UNIT 3 Overview of 8086 microprocessor 5 Hrs]

3.1 Features of 8086 microprocessor
3.2 Functional diagram of 8086 microprocessor
3.3 Registers and Flags
3.4 ALP Development Tools: Editor, Assembler and linker

# Features of 8086 Microprocessor

- 8086 Microprocessor is an **enhanced version of 8085 Microprocessor** that was designed by Intel in **1976.**

- It is a **16-bit Microprocessor having 20 address lines and 16 data lines** that provides up to **1MB storage**.
- It consists of a powerful **instruction set**, which provides operations like **multiplication and division easily**.
- It supports two **modes of operation, i.e. Maximum mode and Minimum mode**.
- Maximum mode is suitable for system having **multiple processors** and
- Minimum mode is suitable for system having a **single processor**.

- **16-bit Microprocessor**: Processes 16 bits of data simultaneously.
- **Clock Speed**: **Initially 5 MHz, with later models up to 10 MHz.**
- **Memory Addressing**: 20-bit address bus, capable of addressing up to 1 MB of memory.
- **Segmentation**: Divides memory into segments of 64 KB each, improving memory management.
- **Instruction Set**: Rich set of instructions with support for arithmetic, logic, control transfer, string manipulation, etc.


- **Operating Modes**: Operates exclusively in real mode.
- **Pipelining**: Pre-fetches instructions, allowing simultaneous fetching and execution.

- **Hardware Multiplication and Division**: Supports efficient execution of complex mathematical operations.
- **Interrupts**: Supports 256 vectored interrupts for handling various events and exceptions.
- **Compatibility**: Pin-compatible with the 8088 and software-compatible with later x86 processors.

- It has an **instruction queue, which is capable of storing six instruction bytes** from the memory resulting in faster processing.

- It was the **first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus** resulting in faster processing.

- It is available in 3 versions based on the frequency of operation −
  - **8086 → 5MHz**
  - **8086-2 → 8MHz**
  - **(c)8086-1 → 10 MHz**

- It uses **two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.**
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It consists of 29,000 transistors.

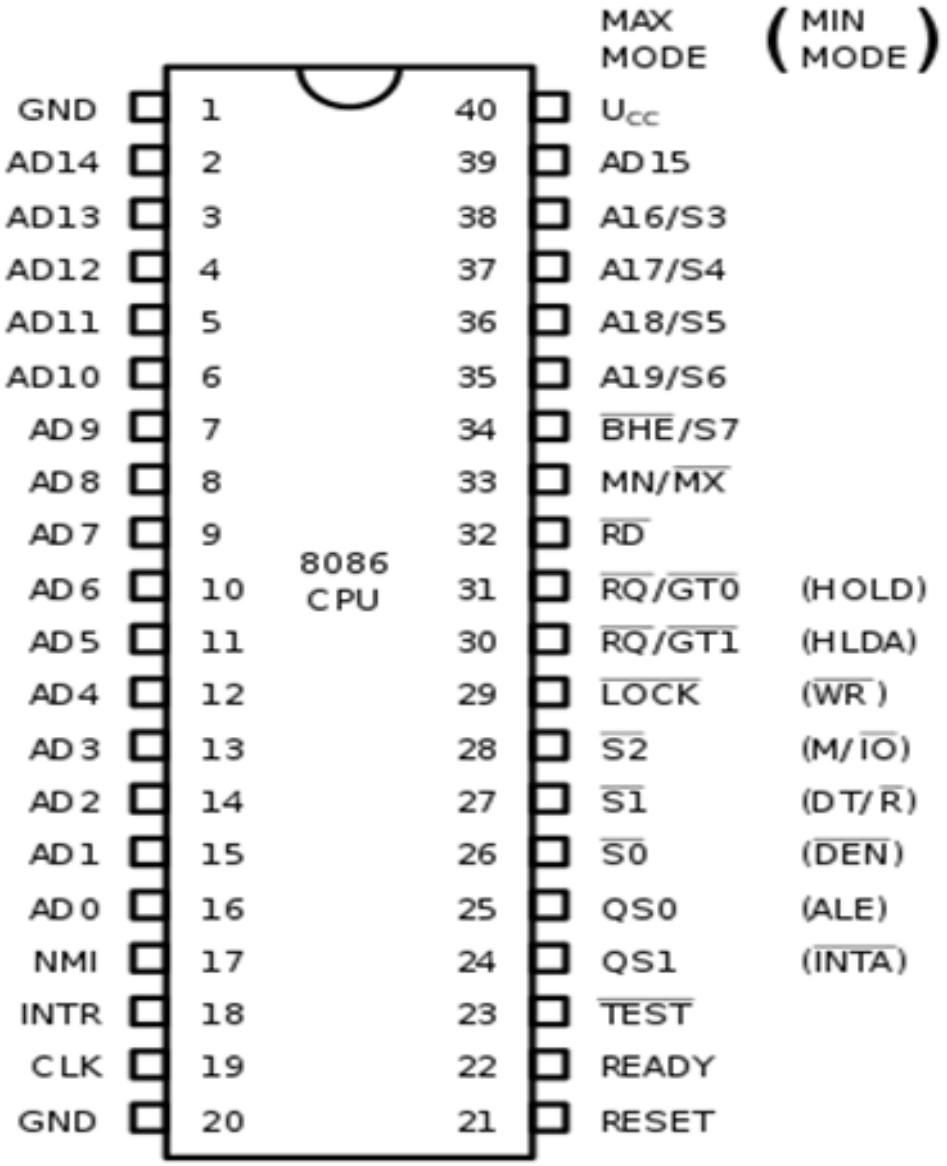
## Comparison between 8085 & 8086

| 8085 Microprocessor | 8086 Microprocessor |
|---|---|
| It is an 8-bit microprocessor. | It is a 16-bit microprocessor. |
| It has a 16-bit address line. | It has a 20-bit address line. |
| It has a 8-bit data bus. | It has a 16-bit data bus. |
| The memory capacity is 64 KB. | The memory capacity is 1 MB. |
| The Clock speed of this microprocessor is 3 MHz | The Clock speed of this microprocessor varies between 5, 8 and 10 MHz for different versions. |
| It has five flags. | It has nine flags. |

| | |
|---|---|
| 8085 microprocessors do not support memory segmentation. | 8086 microprocessor supports memory segmentation. |
| It does not support pipelining. | It supports pipelining. |
| It is accumulator-based processor. | It is general purpose register-based processor. |
| It has no minimum or maximum mode. | It has minimum and maximum modes. |
| In 8085, only one processor is used. | In 8086, more than one processor is used. An additional external processor can also be employed. |
| It contains less number of transistors compare to 8086 microprocessor. It contains about 6500 transistor. | It contains more number of transistors compare to 8085 microprocessor. It contains about 29000 in size. |
| The cost of 8085 is low. | The cost of 8086 is high. |

## 8086 pins configuration

- **AD0-AD15 (Address Data Bus):** PIN 16 15 .. 7 .. 6 5 4 3 2 39 Bidirectional address/data lines. These are low order address bus. They are multiplexed with data.
    - When these lines are used to transmit memory address, the symbol A is used instead of AD, for example, A0- A15.
- **A16 - A19 (Output):** Pin 38 37 36 35 High order address lines. These are multiplexed with status signals.
    - **A16/S3, A17/S4:** A16 and A17 are multiplexed with segment identifier signals S3 and S4.
    - **A18/S5:** A18 is multiplexed with interrupt status S5.
    - **A19/S6:** A19 is multiplexed with status signal S6.




- **BHE/S7 (Output):** Bus High Enable/Status.
  - During T1, it is low.
  - It enables the data onto the most significant half of data bus, D8-D15.
  - 8-bit device connected to upper half of the data bus use BHE signal.
  - It is multiplexed with status signal S7. S7 signal is available during T3 and T4.

    - T1 State (Address State)
    - T2 State (Data Setup State)
    - T3 State (Data Transfer State)
    - T4 State (Idle State)

- **RD (Read):** 32 For read operation.
  - It is an output signal.
  - It is active when LOW.

- **Ready (Input):** 22 The addressed memory or I/O sends acknowledgment through this pin.
    - When HIGH, it denotes that the peripheral is ready to transfer data.

- **RESET (Input):** 21 System reset. The signal is active HIGH.
- **CLK (input):** 19 Clock 5, 8 or 10 MHz.
- **INTR:** 18 Interrupt Request.
- **NMI (Input):** 17 Non-maskable interrupt request.
- **TEST (Input):** 23 Wait for test control.
    - When LOW the microprocessor continues execution otherwise waits.

- **VCC:** 40 Power supply +5V dc.
- **GND:** 20 Ground.

## Operating Modes of 8086

There are two operating modes of operation for Intel 8086, namely the

**minimum mode**

**maximum mode**.

- When only one 8086 CPU is to be used in a microprocessor system, the 8086 is used in the **Minimum mode** of operation.
- In a multiprocessor system 8086 operates in Maximum **mode**.

## Pin Description for Minimum Mode

- In this minimum mode of operation, the pin MN/MX 33 is connected to 5V D.C. supply i.e. MN/MX = VCC.

- **The description about the pins from 24 to 31 for the minimum mode is as follows:**

- **INTA (Output):** Pin number 24 interrupts acknowledgement.
    - On receiving interrupt signal, the processor issues an interrupt acknowledgment signal.
    - It is active LOW.

- **ALE (Output):** Pin no. 25. Address latch enable.
    - It goes HIGH during T1.

- The microprocessor 8086 sends this signal to latch the address into the Intel 8282/8283 latch.

- **DEN (Output):** Pin no. 26. Data Enable.
  - When Intel 8287/8286 octal bus transceiver is used this signal.
  - It is active LOW.

- **DT/R (output):** Pin No. 27 data Transmit/Receives.
  - When Intel 8287/8286 octal bus transceiver is used this signal controls the direction of data flow through the transceiver.
  - When it is HIGH, data is sent out.
  - When it is LOW, data is received.

- **M/IO (Output):** Pin no. 28, Memory or I/O access.
  - When this signal is HIGH, the CPU wants to access memory.
  - When this signal is LOW, the CPU wants to access I/O device.

- **WR (Output):** Pin no. 29, Write.
  - When this signal is LOW, the CPU performs memory or I/O write operation.

- **HLDA (Output):** Pin no. 30, Hold Acknowledgment.
  - It is sent by the processor when it receives HOLD signal.
  - It is active HIGH signal.
  - When HOLD is removed HLDA goes LOW.

- **HOLD (Input):** Pin no. 31, Hold.
  - When another device in microcomputer system wants to use the address and data bus, it sends HOLD request to CPU through this pin.
  - It is an active HIGH signal.

## Pin Description for Maximum Mode

- In the maximum mode of operation, the pin **MN/MX'** is made LOW.
- It is grounded.
- The description about the pins from 24 to 31 is as follows:

- **QS1, QS0 (Output):** Pin numbers 24, 25, Instruction Queue Status. Logics are given below:

| QS1 | QS0 | Operation |
|---|---|---|
| 0 | 0 | No operation |
| 0 | 1 | 1st byte of opcode from queue. |
| 1 | 0 | Empty the queue |
| 1 | 1 | Subsequent byte from queue |

- **S0, S1, S2 (Output):** Pin numbers 26, 27, 28 Status Signals.
  - These signals are connected to the bus controller of Intel 8288.
  - This bus controller generates memory and I/O access control signals.
  - Logics for status signal are given below:

| S2 | S1 | S0 | Operation |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt acknowledgement |
| 0 | 0 | 1 | Read data from I/O port |
| 0 | 1 | 0 | Write data from I/O port |
| 0 | 1 | 1 | Halt |
| 1 | 0 | 0 | Opcode fetch |
| 1 | 0 | 1 | Memory read |
| 1 | 1 | 0 | Memory write |
| 1 | 1 | 1 | Passive state |

- **LOCK (Output):** Pin no. 29.
  - It is an active LOW signal.
  - When this signal is LOW, all interrupts are masked and no HOLD request is granted.
  - In a multiprocessor system all other processors are informed through this signal that they should not ask the CPU for relinquishing the bus control.

- **RG/GT1, RQ/GT0 (Bidirectional)**: Pin numbers 30, 31,
  - Local Bus Priority Control. Other processors ask the CPU by these lines to release the local bus.

*In the maximum mode of operation signals WR, ALE, DEN, DT/R etc. are not available directly from the processor. These signals are available from the controller 8288.*

# Functional Diagram of 8086 Microprocessor

- **Bus Interface Unit (BIU)**:
  - **Segment Registers**: CS, DS, SS, ES
  - **Instruction Queue**: Pre-fetches instructions to improve performance.
  - **Instruction Pointer (IP)**: Holds the address of the next instruction to be executed.
  - **Address Generation**: Computes physical addresses from segment pairs.

- **Execution Unit (EU)**:
  - **Arithmetic Logic Unit (ALU)**: Performs arithmetic and logical operations.
  - **General Purpose Registers**: AX, BX, CX, DX
  - **Pointer and Index Registers**: SP, BP, SI, DI
  - **Flags Register**: Status and control flags.
  - **Control Unit**: Decodes and executes instructions.

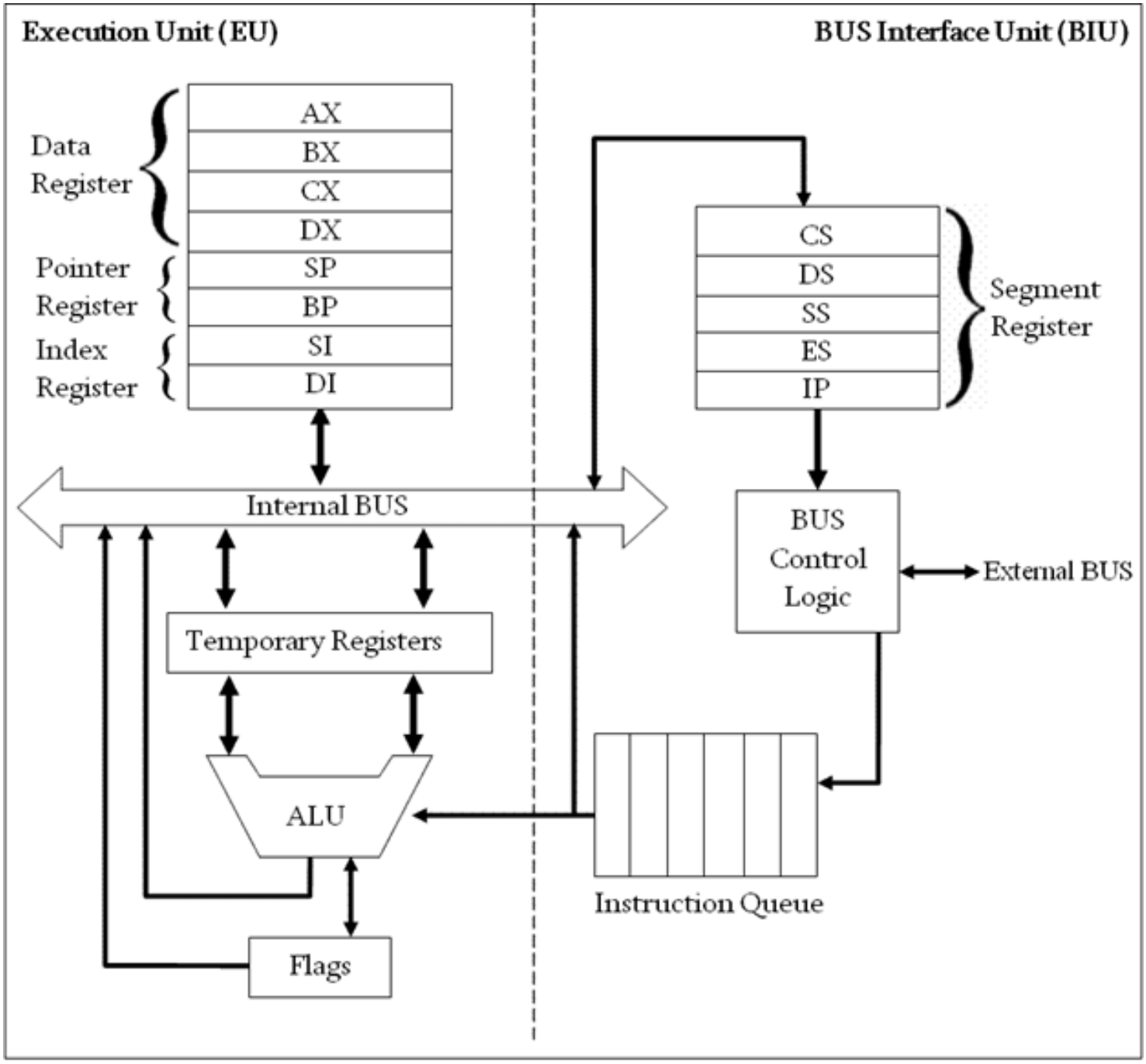



***Fig: Block Diagram of Intel 8086 Microprocessor (8086 Architecture)***

8086 contains two independent functional units: **a Bus Interface Unit (BIU)** and an **Execution Unit (EU)**.

## Bus Interface Unit (BIU)

The segment registers, instruction pointer and 6-byte instruction queue are associated with the bus interface unit (BIU).

The BIU:

- Handles transfer of data and addresses,
- Fetches instruction codes, stores fetched instruction codes in first-in-first-out register set called a **queue**,
- Reads data from memory and I/O devices,
- Writes data to memory and I/O devices,
- It relocates addresses of operands since it gets un-relocated operand addresses from EU. The EU tells the BIU from where to fetch instructions or where to read data.
- It has the following functional parts:
    - **Instruction Queue:** When EU executes instructions, the BIU gets 6-bytes of the next instruction and stores them in the instruction queue and this process is known as instruction pre fetch. This process increases the speed of the processor.
        - **Segment Registers:** A segment register contains the addresses of instructions and data in memory which are used by the processor to access memory locations. It points to the starting address of a memory segment currently being used.

          There are 4 segment registers in 8086 as given below:
            - **Code Segment Register (CS):** Code segment of the memory holds instruction codes of a program.
            - **Data Segment Register (DS):** The data, variables and constants given in the program are held in the data segment of the memory.
            - **Stack Segment Register (SS):** Stack segment holds addresses and data of subroutines. It also holds the contents of registers and memory locations given in PUSH instruction.

- **Extra Segment Register (ES):** Extra segment holds the destination addresses of some data of certain string instructions.
- **Instruction Pointer (IP):** The instruction pointer in the 8086 microprocessor acts as a program counter. It indicates the address of the next instruction to be executed.

**Execution Unit (EU)**

- The **EU** receives opcode of an instruction from the queue, decodes it and then executes it. While Execution, unit decodes or executes an instruction, then the BIU fetches instruction codes from the memory and stores them in the queue.
- The BIU and EU operate in parallel independently. This makes processing faster.
- General purpose registers, stack pointer, base pointer and index registers, ALU, flag registers (FLAGS), instruction decoder and timing and control unit constitute execution unit (EU).

  Let's discuss them:
- **General Purpose Registers:** There are four 16-bit general purpose registers: AX (Accumulator Register), BX (Base Register), CX (Counter) and DX. Each of these 16-bit registers are further subdivided into 8-bit registers as shown below:

| 16-bit registers | 8-bit high-order registers | 8-bit low-order registers |
|---|---|---|
| AX | AH | AL |
| BX | BH | BL |
| CX | CH | CL |
| DX | DH | DL |

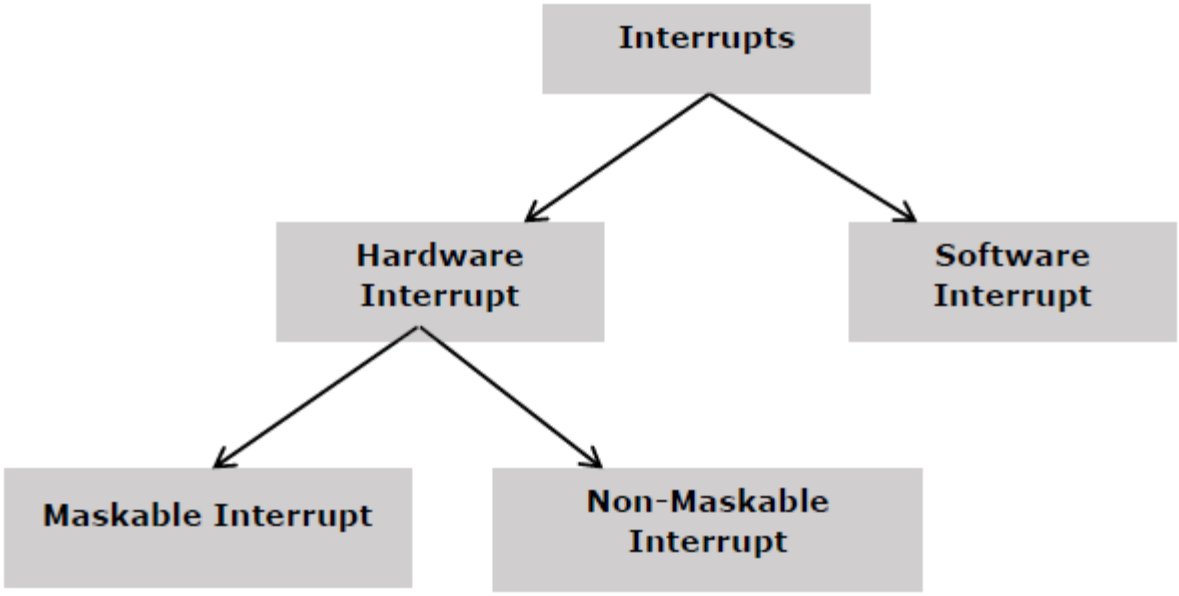
- **Index Register:** The following four registers are in the group of pointer and index registers:
  - Stack Pointer (SP)
  - Base Pointer (BP)
  - Source Index (SI)
  - Destination Index (DI)

- **ALU:** It handles all arithmetic and logical operations. Such as addition, subtraction, multiplication, division, AND, OR, NOT operations.
- **Flag Register:** It is a 16 bit register which exactly behaves like a flip-flop, which means it changes states according to the result stored in the accumulator. It has 9 flags, and they are divided into 2 groups i.e. conditional and control flags.
    - **Conditional Flags:** This flag represents the result of the last arithmetic or logical instruction executed. Conditional flags are:
        - Carry Flag
        - Auxiliary Flag
        - Parity Flag
        - Zero Flag
        - Sign Flag
        - Overflow Flag
    - **Control Flags:** It controls the operations of the execution unit. Control flags are:
        - Trap Flag
        - Interrupt Flag
        - Direction Flag

# Interrupts

- **Interrupt** is a process of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor.Microprocessor responds to these interrupts with an **interrupt service routine (ISR)**, which is a short program or subroutine to instruct the microprocessor on how to handle the interrupt.

**There are different types of interrupt in 8086:**




## Hardware Interrupts

**Hardware interrupts** are that type of interrupt which are caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

**The Intel 8086 has two hardware interrupt pins:**

- NMI (Non-Maskbale Interrupt)
- INTR (Interrupt Request) Maskable Interrupt.

**NMI:** NMI is a single Non-Maskable Interrupt having higher priority than the maskable interrupt.

- It cannot be disabled (masked) by user using software.
- It is used by the processor to handle emergency conditions. **For example:** It can be used to save programs and data in case of power failure. External electronic circuitry is used to detect power failure, and to send an interrupt signal to 8086 through NMI line.

**INTR:** The INTR is a maskable interrupt. It can be enabled/disabled using interrupt flag (IF). After receiving INTR from external device, the 8086 acknowledges through INTA signal.

It executes two consecutive interrupt acknowledge bus cycles.

## Software Interrupt

- A microprocessor can also be interrupted by internal abnormal conditions such as overflow; division by zero; etc.
- A programmer can also interrupt microprocessor by inserting INT instruction at the desired point in the program while debugging a program. Such an interrupt is called a software interrupt.

The interrupt caused by an internal abnormal condition also came under the heading of software interrupt.

**Example of software interrupts are:**

- TYPE 0 (division by zero)
- TYPE 1 (single step execution for debugging a program)
- TYPE 2 represents NMI (power failure condition)
- TYPE 3 (break point interrupt)
- TYPE 4 (overflow interrupt)

**Interrupt pointer table for 8086**

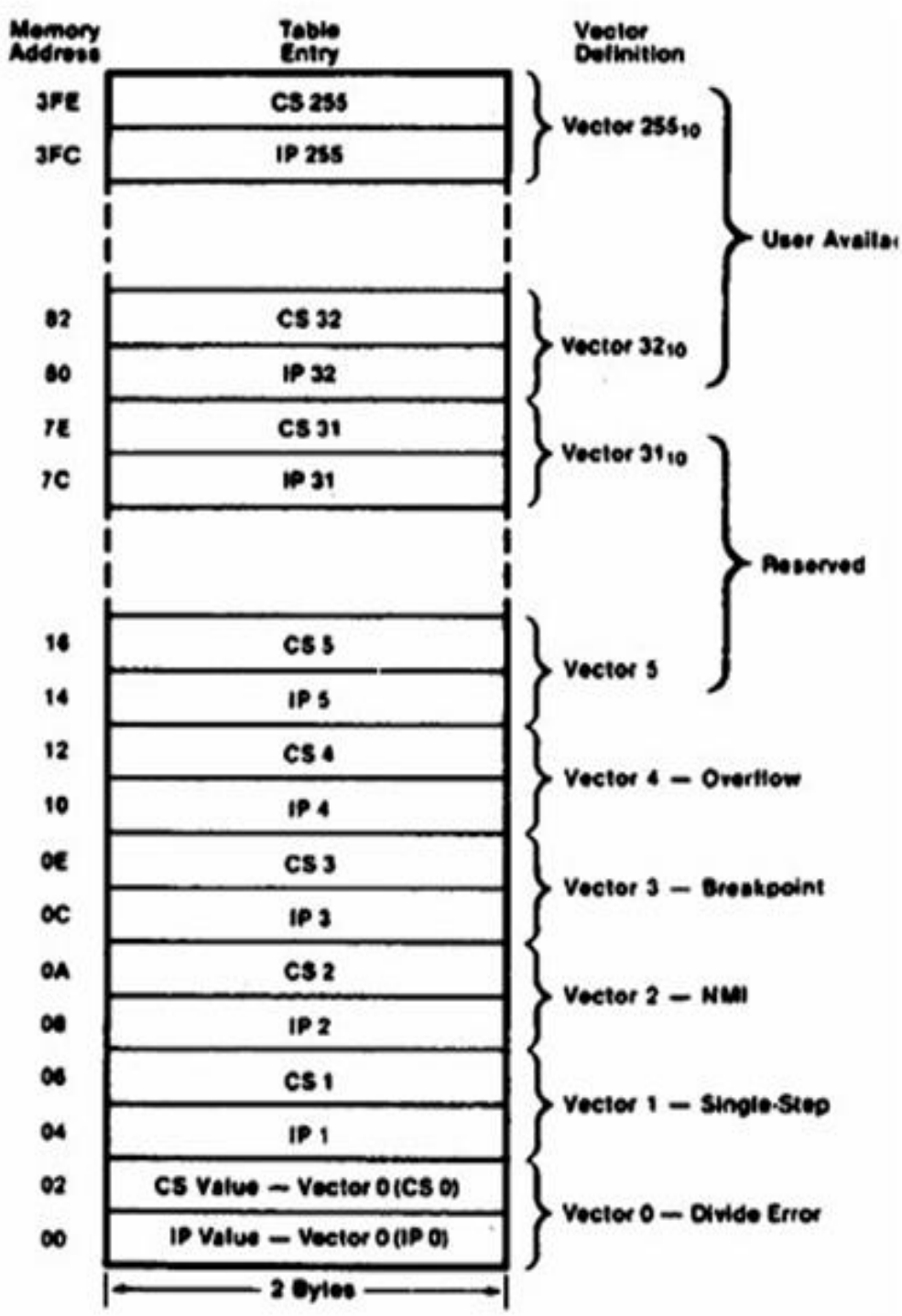



***Fig: Interrupt pointer table for 8086***

- The 8086 can handle up to 256, hardware and software interrupts.
- 1KB memory acts as a table to contain interrupt vectors (or interrupt pointers), and it is called **interrupt vector table** or interrupt pointer table.
- The 256 interrupt pointers have been numbered from 0 to 255 (FF hex). The number assigned to an interrupt pointer is known as type of that interrupt. For example, Type 0, Type 1, Type 2,.....Type 255 interrupt.

## Addressing modes of 8086

The way for which an operand is specified for an instruction in the accumulator, in a general-purpose register or in memory location, is called **addressing mode**.

The 8086 microprocessors have **8 addressing modes.** Two addressing modes have been provided for instructions which operate on register or immediate data.

**These two addressing modes are:**

**Register Addressing:** In register addressing, the operand is placed in one of the 16-bit or 8-bit general purpose registers.

**Example**

- MOV AX, CX
- ADD AL, BL
- ADD CX, DX

**Immediate Addressing:** In immediate addressing, the operand is specified in the instruction itself.

**Example**

- MOV AL, 35H
- MOV BX, 0301H
- MOV [0401], 3598H
- ADD AX, 4836H

**The remaining 6 addressing modes specify the location of an operand which is placed in a memory.**

**These 6 addressing modes are:**

**Direct Addressing:** In direct addressing mode, the operands offset is given in the instruction as an 8-bit or 16-bit displacement element.

**Example**

- ADD AL, [0301]

The instruction adds the content of the offset address 0301 to AL. the operand is placed at the given offset (0301) within the data segment DS.

**Register Indirect Addressing:** The operand's offset is placed in any one of the registers BX, BP, SI or DI as specified in the instruction.

**Example**

- MOV AX, [BX]

It moves the contents of memory locations addressed by the register BX to the register AX.

**Based Addressing:** The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as base register for data segment, and the BP is used as a base register for stack segment.

**Effective address (Offset)** = [BX + 8-bit or 16-bit displacement].

**Example**

- MOV AL, [BX+05]; an example of 8-bit displacement.
- MOV AL, [BX + 1346H]; example of 16-bit displacement.

**Indexed Addressing:** The offset of an operand is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Offset (Effective Address) = [SI or DI + 8-bit or 16-bit displacement]

**Example**

- MOV AX, [SI + 05]; 8-bit displacement.
- MOV AX, [SI + 1528H]; 16-bit displacement.

**Based Indexed Addressing:** The offset of operand is the sum of the content of a base register BX or BP and an index register SI or DI.

Effective Address (Offset) = [BX or BP] + [SI or DI]

Here, BX is used for a base register for data segment, and BP is used as a base register for stack segment.

**Example**

- ADD AX, [BX + SI]
- MOV CX, [BX + SI]

**Based Indexed with Displacement:** In this mode of addressing, the operand's offset is given by:

**Effective Address (Offset)** = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

**Example**

- MOV AX, [BX + SI + 05]; 8-bit displacement
- MOV AX, [BX + SI + 1235H]; 16-bit displacement

# ALP Development Tools: Editor, Assembler and linker

## Instruction Set of 8086

Instructions are classified on the basis of functions they perform. They are categorized into the following main types:

### Data Transfer instruction

- All the instructions which perform data movement come under this category.
- The source data may be a register, memory location, port etc. the destination may be a register, memory location or port.
- The following instructions come under this category:

| Instruction | Description |
|---|---|
| MOV | Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc. |
| LDS | Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register. |
| LES | Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register. |
| LEA | Loads offset address into the specified register. |

| | |
|---|---|
| LAHF | Loads low order 8-bits of the flag register into AH register. |
| SAHF | Stores the content of AH register into low order bits of the flags register. |
| XLAT/XLATB | Reads a byte from the lookup table. |
| XCHG | Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations. |
| PUSH | Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack. |
| POP | Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s). |
| POPF | Pops (reads) two bytes from the top of the stack and keeps them in the flag register. |
| IN | Transfers data from a port to the accumulator or AX, DX or AL register. |
| OUT | Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction. |

## Arithmetic Instructions

- Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

**The following instructions come under this category:**

| Instruction | Description |
|---|---|
| ADD | Adds data to the accumulator i.e. AL or AX register or memory locations. |
| ADC | Adds specified operands and the carry status (i.e. carry of the previous stage). |
| SUB | Subtract immediate data from accumulator, memory or register. |
| SBB | Subtract immediate data with borrow from accumulator, memory or register. |
| MUL | Unsigned 8-bit or 16-bit multiplication. |

| | |
|---|---|
| IMUL | Signed 8-bit or 16-bit multiplication. |
| DIV | Unsigned 8-bit or 16-bit division. |
| IDIV | Signed 8-bit or 16-bit division. |
| INC | Increment Register or memory by 1. |
| DEC | Decrement register or memory by 1. |
| DAA | **Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD. |
| DAS | **Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD. |
| AAA | **ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD. |
| AAD | **Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte. |
| AAM | **Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD. |
| AAS | **ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number. |
| CBW | Convert signed Byte to signed Word. |
| CWD | Convert signed Word to signed Doubleword. |
| NEG | Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s). |
| CMP | Compare Immediate data, register or memory with accumulator, register or memory location(s). |

## Logical Instructions

- Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations.
- **The following instructions come under this category:**

| Instruction | Description |
|---|---|
| AND | Performs bit by bit logical AND operation of two operands and places the result in the specified destination. |
| OR | Performs bit by bit logical OR operation of two operands and places the result in the specified destination. |
| XOR | Performs bit by bit logical XOR operation of two operands and places the result in the specified destination. |
| NOT | Takes one's complement of the content of a specified register or memory location(s). |
| TEST | Perform logical AND operation of a specified operand with another specified operand. |

## Rotate Instructions

**The following instructions come under this category:**

| Instruction | Description |
|---|---|
| RCL | Rotate all bits of the operand left by specified number of bits through carry flag. |
| RCR | Rotate all bits of the operand right by specified number of bits through carry flag. |
| ROL | Rotate all bits of the operand left by specified number of bits. |
| ROR | Rotate all bits of the operand right by specified number of bits. |

## Shift Instructions

**The following instructions come under this category:**

| Instruction | Description |
|---|---|
| SAL or SHL | Shifts each bit of operand left by specified number of bits and put zero in LSB position. |
| SAR | Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB. |
| SHR | Shift each bit of operand right by specified number of bits and put zero in MSB position. |

## Branch Instructions

- It is also called program execution transfer instruction.
- Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target.
- The following instructions come under this category:

| Instruction | Description |
|---|---|
| JA or JNBE | Jump if above, not below, or equal i.e. when CF and ZF = 0 |
| JAE/JNB/JNC | Jump if above, not below, equal or no carry i.e. when CF = 0 |
| JB/JNAE/JC | Jump if below, not above, equal or carry i.e. when CF = 0 |
| JBE/JNA | Jump if below, not above, or equal i.e. when CF and ZF = 1 |
| JCXZ | Jump if CX register = 0 |
| JE/JZ | Jump if zero or equal i.e. when ZF = 1 |
| JG/JNLE | Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF |
| JGE/JNL | Jump if greater, not less or equal i.e. when SF = OF |

| | |
|---|---|
| JL/JNGE | Jump if less, not greater than or equal i.e. when SF ≠ OF |
| JLE/JNG | Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF |
| JMP | Causes the program execution to jump unconditionally to the memory address or label given in the instruction. |
| CALL | Calls a procedure whose address is given in the instruction and saves their return address to the stack. |
| RET | Returns program execution from a procedure (subroutine) to the next instruction or main program. |
| IRET | Returns program execution from an interrupt service procedure (subroutine) to the main program. |
| INT | Used to generate software interrupt at the desired point in a program. |
| INTO | Software interrupts to indicate overflow after arithmetic operation. |
| LOOP | Jump to defined label until CX = 0. |
| LOOPZ/LOOPE | Decrement CX register and jump if CX ≠ 0 and ZF = 1. |
| LOOPNZ/LOOPNE | Decrement CX register and jump if CX ≠ 0 and ZF = 0. |

Here, CF = Carry Flag
ZF = Zero Flag
OF = Overflow Flag
SF = Sign Flag
CX = Register

## Flag Manipulation and Processor Control Instructions

- Instructions of this instruction set are related to flag manipulation and machine control.
- The following instructions come under this category:

| Instruction | Description |
|---|---|

| | |
|---|---|
| CLC | **Clear Carry Flag:** This instruction resets the carry flag CF to 0. |
| CLD | **Clear Direction Flag:** This instruction resets the direction flag DF to 0. |
| CLI | **Clear Interrupt Flag:** This instruction resets the interrupt flag IF to 0. |
| CMC | This instruction take complement of carry flag CF. |
| STC | Set carry flag CF to 1. |
| STD | Set direction flag to 1. |
| STI | Set interrupt flag IF to 1. |
| HLT | Halt processing. It stops program execution. |
| NOP | Performs no operation. |
| ESC | **Escape:** makes bus free for external master like a coprocessor or peripheral device. |
| WAIT | When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing. |
| LOCK | It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction. |

## String Instructions

- String is series of bytes or series of words stored in sequential memory locations.
- The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

**The following instructions come under this category:**

| Instruction | Description |
|---|---|
| MOVS/MOVSB/MOVSW | Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register. |
| CMPS/CMPSB/CMPSW | Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register. |

| | |
|---|---|
| SCAS/SCASB/SCASW | Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES. |
| LODS/LODSB/LODSW | Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register. |
| STOS/STOSB/STOSW | Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register. |
| REP | Repeats the given instruction until $CX \neq 0$ |
| REPE/ REPZ | Repeats the given instruction till $CX \neq 0$ and $ZF = 1$ |
| REPNE/REPNZ | Repeats the given instruction till $CX \neq 0$ and $ZF = 0$ |