**Sr. No: 11**

**Name: Soham Desai**

**Date: 11/01/22**

# Experiment 1

**Aim:** Study of pc motherboard technology(south bridge and north bridge), various connections and parts used in computer communication

**LO & Statement** - LO: 1) Demonstrate various components and peripheral of computer system

## MOTHERBOARD:-

A **motherboard** is the main printed circuit board (PCB) in general-purpose computers and other expandable systems. It holds and allows communication between many of the crucial electronic components of a system, such as the central processing unit (CPU) and memory, and provides connectors for other peripherals. Unlike a backplane, a motherboard usually contains significant sub-systems, such as the central processor, the chipset's input/output and memory controllers, interface connectors, and other components integrated for general use.A motherboard provides the electrical connections by which the other components of the system communicate. Unlike a backplane, it also contains the central processing unit and hosts other subsystems and devices.A typical desktop computer has its microprocessor, main memory, and other essential components connected to the motherboard. Other components such as external storage, controllers for video display and sound, and peripheral devices may be attached to the motherboard as plug-in cards or via cables; in modern microcomputers, it is increasingly common to integrate some of these peripherals into the motherboard itself.Motherboards are produced in a variety of sizes and shape called computer form factor, some of which are specific to individual computer manufacturers. However, the motherboards used in IBM-compatible systems are designed to fit various case sizes. As of 2005, most desktop computer motherboards use the ATX standard form factor — even those found in Macintosh and Sun computers, which have not been built from commodity components. A case's motherboard and power supply unit (PSU) form factor must all match, though some smaller form factor motherboards of the same family will fit larger cases. With the steadily declining costs and size of integrated circuits, it is now possible to include support for many peripherals on the motherboard. By combining many functions on one PCB, the physical size and total cost of the system may be reduced; highly integrated motherboards are thus especially popular in small form factor and budget computers.

A typical motherboard will have a different number of connections depending on its standard and form factor. A standard, modern ATX motherboard will typically have two or three PCI-Express x16 connection for a graphics card, one or two legacy PCI slots for various expansion cards, and one or two PCI-E x1 (which has superseded PCI). A standard EATX motherboard will have two to four PCI-E x16 connection for graphics cards, and a varying number of PCI and PCI-E x1 slots. It can sometimes also have a PCI-E x4 slot (will vary between brands and models). Some motherboards have two or more PCI-E x16 slots, to allow more than 2 monitors without special hardware, or use a special graphics technology called SLI (for Nvidia) and Crossfire (for AMD). These allow 2 to 4 graphics cards to be linked together, to allow better performance in intensive graphical computing tasks, such as gaming, video editing, etc.In newer motherboards, the M.2 slots are for SSD and/or Wireless network interface controller.

**Fig 1**

**Fig 2**

## INTEGRATED CIRCUIT (IC):-

An **integrated circuit** or monolithic integrated circuit (also referred to as an IC, a chip, or a microchip) is a set of electronic circuits on one small flat piece (or "chip") of semiconductor material, normally silicon. The integration of large numbers of tiny transistors into a small chip results in circuits that are orders of magnitude smaller, cheaper, and faster than those constructed of discrete electronic components. An IC can function as an amplifier, oscillator, timer, counter, logic gate, computer memory, microcontroller or microprocessor. An IC is the fundamental building block of all modern electronic devices.In 1958 Jack Kilby of Texas Instruments, Inc., and Robert Noyce of Fairchild Semiconductor Corporation independently thought of a way to reduce circuit size further. They laid very thin paths of metal (usually aluminum or copper) directly on the same piece of material as their devices. These small paths acted as wires. With this technique an entire circuit could be "integrated" on a single piece of solid material and an integrated circuit (IC) thus created. ICs can contain hundreds of thousands of individual transistors on a single piece of material the size of a pea. Working with that many vacuum tubes would have been unrealistically awkward and expensive. The invention of the integrated circuit made technologies of the Information Age feasible. ICs are now used extensively in all walks of life, from cars to toasters to amusement park rides.

**Fig 3**

**Fig 4**

## NORTHBRIDGE:-

A **northbridge**, (also host bridge, or memory controller hub) is one of two chips comprising the core logic chipset architecture on a PC motherboard. A northbridge is connected directly to a CPU via the front-side bus (FSB) to handle high-performance tasks, and is usually used in conjunction with a slower southbridge to manage communication between the CPU and other parts of the motherboard.The northbridge typically handles communications among the CPU, in some cases RAM, and PCI Express (or AGP) video cards, and the southbridge.[9][10] Some northbridges also contain integrated video controllers, also known as a Graphics and Memory Controller Hub (GMCH) in Intel systems. Because different processors and RAM require different signaling, a given northbridge will typically work with only one or two classes of CPUs and generally only one type of RAM.The northbridge plays an important part in how far a computer can be overclocked, as its frequency is commonly used as a baseline for the CPU to establish its own operating frequency. This chip typically gets hotter as processor speed becomes faster, requiring more cooling.

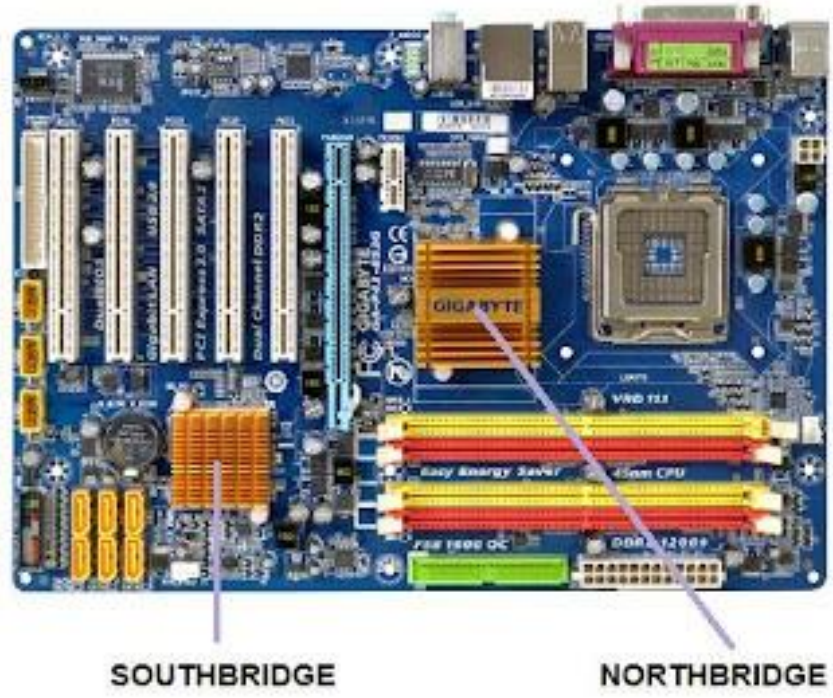



**Fig 5**

## SOUTHBRIDGE:-

The **southbridge** is one of the two chips in the core logic chipset on a personal computer (PC) motherboard, the other being the northbridge. The southbridge typically implements the slower capabilities of the motherboard in a northbridge/southbridge chipset computer architecture. In systems with Intel chipsets, the southbridge is named I/O Controller Hub (ICH), while AMD has named its southbridge 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 southbridge can usually be distinguished from the northbridge by not being directly connected to the CPU. Rather, the northbridge ties the southbridge to the CPU. Through the use of controller integrated channel circuitry, the northbridge can directly link signals from the I/O units to the CPU for data control and access.

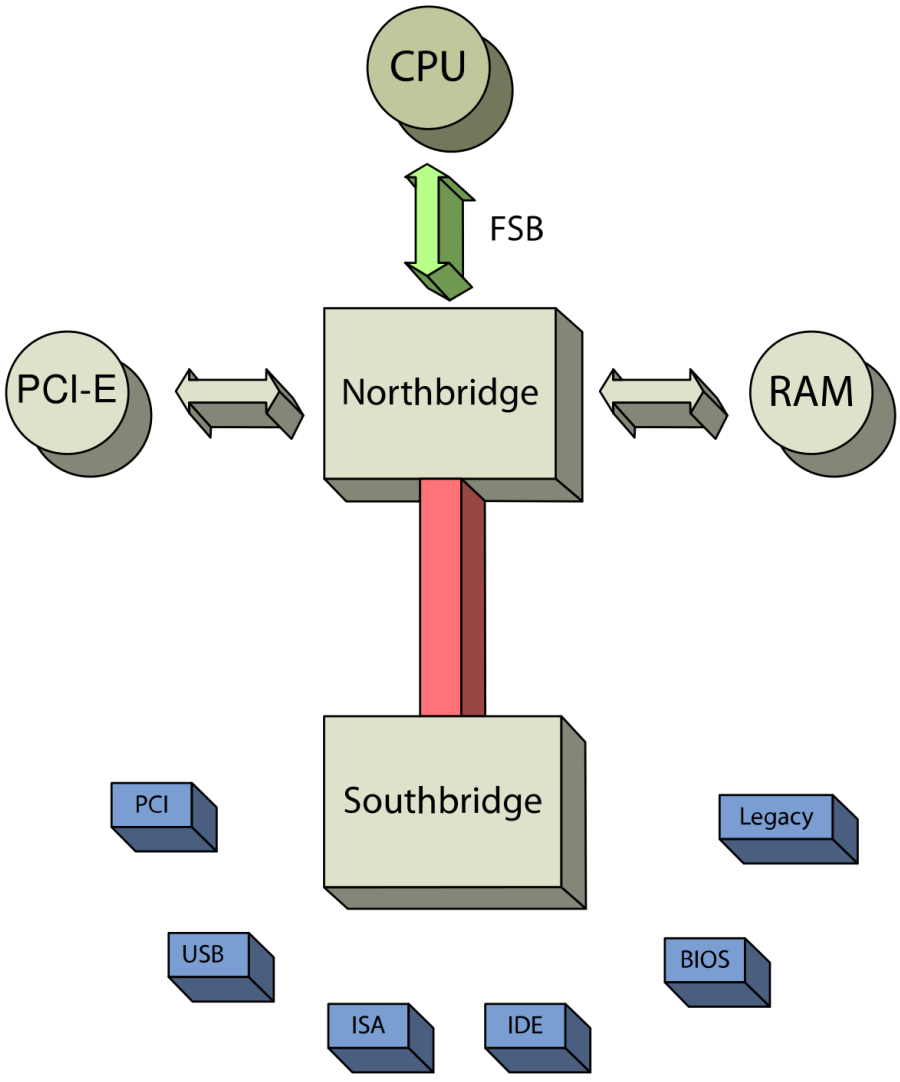



**Fig 6**

## CONNECTORS:-

The CPU socket is the array of hundreds of holes or metal plates to which a computer's central processing unit connects. The CPU socket supplies power to the processor and allows data to be sent to and from the processor from the computer's memory.

A typical motherboard has at least two sockets for **Random Access Memory (RAM).** RAM acts as a high-speed system for temporarily storing the data needed by programs while they are running. When the processor needs instructions, it receives them from the RAM, and when you save a document or file, it goes from the RAM to the hard drive.

Generally, a motherboard has at least two **hard drive connectors**. Current motherboards use Serial Advanced Technology Attachment (SATA) hard drive connectors, which have L-shaped curves to ensure that cables are connected in the correct direction. The older Integrated Drive Electronics (IDE) connector uses two rows of 20 pins each. Some motherboards have connectors for both SATA and IDE drives. The computer's CD or DVD drive also connects to an IDE or SATA interface.

Motherboards have connectors for different types of **peripherals**, usually located on a back plane that remains exposed on the back of the computer case when the tower is closed. The most common peripheral connection is the Universal Serial Bus (USB) connection, while some motherboards also have connections for audio speakers along with ports for FireWire, serial and parallel devices. Some motherboards have additional "headers," or banks of pins, that can be used to connect additional peripheral ports on the front of the computer case.

Many motherboards have connectors for computer **add-on cards**. These connectors are long slots into which the cards are inserted. There are several types of add-on card connectors. Some of the most common include Peripheral Component Interconnect Express (PCIe) and Accelerated Graphics Port (AGP), used mainly for video cards, and conventional Peripheral Component Interconnect (PCI), used for other types of add-on cards such as sound cards and storage controllers.

Every motherboard has at least one **power connector**. This connector is used to bring power from the computer's main power supply to all of the computer's components. Because some of today's desktop computers have very high power requirements, some motherboards have additional ports for auxiliary power connectors.

A **power supply unit** (**PSU**) converts mains AC to low-voltage regulated DC power for the internal components of a computer. Modern personal computers universally use switched-mode power supplies. Some power supplies have a manual switch for selecting input voltage, while others automatically adapt to the mains voltage.

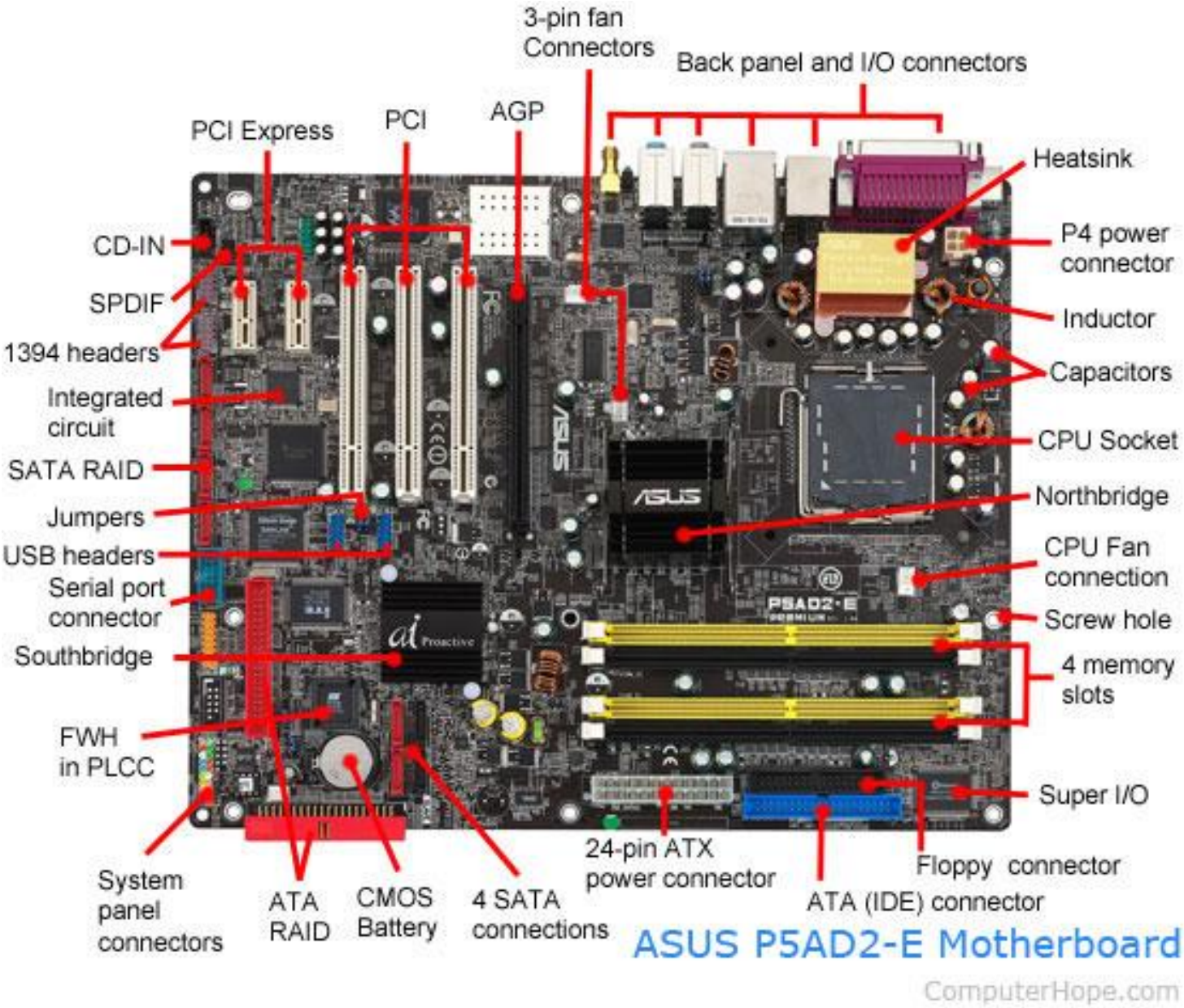

3-pin fan Connectors
Back panel and I/O connectors
PCI Express
PCI
AGP
Heatsink
CD-IN
SPDIF
1394 headers
Integrated circuit
SATA RAID
Jumpers
USB headers
Serial port connector
Southbridge
FWH in PLCC
P4 power connector
Inductor
Capacitors
CPU Socket
Northbridge
CPU Fan connection
Screw hole
4 memory slots
Super I/O
Floppy connector
ATA (IDE) connector
24-pin ATX power connector
System panel connectors
ATA RAID
CMOS Battery
4 SATA connections
ASUS P5AD2-E Motherboard
ComputerHope.com

**Fig 7**

**Sr. No: 11**

**Name: Soham Desai**

**Date: 18/01/22**

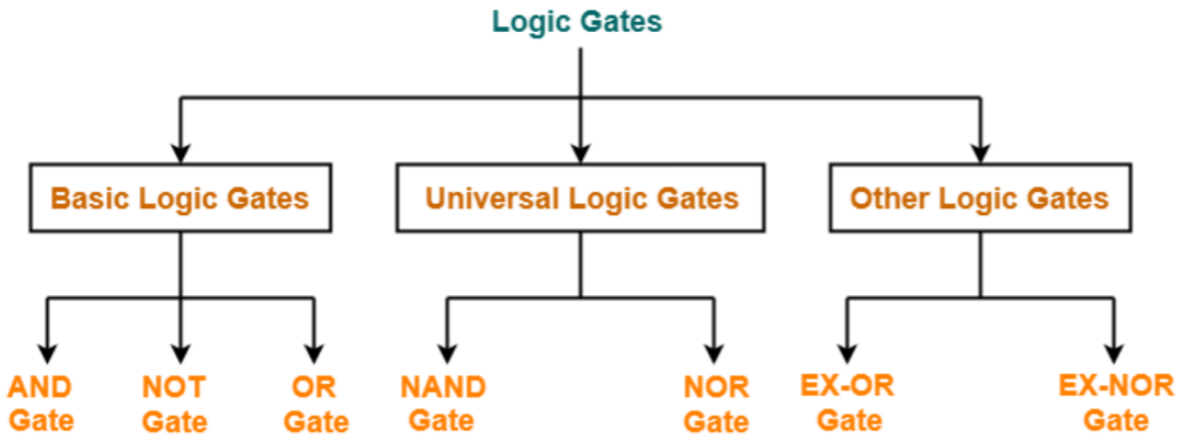
# Experiment 2

**Aim:** Verify the truth table of various logic gates (basic and universal gates)

**LO & Statement :** LO: 2 ) Analyze and design combinational circuits

**Software Requirements:** Logisim software

**Theory:**

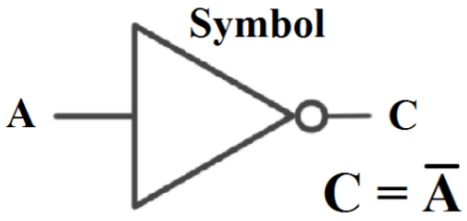



Types of Logic Gates

1)NOT GATE -

The NOT gate is a single input single output gate. This gate is also known as Inverter because it performs the inversion of the applied binary signal, i.e., it converts 0 into 1 or 1 into 0. The gate which has a high input signal only when their input signal is low such type of gate is known as the not gate.

Fig 1




**Truth Table**

| INPUT | OUTPUT |
|---|---|
| A | NOT A |
| 0 | 1 |
| 1 | 0 |

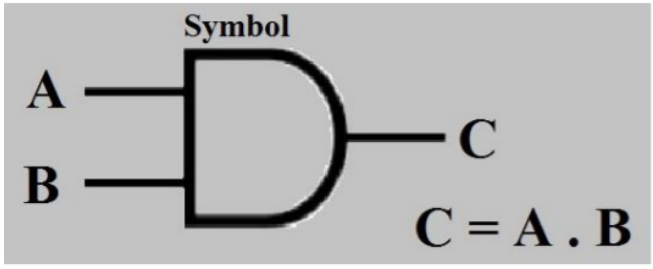
**Sr. No: 11**

**Name: Soham Desai**

**Date: 18/01/22**

2) AND GATE -

The AND gate plays an important role in the digital logic circuit. The output state of the AND gate will always be low when any of the inputs states is low. Simply, if any input value in the AND gate is set to 0, then it will always return low output(0). The logic or Boolean expression for the AND gate is the logical multiplication of inputs denoted by a full stop or a single dot



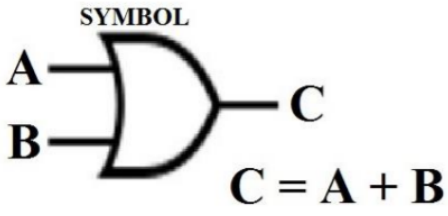

Truth Table

| INPUT | | OUTPUT |
|---|---|---|
| A | B | A AND B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Fig 2

3) OR GATE -

The OR gate is a mostly used digital logic circuit. The output state of the OR gate will always be low when both of the inputs states is low. Simply, if any input value in the OR gate is set to 1, then it will always return high-level output(1). The logic or Boolean expression for the OR gate is the logical addition of inputs denoted by plus sign(+)




TRUTH TABLE

| INPUT | | OUTPUT |
|---|---|---|
| A | B | A OR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Fig 3

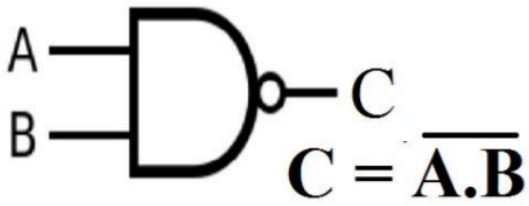
**Sr. No: 11**

**Name: Soham Desai**

**Date: 18/01/22**

4) NAND GATE -

The NAND gate is a special type of logic gate in the digital logic circuit. The NAND gate is the universal gate. It means all the basic gates such as AND, OR, and NOT gate can be constructed using a NAND gate. The NAND gate is the combination of the NOT-AND gate. The output state of the NAND gate will be low only when all the inputs are high. Simply, this gate returns the complement result of the AND gate.



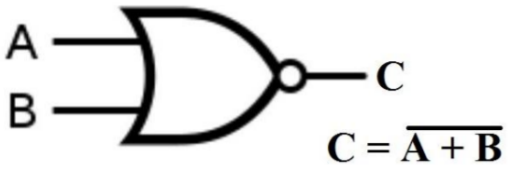

**Truth Table**

| INPUT | | OUTPUT |
|---|---|---|
| A | B | A NAND B |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Fig 4

5) NOR GATE -

The NOR gate is also a universal gate. So, we can also form all the basic gates using the NOR gate. The NOR gate is the combination of the NOT-OR gate. The output state of the NOR gate will be high only when all of the inputs are low. Simply, this gate returns the complement result of the OR gate.




| INPUT | | OUTPUT |
|---|---|---|
| A | B | A NOR B |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Fig 5

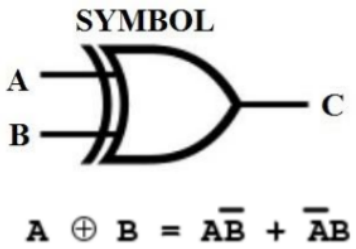
**Sr. No: 11**

**Name: Soham Desai**

**Date: 18/01/22**

6) XOR GATE -

The XOR gate stands for the Exclusive-OR gate. This gate is a special type of gate used in different types of computational circuits. Apart from the AND, OR, NOT, NAND, and NOR gate, there are two special gates, i.e., Ex-OR and Ex-NOR. These gates are not basic gates in their own and are constructed by combining with other logic gates. Their Boolean output function is significant enough to be considered as a complete logic gate. The XOR and XNOR gates are the hybrids gates. The 2-input OR gate is also known as the Inclusive-OR gate because when both inputs A and B are set to 1, the output comes out 1(high). In the Ex-OR function, the logic output "1" is obtained only when either A="1" or B="1" but not both together at the same time. Simply, the output of the XOR gate is high(1) only when both the inputs are different from each other.




| INPUT | | OUTPUT |
|---|---|---|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Fig 6

**Sr. No: 11**

**Name: Soham Desai**

**Date: 18/01/22**

7) XNOR GATE -

The XNOR gate is the complement of the XOR gate. It is a hybrid gate. Simply, it is the combination of the XOR gate and NOT gate. The output level of the XNOR gate is high only when both of its inputs are the same, either 0 or 1. The symbol of the XNOR gate is the same as XOR, only complement sign is added. The XNOR gate is also called the Equivalence gate.

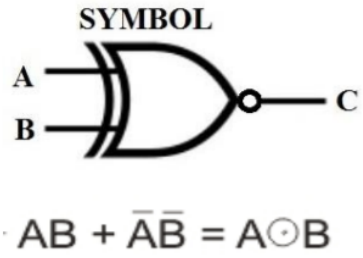
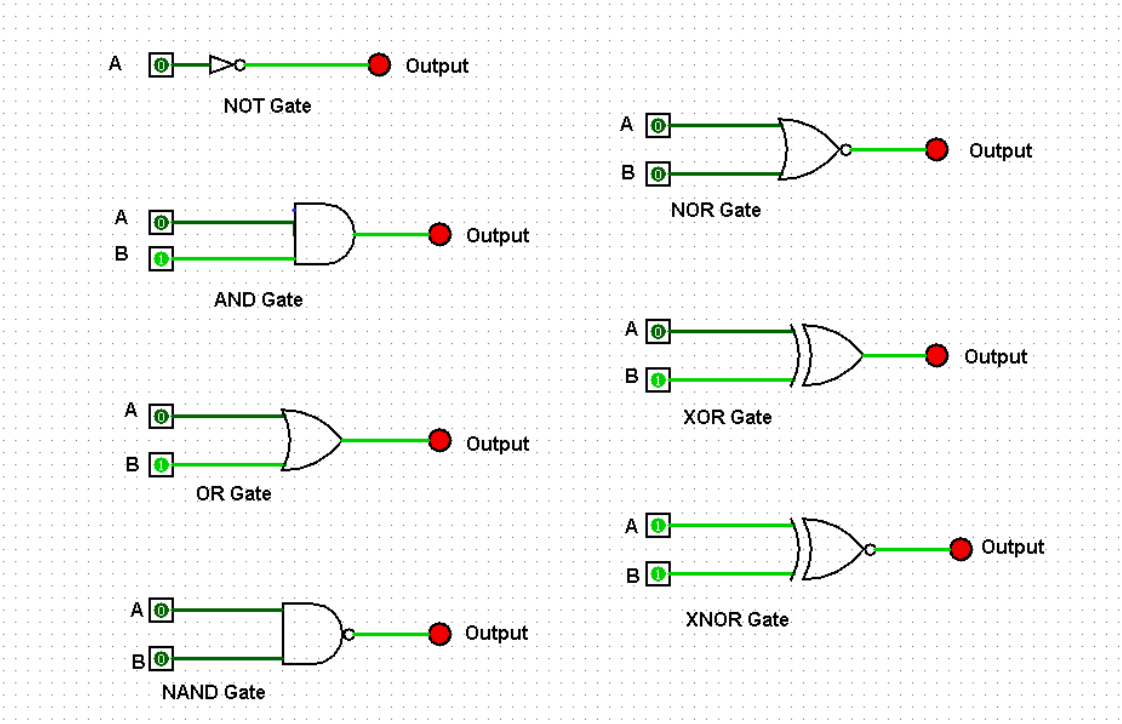



Fig 7

| Input | | Outpu |
|---|---|---|
| A | B | A XNOR |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## Output :




**Conclusion :** All the logic gates are created successfully and are running as desired.

**Sr. No: 11**

**Name: Soham Desai**

**Date: 25/01/22**

# Experiment 3

**Aim:** Realize Half adder and Full adder

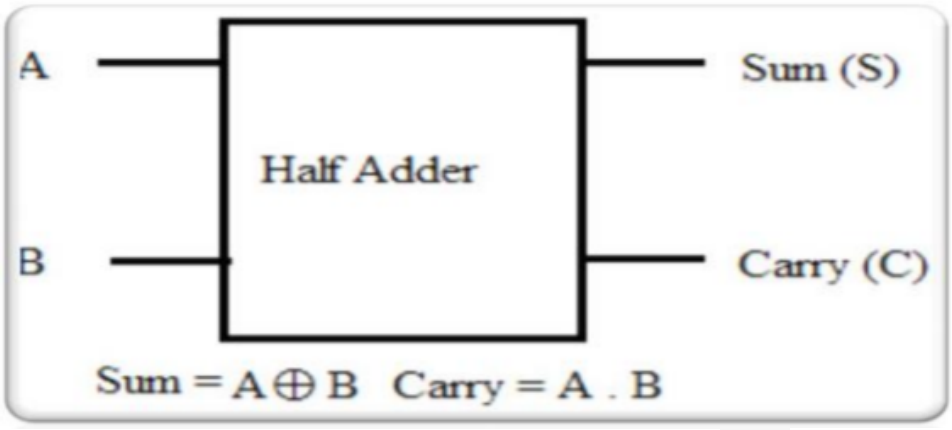
**LO & Statement :** LO: 2 ) Analyze and design combinational circuits

**Software Requirements:** Logisim software

## Theory:

### Half Adder

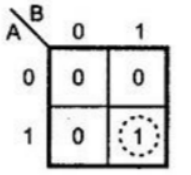
• The half adder is an example of a simple, functional digital circuit built from two logic gates.
• The half adder adds to one-bit binary numbers (A,B).
• The output is the sum of the two bits (S) and the carry




# Half adder truth table

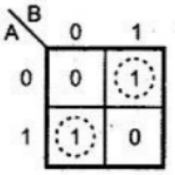
| A | B | Sum | Carry-Out |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**For carry**




Carry = AB

**For Sum**




$S = \bar{A}B + A\bar{B}$

**S=A ⊕ B (Exclusive OR)**

**C=A.B (AND)**

**Sr. No: 11**

**Name: Soham Desai**

**Date: 25/01/22**

## Full Adder

• The full adder accepts two inputs bits and an input carry and generates a sum output and an output carry.
• The full-adder circuit adds three one-bit binary numbers (Cin, A ,B) and outputs two one-bit binary numbers, a sum (S) and a carry (Cout).
• The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.


## HALF ADDER TRUTH TABLE

| 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 |

For S:


$S = A \oplus B \oplus C_{in}$

For $C_{out}$:

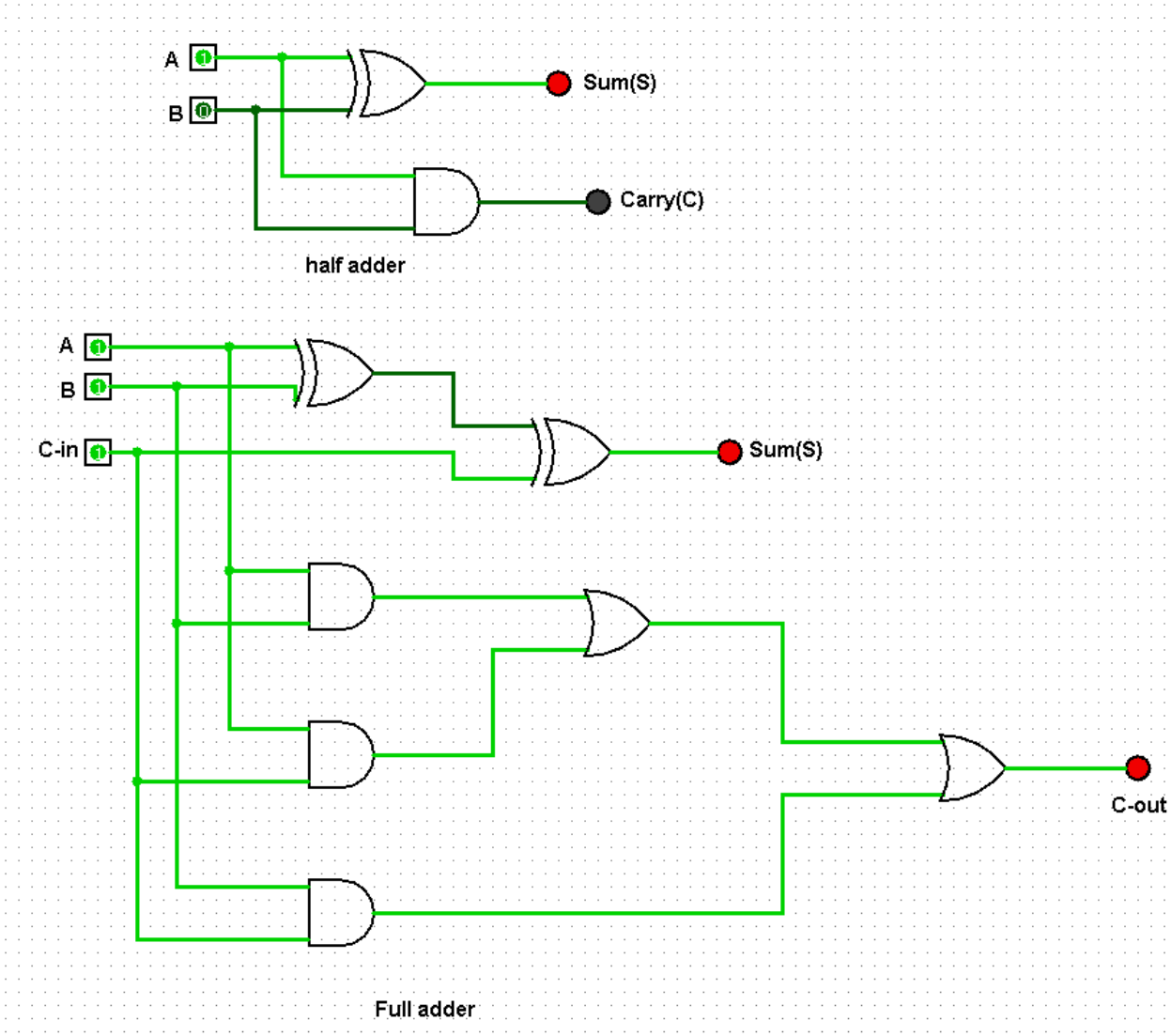

$C_{out} = AB + BC_{in} + C_{in}A$

**Sr. No: 11**

**Name: Soham Desai**

**Date: 25/01/22**

## Output:




## Conclusion:

The half adder and full adder is working as expected

**Sr. No: 11**

**Name: Soham Desai**

**Xavier ID: 202003021**

**Date: 02/02/22**

# Experiment 4

**Aim:** Implementation of MUX and DeMUX

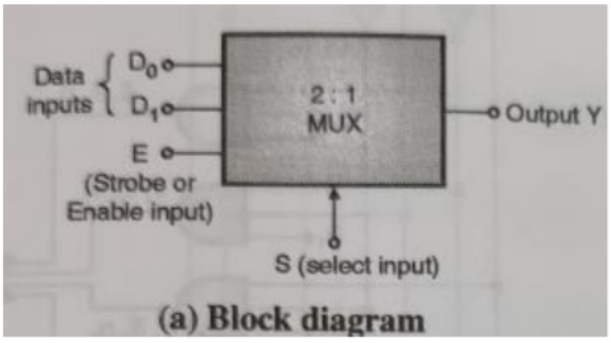
**LO & Statement :** LO: 2 ) Analyze and design combinational circuits

**Software Requirements:** Logisim software

**Theory:**

Multiplexer

- A Multiplexers (MUX) is a combinational logic component that has several inputs and only one output.
- MUX directs one of the inputs to its output line by using a control bit word (selection line) to its select lines.
- The multiplexer is sometimes called a data selector.
- The multiplexer acts like an electronic switch that selects one from different.




(a) Block diagram

| Enable | Select input S | Output Y |
|---|---|---|
| 0 | X | 0 |
| 1 | 0 | $D_0$ |
| 1 | 1 | $D_1$ |

X = Don't care

(b) Truth table

| Enable E | Select S | $D_1$ | $D_0$ | Output Y |
|---|---|---|---|---|
| 0 | X | X | X | 0 |
| 1 | 0 | X | 0 | 0 |
| 1 | 0 | X | 1 | 1 |
| 1 | 1 | 0 | X | 0 |
| 1 | 1 | 1 | X | 1 |

**Sr. No: 11**

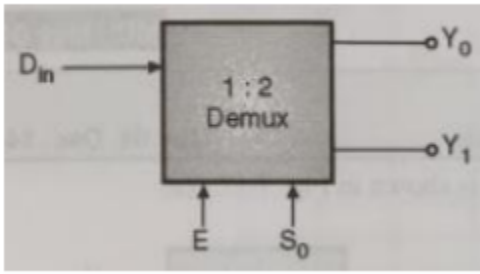
**Name: Soham Desai**

**Xavier ID: 202003021**

**Date: 02/02/22**

# Demultiplexer

- The demultiplexer is a combinational logic circuit that performs the reverse operation of a multiplexer (Several output lines, one input line).
- Few types of multiplexer are 1-to-2, 1-to-4, 1-to-8, 1-to-16 multiplexer
- The digitally controlled analogue switches of the demultiplexer select an input resistor to vary the value of Rin.
- The combination of these resistors will determine the overall voltage gain of the amplifier, (Av)




| Enable | Select | Outputs | |
|---|---|---|---|
| E | $S_0$ | $Y_1$ | $Y_0$ |
| 0 | X | 0 | 0 |
| 1 | 0 | 0 | $D_{in}$ |
| 1 | 1 | $D_{in}$ | 0 |

**Table 7.13.2 : Detail truth table of demux 1 : 2**

| Enable | Data | Select | Outputs | | |
|---|---|---|---|---|---|
| E | $D_{in}$ | $S_0$ | $Y_1$ | $Y_0$ | |
| 0 | X | X | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | $Y_0$ is connected to $D_{in}$ |
| 1 | 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | 0 | $Y_1$ is connected to $D_{in}$ |
| 1 | 1 | 1 | 1 | 0 | |

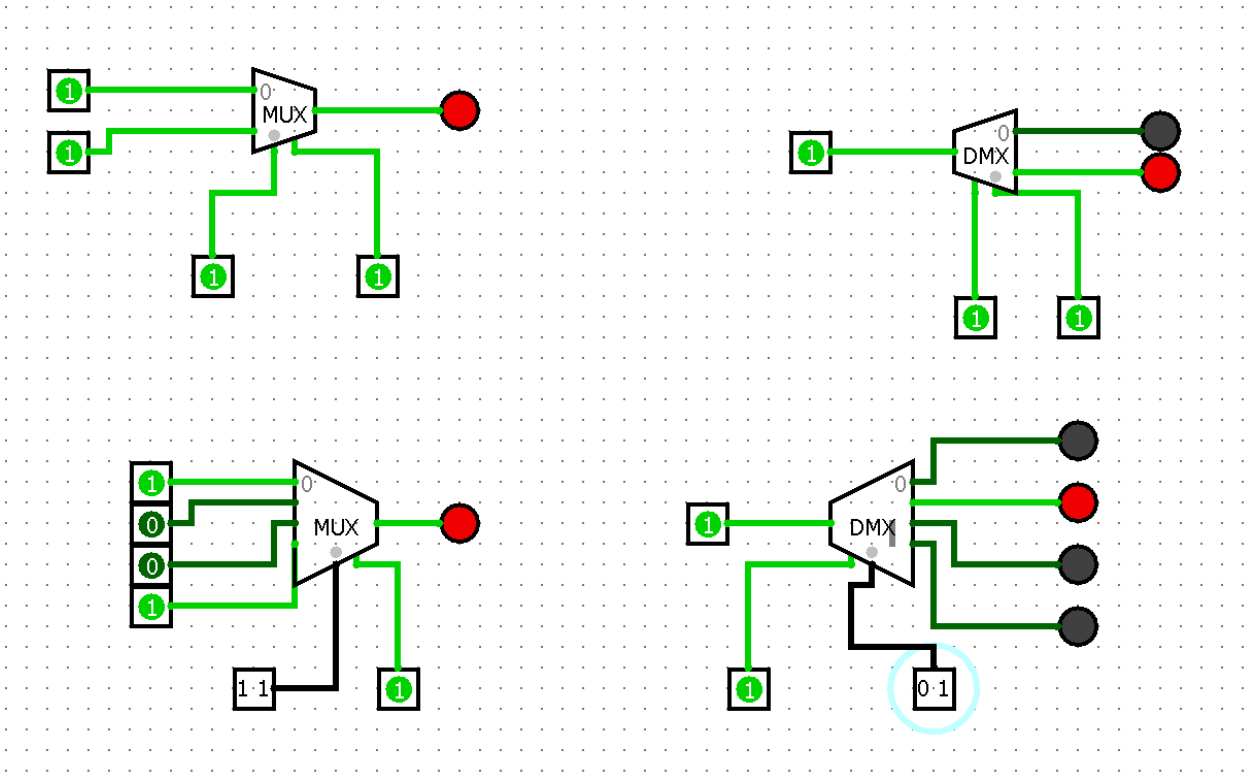
**Sr. No: 11**

**Name: Soham Desai**

**Xavier ID: 202003021**

**Date: 02/02/22**

## Output:




## Conclusion:

The implementation of MUX and DEMUX is carried out successfully.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 9/3/22**

# EXPERIMENT 5

**Aim:** Program for 16 bit BCD addition

**LO:** 3

**LO STATEMENT:** Build a program on a microprocessor using arithmetic & logical instruction set of 8086.

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction:

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**

Move Destination, Source

**Example:**

MOV Ax, Bx

### 2. ADD Instructions:

The ADD instructions are used for performing simple addition of binary data in byte, word and doubleword size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.

The ADD/SUB instruction can take place between −

- Register to register
- Memory to register
- Register to memory
- Register to constant data
- Memory to constant data

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 9/3/22**

**Syntax:**

ADD destination, source

**Example:**

ADD AX,BX

### 3. DAA Instruction:

The DAA (Decimal Adjust after Addition) instruction allows addition of numbers represented in 8-bit packed BCD code. It is used immediately after normal addition instruction operating on BCD codes. This instruction assumes the AL register as the source and the destination, and hence it requires no operand.

**Syntax:**

DAA

**Example:**

ADD AL,BL

DAA

### 4. ADC Instruction:

Adds the destination operand (first operand), the source operand (second operand), andthe carry (CF) flag and stores the result in the destination operand.

The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.)

The state of the CF flag represents a carry from a previous addition When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format. The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF

flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result. The ADC instruction is usually executed as part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 9/3/22**

**Example:**

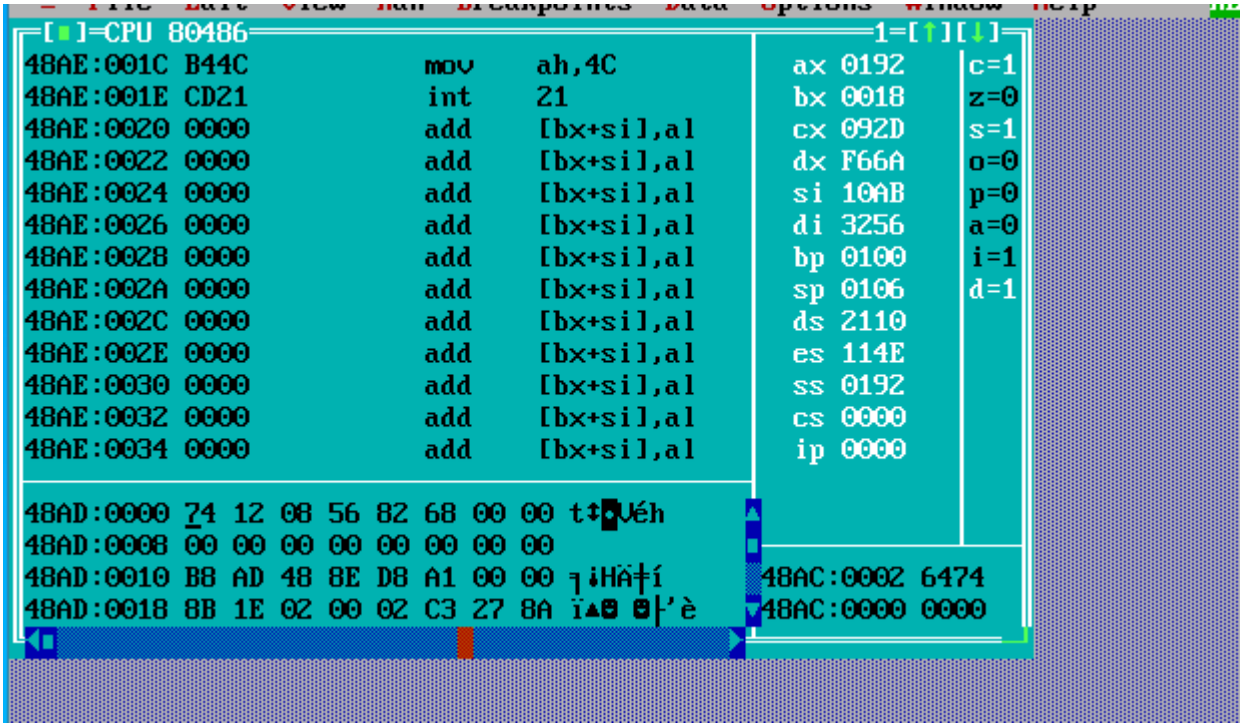
ADC AX,BX

## Code:

```
Assume CS: Code, DS: Data
Data Segment
n1 dw 1274H
n2 dw 5608H
ans dw 01 dup(?)
Data ends
Code Segment
Start: Mov Ax,Data
Mov Ds,Ax
Mov Ax,n1
Mov Bx,n2
Add AL,BL
DAA
Mov CL,AL
Mov AL,AH
Adc AL,BH
DAA
Mov CH,AL
Mov ans,Cx
Mov AH,4CH
INT 21H
Code ends
end Start
```

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 9/3/22**

## Output:




## Conclusion:

From this experiment we have learned how to run a 64 bit BCD addition using TASM software.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/3/22**

# EXPERIMENT 6

**Aim:** Program to evaluate given logical expression – NOT[(A OR B) AND (B OR C)]

**LO:** 3

**LO STATEMENT:** Build a program on a microprocessor using arithmetic & logical instruction set of 8086.

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction:

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**

Move Destination, Source

**Example:**

MOV Ax, Bx

### 2. AND Instruction:

The AND instruction perform logical AND operation between two operands. The source can be an immediate, register, or a memory location and the destination can be either a register or a memory location. Both source and destination operands cannot be a memory location. It ANDs each bit of source operand with the destination operand and stores the result back into the destination operand.

**Syntax:**

AND Destination, Source

**Example:**

AND Ax, Bx

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/3/22**

**3. Not Instruction:**

The NOT instruction implements the bitwise NOT operation. NOT operation reverses the bits in an operand. The operand could be either in a register or in the memory for negating 8-bit, 16-bit or 32-bit operands, respectively.

**Syntax:**

NOT destination

**Example:**

NOT AX

**4. OR Instruction:**

It performs the OR operation between two operands and stores the result back into the destination operand. The destination operand can be a register or a memory location whereas the source can be immediate, register, or a memory location.

**Syntax:**

OR Destination, Source

**Example:**

OR Ax, BX

**5. INT instruction:**

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

**Example:**

INT 21H

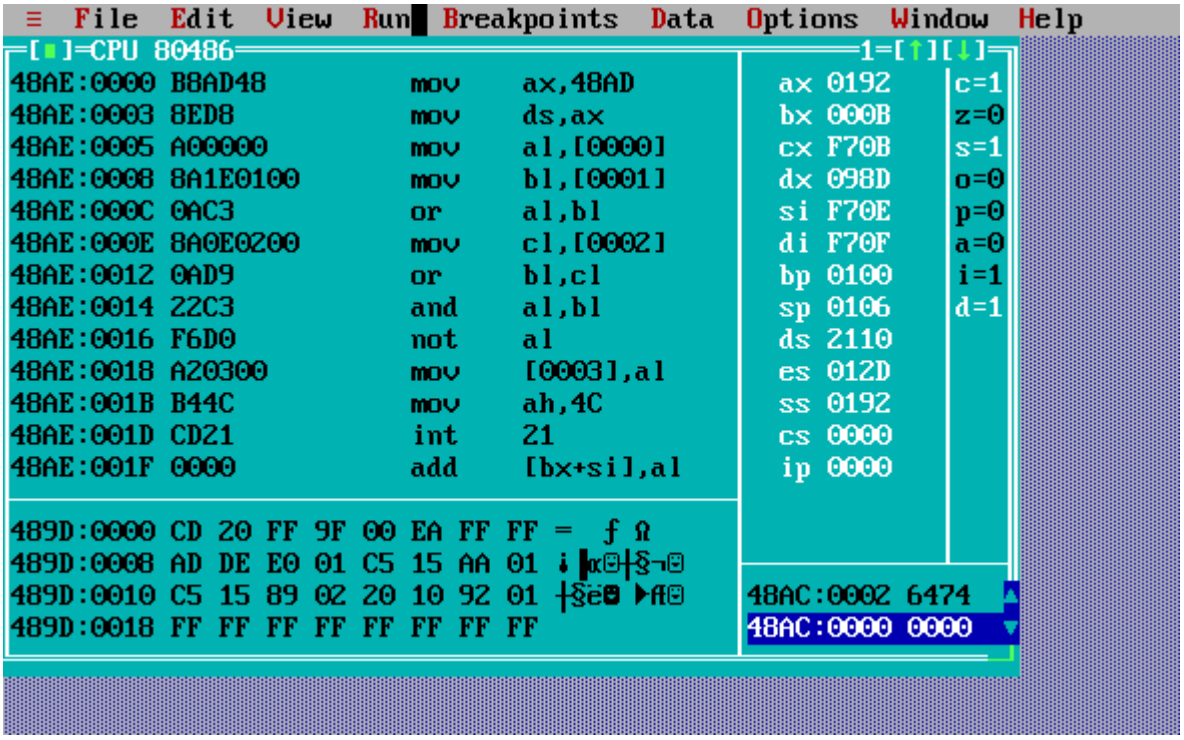
**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/3/22**

## Code:

```
assume cs:code,ds:data

data segment

A db 11

B db 34

C db 55

y db 01 dup(?)

data ends

code segment

start:

    Mov Ax,data

    Mov Ds,Ax


    Mov Al,A

    Mov Bl,B

    OR Al,Bl

    Mov Cl,C

    OR Bl,Cl

    AND Al,Bl

    NOT Al

    Mov y,Al


   Mov AH,4CH

    INT 21H

    code ends

    end start
```

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/3/22**

## Output:




**Conclusion:** From this experiment we have learnt how to evaluate a logical expression using TASM software.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 23/3/22**

# EXPERIMENT 7

**Aim:** Program to move set of numbers from one memory block to another.

**LO:** 4

**LO STATEMENT:** Develop the assembly level programming using 8086 loop instruction set

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**
Move Destination, Source

**Example:**
MOV Ax, Bx

### 2. MOVSB Instruction

MOVSB instruction will perform all the actions in repeat unit loop. The MOVSB instruction will copy a byte from the location pointed to by the Direct Index Register.It will then automatically increment SI to point to the next source location. If you add a special prefix called the repeat prefix in front of the MOVSB instruction, the MOVSB instruction will be repeated and CX decremented until CX is counted down to zero.

**Syntax:**
Move Destination, Source

**Example:**
MOVSB Ax, Bx

### 3. LEA Instruction

LEA is Used to load the address of operand into the provided register. LES − Used to load ES register and other provided register from the memory. The lea instruction places the address specified by its first operand into the register specified by its second operand. Note, the contents of the memory location are not loaded, only the effective address is computed and placed into the register.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 23/3/22**

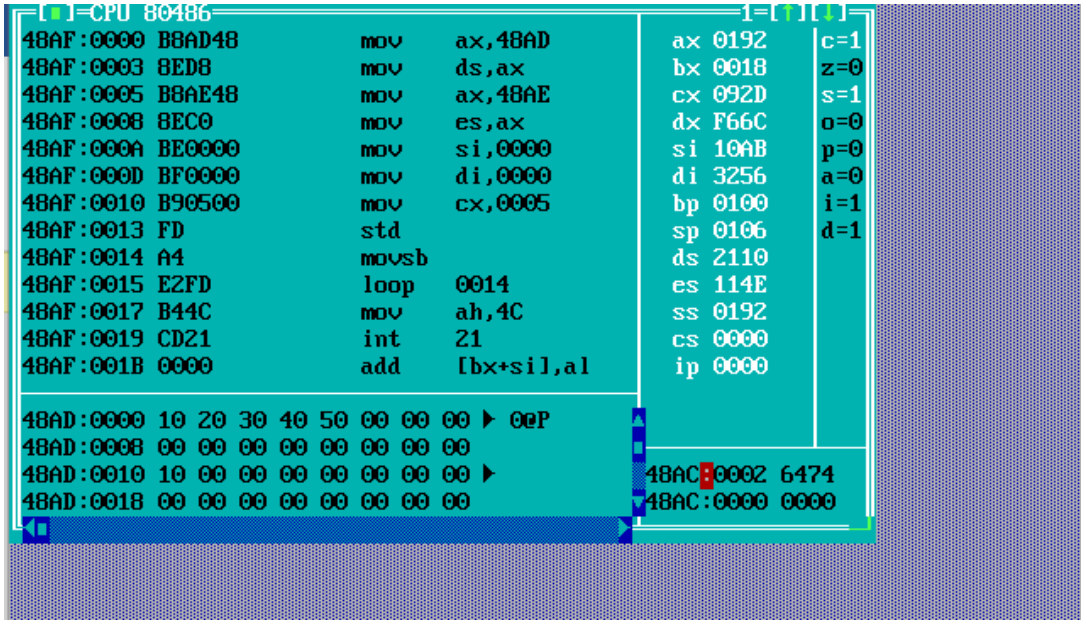
## Code:

```
assume cs:code,ds:code,es:extra
data segment
    blk1 db 10H,20H,30H,40H,50H
data ends
extra segment
    blk2 db 05 dup(?)
    extra ends
code segment
start:
     mov Ax,data
     mov Ds,Ax
     mov Ax,extra
     mov es,Ax
     lea si,blk1
     lea di,blk2
     mov Cx,05H
     std
back:movsb
    loop back
    mov AH,4CH
    int 21H
code ends
end start
```

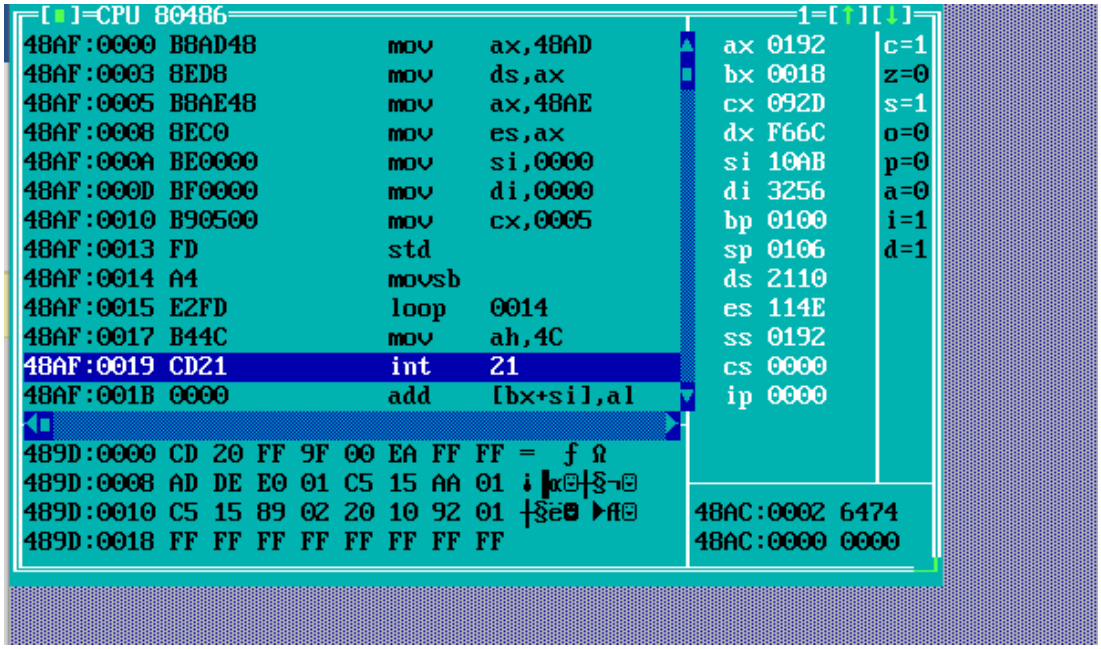
**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 23/3/22**

## Output:

For ds:




For es:




**Conclusion:** From this experiment we learn how to move numbers from one memory block to another.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

# EXPERIMENT 8

**Aim:** Program to count number of1's and 0;s in a given 8 bit number

**LO:** 4

**LO STATEMENT:** Develop the assembly level programming using 8086 loop instruction set

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**
Move Destination, Source

**Example:**
MOV Ax, Bx

### 2. SHR Instruction

SHR shifts the bits within the destination operand to the right, where right is toward the least-significant bit (LSB). The number of bit positions shifted may be specified either as an 8-bit immediate value, or by the value in CL—*not* CX or ECX. (The 8086 and 8088 are limited to the immediate value 1.) Note that while CL may accept a value up to 255, it is meaningless to shift by any value larger than 16—or 32 in 32-bit mode—*even though the shifts are actually performed on the 8086 and 8088*. (The 286 and later limit the number of shift operations performed to the native word size except when running in Virtual 86 mode.) The rightmost bit of the operand is shifted into the Carry flag; the leftmost bit is cleared to 0. The Auxiliary carry flag (AF) becomes undefined after this instruction. OF is modified *only* by the shift-by-one forms of SHL; after shift-by-CL forms, OF becomes undefined.

**Syntax:**

SHR Register, Bits to be shifted

**Example:**

SHR AX, 2

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

### 3. INT instruction:

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

**Example:**

INT 21H

### 4. JC Instruction:

JC stands for 'Jump if Carry' . It checks whether the carry flag is set or not. If yes, then jump takes place, that is: If CF = 1, then jump.

**Example:**

JC me

### 5. JMP Instruction:

Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps. The JMP instruction provides a label name where the flow of control is transferred immediately.

**Syntax:**

JMP label

**Example:**

JMP next

### 6. INC Instruction:

The INC instruction adds one to the destination operand,
while preserving the state of the carry flag CF. The destination operand can be a register or a memory location. This instruction allows a *loop counter* to be updated without disturbing the CF flag.

**Syntax:** INC destination

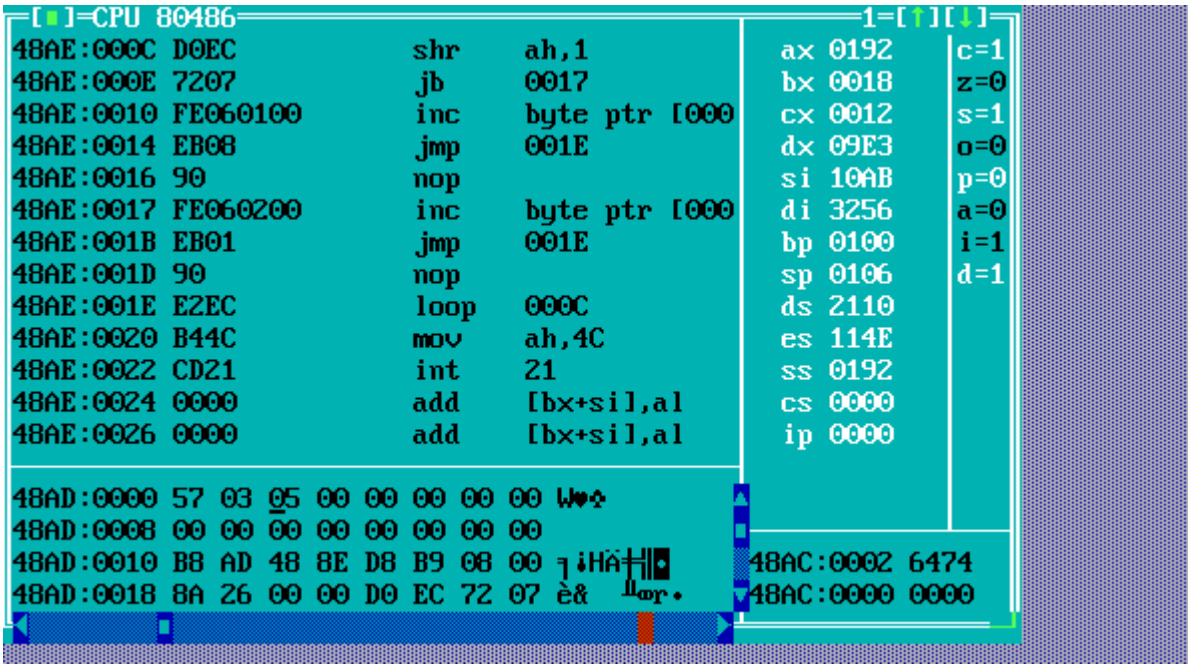
**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

## Code:

```
assume ds:data,cs:code

data segment

 no db 57H

 c0 db 01 dup(?)

 c1 db 01 dup(?)

data ends

code segment

start:mov Ax,data

      mov Ds,Ax

      mov cx,08H

      mov Ah,no

up:   SHR Ah,1

      JC down

      INC c0

      JMP next

down: INC c1

      JMP next

next: LOOP up

      mov AH,4CH

      INT 21H

code ends

end start
```

## Output:

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**




## Conclusion:

From this experiment we have learned how to use different types of commands and to count the number of 0 and 1 in a binary number.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

# EXPERIMENT 9

**Aim:** Program to search for a given number

**LO:** 4

**LO STATEMENT:** Develop the assembly level programming using 8086 loop instruction set

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**
Move Destination, Source

**Example:**
MOV Ax, Bx

### 2. LEA Instruction

LEA is Used to load the address of operand into the provided register. LES − Used to load ES register and other provided register from the memory. The lea instruction places the address specified by its first operand into the register specified by its second operand. Note, the contents of the memory location are not loaded, only the effective address is computed and placed into the register.

### 3. INC Instruction:

The INC instruction adds one to the destination operand, while preserving the state of the carry flag CF. The destination operand can be a register or a memory location. This instruction allows a loop counter to be updated without disturbing the CF flag.

**Syntax:**

INC destination

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

**4. JMP Instruction:**

Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps. The JMP instruction provides a label name where the flow of control is transferred immediately.

**Syntax:**

JMP label

**Example:**

JMP next

**5. JZ Instruction:**

he jz instruction is a conditional jump that follows a test.It jumps to the specified location if the Zero Flag (ZF) is set (1). jz is commonly used to explicitly test for something being equal to zero whereas je is commonly found after a cmp instruction.

**Syntax:**

jz location

**6. CMP Instruction:**

The cmp instruction is used to perform comparison. It's identical to the sub instruction except it does not affect operands. It impacts the Zero Flag (ZF) as well as the Carry Flag (CF)

**Syntax:**

cmp destination, source

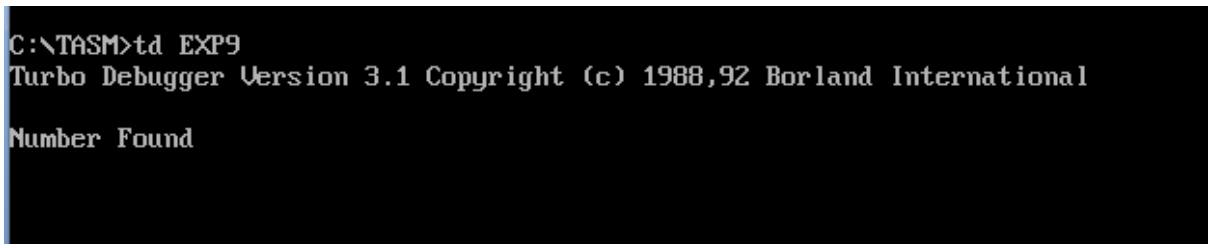
**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

**Code:**

```
assume cs:code,ds:data
data segment
a1 db 10H,34H,13H,78H,56H
msg1 db 0AH,0DH,'Number Found$'
msg2 db 0AH,0DH,'Number Absent$'
num db 10H
data ends
code segment
start:mov Ax,data
    mov Ds,Ax
    mov Cx,05H
    lea Si,a1
    mov AH,num
to :  mov Al,[Si]
    CMP Al,AH
    JZ me
    INC Si
    DEC Cx
    JNZ to
    lea Dx,msg2
    mov AH,09H
    int 21H
    JMP last
me :  lea Dx,msg1
    mov AH,09H
    int 21H
last: mov AH,4CH
    int 21H
code ends
 end start
```

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 30/3/22**

## Output:





## Conclusion:

From this experiment we learned about inc and dec commands and also how to search for a number

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

# EXPERIMENT 10

**Aim:** Compute the factorial of a positive integer 'n' using recursive procedure

**LO:** 5

**LO STATEMENT:** Write programs based on string and procedure for 8086 microprocessor

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**
Move Destination, Source

**Example:**
MOV Ax, Bx

### 2. INT instruction:

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (lnterrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

**Example:**

INT 21H

### 3. INC Instruction:

The INC instruction adds one to the destination operand, while preserving the state of the carry flag CF. The destination operand can be a register or a memory location. This instruction allows a loop counter to be updated without disturbing the CF flag.

**Syntax:**

INC destination

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

**4. JMP Instruction:**

Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward, to execute a new set of instructions or backward, to re-execute the same steps. The JMP instruction provides a label name where the flow of control is transferred immediately.

**Syntax:**

JMP label

**Example:**

JMP next

**5. LEA Instruction:**

LEA is Used to load the address of operand into the provided register. LES Used to load ES register and other provided register from the memory. The lea instruction places the address specified by its first operand into the register specified by its second operand. Note, the contents of the memory location are not loaded, only the effective address is computed and placed into the register.

**6. CMP Instruction:**

The cmp instruction is used to perform comparison. It's identical to the sub instruction except it does not affect operands. It impacts the Zero Flag (ZF) as well as the Carry Flag (CF)

**Syntax:**

cmp destination, source

**7. JNZ instruction:**

The jnz (or jne) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). jnz is commonly used to explicitly test for something not being equal to zero whereas jne is commonly found after a cmp instruction.

**Syntax:**

jnz location

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

## Code:

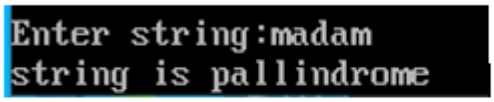
```
assume ds:data , cs:code

data segment

n1 db 0ah,0dh,'Enter string:$'

n2 db 0ah,0dh,'string is pallindrome$'

n3 db 0ah,0dh,'string is not pallindrome$'

buff db 06h

    db 00h

    db 50h

data ends

print macro msg

mov ah,09h

lea dx,msg

int 21h

endm

code segment

start:

Mov ax,data

mov ds,ax

print n1

mov ah,0ah

lea dx,buff

int 21h

lea bx,buff+2

mov ch,00h

mov cl,buff+1
```

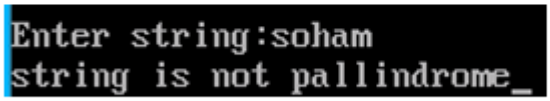
**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

```
mov di,cx

dec di

mov si,00h

shr cl,01h

back:

mov al,[bx+si]

mov ah,[bx+di]

cmp al,ah

jnz last

dec di

inc si

dec cx

jnz back

print n2

jmp final

last:

print n3

final:

mov ah,4ch

int 21h

code ends

end start
```

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

**Output:**




**Conclusion:**

From this experiment we have studied have to use various instructions like jnz and buff and also how to find out if a string is palindrome or not in 8086.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

# EXPERIMENT 11

**Aim:** Compute the factorial of a positive integer 'n' using recursive procedure

**LO:** 5

**LO STATEMENT:** Write programs based on string and procedure for 8086 microprocessor

## Software and Hardware Requirements: TASM Software

## Theory:

### 1. MOV Instruction

The MOV instruction is the most important command in the 8086 because it moves data from one location to another. It also has the widest variety of parameters; so the assembler programmer can use MOV effectively, the rest of the commands are easier to understand. MOV copies the data in the source to the destination. The data can be either a byte or a word. Sometimes this has to be explicitly stated when the assembler cannot determine from the operands whether a byte or word is being referenced.

**Syntax:**
Move Destination, Source

**Example:**
MOV Ax, Bx

### 2. INT instruction:

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (lnterrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

**Example:**

INT 21H

### 3. LOOP instruction:

Loop instructions are Used to simplify the decrementing. testing and branching portion of the loop .A loop instruction is used to loop a group of instructions until the condition satisfies, i.e., CX = 0. To get the loop instruction to work first you have to define a label, set the value in cx which would be the number of times the loop should execute.

**Example:**

LOOP back

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

### 4. Proc near Instruction:

A near procedure refers to a procedure which is in the same code segment from that of the call instruction It is also called intra-segment procedure A near procedure call replaces the old IP with new IP

**Example:**

CALL fact

### 5. RET Instruction:

The ret instruction transfers control to the return address located on the stack. This address is usually placed on the stack by a call instruction. Issue the ret instruction within the called procedure to resume execution flow at the instruction following the call.

**Syntax:**

RET

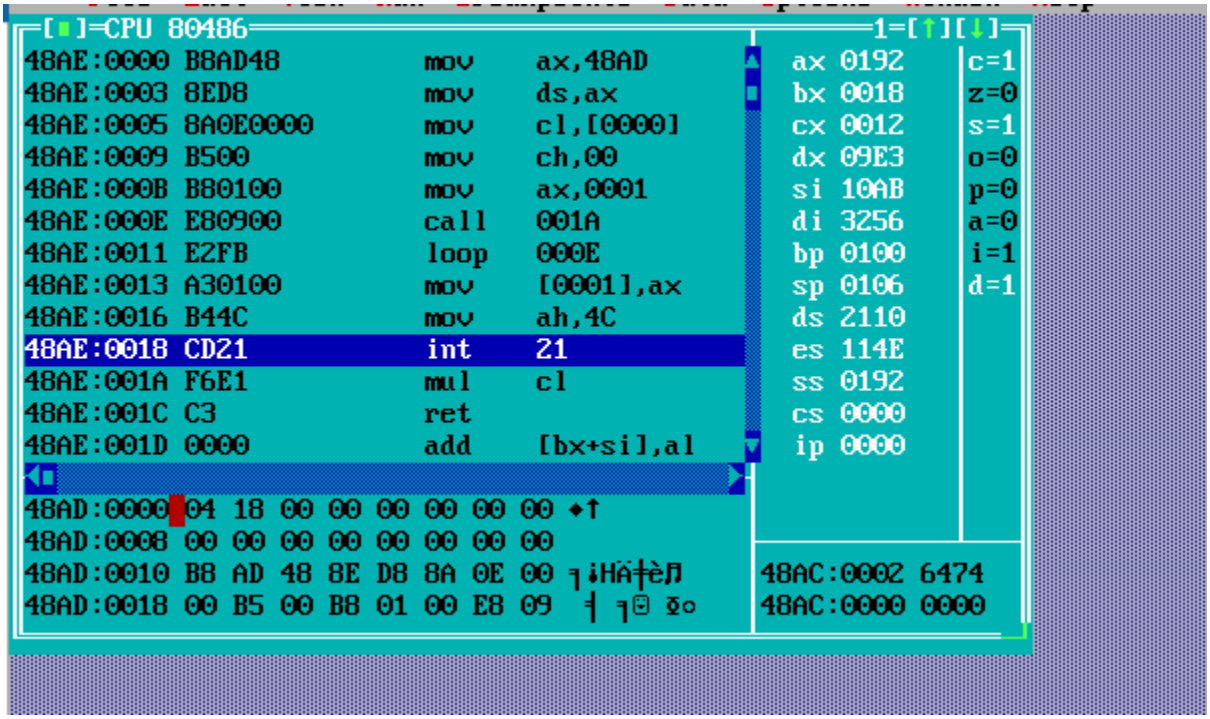
### Code:

```
assume cs:code,ds:data

data segment

num db 04H

ans dw ?

data ends

code segment

start:

    mov Ax,data

    mov Ds,Ax

    mov Cl,num

    mov Ch,00H

    mov Ax,0001H
```

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

```
back:
    call fact
    LOOP back
    mov ans,Ax
    mov AH,4CH
    INT 21H
fact proc near
   MUL Cl
   RET
   fact ENDP
code ends
end start
```

**Output:**




**Conclusion:** From this experiment we have learned about how to use procedure and call then in 8086 assembly language

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

# EXPERIMENT 12

**Aim:** Interfacing Seven Segment Display

**LO:** 6

**LO STATEMENT:** Design interfacing of peripheral devices with 8086 microprocessor.

**Software and Hardware Requirements:** TASM Software

## Theory:

A seven-segment LED is a kind of LED (Light Emitting Diode) consisting of 7 small LEDs it usually comes with the microprocessor's as we commonly need to interface them with microprocessors like 8086.

It can be used to represent numbers from 0 to 8 with a decimal point. We have eight segments in a Seven Segment LED display consisting of 7 segments which include '.' The seven segments are denoted as "a, b, c, d, e, f, g, h" respectively, and "." is represented by "h "

Alphanumeric LED displays are available in three common formats. For displaying only numbers, letters and hexadecimal letters, simple 7 segment displays are used. To display numbers and the entire alphabet 18 segment displays or 5 by 7 dot matrix displays can be used. The 7 segment type is the least expensive, most commonly used and easiest to interface.

Seven segment indicators are of two type's namely common anode type and common cathode type. In a common anode type all the anodes are connected together

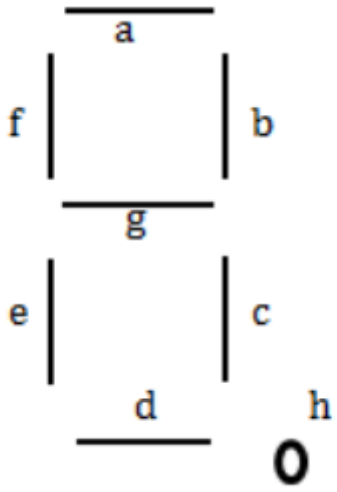
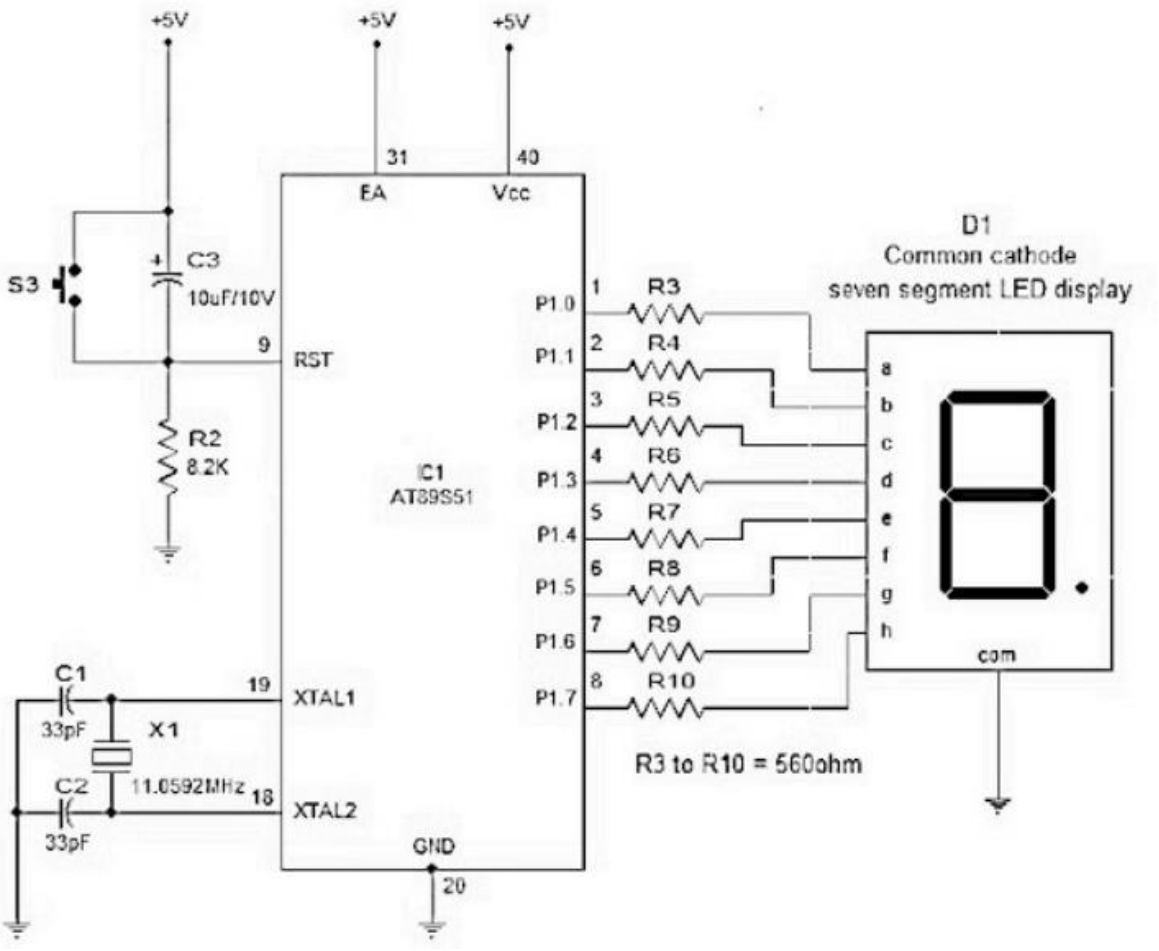


Fig.1 Seven segment indicator

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

Seven-segment displays may use a liquid crystal display (LCD), a light-emitting diode (LED) for each segment, an electrochromic display, or other light-generating or controlling techniques such as cold cathode gas discharge (Panaplex), vacuum fluorescent (VFD), incandescent filaments (Numitron), and others. For gasoline price totems and other large signs, vane displays made up of electromagnetically flipped light-reflecting segments (or "vanes") are still commonly used. A precursor to the 7-segment display in the 1950s through the 1970s was the cold-cathode, neon-lamp-like nixie tube. Starting in 1970, RCA sold a display device known as the *Numitron* that used incandescent filaments arranged into a seven-segment display. In USSR, the first electronic calculator "Vega", which was produced from 1964, contains 20 decimal digits with seven-segment electroluminescent display.

Each LED segment has one of its pins brought out of the rectangular package. Other pins are connected together to a common terminal. Seven segment displays can only display O to 9 numbers. These seven LEDs indicate seven segments of the numbers and a dot point Seven segment displays are seen associated with a great number of devices such as clocks, digital home appliances, signal boards on roads etc.



**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 6/4/22**

**Sample Code:**

```
MOV AL,10000000B

OUT PPI_C, AL

STR: MOV AL, 10011001B

OUT PPIA,AL

MOV CX, FFFFH

DEL: NOP

NOP

NOP

LOOP DEL

MOV AL, 11111111B

OUT PPIA, AL

MOV CX, FFFFH

DEL1: NOP

NOP

NOP

LOOP DEL1

JMP STR

HLT
```

**Conclusion:** From this experiment we have learned about the 7 segment display and how it works with the 8086 microprocessor language.

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/4/22**

# COA CASE STUDY

**Aim:** Study of Flip-flop

**LO:** 6

**LO STATEMENT:** Design interfacing of peripheral devices with 8086 microprocessor.

**Software and Hardware Requirements:** TASM Software

## Theory:

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously.

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs.

There are basically four different types of flip flops and these are:

1. Set-Reset (SR) flip-flop or Latch.

2. JK flip-flop.

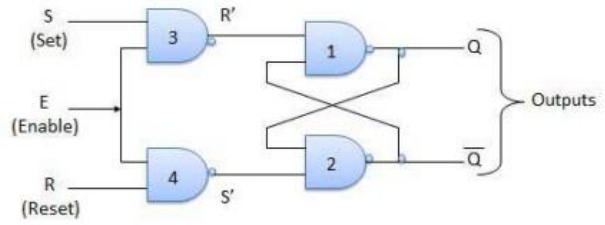
3. D (Data or Delay) flip-flop.

4. T (Toggle) flip-flop.

**S-R Flip Flop :**

It is basically S-R latch using NAND gates with an additional enable input. For this circuit, output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

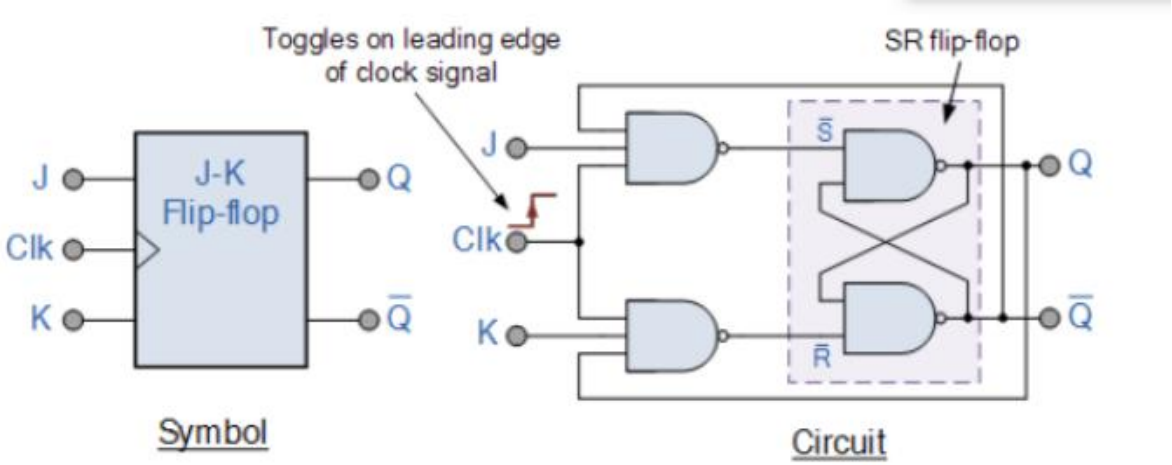
Circuit diagram



**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/4/22**

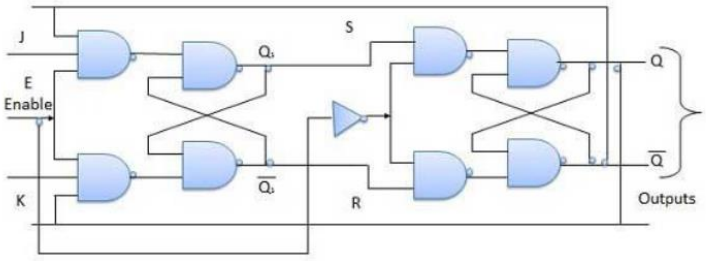
**JK Flip-flop :**

The JK flip flop is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1". Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle"




**Master Slave JK Flip Flop** :

Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.



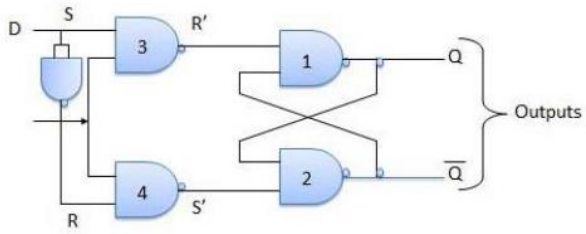

| Inputs | | | Outputs | | Comments |
|---|---|---|---|---|---|
| E | J | K | $Q_{n+1}$ | $\overline{Q}_{n+1}$ | |
| 1 | 0 | 0 | $Q_n$ | $\overline{Q}_n$ | No change |
| 1 | 0 | 1 | 0 | 1 | Rset |
| 1 | 1 | 0 | 1 | 0 | Set |
| 1 | 1 | 1 | $\overline{Q}_n$ | $Q_n$ | Toggle |

**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/4/22**

**Delay Flip Flop / D Flip Flop :**

Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. • The input data is appearing at the output after some time. Due to this data delay between i/p and o/p, it is called delay flip flop. • S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoided by SR = 00 and SR = 1 conditions.

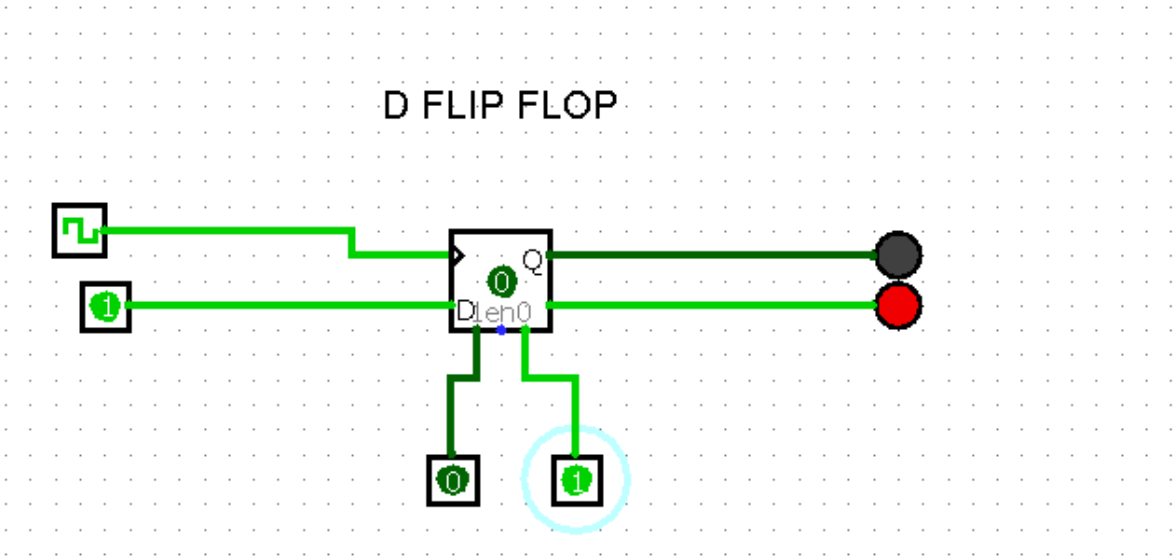
Circuit diagram



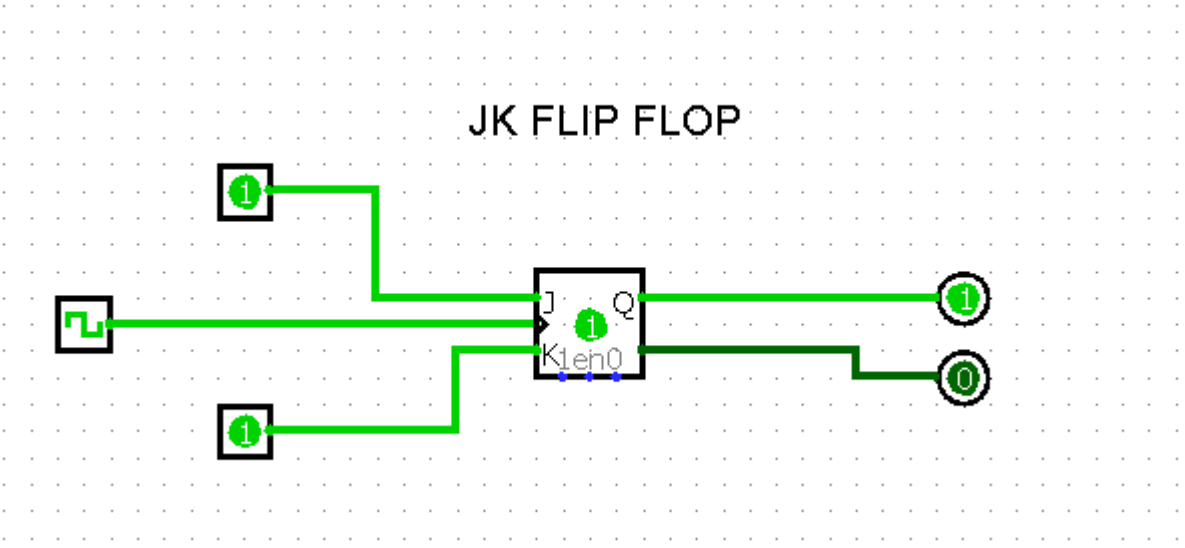

Truth Table

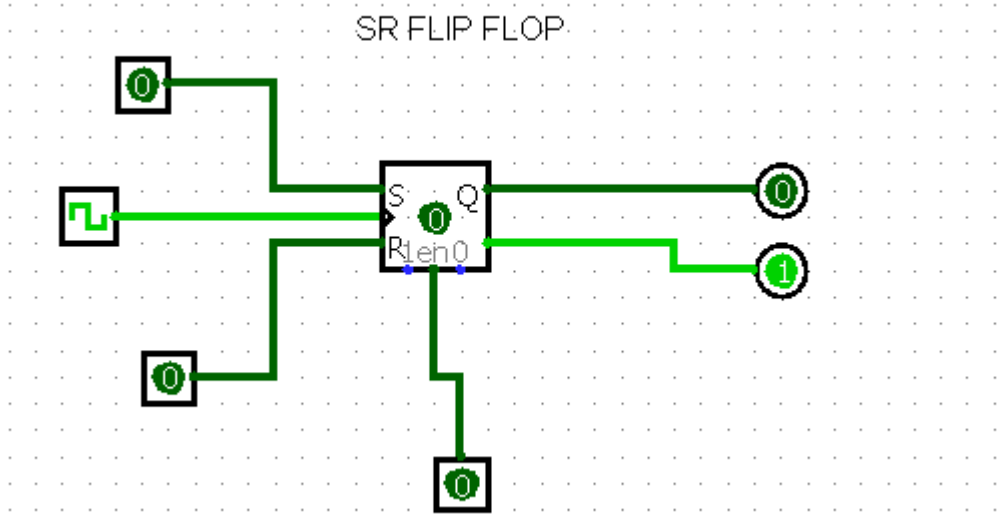
| Inputs | | Outputs | | Comments |
|---|---|---|---|---|
| E | D | $Q_{n+1}$ | $\overline{Q}_{n+1}$ | |
| 1 | 0 | 0 | 1 | Rset |
| 1 | 1 | 1 | 0 | Set |

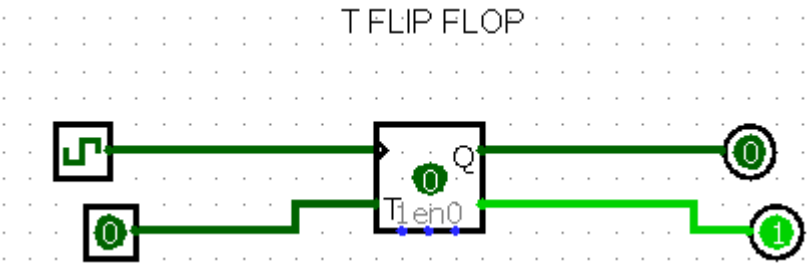
**Logism Outputs:**



**Roll No: 11**
**Name: Soham Desai**
**Xavier ID : 202003021**
**Date: 21/4/22**




**Conclusion:** From this study we have learned about the different types of flip flops and also how to design them in Logism software.