TUSHAR NANKANI
1902112
C23



# ASSIGNMENT No. 1

Q1. Write a short note on cache mapping technique (Direct, Associative, set associative) with examples for each.

Ans. Cache mapping is a technique by which the content of the main memory are brought into the cache memories. There are 3 different types of mapping:

(i) Direct Mapping: The simplest way to determine cache location is in which store memory block is direct mapping. We assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block, then a new block needs to be loaded, the old block is trashed.

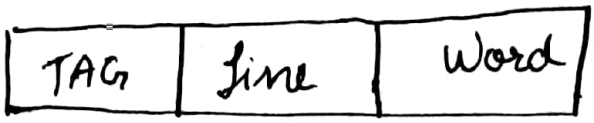
(ii) Associative Mapping: This is a more flexible mapping method, in which main memory block can be placed into any cache block position. While searching, the tag bits of an address required from the processor are compared to the tag bits of cache block to see, if desired block is present.

## (i) Direct Mapping

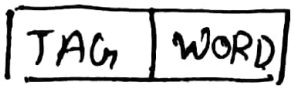
$$i = J \text{ modulo } m$$

where, i = cache line number.
J = main memory block number.
m = number of lines in the cache.

main memory ⇒




## (ii) Associative mapping




## (iii) Set-associative mapping

(i) Examples of Direct mapping:

eg: Computer has 32 bit MM address space

- Cache size = 8KB = $8 \times 2^{10} = 2^{13}$B
- Block size = 16B = $2^4$B

∴ No of words = $2^{32}$ (∵ 32 bit, given)

∴ No of blocks = $2^{32}/2^4 = 2^{28}$ blocks in MM

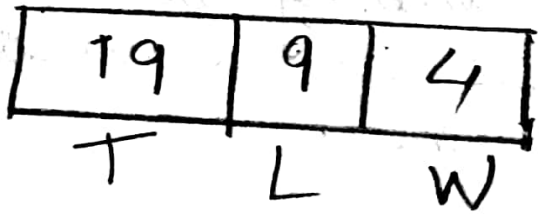
∴ No of line fields = $2^{13}/2^4 = 2^9$ lines in cache

Ans: line field ($2^9$) = 9 bits

word field ($2^4$) = 4 bits

∴ TAG = 32 − 9 − 4 = 19 bits.

MM address



(contd..)
This is known as associative mapping technique is higher than direct mapping, since searching time is increased. This is called an associative mapping.

(iii) **Set - associative mapping:** This form of mapping is an enhanced form of direct mapping and associative mapping. A block is mapped to in the cache, and then the block of memory is mapped to any of the lines that are grouped together.

This type of mapping combines the best of direct and associative cache mapping technique.

* $m = v \times K$
* $i = J \bmod v$

where $i$ = cache set number, $J$ = MM block no.
$v$ = number of sets
$m$ = number of lines in ~~one~~ all sets
$K$ = no of lines in each set.

## (ii) Examples of associative mapping.

eg: $n = 32$ bit
cache size = 8 KB = $2^{13}$ B
block size = 16 B = $2^4$ B

∴ Word field = $2^4$ = 4 bits
TAG = 32 − 4 = 28 bits

∴ MM add ⇒

| 28 | 4 |
|---|---|
| TAG | WORD |

## (iii) Example of set-associative mapping.

eg: $n$ = 32 bits = $2^5$ B
$k$ = 4 bits = $2^2$ B
cache size = 8 KB = $2^{13}$ B
block size = 16 B = $2^4$ B.

∴ No. of lines in cache = $2^{13}/2^4 = 2^9$ lines

∴ Total no. of sets in cache = $2^9/2^2 = 2^7$ sets.

Ans: Word ($2^4$) = 4 bits
set field ($2^7$) = 7 bits.
TAG ($32-7-4$) = 21 bits

| TAG | SET | WORD |
|---|---|---|
| 21 | 7 | 4 |

TUSHAR NANKANI
1902112
C23



# Assignment No. 2

Q2. Write a short note on multicore architecture.

Ans. Multicore refers to an architecture in which a single physical processor incorporates the core logic of more than 1 processor. A single integrated circuit is used to package or hold these processor. Multicore architecture can be homogeneous or heterogeneous.

Homogeneous processors are those that have identical cores on the chip, whereas heterogeneous have different types of chip on the cores.

A multicore processor implements multi-processing in a single physical package. Designer may or may not share cache and they may implement message passing or shared memory inter-core communication methods.

26/08/2020

TUSHAR NANKANI
1902112
C23

37

# ASSIGNMENT No. 1(a).

**AIM:** Write a program to accept a decimal number, and convert it to binary and vice-versa.

**THEORY:**

- The easiest way to convert a decimal to a binary is using repeated division by 2 and appending its remainder to form the binary number.

- The easiest method to convert binary to decimal is by repeated multiplication by powers of 2, starting from 0, and adding them up to form the decimal number.

1. Decimal ⟶ Binary.

eg: $(41)_{10} = (?)_2$

| | | | |
|---|---|---|---|
| 2 | 41 | 1 | ⟶ LSB |
| 2 | 20 | 0 | |
| 2 | 10 | 0 | |
| 2 | 5 | 1 | |
| 2 | 2 | 0 | |
| | 1 | 1 | ⟶ MSB |

$(41)_{10} = (101001)_2$

2. Binary to Decimal.

eg. $(101011)_2 = (?)_{10}$

$= 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5$

$= 1 + 2 + 8 + 32$

$= \underline{43}$

$\therefore (101011)_2 = (43)_{10}$

Eg.1: $(8)_{10}$ to binary:

| | | |
|---|---|---|
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| . | 1 | 1 |

Ans ⇒ 01000

Eg.2: $(-9)_{10}$ to binary.

- Converted magnitude to binary.

| | | |
|---|---|---|
| 2 | 9 | 1 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | 1 |

01001

↓

- taking is complement

10110

↓

- adding 1

10110

\+ 1

10111

# ASSIGNMENT No. 1(b)

AIM: Accept a signed decimal number (positive/ negative) and find the 2's complement of the number.

THEORY:

- For positive:

1) Convert the given decimal to binary.

2) Add '0' to the left of MSB.

- For negative:

1) Convert magnitude to binary.

2) Add 0 to the left of MSB.

3) Invert bits to achieve 1's complement.

4) Add 1 to achieve 2's complement.

Name: Tushar Nankani Roll No: 1902112 Batch: C23

Experiment 1: Decimal to binary and vice versa conversion and 2's complement

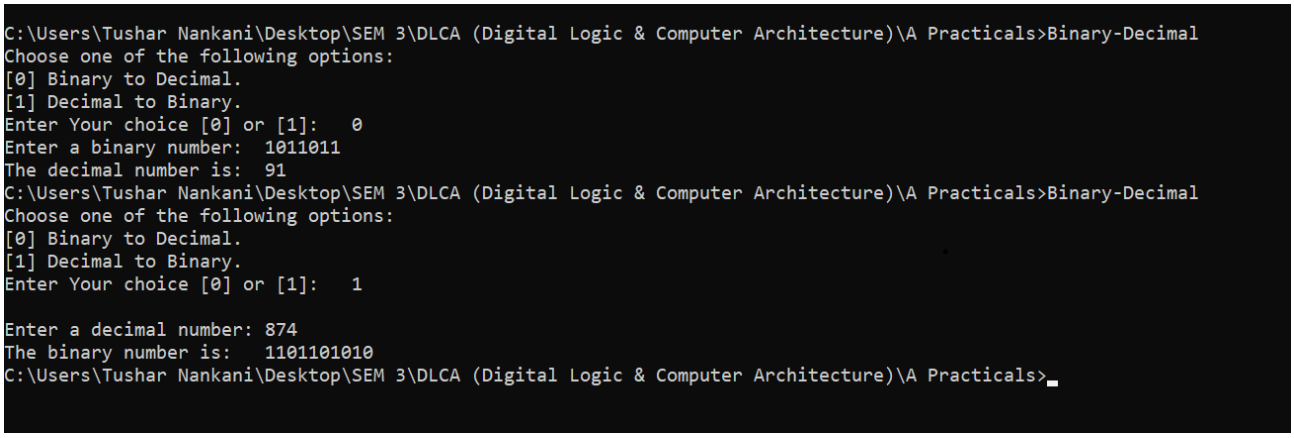
**CODE:**

```
#include <conio.h>
#include <stdio.h>
#include <math.h>

void binary()
{
    // Decimal to Binary;
    int n, a[50], i, flag = 0;
    printf("\nEnter a decimal number:\t");
    scanf("%d", &n);

    // for positive;
    if(n > 0)
    {
        for(i=0; n>0; i++)
        {
            a[i] = n % 2;
            n = n / 2;
        }
        printf("The binary number is: \t");
        for(i = i - 1; i >= 0; i--)
        {
            printf("%d", a[i]);
        }
    }
    // For negative;
    else
    {
        n *= -1;
        for(i = 0; n>0; i++)
        {
            a[i] = n % 2;
            n = n / 2;
        }
        //
```

```
    a[i] = 0;
    int t = i;

    // debug
    for(i = t; i >= 0; i--)
        printf("%d", a[i]);
    printf("\n");
    // debug

    for(i = t; i >= 0; i--)
        a[i] = !a[i];
        // 1 --> 0

    // debug
    for(i = t; i >= 0; i--)
        printf("%d", a[i]);
    printf("\n");
    // debug

    int carry = 0;

    if(a[0] == 1) {
        a[0] = 0;
        carry = 1;
    }
    else {
        a[0] = 1;
        carry = 0;
    }
    for(i = 1; i <= t; i++) {
        if(a[i] && carry)
            a[i] = 0, carry = 1;
        else if(a[i] == 0 && carry)
            a[i] = 1, carry = 0;
    }

    printf("The binary number is: \t");
    if(carry)
        printf("1");
    for(i = t; i >= 0; i--)
        printf("%d", a[i]);
}
```

```
}
void decimal()
{
    int ans = 0, i = 0, r, n;
    printf("Enter a binary number:\t");
    scanf("%d", &n);
    while (n!=0)
    {
        r = n%10;
        n /= 10;
        ans += r * pow(2,i);
        ++i;
    }
    printf("The decimal number is:\t");
    printf("%d", ans);
}


void main()
{
    printf("Choose one of the following options:\n");
    printf("[0] Binary to Decimal.\n");
    printf("[1] Decimal to Binary.\n");
    printf("Enter Your choice [0] or [1]: \t");
    int choice;
    scanf("%d", &choice);
    if(choice)
        binary();
    else
        decimal();
}
```

**OUTPUT:**


C:\Users\Tushar Nankani\Desktop\SEM 3\DLCA (Digital Logic & Computer Architecture)\A Practicals>Binary-Decimal
Choose one of the following options:
[0] Binary to Decimal.
[1] Decimal to Binary.
Enter Your choice [0] or [1]: 0
Enter a binary number: 1011011
The decimal number is: 91
C:\Users\Tushar Nankani\Desktop\SEM 3\DLCA (Digital Logic & Computer Architecture)\A Practicals>Binary-Decimal
Choose one of the following options:
[0] Binary to Decimal.
[1] Decimal to Binary.
Enter Your choice [0] or [1]: 1
Enter a decimal number: 874
The binary number is: 1101101010
C:\Users\Tushar Nankani\Desktop\SEM 3\DLCA (Digital Logic & Computer Architecture)\A Practicals>

02/09/2020

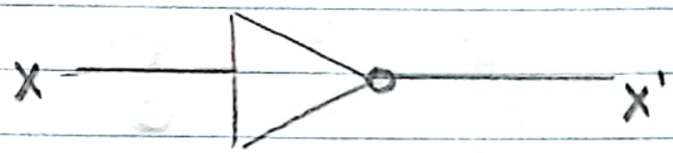
TUSHAR NANKANI
1902112
C23



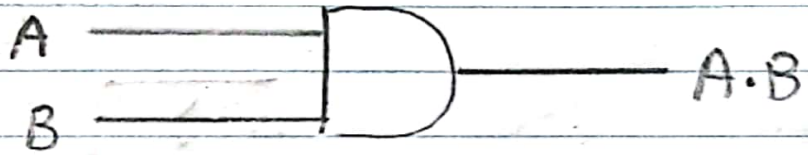
# ASSIGNMENT NO 2(a)

AIM: Verify the truth tables of the following gates (assume 2 inputs)

THEORY: 1. NOT gate: takes 1 input and inverts it.

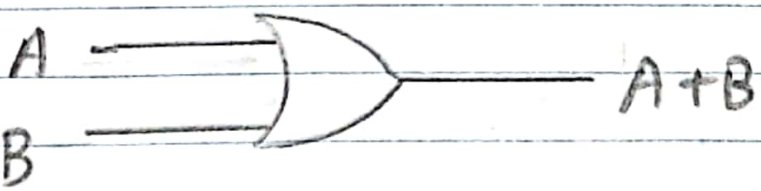



2. AND gate:




expression ⇒ A·B

3. OR gate:




expression ⇒ A+B

1. NOT

| X | $\overline{X}$ |
|---|---|
| 1 | 0 |
| 0 | 1 |

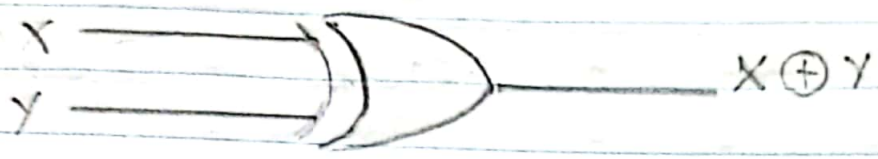
2. AND

| X | Y | X · Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

3. OR

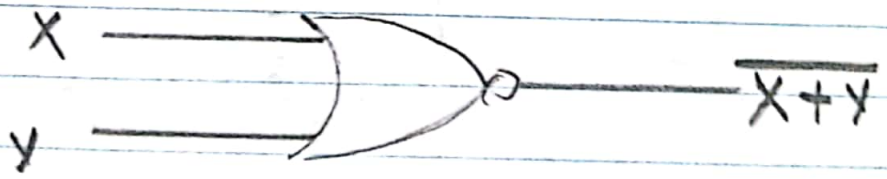
| X | Y | X + Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

4) XOR gate: - not a basic operation.
- output only true, when the inputs differ.




5) NAND gate: - the NOT-AND gate is NAND gate.


6) NOR gate: - the NOT-OR gate is NOR gate.



4) XOR

| X | Y | $X \oplus Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

5) NAND

| X | Y | $\overline{X \cdot Y}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

6) NOR

| X | Y | $\overline{X + Y}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

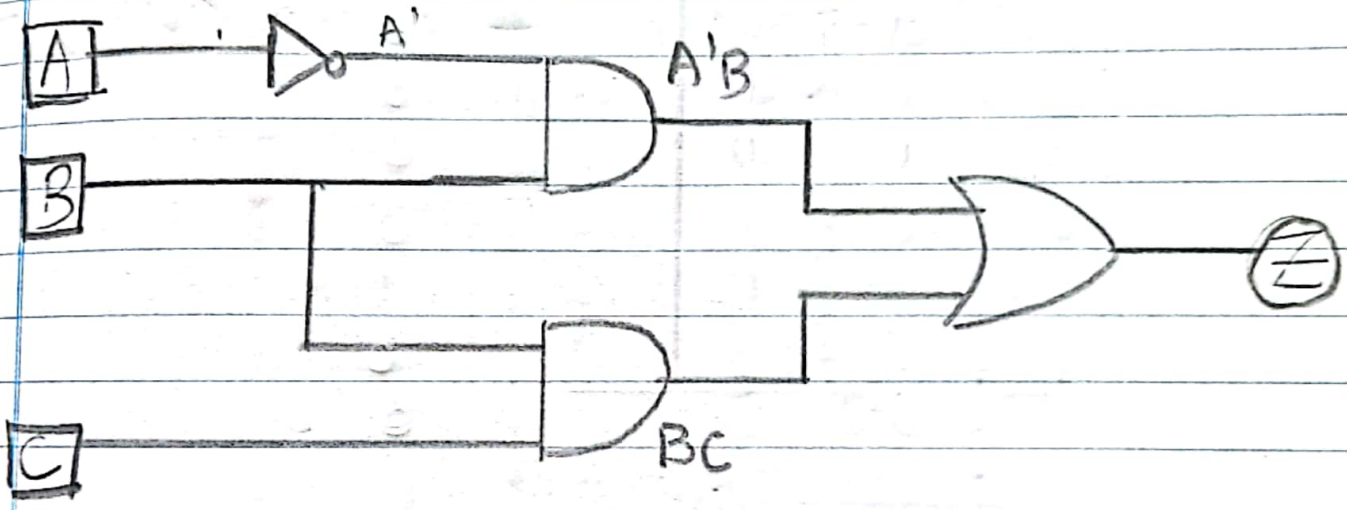
TUSHAR NANKANI
1902112
C23



# ASSIGNMENT No. 2(b)

AIM: Design a logic circuit to realize the expressions: 1) $A'B + BC$
2) $A(B+C) + A'BCD$.

THEORY: 1) $A'B + BC$

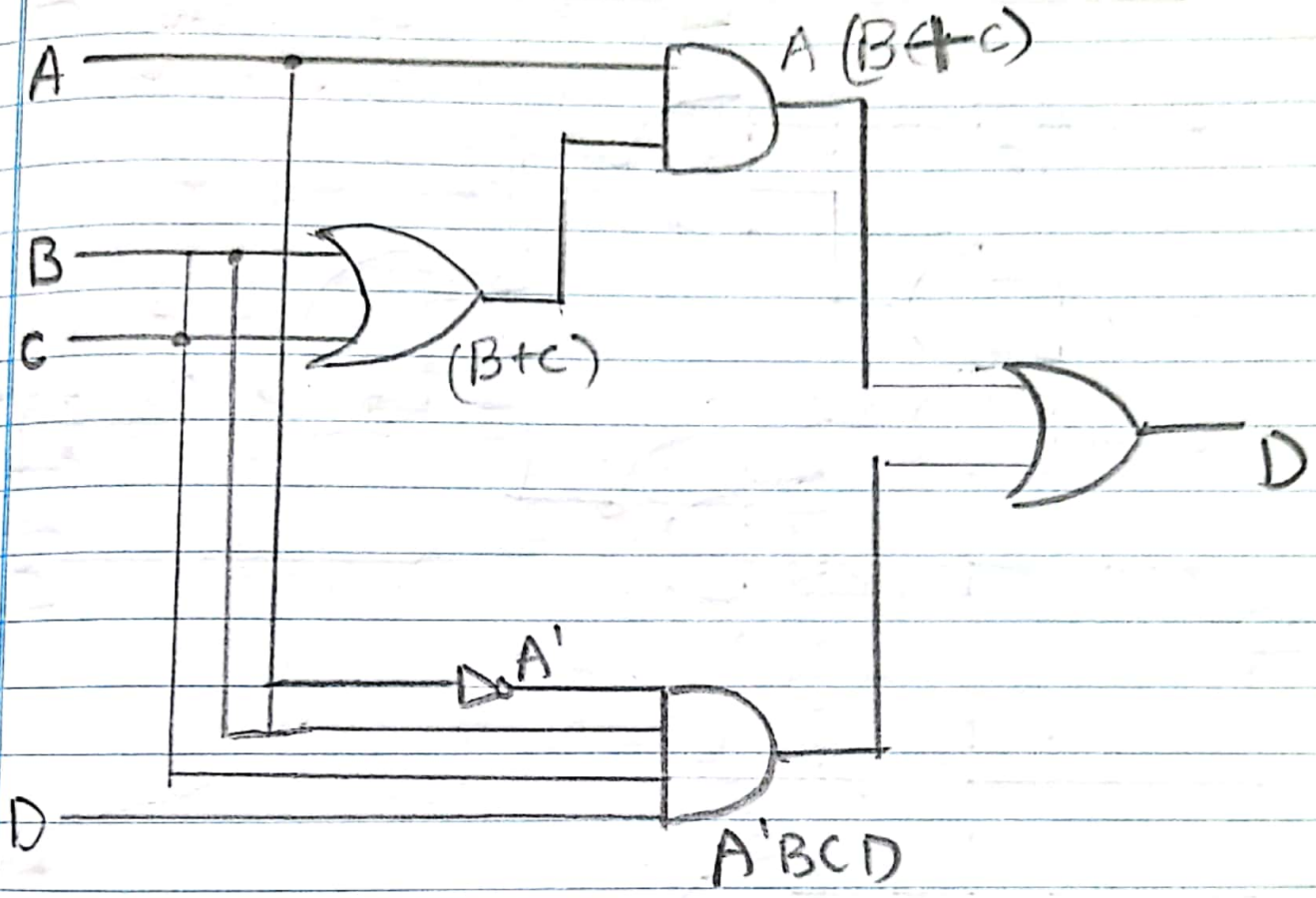



$$Z = A'B + BC$$

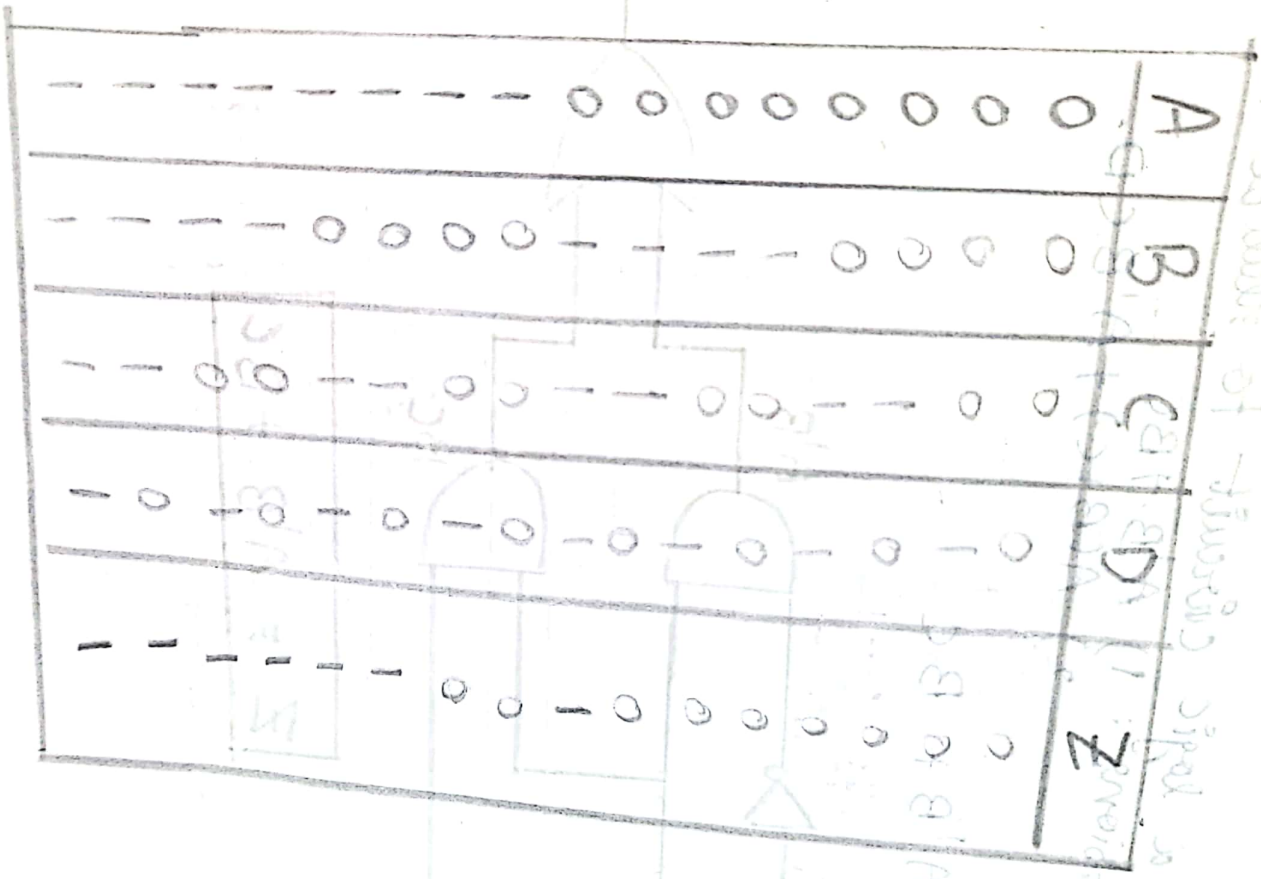
# TRUTH TABLES:

| A | B | C | A'B+BC |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

(ii) $A(B+C) + A'BCD$.

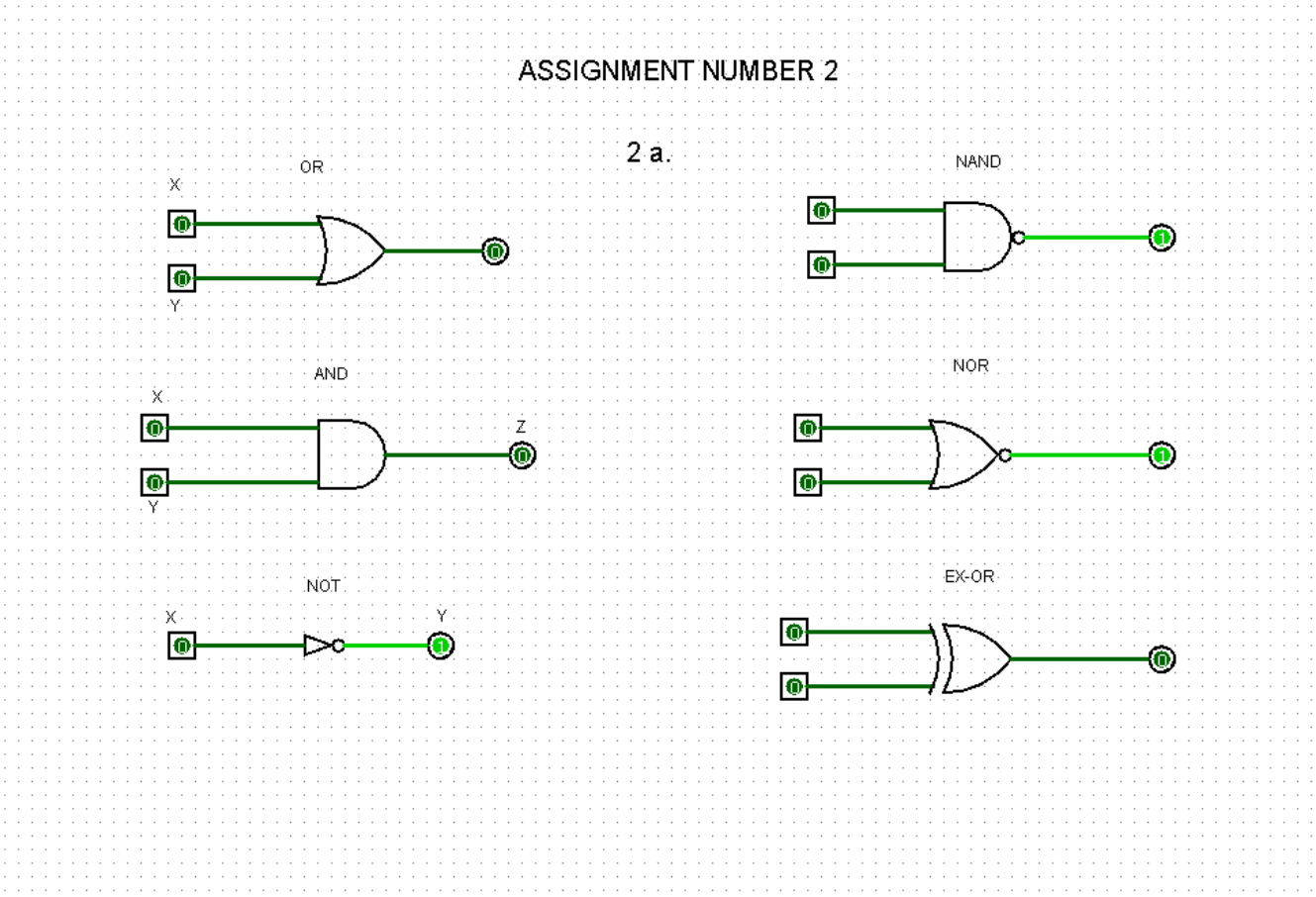



$$Z = A(B+C) + A'BCD$$



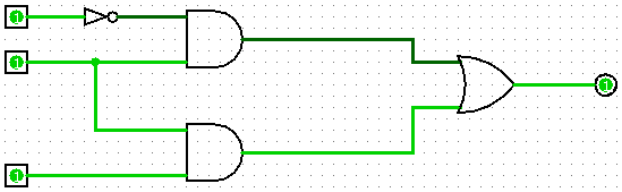
ASSIGNMENT NUMBER 2

2 a.

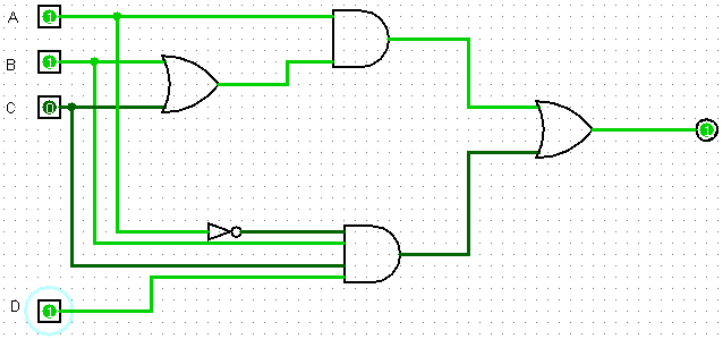


ASSIGNMENT NUMBER 2

i. A'B + BC



ii. A(B + C) + A'BCD



09/09/2020

TUSHAR NANKANI
C23
1902112



# ASSIGNMENT No 3

**AIM:** Using the Logisim Simulator - implement NOT, AND, OR & EXOR gates using the universal gates NAND and NOR.

**THEORY:** - A universal logic gate is a logic gate that can be used to construct all other logic gates.

- Nand and Nor are called universal gates, i.e, any logic gates such as AND OR, EXOR can be made using the NAND and NOR gates.

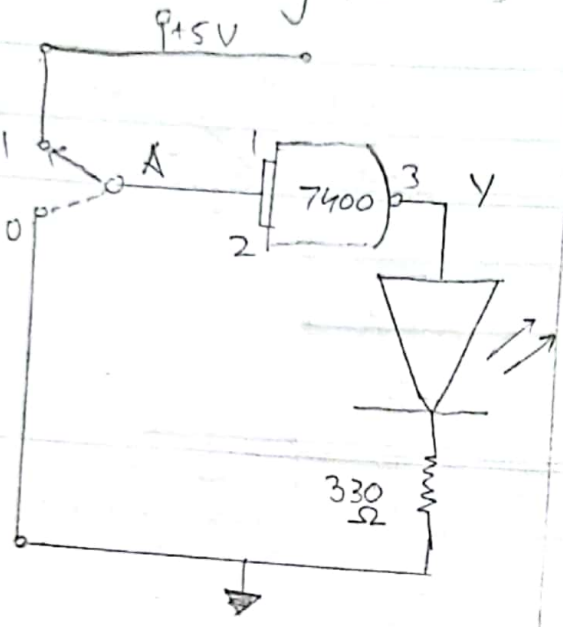
- To implement NAND as AND and OR, we will use the Demorgan's theorem:

(1) $\overline{X}\ \overline{Y} = \overline{\overline{X} + \overline{Y}} = X + Y$ (OR)

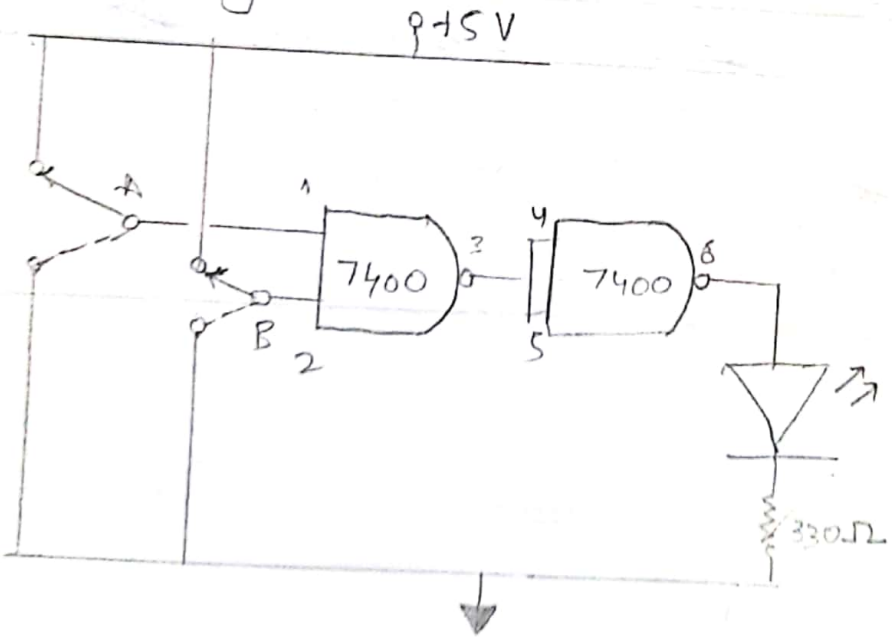
(2) $\overline{\overline{X} + \overline{Y}} = \overline{\overline{X}} \cdot \overline{\overline{Y}} = X \cdot Y$ (AND)

(A)

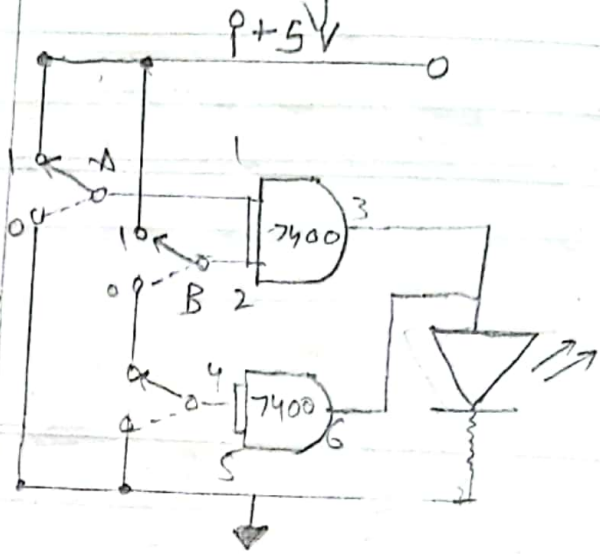
I) NOT using NAND:




II) AND using NAND:

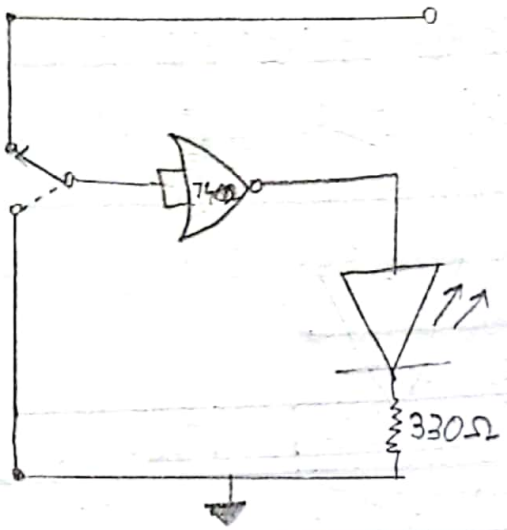



III) OR using AND:

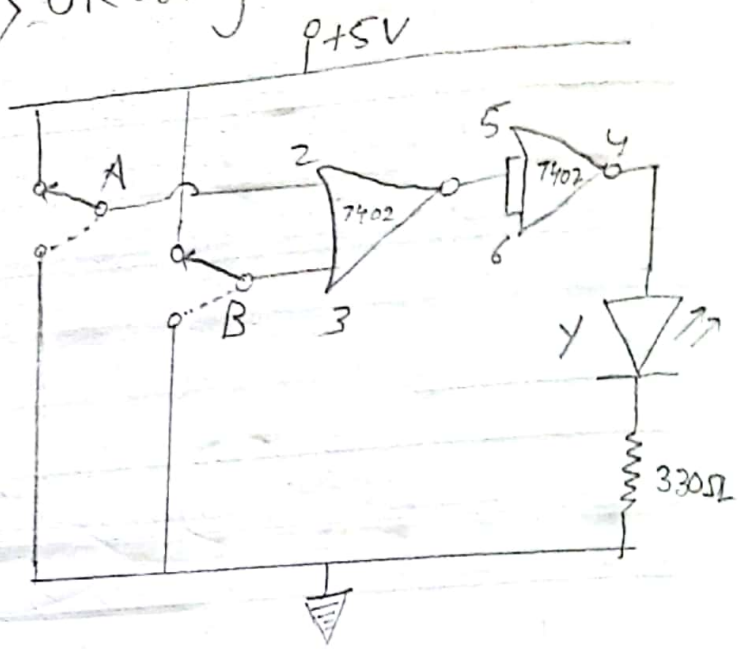


(B)

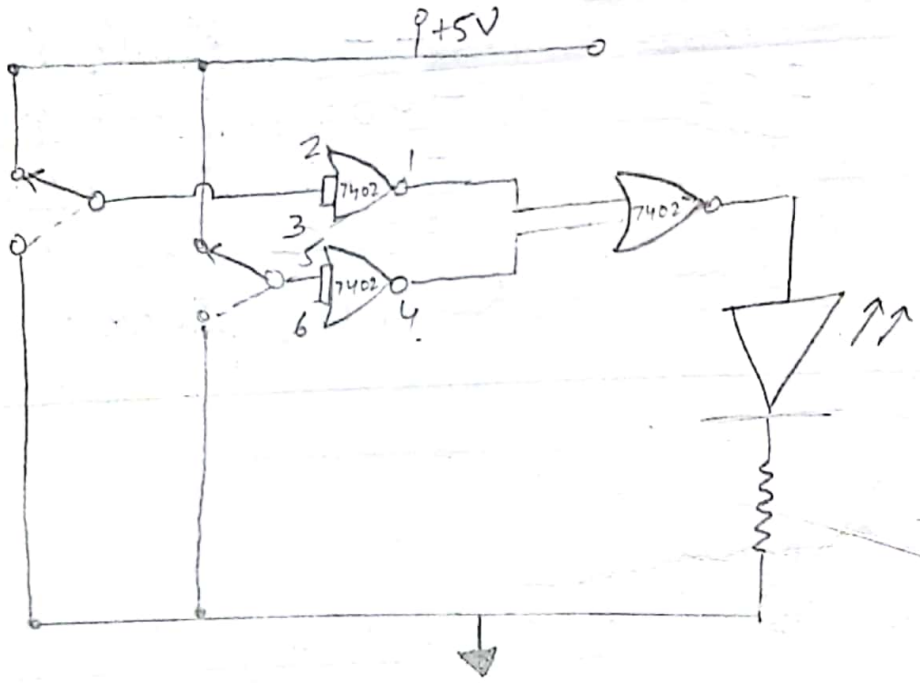
1) NOT using OR




2) OR using NOR.




3)



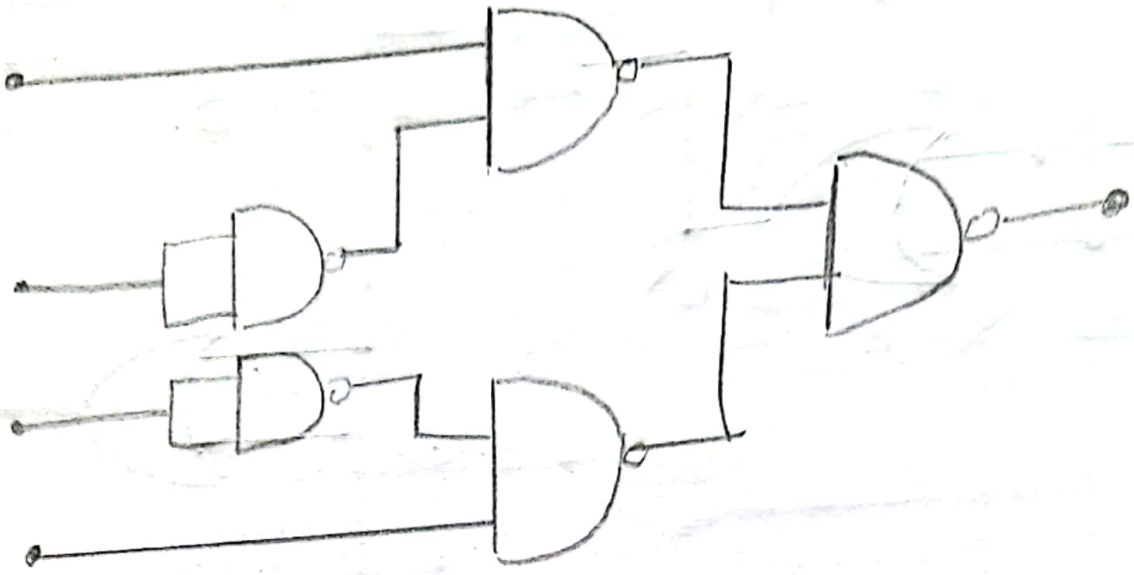

AND using NOR.

⇒ Advantages of Universal gates :

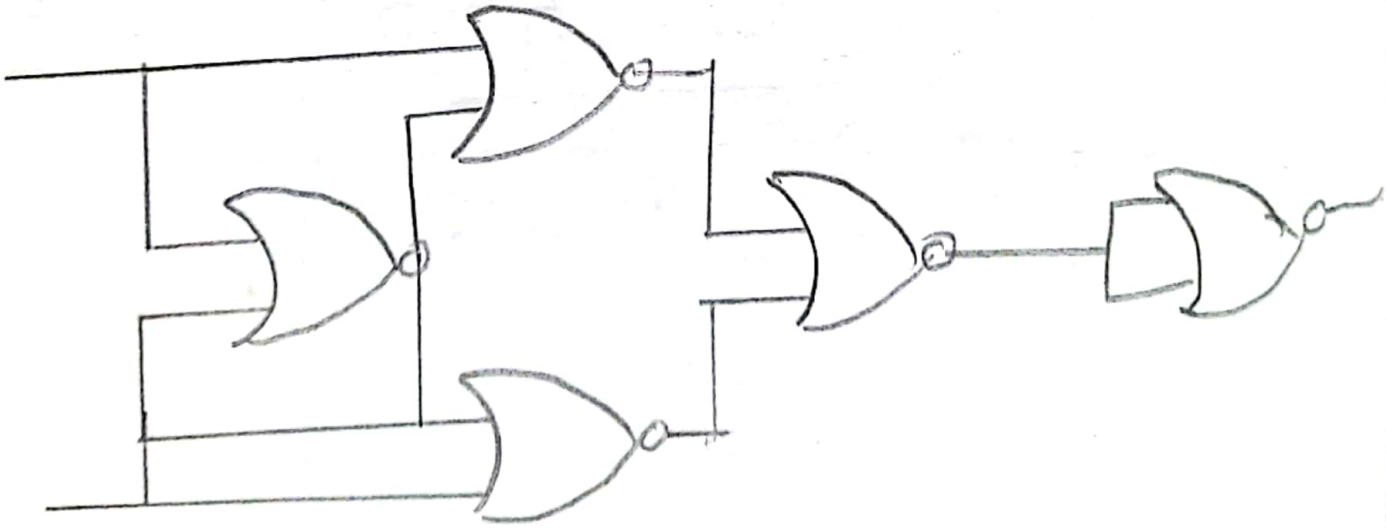
1. Using a single gate type, i.e. NAND or NOR will reduce the number of Integrated Circuits (ICs) required to implement the logic circuit.

2. Lesser the ICs required, less will be the total cost.

## NAND gate as EXOR:

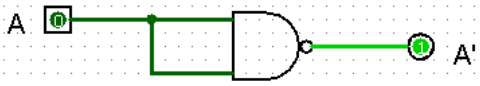


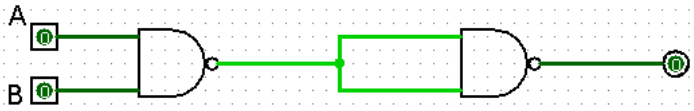
## NOR gate as EXOR:

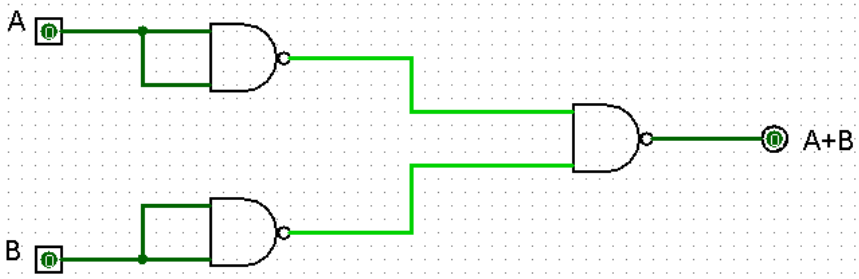


DLCA Assignment No. 3

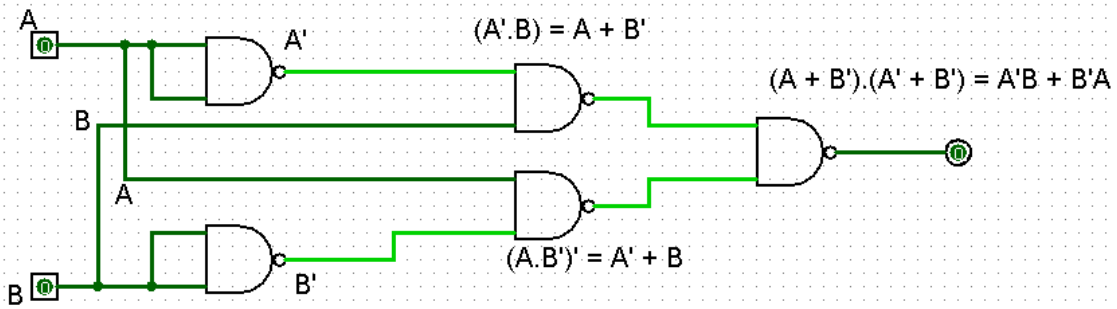
NAND

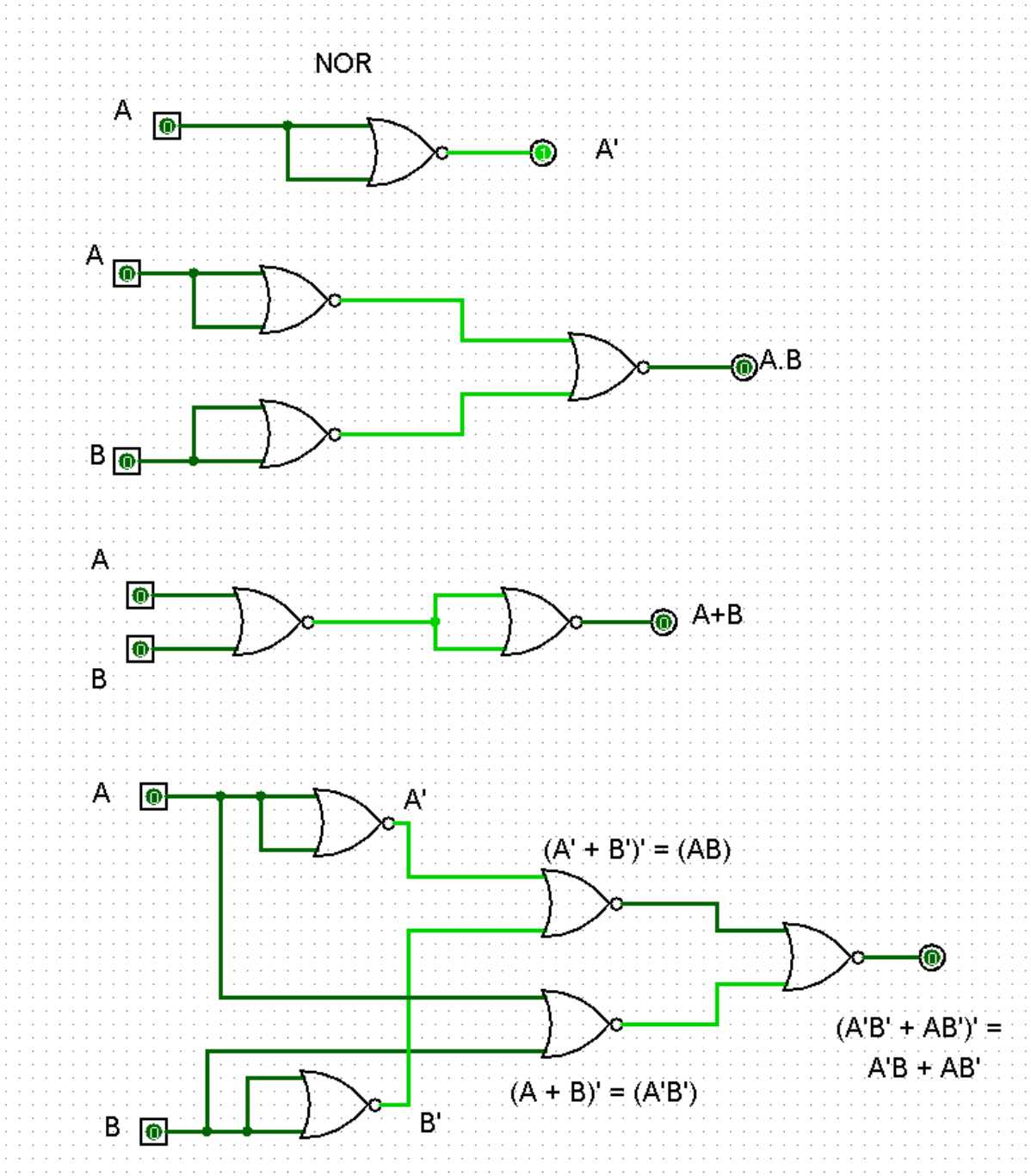



A'B + AB'




NOR
A
A'
A
A.B
B
A
A+B
B
A
A'
(A' + B')' = (AB)
(A'B' + AB')' =
A'B + AB'
(A + B)' = (A'B')
B
B'

16/09/2020

TUSHAR NANKANI
1902112
C23



# Assignment No. 4

**AIM:** Using Logisim Simulator - Implement Half-adder and Full adder.

**THEORY:**

(i) An adder is a digital circuit that performs addition of numbers.

(ii) In many computers and other kinds of processor adders are used in the ALU.

(iii) There are 2 adders, one is half adder and the other is full adder.

(iv) The half adder adds to single binary digits A and B and it has sum and carry.

(v) The truth table and digram is behind for half adder.

(vi) The half adder adds 2 inputs bits and it generates 2 things, one is carry and second is sum.

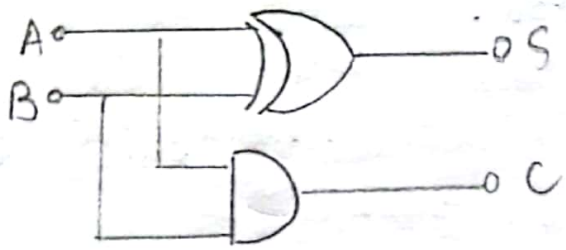
→ HALF-ADDER:

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

$C = A \cdot B$

$S = A'B + AB'$

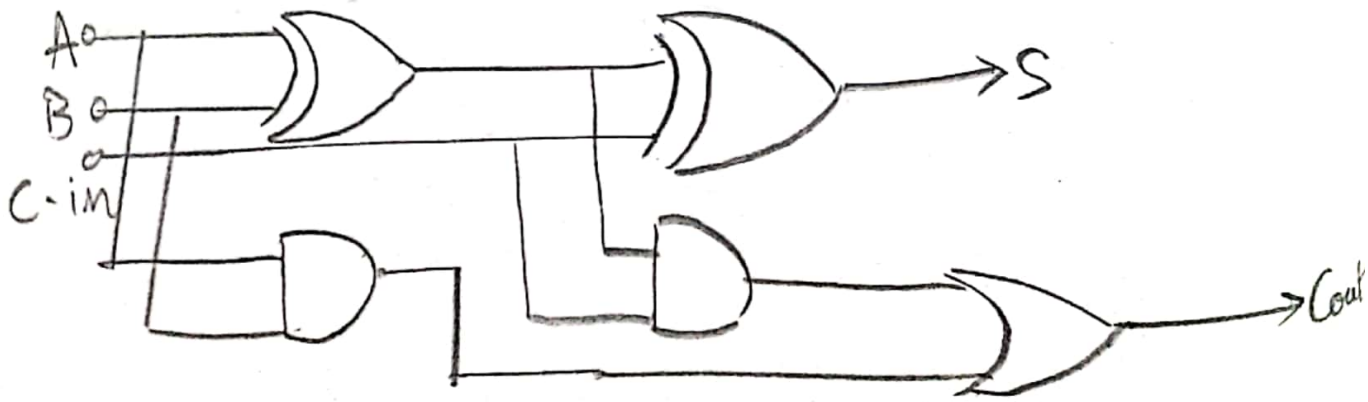
DIAGRAM:

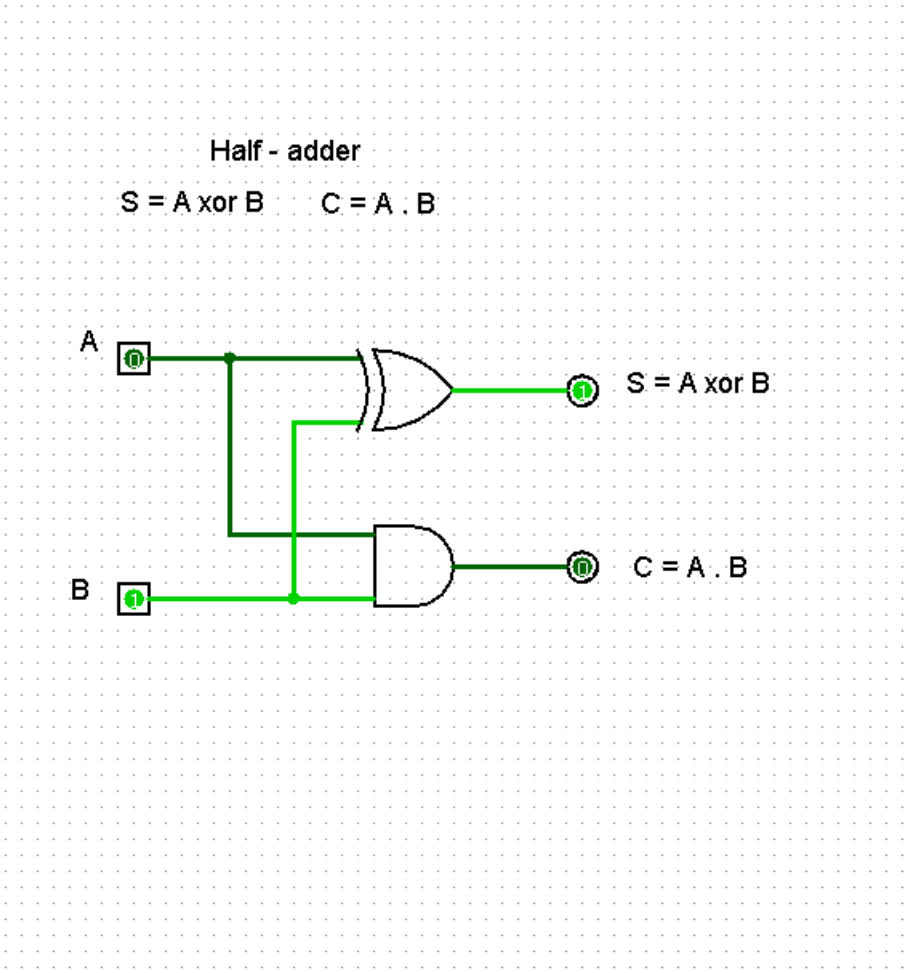



→ FULL ADDER:

| Cin | A | B | 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 |

DIAGRAM:

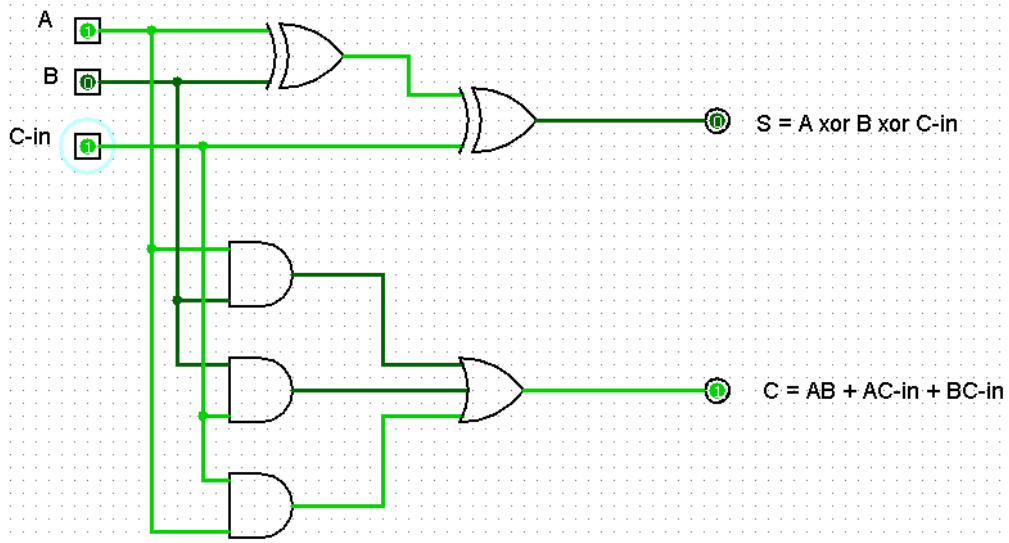



Half - adder
S = A xor B
C = A . B
A
B
S = A xor B
C = A . B

Assignment No. 4

Full - adder

S = A xor B xor C-in  C = AB + AC-in + BC-in



23/09/2020

TUSHAR
1902112
C23



# Assignment No. 5

AIM: Using the logisim simulator, implement conversion as 4 bit binary to gray code and vice versa

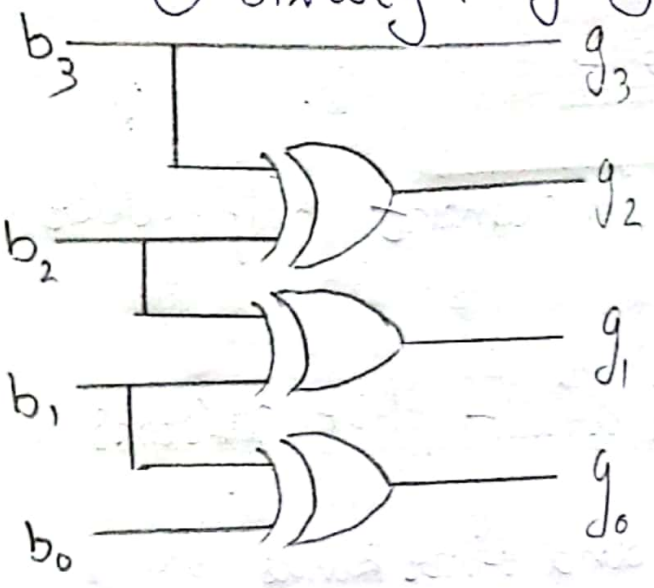
THEORY: Gray code is a reflected binary code is an ordering of binary numerical system such 2 successive value differ in only bit.

(i) Binary to gray code:
Suppose binary code 101101 is given, we retain the MSB and then we do XOR operation b/w 2 bits from the MSB.
So, the gray code converted is 111011.

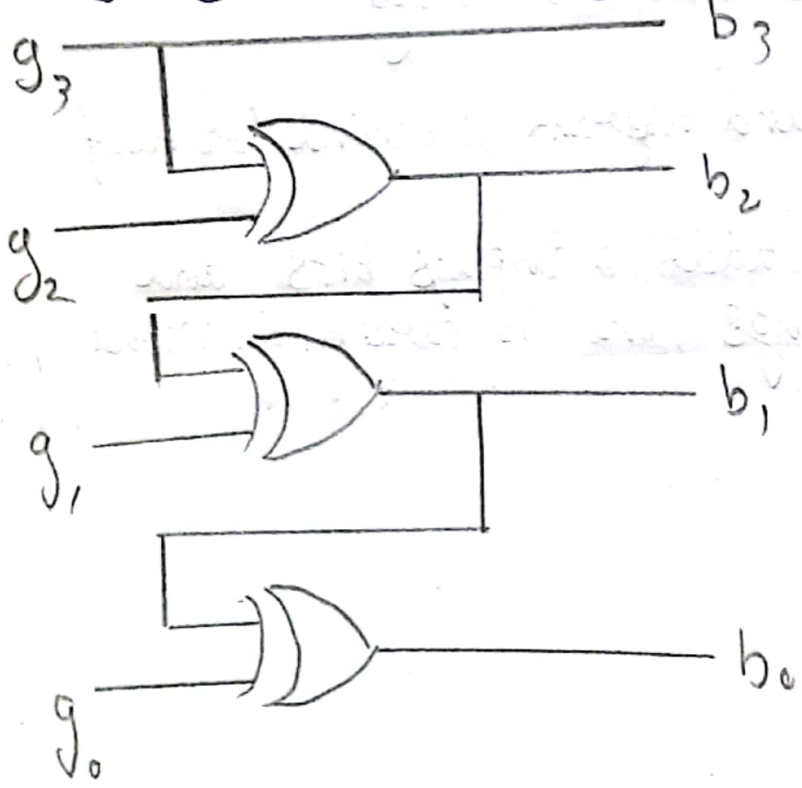
(iii) Gray to binary code:

Suppose gray code 110111 is given, we return the MSB and then we do XOR operation b/w the output of 1st bit and the input of the second bit. In this case, 1st bit and the input is MSB, output of 2nd bit is 1 so XOR is $1 \oplus 1 = 0$, and so on and gray code converted to binary 100101.

## ① Binary to gray.

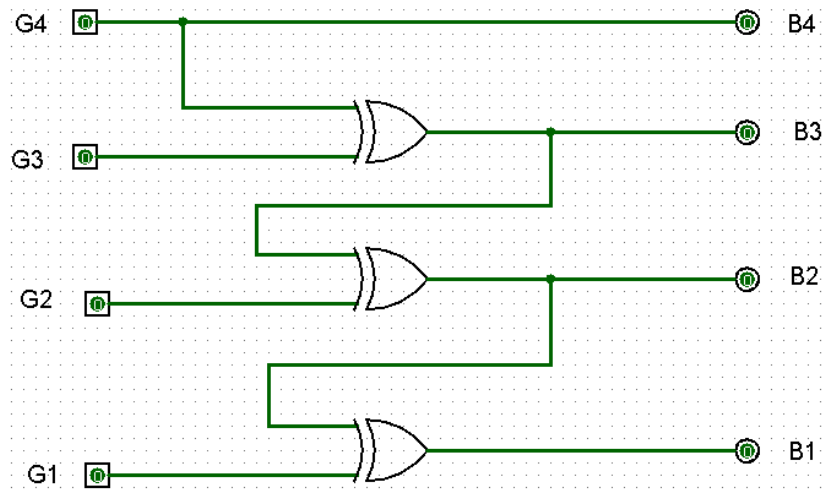


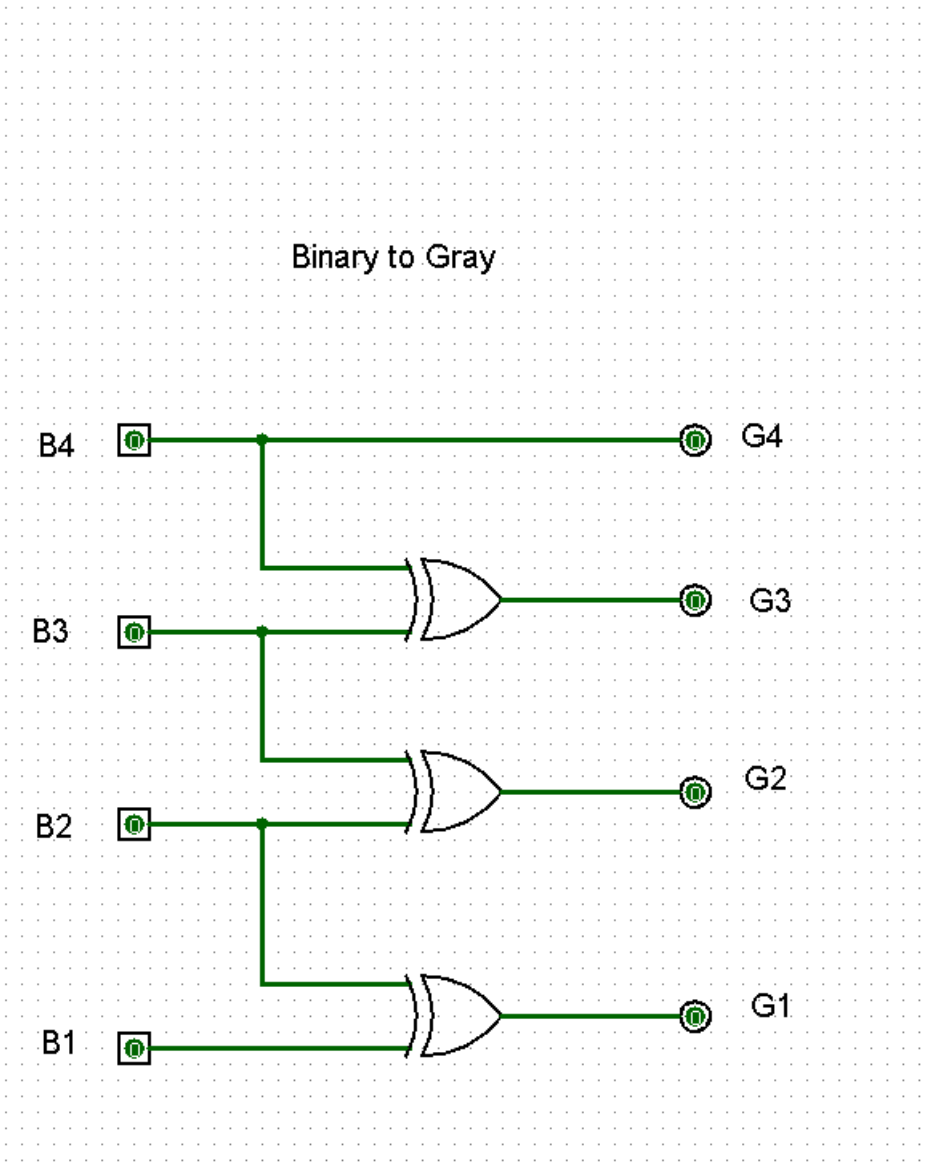

## ② gray to binary.



ASSIGNMENT NO. 5

Gray to Binary




Binary to Gray
B4
B3
B2
B1
G4
G3
G2
G1

30/09/2020

TUSHAR
1902112

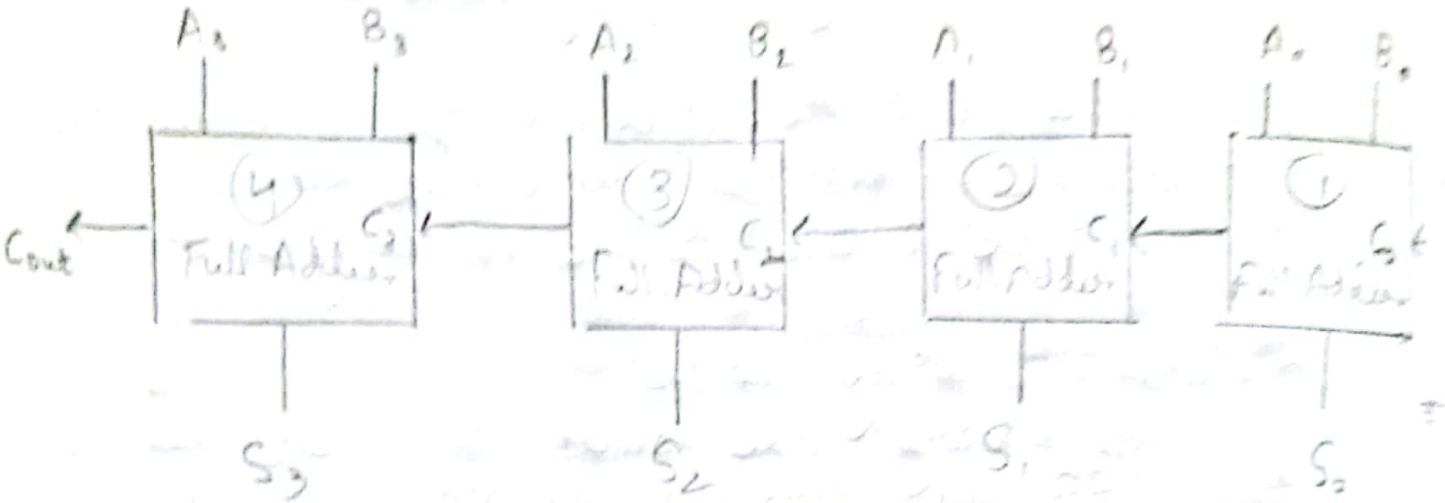
57

# Assignment No. 6

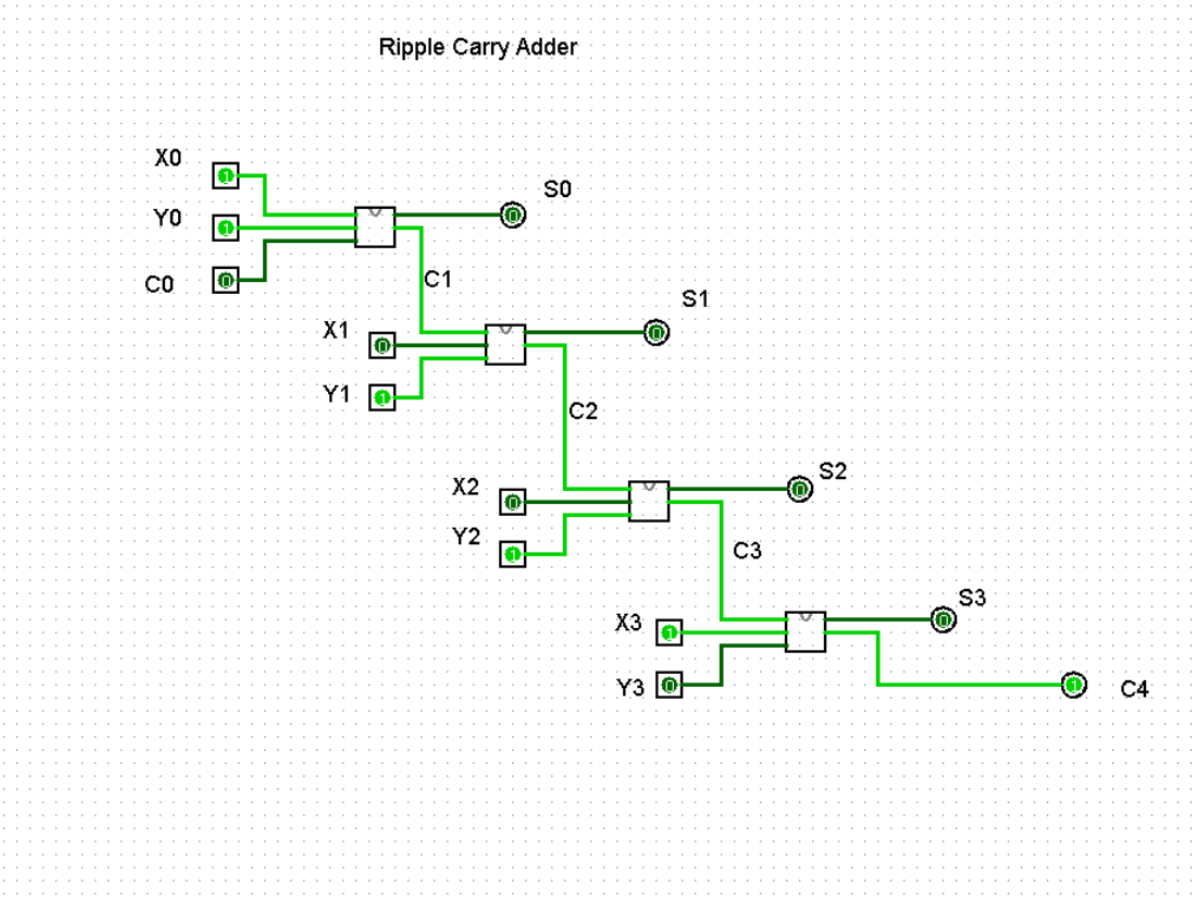
AIM: Implement a 4 bit ripple carry adder. Use a 1-bit full adder to implement ripple carry adder (4 1-bit full adders)

THEORY: An adder is a digital circuit that performs addition. In many computers and other kinds of processors, adders are used in ALU.

Each full adder inputs a line which is the cout of the previous adder. This kind of adder is called ripple carry adder. (RCA), since each carry bit ripples to the next full adder.




4-bit ripple carry adder.


Ripple Carry Adder
X0
Y0
C0
S0
C1
X1
Y1
S1
C2
X2
Y2
S2
C3
X3
Y3
S3
C4

# Assignment No. 7

**AIM:** Write a program to implement booth's algorithm.

**THEORY:** Algorithm:

- n → no. of iterations

Initialize A = 0, append 0-bit to $Q_{-1}$ posn.

do n times...

A = Accumulator.
Q = Multiplier Register
M = Multiplicand.

(1) Examine bits $Q_0 Q_{-1}$

if $Q_0 Q_{-1} = 01$, then
A = A+M
RS(A·Q)

if 10, then
A = A-M
RS(A·Q)

else 00 / 11, then
RS(A·Q)

Finally we discard the LSB from Q.

Example:

$+5 = 0101$
$+2 = 010$

$M = 01010$
$A = 0000$
$Q = 0100$

$M' = 1010 + 1 = 1011$

no. of iterations = 3

| | M | A | Q |
|---|---|---|---|
| | 0101 | 0000 | 0100 |
| (1) RS(A.Q) | | 0000 | 0010 |
| (2) (i) A = A - M | | + 1011 / 1011 | 0010 |
| (ii) RS(A.Q) | | 1101 | 1001 |
| (3) A = A + M | | + 0101 / (1) 0010 | 1001 |
| RS(A.Q) | | 0001 | 0100 |

we discard the last bit

Ans = $0001\underline{010}$ = 10 in decimal.

$5 \times 2 = 10$

Hence, verified.

Name: **Tushar Nankani** Roll No: **1902112** Batch: **C23**

Experiment 7: Implement Booth's Algorithm

```
#include <bits/stdc++.h>

using namespace std;

string M, A, Q, MD;

string binary(int n)
{
    int a[50], i, flag = 0;
    string s = "";

    // for positive;
    if(n > 0)
    {
        for(i=0; n>0; i++)
        {
            a[i] = n % 2;
            n = n / 2;
        }
        // Adding 0 for the sign bit;
        s += "0";
        for(i = i - 1; i >= 0; i--)
        {
            s += ('0' + a[i]);
            // printf("%d", a[i]);
        }
    }
    // For negative;
    else
    {
        n *= -1;
        for(i = 0; n>0; i++)
        {
            a[i] = n % 2;
            n = n / 2;
        }
        //
```

```
        a[i] = 0;
        int t = i;
        for(i = t; i >= 0; i--)
            a[i] = !a[i];
            // 1 --> 0

        int carry = 0;

        if(a[0] == 1) {
            a[0] = 0;
            carry = 1;
        }
        else {
            a[0] = 1;
            carry = 0;
        }
        for(i = 1; i <= t; i++) {
            if(a[i] && carry)
                a[i] = 0, carry = 1;
            else if(a[i] == 0 && carry)
                a[i] = 1, carry = 0;
        }

        if(carry)
        {
            // printf("1");
            s += "1";
        }
        for(i = t; i >= 0; i--)
            s += ('0' + a[i]);
    }
    cout << "Binary: " << s << endl;
    return s;
}

int checkLastTwo(string s)
{
    int len = s.length(), ok = -1;
    if(s[len - 2] == s[len - 1])    // 00 || 11
        ok = 0;
    else if(s[len - 2] == '0' && s[len - 1] == '1')     // addition;
        ok = 1, cout << "A = A + M\t";
```

```
    else if(s[len - 2] == '1' && s[len - 1] == '0')     // subtracti
on;
        ok = 2, cout << "A = A - M\t";

    return ok;
}

void rightShift()
{
    // A, Q;
    int ql = Q.length(), al = A.length();
    //  0000 0101
    for(int i = ql - 1; i >= 0 ; --i)
    {
        if(i == 0)
            Q[i] = A[al - 1];
        else
            Q[i] = Q[i - 1];
    }
    for(int i = al - 1; i >= 0 ; --i)
    {
        if(i == 0)
            continue;
        else
            A[i] = A[i - 1];
    }
}

void add(string y)
{
    int l = A.length(), carry = 0;
    for(int i = l - 1; i >= 0; --i)
    {
        if(A[i] == '1' && y[i] == '1' && carry)
            A[i] = '1', carry = 1;
        else if(((A[i] == '1' && y[i] == '0') || (A[i] == '0' && y[i
] == '1')) && carry)
            A[i] = '0', carry = 1;
        else if(A[i] == '0' && y[i] == '0' && carry)
            A[i] = '1', carry = 0;

        else if(A[i] == '1' && y[i] == '1')
```

```
            A[i] = '0', carry = 1;
        else if((A[i] == '1' && y[i] == '0') || (A[i] == '0' && y[i]
 == '1'))
            A[i] = '1', carry = 0;
        else if(A[i] == '0' && y[i] == '0')
            A[i] = '0', carry = 0;
    }
}

// A = A + M or A = A - M;
void operate(int ok)
{
    //cout << "\nA: " << A << endl;
    if(ok == 1)
        add(M);
    else if(ok == 2)
        add(MD);
    //cout << "\nA: " << A << endl;
}

string twoComplement(string x)
{
    int t = x.length();
    string a = x;

    for(int i = t; i >= 0; i--)
            a[i] = ('0' + (1 - (a[i] - '0')));
    int carry = 0;
    if(a[t - 1] == '1') {
        a[t - 1] = '0';
        carry = 1;
    }
    else {
        a[t - 1] = '1';
        carry = 0;
    }
    for(int i = t - 2; i >= 0; --i) {
        if(a[i] == '1' && carry)
            a[i] = '0', carry = 1;
        else if(a[i] == '0' && carry)
            a[i] = '1', carry = 0;
    }
```

```
    return a;
}

void go()
{
    cout << "M: " << M;
    cout << "   A: " << A;
    cout << "   Q: " << Q;
    cout << endl;
}

int main()
{
    cout << "\nWelcome to Booth's Multiplication Algorithm!" << endl
;
    int a, b;
    cout << "\nEnter two numbers (a and b) for multiplication: " <<
endl;
    cout << "Enter a:  ";
    cin >> a;
    cout << "Enter b:  ";
    cin >> b;
    cout << endl;
    M = binary(a), Q = binary(b);
    if(M.length() < Q.length())
        swap(M, Q);

    int cnt = Q.length();
    Q += '0';
    int ml = M.length(), ql = Q.length();
    A = "", MD = twoComplement(M);
    cout << "MD: " << MD << endl;

    for(int i = 0; i < ml; ++i)
        A += "0";

    go();
    for(int i = 0; i < cnt; ++i)
    {
        cout << "\nIteration " << i + 1 << ":" << endl;
        int opt = checkLastTwo(Q);
        if(opt) {
```

```
            operate(opt);
            go();
        }
        cout << "RS(A.Q)\t\t";
        rightShift();
        go();
    }

    cout << "\nFinal Answer:  " << A << Q.substr(0, ql - 1) << endl;
    cout << "In decimal: " << a * b << endl;
    return 0;
}
```

```
C:\Users\Tushar Nankani\Desktop\SEM 3\DLCA (Digital Logic & Computer Architecture)\A Practicals>booths

Welcome to Booth's Multiplication Algorithm!

Enter two numbers (a and b) for multiplication:
Enter a:  8
Enter b:  12

Binary: 01000
Binary: 01100
MD: 11000
M: 01000   A: 00000   Q: 011000

Iteration 1:
RS(A.Q)         M: 01000   A: 00000   Q: 001100

Iteration 2:
RS(A.Q)         M: 01000   A: 00000   Q: 000110

Iteration 3:
A = A - M       M: 01000   A: 11000   Q: 000110
RS(A.Q)         M: 01000   A: 11100   Q: 000011

Iteration 4:
RS(A.Q)         M: 01000   A: 11110   Q: 000001

Iteration 5:
A = A + M       M: 01000   A: 00110   Q: 000001
RS(A.Q)         M: 01000   A: 00011   Q: 000000

Final Answer:  0001100000
In decimal: 96
```

21/10/2020

TUSHAR
1902112
C23

61

# Assignment No. 8

**AIM:** To implement carry look ahead generator.

**THEORY:** A carry look ahead adder (CLA) is a type of electronics adder used in digital logic.

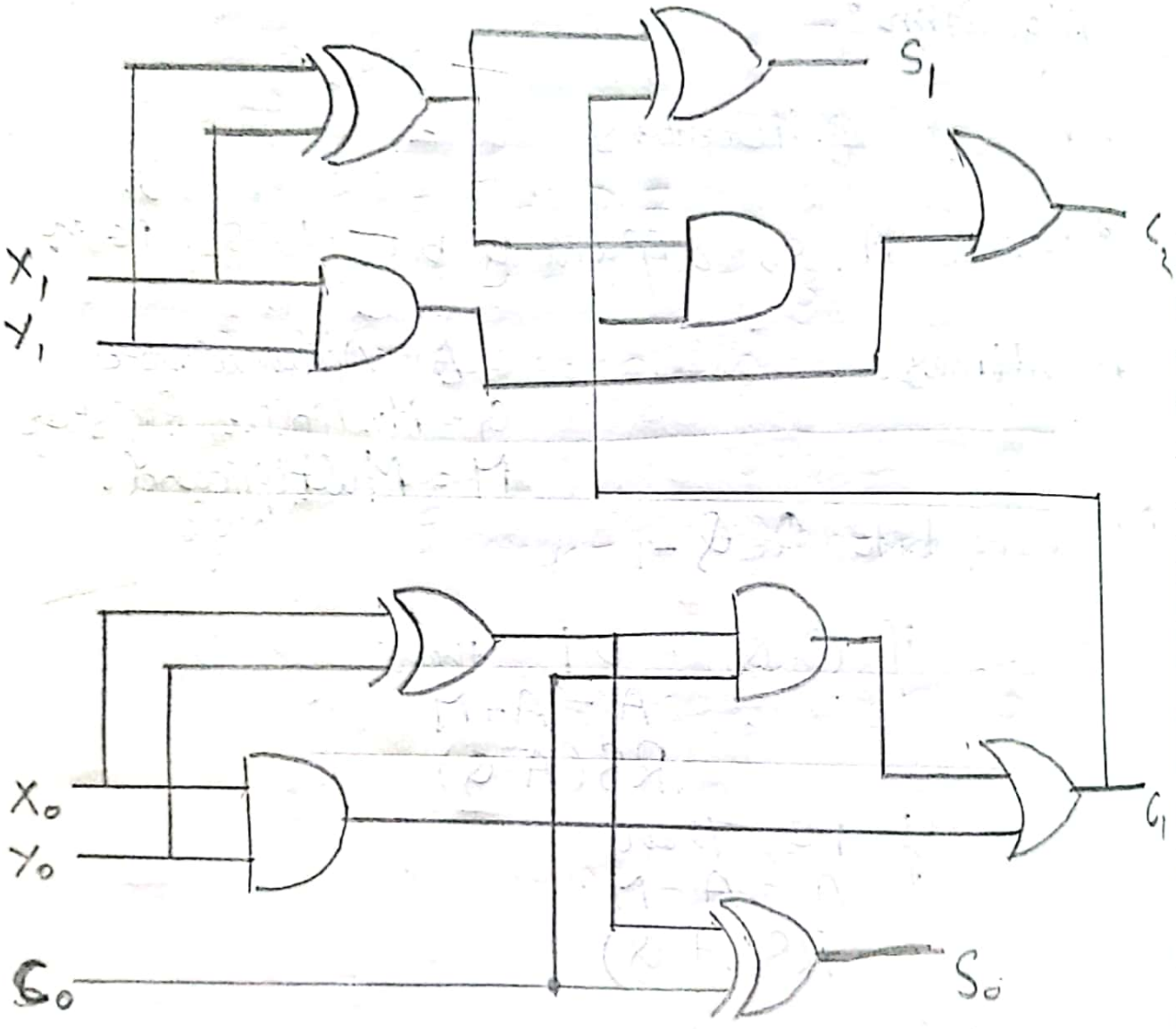
- CLA improves speed by reducing the amount of time required to determine carry bits.
- CLA calculates one or more carry bits before the sum which reduces wait time to calculate values result of larger value bits of the adder.

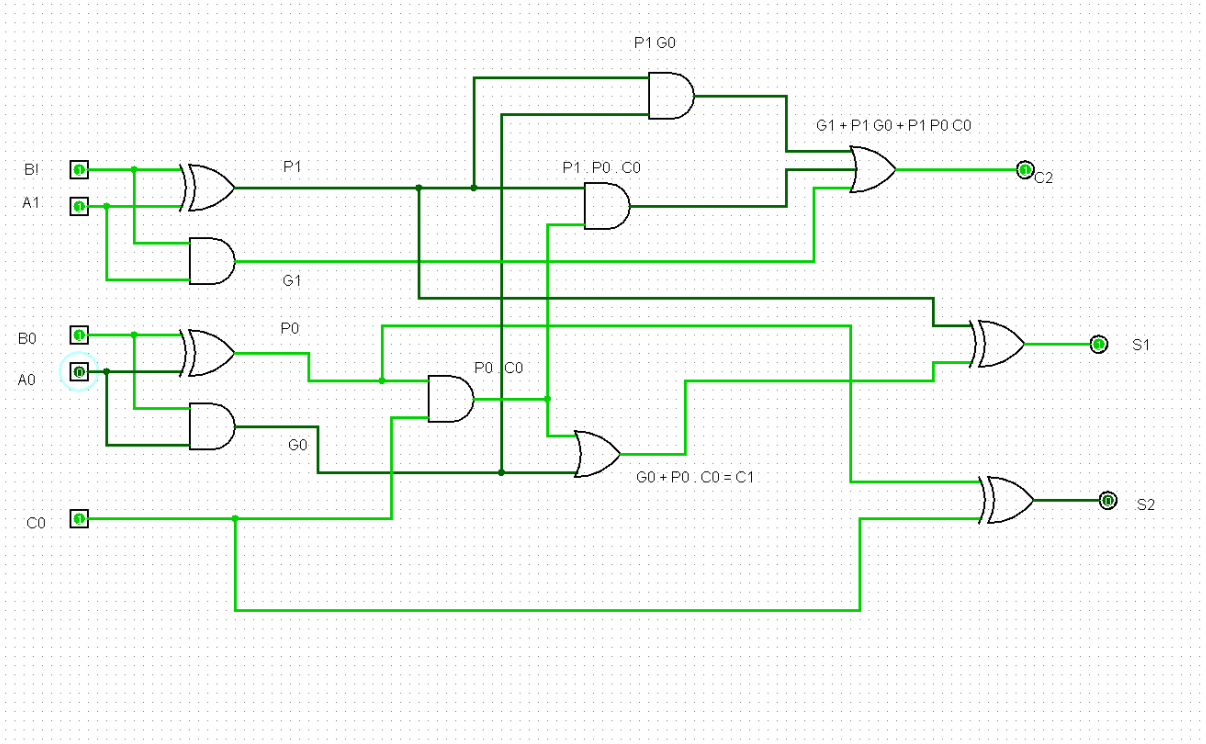
$$C_{i+1} = G_i + P_i C_i$$

$$P_i = A_i \oplus B_i$$
$$G_i = A_i + B_i$$

$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1$$
$$= G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

# Carry Look Ahead Generator




P1 G0
G1 + P1 G0 + P1 P0 C0
BI
A1
P1
P1 . P0 . C0
C2
G1
B0
A0
P0
P0 . C0
S1
G0
G0 + P0 . C0 = C1
C0
S2

TUSHAR
1902312



# Assignment No. 9.

**AIM:** Write a program to convert into IEEE 754 conversion.

**THEORY:** IEEE Standard 754 for floating point computation which was established in 1985 by Institute of Electrical and Electronics Engineers

IEEE 754 has 3 basic components:

(1) the sign of the mantissa
(2) the biased exponent
(3) the normalized mantissa.

32 bits.

| Sign | Exponent | Mantissa |
|---|---|---|
| 1 bit | 8 bit | 23 bit |

eg: Convert IEEE 754 representation from decimal.

① $(21)_{10} \Rightarrow (10101.0000)_2$

$\Rightarrow 1.0101 0000 \times 2^4$

$\Rightarrow E = 4$

Hence $\Rightarrow$ sign is positive $\Rightarrow 0$

$\Rightarrow E' = E + 127$

$= 4 + 127 = 131$

$\Rightarrow M \Rightarrow 0101000 \cdots 0$

| 0 | 10000011 | 0101000...0 |
|---|---|---|
| Sign | Exponent | Mantissa. |

To decimal.

② IEEE:

| 1 | 10000001 | 0100...0 |
|---|---|---|

$\because S = 1 \Rightarrow$ -ve

$\because E' = (10000001)_2 = (129)_{10}$

$E = 129 - 127$

$E = 2$

$\therefore$ Ans $\Rightarrow (-1) \times (1.01) \times 2^{(2)} = -101.00$

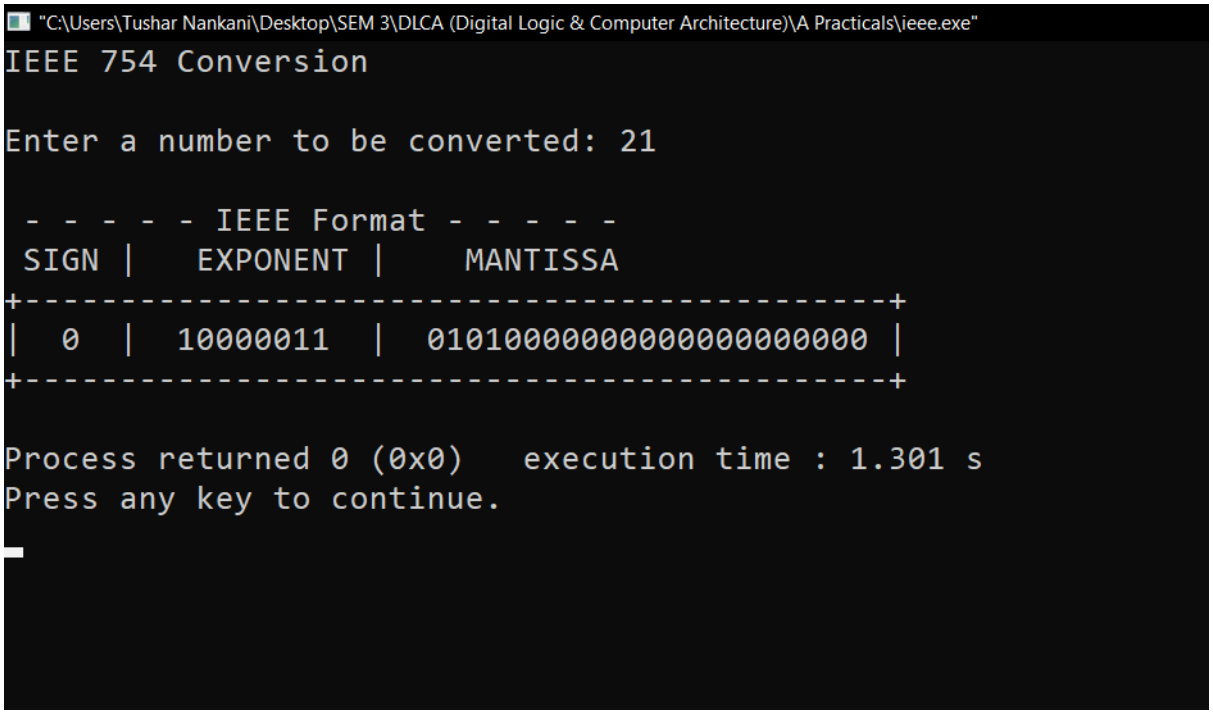
$= \boxed{-5}$ //

Name: **Tushar Nankani** Roll No: **1902112** Batch: **C23**

Experiment 7: IEEE 754 Conversion from decimal

```
#include <bits/stdc++.h>

using namespace std;

string binary(int n)
{
    int a[50], i, flag = 0;
    string s = "";

    for(i=0; n>0; i++)
    {
        a[i] = n % 2;
        n = n / 2;
    }
    for(i = i - 1; i >= 0; i--)
    {
        s += ('0' + a[i]);
    }
    return s;
}


int main()
{
    cout << "IEEE 754 Conversion\n" << endl;
    int n;
    cout << "Enter a number to be converted: ";
    cin >> n;

    const int extra = 127, exp = 8, man = 23;
    string exponent, mantissa;
    char sign = ((n > 0) ? '0' : '1');

    string binaryNum = binary(abs(n));

    exponent = binary(binaryNum.length() - 1 + 127);
    mantissa = binaryNum.substr(1);
```

```
    for(int i = 0; i < man - binaryNum.length() + 1; ++i)
        mantissa += '0';

    cout << "\n - - - - - IEEE Format - - - - - \n";
    cout << " SIGN |   EXPONENT |    MANTISSA  " << endl;
    cout << "+----------------------------------------
+" << endl;
    cout << "|  " << sign << "  |  " << exponent << "  |  " << manti
ssa << " |" << endl;
    cout << "+----------------------------------------
+" << endl;


    return 0;
}
```


"C:\Users\Tushar Nankani\Desktop\SEM 3\DLCA (Digital Logic & Computer Architecture)\A Practicals\ieee.exe"
IEEE 754 Conversion
Enter a number to be converted: 21
- - - - - IEEE Format - - - - -
SIGN | EXPONENT | MANTISSA
+-----------------------------------------------+
| 0 | 10000011 | 01010000000000000000000 |
+-----------------------------------------------+
Process returned 0 (0x0) execution time : 1.301 s
Press any key to continue.

25/11/2020

TUSHAR NANKANI
1902112
C23



# Assignment No. 10

**AIM:** To implement a 8:1 Mux and a 3:8 decoder.

**THEORY:-** A multiplexer is a device that selects b/w several analog or digital inputs signals.

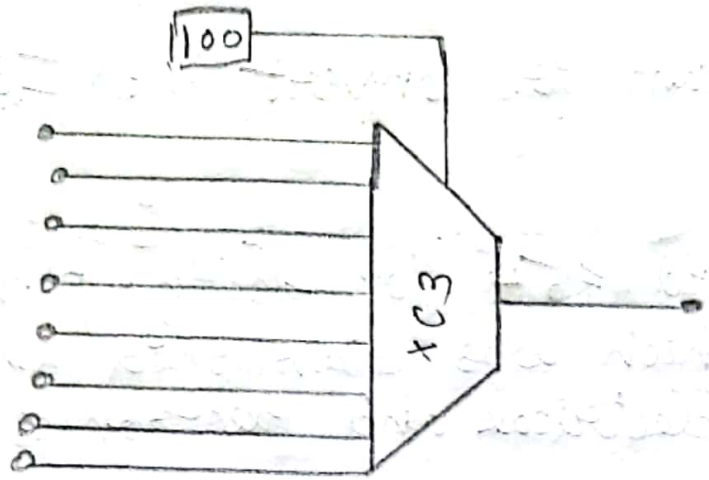
- It is a digital switch that has multiple inputs and single output is called a decoder.

- Types of MUX: 2:1, 4:1, 8:1, 16:1

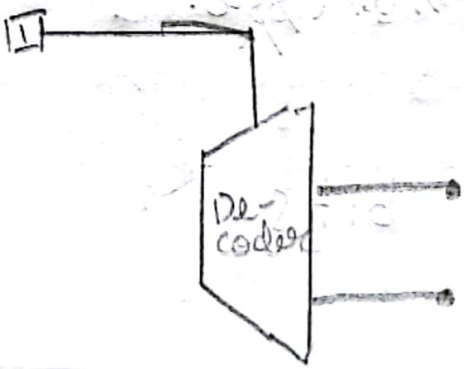
- Applications of Multiplexers and Decoders.

1. Communication system, computer memory, telephone networks are some examples of multiplexers.

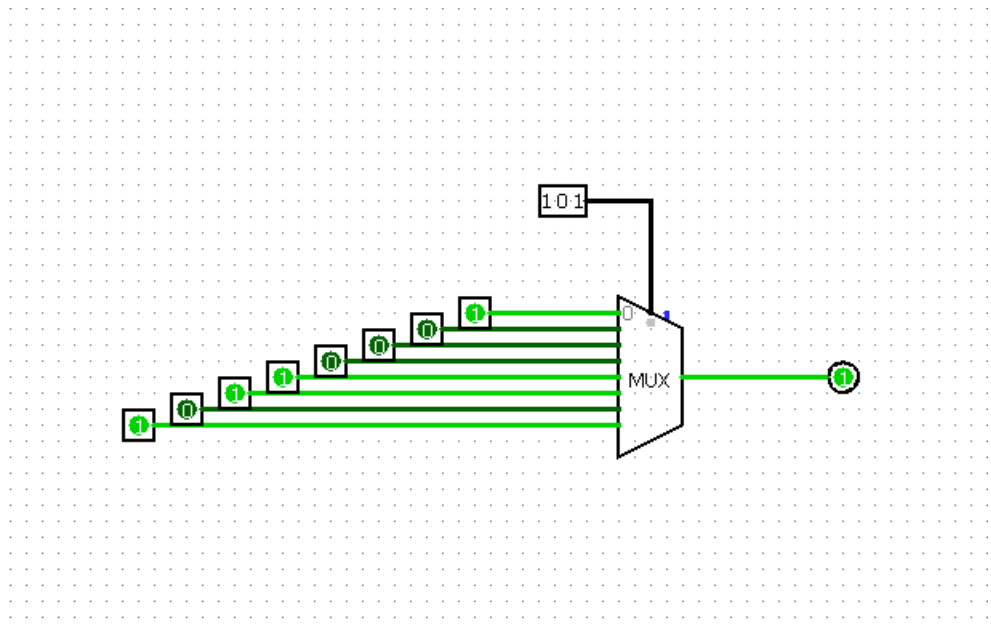
2. Used in logical circuits, data transfer, electronic circuits to convert instruction into CPU control signals are application of decoders.

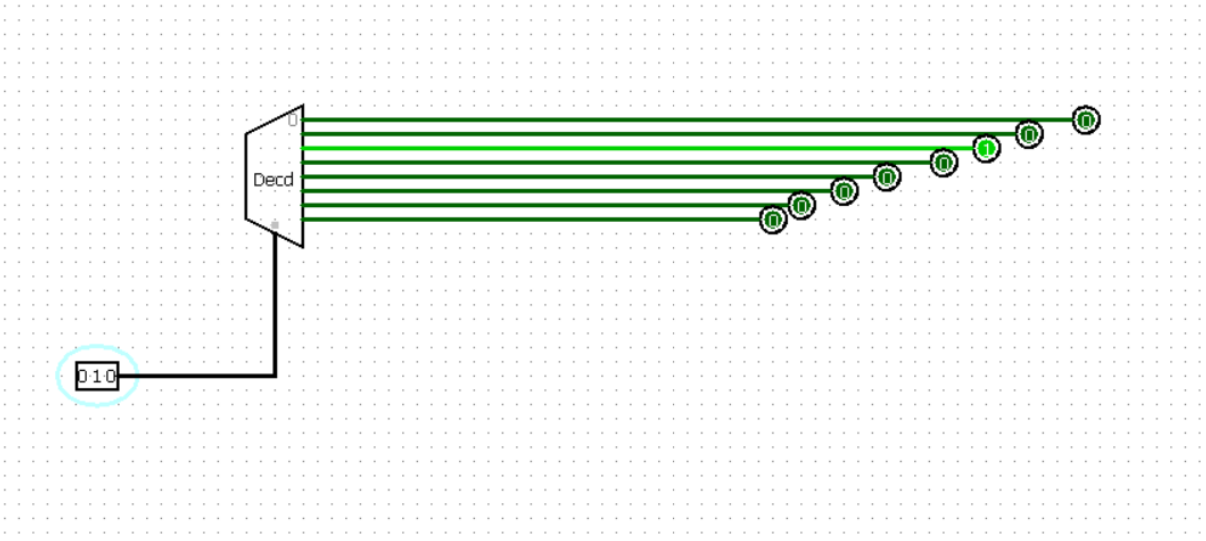



3:1 Multiplexer




1:2 decoder.


101
MUX


Decd
010