LOKMANYA TILAK COLLEGE OF ENGINEERING

Sector No. 4., Vikas Nagar, Koparkhairane, Navi Mumbai -400 709.

Computer Science and Engineering

(Artificial Intelligence & Machine Learning)

AY 2021-22 (Odd)

(SEM. III)

Practical file

for

Digital Logic &

Computer Architecture Lab (DLCA)

(CSL-302)

Name of Student: **SINGH SUDHAM DHARMENDRA**

Roll no. : **AIMLD50**

Faculty Incharge: Prof. Gauri Ashtankar

DLCA PRACTICALS

NAME : SINGH SUDHAM DHARMENDRA

ROLL NO. : AIMLD50

BATCH : A2

BRANCH : CSE(AIML)

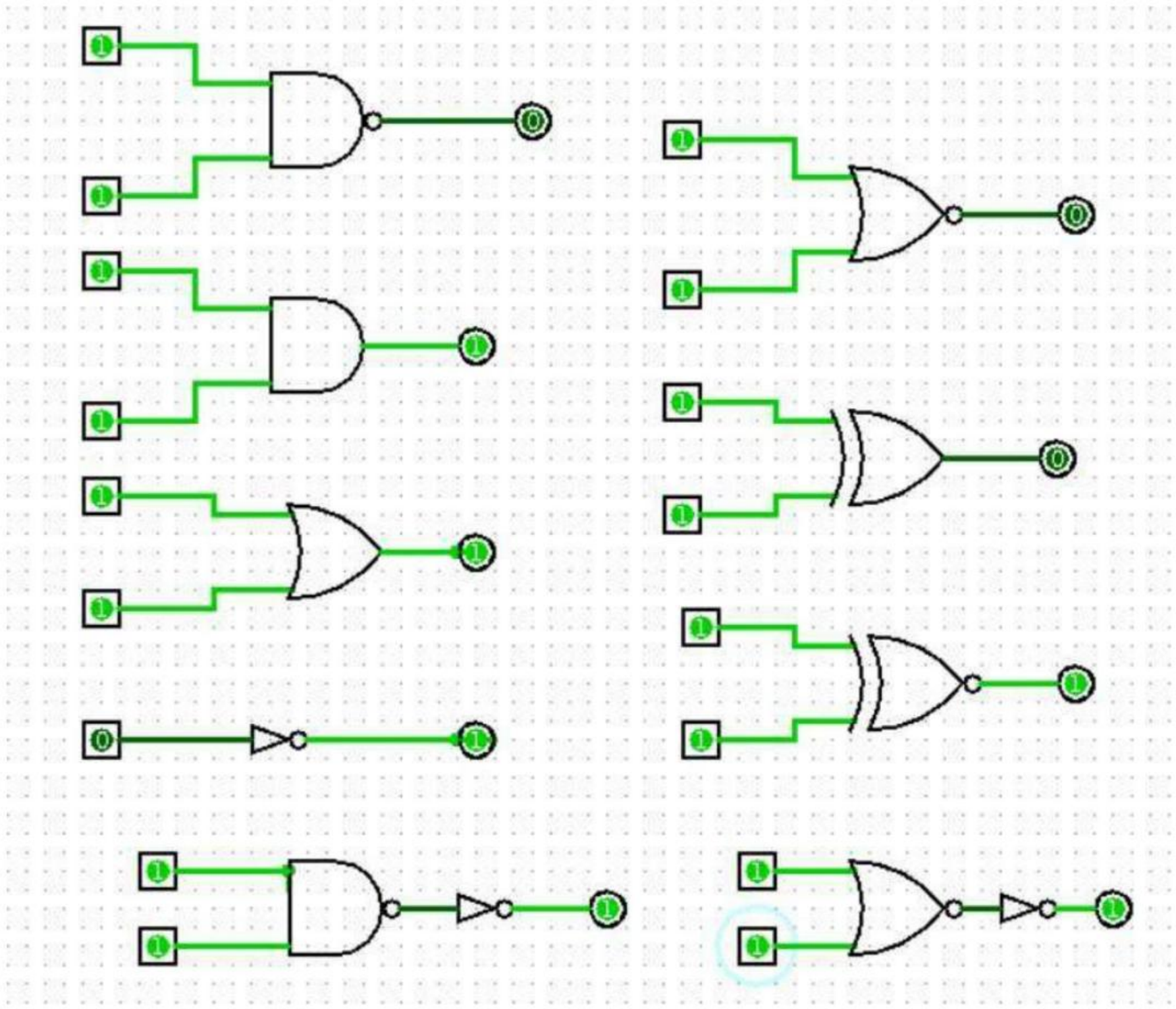
DATE OF SUBMISSION : 26/11/2021

INDEX

EXPERIMENT NO: 01

STUDY OF LOGIC GATES.

- **DATE OF PERFORMANCE: -** 21/09/2021
- **AIM: -** To study different types of logic gates.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



- **RESULT: -**

AND GATE		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NAND GATE		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

EX-OR		
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

NOR GATE		
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

EX-NOR		
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	1

NOT GATE	
A	**Y**
0	1
1	0

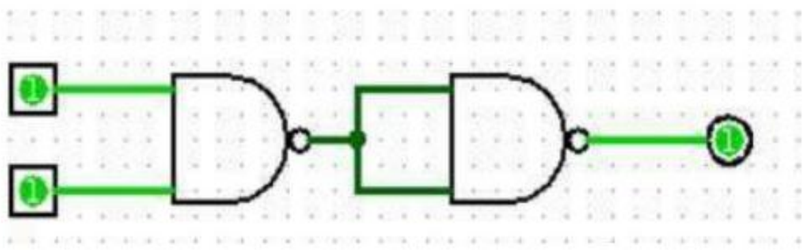
CONCLUSION: -

Different logic gates are studied and verified by using truth tables.

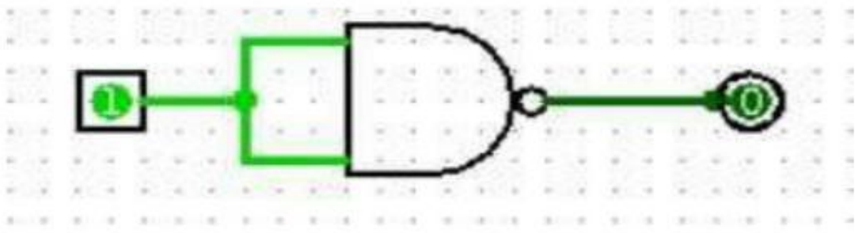
EXPERIMENT NO: 02

Verification of NAND gate as Universal gate and design a NANDlogic circuit that is equivalent to the AOI circuit.

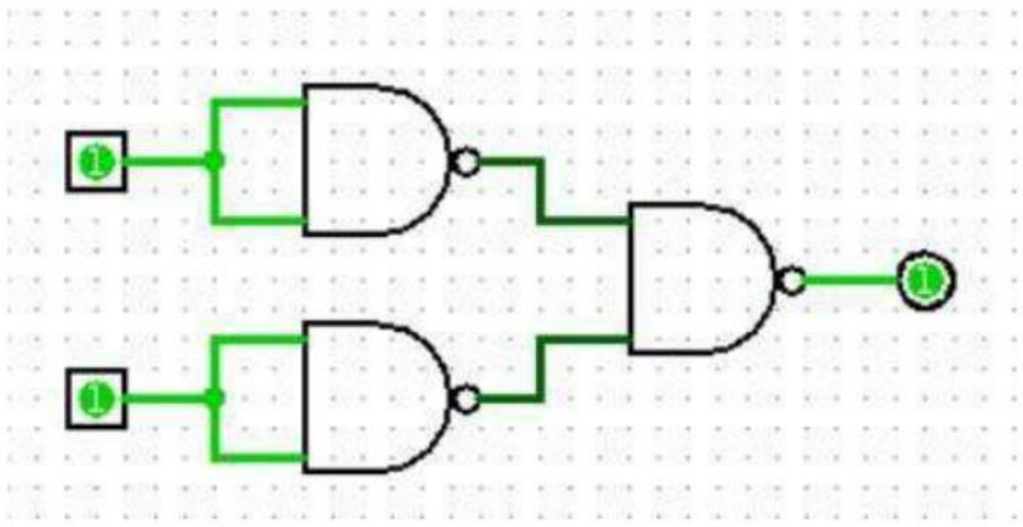
- **DATE OF PERFORMANCE: -** 28/09/2021
- **AIM: -** To verify NAND gate as Universal gate and design a NAND logic circuitthat is equivalent to the AOI circuit shown below.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



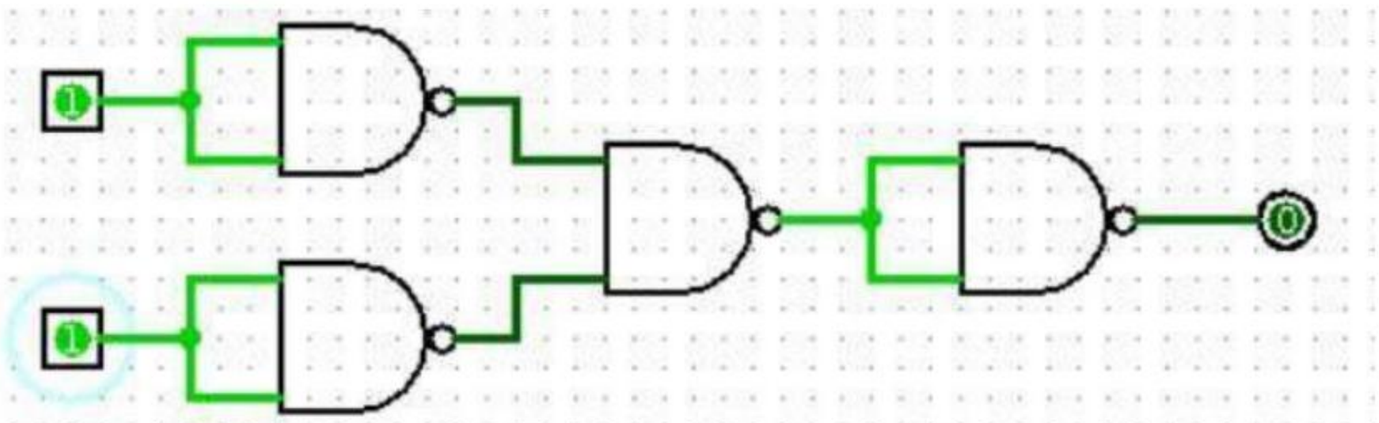
AND gate using NAND gates



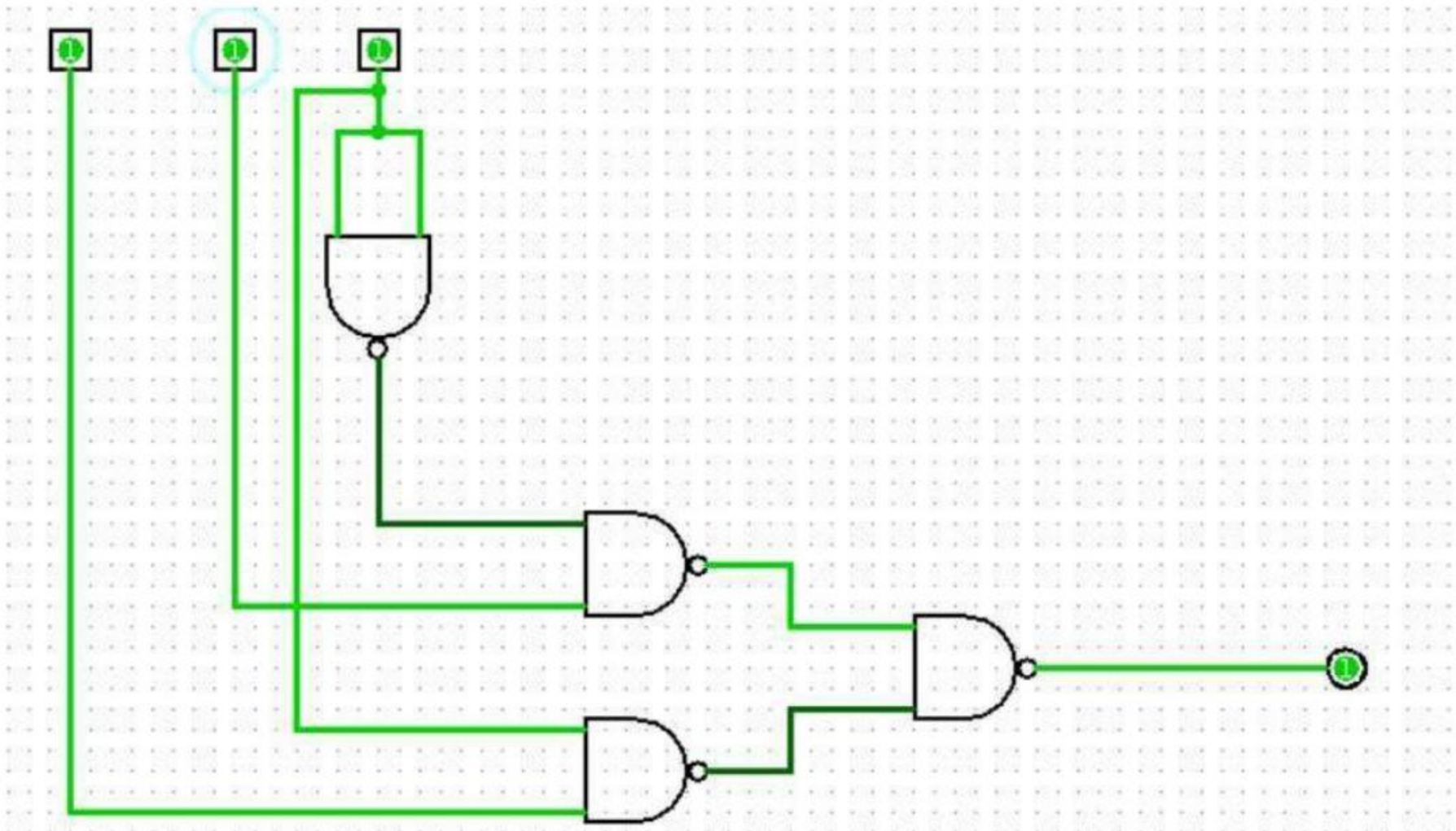
NOT gate using NAND gate



OR gate using NAND gates



NOR gate using NAND gates



NAND logic circuit equivalent to AOI Circuit

RESULT: -

AND using NAND: -

To make the AND function from NAND gates, all that is needed is an inverter (NOT) stage on the output of a NAND gate.

AND gate using NAND gate		
A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

OR using NAND: -

The NAND gate requires inversion of all inputs to mimic the OR function.

OR gate using NAND gate		
A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

NOR using NAND: -

All the inputs and output must be inverted in order to make NAND function as a NOR gate.

NOR gate using NAND gate		
A	B	OUTPUT
0	0	1
0	1	0
1	0	0
1	1	0

NOT using NAND: -

NOT gate usingNAND gate	
A	OUTPUT
0	1
1	0

3 INPUTS LOGIC NAND GATE			
A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

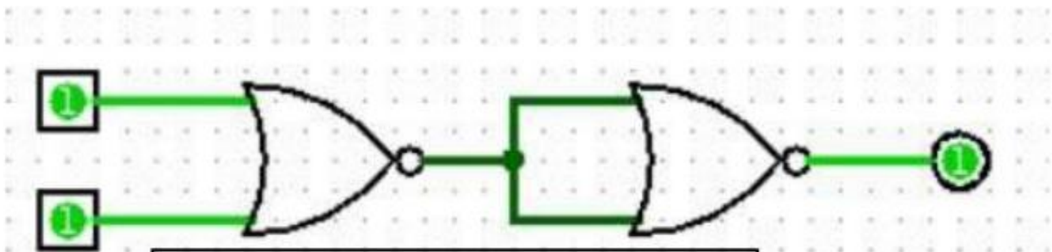
CONCLUSION: -

From the experiment we conclude that all the NAND gate as Universal gate and NAND Logic Circuits that are equivalent to the AOI circuit are verified correctly.

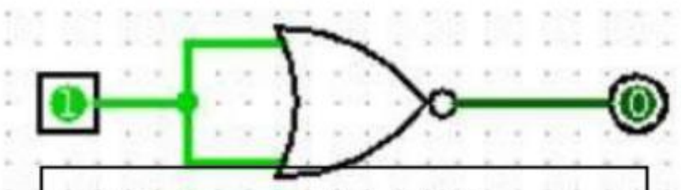
EXPERIMENT NO: 03

Verification of NOR gate as Universal gate and design a NORlogic circuit that is equivalent to the AOI circuit shown.

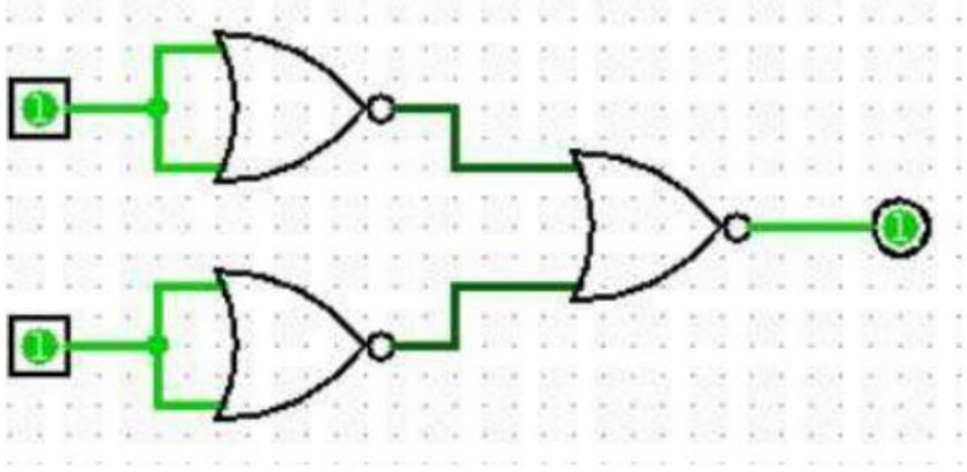
- **DATE OF PERFORMANCE: -**13/10/2021
- **AIM: -** To verify NOR gate as Universal gate and design a NOR logic circuit thatis equivalent to the AOI circuit shown.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



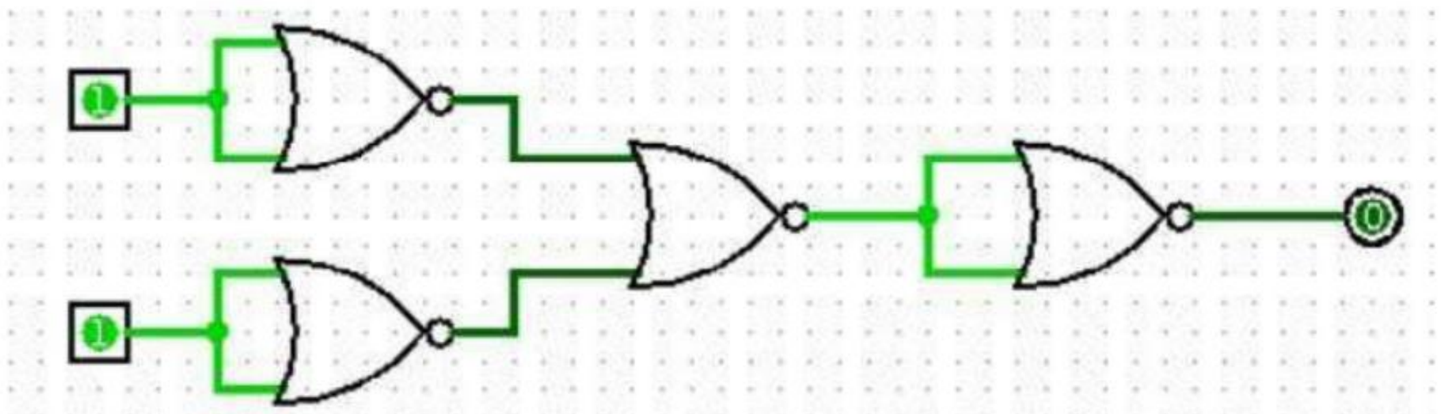
OR gate using NOR gates



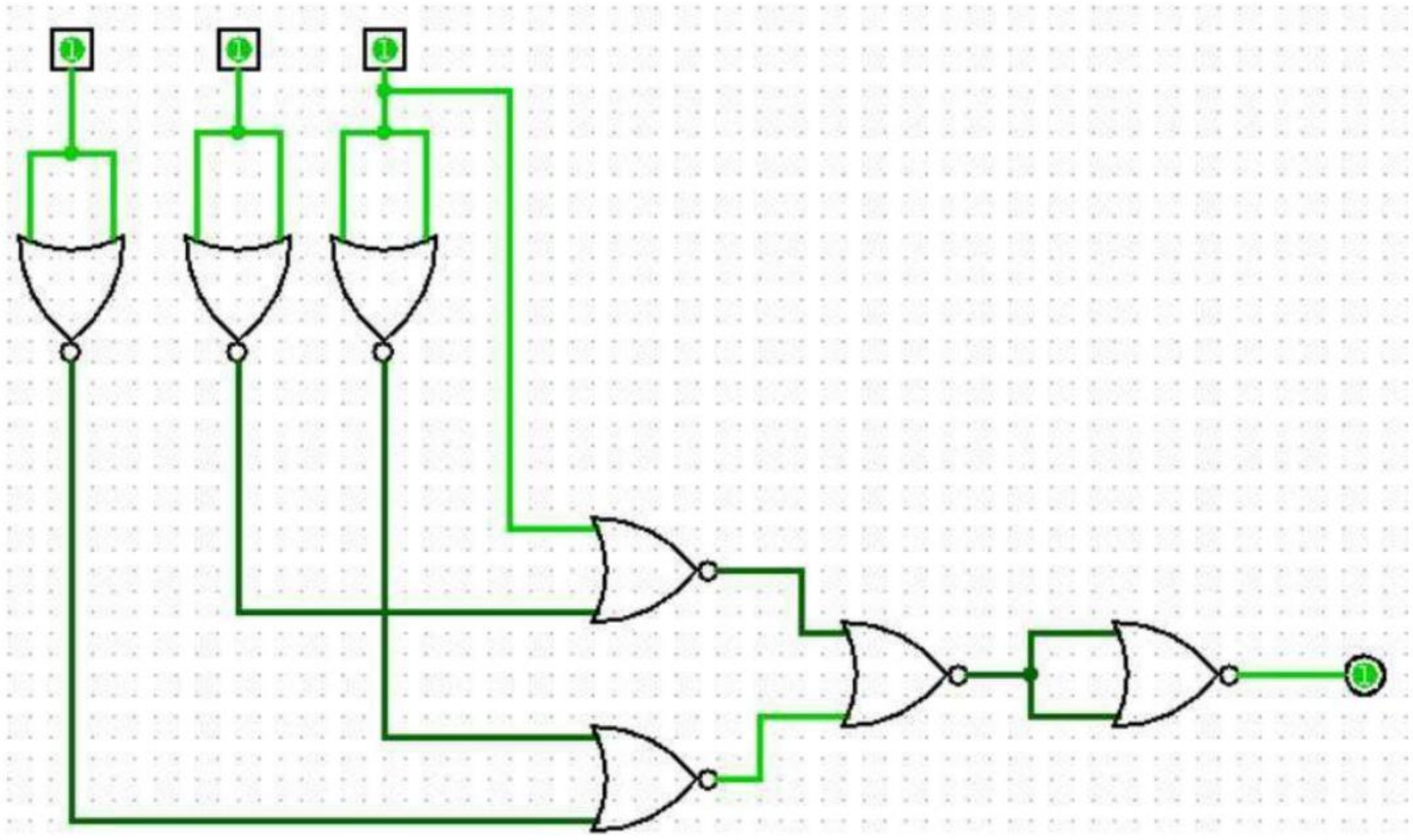
NOT gate using NOR gate



AND gate using NOR gates



NAND gate using NOR gates



NOR Logic circuit equivalent to AOI circuit.

- **RESULT: -**

NAND using NOR: -

To make a NOR gate perform the NAND function, we must invert all the inputs to the NOR gate as well as the NOR gate's output.

NAND gate using NOR gate		
A	**B**	**OUTPUT**
0	0	1
0	1	1
1	0	1
1	1	0

AND gate using NOR gate		
A	**B**	**OUTPUT**
0	0	0
0	1	0
1	0	0
1	1	1

OR gate using NOR gate		
A	**B**	**OUTPUT**
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate using NOR gate	
A	**OUTPUT**
0	1
1	0

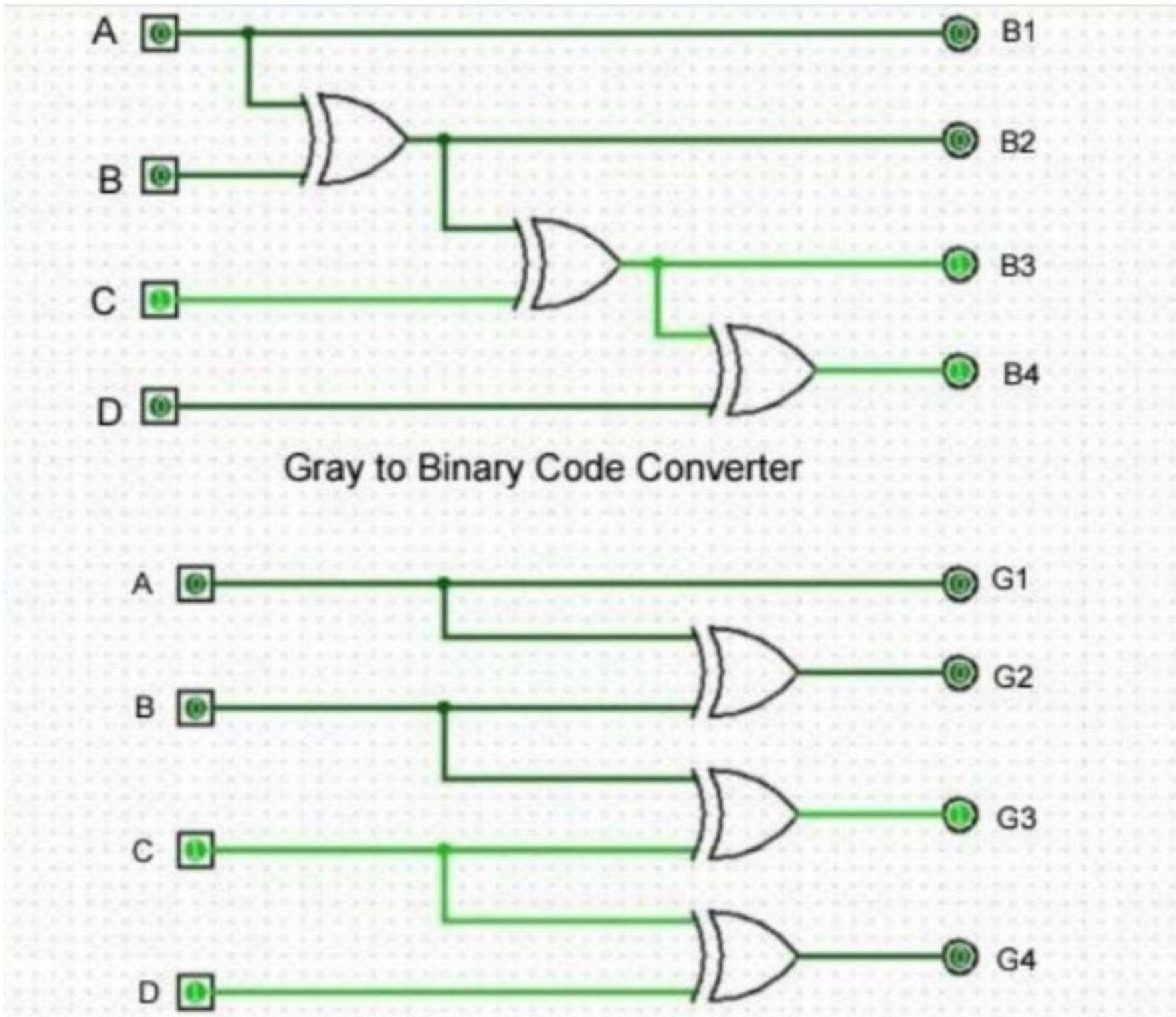
- **CONCLUSION: -**

From the experiment we conclude that all the NOR gate as Universal gate and the NOR logic circuits that are equivalent to the AOI circuit are verified correctly.

EXPERIMENT NO: 04

Study of Binary Gray code converter and Gray to Binary.

- **DATE OF PERFORMANCE: -**20/10/2021
- **AIM: -** To study Binary Gray code converter and Gray to Binary.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



- **RESULT: -**

Truth Table

GRAY CODE				BINARY CODE			
A/G1	B/G2	C/G3	D/G4	B1/A	B2/B	B3/C	B4/D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

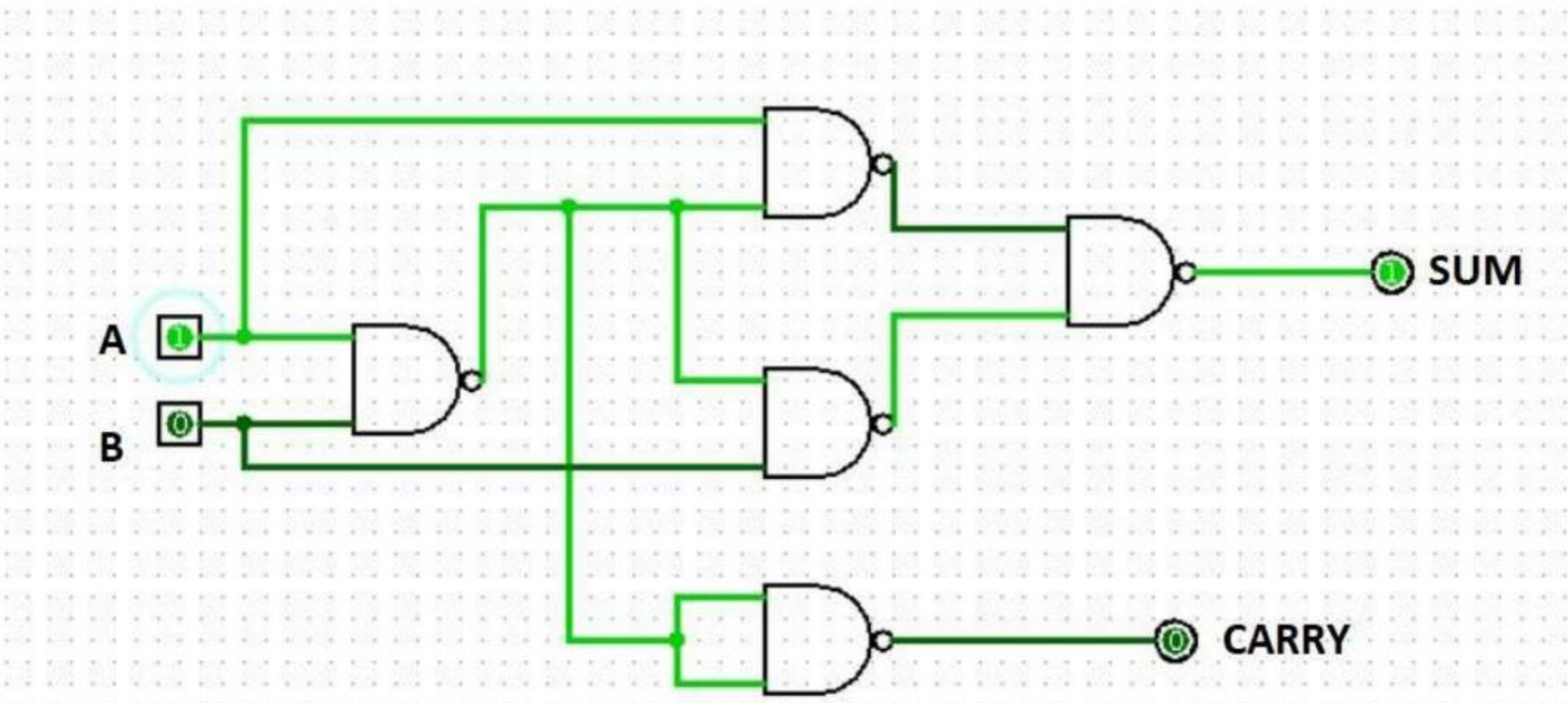
- **CONCLUSION: -**

From the above experiment we have verified the Binary code to Gray code and Gray code to Binary code conversion using truth table.

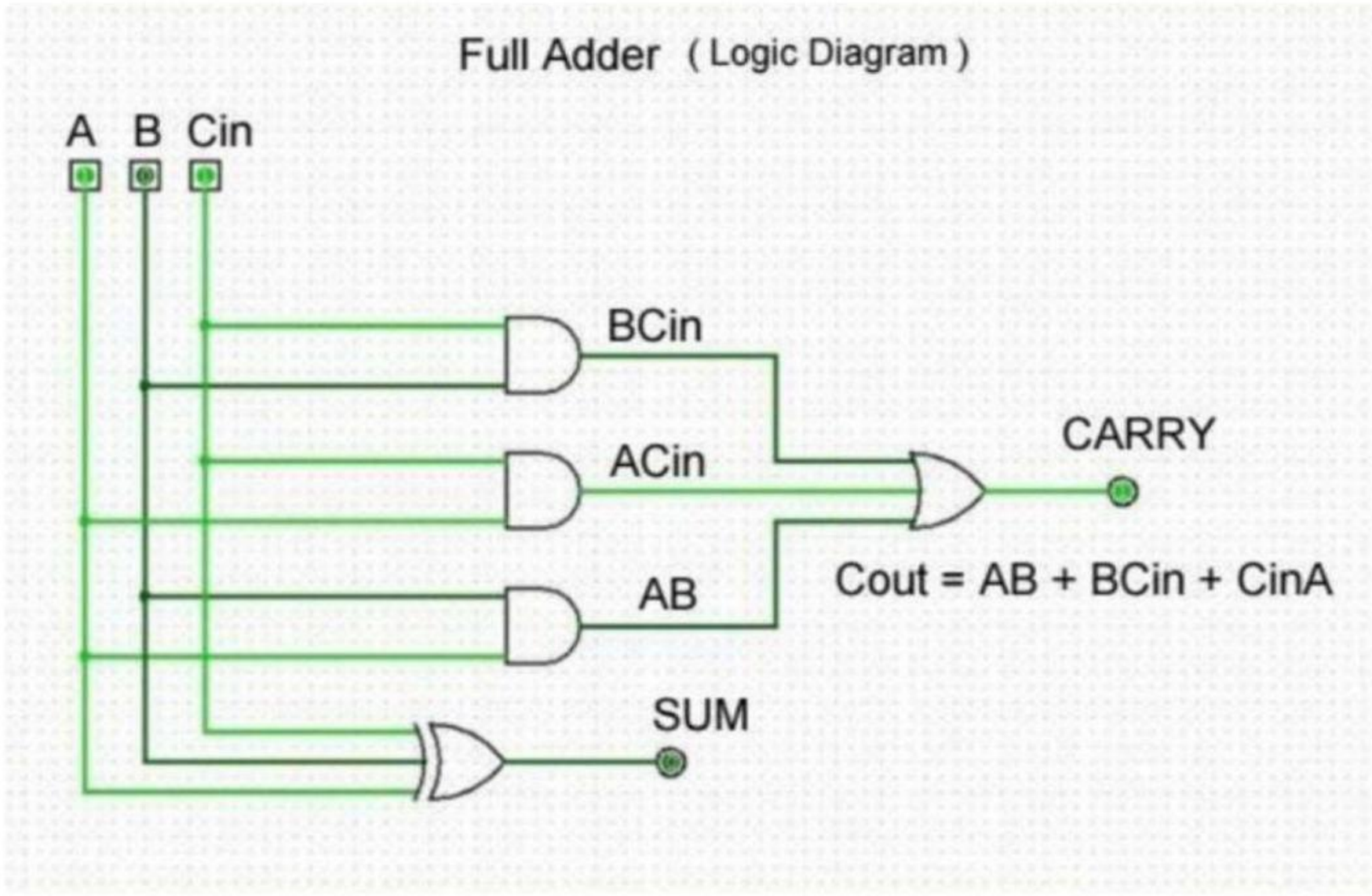
EXPERIMENT NO: 05

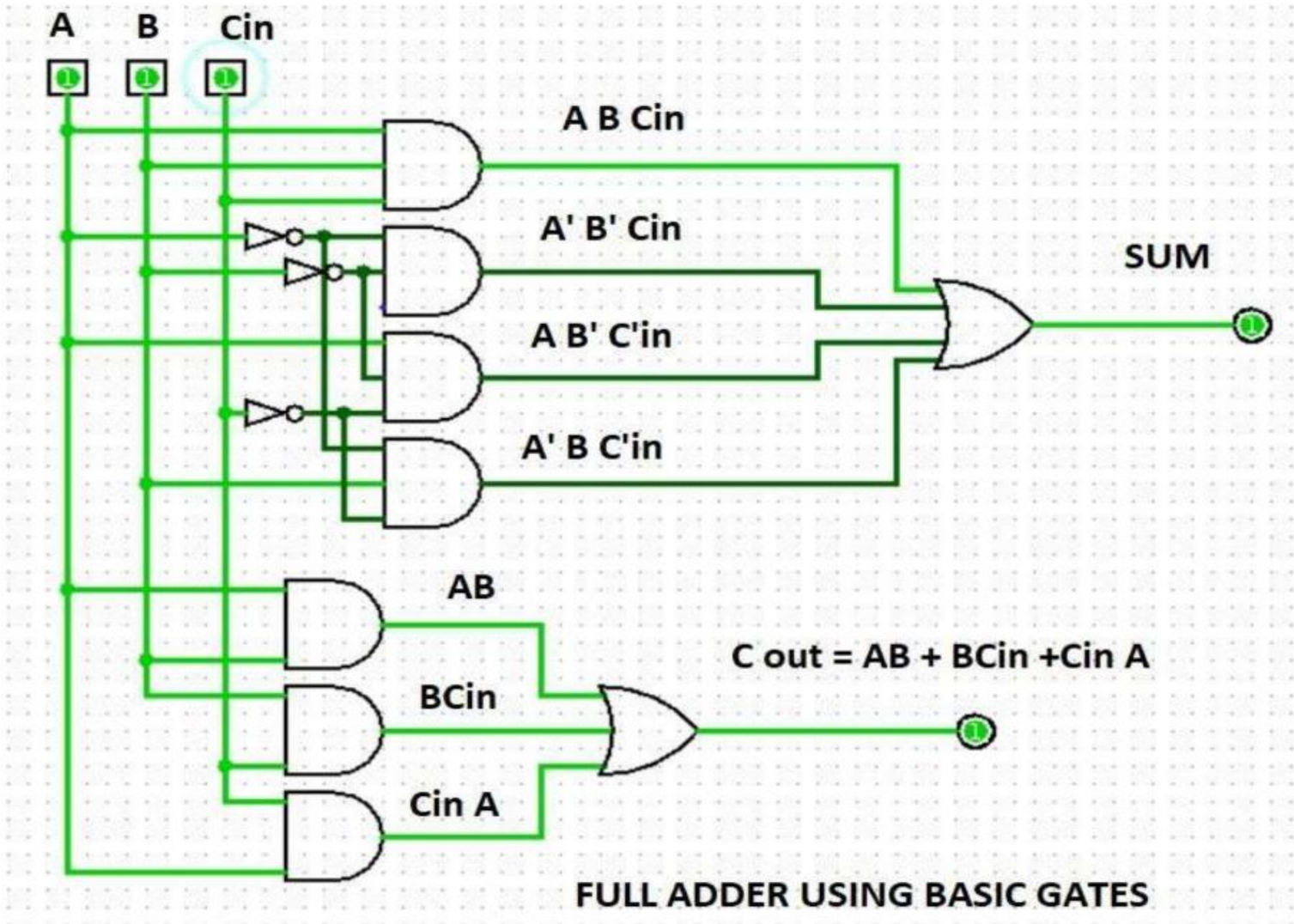
Study of Half and Full adder.

- **DATE OF PERFORMANCE: -**20/10/2021
- **AIM: -** To study half adder and full adder.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



HALF ADDER USING NAND





- **RESULT: -**

HALF ADDER			
INPUTS		OUTPUTS	
AUGEND(A)	AUGEND(B)	SUM(S)	CARRY(C)
0	**0**	**0**	**0**
0	1	1	0
1	0	1	0
1	1	0	1

FULL ADDER				
INPUTS			OUTPUTS	
A	B	Cin	SUM(S)	CARRY (Co)
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
Cin = Third input carry from previous lower significant position				

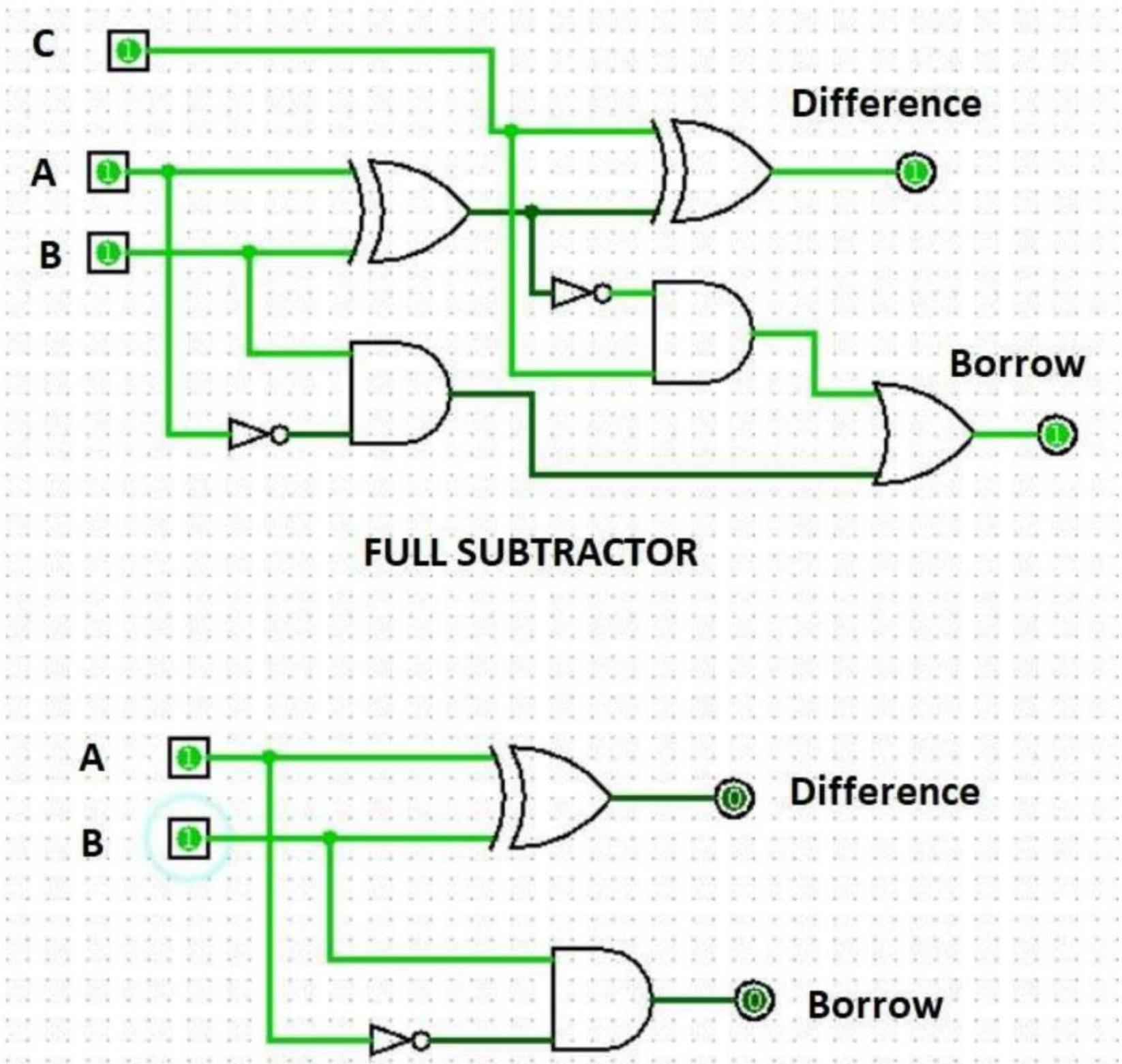
- **CONCLUSION: -**

From these experiments we learned about the Half Adder using Universal Gateand Full Adder using Basic Gates by verifying the truth tables for both the circuits.

EXPERIMENT NO: 06

Study of Half and Full subtractor.

- **DATE OF PERFORMANCE: -** 27/10/2021
- **AIM:-**To study Half Subtractor and Full Subtractor.
- **SOFTWARE: -** Logisim
- **SCREEN SHOTS: -**



- **RESULT: -**

HALF SUBTRACTOR			
INPUTS		OUTPUTS	
MINUEND(A)	SUBTRAHEND(B)	DIFFERENCE(D)	BORROW (Bo)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

FULL SUBTRACTOR				
INPUTS			OUTPUTS	
MIINUEND	SUBTRAHEND	BORROW	DIFFERENCE	BORROW
A	B	B in	D	Bo
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
B in = borrow input				

- **CONCLUSION: -**

Form this experiment we successfully designed the Half Subtractor and FullSubtractor by verifying the truth tables for both the circuits.

EXPERIMENT NO: 07

Implementation of Booth Algorithm for Binary multiplication.

- **DATE OF PERFORMANCE: -**24/11/2021
- **AIM: -** To implement Booth Algorithm for Binary multiplication.
- **SOFTWARE: -** Visual Studio Code.
- **SOURCE CODE:-**

```
#include<stdio.h>
#include<math.h>
#define BIT 4
void
d_b (int dec_s, int bin[BIT])
{

int carry, dec, i;

carry = 0;

dec = dec_s > 0 ? dec_s : -dec_s;

for (i = 0; i < BIT; i++)

    {

bin[i] = (dec % 2);
      dec /= 2;
    }

if (dec_s >= 0)

return;

  else
    {

for (int j = 0; j < BIT; j++)
    {

bin[j] = bin[j] == 0 ? 1 : 0;
      }
if (bin[0] + 1 <= 1)

    {

bin[0] += 1;
```

```
return;

}

      else

    {

carry = 1;
      for (int i = 0; i < BIT; i++)

        {

if (bin[i] + carry <= 1)

        {

bin[i] += carry;

carry = 0;

return;

}

          else

        {

bin[i] = (bin[i] + carry) % 2;

}

}

}

}

}

}


int
b_d (int a[BIT], int q[BIT])
{

int i = 0;

int ans = 0;
```

```
for (; i < BIT; i++)

    {

ans += q[i] * pow (2, i);

}

for (int k = 0; k < BIT; k++, i++)

    {

ans += a[k] * pow (2, i);

}
return ans;

}


void
twos (int m[BIT], int m2[BIT])
{

for (int i = 0; i < BIT; i++)
    {

m2[i] = m[i] == 0 ? 1 : 0;
    }
int carry = 1;
  for (int i = 0; i < BIT; i++)
    {

if (m2[i] + carry <= 1)
    {

m2[i] += carry;

carry = 0;

return;

}

      else
    {

m2[i] = (m2[i] + carry) % 2;
    }
```

```
}

}


int
add (int a[BIT], int m[BIT])
{

int carry = 0;

for (int i = 0; i < BIT; i++)
    {

carry += m[i];

if (a[i] + carry <= 1)

    {

a[i] += carry;

carry = 0;

}

      else
    {

a[i] = (a[i] + carry) % 2;

carry = 1;

}

}

return carry;

}


int
ashr (int a[BIT], int q[BIT], int q_1)
{

q_1 = q[0];

for (int i = 1; i < BIT; i++)
```

```
    {

q[i - 1] = q[i];

}
q[BIT - 1] = a[0];

for (int i = 1; i < BIT; i++)

    {

a[i - 1] = a[i];

}
return q_1;

}


int
main ()
{

int multiplicand, multiplier, q_1, count;

printf (" Enter the multiplicand: ");

scanf ("%d", &multiplicand);

printf (" Enter the multiplier: ");

scanf ("%d", &multiplier);

q_1 = 0;

int a[BIT] = { 0 };

int m[BIT] = { 0 };

int m2[BIT] = { 0 };

int q[BIT] = { 0 };

count = BIT;

d_b (multiplicand, m);
  d_b (multiplier, q);

twos (m, m2);
```

```
printf ("\n Entered Multiplicand: ");

for (int k = BIT - 1; k >= 0; k--)
    {

printf (" %d", m[k]);

}
printf ("\n Entered Multiplier: ");

for (int k = BIT - 1; k >= 0; k--)
    {

printf (" %d", q[k]);

}
printf ("\n\n\t\tA\t Q\t\tQ-1\tOPERATION");

int a1 = 1;

for (int i = count; i != 0; i--)
    {

printf ("\n LOOP NO:%d ", a1);

a1++;

for (int k = BIT - 1; k >= 0; k--)
    {

printf (" %d", a[k]);

}
printf ("\t");

for (int k = BIT - 1; k >= 0; k--)

    {

printf (" %d", q[k]);

}
printf ("\t %d", q_1);

if (q[0] == 1 && q_1 == 0)

    {

add (a, m2);
```

```
printf ("\n ");

for (int k = BIT - 1; k >= 0; k--)

        {

printf (" %d", a[k]);

}
printf ("\t");

for (int k = BIT - 1; k >= 0; k--)

        {

printf (" %d", q[k]);

}
printf ("\t %d\t ADD M", q_1);

}

      else if (q[0] == 0 && q_1 == 1)

    {

add (a, m);

printf ("\n ");

for (int k = BIT - 1; k >= 0; k--)

        {

printf (" %d", a[k]);

}
printf ("\t");

for (int k = BIT - 1; k >= 0; k--)

        {

printf (" %d", q[k]);

}
printf ("\t %d\t SUB M", q_1);

}
```

```
q_1 = ashr (a, q, q_1);

printf ("\n ");

for (int k = BIT - 1; k >= 0; k--)

    {

printf (" %d", a[k]);

}
printf ("\t");

for (int k = BIT - 1; k >= 0; k--)

    {

printf (" %d", q[k]);

}
printf ("\t %d\t ASHR", q_1);


}
printf ("\n\nFINAL RESULT: \n");

printf ("\n The answer in binary form is:");

for (int k = BIT - 1; k >= 0; k--)

    {

printf (" %d", a[k]);

}
for (int k = BIT - 1; k >= 0; k--)

    {

printf (" %d", q[k]);

}
int sign = a[BIT - 1];

if (sign)

    {

for (int k = BIT - 1; k >= 0; k--)
```

```
    {

a[k] = a[k] == 1 ? 0 : 1;

q[k] = q[k] == 1 ? 0 : 1;

}
int one[BIT] = { 0 };

one[0] = 1;

int carry = 0;

carry = add (q, one);

if (carry)

carry = add (a, one);

}

int ans = b_d (a, q);

printf ("\n The answer in decimal form is: ");

if (sign)

printf ("-");

printf ("%d \n", ans);
}
```

- **RESULT: -**

```
Enter the multiplicand: 5
Enter the multiplier: 3

Entered Multiplicand:  0 1 0 1
Entered Multiplier:  0 0 1 1

                A       Q               Q-1     OPERATION
LOOP NO:1  0 0 0 0      0 0 1 1          0
 1 0 1 1        0 0 1 1          0       ADD M
 1 1 0 1        1 0 0 1          1       ASHR
LOOP NO:2  1 1 0 1      1 0 0 1          1
 1 1 1 0        1 1 0 0          1       ASHR
LOOP NO:3  1 1 1 0      1 1 0 0          1
 0 0 1 1        1 1 0 0          1       SUB M
 0 0 0 1        1 1 1 0          0       ASHR
LOOP NO:4  0 0 0 1      1 1 1 0          0
 0 0 0 0        1 1 1 1          0       ASHR

FINAL RESULT:

 The answer in binary form is: 0 0 0 0 1 1 1 1
 The answer in decimal form is: 15
```

- **CONCLUSION: -**

From this experiment we learned to implement Booth Algorithm for Binarymultiplication.

EXPERIMENT NO: 08

Implementation of Restoring division Algorithm for Binary number.

- **DATE OF PERFORMANCE: -**24/11/2021
- **AIM: -** To implement Restoring Division Algorithm for Binary number.
- **SOFTWARE: -** VS code
- **SOURCE CODE: -**

```
#include       <stdio.h>

#include       <conio.h>

#include <math.h>

int a = 0, b = 0, c = 0, com[5] = {1, 0, 0, 0,  0}, s = 0;

int anum[5] = {0}, anumcp[5] = {0}, bnum[5]  = {0};

int acomp[5] = {0}, bcomp[5] = {0}, rem[5] = {0}, quo[5] = {0}, res[5] = {0};void
binary()

{

    a = fabs(a);b =

    fabs(b);

    int r, r2, i, temp; for (i

    = 0; i < 5; i++)

    {

        r = a % 2; a

        = a / 2; r2 =

    b % 2;

    b = b / 2;

    anum[i] = r;

    anumcp[i] = r;
```

```
bnum[i] = r2;
if (r2 == 0)
{
   bcomp[i] = 1;
}
   if (r == 0)
   {
      acomp[i] = 1;
   }
}
//part for two's complementing
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];

}
void add(int num[])
```

```
{
   int i;
   c = 0;
   for (i = 0; i < 5; i++)
   {
      res[i] = rem[i] + num[i] + c;
      if (res[i] >= 2)
      {
         c = 1;
      }
      else
         c = 0;
      res[i] = res[i] % 2;
}
for (i = 4; i >= 0; i--)
{
   rem[i] = res[i];
   printf("%d", rem[i]);
}
printf(":");
for (i = 4; i >= 0; i--)
{
   printf("%d", anumcp[i]);
   }
void shl()
{ //for shift left
   int i;
   for (i = 4; i > 0; i--)
```

```
{ //shift the remainder
   rem[i] = rem[i - 1];
}
rem[0] = anumcp[4];
for (i = 4; i > 0; i--)
{ //shift the remtient
   anumcp[i] = anumcp[i - 1];
}
anumcp[0] = 0;
printf("\nSHIFT LEFT: "); //display together
for (i = 4; i >= 0; i--)
{
   printf("%d", rem[i]);
}
   printf(":");
   for (i = 4; i >= 0; i--)
   {
      printf("%d", anumcp[i]);
   }
}
void main()
{
   int i;
```

```
printf("\t\tRESTORING DIVISION ALGORITHM");
printf("\nEnter two numbers to multiply: ");
printf("\nBoth must be less than 16");
//simulating for two numbers each below 16
do
{
   printf("\nEnter A: ");
   scanf("%d", &a);
   printf("Enter B: ");
   scanf("%d", &b);
} while (a >= 16 || b >= 16);
printf("\nExpected Quotient = %d", a / b);
printf("\nExpected Remainder = %d", a % b);
if (a * b < 0)
{
   s = 1;
}
binary();
printf("\n\nUnsigned Binary 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-->");
//division part
shl();
for (i = 0; i < 5; i++)
{
    printf("\n-->"); //start with subtraction
    printf("\nSUB B: ");
add(bcomp);
if (rem[4] == 1)
{ //simply add for restoring
    printf("\n-->RESTORE");
    printf("\nADD B: ");
    anumcp[0] = 0;
    add(bnum);
}
else
{
    anumcp[0] = 1;
}
if (i < 4)
```

```
        shl();
}
printf("\n.....");
printf("\nSign of the result = %d", s);
printf("\nRemainder is = ");
for (i = 4; i >= 0; i--)
{
   printf("%d", rem[i]);
}
printf("\nQuotient is = ");
for (i = 4; i >= 0; i--)
{
   printf("%d", anumcp[i]);
}
getch();
```

- **RESULT: -**

```
                RESTORING DIVISION ALGORITHM
Enter two numbers to multiply:
Both must be less than 16
Enter A: 5
Enter B: 5

Expected Quotient = 1
Expected Remainder = 0

Unsigned Binary Equivalents are:
A = 00101
B = 00101
B'+ 1 = 11011

-->
SHIFT LEFT: 00000:01010
-->
SUB B: 11011:01010
-->RESTORE
ADD B: 00000:01010
SHIFT LEFT: 00000:10100
-->
SUB B: 11011:10100
-->RESTORE
ADD B: 00000:10100
SHIFT LEFT: 00001:01000
-->
SUB B: 11100:01000
-->RESTORE
ADD B: 00001:01000
SHIFT LEFT: 00010:10000
-->
SUB B: 11101:10000
-->RESTORE
ADD B: 00010:10000
SHIFT LEFT: 00101:00000
-->
SUB B: 00000:00000
---------------------------
Sign of the result = 0
Remainder is = 00000
Quotient is = 00001
```

- **CONCLUSION: -**

From this experiment we have learned to implement the Restoring DivisionAlgorithm for Binary number.

EXPERIMENT NO: 09

Implementation of Non-Restoring division Algorithm for Binary number.

- **DATE OF PERFORMANCE: -**24/11/2021
- **AIM: -** To implement Non-restoring Division Algorithm for Binary number.
- **SOFTWARE: -** VS code.
- **SOURCE CODE: -**

```
#include<stdio.h>
void main() {
int a[10],m[10],q[10],i,j,c=0,b[10],mc[10],z=0,s[10],x[10];
printf("\nEnter Divisor [M] (BINARY FORM): ");
for(i=4;i>=1;i--){
scanf("%d",&m[i]);
mc[i]=!m[i];
a[i]=0, x[i]=0;
}
x[1]=1, x[5]=0, m[5]=0, mc[5]=1;
for(i=1;i<=5;i++){
s[i]=x[i]^mc[i]^z;
c=(x[i]&&mc[i])||(x[i]&&z)||(mc[i]&&z);
z=c;
mc[i]=s[i];
}
printf("\nEnter Divident [Q] (BINARY FORM): ");
for(i=4;i>=1;i--){
scanf("%d",&q[i]);
```

```
}
printf("\n Step \t\t Action Performed \t\t A \t\tQ\n");
printf("\n\n 0\t\t Initialization\t\t 0 0 0 0 0 \t ");
for(i=4;i>=1;i--){
printf(" %d",q[i]);
}
for(j=1;j<=4;j++) {
printf("\n\n %d",j);
printf("\t\t Left Shift \t\t ");
for(i=5;i>=2;i--){
a[i]=a[i-1];
}
a[1]=q[4];
for(i=4;i>=2;i--) {
q[i]=q[i-1];
}
for(i=5;i>=1;i--) {
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=2;i--) {
printf(" %d",q[i]);
}
if(a[5]==0){
z=0;
for(i=1;i<=5;i++) {
```

```
s[i]=a[i]^mc[i]^z;
c=(a[i]&&mc[i])||(a[i]&&z)||(mc[i]&&z); z=c;
a[i]=s[i];
}
if(a[5]==1) {
q[1]=0;
}
else{
q[1]=1;
}
printf("\n\n\t\t a = a-m\t\t ");
for(i=5;i>=1;i--){
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=1;i--){
printf(" %d",q[i]);
}
}
else{
z=0;
for(i=1;i<=5;i++){
s[i]=a[i]^m[i]^z;
c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z); z=c;
a[i]=s[i];
}
```

```
if(a[5]==1){
q[1]=0;
}
else{
q[1]=1;
}
printf("\n\n\t\t a = a+m\t\t ");
for(i=5;i>=1;i--){
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=1;i--){
printf(" %d",q[i]);
}
}
}
if(a[5]==1){
printf("\n\n\n 5");
for(i=1;i<=5;i++){
s[i]=a[i]^m[i]^z;
c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z); z=c;
a[i]=s[i];
}
printf("\t\t a = a+m\t\t ");
for(i=5;i>=1;i--){
printf(" %d",a[i]);
```

```
}
printf("\t ");
for(i=4;i>=1;i--){
printf(" %d",q[i]);
}
}
printf("\nQuotient [Q] :");
for(i=4;i>=1;i--){
printf(" %d",q[i]);
}
printf("\nRemainder [A] :");
for(i=4;i>=1;i--){
printf(" %d",a[i]);
}
}
```

- **RESULT: -**

```
Enter Divisor [M] (BINARY FORM): 1
1
 0              Initialization          0 0 0 0 0       0 1 1 0

 1              Left Shift               0 0 0 0 0      1 1 0

                a = a-m                 1 0 1 0 0       1 1 0 0

 2              Left Shift              0 1 0 0 1       1 0 0

                a = a-m                 1 1 1 0 1       1 0 0 0

 3              Left Shift              1 1 0 1 1       0 0 0

                a = a+m                 0 0 1 1 1       0 0 0 1

 4              Left Shift              0 1 1 1 0       0 0 1

                a = a-m                 0 0 0 1 0       0 0 1 1
Quotient [Q] : 0 0 1 1
Remainder [A] : 0 0 1 0
```

- **CONCLUSION: -**

From this experiment we learned to implement the Non-restoring DivisionAlgorithm for Binary number.

EXPERIMENT NO: 10

IEEE PROGRAM

- **DATE OF PERFORMANCE: -**8/12/2021
- **AIM: -** Conversion of Float number into Binary number.
- **SOFTWARE: -** VS code.
- **SOURCE CODE: -**

```
#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;
    }
```

- **RESULT:**

Output

```
/tmp/j7WRLwmq7n.o
Enter any float number: 123.98
0 10000101 11101111111010111000011
```

- **CONCLUSION:**

From this experiment we learned how to convert Float number into Binary number.

Page no :- (1)

Name :- Singh Sudham Dharmendra

Roll no :- AIML D-50

Branch :- CSE (AI & ML)

Subject :- Digital Logic & Computer Architecture Lab (DLCA)

Topic :- Assignment No :- 1

Date of Submission :- 10/12/2021

(Q.1) Convert Decimal number 123.45 into binary, octal, hexadecimal and Base - 5 system.

→ (i) Decimal to Binary :-

For integer $(123)_{10}$

123 ÷ 2 = 61	remainder	1
61 ÷ 2 = 30	remainder	1
30 ÷ 2 = 15	remainder	0
15 ÷ 2 = 7	remainder	1
7 ÷ 2 = 3	remainder	1
3 ÷ 2 = 1	remainder	1
1 ÷ 2 = 0	remainder	1

For fractional part $(0.45)_{10}$

0.45 X 2 = 0.9	integer	0
0.9 x 2 = 1.8	integer	1
0.8 x 2 = 1.6	integer	1
0.6 X 2 = 1.2	integer	1
0.2 X 2 = 0.4	integer	0
0.4 X 2 = 0.8	integer	0
0.8 X 2 = 1.6	integer	1

$\therefore (123.45)_{10} = (1111011.0111001)$

(ii) Decimal to Octal :-

For integer $(123)_{10}$

123 ÷ 8 = 15	remainder	3
15 ÷ 8 = 1	remainder	7
1 ÷ 8 = 0	remainder	1

$\therefore (123)_{10} = (173)_8$

For Fractional $(0.45)_{10}$

0.45 X 8 = 3.6	integer	3
0.6 X 8 = 4.8	integer	4
0.8 X 8 = 6.4	integer	6
0.4 X 4 = 3.2	integer	3
0.2 X 8 = 1.6	integer	1

$\therefore (0.45)_{10} = (34631)$

$\therefore (123.45)_{10} = (173.34631)_8$

(iii) Decimal to Hexadecimal :-

For integer $(123)_{10}$

123 ÷ 16 = 7	remainder	B
7 ÷ 16 = 0	remainder	7

$\therefore (123)_{10} = (7B)_{16}$

For Fractional $(0.45)_{10}$

0.45 X 16 = 7.2	integer	7
0.2 X 16 = 3.2	integer	3
0.2 X 16 = 3.2	integer	3
0.2 X 16 = 3.2	integer	3

$\therefore (123.45)_{10} = (7B.7333)_{16}$

(iv) Decimal to Base-5 system :-

For integer $(123)_{10}$

$123 \div 5 = 24$	remainder 3
$24 \div 5 = 4$	remainder 4
$4 \div 5 = 0$	remainder 4

$\therefore (123)_{10} = (443)_5$

For ~~decimal~~ fractional part $(0.45)_{10}$

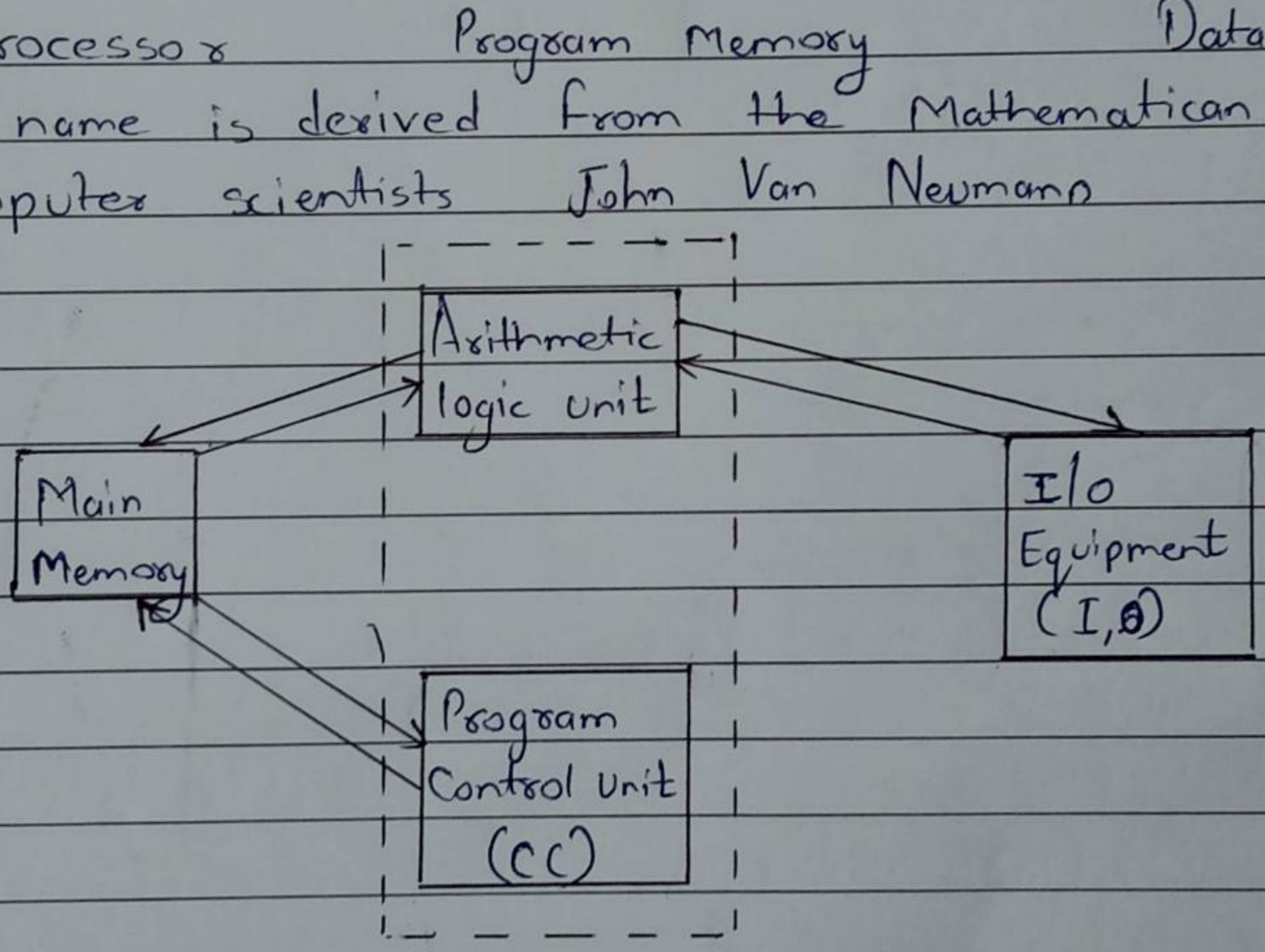
$0.45 \times 5 = 2.25$	integer 2
$0.25 \times 5 = 1.25$	integer 1
$0.25 \times 5 = 1.25$	integer 1
$0.25 \times 5 = 1.25$	integer 1

$\therefore (0.45)_{10} = (0.2111)_5$

$\therefore (123.45)_{10} = (443.2111)_5$

Q.2) Explain the von-neuman architecture in detail.

→



① The name is derived from the Mathematican and early computer scientists John Van Neumann

② The computer has a common memory for as well as code to be executed

③ The processor needs two clock cycles to complete an instruction first to get an instruction and second to get the data.

④ The system has 3-unit CPU, Memory & other devices Key features of a Van Neumann Machine.

(5) The Von Neumann machine uses stored program concept.
(6) The program & data are stored in the same memory unit
(7) Each location of the memory has a unique address
i.e, No two location have the same memory location.

Input unit:- A compiler accepts inputs from the user through input devices.

Output unit:- The result is given back by the compiler to the user through an output devices.

ALU :- Arithmetic or logical operations like Multiplication, addition, division, AND, OR, XOR, etc are performed.

Control Unit:- It controls all the operations of the given system inside and outside the processor.

Memory unit:- Memory is used to store the program and data for the computer.

Q.3] Perform hexadecimal addition of (DADA) & (ABBA).

→ Hexadecimal Addition example 1:- $(DADA)_H + (ABBA)_H$

1st Num	D	A	D	A
2nd Num	A	B	B	A
	1	1	1	
1	$(D+A+1)_H$	$(A+B+1)_H$	$(D+B+1)_H$	$(A+A)_H$
	↓	↓	↓	↓
	$(13+10+1)_{10}$	$(10+11+1)_{10}$	$(13+11+1)_{10}$	$(10+10)_{10}$
	$=(24)_{10}-(16)_{10}$	$=(22)_{10}-(16)_{10}$	$=(25)_{10}-(16)_{10}$	$=(20)_{10}-(16)_{10}$
↓	$=(8)_{10}$	$=(6)_{10}$	$=(9)_{10}$	$=(4)_{10}$
1	= 8	= 6	= 9	= 4

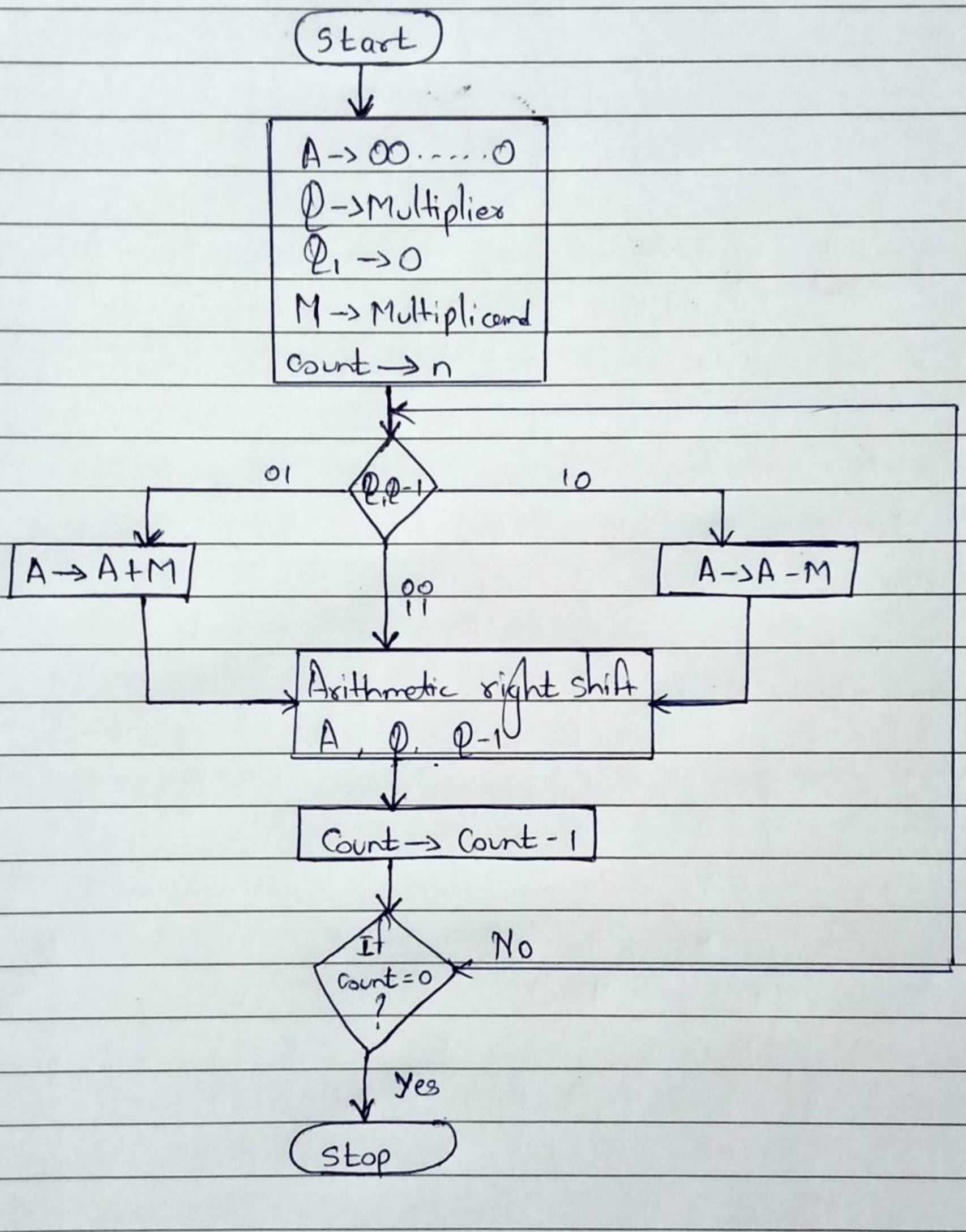
Hence, $(DADA)_H + (ABBA)_H$

$=(18694)_H$

Page no:- (5)

(Q.4) Draw the flowchart for the Booth's multiplication and show all the iteration steps for multiplication of (14) & (-9)

→



(14) x (-9)

(M) -> Multiplicand = $(14)_{10}$ = $(01110)_2$

(Q) -> Multiplier = $(-9)_{10}$ = $(10111)_2$

2's completement of Q = $M^2 + 1$

= 01000 + 1

= 01001

A	Q	Q_{-1}	M	Operation.
00000	10111	0	01110	Intial value
10010	10111	0	01110	A -> A-M
11001	01011	1	01110	shift right
11100	10101	1	01110	shift right
11110	01010	1	01110	shift right
01100	01010	1	01110	A -> A+M
00110	00101	0	01110	shift right
11000	00101	0	01110	A -> A-M
11100	00010	1	01110	shift right

Product (P) = (11100 00010) = is negative

2's complement of P

= 00011 11101

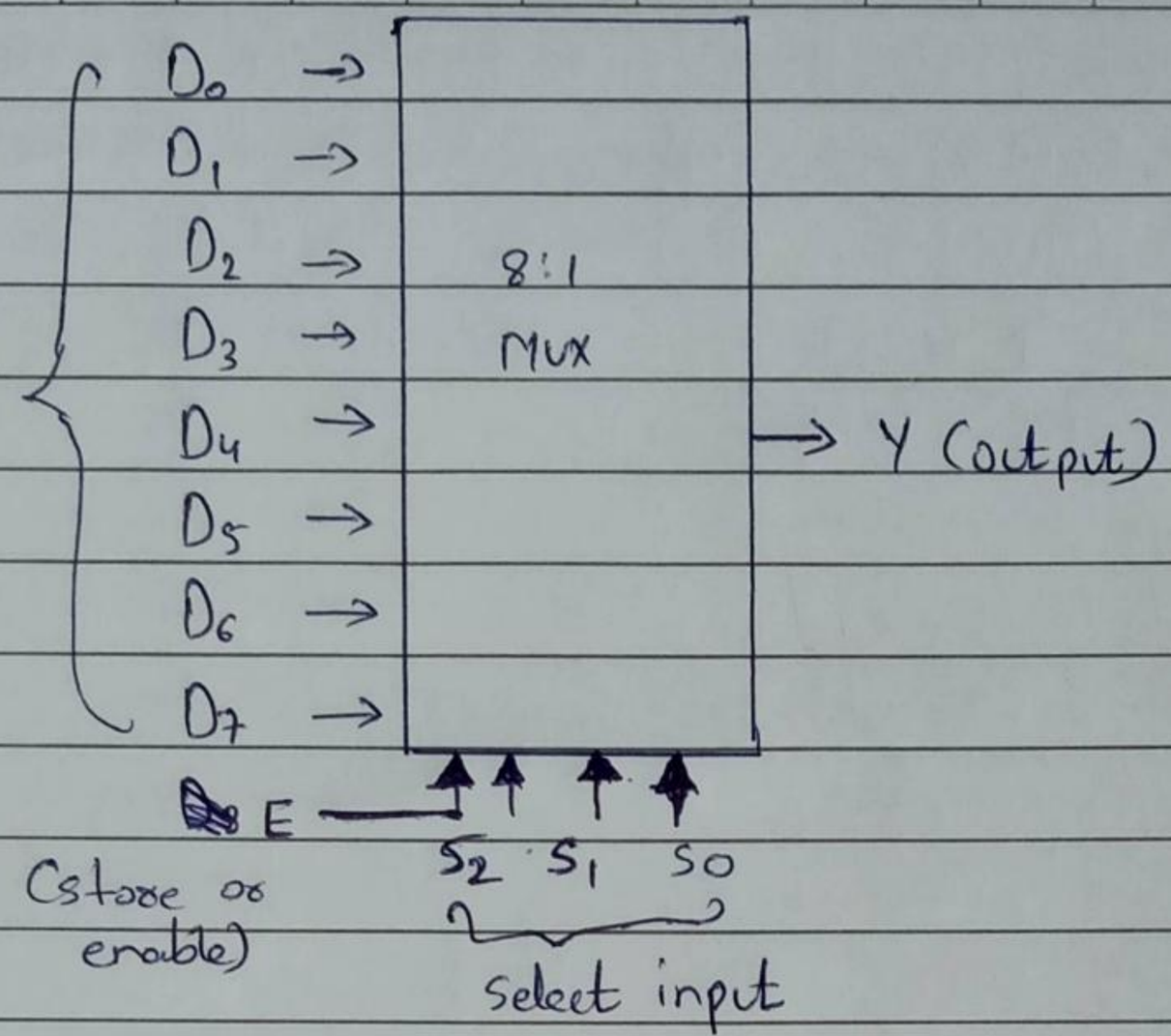
\+ 1

(00011 11110)$_2$

= $(-126)_2$

Q.5) Show all the steps in the design of 8:1 MUX.

→ It has 8 data input, one enable, input, 3 select input and one output.



Block diagram:-

Enable (E)	Select input			Output
	S_2	S_1	S_0	'Y'
0	x	x	x	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

Operator Principle:-

① When the strobe or enable input is 0 the output of the multiplexer will be 0 irrespective of any input.

② With E=1, we can select and one of the eight data input and connect it to the output.

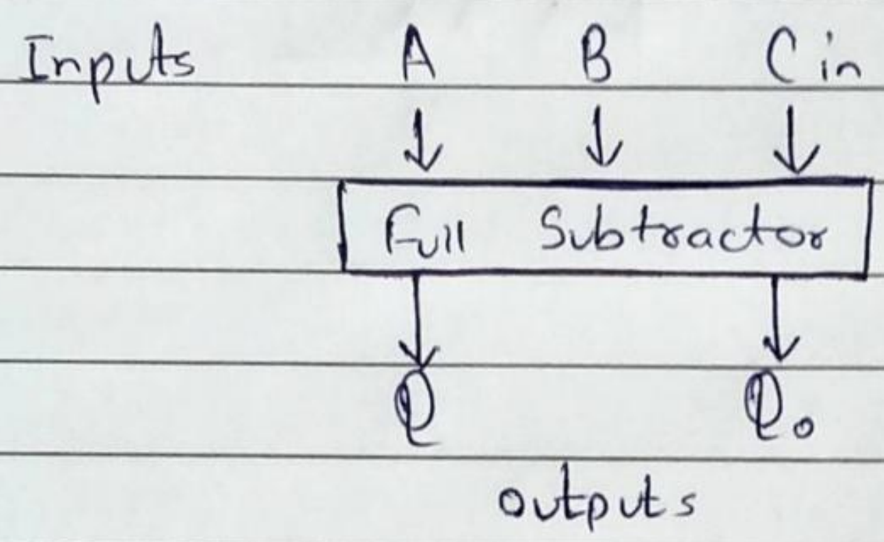
③ For eg if $S_0, S_1, S_2 = 110$, then the data input D_3 is selected and output Y will follows the selected input D_5

Q.6) Design a full subtractor using standard design procedure.

→ ① A full subtractor is a combination circuit with 3 inputs A, B & Cin and two outputs D & D_o

② A is the minvend, B is subtracted, Cin is the borrow produced by the previous stage, D is the difference output and D_o is the borrow output.

③ Show the symbol of a full subtractor. The disadvantage of half subtractor is overcome if we use the full subtractor



Q.7) List & Explain various addressing modes of generic processor.

→ Addressing modes are :-

① ~~Indirect~~ Immediate, Direct, Indirect, Register, Register Indirect, Displacement, Relative, stack.

① Immediate :- In this case the operand is part of instruction The operand is in immediate next location of the op-code

For eg :- ADD AX, 0005H

② Direct Addressing mode :- In this case the address field contains memory address of the operand

for eg :- Intel 8085 instruction : LDA 1000

Loads 8 bit data from address $(1000)_H$ to accumulator.

③ Indirect addressing Mode:- In this case memory location has the address of the operand.

eg:- ADD AX, [[1000]]

-Disadvantage is that it is slower.

④ Register addressing mode:- In this case the operand is held in the register named in operand address field.

Advantage is that there is no memory access.

eg:- ADD AX, BX

⑤ Register addressing mode:- In this case the operand memory address in pointed to be content of register R.

It requires one less memory access then indirect addressing mode.

⑥ Displacement addressing mode:- In this type of addressing mode, there are two address fields that holds the base address and the displacement.

⑦ Relative Addressing mode:- It is a version of displacement addressing where the register is the program counter, PC

It is used for Branching or transfer of the instructor

⑧ Stack Addressing mode:- Used to access the data from the top of the stack.

Used to PUSH & POP instruction to access the stack.

Page no:- (1)

Name :- Singh Sudham Dharmendra

Roll no:- AIML D-50

Branch :- CSE (AI & ML)

Subject :- Digital Logic & Computer Architecture Lab (DLCA)

Topic :- Assignment No:- 2

Date of Submission :- 10/12/2021

Q.1) Enlist the properties of Memory systems and explain each in Brief:

→ ① Location & Capacity of memory system :-

Location	Size	Usage	Access time
CPU register	Few kB register	stores data	1 cycle
L1 cache	Few kB	Stores locality of program or data	1 to 3 cycles
L2 cache	Larger than L1	stores locality of Program or data	1 to 3 cycles
Main Memory	Larger than L2	Stores program and data	5 cycles or more
Secondary memory	GB or more	stores all program	milliseconds

② Capacity memory :-
Define the size of memory
Example - memory ship capacity = 8k x 8 bit
word size = 8 bits
Number of words = 8k

③ Performance :-
(i) Access time - Time elapsed to access memory location after giving address. It is from nanoseconds to milliseconds
(ii) Memory cycle time - Memory cycle time = Access time + Additional time
Sum of memory access time Plus additional time required before starting the next access.
(iii) Memory bandwidth :- Defines the "data transfer rate" Memory bandwidth is reciprocal of memory cycle

(4) Physical Characteristics:-

(i) Volatile / Nonvolatile

Volatile - Memory contents are lost after a power off eg:- Semiconductor RAM.

Nonvolatile - Memory contents are unchanged even after a power off eg:- ROM, megnetic, optical memories.

(ii) Erasable / Non erasable memory:-

Erasable - Memory contents are altered by user eg: RAM, EPROM

Non-erasable - eg. ROM, PROM

(iii) Destructive Read-out (DRO):-

Properties of DRAM where the contents are destructed after energy read operation.

(Q.2) Explain cache mapping methods. Performs and show the cache mapping using all methods with following specifications: 32 kBytes cache memory, 32 MBytes of Main memory & size of cache line 16 Bytes.

→ Mapping defines Cache & line, where the MM block is to be copied.

Various mapping method:-

1. Direct Mapping -

Cache Memory size = 32kB = 2^{15} bytes.

Main memory Size = 32MB = 2^{25} bytes.

Cache line ~~size~~ = 16 bytes. = M block size.

In direct mapping, cache has one tank of 32kB

size of mm address = 25 bits

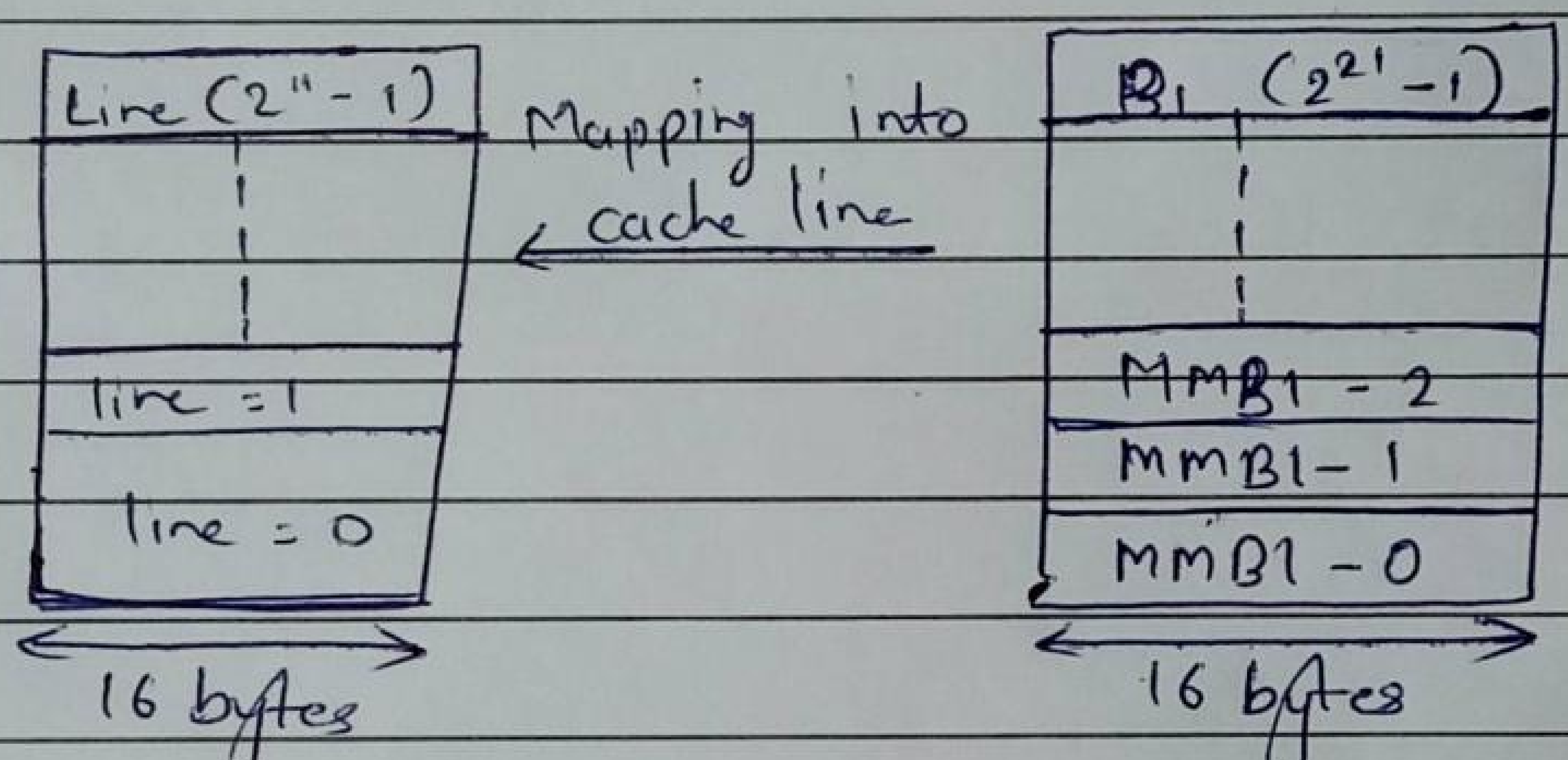
no. of cache lines = $2^{15}/16 = 2^{11}$

no. of mm blocks = $2^{25}/16 = 2^{21}$

Page no:- (4)

Mapping function:-

In coming mm block in mapped in one cache lines
CL = (Incoming mm block) Mod (No. of cache lines)
MM address of bits is used as:-
LSB 2 bits define byte within cache line
Middle 11 bits identifies one particular line out of 2^{11} lines
MSB 11 bits are directory bits.



2. Way of set associative mapping:-

In two way set associative mapping, two tank
Bank 'A' = Bank 'B' = 16 KB = 2^{14} bytes.
no. of cache lines per bank = $2^{14}/16 = 2^{10}$ bytes
no. of mm blocks = $2^{25}/16 = 2^{21}$ bytes
Incoming MM block is mapped into one particular cache line of either Bank 'A' or Bank 'B'.
LSB 2 bits defines bytes when within cache line
Middle 10 bits ~~definiti~~ identifies one particular line out of 2^{10} lines MSB 13 bits are tag (directory) bits

③ Full Associative mapping:-

Cache memory has single block of 32 kB

no. of cache lines = $2^{15}/16 = 2^{11}$ bytes.

no. of mm blocks = $2^{25}/16 = 2^{21}$ bytes

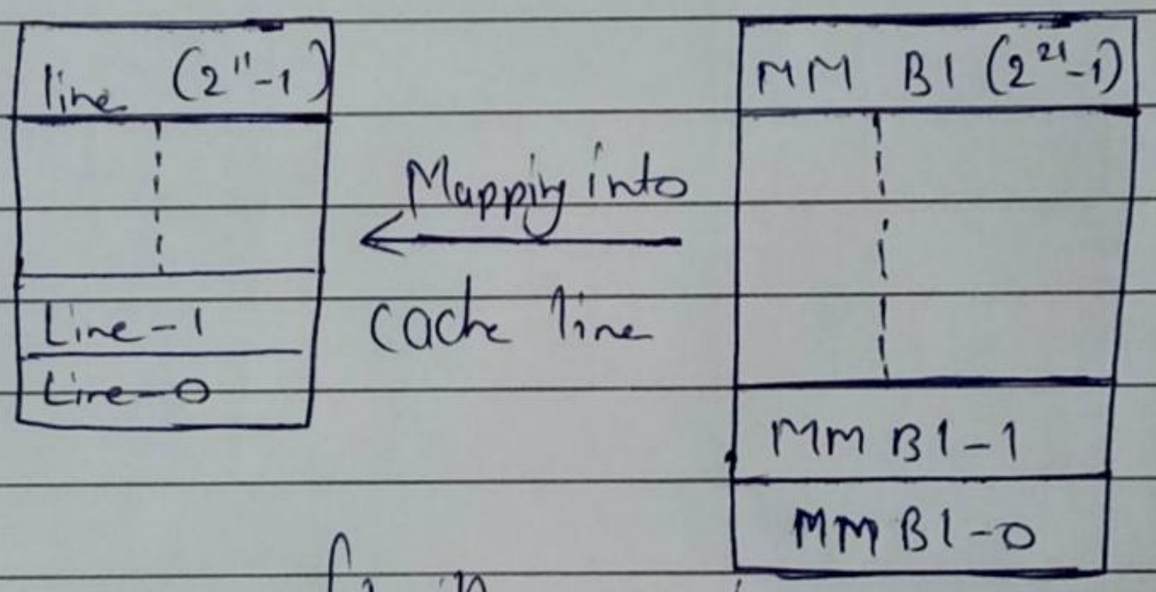
~~no. of mm blocks~~ =

In coming MM block is assigned into any available cache line

main memory 25 bits:

2 bits define byte within cache line.

23 bits are tag bits for the cache line.

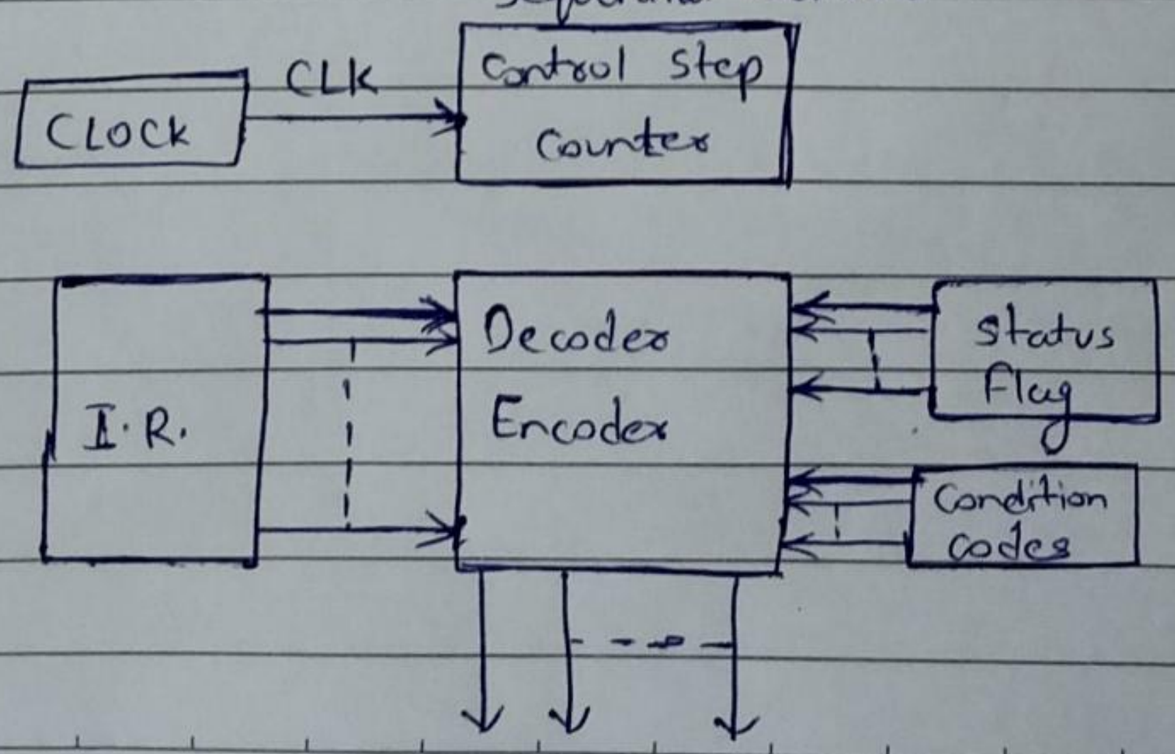


Advantage - more flexible mapping.

Drawback - more memory required for tag/directory

(Q.3) Explain in details the concept of Hardwired control unit design.

→



Block diagram of Hardwired control unit design:

① A hardwire control is a mechanism of producing signals using finite state Machines (FSM) appropriately.

② It is designed as a sequential logic circuit.

③ The final circuit is constructed by physically connecting the components such as gates flip flops, & drums. Hence, it is named a hardwired controller.

④ The figure shows a 2-bit sequence cantex, which is used to develop control signals.

⑤ The output obtained from these signals is decoded to generate the required signals in sequential order.

⑥ The hardwired control consists of a combinational circuit that outputs desired controls for decoding & encoding functions. The instruction that is loaded in the IR is decoded by the instruction decoder. If the IR is an 8-bit register then the instruction decoder generates 2^8 (256) lines.

⑦ Inputs to the encoder are given from the instruction step decoder, external inputs, and condition codes. All these inputs are used and individuals control signals are generated. The end signal is generated after all the instructions get executed. Furthermore it results in the resetting of the control step counter, making it ready to generate the control step for the next instruction.

⑧ The major goal of implementing the hardwired control is to minimize the cost of the circuit and to achieve greater efficiency in the operation speed.

Q.4) Classify the computer system as per the flynn's classification scheme and explain the each in brief.

→ M.I. Flynn proposed a classification for the organization of a computer system by the number of instruction and data items that are manipulated simultaneously.

Flynn's Classification divides computer into 4 major groups:-

① Single instruction stream, Single data stream (SISD):-

It represents the organization of a single computer containing a control unit, a processor unit and a memory unit.

Instructions are executed sequentially, and the system may or may not internal parallel processing capabilities.

Most conventional computers have SISD architecture parallel processing may be achieved by means of multiple functional units or by pipeline processing.

② Single instruction and multiple data stream (SIMD)

All programs receives the same instruction from the control unit that operate on different items of data

SIMD is mainly dedicated to energy processing machines.

③ Multiple instruction & single data stream (MISD)

In MISD, multiple processing units operate on one single data stream. Each processing unit operates on the data independently via, separate instruction stream.

④ Multiple Instruction & Multiple Data stream (MIMD):-

In MIMD, each processor has a separate program and instruction stream is generated from and each program

eg:- IBM-SP2.

Q. 5) Enlist and explain various types of pipeline hazards.

→ Instruction hazards over when instruction read or write registers that are used by either instructions.

(i) Structural Hazard :-
These hazards occur when processor hardware is not capable of executing all the instruction in pipline simultaneasly. This can be resolved by using seperate instruction and data memories.

(ii) Data Hazards :- This hazards arises when the result of a previous instructions, but this result is not yet available

(iii) RAW Hazard :- Read after write hazard access when two instructions read from same registes.

(iv) RAR Hazard :- This hazard occurs when an instruction reads a register that was written by previous instruction.

(v) WAR Hazard :- This hazard occurs when the out put register of an instruction has been ~~written~~ either of read of written by previous instruction.

(vi) Branch Hazard :- Branch instructions, particularly conditional branch instructions create data dependencies between the branch instruction and the previous instruction fetch stage of the pipeline.