

Lab 01: Intro to Assembly Language

Assembly language is a low-level programming language designed for a specific family of processors. It uses symbolic code (mnemonics) to represent machine-level instructions, making it easier for humans to read and write.

Each computer has a microprocessor that performs arithmetic, logic, and control operations. These processors operate using **machine language**, which consists of binary code (1s and 0s). Because machine language is difficult for humans to work with, assembly language was developed as a more readable alternative.

Assembly language allows programmers to write instructions that are then translated into machine code by an **assembler**. It is specific to each processor family, as each has its own unique instruction set for tasks like input/output, display, and computation.

Pros:

1. **High performance** – Executes faster and uses less memory.
2. **Hardware control** – Allows direct interaction with hardware.
3. **Time-critical suitability** – Ideal for real-time and interrupt-driven tasks.
4. **System-level insight** – Enhances understanding of how hardware and software interact.

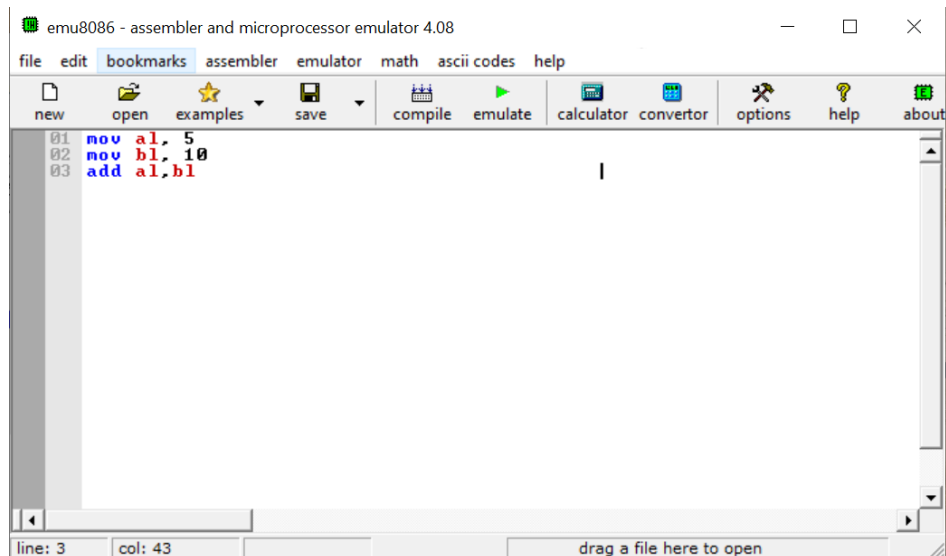
Cons:

1. **Hard to learn** – Complex syntax and structure.
2. **Not portable** – Code is processor-specific.
3. **Slow development** – Writing and debugging takes more time.
4. **Difficult maintenance** – Hard to read, update, or scale.

Introduction to EMU8086

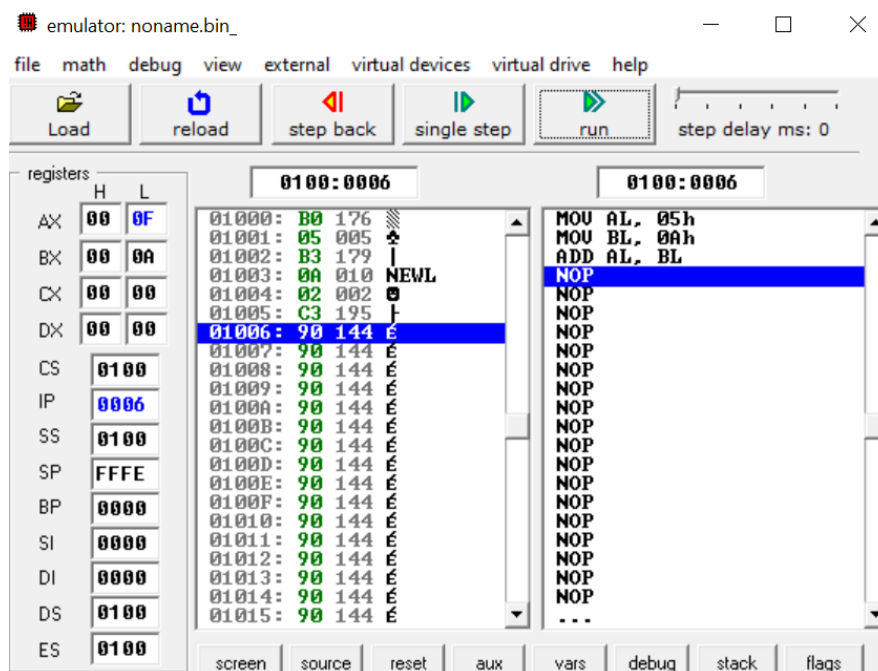
What is EMU8086?

- **EMU8086** is a **microprocessor emulator** (simulator) for the Intel 8086 CPU.
- It allows you to **write, run, and debug assembly programs** without needing real hardware.
- Students can see how registers, memory, and instructions work **step by step**.



Features of EMU8086

1. Built-in assembler, emulator, and debugger in one tool.
2. Real-time register, memory, and flag visualization.
3. Step-by-step program execution for easy tracing.
4. Supports both structured assembly (with .data, .code) and flat assembly (direct instructions).
5. Provides DOS/BIOS interrupt support for I/O operations.
6. User-friendly interface designed for education.



Assembler:

An assembler is a tool that converts **assembly language** (human-readable instructions close to machine code) into **machine code** (binary instructions that the CPU can execute). It acts as a bridge between low-level programming and the processor, allowing programmers to write instructions using mnemonics like MOV, ADD, or SUB instead of raw binary.

MASM (Microsoft Macro Assembler)

MASM is Microsoft's assembler for x86 processors. It is widely used in education because of its clear and readable syntax, which looks close to English. MASM is common in Windows and DOS programming, and tools like EMU8086 often follow MASM-style code. It is usually recommended for beginners learning assembly programming.

NASM (Netwide Assembler)

NASM is an open-source assembler that works across many platforms like Windows, Linux, and macOS. It is known for being lightweight, flexible, and widely used in system-level programming and operating system development. NASM is popular in professional and research environments where cross-platform assembly is needed.

Feature	MASM (Microsoft Macro Assembler)	NASM (Netwide Assembler)
Developer	Microsoft	Open-source community
Platforms	Windows, DOS	Windows, Linux, macOS
Common Use	Education, learning assembly	System programming, OS development
Availability	Proprietary (but widely used in tools like EMU8086)	Free and open-source

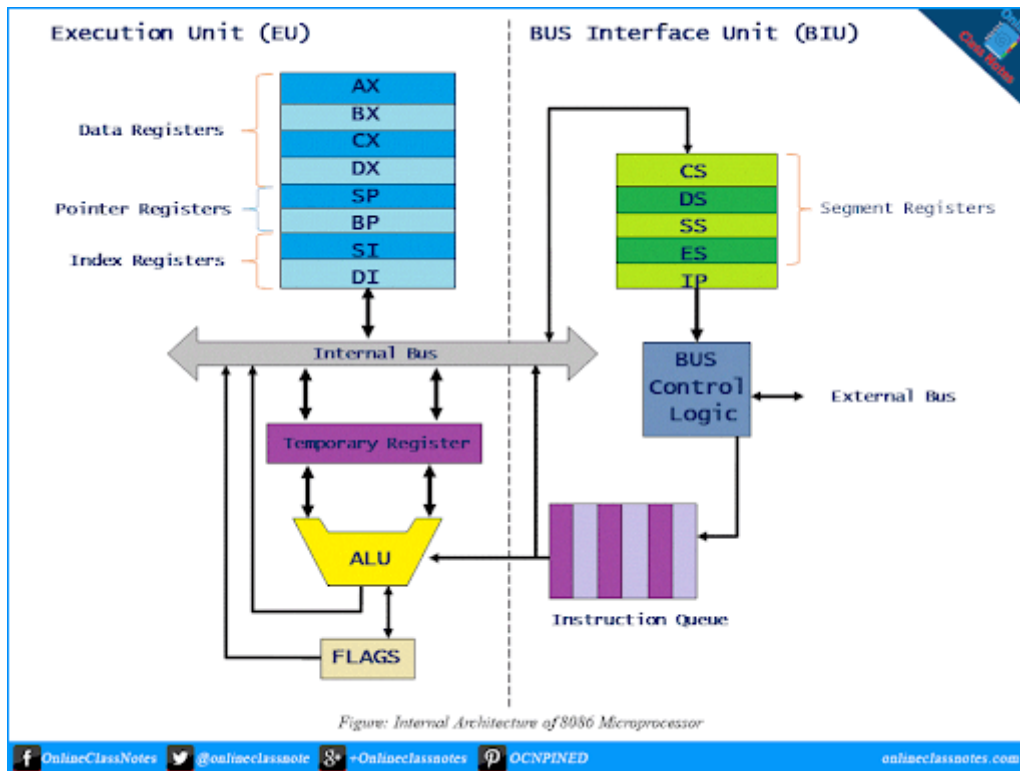
8086 Microprocessor Architecture

The **Intel 8086** is a 16-bit microprocessor introduced by Intel in 1978. It is based on the **CISC (Complex Instruction Set Computer)** design and forms the foundation of the x86 family of processors that modern computers still use today.

Key Features of 8086

- **16-bit processor:** Processes 16 bits of data at a time.
- **20-bit address bus:** Can access up to 1 MB (2^{20} bytes) of memory.
- **Segmented memory:** Uses segments (Code, Data, Stack, Extra) for easier memory management.

- **Registers:** Provides general-purpose, segment, index, and pointer registers.
- **Instruction set:** Large set of instructions (CISC).
- **Clock speed:** Originally 5–10 MHz.



Architecture Overview

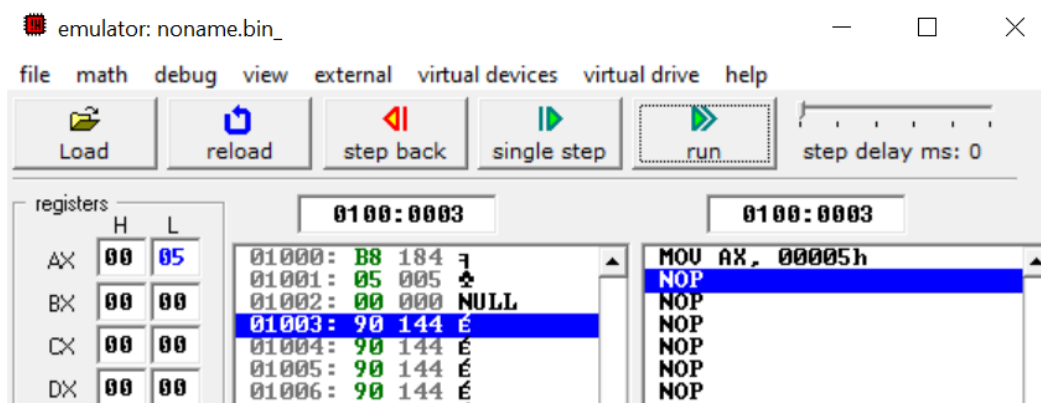
The 8086 architecture is divided into **two main units**:

1. Bus Interface Unit (BIU)

- Handles **addressing and data transfer** with memory and I/O.
- Responsible for fetching instructions and queuing them before execution.
- Contains:
 - **Segment Registers:** CS (Code), DS (Data), SS (Stack), ES (Extra)
 - **Instruction Pointer (IP)**
 - **Address Generation**

2. Execution Unit (EU)

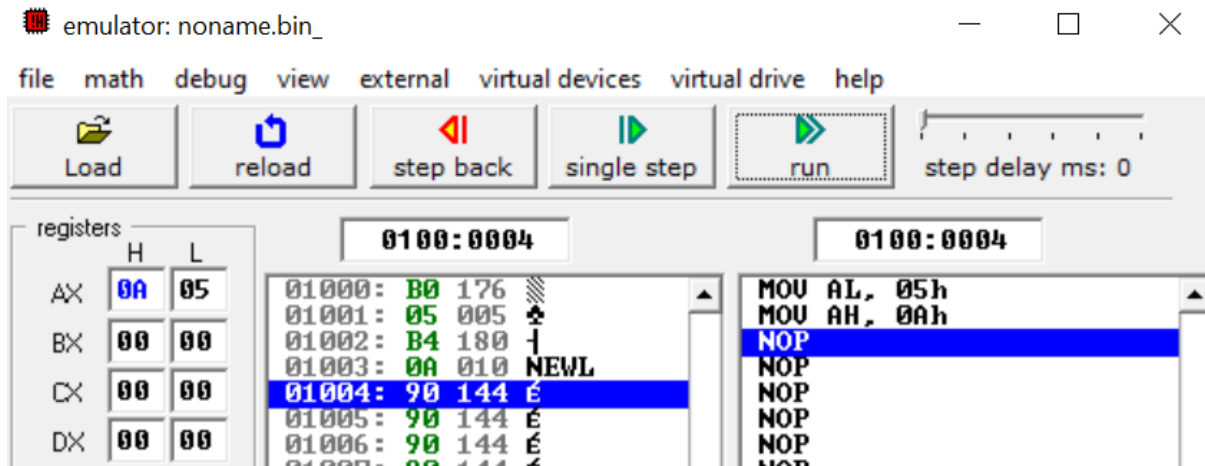
- Executes instructions fetched by BIU.
- Performs arithmetic and logic operations.



Note: Registers are divided in two 8-bit halves: higher and lower bits. That means we can use either lower, higher or both halves of the register.

Accessing both halves

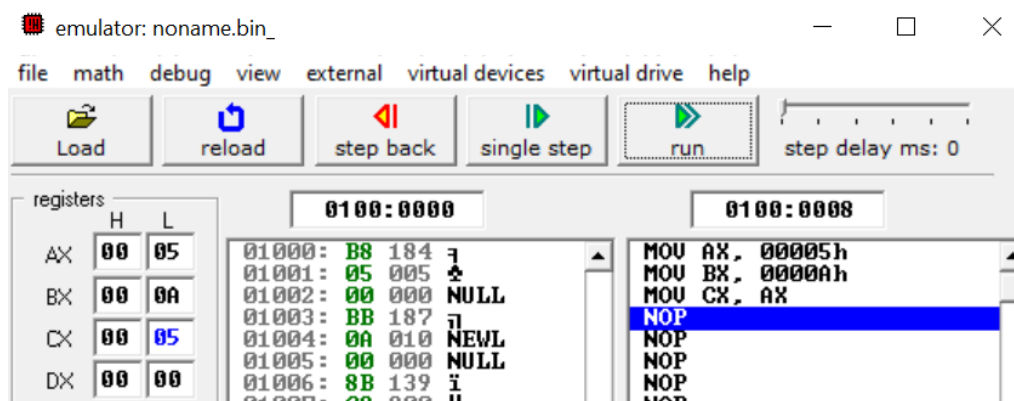
```
Mov al, 5  
Mov ah, 10
```



Moving data between registers

```
MOV AX, 5  
MOV BX, 10  
MOV CX, AX ; Copy AX to CX
```

Output:



Terminating the program (Releasing Memory)

```
MOV AX, 6      ; Load numeric value 6 into AX  
MOV AH, 4Ch    ; DOS terminate program function  
INT 21h        ; Call DOS interrupt
```

Why call DOS interrupt?

We call a DOS interrupt (INT 21h) because the 8086 CPU itself has no built-in instructions for tasks like:

- Printing text on the screen
- Reading from the keyboard
- Ending a program
- Handling files, etc.

The CPU only understands basic machine instructions (add, move, jump, etc.). To do higher-level things, it relies on the Operating System (DOS).

INT 21h is like a *gateway* to DOS services.

- The value in AH selects the function.
- Example:
 - AH = 09h → print a string
 - AH = 01h → read a character
 - AH = 4Ch → terminate program

So, when we write:


- MOV AH, 4Ch
- INT 21h

we're telling DOS: *"Please run function 4Ch (terminate program) for me."* This way, the OS takes over and does the cleanup and exit safely.

Displaying a character

```
MOV DL, 'A'      ; Put ASCII of 'A' into DL
MOV AH, 02H      ; DOS service: print character
INT 21H          ; Call DOS

MOV AH, 4CH      ; DOS service: exit program
INT 21H
```

 emulator screen (80x25 chars)

Exercise

- **Task 1:** Use XCHG command in an example and explain.
- **Task 2:** Find the maximum value that can be stored in a register in 8086 microprocessor and find out what happens if we exceed the value.
- **Task 3:** Instead of character display the number and see what happens.
- **Task 4:** Try printing more than one letter.