; Repeat MOVSB CX times to copy string

5. Branch Instructions

These instructions change the flow of program execution.

A. Call and Return

- CALL: Calls a procedure (function) and saves the return address on the stack.
- RET: Returns from the procedure to the place where it was called.

B. Jump Instructions

- JMP: Unconditionally jumps to a specified address.
- Conditional jumps (jump only if condition is true):
- JA / JNBE: Jump if above (no carry and not equal).
- JAE / JNB: Jump if above or equal (no carry).
- JBE / JNA: Jump if below or equal (carry or zero).
- JC: Jump if carry flag is set (CF=1).
- JE / JZ: Jump if equal or zero flag is set (ZF=1).
- JG / JNLE: Jump if greater (signed).
- JGE / JNL: Jump if greater or equal (signed).



- JL / JNGE: Jump if less (signed).
- JLE / JNG: Jump if less or equal (signed).
- JNC – Used to jump if no carry flag ($CF = 0$)
- JNE/JNZ – Used to jump if not equal/zero flag $ZF = 0$
- JNO – Used to jump if no overflow flag $OF = 0$
- JNP/JPO – Used to jump if not parity/parity odd $PF = 0$
- JNS – Used to jump if not sign $SF = 0$
- JO – Used to jump if overflow flag $OF = 1$
- JP/JPE – Used to jump if parity/parity even $PF = 1$
- JS – Used to jump if sign flag $SF = 1$
- JCXZ – Used to jump to the provided address if $CX = 0$

6. Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values. Include these instructions

- STC – Used to set carry flag CF to 1
- CLC – Used to clear/reset carry flag CF to 0
- CMC – Used to put complement at the state of carry flag CF .
- STD – Used to set the direction flag DF to 1
- CLD – Used to clear/reset the direction flag DF to 0
- **STI** (Set Interrupt flag): This instruction enables hardware interrupts by setting the Interrupt Flag (IF) to 1. When enabled, the CPU will respond to maskable interrupts (like keyboard or timer interrupts).
- **CLI** (Clear Interrupt flag): This instruction disables hardware interrupts by clearing the Interrupt Flag (IF) to 0. When disabled, the CPU ignores maskable interrupts temporarily, useful to protect critical code sections.



Tips: Interrupts are signals from hardware devices, and controlling when the CPU accepts them helps avoid errors during sensitive operations.

- Use CLI before a critical section of code to prevent interrupts from stopping it.
- Use STI after the critical section to allow interrupts again.

7. Loop & Iteration Instructions

- These instructions help run a set of instructions multiple times. Include:
- LOOP – Used to loop a group of instructions until the condition satisfies, i.e., $CX = 0$
- LOOPE/LOOPZ – Used to loop a group of instructions till it satisfies $ZF = 1$ & $CX = 0$
- LOOPNE/LOOPNZ – Used to loop a group of instructions till it satisfies $ZF = 0$ & $CX = 0$

Example: code increases BX 5 times, controlled by the loop counter in CX.

```
MOV CX, 5      ; Set loop counter to 5
INC BX
LOOP START_LOOP ; Decrement CX and repeat if CX != 0
```

8. Interrupt Instructions

- INT: Interrupt instruction—stops the current program and runs a special service routine (like a function) for that interrupt.
- INTO: Interrupt only if the overflow flag (OF) is set to 1 (used after operations that might overflow).
- IRET: Return from an interrupt service routine back to where the program was interrupted.

Example: Interrupt handler prints message on screen message DB 'Hello, Interrupt!', '\$'

```
MOV AH, 09h ,    MOV DX,
OFFSET message , INT 21h ; Call DOS interrupt to display a message
```