INTEL 8086 MICROPROCESSOR

1.1 INTRODUCTION

- It is a semiconductor device consisting of electronic logic circuits manufactured by using either a Large scale (LSI) or Very Large Scale (VLSI) Integration Technique.
- It includes the ALU, register arrays and control circuits on a single chip. The microprocessor has a set of instructions, designed internally, to manipulate data and communicate with peripherals.
- The era of microprocessors began in the year 1971, the Intel introduced the first 4-bit microprocessor is 4004. Using this the first portable calculator is designed.
- The 16-bit Microprocessor families are designed primarily to complete with microcomputers and are oriented towards high-level languages. They have powerful instruction sets and capable of addressing mega bytes of memory.
- The era of 16-bit Microprocessors began in 1974 with the introduction of PACE chip by National Semiconductor. The Texas Instruments TMS9900 was introduced in the year 1976. The Intel 8086 was commercially available in the year 1978, Zilog Z800 in the year 1979, The Motorola MC68000 in the year 1980.
- The 16-bit Microprocessors are available in different pin packages. Ex: Intel 8086/8088 40 pin package Zilog Z8001 40 pin package, Digital equipment LSI-II 40 pin package, Motorola MC68000 64 pin package National Semiconductor NS16000 48 pin package.
- The primary objectives of this 16-bit Microprocessor can be summarized as follows.
  1. Increase memory addressing capability
  2. Increase execution speed
  3. Provide a powerful instruction set
  4. Facilitate programming in high-level languages.

1.2 Microprocessor Architecture:

- The 8086 CPU is divided into two independent functional parts, the Bus interface unit (BIU) and execution unit (EU).
  The Bus Interface Unit contains Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue. The BIU sends out address, fetches the instructions from memory, read data from ports and memory, and writes the data to ports and memory.
- **The execution unit:** contains the Data and Address registers, the Arithmetic and Logic Unit, the Control Unit and flags. tells the BIU where to fetch instructions or data from, decodes instructions and executes instruction. The EU contains control circuitry which directs internal operations. A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out. The EU is has a 16-bit ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers. The EU is decoding an instruction or executing an instruction which does not require use of the buses.
  In other words the BIU handles all transfers of data and addresses on the buses for the execution unit.
- **The Queue**: The BIU fetches up to 6 instruction bytes for the following instructions. The BIU stores these prefetched bytes in first-in-first-out (FIFO) register set called a queue. When the EU is ready for its next instruction it simply reads the instruction byte(s) for the instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes.

.

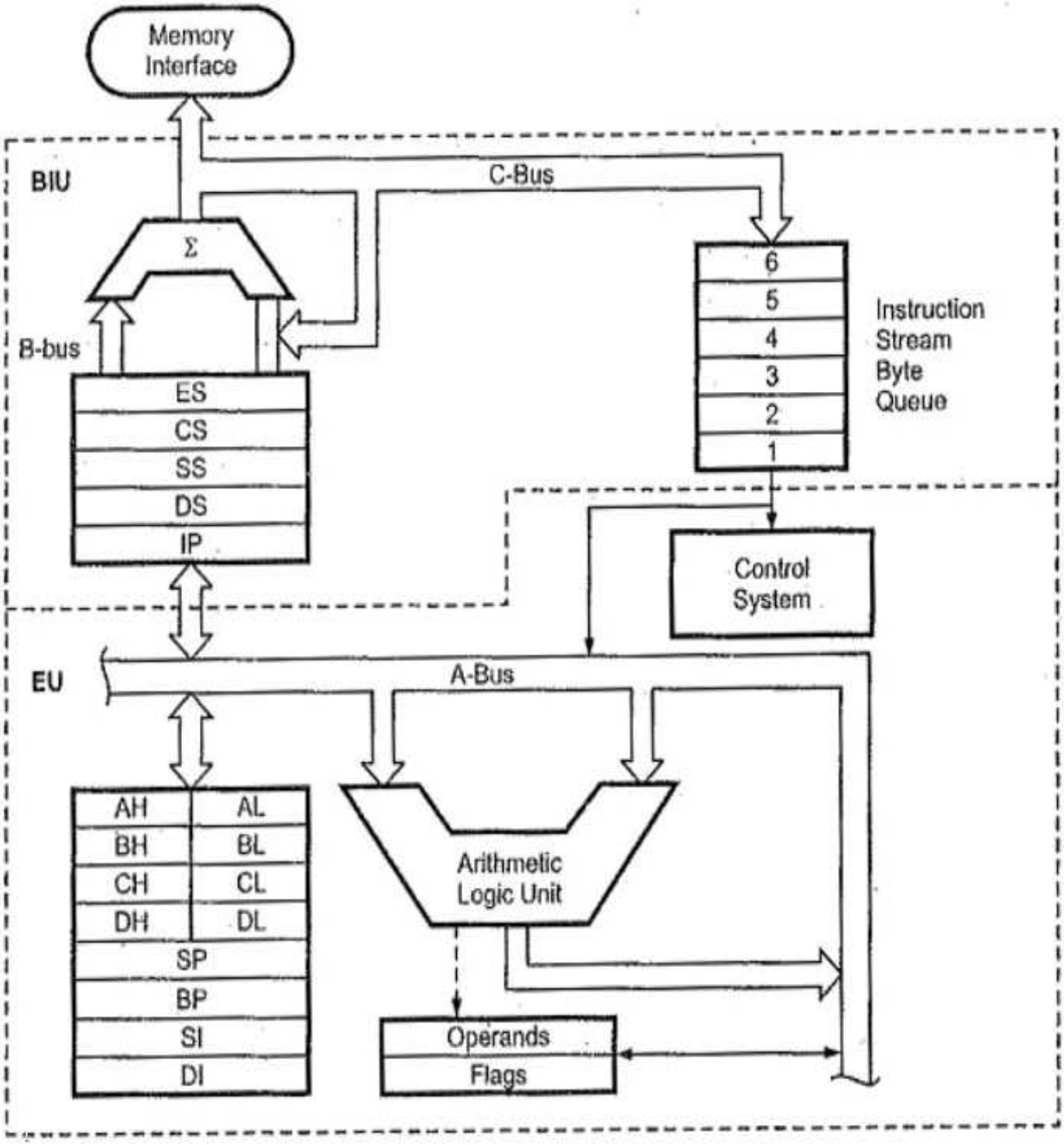

Fig.1.1 8086 Architecture

Except in the case of JMP and CALL instructions, where the queue must be dumped and then reloaded starting from a new address, this prefetch-and-queue scheme greatly speeds up processing. Fetching the next instruction while the current instruction executes is called pipelining.

- **Word Read:** Each of 1 MB memory address of 8086 represents a byte wide location.16-bit words will be stored in two consecutive memory locations. If first byte of the data is stored at an even address**,** 8086 can read the entire word in one operation. For example if the 16 bit data is stored at even address 00520H is 9634H

  MOV BX, [00520H]

  8086 reads the first byte and stores the data in BL and reads the 2nd byte and stores the data in BH

  BL= (00520H) i.e. BL=34H

  BH= (00521H) BH=96H

  If the first byte of the data is stored at an odd address, 8086 needs two operations to read the 16 bit data.

  For example if the 16 bit data is stored at even address 00521H is 3897H

  MOV BX, [00521H]

  In first operation, 8086 reads the 16 bit data from the 00520H location and stores the data of 00521H location in register BL and discards the data of 00520H location In 2nd operation, 8086 reads the 16 bit data from the 00522H location and stores the data of 00522H location in register BH and discards the data of 00523H location.

  BL= (00521H) i.e. BL=97H

  BH= (00522H) BH=38H

- **Byte Read**:

  MOV BH, [Addr]

  For Even Address:

  Ex: MOV BH, [00520H]

  8086 reads the first byte from 00520 location and stores the data in BH and reads the 2nd byte from the 00521H location and ignores it

  BH =[00520H]

  For Odd Address

  MOV BH, [Addr]

  Ex: MOV BH, [00521H]

  8086 reads the first byte from 00520H location and ignores it and reads the 2nd byte from the 00521 location and stores the data in BH

  BH = [00521H]

- **Physical address formation**:

  The 8086 addresses a segmented memory. The complete physical address which is 20-bits long is generated using segment and offset registers each of the size 16-bit.The content of a segment register also called as segment address, and content of an offset register also called as offset address. To get total physical address, put the lower nibble 0H to segment address and add offset address. The fig 1.3 shows formation of 20-bit physical address.

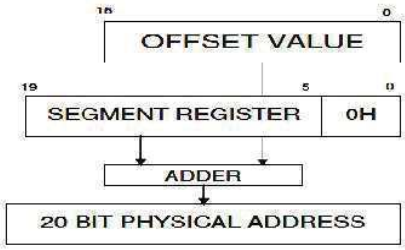

Fig 1.2 Physical Adress formation

- **Register organization of 8086:**

All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes. Fig 1.3

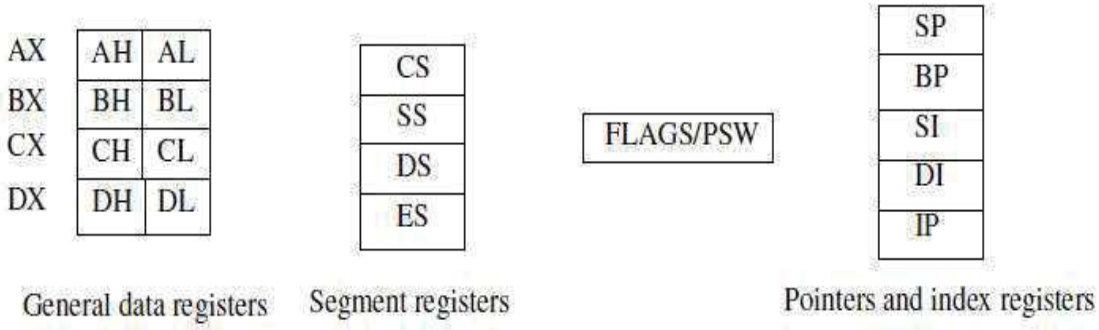

Fig 1.3 Register organization of 8086

- ✓ **AX Register: Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.
- ✓ **BX Register:** This register is mainly used as a **base register**. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.
- ✓ **CX Register:** It is used as default counter - **count register** in case of string and loop instructions.
- ✓ **DX Register: Data register** can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.4. Each segment contains 64Kbyte of memory. There are four segment registers.

- ✓ **Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly.

The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

- ✓ **Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.
- ✓ **Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.
- ✓ **Extra segment** (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory.

- ✓ It also contains data.

Fig1.4. Memory segmentation

- ✓ **Pointers and index registers.**

The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

✓ **Flag Register:**

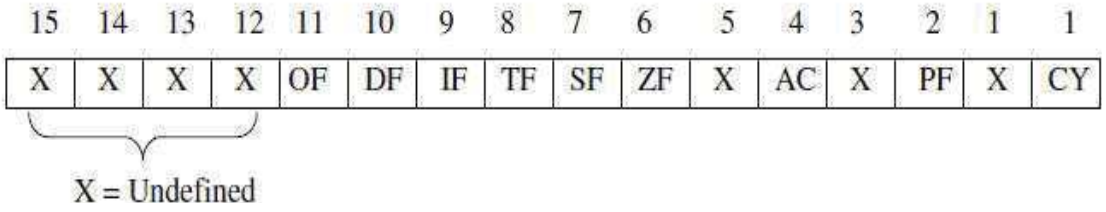

Fig. 1.5 Flag Register

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. The 8086 flag register as shown in the fig 1.5. 8086 has 9 active flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

✓ **Conditional Flags**

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. $D_0 – D_3$) to upper nibble (i.e. $D_4 – D_7$), the AC flag is set i.e. carry given by D_3 bit to D_4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

✓ **Control Flags**

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction SIT and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

1.3 Addressing Modes

The 8086 has 12 addressing modes can be classified into five groups.

- Addressing modes for accessing immediate and register data (register and immediate modes).
- Addressing modes for accessing data in memory (memory modes)

- Addressing modes for accessing I/O ports (I/O modes)
- Relative addressing mode
- Implied addressing mode

✓ **Immediate addressing mode:**

In this mode, 8 or 16 bit data can be specified as part of the instruction - OP Code Immediate Operand

Example 1: MOV CL, 03 H:Moves the 8 bit data 03 H into CL

Example 2: MOV DX, 0525 H: Moves the 16 bit data 0525 H into DX

In the above two examples, the source operand is in immediate mode and the destination operand is in register mode.

A constant such as "VALUE" can be defined by the assembler EQUATE directive such as VALUE EQU 35H

Example: MOV BH, VALUE Used to load 35 H into BH

✓ **Register addressing mode:**

The operand to be accessed is specified as residing in an internal register of 8086. Table 1.1 below shows internal registers, anyone can be used as a source or destination operand, however only the data registers can be accessed as either a byte or word.

Table 1.1 Internal registers of 8086

Register	Operand sizes	
	Byte (Reg 8)	Word (Reg 16)
Accumulator	AL, AH	Ax
Base	BL, BH	Bx
Count	CL, CH	Cx
Data	DL, DH	Dx
Stack pointer	-	SP
Base pointer	-	BP
Source index	-	SI
Destination index	-	DI
Code Segment	-	CS
Data Segment	-	DS
Stack Segment	-	SS
Extra Segment	-	ES

Example 1: MOV DX (Destination Register) , CX (Source Register)

Which moves 16 bit content of CS into DX.

Example 2: MOV CL, DL

Moves 8 bit contents of DL into CL

MOV BX, CH is an illegal instruction.

* The register sizes must be the same.

✓ **Direct addressing mode:**

The instruction Opcode is followed by an affective address, this effective address is directly used as the 16 bit offset of the storage location of the operand from the location specified by the current value in the selected segment register. The default segment is always DS.

The 20 bit physical address of the operand in memory is normally obtained as PA = DS: EA

But by using a segment override prefix (SOP) in the instruction, any of the four segment registers can be referenced,

PA = { CS / DS / SS / ES } : { Direct Address }

Fig 1.6 Physical address generation of 8086

The Execution Unit (EU) has direct access to all registers and data for register and immediate operands. However the EU cannot directly access the memory operands. It must use the BIU, in order to access memory operands.
In the direct addressing mode, the 16 bit effective address (EA) is taken directly from the
displacement field of the instruction.
Example 1: MOV CX, START
If the 16 bit value assigned to the offset START by the programmer using an assembler pseudo instruction such as DW is 0040 and [DS] = 3050. Then BIU generates the 20 bit physical address 30540 H.
The content of 30540 is moved to CL
The content of 30541 is moved to CH
Example 2: MOV CH, START
If [DS] = 3050 and START = 0040
8 bit content of memory location 30540 is moved to CH.
Example 3: MOV START, BX
With [DS] = 3050, the value of START is 0040.
Physical address: 30540
MOV instruction moves (BL) and (BH) to locations 30540 and 30541 respectively.

- ✓ **Register indirect addressing mode:**

The EA is specified in either pointer (BX) register or an index (SI or DI) register. The 20 bit physical address is computed using DS and EA.
Example: MOV [DI], BX register indirect
If [DS] = 5004, [DI] = 0020, [Bx] = 2456 PA=50060.
The content of BX(2456) is moved to memory locations 50060 H and 50061 H.

PA = { CS / DS / SS / ES } = { BX / SI / DI }

Based addressing mode:

PA = { CS / DS / SS / ES } : { BX / or / BP } + displacement

when memory is accessed PA is computed from BX and DS when the stack is accessed PA is computed from BP and SS.
Example: MOV AL, START [BX]
or
MOV AL, [START + BX]
based mode

EA: [START] + [BX]
PA: [DS] + [EA]
The 8 bit content of this memory location is moved to AL.

✓ **String addressing mode:**
The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented (by clearing DF to 0 by CLD instruction) to point to the next byte or word.
Example: MOV S BYTE
If [DF] = 0, [DS] = 2000 H, [SI] = 0500,
[ES] = 4000, [DI] = 0300
Source address: 20500, assume it contains 38
PA: [DS] + [SI]
Destination address: [ES] + [DI] = 40300, assume it contains 45

Indexed addressing mode:

PA = { CS / DS / SS / ES } : { SI / or / DI } + 8 or 16bit displacement

Example : MOV BH, START [SI]
PA : [SART] + [SI] + [DS]
The content of this memory is moved into BH.

Based Indexed addressing mode:

PA = { CS / DS / SS / ES } : { BX / or / BP } + { SI / or / DI } + 8 or 16bit displacement

Example : MOV ALPHA [SI] [BX], CL
If [BX] = 0200, ALPHA – 08, [SI] = 1000 H and [DS] = 3000
Physical address (PA) = 31208
8 bit content of CL is moved to 31208 memory address.

After executing MOV S BYTE,
[40300] = 38
[SI] = 0501 } incremented
[DI] = 0301

✓ **I/O mode (direct):**
Port number is an 8 bit immediate operand.
Example: OUT 05 H, AL
Outputs [AL] to 8 bit port 05 H
I/O mode (indirect):
The port number is taken from DX.
Example 1: IN AL, DX
If [DX] = 5040
8 bit content by port 5040 is moved into AL.
Example 2: IN AX, DX
Inputs 8 bit content of ports 5040 and 5041 into AL and AH respectively.

✓ **Relative addressing mode:**
Example: JNC START

If CY=O, then PC is loaded with current PC contents plus 8 bit signed value of START,
otherwise the next instruction is executed.

- ✓ **Implied addressing mode:**
  Instruction using this mode have no operands.
  Example: CLC which clears carry flag to zero.

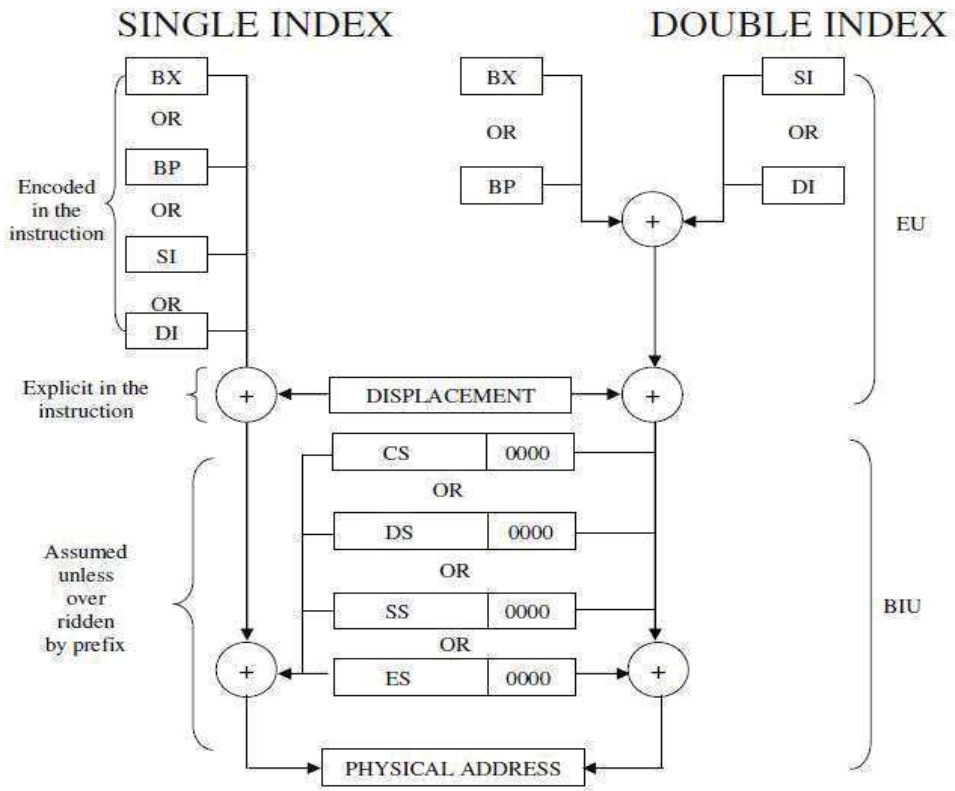

Fig 1.7 Summary of 8086 addressing modes

1.4 INSTRUCTION SET OF 8086

The 8086 instructions are categorized into the following main types.

1. Data Copy / Transfer Instructions
2. Arithmetic and Logical Instructions
3. Shift and Rotate Instructions
4. Loop Instructions
5. Branch Instructions
6. String Instructions
7. Flag Manipulation Instructions
8. Machine Control Instructions

1.4.1 Data Copy / Transfer Instructions:

MOV: