BACHELOR OF COMPUTER APPLICATION LAB MANUAL

1st Semester

Prepared By

Pure and Applied Science Dept.

Computer Application

MIDNAPORE CITY COLLEGE

INSTRUCTIONS TO STUDENTS

- Before entering the lab, the student should carry the following things (MANDATORY)
    1. Identity card issued by the college.
    2. Class notes
    3. Lab observation book
    4. Lab Manual
    5. Lab Record
- Student must sign in and sign out in the register provided when attending the lab sessionwithout fail.
- Come to the laboratory in time. Students, who are late more than 10 min., will not beallowed to attend the lab.
- Students need to maintain 80% attendance in lab if not a strict action will be taken.
- All students must follow a Dress Code while in the laboratory
- Foods, drinks are NOT allowed.
- All bags must be left at the indicated place.
- Refer to the lab staff if you need any help in using the lab.
- Respect the laboratory and its other users.
- Workspace must be kept clean and tidy after experiment is completed.
- Read the Manual carefully before coming to the laboratory and be sure about what youare supposed to do.
- Do the experiments as per the instructions given in the manual.
- Copy all the programs to observation which are taught in class before attending the labsession.
- Students are not supposed to use floppy disks, pen drives without permission of lab- incharge.
- Lab records need to be submitted on or before the date of submission.

C PROGRAMMING LABORATORY MANUAL

(Course Code: BCA-1196)

OBJECTIVES

1. To introduce students to the basic knowledge of programming fundamentals of C language.

2. To impart writing skill of C programming to the students and solving problems.

3. To impart the concepts like looping, array, functions, pointers, file, structure.

COURSE OUTCOME:

After completing this lab course, you will be able to:

1. Understand the logic for a given problem.

2. Write the algorithm of a given problem.

3. Draw a flow chart of a given problem.

4. Recognize and understand the syntax and construction of C programming code.

5. Gain experience of procedural language programming.

6. Know the steps involved in compiling, linking and debugging C code.

7. Understand using header files.

8. Learn the methods of iteration or looping and branching.

9. Make use of different data-structures like arrays, pointers, structures and files.

10. Understand how to access and use library functions.

11. Understand function declaration and definition.

12. Understand proper use of user defined functions.

13. Write programs to print output on the screen as well as in the files.

14. Apply all the concepts that have been covered in the theory course, and

15. Know the alternative ways of providing solution to a given problem.

S.No.	Experiments
1.	To find the sum of individual digits of a given number
2.	To print the Fibonacci series for 1 to n value
3.	To print a prime number up to 1 to n
4.	To calculate the sum. Sum=1-x2/2! + x4/4! - x6/6! + x8/8! - x10/10!
5.	Programs that use recursive function to find the factorial of a given integer.
6.	Program that uses non recursive function to find the factorial of a given integer.
7.	To find the GCD of two given integers by using the recursive function
8.	Two integer operands and one operator form user, performs the operation and then prints the result. (Consider the operators +, -, *, /, % and use Switch Statement)
9.	To find both the largest and smallest number in a list of integers.
10.	To perform the addition of two matrices
11.	Functions to insert a sub string into given main string from a given position.
12.	To generate Pascal ‘s triangle
13.	To convert the given binary number to 2‘s complement
14.	To read the two complex numbers and perform the addition and multiplication of these two numbers.
15.	Program which copies one file to another

Objective: To find the sum of individual digits of a given number.

Description:

Sum of the individual digits means adding all the digits of a number Ex: 123 sum of digits is 1+2+3=6

Algorithm:

Step 1: start
Step 2: read n
Step 3: initialize the s=0
Step 4: if n<0 goto Step 7
Step 5: if n!=0 goto Step 6 else goto step 7
Step 6: store n%10 value in p
Add p value to s
Assign n/10 value to n
Goto Step 5
Step 7: print the output
Step 8:stop

Program:

```
#include<stdio.h>
main()
{
intn,s,p;
clrscr();
printf("enter the vaue for n:\n");
scanf("%d",&n);
s=0;
if(n<0)
printf("The given number is not valid");
else
{
while(n!=0) /* check the given value =0 or not */
{
p=n%10;
n=n/10;
s=s+p;
}
printf("sum of individual digits is %d",s);
}
getch();
}
```

Output:

1.Enter the value for n: 333
Sum of individual digits is 9
2.Enter the value for n: 4733

Sum of individual digits is 17

Outcome:

1.To understand basic structure of a C program.
2. To understand handling of number system in a C program

Objective 2: To print the Fibonacci series for 1 to n value

Description

A Fibonacci series is defined as follows
The first term in the sequence is 0
The second term in the sequence is 1
The sub sequent terms 1 found by adding the preceding two terms in the sequence
Formula: let t1, t2,……tn be terms in Fibonacci sequence
t1=0, t2=1tn=tn-2+tn-1……where n>2

Algorithm:

Step 1: start
Step 2: initialize the a=0, b=1
Step 3: read n
Step 4: if n== 1 print a go to step 7. else goto step 5
Step 5: if n== 2 print a, b goto step 7 else print a, b
Step 6: initialize i=3
i)ifi<= n do as follows. If not goto step 7
c=a+b
print c
a=b
b=c
increment I value
goto step 6(i)
Step 7: stop

Program:

```
#include<stdio.h>
void main()
{
inta,b,c,n,i;
clrscr();

printf("enter n value");
scanf("%d",&n);a=0;
b=1;
if(n==1)
printf("%d",a);
else
if(n==2)
printf("%d%d",a,b);
else
{
printf("%d%d",a,b);
//LOOP WILL RUN FOR 2 TIME LESS IN SERIES AS THESE WAS
PRINTED IN ADVANCE
for(i=2;i<n;i++)
{
c=a+b;
printf("%d",c);
a=b;
b=c;
```

```
}
getch();
}
}
```

Output:

1. Enter n value: 5
0 1 1 2 3
2. Enter n value: 7
0 1 1 2 3 5 8

Outcome:

1. To learn basic looping contracts in a C program,

Objective 3: To print a prime number up to 1 to n

Description:

Prime number is a number which is exactly divisible by one and itself only Ex: 2, 3,5,7,……;

Algorithm:

Step 1: start
Step 2: read n
Step 3: initialize i=1, c=0
Step 4: if i<=n goto step 5
If not goto step 10
Step 5: initialize j=1
Step 6: if j<=1 do as the follow. If no goto step 7
i)ifi%j==0 increment c
ii) increment j
iii) goto Step 6
Step 7: if c== 2 print i
Step 8: increment i
Step 9: goto step 4
Step 10: stop

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, i, fact, j;
clrscr();
printf("enter the number:");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
fact=0;
//THIS LOOP WILL CHECK A NO TO BE PRIME NO. OR NOT.
for(j=1; j<=i; j++)
{
if(i%j==0)
fact++;
}
if(fact==2)
printf("\n %d",i);
}
getch( );
}
```

Output:

Enter the number: 5
2 3 5
Enter the number: 10
2 3 5 7

Enter the number: 12
2 3 5 7 11

Outcome:

1. To understand basic looping constructs in C programs
2. To handle various mathematical problems.

Objective 4: To calculate the sum. Sum=1-x2/2! + x4/4! - x6/6! + x8/8! - x10/10!

Algorithm: main program:
Step 1: start

Step 2: declare x, i, n, s=0, c
Step 3: read x value
Step 4: for i=0, n=0; i<=10; i=i+2, n++ goto step 5
Step 5: s=s+(pow(-1, n)*pow(x,i)/fact(i))
Step 6: print s value
Step 7: stop
Sub program:
Step 1: while x!=0 goto Step 2
Step 2: y=y+x; x—
Step 3: return y
Step 4: return to main program

Program:

```
#include<stdio.h>
#include<math.h>
long fact(int);
void main()
{
intx,i,n;
float s=0,c;
clrscr();
printf("\n enter the value of x\t");
scanf("%d",&x);
/*perform the looping operation*/
for(i=0,n=0;i<=10;i=i+2,n++)
s=s+(pow(-1,n)*pow(x,i)/fact(i));
printf("\n the result is %f",s);
getch();
}
/* calling sub program*/
long fact(int x)
{ longint y=1;
 while(x!=0)
{
y=y*x; x--;
}
return y;
}
```

Output: 1. Enter the value of x: 1 The result is 0.540302 2 Enter the value of x: 2 The result is -0.416155

Outcome: 1. To handle and calculate series-based problems

Objective 5: Programs that use recursive function to find the factorial of a given integer.

Description:

Factorial of a number is nothing but the multiplication of numbers from a given number to 1

Algorithm: main program
Step 1: start
Step 2: read n
Step 3: call sub program as f=fact(n)
Step 4: print f value
Step 5: stop
Sub program:
Step 1: initialize the f
Step 2: if n= = 0 or n == 1 return 1 to main program if not goto step 3
Step 3: return n*fact(n-1) to main program

Program:

```
#include<stdio.h>
#include<conio.h>
int fact(int n)
{
int f;
if((n==0)||(n==1)) // check the condition for the n value
return(n);
else
f=n*fact(n-1); //calculate the factorial of n
return(f);
}
void main()
{
int n;
clrscr();
printf("enter the number :");
scanf("%d", &n);

printf("factorial of number %d", fact(n));
getch();
}
```

Output:
1. Enter the number: 5
Factorial of number: 120
2. Enter the number: 3
Factorial of number: 6

Outcome:

1. To understand recursion-based concepts in C programming.

Objective 6: Program that use non recursive function to find the factorial of a given integer.

Description:

Factorial of a number is nothing but the multiplication of numbers from a given number to 1
Ex: 5! =5*4*3*2*1= 120

Algorithm:

main program
Step 1: start
Step 2: read n
Step 3: call the sub program fact(n)
Step 4: print the f value
Step 5: stop

Sub program:

Step 1: initialize the f=1
Step 2: if n==0 or n=1 return 1 to main program. If not goto step 3
Step 3: perform the looping operation as follows
For i=1 i<=n; i++
Step 4: f=f*i
Step 5: return f value to the main program

Program:

```
#include<stdio.h>
#include<conio.h>
int fact(int n) //starting of the sub program
{
int f=1,i;
if((n==0)||(n==1)) // check the condition for n value
return(1);
else
for(i=1;i<=n; i++) // perform the looping operation for calculating the factorial
f=f*i;
return(f);
}
void main()

{
int n;
clrscr();
printf("enter the number :");
scanf("%d", &n);
printf("factorial of number%d", fact(n));
getch();
}
```

Output:

1.Enter the number: 7
Factorial of number: 5040

2. Enter the number: 6
Factorial of number: 720
3. Enter the number: 8
Factorial of number: -25216

Outcome:

1. To handle recursive problems without recursion-based function call.

Objective 7: To find the GCD of two given integers by using the recursive function

Description:

GCD means Greatest Common Divisor. i.e the highest number which divides the given number
Ex: GCD (12,24) is 12
Formula: GCD= product of numbers/ LCM of numbers

Algorithm:

main program
Step 1: start
Step 2: read a, b
Step 3: call the sub program GCD (a, b) for print the value
Step 4: stop
Sub program:
Step 1: if n>m return GCD (n, m)
Step 2: if n==0 return m else goto step 3
Step 3: return GCD (n, m%n)
Step 4: return to main program

Program:

```
#include<stdio.h>
#include<conio.h>
int gcdrecursive(int m, int n) // starting of the sub program
{
if(n>m)
return gcdrecursive(n, m);
if(n==0)
return m;
else
return gcdrecursive(n, m%n); // return to the main program
}
void main()
{
inta,b,igcd;
clrscr();

printf("enter the two numbers whose gcd is to be found:");
scanf("%d%d",&a,&b);
printf("GCD of a,b is %d",gcdrecursive(a,b)); // return to the sub program
getch();
}
```

Output:

1. enter the two numbers whose gcd is to be found:5,25
GCD of a,b is : 5

2. enter the two numbers whose gcd is to be found:36,54
GCD of a,b is : 18
3. enter the two numbers whose gcd is to be found:11,13
GCD of a,b is : 1

Outcome:

1. To understand basic structure of a C program.
2. To understand handling of number system in a C program

Objective 8: Two integer operands and one operator form user, performs the operation and then prints the result. (Consider the operators +,-,*, /, % and use Switch Statement)

Description:

To take the two integer operands and one operator from user to perform the some arithmetic operations by using the following operators like+,-,*, /, % ,Ex: 2+3=5

Algorithm:

Step 1: Start
Step 2: Read the values of a,b and operator
Step 3: if the operator is _+‘ then
R=a+b
Go to step 8
Break
Step 4: Else if the operator is _-_ then
R=a-b
Go to step 8
Step 5: Else if the operator is _*_ then
R=a*b
Go to step 8
Step 6: Else if the operator is _/_ then
R=a/b
Go to step 8
Step 7: Else if the operator is _%_ then
R=a%b
Go to step 8
Step 8: write R
Step 9:End

Program:

```
#include<stdio.h>
main()
{
char op;

floata,b,c;
clrscr();
printf("enter two operands:");
scanf("%d%d",&a,&b);
printf("enter an operator:");
scanf(" %c",&op);
switch(op) // used to select particular case from the user
{
case '+': printf("sum of two numbers % 2d %2d is: %d",a,b,a+b);
break;
case '-': printf("subtraction of two numbers %2d %2d is: %d",a,b,a-b);
break;
case '*': printf("product of two numbers %2d %2d is: %d",a,b,a*b);
break;
case '/':printf("quotient of two numbers %2d %2d is:%d",a,b,a/b);
break;
case '%':printf("reminder of two numbers %2d %2d is:%d",a,b,c);
break;
default: printf("please enter correct operator");
```

```
break;
}
getch();
}
```

Output:

```
1.enter two operands:2 3
enter an operator: +
sum of two numbers 2 3 is: 5
2.enter two operands:3 4
enter an operator: -
subtraction of two numbers 3 4 is: -1
3.enter two operands:3 5
enter an operator: *
product of two numbers 3 5 is: 15
4.enter two operands:5 2
enter an operator:/
quotient of two numbers 5 2 is: 2
5. enter two operands:5 2
enter an operator: %
reminder of two numbers 5 2 is: 1.
```

Outcome:

1 .To understand basic structure of a C program.
2. To understand handling of number system in a C program

Objective 09: To find both the largest and smallest number in a list of integers

Description:

This program contains n number of elements, in these elements we can find the largest and smallest numbers and display these two numbers

Algorithm:

Step 1: start
Step 2: read n
Step 3: initialize i=0
Step 4: if i<n do as follows. If not goto step 5
Read a[i]
Increment i
Goto step 4
Step 5: min=a[0], max=a[0]
Step 6: initialize i=0
Step 7: if i<n do as follows. If not goto step 8
If a[i]<min
Assign min=a[i]
Increment igoto Step 7
Step 8: print min,max
Step 9: stop

Program:

```
#include<stdio.h>
void main()
{
int a[10],i,n,min,max;
clrscr();
printf("enter the array size:");
scanf("%d",&n);
printf("Enter the elements of array");

for(i=0;i<n;i++) // read the elements of an array
scanf("%d",&a[i]);
min=a[0];
max=a[0];
for(i=0;i<n;i++)// read the elements of an array
{
if(a[i]<min)// check the condition for minimum value
min=a[i];
if(a[i]>max)//check the condition for maximum value
max=a[i];
}
printf("maximum value is:%d\n",max);
printf("minimum value is:%d\n",min);
getch();
}
```

Output:

1.enter the array size:4
Enter the elements of array 36 13 2 45
maximum value is:45

minimum value is:2
2.enter the array size:5
Enter the elements of array 6 2 1 3 8
maximum value is:8
minimum value is:1
3.enter the array size:5
Enter the elements of array-6 9 -9 2 5
maximum value is:9
minimum value is: - 9

Outcome:

1. To understand basic structure of a C program.
2. To understand handling of number system in a C program

Objective 10: To perform the addition of two matrices

Description:

To program takes the two matrixes of same size and performs the addition an also takes the two matrixes of different sizes and checks for possibility of multiplication and perform multiplication if possible.

Algorithm:

Step 1: start
Step 2: read the size of matrices A,B – m, n
Step 3: read the elements of matrix A
Step 4: read the elements of matrix B
Step 5: select the choice for you want. If you select case 1 then goto matric addition. Else goto Step 7.
Step 6: print Sum of matrix A and B
Step 7: if you select case 2 then goto matrix multiplication
Step 8: check if n=p, if not print matrices cannot be multiplied
Step 9: Otherwise perform the multiplication of matrices
Step 10: Print the resultant matrix
Step 11: Stop

Program:

```
#include<stdio.h>
void main()
{
int ch,i,j,m,n,p,q,k,r1,c1,a[10][10],b[10][10],c[10][10];
clrscr();
printf("************************************");
printf("\n\t\tMENU");
printf("\n**********************************");
printf("\n[1]ADDITION OF TWO MATRICES");
printf("\n[2]MULTIPLICATION OF TWO MATRICES");
printf("\n[0]EXIT");
printf("\n**********************************");
printf("\n\tEnter your choice:\n");
scanf("%d",&ch);
if(ch<=2 &ch>0)
{
printf("Valid Choice\n");
}
switch(ch)
{

case 1:
printf("Input rows and columns of A & B Matrix:");
scanf("%d%d",&r1,&c1);
printf("Enter elements of matrix A:\n");
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
scanf("%d",&a[i][j]);
}
```

```
printf("Enter elements of matrix B:\n");
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
scanf("%d",&b[i][j]);
}
printf("\n =====Matrix Addition=====\n");
for(i=0;i<r1;i++)
{
For(j=0;j<c1;j++)
printf("%5d",a[i][j]+b[i][j]);
52
printf("\n");
}

break;
case 2:
printf("Input rows and columns of A matrix:");
scanf("%d%d",&m,&n);
printf("Input rows and columns of B matrix:");
scanf("%d%d",&p,&q);
if(n==p)
{
printf("matrices can be multiplied\n");
printf("resultant matrix is %d*%d\n",m,q);
printf("Input A matrix\n");
read_matrix(a,m,n);
printf("Input B matrix\n");
/*Function call to read the matrix*/
read_matrix(b,p,q);
/*Function for Multiplication of two matrices*/
printf("\n =====Matrix Multiplication=====\n");
for(i=0;i<m;++i)
for(j=0;j<q;++j)
{
c[i][j]=0;
for(k=0;k<n;++k)
c[i][j]=c[i][j]+a[i][k]*b[k][j];

}
printf("Resultant of two matrices:\n");
write_matrix(c,m,q);
}
/*end if*/
else
{
printf("Matrices cannot be multiplied.");
}
/*end else*/
break;
case 0:
```

```
printf("\n Choice Terminated");
exit();
break;
default:
printf("\n Invalid Choice");
53
}
getch();
}
/*Function read matrix*/

intread_matrix(int a[10][10],intm,int n)
{
inti,j;
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
return 0;
}
/*Function to write the matrix*/
intwrite_matrix(int a[10][10],intm,int n)
{
inti,j;
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("%5d",a[i][j]);
printf("\n");
}
return 0;
}
```

Output:

```
1.
**********************************
MENU

**********************************
[1] ADDITION OF TWO MATRICES
[2] MULTIPLICATION OF TWO MATRICES
[0] EXIT
**********************************
Enter your choice:
1
Valid Choice
Input rows and columns of A & B Matrix:2
2
Enter elements of matrix A:
222
54
2
```

```
Enter elements of matrix B:
2222
=====Matrix Addition=====
4 4
4 4
**********************************

MENU
**********************************
[1]ADDITION OF TWO MATRICES
[2]MULTIPLICATION OF TWO MATRICES
[0]EXIT
**********************************
Enter your choice:
2
Valid Choice
Input rows and columns of A matrix:2
3
Input rows and columns of B matrix:2
2
Matrices cannot be multiplied.
**********************************
MENU
**********************************
[1]ADDITION OF TWO MATRICES
[2]MULTIPLICATION OF TWO MATRICES
[0]EXIT
**********************************

Enter your choice:
2
Valid Choice
Input rows and columns of A matrix:2

2
Input rows and columns of B matrix:2
2
matrices can be multiplied
resultant matrix is 2*2
Input A matrix
222
55
2
Input B matrix
2222
=====Matrix Multiplication=====
Resultant of two matrices:
8 8
8 8
```

Outcome:

1. To understand basic structure of a C program.
2. To understand handling of number system in a C program

Objective 11: Functions to insert a sub string into given main string from a given position

Description:

In this program we need to insert a string into another string from a specified position.

Algorithm:

Step 1: start
Step 2: read main string and sub string
Step 3: find the length of main string(r)
Step 4: find length of sub string(n)
Step 5: copy main string into sub string
Step 6: read the position to insert the sub string (p)
Step 7: copy sub string into main string from position p-1
Step 8: copy temporary string into main string from position p+n-1
Step 9: print the strings
Step 10: stop

Program:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
char a[10];
char b[10];
char c[10];
int p=0,r=0,i=0;
int t=0;
intx,g,s,n,o;
clrscr();
puts("Enter First String:");
gets(a);

puts("Enter Second String:");
gets(b);
printf("Enter the position where the item has to be inserted: ");
scanf("%d",&p);
r = strlen(a);
n = strlen(b);
i=0;
// Copying the input string into another array
while(i<= r)
{
c[i]=a[i];
i++;
}s
= n+r;
o = p+n;
// Adding the sub-string
for(i=p;i<s;i++)
{
x = c[i];
if(t<n)
{
```

```
a[i] = b[t];
t=t+1;
}
a[o]=x;
o=o+1;

60
}
printf("%s", a);
getch();
}
```

Output:

1.enter first string:
computer
2.enter second string:
gec
3.enter the position where the item has to be inserted:3
comgecputer

Objective 12: To generate Pascal ‘s triangle

Description:

Pascal‘s triangle which is used for a coefficient in the equation in polynomials.

Alogrithm:

Step 1: Start
Step 2: Initialize m=0
Step 3: Read n
Step 4: If m<n goto step 5.if not goto step 12
Step 5: initialize i=40-m
Step 6: If i>0 is true do as follows. If not goto step 7
i. print white space
ii. decrementi
iii. goto Step 6
Step 7: Initialize j=0
Step 8: If j=m do as follows. If not goto Step 10
i) if(j==0||m==0)
ii) Initialize b=1 if not b=b*(m-j+1)/j
iii) Print white space, b .
iv) Goto Step 9
Step 9: increment j, goto Step 8
Step 10: print new line control
Step 11: increment m, goto step 4
Step 12: Stop

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
inti,p,r,x,binom=1;
clrscr();

printf("enter the how many lines to print");
scanf("%d",&p);
i=0;
while(i<p) // check the condition
{
for(r=40-i;r>0;r--) // perform the looping operation until 0
printf(" ");
for(x=0;x<=i;x++)
{
if((x==0)||(i==0)) // check the condition
binom=1;
else
binom=binom*(i-x+1)/x;
printf("%d",binom);
printf(" ");
}
printf("\n");
i++;
}
getch();
```

```
}
```

Output:

```
enter the how many lines to print5
        1
      1 1
    1 2 1
  1 3 3 1
1 4 6 4 1
```

Objective 13: To convert the given binary number to 2‘s complement

Description:

In this program the given binary number is first covert the numbers 0 to1 and1 to 0. And finally add the 1 to the converted number. Then we will get the2‘s complement number.

Algorithm:

main program
Step 1: Start
Step 2: declare the subprogram ―complement(char *a)‖
Step 3: initialize the variable i
Step 4: read the binary number
Step 5: perform the loop operation. if it is true then follows. if not goto step 7
i) for(i=0;a[i]!=‘\0‘;i++)
ii) if(a[i]!=‘0‘&&a[i]!=‘1‘) then displayed the number is not valid.enter the correct number.
iii) Exit the loop
Step 6: call sub program _complemt(a)‘
Step 7: stop

Sub program:

Step 1: initialize the variable I,c=0,b[160
Step 2: 1=strlen(a)
Step 3: perform the loop operation. if it is true then follows. if not goto
i)for(i=l-1;i>=0;i--)
ii)if(a[i]==‘0‘) then b[i]=‘1‘ else
iii)b[i]=‘0‘
Step 4: for(i=l-1;i>=0;i--) is true
i) if(i==l-1) then
ii) if(b[i]==‘0‘) then b[i]=‘1‘ else
iii) b[i]=‘0‘,c=1 if not goto step 5
Step 5: if(c==1&&b[i]==‘0‘) is true then

i) b[i]=‘1‘, c=0 if not goto Step 6
Step 6: if(c==1&&b[i]==‘1‘) then b[i]=‘0‘,c=1
Step 7: displayed b[l]=‘\0‘
Step 8: print b and return to main program

Program:

```
#include <stdio.h>
#include<conio.h>
void complement (char *a);
void main()
{
char a[16];
inti;
clrscr();
printf("Enter the binary number");
gets(a);
for(i=0;a[i]!='\0'; i++)
{
if (a[i]!='0' && a[i]!='1')
{
printf("The number entered is not a binary number. Enter the
correct number");
exit(0);
}
} complement(a);
```

```
getch();
A
If c==1
&&
b[i]==1

B[i]='0]
C=1
true
B
C
B[i]='\0'
Print _b'
Return to
main
program
93
}
void complement (char *a)
{
int l, i, c=0;

char b[16];
l=strlen(a);
for (i=l-1; i>=0; i--)
{
if (a[i]=='0')
b[i]='1';
else
b[i]='0';
}
for(i=l-1; i>=0; i--)
{
if(i==l-1)
{
if (b[i]=='0')
b[i]='1';
else
{
b[i]='0';
c=1;
}
}
else
{
if(c==1 && b[i]=='0')

{
b[i]='1';
c=0;
}
```

```
else if (c==1 && b[i]=='1')
{
b[i]='0';
c=1;
}
}
}
b[l]='\0';
printf("The 2's complement is %s", b);
}
```

Output:

```
1.Enter the binary number101010
The 2's complement is 010110
Enter the binary number11111
The 2's complement is 00001
Enter the binary number2222
The number entered is not a binary number. Enter the correct number
```

Objective 14: To read the two complex numbers and perform the addition and multiplication of these two numbers.

Description:

In this program the complex number means it contains the two parts. first one is real part and second one is imaginary part(2+3i).by taking these two complex numbers we can perform the addition and multiplication operation.

Algorithm:

Step 1: Start
Step 2: declare structure for complex numbers
Step 3: read the complex number
Step 4: read choice
Step 5: if choice=1 then addition operation will perform and it contains following steps
i) w.realpart = w1.realpart+w2.realpart;
ii) w.imgpart = w1.imgpart+w2.imgpart; goto step 4
Step 6: if choice=2 then multiplication operation will perform and it contains following steps
i) w.realpart=(w1.realpart*w2.realpart)-(w1.imgpart*w2.imgpart);
ii) w.imgpart=(w1.realpart*w2.imgpart)+(w1.imgpart*w2.realpart); goto step 4
Step 7: if choice=0 then exit operation will perform
Step 8:ifw.imgpart>0 then print realpart+imgpart else
Print realpart.
Step 9: Stop

Program:

```
#include<stdio.h>
#include<math.h>
void arithmetic (int opern);
struct comp
{

doubl erealpart;
doubl eimgpart;
};
void main()
{
intopern;
clrscr();
printf("\n\n \t\t\t***** MAIN MENU *****");
printf("\n\n Select your option: \n 1 : ADD\n 2 : MULTIPLY\n 0 : EXIT \n\n\t\t Enter your Option [ ]\b\b");
scanf("%d",&opern);
if(opern>2)
{
printf("invalid option");
}
else
{
switch(opern)
{
case 0:
```

```
exit(0);
case 1:
case 2:
arithmetic(opern);

default:
main();
}
}
getch();
}
void arithmetic(intopern)
{
struct comp w1, w2, w;
printf("\n Enter two Complex Numbers (x+iy):\n Real Part of First Number:");
scanf("%lf",&w1.realpart);
printf("\n Imaginary Part of First Number:");
scanf("%lf",&w1.imgpart);
printf("\n Real Part of Second Number:");
scanf("%lf",&w2.realpart);
printf("\n Imaginary Part of Second Number:");
scanf("%lf",&w2.imgpart);
switch(opern)
{
/*addition of complex number*/
case 1:

w.realpart = w1.realpart+w2.realpart;
w.imgpart = w1.imgpart+w2.imgpart;
break;
/*multiplication of complex number*/
case 2:
w.realpart=(w1.realpart*w2.realpart)-(w1.imgpart*w2.imgpart);
w.imgpart=(w1.realpart*w2.imgpart)+(w1.imgpart*w2.realpart);
break;
}
if (w.imgpart>0)
printf("\n Answer = %lf+%lfi",w.realpart,w.imgpart);
else
printf("\n Answer = %lf%lfi",w.realpart,w.imgpart);
getch();
main();
}
```

Output:

```
***** MAIN MENU *****
Select your option:
1: ADD
```

```
2: MULTIPLY
0: EXIT
Enter your Option [ 1]
Enter two Complex Numbers (x+iy):
Real Part of First Number:2
Imaginary Part of First Number:2
Real Part of Second Number:2
Imaginary Part of Second Number:2
Answer = 4.000000+4.000000i
```

Objective 15: Program which copies one file to another

Description:

In this program we have to use the file functions to perform the copy operation from one file to another file.

Algorithm:

Step 1: Start
Step 2: read command line arguments
Step 3: check if no of arguments =3 or not. If not print invalid no of arguments
Step 4: open-source file in read mode
Step 5: if NULL pointer, then print source file cannot be open
Step 6: open destination file in write mode
Step 7: if NULL pointer, then print destination file cannot be open
Step 8: read a character from source file and write to destination file until EOF
Step 9: Close source file and destination file
Step 10: Stop

Program:

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
void main(intargc, char *argv[])
{
FILE *fs,*ft;
charch;
clrscr();
if(argc!=3)
{
puts("Invalid number of arguments.");
exit(0);

}
fs = fopen(argv[1],"r");
if(fs==NULL)
{
puts("Source file cannot be opened.");
exit(0);
}
ft = fopen(argv[2],"w");
if (ft==NULL) // check the condition if the file pointer is NULL or not
{
puts("Target file cannot be opened.");
fclose(fs);
exit(0);
}
while(1)
{
ch=fgetc(fs);
if (ch==EOF) // check the condition if the file is end or not
break;
else
```

```
fputc(ch,ft);
}
fclose(fs);
fclose(ft);
getch();

}
```

Output:

```
source.c
this is source text
ouput.c
Command line arguments
source.couput.c
source.c
this is source text
ouput.c
this is source text
Command line arguments
source.c
Invalid number of arguments.
```

Outcome:

1. Construct programs that demonstrate effective use of file handling in C language.
2. To understand various file related functions.

DIGITAL ELECTRONIC AND LOGIC LABORATORY MAUAL

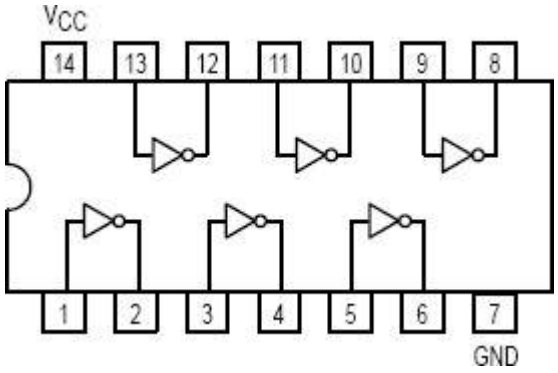
(Course Code: BCA-1197)

CONTENTS

Experiment No

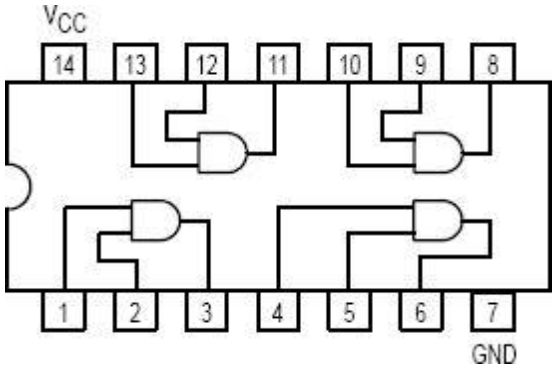
1. ***Verification of Gates***
2. ***Half/Full Adder/Subtractor***
3. ***Parallel Adder/Subtractor***
4. ***Excess-3 to BCD & Vice Versa***
5. ***Binary-Grey & Grey-Binary Converter***
6. ***MUX/DEMUX***
7. ***MUX/DEMUX using only NAND Gates***
8. ***Comparators***
9. ***Encoder/Decoder***
10. ***Flip-Flops***
11. ***Counters***
12. ***Shift Registers***
13. ***Johnson/Ring Counters***
14. ***Sequence Generator***
15. ***Multivibrators***
16. ***Static RAM***

Inverter Gate (NOT Gate) 7404LS



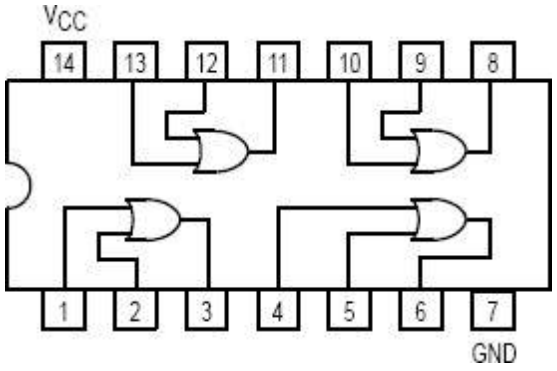
A	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)	Y5 (V)	Y6 (v)
0	1						
1	0						

2-Input AND Gate 7408LS



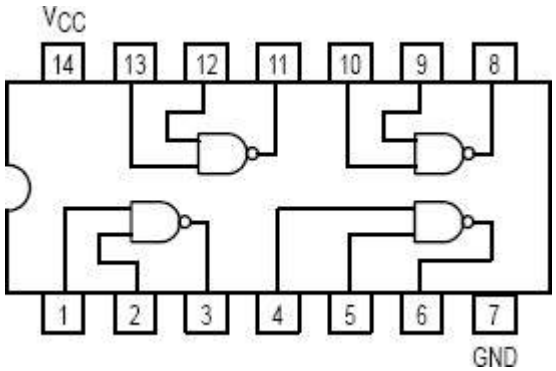
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	0				
1	0	0				
1	1	1				

2-Input OR Gate 7432LS



A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	0				
1	0	0				
1	1	1				

2-Input NAND Gate 7400LS



A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

Experiment No: Date:__/__/

VERIFICATION OF GATES

Aim: - To study and verify the truth table of logic gates

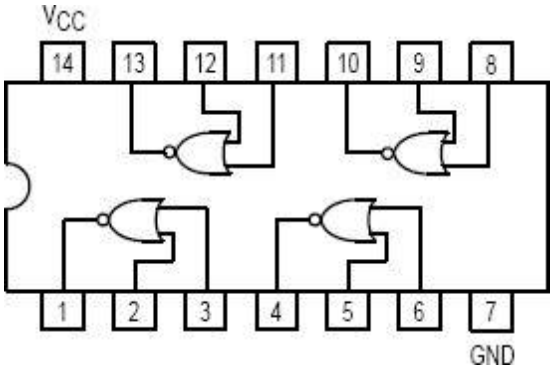
Apparatus Required: -

All the basic gates mention in the fig.

Procedure: -

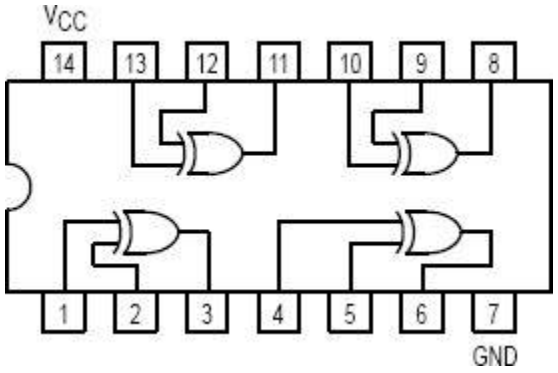
1. **Place the IC-on-IC Trainer Kit.**
2. **Connect V_{CC} and ground to respective pins of IC Trainer Kit.**
3. **Connect the inputs to the input switches provided in the IC Trainer Kit.**
4. **Connect the outputs to the switches of O/P LEDs,**
5. **Apply various combinations of inputs according to the truth table and observe condition of LEDs.**
6. **Disconnect output from the LEDs and note down the corresponding multimeter voltage readings for various combinations of inputs.**

2-Input NOR Gate 7402LS



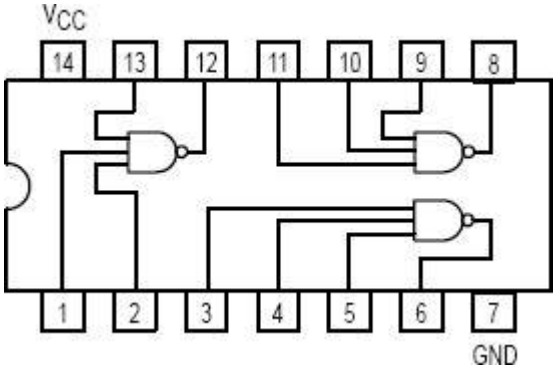
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

2 Input EX-OR Gate 7486LS



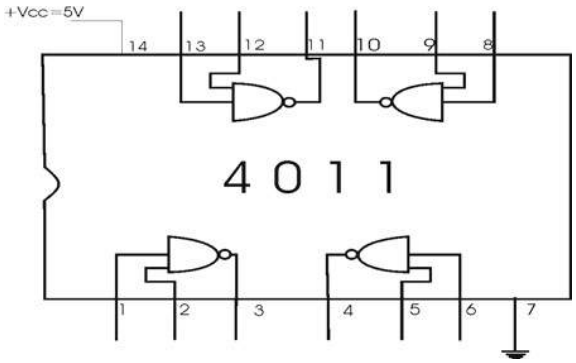
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	0				
0	1	1				
1	0	1				
1	1	0				

3 Input NAND Gate 7410LS



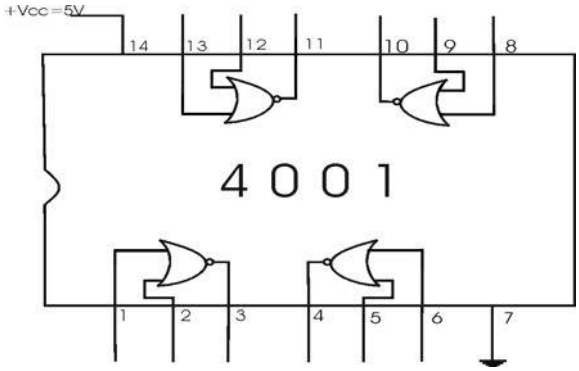
A	B	C	O/P	Y1 (V)	Y2 (V)	Y3 (V)
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	0			

2-Input NAND Gate CD4011



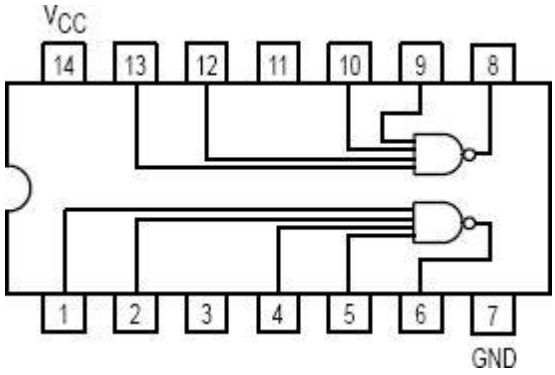
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	1				
1	0	1				
1	1	0				

2-Input NOR Gate ← CD4001



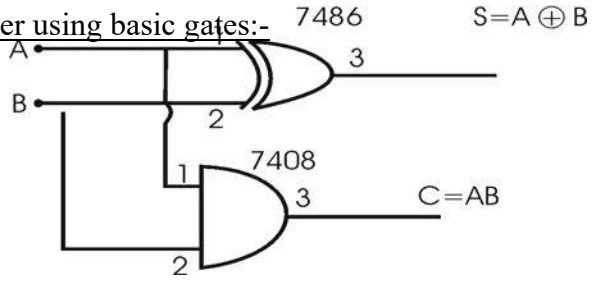
A	B	O/P	Y1 (V)	Y2 (V)	Y3 (V)	Y4 (V)
0	0	1				
0	1	0				
1	0	0				
1	1	0				

4-Input NAND Gate 7420LS

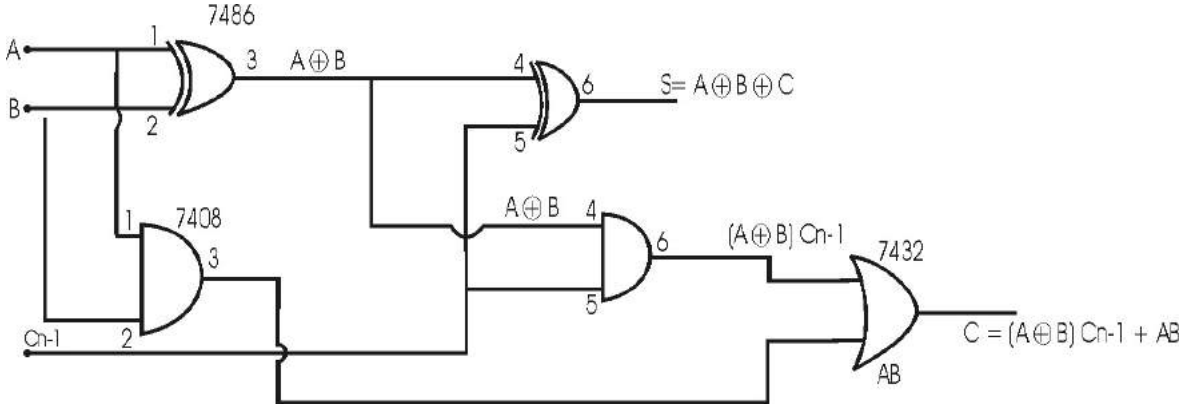


A	B	C	D	O/P	Y1 (V)	Y2 (V)	Y3 (V)
0	0	0	0	1			
0	0	0	1	1			
0	0	1	0	1			
0	0	1	1	1			
0	1	0	0	1			
0	1	0	1	1			
0	1	1	0	1			
0	1	1	1	1			
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	1			
1	0	1	1	1			
1	1	0	0	1			
1	1	0	1	1			
1	1	1	0	1			
1	1	1	1	0			

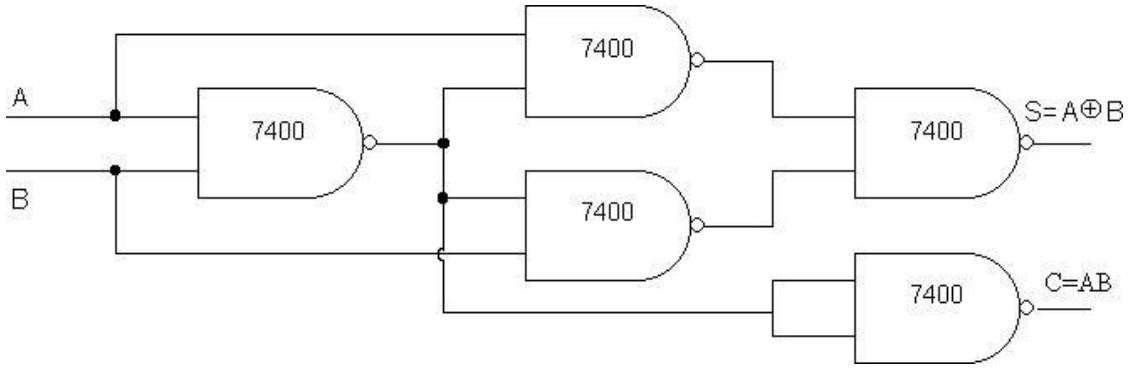
Half Adder using basic gates:-



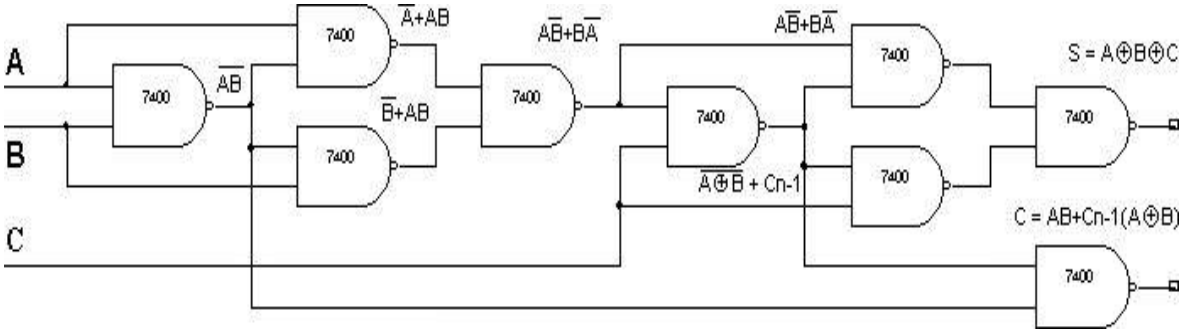
Full Adder using basic gates:-



Half Adder using NAND gates only:-



Full Adder using NAND gates only:-



Experiment No: Date:__/__/

HALF/FULL ADDER & HALF/FULL SUBTRACTOR

Aim: - To realize half/full adder and half/full subtractor.

i. **Using X-OR and basic gates**

ii. **Using only nand gates.**

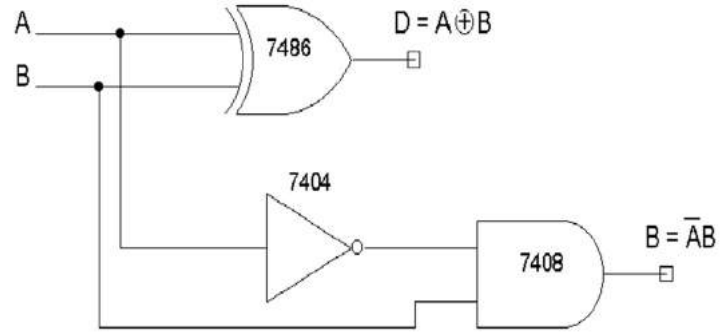
Apparatus Required: -

IC 7486, IC 7432, IC 74 08, IC 7400, etc.

Procedure: -

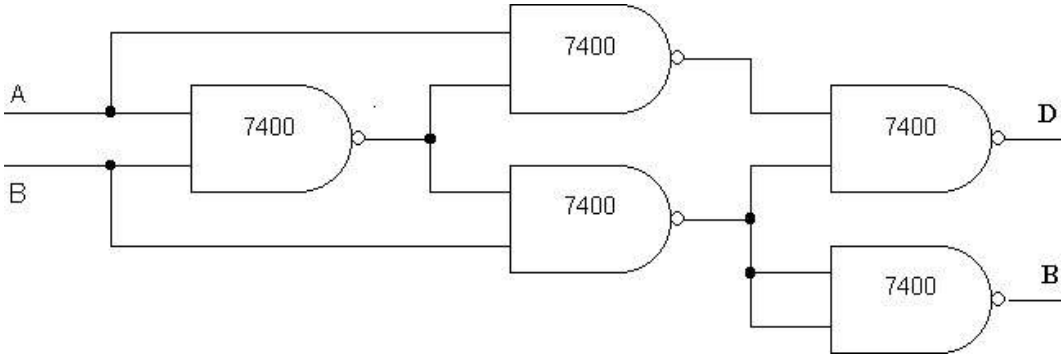
1. **Verify the gates.**
2. **Make the connections as per the circuit diagram.**
3. **Switch on V_{CC} and apply various combinations of input according to the truth table.**
4. **Note down the output readings for half/full adder and half/full subtractor sum/difference and the carry/borrow bit for different combinations of inputs.**

Using X – OR and Basic Gates (a)Half Subtractor

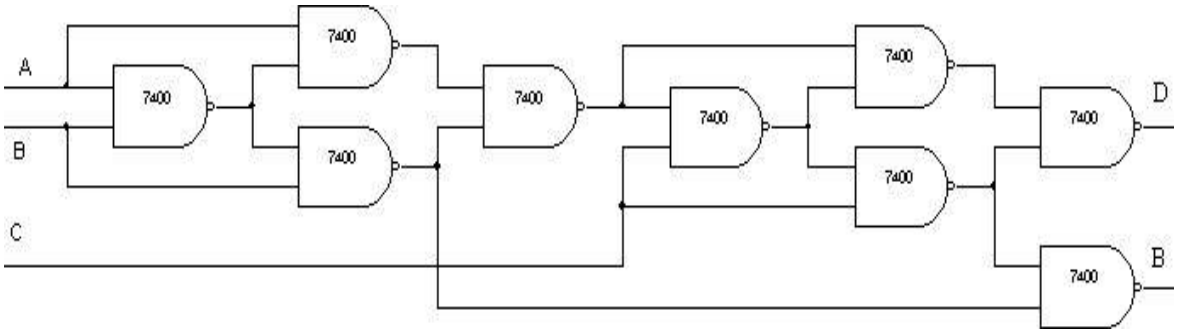


Full Subtractor

(i) Using only NAND gates (a) Half subtractor



(b) Full Subtractor



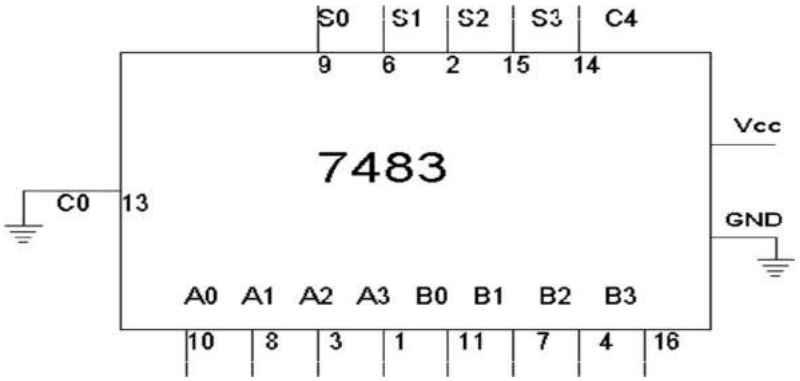
Half Adder					
A	B	S	C	S(V)	C(V)
	0	0	0		
0	1	1	0		
1	0	1	0		
1	1	0	1		

Half Subtractor					
A	B	D	B	D(V)	B(V)
0	0	0	0		
0	1	1	1		
1	0	1	0		
1	1	0	0		

Full Adder						
A	B	Cn-1	S	C	S(V)	C(V)
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		

Full Subtractor						
A	B	Cn-1	D	B	D(v)	B(v)
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		

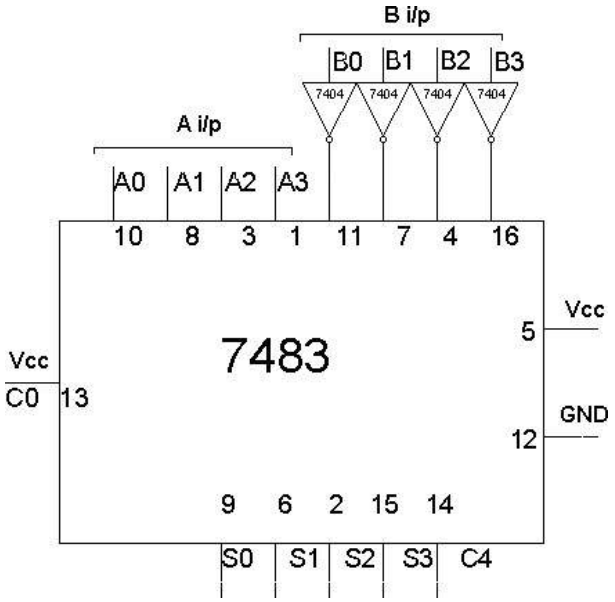
Adder : -



Truth Table: -

A3	A2	A1	A0	B3	B2	B1	B0	C4 (V)	S3(V)	S2(V)	S1(V)	S0(V)
0	0	0	1	0	0	1	0	0	0	0	1	1
0	1	0	1	1	0	1	1	1	1	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	0	0	1	1	0	1	0	1	0

Subtractor:-



Experiment No: Date:__/__/

PARALLEL ADDER AND SUBTRACTOR USING 7483

Aim: - To realize IC7483 as parallel adder / Subtractor.

Apparatus Required: -

IC 7483, IC 7404, etc.

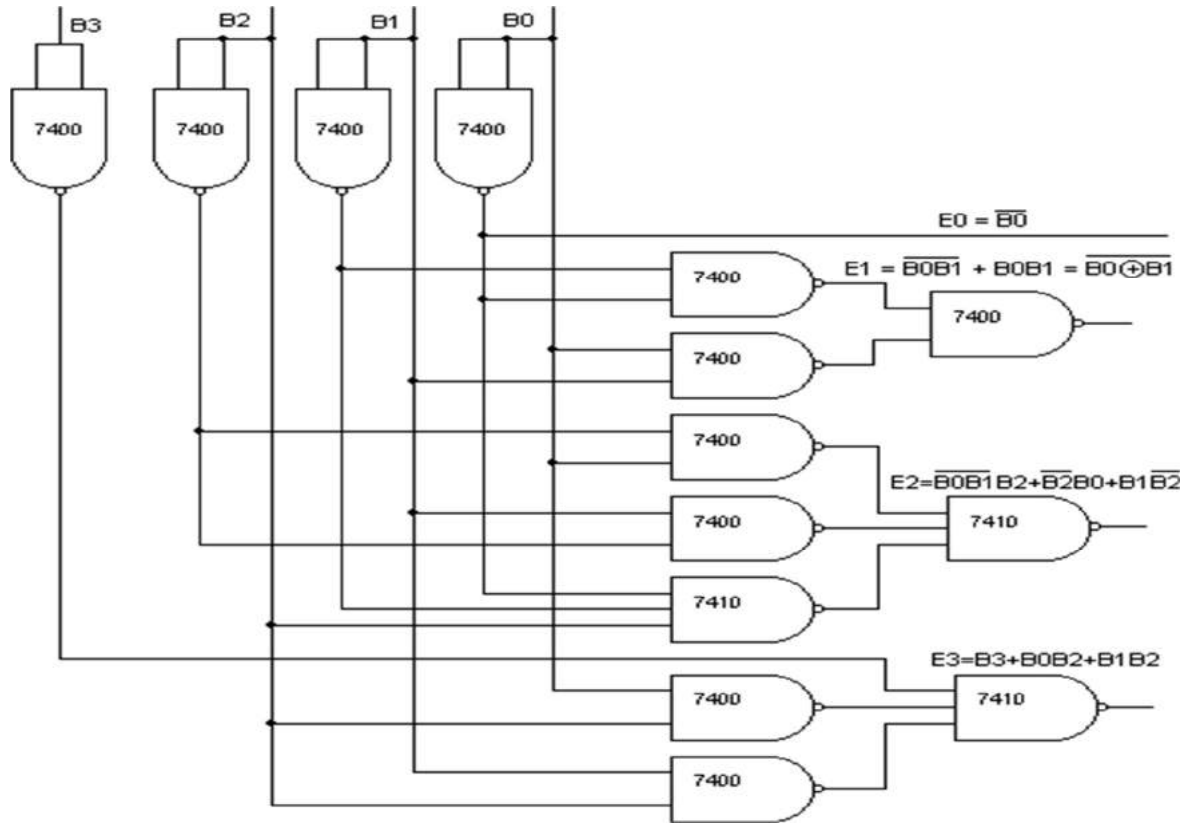
Procedure: -

1. **Apply the inputs to A0 to A3 and B0 to B3.**
2. **Connect C0 to the Ground.**
3. **Check the output sum on the S0 to S3 and also C4.**
4. **For subtraction connect C0 to Vcc, Apply the B input through NOT gate, which gives the complement of B.**
5. **The truth table of adder and Subtractor are noted down.**

Truth Table for Subtractor

A3	A2	A1	A0	B3	B2	B1	B0	C4(V)	S3(V)	S2(V)	S1(V)	S0(V)
0	0	1	0	0	0	0	1	1	0	0	0	1
0	1	0	1	0	0	1	1	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	1	0
1	0	1	0	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	1	0	1	0	0	1
1	0	1	0	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	1	0	1	0	0	1

BCD To Excess-3



Truth Table For Code Conversion: -

Inputs				Outputs			
B3	**B2**	**B1**	**B0**	**E3 (v)**	**E2 (v)**	**E1 (v)**	**E0 (v)**
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**	**0**	**1**
0	**0**	**1**	**1**	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**	**1**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**	**0**	**0**
0	**1**	**1**	**0**	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**0**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**0**	**0**

Experiment No: Date:__/__/____

BCD to Excess 3 AND Excess 3 to BCD

Aim: - To verify BCD to excess –3 code conversion using NAND gates. To study and verify the truth table of excess-3 to BCD code converter

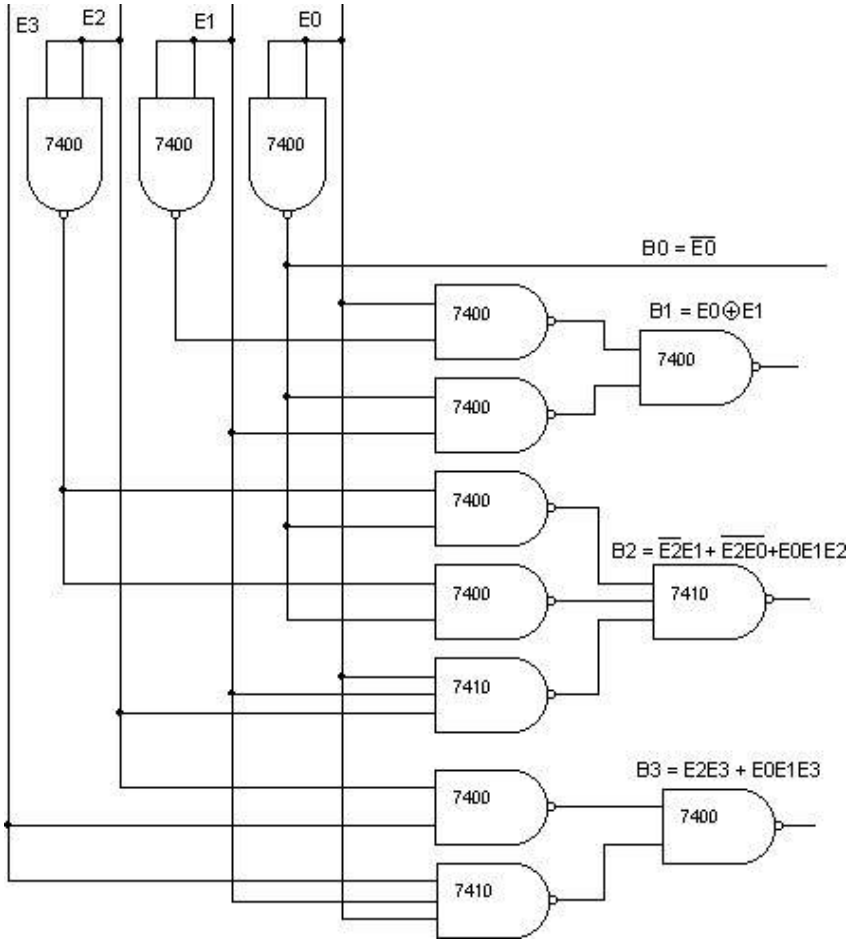
Apparatus Required: -

IC 7400, IC 7404, etc.

Procedure: - (BCD Excess 3 and Vice Versa)

1. **Make the connections as shown in the fig.**
2. **Pin [14] of all IC'S are connected to +5V and pin [7] to the ground.**
3. **The inputs are applied at E3, E2, E1, and E0 and the corresponding outputs at B3, B2, B1, and B0 are taken for excess – 3 to BCD.**
4. **B3, B2, B1, and B0 are the inputs, and the corresponding outputs are E3, E2, E1 and E0 for BCD to excess – 3.**
5. **Repeat the same procedure for other combinations of inputs.**
6. **Truth table is written.**

Excess-3 To BCD :-



Truth Table For Code Conversion: -

Inputs				Outputs			
E3	**E2**	**E1**	**E0**	**B3 (v)**	**B2 (v)**	**B1 (v)**	**B0(v)**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**0**	**1**	**0**	**1**
1	**0**	**0**	**1**	**0**	**1**	**1**	**0**
1	**0**	**1**	**0**	**0**	**1**	**1**	**1**
1	**0**	**1**	**1**	**1**	**0**	**0**	**0**
1	**1**	**0**	**0**	**1**	**0**	**0**	**1**

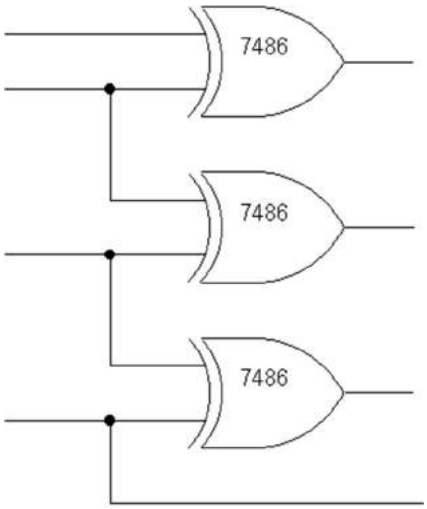
Exercise: -

1. **Obtain the expression for E3, E2, E1 and E0**

2. **Obtain the expression for B3, B2, B1 and B0**

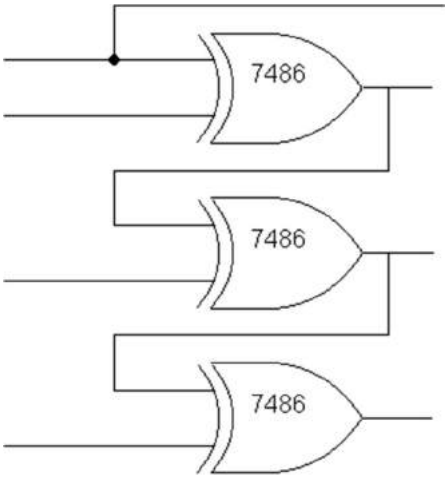
Circuit Diagram: -

Binary To Gray



Using EX-OR gates

Gray To Binary



Using EX-OR gates

Truth Table For Both: -

Inputs				Outputs			
B3	**B2**	**B1**	**B0**	**G3 (V)**	**G2 (V)**	**G1 (V)**	**G0 (V)**
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**

Experiment No: Date:__/__/____

BINARY TO GRAY AND GRAY TO BINARY

CONVERSION

Aim: - To convert given binary numbers to gray codes.

Apparatus Required: -

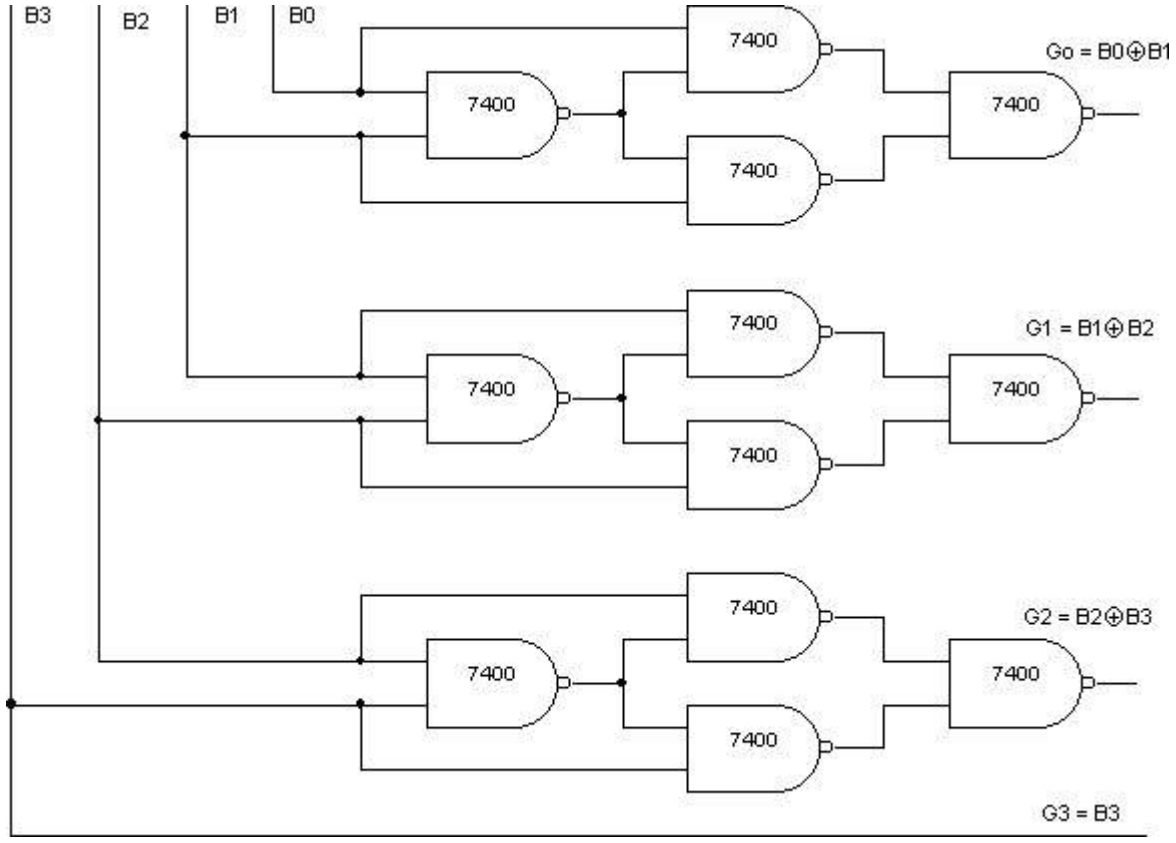
IC 7486, etc

Procedure: -

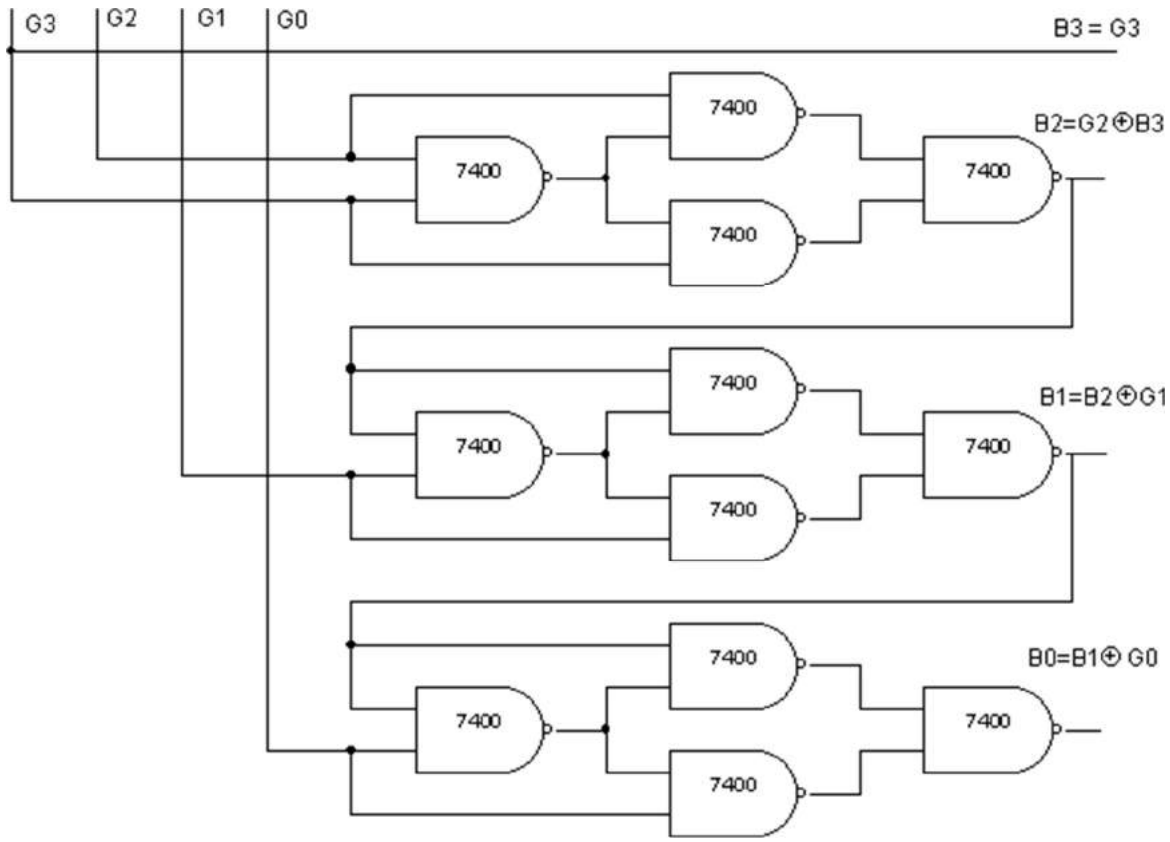
1. **The circuit connections are made as shown in fig.**
2. **Pin (14) is connected to +Vcc and Pin (7) to ground.**
3. **In the case of binary to gray conver sion, the inputs B0, B1, B2 and B3 are given at respective pins and outputs G0, G1, G2, G3 are taken for all the 16 combinations of the input.**
4. **In the case of gray to binary conver sion, the inputs G0, G1, G2 and G3 are given at respective pins and outputs B0, B1, B2, and B3 are taken for all**

   the 16 combinations of inputs.
5. **The values of the outputs are tabulated.**

Using Nand Gates
Only: - Binary To Gra



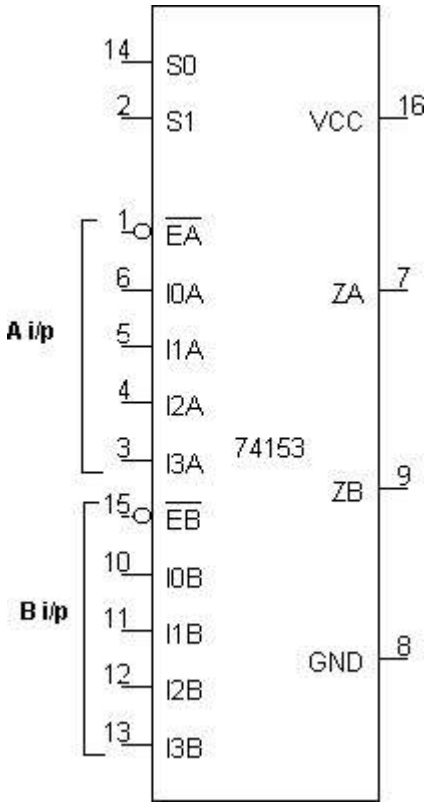
Gray Code



Truth Table For Both: -

Inputs				Outputs			
B3	**B2**	**B1**	**B0**	**G3 (V)**	**G2 (V)**	**G1 (V)**	**G0 (V)**
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**

Pin Details: -



Truth Table: -

CHANNEL – A							
INPUTS					SELECT LINES		O/P
Ēa	Ioa	I1a	I2a	I3a	S1	S2	Za(v)
1	X	X	X	X	X	X	0
0	0	X	X	X	0	0	0
0	1	X	X	X	0	0	1
0	X	0	X	X	0	1	0
0	X	1	X	X	0	1	1
0	X	X	0	X	1	0	0
0	X	X	1	X	1	0	1
0	X	X	X	0	1	1	0
0	X	X	X	1	1	1	1

CHANNEL – B							
INPUTS					SELECT LINES		O/P
Ēa	Iob	I1b	I2b	I3b	S1	S2	Za(v)
1	X	X	X	X	X	X	0
0	0	X	X	X	0	0	0
0	1	X	X	X	0	0	1
0	X	0	X	X	0	1	0
0	X	1	X	X	0	1	1
0	X	X	0	X	1	0	0
0	X	X	1	X	1	0	1
0	X	X	X	0	1	1	0
0	X	X	X	1	1	1	1

Experiment No: Date: __/__/

MUX/DEMUX USING 74153 & 74139

Aim: - To verify the truth table of multiplexer using 74153 & to verify a demultiplexer using 74139. To study the arithmetic circuits half-adder half Subtractor, full adder and full Subtractor using multiplexer.

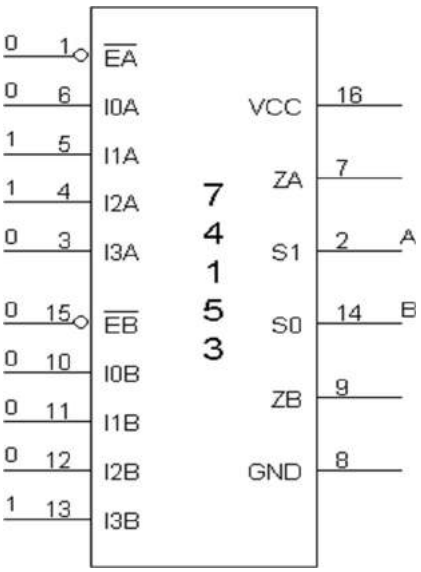
Apparatus Required: -

IC 74153, IC 74139, IC 7404, etc.

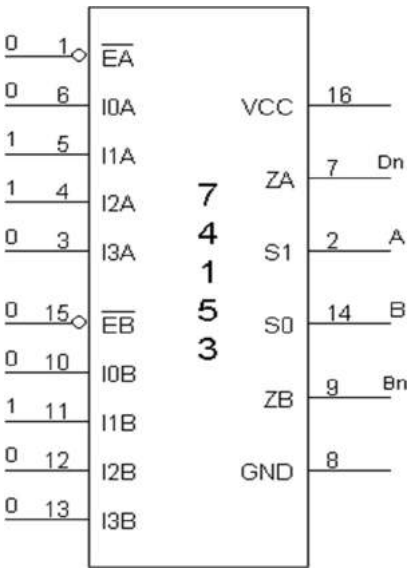
Procedure: - (IC 74153)

1. **The Pin [16] is connected to + Vcc.**
2. **Pin [8] is connected to ground.**
3. **The inputs are applied either to 'A' input or 'B' input.**
4. **If MUX 'A' has to be initialized, Ea is made low and if MUX 'B' has to be initialized, E_b is made low.**
5. **Based on the selection lines one of the inputs will be selected at the output and thus the truth table is verified.**
6. **In case of half adder using MUX, sum and carry is obtained by applying a constant inputs at I_{0a}, I_{1a}, I_{2a}, I_{3a} and I_{0b}, I_{1b}, I_{2b} and I_{3b} and the corresponding values of select lines are changed as per table and the output is taken at Z0a as sum and Z0b as carry.**
7. **In this case, the channels A and B are kept at constant inputs according to the table and the inputs A and B are varied. Making Ea and Eb zero and the output is taken at Za, and Zb.**
8. **In full adder using MUX, the input is applied at Cn-1, An and Bn. According to the table corresponding outputs are taken at Cn and Dn.**

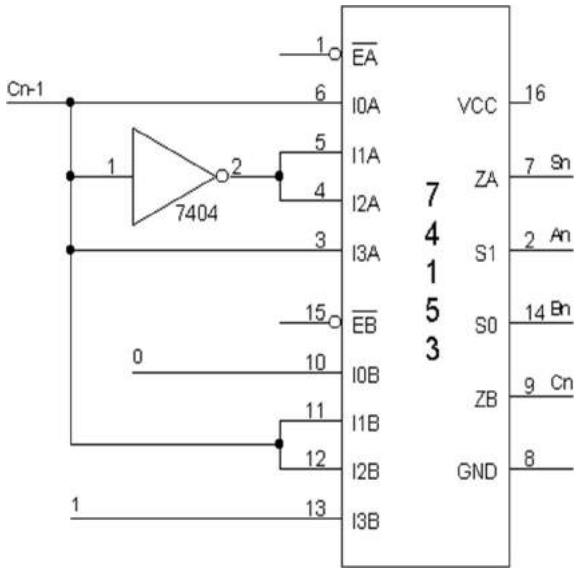
Half Adder Using 74153 –



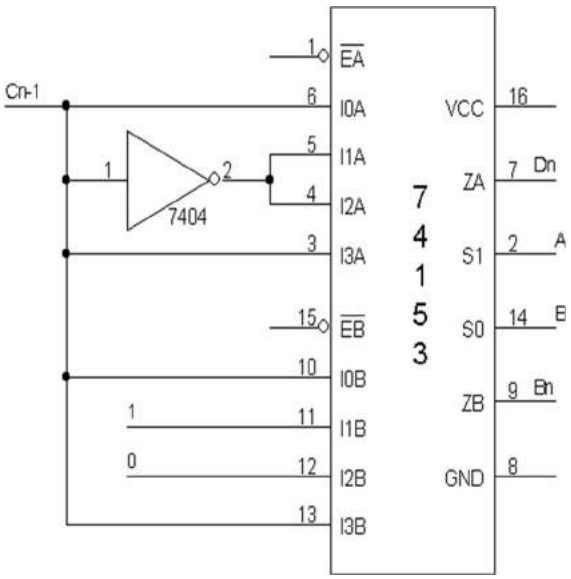
Half Subtractor: -



Full Adder Using 74153: -



Full Subtractor Using 74153: -



Truth Tables: - Same for both Subtractor and adder

Half adder/subtractor			
A	**B**	**Sn/Dn (V)**	**Cn/Bn (V)**
0	**0**		
0	**1**		
1	**0**		
1	**1**		

Full Adder/subtractro				
An	**Bn**	**Cn-1**	**Sn/Dn (V)**	**Cn/Bn (V)**
0	**0**	**0**		
0	**0**	**1**		
0	**1**	**0**		
0	**1**	**1**		
1	**0**	**0**		
1	**0**	**1**		
1	**1**	**0**		
1	**1**	**1**		

Pin Details: -

Truth Table For Demux: -

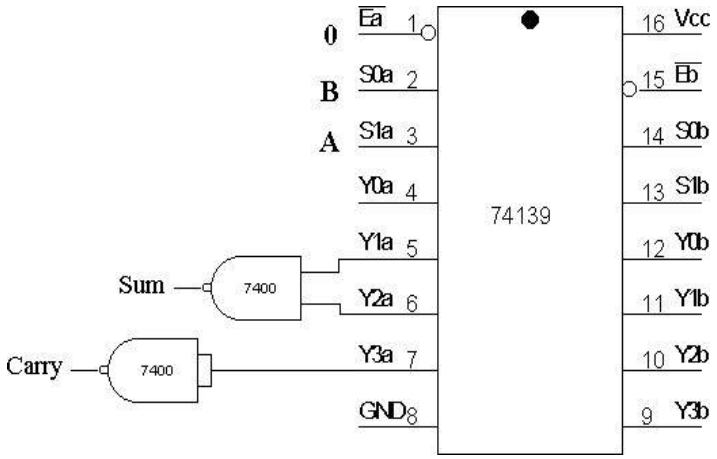
CHANNEL – A						
Inputs			Outputs			
Ēa	S1a	S0a	Y0a	Y1a	Y2a	Y3a
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

CHANNEL – B						
Inputs			Outputs			
Ēb	S1b	S0b	Y0b	Y1b	Y2b	Y3b
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Procedure: - (IC 74139)

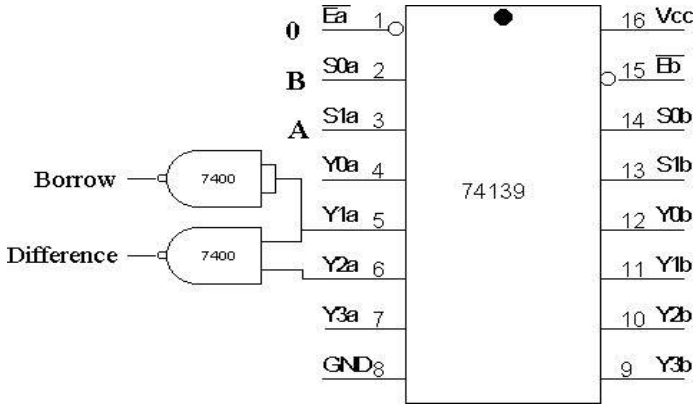
1. **The inputs are applied to either 'a' input or 'b' input**
2. **The demux is activated by making Ea low and Eb low.**
3. **The truth table is verified.**

Half adder



Half Adder			
A	B	Sn (V)	Cn (V)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half subtractor:-

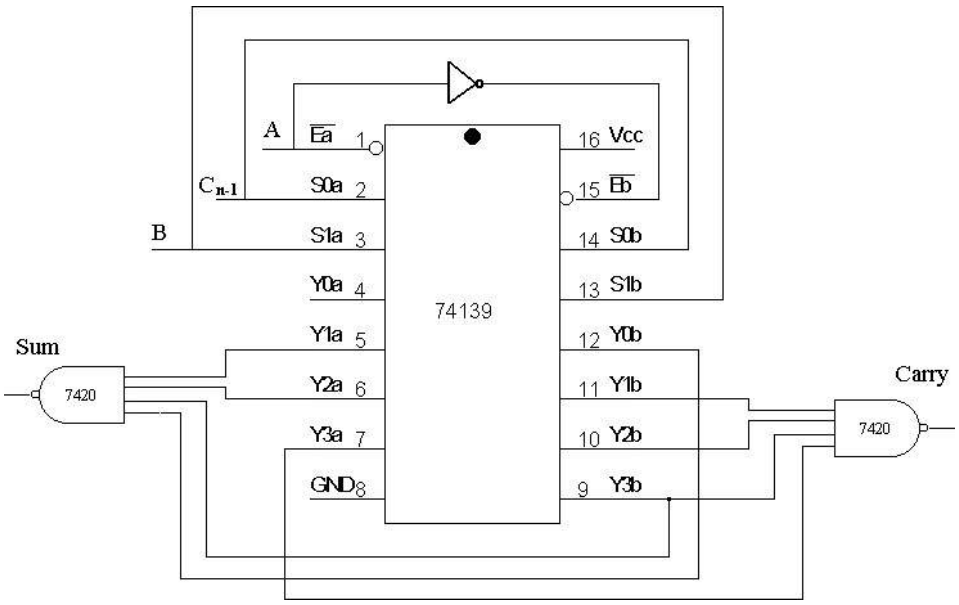


Half Subtractor			
A	B	Dn (V)	Bn (V)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

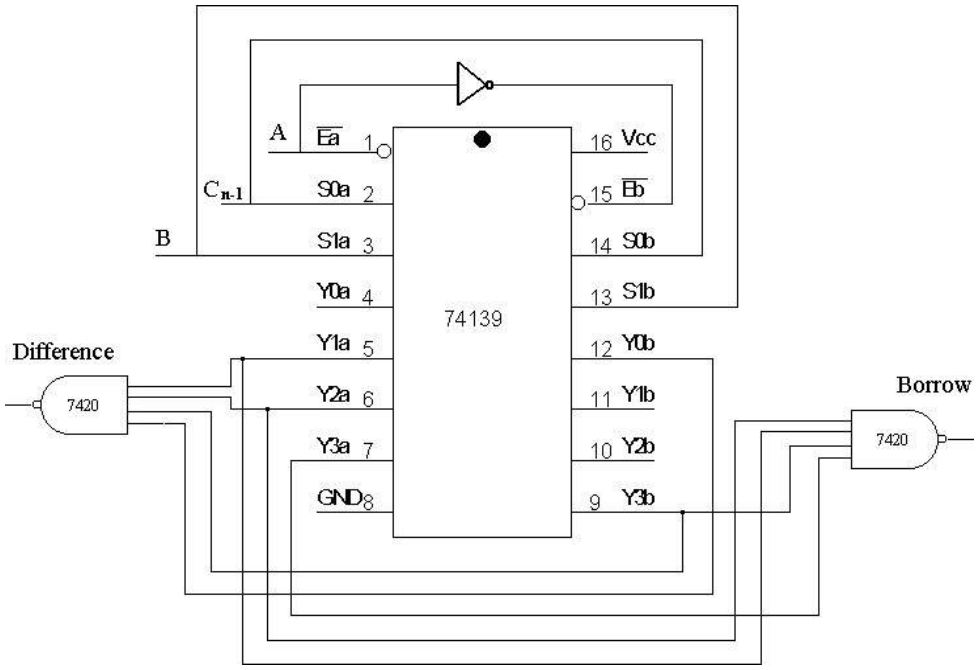
Exercise:-

- **Repeat the experiment to verify**

Channel B. Full Adder using IC 74139:-



Full subtractor using IC 74139:-

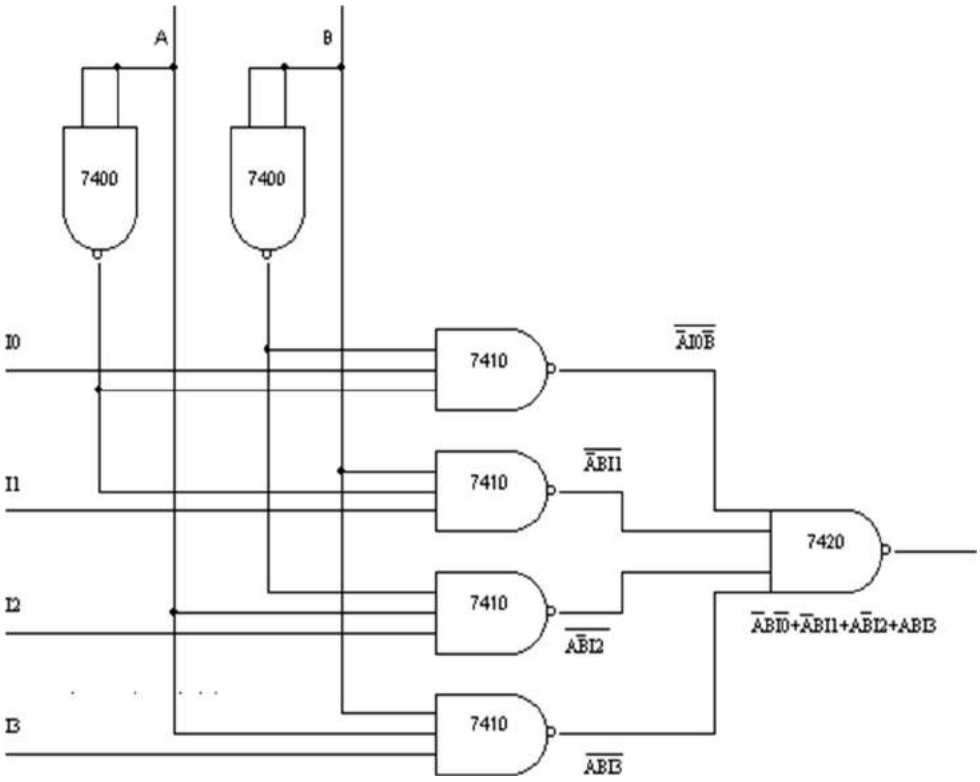


Truth Tables:-

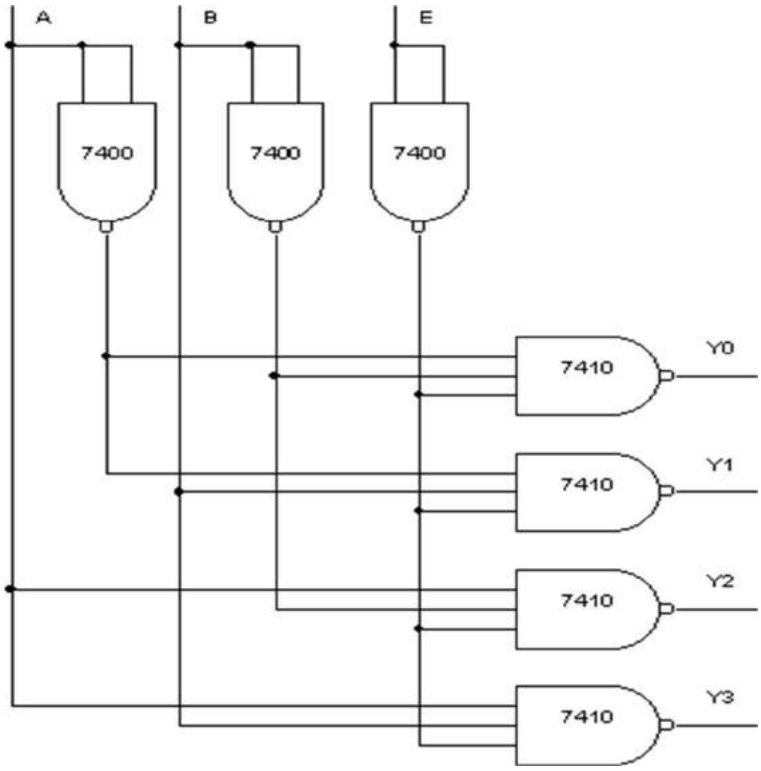
Full Adder				
An	**Bn**	**Cn-1**	**Sn (V)**	**Cn (V)**
0	**0**	**0**		
0	**0**	**1**		
0	**1**	**0**		
0	**1**	**1**		
1	**0**	**0**		
1	**0**	**1**		
1	**1**	**0**		
1	**1**	**1**		

Full Subtractor				
An	**Bn**	**Cn-1**	**Dn (V)**	**Bn (V)**
0	**0**	**0**		
0	**0**	**1**		
0	**1**	**0**		
0	**1**	**1**		
1	**0**	**0**		
1	**0**	**1**		
1	**1**	**0**		
1	**1**	**1**		

MUX USING NAND GATES ONLY: -



DEMUX USING NAND GATES ONLY: -



Experiment No: DATE: __/__/

MUX AND DEMUX USING NAND GATES

AIM: - To verify the truth table of MUX and DEMUX using NAND.

APPARATUS REQUIRED: -

IC 7400, IC 7410, IC 7420, etc.

PROCEDURE: -

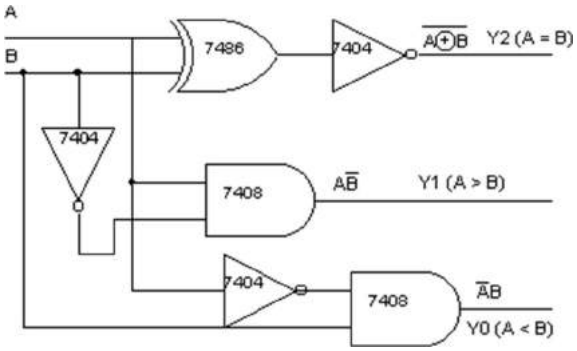
1. **Connections are made as shown in the Circuit diagram.**
2. **Change the values of the inputs as per the truth table and note down the outputs readings using multimeter.**

TRUTH TABLES: -

INPUT						OUPUT
A	B	I0	I1	I2	I3	Y (V)
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

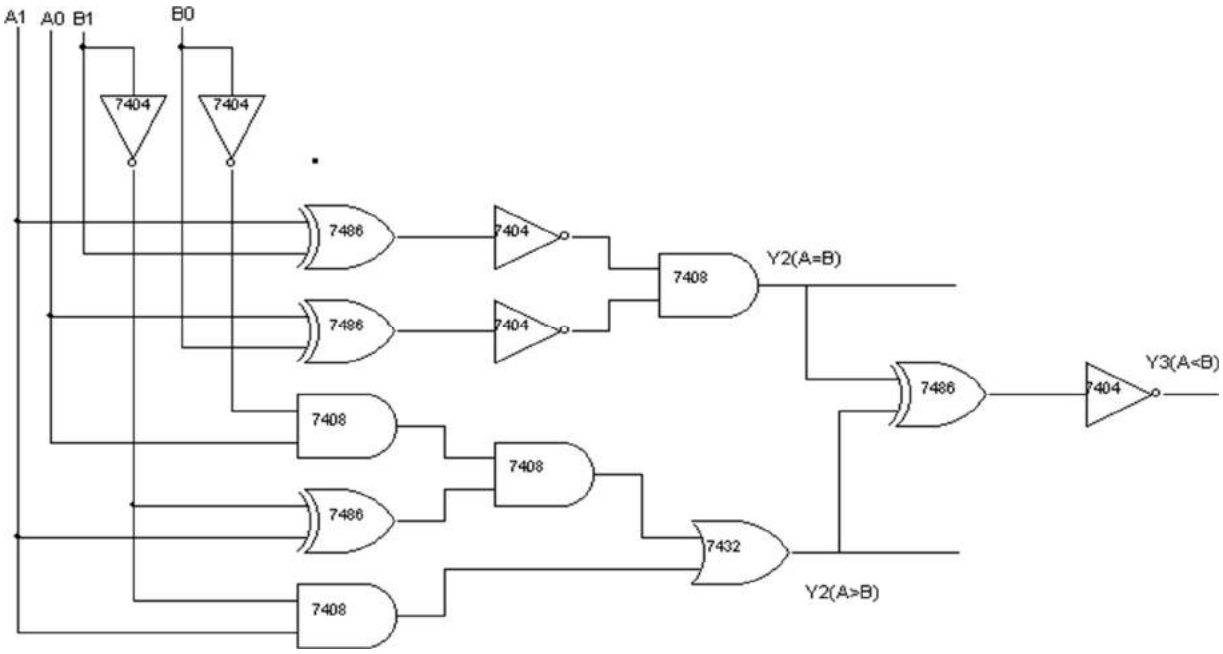
INPUT			OUPUT			
$\bar{E}$	A	B	Y0 (V)	Y1 (V)	Y2 ()	Y3 (V)
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

One Bit Comparator: -



A	B	Y1 (A>B)	Y2 (A = B)	Y3 (A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Two Bit Comparator: -



Two-Bit Comparator: -

A1	A0	B1	B0	Y1 (A > B)	Y2 (A = B)	Y3 (A < B)
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Experiment No: Date:__/__/____

COMPARATORS

Aim: - To verify the truth t able of one bit and two bit comparators using logic gates.

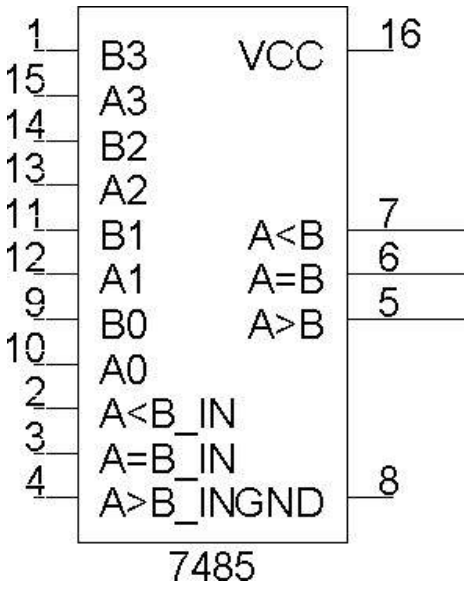
Apparatus Required: -

IC 7486, IC 7404, IC 7408, et c.

Procedure: -

1. **Verify the gates.**
2. **Make the connections as per the circuit diagram.**
3. **Switch on Vcc.**
4. **Applying i/p and Check for the outputs.**
5. **The voltameter readings of outputs are taken and tabulated in tabular column.**
6. **The o/p are verified.**

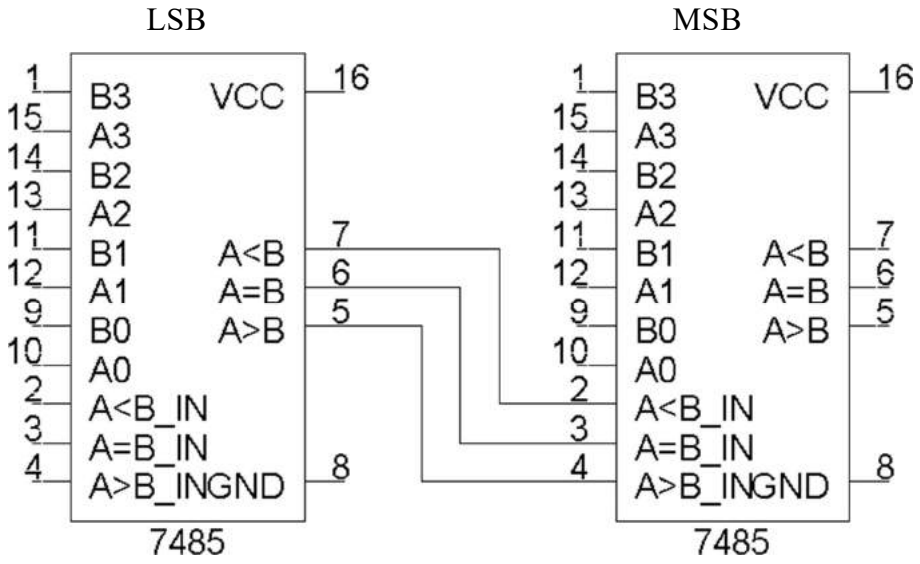
2- bit Comparator



Tabular Coloumn For 8-Bit Comparator: -

A_3 B_3	**A_2 B_2**	**A_1 B_1**	**A_0 B_0**	**A>B**	**A=B**	**A<B**	**A>B**	**A=B**	**A<B**
$A_3>B_3$	**X**	**X**	**X**	**X**	**X**	**X**			
$A_3<B_3$	**X**	**X**	**X**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2>B_2$**	**X**	**X**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2<B_2$**	**X**	**X**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1>B_1$**	**X**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1<B_1$**	**X**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1=B_1$**	**$A_0>B_0$**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1=B_1$**	**$A_0<B_0$**	**X**	**X**	**X**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1=B_1$**	**$A_0=B_0$**	**1**	**0**	**0**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1=B_1$**	**$A_0=B_0$**	**0**	**1**	**0**			
$A_3=B_3$	**$A_2=B_2$**	**$A_1=B_1$**	**$A_0=B_0$**	**0**	**0**	**1**			

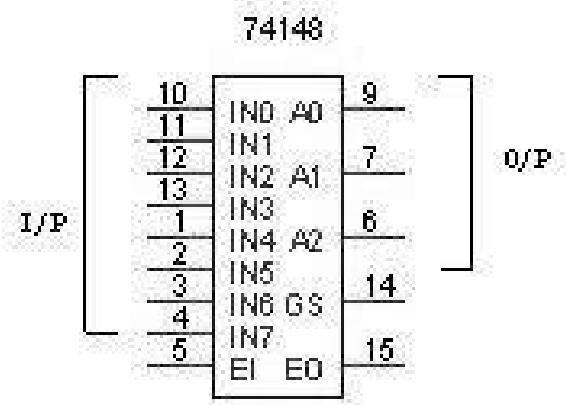
8-Bit Comparator: -



Exercise:-

- **Write the truth table for 8-bit comparator and verify the same for the above circuit.**

PIN DETAILS:-



TRUTH TABLE:-

E_n	A	B	C	D	E	F	G	H	Q_2(V)		Q_1(V)		Q_0(V)		E_S(V)		E_O(V)	
1	X	X	X	X	X	X	X	X	1		1		1		1		1	
0	0	1	1	1	1	1	1	1	1		1		1		0		1	
0	X	0	1	1	1	1	1	1	1		1		0		0		1	
0	0	X	0	1	1	1	1	1	1		0		1		0		1	
0	0	0	X	0	1	1	1	1	1		0		0		0		1	
0	0	0	0	X	0	1	1	1	0		1		1		0		1	
0	0	0	0	0	X	0	1	1	0		1		0		0		1	
0	0	0	0	0	0	X	0	1	0		0		1		0		1	
0	0	0	0	0	0	0	0	0	0		0		0		0		1	
0	1	1	1	1	1	1	1	1	1		1		1		1		0	

Experiment No: DATE:__/__/____

ENCODER & DECODER

AIM:-To convert a given octal input to the binary output and to study the LED display using 7447 7-segment decoder/ driver.

APPARATUS REQUIRED: -

IC 74148, IC 7447, 7-segment display, etc.

PROCEDURE: - (Encoder)

1. **Connections are made as per circuit diagram.**
2. **The octal inputs are given at the corresponding pins.**
3. **The outputs are verified at the corresponding output pins.**

PROCEDURE: - (Decoder)

1. **Connections are made as per the circuit diagram.**
2. **Connect the pins of IC 7447 to the respective pins of the LED display board.**
3. **Give different combinations of the inputs and observe the decimal numbers displayed on the board.**

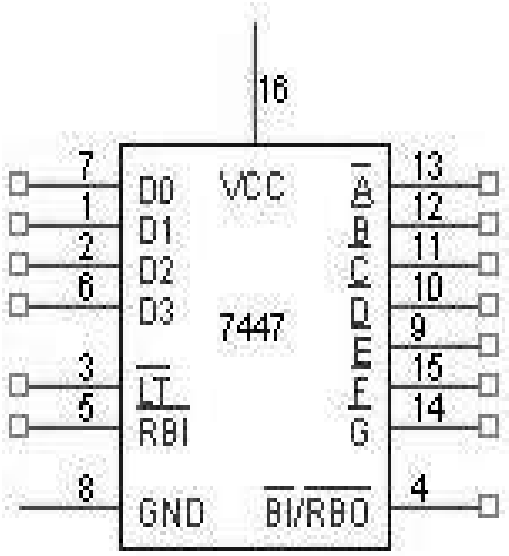
RESULT: -

The given octal numbers are converted into binary numbers. The given data is displayed using &-segment LED decoder.

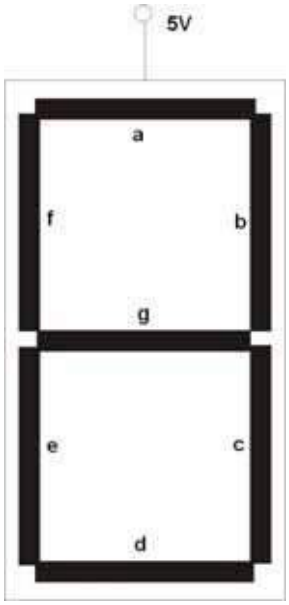
TABULAR COLUMN:-

Q4	Q3	Q2	Q1	O/P	Display	Glowing LEDs
0	0	0	0	0		a,b,c,d,e,f
0	0	0	1	1		b,c
0	0	1	0	2		a,b,d,e,g
0	0	1	1	3		a,b,c,d,g
0	1	0	0	4		b,c,f,g
0	1	0	1	5		a,c,d,f,g
0	1	1	0	6		a.c.d.e.f.g
0	1	1	1	7		a.b.c
1	0	0	0	8		a,b,c,d,e,f,g
1	0	0	1	9		a,b,c,d,f,g
1	0	1	0	10		d,e,g
1	0	1	1	11		c,d,g
1	1	0	0	12		c,d,e
1	1	0	1	13		a,g,d
1	1	1	0	14		d,e,f,g
1	1	1	1	15		blank

PIN DETAILS:-

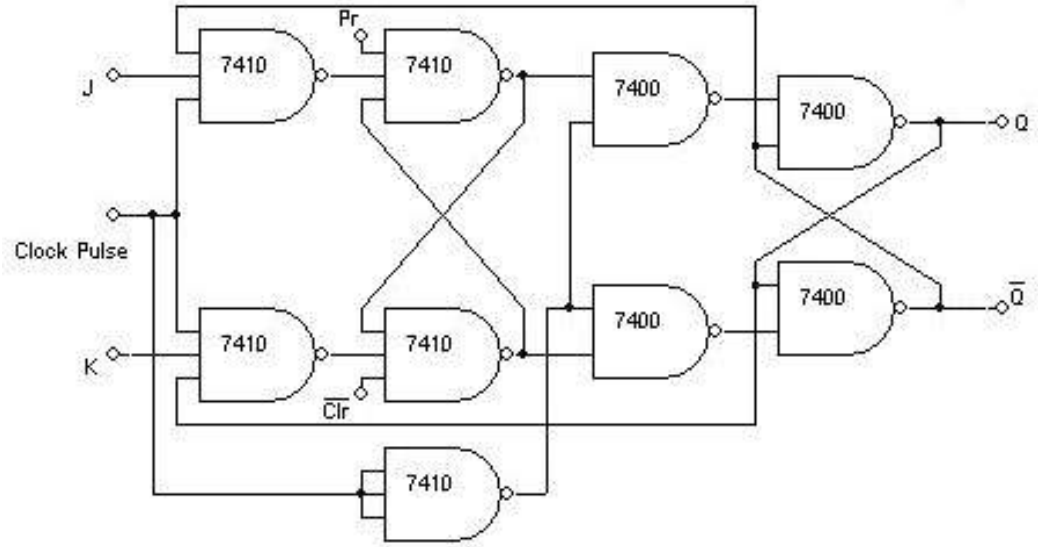


DISPLAY:-

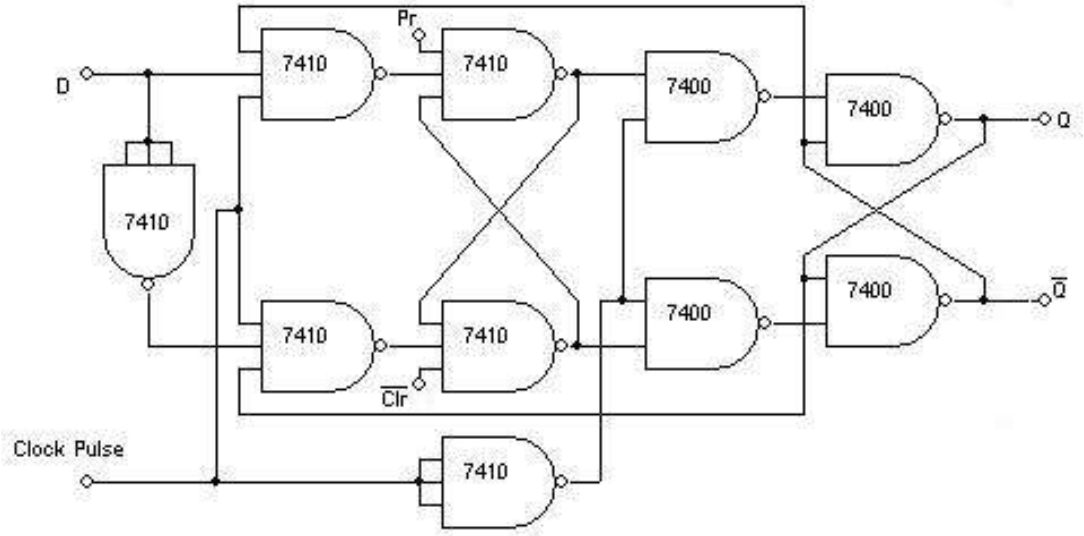


Conclusion:-

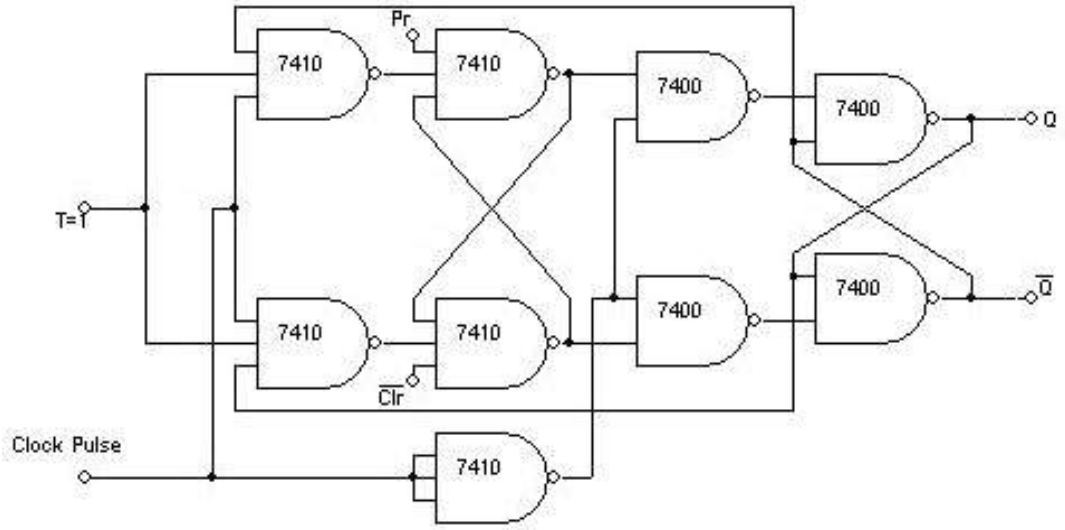
Circuit Diagram: - (Mas ter Slave JK Flip-Flop)



D Flip-Flop:-



T-Flip Flop


Pr
7410
7410
7400
7400
Q
T=1
7410
7410
7400
7400
$\overline{Q}$
$\overline{Clr}$
Clock Pulse
7410

Experiment No: Date:__/__/

FLIP-FLOP

Aim:- Truth table verification of Flip-Flops : (i) JK Master Slave
(ii) D- Type
(iii) T- Type.

Apparatus Required: -
IC 7410, IC 7400, etc.

Procedure: -
1. **Connections are made as per circuit diagram.**
2. **The truth table is verified for various combinations of inputs.**

Truth Table:- (Master Slave JK Flip-Flop)

Preset	Clear	J	K	Clock	Qn+1	$\overline{Qn}$ □ 1	
0	1	X	X	X	1	0	Set
1	0	X	X	X	0	1	Reset
1	1	0	0	⎍	Qn	$\overline{Qn}$	No Change
1	1	0	1	⎍	0	1	Reset
1	1	1	0	⎍	1	0	Set
1	1	1	1	⎍	$\overline{Qn}$	Qn	Toggle

D Flip-Flop:-

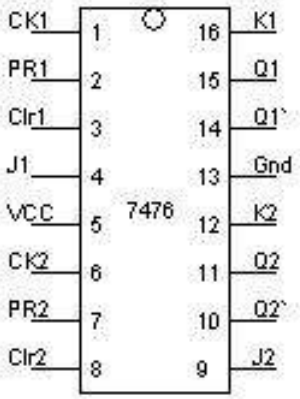
Preset	Clear	D	Clock	Qn+1	$\overline{Qn}$ □ 1
1	1	0	⎍	0	1
1	1	1	⎍	1	0

T Flip-Flop:-

Preset	Clear	T	Clock	Qn+1	$\overline{Qn}$ □ 1
1	1	0	⎍	Qn	$\overline{Qn}$
1	1	1	⎍	$\overline{Qn}$	Qn

Exercise:-
- **Write the timing diagrams for all the above Flip-Flops**

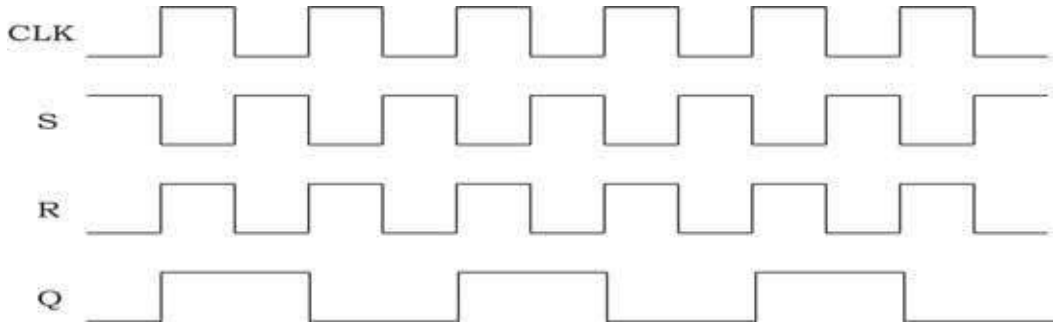
Pin Details: -



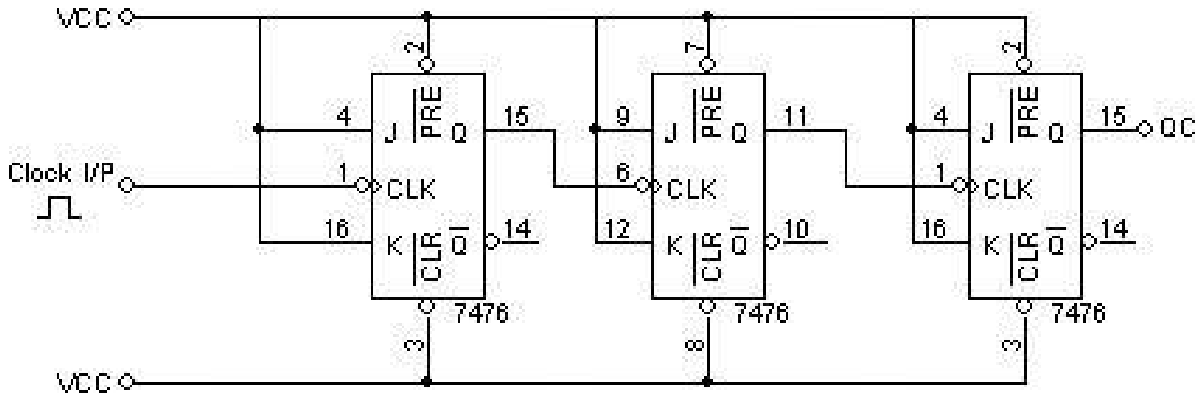
Truth Table:-

Clock	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

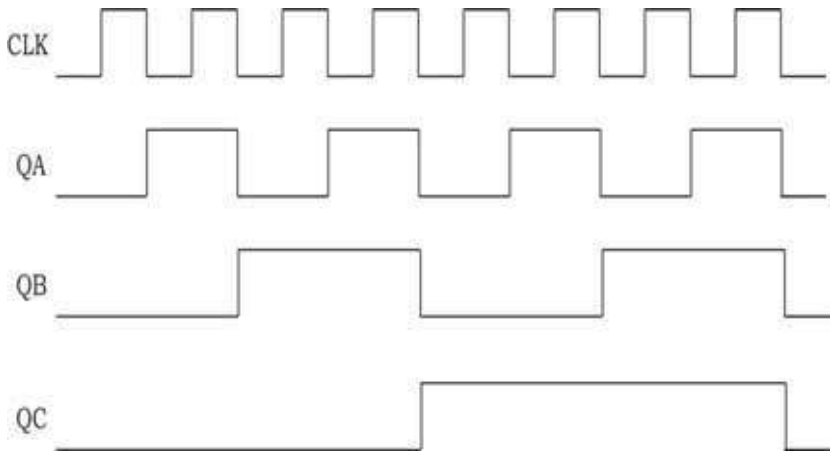
Timing Diagram:-



Circuit Diagram: - 3-Bit Asynchronous Up Counter



3-bit Asynchronous up counter			
Clock	**QC**	**QB**	**QA**
0	**0**	**0**	**0**
1	**0**	**0**	**1**
2	**0**	**1**	**0**
3	**0**	**1**	**1**
4	**1**	**0**	**0**
5	**1**	**0**	**1**
6	**1**	**1**	**0**
7	**1**	**1**	**1**
8	**0**	**0**	**0**



Experiment No: Date:__/__/

COUNTERS

Aim:- Realization of 3-bit counters as a sequential circuit and Mod-N counter design (7476, 7490, 74192, 74193).

Apparatus Required: -

IC 7408, IC 7476, IC 7 490, IC 74192, IC 74193, IC 7400, IC 7416, IC 7432 etc.

Procedure: -

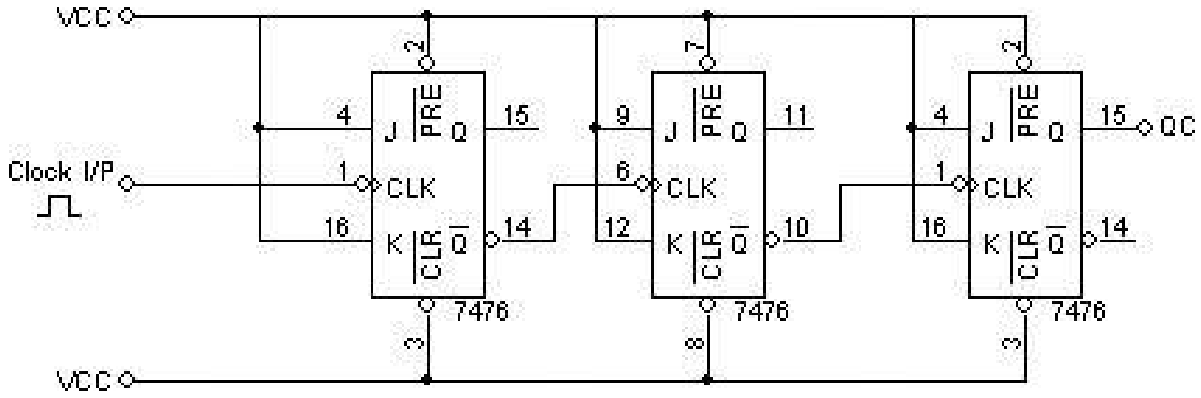
1. **Connections are made as per circuit diagram.**
2. **Clock pulses are applied one by one at the clock I/P and the O/P is observed at QA, QB & QC for IC 7476.**
3. **Truth table is verified.**

Procedure (IC 741 92, IC 74193):-

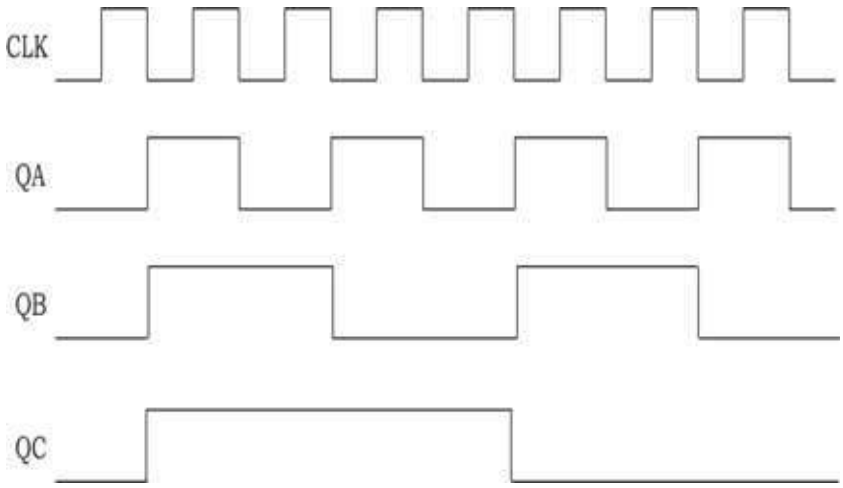
1. **Connections are made as per the circuit diagram except the connection from output of NAND gate to the load input.**
2. **The data (0011) = 3 is made available at the da ta i/ps A, B, C & D respectively.**
3. **The load pin made low so that the data 0011 appears at QD, QC, QB & QA respectively.**
4. **Now connect the output of the NAND gate to the load input.**
5. **Clock pulses are applied to "count up" pin and the truth table is verified.**
6. **Now apply (1100) = 12 for 12 to 5 count er and remaining is same as for 3 to 8 counter.**

7. **The pin diagram of IC 7419 2 is same as that of 74193. 74192 can be configured to count between 0 and 9 in either direction. The starting value can be any number between 0 and 9.**

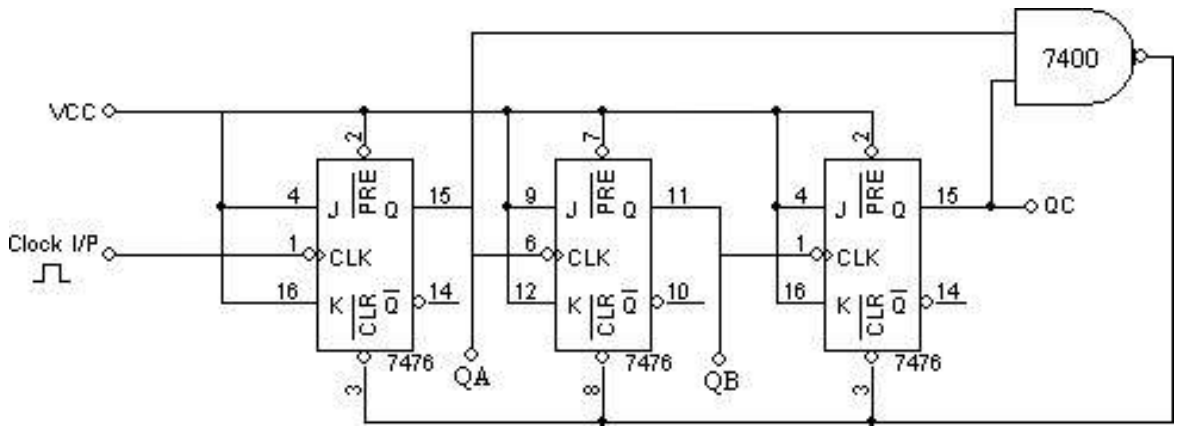
Circuit Diagram: - 3-Bit Asynchronous Down Counter



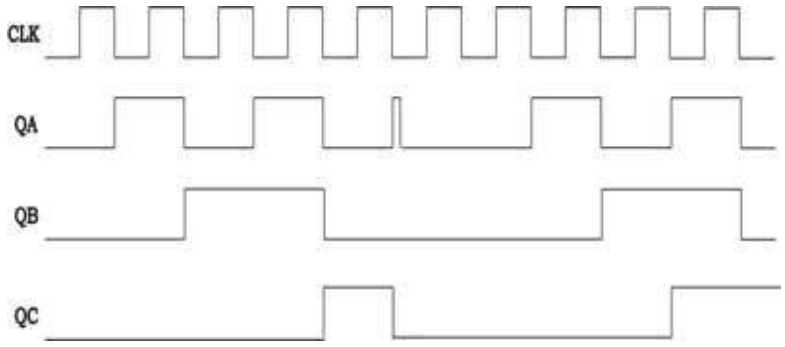
3-bit Asynchronous down counter			
Clock	**QC**	**QB**	**QA**
0	**1**	**1**	**1**
1	**1**	**1**	**0**
2	**1**	**0**	**1**
3	**1**	**0**	**0**
4	**0**	**1**	**1**
5	**0**	**1**	**0**
6	**0**	**0**	**1**
7	**0**	**0**	**0**
8	**1**	**1**	**1**
9	**1**	**1**	**0**



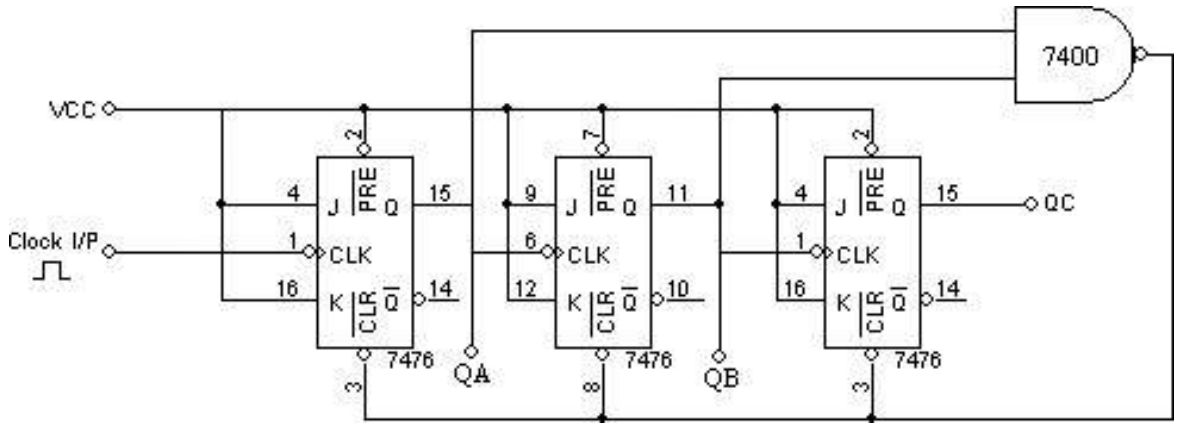
Mod 5 Asynchronous Counter: -



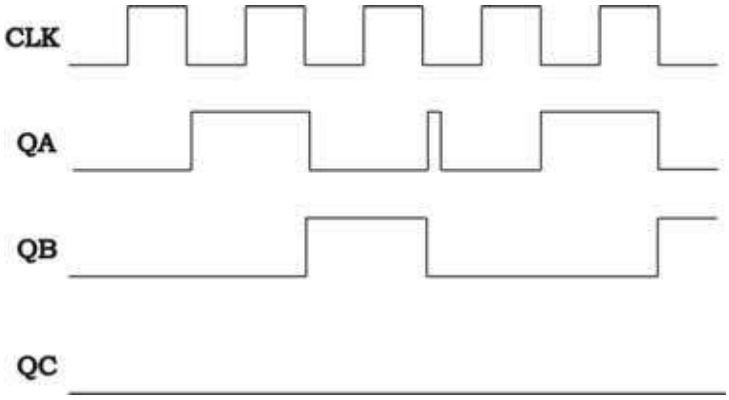
MOD 5 Asynchronous counter			
Clock	**QC**	**QB**	**QA**
0	**0**	**0**	**0**
1	**0**	**0**	**1**
2	**0**	**1**	**0**
3	**0**	**1**	**1**
4	**1**	**0**	**0**
5	**0**	**0**	**0**



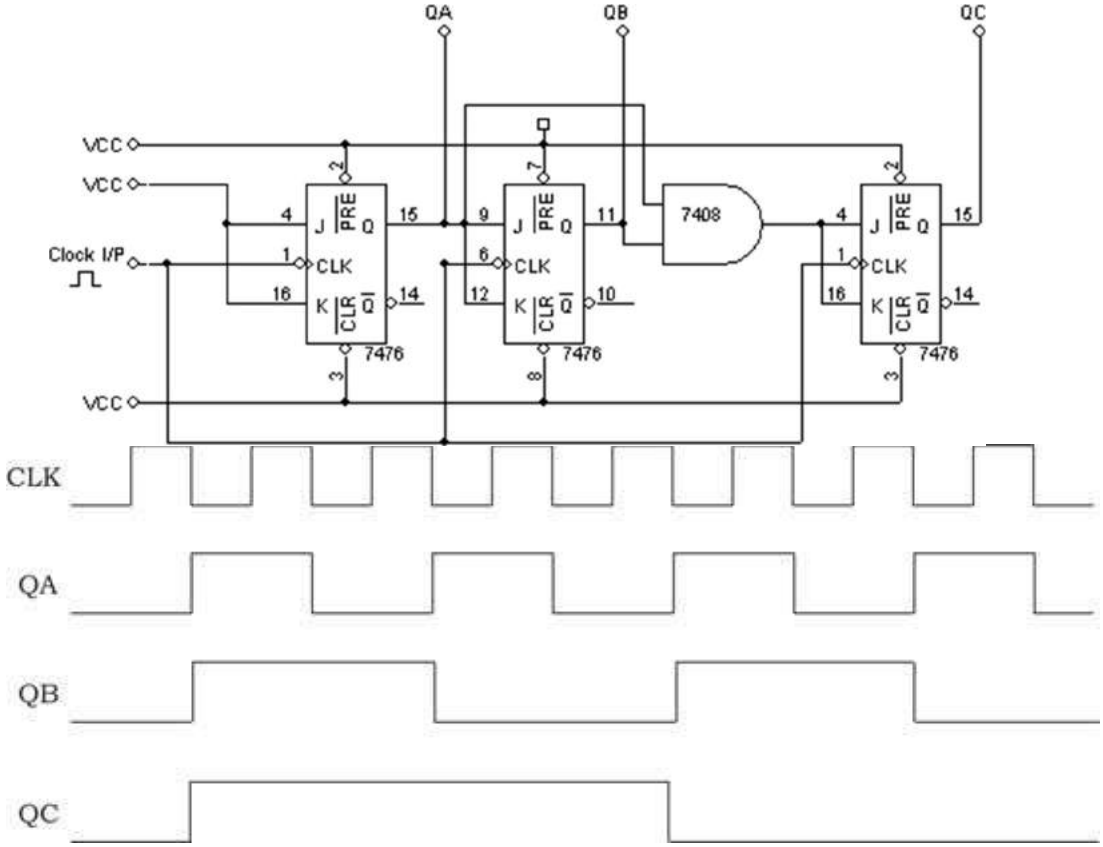
MOD 3 Asynchronous Counter:-



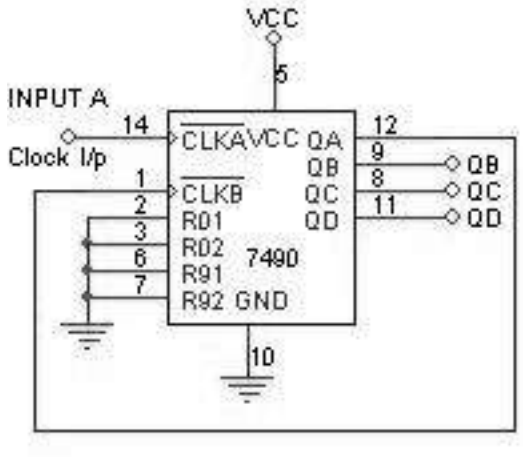
Mod 3 Asynchronous counter			
Clock	**QC**	**QB**	**QA**
0	**0**	**0**	**0**
1	**0**	**0**	**1**
2	**0**	**1**	**0**
3	**0**	**0**	**0**
4	**0**	**0**	**1**
5	**0**	**1**	**0**



3-bit Synchronous Counter:-

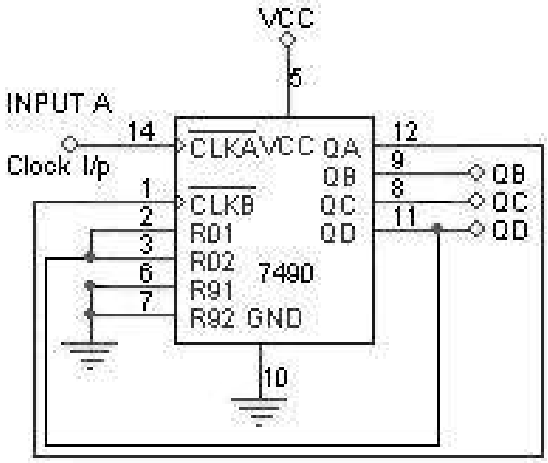


IC 7490 (Decade Counter):-



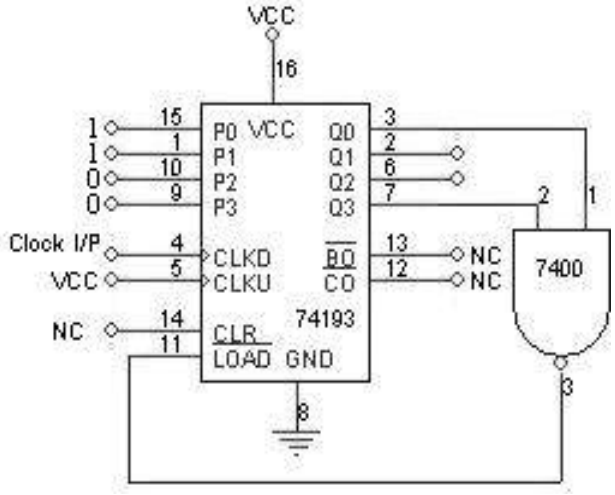
Clock	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

IC 7490 (MOD-8 Counter):-



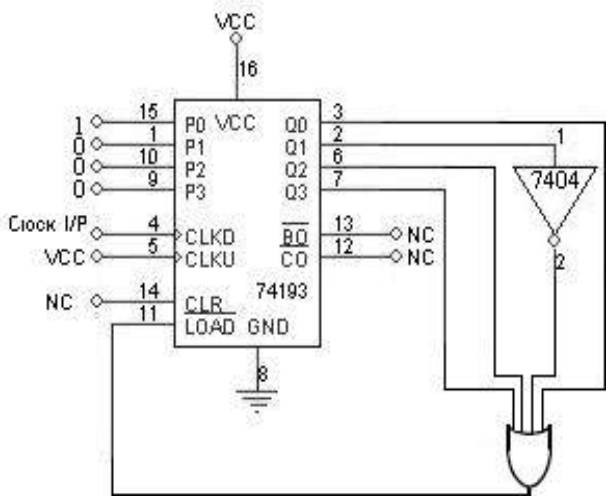
Clock	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	0	0	0	0
9	0	0	0	1

Circuit Diagram (IC 74193) To Count from 3 to 8:-



Clock	QD	QC	QB	QA	Count in Decimal
0	0	0	1	1	3
1	0	1	0	0	4
2	0	1	0	1	5
3	0	1	1	0	6
4	0	1	1	1	7
5	1	0	0	0	8
6	0	0	1	1	3
7	repeats				4

Circuit Diagram (IC 74193) To Count from 8 to 3:-

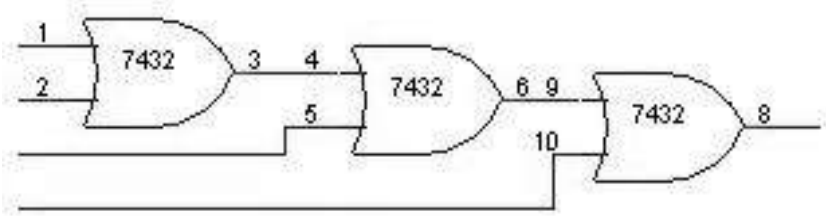


Clock	QD	QC	QB	QA	Count in Decimal
0	0	1	0	1	5
1	0	1	1	0	6
2	0	1	1	1	7
3	1	0	0	0	8
4	1	0	0	1	9
5	1	0	1	0	10
6	1	0	1	1	11
7	1	1	0	0	12
8	0	1	0	1	5
9	repeats				6

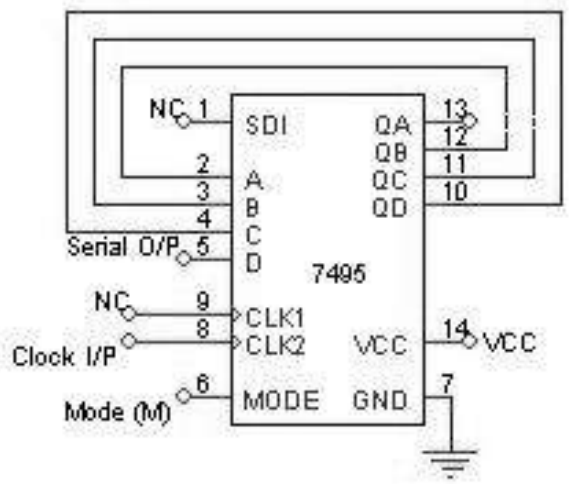
Function Table for 7490:-

Clock	R1	R2	S1	S2	QD	QC	QB	QA	
X	H	H	L	X	L	L	L	L	RESET
X	H	H	X	L	L	L	L	L	RESET
X	X	X	H	H	H	L	L	H	SET TO 9
⎍	X	L	X	L	COUNT				
⎍	L	X	L	X	COUNT				
⎍	L	X	X	L	COUNT				
⎍	X	L	L	X	COUNT				

4 I/P OR Gate can be realized as follows:-

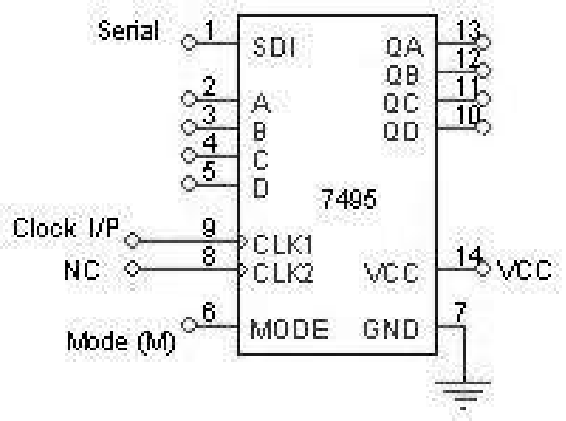


Circuit Diagram: - Shift Left



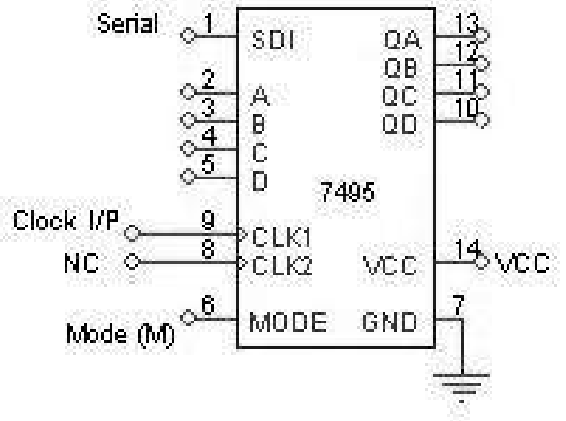
Clock	Serial i/p	QA	QB	QC	QD
1	1	X	X	X	1
2	0	X	X	1	0
3	1	X	1	0	1
4	1	1	0	1	1

SIPO (Right Shift):-



Clock	Serial i/p	QA	QB	QC	QD
1	0	0	X	X	X
2	1	1	0	X	X
3	1	1	1	0	X
4	1	1	1	1	0

SISO:-



Clock	Serial i/p	QA	QB	QC	QD
1	do=0	0	X	X	X
2	d1=1	1	0	X	X
3	d2=1	1	1	0	X
4	d3=1	1	1	1	0=do
5	X	X	1	1	1=d1
6	X	X	X	1	1=d2
7	X	X	X	X	1=d3

SHIFT REGISTERS

Aim: - Realization of 3-bit counters as a sequential circuit and Mod-N counter design (7476, 7490, 74192, 74193).

Apparatus Required: -

IC 7495, etc.

Procedure: -

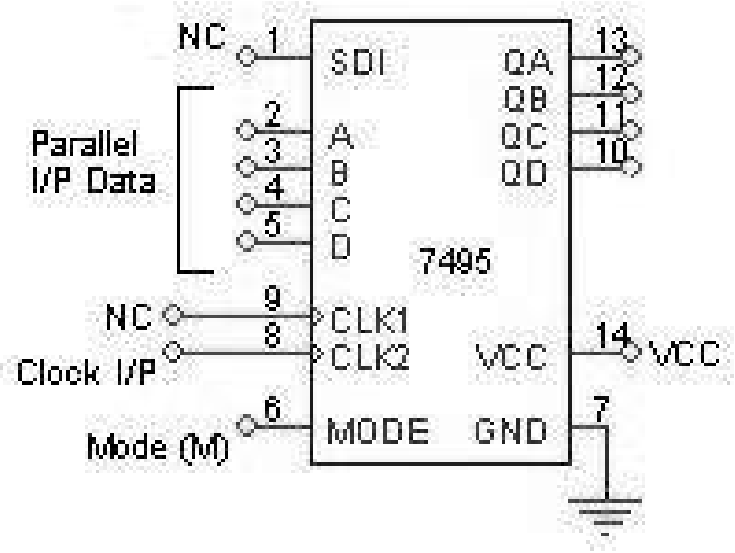
Serial In Parallel Out:-

1. **Connections are made as per circuit diagram.**
2. **Apply the data at serial i/p**
3. **Apply one clock pulse at clock 1 (Right Shift) observe this data at QA.**
4. **Apply the next data at serial i/p.**
5. **Apply one clock pulse at clock 2, observ e that the data on QA will shift to QB and the new data applied will appear at QA.**
6. **Repeat steps 2 and 3 till all the 4 bits data are entered on e by one into the shift register.**

Serial In Serial Out:-

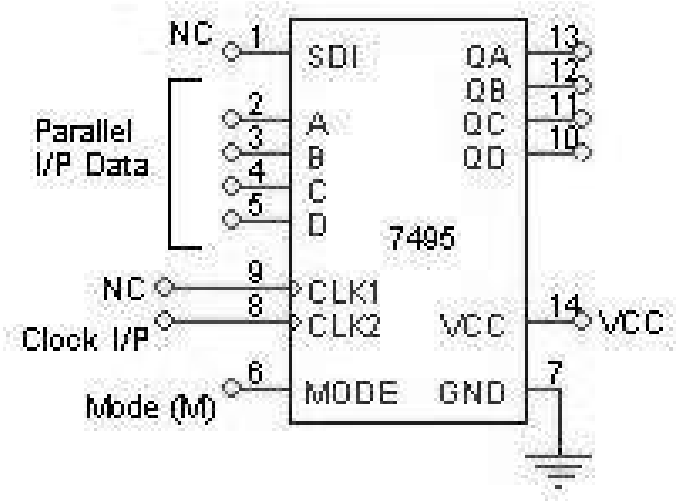
1. **Connections are made as per circuit diagram.**
2. **Load the shift register with 4 bits of data one by one serially.**
3. **At the end of 4th clock pulse the first data 'd0' appears at QD.**
4. **Apply another clock pulse; the second data 'd1' appears at QD.**
5. **Apply another clock pulse; the third data appears at QD.**
6. **Application of next clock pulse will enable the 4 th data 'd3' to appear at QD. Thus the data applied serially at the input comes out serially at QD**

PISO:-



Mode	Clock	Parallel i/p				Parallel o/p			
		A	B	C	D	QA	QB	QC	QD
1	1	1	0	1	1	1	0	1	1
0	2	X	X	X	X	X	1	0	1
0	3	X	X	X	X	X	X	1	0
0	4	X	X	X	X	X	X	X	1

PIPO:-



Clock	Parallel i/p				Parallel o/p			
	A	B	C	D	QA	QB	QC	QD
1	1	0	1	1	1	0	1	1

Parallel In Parallel Out:-

1. **Connections are made as per circuit diagram.**
2. **Apply the 4 bit data at A, B, C and D.**
3. **Apply one clock pulse at Clock 2 (Note: Mode control M=1).**
4. **The 4 bit data at A, B, C and D appears at QA, QB, QC and QD respectively.**

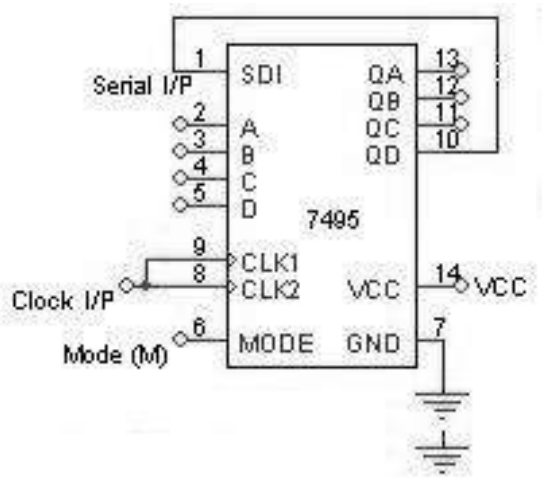
Parallel In Serial Out:-

1. **Connections are made as per circuit diagram.**
2. **Apply the desired 4 bit data at A, B, C and D.**
3. **Keeping the mode control M=1 apply one clock pulse. The data applied at A, B, C and D will appear at QA, QB, QC and QD respectively.**
4. **Now mode control M=0. Apply clock pulses one by one and observe the data coming out serially at QD.**

Left Shift:-

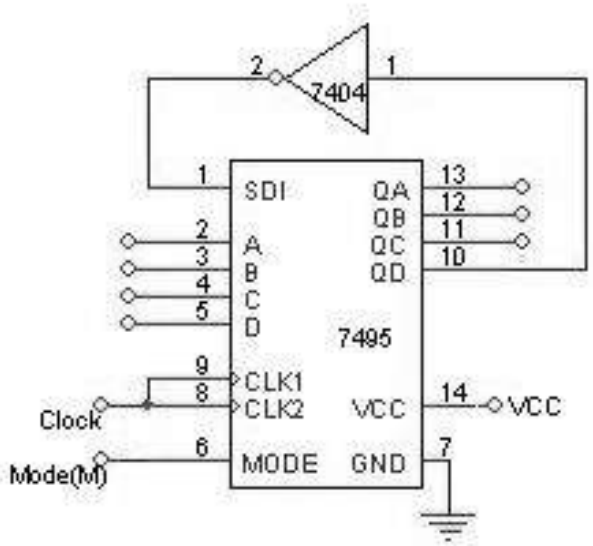
1. **Connections are made as per circuit diagram.**
2. **Apply the first data at D and apply one clock pulse. This data appears at QD.**
3. **Now the second data is made available at D and one clock pulse applied. The data appears at QD to QC and the new data appears at QD.**
4. **Step 3 is repeated until all the 4 bits are entered one by one.**
5. **At the end 4th clock pulse, the 4 bits are available at QA, QB, QC and QD.** **Conclusion: -**

Circuit Diagram: - Ring Counter



Mode	Clock	QA	QB	QC	QD
1	1	1	0	0	0
0	2	0	1	0	0
0	3	0	0	1	0
0	4	0	0	0	1
0	5	1	0	0	0
0	6	repeats			

Johnson Counter:-



Mode	Clock	QA	QB	QC	QD
1	1	1	0	0	0
0	2	1	1	0	0
0	3	1	1	1	0
0	4	1	1	1	1
0	5	0	1	1	1
0	6	0	0	1	1
0	7	0	0	0	1
0	8	0	0	0	0
0	9	1	0	0	0
0	10	repeats			

Experiment No: Date:__/__/

JOHNSON COUNTERS / RING COUNTER

Aim:- Design and testing of Ring counter/ Johnson counter.

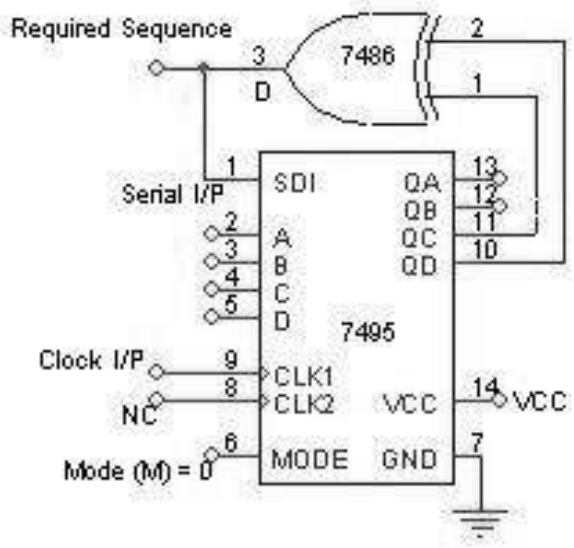
Apparatus Required: -

IC 7495, IC 7404, etc.

Procedure: -

1. **Connections are made as per the circuit diagram.**
2. **Apply the data 1000 at A, B, C and D respectively.**
3. **Keeping the mode M = 1, apply one clock pulse.**
4. **Now the mode M is made 0 and clock pulses are applied one by one, and the truth table is verified.**
5. **Above procedure is repeated for Johnson counter also.**

Circuit Diagram: - Sequence Generator



Truth Table:-

Map Value	Clock	QA	QB	QC	QD	o/p D
15	1	1	1	1	1	0
7	2	0	1	1	1	0
3	3	0	0	1	1	0
1	4	0	0	0	1	1
8	5	1	0	0	0	0
4	6	0	1	0	0	0
2	7	0	0	1	0	1
9	8	1	0	0	1	1
12	9	1	1	0	0	0
6	10	0	1	1	0	1
11	11	1	0	1	1	0
5	12	0	1	0	1	1
10	13	1	0	1	0	1
13	14	1	1	0	1	1
14	15	1	1	1	0	1

Karnaugh Map for D:-

QCQD \ QA QB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Experiment No: Date: __/__/

SEQUENCE GENERATOR

Aim:- Design of Sequence Generator.

Apparatus Required: -

IC 7495, IC 7486, etc.

Design:-

To generate a sequence of length S it is necessary to use at least N number of Flip-Flops, which satisfies the condition $S \leq 2^N - 1$.

The given sequence length S = 15.

Therefore N = 4.

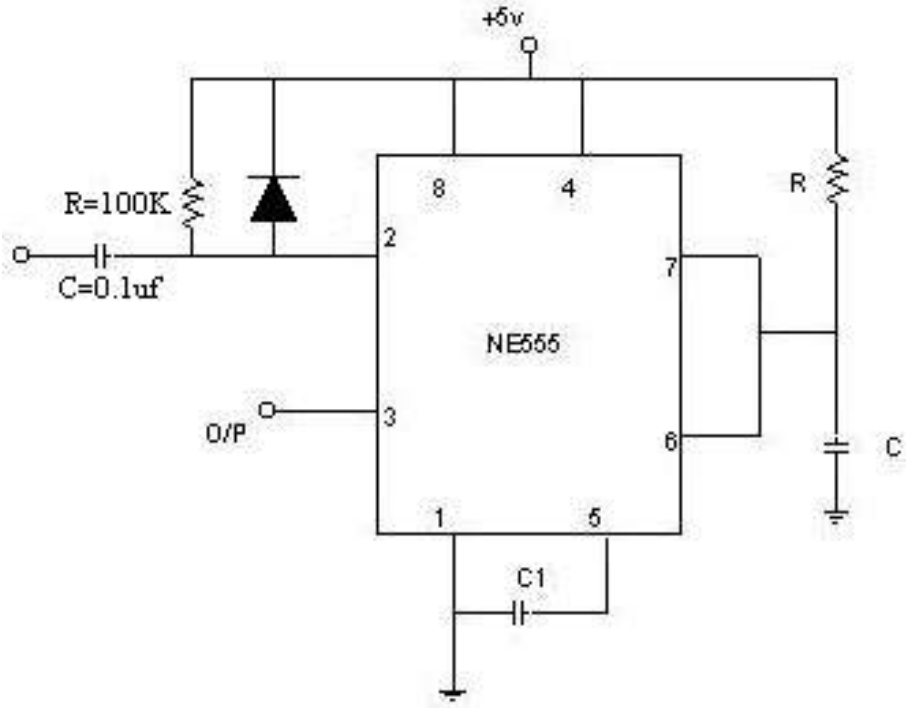
Note: - There is no guarantee that the given sequence can be generated by 4 f/fs. If the sequence is not realizable by 4 f/fs then 5 f/fs must be used and so on.

Procedure: -

1. **Connections are made as per the circuit diagram.**
2. **Clock pulses are applied one by one and truth table is verified.**

Conclusion:-

Circuit Diagram: - Monostable Multivibrator



Waveform:-