UNIT-1

Introduction to Micro Processors

Definition of microprocessor:

We can define the microprocessor based on 3 things. They are

1) Based on the application of the device.
   The CPU of any microcomputer is called microprocessor.
2) Based on the name of the device.
   A small device which is able to do data processing is called microprocessor.
3) Based on the construction and operation of the device.
   Microprocessor is a VLSI/ULSI chip. It accepts binary data from either an i/p device or from the memory and it access the instruction from the memory and it perform the operation of the received data according to the instruction & produces the results those are sent to either an o/p device or memory.

History of microprocessors: - Main parameter → word length

Definitions of word length: -

1) The no. of bits processed by the CPU at a time are called word length.
2) The no. of bits transmitted or received by the CPU at a time.

   (or)

   The no. of bits identified by the CPU at a time are called word length.

1ST Generation Microprocessors: -For these processors Word length = 4bits.Examples for this generation processors are: 4004, 4040. These processors designed by using PMOS technology.4004 was the first processor in the world. It was introduced in the year 1971 by Intel Corporation. It is designed basically for calculator (Abacus).4040 is the advanced version in 4-bit processors. It was introduced in the year 1972. The difference between the 4004 and 4040 is the operating clock frequency.

2ND Generation Microprocessors: - For these processors Word length = 8bits.Examples for this generation processors are:8008, 8080, 8085, M6800 and Z80.The processor 8080 requires 3-power supplies for its operation, those are +12V, -12V and +5V.The 8085 Introduced in 1975 this is most popular processor in the 2nd generation processors. NMOS technology was used for the designing of this processor. The Speed of these processors is high when compared to 1st generation microprocessors.

3RD Generation Microprocessors: - For these processors Word length = 16bits. Examples for this generation processors are: 8086, 8088, 80186, 80286, M68000 and Z8080. The first 16-bit processor is 8086it was introduced in the year 1978.The HMOS technology was used for 8086 processor. The 8088 is the combination of 8085 & 8086 features. Its operation like 8086 when it performs internal operations and its operation is like 8085 when it performs external devices related operations. The speed of these processors is high as compared to previous generation (2nd generation) processors.

4TH Generation Microprocessors: - For these processors Word length = 32bits.Examples for this generation processors are: 80386, 80486, M68020, M68030.Virtual memory and cache memory concepts were introduced from 80386 onwards. Instruction pipe-lining concept was introduced from 80486 onwards. In this 80486 processor there is a 4-stage Instruction pipe lining. The speed of these processors is high when compared to 3rd generation microprocessors.

5TH Generation Microprocessors: - For these processors Word length = 64bits.Examples for this generation processors are: Pentium [80586], Pentium pro, Pentium -2 etc. The Speed of these processors is high when compared to 4th generation microprocessors.

Features of 8086: -

1. It is a 16-bit Microprocessor. It's ALU, internal registers works on 16-bit binary word.
2. It was implemented in the year 1978 by Intel corp. by using HMOS (hybrid metal oxide semi-conductor or high speed MOS or high density MOS) technology.
3. 8086 processor has 20 address lines A19-A0, and 16 data lines D15-D0. The data lines are multiplexed with lower order 16 address lines, and then the multiplexed address and data lines areAD15-AD0. The remaining higher order 4 address lines A16-A19 are multiplexing with the status lines S3-S6.
4. 8086 processor is available in 40-pin DIP(Dual in line Package).
5. The operating clock frequencies of 8086 processors are 5MHz, 8MHz, and 10MHz.
6. 8086 processor supports 256 interrupts.
7. It supports full duplex asynchronous serial communication and half duplex synchronous serial communication.
8. 8086 processor have 4 general purpose registers, 4 segment registers, 3 pointer registers, 2 index registers and 1 flag register. Size of all these registers is 16-bit.
9. 8086 processor supports segmented version of memory (1MB size). Size of each segment is 64KB.
10. 8086 operates in two different modes. (1) Minimum mode or Single-Processor mode and (2) Maximum mode or Multi-Processor mode.
11. 8086 includes few features, which enhance multiprocessing capability (it can be used with math coprocessors like 8087, I/O processor 8089 etc.).
12. 8086 available in three different versions 8086, 8086_1 and 8086_2.

Architecture of 8086: - The Internal Block Diagram of 8086

The entire architecture is divided into two independent functional parts. They are

1) BIU: Bus Interface Unit 2) EU: Execution Unit

BIU (Bus Interface Unit): -

The functions are performed by BIU

a). Fetch the instruction or data from memory.
b). Write the data to memory.
c). Write the data to the port.
d). Read data from the port.

BIU has the following functional parts these are

a). Summer circuit (or) adder circuit.

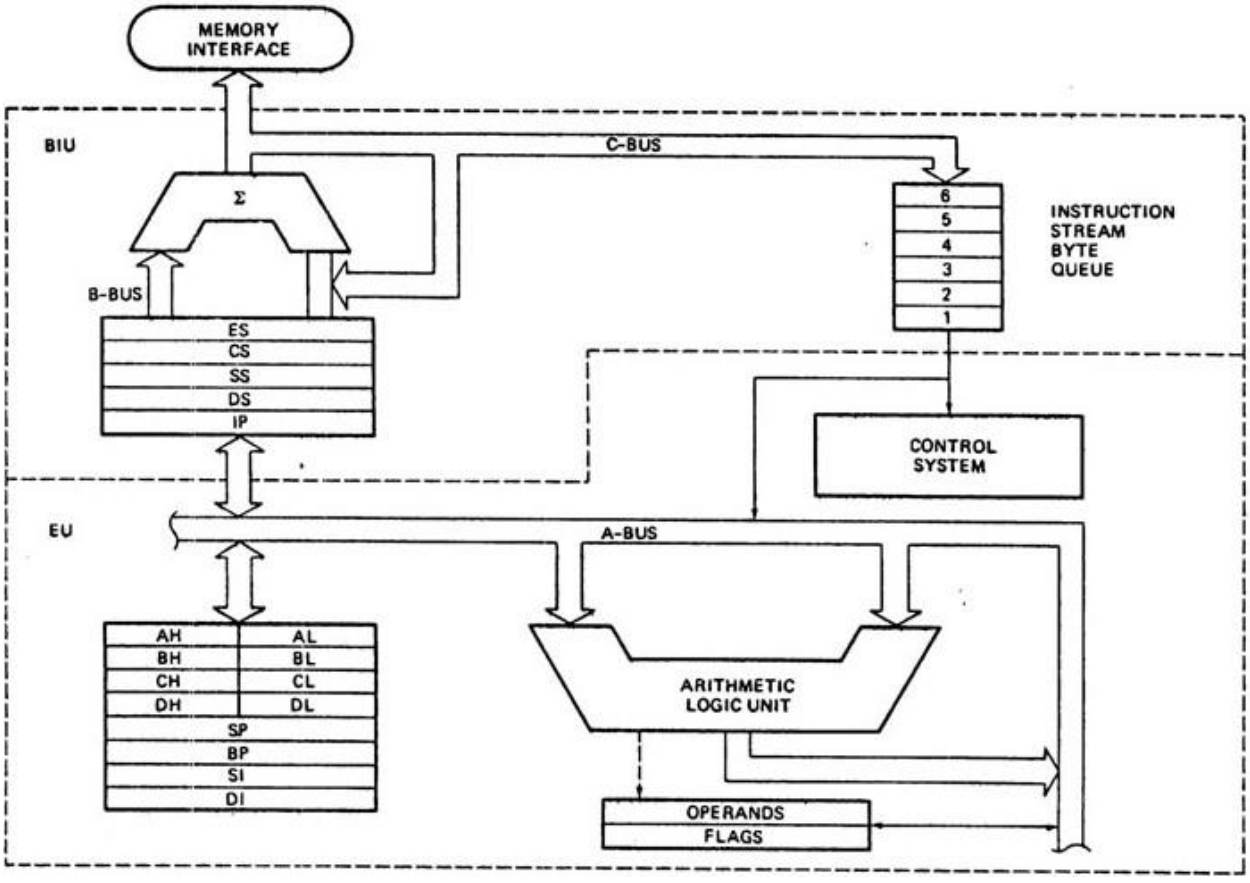
b). Segment registers & IP.

c). 6B instruction pre-fetch queue.

a) Summer Circuit: - It produces a 20-bit physical address.

b) Segment Registers: -The BIU contains four 16-bit segment registers. They are: the extra segment (ES) register, the code segment (CS) register, the data segment (DS) register, and the stack segment (SS) register. These segment registers are used to hold the upper 16 bits of 20-bit **Physical Address.** It is called the **Segment Base** and it is also called the **Starting Address** or **Base Address** for each of the segment. The part of a segment starting address stored in a segment register is often called the **Segment Base**.

Instruction pointer: - Its functionality is similar to that of PC (**Program Counter**). It holds the address of next instruction byte to be fetched from the code-segment. It means it gives **offset address** of the instruction code bytes in the **Code Segment**.



6B Instruction Pre-Fetch Queue: The functions of instruction Queue in BIU of 8086 are,

1. To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory.

2. All six bytes are then held in first in first out 6 byte register called instruction queue.

3. Then all bytes have to be given to EU one by one.

4. This pre fetching operation of BIU may be in parallel with execution operation of EU, which improves the speed execution of the instruction

If the Queue size is greater than 6B then wait state period of processor is increased because any instruction in the queue is related to **Branching,** then instructions in the queue must be cleared because the next instructions must be taken from the branch specified location. In this period processor must be in wait state.

If the queue size less than the 6B then the maximum size instruction in the 8086 processor is unable to execute because maximum length of instruction is 6B.

EU (Execution Unit): -

The functions are performed by EU

a). It tells the BIU where to fetch instructions or data from
b). Decodes instructions, and
c). Executes instructions.

The EU contains the control circuitry to perform various internal operations. A decoder in EU decodes the instruction which is fetched from memory and to generate different internal or external control signals required to perform the operation. EU has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit.

EU has the following functional parts these are

a). ALU
b).Register Set.
c). Operand & Flag Register.
d). Control System.

a).ALU (Arithmetic and Logical Unit):- It performs Arithmetic and Logical operations on 8 bits/16bits. The Bit Capacity of ALU is 16bits.

It can do the following arithmetic operations

i) Addition ii) Subtraction iii) Multiplication
iv) Division v) Increment vi) Decrement

Arithmetic operations may be performed on four types of numbers

- Unsigned binary numbers
- Signed binary numbers (Integers)
- Unsigned packed decimal numbers
- Unsigned unpacked decimal numbers

The ALU can also perform logical operations such as

i) NOT ii) AND iii) OR
iv) EX-OR v) TEST vi) Logical Shift
vii) Arithmetic Shift viii) Circular Shift (or) Rotate

b).Register Set:- It is used to hold the 16-bit information. The information is address, data or result of some operation.

In 8086 there are 14 registers. These are divides into 5 groups and these are User accessible registers, each register size is 16-bit.

I. General Purpose Registers [GPRs] ---4
II. Segment Registers---4
III. Pointer Registers---3
IV. Index Registers---2
V. Flag register---1

c).Operand & Flag Register: -Operand holds the result produced by ALU. Flags are also called as PSW (program status word) of 8086. Each single bit is called flag. 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.It is also called PSW of 8086.

U	U	U	U	OV	DF	IF	TF	SF	ZF	U	AC	U	PF	U	CY

U--- Unused bit positions (7)
Remaining 9 are active bit positions. Active flags of 8086.
OV=overflow flag DF=direction flag TF=trap flag IF=interrupt flag SF=sign flag
ZF=zero flag AC=auxiliary carry flag PF=parity flag CY=carry flag
9 active flags can be divided into 2 groups.
a) Conditional flags----6 b) Control flags----3

a) Conditional Flags: -

- Also called status flags.
- Gives the information of conditions of the result produced by the ALU.
- Gives the status of ALU after completion of arithmetic/logical operations.
- All the flags in LSByte are status flags along with OV in MSByte.

CY=1: - When

1) If the processor performs any addition operation, the result contains a carry [final carry]. Carry generated at MSB position.
2) If the processor execute subtraction related instructions, if the minued value is less than subtrahend.

PF=1: - When

1) The parity flag is set to 1, if the result contains even no. of 1's in its LSByte.

AC=1: - When

1) In addition process, the auxiliary carry is generated from the least significant nibble, i.e. at b3.
2) In subtraction process, the auxiliary carry flag indicates if the borrow is taken by least significant nibble, i.e. by b3 from b4.
   This is not a general-purpose flag; it is used internally by the processor to perform Binary to BCD conversion

ZF=1: - When

1) The zero flag is set to1 when the result of any arithmetic or logical operation is zero.

SF=1: - When

1) Sign flag is set to 1, if the sign of result is –ve. For signed operations, for unsigned operations, the MSB of the result is 1. It means the MSB of the result is copied to sign flag.

OV=1: - When

1) OV is set to 1, if the size of the result is greater than the size of the destination.
2) If any arithmetic/logic operation satisfies the overflow rule, OV=1.

b) Control Flags: - Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows: These flags are set or reset by user.

1. Trap Flag (TP): -

a. It is used for single step control.

b. It allows user to execute one instruction of a program at a time for debugging.

c. When trap flag is set, program can be run in single step mode.

2. Interrupt Flag (IF): -

a. It is an interrupt enable/disable flag.

b. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.

c. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

3. Direction Flag (DF): -

a. It is used in string operation.

b. If it is set, string bytes are accessed from higher memory address to lower memory address. Processor operates in auto-increment mode of operation.

c. When it is reset, the string bytes are accessed from lower memory address to higher memory address. Processor operates in auto-decrement mode of operation.

d).Control System: - It is divided into 2 parts. They are

1) Decoding circuit----it decodes the instruction.
2) Timing circuit---- it generates control signals at appropriate times.

Register Organization of 8086: -

In general there are 2 types of registers in any CPU. They are

1. Machine accessible registers – These Registers are only used by the Machines. User was unable to use these registers by programming instructions.

2. User accessible registers: -These Registers are used by the programmer in programming. These are called by the user in program by using programming instructions.

The 8086 processor also contains both types of registers.

In 8086 there are 14 registers. These are divides into 5 groups and these are User accessible registers, each register size is 16-bit.

I. General Purpose Registers [GPRs] ---4
II. Segment Registers---4
III. Pointer Registers---3
IV. Index Registers---2
V. Flag register---1

I). General Purpose Registers [GPRs]: -There are 4 GPRs. They are

i). AX---Accumulator Register ii). BX---Base Register
iii). CX---Counter Register iv). DX---Data Register

- These registers are able to hold data, address or results on temporary basis.
- After completion of any operation by CPU, the result is loaded on to GPRs in some cases.
- For 8-bit operations, each 16-bit register can be divides into two 8-bit registers.
  AX→ AH+AL BX→ BH+BL
  CX→ CH+CL DX→ DH+DL

Special Functions of GPRs: -

- **AX (Accumulator): -**
  a) AX acts as source/destination for some arithmetic/logical operations.
  b) If the processor is communicating with any input or output device, then entire communication is done through accumulator register.
  c) It acts as interface between CPU and I/O device.
- **BX (Base Register): -**
  This register is used to hold the off-set value or the part of off-set value of an operand in some addressing modes.
- **CX (Counter Register): -**
  a) This register is used as a counter to perform repeated operations by using **loop** instruction.
  b) If the processor performs any **string** related operations, count value for these operations must be the content of CX register.
- **DX (Data Register): -**
  a) DX register is used to hold the address of an I/O device in I/O indirect addressing.

b) Whenever the processor executes 16bit*16bit multiplication, the result size is 32-bit, in that higher order 16-bits are loaded onto DX and lower order 16-bits are loaded on AX.
c) If the processor performs 32-bit/16-bit division, in that dividend 32-bit, higher order 16-bit are content of DX. After completion of division remainder is loaded onto DX.

II). Segment Registers: -There are 4 segment registers; each register size is 16-bit.

1) CS[code segment registers]
2) DS[data segment registers]
3) SS[stack segment registers]
4) ES[extra segment registers]

These segment registers are used to hold the upper 16 bits of the **Starting Address** or **Base Address** for each of the segments. The part of a segment starting address stored in a segment register is often called the **Segment Base**.

Advantages due to Memory Segmentation: -

1) By using 16bit registers, we can identify any memory location in 1MB main memory.
2) By memory segmentation, we can use memory more efficiency.
3) Segmented version memory supports parallel processing without any difficulty.
4) Segmentation gives more security.

All 4 segment registers give the base addresses of selected segments.
Segments starting address are like→00000H, 10000H, 20000h, 30000H…. F0000H
Offset address range in any segment is →from 0000H to FFFFH
Address range of total 1MB is →from 00000H to FFFFFH

Physical Address=Base Address * 10 + Offset Address

No. of locations are getting down from segment base to desired address in the segment is called **Offset Address**.
Physical Address means actual location address in the memory system.

III). Pointer Registers: -There are 3 pointer registers, with 16-bit length, those registers are

1) Instruction pointer(IP)
2) Stack pointer(SP)
3) Base pointer(BP)

IP (Instruction Pointer): - Holds the address of next instruction byte in code segment. It provides offset address in code segment.
SP (Stack Pointer): - Gives the offset value of stack segment (SS). It holds the address of the Top of the Stack. (Or) it holds the address of the stack where the last stack related operation is performed.
BP (Base Pointer): - Gives the offset value of DS or SS. It is used primarily to access parameters passed via the stack.

IV) Index Registers: - There are 2 registers, each 16-bit size.

1) SI (Source index register)
2) DI (Destination index register)

- ❖ SI,DI are used as offset registers for data registers.
- ❖ These are used for string related operations
- ❖ If the processor executes any string related operation, the source address of the string is provide by the combination of DS and SI registers. SI provides Offset address relative to DS.
- ❖ Similarly, destination address is provided by the combination of ES and DI registers. DI provides Offset address relative to ES

V) 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.It is also called PSW of 8086.

U	U	U	U	OV	DF	IF	TF	SF	ZF	U	AC	U	PF	U	CY

U--- Unused bit positions (7)

Remaining 9 are active bit positions. Active flags of 8086.

OV=overflow flag DF=direction flag TF=trap flag IF=interrupt flag SF=sign flag

ZF=zero flag AC=auxiliary carry flag PF=parity flag CY=carry flag

9 active flags can be divided into 2 groups.

b) Conditional flags----6 b) Control flags----3

a) Conditional Flags: -

- ❖ Also called status flags.
- ❖ Gives the information of conditions of the result produced by the ALU.
- ❖ Gives the status of ALU after completion of arithmetic/logical operations.
- ❖ All the flags in LSByte are status flags along with OV in MSByte.

CY=1: - When

3) If the processor performs any addition operation, the result contains a carry [final carry]. Carry generated at MSB position.
4) If the processor execute subtraction related instructions, if the minued value is less than subtrahend.

PF=1: - When

2) The parity flag is set to 1, if the result contains even no. of 1's in its LSByte.

AC=1: - When

3) In addition process, the auxiliary carry is generated from the least significant nibble, i.e. at b3.
4) In subtraction process, the auxiliary carry flag indicates if the borrow is taken by least significant nibble, i.e. by b3 from b4.

   This is not a general-purpose flag; it is used internally by the processor to perform Binary to BCD conversion

ZF=1: - When

2) The zero flag is set to1 when the result of any arithmetic or logical operation is zero.

SF=1: - When

2) Sign flag is set to 1, if the sign of result is –ve. For signed operations, for unsigned operations, the MSB of the result is 1. It means the MSB of the result is copied to sign flag.

OV=1: - When

3) OV is set to 1, if the size of the result is greater than the size of the destination.
4) If any arithmetic/logic operation satisfies the overflow rule, OV=1.

b) Control Flags: - Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows: These flags are set or reset by user.

1. Trap Flag (TP): -

a. It is used for single step control.

b. It allows user to execute one instruction of a program at a time for debugging.

c. When trap flag is set, program can be run in single step mode.

2. Interrupt Flag (IF): -

a. It is an interrupt enable/disable flag.

b. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.

c. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

3. Direction Flag (DF): -

a. It is used in string operation.

b. If it is set, string bytes are accessed from higher memory address to lower memory address. Processor operates in auto-increment mode of operation.

c. When it is reset, the string bytes are accessed from lower memory address to higher memory address. Processor operates in auto-decrement mode of operation.

Addressing Modes of 8086: -

8086 supports 15 addressing modes. An Addressing mode shows how the data is represented in the instruction.

1. Immediate Addressing Mode: -

a) In this AM instructions, the data is directly placed in the source operand field of an instruction.
b) The immediate data size is either 8bit or 16bit.
c) The length of the instruction for this AM is either 3byte or 4byte, if data is moved to register.
d) If data is moved to a memory location, then the length of instruction is from 3bytes-6bytes.
Ex: MOV [BX], 2050H

2. Register Addressing Mode: -

a) In this AM, data is represented in the source operand field through a register.
b) We can use any register except IP.
c) Length of this AM instruction is 2bytes.
Ex: MOV AX, BX

3. Direct Addressing Mode: -

a) In this AM, the address of the data is directly placed in the operand fields of an instruction.
b) In this addressing mode, the address of the data is nothing but the memory location address.
Ex: MOV AX, [2050]

4. Register Indirect Addressing Mode: -

a) In this AM, the address of the data indirectly represented in operand fields of the instruction by using registers.
b) The registers must be BX, BP registers.

Ex: MOV AX, [BX].

5. Register Relative Addressing Mode: -

In this AM instruction, the operand address is provided by the combination of base address and displacement value.

Ex: MOV AX, 50H[BX]

6. Based Indexed Addressing Mode: -

In this AM, operand address is provided by combination of base register and index register.

Ex: MOV AX, [BX][SI].

7. Based Indexed Displacement Addressing Mode: -

In this AM, operand address is provided by the combination of base reg and index reg and displacement value.

Ex: MOV AX, 50H[BX][SI]

8. Implied Addressing Mode: -

a) In this AM, the operand implicitly represented in the operation code of an instruction.
b) Length of the instruction in this AM is 1byte.
c) Ex: STD (set DF)

9. I/O Direct Addressing Mode: -

In this AM, the I/O device address is directly placed in the operand field of instruction.

Ex: IN AX, 2050H
OUT 3050H, AL

10. I/O Indirect Addressing Mode: -

In this AM, the device address placed in operand filed through a register in the instruction.
Ex: IN AL, [DX]

11. Indexed Addressing Mode: -

In this AM, the data address is represented in the operand field through an index register in an instruction. It is useful when the processor executes string related operations.It is similar to register indirect AM.
Ex: MOV AX, [SI].

12. Intra Segment Direct Addressing Mode: -

If the both source and destination appears in same segment then it is called intra segment.
In this AM, the destination address is directly placed in the operand field of the instruction.
Ex: JMP 2050H

13. Inter Segment Direct Addressing Mode: -

If the both source and destination appears in different segment then it is called inter segment.
In this AM, the destination address is directly placed in the operand field of the instruction.
Ex: JMP 3000:4050H.

14. Intra Segment Indirect Addressing Mode: -

In this AM, the destination address is represented in the operand filed through a register or a memory location in an instruction.
Ex: JMP word ptr [2050H]

15. Inter Segment Indirect Addressing Mode: -

In this AM, the destination address is represented in the operand filed through a register or a memory location in an instruction.
Ex: JMP dword ptr [2000H].

Instruction Set of 8086:-

Instructions in the instruction set of 8086 are classified in to different types based on the following parameters.

1. Based on number of operand address fields in the instruction.
2. Based on type of operation.

I. Based on Number of Operand Address Fields in the Instruction: -

In this type of classification instructions can be divided into three groups. Those are

1. Zero operand address field Instructions
2. One operand address field Instructions
3. Two operand address field Instructions

In **Zero operand address field instructions** there is no operand fields in the instruction, only opcode part is only exist in the instruction. Ex:- NOP, HLT, STC, WAIT etc.

In **One operand address field instructions** only one operand is exist in the instruction, the another operand for the instruction are the default registers (AX or DX) these are not represented in the operand field of the instruction. Ex:- IN AL, OUT AX, DIV BX, and MUL CL etc.

In **Two operand address field instructions** both operands of the instruction is directly represented in the instruction operand fields. Ex:- MOV AX,BX, ADD AX, DX, SUB AX, [BP] etc.

II. Based on Type of Operation: -

In this type of classification instructions can be divided into nine groups. Those are

1. Data Transfer instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Branching Instructions
5. Machine Control Instruction
6. Flag Manipulation Instructions
7. Loop Related Instructions
8. Shift and Rotate Instructions
9. String Related Instructions

1. Data Transfer Instructions:-

Data Transfer instructions are also called as Data Movement Instructions or Data Copying Instructions. If the processor executes any **Data Transfer Instruction there is no effect on the Flag Register Status**. The **Data Transfer Instructions** are those, which transfers the DATA (means copy of data) from any one source to any one destination.

The 8086 processor doesn't support the following types of Data transfer Operations

1. Memory to Memory data transfer
2. Immediate data to any Segment Register

The following are the data transfer Instructions in the 8086 instruction set

General – Purpose Byte or Word Transfer Instructions	Simple Input and Output Port Transfer Instructions	Special Address Transfer Instructions	Flag Transfer Instructions
MOV PUSH POP XCHG XLAT	IN OUT	LEA LDS LES	LAHF SAHF PUSHF POPF

MOV: - Copy byte or word from specified source to specified destination. The source operand is either an immediate data or a register or a memory location. The destination is either a register or a memory location.

Ex: - MOV CX, 037A H
MOV AL, BL
MOVBX, [0301H]

PUSH: - Copy specified word to top of stack. The specified word is any 16-bit register content or any memory location contents. For this instruction execution the content of SP is decrement by '2'.

Ex: - PUSH BX
PUSH [1250H]

POP: - Copy word form top of stack to specified destination. Destination can be a general purpose register, segment register (except CS) or memory location. For this instruction execution the content of SP is increment by '2'.

Ex: - POP CX
POP [2050H]

XCHG: - This instruction exchanges Source with Destination. It cannot exchange two memory locations directly. Exchange bytes or exchange words.

Ex: - XCHG DX, AX
XCHG DX, [5040H]

XLAT:-Translate a byte using table in the Memory

IN: - Copy a byte or word from specific Input Port to Accumulator.

Ex: - IN AX, 0050H
IN AL

OUT: - Copy a byte or word from accumulator to specific Output Port.
Ex: - OUT 0050H, AX
OUT AL

LEA: - Load effective address of operand into specified register (16-bit).
Ex: - LEA BX, [SI]

LDS: - It loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment base is placed in DS. To use this instruction the word at the lower order memory address must contain the offset and the word at the higher order address must contain the segment base address.
Ex: - LDS BX, [0302 H]

LES: - It loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. This instruction is very similar to LDS except that it initializes ES instead of DS.
Ex: - LES DX, [1030 H]

LAHF: - Load AH with the low byte of flag register.

SAHF: - Store AH register to low byte of flag register.

PUSHF: - Copy flag register to top of stack.

POPF: - Copy word to top of stack to flag register.

2 . Arithmetic Instructions: -

These instructions are useful to perform Arithmetic calculations, such as addition, subtraction, multiplication, division, increment and decrement. They are again classified into four groups. They are:

Addition Instructions	Subtraction Instructions	Multiplication, Increment, & Decrement Instructions	Division Instructions
ADD ADC INC AAA DAA	SUB SBB DEC NEG CMP AAS DAS	MUL IMUL AAM INC DEC	DIV IDIV AAD CBW CWD

ADD Des, Src: - It adds a byte to byte or a word to word. It affects AF, CF, OF, PF, SF, ZF flags.
Ex: - ADD AL, 74H
ADD DX, AX
ADD AX, [BX]

ADC Des, Src: - It adds the two operands with CF. It effects AF, CF, OF, PF, SF, ZF flags.

Ex: - ADC AL, 74H
ADC DX, AX
ADC AX, [BX]

SUB Des, Src: - It subtracts a byte from byte or a word from word. It affects AF, CF, OF, PF, SF, ZF flags. For subtraction, CF acts as borrow flag.

Ex; - SUB AL, 74H
SUB DX, AX
SUB AX, [BX]

SBB Des, Src: - It subtracts the two operands and also the borrow from the result. It affects AF, CF, OF, PF, SF, ZF flags.

Ex: - SBB AL, 74H
SBB DX, AX
SBB AX, [BX]

INC Src: - It increments the byte or word by one. The operand can be a register or memory location. It affects AF, OF, PF, SF, ZF flags. CF is not affected.

Ex: - INC AX
INC [BX]
INC [2050H]

DEC Src: - It decrements the byte or word by one. The operand can be a register or memory location. It affects AF, OF, PF, SF, ZF flags. CF is not affected.

Ex: - DEC AX
DEC [BX]
DEC [2050H]

AAA (ASCII Adjust after Addition): - The data entered from the terminal is in ASCII format. In ASCII, 0 – 9 are represented by 30H – 39H.This instruction allows us to add the ASCII codes. This instruction does not have any operand. Similarly the following are the ASCII related instructions.

Other ASCII Instructions:
AAS (ASCII Adjust after Subtraction)
AAM (ASCII Adjust after Multiplication)
AAD (ASCII Adjust Before Division)

DAA (Decimal Adjust after Addition): - It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number. It only works on AL register. This Instruction is belongs to Implied addressing mode.

DAS (Decimal Adjust after Subtraction): - It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number. It only works on AL register. This Instruction is belongs to Implied addressing mode.

NEG Src: It creates 2's complement of a given number. That means, it changes the sign of a number. The source content is subtracted from the Zero. The source is a register or a memory location.

Ex: - NEG AX NEG [1250H] NEG [BX]

CMP Des, Src: It compares two specified bytes or words. The Src and Des can be a constant, register or memory location. Both operands cannot be memory locations at the same time. And immediate data may not be destination. The comparison is done simply by internally subtracting the source from destination. The value of source and destination does not change, but the flags are modified to indicate the result.

Ex:- CMP AX, BX
CMP AX, [2050H]
CMP BX, 3050H

MUL Src: It is an unsigned multiplication instruction. It multiplies two bytes to produce a word or two words to produce a double word. This instruction assumes one of the operand in AL or AX. Src can be a register or memory location. For two bytes multiplication the result size is a word (16-bit) that is default stored in the AX register. Similarly for two words (16-bit) are multiplied the result size is 32-bit that is stored in the combination of DX and Ax registers.

Ex;- MUL BL
MUL [2050H]
MUL CX

IMUL Src: It is a signed multiplication instruction. The conditions for this are similar to the MUL instruction.

DIV Src: It is an unsigned division instruction. It divides word by byte or double word by word. The dividend is AX, divisor is Src and the result is stored as: AH = remainder AL = quotient. It is for word by byte division (16-bit / 8-bit). But for double word by word division (32-bit / 16-bit) the dividend is the content of combination of two registers DX and AX, the divisor is Src and the result is stored as: AX= quotient and DX=remainder.

Ex: - DIV BL
DIV [2050H]
DIV CX

IDIV Src: It is a signed division instruction. The conditions for this are similar to the DIV instruction.

CBW (Convert Byte to Word): This instruction converts byte in AL to word in AX. The conversion is done by extending the sign bit of AL throughout AH. This Instruction is belongs to Implied addressing mode.

CWD (Convert Word to Double Word): This instruction converts word in AX to double word in DX: AX. The conversion is done by extending the sign bit of AX throughout DX. This Instruction also belongs to implied addressing mode.

3. Logical Instructions & Shift and Rotate Instructions: -

These instructions are used to perform Bit wise operations. These instructions are also called as Bit-Manipulation Instructions. The following are the Logical instructions in 8086 instruction set.

Logical Instructions	Shift Instructions	Rotate Instructions
NOT AND OR XOR TEST	SHL / SAL SHR SAR	ROL ROR RCL RCR

NOT Src: It complements each bit of Src to produce 1'scomplement of the specified operand. The operand can be a register or memory location.

Ex: - NOT AX
NOT [2050H]

AND Des, Src: It performs Logical AND operation of Des and Src. Src can be immediate number, register or memory location. Des can be register or memory location. Both operands cannot be memory locations at the sametime.CF and OF become zero after the operation.PF, SF and ZF are updated.

Ex: - AND AX, BX
AND AX, [3050H]
AND AX, 1200H
AND [3040H], 1250H

OR Des, Src: It performs Logical OR operation of Des and Src. Src can be immediate number, register or memory location. Des can be register or memory location. Both operands cannot be memory locations at the sametime.CF and OF become zero after the operation.PF, SF and ZF are updated.

Ex: - OR AX, BX
OR AX, [3050H]
OR AX, 1200H
OR [3040H], 1250H

XOR Des, Src: It performs Logical XOR operation of Des and Src. Src can be immediate number, register or memory location. Des can be register or memory location. Both operands cannot be memory locations at the sametime.CF and OF become zero after the operation.PF, SF and ZF are updated.

Ex: - XOR AX, BX
XOR AX, [3050H]
XOR AX, 1200H
XOR [3040H], 1250H

4. Shift and Rotate Instructions:

SHL Des, Count: It shift bits of byte or word left, by count. It puts zero(s) in LSBs.MSB is shifted into carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register.

Ex: - SHL AX
SHL AX, 02H

SHR Des, Count: It shift bits of byte or word right, by count. It puts zero(s) in MSBs.LSB is shifted into carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register.

Ex: - SHR AX
SHR AX, 02H

ROL Des, Count: It rotates bits of byte or word left, by count.MSB is transferred to LSB and also to CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register.

Ex: - ROL AX
ROL AX, 02H

ROR Des, Count: It rotates bits of byte or word right, by count.LSB is transferred to MSB and also to CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register.

Ex: - ROR AX
ROR AX, 02H

NEG Des: It is an instruction which forms the 2's complement (i.e., 1's complement + 1) of an operand.

Ex: - NEG AX
NEG [2002H]

TEST Des, Src: This instruction ANDs the contents of a source byte or word with the contents of specified destination byte or word. Flags are updated but neither operand is changed .TEST instruction is often used to set flags before a condition jump instruction.

Ex: -TEST AL, BH
TEST CX, 0001H

SAL Des, Count: It Arithmetic shift bits of byte or word left, by count. It puts zero(s) in LSbs. MSb is shifted into carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register. This type of operation already exists in SHL instruction, so SAL is equal to SHL.

Ex: - SAL AX
SAL AX, 02H

SAL Des, Count: It Arithmetic shift bits of byte or word Right, by count. It puts Sign bit as it is in the MSb position and he next vacant positions are copied with sign bit value. LSb is shifted into carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in CL register. Because of this operation SAR is not equal to SHR.

Ex: - SAR AX
SAR AX, 02H

5. Program Flow Control Instructions:

These instructions cause change in the sequence of the execution of instruction. This change can be through a condition or sometimes unconditional. The conditions are represented by flags. These instructions are also called as **Branching Instructions** or **Transfer of control Instructions**.

There are two types of Program Flow Control Instructions.

1. Unconditional Branching Instructions
2. Conditional Branching Instructions

1. Unconditional Branching Instructions: - These instructions are used to change the program execution flow control from one location to another location unconditionally means without taking any condition from conditional flags of 8086. By execution any instruction in this group program execution definitely modifies.

JMP Des: This instruction is used for unconditional jump from one place to another place. The destination address is any location in the 1MB of addressable memory of 8086. Generally this jump is long jump. The destination address is directly or indirectly represented in the operand field of the instruction.

Ex: - JMP 2000H
JMP [SI]

CALL Des: This instruction is used to call a subroutine or function or procedure into main program. The address of next instruction (both CS & IP values) after CALL is saved on to stack. For this instruction execution the SP content is decrement by 2 or 4. There are two types of CALL instructions in 8086. 1. NEAR CALL and 2. FAR CALL.

1. NEAR CALL: -This **CALL** instruction is used when the subprogram and main program both appears in same segment. For this instruction only offset address of subprogram placed in the operand field of the instruction. Ex: - CALL 2000H.

2. FAR CALL: - This **CALL** instruction is used when the subprogram and main program appears in different segments. For this instruction both base address and offset address of subprogram placed in the operand field of the instruction. Ex: - CALL 2000H: 3050H

RET: It returns the control from procedure to calling program. Every CALL instruction should have a RET. For this instruction execution the SP content is increment by 2 or 4. This instruction belongs to Implied addressing mode and its length is 1Byte. This instruction is used as last instruction for any subprogram or procedure.

Ex: - RET

IRET: It returns the control from **ISR** (**Interrupt Service Routine**) to calling program. For this instruction execution the SP content is increment by 6. This instruction belongs to Implied

addressing mode and its length is 1Byte. This instruction is used as last instruction for any ISR. The SP increment by 2 for first time to get the IP value from the stack, again SP increment by 2 for second time to get the CS value from the stack, again SP increment by 2 for third time to get the Flag Register value from the stack.
Ex: - IRET

INT n: This instruction is used to generate the software interrupt in 8086. The value of 'n' is varies from 00H to 0FFH or 0 to 255d.
Ex: - INT 02H
INT 21H

2. Conditional Branching Instructions: - These instructions are used to change the program execution flow control from one location to another location conditionally means with taking condition from conditional flags of 8086. By execution any instruction in this group program execution may or may not be modifies. It is entirely depends on the condition is satisfied or not.
The Conditional Branching Instructions are designed based on the conditions of the conditional flags except AC. All conditional jumps are relative jumps means operand address is 8-bit signed numbers.
Conditional Branching Instructions based conditions in the CY flag
CY = 0 **JNC** CY = 1 **JC**

Conditional Branching Instructions based conditions in the PF
PF = 0 **JNP** PF = 1 **JP**

Conditional Branching Instructions based conditions in the ZF
ZF = 0 **JNZ** ZF = 1 **JZ**

Conditional Branching Instructions based conditions in the SF
SF = 0 **JNS** SF = 1 **JS**

Conditional Branching Instructions based conditions in the OV flag
OV = 0 **JNO** OV = 1 **JO**
Some conditional branching instructions are designed based on conditions of 2 flags. Those 2 flags are CY and ZF.

JB / JNAE = Jump on Below / Jump on not Above or not Equal (CY = 1 & ZF = 0).
JNB / JAE = Jump on not Below / Jump on Above or Equal (CY = 0 & ZF = 1).
JBE / JNA = Jump on Below or Equal / Jump on not Above (CY = 1 & ZF = 1).
JNBE / JA = Jump on not Below or not Equal / Jump on Above (CY = 0 & ZF = 0).
These set of instructions are used when the comparison between two unsigned input data's.

JL / JNGE = Jump on Lesser / Jump on not Greater or not Equal (CY = 1 & ZF = 0).
JNL / JGE = Jump on not Lesser / Jump on Greater or Equal (CY = 0 & ZF = 1).
JLE / JNG = Jump on Lesser or Equal / Jump on not Greater (CY = 1 & ZF = 1).
JNLE / JG = Jump on not Lesser or not Equal / Jump on Greater (CY = 0 & ZF = 0).
These set of instructions are used when the comparison between two signed input data's.

JCXZ Des: If the processor executes this instruction the execution shits specified destination only when the count value in the counter register (CL or CX) is equal to zero.

6. Machine Control Instructions: -

These instructions are used to control the machine operation. These instructions are also called 'Processor Control Instructions'.
HLT: - To stop the execution of any ongoing program.
Wait: - To Enter the processor in wait state.

7. Loop Related Instructions: -

LOOP Des: -This is a looping instruction. The number of times looping is required is placed in the CX register. With each iteration the contents of CX are decremented. ZF is checked whether to loop again or not. If the ZF is not equal to '1' then the execution goes to the specified location.
Ex: - LOOP 2040H

8. Flag Manipulation Instructions

These instructions are used to set or reset Flag register bits.
Ex. STD - To set direction flag.
STI - To set the interrupt flag.
And CLI, CLD etc.

9. String related instructions
These instructions are used to perform string operation. For example –
MOVSB: - This instruction is used to transfer a byte from memory location DS:SI to memory location ES:DI.
MOVSW: - This instruction is used to transfer a byte from memory location DS:SI to memory location ES:DI.

Interrupts: -

Definition: The meaning of 'interrupts' is to break the sequence of operation. While the CPU is executing a program, on 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).After executing ISR , the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

Types of Interrupts: There are two types of Interrupts in 8086. They are:

(i) **Hardware Interrupts** (ii) **Software Interrupts**

(i) Hardware Interrupts (External Interrupts). The Intel microprocessors **support hardware interrupts** through:

- Two pins that allow interrupt requests, INTR and NMI
- One pin that acknowledges, INTA, the interrupt requested on INTR.

INTR and NMI

- **INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.
- **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.
- – **Ex: NMI, INTR.**

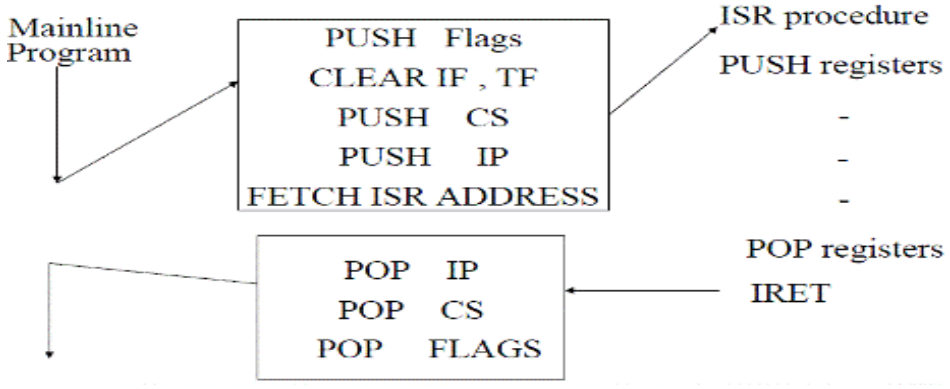
(ii) Software Interrupts (Internal Interrupts and Instructions) .S**oftware interrupts** can be caused by:

INT 3 instruction - breakpoint interrupt. This is a type 3 interrupt.
INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
INTO instruction - interrupt on overflow
Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape Opcode (type 7).
Software interrupt processing is the same as for the hardware interrupts.
Ex: INT n (Software Instructions)
Control is provided through:

IF and TF flag bits

IRET and IRETD

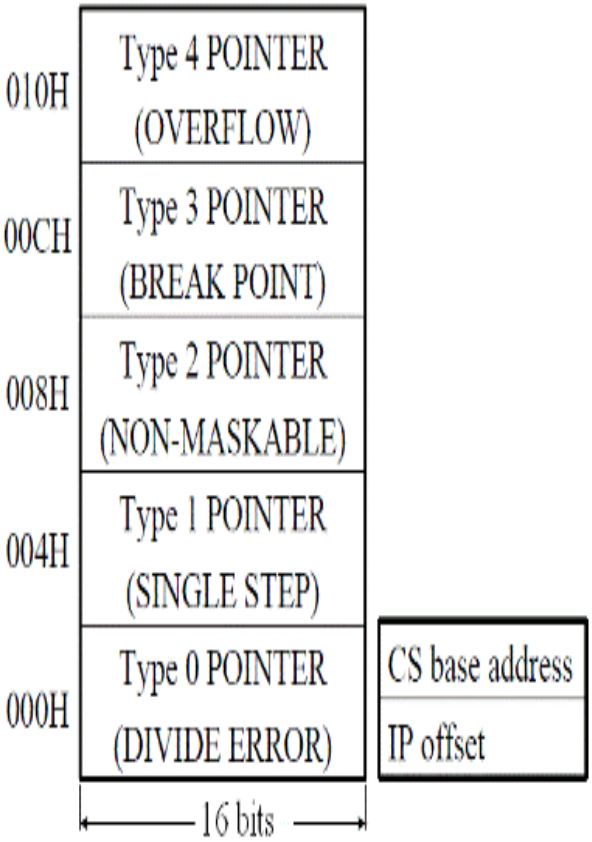
Performance of Software Interrupts: -

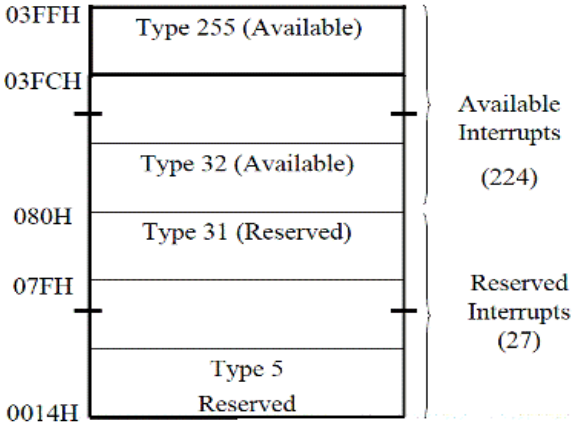


1. It decrements SP by 2 and pushes the flag register on the stack.
2. Disables INTR by clearing the IF.
3. It resets the TF in the flag Register.

5. It decrements SP by 2 and pushes CS on the stack.
6. It decrements SP by 2 and pushes IP on the stack.
6. Fetch the ISR address from the interrupt vector table.

Interrupt Vector Table: -





INT Number	Physical Address
INT 00	00000
INT 01	00004
INT 02	00008
:	:
:	:
INT FF	003FC

INT 00 (divide by zero error): -

- INT 00 is invoked by the microprocessor whenever there is an attempt to divide a number by zero.
- ISR is responsible for displaying the message “Divide Error” on the screen

INT 01 (Single-step Interrupt): -

- For single stepping the trap flag must be 1
- After execution of each instruction, 8086 automatically jumps to 00004H to fetch 4 bytes for CS: IP of the ISR.
- The job of ISR is to dump the registers on to the screen.

INT 02 (Non-Maskable Interrupt): -

- Whenever NMI pin of the 8086 is activated by a high signal (5v), the CPU Jumps to physical memory location 00008 to fetch CS: IP of the ISR associated with NMI.

INT 03 (break point Interrupt): -

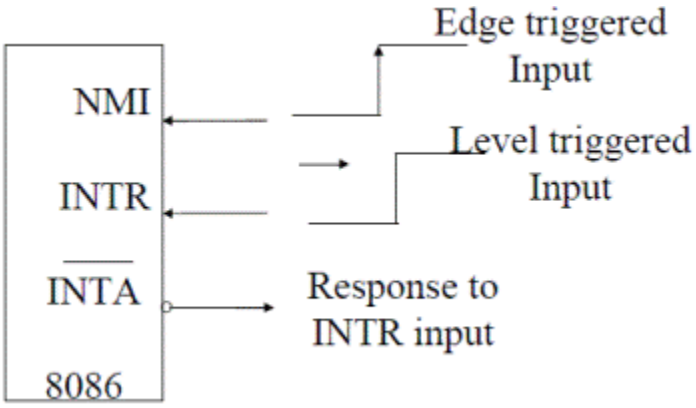
- A break point is used to examine the CPU and memory after the execution of a group of Instructions.
- It is one byte instruction whereas other instructions of the form "INT nn" are 2 byte instructions.

INT 04 (Signed number overflow Interrupt): -

- There is an instruction associated with this INT 04(interrupt on overflow).
- If INT 04 is placed after a signed number arithmetic as IMUL or ADD the CPU will activate INT 04 if 0F = 1.
- In case where 0F = 0, the INT 04 is not executed but is bypassed and acts as a NOP.

Performance of Hardware Interrupts: -

- NMI : Non Maskable interrupts - TYPE 2 Interrupt
- INTR : Interrupt request - Between 20H and FFH



Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	
INTR	
Single Step	Lowest