ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES::TIRUPATI
II B.Tech II Semester

L	T	P	C
0	0	2	1

(19APC0504) Computer Organization Lab
Course Objectives:

1. Understanding the behavior of logic gates, adders, decoders, multiplexers and flipflops.
2. Understanding the behavior of ALU, RAM, STACK and PROCESSOR from working modules and the modules designed by the student as part of the experiment.

Exercises in Digital Logic Design:

- Implement Logic gates using NAND and NOR gates
- Design a Full adder using gates
- Design and implement the 4:1 MUX, 8:1 MUX using gates /ICs.
- Design and Implement a 3 to 8 decoder using gates
- Design a 4 bit comparator using gates/IC
- Design and Implement a 4 bit shift register using Flip flops
- Design and Implement a Decade counter

Exercises in Computer Organization:

- Implement a C program to convert a Hexadecimal, octal, and binary number to decimal number vice versa.
- Implement a C program to perform Binary Addition & Subtraction.
- Implement a C program to perform Multiplication of two binary numbers
- Implement a C program to perform Multiplication of two binary numbers (signed) using Booth's Algorithms.
- Implement a C program to perform division of two binary numbers (Unsigned) using restoring division algorithm.
- Implement a C program to perform division of two binary numbers (Unsigned) using non-restoring division algorithm.

Task1:
Implement Logic gates using NAND and NOR gates.

NAND :

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a[5] = { 1, 0, 1, 0, 1 };
    int b[5] = { 0, 1, 1, 0, 0 };
    int i, ans;

    for (i = 0; i < 5; i++) {
        if (a[i] == 1 && b[i] == 1)
            ans = 0;
        else
            ans = 1;
        printf("\n %d NAND %d = %d", a[i], b[i], ans);
    }
}
```

Output:

```
1 NAND 0 = 1
0 NAND 1 = 1
1 NAND 1 = 0
0 NAND 0 = 1
1 NAND 0 = 1
```

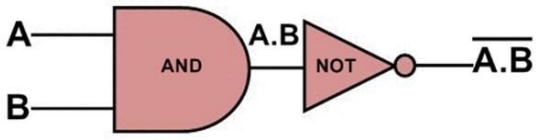
NOR:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a[5] = { 1, 0, 1, 0, 1 };
    int b[5] = { 0, 1, 1, 0, 0 };
    int i, ans;

    for (i = 0; i < 5; i++) {
        ans = !(a[i] + b[i]);
        printf("\n %d NOR %d = %d", a[i], b[i], ans);
    }
}
```

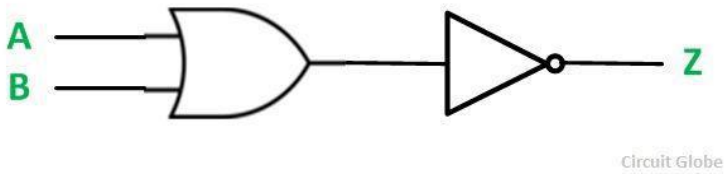
Output:

```
1 NOR 0 = 0
0 NOR 1 = 0
1 NOR 1 = 0
0 NOR 0 = 1
1 NOR 0 = 0
```

Logic Gate:



NAND GATE



NOR GATE

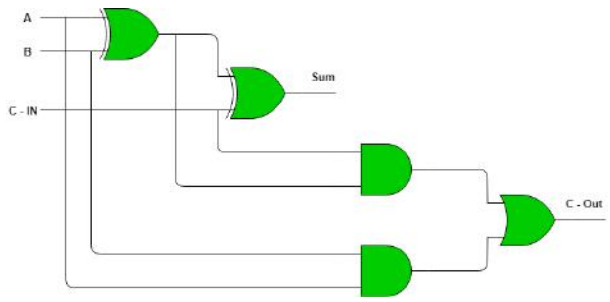
Task2: Design a Full adder using gates

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

char* addBinary(char* a, char* b)
{
	int la = strlen(a);
	int lb = strlen(b);
	int lr = la > lb ? la : lb;
	int carry = 0;
	char *res = (char*)calloc(lr + 2, sizeof(char));
	res[lr + 1] = '\0';
	la--; lb--;
	while (la >= 0 || lb >= 0)
	{
		int ba = la >= 0 ? a[la--] - '0' : 0;
		int bb = lb >= 0 ? b[lb--] - '0' : 0;
		int br = ba ^ bb ^ carry;
		carry = (ba & bb) | (carry & (ba ^ bb));
		res[lr--] = br + '0';
	}
	if (!carry) return res + 1;
	res[0] = '1';
	return res;
}

void main()
{
	clrscr();
	printf("%s",addBinary("1110","1011"));
	getch();
}
```

Output:

11001

Logic Gate:



EXPERIMENT -3

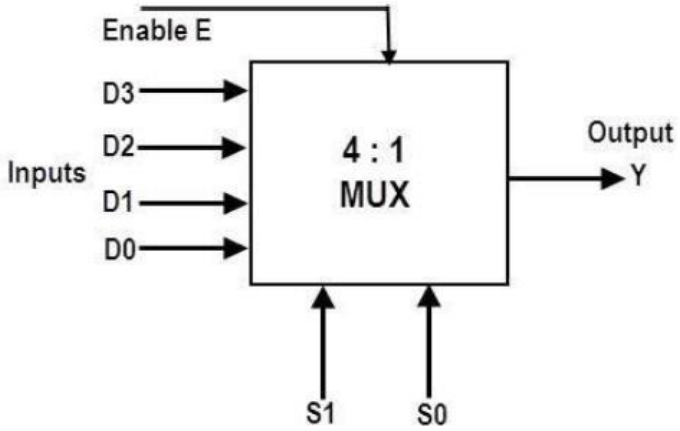
MULTIPLEXER

AIM: To design and implement the 4:1 MUX, 8:1 MUX using gates /ICs.

Theory:

Multiplexers are very useful components in digital systems. They transfer a large number of information units over a smaller number of channels, (usually one channel) under the control of selection signals. Multiplexer means many to one. A multiplexer is a circuit with many inputs but only one output. By using control signals (select lines) we can select any input to the output. Multiplexer is also called as data selector because the output bit depends on the input data bit that is selected. The general multiplexer circuit has 2n input signals, n control/select signals and 1 output signal.

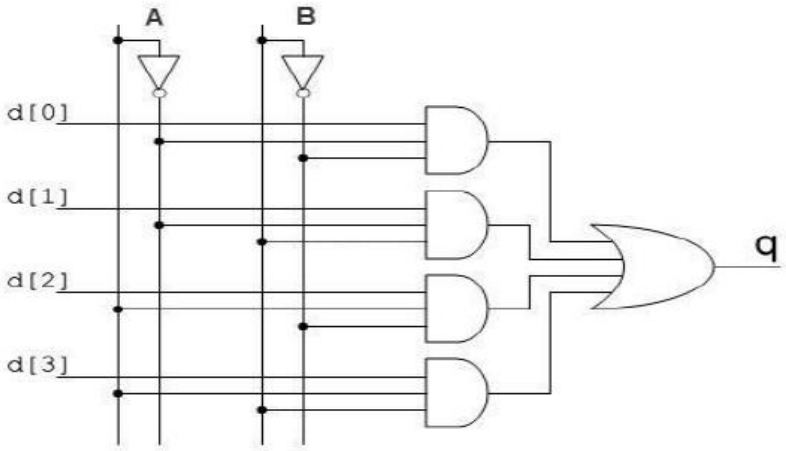
i) 4:1 MULTIPLEXER



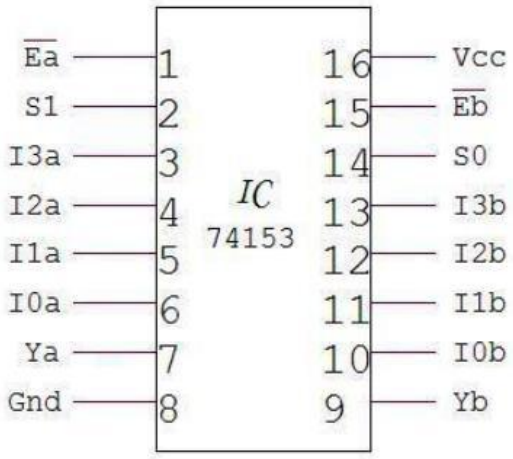
To get the total data output from the multiplexer, all these product terms are to be summed and then the final Boolean expression of this multiplexer is given as

$$Y = D0\,\overline{S1}\,\overline{S0} + D1\,\overline{S1}\,S0 + D2\,S1\,\overline{S0} + D3\,S1\,S0$$

Logic Gate:



Pin Diagram:



Truth Table:

Select inputs		Enable input	Inputs				Output
S1	S0	E	I0	I1	I2	I3	Y
X	X	1	X	X	X	X	0
0	0	0	0	X	X	X	0
0	0	0	1	X	X	X	1
0	1	0	X	0	X	X	0
0	1	0	X	1	X	X	1
1	0	0	X	X	0	X	0
1	0	0	X	X	1	X	1
1	1	0	X	X	X	0	0
1	1	0	X	X	X	1	1

PROCEDURE:

1) Connect the circuit as shown in figure.
2) Apply Vcc & ground signal to every IC.
3) Observe the input & output according to the truth table.

(ii) 8:1 MULTIPLEXER

AIM: To verify the truth table of Multiplexer.

Theory:

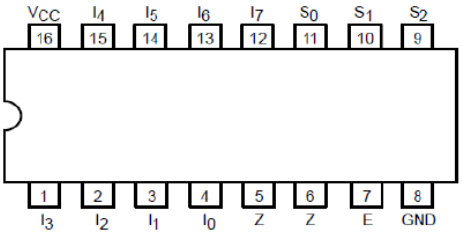
The TTL/MSI SN54/74LS151 is a high speed 8-input Digital Multiplexer. It provides, in one package, the ability to select one bit of data from up to eight sources. The LS151 can be used as universal function generator to generate any logic function of 4 variables. Both assertion and negation outputs are provided.

- Schottky Process for high speed
- Multifunction capability
- On-chip select logic decoding.
- Fully buffered complementary outputs
- Input clam diodes limit high speed termination effects.

Functional description:

The LS151 is a logical implementation of a single pole, 8-position switch with the switch position controlled by the state of three select inputs, s0, s1, s2. Both assertion and negation outputs are provided. The enable input (E) is active LOW. When it is not activated, the negation output is HIGH and assertion output is LOW regardless of all other inputs.

CONNECTION DIAGRAM DIP (TOP VIEW)



Truth Table:

TRUTH TABLE

$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Z}$	Z
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

PROCEDURE:

1. Connect Inputs S2,S1,S0,I0,I1,I2,I,3I4,I5,I6,I7 and Enable(E)to Logic Input Sockets.
2. Connect Output terminal Z & $\bar{Z}$ to Logic Output Sockets.
3. Verify output with given Truth Table.

CONCLUSION: Hence verified the truth table of Multiplexer.

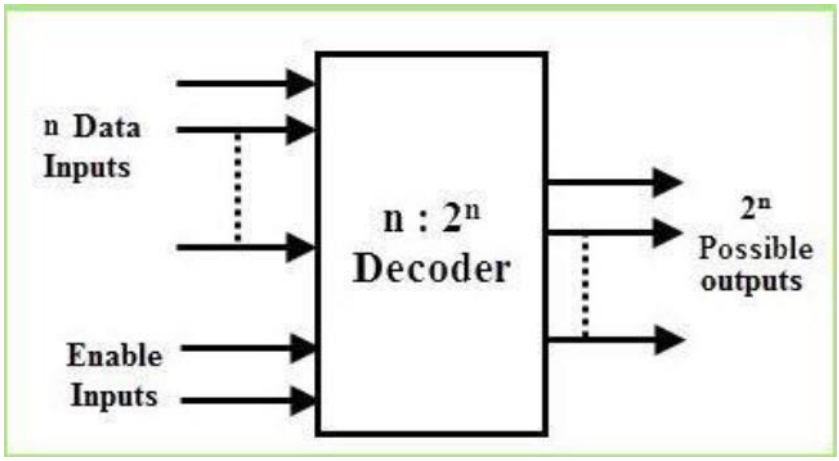
EXPERIMENT -4

3 TO 8 DECODERS

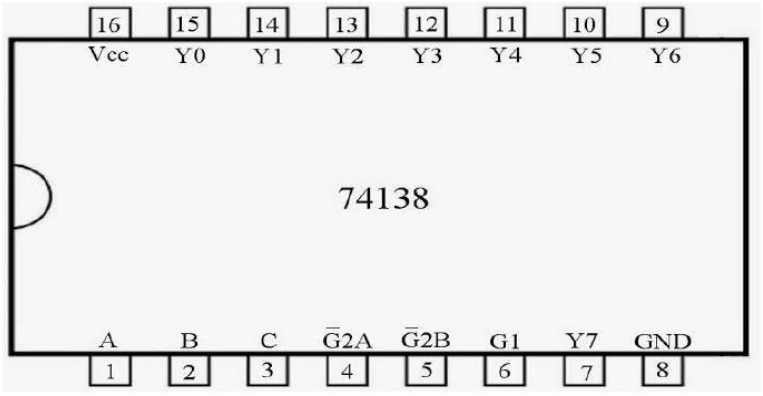
Aim: To design and Implement a 3 to 8 decoder using gates

Theory:

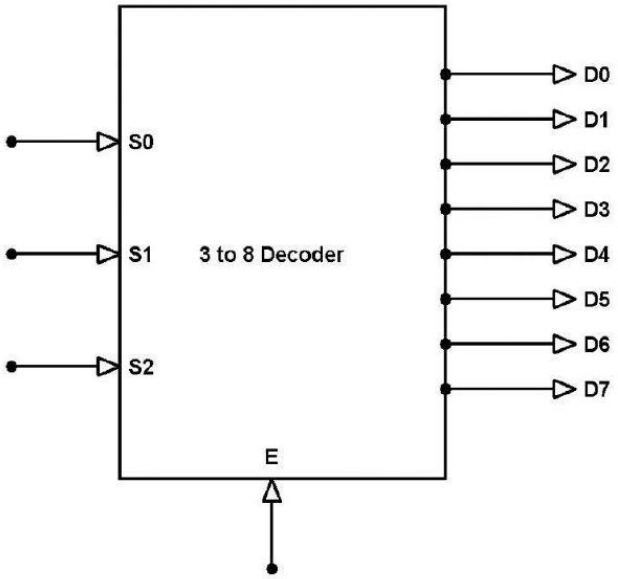
A decoder is a device which does the reverse operation of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode. It is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines. In digital electronics, a decoder can take the form of a multiple- input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. e.g. n-to-2n , binary-coded decimal decoders. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. In case of decoding all combinations of three bits eight (23=8) decoding gates are required. This type of decoder is called 3-8 decoder because 3 inputs and 8 outputs. For any input combination decoder outputs are 1. This decoder circuit gives 8 logic outputs for 3 inputs and has a enable pin. The circuit is designed with AND and NAND logic gates. It takes 3 binary inputs and activates one of the eight outputs. 3 to 8 line decoder circuit is also called as binary to an octal decoder.



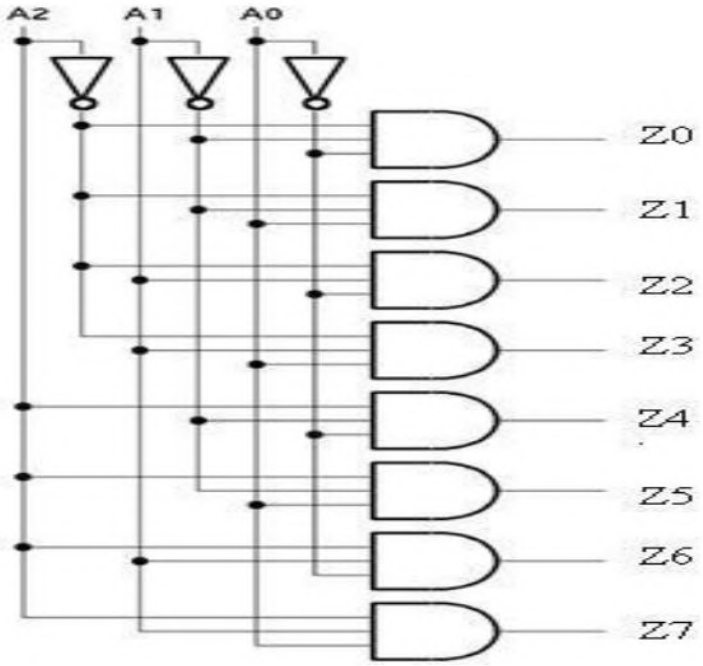
Pin Diagram of 74138 IC:



3 Line to 8 Line Decoder:



Logic Diagram:



Truth Table:

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

PROCEDURE:

1. Connections are given as per circuit diagram
2. Logical inputs are given as per circuit diagram.
3. The decoder circuit works only when the Enable pin (E) is high.
4. S0, S1 and S2 are three different inputs and D0, D1, D2, D3. D4. D5. D6. D7 are the eight outputs.
5. When the Enable pin (E) is low all the output pins are low.
6. Observe the output and verify the truth table.

Result: Hence verified the truth table of 3:8 decoders.

EXPERIMENT -5

4-BIT COMPARATOR

AIM: To verify the truth table of 4-bit Comparator.

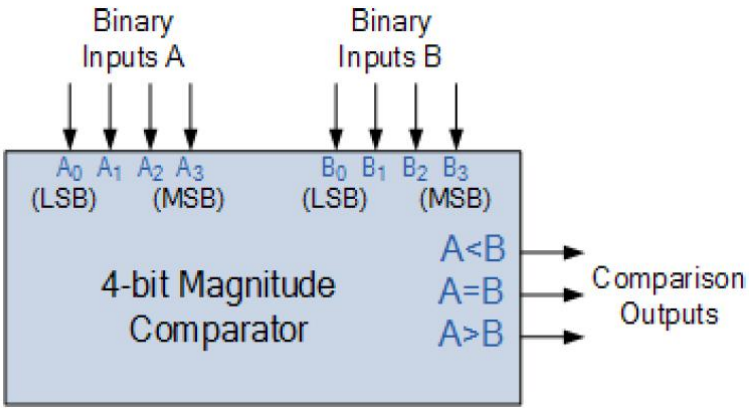
Theory:

The SN54/74LS85 is a 4-Bit Magnitude comparator which compares two 4-bit words (A,B), each word having four parallel inputs (A0-A3, B0-B3); A3,B3being the most significant inputs. Operation is not restricted to binary codes; the device will work with any monotonic code. Three outputs are provided: "A greater than B" ($O_{A>B}$), "A less than B" ($O_{A<B}$), "A equal to B" ($O_{A=B}$). Three expander inputs $I_{A>B}$, $I_{A<B}$, $I_{A=B,}$ allow cascading without external gates. For proper compare operation, the expander inputs to the least significant position must be connected as follows: $I_{A<B}$= $I_{A>B}$=L, $I_{A=B}$=H. For serial (ripple) expansion, $O_{A>B}$, $O_{A<B}$, and $O_{A=B}$ outputs are connected respectively to the $I_{A>B}$,$I_{A<B}$, and $I_{A=B}$ inputs of the next most significant comparator, as shown in figure1. Refer to application section of data sheet for high speed method of comparing large words.

The truth table on the following page describes the operation of SN54/74LS85 under all possible logic conditions. The upper 11 lines describe the normal operation under all conditions that will occur in a single device or in a series expansion scheme. The lower five lines describe the operation under abnormal conditions on the cascading inputs. These conditions occur when the parallel expansion technique is used.

- Easily expandable.
- Binary or BCD Comparison
- $O_{A>B}$, $O_{A<B}$, and $O_{A=B}$ outputs available.

4-bitMagnitudeComparator:

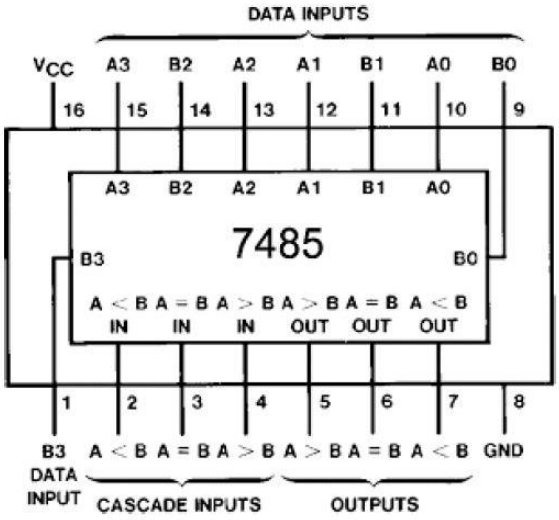


Some commercially available digital comparators such as the TTL74LS85 or CMOS4063 4-bit magnitude comparator have additional input terminals that allow more individual comparator strobe "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8,16 or even 32-bit words.

When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit(MSB) first. If equality exists, A=B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

If in equality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits tops. Digital Comparator is used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations. Magnitude Comparator is a logical circuit, which compares two signals A and B and generates three logical outputs, whether A>B, A=B, or A<B. IC7485 is a high speed 4-bit Magnitude comparator, which compares two 4-bit words. The A=B Input must be held high for proper compare operation.

PINDIAGRAM:



PROCEDURE:

1. Connect Inputs (A0,A1,A2,A3,B0,B1,B2,B3) to Input Switches
2. Connect Cascade Inputs(A<B,A>B,A=B)to Input Switches
3. Connect Outputs (A<B,A>B,A=B)to Output Switches
4. Observe the Truth Table.

Truth Table:

COMPARING INPUTS				CASCADINGINPUTS			OUTPUTS		
A3,B3	**A2,B2**	**A1,B1**	**A0,B0**	**A>B**	**A<B**	**A=B**	**A>B**	**A<B**	**A=B**
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L

CONCLUSION: Verified the truth table of 4- bit Comparator.

EXPERIMENT -6

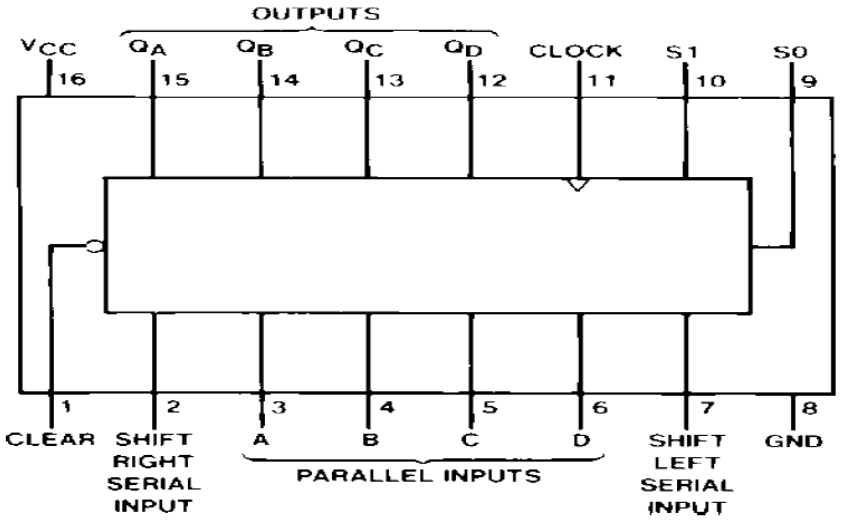
UNIVERSAL SHIFT REGISTER

AIM: To Verify the Truth Table of Universal Shift Register.

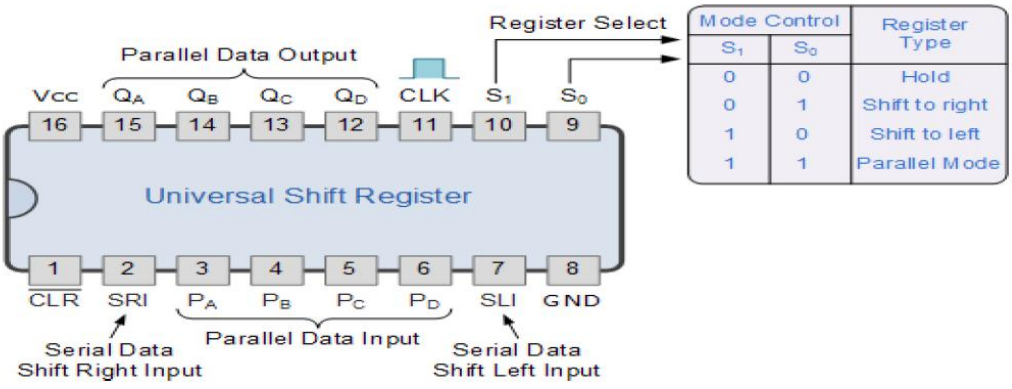
THEORY:

A register is simply a group of flip flops that can be used to store a binary number. A shift register is a group of flip flops connected such that the binary number can be entered (shifted) into the register and possibly shifted out. There are two ways to shift the data (bits in the binary number) from one place to another. The first method involves shifting the data 1 bit at a time in a serial fashion, beginning with either MSB or LSB. This technique is referred to as serial shifting. These methods involve shifting all the data bits simultaneously and are referring red to as parallel shifting. There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift data out of the register. This leads to the construction off our basic types of registers.

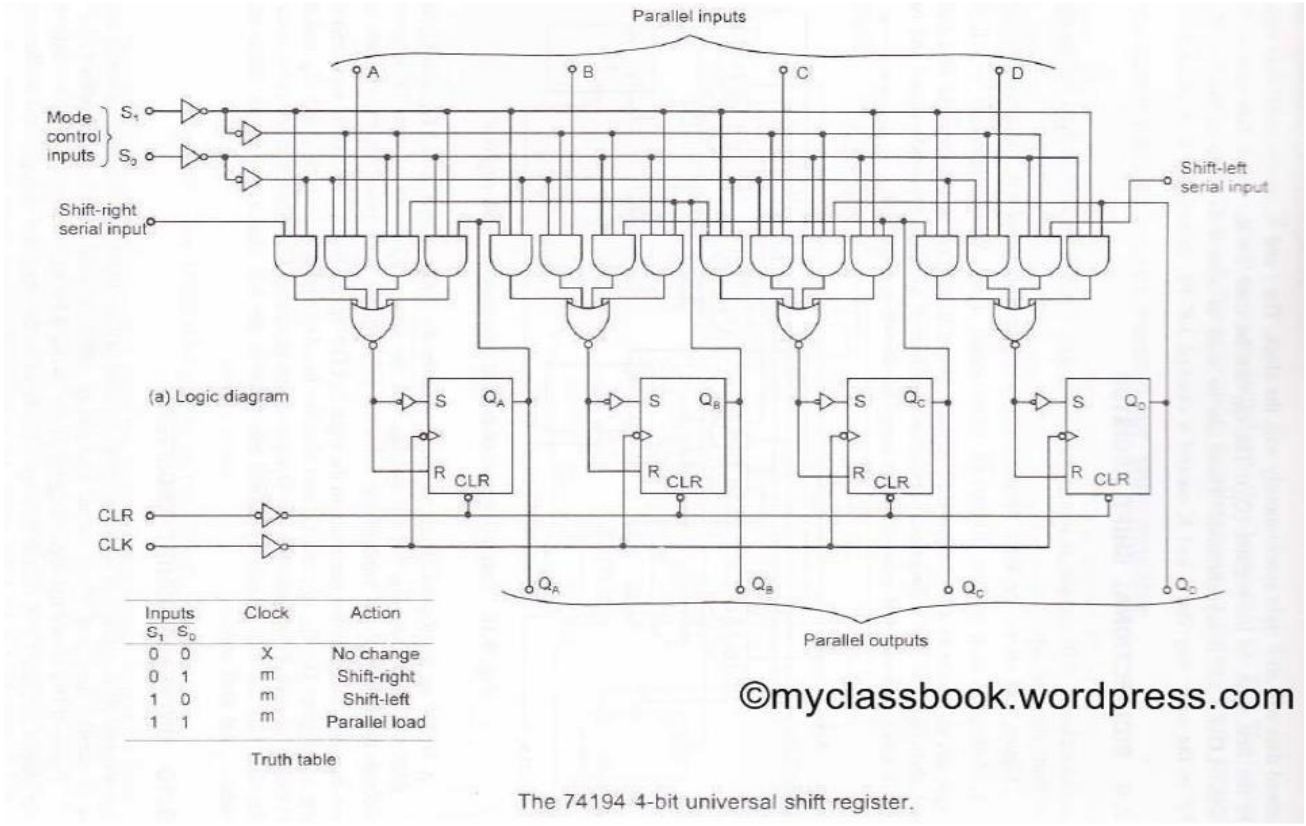
1. Serial in–Serial out shift register.
2. Serial in–Parallel out shift register.
3. Parallel in–Serial out shift register.
4. Parallel in–Parallel out shift register.



Modes of Operation:



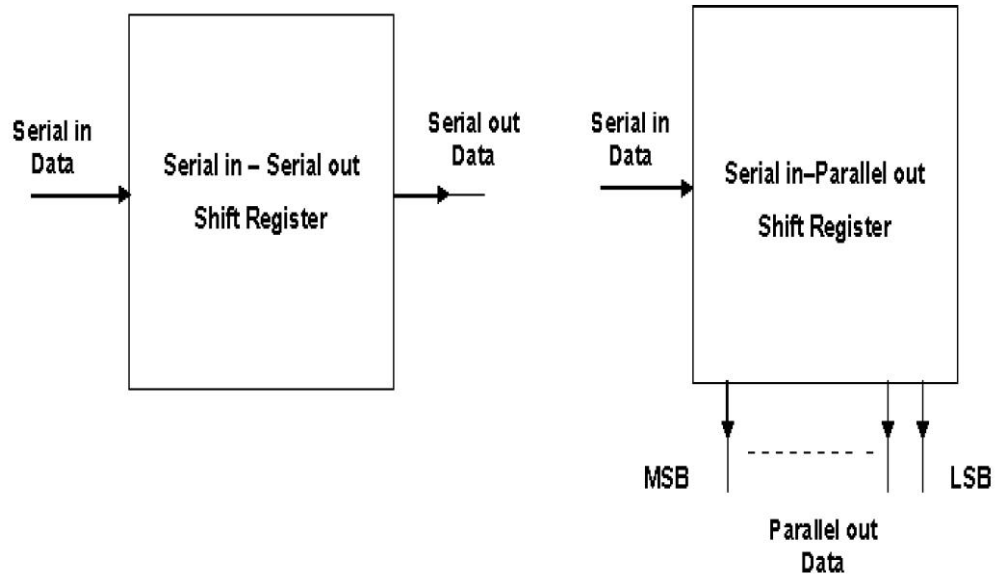
INTERNAL DIAGRAM OF 74194:

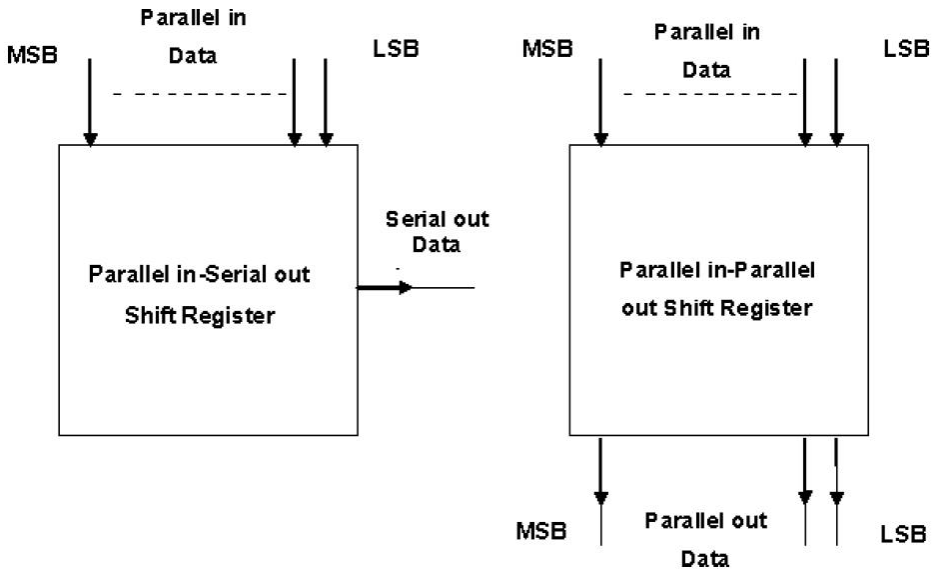


Inputs S_1	Inputs S_0	Clock	Action
0	0	X	No change
0	1	m	Shift-right
1	0	m	Shift-left
1	1	m	Parallel load

Truth table

Block diagram of 4 types of shift registers (n-bit):





Truth table:

INPUTS										OUTPUTS			
CLEAR	MODE		CLOCK	SERIAL		PARALLEL				QA	QB	QC	QD
	S1	S0		LEFT	RIGHT	A	B	C	D				
H	X	X	⌞	X	X	X	X	X	X	QA0	QB0	QC0	QD0
H	H	H	⌟	X	X	a	b	c	d	a	b	c	d
H	L	H	⌟	X	H	X	X	X	X	H	QAn	QBn	QCn
H	L	H	⌟	X	L	X	X	X	X	L	QAn	QBn	QCn
H	H	L	⌟	H	X	X	X	X	X	QBn	QCn	QDn	H
H	H	L	⌟	L	X	X	X	X	X	QBn	QCn	QDn	L
H	L	L	X	X	X	X	X	X	X	QA0	QB0	QC0	QD0

PROCEDURE:

1. Derive the wiring diagram.
2. Connect Input terminals to the Logic Input Sockets.
3. S1,S0, SERIAL LEFT,SERIAL RIGHT,PARALLEL DATA-A,B,C and D to Logic Input Sockets.
4. Connect CLEAR to VCC.
5. Connect External Clock to the CLOCK Terminal.
6. Connect QA, QB, QC and QD to the Logic Output terminals.
7. Observe output changes at O/P Terminals with given Truth Table.

CONCLUSION: Verified the Truth Table of Universal Shift Register.

EXPERIMENT -7

DECADE COUNTER

AIM: To Verify The Truth Table Of Decade Counter.

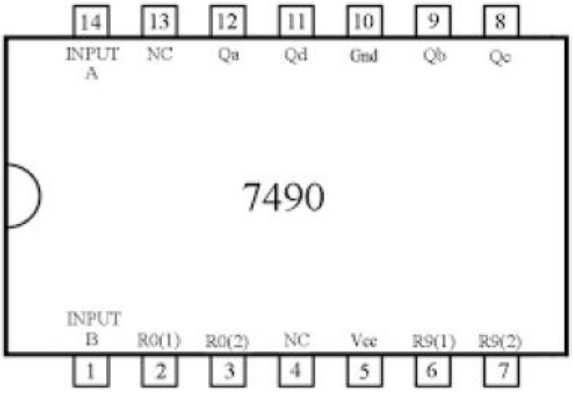
THEORY:

The counters four outputs are design at ed by the letter symbol Q with a numeric sub script equal to the binary weight of the corresponding bit in the BCD counter circuits code. So, for example, QA, QB, QC and QD. The 74LS90 counting sequence is triggered on the negative going edge of the clock signal, that is when the clock signal CLK goes from logic 1 (HIGH) to logic 0 (LOW).

The additional input pins R1 and R2 are counter "reset" pins while inputs S1 and S2 are "set" pins. When connected to logic 1, the Reset inputs R1 and R2 reset the counter back to zero, 0 (0000), and when the Set inputs S1 and S2 are connected to logic 1, they Set the counter to maximum, or 9 (1001) regardless of the actual count number or position.

As we said before, the 74LS90 counter consists of a divide-by-2 counter and a divide-by-5 counter with in the same package. Then we can use counter to produce a divide-by-2 frequency counter only, adivide-by-5 frequency counter only or the two together to produce our desired divide-by-10 BCD counter. With the four flip-flops making up the divide-by-5 counter section disabled, if a clock signal is applied to input pin 14 (CLKA) and the output taken from pin 12(QA), we can produce a standard divide-by-2 binary counter for use in Pin Diagram circuits as shown below.

Pin Diagram of 7490IC is as below:



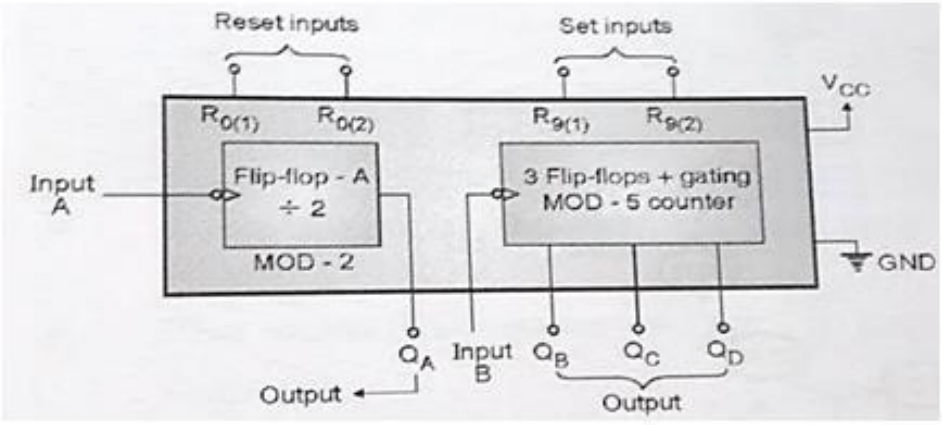
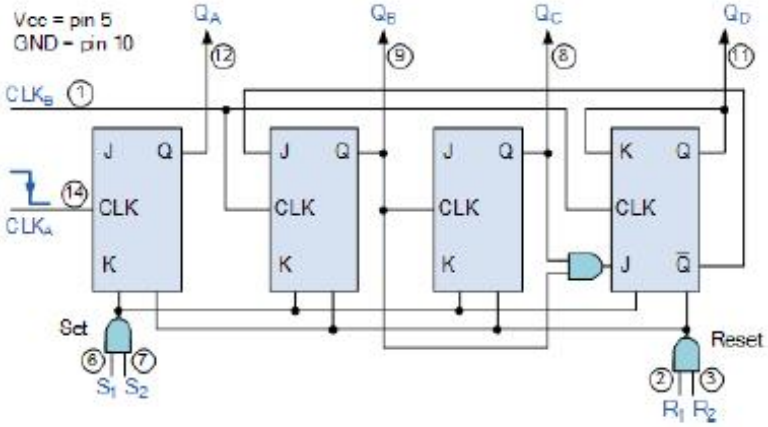
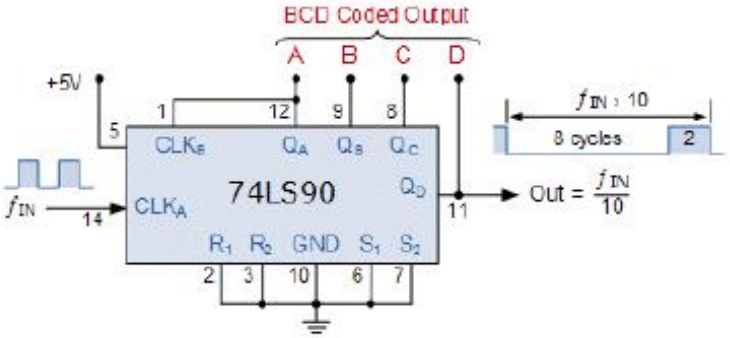


Fig10. The basic internal structure of IC 7490





PROCEDURE:

1. Connect inputs RESET0 (pin2), RESET0 (pin3), R3, R2 to the inputs witches and Qa, Qb, Qc, Qd to the outputs witches.
2. Connect Clock A to clock pulse.
3. Connect Clock B to Qa.
4. Feed the logic signal either H/L as shown in the truth table.
5. Monitor the outputs Qa,Qb,Qc,Qd.
6. Verify the truth table.

TRUTHTABLE:

ResetI/p				Outputs			
RESET0 (pin2)	RESET0 (pin3)	R3	R2	Qd	Qc	Qb	Qa
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

Conclusion: Hence verified the truth table of Decade Counter.

PART-2

Program 1: Implement a C program to convert a Hexadecimal, octal, and binary number to decimal number vice versa.

/*convert decimal to binary*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
long int num,i,j,bnum=0,dnum;
clrscr();
printf("\nEnter a decimal number : ");
scanf("%ld",&num);
dnum=num;
i=1;
for(j=num;j>0;j=j/2)
{
        bnum=bnum+(num%2)*i;
        i=i*10;
        num=num/2;
}
printf("\n The binary of %ld is %ld",dnum,bnum);
getch();
}
```

Output:

```
Enter a decimal number :  25
The binary of  25 is  11001
```

/*convert decimal to octal number*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
long int num,i,j,octnum=0,dnum;
clrscr();
printf("\nEnter a number : ");
scanf("%ld",&num);
dnum=num;
i=1;
for(j=num;j>0;j=j/8)
```

```
{
        octnum=octnum+(num%8)*i;
        i=i*10;
        num=num/8;
}
printf("\n The Octal of %ld is %ld",dnum,octnum);
getch();
}
```

Output:

```
Enter a number :  79
The Octal of 79 is  117
```

/* convert number from decimal to hexadecimal*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
long int dnum,rem,q,dn=0,i,j,temp;
char ch;
clrscr();
printf("\nEnter a decimal number : ");
scanf("%ld",&dnum);
q=dnum;
for(i=q;i>0;i=i/16)
{
        temp=i%16;
        if(temp<10)
                temp=temp+48;
        else
                temp=temp+55;
        dn=dn*100+temp;
}
printf("\nThe equivalent hexadecimal number is : ");
for(j=dn;j>0;j=j/100)
{
        ch=j%100;
        printf("%c",ch);
}
printf("\n");
```

```
getch();
}
```

Output:

```
Enter a decimal number :  79
The equivalent hexadecimal number is :  4F

Enter a decimal number :  775
The equivalent hexadecimal number is :  307
```

/*convert binary number to decimal*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
long int bnum;
long int decnum=0,i=0,d;
clrscr();
printf("\nEnter the binary number : ");
scanf("%ld",&bnum);
while(bnum!=0)
{
        d=bnum%10;
        decnum=decnum+d*pow(2,i);
        bnum=bnum/10;
        i++;
}
printf("\nThe equivalent decimal number is : %ld",decnum);
getch();
}
```

Output:

```
Enter the binary number : 1010100
The equivalent decimal number is : 84
```

/*convert octal number to decimal number*/

```
#include<stdio.h>
#include<conio.h>
```

```
#include<math.h>
void main()
{
int num,decnum=0,i=0,rem;
clrscr();
printf("\nEnter an octal number(using the digits 0---7) : ");
scanf("%d",&num);
while(num!=0)
{
        rem=num%10;
        decnum=decnum+rem*pow(8,i);
        num=num/10;
        i++;
}
printf("\nThe equivalent decimal number is : %d",decnum);
getch();
}
```

Output:

Enter an octal number(using the digits 0---7) : 745
The equivalent decimal number is : 485

Enter an octal number(using the digits 0---7) : 25
The equivalent decimal number is : 21

2. Binary Addition & Subtraction:

Aim: To write C program for sum of two binary numbers

Source Code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int c = 0,numa[8]={0},numb[8]={0},diff[8]={0},sum[8]={0};
void add();
void sub();
void main()
{
	int ch,i;
	do
	{

	for(i=0; i<8; i++)
			{
		numa[i] = 0;
		numb[i] = 0;
		diff[i] = 0;
		sum[i] = 0;
	}
	c = 0;
	printf("\tADDITION-SUBTRACTION USING TWO'S COMPLEMENT");
	printf("\n1.ADD\n2.SUBTRACT\nEnter your choice: ");
	scanf("%d",&ch);
		switch(ch){
			case 1: add();
				break;
			case 2: sub();
				break;
			default: printf("\nInvalid Choice: ");
		}
	printf("\nPress 1 to Continue...");
	scanf("%d",&ch);
	}while(ch == 1);
}
void add(){
	int i;
```

```
                printf("\nEnter two 8-bit binary numbers\n");
        printf("\nEnter first number: ");
        for(i = 0; i<8; i++)
                {
                scanf("%d",&numa[i]);
        }
        printf("\nEnter second number: ");
        for(i = 0; i<8; i++){
                scanf("%d",&numb[i]);
        }
        for(i = 7; i >= 0; i--){
                sum[i] = numa[i]+ numb[i] + c;
                if(sum[i]>=2){
                        c = 1;
                }
                else
                        c = 0;
                sum[i] = sum[i]%2;
        }
        printf("\nSum is: ");
        for(i = 0; i<8; i++){
                printf("%d",sum[i]);
        }
}
void sub()
{
        int i;
        printf("\nEnter two 8-bit binary numbers\n");
        printf("\nEnter first number: ");
        for(i = 0; i<8; i++){
                scanf("%d",&numa[i]);
        }
        printf("\nEnter second number: ");
        for(i = 0; i<8; i++){
                scanf("%d",&numb[i]);
        }
        for(i = 7; i >= 0; i--){
                diff[i] = numa[i] - numb[i];
                if(diff[i] < 0){
                        numa[i-1] = numa[i-1] - 1;
                }
```

```
                diff[i] = fabs(diff[i]%2);
        }
        printf("\nDifference is: ");
        for(i = 0; i<8; i++){
                printf("%d",diff[i]);
        }
}
```

Task 3: Implement a C program to perform Multiplication of two binary numbers

```
#include <stdio.h>

int binaryproduct(int, int);

int main()
{

   long binary1, binary2, multiply = 0;
   int digit, factor = 1;

   printf("Enter the first binary number: ");
   scanf("%ld", &binary1);
   printf("Enter the second binary number: ");
   scanf("%ld", &binary2);
   while (binary2 != 0)
   {
      digit =  binary2 % 10;
      if (digit == 1)
      {
         binary1 = binary1 * factor;
         multiply = binaryproduct(binary1, multiply);
      }
      else
         binary1 = binary1 * factor;
      binary2 = binary2 / 10;
      factor = 10;
   }
   printf("Product of two binary numbers: %ld", multiply);
   return 0;
}
```

```
int binaryproduct(int binary1, int binary2)
{
   int i = 0, remainder = 0, sum[20];
   int binaryprod = 0;

   while (binary1 != 0 || binary2 != 0)
   {
      sum[i++] =(binary1 % 10 + binary2 % 10 + remainder) % 2;
      remainder =(binary1 % 10 + binary2 % 10 + remainder) / 2;
      binary1 = binary1 / 10;
      binary2 = binary2 / 10;
   }
   if (remainder != 0)
      sum[i++] = remainder;
   --i;
   while (i >= 0)
      binaryprod = binaryprod * 10 + sum[i--];
   return binaryprod;
}
```

Task 4: Multiplication of two binary numbers (signed) using Booth's Algorithms.

```
#include <stdio.h>
#include <math.h>

int a = 0,b = 0, c = 0, a1 = 0, b1 = 0, com[5] = { 1, 0, 0, 0, 0};
int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};
int acomp[5] = {0}, bcomp[5] = {0}, pro[5] = {0}, res[5] = {0};

void binary(){
        a1 = fabs(a);
        b1 = fabs(b);
        int r, r2, i, temp;
        for (i = 0; i < 5; i++){
                r = a1 % 2;
                a1 = a1 / 2;
                r2 = b1 % 2;
```

```
                b1 = b1 / 2;
                anum[i] = r;
                anumcp[i] = r;
                bnum[i] = r2;
                if(r2 == 0){
                        bcomp[i] = 1;
                }
                if(r == 0){
                        acomp[i] =1;
                }
  }
//part for two's complementing
c = 0;
for ( i = 0; i < 5; i++){
                res[i] = com[i]+ bcomp[i] + c;
                if(res[i] >= 2){
                        c = 1;
                }
                else
                        c = 0;
                res[i] = res[i] % 2;
  }
for (i = 4; i >= 0; i--){
  bcomp[i] = res[i];
}
//in case of negative inputs
if (a < 0){
        c = 0;
  for (i = 4; i >= 0; i--){
                res[i] = 0;
  }
  for ( i = 0; i < 5; i++){
                res[i] = com[i] + acomp[i] + c;
                if (res[i] >= 2){
                        c = 1;
                }
                else
                        c = 0;
                res[i] = res[i]%2;
  }
  for (i = 4; i >= 0; i--){
```

```
                        anum[i] = res[i];
                        anumcp[i] = res[i];
            }

          }
          if(b < 0){
            for (i = 0; i < 5; i++){
                        temp = bnum[i];
                        bnum[i] = bcomp[i];
                        bcomp[i] = temp;
            }
          }
}
void add(int num[]){
           int i;
           c = 0;
           for ( i = 0; i < 5; i++){
                        res[i] = pro[i] + num[i] + c;
                        if (res[i] >= 2){
                                    c = 1;
                        }
                        else{
                                    c = 0;
                        }
                        res[i] = res[i]%2;
            }
            for (i = 4; i >= 0; i--){
                       pro[i] = res[i];
                       printf("%d",pro[i]);
            }
          printf(":");
          for (i = 4; i >= 0; i--){
                        printf("%d", anumcp[i]);
            }
}
void arshift(){//for arithmetic shift right
           int temp = pro[4], temp2 = pro[0], i;
           for (i = 1; i < 5  ; i++){//shift the MSB of product
                    pro[i-1] = pro[i];
           }
           pro[4] = temp;
```

```
        for (i = 1; i < 5  ; i++){//shift the LSB of product
                anumcp[i-1] = anumcp[i];
        }
        anumcp[4] = temp2;
        printf("\nAR-SHIFT: ");//display together
        for (i = 4; i >= 0; i--){
                printf("%d",pro[i]);
        }
        printf(":");
        for(i = 4; i >= 0; i--){
                printf("%d", anumcp[i]);
        }
}

void main(){
        int i, q = 0;
        printf("\t\tBOOTH'S MULTIPLICATION ALGORITHM");
        printf("\nEnter two numbers to multiply: ");
        printf("\nBoth must be less than 16");
        //simulating for two numbers each below 16
        do{
                printf("\nEnter A: ");
                scanf("%d",&a);
                printf("Enter B: ");
                scanf("%d", &b);
         }while(a >=16 || b >=16);

        printf("\nExpected product = %d", a * b);
        binary();
        printf("\n\nBinary Equivalents are: ");
        printf("\nA = ");
        for (i = 4; i >= 0; i--){
                printf("%d", anum[i]);
        }
        printf("\nB = ");
        for (i = 4; i >= 0; i--){
                printf("%d", bnum[i]);
        }
        printf("\nB'+ 1 = ");
        for (i = 4; i >= 0; i--){
                printf("%d", bcomp[i]);
```

```
        }
        printf("\n\n");
        for (i = 0;i < 5; i++){
                        if (anum[i] == q){//just shift for 00 or 11
                                printf("\n-->");
                                arshift();
                                q = anum[i];
                        }
                        else if(anum[i] == 1 && q == 0){//subtract and shift for 10
                                printf("\n-->");
                                printf("\nSUB B: ");
                                add(bcomp);//add two's complement to implement subtraction
                                arshift();
                                q = anum[i];
                        }
                        else{//add ans shift for 01
                                printf("\n-->");
                                printf("\nADD B: ");
                                add(bnum);
                                arshift();
                                q = anum[i];
                        }
        }

        printf("\nProduct is = ");
        for (i = 4; i >= 0; i--){
                        printf("%d", pro[i]);
        }
        for (i = 4; i >= 0; i--){
                        printf("%d", anumcp[i]);
        }
}
```

Output:

```
(Inactive C:\TC
Enter A: -13
Enter B: +11

Expected product = -143

Binary Equivalents are:
A = 10011
B = 01011
B'+ 1 = 10101


-->
SUB B: 10101:10011
AR-SHIFT: 11010:11001
-->
AR-SHIFT: 11101:01100
-->
ADD B: 01000:01100
AR-SHIFT: 00100:00110
-->
AR-SHIFT: 00010:00011
-->
SUB B: 10111:00011
AR-SHIFT: 11011:10001
Product is = 1101110001
```

Task 5: Implement a C program to perform division of two binary numbers (Unsigned) using restoring division algorithm.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int a=0,b=0,c=0,com[5]={1,0,0,0,0},s=0;
int anum[5]={0},anumcp[5] ={0},bnum[5]={0};
int acomp[5]={0},bcomp[5]={0},rem[5]={0},quo[5]={0},res[5]={0};

void binary(){
    a = fabs(a);
    b = fabs(b);
    int r, r2, i, temp;
    for(i = 0; i < 5; i++){
        r = a % 2;
        a = a / 2;
        r2 = b % 2;
        b = b / 2;
        anum[i] = r;
        anumcp[i] = r;
        bnum[i] = r2;
        if(r2 == 0){
            bcomp[i] = 1;
        }
        if(r == 0){
            acomp[i] =1;
        }
    }
  //part for two's complementing
  c = 0;
  for( i = 0; i < 5; i++){
        res[i] = com[i]+ bcomp[i] + c;
        if(res[i]>=2){
            c = 1;
        }
        else
            c = 0;
        res[i] = res[i]%2;
    }
  for(i = 4; i>= 0; i--){
```

```
    bcomp[i] = res[i];
   }
}
void add(int num[]){
    int i;
    c = 0;
    for( i = 0; i < 5; i++){
        res[i] = rem[i]+ num[i] + c;
        if(res[i]>=2){
            c = 1;
        }
        else
            c = 0;
        res[i] = res[i]%2;
    }
    for(i = 4; i>= 0; i--){
        rem[i] = res[i];
        printf("%d",rem[i]);
    }
    printf(":");
    for(i = 4; i>= 0; i--){
        printf("%d",anumcp[i]);
    }
}
void shl(){//for shift left
    int i;
    for(i = 4; i > 0  ; i--){//shift the remainder
        rem[i] = rem[i-1];
    }
    rem[0] = anumcp[4];
    for(i = 4; i > 0  ; i--){//shift the remtient
        anumcp[i] = anumcp[i-1];
    }
    anumcp[0] = 0;
    printf("\nSHIFT LEFT: ");//display together
    for(i = 4; i>= 0; i--){
        printf("%d",rem[i]);
    }
    printf(":");
    for(i = 4; i>= 0; i--){
        printf("%d",anumcp[i]);
```

```
    }
}

void main(){
    clrscr();
    int i;
    printf("\t\tRESTORING DIVISION ALGORITHM");
    printf("\nEnter two numbers to multiply: ");
    printf("\nBoth must be less than 16");
    //simulating for two numbers each below 16
    do{
        printf("\nEnter A: ");
        scanf("%d",&a);
        printf("Enter B: ");
        scanf("%d",&b);
    }while(a>=16 || b>=16);

    printf("\nExpected Quotient = %d", a/b);
    printf("\nExpected Remainder = %d", a%b);
    if(a*b <0){
        s = 1;
    }

    binary();
    printf("\n\nUnsigned Binary Equivalents are: ");
    printf("\nA = ");
    for(i = 4; i>= 0; i--){
        printf("%d",anum[i]);
    }
    printf("\nB = ");
    for(i = 4; i>= 0; i--){
        printf("%d",bnum[i]);
    }
    printf("\nB'+ 1 = ");
    for(i = 4; i>= 0; i--){
        printf("%d",bcomp[i]);
    }
    printf("\n\n-->");
    //division part
    shl();
    for(i=0;i<5;i++){
```

```
        printf("\n-->"); //start with subtraction
        printf("\nSUB B: ");
        add(bcomp);
        if(rem[4]==1){//simply add for restoring
            printf("\n-->RESTORE");
            printf("\nADD B: ");
            anumcp[0] = 0;
            add(bnum);
        }
        else{
            anumcp[0] = 1;
        }
        if(i<4)
            shl();

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

}
```

Task 6: Implement a C program to perform division of two binary numbers (Unsigned) using non-restoring division algorithm.

```
#include<stdio.h>
void comp();
void nonresdiv();
void shiftleft();
void a_minus_b();
void a_plus_b();
```

```
int a[5]={0,0,0,0,0},q[4],b[5],b2c[5];

void comp()
{
int i=4;
do
{
b2c[i]=b[i];
i--;
}while(b[i+1]!=1);
while(i>=0)
{
b2c[i]=(b[i]+1)%2;
i--;
}
printf("\n\tB's complement:");
for(i=0;i< 5;i++)
printf("%d",b2c[i]);
printf("\n");
}


void nonresdiv()
{
shiftleft();
if(a[0]==0)
a_minus_b();
else
a_plus_b();
q[3]=(a[0]+1)%2;
}

void shiftleft()
{
int i;
for(i=0;i< 4;i++)
a[i]=a[i+1];
a[4]=q[0];
for(i=0;i< 3;i++)
q[i]=q[i+1];
}
```

```
void a_minus_b()
{
int i,carry=0,sum=0;
for(i=4;i>=0;i--)
{
sum=(a[i]+b2c[i]+carry);
a[i]=sum%2;
carry=sum/2;
}
}

void a_plus_b()
{
int i,carry=0,sum=0;
for(i=4;i>=0;i--)
{
sum=(a[i]+b[i]+carry);
a[i]=sum%2;
carry=sum/2;
}
}


void main()
{
int i,j,k;
printf("Enter dividend in binary form\t: ");
for(i=0;i< 4;i++)
scanf("%d",&q[i]);
printf("Enter divisor in binary form\t: ");
for(i=0;i< 5;i++)
scanf("%d",&b[i]);
comp();
printf("\n\t[A]\t[M]\n");
for(i=0;i< 4;i++)
{
nonresdiv();
printf("\t");
for(j=0;j< 5;j++)
printf("%d",a[j]);
```

```
printf("\t");
for(k=0;k< 4;k++)
printf("%d",q[k]);
printf("\n");
}
if(a[0]==1)
a_plus_b();
printf("\t");
for(j=0;j< 5;j++)
printf("%d",a[j]);
printf("\t");
for(k=0;k< 4;k++)
printf("%d",q[k]);
printf("\n");
printf("\n\tThe Quotient Is\t: ");
for(k=0;k< 4;k++)
printf("%d",q[k]);
printf("\n\tThe Remainder Is\t: ");
for(j=0;j< 5;j++)
printf("%d",a[j]);
getch();
}
```