## Computer Software:

Practical computer systems divide software systems into three major classes: operating System, Application programs, and programming languages.

a. **Operating System**: is an interface between hardware and user. An OS is responsible for the management and coordination of activities and the sharing of the resources of the computer. The operating system acts as a host for computing applications run on the machine. As a host, one of the purposes of an operating system is to handle the details of the operation of the hardware. This relieves application programs from having to manage these details and makes it easier to write applications (ex: MS-DOS, Windows, Linux, … etc).

b. **Application Programs:** an application program (sometimes shortened to application) is any program designed to perform a specific function directly for the user or, in some cases, for another application program. Examples of application programs include word processors; database programs; Web browsers; development tools; drawing, paint, and image editing programs; and communication programs. Application programs use the services of the computer's operating system and other supporting programs.

c. **Programming Languages:** A programming language is an artificial language designed to express computations that can be performed by a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine, to express algorithms precisely, or as a mode of human communication.

   There are various types of programming languages that one may write for a computer, but the computer can execute programs written for a computer may be in one of the following categories:

   - **Machine language:** a system of instructions and data executed directly by a computer's central processing unit. It is the lowest-level programming language that is understood by computers. While it is easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers (in binary or octal or hexadecimal code).
   - **Assembly Language:** is a programming language consisting mostly of symbolic equivalents of a particular computer's machine language called mnemonics. Computers produced by different manufacturers have different machine languages and require different assembly languages. Each symbolic instruction can be translated into one binary coded instruction and that done by special program called assembler.

- **High – Level Language:** is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of a computer. Such languages are considered high-level because they are closer to human languages and further from machine languages.

# The 8086 Microprocessor

The 8086 microprocessor is a 16-bit microprocessor chip designed by Intel and introduced on the market in 1978, which gave rise to the x86 architecture. 8086 microprocessor has 16-bit data bus, so it can read or write data to memory or I/O ports either 16-bits or 8-bits at a time. It has 20 bits address bus, so it can address up to 1 Mbytes of memory ($2^{20} = 1048576$ bytes $=$ 1 Mbytes).

## Internal Architecture of the 8086 Microprocessor:

The internal architecture of a microprocessor describes its functional components and their interaction. Figure (1.4) illustrates the internal architecture of 8086 microprocessor.

This architecture can be divided into two units: Bus Interface Unit (BIU) and Execution unit (EU).

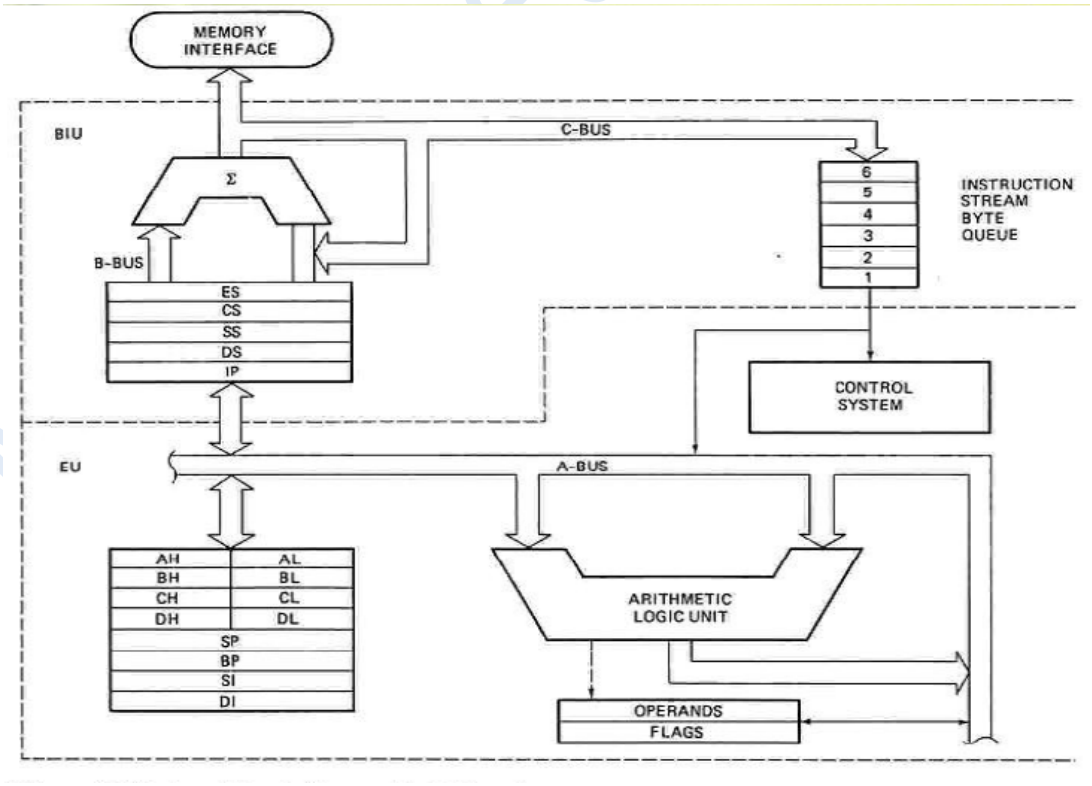


**Fig. (1.4)** ***Internal Architecture of 8086 Microprocessors***

## 1. Bus Interface Unit (BIU)

The BIU handles all transactions of data and addresses on the buses for EU. It provides full 16-bit bidirectional data bus and 20-bit unidirectional address bus. The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory (**Address Generation**) operands. It The instruction bytes are transferred to the instruction queue then, transferred to the EU to be executed. The BIU contains on the following:

**i. Segment registers:**

**Code Segment register (CS)** - points at the segment containing the current program.

**Data Segment register (DS)** - generally points at segment where variables are defined.

**Extra Segment register (ES)** - points to the segment of the memory that deals with ***string operations***.

**Stack Segment register (SS)** - points at the segment containing the stack.

Figure (1.5) shows the memory and its segments.

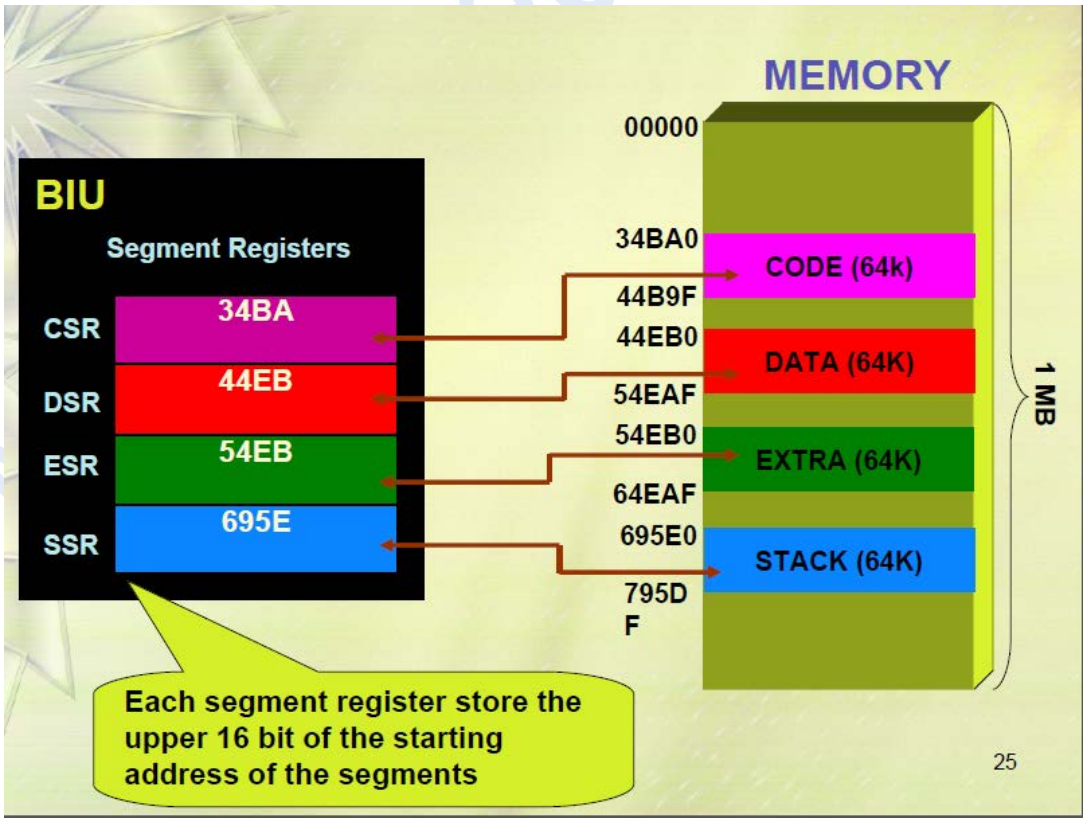


**Fig. (1.5)** ***Memory Segments with Segment Registers***

**ii. Instruction pointer (IP):**

1. Always points to next instruction to be executed
2. Offset address relative to CS

**IP** register always works together with **CS** segment register and it points to currently

**iii. Instruction Queue:**

BIU contains an object code queue that permits prefetching of up to six bytes of instruction code. The instruction queue is used as a temporary memory storage area of data instructions that are to be executed by the microprocessor. In effect, the BIU and EU work in parallel. The instruction queue is first-in, first-out (FIFO) in the memory. This means that the first instruction loaded into the instruction queue by the bus control unit will be the first instruction to be used the ALU.

**iv. Address Generation:**

The address-generator unit is used with the segment registers to generate the 20-bit physical address required to identify all the possible memory addresses. Since the maximum width of the 8086 registers is 16-bit, the address generator used to change the 16-bit address (usually called ***Logical Address*** (**LA**)) to a 20-bit address (usually called ***Physical Address*** (**PA)**). The physical address is obtained by shifting the segment base value four bit positions (one hexadecimal position to the left) and adding the 16-bit offset or the logical address. This operation is illustrated in figure (1.6).

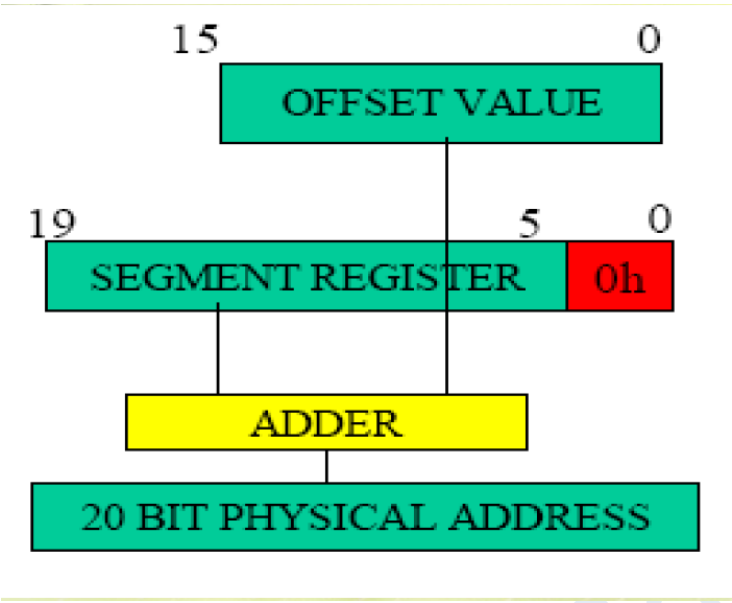


**Fig. (1.6)** ***Address Generation Operation***

The following formula used to calculate the PA from a given LA and segment register.

PA=Segment register (base address)*10h+LA (offset)

Or

PA=SR(0h)+offset

Table (1.2) represents each segment with related offset.

| Segment | Offset |
|---|---|
| CS | IP |
| DS | BX, SI, DI |
| ES | DI |
| SS | SP, BP |

**Table (1.2)** ***All Segment with Related Offsets***

**Example 1:** You have CS=1000h, IP=2000h. Calculate the PA.

Sol.: PA=1000(0h) + 2000

=10000+2000=12000h.