THADOMAL SHAHANI ENGINEERING COLLEGE
BANDRA (W), MUMBAI – 400050

CERTIFICATE

This is to certify that Mr./Miss Krish Naik of Computer Engineering Department, Semester 3 With Roll No. **2003120** has completed a course of the necessary experiments in the subject DLCA under my supervision in the **THADOMAL SHAHANI ENGINEERING COLLEGE** Laboratory in the year 20 - 20

_____ _____

Teacher in-charge **Head of Department**

Date: 10\12\2021 **Principal**

List of DLCA Assignments (C1/C2/C3)

Assignment No	Assignment
1	Decimal to binary and vice versa conversion and 2's complement
2	Verify the truth table and realize the logic expression
3	Implement NOT, AND ,OR and EXOR gates using the universal gates NAND and NOR
4	Implement Half Adder and Full Adder.
5	Implement conversion of 4-bit binary to gray code and vice versa.
6	Implement a 4-bit ripple carry adder.
7	Booth's Algorithm
8	Carry look ahead Generator
9	IEEE 754 conversion
10	Implement a 8:1 Mux and 3:8 Decoder
	Written Assignment
1	Write a short note on cache mapping techniques (Direct, Associative and Set associative mapping) with examples for each
2	Write a short note on multicore architecture

Prof . Tasneem Mirza

Prof. Vaishali Suryavanshi

Prof. Vaibhav Ambhire

Roll No :- 2003120

Page No. :
Date : / /

Experiment No 1

Aim :- A) Write a program to accept a decimal number and convert it to binary and vice ~~versa~~ versa.

B) Accept a signed decimal no (positive/negative) and find the 2's complement of the no.

Theory :-

a) ex :- $(10)_{10} = (01010)_2$

$(01100)_2 = (12)_{10}$

b) 2's Complement ex :- 5 = 0101

-5 = 1011

Program with output :-

tart here ✕ Decimal to Binary and vice versa.c ✕

```c
#include <stdio.h>
void main()
{
int a[10], number, i, j;
printf("\nEnter the Number You want to Convert  :  ");
scanf("%d", &number);
for(i = 0; number > 0; i++)
{
a[i] = number % 2;
number = number / 2;
}
printf("\nBinary Number of a Given Number = 0 ");
for(j = i - 1; j >= 0; j--)
{
printf(" %d ", a[j]);
}
printf("\n");


int num, binary, decimal = 0, base = 1, rem;
printf (" Enter a binary number with the combination of 0s and 1s \n");
scanf (" %d", &num);
binary = num;
while ( num > 0)
{
rem = num % 10;
decimal = decimal + rem * base;
num = num / 10;
base = base * 2;
}
printf ( " The binary number is %d \t", binary);
printf (" \n The decimal number is %d \t", decimal);
}
```

"C:\Users\krish\OneDrive\Documents\Decimal to Binary and vice versa.exe"

```
Enter the Number You want to Convert  :  11

Binary Number of a Given Number = 0  1  0  1  1
 Enter a binary number with the combination of 0s and 1s
1011
 The binary number is 1011
 The decimal number is 11
Process returned 29 (0x1D)   execution time : 12.640 s
Press any key to continue.
```

Start here | 2's compliment.c

```c
#include <stdio.h>
#include <math.h>
void main()
{
int n, absN, i, j, bin[50] = {};
printf("Enter Number\n");
scanf("%d", &n);
absN = abs(n);
for(i = 0; absN != 0; i++)
{
bin[i] = absN % 2;
absN /= 2;
}
printf("Your number in binary is ");
if(n < 0)
{
for(j = 0; j <= i; j++)
{
bin[j] = (bin[j] == 1) ? 0 : 1;
}
for(j = 0; bin[j] == 1; j++)
bin[j] = 0;
bin[j] = 1;
for(; i >= 0; i--)
printf("%d", bin[i]);
}
else
{
printf("0");
for(i = i-1; i >= 0; i--)
printf("%d", bin[i]);
}
}
```

"C:\Users\krish\OneDrive\Documents\2's compliment.exe"

```
Enter Number
-5
Your number in binary is 1011
Process returned 1 (0x1)   execution time : 4.516 s
Press any key to continue.
```

Roll No :- 2003120

Page No. :
Date : / /

Experiment No 2

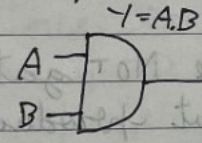
Aim :-

a) Verify the truth tables of the following gates

1) NOT
2) AND
3) OR
4) EXOR
5) NAND
6) NOR

Theory :-

a) 1) AND ~~Gate~~ Gate :- The And gate is an electronic circuit that gives a high input (1) only if all its inputs are high. A dot (.) is used to show And operation. A.B or AB.

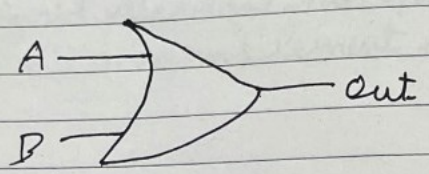


input		output
A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of AND Gate.

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

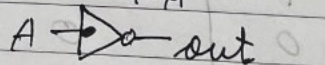
$Y = A + B$



input		output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

3) NOT Gate : The NOT gate is an electronic circuit ~~pt~~ that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. It is also known as A'

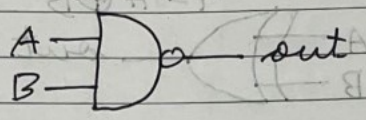
$Y = A'$



Input	Output
A	Y
0	1
1	0

4) NAND Gate : This is a NOT-AND gate which is equal to an AND gate. The outputs of all NAND gates are high if any of the inputs are low.

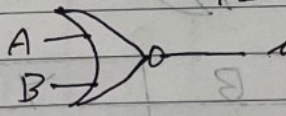
$Y = \overline{AB}$



Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

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

$Y = \overline{A+B}$

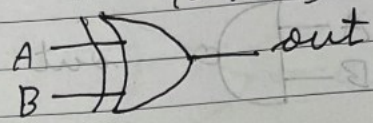


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Page No.:
Date: / /

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

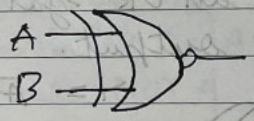
$Y = A \oplus B$



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

7) EXNOR Gate :- The EXNOR gate circuit does the opposite to the EXOR gate. It will give a low output if either, but not both inputs are high. ~~Th~~

$Y = \overline{A \oplus B}$



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

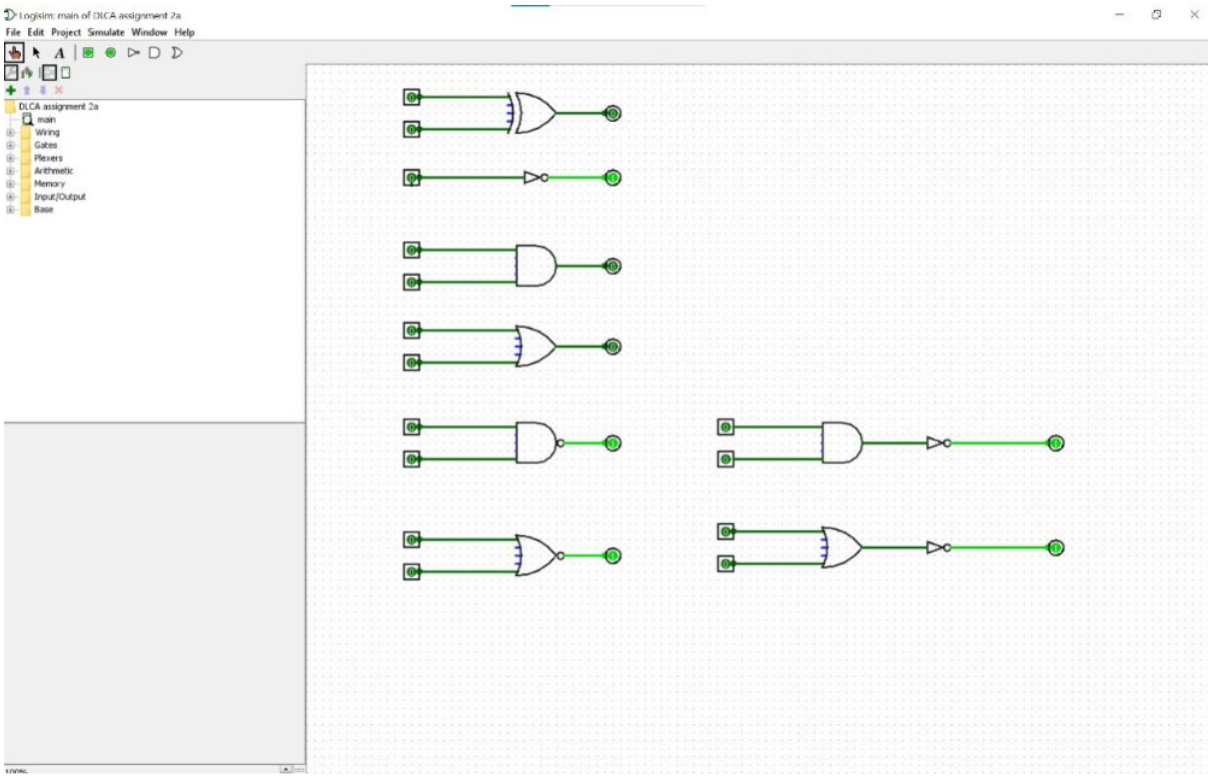
Page No. :
Date : / /

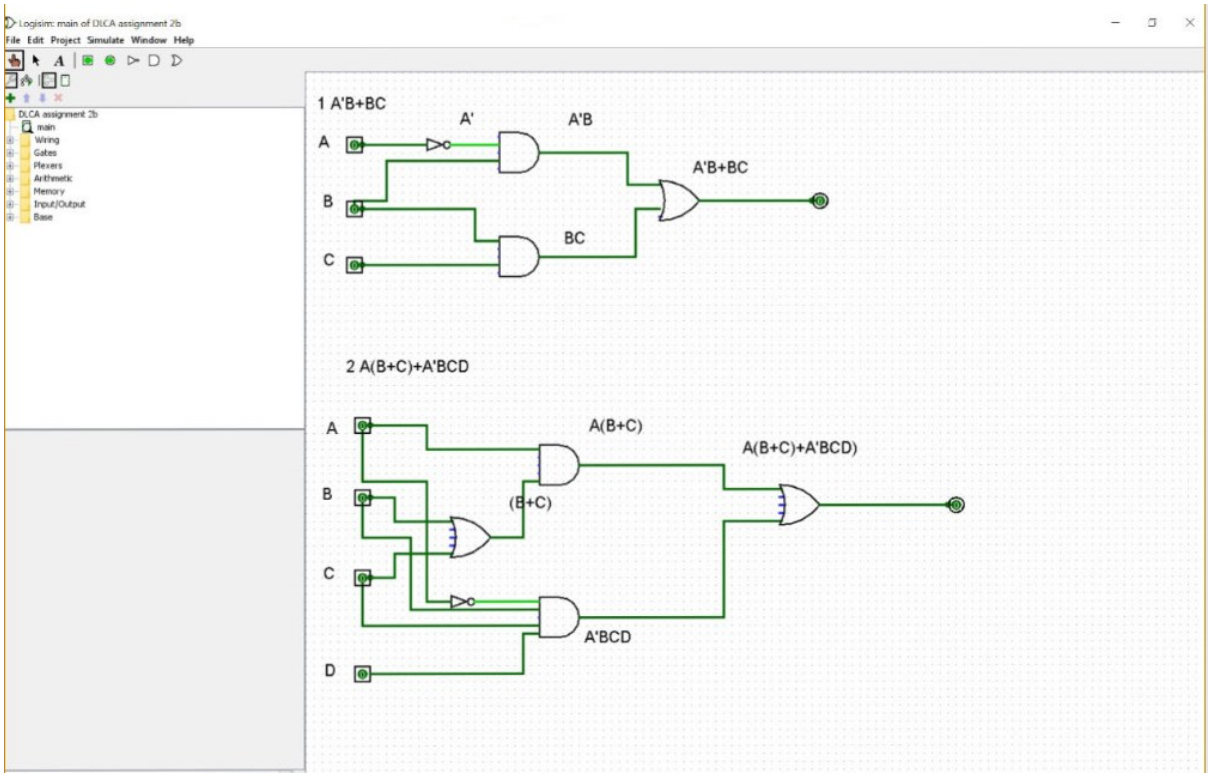
(a) 1) $A'B + BC$

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

2) $A(B+C) + A'BCD$

A	B	C	D	B+C	A(B+C)	A'BCD	A(B+C) + A'BCD
1	1	1	1	1	1	0	1
1	1	1	0	1	1	0	1
1	1	0	1	1	1	0	1
1	1	0	0	1	1	0	1
1	0	1	1	1	1	0	1
1	0	1	0	1	1	0	1
1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0
0	1	1	1	1	0	1	1
0	1	1	0	1	0	0	0
0	1	0	1	1	0	0	0
0	1	0	0	1	0	0	0
0	0	1	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0


Logisim: main of DLCA assignment 2a
File Edit Project Simulate Window Help
DLCA assignment 2a
main
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base


Logisim: main of DLCA assignment 2b
File Edit Project Simulate Window Help
DLCA assignment 2b
main
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base
1 A'B+BC
A
B
C
A'
A'B
A'B+BC
BC
2 A(B+C)+A'BCD
A
B
C
D
A(B+C)
(B+C)
A(B+C)+A'BCD
A'BCD

Roll No: 2003120

Page No.:
Date: / /

Experiment No 3

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

Theory:-

1.1) NAND gates as OR gate:- From DeMorgan's theorems:-
$(A.B)' = A' + B'$
$(A'.B')' = A + B$
So, give the inverted inputs to a NAND gate, obtain OR operation at output.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

1.2) NAND gates as AND gate:- An NAND produces complement of ~~AD~~ AND gate. So, if the output of a NAND gate is inverted the output will be of AND gate.

A	B	$Y = A.B$
0	0	0
0	1	0
1	0	0
1	1	1

1.3) NAND gates as Ex OR gate :- The output of a two input Ex OR gate is shown by: $Y = A'B + AB'$. This can be achieved with the logic diagram.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

1.4) NAND gates as Ex NOR gate :- An Ex-NOR gate is actually Ex OR gate followed by NOT gate. So give the output of Ex-OR gate to a NOT gate, overall output is of Ex-NOR gate.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

2.1) NOR gates as OR gate :- A NOR produces complement of OR gate. So, if output of a NOR gate is inverted, overall output will be of an OR gate.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

2.2) NOR gates as AND gate: from De Morgan's theorems:
$(A+B)' = A'B'$
$(A'+B')' = A''B'' = AB$.
So, give the inverted inputs to a NOR gate, obtain AND operation at output.

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

2.3) NOR gates as EX-OR gate: EX-OR gate is actually An EX-NOR gate followed by a NOT gate. So, give output of EX-NOR gate to a NOT gate, overall output is that of EX-OR gate.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

2.4) NOR gates as EX-NOR gate: The output of a two input EX-NOR gate is achieved by $Y = AB + A'B'$.

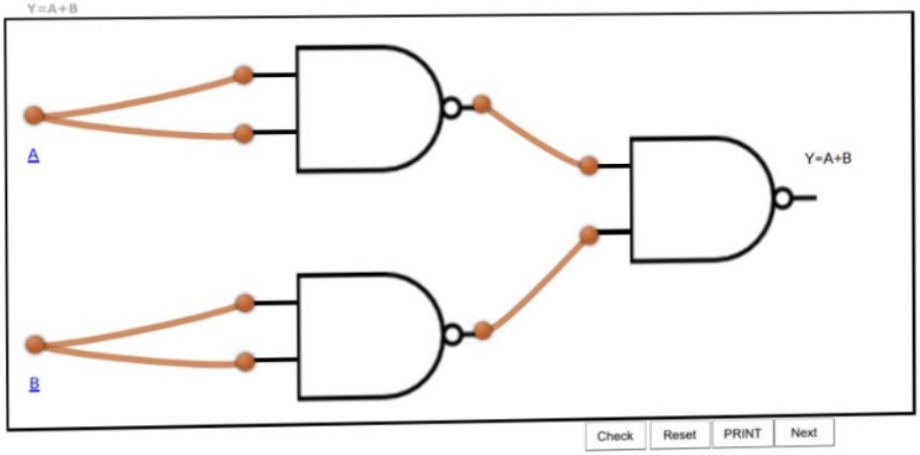
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

1- NAND Gates as OR Gate

INSTRUCTIONS

Experiment to perform logic of Or Using Nand on kit

Y=A+B



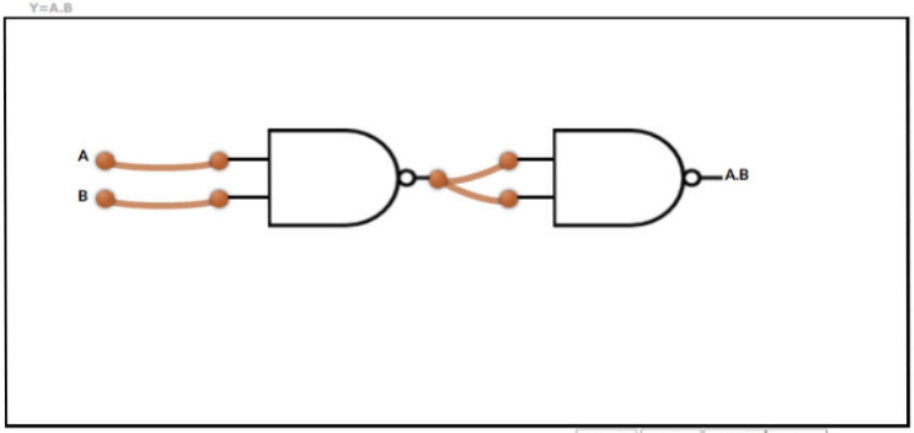
Check | Reset | PRINT | Next

2- NAND Gates as AND Gate

INSTRUCTIONS

Experiment to perform logic of AND Using NAND on kit

Y=A.B



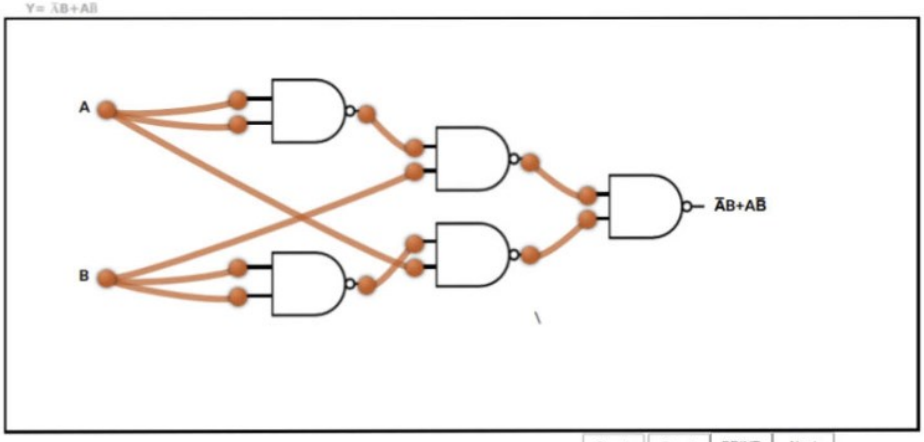
Check | Reset | PRINT | Next

3- NAND Gates as XOR Gate

INSTRUCTIONS

Experiment to perform logic of Ex-OR Using Nand on kit

Y= $\bar{A}B+A\bar{B}$



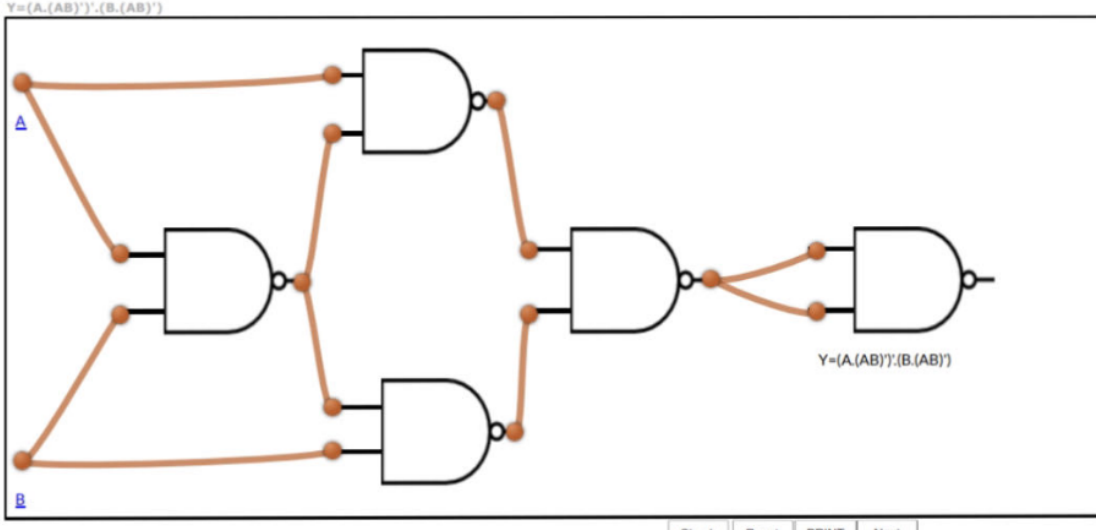
Check | Reset | PRINT | Next

4- NAND Gates as XNOR Gate

INSTRUCTIONS

Experiment to perform logic of Ex-NOR Using Nand on kit

Y=(A.(AB)')'.(B.(AB)')



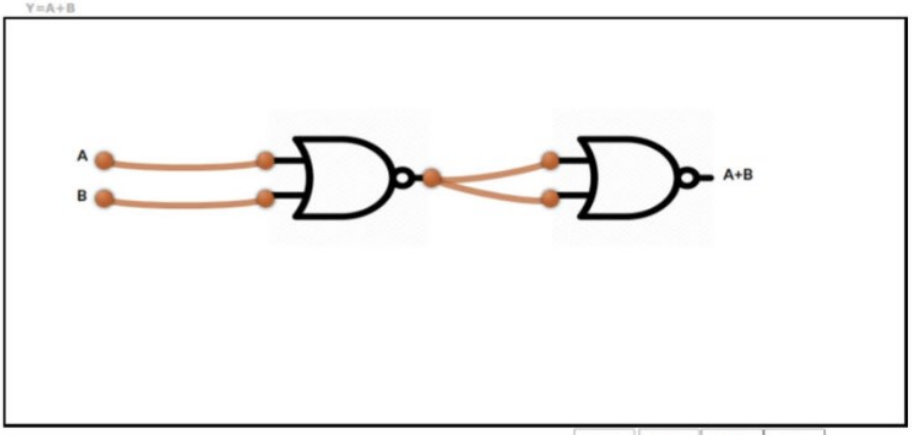
Check | Reset | PRINT | Next

1- NOR Gates as OR Gate

INSTRUCTIONS

Experiment to perform logic of OR Using NOR on kit

Y=A+B



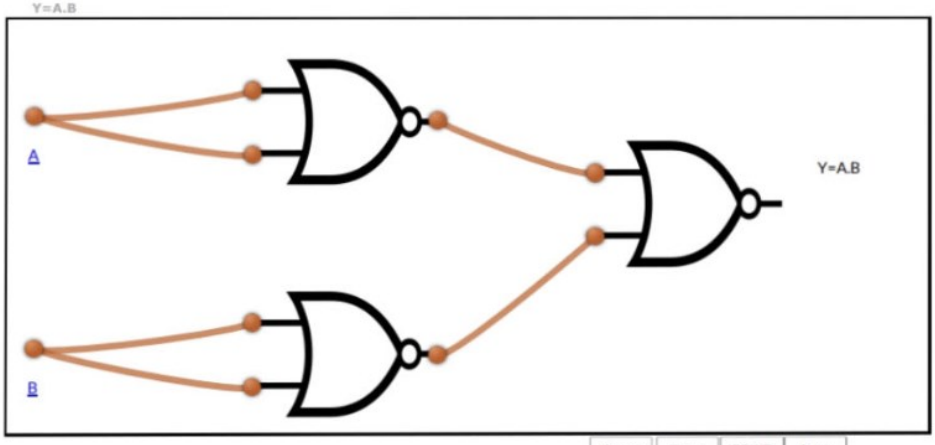
Check | Reset | PRINT | Next

2-NOR Gates as AND Gate

INSTRUCTIONS

Experiment to perform logic of And Using Nor on kit

Y=A.B

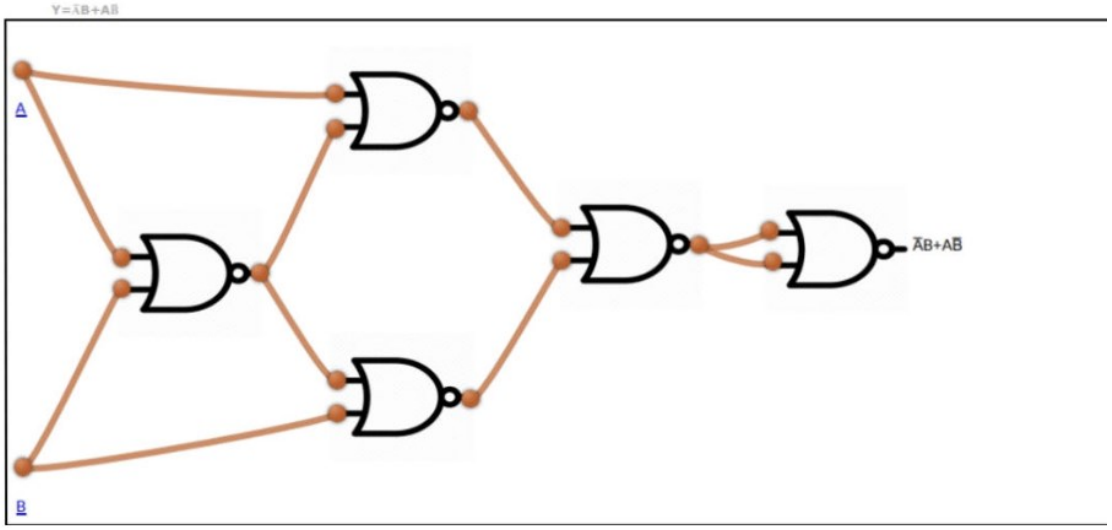


Check | Reset | PRINT | Next

3-NOR Gates as XOR Gate

INSTRUCTIONS

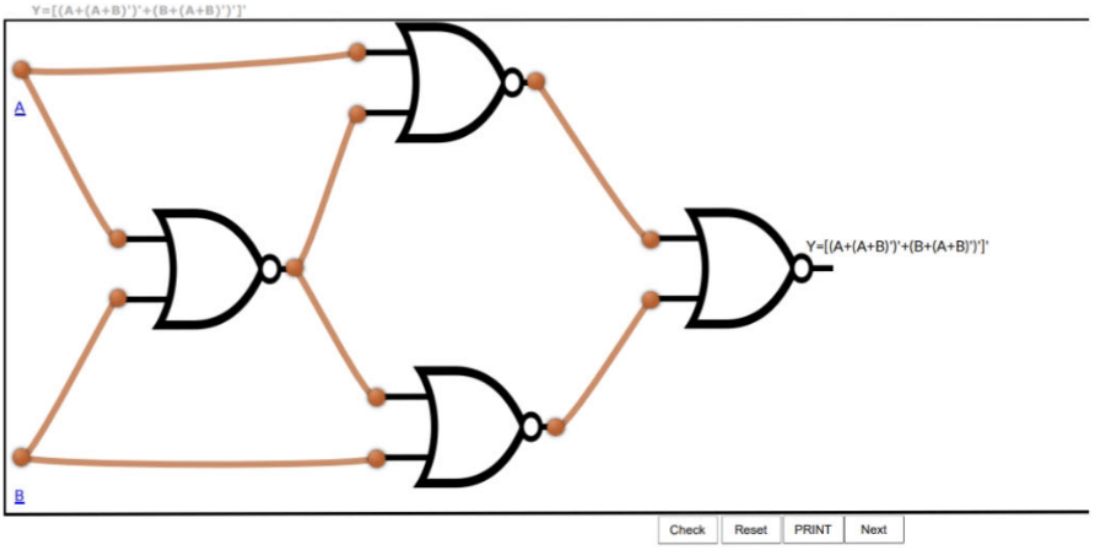
Experiment to perform logic of Ex-OR using NOR on kit



4-NOR Gates as XNOR Gate

INSTRUCTIONS

Experiment to perform logic of Ex-NOR Using Nor on kit



Check | Reset | PRINT | Next

Roll No: 2003120

Page No.:
Date: / /

Experiment No 4

Aim:- Using the logisim simulator:
Implement half-adder and full-adder.

Draw the truth table and realize the expression for half and full adder.

Theory:-

1) Half Adder: It is a combinational circuit that performs simple addition of two ~~to~~ binary numbers. If we assume A and B as the two bits whose addition is to be performed, the truth table for half adder with A, B as inputs and Sum, Carry as outputs can be calculated as

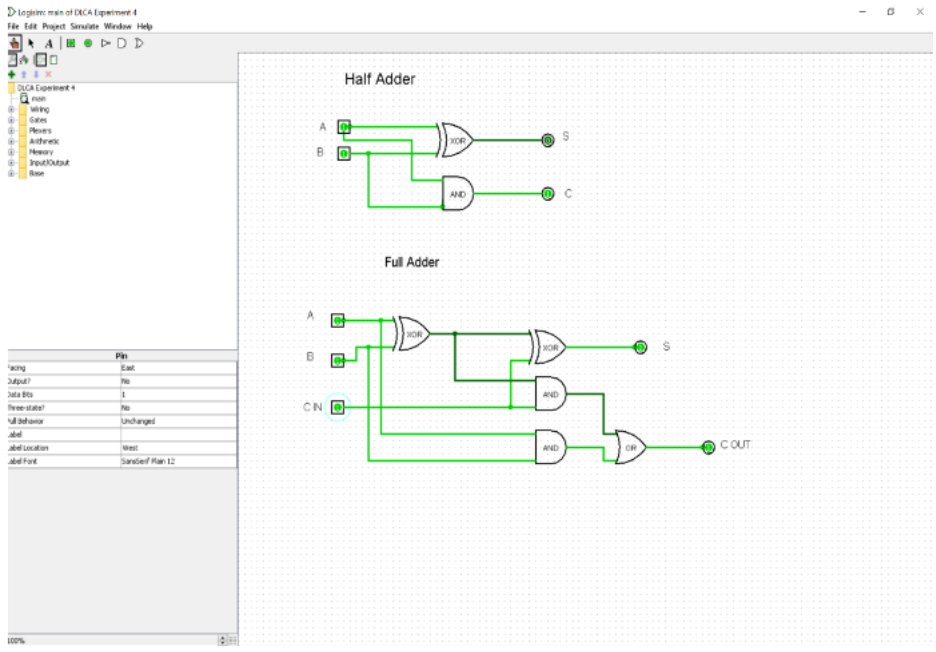
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder: It is a digital circuit used to calculate the sum of three binary bits. Full adders are complex and difficult to implement when compared to half adders. Two of the three bits are same as before which are A and B.

The additional third bit is carry bit from the previous stage and is called carry generally represented by CIN. It calculates the sum of three bits along with the carry. The output carry is called carry out and is represented by COUT.

The truth table of full adder with A, B and CIN as inputs and Sum and Cout as outputs is:

Input			Output	
A	B	CIN	Sum	Carry
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



Roll No: 2003120

Page No.:
Date: / /

Experiment No 5.

Aim: Using logisim simulator:
Implement conversion of 4-bit binary to gray code and vice versa.

Theory:

1) Binary to gray conversion:
a) MSB of the gray code is always equal to MSB of the given binary code.

b) Other bits of the output gray code can be obtained by ~~EXOR~~ to EXORing binary code bit at the index and previous index.

There are 4 inputs and outputs. The input variables are defined as B_3, B_2, B_1, B_0 and output variables are defined as G_3, G_2, G_1, G_0

$B_3 = G_3$

$B_2 \oplus B_3 = G_2$

$B_1 \oplus B_2 = G_1$

$B_0 \oplus B_1 = G_0$

Natural-binary code				gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
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	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

2) ~~by~~ Gray to binary conversion:

1) MSB of the binary code is always equal to MSB of given binary number.

2) Other bits of the output binary code can be obtained by checking gray code bit at that index. If current gray code bit is 0, then copy previous binary code bit, else copy inverse of previous binary code bit

There are 4 inputs and outputs. The input variable are defined as G_3, G_2, G_1, G_0 and output variables are defined as B_3, B_2, B_1, B_0.

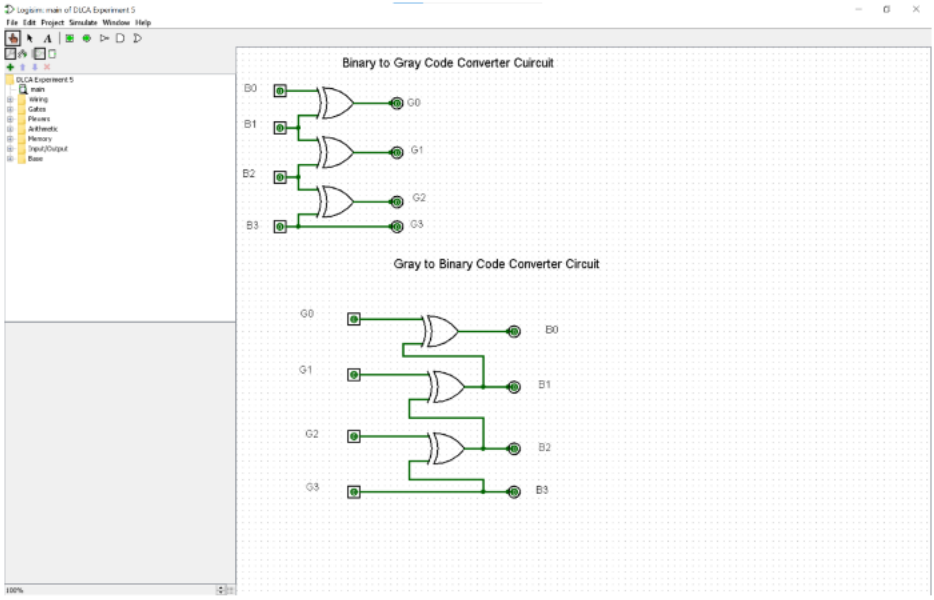
$G_0 \oplus G_1 \oplus G_2 \oplus G_3 = B_0$

$G_1 \oplus G_2 \oplus G_3 = B_1$

$G_2 \oplus G_3 = B_2$

$G_3 = B_3$

Gray code				Natural - binary code			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
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



Roll No:- 2003120

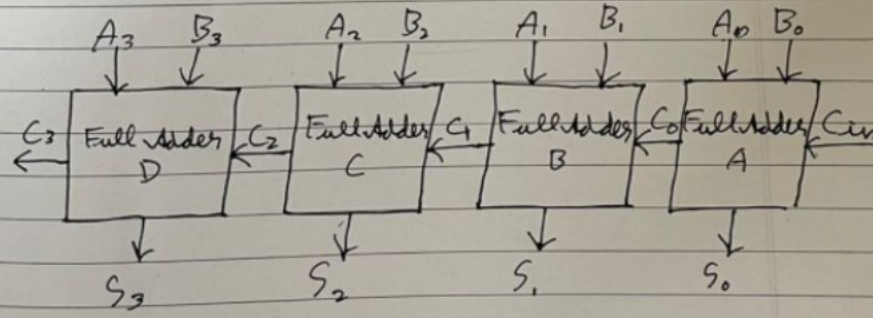
Page No.:
Date: / /

Experiment no. 6

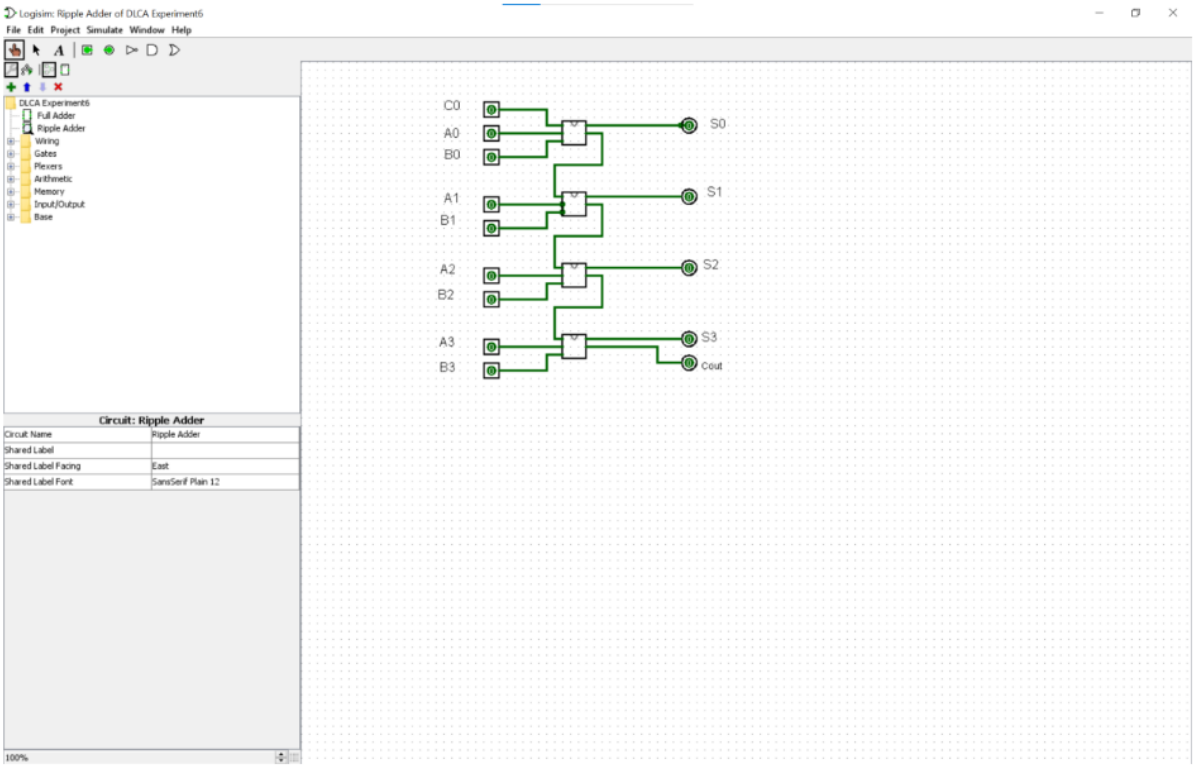
Aim:-
Using the logisim simulator -
Implement a 4-bit ripple carry adder.

Use the 1-bit Full adder to implement the ripple carry adder (four 1-bit full adders)

Theory:-
Ripple Carry Adder is a combinational logic circuit. It is used for the purpose of adding two n-bit binary numbers. It requires n full adders in its circuit for adding two n-bit binary numbers. It is also known as n-bit parallel adder. 4-bit ripple carry adder is used for the purpose of adding two 4-bit binary numbers.



4-bit Ripple Carry Adder


Logisim: Ripple Adder of DLCA Experiment6
File Edit Project Simulate Window Help
DLCA Experiment6
Full Adder
Ripple Adder
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base
Circuit: Ripple Adder
Circuit Name
Ripple Adder
Shared Label
Shared Label Facing
East
Shared Label Font
SansSerif Plain 12
100%
C0
A0
B0
A1
B1
A2
B2
A3
B3
S0
S1
S2
S3
Cout

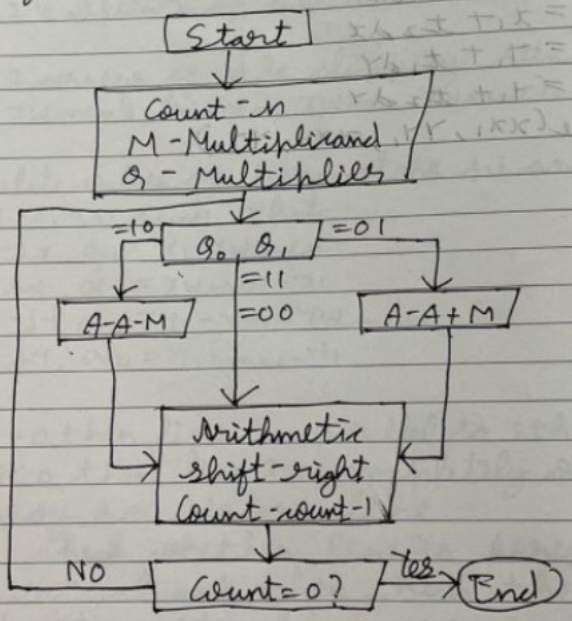
Roll No: 2003120

Page No.:
Date: / /

Experiment no 7

Aim: Implement Booth's algorithm

Theory:



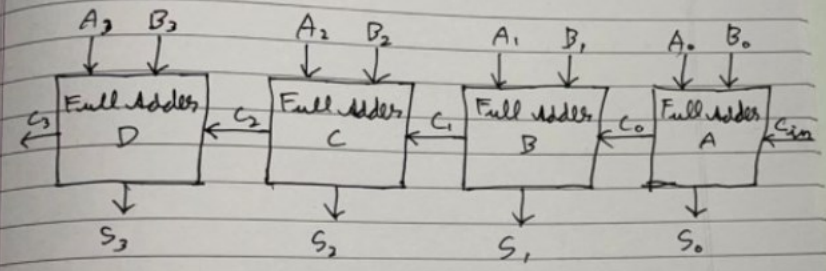
Roll No :- 2003120

Page No. :
Date : 1 1

Experiment no 8

Aim :- Implement Carry look ahead generator.

Theory :-



A	B	Cin	Carry
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

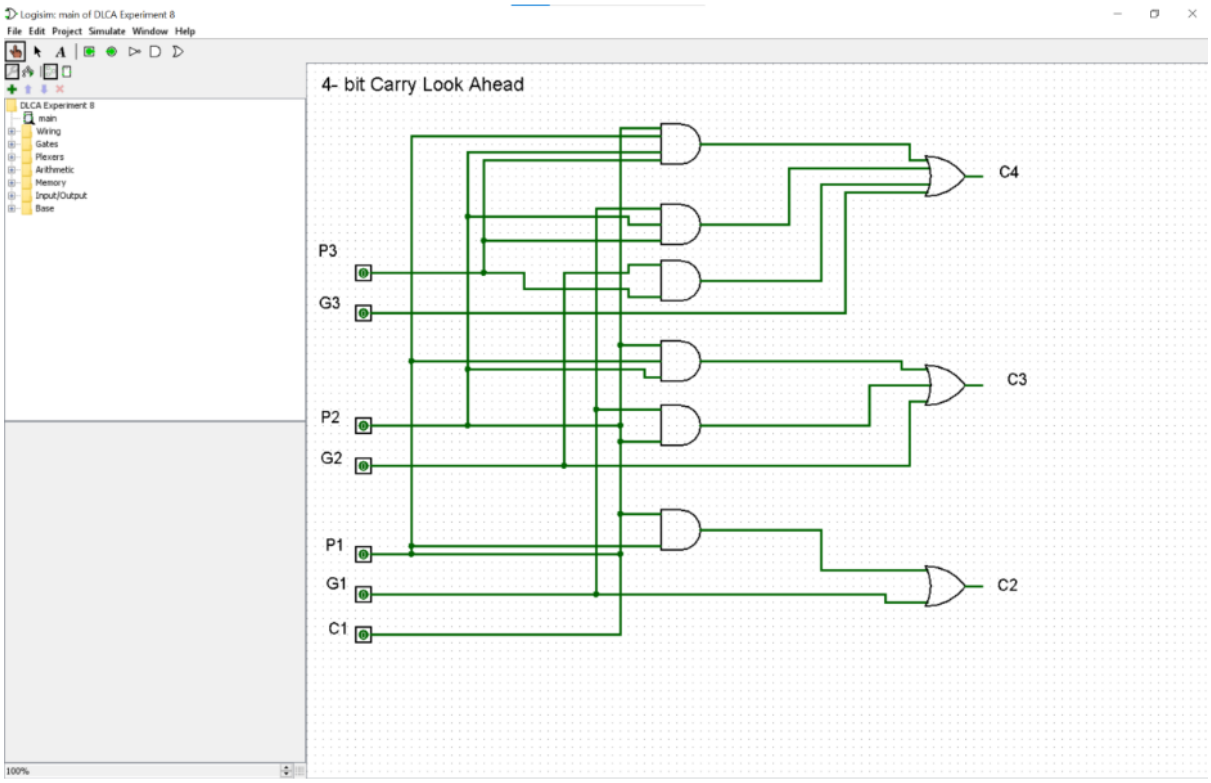
As $C_{in} = 0$

$C_1 = G_0$

$C_2 = G_1 + P_1 C_1$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 C_1$

$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$


Logisim: main of DLCA Experiment 8
File Edit Project Simulate Window Help
DLCA Experiment 8
main
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base
100%
4- bit Carry Look Ahead
P3
G3
P2
G2
P1
G1
C1
C4
C3
C2

Roll No: 2003120

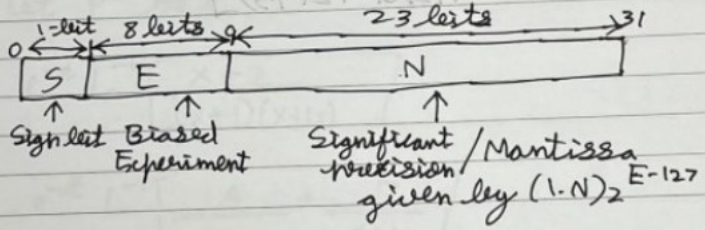
Experiment No 9

Aim: Program to implement IEEE-754 floating point single precision format.

Theory: IEEE single-precision floating point format is a binary computing format that occupies (32 bits) 4 bytes in computer memory.

Three numbers are associated with a floating point to number of single precision.

1. Sign bit: 1 bit
2. Exponent width: 8 bits
3. Significant precision: 23 bits



1) The sign of Mantissa: 0 represents a positive number while 1 represents a negative number.

2) The Biased experiment: The exponent field needs to represent both positive and negative exponents. A bias is added to get the stored exponent.

3) Mantissa: It is a floating point number consisting of its significant digits.

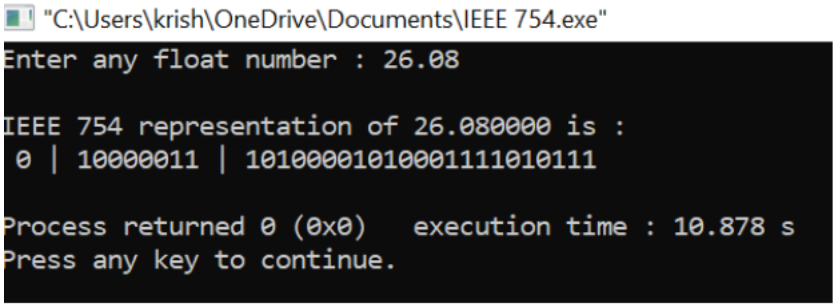
Program:

```
#include <stdio.h>

void printBinary(int n, int i)

{

 int k;

 for (k = i - 1; k >= 0; k--)
```

```
 {
 if ((n >> k) & 1)
 printf("1");
 else
 printf("0");
 }
}
typedef union
{
 float f;
 struct
 {
 unsigned int mantissa : 23;
 unsigned int exponent : 8;
 unsigned int sign : 1;
 } raw;
} myfloat;
void printIEEE(myfloat var)
{

 printf(" %d | ", var.raw.sign);
 printBinary(var.raw.exponent, 8);
 printf(" | ");
 printBinary(var.raw.mantissa, 23);
 printf("\n");
}
int main()
```

```
{
 myfloat var;
 printf("Enter any float number : ");
 scanf("%f",&var);
 printf("\nIEEE 754 representation of %f is : \n", var.f);
 printIEEE(var);
 return 0;
}
```

Output:



Roll No: 2003120

Page No.:
Date: / /

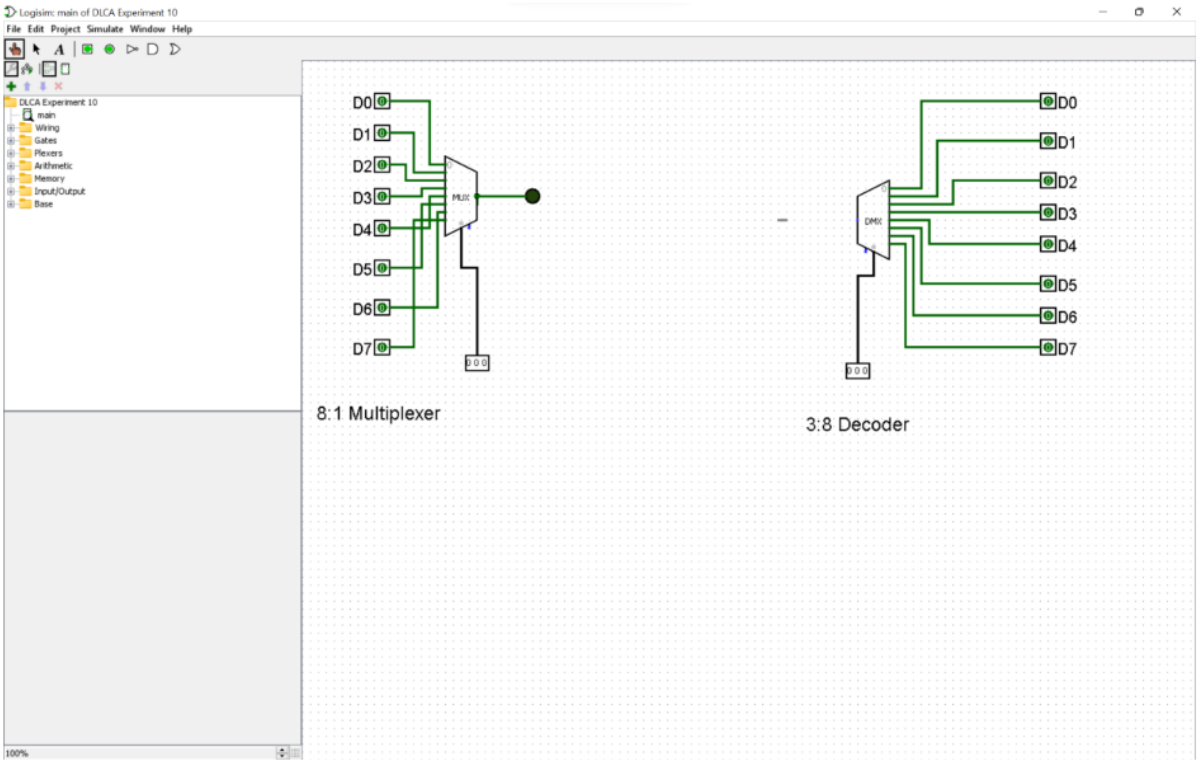
Experiment no 10

Aim: Implement 8:1 multiplexer and 3:8 decoder.

Theory: ~~Mux~~ Multiplexer is a device that has 2^n input lines but has only 1 output line. where n = number of input selector line.

Truth table

Din	Cin	Bin	Ain	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	0	1	1	1	1	1	1	1	D
0	0	0	1	0	1	1	0	0	0	0	0
0	0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1	1
0	1	0	1	1	0	1	1	0	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0


Logisim: main of DLCA Experiment 10
File Edit Project Simulate Window Help
DLCA Experiment 10
main
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base
100%
D0
D1
D2
D3
D4
D5
D6
D7
MUX
000
8:1 Multiplexer
D0
D1
D2
D3
D4
D5
D6
D7
DMX
000
3:8 Decoder

Roll No: 2003120

Page No.:
Date: / /

Assignment No. 1

Cache Mapping Techniques: Mapping function and replacement algorithm together decides where a line from the main memory can reside in the cache.

1) Direct Mapping Technique:
In this case each block of main memory can map to only one cache line. A given block maps to any line (i mod j), where i is the line number of the main memory to be mapped and j is the total number of lines in the cache memory.

The address is divided into 3 parts: the word selector, line selector and tag. Least Significant w bits identify unique word of a particular line. Most significant s bits specify one memory block to which the cache line corresponds. The MSB's are split into a cache line field r and a tag of s-r.

Example: let cache be of 64 kByte that is divided into blocks of 4 bytes hence cache is 16 k (2^{14}) lines of 4 bytes.
let the main memory size be 16 MBytes that requires 24 bit address lines (2^{24}=16 M).

Tag(s-r) (8 bits)	Line(r) (14 bits)	Word(w) (2 bits)

Page No.:
Date: / /

Advantages:-
1) Simple implementation
2) Inexpensive

Disadvantage:-
Fixed location for given block hence if a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

2) Associative Mapping:-

In this case a main memory block can load into any line of cache. There are only two fields in the address as tag and word. The tag uniquely identifies block of memory from where the line has been copied into the cache memory. To search a particular data the tag of every line is to be examined for a match. Thus cache searching gets expensive in terms of time required.

Example: let cache be of 64 k byte that is divided into blocks of 4 bytes hence cache is 16 k (2^{14}) lines of 4 bytes.
let the main memory size be 16 M bytes that requires 24 bit address lines (2^{24}=16 M).

The associative Mapping Address structure for this example considered: 22 bit tag stored with each 4-word block of data.

Compare tag field with tag entry in cache to check for least Significant 2 bits of address identify which word is required from 4-word data block.

Tag (22 bits)	Word (2 bits)

Advantages:

1) If a program accesses 2 blocks repeatedly, cache misses will not occur.

Disadvantages:

1) Complex design for many parallel comparisons of tag.
2) Expensive due to implementation of parallel comparator.

3) Set Associative Mapping:

In this case cache is divided into a number of sets. Each set contains a number of lines. A given block maps to any line in a given set (i mod j), where i is the line number of the main memory to be mapped and j is the total number of sets in the cache memory.

Example: let cache be of 64kByte that is divided into blocks of 4 bytes hence cache is 16k (2^{14}) lines of 4 bytes. let the main memory size be 16MBytes that requires

24 bit address lines (2^{24} = 16M).
For this example for set associative mapping address structure: 2 bits for one of the 4 words, 8K lines in each of the 2 sets hence 13 bits to select a set (2^{13} = 8K) and remaining (24-13-2=9) bits for tag.

Tag (9 bits)	Set (13 bits)	Word (2 bits)

Advantages:

1) If a program accesses 2 blocks that map to the same set repeatedly, cache misses will not occur because they would go into different lines of the set.

2) Not very complex because of just 2, 4 or 8 parallel comparisons.

3) Not much expensive again because of simple implementation.

Roll No: 2003120

Assignment No 2:

Multicore refers 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 these processors. These single integrated circuits are known as a die. Multicore architecture places multiple processor cores and bundles 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.

This technology is most commonly used in multicore processors, where two or more processors chips or cores run concurrently as a single system. Multicore-based processors are used in mobile devices, desktops, workstations and servers.

The concept of multicore technology is mainly centred 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 reduces the system's heat and power consumption. This means much better performance with less or the same amount of energy.

The architecture of a multicore processor enables communication between all available cores to ensure that the processing tasks are divided and assigned accurately. At the time of task completion, the processed data from each core is delivered back to the motherboard by means of a single shared gateway. This technique significantly enhances performance compared to a single-core processor of similar speed.

Multicore technology is very effective in challenging tasks and applications, such as encoding, 3-D gaming and video editing.