THADOMAL SHAHANI ENGINEERING COLLEGE

Bandra (W), Mumbai 400-050

CERTIFICATE

Certify that **Mr. ABHAY RANGNANI** of Department of Information Technology, **Semester IV** with Roll number **78** of batch **S21** has satisfactorily completed a course of the necessary experiments in the subject **MicroProcessor Lab** under my supervision in Thadomal Shahani Engineering College in the academic year 2020-21.

Name of faculty: Reshma Malik

Date: 07-05-2021

INDEX

	language programming	2. Compute the factorial of a positive 66integer ‘n’ using procedure. OR Generate the first ‘n’ Fibonacci numbers.	59
6	Interfacing with 8086 microprocessors	Interfacing Seven Segment Display 2. Interfacing keyboard matrix 3. Interfacing DAC (any one)	61

ASSIGNMENT 1

AIM:

PC Assembly:

- Study of PC Motherboard Technology (south Bridge and North Bridge), Internal Components and Connections used in computer systems.

THEORY:

A computer is an electronic data processing device, which accepts and stores data input, processes the data input, and generates the output in a required format.

Basic components required in PC assembly are:

Case

Motherboard

CPU [Processor]

GPU [Graphics Card] (if no integrated GPU)

RAM [Memory]

Storage Device (SSD, NVME SSD, HDD)

Cooling (CPU, Chassis)

PSU [Power Supply Unit]

Display device, Monitor

Operating System [OS]

Input Devices, Mouse, Keyboard

Input Unit:

This unit contains devices with the help of which we enter data into the computer. This unit creates a link between the user and the computer. The input devices translate the information into a form understandable by the computer.

For eg. Keyboard, mouse, pen or stylus, microphone for voice as input, webcam, etc.

CPU (Central Processing Unit):

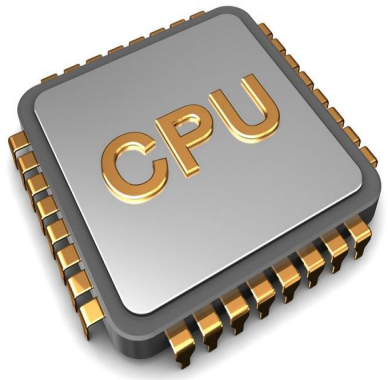
CPU is considered as the brain of the computer. CPU performs all types of data processing operations. It stores data, intermediate results, and instructions (program). It controls the operation of all parts of the computer.

CPU itself has the following three components −

ALU (Arithmetic Logic Unit)

Memory Unit

Control Unit



Output Unit:

The output unit consists of devices with the help of which we get the information from the computer. This unit is a link between the computer and the users. Output devices translate the computer's output into a form understandable by the users.

For eg. Monitor, printers, speakers, etc.

Memory:

Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into a large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory locations. The address of these locations varies from 0 to 65535.

Memory is primarily of three types −

Cache Memory

Primary Memory/Main Memory

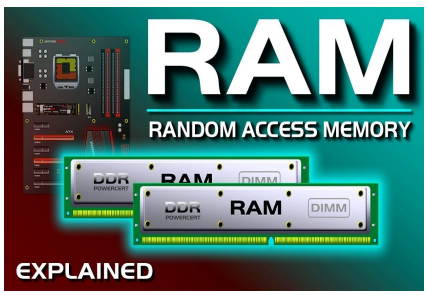
Secondary Memory

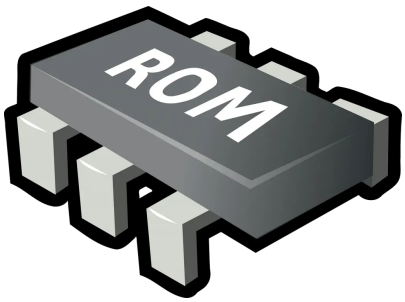
Cache Memory

Cache memory is a very high-speed semiconductor memory which can speed up the CPU. It acts as a buffer between the CPU and the main memory. It is used to hold those parts of data and program which are most frequently used by the CPU. The parts of data and programs are transferred from the disk to cache memory by the operating system, from where the CPU can access them.

Primary Memory (Main Memory):

Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off. It is generally made up of semiconductor device. These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.





Secondary Memory:

This type of memory is also known as external memory or non-volatile. It is slower than the main memory. These are used for storing data/information permanently. CPU directly does not access these memories, instead they are accessed via input-output routines. The contents of secondary memories are first transferred to the main memory, and then the CPU can access it. For example, disk, CD-ROM, DVD, etc.

MOTHERBOARD:

Features of motherboard:

The features of a computer motherboard are as follows:

The motherboard acts as the central backbone of a computer on which other modular parts are installed such as the CPU, RAM and hard disks.

The motherboard also acts as the platform on which various expansion slots are available to install other devices / interfaces.

The motherboard is also responsible to distribute power to the various components of the computer.

They are also used in the coordination of the various devices in the computer and maintain an interface among them.

Some of the Sizes in which the motherboards are available are : BTX, ATX, mini-ATX, micro-ATX, LPX, NLX etc..

Description of motherboard:

A motherboard (also called mainboard, main circuit board, system

board, baseboard, planar board, logic board, or mobo) 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.

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.

Similarly, the term mainboard describes a device with a single board and no additional expansions or capability, such as controlling boards in laser printers, television sets, washing machines, mobile phones, and other embedded systems with limited expansion abilities.

Components:

Mouse & keyboard

USB

Parallel port

CPU Chip

RAM slots

Floppy controller

IDE controller

PCI slot

ISA slot

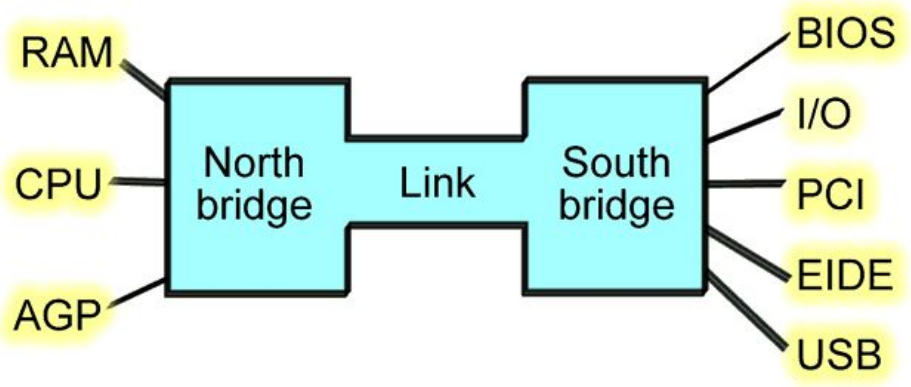
CMOS Battery

AGP slot

CPU slot

Power supply plug in

NORTH BRIDGE AND SOUTH BRIDGE:



The main difference between northbridge and southbridge is that the northbridge is a chip in the chipset of a motherboard that directly connects to the CPU while the southbridge is a chip in the chipset of a motherboard that does not directly connects to the CPU.

Modern computers are electronic devices that are designed to perform multiple tasks. There are various components in a computer. The CPU or the processor manages the functionalities of the other components. A motherboard is a Printed Circuit Board (PCB) that allows communication between various components of the system. A motherboard consists of a chipset. Northbridge and southbridge are two chips in that chipset. High-speed components are connected to the northbridge while lower speed components are connected to the southbridge.

Characteristics of Ports:

- A port has the following characteristics −
- External devices are connected to a computer using cables and ports.
- Ports are slots on the motherboard into which a cable of external device is plugged in.
- Examples of external devices attached via ports are the mouse, keyboard, monitor, microphone, speakers, etc.

Firewire Port

- Transfers large amounts of data at very fast speed.
- Connects camcorders and video equipment to the computer.
- Data travels at 400 to 800 megabits per second.

- Invented by Apple.
- It has three variants: 4-Pin FireWire 400 connector, 6-Pin FireWire 400 connector, and 9-Pin FireWire 800 connector.

PS/2 Port

- Used for old computer keyboard and mouse
- Also called mouse port
- Most of the old computers provide two PS/2 port, each for the mouse and keyboard
- IEEE 1284-compliant Centronics port

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 printer port
- 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 second.
- USB compliant devices can get power from a USB port.

VGA Port

- Connect monitor to a computer's video card.
- It has 15 holes.
- Similar to the serial port connector. However, the serial port connector has pins, and the VGA port has holes.

Power Connector

- Three-pronged plug.
- Connects to the computer's power cable that plugs into a power bar or wall socket.

Modem Port

- Connects a PC's modem to the telephone network.

Ethernet Port

- Connects to a network and high-speed Internet.
- Connects the network cable to a computer.
- This port resides on an Ethernet Card.
- Data travels at 10 megabits to 1000 megabits per seconds depending upon the network bandwidth.

Game Port

- Connect a joystick to a PC
- Now replaced by USB
- Digital Video Interface, DVI port
- Connects Flat panel LCD monitor to the computer's high-end video graphic cards.
- Very popular among video card manufacturers.

Sockets

- Sockets connect the microphone and speakers to the sound card of the computer.

Power Supply Unit (PSU):

A power supply unit 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.

Switched-Mode Power Supply: It is an electronic power supply that uses a switching regulator to convert electrical power efficiently. It is also known as Switching Mode Power Supply. It is a power supply unit (PSU) generally used in computers to convert the voltage into the computer acceptable range.

This device has the power handling electronic components that converts electrical power efficiently. Switched Mode Power Supply uses a great power conversion technique to reduce overall power loss.

Assignment 02:

AIM:

Implementation of combinational circuits.

Theory:

1. **Verify Truth tables of various logic gates:**

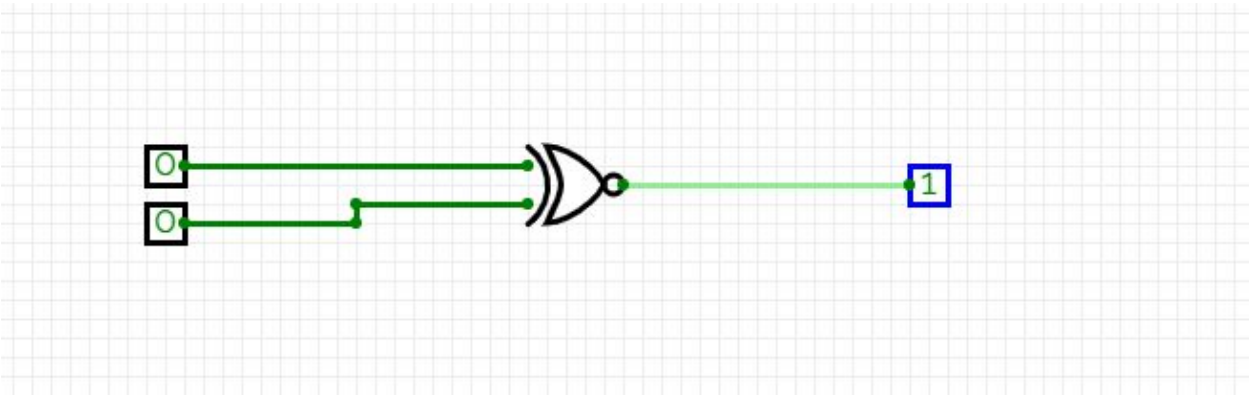
What are logic gates:
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic.

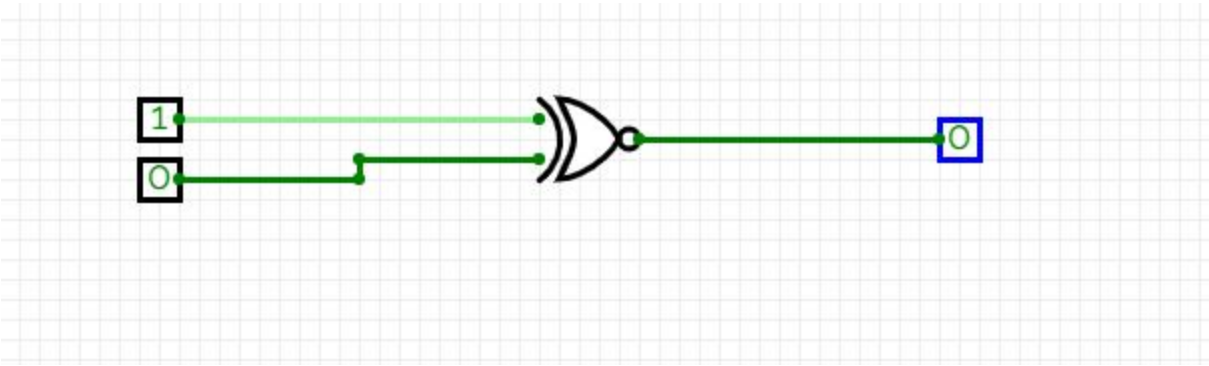
Why do we need Logic Gates:
Logic gates are used to perform various operations like in microprocessors and other electronics devices all electronic devices use logical operations by using binary values.

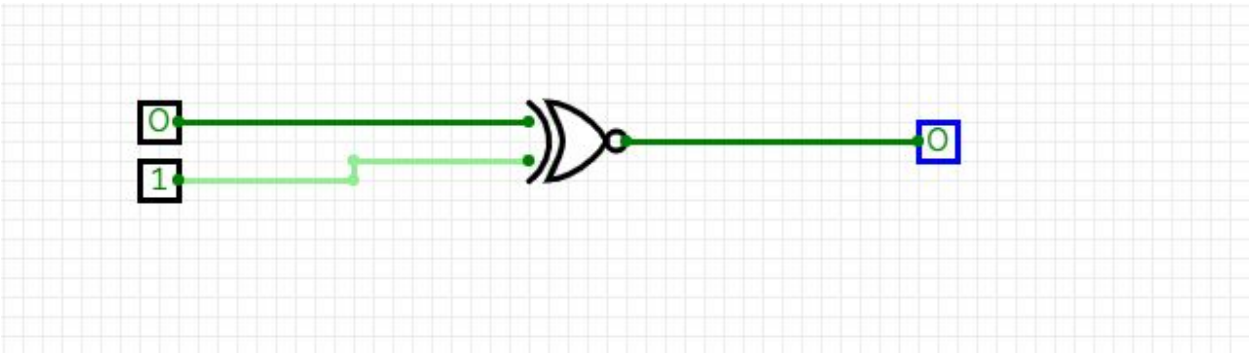
Use of Logic Gates:
Logic gates are primarily implemented using diodes or transistors acting as electronic switches, but can also be constructed using vacuum tubes, electromagnetic relays (relay logic), fluidic logic, pneumatic logic, optics, molecules, or even mechanical elements. With amplification, logic gates can be cascaded in the same way that Boolean functions can be composed, allowing the construction of a physical model of all of Boolean logic, and therefore, all of the algorithms and mathematics that can be described with Boolean logic.

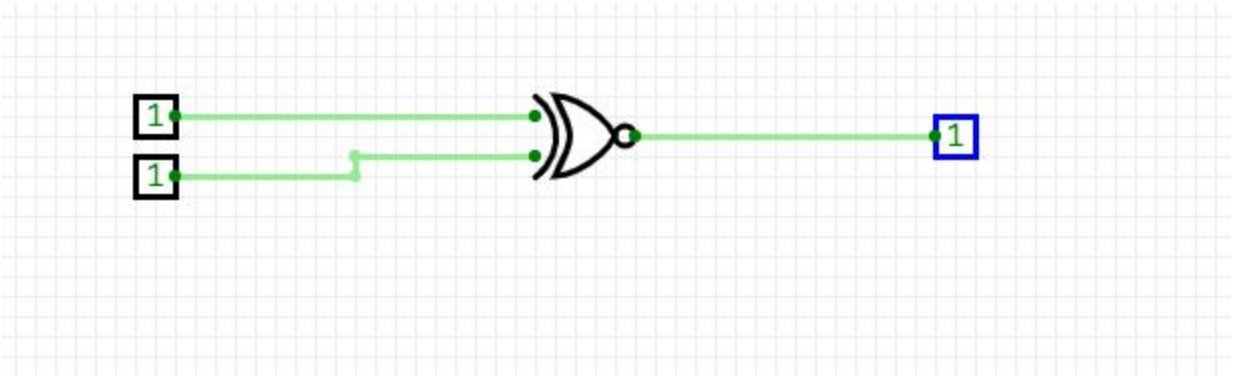
XNOR Gate:

The Exclusive-NOR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-NOR. It performs based on the operation of the NOR gate. When both the inputs of this gate are high, then the output of the EX-NOR gate will be high. But, if any one of the inputs is high (but not both), then the output will be low.







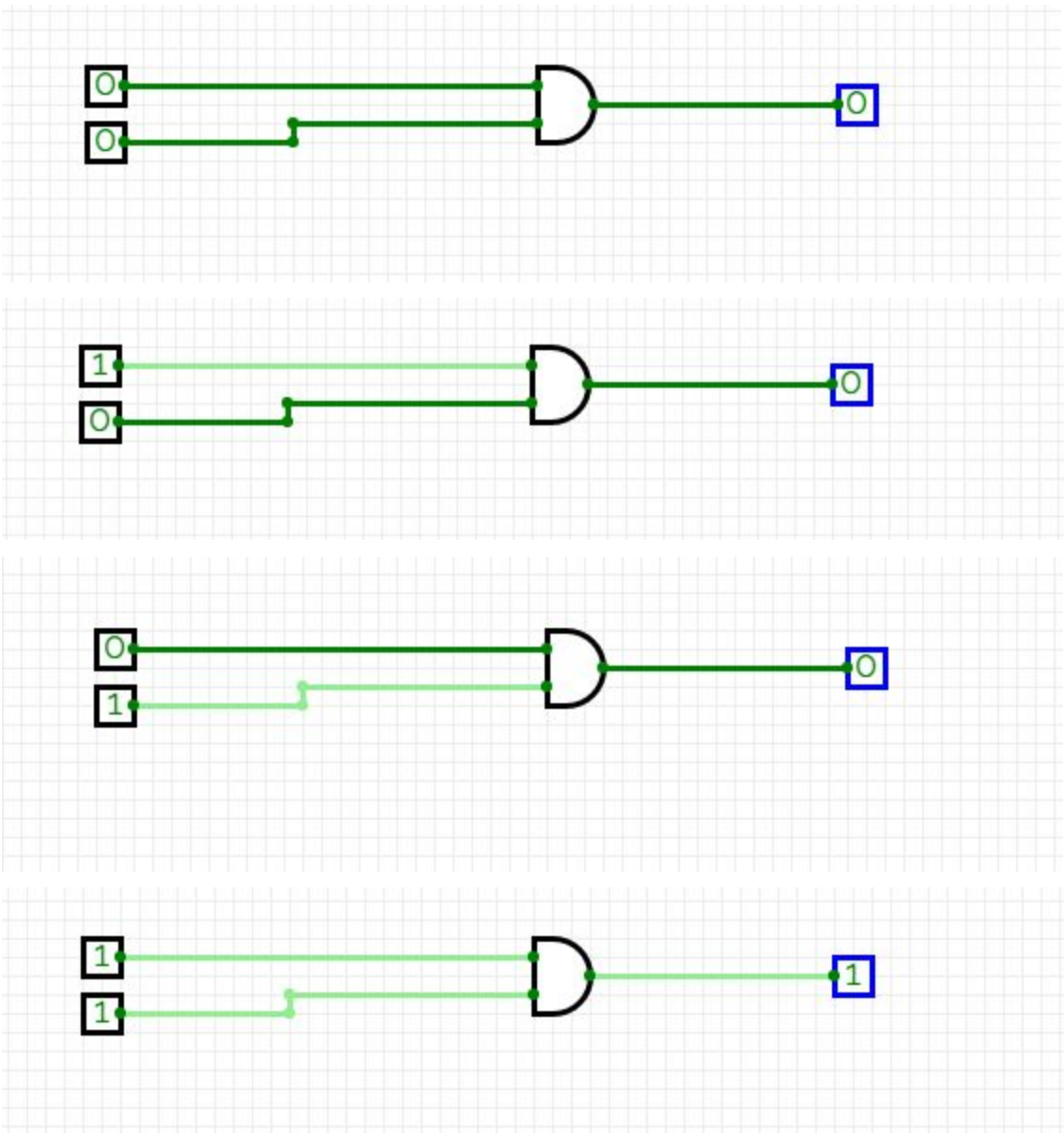


Truth Table of XNOR gate:

INPUT		OUTPUT
A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

AND Gate:

The AND gate is a digital logic gate which performs logical conjunction based on the combinations of its inputs. The output of this gate is true only when all the inputs are true. When one or more inputs of the AND gate's i/ps are false, then only the output of the AND gate is false.

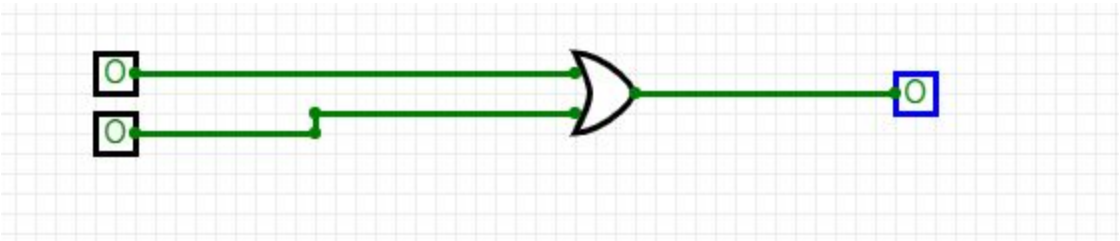


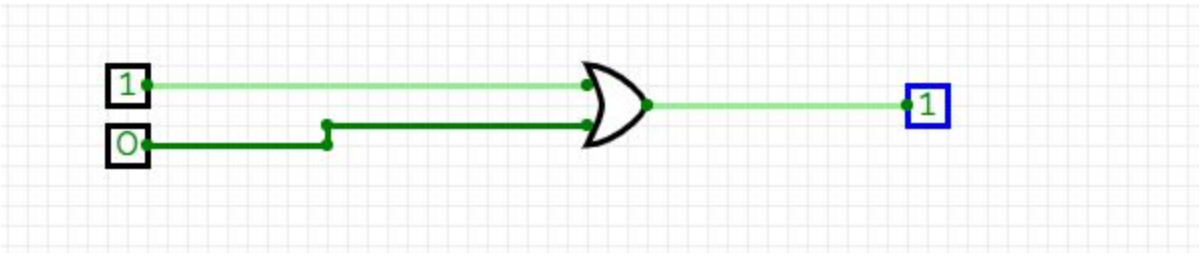
Truth Table of AND Gate:

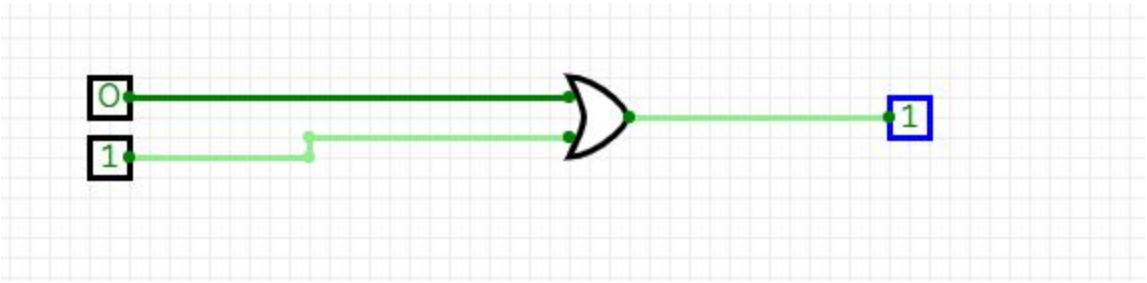
INPUT	INPUT	OUTPUT
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

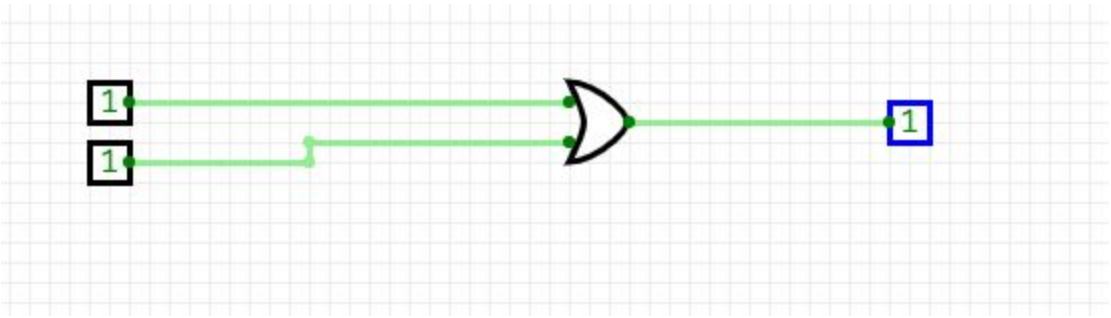
OR Gate:

The OR gate is a digital logic gate with ‘n’ i/ps and one o/p, that performs logical conjunction based on the combinations of its inputs. The output of the OR gate is true only when one or more inputs are true. If all the i/ps of the gate are false, then only the output of the OR gate is false.







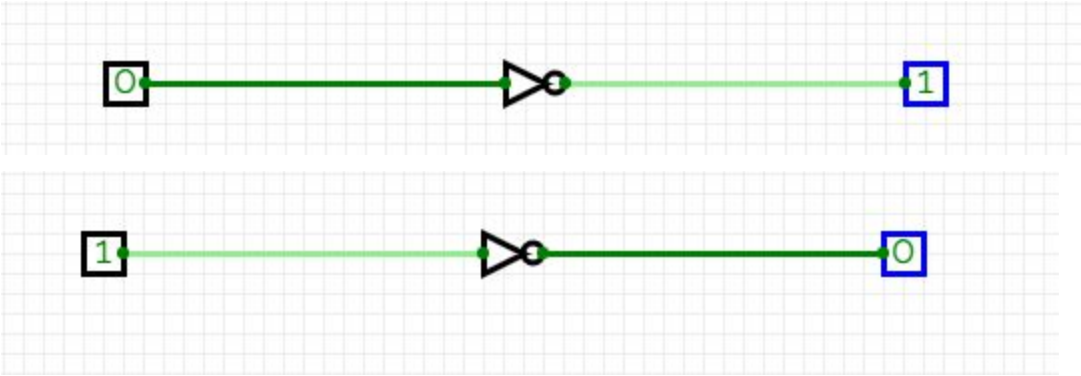


Truth Table of OR Gate:

INPUT		OUTPUT
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate:

The NOT gate is a digital logic gate with one input and one output that operates an inverter operation of the input. The output of the NOT gate is the reverse of the input. When the input of the NOT gate is true then the output will be false and vice versa.

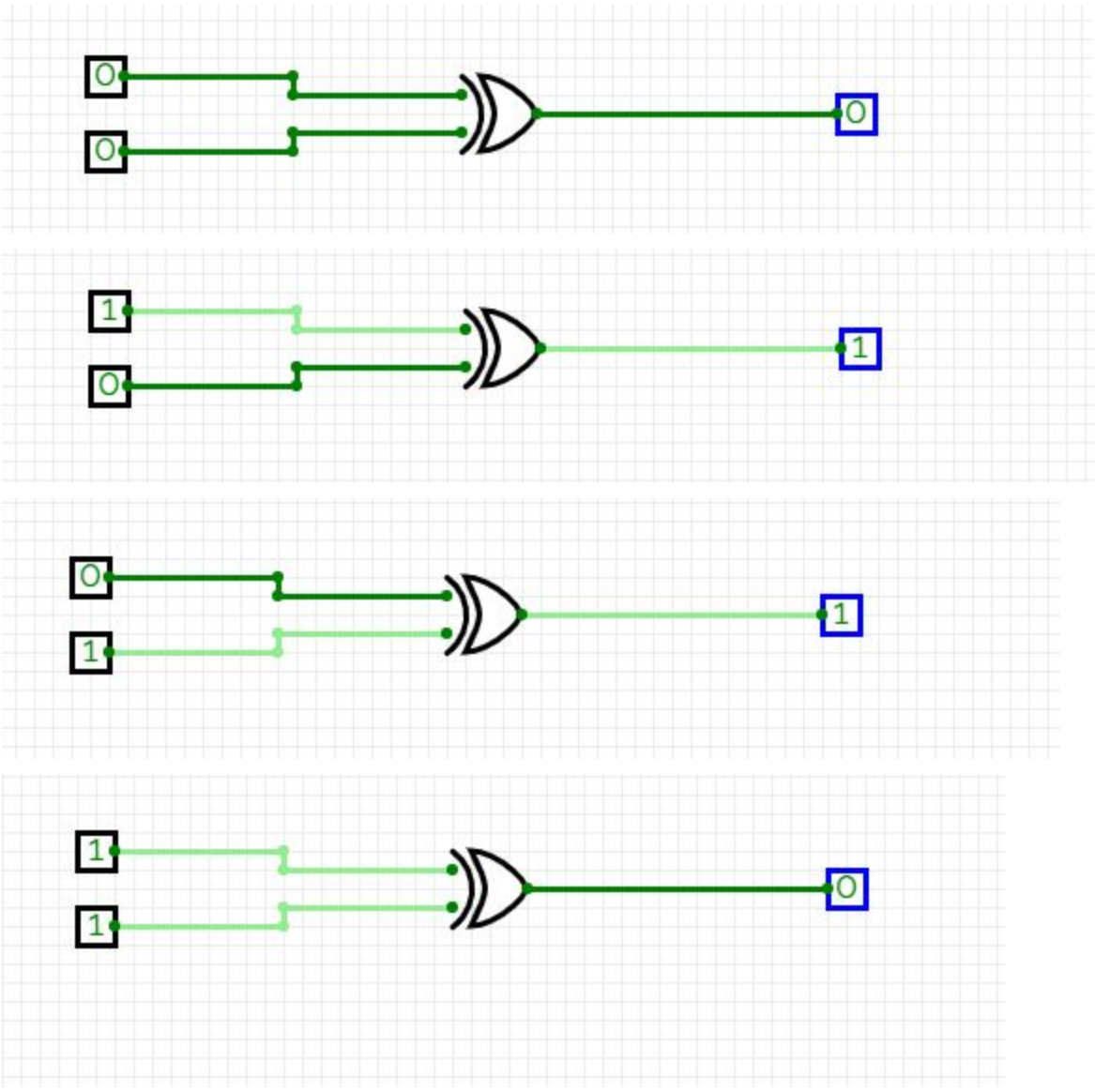


Truth Table of NOT Gate:

INPUT	OUTPUT
A	**O**
0	**1**
1	**0**

XOR Gate:

The Exclusive-OR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-OR. It performs based on the operation of the OR gate. . If any one of the inputs of this gate is high, then the output of the EX-OR gate will be high.

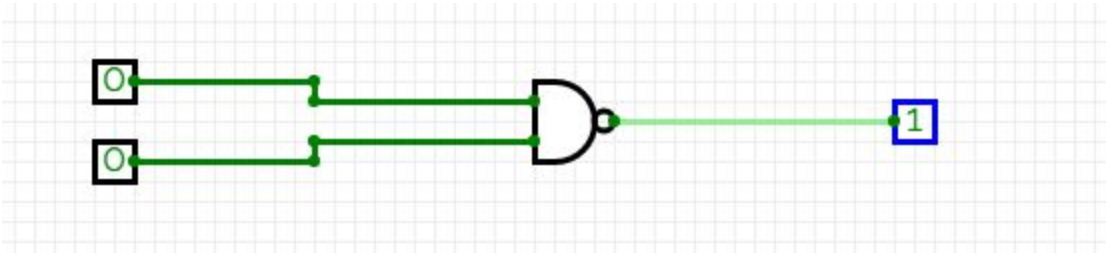


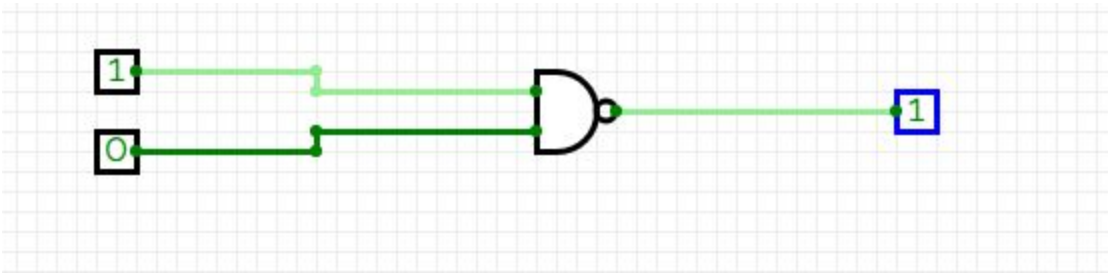
Truth Table of XOR Gate:

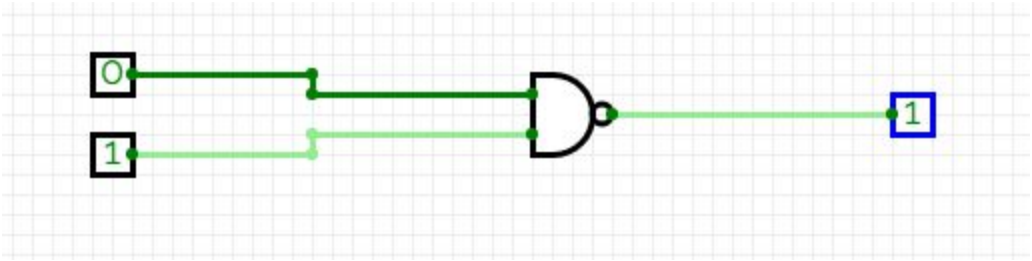
INPUT		OUTPUT
A	**B**	**O**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

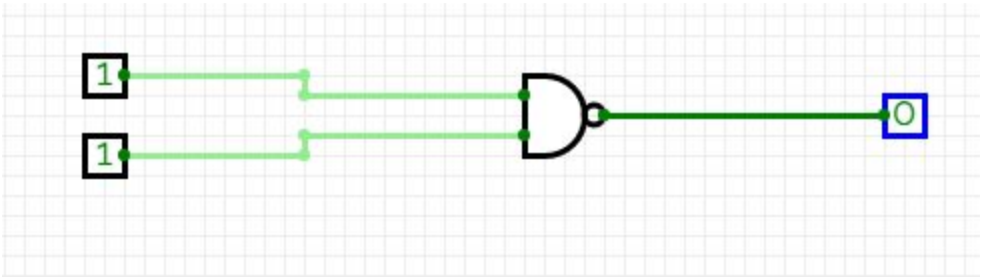
NAND Gate:

The NAND gate is a digital logic gate with ‘n’ i/ps and one o/p, that performs the operation of the AND gate followed by the operation of the NOT gate. NAND gate is designed by combining the AND and NOT gates. If the input of the NAND gate high, then the output of the gate will be low.







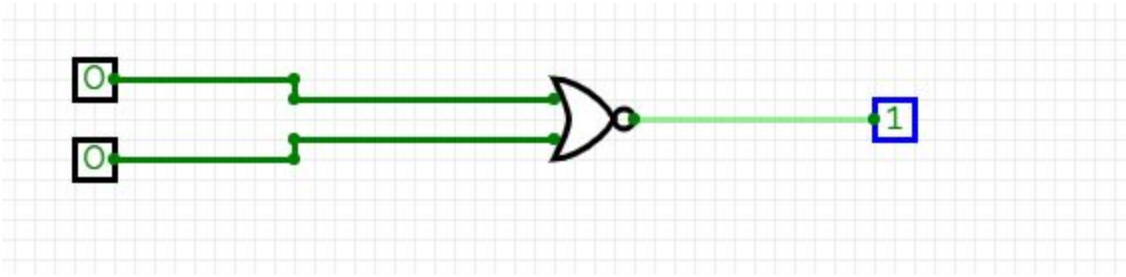


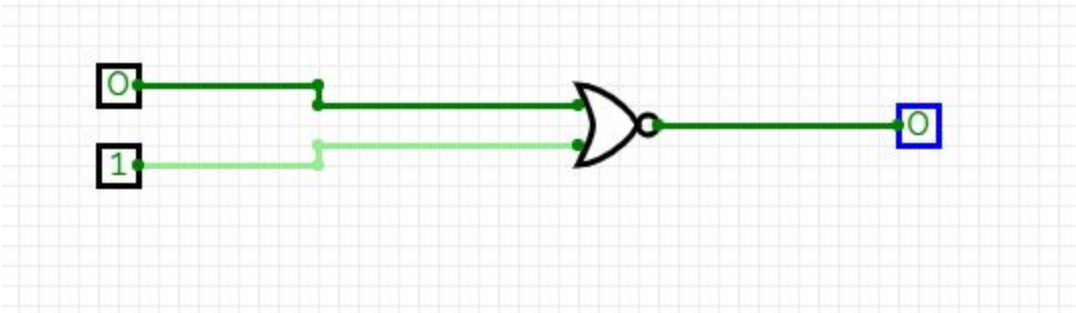
Truth Table of NAND Gate:

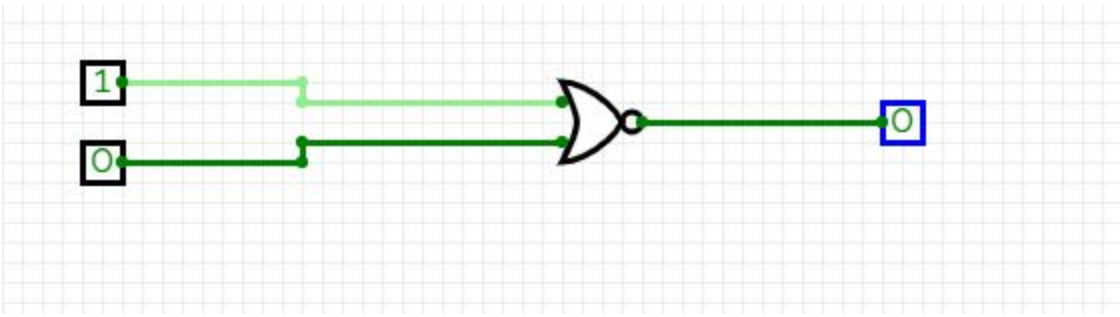
INPUT		OUTPUT
A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

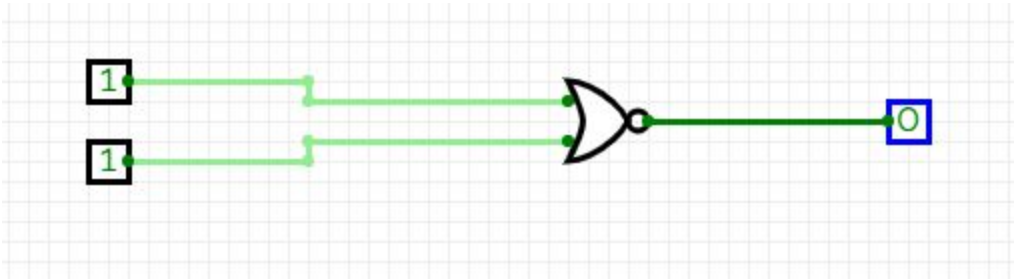
NOR Gate:

The NOR gate is a digital logic gate with n inputs and one output, that performs the operation of the OR gate followed by the NOT gate. NOR gate is designed by combining the OR and NOT gate. When any one of the i/ps of the NOR gate is true, then the output of the NOR gate will be false.









Truth Table of NOR Gate:

INPUT		OUTPUT
A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

CONCLUSION:

The concept of various gates and their truth table is understood and successfully implemented using CircuitVerse - Online digital logic circuit simulator.

Combinational Logic Circuit:

Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs.
Unlike Sequential Logic Circuits whose outputs are dependent on both their present inputs and their previous output state giving them some form of Memory. The outputs of Combinational Logic Circuits are only determined by the logical function of their current input state, logic “0” or logic “1”, at any given instant in time.
Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are “combined” or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

Example:
Half Adders, Full Adders, Multiplexers, Demultiplexers, Encoders and Decoders.

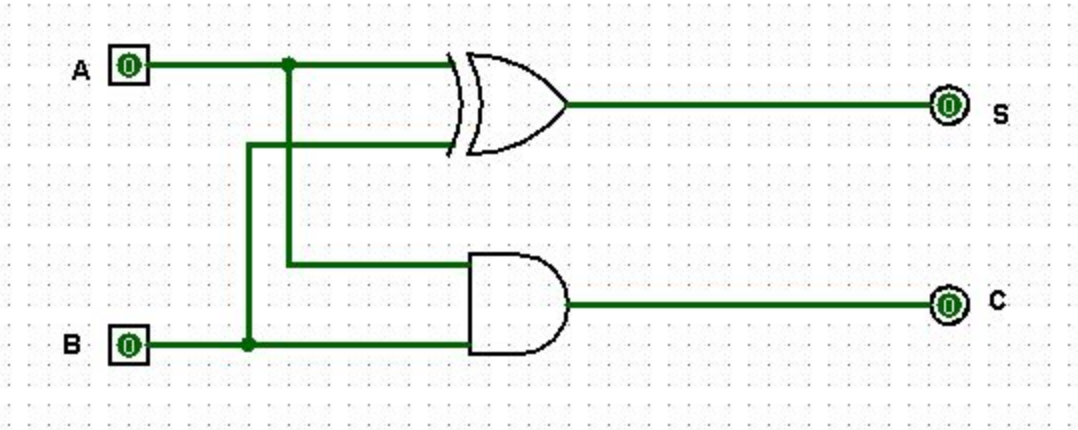
2. Realize Half Adder and Full Adder:

Half Adder:
A Half Adder is defined as a basic four terminal digital device which adds two binary input bits. It outputs the sum binary bit and a carry binary bit.
A half adder is a simple digital circuit used to digitally add two binary bits. A binary bit is either 0 or 1. Hence, there will be four additional combinations of these two binary digits and those will be 0 + 0, 0 + 1, 1 + 0 and 1 + 1.
Binary two is the smallest double digits number in the binary system. When we add binary 1 with binary 1 we will get both sum and carry since 10 is two digits binary number. When we add 0 to 0, 0 to 1 and 1 to 0, we get the sum 0 and 1 respectively and both of them are one digit binary numbers. Hence, in these three cases there will be no carry during addition or carry is 0 here. We can summarise this in a truth table for the half adder.

Truth table of Half Adder:

A	**B**	**Sum**	**Carry**
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Design:



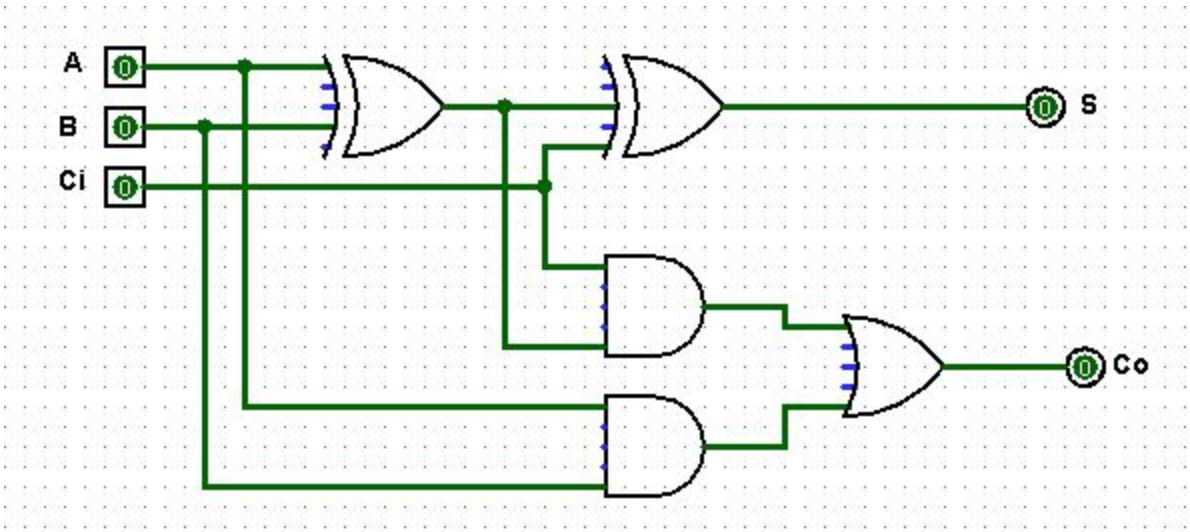
Full Adder:

The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas a half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

Truth table of Full Adder:

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

Design:



CONCLUSION:

The concept of half adder and full adder is understood and successfully implemented using Logisim software.

3. Implementation of MUX and DEMUX:

MUX:

Multiplexer is a combinational circuit that has a maximum of 2n data inputs, ‘n’ selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are ‘n’ selection lines, there will be 2n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called a Mux.

Application of Multiplexer:

A. Communication System. A communication system has both a communication network and a transmission system.

B. Computer Memory.

C. Telephone Network.

D. Transmission from the Computer System of a Satellite.

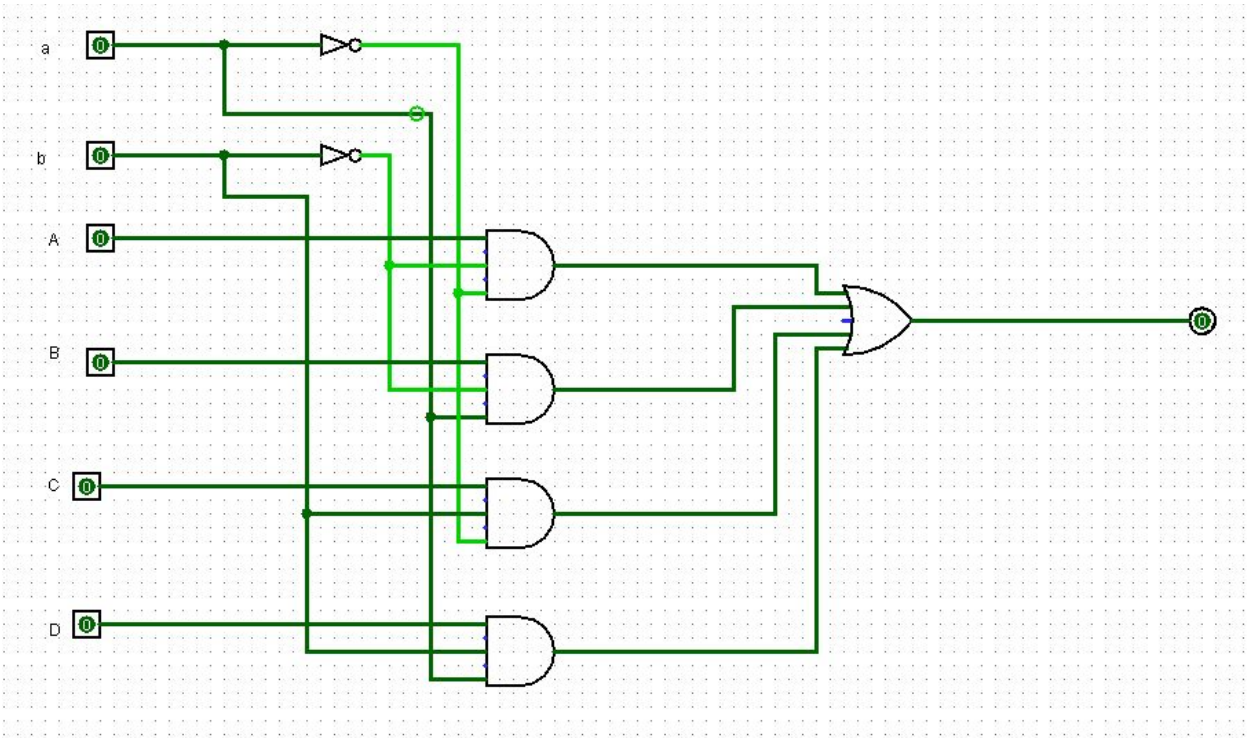
E. Communication System.

F. Arithmetic Logic Unit.

G. Serial to Parallel Converter.

H. Photo Credits.

Design:



Truth Table of Multiplexer:

Selection Lines		Output
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

DEMUX:
De-Multiplexer is a combinational circuit that performs the reverse operation of a Multiplexer. It has single input, ‘n’ selection lines and maximum of 2n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are ‘n’ selection lines, there will be 2n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called Demux.

Application of Demultiplexer:

A. In multiplexer, the usage of a number of wires can be decreased

B. It reduces the cost as well as the complexity of the circuit

C. The implementation of a number of combination circuits can be possible by using a multiplexer

D. Mux doesn’t require K-maps & simplification

E. The multiplexer can make the transmission circuit less complex & economical

F. The dissipation of heat is less because of the analog switching current which ranges from 10mA to 20mA.

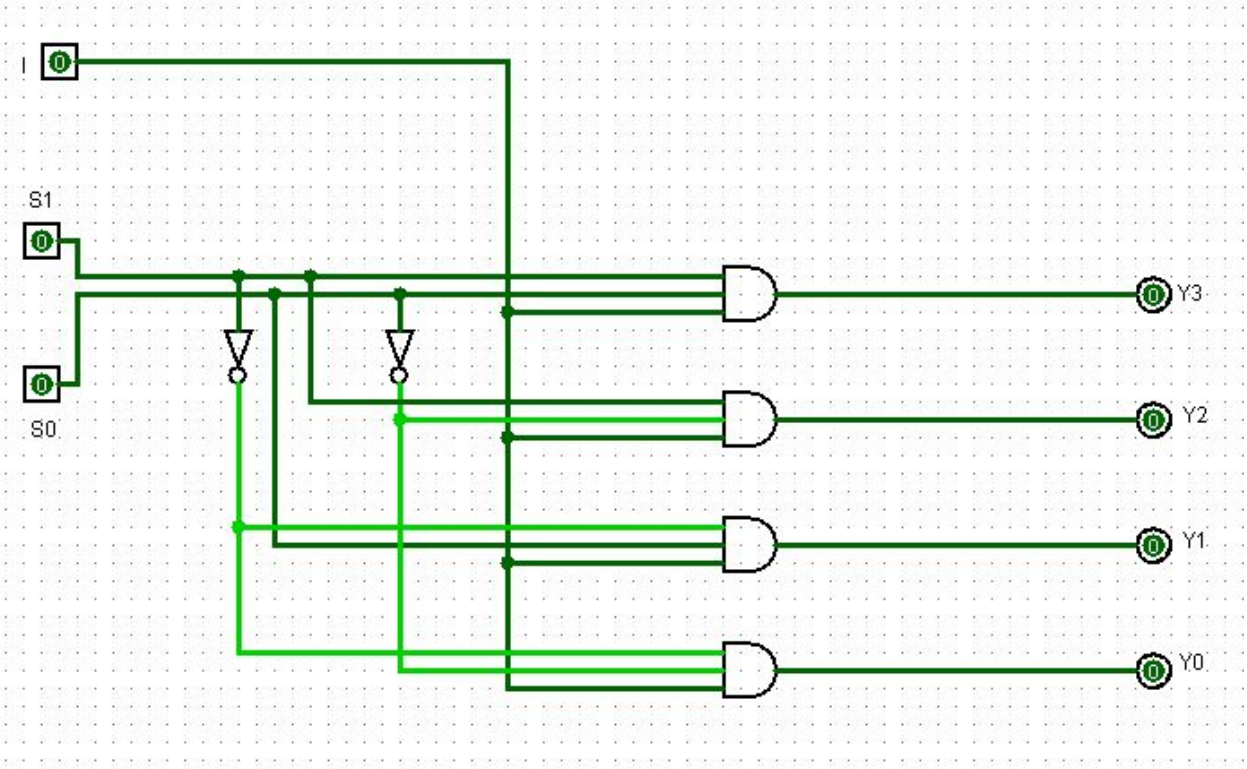
G. The multiplexer ability can be extended to switch audio signals, video signals, etc.

H. The digital system reliability can be improved using a MUX as it decreases the number of exterior wired connections.

I. MUX is used to implement several combinational circuits

J. The logic design can be simplified through MUX

Design:



Truth Table of Demultiplexer:

Selection	Lines			Output	
s_1	s_0	Y_3	Y_2	Y_1	Y_O
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

CONCLUSION:

The concept of Multiplexer and Demultiplexer is understood and successfully implemented using Logisim software.

Assignment 3.A

Arithmetic and logical operations in 8086 Assembly language programming

AIM:
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 alongwith 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 :

```
    2  9
+   3  8
    6  1   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 nibbles of addition is greater than 9, so add 6.

```
    7  D
+   0  6
    8  3
```

Final result : 8367 BCD.

Algorithm :

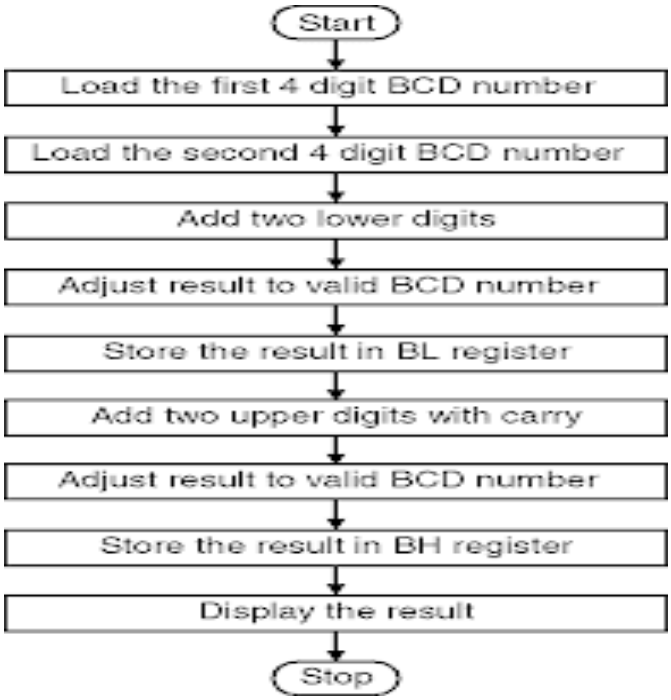
Step I : Initialize the data memory.
Step II : Load the first number into AX register.
Step III : Load the second number into BX register.
Step IV : Add two lower digits.
Step V : Adjust result to valid BCD number.
Step VI : Store the result in BL.
Step VII : Add the two upper digits with carry.
Step 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
l2: 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   l4
    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
```

Result :

```
C:\TASM>tasm 16bitadd.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   16bitadd.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k


C:\TASM>tlink 16bitadd.obj
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
Warning: no stack

C:\TASM>16bitadd.exe
8367
```

Assignment 3.B

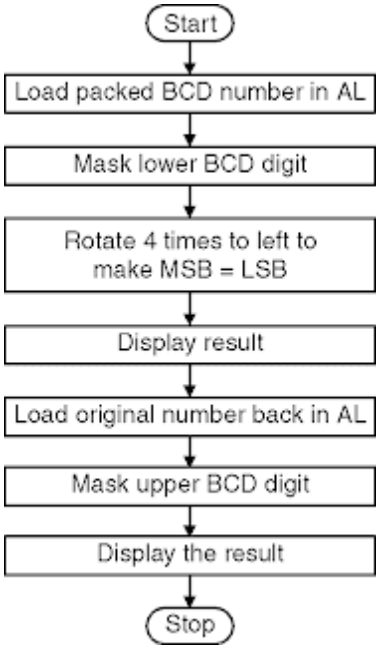
Arithmetic and logical operations in 8086 Assembly language programming

AIM:
Convert two digit Packed BCD to Unpacked 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 be 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 OF0 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
l2:   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
l4:   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
```

OUTPUT:

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

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


C:\TASM>tlink BCD.obj
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
Warning: no stack

C:\TASM>BCD.exe
09 02
```

Assignment 04:

AIM:

To understand Loop Operations in 8086 Assembly Programming. Write Programs:
1. To Move set of numbers from one memory block to another.
2. To count the number of 1's and 0's in a given 8-bit number.
3. Find even and odd numbers from a given list.

THEORY:

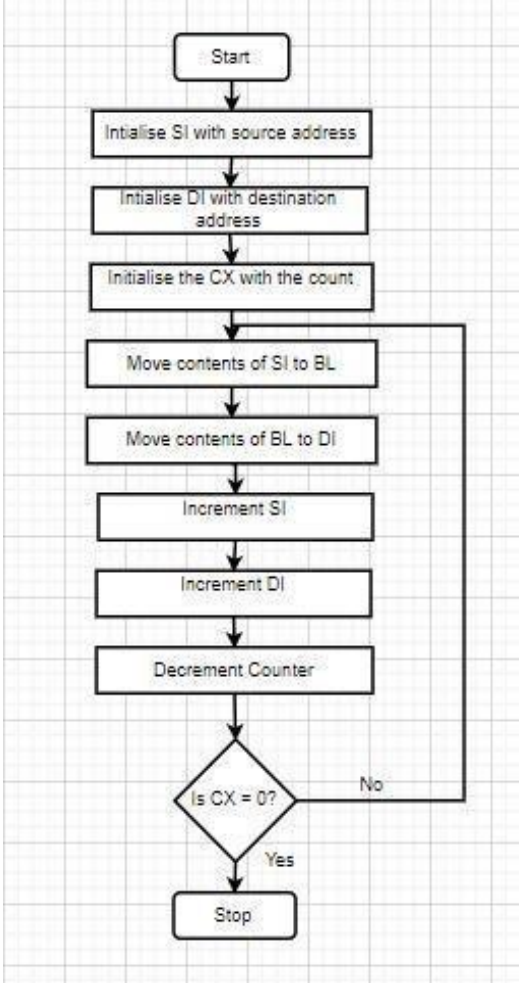
PROGRAM TO MOVE SET OF NUMBERS FROM ONE MEMORY BLOCK TO ANOTHER:

We initialize SI with the first memory address and DI with the destination memory address. Then 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 and check if it's 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 IV if CX not zero
Step 10: Display the result
Step 11: Stop

Flowchart:



Code:

```
.model small
.data
msg1 db "Data from 0009h are$"
 msg2 db "Data from 4200h are$"
msg3 db "After transfer$"
.code

start:
Mov AX, @data
Mov DS, AX

MOV SI,0003h ; Initialise SI with the Source memory address
MOV DI,4000h ; Initialise DI with the Destination memory address
```

```
MOV CL,05h ; Initialise the CX with the count
PUSH CX ; Save the count in stack
PUSH SI ; Save the source address in stack
Lea DX, [msg1] ; print the message
Mov AH, 09H
Int 21h
call n_line ; print a newline using pre-written procedure

printsi: ;printing the memory block SI holds
MOV BL, [SI] ;Move contents of SI to BL
MOV BH, 00H
call printhex ; printing the content of BX call n_line
INC SI ; incrementing the si
Dec CL ; decrementing Cl
JNZ printsi ; go to printsi if cx not zero
POP SI ; recovering the value of SI
POP CX ; recovering the value of CX
call displaydi ; printing the memory block DI holds for the count
CX LOOP1:
MOV bl,[SI] ;Move contents of SI to BL
MOV [DI],bl ;Move contents of temporary register to DI
INC SI ;Increment SI
INC DI ;Increment DI
DEC CL ;Decrement CX
JNZ LOOP1 ;Go to Step IV if CX not zero
MOV CL, 05H
MOV DI, 4000h
Lea DX, [msg3]
Mov AH, 09H
Int 21h
call n_line
call displaydi ; displaying the memory block DI holds for the count
CX MOV AH, 4CH
INT 21H


;description proc displaydi
PUSH CX
PUSH DI
Lea DX, [msg2]
Mov AH, 09H
Int 21h
call n_line printdi:
MOV BL, [DI]
MOV BH, 00H
CALL printhex
```

```
CALL n_line
INC DI
Dec CL JNZ printdi
POP DI
POP CX RET
endp
displaydi

; Prints new line

proc n_line mov dx,13
mov ah,2
int 21h
mov dx,10
mov ah,2
int 21h
ret
endp n_line

; Prints a 4 digit hex number. Store the hex number in BX register.
proc printhex
PUSH DX
PUSH BX
AND BH, 0F0H
ROL BH, 04
MOV DL, BH
call showit
POP BX
PUSH BX
AND BH, 0FH
MOV DL, BH
call showit
POP BX
PUSH BX
AND BL, 0F0H
ROL BL, 04
MOV DL, BL
call showit
POP BX
PUSH BX
AND BL, 0FH
MOV DL, BL
call showit
POP BX
POP DX RET
endp printhex
```

```
;The below procedure prints a hexdecimal digit. Store the single digit in DL register.
proc showit
PUSH AX

PUSH DX
CMP DL, 0Ah
JC numbers
MOV DH, 00H
ADD DX, 57H
MOV AH, 02H
INT 21h
JMP finish numbers: MOV DH, 00H

ADD DX, 48
MOV AH, 02H
INT 21h
JMP finish finish:
POP DX
POP AX RET
endp showit
end start
End
```

```
C:\TASM>move.exe
Data from 0009h are
0000
0044
0061
0074
0061
Data from 4200h are
00b7
0001
003a
00e9
0075
After transfer
Data from 4200h are
0000
0044
0061
0074
0061
```

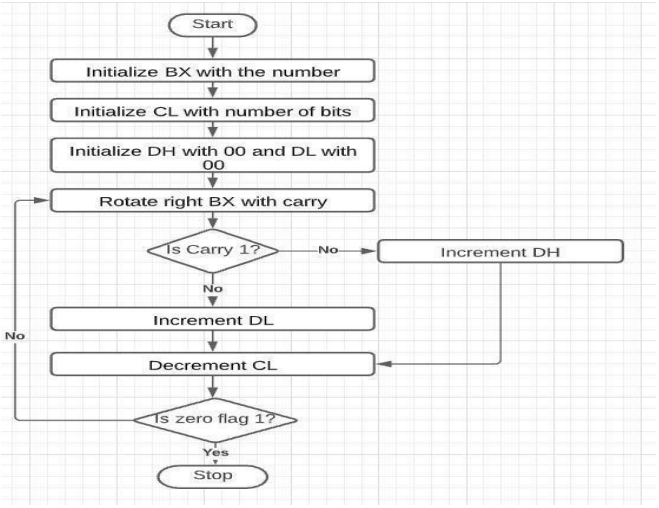
PROGRAM TO COUNT THE NUMBER OF 1's AND 0's IN A GIVEN 8-BIT NUMBER:

We start with initializing BX Register with the number. We initialise CX with 8. Now we rotate the number one along with the carry such that the last digit enters the carry. Now if the 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
Step 2: Initialize CL with the number of bits
Step 3: Initialize DH with 00 and DL with 00
Step 4: Rotate right with carry
Step 5: If Carry is set to one, go to Step VII
Step 6: Increment DL
Step 7: Go to Step IX
Step 8: Increment DH
Step 9: Decrement CL
Step 10: If Zero flag is not set to zero, go to Step IV
Step 11: Display Result
Step 12: Stop

Flowchart:



Code:

```
.model small
.data
no db 20H
bitCount db 08H
msg1 db "The no of zeroes in 20H are $"
msg2 db "The no of ones in 20H are $"

.code start:
Mov AX, @data Mov DS, AX
MOV BL, [no] ; Initializing BL with the number
MOV CL, [bitCount] ; Initializing CL with the count
MOV DX, 0000H ; Initializing DX for storing the count LOOP1:
RCR BX, 01H ; Rotate BX once with carry
JC NEXT ; Jump to next if carry is set to one
INC DL ; Increment DL i.e the number of Zeros
JMP NEXT2 ; Jump unconditionally to next2

NEXT:
INC DH ; Increment DL i.e the number of Ones NEXT2:
DEC CL ; Decrement the counter
JNZ LOOP1 ; Repeat from Loop1 if Zero flag isnt set to one
MOV BX, DX ; Storing the result in BX register
LEA DX, [msg1]
MOV Ah, 09H
INT 21H
MOV DL, BL
MOV DH, 00H
ADD DX, 48
MOV Ah, 02H
INT 21H
mov dx,13
mov ah,2
int 21h
mov dx,10
mov ah,2
int 21h
LEA DX, [msg2]
MOV Ah, 09H
INT 21H
MOV DL, BH
MOV DH, 00H
ADD DX, 48
MOV Ah, 02H
INT 21H
```

```
MOV AH, 4CH
INT 21H
END
```

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


C:\TASM>tlink count.obj
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
Warning: no stack

C:\TASM>count.exe
The no of zeroes in 20H are 7
The no of ones in 20H are 1
C:\TASM>m
```

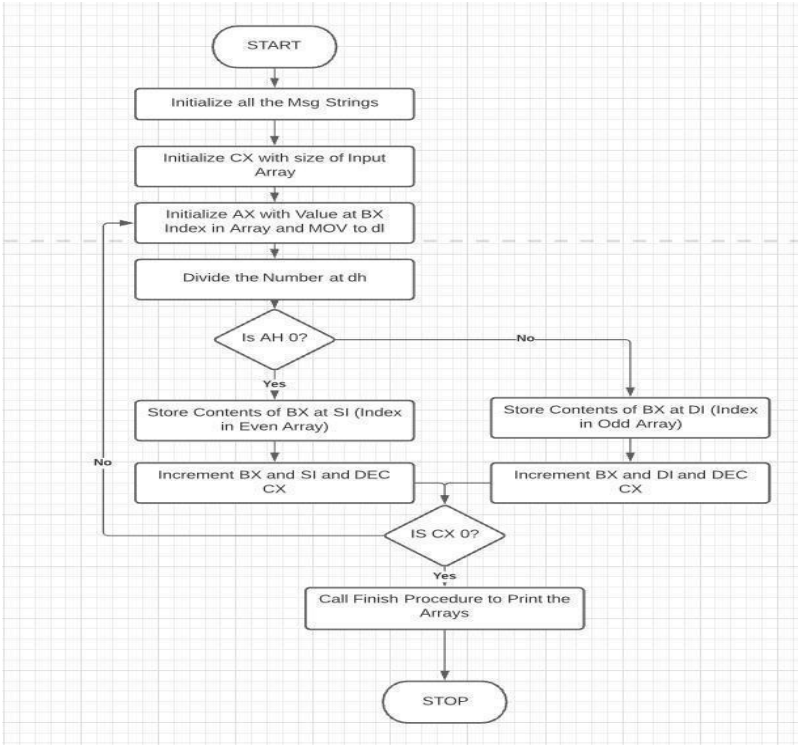
PROGRAM TO FIND EVEN AND ODD NUMBERS FROM A GIVEN LIST:

We create three arrays, 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:

Step 1: Initialize all messages, new line characters, etc which are 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 the array, divide it by 2 and compare the remainder.
Step 4: If remainder is zero, jmp to is evenProc and store the dl 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.
Step 6: Else, it's odd so store contents in an 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:



Code:

```
.model small
.data
msg db 'Items in Input Array are: ', '$';for input msg
msg1 db 'Items in Even Array are: ', '$'; for even msg
msg2 db 'Items in Odd Array are: ', '$'; for odd msg n_line
DB 0AH,0DH,"$" ;for new line
InputArray db 1, 2, 3, 9, 6, 5, 2, 4, 9, 4 ; initialize input Array
EvenArray db 10 dup(?); create empty even array of size of input array as all items
could be even OddArray
db 10 dup(?); create empty odd array of size of input array

as all items could be odd
.code
mov ax,@data; Load Data Segment
mov ds,ax;
LEA BX,InputArray; Load Effective Address of Input Array in BX Register
LEA SI,EvenArray; Load Effective Address of Even Array in SI Register
LEA DI,OddArray; Load Effective Address of Odd Array in DI Register
mov cx,10; Initialize count to traverse Input Array in CX Register
mov dh,02

loopThroughInput:
mov ah,00
mov al,[BX]; Load Value in al
mov dl,al; Mov it dl register
div dh; Divide Number at Index
cmp ah,00; If Rem is zero, its even so jmp to even proc to add it in even atray
je isEven; If Even, store in even array
mov [DI],dl;
mov dl contents to pos in Odd arr if Jump to isEven is False
INC DI; inc di i.e. index of odd arr
INC BX; inc bx i.e. index of inputArray
Loop loopThroughInput; loop untl all elements in array are done
jmp finalProc; call final proc to print all


isEven:; Store in even Array Procedure
mov [SI],dl;
mov dl contents to pos in Even arr if Jump to isEven is True
INC SI ; inc si i.e. index of even arr
INC BX; inc bx i.e. index of inputArray Loop loopThroughInput

finalProc:
LEA DX,n_line ;load n_line
```

```
MOV AH,9
INT 21H ;print new line
LEA DX, msg ;load input_msg
MOV AH,9

INT 21H ;print input msg
mov si, offset InputArray; load Input array
mov cx, 10; create a counter with number of items in array
displnput:; proc to display all elements in input array
mov dl, [si]; to load value at index in array
add dl, 48
mov ah, 02h
int 21h
; To Print Spaces mov dl, 32
mov ah, 02h
int 21h
inc si;
inc index Loop displnput
LEA DX,n_line ;load n_line
MOV AH,9
INT 21H ;print new line
mov ax,0000
mov bx,0000
LEA DX, msg1 ;load even_msg
MOV AH,9
INT 21H ;print even msg
mov si, offset EvenArray; load Even array
mov cx, 10; create a counter with number of items in array
dispEven:; proc to display all elements in even array
mov dl, [si]; to load value at index in array
add dl, 48
mov ah, 02h
int 21h
; To Print Spaces mov dl, 32
mov ah, 02h
int 21h
inc si;
inc index

Loop dispEven

LEA DX,n_line ;load n_line
MOV AH,9
INT 21H ;print new line
LEA DX, msg2 ;load odd_msg
MOV AH,9
```

```
INT 21H ;print odd msg
mov si, offset OddArray; load Odd array
mov cx, 10; create a counter with number of items in array
dispOdd:; proc to display all elements in odd array
mov dl, [si]; to load value at index in array
add dl, 48
mov ah, 02h
int 21h ; To Print Spaces mov dl, 32
mov ah, 02h
int 21h
inc si; inc index Loop dispOdd
mov ax,4C00h; exit
int 21h
End
```

CONCLUSION:

Loop operations of 8086 were implemented in the lab using tasm.

Name - Abhay Rangnani Batch - S21 Roll Number - 78

Assignment 05:

AIM:

1] Check if a string is palindrome or not
2] Computer factorial of a positive integer n using procedure Or
Generate the first n Fibonacci series

THEORY:

CHECK IF A STRING IS PALINDROME OR NOT:

A palindrome is a string of characters that reads the same forwards as backwards. For example, the following are both palindromes. 1457887541, madam.

Input String: "madam"
Output: String is palindrome

Input String: "inaaya"
Output: String is not palindrome

Algorithm:

1. Create a string
2. Traverse to the end of the string
3. Get the address of the end of the string, DI
4. Load the starting address of the string, SI
5. Compare the value stored at the address
6. Increment the pointer, SI
7. Decrements the pointer, DI
8. Compare again the value stored at si and di
9. Repeat the steps until SI<=DI
10. If all the characters match print string is palindrome else print not palindrome

Code:

```
.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
```

```
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
String is palindrome
C:\TASM>_
```

COMPUTE FACTORIAL OF A POSITIVE INTEGER N USING PROCEDURE:

Assumptions –
Starting address of program: 0400
Input memory location: 0500
Output memory location: 0600 and 0601

If the Given Number is a 16-bit number, the AX register is automatically used as the second parameter and the product is stored in the DX:AX register pair. This means that the DX register holds the high part and the AX register holds the low part of a 32-bit number.

Algorithm:

1. MOV CX, [0500] loads 0500 Memory location content to CX Register
2. MOV AX, 0001 loads AX register with 0001
3. MOV DX, 0000 loads DX register with 0000
4. MUL CX multiply AX with CX and store result in DX:AX pair
5. LOOP 040A runs loop till CX not equal to Zero
6. MOV [0600], AX store AX register content to memory location 0600
7. MOV [0601], DX store DX register content to memory location 0601
8. HLT stops the execution of program

Code:

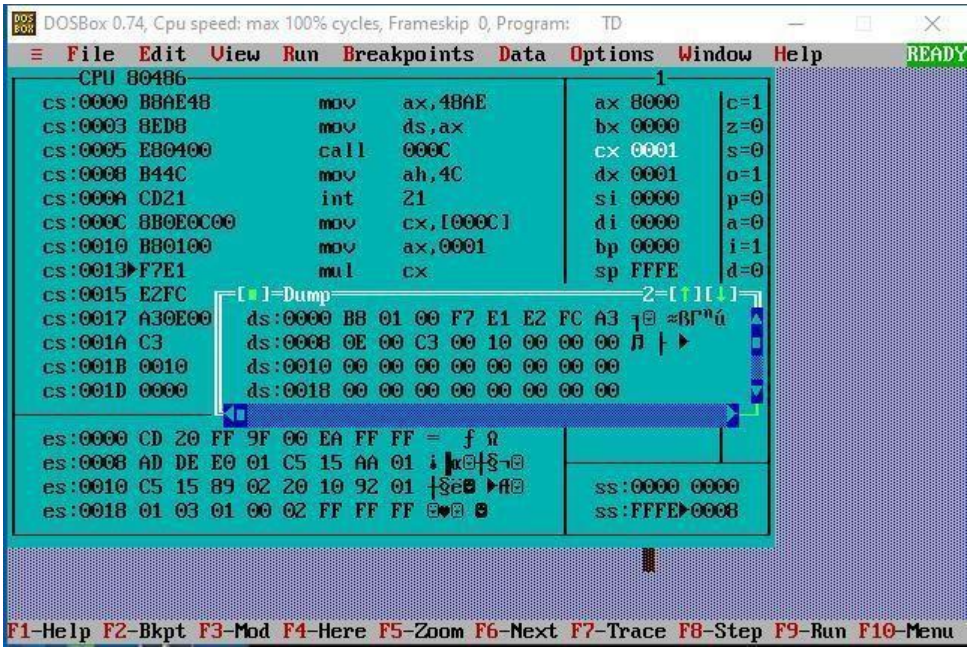
```
.model small
.data
num1 dw 010H ; Initialize num1 with any number we want to find factorial of result
dw ? ; This will store the result

.code
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
CALL Factorial ; Calling simple Procedure

;Interrupt to exit
 MOV AH, 4CH
INT 21H
MAIN ENDP

Factorial PROC; We move the number into CS register because CS acts as a count
MOV CX, num1 ;
```

MOV num1 into the code segment CX
MOV AX,01H;
MOV 01H to AX register so that we can multiply

MULTIPLY: MUL CX

LOOP MULTIPLY ; We loop through the MULTIPLY label so to multiply the value in CX with AX also we decrement the value in CX by 1 with LOOP

MOV result,AX ; At last we store the value of AX into result
RET ; Returning the Procedure
Factorial ENDP
END MAIN



CONCLUSION:
Programs on string were implemented using TASM in the lab successfully.

Assignment 06 - Interfacing with 8086 Microprocessor:

AIM:
Interfacing Seven Segment display.

THEORY:
Proteus is a simulation and design software tool developed by Labcenter Electronics for Electrical and Electronic circuit design. It also possesses 2D CAD drawing features. It deserves to bear the tagline “From concept to completion”.
About Proteus:

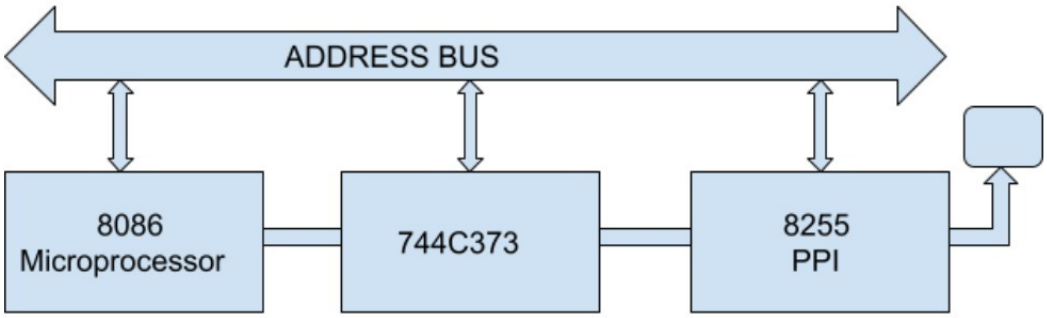
It is a software suite containing schematic, simulation as well as PCB designing.

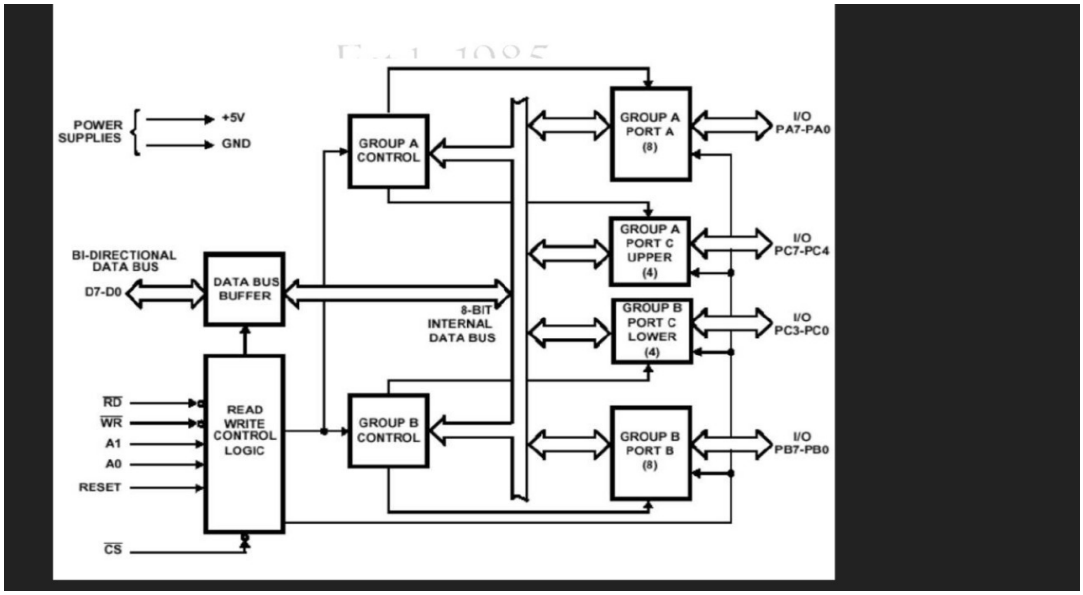
- ISIS is the software used to draw schematics and simulate the circuits in real time.The simulation allows human access during run time,thus providing real time simulation.
- ARES is used for PCB designing.It has the feature of viewing output in 3D view of the designed PCB along with components.
- The designer can also develop 2D drawings for the product.

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 labelled from a through to 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).

Block diagram of 8086 interfacing 8255 pi:





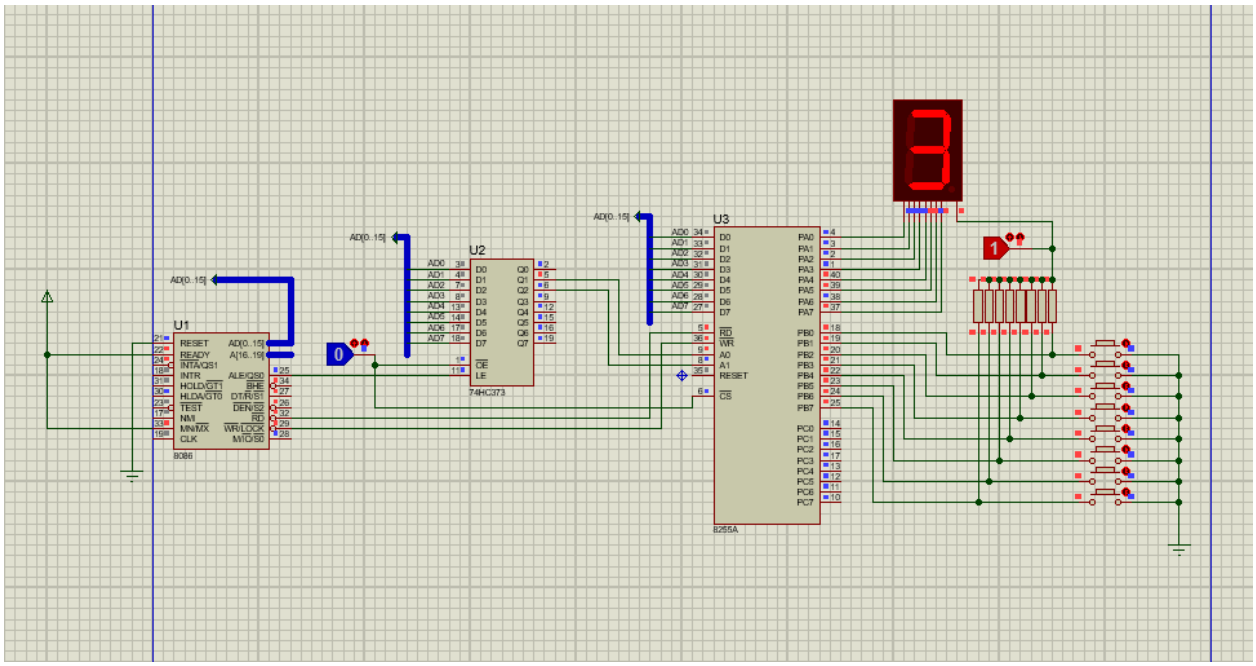
CODE:

```
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:
MOV AL, 10011001B
JMP DISPLAY
S5:
MOV AL, 10010010B
JMP DISPLAY
S6:
```

```
MOV AL, 10000010B
JMP DISPLAY
S7:
MOV AL, 11111000B
JMP DISPLAY


CODE ENDS
END
```

CIRCUIT:



CONCLUSION:

Hence, we have implemented 7 segments using Proteus software and it maps to loc.