COMPUTER ARCHITECTURE

**EX.1**

**8 BIT ADDITION**

```
LDA 2050
MOV B,A
LDA 2052
ADD B
STA 2054
HLT
```

EX. 2

8 BIT SUBTRACTION

```
LDA 2050
MOV B,A
LDA 2052
SUB B
STA 2054
HLT
```

EX. 3

8 BIT MULTIPLICATION

```
        MVI D,00
        MVI A,00
        LXI H,4150
        MOV B.,M
        INX H
        MOV C,M
LOOP: ADD B
    JNC NEXT
    INR D
NEXT: DCR C
    JNZ LOOP
    STA 4152
    MOV A,D
```

```
    STA 4153
    HLT
```

EX. 4

8 BIT DIVISION

```
LXI H,1100
MOV B,M
MVI C,00
INX H
MOV A,M
LOOP: CMP B
JC SKIP
SUB B
INR C
JMP LOOP
SKIP: STA 1102
MOV A,C
STA 1103
HLT
```

EX. 5

16 BIT ADDITION

```
LHLD 2500
XCHG
LHLD 2502
MOV A,E
ADD L
MOV L,A
MOV A,D
ADC H
MOV H,A
SHLD 2504
HLT
```

EX.6

16 BIT SUBTRACTION

```
LHLD 2500
XCHG
LHLD 2502
MOV A,E
SUB L
MOV L,A
MOV A,D
ADC H
MOV H,A
SHLD 2504
HLT
```

EX.7

16 BIT MULTIPLICATION

```
LHLD 2050
SPHL
LHLD 2052
XCHG
LXI H,0000H
LXI B,0000H
DAD SP
JNC 2013
INX B
```

DCX D

MOV A,E

ORA D

JNZ 200E

SHLD 2054

MOV L,C

MOV H,B

SHLD 2056

HLT

EX.8

16 BIT DIVISION

| |
|---|
| LXI B,0000H |
| LHLD FC02H |
| XCHG |
| LHLD FC00H |
| MOV A,L |
| SUB E |
| MOV L,A |
| MOV A,H |
| SBB D |
| MOV H,A |
| JC SKIP |
| INX B |

| |
|---|
| JMP LOOP |
| DAD D |
| SHLD FC06H |
| MOV L,C |
| MOV H,B |
| SHLD FC04H |
| HLT |

HALF ADDER

**EXP.NO: 9**

**AIM:**

To design and implement the two bit half adder using Logisim simulator.

**TRUTH TABLE:-**

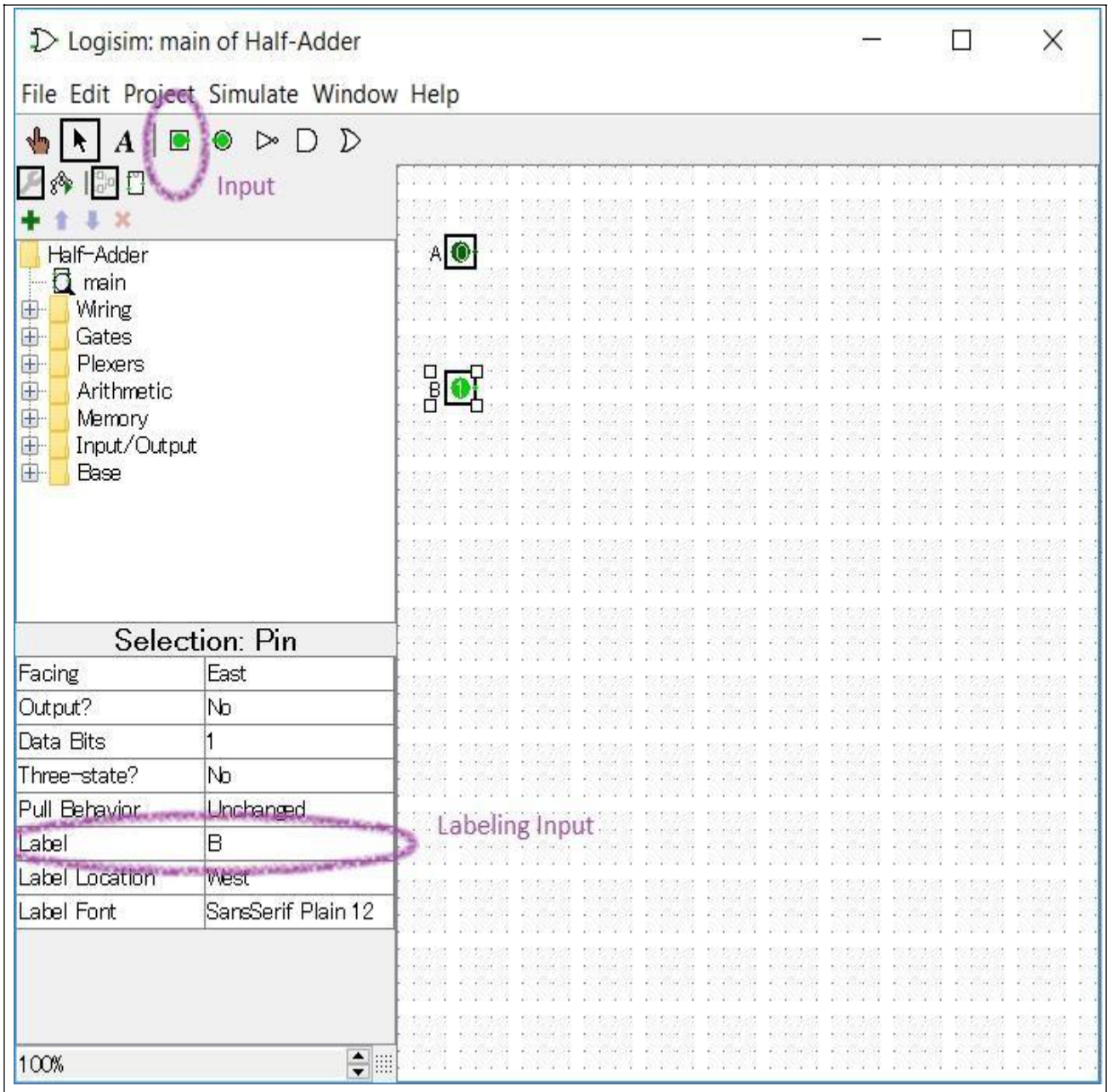
| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

S = A XOR B . C = A AND B .

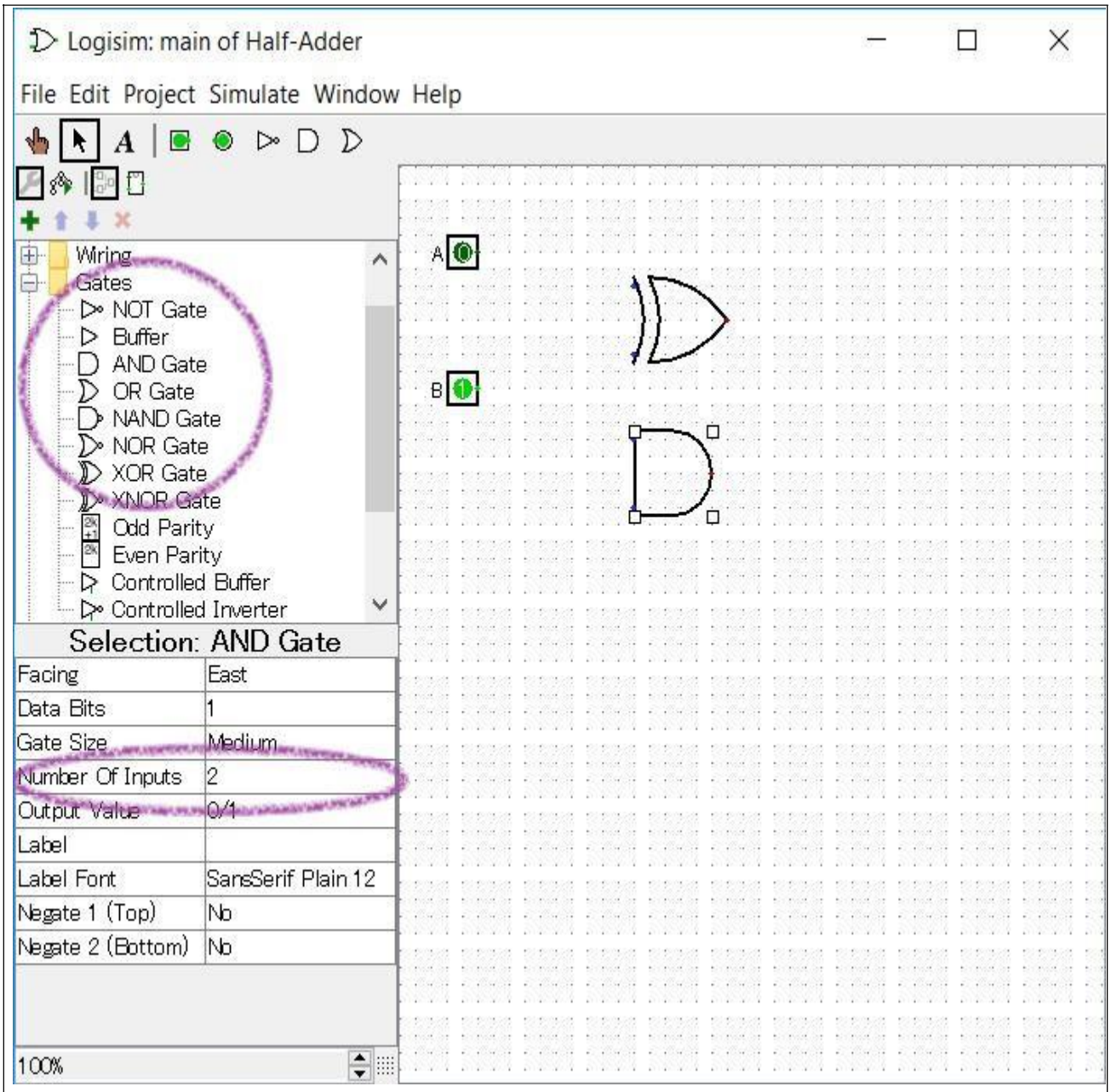
THE FOLLOWING STEPS IS USED TO DRAW A HALF-ADDER CIRCUIT.

- Insert 2 inputs into the canvas.
    - Label the inputs (A & B) by setting the attribute 'Label' in the attribute table.

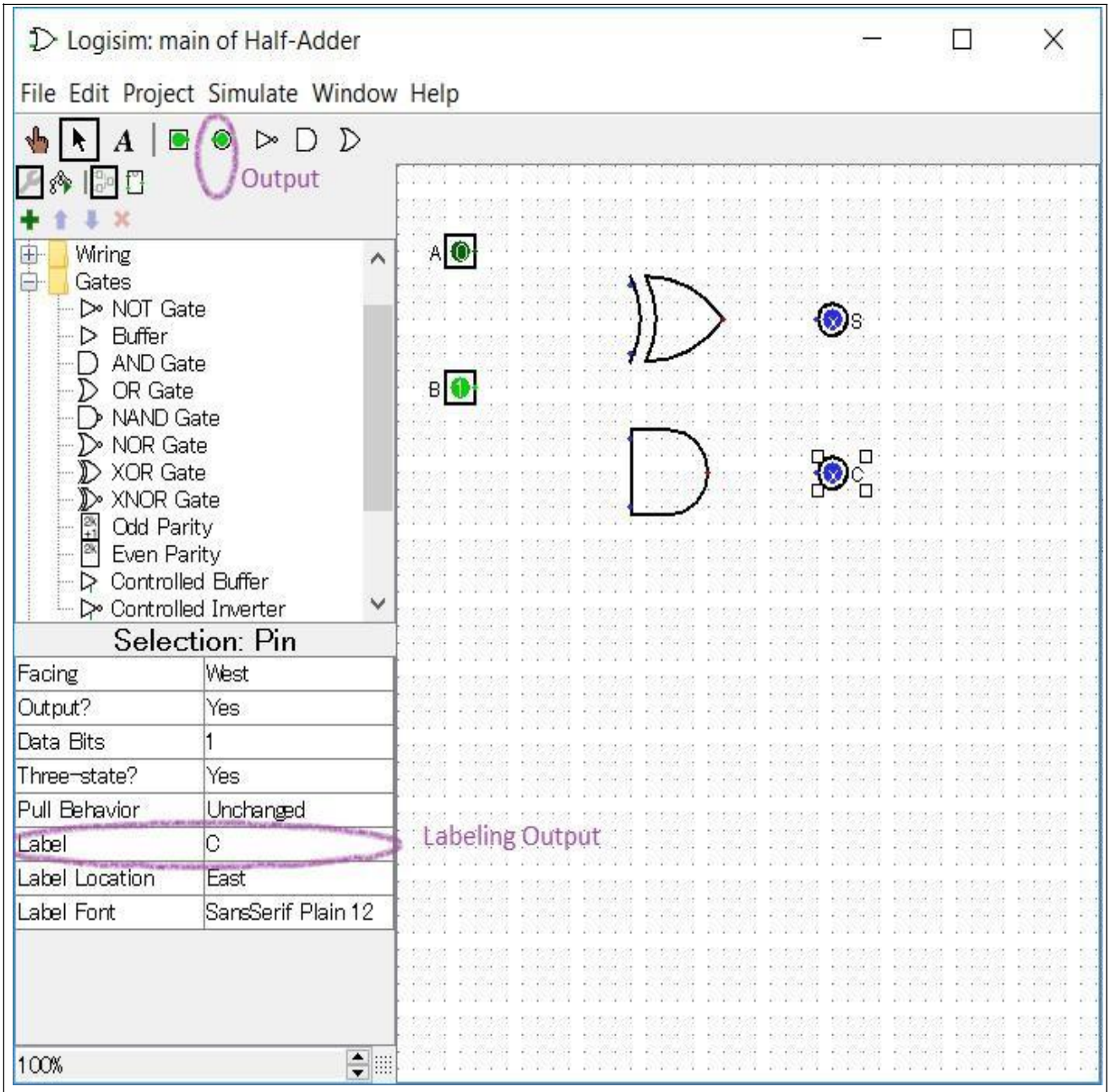
Note that both inputs have now 0s inside their green spots. These are the current bit value of the input.



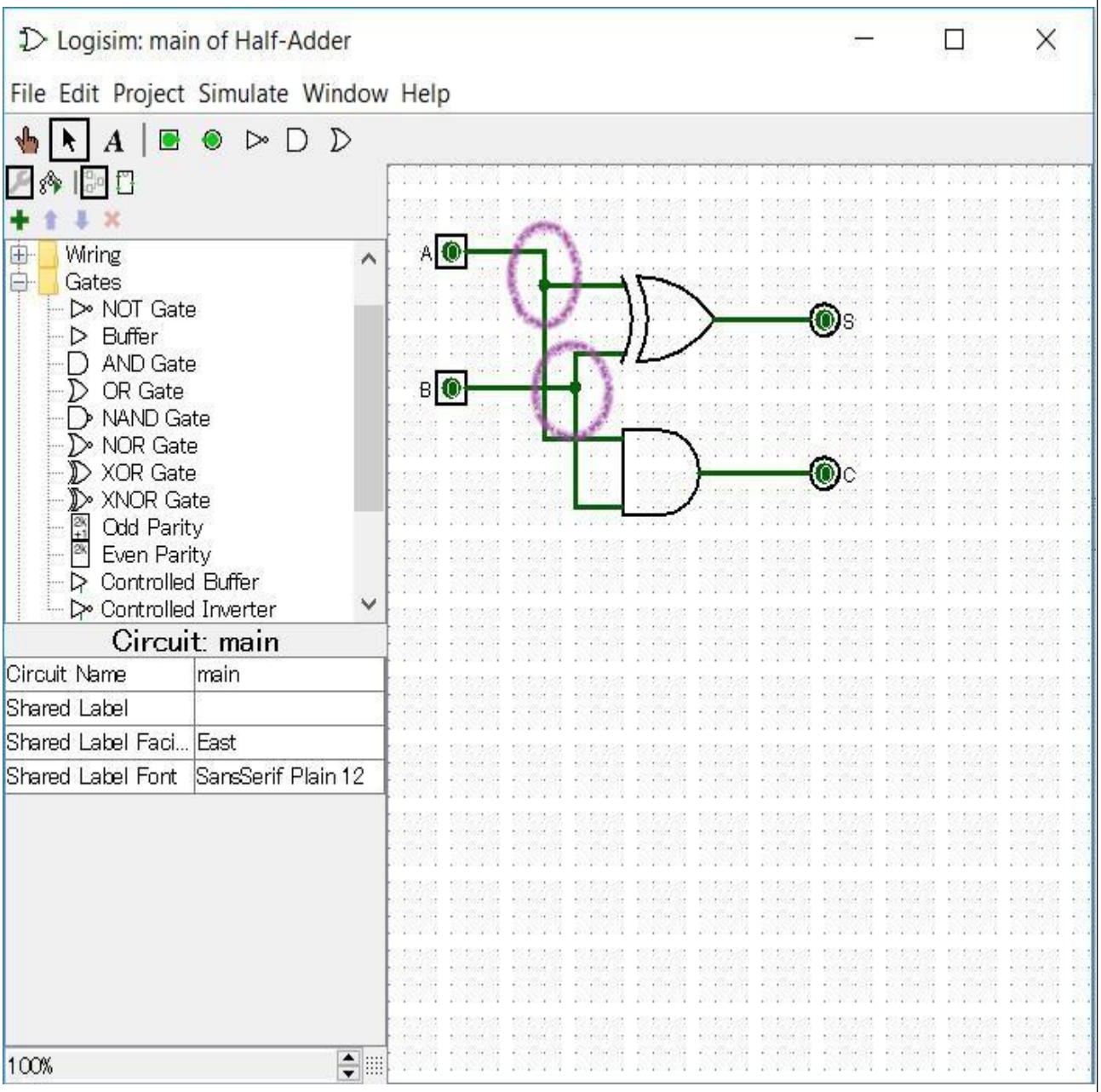

- Insert one XOR gate and one AND gate into the canvas.
    - The two gates are located inside the 'Gates' library in the explorer pane.
    - Change the 'Number of Inputs' in the attribute table to 2.




- Insert 2 outputs in the canvas
    - Label the outputs (S & C).
    - Note that both outputs have X inside the dots. X indicates an invalid value for the output.




- Connect the inputs to the XOR gate.
- Connect the inputs to the AND gate.
    - Start the connection on the AND gate input and finish it on the wire reaching the designated input.
- ~~Connect the outputs to the gates.~~




**RESULT:-**

Hence the designing of the 2-bit half adder using logisim simulator has been implemented successfully.

## 32-BIT FULL ADDER USING LOGISIM SIMULATOR

**EXP.NO: 10**

**AIM:**

To design and implement 32-bit full adder by using logisim simulator.

**TRUTH TABLE:-**

| Row | Inputs | | | Outputs | | Comment |
|---|---|---|---|---|---|---|
| | x | y | $c_{in}$ | $c_{out}$ | s | |
| 0 | 0 | 0 | 0 | 0 | 0 | $0+0+0=00_2$ |
| 1 | 0 | 0 | 1 | 0 | 1 | $0+0+1=01_2$ |
| 2 | 0 | 1 | 0 | 0 | 1 | $0+1+0=01_2$ |
| 3 | 0 | 1 | 1 | 1 | 0 | $0+1+1=10_2$ |
| 4 | 1 | 0 | 0 | 0 | 1 | $1+0+0=01_2$ |
| 5 | 1 | 0 | 1 | 1 | 0 | $1+0+1=10_2$ |
| 6 | 1 | 1 | 0 | 1 | 0 | $1+1+0=10_2$ |
| 7 | 1 | 1 | 1 | 1 | 1 | $1+1+1=11_2$ |

**STEPS TO BE FOLLOWED:-**

**STEP-0:-**

Make a truth table with input coloumns X,Y and cin,in one column give the result of X XOR Y.In another column,give the result for(x XOR y)XOR cin,you should now see that :

- S = x XOR y XOR Cin.

**STEP-1:-**

The sum-of-products equation for the carry output (Cout) is:Cout = $x'\cdot y\cdot C_{in} + x\cdot y'\cdot C_{in} + x\cdot y\cdot C_{in} + x\cdot y\cdot C_{in}'$, is not a minimal expression. Show step by step how you can reduce the expression for Cout to end up with:

- Cout = Cin·(x XOR y) + x·y.

**STEP-2:-**

It's now time to implement your 1-bit full adder in Logisim.

- Start Logisim. On the department Unix System, type logisim in a shell and press enter. If you work on a laptop or form home, downlad and install Logisim from here.

Open up add.circ in Logisim. Start by double-click on add1 to select the add1 circuit.

**STEP-3:-**

Complete the add8 circuit by combining eight 1-bit adders.

- Add three splitters to the circuit. Each splitter should have an input bit width of 8 and a fan out of 8. Attach two east-facing splitters to the 8-bit inputs A and B. Attach a west-facing splitter to the 8-bit output S.
- Create eight instances of the add1 circuit.
- connect the S outputs of the eight add1 instances to the splitter for the 8-bit S output.
- Connect the carry inputs and outputs of the eight add1 instances so that carries will propagate appropriately from the Ci input, through the 1-bit adders, to the Co output.
- Connect the A inputs of the eight add1 instances to the splitter for the Ainput.
- Connect the B inputs of the eight add1 instances to the splitter for the Binput.

Change the values of the $C_i$, A, and B inputs and observe the $C_o$ and S outputs to verify the correct operation of the circuit.

**STEP-4:-**

Complete the add32 circuit by combining four 8-bit adders.

- You will find three splitters in the circuit. Each splitter has an input bit width of 32 and a fan out of 4. Thus, each connection to a splitter represents 8 bits.
- Create four instances of the add8 circuit.
- Connect the 8-bit S outputs of the four add8 instances to the splitter for the 32-bit S output.
- connect the carry inputs and outputs of the four add8 instances so that carries will propagate appropriately from the Ci input, through the 8-bit adders, to the Co output.
- Connect the 8-bit A inputs of the four add8 instances to the splitter for the Ainput.
- Connect the 8-bit B inputs of the four add8 instances to the splitter for the Binput.

**STEP-5:-**

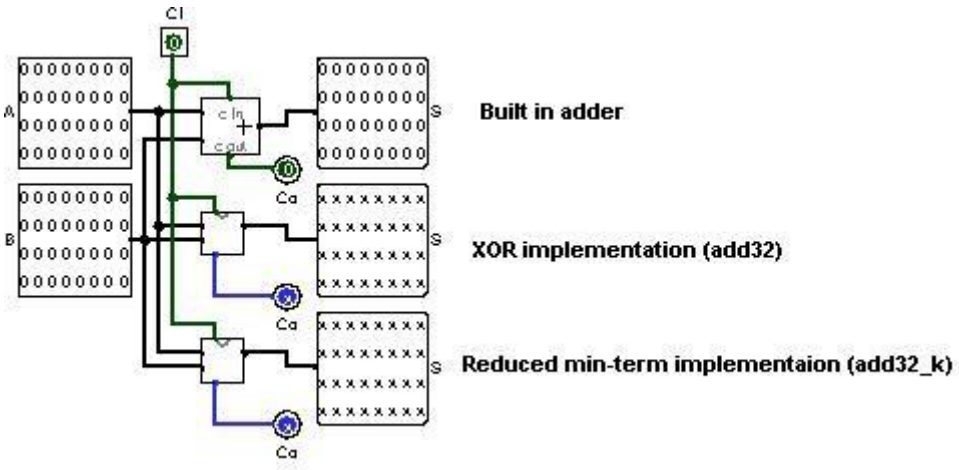
- Within the main circuit, you will find a 32-bit adder connected side-by-side with your add 32 circuit.
- Change the values of the $C_i$, A, and B inputs and observe the Co and S outputs to verify the correct operation of your add32 circuit.

**STEP-6:-**

- There is more than one way to implemt a logical function. An alternative expression for $C_{out}$ can be found by reducing the expression using only min-terms.

- Use a Karnaugh map to reduce the expression for Cout.Note: using Karnaugh map to reduce the expression for the Sum will no be possible, it will result in the original sum-of-products for the Sum.
- Add a new circuit to the project named add1_k and implement a new version of a 1 bit full adder using the new expression for Cout and the original sum-of-producs exreppsions for the SumSimlary, add new circuits named add8_k and add32_k to construct an alternate version of the 32 bit full adder.
- Add the add32_k component to the main circuit along with the other adders and verify that they give the same results.

**CIRCUIT DIAGR**




**RESULT:-**

Thus the designing of the 32-bit full adder using logisim simulator has been implemented successfully.

## EX.12 FACTORIAL OF A GIVEN NUMBER USING 8085 MICROPROCESSOR

**PROGRAM:**

| ADDRESS | MNEMONICS | COMMENTS |
|---|---|---|
| 8100 | LXI H,8500H | Load data from memory |
| 8103 | MOV B,H | Load data to B register |
| 8104 | MVI D,01H | Set D register with 1 |
| 8106 | CALL MULTIPLY | Subroutine call for multiplication |
| 8109 | DCR B | Decrement B |
| 810A | JNZ FACTORIAL | Call factorial till B becomes 0 |
| 810D | INX H | Increment memory |
| 810E | MOV M,D | Store result in memory |
| 810F | HLT | Halt |
| 8200 | MOV E,B | Transfer contents of B to C |
| 8201 | MVI A,00H | Clear accumulator to store result |
| 8203 | ADD D | Add contents of D to A |
| 8204 | DCR E | Decrement E |
| 8205 | JNZ MULTIPLYLOOP | Repeated addition |
| 8208 | MOV D,A | Transfer contents of A to D |
| 8209 | RET | Return from subroutine |

**INPUT:**

8500 05H

**OUTPUT:**

8501 78H

## LARGEST NUMBER IN AN ARRAY

**EXP NO: 13**

**AIM: To find the largest number from an array using 8086 processor.**

**PROGRAM:**

| ADDRESS | MNEMONICS | COMMENTS |
|---|---|---|
| 1100 | LXI H, 1200H | Point to get array size |
| 1103 | MOV C, M | Get the size of array |
| 1104 | INX H | Point to actual array |
| 1105 | MOV B, M | Load the first number into B |
| 1106 | DCR C | Decrease C |
| 1107 | LOOP: INX H | Point to next location |

| | | |
|---|---|---|
| 1108 | MOV A, M | Get the next number from memory to Acc |
| 1109 | CMP B | Compare Acc and B |
| 110A | JC SKIP | if B > A,then skip |
| 110D | MOV B, A | If CY is 0, update B |
| 110E | SKIP: DCR C | Decrease C |
| 110F | JNZ LOOP | When count is not 0, go to LOOP |
| 1112 | LXI H,1300H | Point to destination address |
| 1115 | MOV M, B | Store the MAXIMUM number |
| 1116 | HLT | Terminate the program |

**INPUT:**

**1200 H , 05 H, 09 H, 01 H, 02 H, 07 H, 03 H**

**OUTPUT:**

**1300 H 09 H**

EX.NO.14

2 stage pipeline for addition and subtraction of two numbers using any high level language.

```
# 2 stage pipeline in python
counter=1
a=int(input("ENTER NUMBER-1-"))
counter=counter+1
b=int(input("ENTER NUMBER-2-"))
counter=counter+1
print("1-ADDITION 2-SUBTRACTION 3-MULTIPLICATION 4-DIVISION")
print("Enter Your Choice")
choice=int(input())
if choice==1:
 print("Performing Addition...")
```

```
 res=a+b
 counter=counter+1
if choice==2:
 print("Performing Subtraction...")
 res=a-b
 counter=counter+1
if choice==3:
 print("Performing Multiplication")
 res=a*b
 counter=counter+1
if choice==4:
 if b==0:
 print("Denominator can't be Zero")
 print("Performing Division")
 res=a/b
 counter=counter+1
if choice>=5:
 print("Enter Correct Input")
print(res)
counter=counter+1
print("CYCLE VALUE IS:",counter)
ins=int(input("Enter the No.of instructions:"))
performance_measure =ins/counter
print(“performance measure is:” performance_measure)
```

EX.15

```
# 3 stage AND operation
counter=1
a=int(input("ENTER NUMBER-1-"))
counter=counter+1
b=int(input("ENTER NUMBER-2-"))
counter=counter+1
res= a and b
counter=counter+1
```

```
print(res)
counter=counter+2
INS=int(input("enter no. of instructions:"))
performance_measure=INS/counter
print("performance measure:",performance_measure)
```

EX.NO. 16

```
# 4 stage pipeline
counter=1
a=int(input("ENTER NUMBER-1-"))
counter=counter+1
b=int(input("ENTER NUMBER-2-"))
counter=counter+1
print("1-ADDITION 2-SUBTRACTION 3-MULTIPLICATION 4-DIVISION")
print("Enter Your Choice")
choice=int(input())
if choice==1:
 print("Performing Addition...")
 res=a+b
 counter=counter+1
if choice==2:
 print("Performing Subtraction...")
 res=a-b
 counter=counter+1
if choice==3:
 print("Performing Multiplication")
 res=a*b
 counter=counter+1
if choice==4:
 if b==0:
 print("Denominator can't be Zero")
 print("Performing Division")
 res=a/b
 counter=counter+1
```

```
if choice>=5:
 print("Enter Correct Input")
print(res)
counter=counter+3
print("CYCLE VALUE IS:",counter)
ins=int(input("Enter the No.of instructions:"))
performance_measure =ins/counter
print("performance measure is:",performance_measure)
```

EX.17

BOOTH ALGORITHM

```
#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;
        }
  }
//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];
}
//in case of negative inputs
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(){//for arithmetic shift right
    int temp = pro[4], temp2 = pro[0], i;
    for (i = 1; i < 5  ; i++){//shift the MSB of product
       pro[i-1] = pro[i];
    }
    pro[4] = temp;
    for (i = 1; i < 5  ; i++){//shift the LSB of product
        anumcp[i-1] = anumcp[i];
    }
    anumcp[4] = temp2;
    printf("\nAR-SHIFT: ");//display together
    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("\t\tBOOTH'S MULTIPLICATION 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 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){//just shift for 00 or 11
          printf("\n-->");
          arshift();
          q = anum[i];
      }
      else if(anum[i] == 1 && q == 0){//subtract and shift for 10
         printf("\n-->");
         printf("\nSUB B: ");
         add(bcomp);//add two's complement to implement subtraction
         arshift();
         q = anum[i];
      }
      else{//add ans shift for 01
         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]);
    }
}
```

EX. 18

INTEGER RESTORATION DIVISON METHOD:

```
#include<stdlib.h>
#include<stdio.h>
int acum[100]={0}     ;
void add(int acum[],int b[],int n);
int q[100],b[100];
int main()
{
int x,y;
printf("Enter the Number :");
scanf("%d%d",&x,&y);
int i=0;
while(x>0||y>0)
{
if(x>0)
{
q[i]=x%2;
x=x/2;
}
else
{
```

```
q[i]=0;
}
if(y>0)
{
b[i]=y%2;
y=y/2;
}
else
{
b[i]=0;
}
i++;
}

int n=i;
int bc[50];
printf("\n");
for(i=0;i<n;i++)
{
if(b[i]==0)
{
bc[i]=1;
}
else
{
bc[i]=0;
}
}
bc[n]=1;
for(i=0;i<=n;i++)
{
if(bc[i]==0)
{
bc[i]=1;
```

```
i=n+2;
}
else
{
bc[i]=0;
}
}
int l;
 b[n]=0;
int k=n;
int n1=n+n-1;
int j,mi=n-1;
for(i=n;i!=0;i--)
{
for(j=n;j>0;j--)
{
acum[j]=acum[j-1];

}
acum[0]=q[n-1];
for(j=n-1;j>0;j--)
{
q[j]=q[j-1];
}

add(acum,bc,n+1);
if(acum[n]==1)
{
q[0]=0;
add(acum,b,n+1);
}
else
{
q[0]=1;
```

```
}
}
printf("\nQuoient   : ");

for(  l=n-1;l>=0;l--)
{
printf("%d",q[l]);

}
printf("\nRemainder : ");
for( l=n;l>=0;l--)
{
printf("%d",acum[l]);
}
return 0;
}
void add(int acum[],int bo[],int n)
{
int i=0,temp=0,sum=0;
for(i=0;i<n;i++)
{
sum=0;
sum=acum[i]+bo[i]+temp;
if(sum==0)
{
acum[i]=0;
temp=0;
}
else if (sum==2)
{
acum[i]=0;
temp=1;
}
else if(sum==1)
```

```
{
acum[i]=1;
temp=0;
}
else if(sum==3)
{
acum[i]=1;
temp=1;
}
}
}
```

INTEGER RESTORATION DIVISION

```
Enter the Number :8
3
Quoient   : 0010
Remainder : 00010
```

EX. 19 CACHE HIT RATIO AND CACHE MISS RATIO

```
#include <stdio.h>

int main() {
    float cachehit, cachemiss;
    float cachehitratio;
    printf("\n enter the total number of cache hits:");
    scanf("%d",&cachehit);
     printf("\n enter the number of cache misses:");
    scanf("%d",&cachemiss);
    cachehitratio=cachehit/(cachehit+cachemiss);
    printf("\n Cache Hit Ratio: %f",cachehitratio);
     printf("\n Cache Miss Ratio: %f",1-cachehitratio);
    return 0;
```

}

enter the total number of cache hits:43

enter the number of cache misses:11

Cache Hit Ratio: 0.796296

Cache Miss Ratio: 0.203704

EX. 20

**1’s & 2’s COMPLEMENT USING DIFFERENT LOCATIONS**

| | |
|---|---|
| | |

**AIM:**

The program to find 1's and 2's complement of 8-bit number where starting address is 2000 and the number is stored at 3000 memory address and store result into 3001 and 3002 memory address.

**ALGORITHM:-**

1.Load the data from memory 3000 into A (accumulator)

2.Complement content of accumulator

3.Store content of accumulator in memory 3001 (1's complement)

4.Add 01 to Accumulator content

5.Store content of accumulator in memory 3002 (2's complement)

6.Stop .

**EXPLANATION:-**

1.A is an 8-bit accumulator which is used to load and store the data directly

2.LDA is used to load accumulator direct using 16-bit address (3 Byte instruction)

3.CMA is used to complement content of accumulator (1 Byte instruction)

4.STA is used to store accumulator direct using 16-bit address (3 Byte instruction)

5.ADI is used to add data into accumulator immediately (2 Byte instruction)

6.HLT is used to halt the program

**PROGRAM:-**

| MEMORY | MNEMONICS | OPERANDS | COMMENT |
|---|---|---|---|
| 2000 | LDA | [3000] | [A] <- [3000] |
| 2003 | CMA | | [A] <- [A^] |

| MEMORY | MNEMONICS | OPERANDS | COMMENT |
|---|---|---|---|
| 2004 | STA | [3001] | 1's complement |
| 2007 | ADI | 01 | [A] <- [A] + 01 |
| 2009 | STA | [3002] | 2's complement |
| 200C | HLT | | Stop |

**RESULT:-**

Thus the 1's & 2's complement using different locations has been executed successfully.

EX. 21

C PROGRAM TO CONVERT DECIMAL TO BINARY

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10],n,i;
printf("Enter the number to convert: ");
scanf("%d",&n);
for(i=0;n>0;i++)
{
a[i]=n%2;
n=n/2;
}
printf("\nBinary of Given Number is=");
for(i=i-1;i>=0;i--)
{
printf("%d",a[i]);
}
return 0;
}
```

EX. 22

C PROGRAM TO CONVERT DECIMAL TO OCTAL

```
#include <stdio.h>

int main()
```

```
{
   long decimalnum, remainder, quotient,octalnum=0;
   int octalNumber[100], i = 1, j;

   printf("Enter the decimal number: ");
   scanf("%ld", &decimalnum);
   quotient = decimalnum;

   //Storing remainders until number is equal to zero
   while (quotient != 0)
   {
      octalNumber[i++] = quotient % 8;
      quotient = quotient / 8;
   }

   //Converting stored remainder values in corresponding octal number
   for (j = i - 1; j > 0; j--)
      octalnum = octalnum*10 + octalNumber[j];
   printf("Equivalent octal value of decimal no %d is: %d  ", decimalnum,octalnum);
   return 0;
}
```

EX. 23

C PROGRAM TO CONVERT BINARY TO DECIMAL

```
#include <stdio.h>
void main()
{
   // declaration of variables
   int num, binary_num, decimal_num = 0, base = 1, rem;
   printf (" Enter a binary number with the combination of 0s and 1s \n");
```

```
   scanf (" %d", &num); // accept the binary number (0s and 1s)

   binary_num = num; // assign the binary number to the binary_num variable


   while ( num > 0)
   {
      rem = num % 10; /* divide the binary number by 10 and store the remainder in rem variable. */
      decimal_num = decimal_num + rem * base;
      num = num / 10; // divide the number with quotient
      base = base * 2;
   }

   printf ( " The binary number is %d \t", binary_num); // print the binary number
   printf (" \n The decimal number is %d \t", decimal_num); // print the decimal
}
```

EX. 24

CPU PERFORMANCE

```
#include <stdio.h>

int main()
{
  float cr;
  int p,p1,i;
  float cpu[5];
  float cpi,ct,max;
  int n=1000;
  for(i=0;i<=4;i++)
  {
```

```
    cpu[5]=0;
  }
  printf("\n Enter the number of processors:");
  scanf("%d",&p);
  p1=p;
  for(i=0;i<p;i++)
{
    printf("\n Enter the Cycles per Instrcution of processor:");
  scanf("%f",&cpi);
  printf("\n Enter the clockrate in GHz:");
  scanf("%f",&cr);
  ct=1000*cpi/cr;
  printf("The CPU time is: %f",ct);
  cpu[i]=ct;
}
max=cpu[0];
//printf("%f", max);
for(i=0;i<p1;i++)
{
    if(cpu[i]<=max)
    max=cpu[i];
}
printf("\n The processor has lowest Execution time is: %f ", max);
    return 0;
}
```

```
Enter the number of processors:3
Enter the Cycles per Instrcution of processor:1.5
Enter the clockrate in GHz:3
The CPU time is: 500.000000
```

Enter the Cycles per Instrcution of processor:1

Enter the clockrate in GHz:2.2

The CPU time is: 454.545441

Enter the Cycles per Instrcution of processor:2

Enter the clockrate in GHz:4

The CPU time is: 500.000000

The processor has lowest Execution time is: 454.545441

EX. 25

swap two 8-bit data using 8085

| | | |
|---|---|---|
| 2000 | LDA 2500 | A<-[2500] |
| 2003 | MOV B, A | B<-A |
| 2004 | LDA 2501 | A<-[2501] |
| 2007 | STA 2500 | 2500<-[A] |
| 200A | MOV A, B | A<-B |
| 200B | STA 2501 | 2501<-[A] |
| 200E | HLT | Terminates the program |

**Explanation –**