UNIVERSITY OF BAGHDAD

College of Education for Pure Science

(Ibn Al-Haitham)

Department of Computer Science

Microprocessor 8086

Second Stage

Dr. Alaa Abdul Hameed Abdul Lateef

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.

Types of Computers

There are four types of computer:

1. **Microcomputers - Personal Computers**

Microcomputers sit on, beside or under a desk. They process data quickly and are designed for one user. The cost is relatively low, depending on the type, model and features. These computers are found in small businesses, schools and homes. The size of the components that can be used in or with a computer have decreased significantly as well as becoming more cost effective, There are many different types of programs that can be used with a microcomputer, there are generally two types of microcomputers: the PC (personal computer) based on the original IBM machine, or the Macintosh designed by Apple.

Microsoft Office is available for both the PC and Apple machines. The software works exactly the same on either machine. The majority of companies use PCs to handle their work requirements. This may primarily be due to the wide variety of software programs available to the PC machine and the relative lower cost of a PC as compared to an Apple machine.

2. **Supercomputer**

A supercomputer is considered to be the fastest type of computer available. It is very expensive due to the amount of information that needs to be processed on a daily basis. The first supercomputer was created in 1958 with tremendous advancement in technology over the past years. Supercomputers generally have specialized software programs installed and are typically used for scientific and engineering tasks such as forecasting weather, nuclear energy research, resource exploration, or animation. Supercomputers put all its power into running and performing calculations on a few programs as quickly as it can. Typically, it can handle very large databases and perform a huge number of calculations very quickly.

3. **Mainframe computers**

Mainframe computers are large enough to fill an entire room and require a large capital investment. They can simultaneously handle hundreds of different programs and users without sacrificing performance. They process large volumes of data at an incredible speed. Mainframes are commonly found in government agencies or large organizations, e.g., telephone companies, credit card companies, airlines, or universities. For instance, you access a mainframe computer whenever you use your bankcard at an Automated Teller Machine (ATM). Mainframe computers can sometimes be called centralized systems as they control the flow of data to and from computers or terminals.

4. **Minicomputers**

Minicomputers can fill part of a room, and often cost tens of thousands of dollars. They process data at a slower rate and in smaller volumes than the mainframe computers. Several people can use a minicomputer simultaneously; but as the number of users increases, each user would notice a reduction in speed. Minicomputers are commonly found in medium-sized manufacturing companies, legal or accounting firms, and department store where the scanners in a grocery store would link to a minicomputer.

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 Blocks of a microcomputer

All microcomputers consist of (at least):

1. Microprocessor Unit (MPU) MPU is the brain of microcomputer
2. Program Memory (ROM)
3. Data Memory (RAM)
4. Input / Output ports
5. Bus System

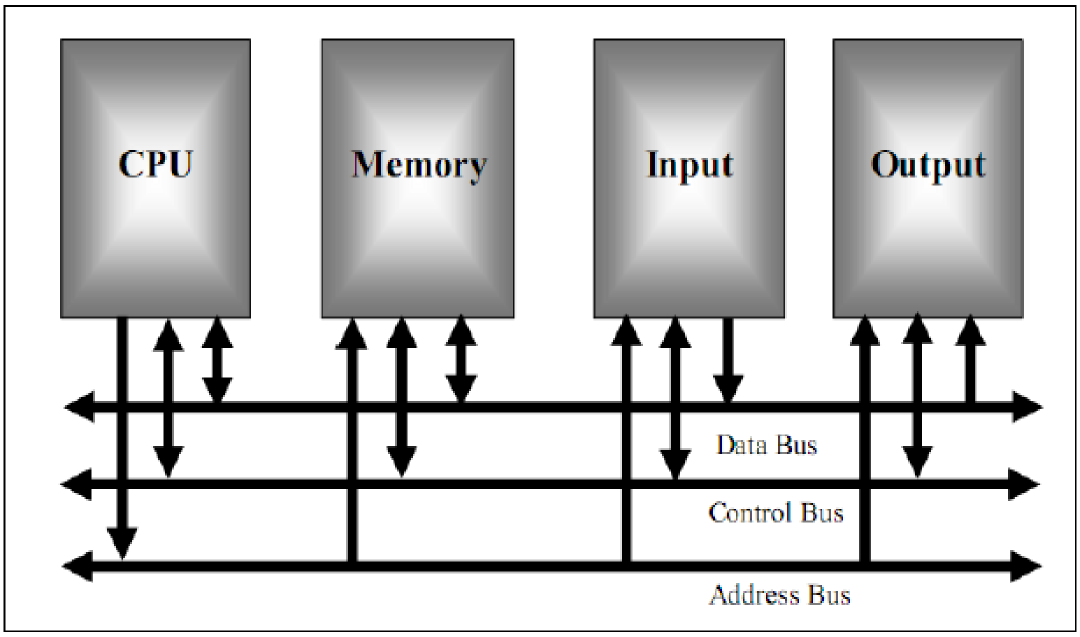


Fig. 1: Basic Block of a Microcomputer

Central Processing Unit (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 CPU 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:** Stores the data that is to be executed next, "very fast storage area".

RAM:

Random Access Memory (RAM): 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.

ROM:

Read Only Memory (ROM): 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 or other information appliance.

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.

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 2N memory locations.

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. To read a byte of data from a memory location, for example, the CPU sends out the address of the desired byte on the address bus and then sends out a memory read signal on the control bus.

Microprocessor 8086

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 220 =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 (Active low). 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 220 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. It requires single phase clock with 33% duty cycle to provide internal timing.
8. Prefetches up to 6 instruction bytes from memory and queues them in order to speed up the processing.
9. 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.

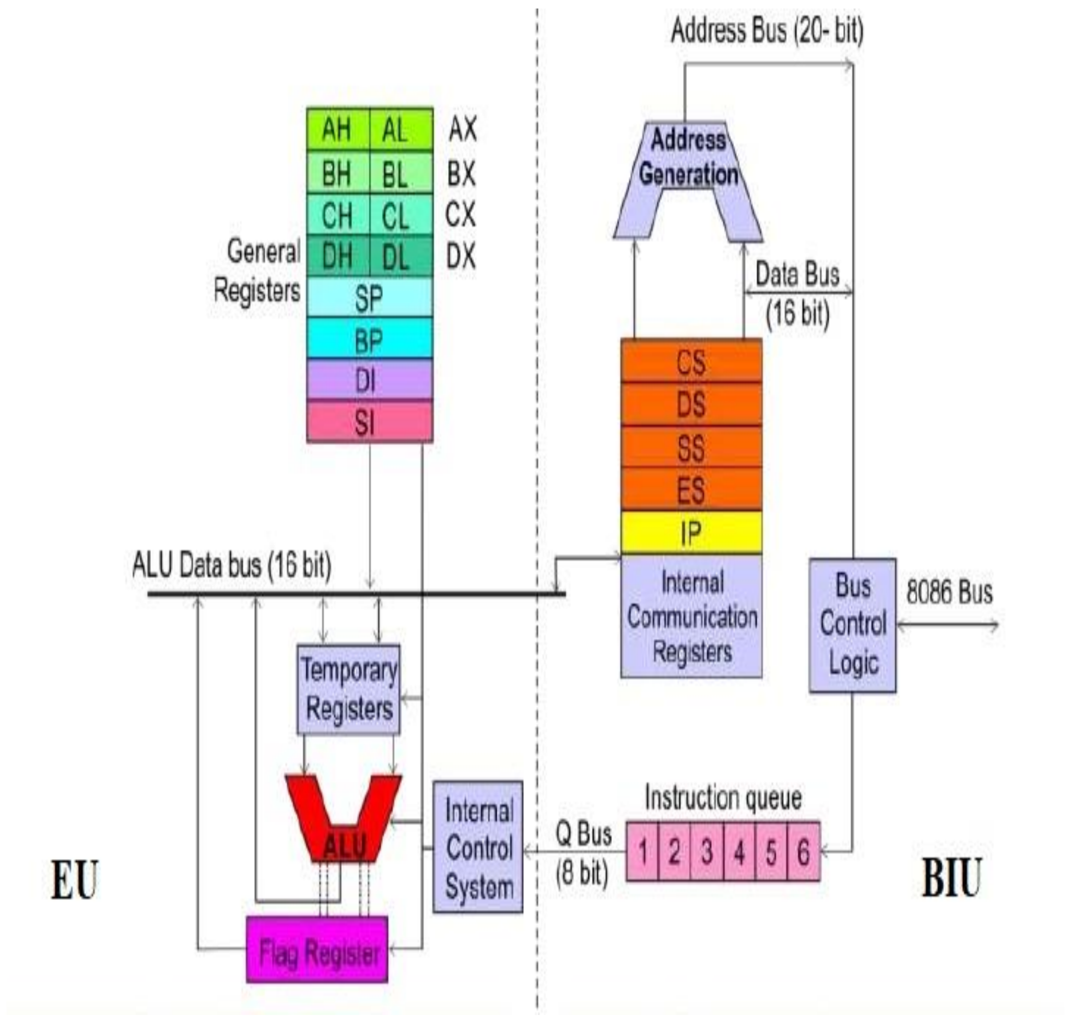


Fig.2: Architecture of 8086 Microprocessor

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 exe uting 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 t e BIU, decodes the ,generates operands if necessary, passes them to the IU 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.

8086 Has Pipelining Architecture:

- 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**.

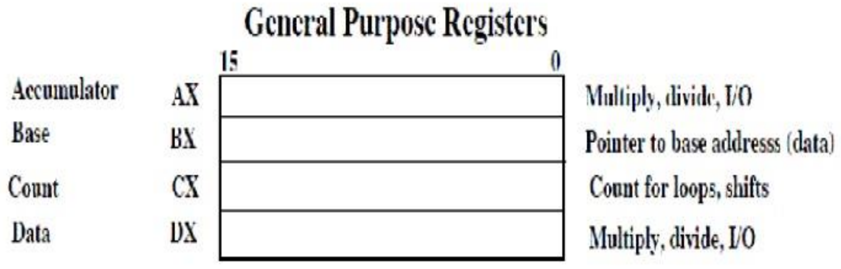
Register Organization

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

S.NO	Type	Register width	Name of the Registers
1	General Purpose Register(4)	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 (PSW)	16-bit	Flag Register

1. General Purpose Registers:

- ❖ **Accumulator register**: consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high order byte. Accumulator can be used for I/O operations and string manipulation.
- ❖ **Base register**: consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low order byte of the word, and BH contains the high order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

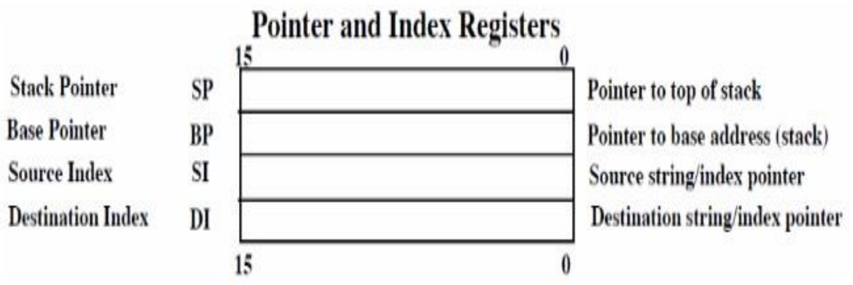


- ❖ **Count register:** consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. CL in this case contains the low order byte of the word, and CH contains the high order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation.
- ❖ **Data register**: consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. DL in this case contains the low order byte of the word, and DH contains the high order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high order word of the resulting number.

2. **Index and Pointer Register**

These registers can also be called as special purpose registers.

- ❖ **Source Index (SI)**: is a 16-bit register. SI is used foe 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. In short, Destination Index and SI Source Index registers are used to hold address.

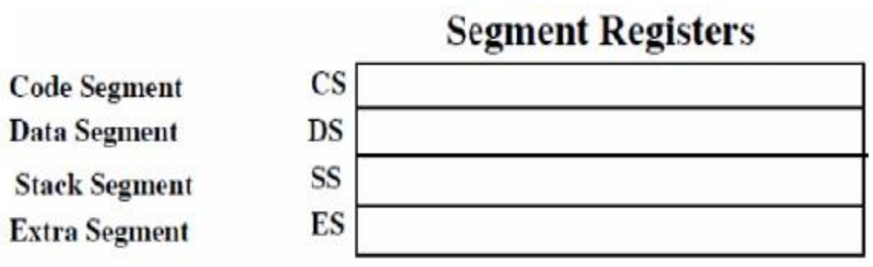

Pointer and Index Registers
15
0
Stack Pointer
SP
Pointer to top of stack
Base Pointer
BP
Pointer to base address (stack)
Source Index
SI
Source string/index pointer
Destination Index
DI
Destination string/index pointer
15
0

- ❖ **Stack Pointer (SP)**: is a 16-bit register pointing to program 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 program 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.

3. Segment Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses for code segment register.

- ❖ **Code Segment (CS)** is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.



- ❖ **Stack segment (SS)** is a 16- bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.
- ❖ **Data segment (DS)** is a 16- bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

- **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.

4. **Instruction Pointer (IP)** is a 16- bit register. This is a crucially 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. The CPU checks the program counter to ascertain which instruction to carry out next. It then updates the program counter to point to the next instruction. Thus the program counter will always point to the next instruction to be executed.

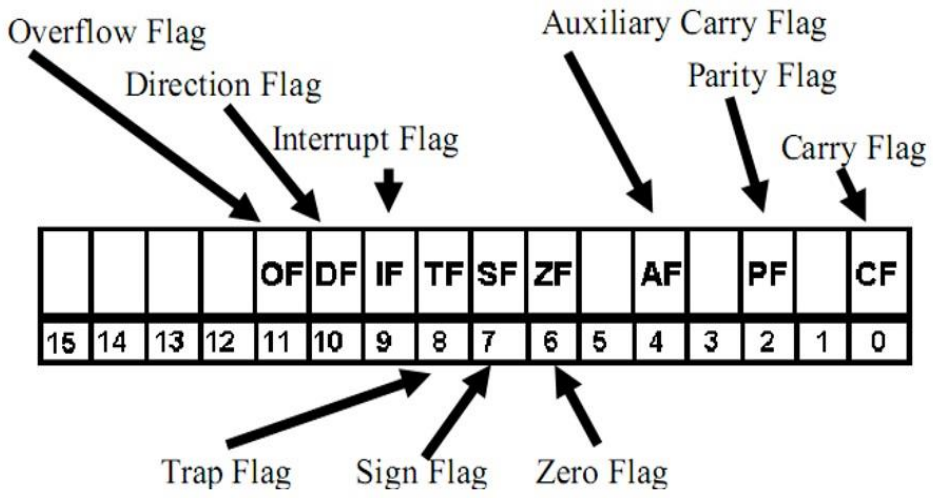
5. **Flag Register** determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. 8086 has 9 flags and they are divided into two categories:

1. **Status Flags**

Status Flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:

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

- **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 are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.



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 sit and can be cleared by executing CLI instruction.

❖ Direction Flag (DF):

- ➢ It is used in string operation.
- ➢ If it is set, string bytes are accessed from higher memory address to lower memory address.
- ➢ When it is reset, the string bytes are accessed from lower memory address to higher memory address.

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.

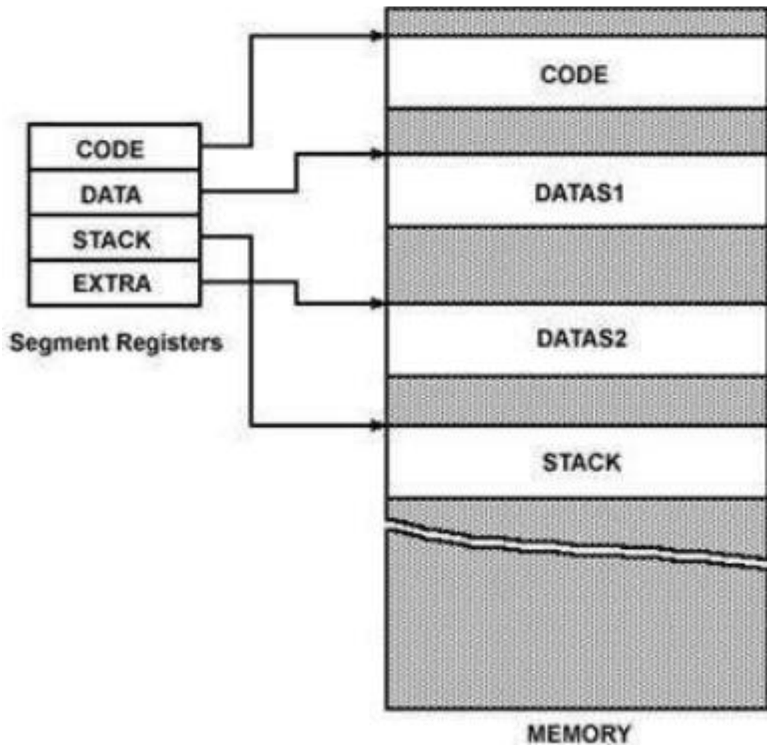


Fig. 3: Memory Segmentation

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.

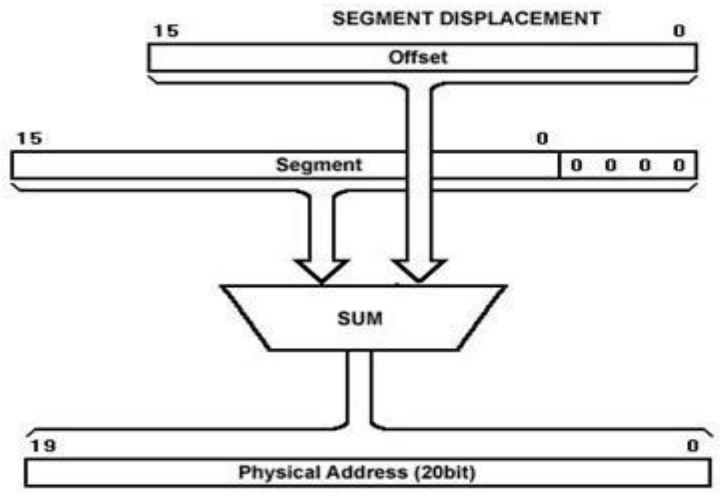


Fig. 4: Generating a physical address

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

1. Allows the memory capacity to be 1MB although the actual addresses to be handled are of 16-bit size.
2. Allows the placing of code, data and stack portions of the same program in different parts (segments) of memory, for data and code protection.

3. Permits a program and/or its data to be put into different areas of memory each time the program is executed, i.e., provision for relocation is done.

Addressing modes:

The different ways in which a source operand is denoted in an instruction are known as the addressing modes. There are 8 different addressing modes in 8086 programming. They are

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. Based, Indexed with displacement.

1. **Immediate addressing mode**: In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.

Example:

MOV AX, 0005H

In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

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 BX, AX

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 AX, [5000H]

Here, data resides in a memory location in the data segment, whose effective address may be completed using 5000H as the offset address and content of DS as segment address. The effective address here, is 10H * DS + 5000H.

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 register. The default segment is either DS or ES. 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, [BX]

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

5. **Based addressing mode**: In the based addressing mode, the effective address of the operand is obtained by adding a direct or

indirect displacement to the contents of either base register BX or base pointer register BP.

Example:

MOV [BX] + 1234H, AL;

EA=BX+1234H

PH=DS*10+EA

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.

Example:

MOV BX, [SI+06];

EA=SI+06

PH=DS*10+EA

7. **Based -index addressing mode*:*** The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example:

MOV AX, [BX][SI]

Here, BX is the base register and SI is the index register the effective address is computed as 10H * DS + [BX] + [SI].

8. **Based indexed with displacement mode:** The effective address is formed by adding an 8 or 16-bit displacement with the sum of the

contents of any one of the base register (BX or BP) and any one of the index register, in a default segment.

Example:

MOV AX, 50H [BX] [SI]

Here, 50H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as

10H * DS + [BX] + [SI] + 50H

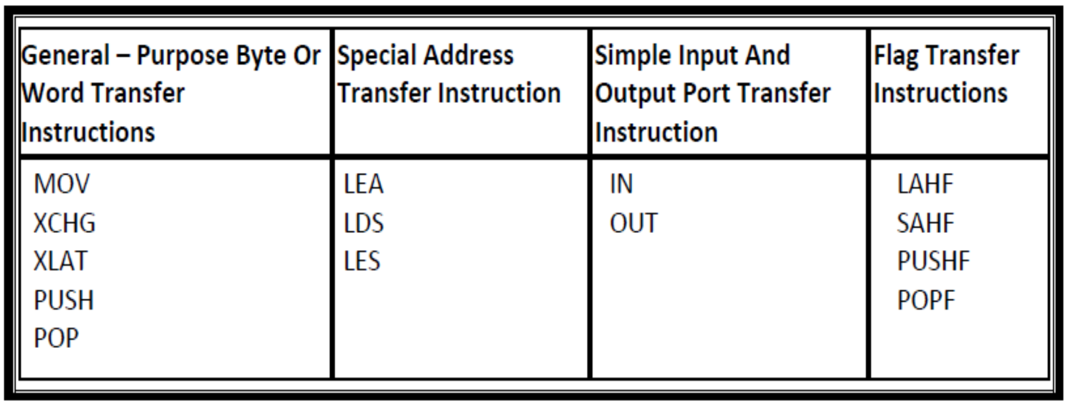
Instruction Set of 8086

The 8086 microprocessor supports 6 types of Instructions. They are

1. Data Transfer Instructions

The Data Transfer Instructions are those, which transfers the DATA from any one source to any one destination. The data's may be of any type. They are again classified into four groups as table 1:

Table 1: Data Transfer Instructions



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

- **MOV instruction**

It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.

General Form:

MOV destination, source

MOV BX, 00F2H ;	load the immediate number 00F2H in BX register
MOV CL, [2000H] ;	Copy the 8 bit content of the memory location, at a displacement of 2000H from data segment base to the CL register
MOV [589H], BX;	Copy the 16 bit content of BX register on to the memory location, which at a displacement of 589H from the data segment base.
MOV DS, CX ;	Move the content of CX to DS

- **XCHG instruction**

The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.

General Format

XCHG Destination, Source

XCHG BX, CX;	exchange word in CX with the word in BX
XCHG AL, CL;	exchange byte in CL with the byte in AL
XCHG AX, SUM[BX];	here physical address, which is DS+SUM+[BX]. The content at physical address and the content of AX are interchanged.

- **LEA Instruction – Load Effective Address**

General Form

LEA register, memory

This instruction indicates the offset of the variable or memory location named as source and put this offset in indicated 16-bit register.

Example:

LEA BX,m ;	Load BX with offset of m in DS
LEA BP,SS:STAK ;	Load BP with offset of Stack in SS
LEA CX, [BX][DI] ;	Load CX with EA=BX+DI

- **PUSH instruction**

The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be a general purpose register, segment register or a memory location.

The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2. Push instruction does not affect any flags.

Example:

PUSH CX;	Decrements SP by 2, copy content of CX to the stack
PUSH DS;	Decrement SP by 2 and copy DS to stack

- **POP instruction**

The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.

The execution pattern is similar to that of the PUSH instruction.

Example:

POP CX ; Copy a word from the top of the stack to CX and increment SP by 2.

- **IN & OUT instructions**

The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL and if 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.

Both IN and OUT instructions can be done using direct and indirect addressing modes.

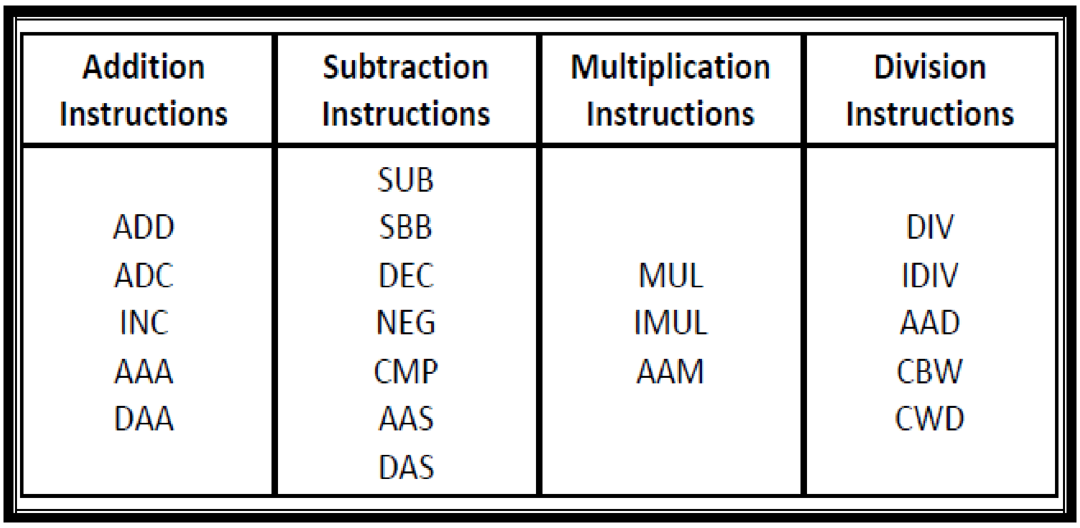
Example

IN AL, 0F8H;	Copy a byte from the port 0F8H to AL
MOV DX, 30F8H;	Copy port address in DX
IN AL, DX;	Move 8 bit data from 30F8H port
IN AX, DX;	Move 16 bit data from 30F8H port
OUT 047H, AL;	Copy contents of AL to 8 bit port 047H
MOV DX, 30F8H;	Copy port address in DX
OUT DX, AL;	Move 8 bit data to the 30F8H port
OUT DX, AX;	Move 16 bit data to the 30F8H port

2. Arithmetic Instructions

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

Table 2: Arithmetic Instructions



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

- **ADD instruction**

Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:

ADD Destination, Source

Example"

ADD AL, 0FH;	Add the immediate content, 0FH to the content of AL and store the result in AL
ADD AX, BX;	AX = AX+BX
ADD AX, 0100H;	Immediate
ADD AX, BX;	Register
ADD AX,[SI];	Register Indirect or Indexed
ADD AX, [5000H];	Direct
ADD [5000H], 0100H;	Immediate
ADD 0100H;	Destination AX (IMPLICT)

- **ADC: ADD with Carry**

This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculation) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:

Example:

ADC AX,BX;	Register
ADC AX,[SI];	Register Indirect or Indexed
ADC AX, [5000H];	Direct
ADC [5000H], 0100H;	Immediate
ADC 0100H;	Immediate AX (IMPLICT)

- **SUB instruction**

SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:

SUB Destination, Source

Example:

SUB AL, 0FH ;	subtract the immediate content, 0FH from the content of AL and store the result in AL
SUB AX, BX ;	AX = AX-BX
SUB AX,0100H ;	Immediate Destination AX
SUB AX,BX;	Register
SUB AX,[5000H];	Direct
SUB [5000H], 0100H;	Immediate

- **SBB: SUBTRACT with Borrow**

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

The result is stored in the destination operand. All the flags are affected (condition code) by this instruction. The examples of this instruction are as follows:

Example:

SBB AX, 0100H ;	Immediate Destination AX
SBB AX, BX ;	Register
SBB AX,[5000H] ;	Direct
SBB [5000H], 0100H ;	Immediate

- **CMP: Compare**

The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:

Example:

CMP BX,0100H ;	Immediate
CMP AX,0100H ;	Immediate
CMP [5000H] ;	Direct
CMP BX,[SI] ;	Register Indirect or Indexed
CMP BX, CX ;	Register

- **INC & DEC instructions**

INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.

Example:

INC AL;	AL= AL + 1
INC AX;	AX=AX + 1
DEC AL;	AL= AL – 1
DEC AX;	AX=AX – 1

- **MUL Instruction - Multiply unsigned bytes or words**

General Format

MUL OP

This instruction multiplies an unsigned multiplication of the accumulator by the operand specified by op. The type of op may be a register or memory operand,

When operand is a **byte:**

AX = AL * operand.

When operand is a **word:**

(DX AX) = AX * operand.

Example:

MUL BL;   AX=AL*CL

MUL CX;   (DX) (AX) = (AX)*(CX)

MUL [BX];   (AX) = (AL)*(DS) 0+ (BX))

- **IMUL Instruction - Multiply signed number**

General Format

IMUL op

This instruction performs a signed multiplication.

When operand is a byte:

AX = AL * operand.

When operand is a word: (DX AX) = AX * operand.

Example:

IMUL BH;   Signed byte in AL times multiplied by

; signed byte in BH and result in AX.

- **DIV Instruction - Unsigned divide**

General Format

Div op ; op (register or memory)

When operand is a byte:

AL=AX/operand

AH=remainder (modulus)

When operand is a word:

AX= (DX AX)/operand

DX=remainder (modulus)

If an attempt is made to divide by zero or quotient is too large to fit in AX (greater than **FFFFH)** the 8086 will do a type of 0 interrupt.

If you want to divide a byte by a byte, you must first put the dividend byte in **AL and fill** AH with all 0's. The SUB AH, AH instruction is a quick way to do. If you want to divide a word by a word, put the dividend word in AX and fill DX with all O's, The SUB DX, DX instruction does this quickly.

Example:

DIV CX ; (Quotient) AX= (DX: AX)/CX

;(Reminder) DX= (DX: AX) % CX

; AX = 37D7H = 14, 295 decimal and BH =97H = 151 decimal

DIV BH ; AX/ BH

; AX = Quotient = 5EH = 94 decimal and AH = Remainder = 65H = 101; decimal.

- **IDIV Instruction - Divide by signed byte or word**

General Format

IDIV op   ; op (reg or memory)

This instruction is used to divide a signed word by a signed byte or to divide a signed double word by a signed word.

WHEN operand is a byte:

AL=AX/operand

AH=remainder (modulus)

When operand is a word:

AX= (DX AX)/operand

DX=remainder (modulus)

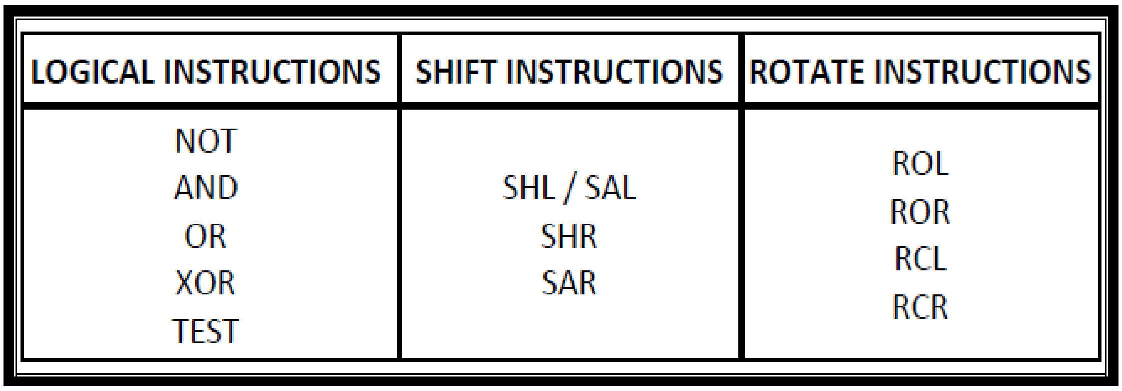
Example:

```
MOV AX,-203; AX=0FF35h

MOV BL, 4;

IDIV BL;    AL=-50(0CEh), AH=-3(0FDH)

RET
```

3. BIT Manipulation Instructions:

These instructions are used to perform Bit wise operations.

Table 3: BIT Manipulation Instructions



LOGICAL INSTRUCTIONS	SHIFT INSTRUCTIONS	ROTATE INSTRUCTIONS
NOT AND OR XOR TEST	SHL / SAL SHR SAR	ROL ROR RCL RCR

- **AND instruction**

This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:

AND Destination, Source

Example

AND BH, CL ;	AND byte in CL with byte in BH ;result in bH
AND BX,OOFFh ;	AND word in BX with immediate OOFFH. Mask upper byte, leave lower unchanged
AND CX, [SI] ;	AND word at offset [SI] in data segment with word in CX register. Result in CX register

- **OR Instruction**

This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:

OR Destination, Source

OR AH,CL ;	CL is OR'ed with AH, result in AH;
	CX=00111110 10100101

OR CX,FF00h ;	OR CX with immediate FF00h result in CX=11111111 10100101 upper byte are all 1's lower bytes are unchanged.

- **XOR Instruction**

The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:

```
XOR AX,0098H;
XOR AX,BX;
XOR AX,[5000H];
```

❖ Shift / Rotate Instructions

Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of

2^{+n} and division of powers of 2^{-n} .

There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.

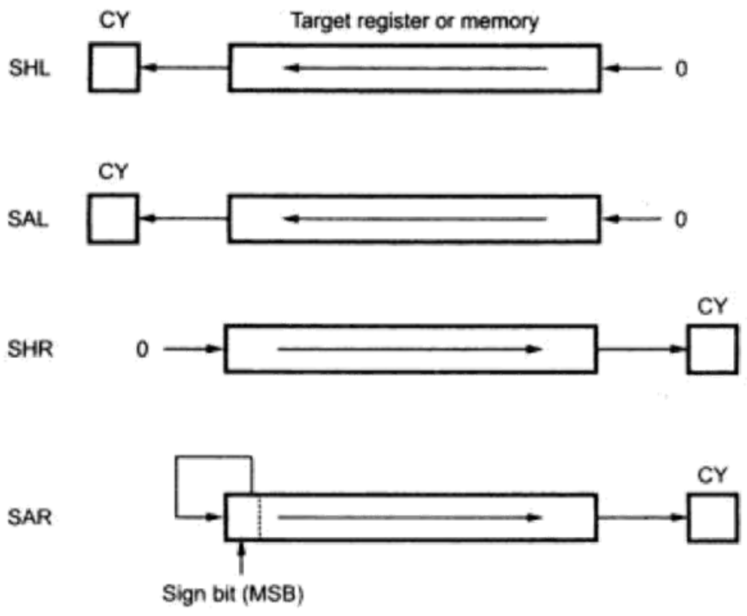


Fig. 5: Shift operations

Rotate on the other hand rotates the information in a register or memory either from one end to another or through the carry flag.

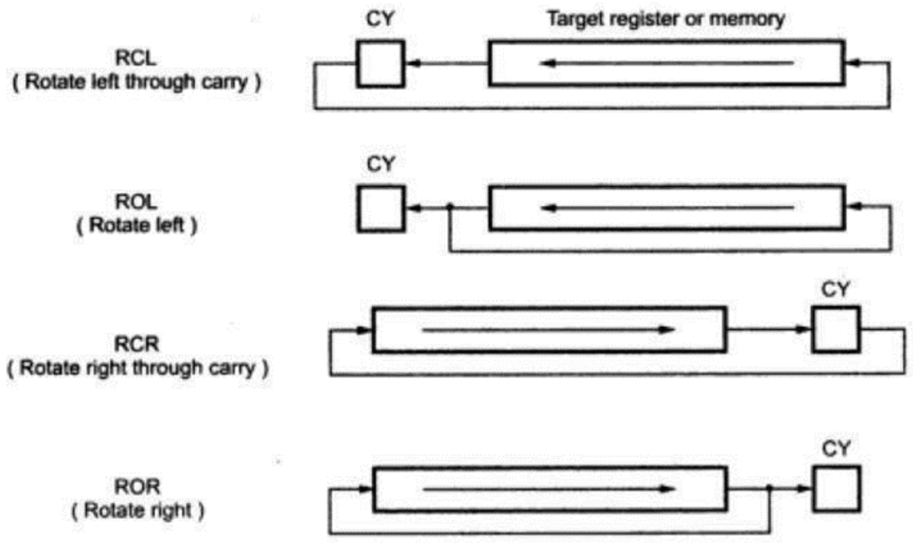

CY
Target register or memory
RCL
(Rotate left through carry)
CY
ROL
(Rotate left)
CY
RCR
(Rotate right through carry)
CY
ROR
(Rotate right)

Fig. 6: Rotate operations.

- **SHL/SAL instruction.**

Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected.

General Format:

SAL/SHL destination, count

Example:

MOV BL, B7H	;	BL is made B7H
SAL BL, 1	;	shift the content of BL register one place to left. Before

execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

After the execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
1	0	1	1	0	1	1	1	0

- **SHR instruction**

This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected.

General Format:

SHR destination, count

Example:

MOV BL, B7H ;	BL is made B7H
SHR BL, 1 ;	shift the content of BL register one place to the right.

- **ROL instruction**

This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format:

ROL destination, count

MOV BL, B7H	;	BL is made B7H
ROL BL, 1	; rotates the content of BL register one place to the left.	

Before execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

After the execution,

CY
B7 B6 B5 B4 B3 B2 B1 B0

1 0 1 1 0 1 1 1 1

- **ROR instruction**

This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected.

General Format:

ROR destination, count

Example:

MOV BL, B7H ; BL is made B7H

ROR BL, 1 ; shift the content of BL register one place to the right.

- **RCR instruction**

This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format:

RCR destination, count

Example:

MOV BL, B7H ; BL is made B7H

RCR BL, 1 ; shift the content of BL register one place to the right.

4. String Instructions

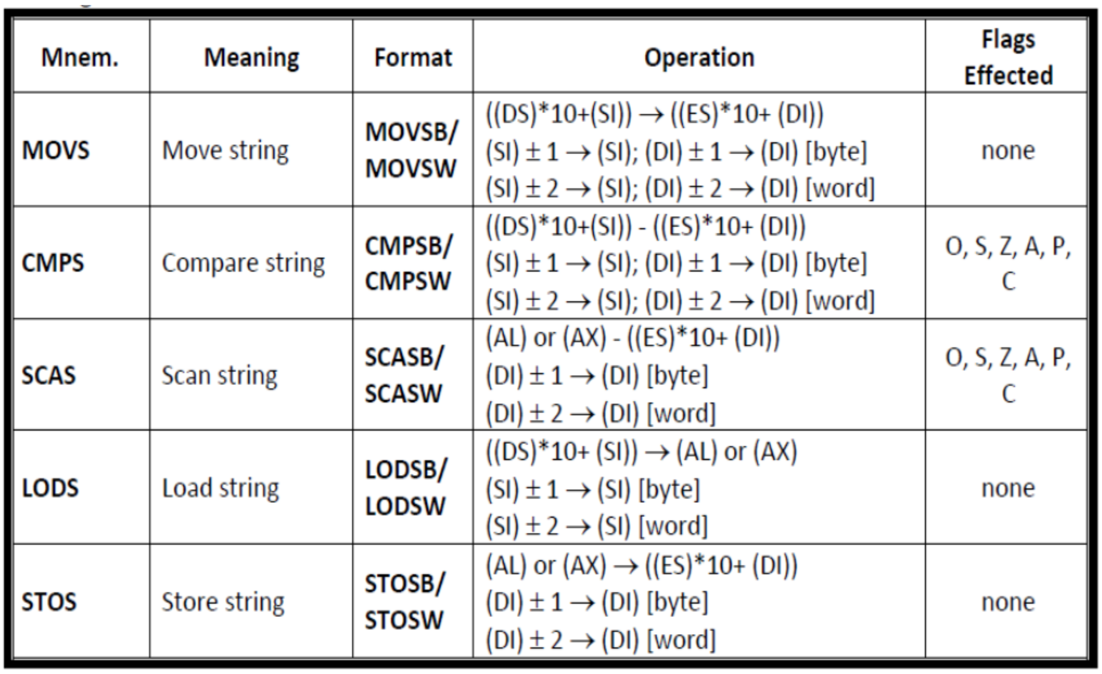
The string instructions function easily on blocks of memory. They are user friendly instructions, which help for easy program writing and execution. They can speed up the manipulating code. They are useful in array handling, tables and records. By using these string instructions, the size of the program is considerably reduced.

Five basic String Instructions define operations on one element of a string:

- Move byte or word string MOVSB/MOVSW
- Compare string CMPSB/CMPSW
- Scan string SCASB/SCASW
- Load string LODSB/LODSW
- Store string STOSB/STOSW

The general forms of these instructions are as shown below:

Table 4: string instructions



Mnem.	Meaning	Format	Operation	Flags Effected
MOVS	Move string	**MOVSB/ MOVSW**	((DS)*10+(SI)) → ((ES)*10+ (DI)) (SI) ± 1 → (SI); (DI) ± 1 → (DI) [byte] (SI) ± 2 → (SI); (DI) ± 2 → (DI) [word]	none
CMPS	Compare string	**CMPSB/ CMPSW**	((DS)*10+(SI)) - ((ES)*10+ (DI)) (SI) ± 1 → (SI); (DI) ± 1 → (DI) [byte] (SI) ± 2 → (SI); (DI) ± 2 → (DI) [word]	O, S, Z, A, P, C
SCAS	Scan string	**SCASB/ SCASW**	(AL) or (AX) - ((ES)*10+ (DI)) (DI) ± 1 → (DI) [byte] (DI) ± 2 → (DI) [word]	O, S, Z, A, P, C
LODS	Load string	**LODSB/ LODSW**	((DS)*10+ (SI)) → (AL) or (AX) (SI) ± 1 → (SI) [byte] (SI) ± 2 → (SI) [word]	none
STOS	Store string	**STOSB/ STOSW**	(AL) or (AX) → ((ES)*10+ (DI)) (DI) ± 1 → (DI) [byte] (DI) ± 2 → (DI) [word]	none

- **MOVS/MOVSB/MOVSW**

These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.

When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.

MOVS affect no flags. MOVSB is used for byte sized movements while MOVSW is for word sized.

MOVSB

Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.

Algorithm:

- ES :[DI] = DS : [SI]
- if DF = 0 then
- SI = SI + 1
- DI = DI + 1

Else

- SI = SI - 1
- DI = DI-1

Example:

```
ORG 100h
CLD
LEA SI , a1
LEA DI , a2
MOV CX , 5
REP MOVSB
RET


a1 DB 1 , 2 , 3 , 4 , 5
a2 DB 5 DUP( 0 )
```

MOVSW

Copy word at DS:[SI] to ES:[DI]. Update SI and DI.

Algorithm:

- ES :[DI] = DS : [SI]
- if DF = 0 then O SI = SI + 2 O DI = DI + 2

Else

- SI = SI – 2
- DI = DI – 2

Example:

```
ORG 100h CLD
LEA SI , a1
LEA DI , a2
MOV CX , 5
REP MOVSW
RET
a1 DW 1 , 2 , 3 , 4 , 5
a2 DW 5 DUP( 0 )
```

- **REP/REPE/REP2/REPNE/REPNZ**

REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

Example:

REP REPE/REPZ

- CX=0
- CX=0 OR ZF=0
- REPNE/REPNZ   CX=0 OR ZF=1

- **LODS/LODSB/LODSW**

This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX.LODS does not affect any flags. LODSB copies byte and LODSW copies word.

LODSB

Load byte at DS:[SI] into AL. Update SI.

Algorithm:

- AL = DS : [SI]
- if DF = 0 then
- SI = SI + 1

Else

- SI = SI – 1

Example:

```
ORG 100h

LEA SI , a1
MOV CX , 5
MOV AH , 0Eh

m : LODSB
INT 10h
LOOP m

RET

a1  DB  ' H ' , ' e ' , ' l ' ,   ' l ' ,   ' o '
```

LODSW

Load word at DS:[SI] into AX. Update SI. Algorithm:

AX = DS : [SI]

- if DF = 0 then
- SI = SI + 2

else

- SI = SI - 2

Example:

```
ORG 100h
LEA SI , a1
MOV CX , 5
REP LODSW       ; finally there will be 555h in  AX .
RET
a1 dw 111h , 222h , 333h , 444h , 555h
```

- **STOS/STOSB/STOSW**

The STOS instruction is used to store a byte/word contained in AL/AX to the offset contained in the DI register. STOS does not affect any flags. After copying the content DI is automatically incremented or decremented, based on the value of direction flag.

STOSB

Store byte in AL into ES:[DI]. Update DI.

Algorithm:

- ES :[DI] = AL
- if DF = 0 then
- DI = DI + 1

Else

- DI = DI - 1

Example:

```
ORG 100h
LEA DI , a1
MOV AL , 12h
MOV CX , 5
REP STOSB
RET
a1 DB 5 dup( 0 )
```

STOSW

Store word in AX into ES:[DI]. Update DI.

Algorithm:

- ES :[DI] = AX
- if DF = 0 then
- DI = DI + 2

Else

- DI = DI - 2

Example:

```
ORG 100h

LEA DI , a1
MOV AX , 1234h
MOV CX , 5
REP STOSW
RET
a1 DW 5 dup( 0 )
```

- **CMPS/CMPSB/CMPSW**

CMPS is used to compare the strings, byte wise or word wise. The comparison is affected by subtraction of content pointed by DI from that pointed by SI. The AF, CF, OF, PF, SF and ZF flags are affected by this instruction, but neither operand is affected.

CMPSB

Compare bytes: ES:[DI] from DS:[SI].

Algorithm:

- DS :[SI] - ES : [DI]
- set flags according to result : OF , SF , ZF , AF , PF , CF
- if DF = 0 then
- SI = SI + 1
- DI = DI + 1

Else

- SI = SI - 1
- DI = DI - 1

CMPSW

- DS :[SI] - ES : [DI]
- set flags according to result : OF , SF , ZF , AF , PF , CF
- if DF = 0 then
- SI = SI + 2
- DI = DI + 2

Else

- SI = SI - 2
- DI = DI – 2

- **Scan string SCASB/SCASW**

SCASB

Compare bytes: AL from ES:[DI].

Algorithm:

- AL - ES : [DI]
- set flags according to result : OF , SF , ZF , AF , PF , CF
- if DF = 0 then
- DI = DI + 1

Else

- DI = DI – 1

SCASW

Compare bytes: AX from ES:[DI].

Algorithm:

- AL - ES : [DI]
- set flags according to result : OF , SF , ZF , AF , PF , CF
- if DF = 0 then
- DI = DI + 2

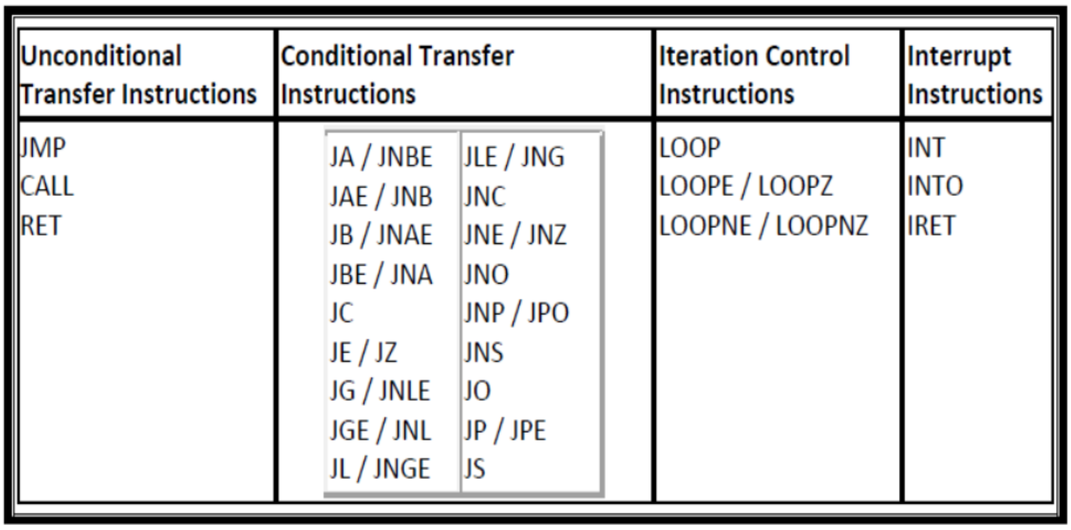
Else

- DI = DI – 2

5. Control Transfer Instructions

These instructions transfer the program control from one address to other address. (Not in a sequence). They are again classified into four groups. They are:

Table 5: Control Transfer Instructions



Unconditional Transfer Instructions	Conditional Transfer Instructions		Iteration Control Instructions	Interrupt Instructions
JMP	JA / JNBE	JLE / JNG	LOOP	INT
CALL	JAE / JNB	JNC	LOOPE / LOOPZ	INTO
RET	JB / JNAE	JNE / JNZ	LOOPNE / LOOPNZ	IRET
	JBE / JNA	JNO		
	JC	JNP / JPO		
	JE / JZ	JNS		
	JG / JNLE	JO		
	JGE / JNL	JP / JPE		
	JL / JNGE	JS		

- **JUMP Instruction**

The purpose of a jump instruction is to alter the execution path of instructions in the program. The code segment register and instruction pointer keep track of the next instruction to be executed. Thus a jump instruction involves altering the contents of these registers. In this way, execution continues at an address other than that of the next sequential instruction. That is, a jump occurs to another part of the program.

There two type of jump instructions:

a) Unconditional jump.
b) Conditional jump.

In an unconditional jump, no status requirements are imposed for the jump to occur. That is, as the instruction is executed, the jump always takes place to change the execution sequence.

Instruction	**Meaning**	**Format**	**Operation**	**Flags affected**
JMP	Unconditional jump	JMP operand	Jump is to the address specified by operand	non

On the other hand, for a conditional jump instruction, status conditions that exist at the moment the jump instruction is executed decide whether or not the jump will occur. If this condition or conditions are met, the jump takes place, otherwise execution continues with the next sequential instruction of the program. The conditions that can be referenced by a conditional jump instruction are status flags such as carry (CF), parity (PF), and overflow (OF).

Instruction	**Meaning**	**Format**	**Operation**	**Flags affected**
JCC	Conditional jump	JCC operand	if the specific condition cc is true, the jump to the address specified by the operand is initiated, otherwise the next instruction is executed	non

6. Process Control Instructions

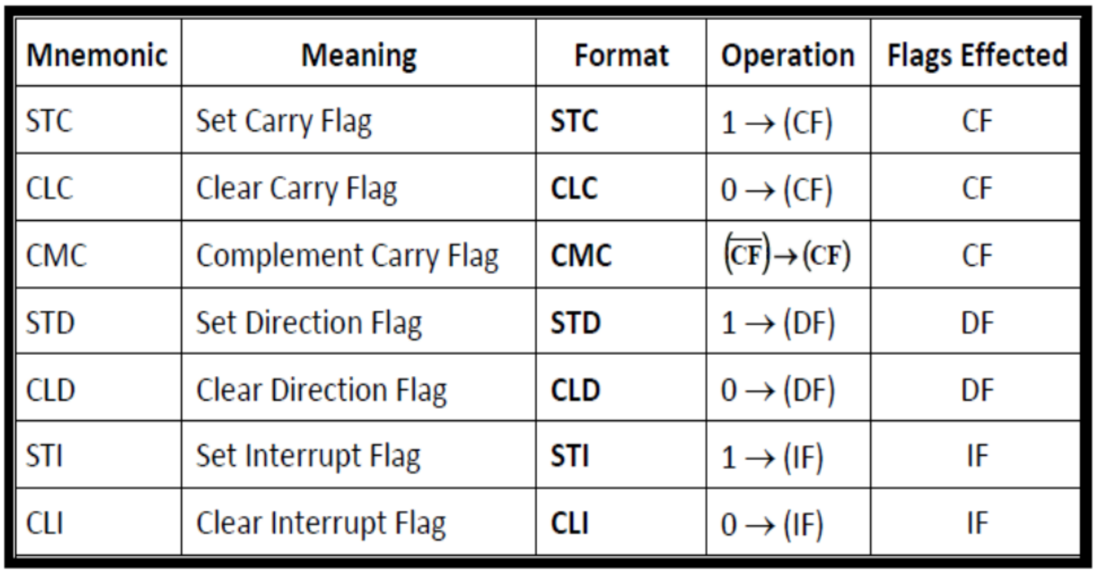
These instructions are used to change the process of the Microprocessor. They change the process with the stored information. They are again classified into two groups. They are:

1. Flag Control Instructions
2. External Hardware Synchronization Instructions

1. Flag Control Instructions:

These instructions directly affected the state of flags. Figure below shows these instructions.

Table 6: **Flag Control Instructions**



Mnemonic	Meaning	Format	Operation	Flags Effected
STC	Set Carry Flag	**STC**	$1 \rightarrow (CF)$	CF
CLC	Clear Carry Flag	**CLC**	$0 \rightarrow (CF)$	CF
CMC	Complement Carry Flag	**CMC**	$(\overline{CF}) \rightarrow (CF)$	CF
STD	Set Direction Flag	**STD**	$1 \rightarrow (DF)$	DF
CLD	Clear Direction Flag	**CLD**	$0 \rightarrow (DF)$	DF
STI	Set Interrupt Flag	**STI**	$1 \rightarrow (IF)$	IF
CLI	Clear Interrupt Flag	**CLI**	$0 \rightarrow (IF)$	IF

2. **Flag manipulation instructions**

1. **STC instruction**

This instruction sets the carry flag. It does not affect any other flag.

2. CLC instruction

This instruction resets the carry flag to zero. CLC does not affect any other flag.

3. CMC instruction

This instruction complements the carry flag. CMC does not affect any other flag.

4. STD instruction

This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instruction. STD does not affect any other flag.

5. CLD instruction

This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.

6. STI instruction

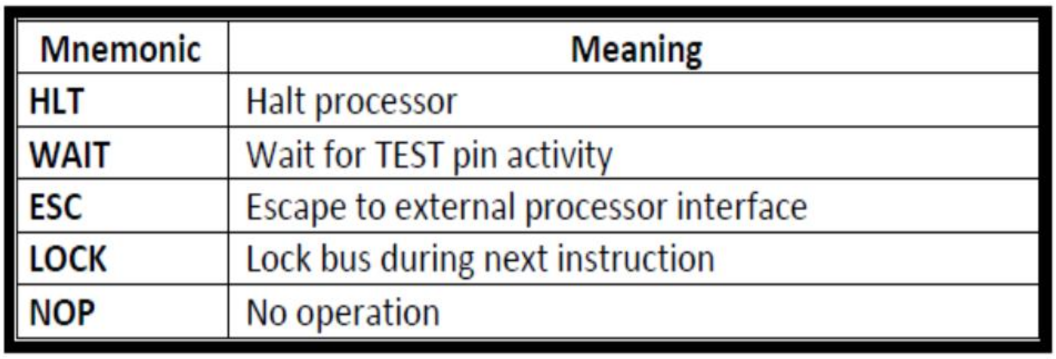
This instruction sets the interrupt flag to 1. This enables INTR interrupt of the 8086. STI does not affect any other flag.

7. CLI instruction

This instruction resets the interrupt flag to 0. Due to this the 8086 will not respond to an interrupt signal on its INTR input. CLI does not affect any other flag.

2. External Hardware Synchronization Instructions:

Table: External Hardware Synchronization Instructions

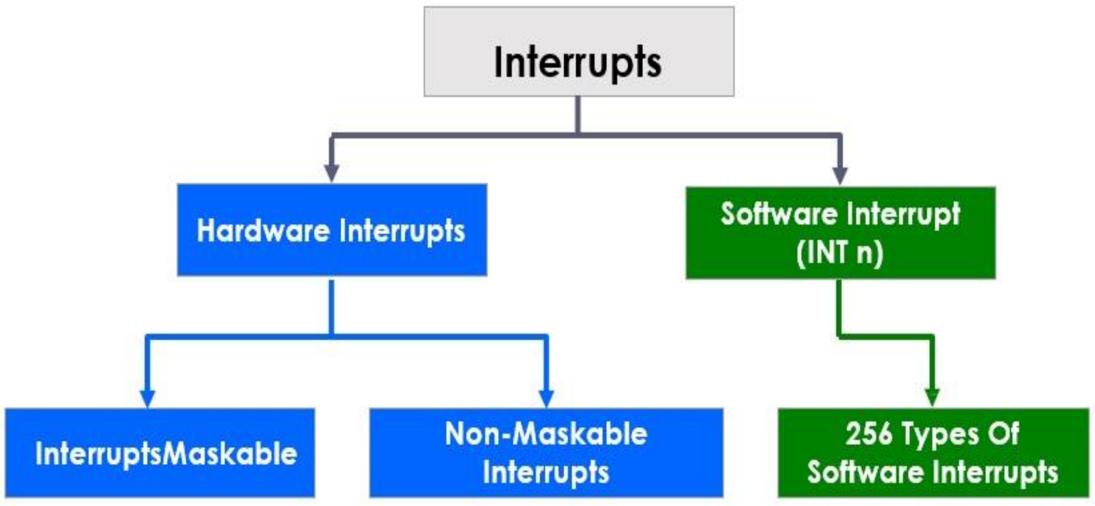


Mnemonic	Meaning
HLT	Halt processor
WAIT	Wait for TEST pin activity
ESC	Escape to external processor interface
LOCK	Lock bus during next instruction
NOP	No operation

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.

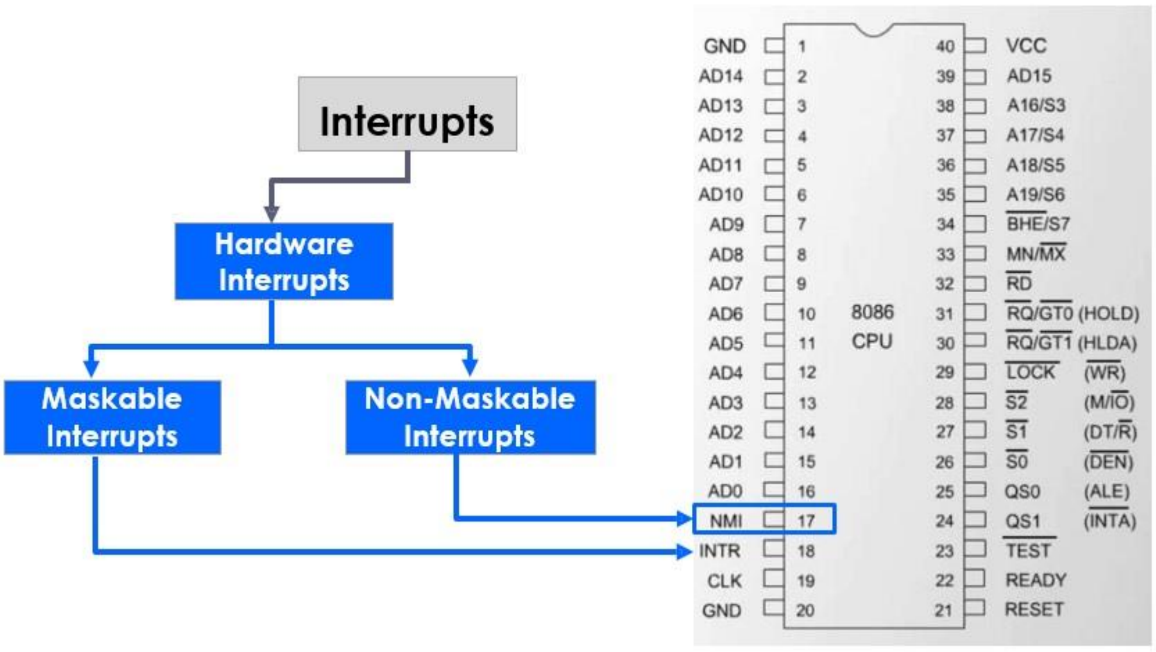
Whenever an interrupt occurs the processor completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR is a program that tells the processor what to do when the interrupt occurs. After the execution of ISR, control returns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

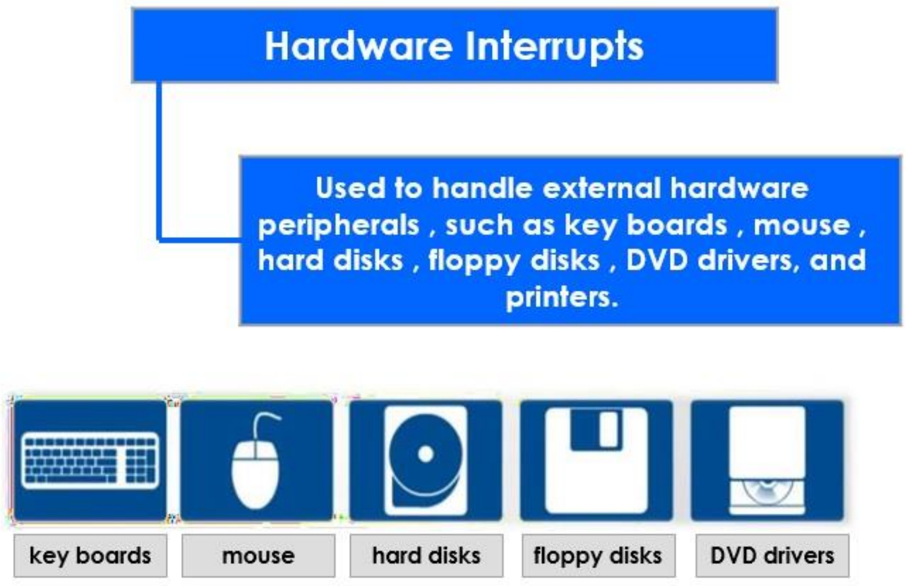


Broadly the interrupts are divided into two types. They are external (hardware) Interrupts and internal (Software) Interrupts. The hardware interrupts are classified as non-maskable and maskable interrupts. The hardware interrupt is caused by any peripheral device by sending a signal through a specified 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. The interrupts initiated by applying appropriate signal to these pins are called hardware interrupts of 8086.




Hardware Interrupts
Used to handle external hardware peripherals , such as key boards , mouse , hard disks , floppy disks , DVD drivers, and printers.
key boards
mouse
hard disks
floppy disks
DVD drivers

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:

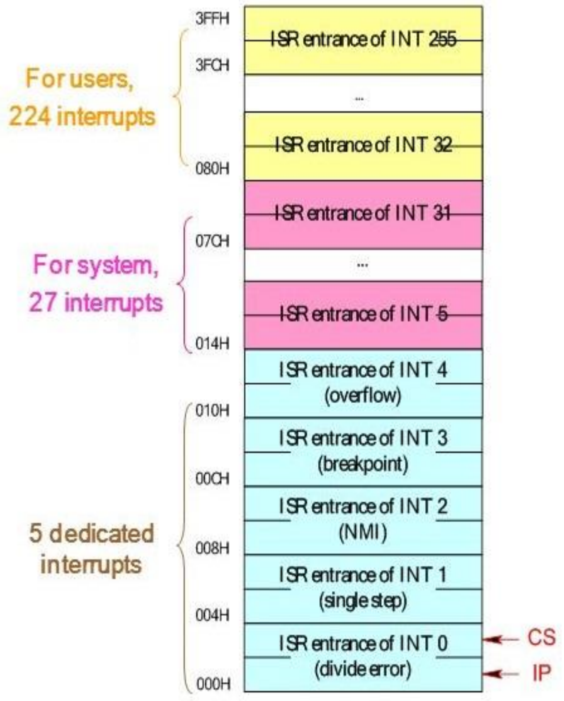
- TYPE 0 interrupt represents division by zero situation.
- TYPE 1 interrupt represents single-step execution during the debugging of a program.
- TYPE 2 interrupt represents non-maskable NMI interrupt.
- TYPE 3 interrupt represents break-point interrupt.
- TYPE 4 interrupt represents overflow interrupt.

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.

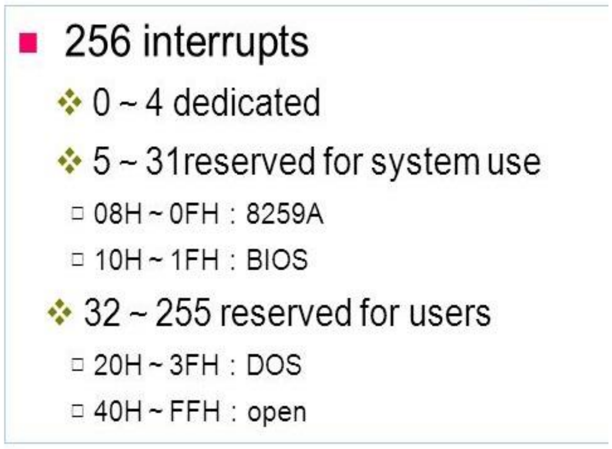
Interrupt vector table

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 as a lookup or jump table to memory address of bios interrupt service routine (f000h to ffffh) or dos interrupt service routine address, 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.



3FFH
3FCH
ISR entrance of INT 255
...
ISR entrance of INT 32
080H
ISR entrance of INT 31
07CH
...
ISR entrance of INT 5
014H
ISR entrance of INT 4 (overflow)
010H
ISR entrance of INT 3 (breakpoint)
00CH
ISR entrance of INT 2 (NMI)
008H
ISR entrance of INT 1 (single step)
004H
ISR entrance of INT 0 (divide error)
000H
CS
IP
For users, 224 interrupts
For system, 27 interrupts
5 dedicated interrupts



- 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

In 8086 microprocessor following tasks are performed when microprocessor encounters an interrupt:

1. The value of flag register is pushed into the stack. It means that first the value of SP (Stack Pointer) is decremented by 2 then the value of flag register is pushed to the memory address of stack segment .
2. The value of starting memory address of CS (Code Segment) is pushed into the stack .
3. The value of IP (Instruction Pointer) is pushed into the stack .
4. IP is loaded from word location (Interrupt type) * 04 .
5. CS is loaded from the next word location .
6. Interrupt and Trap flag are reset to 0.