UNIVERSITY OF BAGHDAD

College of Education for Pure Science

(Ibn Al-Haitham)

Department of Computer Science

Microprocessor 8086

Second Stage

Prof. Dr. Alaa Abdul Hameed

Lecturer Sukaina Shukri

Introduction to Computer

A computer is an electronic device, operating under the control of instructions stored in its own memory that can accept data (input), process the data according to specified rules, produce information (output), and store the information for future use.

Computer Components

Any kind of computers consists of **Software and Hardware:**

- **Software**

Software is a generic term for organized collections of computer data and instructions, often broken into two major categories: system software that provides the basic non-task-specific functions of the computer, and application software which is used by users to accomplish specific tasks.

- **Hardware:**

Computer hardware is the collection of physical elements that constitutes a computer system. Computer hardware refers to the physical parts or components of a computer such as the monitor, mouse, keyboard, etc. all of which are physical objects that can be touched.

Basic Components of a computer

All computers consist of (at least):

1. Central Processing Unit (CPU)
2. Read Only Memory (ROM)
3. Random Access Memory (RAM)
4. Input / Output ports
5. Bus System

CPU

A CPU is brain of a computer. It is responsible for all functions and processes. Regarding computing power, the CPU is the most important element of a computer system.

The MPU is comprised of three main parts:

- **Arithmetic Logic Unit (ALU)**: Executes all arithmetic and logical operations. Arithmetic calculations like as addition, subtraction, multiplication and division. Logical operation like compare numbers, letters, or special characters
- **Control Unit (CU):** controls and co-ordinates computer components.

  1. Read the code for the next instruction to be executed.
  2. Increment the program counter so it points to the next instruction.
  3. Read whatever data the instruction requires from cells in memory.
  4. Provide the necessary data to an ALU or register.
  5. If the instruction requires an ALU or specialized hardware to complete, instruct the hardware to perform the requested operation.

- **Registers:** they are a temporary storage memory that is built into CPU. Registers are performed their tasks quickly. All computers required these registers to manipulate data, and store memory addressing.

  The CU deals with several special-purpose registers and their functions are briefly described below:

  • **Program counter (PC)**: The PC holds the address of the next instruction to be executed.

  • **Instruction register (IR)**: The IR holds the actual instruction being executed currently by the computer.

• **Memory address register (MAR)**: The MAR holds the address of a memory location.

• **Memory data register (MDR)**: The MDR holds a data value that is being stored to or retrieved from the memory location currently addressed by the memory address register.

• **Status register (SR)**: The SR indicates the results of an arithmetic and logic unit operation. For example: carry, overflow, negative.

The Fetch-Execute Cycle

Depending on the complexity of each operation (i.e., task), the computer may take two or more **machine cycles** in order to complete the task. A machine cycle consists of both fetch and execution cycles. In the fetch cycle, the CU brings the program instruction from the memory, decodes it (i.e., translates the instruction into commands), and then sends the data to the ALU for execution. In the execution cycle, the ALU performs an operation and then sends the result to the memory for temporary storage.

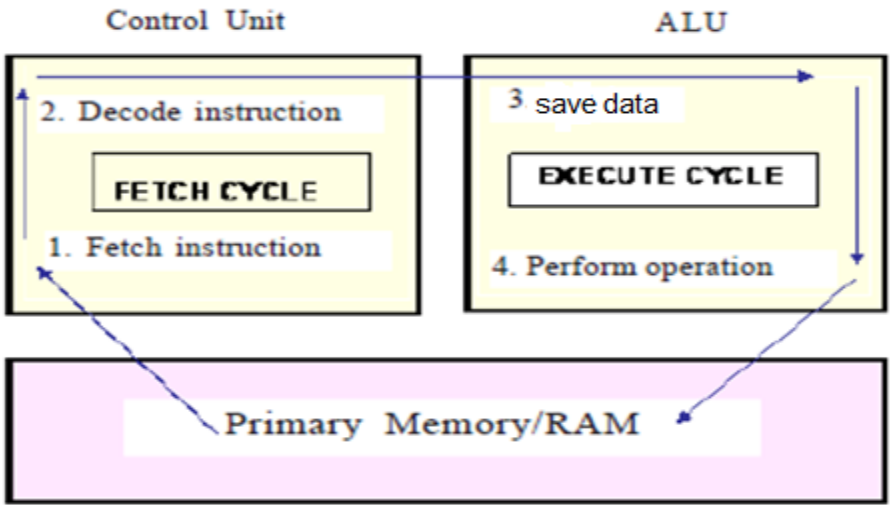


Fig. 1 Fetch and execution cycle

Evolution of Microprocessors

We can categorize the microprocessor according to the generations or according to the size of the microprocessor:

- **First Generation (4 - bit Microprocessors)**

The first generation microprocessors were introduced in the year 1971-1972 by Intel Corporation. It was named **Intel 4004** since it was a 4-bit processor. It was a processor on a single chip. It could perform simple arithmetic and logical operations such as addition, subtraction, Boolean OR and Boolean AND.

- **Second Generation (8 - bit Microprocessor)**

The second generation microprocessors were introduced in 1973 again by Intel. It was a first 8 - bit microprocessor which could perform arithmetic and logic operations on 8-bit words. It was Intel 8008, and another improved version was Intel 8088.

- **Third Generation (16 - bit Microprocessor)**

The third generation microprocessors, introduced in 1978 were represented by **Intel's 8086, Zilog Z800 and 80286**, which were 16 - bit processors with a performance like minicomputers.

- **Fourth Generation (32 - bit Microprocessors)**

Several different companies introduced the 32-bit microprocessors, but the most popular one is the **Intel 80386**.

- **Fifth Generation (64 - bit Microprocessors)**

From 1995 to now we are in the fifth generation. After 80856, Intel came out with a new processor namely Pentium processor followed by **Pentium Pro CPU**, which allows multiple CPUs in a single system to achieve multiprocessing.

Other improved 64-bit processors are **Celeron, Dual, Quad, Octa Core processors**.

RAM:

It is a memory scheme within the computer system responsible for storing data on a temporary basis, so that it can be promptly accessed by the processor as and when needed. It is volatile in nature, which means that data will be erased once supply to the storage device is turned off. RAM stores data randomly and the processor accesses these data randomly from the RAM storage. RAM is considered "random access" because you can access any memory cell directly if you know the row and column that intersect at that cell.

Memory Write Operation

Three basic steps are needed in order the CPU to perform a write operation into a specified memory location:

1. The word to be stored into the memory location is first loaded by the CPU into a specified register, called the memory data register (MDR).
2. The address of the location into which the word is to be stored is loaded by the CPU into a specified register, called the memory address register (MAR).
3. A signal, called write, is issued by the CPU indicating that the word stored in the MDR is to be stored in the memory location whose address in loaded in the MAR.

Memory Read Operation

Three basic steps are needed in order to perform a memory read operation:

1. The address of the location from which the word is to be read is loaded into the MAR.
2. A signal, called read, is issued by the CPU indicating that the word whose address is in the MAR is to be read into the MDR.
3. After some time, corresponding to the memory delay in reading the specified word, the required word will be loaded by the memory into the MDR ready for use by the CPU.

ROM:

It is a permanent form of storage. ROM stays active regardless of whether power supply to it is turned on or off. ROM devices do not allow data stored on them to be modified.

Input Devices:

Input device is any peripheral (piece of computer hardware equipment to provide data and control signals to an information processing system such as a computer).
Input device Translate data from **form** that humans understand to one that the computer can work with. Most common are keyboard and mouse.

Output devices

An output device is any piece of computer hardware equipment used to communicate the results of data processing carried out by an information processing system (such as a computer) which converts the electronically generated information into human-readable form.

I/O devices are connected to the system bus through **I/O controller** (interface) – which acts as interface between the system bus and I/O devices.

There are two main reasons for using I/O controllers

1- I/O devices exhibit different characteristics and if these devices are connected directly, the CPU would have to understand and respond appropriately to each I/O device. This would cause the CPU to spend a lot of time interacting with I/O devices and spend less time executing user programs.

2- The amount of electrical power used to send signals on the system bus is very low. This means that the cable connecting the I/O device has to be very short (a few centimeters at most). I/O controllers typically contain driver hardware to send current over long cable that connects I/O devices**.**

Bus System

- A Bus is a common communications pathway used to carry information between the various elements of a computer system
- The term BUS refers to a group of wires or conduction tracks on a printed circuit board (PCB) though which binary information is transferred from one part of the microcomputer to another
- The individual subsystems of the digital computer are connected through an interconnecting BUS system.

There are three main bus groups

- Address Bus
- Data Bus
- Control Bus

Address Bus: The address bus consists of 16, 20, 24, or more parallel signal lines. On these lines the CPU sends out the address of the memory location that is to be written to or read from. The number of address lines determines the number of memory locations that the CPU can address. If the CPU has N address lines then it can directly address 2^N memory locations. The 8086 microprocessor has 20 bit address bus, therefore it can address a maximum of 1M byte of memory location with an address ranged from (00000)H-(FFFFF)H

Data Bus: The data bus consists of 8, 16, 32 or more parallel signal lines. As indicated by the double-ended arrows on the data bus line, the data bus lines are bi-directional. This means that the CPU can read data in on these lines from memory or from a port as well as send data out on these lines to memory location or to a port. Many devices in a system will have their outputs connected to the data bus, but the outputs of only one device at a time will be enabled.

Control Bus: The control bus consists of 4-10 parallel signal lines. The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices. Typical control bus signals are memory read, memory write, I/O read, and I/O writer.

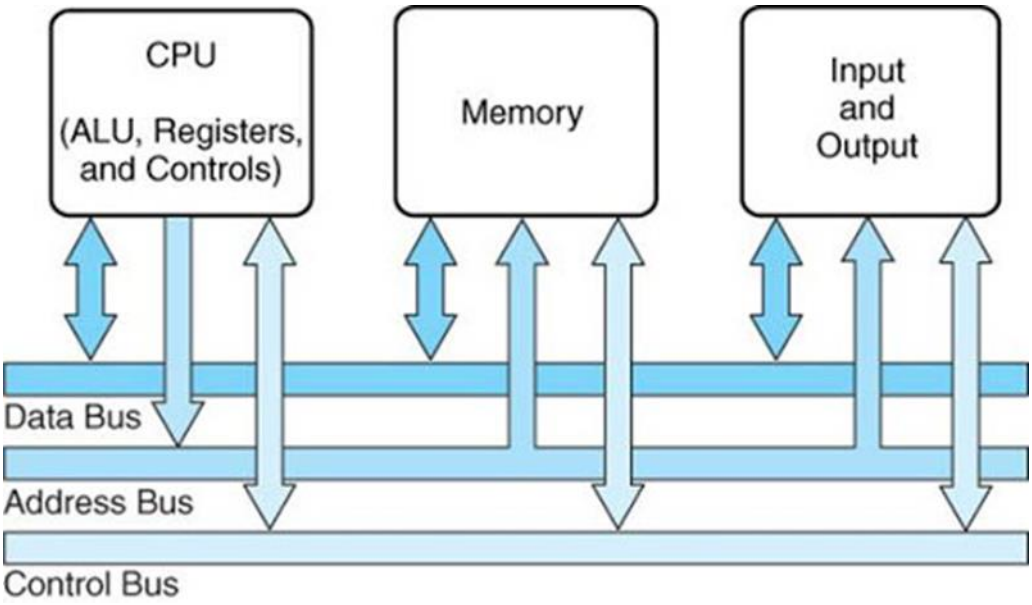


Fig. (2) System bus in computer

Microprocessor 8086

Microprocessor 8086 is the first 16-bit microprocessor from INTEL, released in the year 1978. The term 16 bit means that its ALU, its internal registers and most of the instructions are designed to work with 16 bit binary words. 8086 microprocessor has a 16-bit data bus and 20-bit address bus. So, it can address any one of 2^{20} =1048576=1 megabyte memory locations. INTEL 8088 has the same ALU, same registers and same instruction set as the 8086.But the only difference is 8088 has only 8-bit data bus and 20-bit address bus. Hence the 8088 can only read/write/ports of only 8-bit data at a time. The 8086 microprocessor can work in two modes of operations .They are Minimum mode and Maximum mode. In the minimum mode of operation the microprocessor do not associate with any co-processors and cannot be used for multiprocessor systems. But in the maximum mode the 8086 can work in multi-processor or co-processor configuration. This minimum or maximum

operations are decided by the pin MN/ MX. When this pin is high 8086 operates in minimum mode otherwise it operates in Maximum mode.

8086 Microprocessor features:

1. It is 16-bit microprocessor.
2. It has a 16-bit data bus, so it can read data from or write data to memory and ports either 16-bit or 8-bit at a time.
3. It has 20 bit address bus and can access up to 2^{20} memory locations (1 MB).
4. It can support up to 64K I/O ports.
5. It provides 14, 16-bit registers.
6. It has multiplexed address and data bus AD0-AD15 & A16-A19.
7. Pre-fetches up to 6 instruction bytes from memory and queues them in order to speed up the processing.
8. 8086 supports 2 modes of operation
    a. Minimum mode
    b. Maximum mode

Architecture of 8086 microprocessor:

To improve the performance by implementing the parallel processing concept the CPU of the 8086 /8088 is divided into two independent sections .They are Bus Interface Unit (BIU) and Execution Unit (EU) as shown in figure 2.

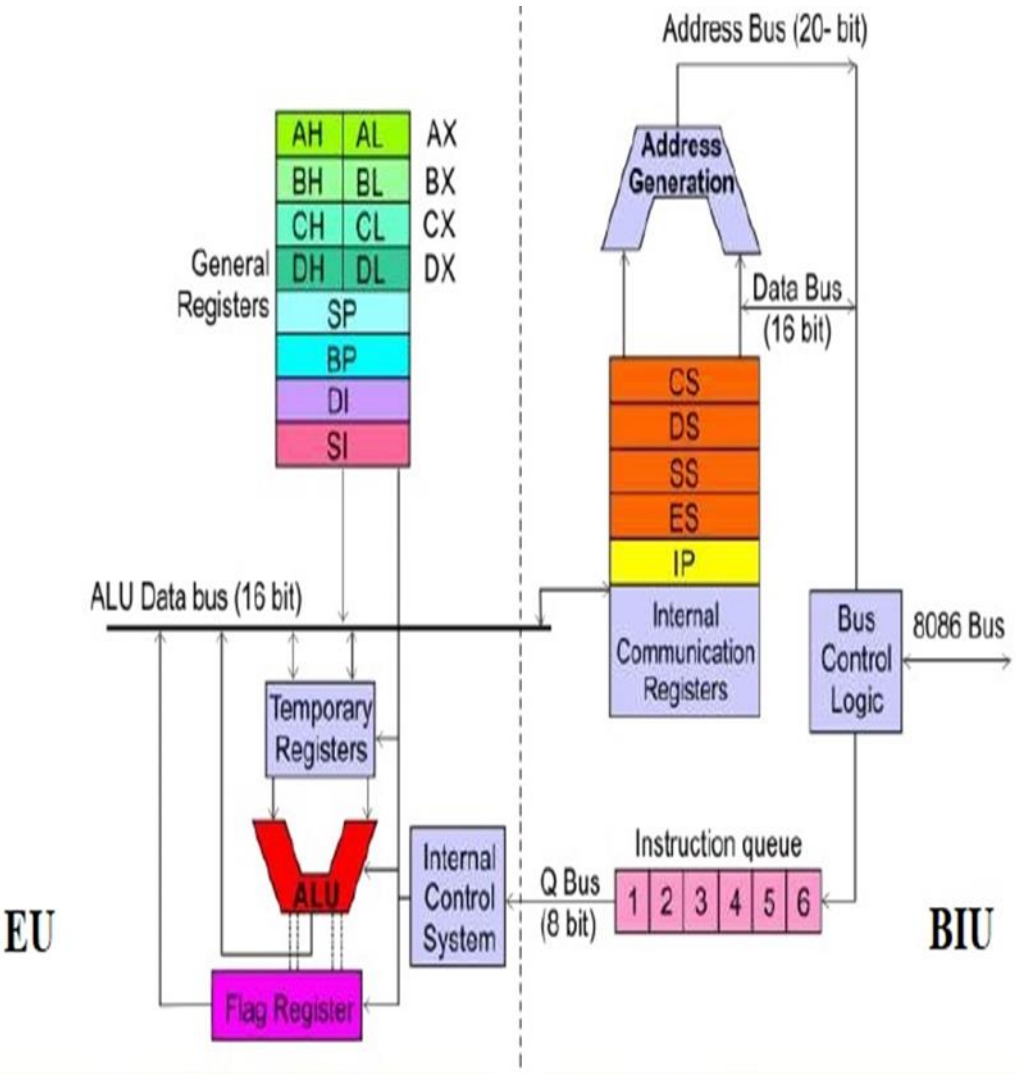


Fig. 3: Architecture of 8086 Microprocessor

Bus Interface Unit (BIU):

- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit connects the microprocessor to external devices.

BIU performs following operations:

- Instruction fetching.
- Reading and writing data of data operands for memory.
- Inputting/outputting data for input/output peripherals.

- And other functions related to instruction and data acquisition.
- To implement above functions, the BIU contains the segment registers, the instruction pointer, address generation adder, bus control logic, and an instruction queue.
- The BIU uses a mechanism known as an instruction stream queue to implement pipeline architecture.

Execution Unit (EU)

- The Execution unit is responsible for decoding and executing all instructions.
- The EU consists of arithmetic logic unit (ALU), status and control flags, general-purpose registers, and temporary-operand registers.
- The EU extracts instructions from the top of the queue in the BIU, decodes them ,generates operands if necessary, passes them to the BIU and requests it to perform the read or write by cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

Pipelining Architecture in 8086mp:

- While the EU is decoding an instruction or executing an instruction, which does not require use of the buses, the BIU fetches up to six instruction bytes for the following instructions.
- The BIU stores these pre-fetched bytes in a first-in-first-out register set called a **queue.**

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

Register Organization

The 14 registers of 8086 microprocessor are categorized into four groups. They are general purpose data registers, Pointer & Index registers, Segment registers, Instruction register, and Flag register as shown in the table below.

S.NO	Type	Register width	Name of the Registers
1	Data Register	16-bit	AX,BX,CX,DX
		8-bit	AL,AH,BL,BH,CL,CH,DL,DH
2	Pointer Registers	16-bit	Stack Pointer (SP) Base Pointer (BP)
3	Index Registers	16-bit	Source Index (SI) Destination Index (DI)
4	Segment Registers	16-bit	Code Segment (CS) Data Segment (DS) Stack Segment (SS) Extra Segment (ES)
5	Instruction	16-bit	Instruction Pointer (IP)
6	Flag	16-bit	Flag Register

1. General Purpose Registers:

8086 CPU has 8 general purpose registers; these registers can be divided into:

a) Data registers: four 16 bits data registers

b) Pointer and index registers: two 16 bits pointer registers and two 16 bits index registers

a) Data Registers: they are four registers (AX, BX, CX, and DX) which used for arithmetic and data movement. Each register can be addressed as either 16-bit or 8 bit value. Example, **AX** register is a 16-bit register, its upper 8-bit is called **AH**, and its lower 8-bit is called **AL**. Bit 0 in AL corresponds to bit 0 in AX and bit 0 in AH corresponds to bit 8 in AX as shown in the figure below.

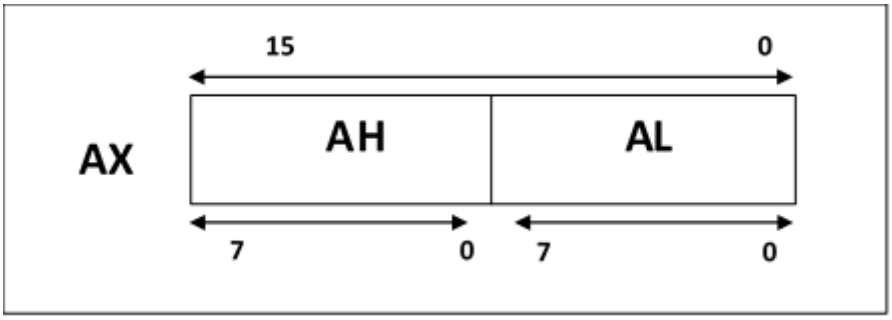


Fig. 4: AX register

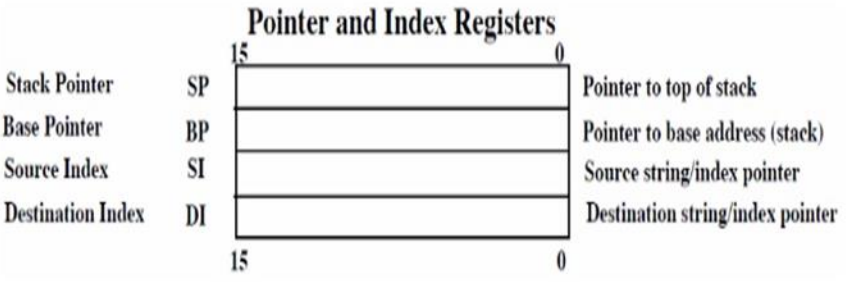
- **Accumulator register (AX)**: It is the accumulator register because it is favored by the CPU for arithmetic operations. Other operations are also slightly more efficient when performed using **AX**.

- **Base register**: The BX register can hold the address of a procedure or variable. Three other registers with this ability are **SI, DI** and **BP**. The **BX** register usually contains a data pointer used for based, based indexed or register indirect addressing. BX register can also perform arithmetic and data movement.

- **Count register:** The CX register acts as a counter for repeating or looping instructions. These instructions automatically repeat and decrement **CX**.
- **Data register**: In integer 32-bit multiply and divide instruction the DX register contains high order word of the resulting number. DX register can be used as a port number in I/O operations.

b) Index and Pointer Register

There are four 16-bits registers two serve as pointers and two serve as indexes. These registers usually store offset address used for addressing within the segment.



- **Source Index (SI)**: is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing. As well as source data address in string manipulation instructions. Used in conjunction with DS register to point to data locations in the data segment.
- **Destination Index (DI)** is a 16-bit register. Used with the ES register in string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.
- **Stack Pointer (SP)**: is a 16-bit register pointing to stack, it is used to hold the address of the top of the stack. The stack is maintained as LIFO with its bottom at the start of the stack segment (Specified by the SS segment register). Unlike the SP register, the BP can be used to specify the offset of other program segments.

- **Base Pointer (BP):** is a 16-bit register pointing to stack segment. It is usually used by subroutine to locate variables that were passed on stack by calling program. BP register is usually used for based, based indexed or register indirect addressing.

2. **Segment Registers**

Within the 1 MB of memory space the 8086/88 defines 16 segments, but four 64K-byte memory blocks are active at a time called the code segment, stack segment, data segment, and extra segment. Each of these blocks of memory is used differently by the processor.

The four segment registers (CS, DS, ES, and SS) are used to "point" at location 0 (the base address) of each segment as shown in fig. (5)

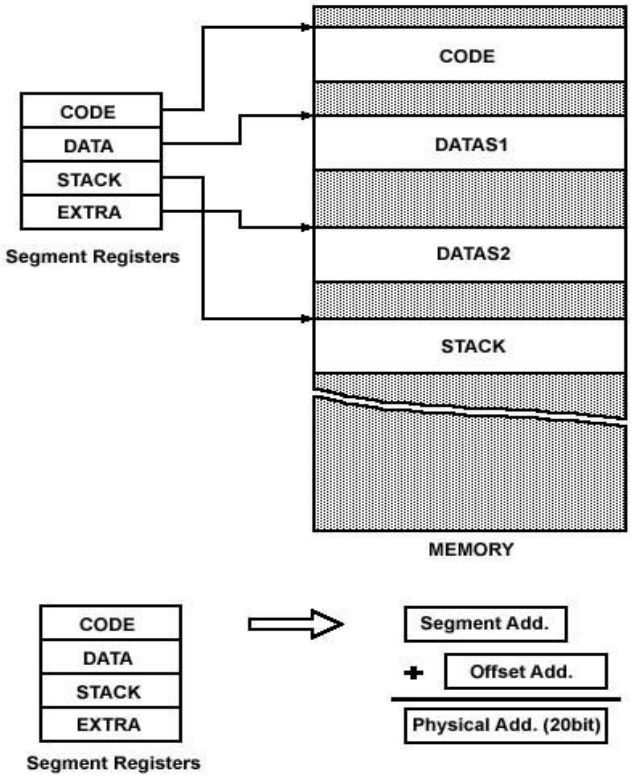


Fig. 5: physical memory organization

1) Code Segment (CS) is a 16-bit register containing address of 64 KB segment with program instructions. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

2) Stack segment (SS) is a 16- bit register containing address of 64KB segment with program stack**.** The stack segment in memory which the value of instruction pointer, status flags, and other registers are pushed in case of interrupt or subroutine call.

3) Data segment (DS) is a 16- bit register containing the starting address of the current data segment in which data are stored. It provides a read/write memory space.

4) Extra segment (ES) used to hold the starting address of Extra segment. Extra segment is provided for programs that need to access a second data segment. Segment registers cannot be used in arithmetic operations.

3. Instruction Pointer (IP) is a 16-bit register. This is important register which is used to control which instruction the CPU executes. The IP, or program counter, is used to store the memory location of the next instruction to be executed (offset address relative to CS). The CPU checks the IP to ascertain which instruction to carry out next, and then updates the IP to point to the next instruction. Thus the IP will always point to the next instruction to be executed.

4. Flag 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. 8086 has 9 flags and they are divided into two categories as shown in figure below:

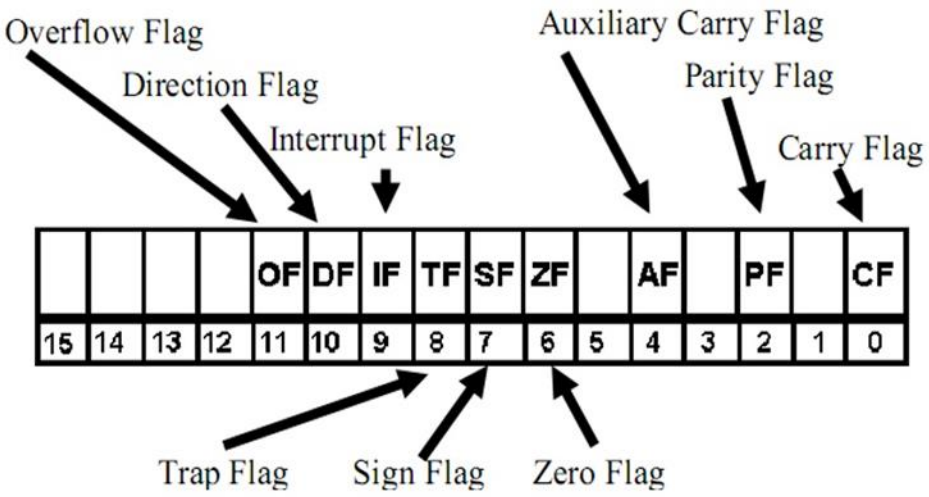


Fig. 6: Flag register

1. **Status Flags**

The status flags are set/reset depending on the results of some arithmetic or logical operations during program execution:

- **Carry Flag (CF):** this flag is set to **1** if there is a carry out or borrow in for the most significant bit of the result during the execution of an arithmetic instruction otherwise, CF is reset.
- **Auxiliary Flag (AF):** If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 is AF 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.
- **Overflow Flag (OF):** It occurs when signed numbers overflow. An OF indicates that the result has exceeded the capacity of machine.
- **Example:** Suppose AL= CFH if we execute the instruction ADD AL, C1H

AL 1100 1111 CF=1 AF=1 PF=1

C1H 1100 0001 ZF=0 SF=1 OF=0

90H 1001 0000

2. Control Flags

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

- **Trap Flag (TP):**

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 STI 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.

Memory Segmentation:

- The memory in an 8086 based system is organized as segmented memory.
- The CPU 8086 is able to access 1MB of physical memory. The complete 1MB of memory can be divided into 16 segments, each of 64KB size and is addressed by one of the segment register.
- The 16-bit contents of the segment register actually point to the starting location of a particular segment. The address of the segments may be assigned as 0000H to F000h respectively.
- To address a specific memory location within a segment, we need an offset address. The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH.
- A program can have more than four segments, but can only access four segments at a time.

Physical address is calculated as below:

Ex:

Segment address =1005H

Offset address =5555H

Segment address =1005H = 0001 0000 0000 0101

Shifted left by 4 Positions=0001 0000 0000 0101 0000 + Offset address = 5555H= 0101 0101 0101 0101

Physical address=155A5H =0001 0101 0101 1010 0101

Physical address = Segment address * 10H + Offset address.

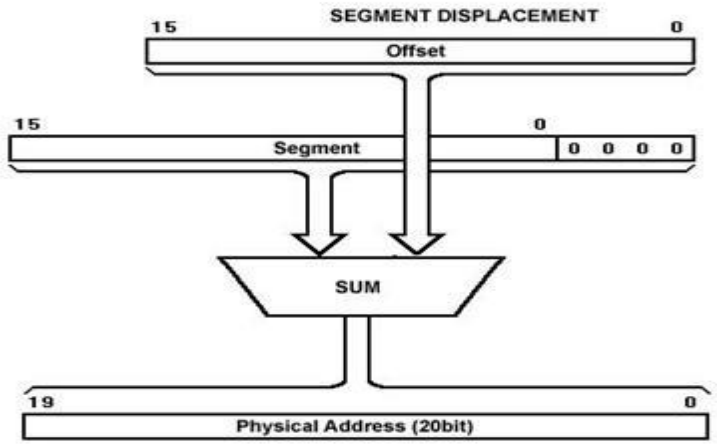


Fig. 7: Generating a physical address

The main advantages of the segmented memory scheme are as follows:

- It provides a powerful memory management mechanism.
- Data related or stack related operations can be performed in different segments.
- Code related operation can be done in separate code segments.
- It allows to processes to easily share data.
- It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.

- It is possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

Addressing modes:

When the 8086 executes an instruction, it performs the specified function on data. The data are called its **operands** and may be part of the instruction reside in one of the internal registers of the 8086, stored at an address in memory, or held at an I/O port. To access these different types of operands, the 8086 is provided with various addressing modes as follow:

1. Immediate addressing mode
2. Register addressing mode
3. Direct addressing mode
4. Register indirect addressing mode
5. Based addressing mode
6. Indexed addressing mode.
7. Based indexed addressing mode
8. String addressing mode.

1. **Immediate addressing mode**: In this type of addressing, immediate data is a part of instruction instead of the contents of a register or memory location, and it may be 8-bit or 16-bit in size.

Example:

MOV AL, 015H

2. **Register addressing mode:** In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.

Example:

MOV AX, BX

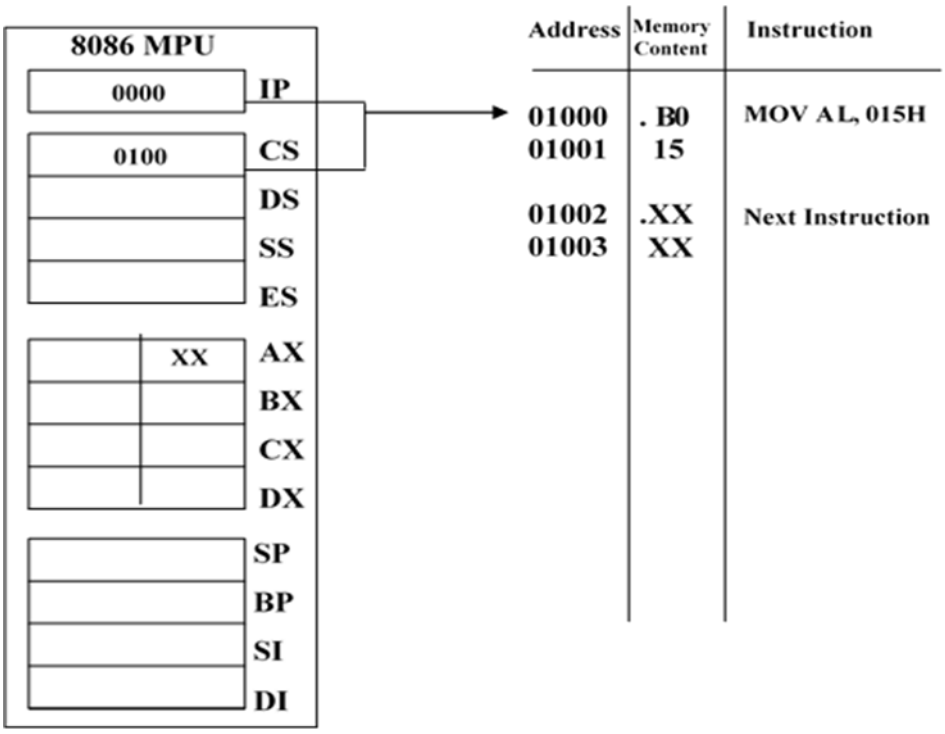


Figure 8 (a): Immediate addressing mode before execution.

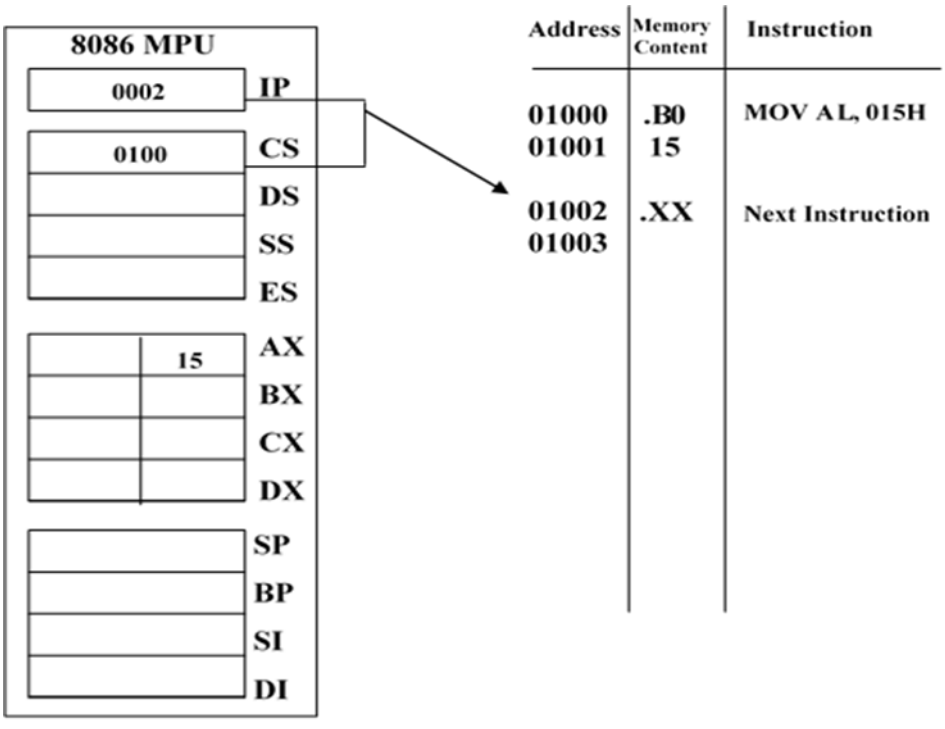


Figure 8 (b): Immediate addressing mode after execution.

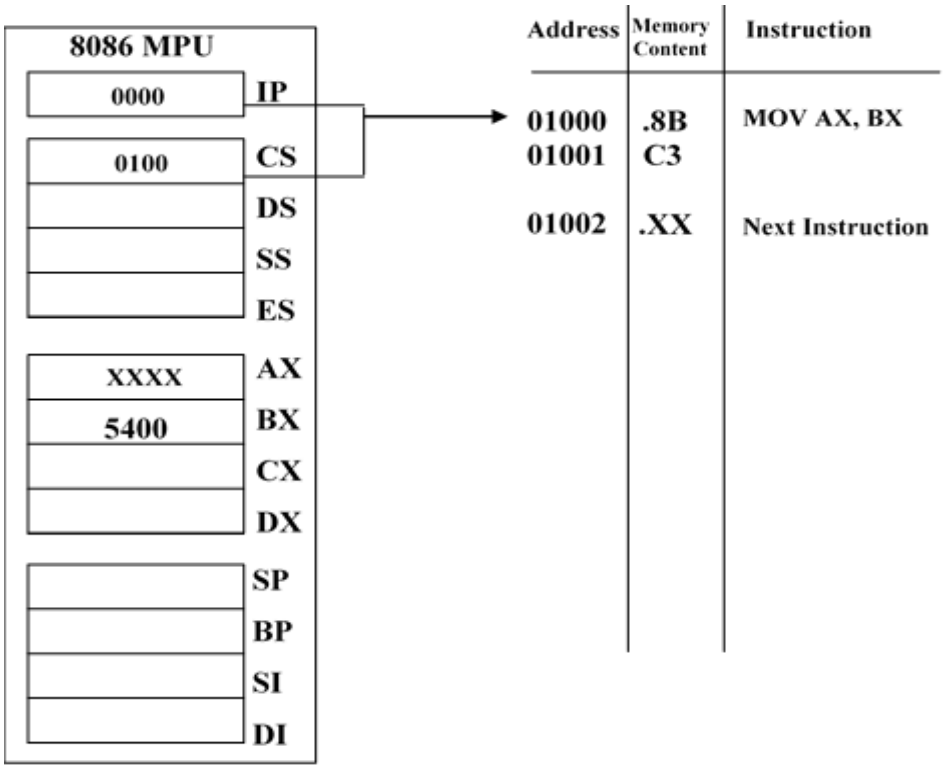


Figure 9 (a): Register addressing mode before execution.

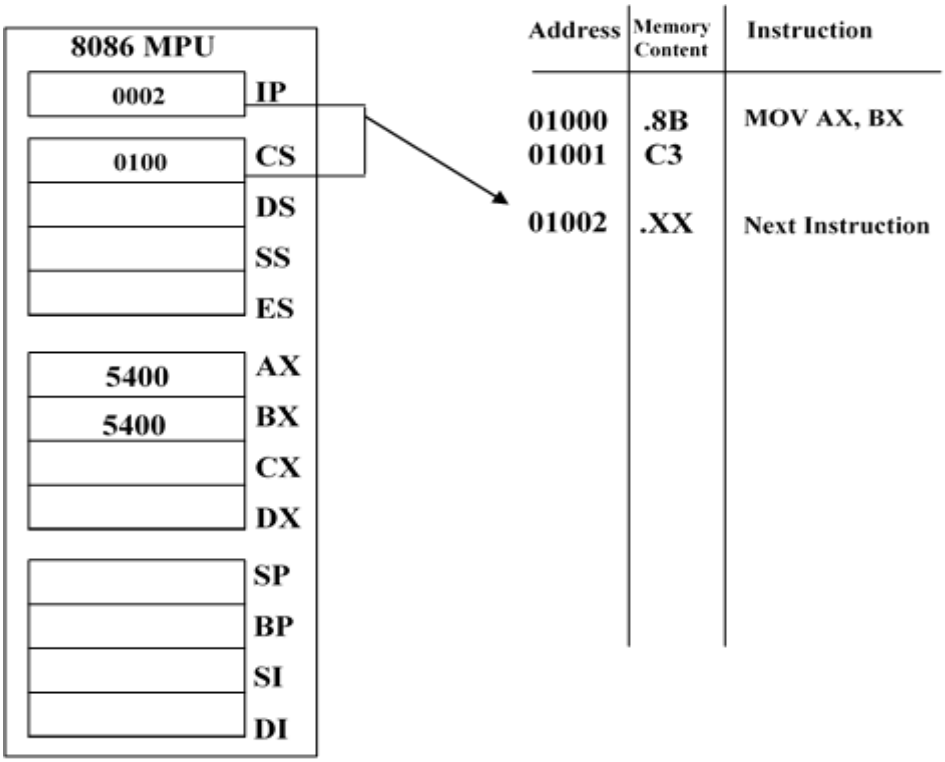


Figure 9 (b): Register addressing mode after execution.

3. Direct addressing mode: In the direct addressing mode a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

Example:

MOV CX, [1234H]

Here, the operand resides in a memory location in the data segment, whose effective address may be completed using 1234H as the offset address and content of DS as segment address. The effective address here, is 10H * DS + 1234H.

4. **Register indirect addressing mode**: Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset register. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI or BP registers. The default segment is either DS or ES or SS. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Example:

MOV AX, [SI]

Here, data is present in a memory location in DS whose offset address is in SI. The effective address of the data is given as 10H * DS + SI.

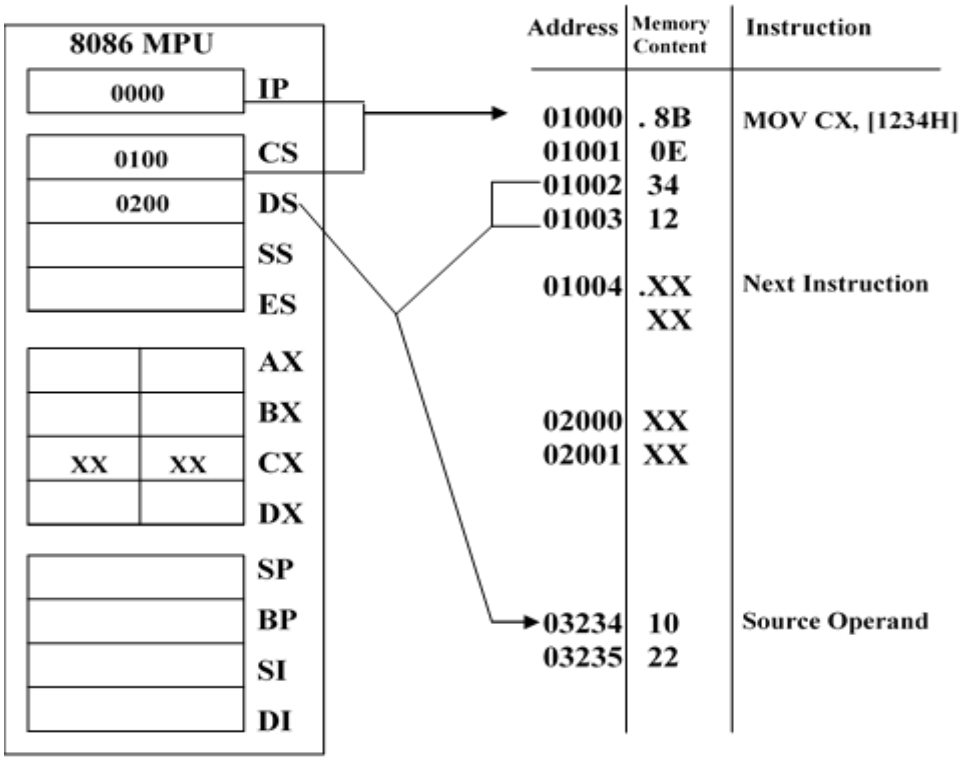


Figure 10 (a): Direct Addressing mode before execution.

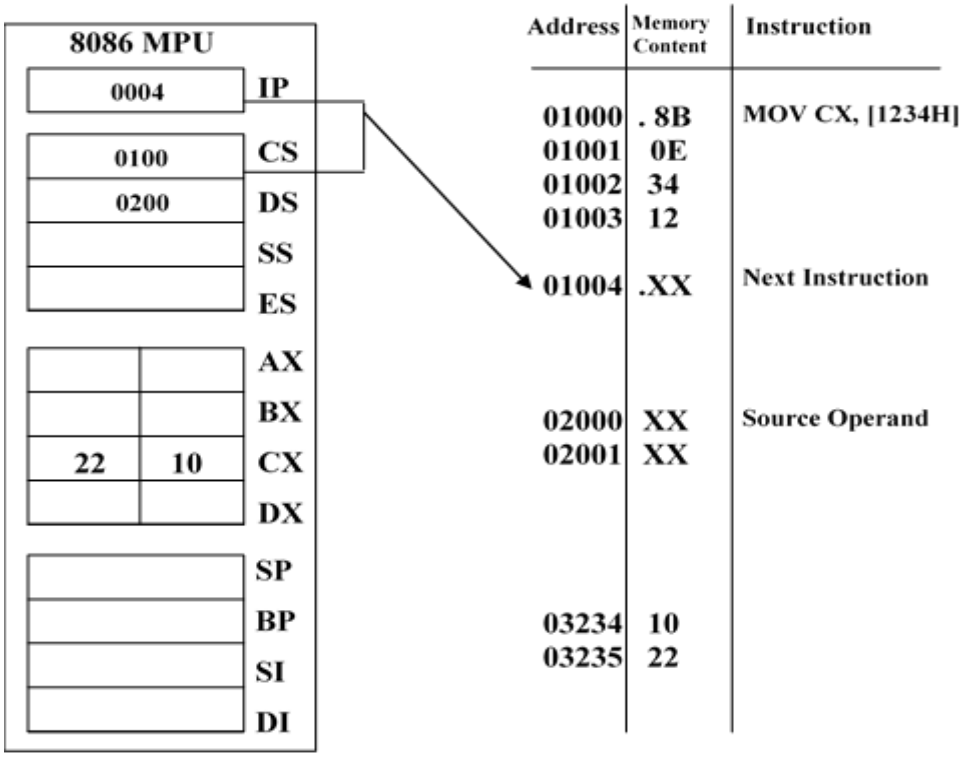


Figure 10 (b): Direct Addressing mode after execution.

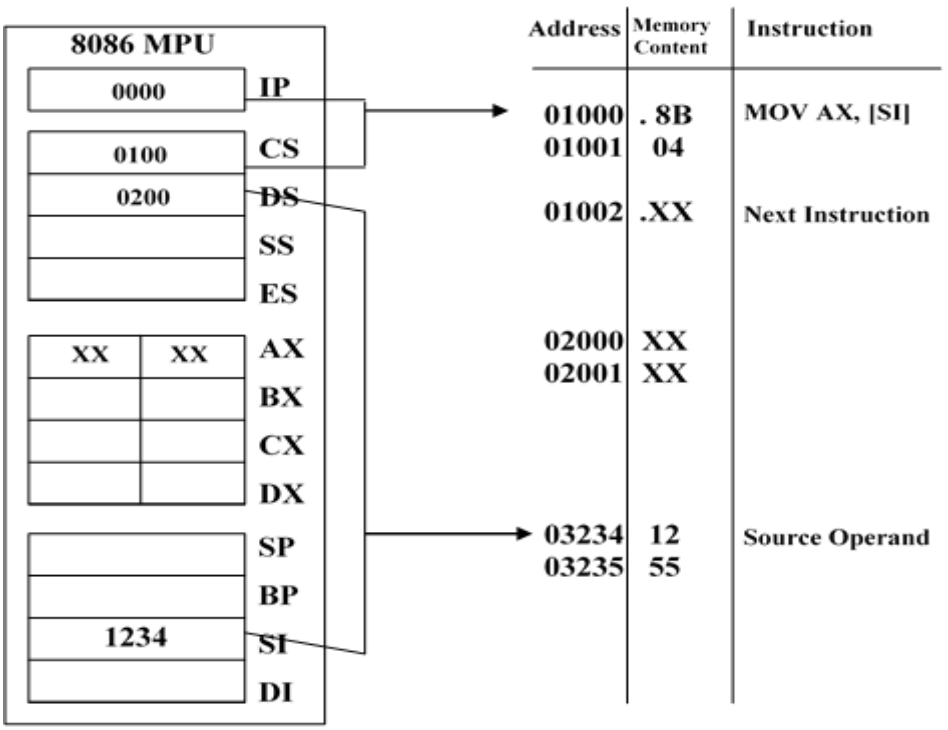


Figure 11 (a): Register Indirect Addressing before execution

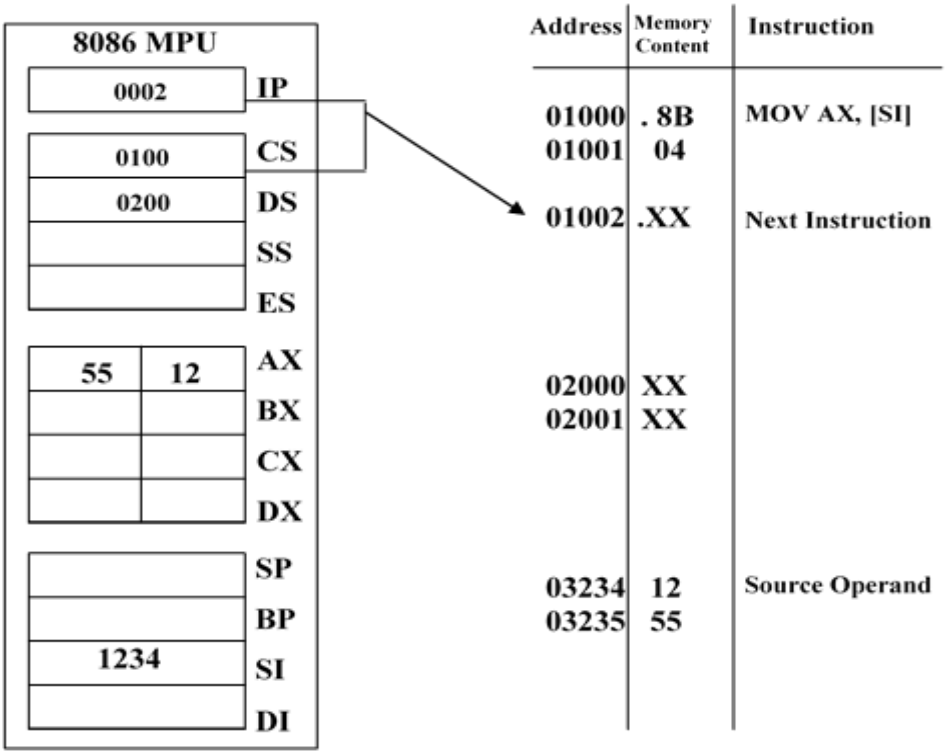


Figure 11(b): Register Indirect Addressing mode after execution.

5. **Based addressing mode**: In the based addressing mode, the physical address of the operand is obtained by adding a direct or indirect displacement to the contents of either BX or BP and the current value in DS and SS, respectively.

Example:

MOV [BX] + 1234H, AL; EA=BX+1234H, PA=DS*10H+EA

After execute the instruction the content of register AL is moved to memory location specified by PA (as shown in fig. 12)

6. **Indexed Addressing mode**: In the Indexed addressing mode, the effective address of the operand is obtained by adding a direct or indirect displacement to the contents of either SI or DI register. Indexed addressing works identically to the based addressing, it uses the contents of one of the index registers, instead of BX or BP, in the generation of the physical address.

Example:

MOV BL, [SI]+1234H; EA=SI+1234H, PA=DS*10H+EA

After execute the instruction the byte of data stored at this location (PA), is read into lower byte BX.

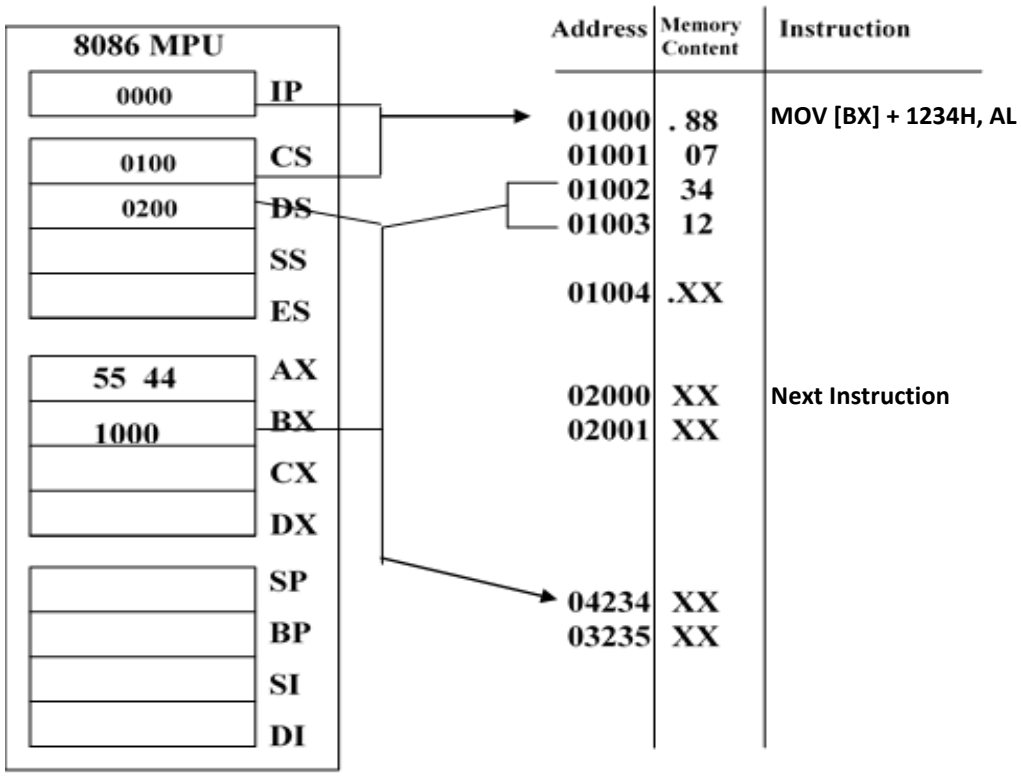


Figure 12(a): Based Addressing before execution

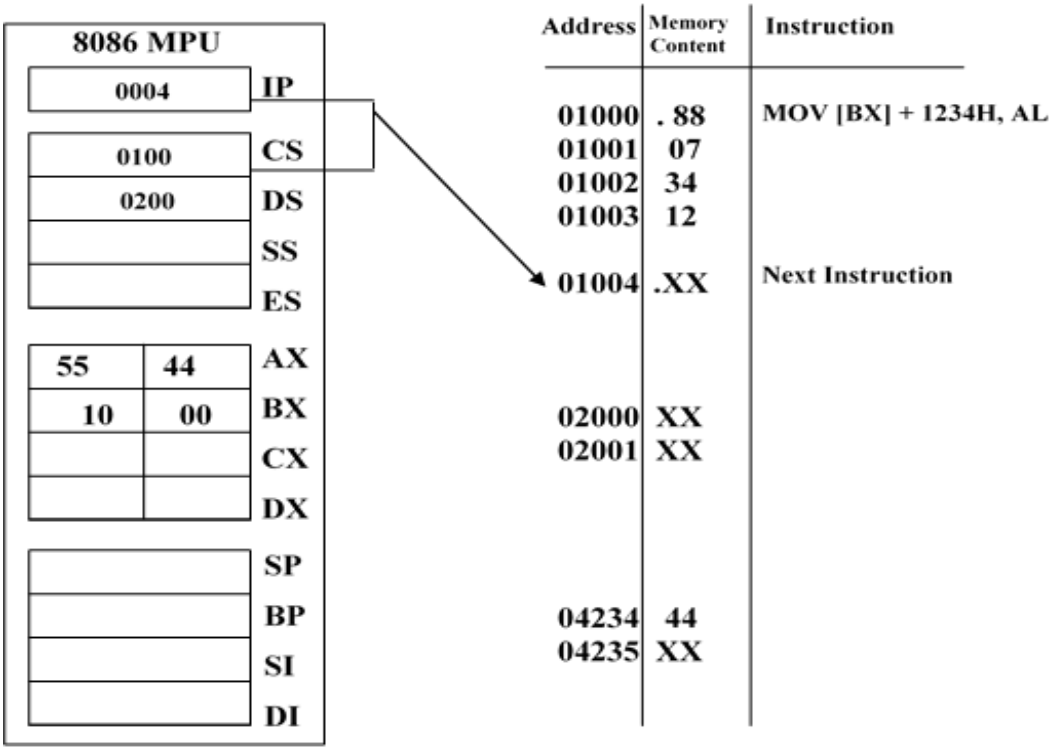


Figure 12(b): Based Addressing mode after execution.

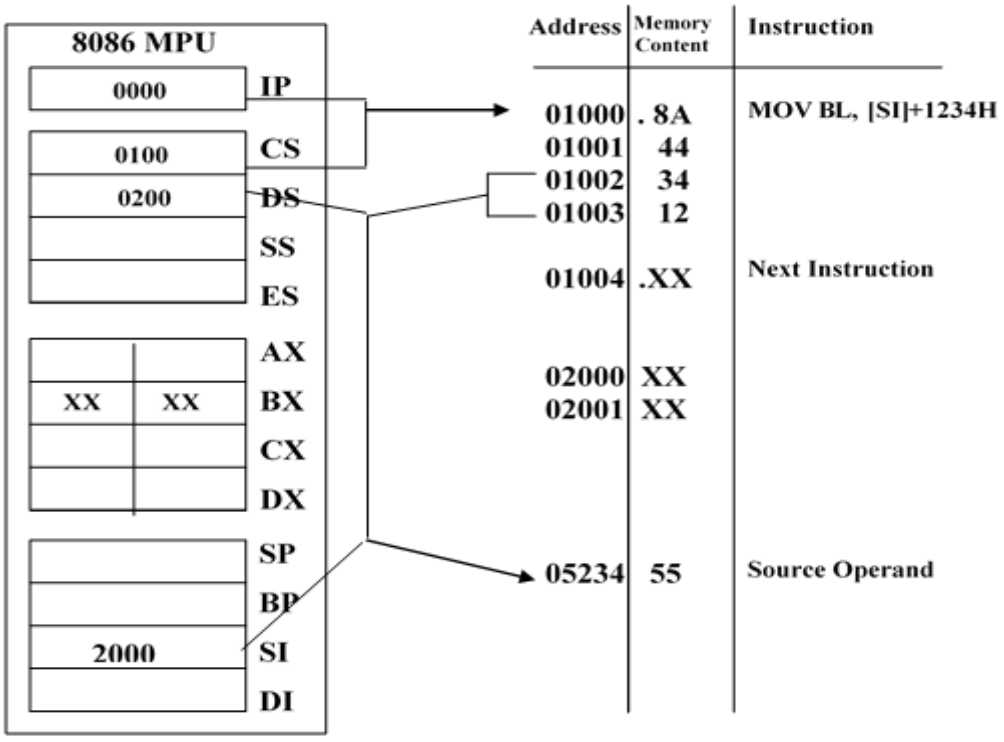


Figure 13(a): Indexed Addressing before execution.

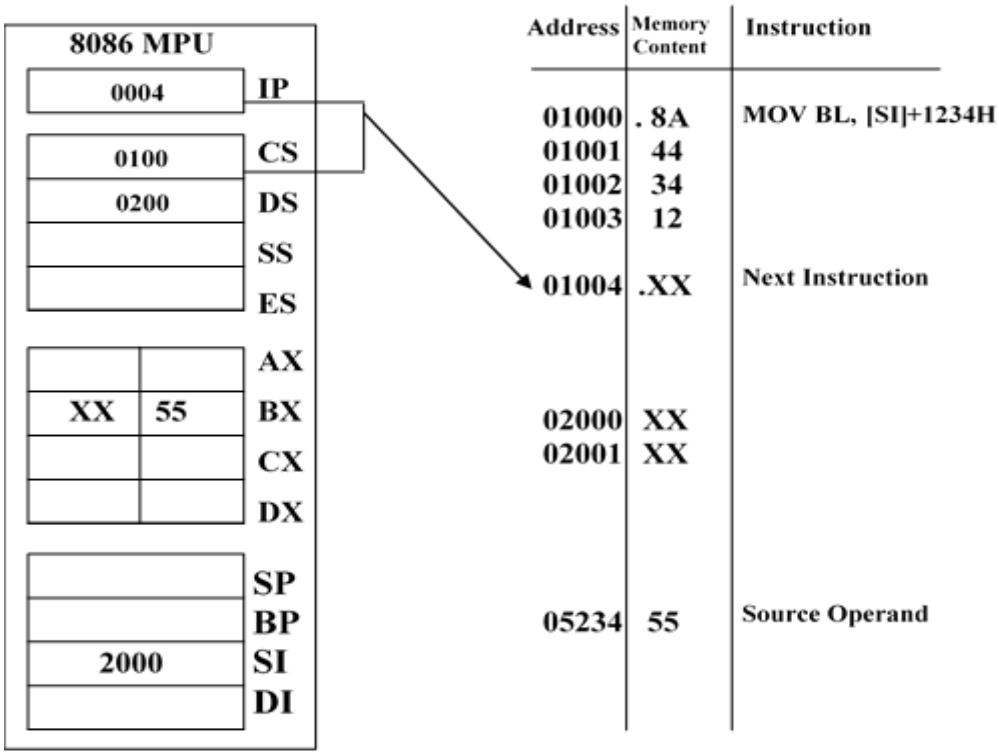


Figure 13(b): Direct Indexed Addressing mode after execution.

7. **Based -Index addressing mode*:*** Combining the based addressing mode and the indexed addressing mode together results in a new, more powerful mode known as based indexed addressing. The effective address of data is formed, by adding an 8 or 16-bit displacement with the content of a base register (any one of BX or BP) and the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example:

MOV AX, 1234H [BX] [SI]

Here, 1234H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as

EA= [BX] + [SI] + 1234H

And the physical address

PA=10H * DS + EA

8. String Addressing Mode

The string instructions of the 8086's instruction set automatically use the source and destination index registers to specify the effective addresses of the source and destination operands, respectively.

Example:

MOVSB

The physical address for the source operand

10H * DS + [SI]

The physical address for the destination operand

10H * ES + [DI]

Notice that neither SI nor DI appears in the string instruction, but both are used during its execution.

9. **Input/Output mode:**

This addressing mode is related with input output operations. Example:

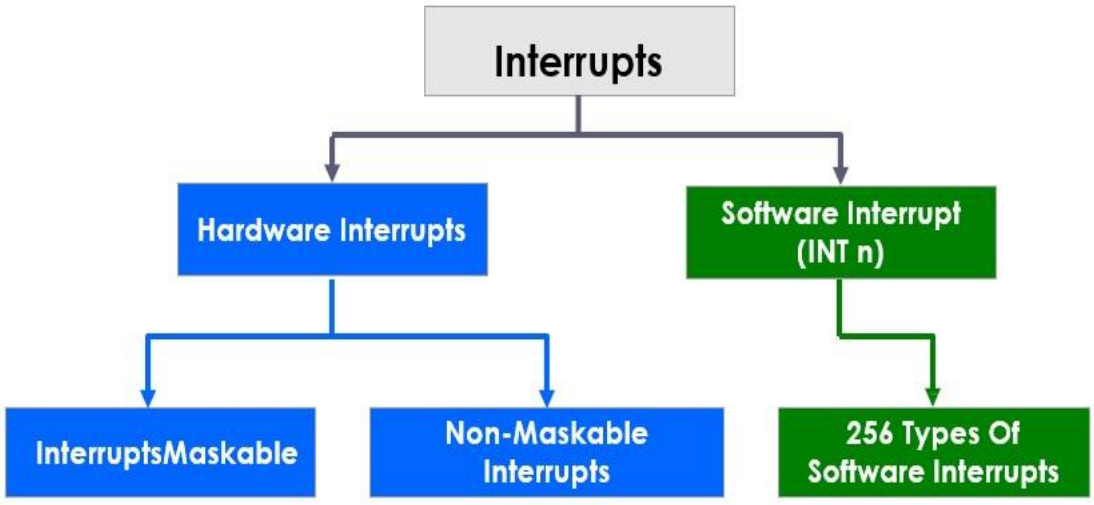
IN Al, 45

OUT DX, AL

Interrupts:

An interrupt is a condition that halts the microprocessor temporarily to work on a different task and then return to its previous task. Interrupt is an event or signal that request to attention of CPU. This halt allows peripheral devices to access the microprocessor.

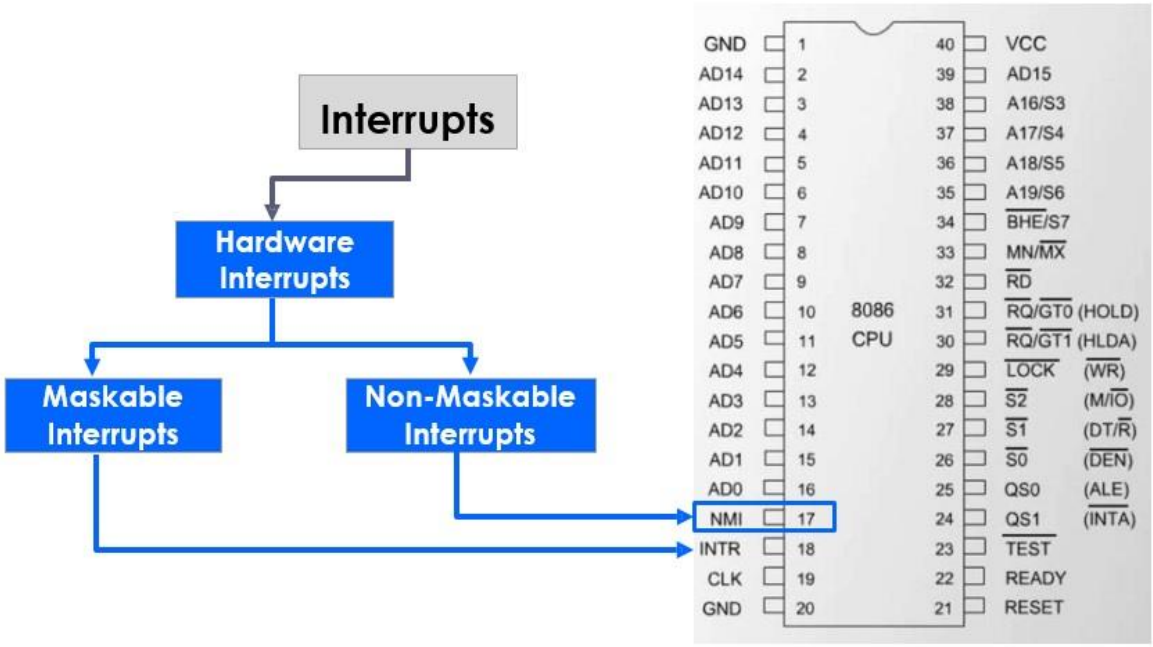
Whenever an interrupt occurs the processor completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR is a program that tells the processor what to do when the interrupt occurs. After the execution of ISR, control returns back to the main routine where it was interrupted.

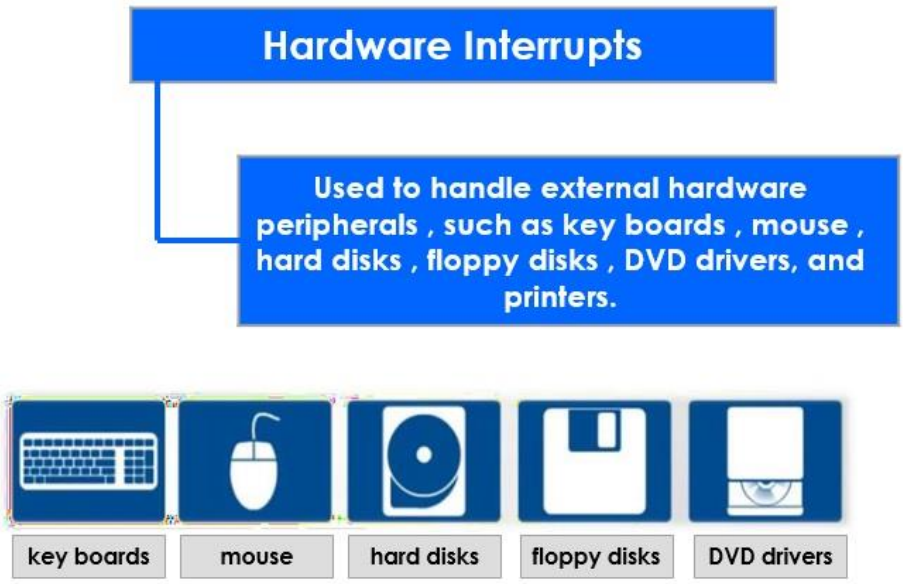


Broadly the interrupts are divided into two types. They are external (hardware) Interrupts and internal (Software) Interrupts. The hardware interrupts are classified as non-maskable and maskable interrupts. The hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor. Whereas internal interrupts are initiated by the state of the CPU (e.g. divide by zero error) or by an instruction. So, the software interrupt is one which interrupts the normal execution of a program of the microprocessor. The 8086 has two hardware interrupt pins namely NMI and INTR. In the two, the NMI is a non-maskable interrupt and the INTR interrupt request is a maskable interrupt which has lower priority .The third pin associated with the hardware interrupts are the INTA called interrupt acknowledge.

Hardware interrupts

The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8086 processor has two interrupt pins INTR and NMI.





Maskable and Non-Maskable Interrupts

- The processor has the facility for accepting or rejecting hardware interrupts. Programming the processor to reject an interrupt is referred to as masking or disabling and programming the processor to accept an interrupt is referred to as unmasking or enabling. In 8086 the interrupt flag (IF) can be set to one to unmask or enable all hardware interrupts and IF is cleared to zero to mask or disable a hardware interrupts except NMI. The interrupts whose request can be either accepted or rejected by the processor are called maskable interrupts.
- The interrupts whose request has to be definitely accepted (or cannot be rejected) by the processor are called non-maskable interrupts. Whenever a request is made by non-maskable interrupt, the processor has to definitely accept that request and service that interrupt by suspending its current program and executing an ISR. In 8086 processor all the hardware interrupts initiated through INTR pin are maskable by clearing interrupt flag (IF). The

interrupt initiated through NMI pin and all software interrupts are non-maskable.

- The programmer cannot control when a Non-Maskable Interrupts is serviced and the processor has to stop the main program to execute the NMI service routine.
- In Maskable Interrupts the programmer can choose to mask specific interrupts and re-enable them later.
- Non-Maskable Interrupts used :

1. during power failure
2. during critical response time
3. during non-recoverable hardware errors
4. watchdog interrupt
5. during memory parity errors

Software interrupts

Coming to the software interrupts, 8086 can generate 256 interrupt types through the instruction INT n .Any of the 256 interrupt types can be generated by specifying the interrupt type after INT instruction. For example the first five types are as follows:

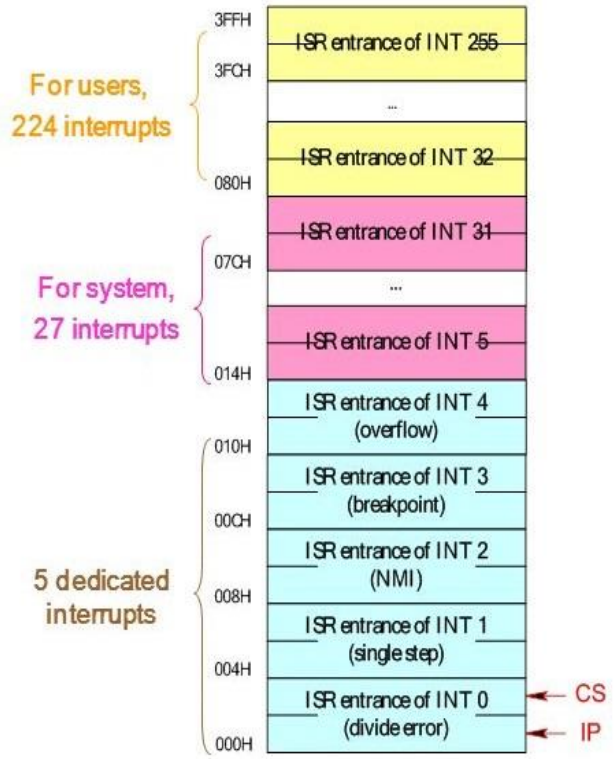
- TYPE 0 interrupt represents division by zero situation.
- TYPE 1 interrupt represents single-step execution during the debugging of a program.
- TYPE 2 interrupt represents non-maskable NMI interrupt.
- TYPE 3 interrupt represents break-point interrupt.
- TYPE 4 interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

Interrupt vector table

Interrupt vector table on 8086 is a vector that consists of 256 total interrupts placed at first 1 kb of memory from 0000h to 03ffh, where each vector consists of segment and offset address for ISR. The call to interrupt service routine is similar to far procedure call.

The size for each interrupt vector is 4 bytes (2 word in 16 bit), where 2 bytes (1 word) for segment and 2 bytes for offset of interrupt service routine address. So it takes 1024 bytes (1 kb) memory for interrupt vector table.



- 256 interrupts
  - 0 ~ 4 dedicated
  - 5 ~ 31reserved for system use
    - 08H ~ 0FH : 8259A
    - 10H ~ 1FH : BIOS
  - 32 ~ 255 reserved for users
    - 20H ~ 3FH : DOS
    - 40H ~ FFH : open

When an interrupt occur, the following action are taken:

1. Push flag register on the stack
2. clear IF and TF

3. Push CS and IP register, on the stack
4. Load CS with the 16-bit data at memory address (INT-type *4+2)
5. Load IP with the 16 bit data at memory address (INT-type *4).

The last instruction of ISR is (IRET) instruction, it actions are:

1. POP the 16-value on top of stack into IP register
2. POP the 16-value on top of stack into CS register
3. POP the 16-value on top of stack into flag register.

How can more than one device interrupt?

Computer typically have more than one I/O device requesting interrupt service, like keyboard, hard disk, floppy disk, printer all generate an INT when they required the attention to CPU.

When more than one device INT CPU, we need a mechanism to priority these INT (if they come at the same time) and forward only one INT request at a time to the CPU while keeping other INT request pending for their service.

Examples of some interrupt instructions

1- INT 10h / AH=0Eh teletype output (print Al value)

Ex:

```
MOV AL, 'A'
MOV AH, 0Eh
INT 10h
```

2- INT 21h / AH=01h

Read character from standard input with echo, result is stored in AL.

EX:

```
MOV AH, 01h
INT 21h
```

3- INT 21h / AH=02h

Write character to standard output entry DL= character to write, after execution AL=DL

EX:

```
MOV AH, 02h
MOV DL, 'a'
INT 21h
```

Ex: Write assembly program to read five value from keyboard and store it in array

```
Org 100h
Mov ah, 01h
Mov cx, 5
aa: int 21h
Mov a[si], al
Inc si
Loop aa
Ret
a db 5 dup (0)
```

Ex: Write assembly program to print the value of array a where a= 1, 2, 3, 4, 5

```
Org 100h
Mov ah, 2
Mov cx, 5
aa: Mov Dl, a[si]
Int 21h
Inc si
Loop aa
Ret
a db 1, 2, 3, 4, 5
```

4- INT 33h (mouse driver interrupt)

- INT 33h / AX= 0001h (show mouse pointer)

Ex:

```
Mov ax, 1
Int 33h
```

-INT 33h / AX=0002h (Hide visible mouse pointer)

Ex:

```
Mov ax, 2
Int 33h
```

Input and Output

Input & Output (I/O) devices provide the means by which a computer system can interact with the outside worlds.

An I/O device can be a purely input device (e.g. KB, Mouse), a purely output device (printer, screen), or both input and output device like (e.g. disk)

Regardless of the intended purpose of I/O devices, all communication with these devices must involve the system bus. However, I/O devices are not directly connected to the system bus. Instead, there is usually, On I/O controller that acts as an interface between the system and the I/O devices.

Accessing I/O devices

As programmer, you can have direct control to any of the I/O devices (through their associated I/O controller).

It is a waste of time and effort if every one had to develop their own routines to access I/O devices. In addition system resource could be abused either intentionally or accidentally. For instance, and improper disk drive could erase the content of a disk due to a bug in the driver routine.

To avoid this problem and to provide a standard way of accessing I/O devices, OS provide routine to convent all access I/O devices. Typically, access to I/O devices can be obtain from two layer of system software, the basic I/O system (BIOS) and

the OS,BIOS is ROM resident and is a collection of routine that control the I/O devices. Both provide access to routine that control I/O devices through a mechanism called INT (interrupt).

I/O Address Space and Data Transfer

As we know I/O ports in the 8086 MPU can be either byte wide or word wide. The port that is accessed for input or output of data is selected by an I/O address. The address is specified as port of the instruction that performs the I/O operation.

I/O addresses are 16 bit in length and are output by the 8086 to the I/O interface over bus lines AD_0 through AD_{15}, the most significant bit A_{16}-A_{19} of the memory address are held at the 0 logic (not used).

Below Figure 19 show a map of I/O address space of the 8086 system. This is an independent 64-KB address space that is dedicated for I/O devices. Notice that its address range is from 0000_{16}-$FFFF_{16}$. Moreover, notice that the eight ports located from address 00F8 to 00FF are specified as reserved. These port addresses are reserved by Intel for use in their future HW and SW products.

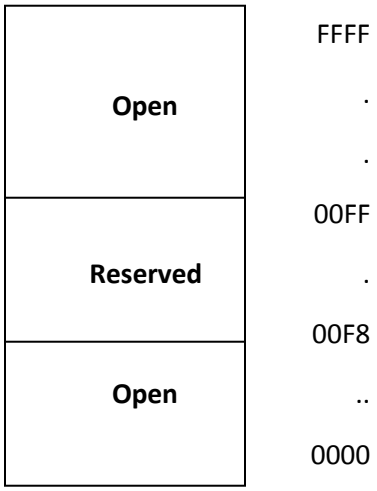


Figure: I/O Address Space

Data transfer between the MPU and I/O devices are performed over the data bus. Word transfer take place over the complete data bus D_0 to D_{15}, and can required either one or two bus cycle.

Ports: a port is a device that connects the processor to the external world through a port processor, receive a signal from an input device and send a signal to an output device.

Input / Output Instruction

The instruction set contains one type of instruction that transfer information to an I/O device (OUT) and another to read information from an I/O device (IN).

Instruction	Meaning	Format	Operation
IN	Input direct	IN ACC, PORT	ACC ⟵ PORT
	Input indirect	IN ACC, DX	ACC ⟵ (DX)
OUT	Output direct	OUT PORT,ACC	PORT ⟵ ACC
	Output indirect	OUT DX, ACC	(DX) ⟵ ACC

- ACC = AL or AX

Example 1: write a sequence of inst that will output FF_{16} to a byte wide output port at address AB_{16} of the I/O addresses space.

Solution: first the AL register is loaded with FF_{16} as an immediate operand in the instruction

MOV AL, $0FF_H$

Now the data in AL can be output to the byte wide output port with the instruction

OUT $0AB_H$, AL

Example2: write a series of instruction that will output FF_{16} to an output port located at address $B000_{16}$ of the I/O address space

Solution: the DX register must first be loaded with the address of the output port

MOV DX, 0B000H

Next, the data that is to be output must be loaded into AL

MOV AL, $0FF_H$

Finally, the data are output with the instruction

OUT DX, AL

Example 3: data are to be read in from two byte wide input port at address AA_{16} and $A9_{16}$ respectively, and then output to a word wide output port at address $B000_{16}$. Write a sequence of instruction to perform this I/O operation:

Solution: we first read in a byte from the port at address AA_{16} into AL and move it to AH

IN AL, $0AA_H$

MOV AH, AL

The other byte can be read into AL

IN AL, $0A9_H$

To writhe out the word of data in AX, we can load DX with the address$B000_{16}$ and use a variable output instruction

MOV DX, $0B000_H$

OUT DX, AX

Isolated and Memory I/O

There are two different method of interfacing I/O to the MPU. In the isolated I/O scheme, the IN, OUT instruction transfer data between the MPU (ACC or memory) and the I/O device.

Isolated I/O: it is the most common I/O transfer techniques. The addressed for insolated I/O device, called ports, are separate from the memory. Because the ports are separate from the memory, because the ports are separate. The user can expand the memory to its full size without using any of memory space for I/O device.
A disadvantage of isolated I/O is that, the data transferred between I/O and the MPU must be accessed by the IN, OUT instruction. See Figure 20.

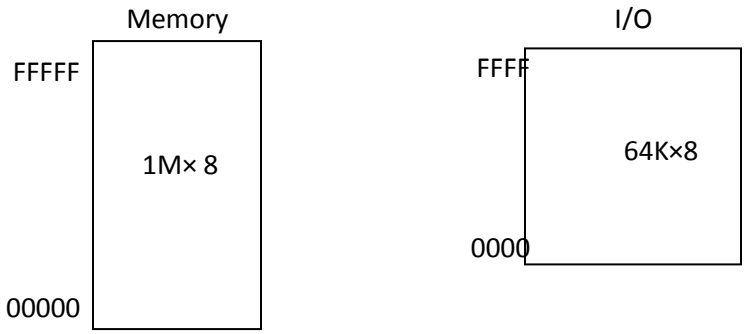


Figure: Isolated I/O.

Memory- Map I/O: Unlike isolated I/O, memory mapped I/O does not use the IN or OUT instruction. Instead, it uses any instruction that transfer data between the MPU and memory. A memory mapped I/O device is treated as a memory location in memory map.
The main advantage of memory-mapped I/O is that any memory transfer instruction can be used to access the I/O

The main disadvantage is that a portion of the memory systems used as the I/O map. This reduced the amount of memory available to application. See Figure 21.

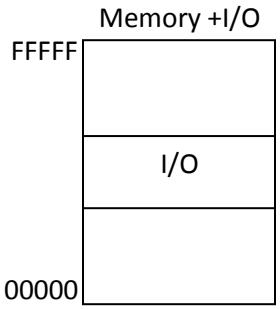


Figure: Memory-Mapped I/O