Thadomal Shahani Engineering College

Bandra (W.), Mumbai – 400050

University of Mumbai

(A Y 2021-2022)

This is to certify that Mr. / ~~Ms~~. **Ratlamwala Idris Ismail** of Department of Computer Engineering, in 3rd Semester with Batch **C31** and Roll No **2003145** has completed the course of necessary experiments and assignments in the subject of **'Digital Logic & Computer Organization and Architecture Lab'** successfully under my supervision in the **'Thadomal Shahani Engineering College'** during academic year 2021 – 22

Teacher In-Charge:

Index

Idris Ratlamwala ; C31 - 2003145

PAGE No. Idris
DATE

Experiment 1.

Aim: To implement
i) Decimal to Binary Conversion & viceversa
ii) 2's complement.

Theory:

1) Decimal to Binary:

i) Divide the decimal by 2.
ii) Note down quotient & remainder as shown
iii) Repeat step 1 & 2 till the quotient is 0.
iv) The last remainder will be quotient MSB of the corresponding binary number, and first remainder will be LSB.

eg. 28.

2 | 28	
2 | 14	→ 0 → LSB
2 | 7	→ 0
2 | 3	→ 1
2 | 1	→ 1
	0 → 1 → MSB

∴ Binary equivalent = 11100

2) Binary to decimal:

Multiply the binary digits (bits) with the corresponding powers of 2, according to their positional weight.

eg. $(11100)_2$ =

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

$= (28)_{10}$

3) 2's complement method of representing signed numbers.

i) First take 1's complement of binary number. ie flip all 1's with 0's & all 0's with 1's.

ii) Then take 2's complement of binary number. ie add 1 to get -ve of a number.

Program and Output :

Decimal to binary :

```c
#include <stdio.h>

void main()
{
    int dec,i,length,a[16];
    printf("\nEnter decimal Number : ");
    scanf("%d",&dec);
    for(i=0;dec!=0;i++)
    {
        a[i]=dec%2;
        dec=dec/2;
    }
    length=--i;
    printf("Binary equivalent : ");
    for(i=length;i>=0;i--)
    {
        printf("%d",a[i]);
    }
    printf("\n");
}
```

```
Enter decimal Number : 62
Binary equivalent : 111110
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"DecToBin1.exe"

Enter decimal Number : 123
Binary equivalent : 1111011
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"DecToBin1.exe"

Enter decimal Number : 15
Binary equivalent : 1111
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs>
```

Binary to decimal :

```c
#include <stdio.h>

int main() {
    int bin,dec=0,pow=1;
    printf("\nEnter a binary number: ");
    scanf("%d", &bin);
    while(bin!=0)
    {
        dec=dec+pow*(bin%10);
        pow=pow*2;
        bin=bin/10;
    }
    printf("Decimal equivalent is : %d\n",dec);
}
```

```
Enter a binary number: 11010
Decimal equivalent is : 26
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"binToDec1.exe"

Enter a binary number: 10001
Decimal equivalent is : 17
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"binToDec1.exe"

Enter a binary number: 100
Decimal equivalent is : 4
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs>
```

2’s complement :

```c
#include <stdio.h>

int main()
{
    int i=0,dig,flag=0,bin,b[16];
    printf("\nEnter the binary number:");
    scanf("%d",&bin);
    while(bin!=0)
    {
        dig=bin%10;
        if(dig!=0 && dig!=1)
        {
        printf("The number entered is not a binary number\n");
        printf("Enter the correct number!\n");
        exit(0);
        }

        if(flag!=1) b[i]=dig;
        else
        {
            if(dig==0) b[i]=1;
            else b[i]=0;
        }
        if(dig==1) flag = 1;
        bin=bin/10;
        i++;
    }

    printf("The 2's Complement is :");
    for (i=i-1;i>=0;i--)
    {
        printf("%d",b[i]);
    }
    printf("\n");
    return 0;
}
```

```
Enter the binary number:10110110
The 2's Complement is :01001010
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"2sCompl.exe"

Enter the binary number:11101101
The 2's Complement is :00010011
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"2sCompl.exe"

Enter the binary number:01000100
The 2's Complement is :0111100
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs>
```

2003145 - C21

PAGE No.
DATE Adnin

Experiment 2.

Aim: Verify truth table and realize the logic expression: i) $AB + C$

ii) $AB + BC + C'A$.

Theory: Verification of truth table.

1)

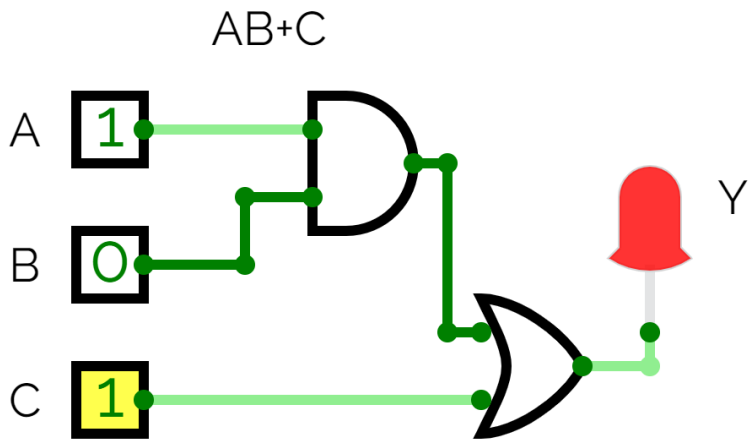
A	B	C	A.B	AB+C	Output of Circuit.
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1

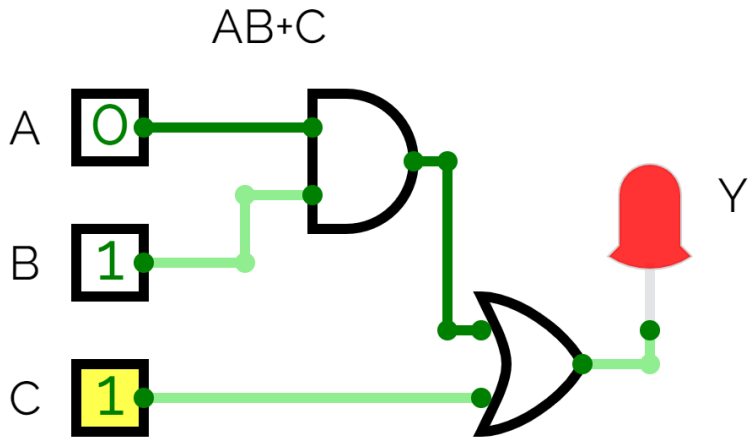
2)

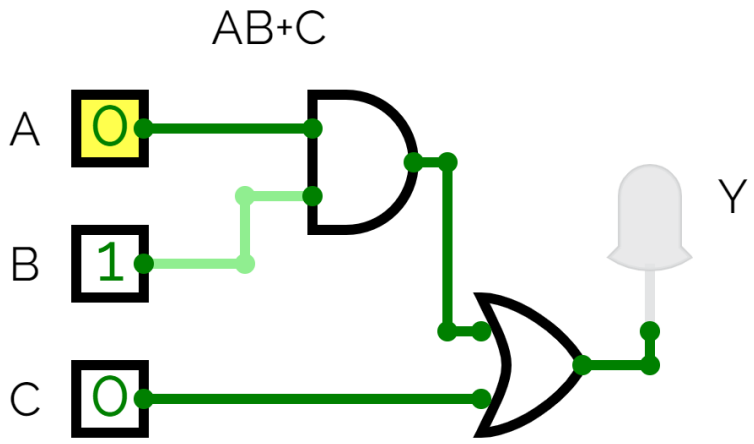
A	B	C	A.B	B.C	C'	C'.A	A.B+BC+C'A
0	0	0	0	0	1	0	1
0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1
1	0	0	0	0	1	1	1
1	0	1	0	0	0	0	0
1	1	0	1	0	1	1	1
1	1	1	1	1	0	0	1

Output :

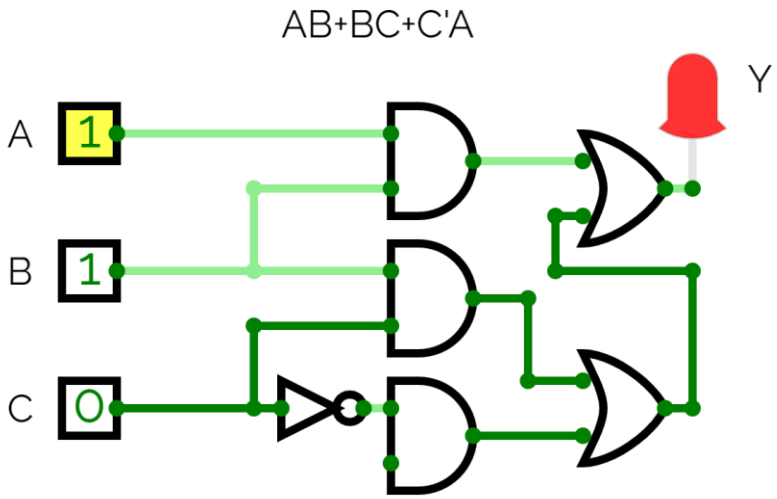
AB+C :

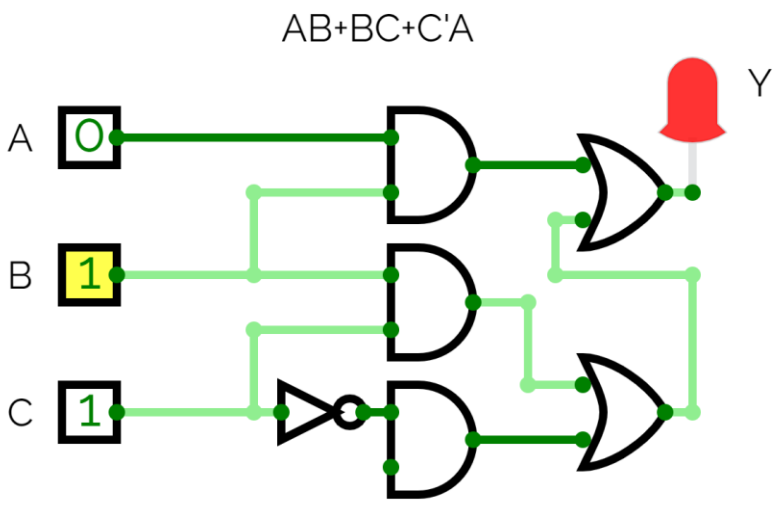


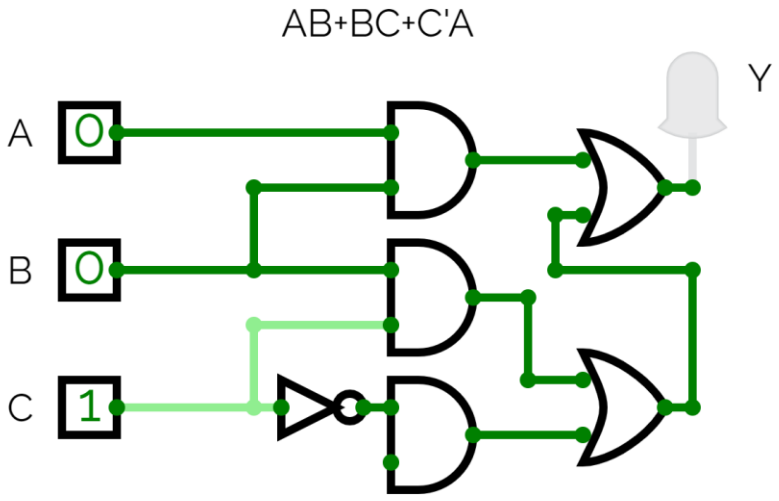




AB+BC+C’A :




AB+BC+C'A
A
0
B
1
C
1
Y


AB+BC+C'A
A
0
B
0
C
1
Y

2003145 - C31

PAGE No.
DATE 16/1/23

Experiment. 3

Aim: Implement NOT, AND, OR and Ex-OR gates using universal gates NAND and NOR.

Theory:

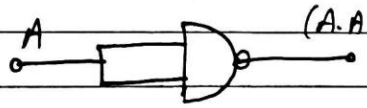
Digital electronics are said to be constructed by using logic gates. These gates are AND, OR, NOT, NAND, NOR, Ex-OR, and Ex-NOR

1) NOT gate

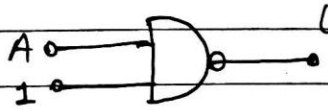
NOT gate is an electronic circuit which produces an inverted version of it's input at the output.

NOT gate can be implemented using NAND gate as:

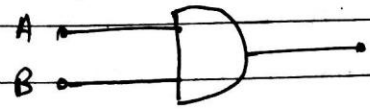
1)



2)

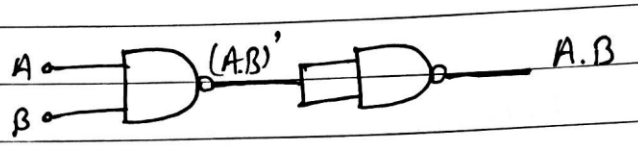


2) AND gate :

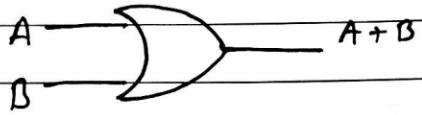


The AND gate produces a high output (1) if and only if all of it's inputs are high.

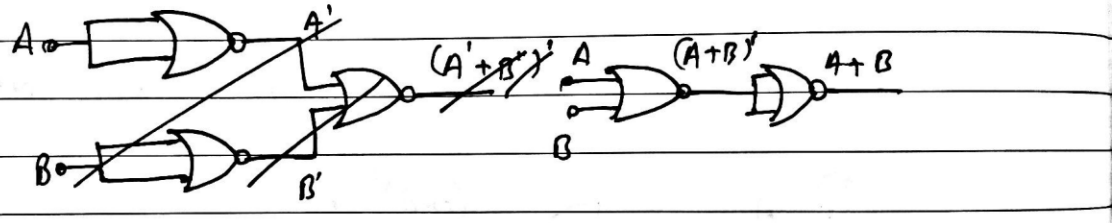
AND gate can be implemented using only NAND gates.



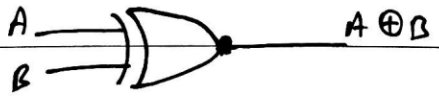
3) OR gate:



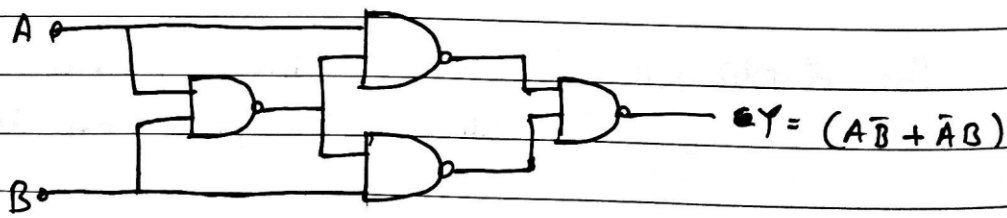
The OR gate gives output low, if and only if all the inputs are low.



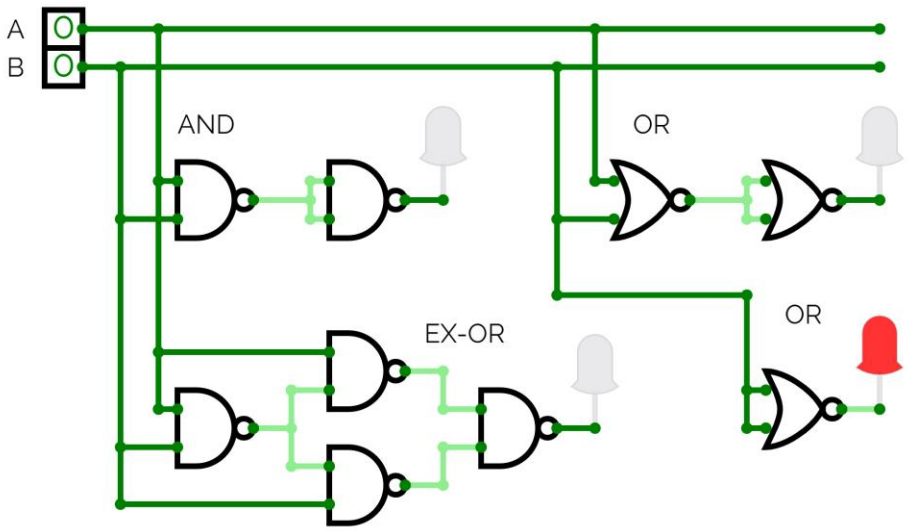
4) Ex-OR gate:

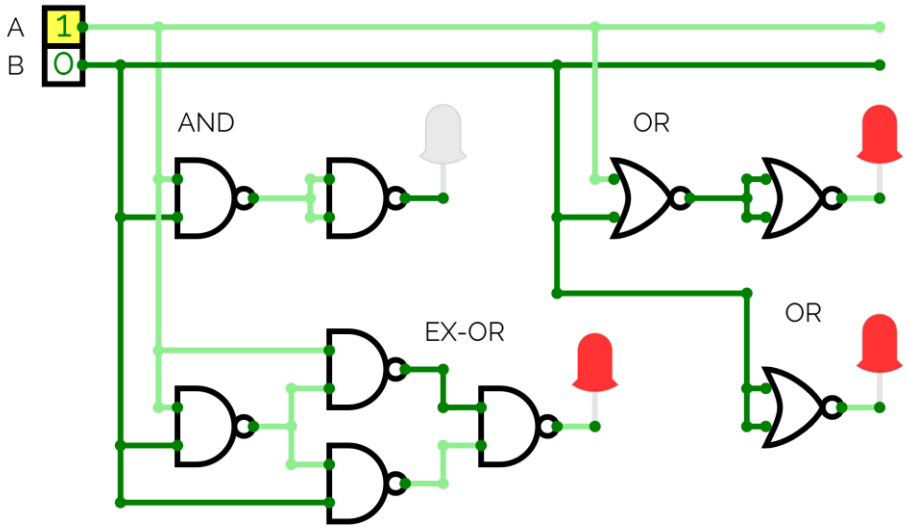


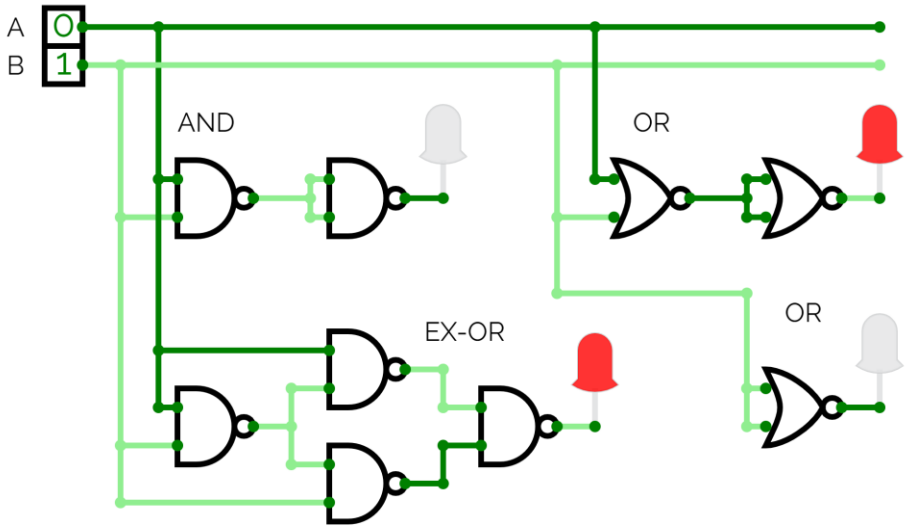
Ex OR gate will ~~none~~ give a high output if either but not both of its inputs are high.

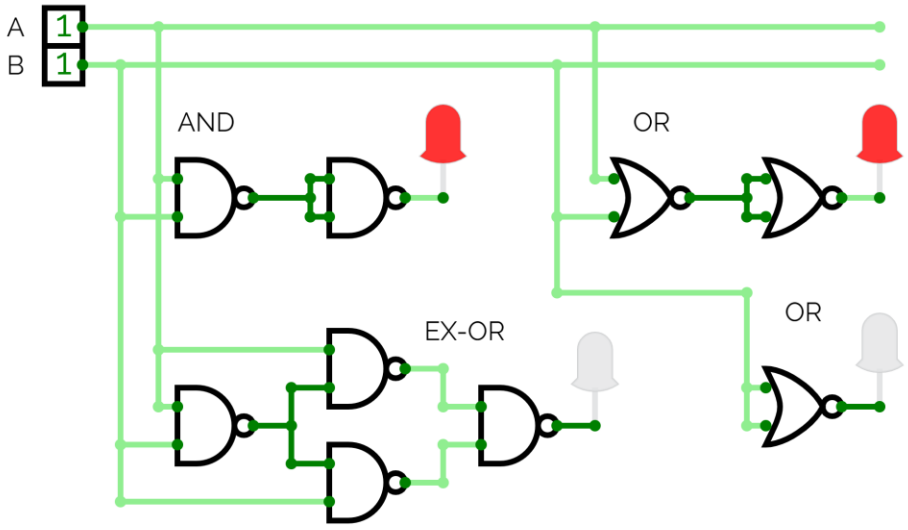


Output :






A
0
B
1
AND
OR
EX-OR
OR


A
1
B
1
AND
OR
EX-OR
OR

200314I-C31

PAGE No.
DATE / /

Experiment. 4.

Aim: Implement half adder and full adder.

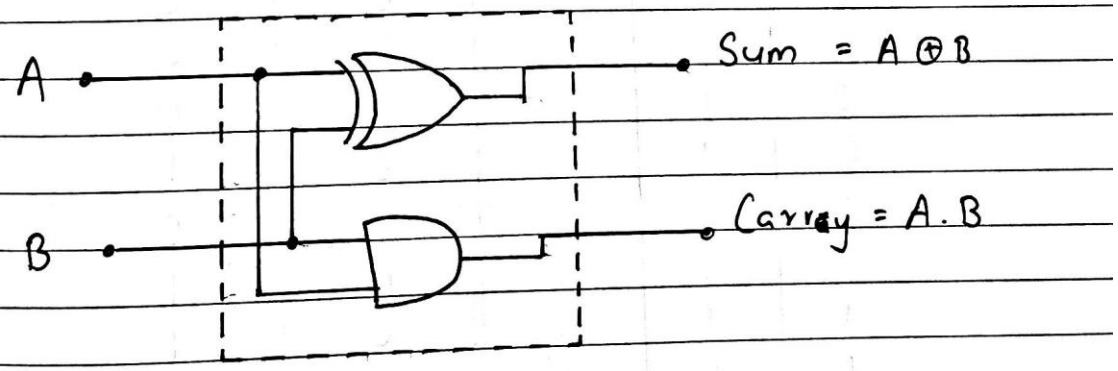
Theory:

Half Adder: The addition of 2 bits is done using a combinational circuit called as half adder. The input variables are augend and addend bits and output variables are sum and carry.

truth table:

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

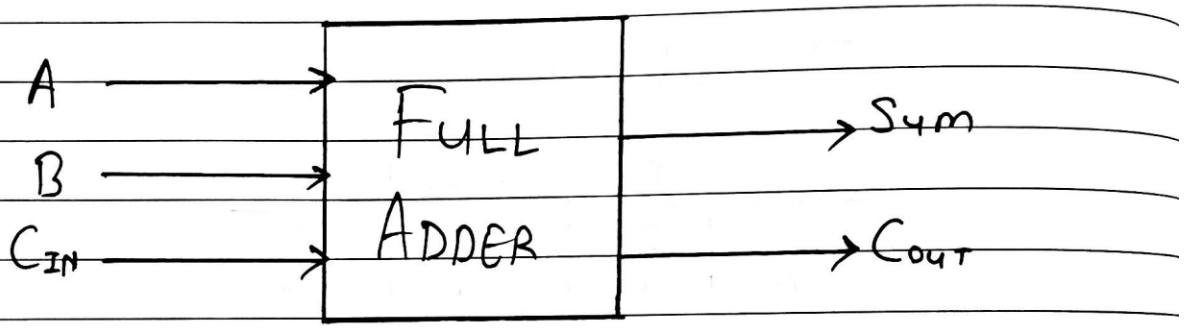
Implementation:



Half adder has only 2 inputs and there is no provision for to add a carry coming from lower order bits.

Full adder :

Full adder is the adder which adds the three inputs and produces 2 outputs. The first 2 inputs are same as half adder ie A & B and 3rd one is the input carry from lower bits (cin). The output is designated as a carry C_{out} and sum 'S'.

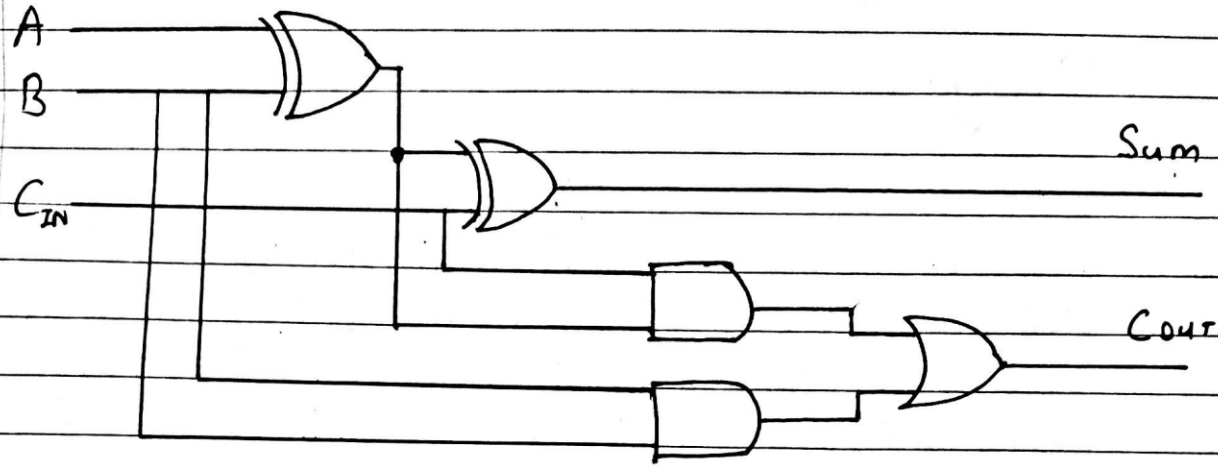


Truth table :

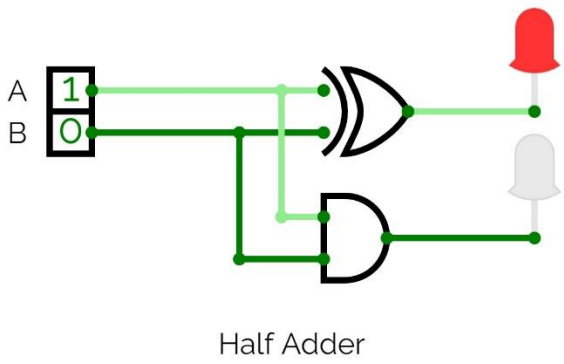
A	B	C_{IN}	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

DATE / /

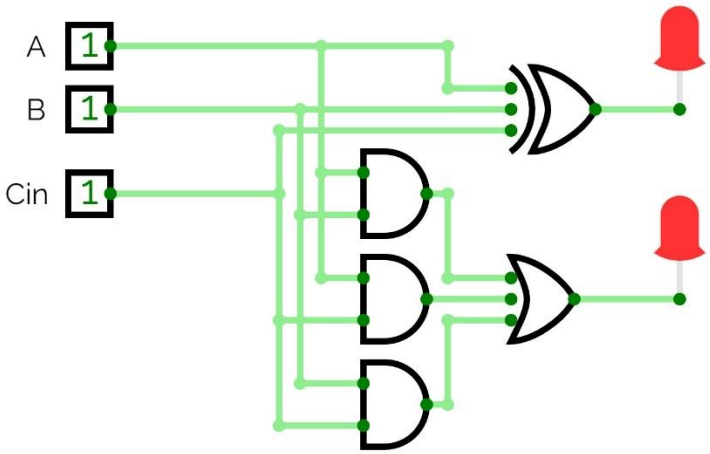
Implementation :



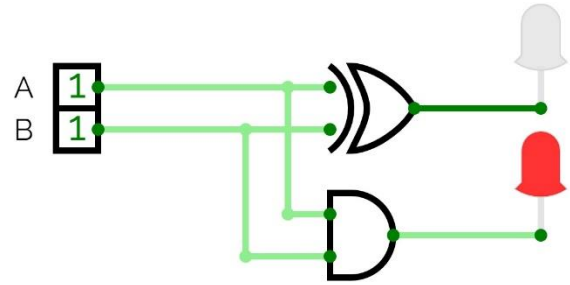
Output :



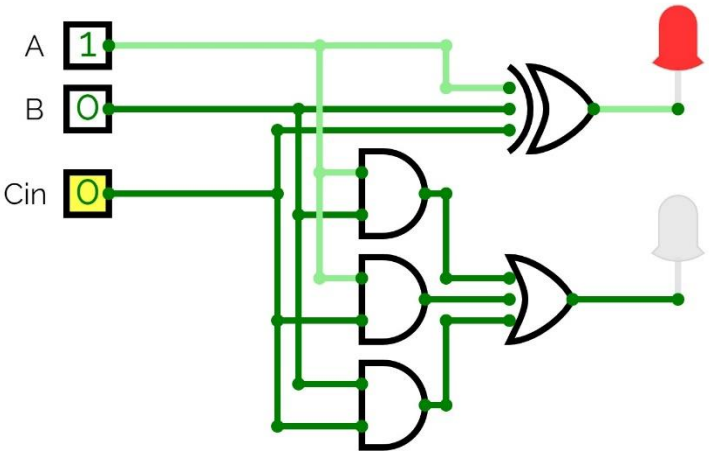
Half Adder



Full Adder



Half Adder



Full Adder

2003145 - C3
PAGE No.
DATE / / Admi

Experiment 5.

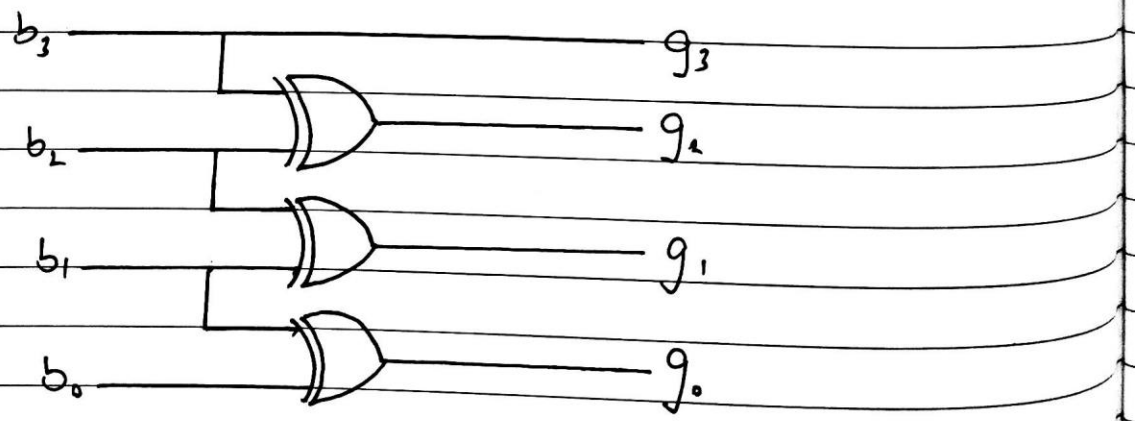
Aim: Implement conversion of 4 bit binary to gray code and viceverca.

Theory:

Gray code system is a binary number system in which each successive pair of number differ in only 1 bit. It is used in applications in which the normal sequence of binary numbers generated by hardware may produce an error or ambiguity during transition from one number to ~~other~~ the next.

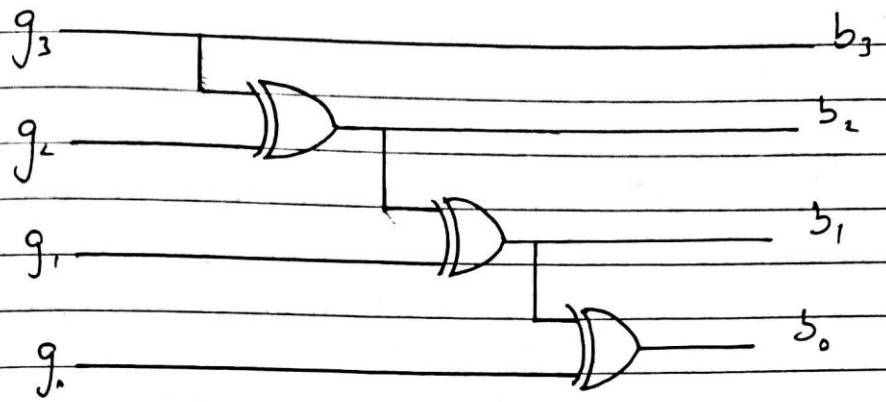
Converting Binary to Gray code:

Let b_0, b_1, b_2 & b_3 be the bits represented binary number, where b_0 is LSB.
& g_0, g_1, g_2, g_3 be the bits representing grey code of binary number where g_0 is LSB.

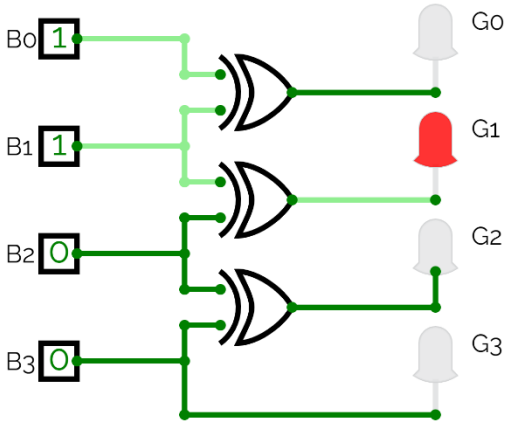


PAGE No.
DATE / /

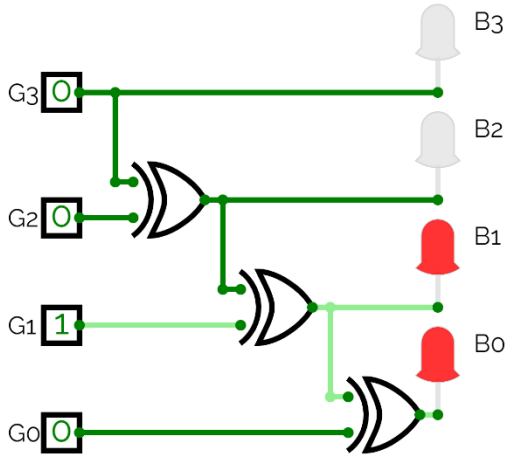
Converting gray code to binary.



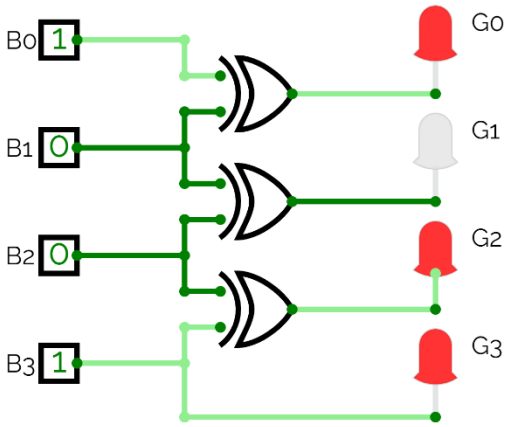
Output :



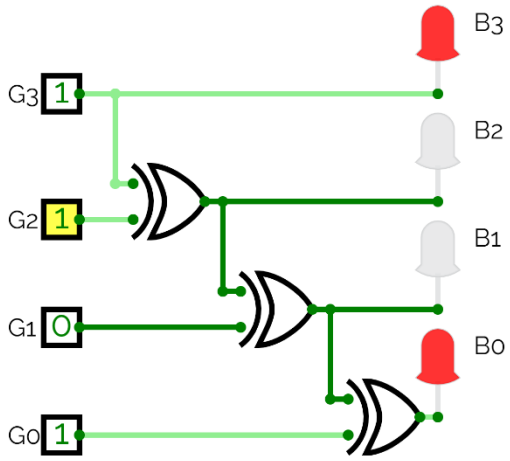
Binary to gray code



Gray code to binary



Binary to gray code



Gray code to binary

2003145 - C_{31}

PAGE No.

DATE / Ashp

Experiment 6

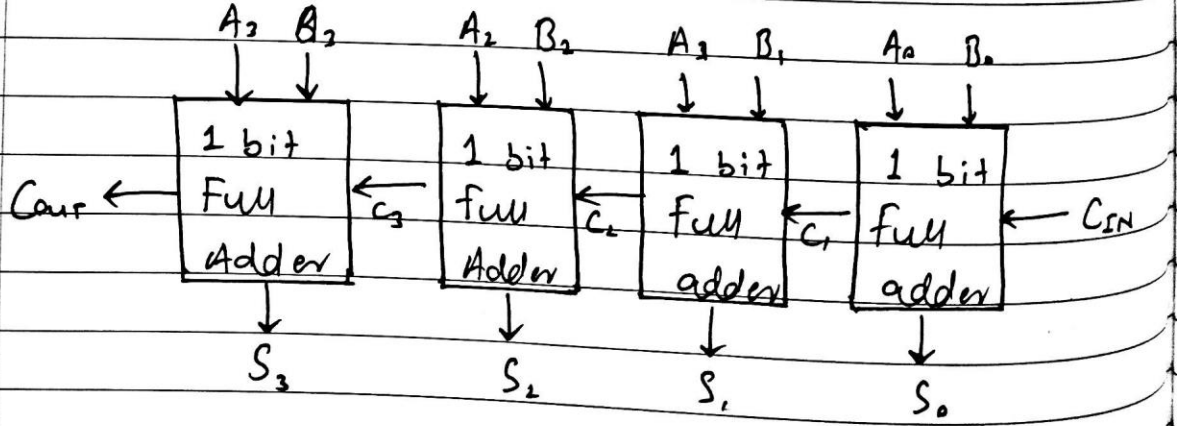
Aim: Implement a 4 bit carry adder.

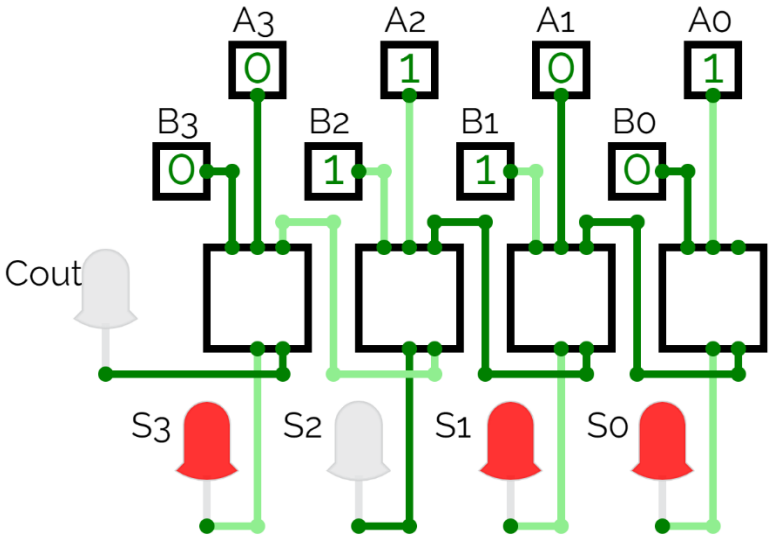
Theory:

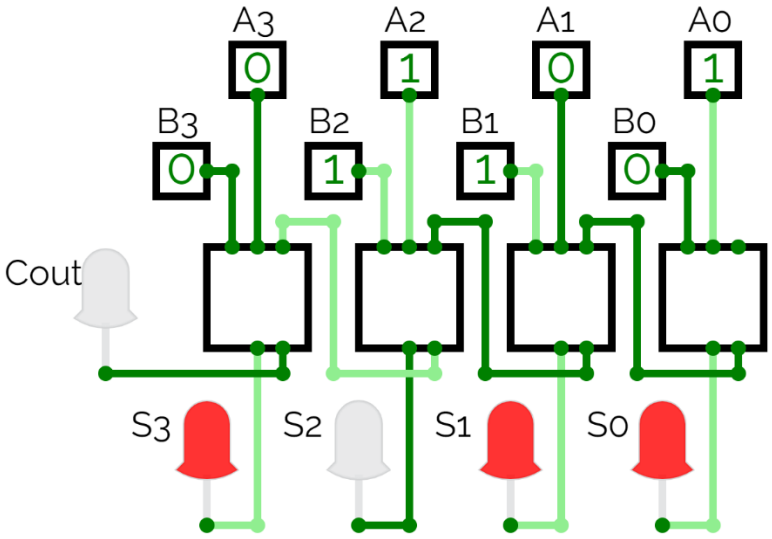
Ripple carry adder: is a combinational logic circuit. Half adders can be used to add two one bit binary numbers. It is also possible to create a logical circuit using full adders to add N-bit binary numbers. Each full adder inputs a C_{IN}, which is Cout of previous adder, this kind of adders are called Ripple carry Adder.

The layout of Ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous bit full adder.

Implementation:




A3
A2
A1
A0
0
1
0
1
B3
B2
B1
B0
0
1
1
0
Cout
S3
S2
S1
S0


A3
A2
A1
A0
0
1
0
1
B3
B2
B1
B0
0
1
1
0
Cout
S3
S2
S1
S0

2003145 - C31

PAGE No.

DATE / /

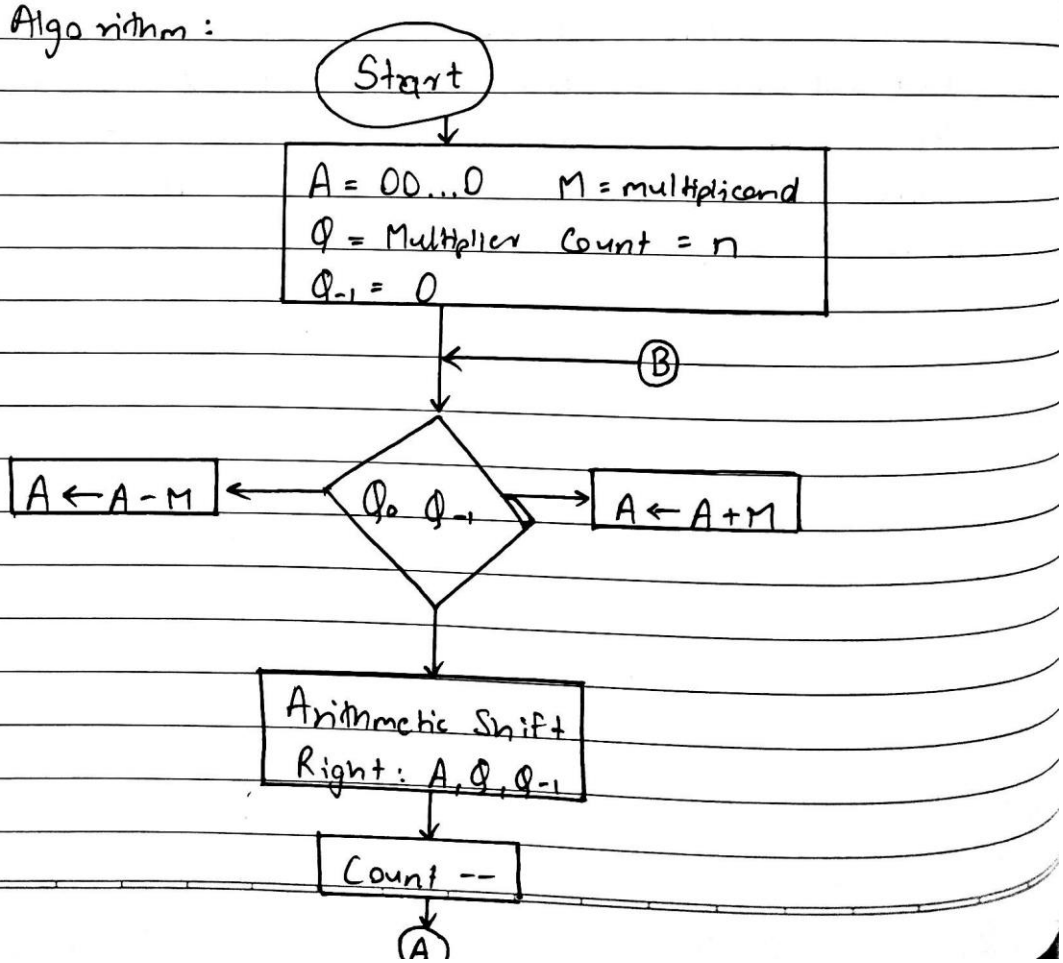
Experiment 7.

Aim: Booth's Algorithm.

Theory: The booth's algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement. It's also used to speed up the performance of the multiplication process. Since its very efficient.

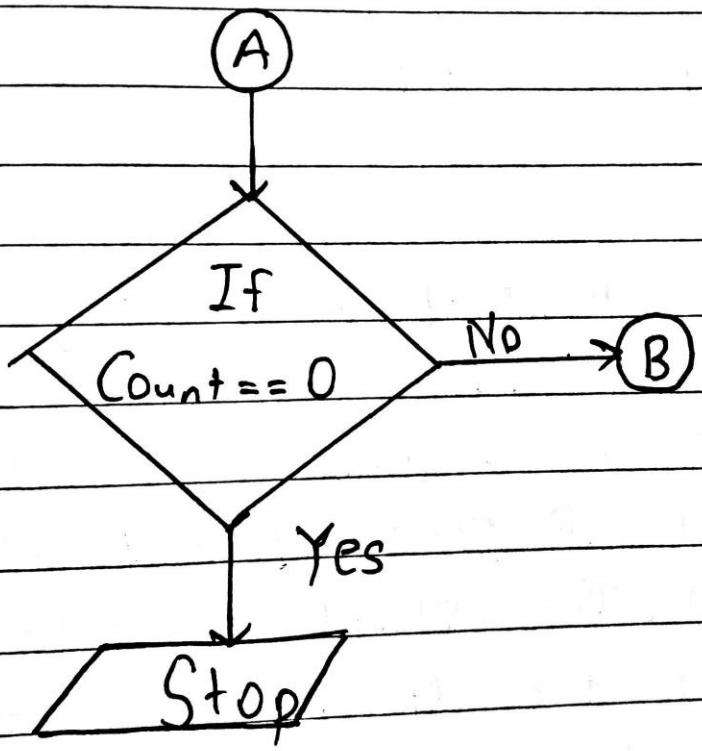
It operates on the fact that strings of 0's in multiplier requires no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to 2^m can be treated as $2^{(k+1)}$ to 2^m.

Algorithm:



PAGE No.

DATE /



Program :

```
#include <stdio.h>
#include <math.h>
int a = 0, b = 0, c = 0, a1 = 0, b1 = 0, com[5] = {1, 0, 0, 0, 0};
int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};
int acomp[5] = {0}, bcomp[5] = {0}, pro[5] = {0}, res[5] = {0};
void binary()
{
    a1 = fabs(a);
    b1 = fabs(b);
    int r, r2, i, temp;
    for (i = 0; i < 5; i++)
    {
        r = a1 % 2;
        a1 = a1 / 2;
        r2 = b1 % 2;
        b1 = b1 / 2;
        anum[i] = r;
        anumcp[i] = r;
        bnum[i] = r2;
        if (r2 == 0)
        {
            bcomp[i] = 1;
        }
        if (r == 0)
        {
            acomp[i] = 1;
        }
    }
    c = 0;
    for (i = 0; i < 5; i++)
    {
        res[i] = com[i] + bcomp[i] + c;
        if (res[i] >= 2)
        {
            c = 1;
        }
        else
            c = 0;
        res[i] = res[i] % 2;
    }
    for (i = 4; i >= 0; i--)
    {
        bcomp[i] = res[i];
    }
    if (a < 0)
    {
        c = 0;
```

```
            for (i = 4; i >= 0; i--)
            {
                res[i] = 0;
            }
            for (i = 0; i < 5; i++)
            {
                res[i] = com[i] + acomp[i] + c;
                if (res[i] >= 2)
                {
                    c = 1;
                }
                else
                    c = 0;
                res[i] = res[i] % 2;
            }
            for (i = 4; i >= 0; i--)
            {
                anum[i] = res[i];
                anumcp[i] = res[i];
            }
        }
        if (b < 0)
        {
            for (i = 0; i < 5; i++)
            {
                temp = bnum[i];
                bnum[i] = bcomp[i];
                bcomp[i] = temp;
            }
        }
    }
    void add(int num[])
    {
        int i;
        c = 0;
        for (i = 0; i < 5; i++)
        {
            res[i] = pro[i] + num[i] + c;
            if (res[i] >= 2)
            {
                c = 1;
            }
            else
            {
                c = 0;
            }
            res[i] = res[i] % 2;
        }
        for (i = 4; i >= 0; i--)
        {
```

```
            pro[i] = res[i];
            printf("%d", pro[i]);
        }
        printf(":");
        for (i = 4; i >= 0; i--)
        {
            printf("%d", anumcp[i]);
        }
    }
    void arshift()
    {
        int temp = pro[4], temp2 = pro[0], i;
        for (i = 1; i < 5; i++)
        {
            pro[i - 1] = pro[i];
        }
        pro[4] = temp;
        for (i = 1; i < 5; i++)
        {
            anumcp[i - 1] = anumcp[i];
        }
        anumcp[4] = temp2;
        printf("\nAR-SHIFT: ");
        for (i = 4; i >= 0; i--)
        {
            printf("%d", pro[i]);
        }
        printf(":");
        for (i = 4; i >= 0; i--)
        {
            printf("%d", anumcp[i]);
        }
    }
    void main()
    {
        int i, q = 0;
        printf("\nEnter two numbers to multiply: ");
        printf("\nBoth must be less than 16");
        do
        {
            printf("\nEnter A: ");
            scanf("%d", &a);
            printf("Enter B: ");
            scanf("%d", &b);
        } while (a >= 16 || b >= 16);
        printf("\nExpected product = %d", a * b);
        binary();
        printf("\n\nBinary Equivalents are: ");
        printf("\nA = ");
        for (i = 4; i >= 0; i--)
```

```
    {
        printf("%d", anum[i]);
    }
    printf("\nB = ");
    for (i = 4; i >= 0; i--)
    {
        printf("%d", bnum[i]);
    }
    printf("\nB'+ 1 = ");
    for (i = 4; i >= 0; i--)
    {
        printf("%d", bcomp[i]);
    }
    printf("\n\n");
    for (i = 0; i < 5; i++)
    {
        if (anum[i] == q)
        {
            printf("\n-->");
            arshift();
            q = anum[i];
        }
        else if (anum[i] == 1 && q == 0)
        {
            printf("\n-->");
            printf("\nSUB B: ");
            add(bcomp);
            arshift();
            q = anum[i];
        }
        else
        {
            printf("\n-->");
            printf("\nADD B: ");
            add(bnum);
            arshift();
            q = anum[i];
        }
    }
    printf("\nProduct is = ");
    for (i = 4; i >= 0; i--)
    {
        printf("%d", pro[i]);
    }
    for (i = 4; i >= 0; i--)
    {
        printf("%d", anumcp[i]);
    }
}
```

Output :

```
Enter two numbers to multiply:
Both must be less than 16
Enter A: 7
Enter B: 8

Expected product = 56

Binary Equivalents are:
A = 00111
B = 01000
B'+ 1 = 11000


-->
SUB B: 11000:00111
AR-SHIFT: 11100:00011
-->
AR-SHIFT: 11110:00001
-->
AR-SHIFT: 11111:00000
-->
ADD B: 00111:00000
AR-SHIFT: 00011:10000
-->
AR-SHIFT: 00001:11000
Product is = 0000111000PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs\7th expt>
```

```
Enter two numbers to multiply:
Both must be less than 16
Enter A: 5
Enter B: -7

Expected product = -35

Binary Equivalents are:
A = 00101
B = 11001
B'+ 1 = 00111


-->
SUB B: 00111:00101
AR-SHIFT: 00011:10010
-->
ADD B: 11100:10010
AR-SHIFT: 11110:01001
-->
SUB B: 00101:01001
AR-SHIFT: 00010:10100
-->
ADD B: 11011:10100
AR-SHIFT: 11101:11010
-->
AR-SHIFT: 11110:11101
Product is = 1111011101PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs\7th expt>
```

2003145 - C31

PAGE No.
DATE / /

Experiment 8

Aim: Carry look ahead generator.

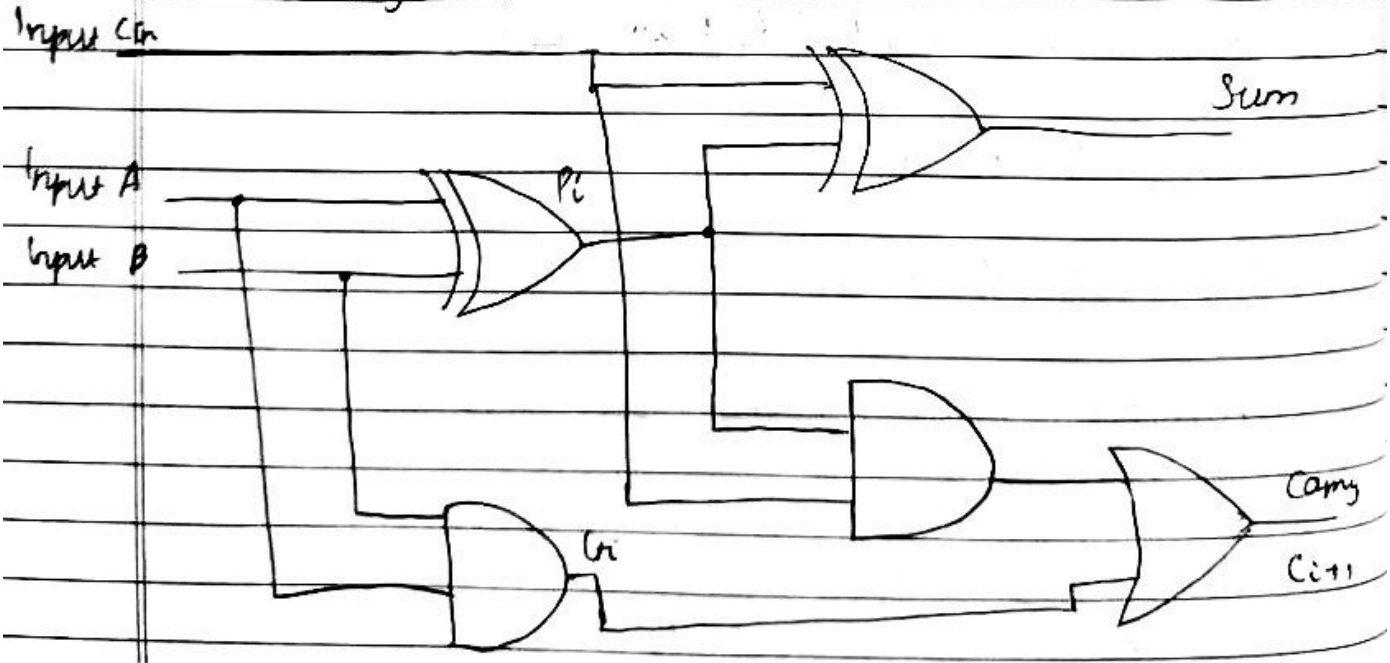
Theory: In ripple carry adders, for each adder blocks, the two bits are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block waits for the carry to arrive from previous block. So its not possible to generate the sum and carry of any block until the input carry is known. The i^{th} block waits for the $i-1^{th}$ block to produce its carry.

Carry Looks-ahead adder is the fasted adder circuit. It reduces the propogation delay, which occurs during addition, by using more complex hardware circuitry. It is designed by transforming the ripple carry adder circuit such that the carry logic of the adder is changed into two-level logic.

It can be observed from the equations that carry C_{i+1} only depends on the carry C_0, not on the immediate carry bits

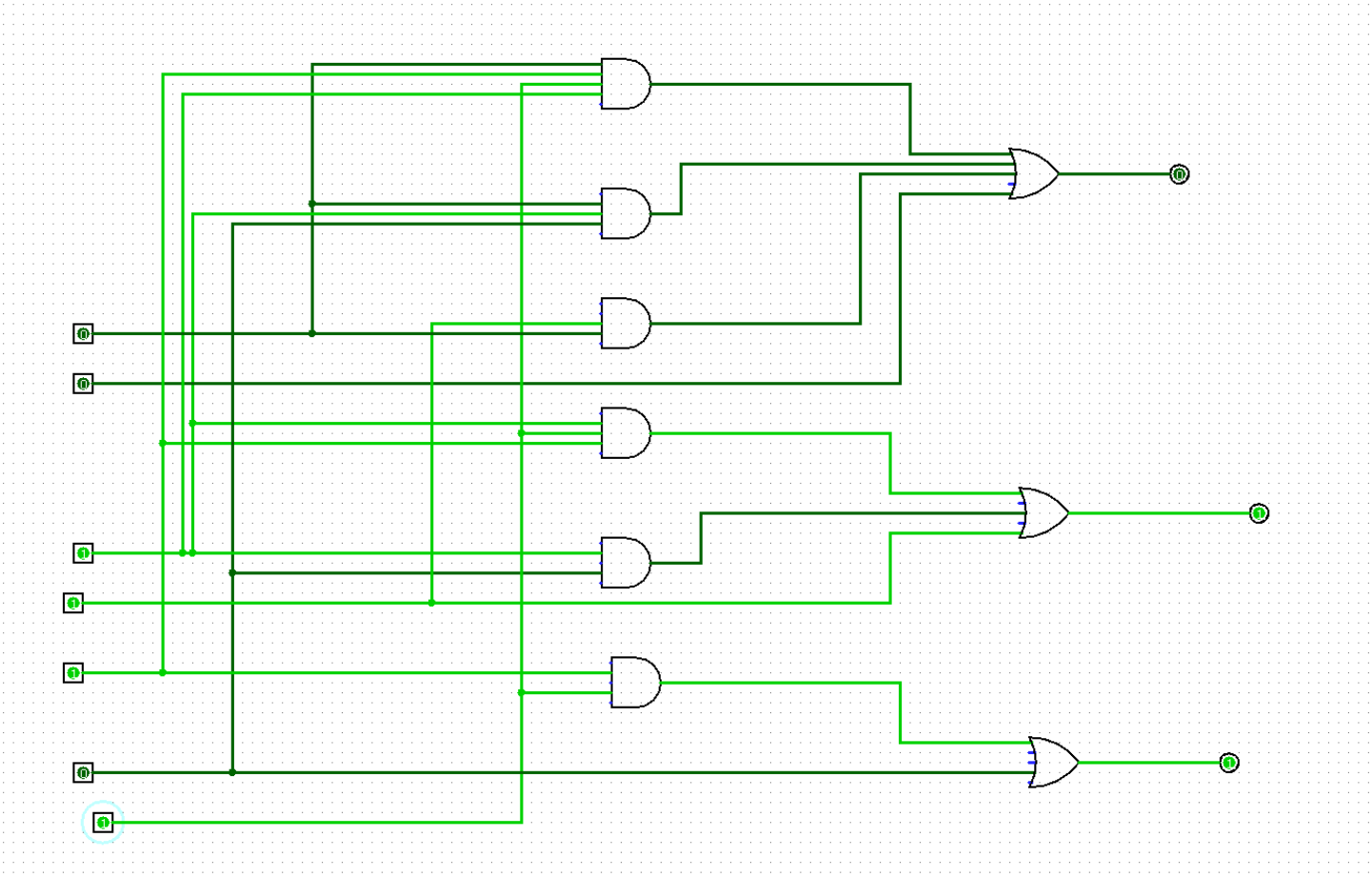
A	B	C_i	C_{i+1}	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

Circuit diagram :



Output :

Circuit Diagram of Carry Look Ahead Adder using Logisim



2007145 - C11

PAGE No.

DATE / /

Experiment 9.

Aim : IEEE 754 conversion.

Theory :

The IEEE standard for floating point arithmetic (IEEE 754) is a technical standard for floating point computation which was established in 1985 by the Institute of Electrical and electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE standard 754 f.p. is the most common representation today for real numbers on computers, including Intel based pc's, mac's, and most Unix platforms.

floating point representation :

Sign bit	Biased exponent	Significand or mantissa.

$= +/- \text{ Significand} \times 1^{\text{exponent.}}$

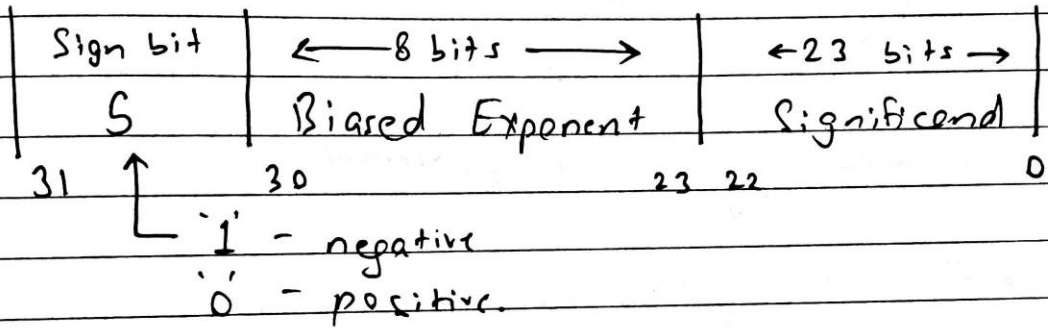
- Exponent signifies the point position.

Normalization :

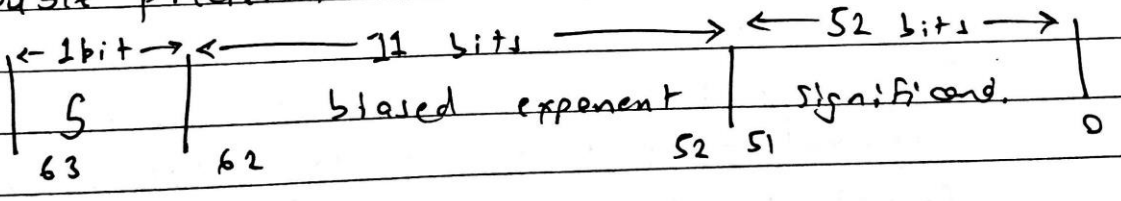
- floating point numbers are usually normalized ie adjusted so that the leading bit of mantissa (MSB) is 1.

- Since its always 1, there is no need to store it.

1]. Single precision format : (32 bit)



2]. Double precision format : (64 bit).



Program :

```c
#include <stdio.h>

int binary(int n, int i)
{
    int k;
    for (i--; i >= 0; i--)
    {
        k = n >> i;
        if (k & 1)
            printf("1");
        else
            printf("0");
    }
}

typedef union
{
    float f;
    struct
    {
        unsigned int mantissa : 23;
        unsigned int exponent : 8;
        unsigned int sign : 1;
    } field;
} myfloat;

int main()
{
    myfloat var;
    printf("Enter any float number: ");
    scanf("%f", &var.f);
    printf("%d ", var.field.sign);
    binary(var.field.exponent, 8);
    printf(" ");
    binary(var.field.mantissa, 23);
    printf("\n");
    return 0;
}
```

Output :

```
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> cd "c:\Users\Ismail
LCA Labs"
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"IEEE754.exe"
Enter any float number: 234.7567
0 10000110 11010101100000110110111
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"IEEE754.exe"
Enter any float number: 100.00
0 10000101 10010000000000000000000
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs> & .\"IEEE754.exe"
Enter any float number: 1024.1024
0 10001001 00000000000001101000111
PS C:\Users\IsmailRatlamwala\Documents\College prog\DLCA Labs>
```

2003145 - C31

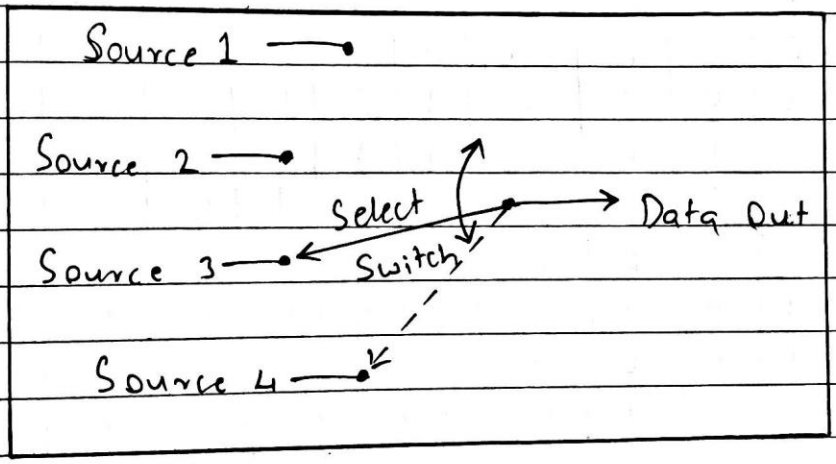
PAGE No.
DATE / /

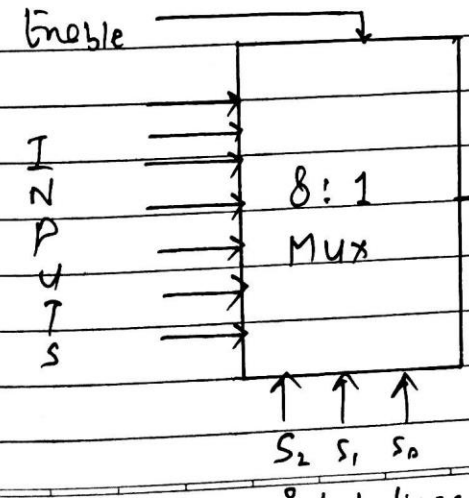
Experiment 10

Aim: Implement a 8:1 Mux and 3:8 Decoder.

Theory:

Multiplexer: The Mux is a digital switch also called as data selector. It is a combinational logic circuit with more than 1 input line and one output line and several select line. It accepts several input lines or sources and depending on the set of select lines, a particular input line is routed to onto a single output line.





Decoder:

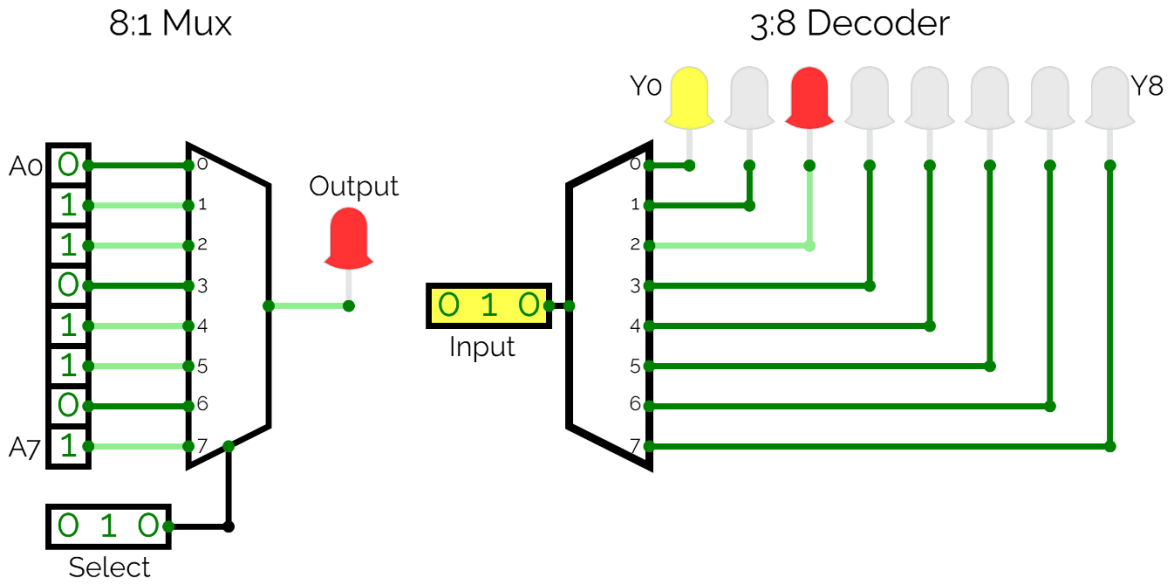
A decoder is a combinational logic circuit that is used to changed the code into set of signals. A decoder takes multiple inputs and multiple outputs It takes 'n' bits of binary data and gives 2^n unique outputs.

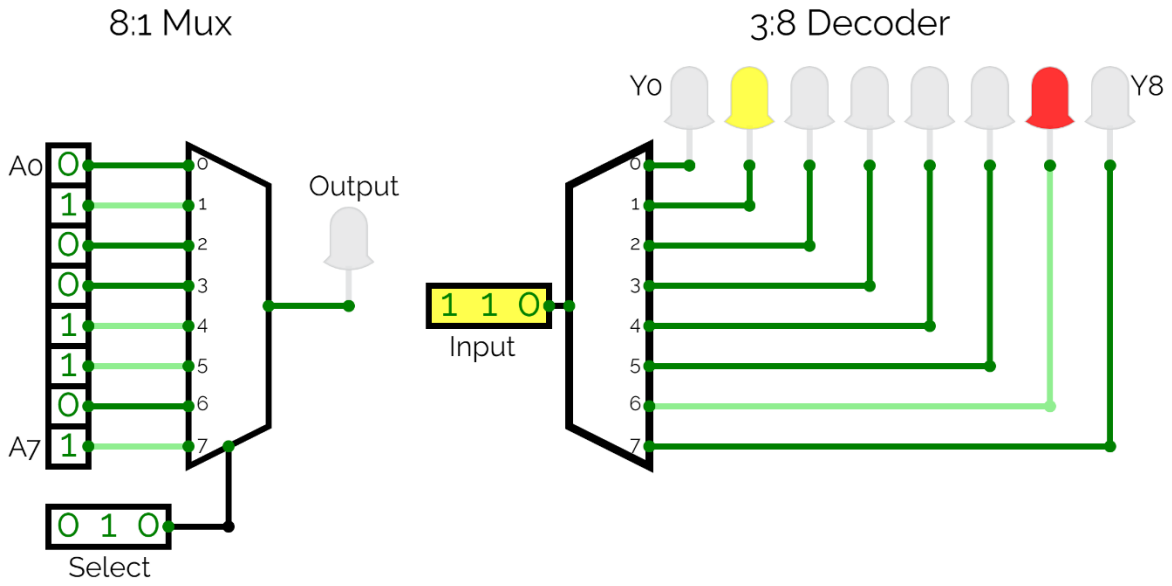
A 3:8 decoder has 3 inputs & 8 outputs, based on the 3 inputs 8 outputs are selected.

Truth Table:

A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Output :


8:1 Mux
3:8 Decoder
A0
A7
Output
Select
Input
Y0
Y8


8:1 Mux
3:8 Decoder
A0
A7
Output
Select
Input
Y0
Y8

Idris Ratlamwala

2003145-C31

PAGE No.

DATE / /

DLCA. Assingment 1.

Q. Write short note on cache mapping:

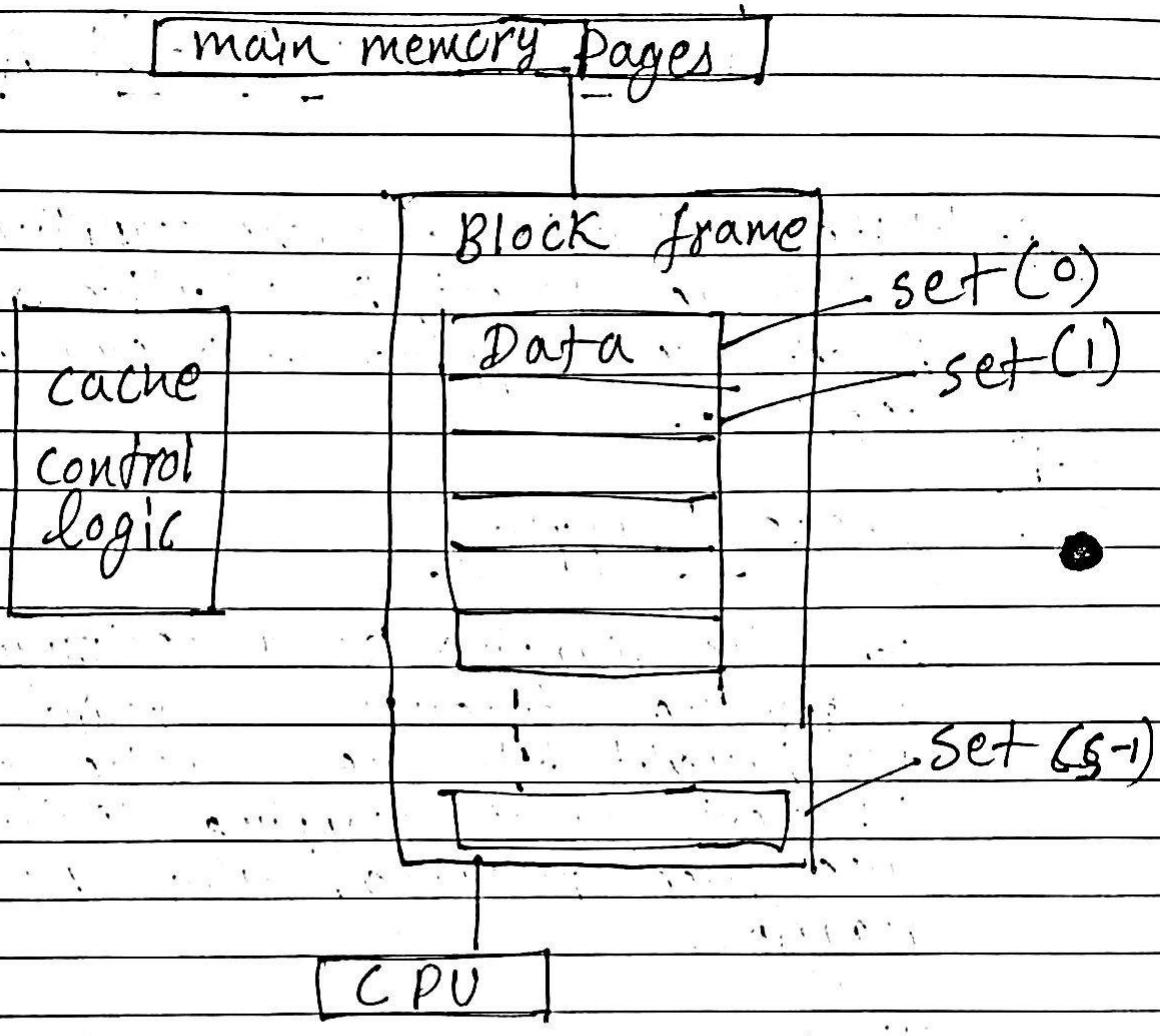
→ Cache Mapping:

Cache mapping is a technique by which the contents of main memmory are brought into the cache memory. Different cache memory techiques are, direct mapping, fully associative mapping.

i) Direct Mapping:

The ~~simplest~~ technique known as direct mapping maps each block of main memory into only one possible cache line or

In direct mapping, assign cache memory block to a specific line in the cache. if a line is previously taken up by a memory block when a new block need to be loaded.



(ii) Associative Mapping

In this type of mapping the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache.

This means that the word id bits are used to identify which word in the block is needed; but the tag becomes all of the remaining bits.

→ The concept of multicore technology is mainly centered on the possibility of parallel computing, which can significantly boost computer speed and efficiency by including two or more central processing units (CPUs) in a single chip.

→ ~~This means~~

→ This reduces the system's heat and power consumption. This means much better performance with less or the same amount of energy.

Assignment - 2 (DLCA)

Q) Write a short note on Multi Core Architecture.

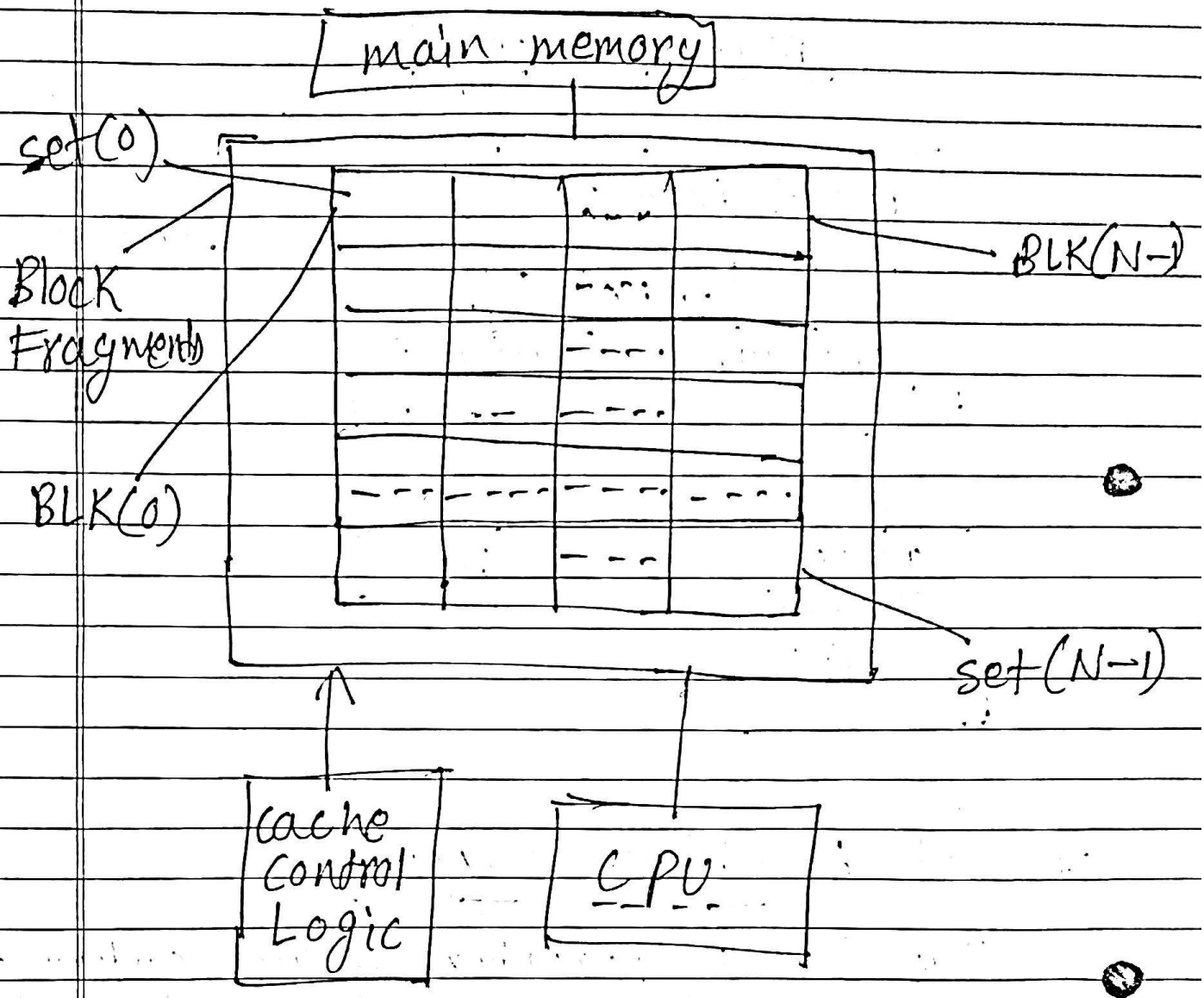
→ multicore referees to an architecture in which a single physical processor incorporates the core logic of more than one processor.

→ A single integrated circuit is used to package or hold the processors. These single integrated circuits are known as a die. Multicore architecture places multiple processor cores and bundels them as a single physical processor.

→ The objective is to create a system that can complete more tasks at the same time, thereby gaining better overall system performance.

→ The technology is most commonly used in multicore processors, where two or more processor chips or cores run concurrently as a single system.

→ Multicore Based processors are used in mobile devices, desktops, workstations and servers.


main memory
set (0)
Block Fragment
BLK(0)
BLK(N-1)
set (N-1)
cache control Logic
CPU