

# **THADOMAL SHAHANI ENGINEERING COLLEGE**

Advocate Nari Gursahani Marg, 37th Road, (Off Linking Road), TPS III,  
Bandra(West), Mumbai- 400 050, India

**Department Of Information Technology**

**LAB MANUAL**

**MICROPROCESSOR LAB**

Semester-IV

**Subject Teachers:**

Ms. Reshma Malik, Ms. Sheetal Gondal

## INTRODUCTION TO 8086

8086 Microprocessor was designed by Intel in 1976. It was the first 16-bit processor having 16-bit ALU, 16-bit registers and 16-bit data bus resulting in faster processing. It is available in 3 versions based on the frequency of operation

8086 → 5MHz

8086 → 8MHz

8086 → 10 MHz

It is a 16-bit Microprocessor having 20 address lines and 16 data lines. The size of 8086 Microprocessor is up to 1MB storage.

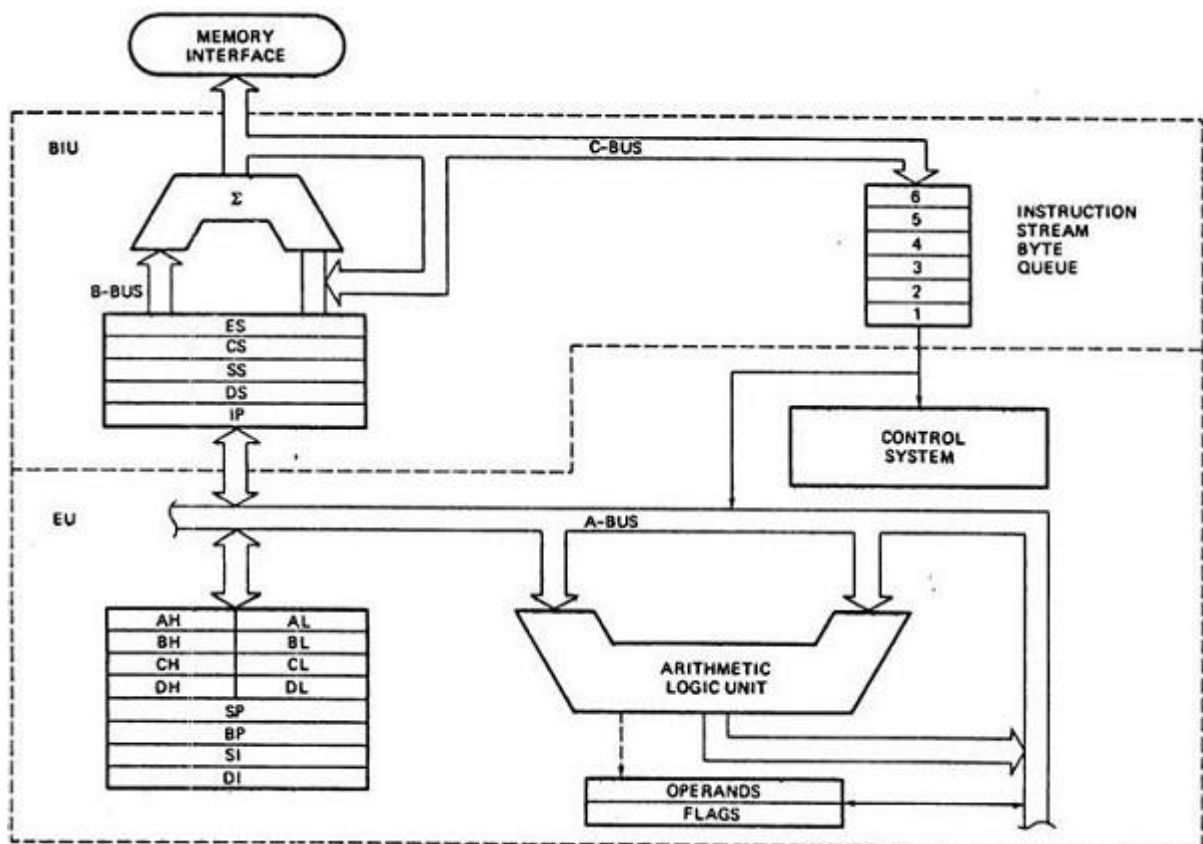
It supports two modes of operation, i.e. Minimum mode and Maximum mode.

Minimum mode is suitable for system having a single processor and Maximum mode is suitable for system having multiple processors.

It has powerful instruction set.

## Architecture of 8086

The following diagram depicts the architecture of 8086 Microprocessor



8086 Microprocessor is divided into two functional units, i.e. **BIU** (Bus Interface Unit) and **EU** (Execution Unit).

## BIU (Bus Interface Unit)

BIU takes care of all data and addresses transfers on the buses for the EU such as fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU and BIU are connected with the Internal Bus.

BIU has divided into following section:

- **Instruction queue** – BIU has instruction queue of 6 bytes. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.
- Fetching the next instruction while the current instruction executes is called **pipelining**.
- **Segment register** – BIU has 4 segment registers of 16 bits i.e. CS, DS, SS & ES. It holds the starting address of the corresponding segments. It also contains pointer register IP, which holds the address of the next instruction to execute by the EU.
  - **CS** – It stands for Code Segment register.
  - **DS** – It stands for Data Segment register.
  - **SS** – It stands for Stack Segment register.
  - **ES** – It stands for Extra Segment register.
- **Instruction pointer** – It is a 16-bit register used to hold the address of the next instruction to be executed.
- 8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip.

## EU (Execution Unit)

Execution unit gets the instruction from instruction queue then decode it with the help of control unit and execute those instructions with the help of ALU.

### ALU

It handles all arithmetic and logical operations, like +, -, ×, /, OR, AND, NOT operations.

### Flag Register

It is a 16-bit register. It has 9 flags and they are divided into 2 groups – Conditional Flags and Control Flags.

### Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags –

- **Carry flag** – This flag indicates an overflow condition for arithmetic operations.
- **Auxiliary flag** – When an operation is performed at ALU, it results in a carry/borrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.

- **Parity flag** – This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
- **Zero flag** – This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** – This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** – This flag represents the result when the system capacity is exceeded.

### Control Flags

Control flags controls the operations of the execution unit.

- **Trap flag** – It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.
- **Interrupt flag** – It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
- **Direction flag** – It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

### General purpose register

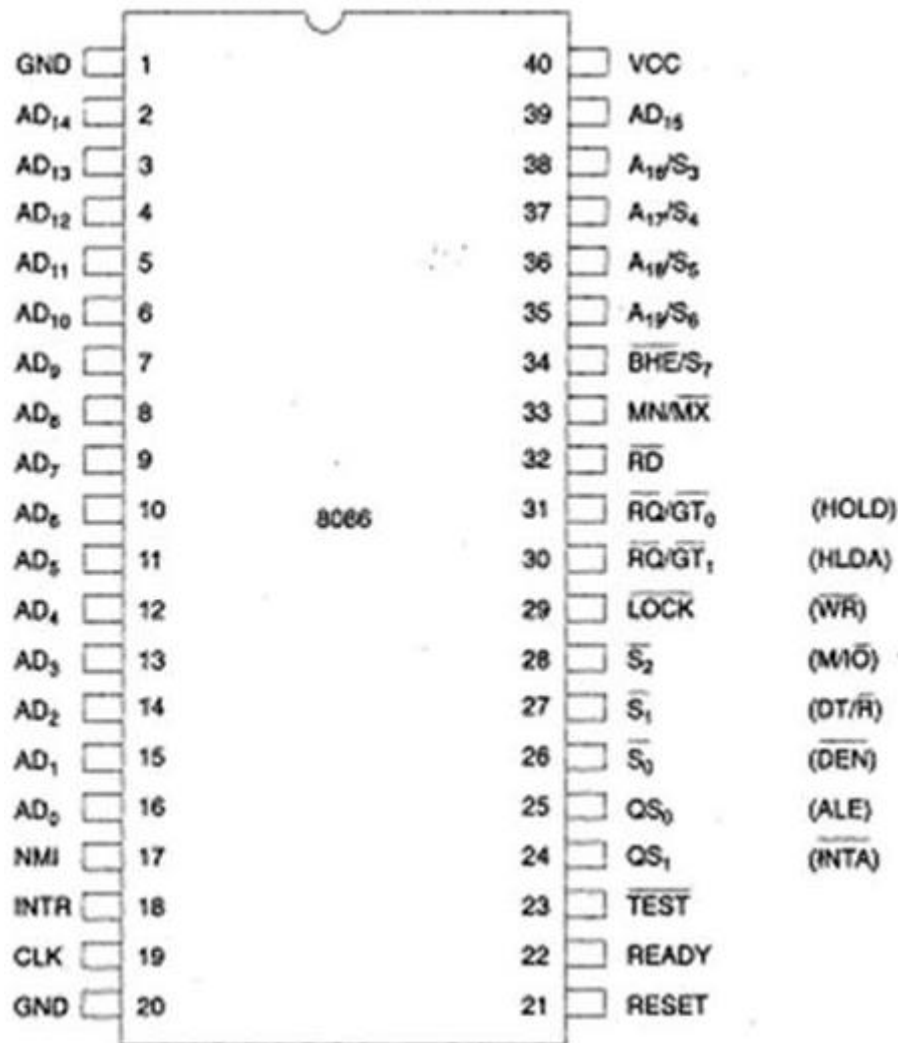
There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register** – It is also known as accumulator register. It is used to store operands for arithmetic operations.
- **BX register** – It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
- **CX register** – It is referred to as counter. It is used in loop instruction to store the loop counter.
- **DX register** – This register is used to hold I/O port address for I/O instruction.

### Stack pointer register

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

## 8086 Pin Diagram



### Power supply and frequency signals

- It uses 5V DC supply at V<sub>CC</sub> pin 40, and uses ground at V<sub>SS</sub> pin 1 and 20 for its operation.

### Clock signal

- Clock signal is provided through Pin-19. It provides timing to the processor for operations. Its frequency is different for different versions, i.e. 5MHz, 8MHz and 10MHz.

### Address/data bus

- AD<sub>0</sub>-AD<sub>15</sub>. These are 16 address/data bus.
- AD<sub>0</sub>-AD<sub>7</sub> carries low order byte data and AD<sub>8</sub>-AD<sub>15</sub> carries higher order byte data. During the first clock cycle, it carries 16-bit address and after that it carries 16-bit data.

### Address/status bus

- A<sub>16</sub>-A<sub>19</sub>/S<sub>3</sub>-S<sub>6</sub>. These are the 4 address/status buses.

- During the first clock cycle, it carries 4-bit address and later it carries status signals.

#### **S7/BHE**

- BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

#### **Read**

- It is available at pin 32 and is used to read signal for Read operation.

#### **Ready**

- It is available at pin 22. It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

#### **RESET**

- It is available at pin 21 and is used to restart the execution. It causes the processor to immediately terminate its present activity. This signal is active high for the first 4 clock cycles to RESET the microprocessor.

#### **INTR**

- It is available at pin 18. It is an interrupt request signal, which is sampled during the last clock cycle of each instruction to determine if the processor considered this as an interrupt or not.

#### **NMI**

- It stands for non-maskable interrupt and is available at pin 17. It is an edge triggered input, which causes an interrupt request to the microprocessor.

#### **TEST**

- This signal is like wait state and is available at pin 23. When this signal is high, then the processor has to wait for IDLE state, else the execution continues.

#### **MN/MX**

- It stands for Minimum/Maximum and is available at pin 33. It indicates what mode the processor is to operate in; when it is high, it works in the minimum mode and vice-a versa.

#### **INTA**

- It is an interrupt acknowledgement signal and is available at pin 24. When the microprocessor receives this signal, it acknowledges the interrupt.

#### **ALE**

- It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

#### **DEN**

- It stands for Data Enable and is available at pin 26. It is used to enable Trans receiver 8286. The trans receiver is a device used to separate data from the address/data bus.

### **DT/R**

- It stands for Data Transmit/Receive signal and is available at pin 27. It decides the direction of data flow through the transreceiver. When it is high, data is transmitted out and vice-a-versa.

### **M/IO**

- This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicates the memory operation. It is available at pin 28.

### **WR**

- It stands for write signal and is available at pin 29. It is used to write the data into the memory or the output device depending on the status of M/IO signal.

### **HLDA**

- It stands for Hold Acknowledgement signal and is available at pin 30. This signal acknowledges the HOLD signal.

### **HOLD**

- This signal indicates to the processor that external devices are requesting to access the address/data buses. It is available at pin 31.

### **QS<sub>1</sub> and QS<sub>0</sub>**

- These are queue status signals and are available at pin 24 and 25. These signals provide the status of instruction queue. Their conditions are shown in the following table –

<b>QS<sub>0</sub></b>	<b>QS<sub>1</sub></b>	<b>Status</b>
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

### **S<sub>0</sub>, S<sub>1</sub>, S<sub>2</sub>**

- These are the status signals that provide the status of operation, which is used by the Bus Controller 8288 to generate memory & I/O control signals. These are available at pin 26, 27, and 28. Following is the table showing their status –

S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Status
0	0	0	Interrupt acknowledgement
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

### **LOCK**

- When this signal is active, it indicates to the other processors not to ask the CPU to leave the system bus. It is activated using the LOCK prefix on any instruction and is available at pin 29.

### **RQ/GT<sub>1</sub> and RQ/GT<sub>0</sub>**

- These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment. RQ/GT<sub>0</sub> has a higher priority than RQ/GT<sub>1</sub>.

**The 8086 microprocessor supports following types of instructions –**

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions



### Data Transfer Instruction

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

## Arithmetic Instructions

Instruction	Description
ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	<b>Decimal Adjust after BCD Addition:</b> When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.
DAS	<b>Decimal Adjust after BCD Subtraction:</b> When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD.
AAA	<b>ASCII Adjust for Addition:</b> When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD.
AAD	<b>Adjust AX Register for Division:</b> It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte.
AAM	<b>Adjust result of BCD Multiplication:</b> This instruction is used after the multiplication of two unpacked BCD.

AAS	<b>ASCII Adjust for Subtraction:</b> This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number.
CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory location(s).

### Logical Instructions

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

### Rotate Instructions

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

ROR	Rotate all bits of the operand right by specified number of bits.
-----	-------------------------------------------------------------------

### Shift Instructions

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

### Branch Instructions

Instruction	Description
JA or JNBE	Jump if above, not below, or equal i.e. when CF and ZF = 0
JAE/JNB/JNC	Jump if above, not below, equal or no carry i.e. when CF = 0
JB/JNAE/JC	Jump if below, not above, equal or carry i.e. when CF = 0
JBE/JNA	Jump if below, not above, or equal i.e. when CF and ZF = 1
JCXZ	Jump if CX register = 0
JE/JZ	Jump if zero or equal i.e. when ZF = 1
JG/JNLE	Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF
JGE/JNL	Jump if greater, not less or equal i.e. when SF = OF
JL/JNGE	Jump if less, not greater than or equal i.e. when SF ≠ OF
JLE/JNG	Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF
JMP	Causes the program execution to jump unconditionally to

	the memory address or label given in the instruction.
CALL	Calls a procedure whose address is given in the instruction and saves their return address to the stack.
RET	Returns program execution from a procedure (subroutine) to the next instruction or main program.
IRET	Returns program execution from an interrupt service procedure (subroutine) to the main program.
INT	Used to generate software interrupt at the desired point in a program.
INTO	Software interrupts to indicate overflow after arithmetic operation.
LOOP	Jump to defined label until CX = 0.
LOOPZ/LOOPE	Decrement CX register and jump if CX $\neq$ 0 and ZF = 1.
LOOPNZ/LOOPNE	Decrement CX register and jump if CX $\neq$ 0 and ZF = 0.

### Flag Manipulation and Processor Control Instructions

Instruction	Description
CLC	<b>Clear Carry Flag:</b> This instruction resets the carry flag CF to 0.
CLD	<b>Clear Direction Flag:</b> This instruction resets the direction flag DF to 0.
CLI	<b>Clear Interrupt Flag:</b> This instruction resets the interrupt flag IF to 0.
CMC	This instruction take complement of carry flag CF.
STC	Set carry flag CF to 1.
STD	Set direction flag to 1.
STI	Set interrupt flag IF to 1.
HLT	Halt processing. It stops program execution.

NOP	Performs no operation.
ESC	<b>Escape:</b> makes bus free for external master like a coprocessor or peripheral device.
WAIT	When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing.
LOCK	It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction.

### String Instructions

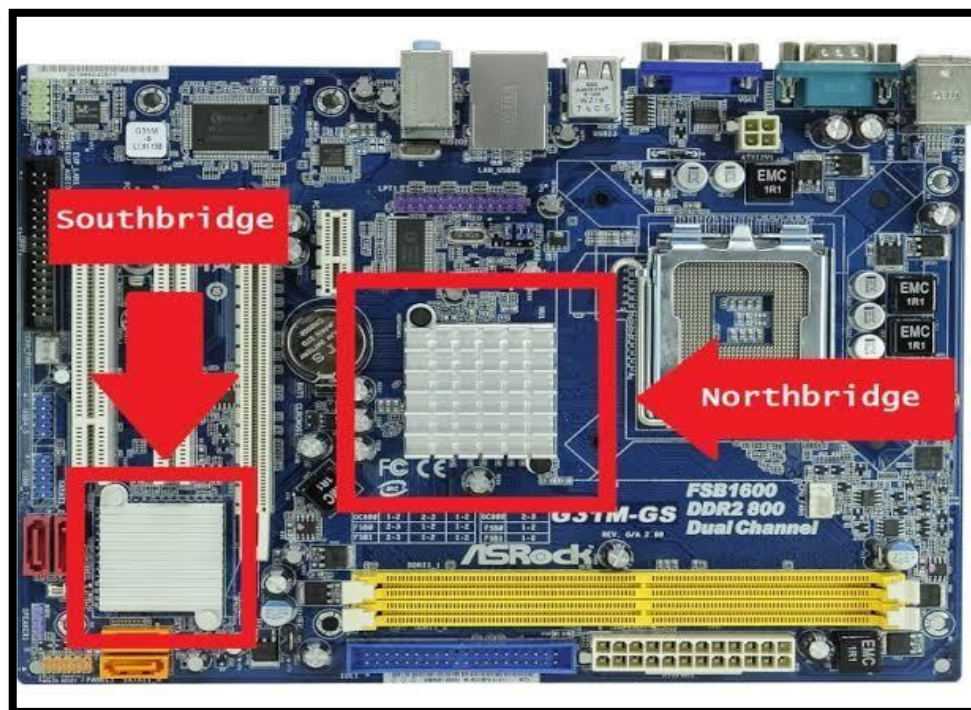
Instruction	Description
MOVS/MOVSb/MOVSW	Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register.
CMPS/CMPSb/CMPSW	Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register.
SCAS/SCASb/SCASW	Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES.
LODS/LODSb/LODSW	Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register.
STOS/STOSb/STOSW	Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register.
REP	Repeats the given instruction until CX $\neq$ 0
REPE/ REPZ	Repeats the given instruction till CX $\neq$ 0 and ZF = 1
REPNE/REPNZ	Repeats the given instruction till CX $\neq$ 0 and ZF = 0

## Module 1: PC Assembly

### Motherboard:

A motherboard (also called mainboard, main circuit board, system board, baseboard, planar board, logic board) is the main printed circuit board (PCB) in general-purpose computers and other expandable systems. It holds and allows communication between central processing unit (CPU) and memory, and provides connectors for other peripherals.

Motherboard means specifically a PCB with expansion capabilities. As the name suggests, this board is often referred to as the “mother” of all components attached to it, which often include peripherals, interface cards, and daughter cards: sound cards, video cards, network cards, host bus adapters, TV tuner cards, IEEE 1394 cards: and a variety of other custom components.



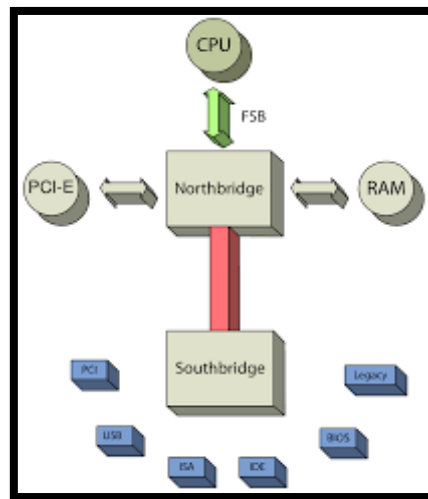
**North Bridge:** Northbridge is an integrated circuit responsible for communications between the CPU interface, AGP, and the memory. The north bridge is directly connected. It acts as a "bridge" for the south bridge chip to communicate with the CPU, RAM, and graphics controller. Today, the north bridge is a single-chip that is north of the PCI bus, however, early computers may have had up to three separate chips that made up the north bridge.

**South Bridge:** The south bridge is an IC on the motherboard responsible for the hard drive controller, I/O controller and integrated hardware. Integrated hardware can include the sound card and video card if on the motherboard, USB, PCI, ISA, IDE, BIOS, and Ethernet.

The south bridge typically implements the slower capabilities of the motherboard in a north bridge/south bridge chipset computer architecture. In systems with Intel chipsets, the south bridge

is named I/O Controller Hub (ICH), while AMD has named its south bridge Fusion Controller Hub (FCH) since the introduction of its Fusion AMD Accelerated Processing Unit (APU) while moving the functions of the Northbridge onto the CPU die, hence making it similar in function to the Platform hub controller.

The south bridge can usually be distinguished from the north bridge by not being directly connected to the CPU. Rather, the north bridge ties the south bridge to the CPU. Through the use of controller integrated channel circuitry, the north bridge can directly link signals from the I/O units to the CPU for data control and access.



**Ports:** A port is a physical docking point using which an external device can be connected to the computer. It can also be a programmatic docking point through which information flows from a program to the computer or over the Internet.

Ports are of two types –

- **Internal port** – It connects the motherboard to internal devices like hard disk drive, CD drive, internal modem, etc.
- **External port** – It connects the motherboard to external devices like modem, mouse, printer, flash drives, etc.

### Serial Port

- Used for external modems and older computer mouse
- Two versions: 9 pin, 25 pin model
- Data travels at 115 kilobits per second

### Parallel Port

- Used for scanners and printers
- Also called printerport
- 25 pin model



- IEEE 1284-compliant Centronics port



### Universal Serial Bus (or USB) Port

- It can connect all kinds of external USB devices such as external hard disk, printer, scanner, mouse, keyboard, etc.
- It was introduced in 1997.
- Most of the computers provide two USB ports as minimum.
- Data travels at 12 megabits per seconds.
- USB compliant devices can get power from a USB port.



### Firewire Port

- Transfers large amount of data at very fast speed.
- Connects camcorders and video equipment to the computer.
- Data travels at 400 to 800 megabits per seconds.
- Invented by Apple.
- It has three variants: 4-Pin FireWire 400 connector, 6-Pin FireWire 400 connector, and 9-Pin FireWire 800 connector.



## Keyboard Port

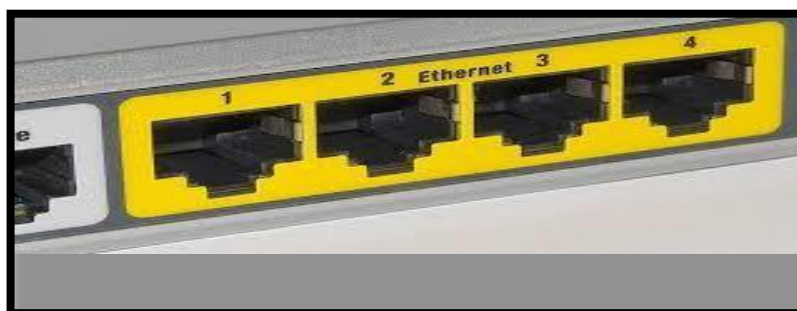
The PS/2 port is a 6-pin mini-DIN connector used for connecting keyboards and mouse to a PC compatible computer system. Its name comes from the IBM Personal System/2 series of personal computers, with which it was introduced in 1987.



## Ethernet Port

This type of ports provides an interface to connect to peripheral devices using a serial protocol. In this port, the rate of transmission of data is one bit at a time through a single communication line. For example, RS-232 signals.

## VGA Port



VGA stands for Video Graphics Array. It is a 3 row with a 15-pin DE-15 connector. It is used in many monitors, laptops, video cards, projectors, etc. Sometimes, this port was used on laptops otherwise other portable devices in place of the full-size VGA connector. The current LCD as well as LED monitors support VGA ports however the quality of the picture can be reduced. This port carries

analog video signals up to 648X480.resolution. Some laptops are inbuilt with VGA ports to unite exteriormonitors otherwise projectors.



## Module 2:

### 2.1 Verify the truth table of various logic gates.

#### LOGIC GATES

Digital electronic circuits operate with voltages of **two logic levels** namely Logic Low and Logic High

The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**. Hence, the Logic gates are the building blocks of any digital system. Logic gates are classified into the following three categories.

- Basic gates
- Universal gates
- Special gates

#### Basic Gates

Boolean functions can be represented either in sum of products form or in product of sums form based on the requirement. So it is implement these Boolean functions by using basic gates. The basic gates are AND, OR & NOT gates.

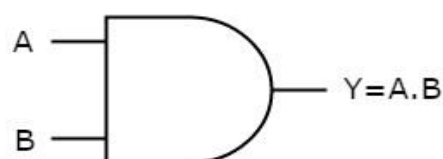
#### AND gate

An AND gate is a digital circuit that has two or more inputs and produces an output, which is the **logical AND** of all those inputs. It is optional to represent the **Logical AND** with the symbol '·'.

The following table shows the **truth table** of 2-input AND gate.

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

The following figure shows the **symbol** of an AND gate, which is having two inputs A, B and one output, Y.



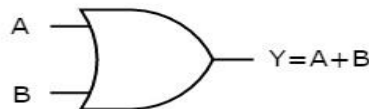
## OR gate

An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. This **logical OR** is represented with the symbol '+'.

The following table shows the **truth table** of 2-input OR gate.

A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

The following figure shows the **symbol** of an OR gate, which is having two inputs A, B and one output, Y.



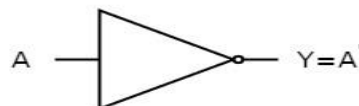
## NOT gate

A NOT gate is a digital circuit that has single input and single output. The output of NOT gate is the **logical inversion** of input. Hence, the NOT gate is also called as inverter.

The following table shows the **truth table** of NOT gate.

A	$Y = A'$
0	1
1	0

The following figure shows the **symbol** of NOT gate, which is having one input, A and one output, Y.



## Universal gates

NAND & NOR gates are called as **universal gates**. Because we can implement any Boolean function, which is in sum of products form by using NAND gates alone. Similarly, we can implement any Boolean function, which is in product of sums form by using NOR gates alone.

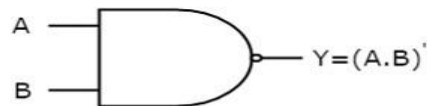
## NAND gate

NAND gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical AND** of all those inputs.

The following table shows the **truth table** of 2-input NAND gate.

A	B	$Y = A.B.A.B'$
0	0	1
0	1	1
1	0	1
1	1	0

The following image shows the **symbol** of NAND gate, which is having two inputs A, B and one output, Y.



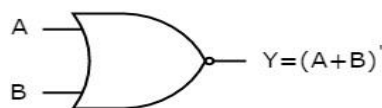
## NOR gate

NOR gate is a digital circuit that has two or more inputs and produces an output, which is the **inversion of logical OR** of all those inputs.

The following table shows the **truth table** of 2-input NOR gate

A	B	$Y = A+BA+B'$
0	0	1
0	1	0
1	0	0
1	1	0

The following figure shows the **symbol** of NOR gate, which is having two inputs A, B and one output, Y.



## Special Gates

Ex-OR & Ex-NOR gates are called as special gates. Because, these two gates are special cases of OR & NOR gates.

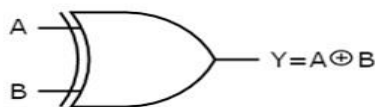
### Ex-OR gate

The full form of Ex-OR gate is **Exclusive-OR** gate. Its function is same as that of OR gate except for some cases, when the inputs having even number of ones.

The following table shows the **truth table** of 2-input Ex-OR gate.

A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

**Symbol** of Ex-OR gate, which is having two inputs A, B and one output, Y.



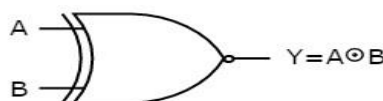
### Ex-NOR gate

The full form of Ex-NOR gate is **Exclusive-NOR** gate. Its function is same as that of NOR gate except for some cases, when the inputs having even number of ones.

The following table shows the **truth table** of 2-input Ex-NOR gate.

A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

**Symbol** of Ex-NOR gate, which is having two inputs A, B and one output, Y.



## 2.2 Realize Half adder, Full adder, Half subtractor and Full subtractor.

- Output depends upon the combination of inputs.
- Output is pure function of present inputs only i.e., Previous State inputs won't have any effect on the output. Also, It doesn't use memory.
- In other words,
  - $OUTPUT=f(INPUT)$
- Inputs are called Excitation from circuits and outputs are called Response of combinational logic circuits.

### Classification of Combinational Logic Circuits:

#### 1. Arithmetic:

- Adders
- Subtractors
- Multipliers
- Comparators

#### 2. Data Handling:

- Multiplexers
- DeMultiplexers
- Encoders and Decoders

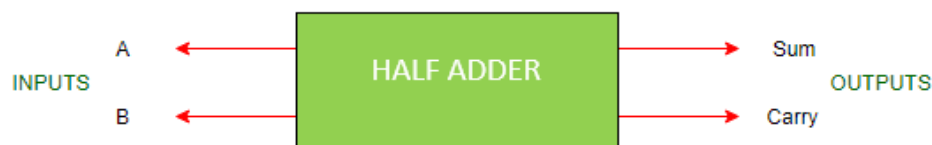
#### 3. Code Converters:

- BCD to Excess-3 code and vice versa
- BCD to Gray code and vice versa
- Seven Segment

### Design of Half Adders and Full Adders:

- A combinational logic circuit that performs the addition of two single bits is called Half Adder.
- A combinational logic circuit that performs the addition of three single bits is called Full Adder.

#### 1. Half Adder:



- It is an arithmetic combinational logic circuit designed to perform addition of two single bits.
- It contains two inputs and produces two outputs.
- Outputs are called Sum and Carry.

Let us observe the **addition of single bits**,

$$0+0=0$$

$$0+1=1$$



$$1+0=1$$

$$1+1=10$$

Since  $1+1=10$ , the result must be two bit output. So, above can be rewritten as,

$$0+0=00$$

$$0+1=01$$

$$1+0=01$$

$$1+1=10$$

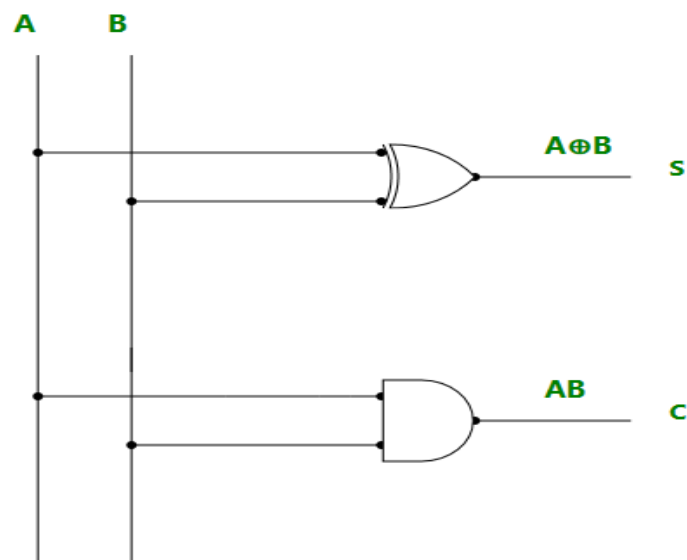
The result of  $1+1$  is 10, where '1' is carry-output ( $C_{out}$ ) and '0' is Sum-output (Normal Output).

### Truth Table of Half Adder:

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Next Step is to draw the Logic Diagram. To draw Logic Diagram, we need Boolean Expression, which can be  $S = AB' + A'B$  which can be logically written as,

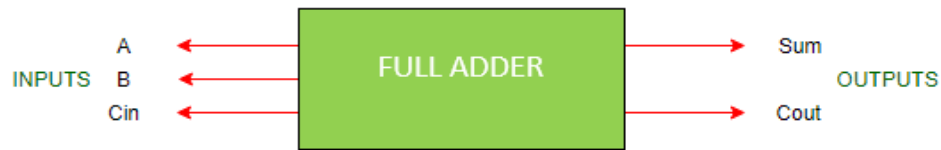
$$S = A \text{ xor } B \text{ and } C = AB$$



### Limitations:

Adding of Carry is not possible in Half adder.

## 2. Full Adder:



- To overcome the above limitation faced with Half adders, Full Adders are implemented.
- It is an arithmetic combinational logic circuit that performs addition of three single bits.
- It contains three inputs (A, B,  $C_{in}$ ) and produces two outputs (Sum and  $C_{out}$ ).
- Where,  $C_{in} \rightarrow$  Carry In and  $C_{out} \rightarrow$  Carry Out

### Truth table of Full Adder:

Inputs			Outputs	
A	B	$C_{in}$	Sum	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The equation obtained is,

$$S = A'B'C_{in} + AB'C_{in}' + ABC + A'BC_{in}'$$

The equation can be simplified as,

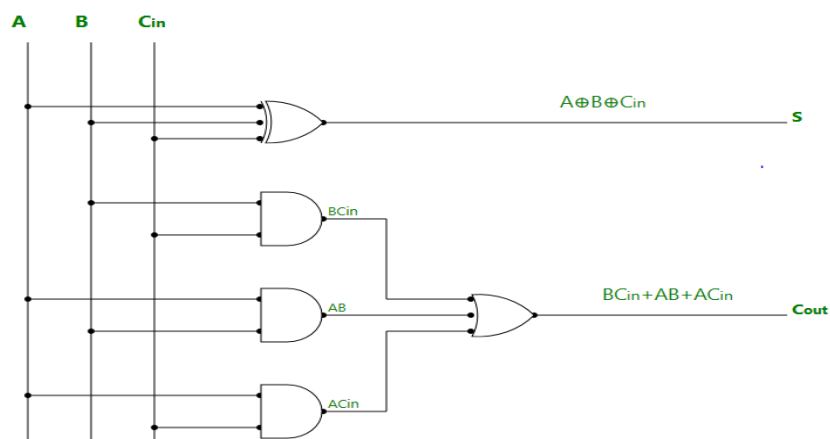
$$S = B'(A'C_{in} + AC_{in}') + B(AC + A'C_{in}')$$

$$S = B'(A \text{ xor } C_{in}) + B(A \text{ xor } C_{in})'$$

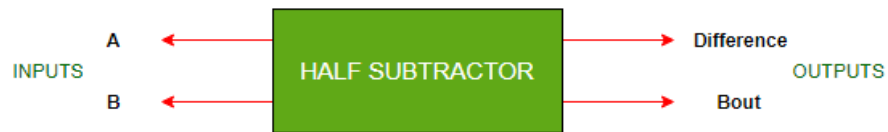
$$S = A \text{ xor } B \text{ xor } C_{in}$$

$$C_{out} = BC_{in} + AB + AC_{in}$$

### Logic Diagram of Full Adder:



### 3. Half Subtractor:



- It is a combinational logic circuit designed to perform subtraction of two single bits.
- It contains two inputs (A and B) and produces two outputs (Difference and Borrow-output).

#### Truth Table of Half Subtractor:

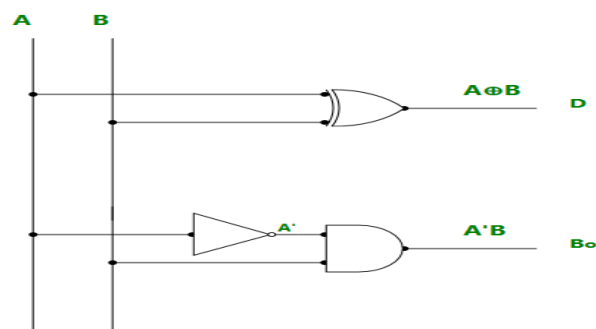
Inputs		Outputs	
A	B	D	B <sub>o</sub>
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The equation is,  $D = A'B + AB'$  which can be logically written as,

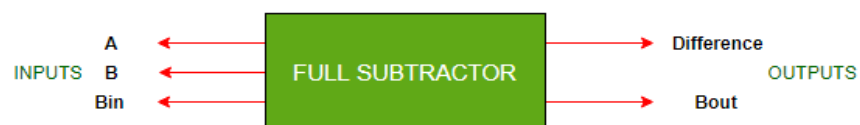
$$D = A \text{ xor } B$$

$$B_{\text{out}} = A'B$$

#### Logic Diagram of Half Subtractor:



### 4. Full Subtractor:



- It is a Combinational logic circuit designed to perform subtraction of three single bits.

- It contains three inputs(A, B, B<sub>in</sub>) and produces two outputs (D, B<sub>out</sub>).
- Where, A and B are called **Minuend** and **Subtrahend** bits.
- And, B<sub>in</sub> -> Borrow-In and B<sub>out</sub> -> Borrow-Out

### Truth Table of Full Subtractor:

Inputs			Outputs	
A	B	B <sub>in</sub>	D	B <sub>out</sub>
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The equation is,

$D = A'B'B_{in} + AB'B_{in}' + ABB_{in} + A'BB_{in}'$  which can be simplified as,

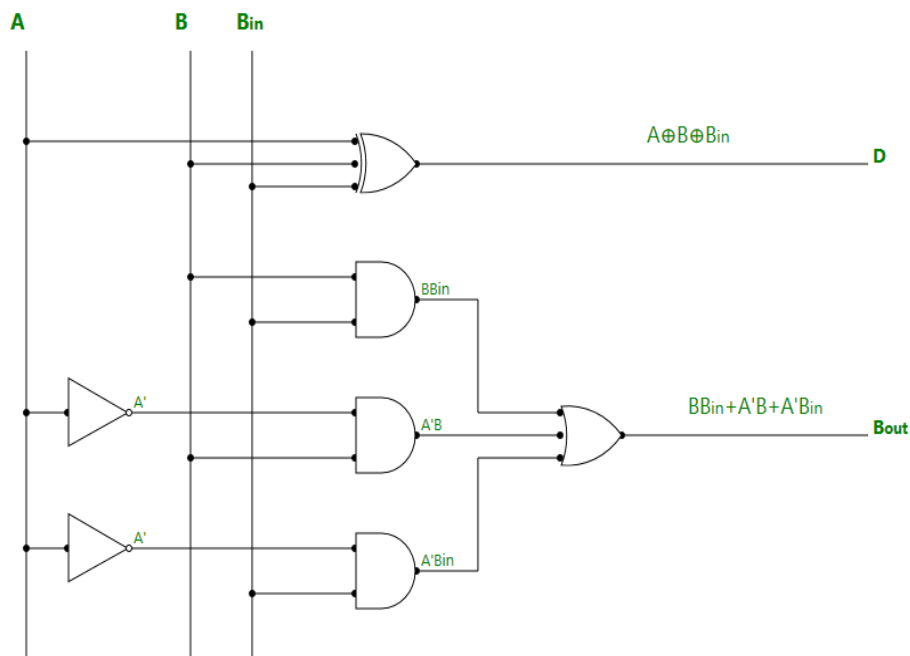
$D = B'(A'B_{in} + AB_{in}') + B(AB_{in} + A'B_{in}')$

$D = B'(A \text{ xor } B_{in}) + B(A \text{ xor } B_{in})'$

$D = A \text{ xor } B \text{ xor } B_{in}$

$B_{out} = BB_{in} + A'B + A'B_{in}$

### Logic Diagram of Full Subtractor:



### Applications:

1. For performing arithmetic calculations in electronic calculators and other digital devices.
2. In Timers and Program Counters.
3. Useful in Digital Signal Processing.

## 2.3 Implementation of Multiplexer and Demultiplexer

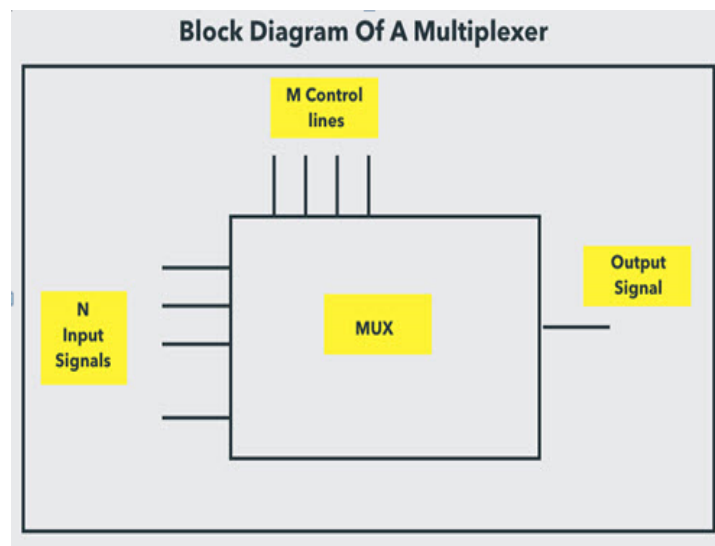
A **Multiplexer** is a circuit that accept many inputs but gives only one output.

A **Demultiplexer** functions exactly in the reverse way of a multiplexer i.e., a demultiplexer accepts only one input and gives many outputs.

Multiplexers can handle two type of data i.e., analog and digital. For analog application, multiplexer are built using relays and transistor switches. For digital application, they are built from standard logic gates.

The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many input but only one output. By applying control signals (also known as Select Signals), we can steer any input to the output. Some of the common types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

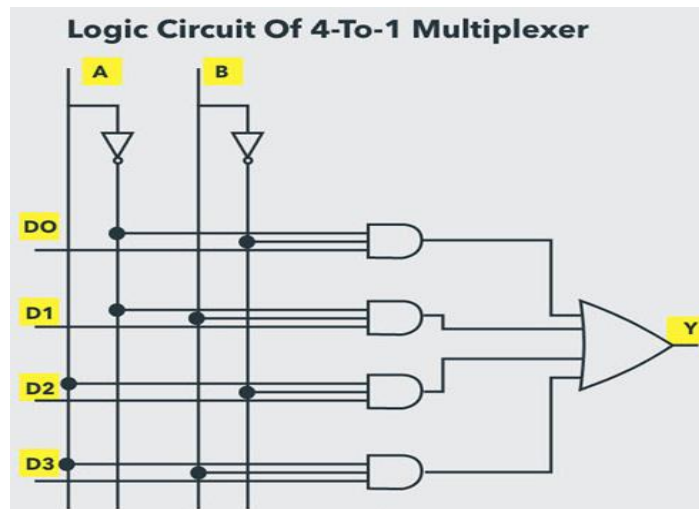
Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.



### 4-to-1 Multiplexer

The 4-to-1 multiplexer has 4 input bits, 2 control or select bits, and 1 output bit. The four input bits are D0, D1, D2 and D3. Only one of this is transmitted to the output Y. The output depends on the values of A and B, which are the control inputs. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in figure, when **A B = 0 0**, the upper AND gate is enabled, while all other AND gates are disabled. Therefore, data bit D0 is transmitted to the output, giving  $Y = D_0$ .



If the control input is changed to  $A B = 1 1$ , all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and  $Y = D3$ .

- An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.
- Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.
- Example of 16-to-1 line multiplexer is IC 74150.

### Applications of Multiplexer

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers –

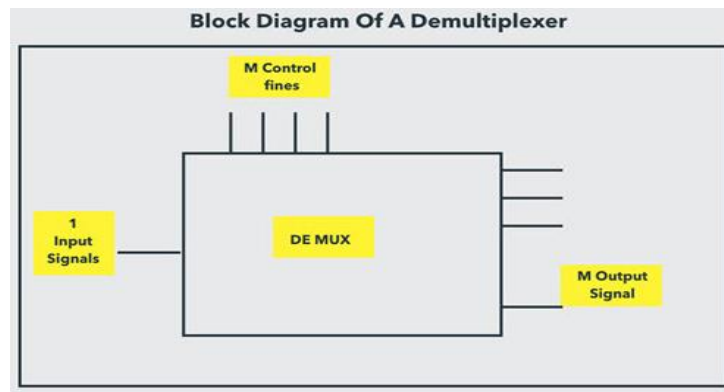
1. **Communication System** – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.
2. **Telephone Network** – In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
3. **Computer Memory** – Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.
4. **Transmission from the Computer System of a Satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

This is just an introduction to the concept of Multiplexer. To learn more about Multiplexers, read this [Multiplexer \(MUX\) and Multiplexing tutorial](#).

## Demultiplexer

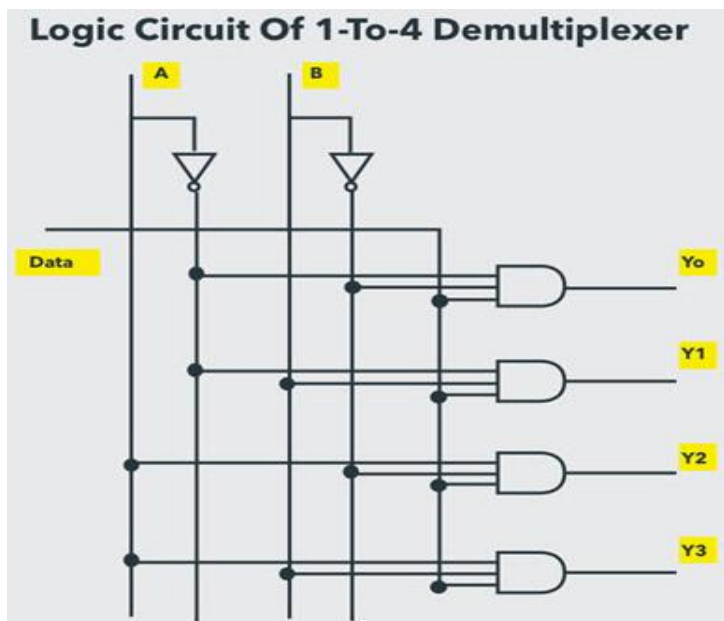
Demultiplexer means one to many. A demultiplexer is a circuit with one input and many outputs. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to-2, 1-to-4, 1-to-8 and 1-to-16 demultiplexer.

Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



### 1-to-4 Demultiplexer

The 1-to-4 demultiplexer has 1 input bit, 2 control or select bits, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-



The input bit is labelled as Data D. This data bit is transmitted to the selected output lines, which depends on the values of A and B, the control or Select Inputs.

When  $A B = 0 1$ , the second AND gate from the top is enabled while other AND gates are disabled. Therefore, data bit D is transmitted to the output Y1, giving  $Y1 = \text{Data}$ .

If D is LOW, Y1 is LOW. If D is HIGH, Y1 is HIGH. The value of Y1 depends upon the value of D. All other outputs are in low state.

If the control input is changed to  $A B = 1 0$ , all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and  $Y2 = \text{Data}$ .

Example of 1-to-16 demultiplexer is IC 74154. It has 1 input bit, 4 control / select bits and 16 output bit.

### Applications of Demultiplexer

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system, where multiplexers are used. Most of the communication system are bidirectional i.e., they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.
2. **Communication System** – Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.
3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. **Serial to Parallel Converter** – A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.



## Module 3:

### 3.1 Program for 16 bit BCD addition

Consider that two words are available in registers AX and BX. We have to add these two words.

- Using add instruction, add the contents, of the lower two bits i.e. add AL and BL.
- The result of this addition is stored in the AL register.
- DAA instruction is then used to convert the result to valid BCD. Now, add the contents of MSB along with carry if generated in LSB addition.
- The result of MSB addition is stored in the AH register. Adjust this result to valid BCD number. The final result is available in the BX register.
- The DAA instruction operates only on the AL register and so here, we have to add LSB and MSB separately without using ADD AX, BX.

eg. : AX = 3629 BCD

BX = 4738 BCD

**Step I :**

$$\begin{array}{r} 29 \\ + 38 \\ \hline 61 \end{array} \text{ and auxiliary carry} \\ = 1$$

As AC = 1, add 6 to make result valid.

$$61 + 6 = 67$$

**Step II :**  $36 + 47 + 0$  (Carry of LSB) = 7 D.

Lower nibble of addition is greater than 9, so add 6.

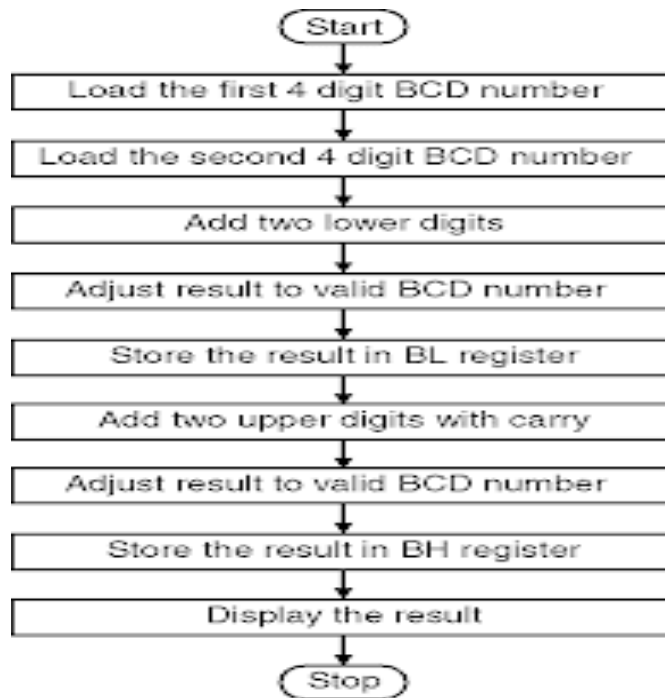
$$\begin{array}{r} 7D \\ + 06 \\ \hline 83 \end{array}$$

Final result : 8367 BCD.

#### Algorithm :

- Step I** : Initialize the data memory.
- Step II** : Load the first number into AX register.
- Step III** : Load the second number into BX register.
- Step IV** : Add two lower digits.
- Step V** : Adjust result to valid BCD number.
- Step VI** : Store the result in BL.
- Step VII** : Add the two upper digits with carry.
- Step VIII** : Adjust result to valid BCD number.
- Step IX** : Store the result in BH.
- Step X** : Display the result.
- Step XI** : Stop.

**Flowchart :** Refer flowchart



### **Program :**

```
.model small
```

```
.data
```

```
a dw 3629H
```

```
b dw 4738H
```

```
.code
```

```
start:
```

```

    mov  ax, @data    ; Initialize data section
    mov  ds, ax
    mov  ax, a        ; Load number1 in ax
    mov  bx, b        ; Load number2 in bx
    add  al, bl       ; add lower two digits. Result in al
    daa              ; adjust result to valid bcd
    mov  bl, al       ; store result in bl
    adc  ah, bh       ; add upper two digits. Result in ah
    mov  al, ah       ; al=ah as daa works on al only
    daa              ; adjust result to valid BCD
    mov  bh, al       ; store result in bh
    mov  ch, 04h      ; Count of digits to be displayed
    mov  cl, 04h      ; Count to roll by 4 bits
12:  rol  bx, cl       ; roll bl so that msb comes to lsb
    mov  dl, bl       ; load dl with data to be displayed
    and  dl, 0FH      ; get only lsb
    cmp  dl, 09       ; check if digit is 0-9 or letter A-F
    jbe  14

```

```

        add    dl, 07          ; if letter add 37H else only add 30H
l4: add    dl, 30H
        mov    ah, 02          ; Function 2 under INT 21H (Display character)
        int    21H
        dec    ch              ; Decrement Count
        jnz    l2
        mov    ah, 4cH         ; Terminate Program
        int    21H
        end start

```

### **Result :**

```

C:\programs>tasm addbcd16.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International
Assembling file: addbcd16.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 438k
C:\programs>tlink addbcd16
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack
C:\programs>addbcd16
8367
C:\programs>

```

## **3.2 Convert two digit Unpacked BCD to Packed BCD.**

A digit BCD number is available in register AL. We have to unpack this BCD number i.e. we have to separate the BCD digits. e.g : If the number = 92 H then in unpack form the two digits will 02 H and 09 H. i.e. we have to mask the lower nibble, first and rotate four times to the right to get the MSB digit. Then to get the LSB digit mask the upper nibble. Display the result. Masking lower nibble means ANDing the number with 0F0 to get MSB.

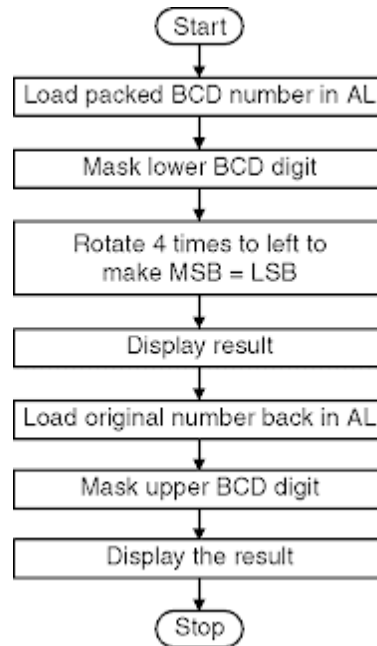
### **Algorithm :**

```

Step I      : Initialize the data memory.
Step II     : Load number into register AL.
Step III    : Mask the lower nibble.
Step IV     : Rotate 4 times left to make
                ; MSB digit = LSB.
Step V      : Display the digit.
Step VI     : Load number in AL.
Step VII    : Mask upper nibble.
Step VIII   : Display the result.
Step IX     : Stop.

```

### Flowchart :



### Program :

```
.model small
.data
a db 92H
.code
start:
    mov     ax, @data    ; Initialize data section
    mov     ds, ax
    mov     al, a        ; Load number1 in al
    and     al, 0f0h      ; mask lower nibble
    rcr     al, 4         ; rotate it 4 times to right to make it 09h
    mov     bh, al        ; store result in bh
    call    disp          ; display the upper nibble
    mov     al, a        ; Load number1 in al
    and     al, 0fh       ; mask upper nibble
    mov     bh, al        ; store result in bh
    call    disp          ; display the lower nibble
    mov     ah, 4cH       ; Terminate Program
    int     21H

disp proc near
    mov     ch, 02h       ; Count of digits to be displayed
    mov     cl, 04h       ; Count to roll by 4 bits
12:  rol     bh, cl        ; roll bl so that msb comes to lsb
    mov     dl, bh        ; load dl with data to be displayed
    and     l, 0fH        ; get only lsb
    cmp     dl, 09        ; check if digit is 0-9 or letter A-F
    jbe     l4
    add     dl, 07         ; if letter add 37H else only add 30H
14:  add     dl, 30H
    mov     ah, 02        ; Function 2 under INT 21H (Display character)
```

```
int    21H
dec    ch          ; Decrement Count
jnz    l2
mov     ah, 02h
mov     dl, ''
int     21h
endp
ret
end start
```

**Result :**

```
C:\programs>tasm unpack.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International
Assembling file:  unpack.asm
Error messages:  None
Warning messages: None
Passes:         1
Remaining memory: 437k
C:\programs>tlink unpack
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack
C:\programs>unpack
09 02
```

## **Module 4:**

### **4.1 Program to move set of numbers from one memory block to another.**

First, we initialize SI with the first memory address and DI with the destination memory address. Secondly, we initialize the counter as to how many memories should be transferred. Then we move contents of SI to a temporary register and then move the temporary registers to DI and then increment SI and DI. Then we decrement the counter to check if its zero. If it is not zero then repeat the steps again until the counter exhausts.

#### **Algorithm:**

**Step 1:** Initialize SI with the Source memory address

**Step 2:** Initialize DI with the Destination memory address

**Step 3:** Initialize the CX with the count

**Step 4:** Move contents of SI to BL

**Step 5:** Move contents of temporary register to DI

**Step 6:** Increment SI

**Step 7:** Increment DI

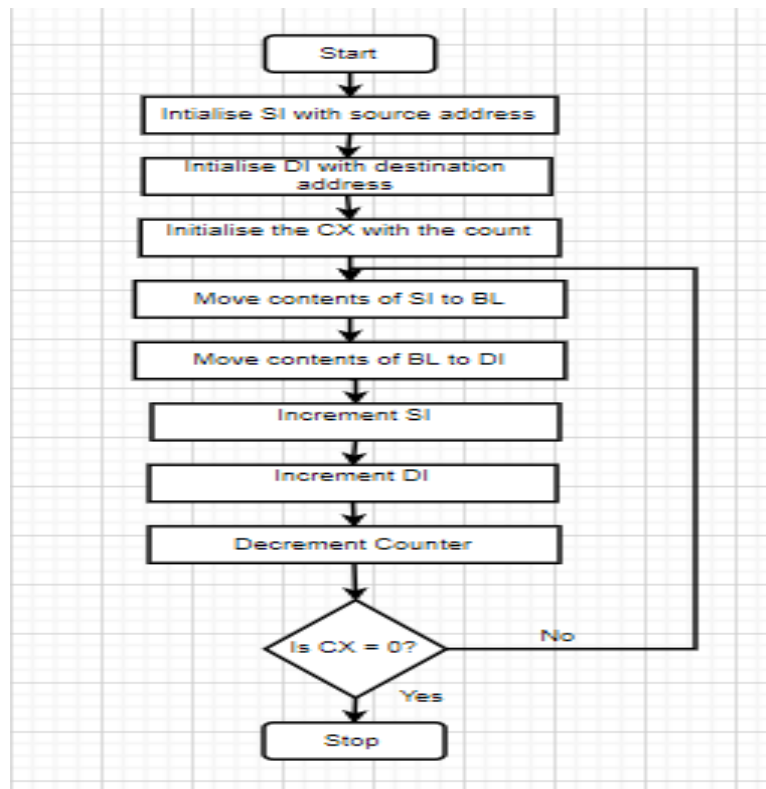
**Step 8:** Decrement CX

**Step 9:** Go to Step 4 if CX not zero

**Step 10:** Display the result

**Step 11:** Stop

### Flowchart :



### Program :

.model small

.code

start:

MOV SI,2000h

MOV DI,4000h

MOV CL,05h

LOOP1: MOV bl,[SI]

MOV [DI],bl

INC SI

INC DI

DEC CL

JNZ LOOP1

int 03h

end start

Result :

```
File Edit View Run Breakpoints Data Options Window Help READY
[ ]=Dump 2 [ ]
ds:1FA3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FB3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FC3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FD3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FE3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FF3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2003 00 00 00 00 00 00 00 00 05 20 00 00 00 00 00 00
ds:2013 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2023 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2033 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2043 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2053 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2063 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2073 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2083 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2093 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20A3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20B3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20D3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20E3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

```
File Edit View Run Breakpoints Data Options Window Help READY
[ ]=Dump 2 [ ]
ds:3FB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3F90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4000 00 00 00 00 00 00 00 00 00 00 00 05 20 00 00 00
ds:4010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:4090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:40A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:40B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:40C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```



## **4.2 Program to count number of 1's and 0's in a given 8 bit number**

First, we initialize BX Register with the number and CX with 8. Then we rotate the number one along with the carry such that the last digit enters the carry. Now if carry flag is set to one, we increment the one's counter by one. Else we increment zero's counter by one. Now we decrement the counter and check if it is zero. If it isn't then repeat the procedure once again, else display the result.

### **Algorithm:**

**Step 1 :** Initialize BL with the number

**Step2 :** Initialize CL with the number of bits

**Step3 :** Initialize DH with 00 and DL with 00

**Step 4 :** Rotate right with carry

**Step 5 :** If Carry is set to one, go to Step7

**Step6 :** Increment DL

**Step7 :** Go to Step 9

**Step8 :** Increment DH

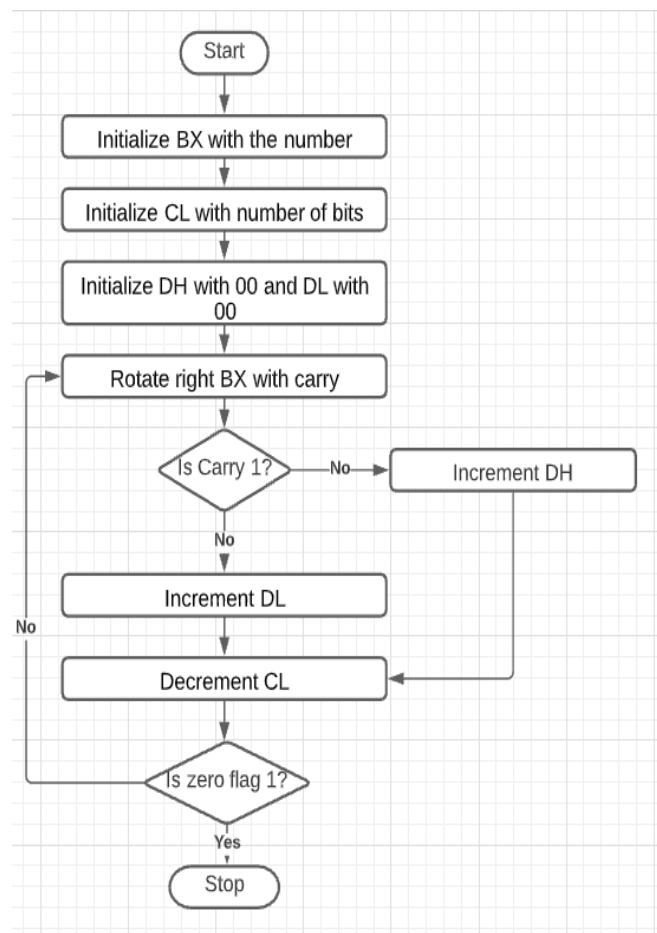
**Step9 :** Decrement CL

**Step 10 :** If Zero flag is not set to zero, go to Step 4

**Step 11 :** Display Result

**Step 12 :** Stop

### **Flowchart:**



### **Program :**

.model small

.data

n1 db 31h

zeros db 1 dup(0)

ones db 1 dup(0)

.code

Start:

mov ax,@data

mov ds,ax

mov cl,08h

mov ah,00h

mov al,n1

mov dx,0000h

```

up:  rcl    al,01H
      JC     next
      inc    dl
      jmp    down
next: inc     dh
down: loop up
      mov     zeros, dh
      mov     ones, dl
      int     03H
      end Start

```

### Result :

File Edit View Run Breakpoints Data Options Window H

[ ]-CPU 80486 ds:0000 = 88 1=[↑][↓]

Address	Instruction	Register/Value
cs:001A	E2F3 loop 000F	ax 0018 c=1
cs:001C	88360700 mov [0007],dh	bx 0000 z=0
cs:0020	88160800 mov [0008],dl	cx 0000 s=0
cs:0024	CC int 03	dx 0305 o=0
cs:0025	0031 add [bx+di],dh	si 0000 p=1
cs:0027	0305 add ax,[di]	di 0000 a=0
cs:0029	0000 add [bx+si],al	bp 0000 i=1
cs:002B	0000 add [bx+si],al	sp 0000 d=0
cs:002D	0000 add [bx+si],al	ds 48AF
cs:002F	0000 add [bx+si],al	es 489D
cs:0031	0000 add [bx+si],al	ss 48AC
cs:0033	0000 add [bx+si],al	cs 48AD
cs:0035	0000 add [bx+si],al	ip 0025

es:0000 CD 20 FF 9F 00 EA FF FF = f R

es:0008 AD DE E0 01 C5 15 AA 01 i |x|S-@

es:0010 C5 15 89 02 20 10 92 01 +Se@ >ff@

es:0018 01 03 01 00 02 FF FF FF @v@ @

ss:0002 6474

ss:0000 0000

### 4.3 Program to find even and odd numbers from a given list

First, we create an input array and two other arrays of equal size to store all even and odd numbers respectively. At the end, we display all 3 Arrays.

#### Algorithm:

**Step1 :** Initialize all messages, new line character etc. which going to be used for printing.

**Step 2 :** Load Starting Addresses of inputArray, evenArray ,oddArray in BX, SI, DI respectively.

**Step 3 :** Initialize a Counter and traverse through inputArray. For each value in array, divide it by 2 and compare the remainder.

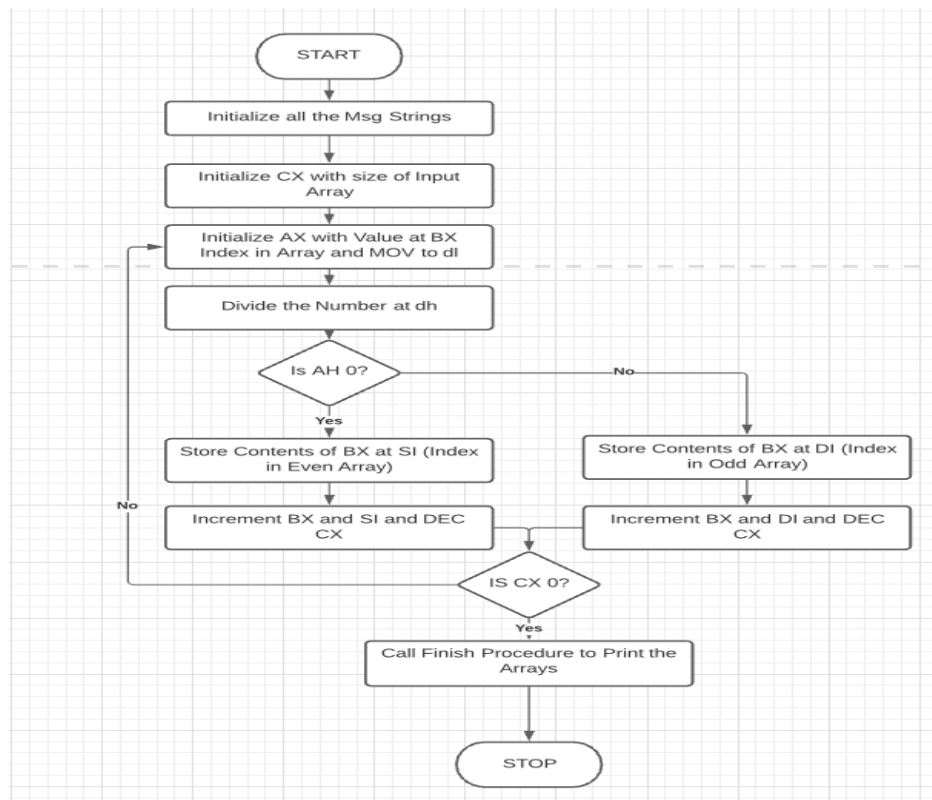
**Step4 :** If remainder is zero,jmp to is evenProc and store the di register's content at SI location i.e. index of even array.

**Step 5 :** Increment SI and BX i.e. shift the pointers to array value.

**Step6 :** Else,it's odd so store contents in odd array and increment pointers accordingly.

**Step 7 :** Call final procedure which will display all the arrays i.e. even, odd and input array and the string msgs.

#### Flowchart:



**Program :**

```
.model small

.data

array db 12h, 23h, 26h, 63h, 25h, 36h, 2fh, 33h, 10h, 35h

.code

start: MOV ax,@data

        MOV ds,ax

        MOV cl,10

        MOV SI,2000h

        MOV DI,2008h

        LEA BP,array

back: MOV AL,DS:[BP]

        MOV BL,AL

        AND AL,01H

        JZ next

        MOV [DI],bl

        INC DI

        JMP skip

next: MOV [SI],bl

        INC SI

skip: INC BP

        DEC CL

        JNZ back

int    03H

end start
```

Result :

```

File Edit View Run Breakpoints Data Options Window
[ ]-CPU 80486
cs:0003 8ED8      mov     ds,ax
cs:0005 B10A      mov     cl,0A
cs:0007 BE0020    mov     si,2000
cs:000A BF0820    mov     di,2008
cs:000D BD0A00    mov     bp,000A
cs:0010 3E8A4600  mov     al,ds:[bp]
cs:0014 8AD8      mov     bl,al
cs:0016 2401      and     al,01
cs:0018 7406      je      0020
cs:001A 881D      mov     [di],bl
cs:001C 47         inc     di
cs:001D EB04      jmp     0023
cs:001F 90         nop

ax 4801  c=0
bx 0035  z=1
cx 0000  s=0
dx 0000  o=0
si 2004  p=1
di 200E  a=0
bp 0014  i=1
sp 0000  d=0
ds 48AF
es 489D
ss 48AC
cs 48AD
ip 002F

es:0128 CC 00 12 23 26 63 25 36 || +&c%6
es:0130 2F 33 10 35 00 00 00 00 /3>5
es:0138 00 00 00 00 00 00 00 00
es:0140 00 00 00 00 00 00 00 00
ss:0002 6474
ss:0000 0000

```

```

File Edit View Run Breakpoints Data Options Window Help
[ ]-Dump
ds:1F90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:1FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2000 12 26 36 10 00 00 00 00 23 63 25 2F 33 35 00 00 +&6> #cx%/35
ds:2010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:2090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:20D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Activate Windows
Go to Settings to activate Windows.

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

## Module 5:

### 5.1 Check whether a given string is a palindrome or not.

A palindrome can be a word, number, phrase or any other sequence of characters which reads the same backward as forward, such as madam or racecar.

#### Algorithm:

**Step 1 :** Create a string

**Step 2 :** Traverse to the end of the string

**Step 3 :** Get the address of the end of the string, DI

**Step 4 :** Load the starting address of the string, SI address

**Step 5 :** Compare the value stored at the

**Step 6 :** Increment the pointer, SI

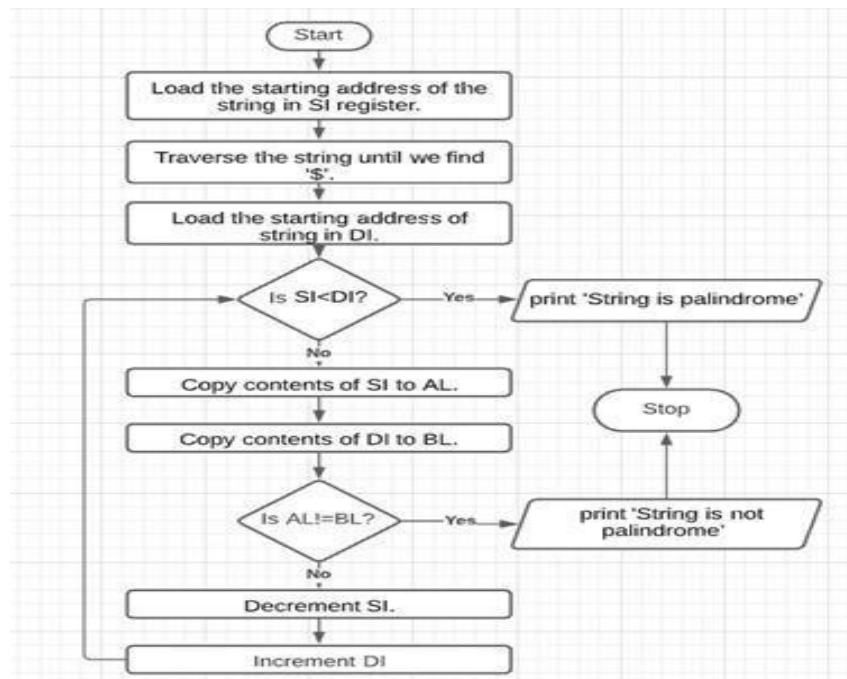
**Step 7 :** Decrements the pointer, DI

**Step 8 :** Compare again the value stored at SI and DI

**Step 9 :** Repeat the steps until  $SI \leq DI$

**Step 10 :** If all the characters match print string is palindrome else print not palindrome

#### Flowchart :



### **Program :**

```
.MODEL SMALL
.STACK 100H
.DATA                                ; The string to be printed
STRING DB 'madam', '$'
STRING1 DB 'String is palindrome', '$'
STRING2 DB 'String is not palindrome', '$'
.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
; check if the string is;
;palindrome or not
CALL Palindrome
;interrupt to exit
MOV AH, 4CH
INT 21H
MAIN ENDP
Palindrome PROC
; load the starting address
; of the string
MOV SI,OFFSET STRING
; traverse to the end of;
;the string
LOOP1 :
MOV AX, [SI]
CMP AL, '$' JE LABEL1
INC SI
JMP LOOP1
;load the starting address;
;of the string
LABEL1 :
MOV DI,OFFSET STRING
DEC SI
; check if the string is palindrome;
;or not
LOOP2 :
CMP SI, DI
JL OUTPUT1
MOV AX,[SI]
MOV BX, [DI]
CMP AL, BL
JNE OUTPUT2
DEC SI
INC DI
JMP LOOP2
OUTPUT1:
;load address of the string
LEA DX,STRING1
```

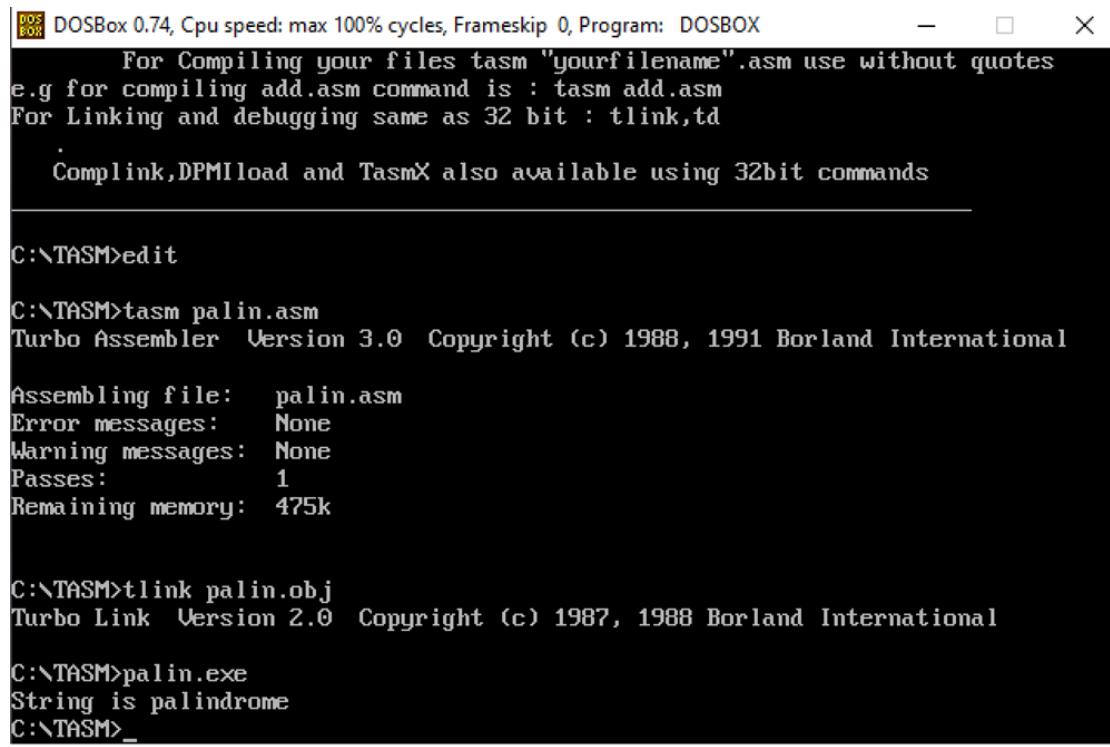


```

; output the string;
;loaded in dx
MOV AH, 09H
INT 21H
RET
OUTPUT2:
;load address of the string
LEA DX,STRING2
; output the string
; loaded in dx
MOV AH,09H
INT 21H
RET
Palindrome ENDP
END MAIN

```

### **Result :**



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: DOSBOX
For Compiling your files tasm "yourfilename".asm use without quotes
e.g for compiling add.asm command is : tasm add.asm
For Linking and debugging same as 32 bit : tlink,td

Complink,DPMIload and TasmX also available using 32bit commands

C:\TASM>edit

C:\TASM>tasm palin.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   palin.asm
Error messages:   None
Warning messages: None
Passes:           1
Remaining memory: 475k

C:\TASM>tlink palin.obj
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\TASM>palin.exe
String is palindrome
C:\TASM>_

```

## 5.2 Compute the factorial of a positive integer 'n' using procedure.

- To compute the factorial of a number means to multiply the number n with (n-1) x (n-2) ..... (2) x (1).  
e.g.: 5! which is 120.
- In our program, we initialize AX=1 and load the number whose factorial is to be computed in BX. And, then we call the procedure fact which will compute the factorial of the number.

### ALGORITHM:

**Step 1:** Initialize the data segment.

**Step 2:** Initialize AX=1.

**Step 3:** Load the Number in BX.

**Step 4:** Call the fact procedure.

**Step 5:** Compare BX with 1. If not 1, goto Step 7

**Step 6:** AX=1 and return back to calling program

**Step 7:** AX = AX x BX

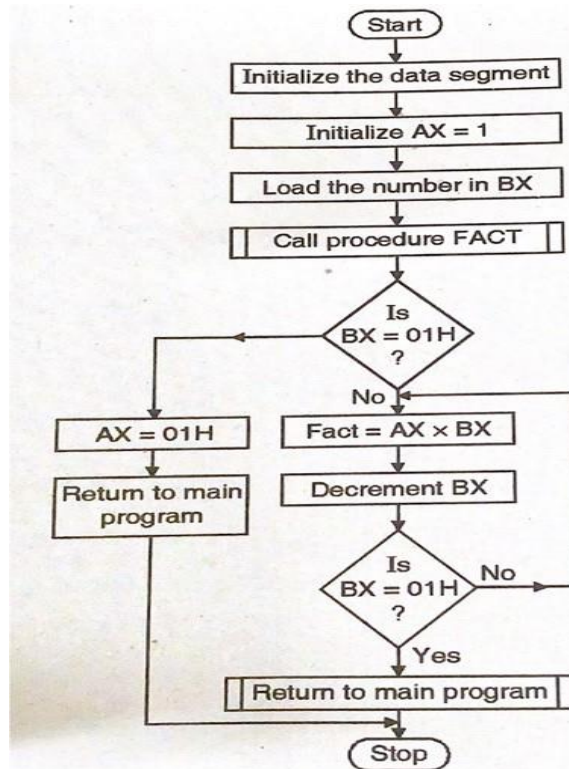
**Step 8:** Decrement BX

**Step 9:** Compare BX with 1, if not to Step 7.

**Step 10:** Return back to calling program

**Step 11:** Stop

### Flowchart :



### Program :

```

.model small
.data
num dw 05h
.code
MOV ax, @data ; initialize the data segment
MOV ds, ax
MOV ax, 01 ; initialize ax = 1
MOV bx, num ; load the number in cx
CALL fact ; call procedure
MOV di, ax ; store lsb of result in di
MOV bp, 2 ; initialize count for no of times display is called
MOV bx, dx ; store msb of result in reg bx
MOV bx, di ; store lsb of result in bx
DEC bp ; decrement bp
MOV ah, 4ch
Int 21h
Fact proc near ; function for finding the factorial
CMP bx, 01 ; if bx=1
JZ l11 ; if yes ax=1
l12: MUL bx ; find factorial
DEC bx ; decrement bx
CMP bx, 01 ; multiply bx=1
JNE l12
RET
  
```

```

l11:MOV ax,01    ;initialize ax=1
RET              ;return to called program
fact ENDP        ;end procedure
END              ;end program

```

### Result :

≡ File Edit View Run Breakpoints Data Options Window Help READY

[ ]=CPU 80486 -1=[↑][↓]

cs:0000	B8B04B	mov	ax,4B0	ax	4C78	c=0
cs:0003	8ED8	mov	ds,ax	bx	0078	z=0
cs:0005	B80100	mov	ax,0001	cx	0000	s=0
cs:0008	BB1E0000	mov	bx,[0000]	dx	0000	o=0
cs:000C	E30E00	call	001D	si	0000	p=0
cs:000F	8BF8	mov	di,ax	di	0078	a=0
cs:0011	BD0200	mov	bp,0002	bp	0001	i=1
cs:0014	8BDA	mov	bx,dx	sp	0000	d=0
cs:0016	8BDF	mov	bx,di	ds	4B0	
cs:0018	4D	dec	bp	es	4B9D	
cs:0019	B44C	mov	ah,4C	ss	4BAC	
cs:001B	CD21	int	21	cs	4BAD	
cs:001D	83FB01	cmp	bx,0001	ip	001B	

es:0000 CD 20 FF 9F 00 EA FF FF = f Ω  
 es:0008 AD DE E0 01 C5 15 AA 01 : |x|S-@  
 es:0010 C5 15 89 02 20 10 92 01 +Se@ >ff@  
 es:0018 01 03 01 00 02 FF FF FF @v@ @

ss:0002 6474  
 ss:0000 0000

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

## Module 6

### 6.1 Interfacing Seven Segment Display using Proteus.

#### Proetus Simulation Software:

The Proteus Design Suite is a proprietary software tool suite used primarily for electronic design automation. The software is used mainly by electronic design engineers and technicians to create schematics and electronic prints for manufacturing printed circuit boards.

The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analogue and digital electronics connected to it. This enables its use in a board spectrum of project prototyping in areas such as motor control, temperature control and user interface design. It also finds use in the general hobbyist community and, since no hardware is required, is convenient to use as a training or technical tool.

#### 7-segment Display:

The emission of these photons occurs when the diode junction is forward biased by an external voltage allowing current to flow across its junction, and in Electronics we call this process electroluminescence. The actual colour of the visible light emitted by an LED, ranging from blue to red to orange, is decided by the spectral wavelength of the emitted light which itself is dependent upon the mixture of the various impurities added to the semiconductor materials used to produce it.



Light emitting diodes have many advantages over traditional bulbs and lamps, with the main ones being their small size, long life, various colours, cheapness and are readily available, as well as being easy to interface with various other electronic components and digital circuits. But the main advantage of light emitting diodes is that because of their small die size, several of them can be connected together within one small and compact package producing what is generally called a 7-segment Display.

The 7-segment display, also written as “seven segment display”, consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a

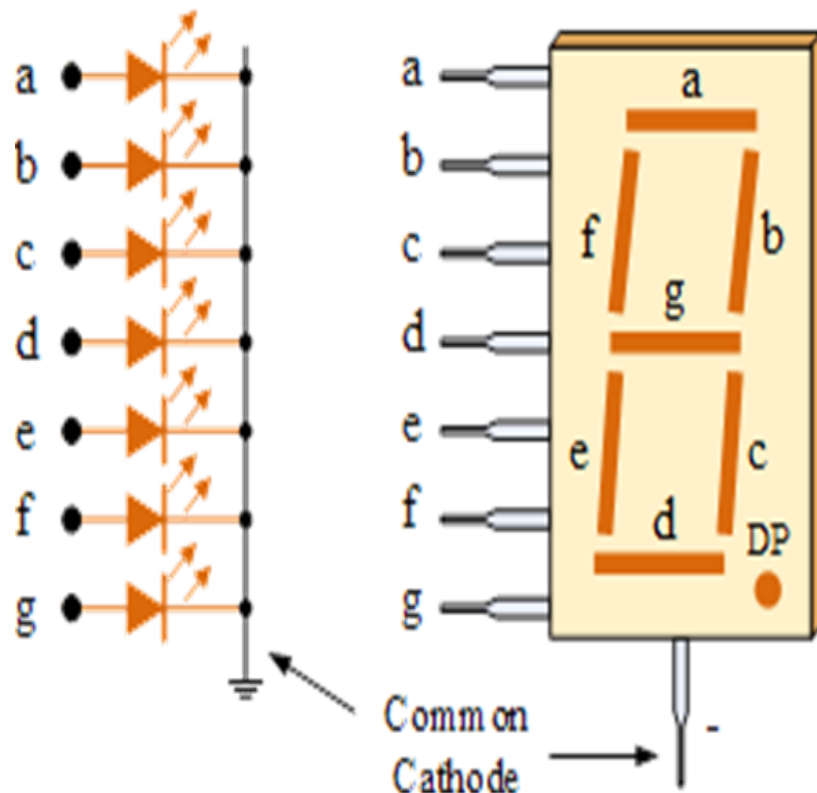
segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point, (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.

Each one of the seven LEDs in the display is given a positional segment with one of its connection pins being brought straight out of the rectangular plastic package. These individually LED pins are labeled from a through g representing each individual LED. The other LED pins are connected together and wired to form a common pin.

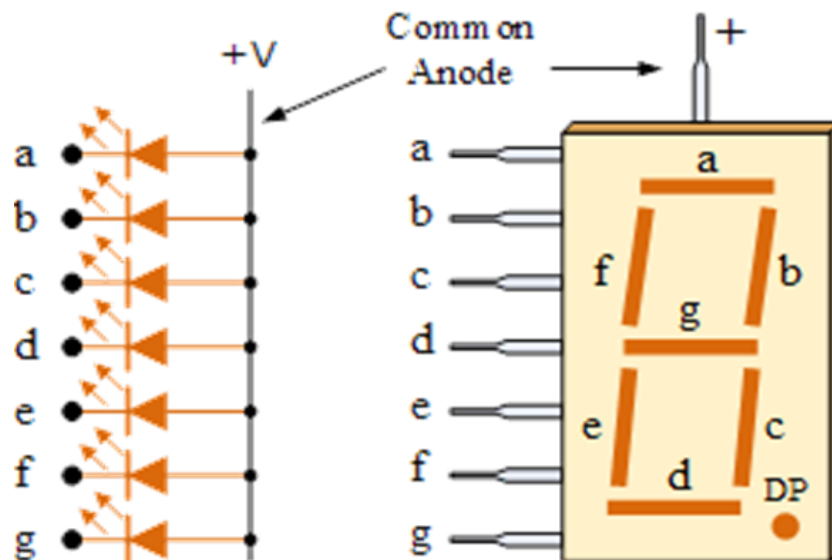
So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will be light and others will be dark allowing the desired character pattern of the number to be generated on the display. This then allows us to display each of the ten decimal digits 0 through to 9 on the same 7-segment display.

The displays common pin is generally used to identify which type of 7-segment display it is. As each LED has two connecting pins, one called the “Anode” and the other called the “Cathode”, there are therefore two types of LED 7-segment display called: Common Cathode (CC) and Common Anode (CA).

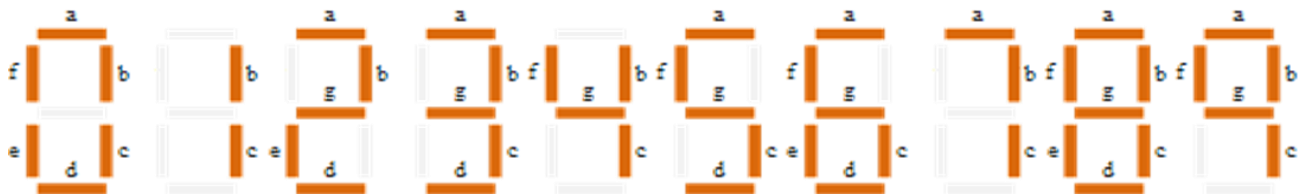
#### **Common Cathode 7-segment Display:**



### Common Anode 7-segment Display:

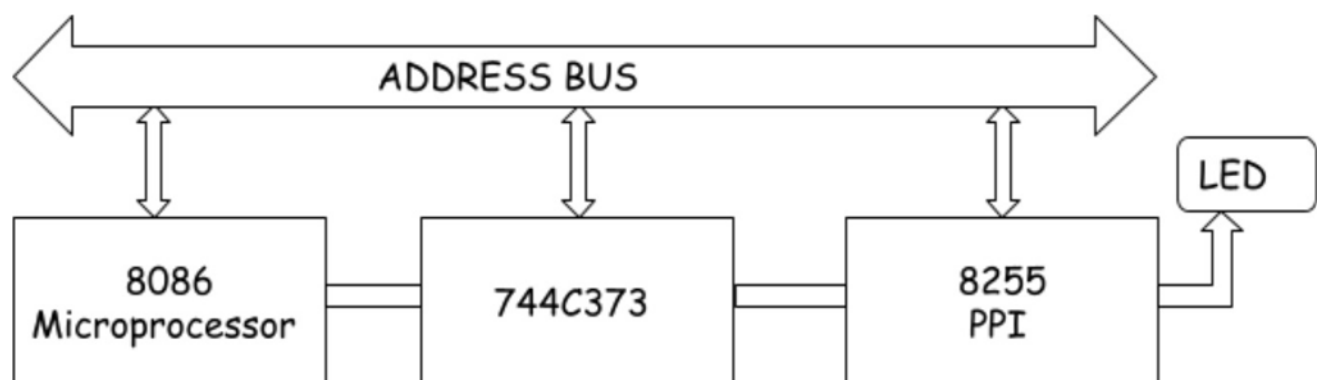


### 7-Segment Display Segments for all Numbers:



Then for a 7-segment display, we can produce a truth table giving the individual segments that need to be illuminated in order to produce the required decimal digit from 0 through 9 as shown below.

### Block Diagram of 7 Segment Display:



### Program :

```
PORTA EQU 00H
```

```
PORTB EQU 02H
```

PORTC EQU 04H

PORT\_CON EQU 06H

CODE SEGMENT ORG 100H

MOV DX, PORT\_CON

MOV AL, 10000010B ; port C (output), port A (output) in mode 0 and PORT B (INPUT) in mode 0 OUT DX, AL

MOV AL, 11000000B MOV DX, PORTA

OUT DX,AL ;Display 0 on 7Segment

START:

MOV DX, PORTB

IN AL, DX ;Check Push Buttons

MOV DX, PORTA ; prepare PORTA For Output to 7Segment

CMP AL, 11111110B JZ S0

CMP AL, 11111101B JZ S1

CMP AL, 11111011B JZ S2

CMP AL, 11110111B JZ S3

CMP AL, 11101111B JZ S4

CMP AL, 11011111B JZ S5

CMP AL, 10111111B JZ S6

CMP AL, 01111111B JZ S7

JMP START DISPLAY:

OUT DX,AL JMP START

S0:

MOV AL, 11000000B JMP DISPLAY

S1:

MOV AL, 11111001B JMP DISPLAY

S2:

MOV AL, 10100100B JMP DISPLAY

S3:

MOV AL, 10110000B JMP DISPLAY

S4:



END

