Chapter 02

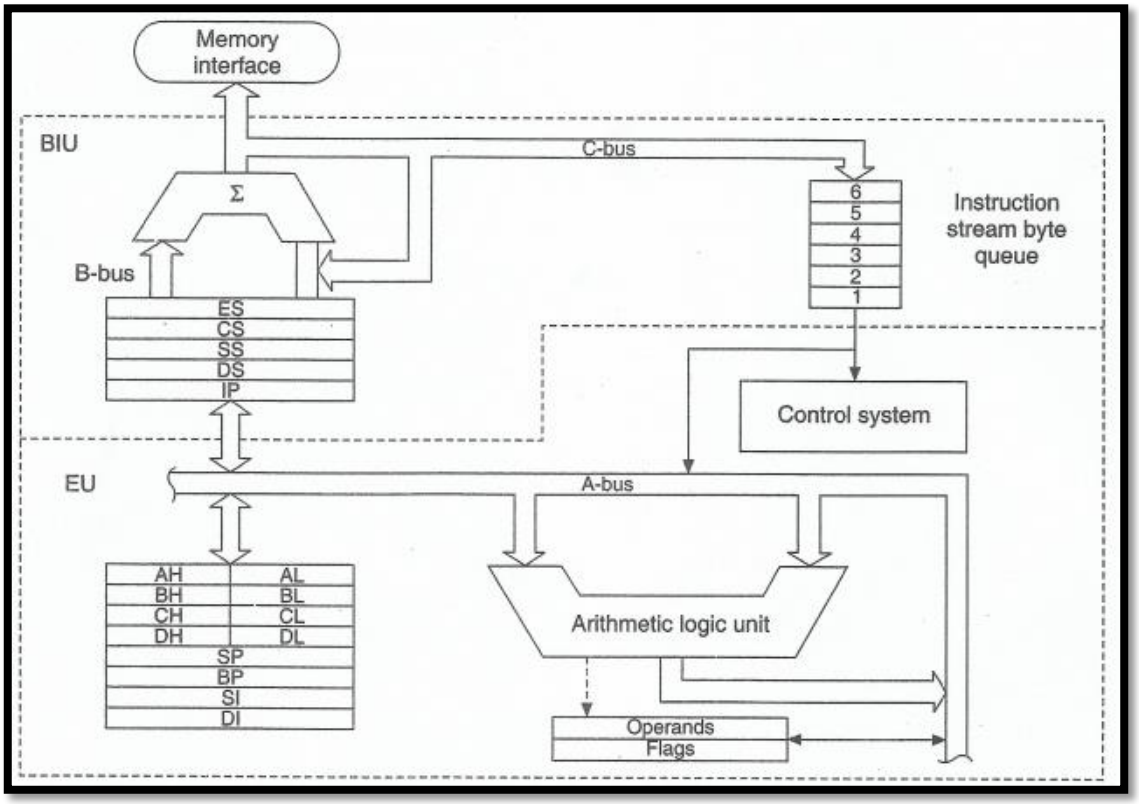
# 8086 Microprocessor

## Table of content

## ■ Terminology

**8086 features**

- Intel's first 16-bit MP
- Packaged in 40 pin dual-in-line package
- The central processing unit has been divided into
  - Bus Interface Unit BIU
    Provides interface with external bus and executes all bus operation
    It has a 6-byte instruction object code queue
  - Execution unit EU
    Takes the instruction from object code queue of BIU and executes it
- 20-bit address bus
- 16-bit data bus
- 16-bit registers




## ■ The Programming Model

The ***programming model*** of the 8086 through the Pentium Pro is considered ***program visible*** because its registers are used during programming and are specified by the instructions. Other registers, not

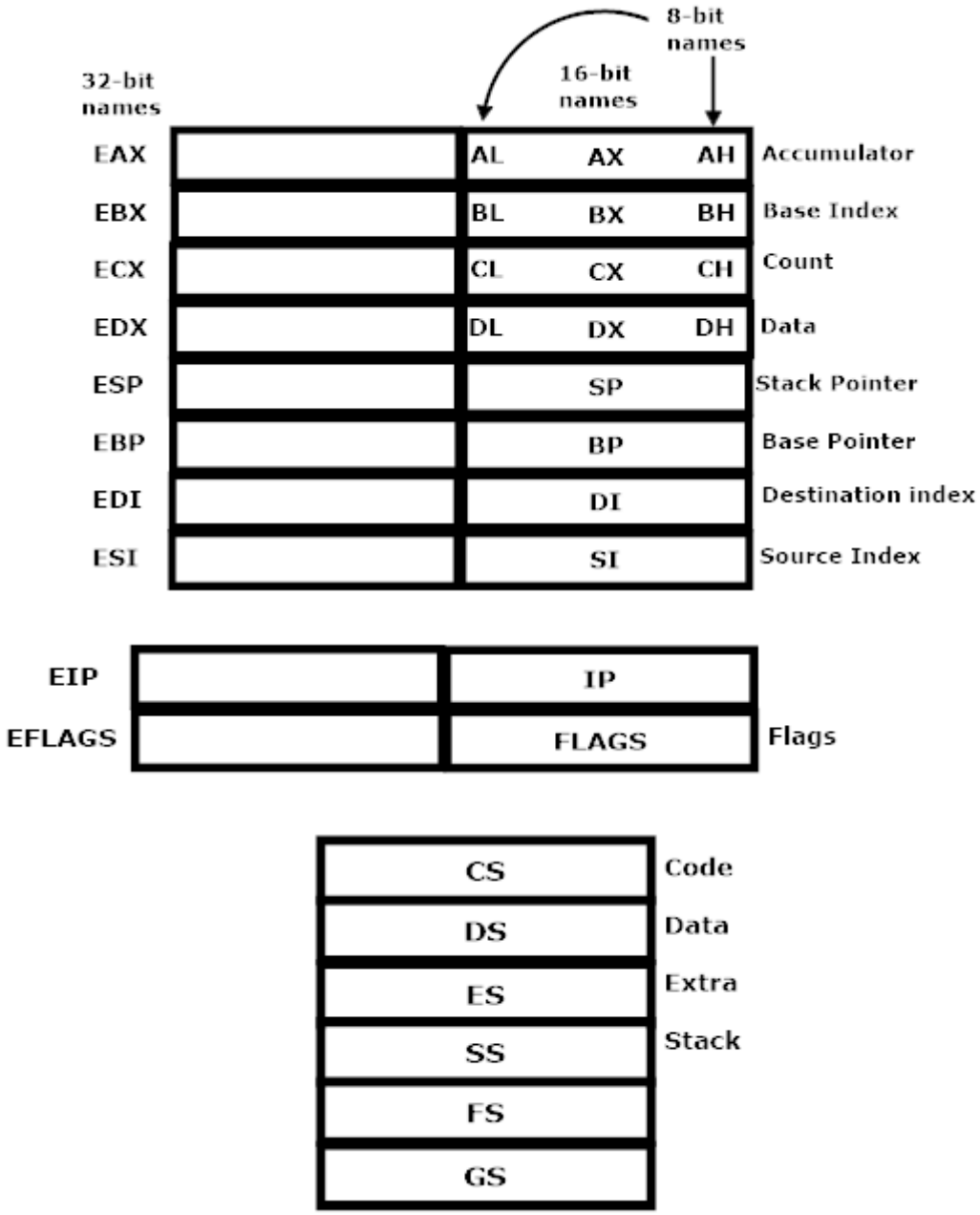
covered in this chapter, are considered ***program invisible*** because they are not addressable directly during applications programming, but may be used indirectly during **system programming**. Only the 80286 and above contain the program-invisible registers used to control and operate the protected memory system.

**REGISTERS for 8088/86**

The registers of the 8088/86 fall into the six categories outlined in table below.

| Category | Bits | Register Names |
|---|---|---|
| **General** | 16 | AX, BX, CX, DX |
| | 8 | AH, AL, BH, BL, CH, CL, DH, DL |
| **Pointer** | 16 | SP (Stack pointer)<br>BP (Base Pointer) |
| **Index** | 16 | SI (Source Index)<br>DI (Destination Index) |
| **Segment** | 16 | CS (Code Segment)<br>DS (Data Segment)<br>SS (Stack Segment)<br>ES (Extra Segment) |
| **Instruction** | 16 | IP (Instruction Pointer) |
| **Flag** | 16 | FR (Flag Register) |

The general-purpose registers in 8088/86 microprocessors can be accessed as either 16-bits or 8-bits register. All other registers can be accessed only as the full 16-bits. In the 8088/86, data types are either 8 or 16-bits. To access 12-bits data, for example, 16-bit register must be used with the highest 4 bits set to 0. The bits of a register are numbered in descending order, as shown below




- The programming model contains 8-, and 16-bit registers. The 8-bit registers are **AH, AL, BH, BL, CH, CL, DH, and DL** and are referred to when an instruction is formed using these two-letter designations. For example, an **ADD AL,AH** instruction adds the 8-bit contents of AH to AL. (Only AL changes due to this

instruction.) The 16-bit registers are **AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, and ES**.

- These registers are also referenced with these two-letter designations. For example, an **ADD DX, CX** instruction adds the 16-bit contents of CX to DX. (Only DX changes due to this instruction.)
- Some registers are **general-purpose** or **multipurpose registers**, while some have **special purposes**.

■ General-purpose registers

| | |
|---|---|
| **AX** (accumulator) | AX is referenced as a 16-bit register (AX), or as either of two 8-bit registers AH and AL. Note that if an 8- bit register is addressed, only that portion of the 16-bit register changes without affecting the remaining bits. *The accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions*. For these instructions, the accumulator has a special purpose, but is generally considered a multipurpose register. |
| BX **(base index)** | BX is addressable as BX, BH, or BL. The BX register *sometimes holds the offset address of a location in the memory system in all versions of the microprocessor*. |

| | |
|---|---|
| CX **(count)** | CX is a general-purpose register that also *holds the count for various instructions*. Instructions that use a count are the repeated string instructions (REP/REPE/REPNE), shift, rotate, and LOOP/LOOPD instructions. |
| DX **(data)** | DX is a general-purpose register that *holds a part of the result from a multiplication or part of the dividend before a division*. |

| | |
|---|---|
| BP **(base pointer)** | BP *points to a memory location in all versions of the microprocessor for memory data transfers*. |
| DI **(destination index)** | DI often *addresses string destination data for the string instructions*. |
| SI **(source index)** | SI is the source index register *often addresses source string data for the string instructions*. |

■ Special-purpose Registers

The **special-purpose registers** include IP, SP, FLAGS, and the segment registers CS, DS, ES, SS, FS, and GS.

| | |
|---|---|
| IP **(instruction pointer)** | IP addresses the next instruction in a section of memory defined as a code segment. This register is IP (16-bits) when the micropro-cessor operates in the real mode and EIP (32-bits) when the 80386 and above operate in the |

| | |
|---|---|
| | protected mode. The instruction pointer, which points to the next instruction in a program, is used by the microprocessor to find the next sequential instruction in a program located within the code segment. The instruction pointer can be modified with a jump or a call instruction. |

| | |
|---|---|
| SP **(stack pointer)** | *SP addresses an area of memory called the stack*. The stack memory stores data through this pointer. |

**Segment Registers**

Additional registers, called segment registers, *generate memory addresses when combined with other registers in the microprocessor*. There are four segment registers in various versions of the microprocessor. Following is a list of each segment register along with its function in the system.

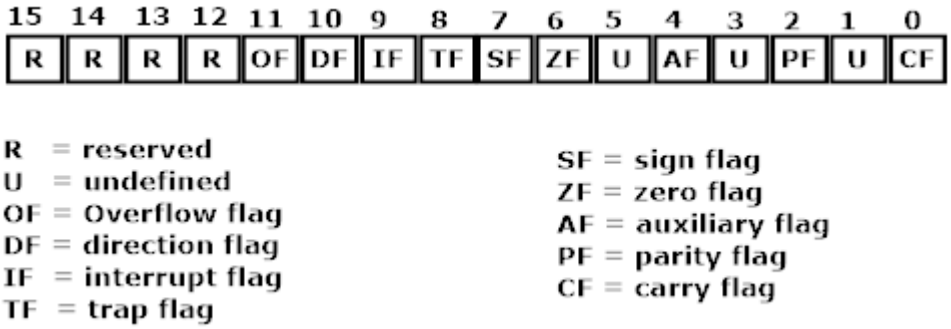
| | |
|---|---|
| CS **(code)** | The code segment is a section of memory that holds the code (programs and procedures) used by the microprocessor. The code segment register defines the starting address of the section of memory holding code. In real mode operation, it defines the start of a 64K byte section of memory; in protected mode, it selects a descriptor that describes the starting address and length of a section of memory holding code. The code segment length is |

| | |
|---|---|
| | limited to 64K bytes in the 8088–80286 and 4G bytes in the 80386 and above when these microprocessors operate in the protected mode. |
| DS **(data)** | The data segment is a section of memory that contains most data used by a program. Data are accessed in the data segment by an offset address or the contents of other registers that hold the offset address. As with the code segment and other segments, the length is limited to 64K bytes in the 8086–80286 and 4G bytes in the 80386 and above. |
| ES **(extra)** | The extra segment is an additional data segment used by some of the string instructions to hold destination data. |
| SS **(stack)** | The stack segment defines the area of memory used for the stack. Thy location of the current entry point in the stack segment is determined by the stack pointer register. The BP register also addresses data within the stack segment. |

**Flag register**

- The flag register is a 16-bit register sometimes referred to as the status register (Program Status Word PSW).
- Although the register is 16 bits wide, only some of the bits are used. The rest are either undefined or reserved by Intel.

- Six of the flags are called conditional flags, meaning that they indicate some condition that resulted after an instruction was executed. These six are CF, PF, AF, ZF, SF, and OF.
- The three remaining flags are sometimes called control flags since they are used to control the operation of the instructions before they are executed.




**Carry flag CF**

This flag is set whenever there is a carry output, either from d7 after an 8-bit operation or from d15 after a 16-bit operation

**Parity flag PF**

After certain operations, the parity of the result's low order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise it is cleared.

**Auxiliary flag AF**

If there is a carry from d3 to d4 of an operation, this bit is set; otherwise, it is cleared (set equal to zero). This flag is used by the instructions that perform BCD arithmetic.

**Zero flag ZF**

The zero flag is set to 1 if the result of an arithmetic or logical operation is zero; otherwise it is cleared

**Sign flag SF**

Binary representation of signed numbers uses the most significant bit as the sign bit, after arithmetic or logic operations. The status of this sign bit is copied into the SF, thereby indicating the sign of the result

**Trap flag TF**

When this flag is set, it allows the program to single-step, meaning to execute one instruction at a time. Single-stepping is used for debugging purposes.

**Interrupt Enable flag IF**

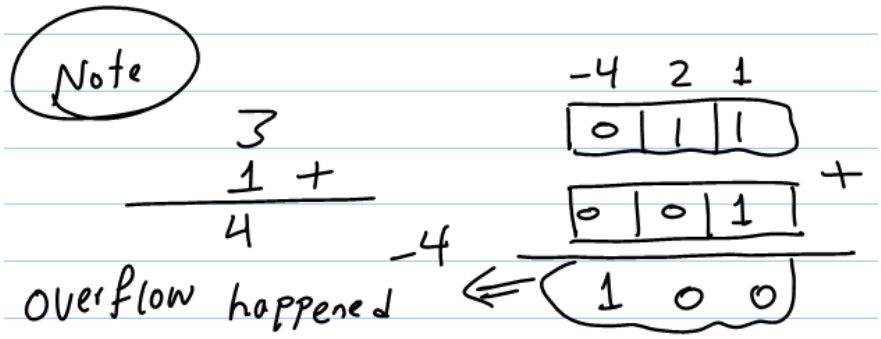
This bit is set or cleared to enable or disable only the external maskable interrupt requests.

**Direction flag DF**

This bit is used to control the direction of string operations

**Overflow flag**

This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations. The overflow flag is only to detect errors in signed arithmetic operations



**Example**

Show how the flag register is affected by the addition of 38H and 2FH.

**Solution**

**MOV BH,38H**

**ADD BH,2FH**

```
    38      0011    1000
+   2F      0010    1111
-------   ---------------
    67      0110    0111
```

CF = 0 since there is no carry beyond d7

PF = 0 since there is an odd number of 1s in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

**Example**

Show how the flag register is affected by

**MOV AL,9CH ; AL = 9CH**

**MOV DH,64H ; DH = 64H**

**ADD AL,DH ; now AL = 0**

**Solution**

```
    9C      1001    1100
+   64      0110    0100
-------   ---------------
    00      0000    0000
```

CF = 1 since there is a carry beyond d7

PF = 1 since there is an even number of 1s in the result

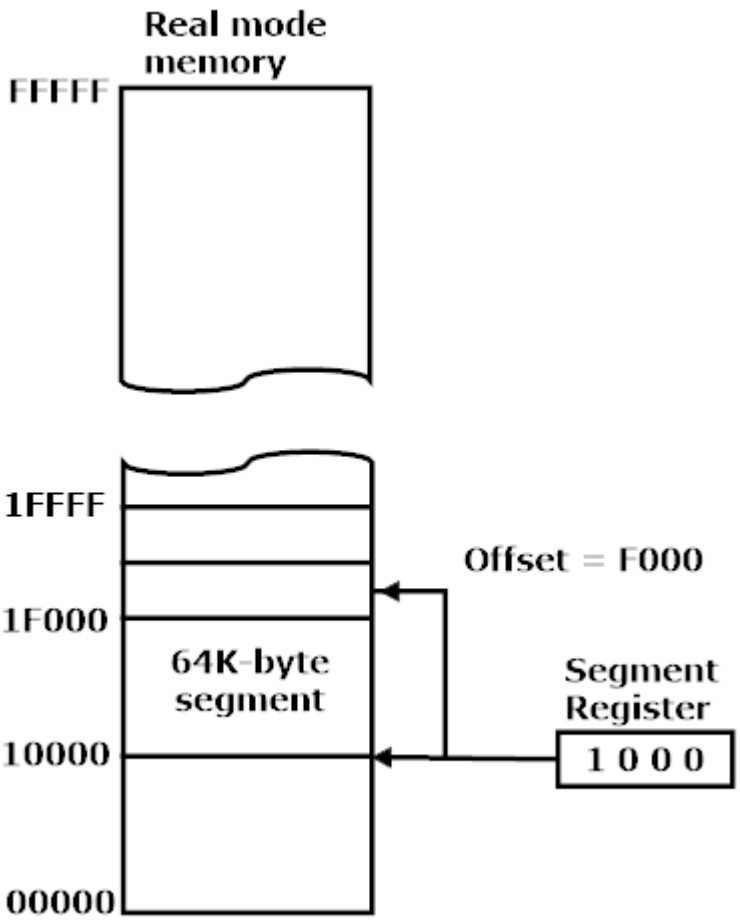
AF = 1 since there is a carry from d3 to d4

ZF = 1 since the result is zero

SF = 0 since d7 of the result is zero

**Segments and Offsets**

- A combination of a **segment address** and an **offset address** access a memory location in the real mode.
- All real mode memory addresses consist of a segment address plus an offset address.
- The segment address, located within one of the segment registers, defines the beginning address of any 64K-byte memory segment.
- The offset address selects any location within the 64K-byte memory segment.
- Figure below shows how the segment plus offset addressing scheme selects a memory location.




- This illustration shows a memory segment that begins at location 10000H and ends at location 1FFFFH -64K bytes in length. It also shows how an **offset**, sometimes called a ***displacement***, of F000H selects location 1F000H in the memory system.
- *Note that the offset or displacement is the distance above the start of the segment*.
- Because a real mode segment of memory is 64K in length, once the beginning address is known, **the ending address is found by adding FFFFH**.

  For example, if a segment register contains 3000H, the first address of the segment is 30000H and the last address is 30000H + FFFFH or 3FFFFH. Table below shows several examples of segment register contents and the starting and ending addresses of the memory segments selected by each segment address.

| Segment Register | Starting Address | Ending Address |
|---|---|---|
| 2000H | 20000H | 2FFFFH |
| 2001H | 20010H | 3000FH |
| 2100H | 21000H | 30FFFH |
| AB00H | AB000H | BAFFFH |
| 1234H | 12340H | 2233FH |

- The offset address is added to the start of the segment to address a memory location in the memory segment.

  **For example**, if the segment address is 1000H and the offset address is 2000H, the microprocessor addresses memory location 12000H. The segment and offset address is sometimes written as **1000:2000** for a segment address of 1000H with an offset of 2000H.

**Default Segment and Offset Registers**

- The microprocessor has a set of rules that apply to segments whenever memory is addressed. These rules, which apply in either the real or protected mode, *define the segment register and offset register combination used by certain addressing modes*. For example, the code segment register is always used with the instruction pointer to address the next instruction in a program.
- The code segment register defines the start of the code segment and the instruction pointer locates the next instruction within the code segment. This combination (**CS:IP**) locates the next instruction executed by the microprocessor.

**For example**, if CS = 1400H and IP = 1200H, the microprocessor fetches its next instruction from memory location 14000H + 1200H or 15200H.

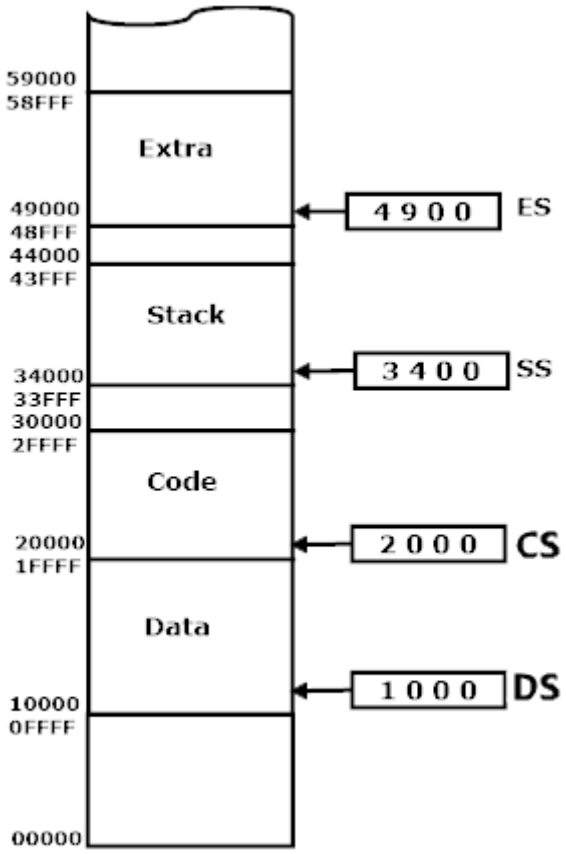
- **Another of the default combinations is the stack**. Stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (**SP**) or the base pointer (**BP**). These combinations are referred to as **SS:SP** or **SS:BP**.

  **For example**, if SS = 2000H and BP = 3000H, the microprocessor addresses memory location 23000H for a stack segment memory location.

- Other defaults are shown in Table below for addressing memory using any Intel microprocessor with 16-bit registers.

| **Segment** | **Offset** | **Special Purpose** |
|---|---|---|
| CS | IP | Instruction address |
| SS | SP or BP | Stack address |
| DS | BX, DI, SI, an 8-bit number or a 16-bit number | Data address |
| ES | DI for string instructions | String destination address |

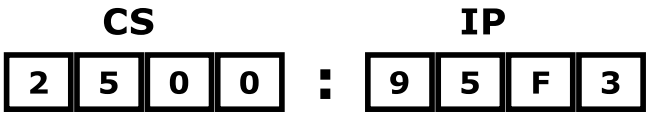
- Figure below shows a system that contains four memory segments. Note that a memory segment can touch or even overlap if 64K bytes of memory are not required for a segment. Think of segments as windows that can be moved over any area of memory to access data or code. Also, note that a program can have more than four or six segments, but can only access four or six segments at a time.



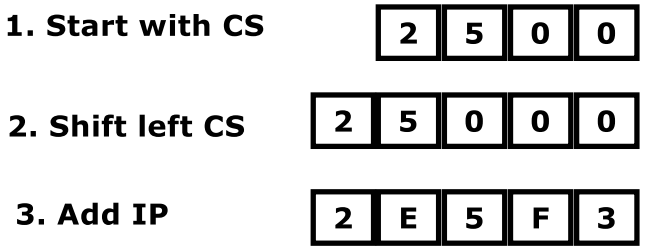

- **Logical address and physical address**
  - As we discussed before there are two types of addresses:
    - Physical address, and
    - Logical address.
  - The physical address is the 20-bit address that is actually put on the address pins of the 8086 microprocessor and decoded by the memory interfacing circuitry. The address can have a range of 00000H to FFFFFH for the 8086. This is an actual physical location in RAM or ROM within the 1 megabyte memory space.
  - The offset address is a location within a 64K-byte segment range. Therefore, an offset address can range from 0000H to FFFFH. The logical address consists of a segment value and an offset address.

**Example:**

To execute a program, the 8086 fetches the instructions (opcode and operands) from the code segment. The logical address of an instruction always consists of a CS (code segment) and an IP (instruction pointer), shown in CS:IP format.




The physical address for the location of the instruction is generated by shifting the CS left one hex digit and then adding it to the IP. IP contains the offset address. The resulting 20-bit address is called the physical address.




The microprocessor will retrieve the instruction from memory location starting at 2E5F3. Since IP can have a minimum value of 0000H and a maximum of FFFFH, the logical address range in the example is 2500:0000 to 2500:FFFF. This means that the lowest memory location of the code segment above will be 25000H (25000 + 0000) and the highest memory location will be 34FFFH (25000 + FFFF)

**Example:**

If CS = 24F6H and IP = 634AH, show:

(a) The logical address

(b) The offset address

(c) The physical address

(d) The lower range

(e) The upper range of the code segment

**Solution:**

(a) 24F6:634A

(b) 634A

(c) 2B2AA (24F60 + 634A)

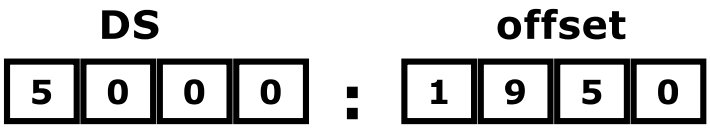
(d) 24F60 (24F60 + 0000)

(e) 34F5F (24F60 + FFFF)

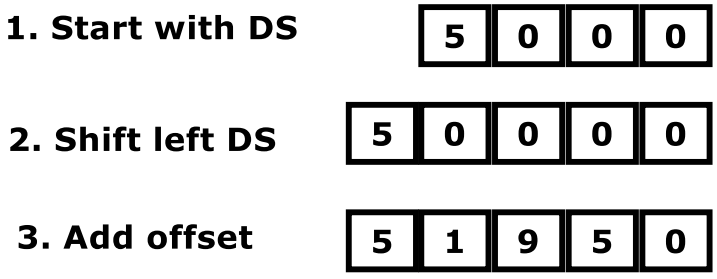
**Example:**

Assume that DS is 5000H and the offset is 1950H, calculate the physical address of the byte

**Solution**




The physical address will be 50000 +1950 = 51950




**Example:**

If DS = 7FA2H and the offset is 438EH,

(a) Calculate the physical address

(b) Calculate the lower range

(c) Calculate the upper range of the data segment

(d) Show the logical address

**Solution**

(a) 83DAE

(b) 7FA20

(c) 8FA1F

(d) 7FA2:438E

✍ **Just to note:**

- **In the code segment**, CS and IP hold the logical address of the instructions to be executed. The following assembly instructions have been assembled (translated into machine code) and stored in memory.

| Logical address **CS:IP** | Machine language Opcode and operand | Assembly language mnemonics and operand |
|---|---|---|
| 1132:0100 | B057 | MOV Al,57 |
| 1132:0102 | B686 | MOV DH,86 |
| 1132:0104 | B272 | MOV DL,72 |
| 1132:0106 | 89D1 | MOV CX,DX |
| 1132:0108 | 88C7 | MOV BH,AL |
| 1132:010A | B39F | MOV BL,9F |
| 1132:010C | B420 | MOV AH,20 |
| 1132:010E | 01D0 | ADD AX,DX |
| 1132:0110 | 01D9 | ADD CX,BX |
| 1132:0112 | 05351F | ADD AX,1F35 |

- The program above shows that the byte at address 1132:0100 contains B0, which is the opcode for moving a value into register AL, and address 1132:0101 contains the operand (in this case 57) to be moved to AL. Therefore, the instruction MOV AL,57 has a machine code of B057, where B0 is the opcode and 57 is the operand.
- Similarly, the machine code B686 is located in memory locations 1132:0102 and 1132:0103 and represents the opcode and the operand for the instruction MOV DH,86. The following are the physical addresses and the contents of each location for the program above

| **Logical address** | **Physical address** | **Machine code contents** |
|---|---|---|
| 1132:0100 | 11420 | B0 |
| 1132:0101 | 11421 | 57 |
| 1132:0102 | 11422 | B6 |
| 1132:0103 | 11423 | 86 |
| 1132:0104 | 11424 | B2 |
| 1132:0105 | 11425 | 72 |
| 1132:0106 | 11426 | 89 |
| 1132:0107 | 11427 | D1 |
| 1132:0108 | 11428 | 88 |
| 1132:0109 | 11429 | C7 |
| 1132:010A | 1142A | B3 |
| 1132:010B | 1142B | 9F |
| 1132:010C | 1142C | B4 |
| 1132:010D | 1142D | 20 |
| 1132:010E | 1142E | 01 |
| 1132:010F | 1142F | D0 |

| | | |
|---|---|---|
| 1132:0110 | 11430 | 01 |
| 1132:0111 | 11431 | D9 |
| 1132:0112 | 11432 | 05 |
| 1132:0113 | 11433 | 35 |
| 1132:0114 | 11434 | 1F |