8086 microprocessor

Features of 8086 microprocessor:-

1. The 8086 is a 16-bit microprocessor. The term "16-bit" means that its arithmetic logic unit, internal registers and most of its instructions are designed to work with 16-bit binary words.

2. The 8086 has a 16-bit data bus, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time.

3. The 8086 has a 20-bit address bus, so it can directly access 2^{20} or 10,48,576 (1Mb) memory locations. Each of the 10, 48, 576 memory locations is byte(8-bit) wide. Therefore, a sixteen-bit words are stored in two consecutive memory locations.

4. The 8086 can generate 16-bit I/0 address, hence it can access 2^{16} = 65536 I/0 ports.

5. The 8086 provides fourteen 16-bit registers.

6. The 8086 has multiplexed address and data bus which reduces the number of pins needed, but does slow down the transfer of data.

7. The 8086 is possible to perform bit, byte, word and block operations in 8086. It performs the arithmetic and logical operations on bit, byte, word and decimal numbers including multiply and divide.

8. The Intel 8086 is designed to operate in two modes, namely the minimum mode and the maximum mode. When only one 8086 CPU is to be used in a microcomputer system, the 8086 is used in the minimum mode of operation. In multiprocessor (more than one processor in the system) system 8086 operates in maximum mode.

9. An interesting feature of the 8086 is that it fetches up to six instruction bytes from memory and queue stores them in order to speed up instruction execution.

10. The 8086 provides powerful instruction set with the following addressing modes Register, immediate, direct, indirect through an index or base, indirect

through the sum of a base and an index register, relative and implied.

Architecture of 8086 :-

Fig. -1- shows a block diagram of the 8086 internal architecture. It is internally divided into two separate functional units. These are the Bus Interface Unit (BIU) and the Execution Unit (EU).

Each unit has dedicated functions and both operate at the same time. this parallel processing effectively makes the fetch and execution of instructions independent operations. This results in efficient use of the system bus and higher performance for 8086 microcomputer systems.

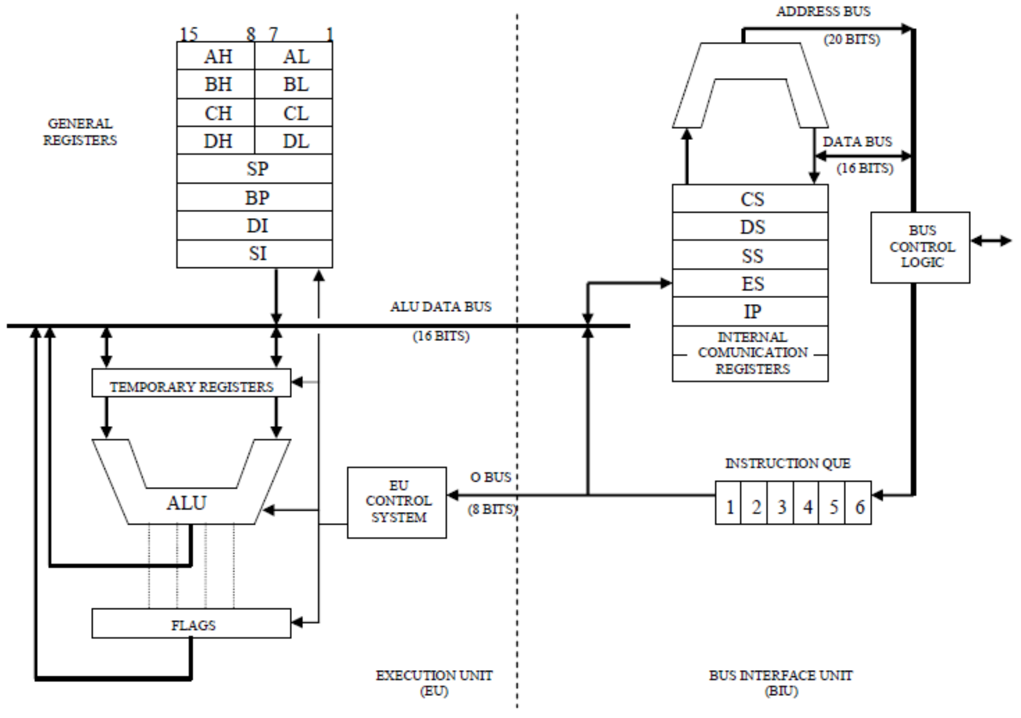


Fig-1-8086 internal block diagram

- **Bus Interface Unit [BIU]**

Is interface the 8086 to the outside world. By interface, we mean the path by which it connects to external devices.

The BIU is responsible for

- performing all external bus operations
- instruction queuing
- address generation.

Functions of Bus Interface Unit

1- It sends address of the memory or I/O.

2- It fetches instruction from memory.

3- It reads data from port/memory.

4- It writes data into port/memory.

5- It supports instruction queuing.

To implement these functions the BIU contains

- the instruction queue
- segment registers
- instruction pointer register
- address generation adder
- bus control logic.

Instruction Queue

To speed up program execution, the BIU fetches **six instruction bytes** ahead of time from the memory. These pre-fetched instruction bytes are held for the execution unit in a group of registers called **Queue.**

With the help of queue it is possible to fetch next instruction when current instruction is in execution.

During this execution time the BIU fetches the next instruction or instructions from memory into the instruction queue instead of remaining idle. The BIU continues this process as long as the queue is not full. Due to this, execution unit gets the ready instruction in the queue and instruction fetch time is eliminated.

The queue operates on the principle first in first out (FIFO). So that the execution unit gets the instructions for execution in the order they are fetched.

Feature of fetching the next instruction while the current instruction is executing is called **pipelining.**

Q/ what is mean of unpipelining ?

Note:

If the queue is full and the EU is not requesting access to data in memory, the BIU does not need to perform any bus operations. These intervals of no bus activity, which occur between bus operations, are known as idle states.

- **Execution Unit [EU]**

The execution unit is responsible for decoding and executing instructions. It consists:

- arithmetic logic unit (ALU)
- status and control flags register
- general-purpose registers
- Pointers and Index registers.

The EU accesses instructions from the output end of the instruction queue and data from the general-purpose registers or memory.
It reads one instruction byte after the other from the output of the queue, decodes them and performs the operation specified by the instruction.
The ALU performs the arithmetic and logic operations required by an instruction.
During execution of the instruction, the EU may test the status and control flags, and updates these flags based on the results of executing the instruction.

- The control circuitry in the EU directs the internal operations.
- A decoder in the EU translates the instructions fetched from memory into a series of actions which the EU performs.
- ALU is 16-bit. It can add, subtract, AND, OR, XOR, increment, decrements, complement and shift binary numbers.

Register Organization

The 8086 has a powerful set of registers. It includes general purpose registers, segment registers, pointers and index registers, and flag register. The Fig.-2- shows the register organization of 8086. The registers shown are accessible to programmer. As shown in the Fig.-2-, all the registers of 8086 are 16-bit registers.

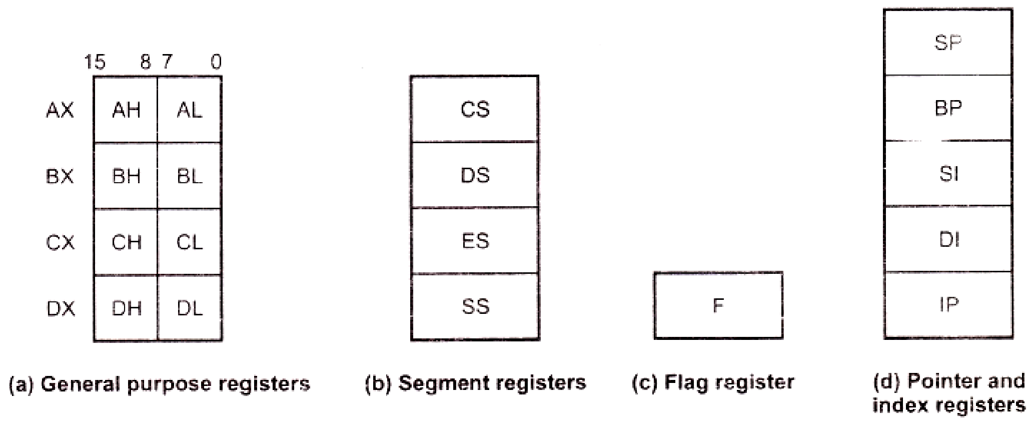


Fig-2- register organization of 8086

- **General Purpose Registers**

The 8086 has four 16-bit general purpose registers labeled AX, BX, CX and DX. Each 16-bit general purpose register can be split into two 8-bit registers.

The letters L and H specify the lower and higher bytes of a particular register. For example, BH means the higher byte (8-bits) of the BX register and BL means the lower byte (8-bits) of the BX register.

The letter X is used to specify the complete 16-bit register.

The general purpose registers are either used for holding data temporarily. They can also be used as a counters or used for storing offset address for some particular addressing modes.

The register AX is used as 16-bit accumulator whereas register AL (lower byte of AX) is used as 8-bit accumulator.

The register BX is also used as offset storage for generating physical addresses in case of certain addressing modes. On the other hand, the register CX is also used as a default counter in case of string and loop instructions.

- **Flag Register**

It is 16 bit flag register. The flag register contains nine active flags which is divided in to two parts as shown in the Fig.-3-.
1.) status flags
2.) control flags

status flags are 6 in number, they are:-
carry flag , parity flag, auxiliary carry flag , zero flag , sign flag , overflow flag

1. **Carry Flag (CF)** In case of addition this flag is set if there is a carry out of the MSB. The carry flag also serves as a borrow flag for subtraction. In case of subtraction it is set when borrow is needed.

2. **Parity Flag (PF)** It is set to 1 if result of byte operation or lower byte of the word operation contain an even number of ones; otherwise it is zero.

3. **Auxiliary Flag (AF):** This flag is set if there is an overflow out of bit 3 i.e., carry from lower nibble to higher nibble (D_3 bit to D_4 bit). This flag is used for BCD operations and it is not available for the programmer.

4. **Zero Flag (ZF)** The zero flag sets if the result of operation in ALU is zero and flag resets if the result is nonzero. The zero flag is also set if a certain register content becomes zero following an increment or decrement operation of that register.

5. **Sign Flag (SF):** After the execution of arithmetic or logical operations, if the MSB of the result is 1, the sign bit is set. Sign bit 1 indicates the result is negative; otherwise it is positive.

6. **Overflow Flag** (OF): When OF is set, it indicates that the signed result is out of range. If the result is not out of range, OF remains reset.

control flags are 3 in number, they are:-
trap flag , interrupt flag , direction flag

7. **Trap Flag (TF):** One way to debug a program is to run the program one instruction at a time and see the contents of used registers and memory variables after execution of every instruction. This process is called **'single stepping'** through a program. Trap flag is used for single

stepping through a program. If set, a trap is executed after execution of each instruction, i.e. interrupt service routine is executed which displays various registers and memory variable contents on the display after execution of each instruction. Thus programmer can easily trace and correct errors in the program.

8. interrupt Flag (IF) : It is used to allow/prohibit the interruption of a program. If set, a certain type of interrupt (a maskable interrupt) can be recognized by the 8086; otherwise, these interrupts are ignored.

9. Direction Flag (DF) :It is used with string instructions. If DF = 0, the string is processed from its beginning with the first element having the lowest address. Otherwise, the string is processed from the high address towards the low address.

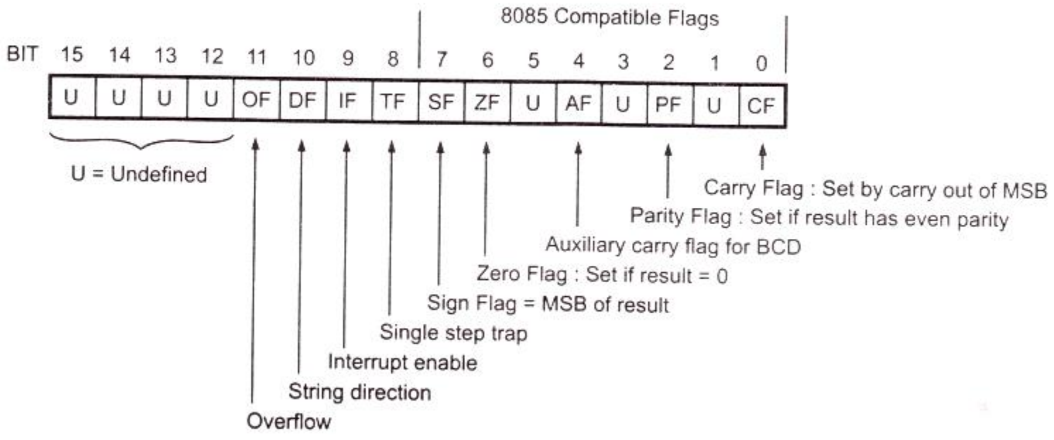

8085 Compatible Flags
BIT 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
U U U U OF DF IF TF SF ZF U AF U PF U CF
U = Undefined
Carry Flag : Set by carry out of MSB
Parity Flag : Set if result has even parity
Auxiliary carry flag for BCD
Zero Flag : Set if result = 0
Sign Flag = MSB of result
Single step trap
Interrupt enable
String direction
Overflow

Fig-3- 8086 flag register bit pattern

Memory segmentation

The physical address of the 8086 is 20-bits wide to access 1 M byte memory locations. However, its registers and memory locations which contain logical addresses are just 16-bits wide. Hence 8086 uses memory segmentation. It treats the 1 M byte of memory as divided into segments, with a maximum size of a segment as 64 Kbytes. Thus any location within the segment can be accessed using 16 bits. The 8086 allows only four active segments at a time, as shown in the Fig. -4- For the selection of the four active segments the 16-bit segment registers are provided by the bus interface unit (BIU) of the 8086.

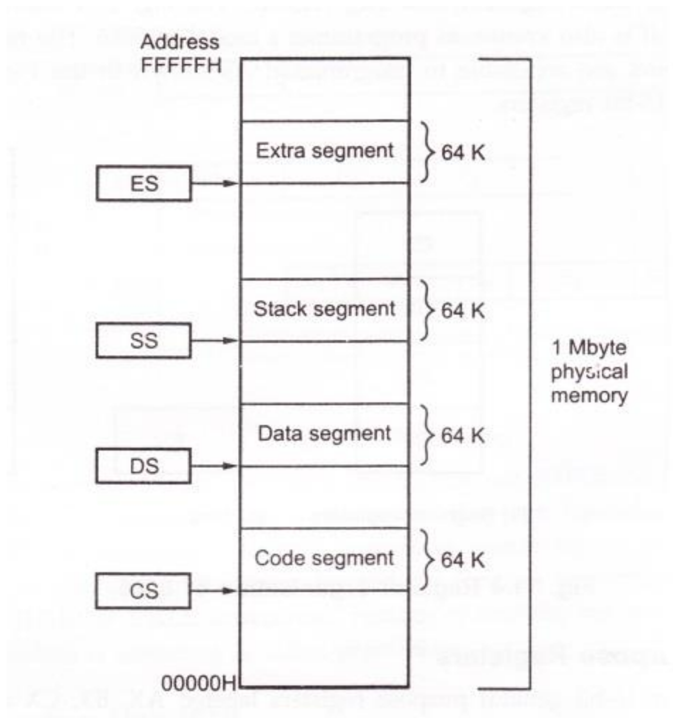


Fig-4- memory segmentation and segment register

- **Segment Registers**
  - Code segment (CS) register
  - Data segment (DS) register
  - Stack segment (SS) register
  - Extra segment (ES) register

These are used to hold the upper 16-bits of the starting addresses of the four memory segments.

For example, the value in CS identifies the starting address of 64 K-byte segment known as code segment. By "starting address", we mean the lowest addressed byte in the active code segment.

- The starting address is also known as **base address** or **segment base**.
- The BIU always inserts zeros for the lower 4 bits (nibble) in the contents of segment register to generate 20-bit base address.

For example, if the code segment register contains 348AH, then code segment will start at address 348A0 H.

- The 16-bit contents of the segment register gives the starting/base address of a particular segment, To address a specific memory location within a segment we need an offset address. The offset address is also 16-bit wide and it is provided by one of the associated pointer or index register.

- **<u>Pointers and Index Registers</u>**

  - pointer registers
    - IP (instruction pointer)
    - BP (base pointer)
    - SP(stack pointer)
  - index registers
    - DI(destination index register)
    - SI (source index register)

The index registers used as a general purpose registers as well as for offset address.

- **<u>Generation of 20-bit Address</u>**

To access a specific memory location from any segment we need 20- bit physical address. The 8086 generates this address using the contents of segment register and the offset register associated with it. The BIU has a dedicated adder for determining physical memory address. Let us see how 8086 access code byte within the code segment.

We know that the CS register holds the base address of the code segment. The 8086 provides an instruction pointer (IP) which holds the 16-bit address of the next code byte within the code segment. The value contained in the IP is referred to as an offset. This value must be offset from (added to) the segment base address in CS to produce the required 20-bit physical address.

The contents of the CS register are shifted by 4 position to the left by inserting 4 zero bits and then the offset (the contents of IP register) are added to the shifted contents of CS to generate physical address. As shown in the Fig. -5-, the contents of CS register are 348AH, therefore the shifted contents of CS register are 348A0H. When the BIU adds the offset of 4214H in the IP to this starting address, we get 38AB4H as a 20-bit physical address of memory. This is illustrated in Fig. -5- (b).

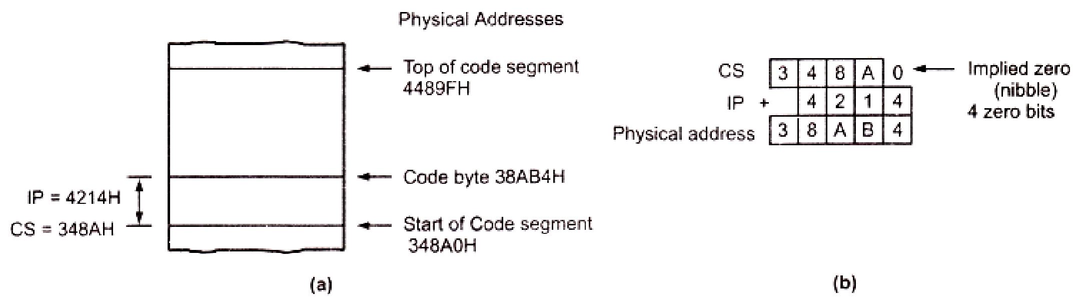

Physical Addresses
Top of code segment 4489FH
Code byte 38AB4H
Start of Code segment 348A0H
IP = 4214H
CS = 348AH
(a)
CS 3 4 8 A 0
IP + 4 2 1 4
Physical address 3 8 A B 4
Implied zero (nibble) 4 zero bits
(b)

Fig-5-

We have seen that how 20-bit physical address is generated within the code segment. In the similar way the 20-bit physical address is generated in the other segments. However, it is important to note that each segment requires particular segment register and offset register to generate 20-bit physical address as shown in table below:

segment	offset	Special purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX,DI,SI,8-bit number, or 16-bit number	Data address
ES	DI for string instruction	String destination address

Example:
If we would like to access memory at the physical address 12345 H we could set DS=1230 H and SI=0045 H. to make a calculation of the physical address by multiplying the segment register by 10 H (or shift it 4 digit to left or adding zeros for the lower 4 bits) and adding the offset memory address (content of SI register) to it :
1230 H * 10 H + 0045H=12345 H (physical address)

- **Logical address and physical address:**

There are three type of addresses :-

- physical address

  The physical address is the 20- bit address that actually put on the address pins of the 8086 microprocessor and decoded by the memory interfacing circuitry. The address can have a range of 00000 H to FFFFF H . this is actual physical location in RAM or ROM within the 1 M byte memory range.

- offset address

  The offset address is a location within 64K-byte segment range . therefore, an offset address can range from 0000 H to FFFF H.

- logical address.

The logical address consists of a segment value and an offset address.

Logical address is specified as **segment: offset**

Ex: Thus the physical address of the logical address A4FB:4872 is

A4FB0 + 4872 = A9822

EX: if the data segment starts at location 1000 H and data reference contains the address 29 H where is the actual data?

```
Offset:          0000 0000 0010 1001                      = 0029 H
Segment:    0001 0000 0000 0000 0000   (shifting 4 bits) = 10000 H
Address:    0001 0000 0000 0010 1001                      = 10029 H
```

Ex:

CS		IP
2500	:	95F3

To execute a program, the 8086 fetch the instructions 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.

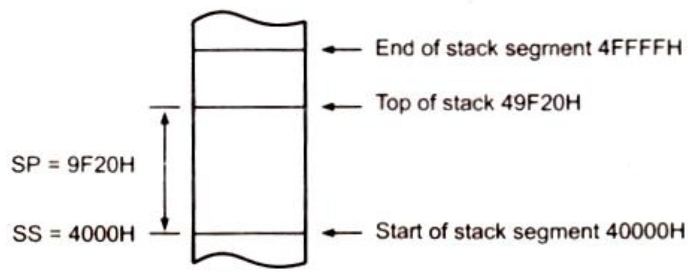
IP contains the offset address. The result 20-bit address is the physical address since it is put on external physical address bus pins to be decoded by the memory decoding circuitry.

The offset address is contained in IP = 95F3 H. the CS contained 2500 H. The logical address is CS:IP or 2500:95F3. The physical address will be 25000+ 95F3= 2E5F3 H

Ex: The stack pointer (SP) register contains the 16-bit offset from the start of the segment to the top of stack. For stack operation, physical address is produced by adding the contents of stack pointer register to the segment base address in SS. To do this the contents of the stack segment register are shifted four bits left and the contents of SP are added to the shifted result. If the contents of SP are 9F20H and SS are 4000H then the physical address is calculated as follows.

SS = 4000H after shifting four bits left SS = 40000H

SS	40000H
+ SP	9F20H
Physical address	49F20H



EX:

If CS= 24F6 H and IP= 634A H, show:

a- The logical address

b- The offset address

And calculate:

c- The physical address

d- The lower range

e- The upper range of the code segment

Solution:

a- 24F6 : 634A

b- 634A H

c- 2B2AA (24F60 +634A)

d- 24F60 (24F60 +0000)

e- 34F5F (24F60 +FFFF)

EX:

If DS=7FA2 H and the offset is 438E H

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- 7FA20 +438E = 83DE H

b- 7FA20 + 0000 = 7FA20

c- 7FA20 + FFFF = 8FA1F

d- 7FA2 : 438E