**LAB – 1**

**VERIFICATION OF LOGIC GATES**

**Description:**

Logic gates are the basic building blocks of any digital system. Logic gates are electronic circuits 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. Based on this, logic gates are named as

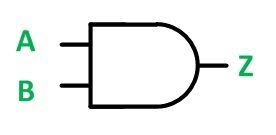
1. AND gate
2. OR gate
3. NOT gate
4. NAND gate
5. NOR gate
6. Ex-OR gate
7. Ex-NOR gate

**AND GATE :**

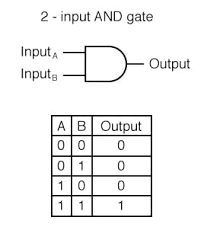
The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B or can be written as Z.

**Y= A.B**

**Circuit diagram :**

****

**Truth table:**



**Procedure**:

1.Open a new blank schematic in multisim.

2.And place a 2-input AND Gate on the sheet by place component.

3.After placing the AND Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input AND gate and probe to the output AND Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

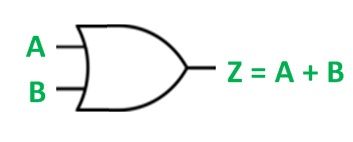
**AND gate in multisim :**

****

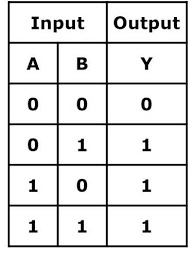
**OR GATE :**

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation. **Y= A+B**

**Circuit diagram :**



**Truth table :**



**Procedure:**

1.Open a new blank schematic in multisim.

2.And place a 2-input OR Gate on the sheet by place component.

3.After placing the OR Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input OR gate and probe to the output OR Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**OR gate in multisim :**

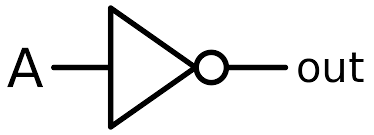


**NOT GATE:**

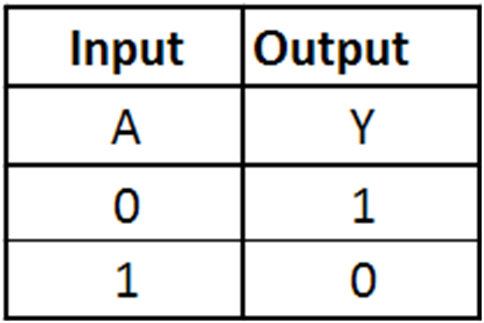
The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A' or A with a bar over the top, as shown at the outputs.

**Y= A'**

**Circuit diagram:**

****

**Truth table :**



**Procedure :**

1.Open a new blank schematic in multisim.

2.And place a NOT Gate on the sheet by place component.

3.After placing the NOT Gate,we have to give one input by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach one key to the input NOT gate and probe to the output NOT Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**NOT gate in multisim :**

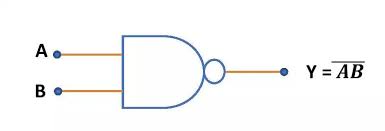
****

**NAND GATE :**

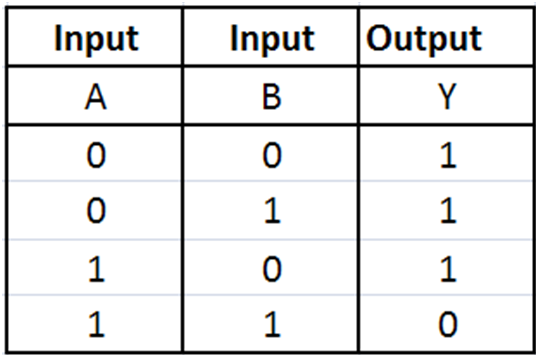
This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

**Y= (AB)’**

**Circuit diagram :**

****

**Truth table :**



**Procedure:**

1.Open a new blank schematic in multisim.

2.And place a 2-input NAND Gate on the sheet by place component.

3.After placing the NAND Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input NAND gate and probe to the output NAND Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**NAND gate in multisim :**

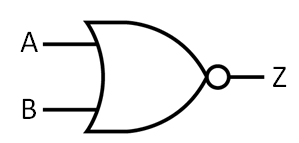


**NOR GATE :**

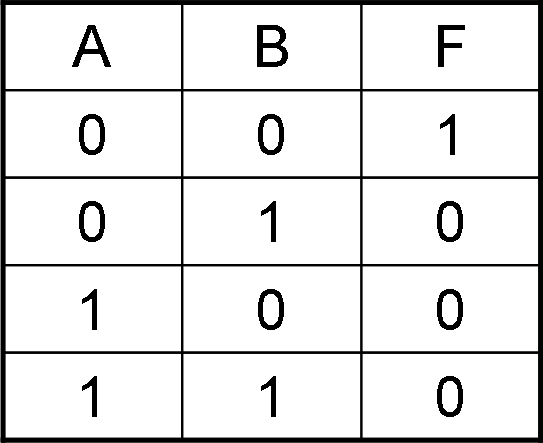
This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

**Y= (A+B)’**

**Circuit diagram :**



**Truth table :**



**Procedure :**

1.Open a new blank schematic in multisim.

2.And place a 2-input NOR Gate on the sheet by place component.

3.After placing the NOR Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input NOR gate and probe to the output NOR Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**NOR gate in multisim :**

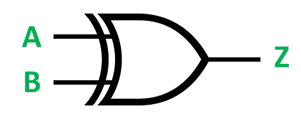


**6) EX-OR GATE :**

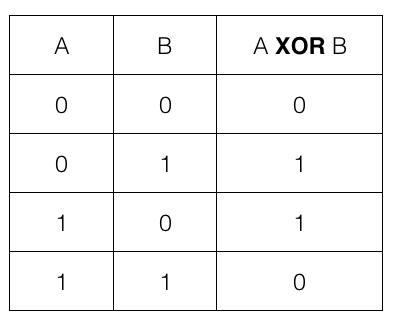
The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign (⊕) is used to show the Ex-OR operation.

**Y= A⊕B**

**Circuit diagram :**

****

**Truth table :**



**Procedure:**

1.Open a new blank schematic in multisim.

2.And place a 2-input EX-OR Gate on the sheet by place component.

3.After placing the EX-OR Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input EX-OR gate and probe to the output EX-OR Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**Ex-or gate in multisim :**

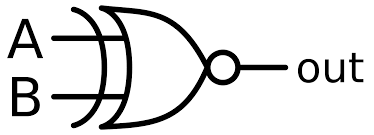


**EX-NOR GATE :**

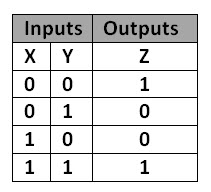
The 'Exclusive-NOR' gate circuit does the opposite to the EX-OR gate. It will give a low output if either, but not both of its two inputs are high. The symbol is an EX-OR gate with a small circle on the output. The small circle represents inversion.

**Y= A⊕B**

**Circuit diagram :**



**Truth table :**

****

**Procedure :**

1.Open a new blank schematic in multisim.

2.And place a 2-input EX-NOR Gate on the sheet by place component.

3.After placing the EX-NOR Gate,we have to give inputs by the Digital Constant Component.

Path:🡪Digital Source🡪Digital\_Constant

4.Select a probe from component.

5.Attach two keys to the input EX-NOR gate and probe to the output EX-NOR Gate.

6.Run the schematic and check the probe whether glowing or not based on the inputs according to the truth table.

**Ex-nor gate in multisim :**



**B]Assembly language program to find largest number in array**

**Code:**

org 100h

.MODEL SMALL

.STACK 100H

.DATA

array DB 2,8,9,5,7

.CODE

MAIN PROC

MOV AX,@DATA

MOV DS,AX

MOV CX,5

MOV DI,0

SUB AL,AL

BIG:

CMP AL,array[DI]

JA NEXT

MOV AL,array[DI]

NEXT:

INC DI

LOOP BIG

MOV AH,2

ADD AL,30H

MOV DL,AL

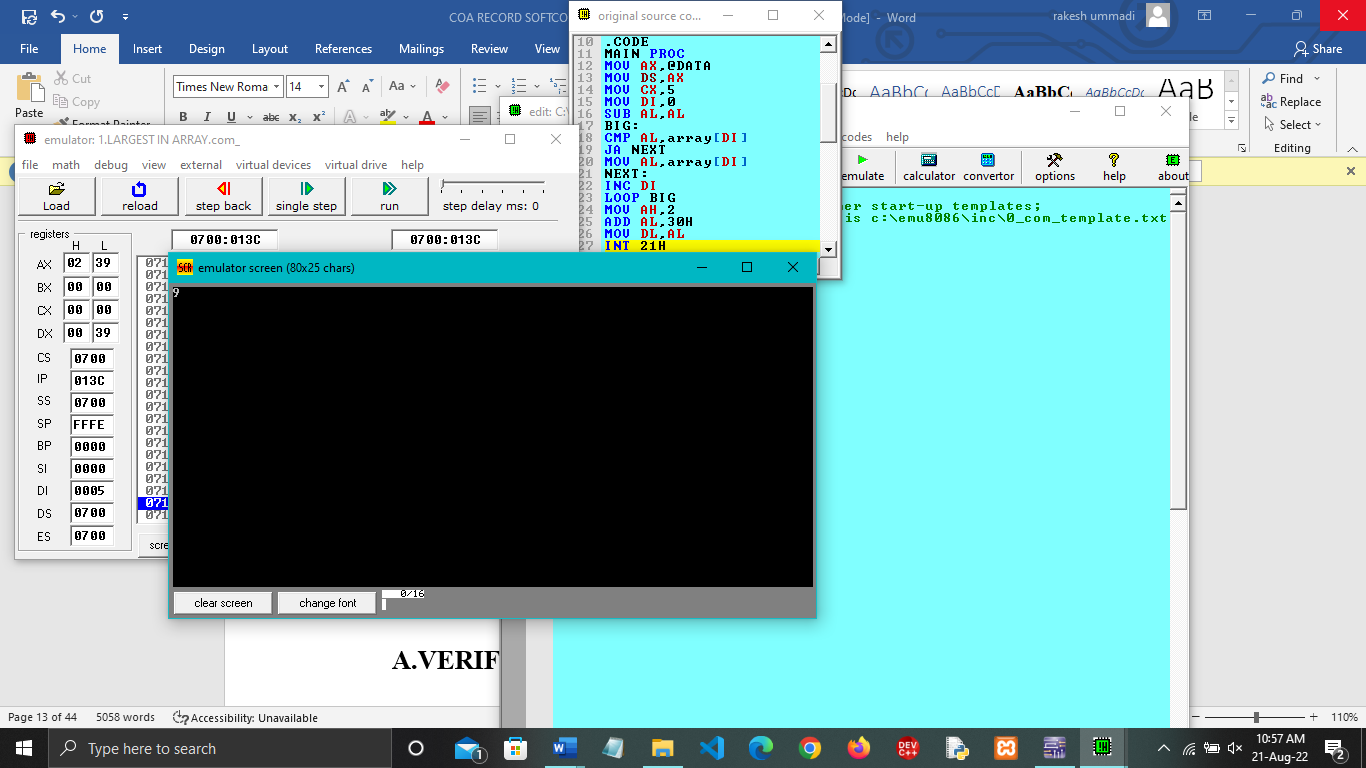
INT 21H

MAIN ENDP

END MAIN

ret

**Output :**



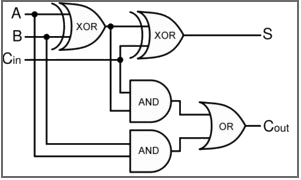
**LAB – 2**

**A]VERIFICATION OF FULL ADDER & FULL SUBTRACTOR**

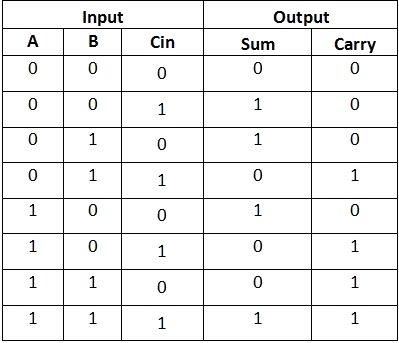
**FULL ADDER :**

Full adder is a digital circuit used to calculate the sum of three binary bits. Full adders are complex and difficult to implement when compared to half adders. Two of the three bits are same as before which are A, the augend bit and B, the addend bit. The additional third bit is carry bit from the previous stage and is called 'Carry' – in generally represented by CIN. It calculates the sum of three bits along with the carry. The output carry is called Carry – out and is represented by Carry OUT.

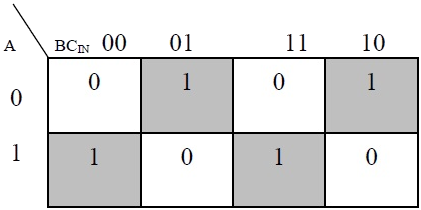
**Circuit diagram :**



**Truth table :**

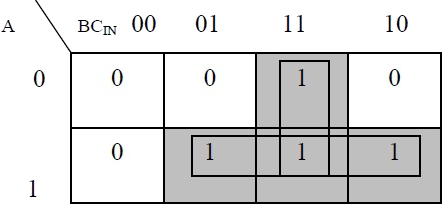


**The K-Map simplified equation for SUM:**



**S = A'B'Cin + A'BCin' + ABCin**

**The K-Map simplified equation for COUT :**



**COUT = AB + ACIN + BCIN**

**Procedure :**

1. Open a new blank schematic in multisim.

2. For construct Half Adder ,we need an EX-OR and AND Gate.

3. Place an EX-OR gate on the schematic from master database 🡪TTL Group and place an AND gate on the schematic in the same process.

4. Place two input keys from Digital\_Source\_Constant to give input for gates.

5. Place a probe from components.

6. Connect probes to the EX-OR gate output and connect another probe to the AND gate ouput

7. Connect the input keys for both AND and EX-OR Gate.

8. Run the schematic to check the Half Adder Implementation.

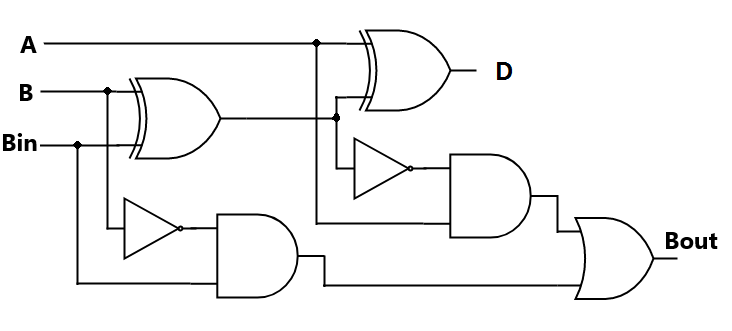
**Full adder in multisim :**

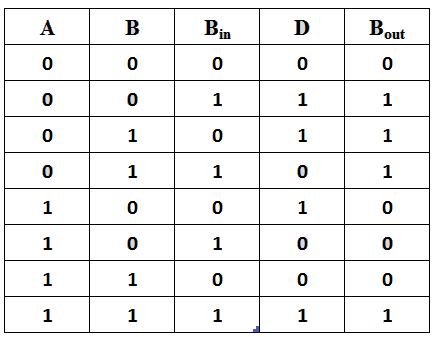


### 2.FULL SUBTRACTOR :

A full subtractor is a combinational circuit that performs subtraction involving three bits, namely A (minuend), B (subtrahend), and Bin (borrow-in) . It accepts three inputs: A (minuend), B (subtrahend) and a Bin (borrow bit) and it produces two outputs: D (difference) and Bout (borrow out).

**Circuit diagram :**



**Truth table :**   


From the above truth table we can find the boolean expression.

**D = A ⊕ B ⊕ Bin  
Bout = A' Bin + A' B + B Bin**

**Procedure :**

1.Open a new blank schematic in multisim.

2.To construct a Full Adder ,we need 2 EX-OR gates,2 two input AND gates,2 NOT gates,1 two I input OR gate.

3.Place all the required gates in the schematic from componet🡪 TTL Groups.

4.Place two Probs in the shematic from Diode Groups.

5.Place three inputs from the Digital Source.

6.Connect all the input gates in shown in the circiute diagram.

7.connect ouput of EX-OR and OR gate to each probe.

8.connect another ends of probes .

9.Run the schematic abd check the implementation of Full Subtractor based on the inputs given according to the truth table.

**Full subtractor in multisim :**



**B] Assembly language program to find smallest number in array**

**Code:**

org 100h

include 'emu8086.inc'

.model small

.stack 100h

.data

array db 7,3,4,6,5

.code

main proc

mov ax,@data

mov ds,ax

mov si,offset array

mov cx,5

mov bl,[si]

loopx:

cmp [si],bl

jle update

resume:

inc si

loop loopx

print 'Smallest vaue from array:'

add bl,48

mov dl,bl

mov ah,02h

int 21h

update:

mov bl,[si]

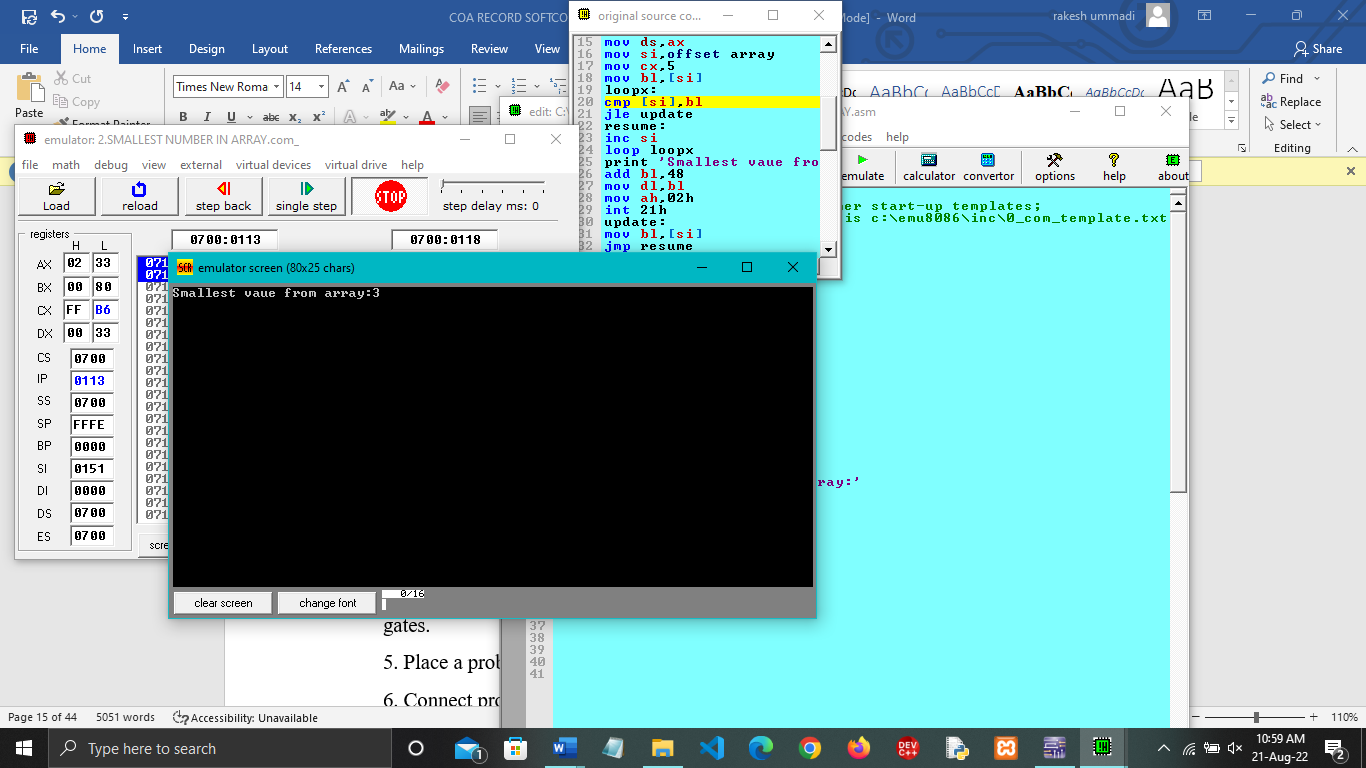
jmp resume

main endp

end main

ret

**Output:**



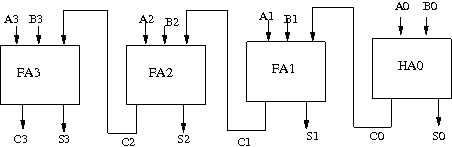
**LAB 3**

**VERIFICATION OF RIPPLE CARRY ADDER & CARRY LOOK AHEAD ADDER.**

**RIPPLE CARRY ADDER:**

Arithmetic operations like addition, subtraction, multiplication, division are basic operations to be implemented in digital computers using basic gates likr AND, OR, NOR, NAND etc. Among all the arithmetic operations if we can implement addition then it is easy to perform multiplication (by repeated addition), subtraction (by negating one operand) or division (repeated subtraction).

**Circuit diagram :**



**Truth table :**



**Procedure:**

1. Start the simulator as directed.This simulator supports 5-valued logic.
2. To design the circuit we need 3 full adder, 1 half adder, 8 Bit switch(to give input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1, For full adder input is in pin-5,6,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component, likewise add 3 full adders,8 Bit switches, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect 4 bit switches to the 4 terminals of a digital display and another set of 4 bit switches to the 4 terminals of another digital display. connect the pin-1 of the full adder which will give the final carry output. connet the sum(pin-4) of all the adders to the terminals of the third digital display(according to the circuit diagram shown in screenshot). After the connection is over click the selection tool in the pallete.
7. 7To see the circuit working, click on the Selection tool in the pallet then give input by double clicking on the bit switch, (let it be 0011(3) and 0111(7)) you will see the output on the output(10) digital display as sum and 0 as carry in bit display.

**Ripple carry adder in multism :**

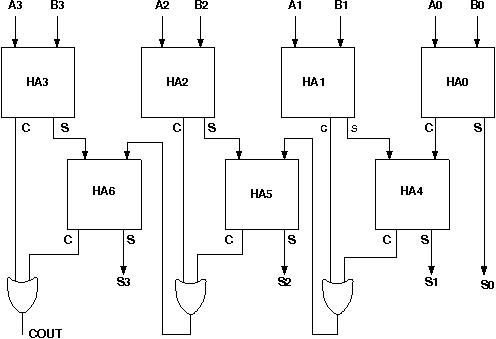


**CARRY LOOK AHEAD ADDER:**

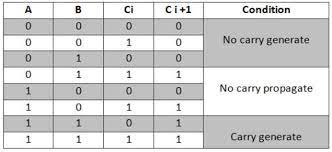
**Theroy :**

To reduce the computation time, there are faster ways to add two binary numbers by using carry lookahead adders. They work by creating two signals P and G known to be **Carry Propagator** and **Carry Generator**. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry ,regardless of input carry.

**Circuit diagram:**



**Truth table:**



**Procedure:**

1. Start the simulator as directed.This simulator supports 5-valued logic.
2. To design the circuit we need 7 half adder, 3 OR gate, 1 V+(to give 1 as input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 6 more full adders(from the Adder drawer in the pallet), 3 OR gates(from Logic Gates drawer in the pallete), 1 V+, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect V+ to the upper input terminals of 2 digital displays according to you input. connect the OR gates according to the diagram shown in the screenshot connect the pin-1 of the half adder which will give the final carry output. connet the sum(pin-4) of those adders to the terminals of the third digital display which will give output sum. After the connection is over click the selection tool in the pallete.
7. See the output, in the screenshot diagram we have given the value 0011(3) and 0111(7) so get 10 as sum and 0 as carry.you can also use many bit switches instead of V+ to give input and by double clicking those bit switches can give different values and check the result.

**Carry look ahead adder in multisim :**



**B . Assembly Language Program For Adding Two Arrays.**

**CODE :**

org 100h

jmp start

arr1 db 1,2,3,4

arr2 db 3,5,6,7

arr3 db ?,?,?,?

start:

lea si,arr1

lea bx, arr2

lea di,arr3

mov cx, 4

sum:

mov al, [si]

add al, [bx]

mov [di], al

inc si

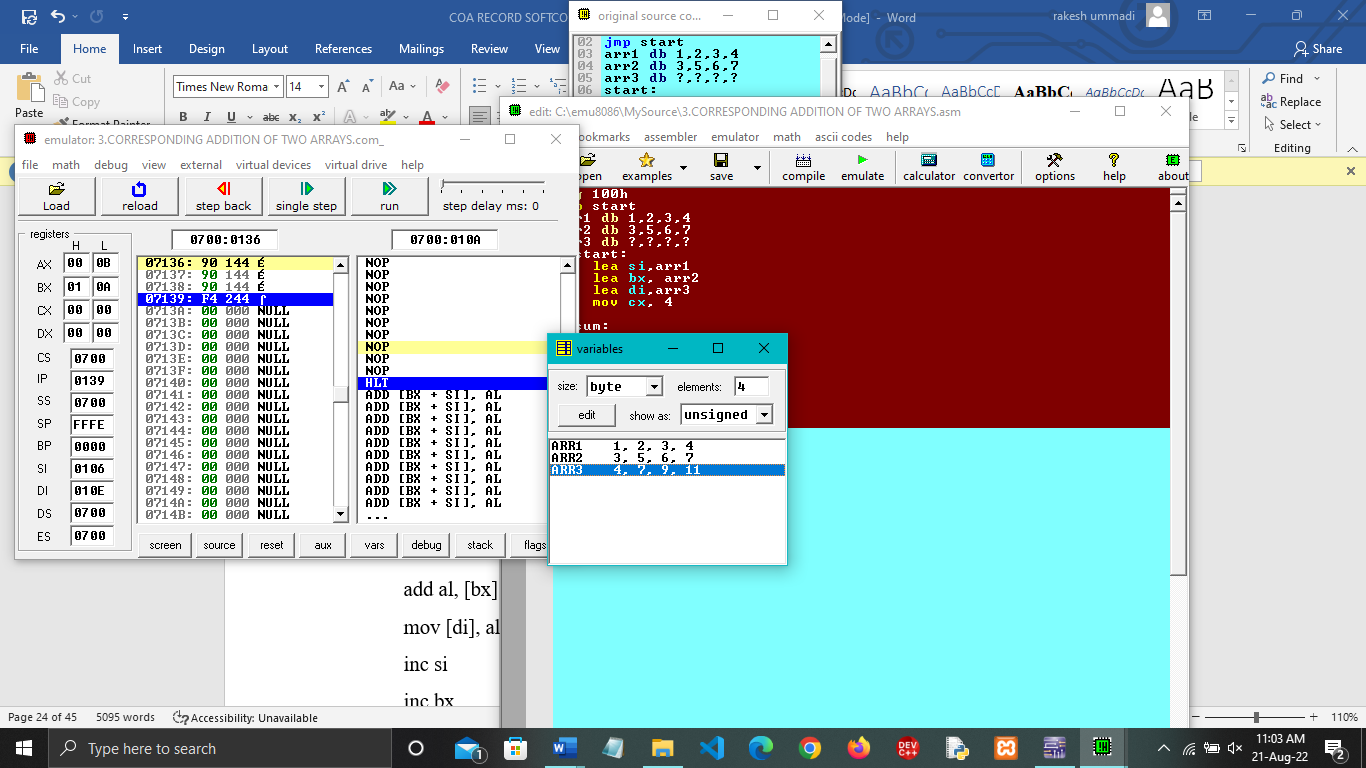
inc bx

inc di

loop sum

end

**OUTPUT :**



**LAB-4**

**COMBINATIONAL MULTIPLIER**

**Description :**

Combinational Multipliers do multiplication of two unsigned binary numbers.Each bit of the multiplier is multiplied against the multiplicand, the product is aligned according to the position of the bit within the multiplier, and the resulting products are then summed to form the final result. Main advantage of binary multiplication is that the generation of intermediate products are simple: if the multiplier bit is a 1, the product is an ppropriately shifted copy of the multiplicand; if the multiplier bit is a 0, the product is simply 0.

**Components :**

1]16 Two-input AND gates

2]8 Full Adders

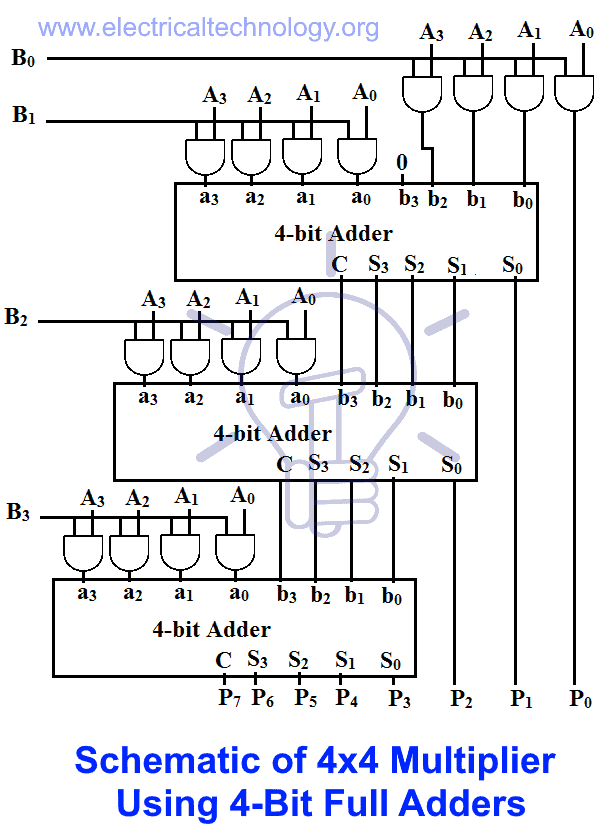
3]4 Half Adders

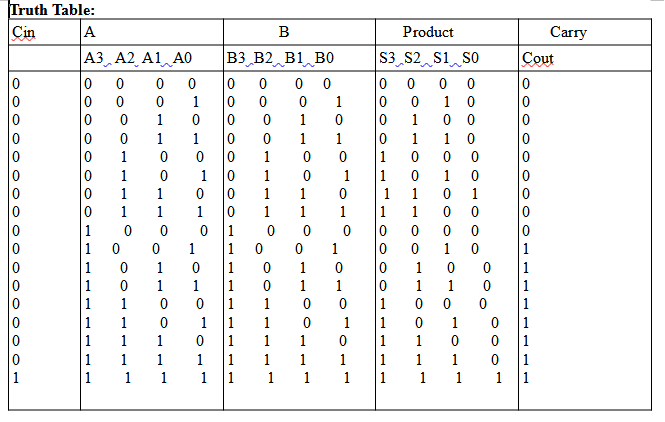
4]Probes to show output

5] wires to connect

**Circuit diagram :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  |  | A3 | A2 | A1 | | A0 |
|  |  | |  |  | B3 | B2 | B1 | | B0 |  |
|  |  | |  |  |  | | | | |  |
|  |  | |  |  | A3 . B0 | A2 . B0 | A1 .  B0 | | A0 . B0 |  |
|  |  | |  | A3 . B1 | A2 . B1 | A1 . B1 | A0 .  B1 | |  |  |
|  | | A3 . B2 | | A2 . B2 | A1 . B2 | A0 . B2 |  |  |  |
| A3 . B3 | | A2 . B3 | | A1 . B3 | A0 . B3 |  |  |  |  |
|  | | | | | | | | |  |
| S6 | | S5 | | S4 | S3 | S2 | S1 | S0 |  |

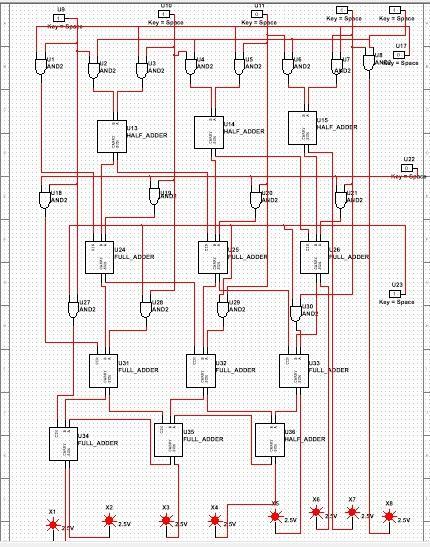




**Procedure:**

1. Start the simulator as directed.This simulator supports 5-valued logic.
2. To design the circuit we need 8 full adders, 4 half adders, 16 AND gates, 8 bit switch (to give input,which will toggle its value with a double click), 8 bit displays (to see the output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner (indicated with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1, For full adder input is in pin-5,6,8 output sum is in pin-4 and carry is in pin-1
5. Click on the half adder component (in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component (no drag and drop, simple click will serve the purpose), likewise add 3 more half adders, 8 full adders (from the Adder drawer in the pallet), 16 AND gates (from Logic Gates drawer of the pallet,if it is not seen scroll down in the drawer), 8 Bit switches, 8 bit Displays (from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect bit switches to the input terminals. connect the output terminals to the Bit display component. After the connection is over click the selection tool in the pallete.
7. See the output, bit switches are used to give input so that you can toggle its value with a double click and see the outputs with different inputs.

**Combinational multiplier in multisim :**



**LAB – 5**

**Assembly language program to find prime numbers between a given range**

**Code:**

org 100h

.model small

.stack

.code

main proc

mov ax,@data

mov ds,ax

mov dl,1

mov cx,10

mov si,offset primenum

L1:mov bl,2

add dl,1

cmp dl,2

je insert

logic:mov ah,0

mov al,dl

div bl

cmp ah,0

je L1

add bl,1

cmp bl,al

jb logic

jmp insert

insert:mov [si],dl

inc si

loop L1

mov ah,4ch

int 21h

main endp

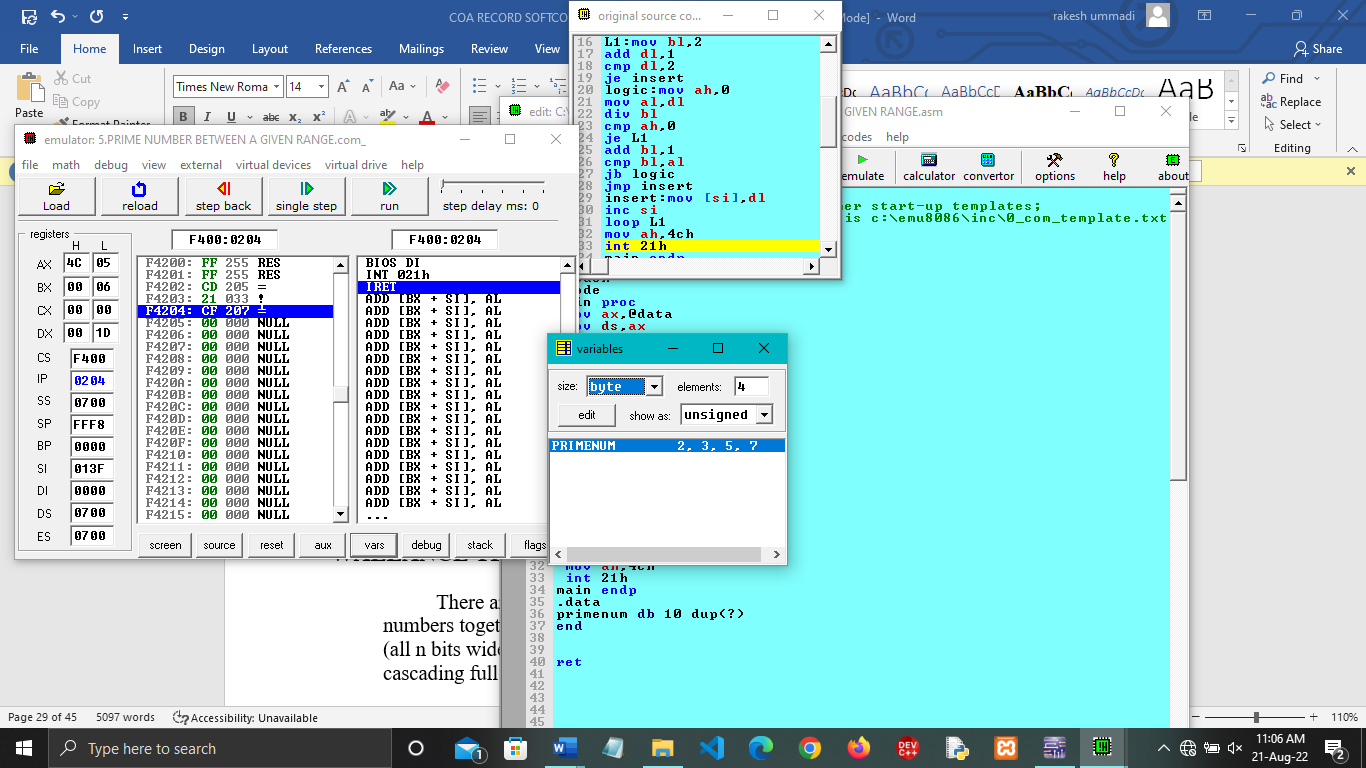
.data

primenum db 10 dup(?)

end

ret

**Output :**



**LAB- 6**

**WALLANCE TREE ADDER**

**Theory:**

There are many cases where it is desired to add more than two numbers together. The straightforward way of adding together m numbers (all n bits wide) is to add the first two, then add that sum to the next using cascading full adders. This requires a total of m − 1 additions, for a total gate delay of O(m lg n) (assuming lookahead carry adders). Instead, a tree of adders can be formed, taking only O(lg m · lg n) gate delays.

A Wallace tree adder adds together n bits to produce a sum of log2n bits.

**Components:**

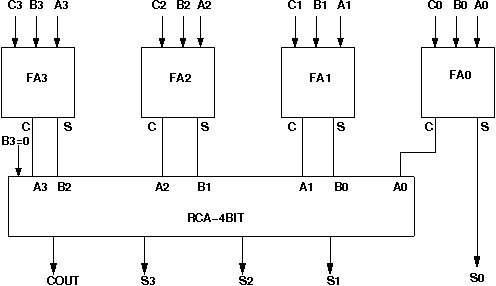
1] 4 - Full Adders

2] 4 – bit Adder

3] Display uints to check the outputs

4] Wires to connect.

**Circuit diagram:**



**Procedure:**

1**.**Start the simulator as directed.This simulator supports 5-valued logic.

2.To design the circuit we need 4 full adder, 1 'RCA 4 bit' component, 12 Bit switch(to give input), 5 Bit displays(for seeing output), wires.

3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

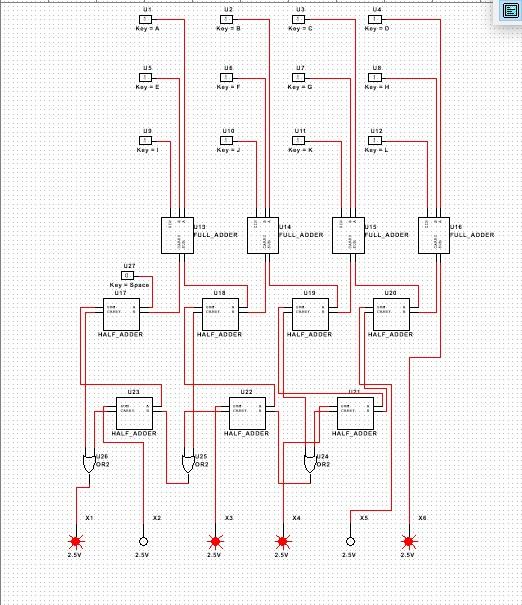
4. For a full adder input is in pin-5,6,8, output sum is in pin-4 and carry is pin-1

5. For a 'RCA 4 bit' input A0=pin-13 A1=pin-14 A2=pin-15 A3=pin-16, B0=pin-17 B1=pin-18 B2=pin-19 B3=20, C0=21, output S0=pin-12 S1=pin-11 S2=pin-10 S3=pin-9 Cout=pin-8

6. Click on the full adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 3 more full adder(from the Adder drawer in the pallet), 1 'RCA 4 bit'(from the Adder drawer in the pallet,if it is not seen scroll down in the drawer), 12 Bit switches and 5 Bit Displays(from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer).connect the Bit switches with the inputs and Bit displays component with the outputs.

7. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components. After the connection is over click the selection tool in the pallete.

**Wallance tree adder in multisim :**



**B] Assembly language program to find factorial of the given number**

**Code :**

org 100h

.DATA

ANS DB ?

.CODE

MAIN PROC

MOV AX,@DATA

MOV DS,AX

MOV AL,5

MOV CL,4

MOV BL,AL

SUB BL,1

L:

MUL BL

SUB BL,1

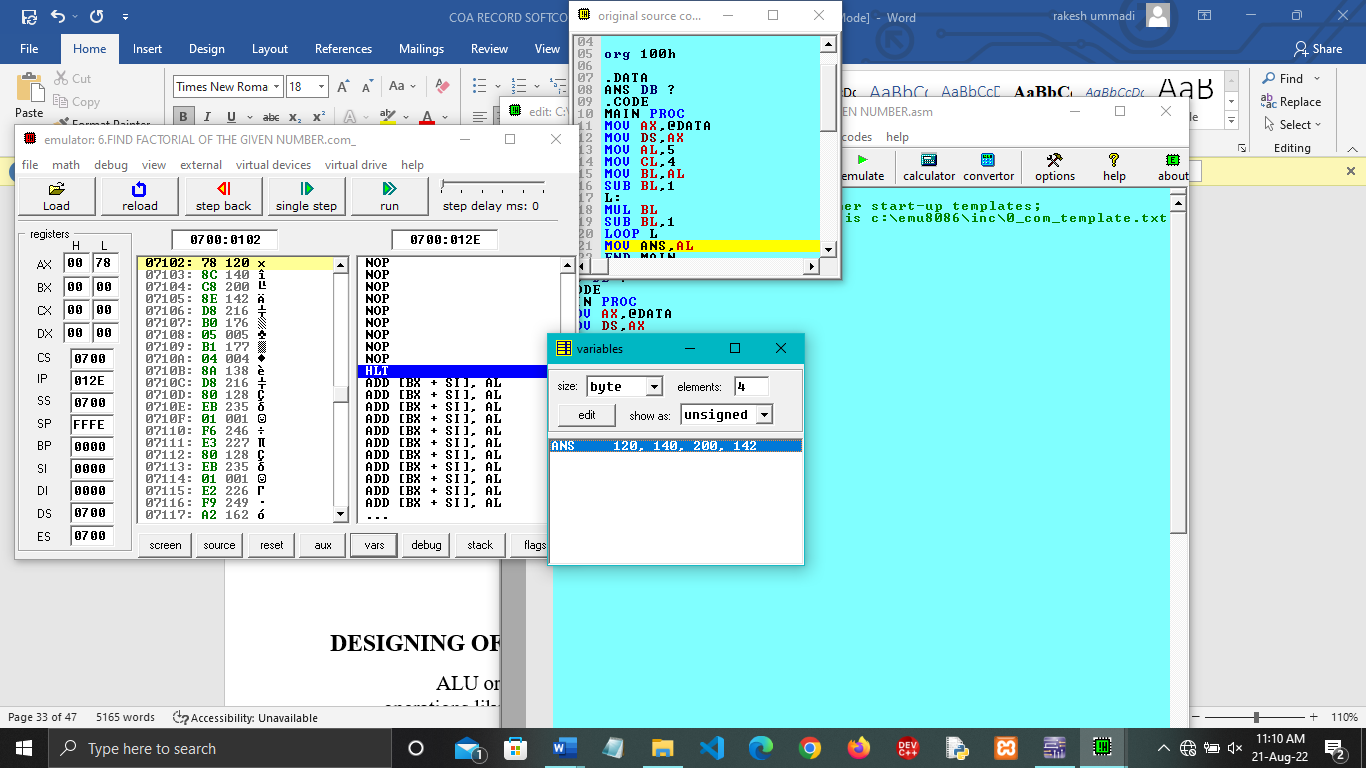
LOOP L

MOV ANS,AL

END MAIN

ret

**Output :**



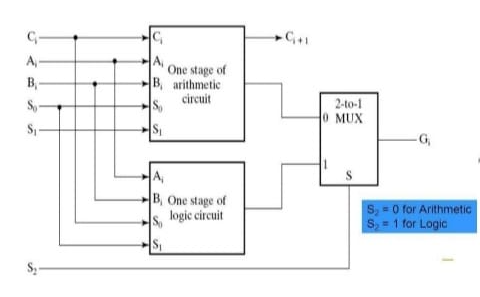
**LAB -7**

**ARITHMETIC LOGIC UINT**

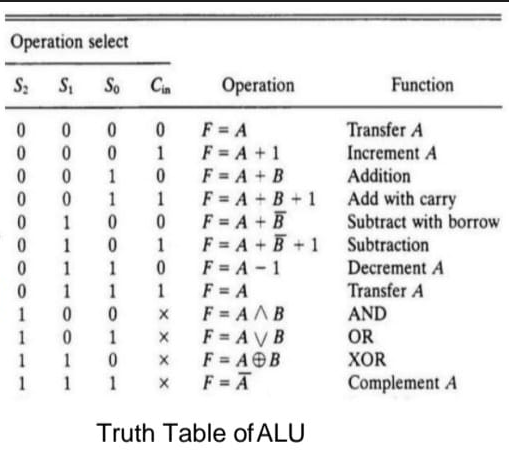
**Description:**

Arithmetic logic unit is implemented by combining the Arithmetic unit and logic unit with the help of 2 by 1 multiplexer.

**Circuit diagram:**

****

**Truth table :**

****

**Components:**

Two 4 by 1 multiplexers,One 2 by 1 multiplexers,Two full adders,One half adder,Two 2-input xor gates,Two 2-input and gates,One or gate,One nand gate,Probes.

**Procedure :**

1.Open a new blank schematic in multisim and place all the components required.

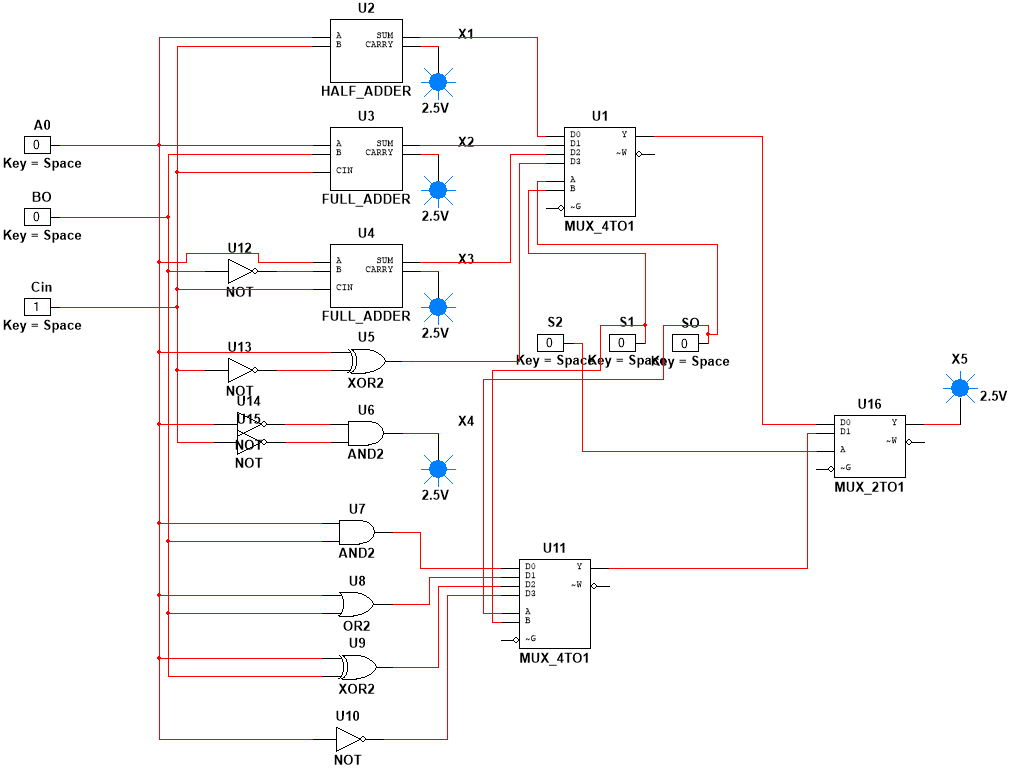
2.Connect all the components with the help of circuit diagram.

3.Give inputs to the Arithmetic unit by using full adders and half adders and to gates by using input keys(switches).

4.Connect probes to the output lines of multiplexer and wherever it is need.

5.Run the schematic and check the implementation of ALU by varying the inputs according to the truth table.

**Multisim circuit:**



**B] Assembly language program to find lcm**

**Code:**

org 100h

DATA SEGMENT

NUM DW 05,04

LCM DW 2 DUP(?)

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:

MOV AX,DATA

MOV DS,AX

MOV DX,0H

MOV AX,NUM

MOV BX,NUM+2

UP:

PUSH AX

PUSH DX

DIV BX

CMP DX,0

JE EXIT

POP DX

POP AX

ADD AX,NUM

JNC DOWN

INC DX

DOWN:

JMP UP

EXIT:

POP LCM+2

POP LCM

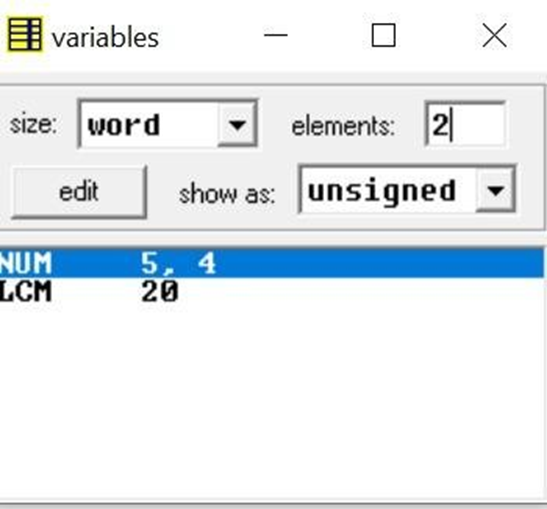
MOV AH,4CH

INT 21H

CODE ENDS

Ret

**Output:**



**LAB-8**

**VERIFICATION OF REGISTERS AND COUNTERS**

**REGISTERS:**

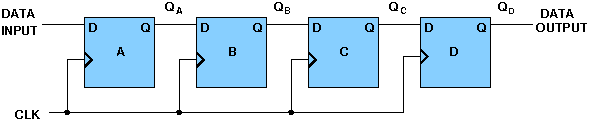
Registerss are high speed memory storing uints .It is an element of the computer processor.It can carry any type of information .The register used by cpu are after termed as processor register

.

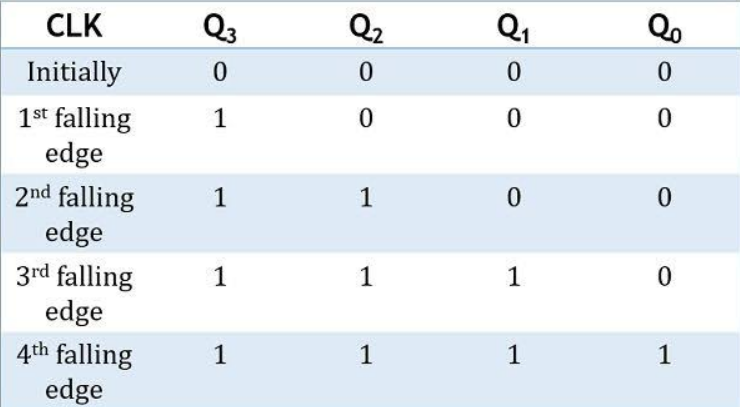
1. **4-BIT SERIAL IN & SERIAL OUT:**

4 bit serial-in serial-out register accepts digital data serially that is one bit at the time on one line. It produces the stored information on its output also in serial form. This is a shift register, as The binary number is "Shifted" one bit at time from one flip flop to the next.

**Cirucuite diagram :**



**Truth table :**



**Procedure :**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Key(1)

3] Clock (Digital Clock)

4] Probe

3.Connect all the components with the help of CIRCUIT diagram.

4.We have to disable the Set & Reset pins in D FlipFlops.So that you can start checking the circuit by taking input.

5.Probe will be glowed as it is serially passed from one Flip Flop to other Flip Flop.

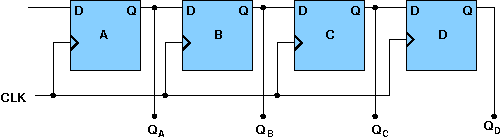
**Ciruite diagram in multisim :**



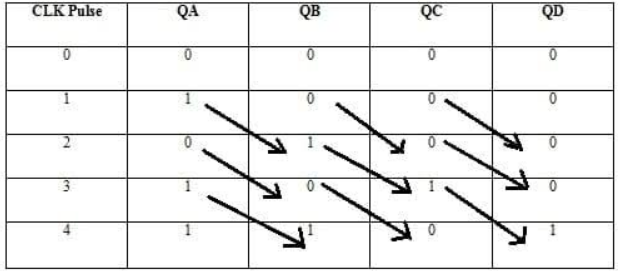
**2 .4-BIT SERIAL IN & PARALLEL OUT :**

In serial-in parallel-out register the data are loaded serially and read out in parallel.

**Circuit diagram:**



**Truth table :**



**Procedure :**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Key(1)

3] Clock (Digital Clock)

4] Probes(4)

3.Connect all the components with the help of CIRCUIT diagram.

4.We have to disable the Set & Reset pins in D FlipFlops.So that you can start checking the circuit by taking input.

5.Check the CIRCUIT by giving input 1,probe will be glow.

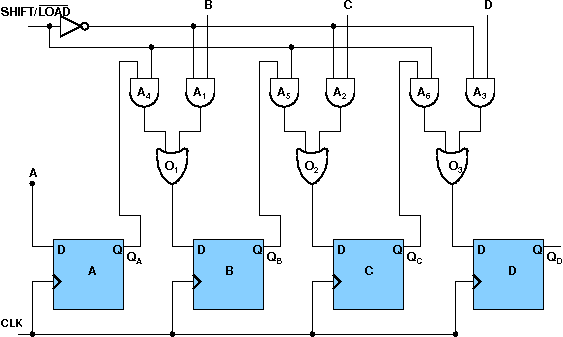
**Circuit diagram in multisim :**



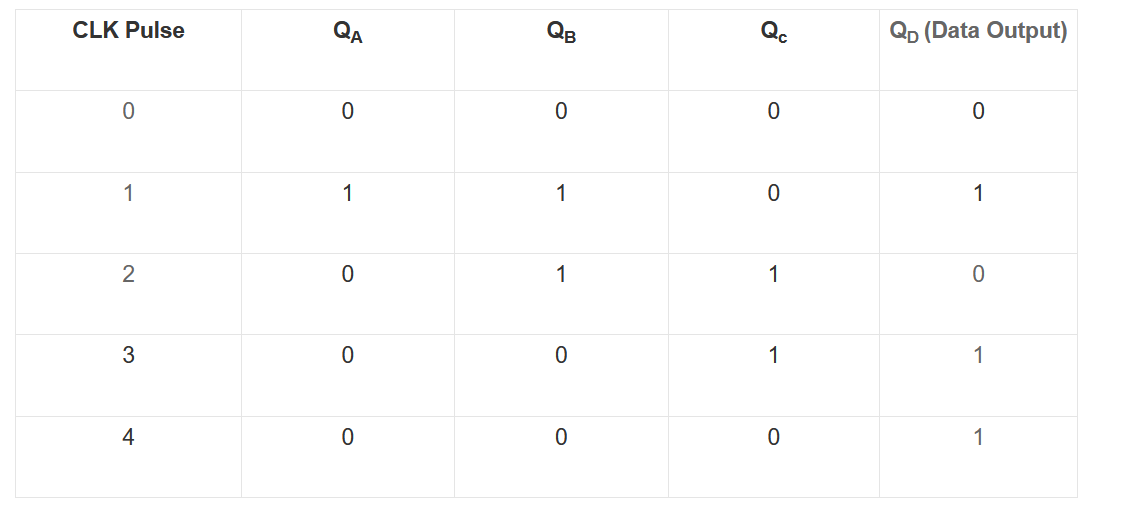
**3 .4-BIT PARALLEL IN SERIAL OUT :**

In parallel-in serial out register the bits are entered simultaneously into their respective stages on parallel-lines, rather than on a bit-by-bit basis on one line as with serial data inputs and output is read out out parallely.

**Circuit diagram:**



**Truth table:**



**Procedure :**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Key(1)

3] 2-i/p AND gates(6) and 2 -i/p OR Gates(3).

4] Probe

3.Connect all the components with the help of CIRCUIT diagram.

4.We have to disable the Set & Reset pins in D FlipFlops.So that you can start checking the circuit by taking input.

5.Check the CIRCUIT by giving 1 to the input for both A and Shift/Load inpute.

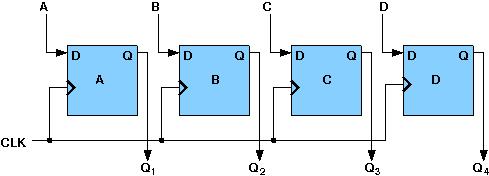
**CIRCUIT DIAGRAM IN MULTISIM :**



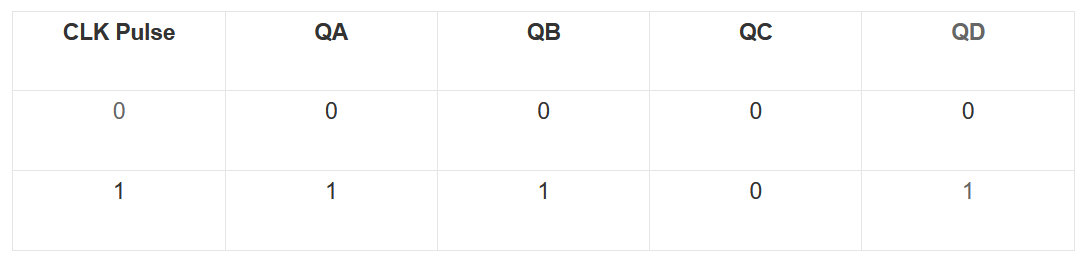
**4 .4-BIT PARALLEL IN PARALLEL OUT :**

In parallel-in parallel out register the data is loaded in parallel and shifted out serially.

**CIRCUIT DIAGRAM:**



**Truth table :**



**PROCEDURE :**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Key(4)

3] Clock (Digital Clock)

4] Probe(4)

3.Connect all the components with the help of CIRCUIT diagram.

4.We have to disable the Set & Reset pins in D FlipFlops.So that you can start checking the circuit by taking input.

5.Check the parallel -in parallel out register by giving the input 1 to verify the register.

**CIRCUIT DIAGRAM IN MULTISIM :**



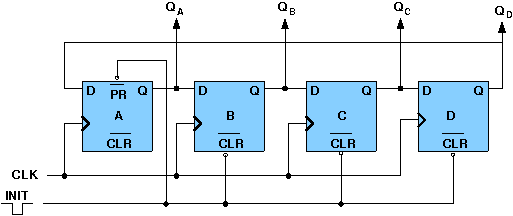
**COUNTERS**

In digital logic and computing a counter is a device which stores(and sometimes displays) the number of times a particular event or process has occurred often in relationship to a clock.

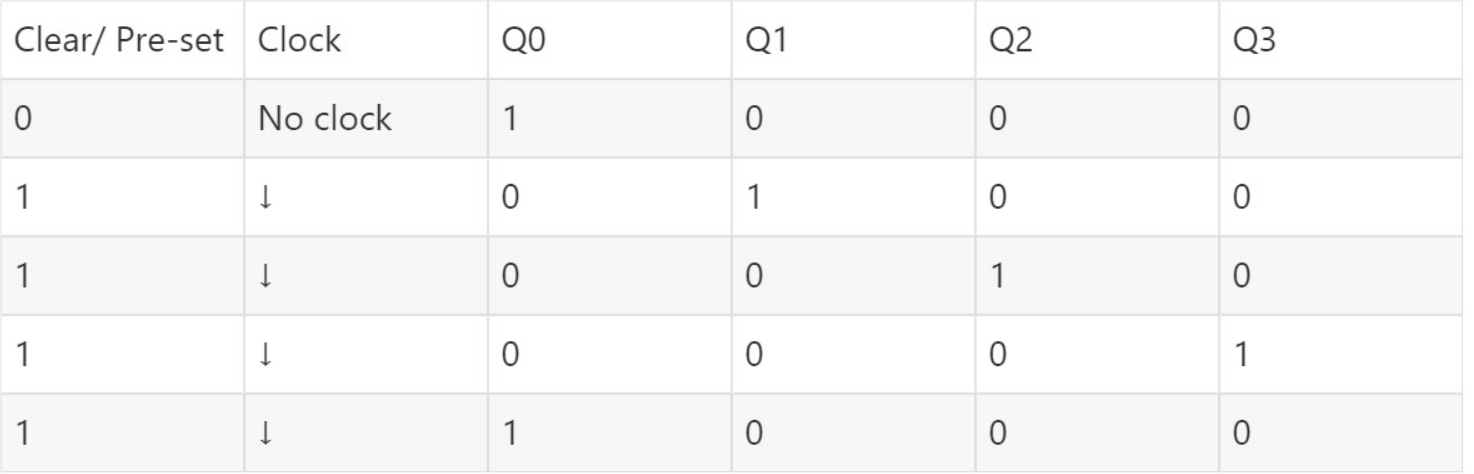
**4-BIT SYNCHRONOUS RING COUNTER :**

If the output of a shift register is fed back to the input. a ring counter results. The data pattern contained within the shift register will recirculate as long as clock pulses are applied.

**Ciruite diagram :**



**Truth table :**



**Procedure:**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Keys

3] Clock (Digital Clock)

4] Probe(4)

3.Connect all the components with the help of CIRCUIT diagram.

4.Disable the NQ pins.

5.Verify the 4 bit synchrounous ring counter by varying the inputs according to truth table.

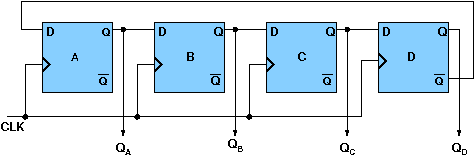
**Circuit in multisim :**



**4-BIT SYNCHRONOUS JOHNSON COUNTER :**

If the complement output of a ring counter is fed back to the input instead of the true output, a Johnson counter results. This "reversed" feedback connection has a profound effect upon the behavior of the otherwise similar circuits. Recirculating a single 1 around a ring counter divides the input clock by a factor equal to the number of stages. Whereas, a Johnson counter divides by a factor equal to twice the number of stages.

**Ciruite diagram :**



**Procedure:**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] D-FlipFlop(4)

2] Input Keys

3] Clock (Digital Clock)

4] Probe(4)

3.Connect all the components with the help of CIRCUIT diagram.

4.Connect a Flip Flop D pin to (D) flipflop to pin.

5.Verify the 4 bit synchrounous Jhonson counter by varying the inputs.

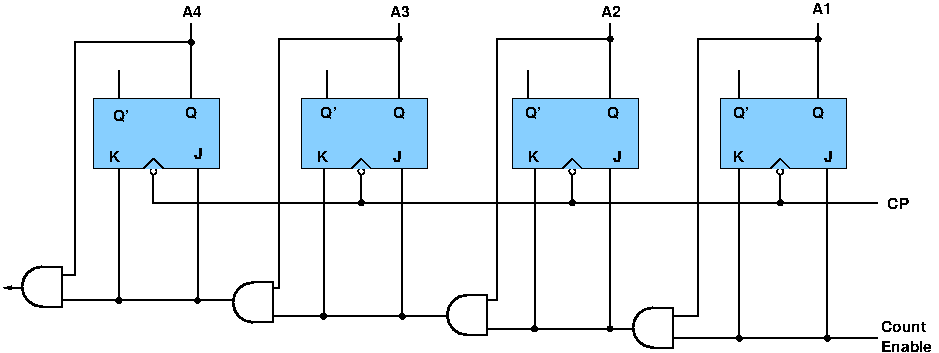
**Circuit in multisim :**



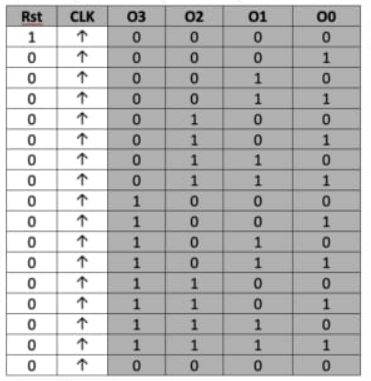
**4-BIT SYNCHRONOUS BINARY COUNTER :**

A counter is a sequential circuit that moves through a predefined sequence of states upon applying of clock pulses. The sequence of states may follow the binary number sequence or an arbitrary manner (no sequence). The simplest example of a counter is the binary counter which follows the binary number sequence. An n-bit binary counter contains n flip-flops and can count binary numbers from 0 to (2n -1)(up counter which is incremental, if it counts decrementally it is then down counter).

**Ciruite diagram:**



**Truth table :**



**Procedure:**

1. Open a new schematic in multisim and place all the components required.

2. Components required :

1] JK-FlipFlop(4)

2] AND Gates.(4)

3] Clock (Digital Clock)

4] Probe(1)

3.Connect all the components with the help of CIRCUIT diagram.

4.Verify the 4 Binary counter by varying the inputs.

**Circuit diagram in multisim:**

****

**B]Assembly language program to find gcd**

**Code:**

org 100h

DATA SEGMENT

NUM1 DW 000AH

NUM2 DW 0004H

GCD DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

UP: CMP AX,BX

JE EXIT

JB EXCG

UP1: MOV DX,0H

DIV BX

CMP DX,0

JE EXIT

MOV AX,DX

JMP UP

EXCG:XCHG AX,BX

JMP UP1

EXIT:MOV GCD,BX

MOV AH,4CH

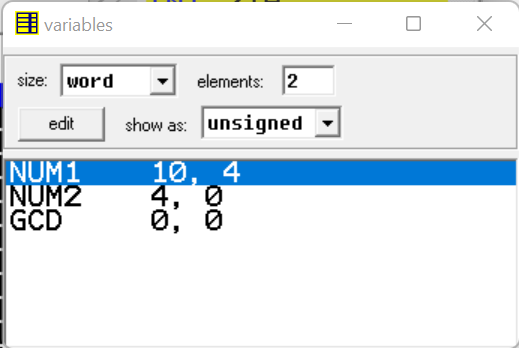
INT 21H

CODE ENDS

END START

Ret

**Output:**



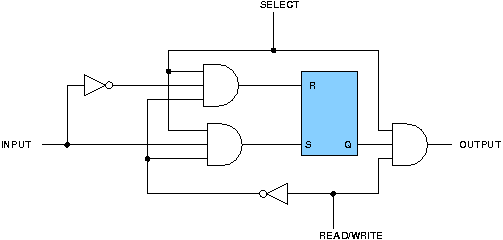
**LAB-9**

**MEMORY DESIGN**

**RAM DESIGN:**

The binary cell has three inputs and one output. The select input enables the cell for reading or writing and the reda/write input determines the cell operation when it is selected. A 1 in the read/write input provides the read operation by forming a path from the flip-flop to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the flip-flop.

**Circuit diagram:**



**Procedure :**

1.Take a new schematic in Multisim.

2.Go to the place menu, select the SR Flip-Flop, AND3 gate, and NOT gate, and then place the component on the screen.

3.Now take input sources and take a probe for output.

4.Connect them accordingly as shown in the circuit diagram.

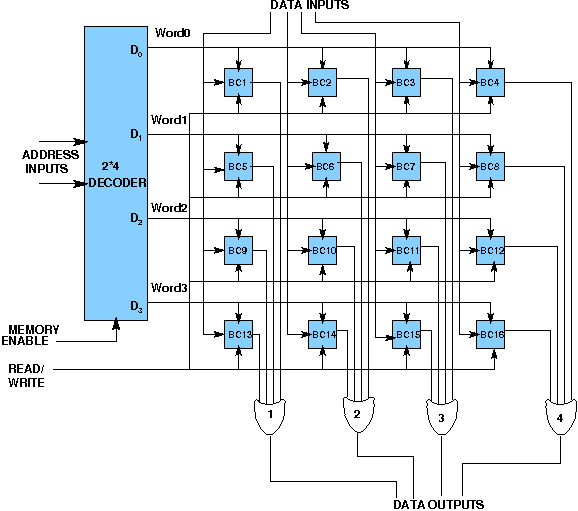
**Multisim circuit:**

****

**4X4 RAM DESIGN**

The logical construction of a small RAM 4X3 is shown below. It consists of 4 words of 3 bits each and has a total of 12 binary cells. Each block labeled BC represents the binary cell with its 3 inputs and 1 output.

**Circuit diagram:**



**Procedure:**

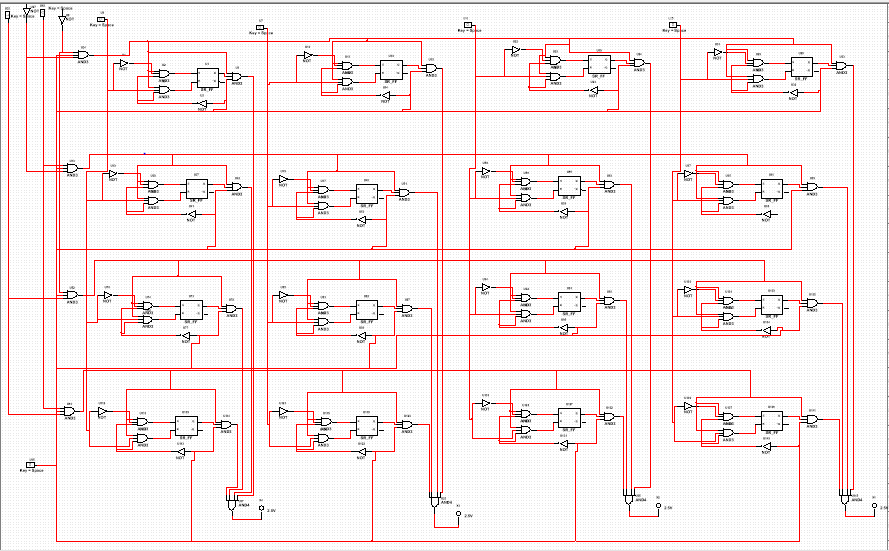
1.Take a new schematic in Multisim.

2.Go to the place menu, select the SR Flip-Flop, AND3 gate, OR gate, and NOT gate, and then 3.place the component on the screen.

4.Now take input sources and take a probe for output.

5.Connect them accordingly as shown in the circuit diagram.

**Multisim circuit :**



**B]Assembly language program to search an element using**

**linear search.**

**Code:**

org 100h

.8086

.stack 100h

.data

a db 01h ,02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 10h

element db 05h

.code

mov cx,10

lea bx,a

mov si,0

mov dx,0

label1: mov al,[bx+si]

cmp al,element

je equal

inc si

loop label1

not\_equal: mov dl,00h

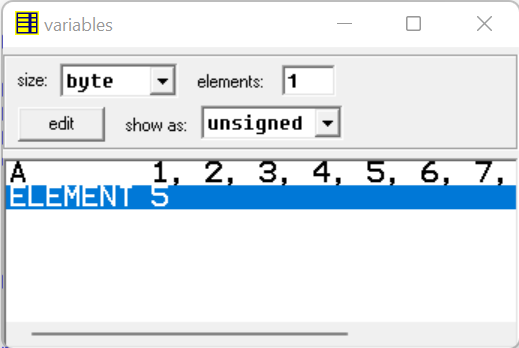
jmp exit

equal: mov dl,0ffh

exit:

ret

**Output:**



**LAB- 10**

**A]DIRECT MAPPED CACHE DESIGN**

**Description :**

A given Memory block can be mapped into one and only cache line.

Block Identification : Let the main memory contains n blocks and

cache contains m blocks ,so n/m different blocks of memory can be

mapped to a cache block.

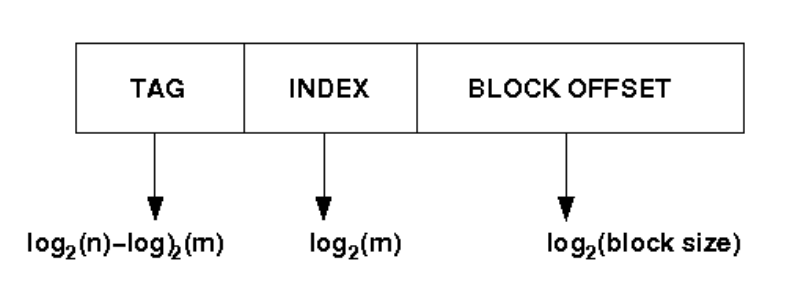
Number of bits in the tag : log2(n/m)

Number of sets in the Cache : m

Number of bits to identify the correct set : log2(m)

The Memory address is divided into 3 parts – tag(most MSB), index,

block offset(most LSB) in order to do the cache mapping.

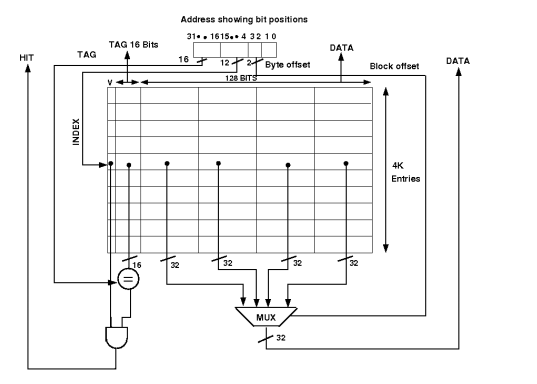


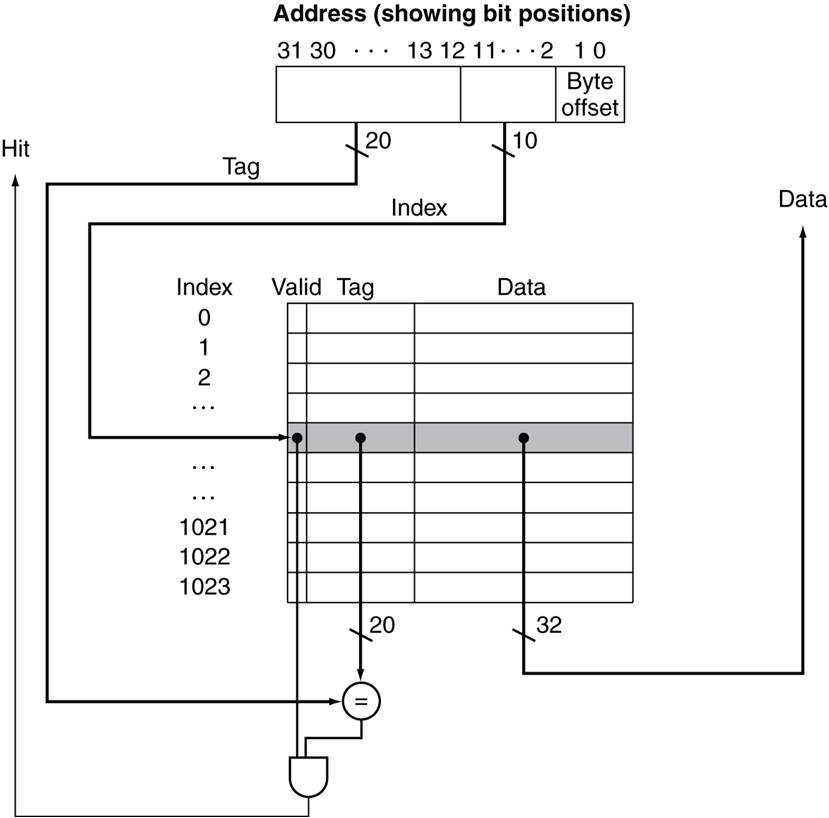
• Select set using index, block from set using tag.

• Select location from block using block offset.

• Tag + index = block addressDiagram of direct mapped cache (here main memory address is of 32

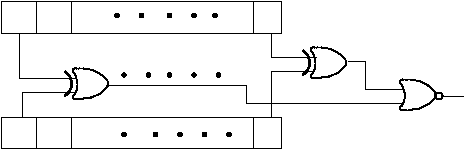
bits and it gives a data chunk of 32 bits at a time)



****

The Comparator Circuit through which tag is compared with specified

bits of address.



**B]Assembly language program to search an element using binary search**

**CODE:**

org 100h

DATA SEGMENT

ARR DW 05H,0111H,2161H,4541H,7161H,8231H

SR EQU 4541H

MSG1 DB 'ELEMENT FOUND AT '

RES DB ' RD POSITION','$'

MSG2 DB 'ELEMENT NOT FOUND','$'

DATA ENDS

ASSUME CS:CODE,DS:DATA

CODE SEGMENT

START: MOV AX,DATA

MOV DS,AX

MOV BX,00H

MOV CX,SR

MOV DX,05H

LP: CMP BX,DX

JA FAILURE

MOV AX,BX

ADD AX,DX

SHR AX,01

MOV SI,AX

ADD SI,SI

CMP CX,ARR[SI]

JAE BIGGER

DEC AX

MOV DX,AX

JMP LP

BIGGER: JE SUCCESS

INC AX

MOV BX,AX

JMP LP

SUCCESS:ADD AL,01H

ADD AL,2FH

MOV RES,AL

LEA DX,MSG1

JMP DISPLAY

FAILURE: LEA DX,MSG2

DISPLAY: MOV AH,09H

INT 21H

MOV AH,4CH

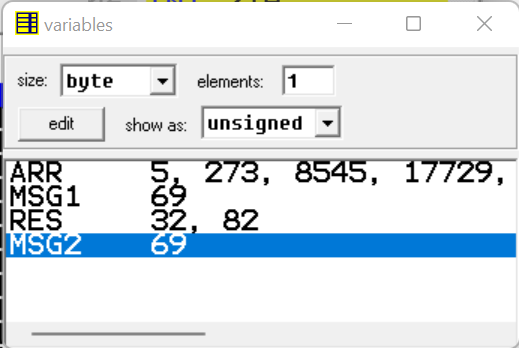
INT 21H

CODE ENDS

END START

Ret

**Output :**

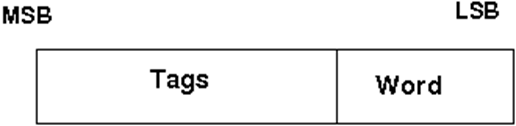


**LAB - 11**

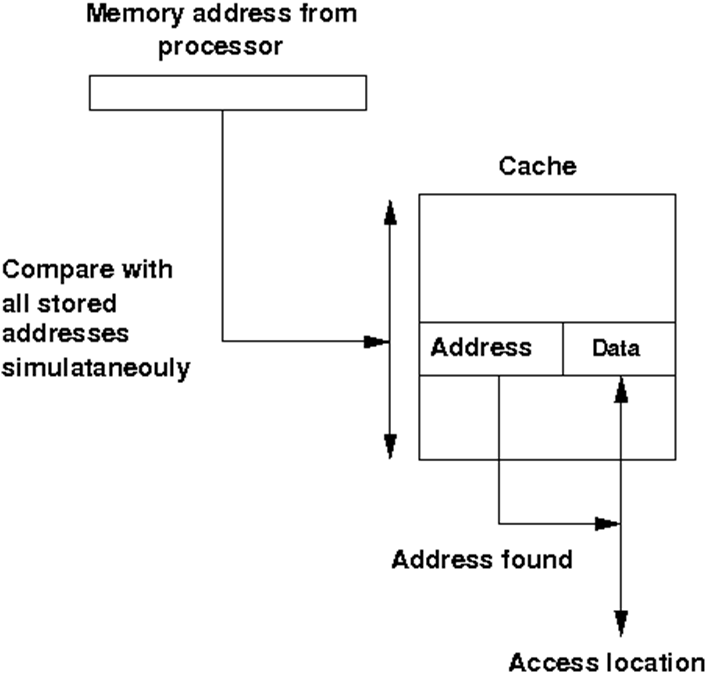
**A] ASSOCIATIVE CACHE DESIGN**

**ASSOCIATIVE CACHE :**

Any main memory block can be mapped into any cache line. Main memory address is divided into two groups which are tags and word bits. Words are low order bits and identify the location of a word within a block and tags are high order bits which identifies the block.



**Block diagram of associated cache:**

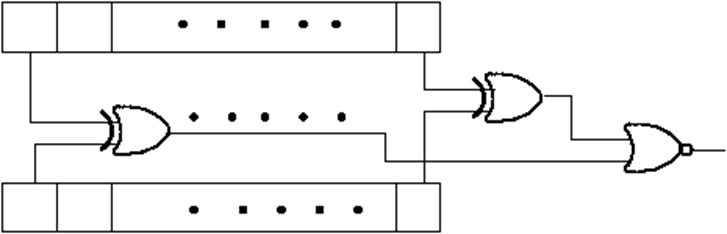


If a miss occurs CPU brings the block from the main memory to the cache, if there is no free block in the corresponding set it replaces a block and puts the new one. CPU uses different replacement policies to decide which block is to replace. The disadvantage of the associative cache is its high cost for implementing parallel tag comparison, but suffer the most from thrashing due to the ‘conflict misses’ giving more miss penalty.

No replacement policy has been implemented in the experiment.

The Comparator circuit through which tag is compared with specified bits of address.

**Circuit diagram:**



**B]Assembly language program to sort numbers using bubble sort.**

**Code:**

.model small

.data

arr db 71h, 36h, 48h, 18h, 65h

len dw 5

.code

main proc

mov ax, @data

mov ds, ax

mov ax, 0 ;outer loop

outer:

mov di, 0

inner:

mov bl, [arr + di] ;arr[j]

mov bh, [arr + di + 1] ;arr[j+1]

cmp bl, bh

jbe innerRest

mov [arr + di], bh

mov [arr + di + 1], bl

innerRest:

mov si, len - 1

sub si, ax

inc di

cmp di,si

jne inner

inc ax

cmp ax, len - 1

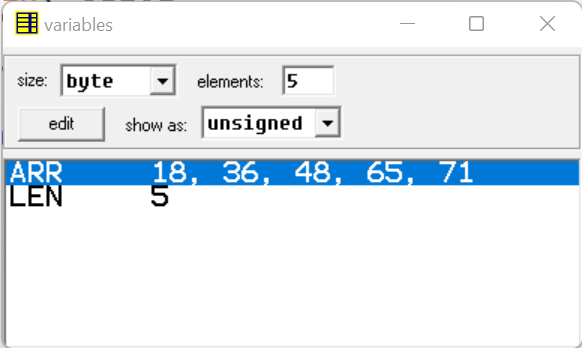
JNE outer

ret

main endp

End

**Output:**

****