

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY  
BELAGAVI – 590 010**



**DIGITAL DESIGN AND HDL  
(17EIL38)**

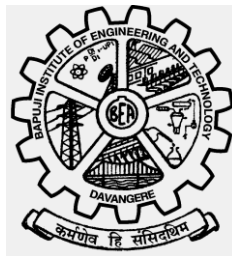
***LABORATORY MANUAL***

**III Semester - B.E.**

Prepared By

**Manjunath K.G., Shubha V. Patel**

**Department of Electronics and Instrumentation  
Engineering**



**Bapuji Institute of Engineering and Technology,  
Davangere - 577 004, Karnataka.**

**EXPERIMENT 1****A). Verification of Basic gates & Universal Gates using their respective Truth Table.****Components Required:**

Serial No.	Component Description	IC No.	Required No.
1	AND Gate	7408	1
2	OR Gate	7432	1
3	Exclusive OR Gate	7486	1
4	NAND Gate	7400	1
5	NOR Gate	7402	1
6	NOT Gate	7404	1
7	Patch Chords	----	10
8	IC Trainer Kit	----	1

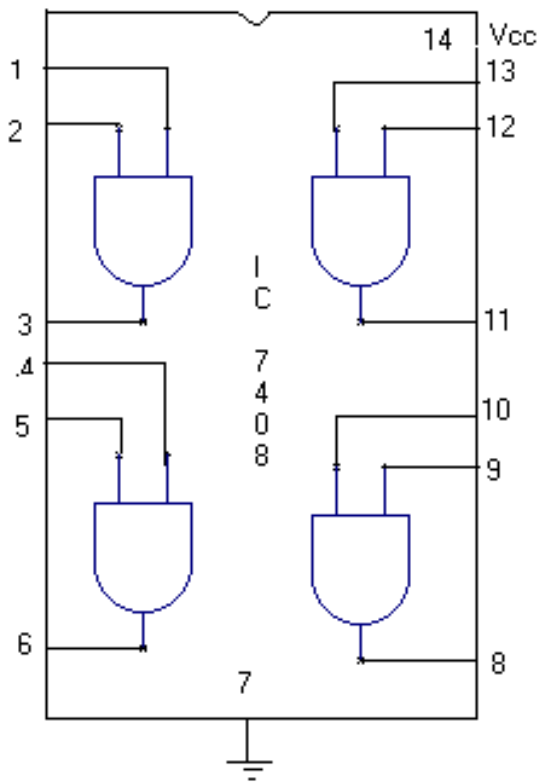
**Procedure:**

1. Check all the components and patch chords whether they are in good condition.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in pin details.
4. Connect pin no.14 to +5V & Pin no. 7 to Gnd.
5. Verify the connections and turn on the Trainer Kit.
6. Verify the Truth Table & observe the Output.

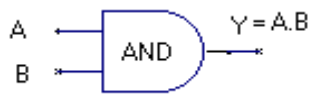
Observation:

Result:

**1. Quad 2-input AND Gate.**



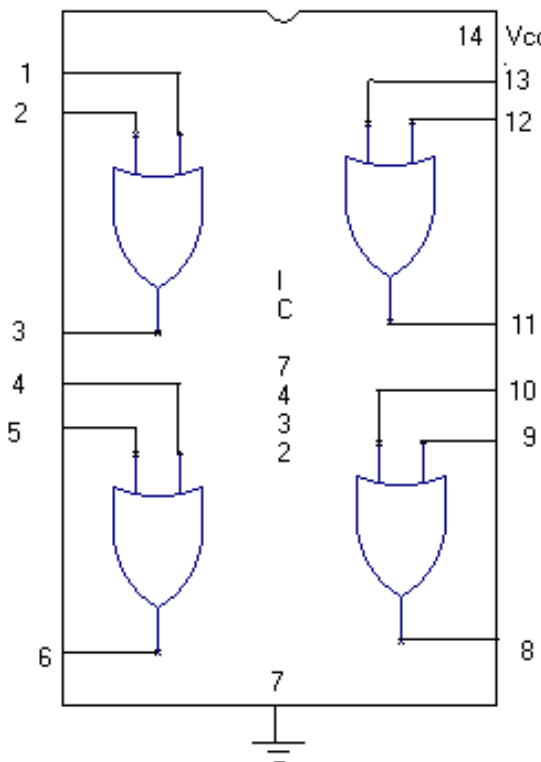
**Logic Symbol**



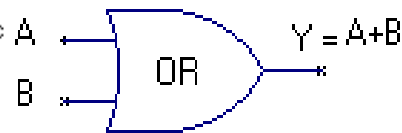
**Truth Table**

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

**2. Quad 2 input OR Gate**



**Logic Symbol**



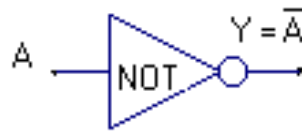
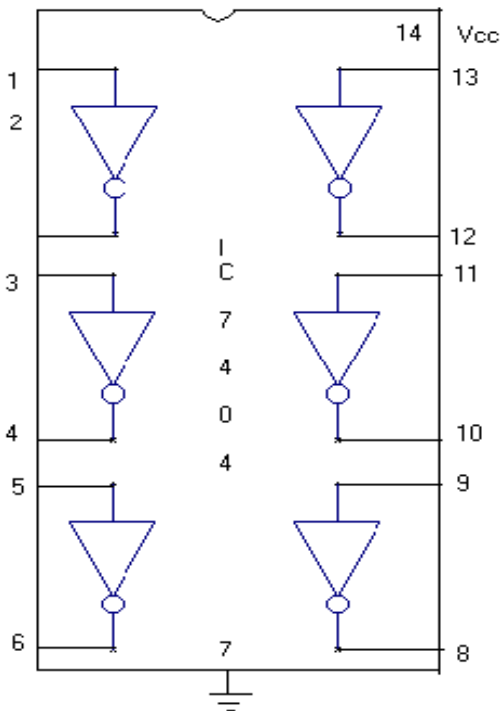
**Truth Table**

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

**3. Hex inverter- NOT Gate**

**Logic Symbol**

**Truth Table**

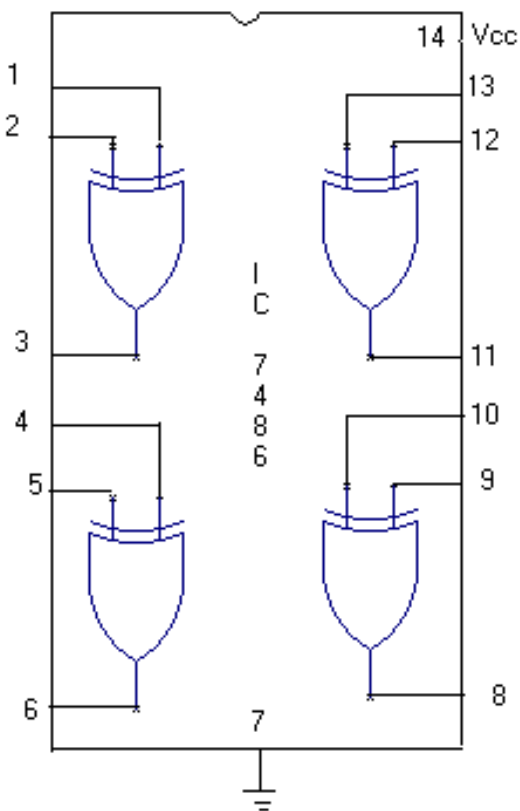


Input	Output
A	Y
0	1
1	0

**4. Quad 2 input Ex-OR Gate**

**Logic Symbol**

**Truth Table**

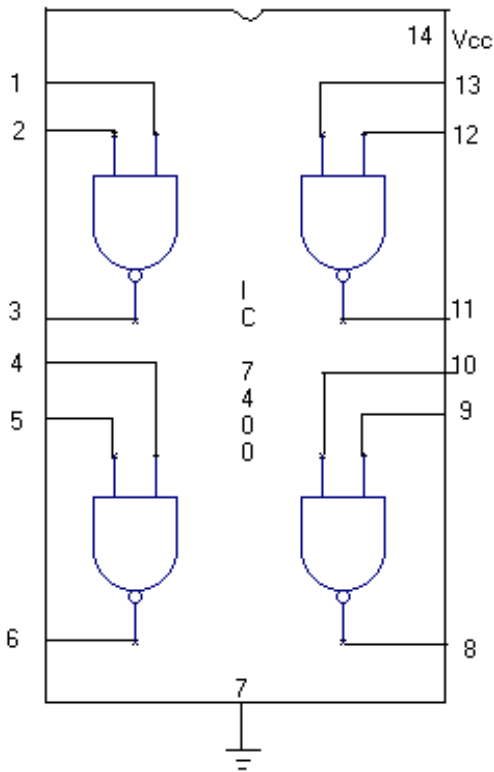


Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

**5. Quad 2 input NAND Gate.**

**Logic Symbol**

**Truth Table**

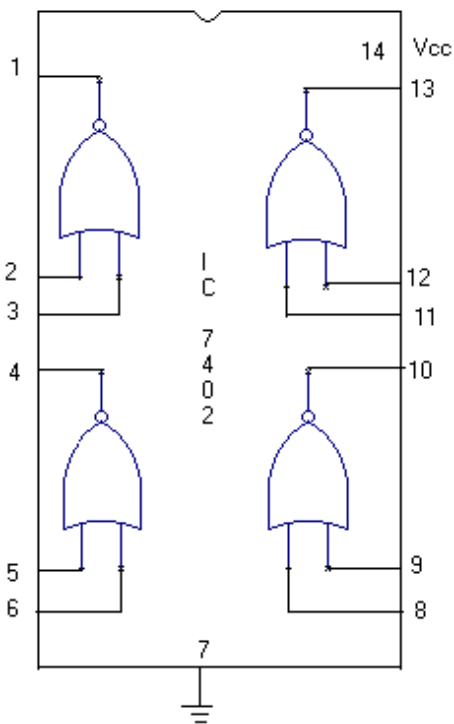


Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

**5. Quad 2 input NOR Gate.**

**Logic Symbol**

**Truth Table**

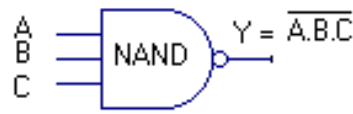
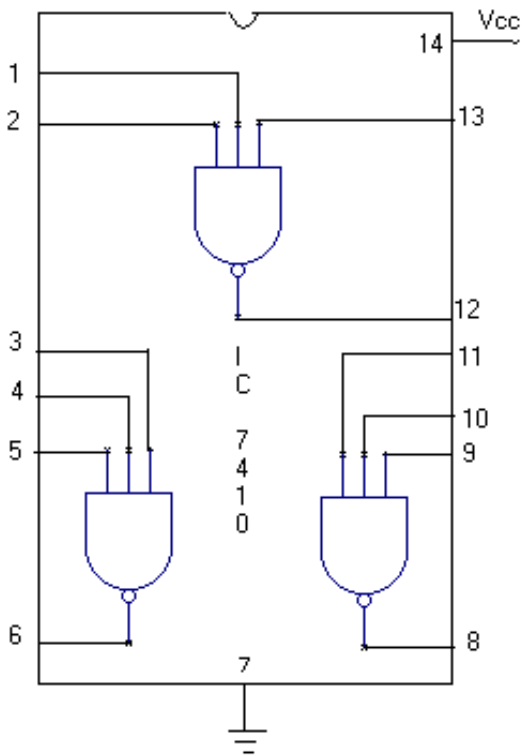


Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

7.3 input NAND Gate

Logic Symbol

Truth Table

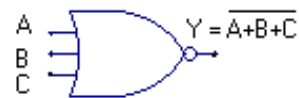
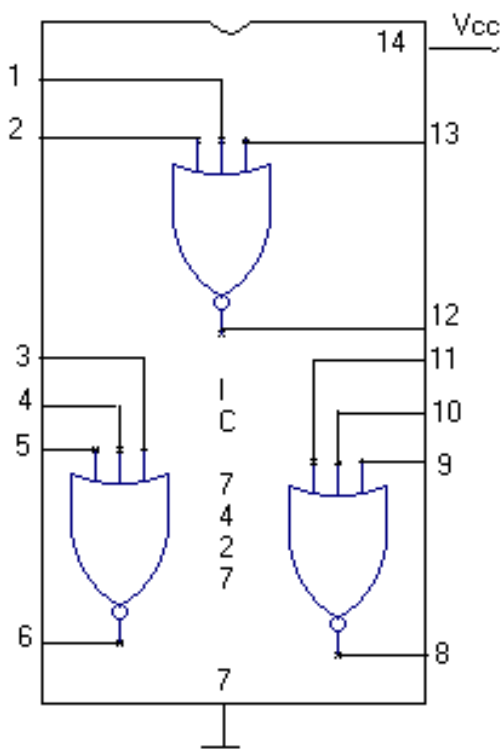


Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

8.3 INPUT NOR Gate.

Logic Symbol

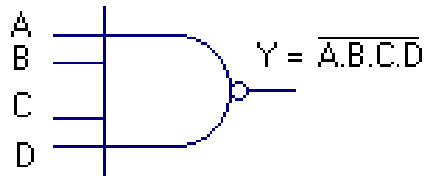
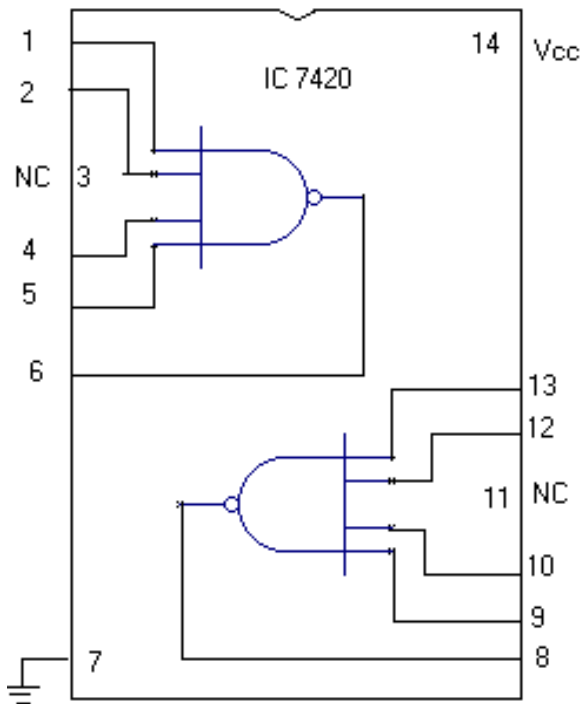
Truth Table



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

9. 4-input NAND Gate.

Logic Symbol



**B). Verification of Logic Gates Using Universal Gates.****Components Required:**

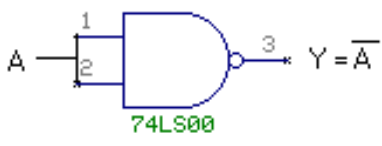

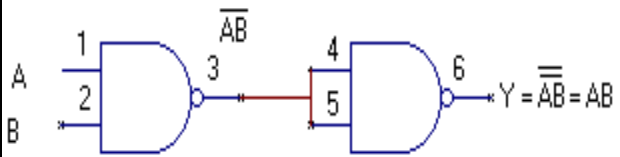
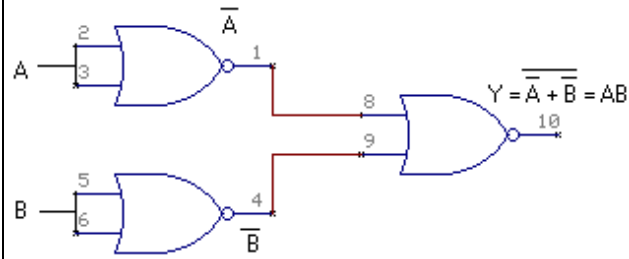
Serial No.	Component Description	IC No.	Required No.
1	NAND Gate	7400	1
2	NOR Gate	7402	2
3	Patch Chords	----	10
4	IC Trainer Kit	----	1

**Procedure:**

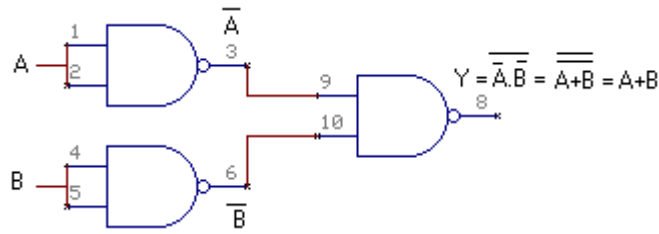
1. Check all the components and patch chords whether they are in good condition.
2. Insert the appropriate IC into the IC base.
3. Make connection as shown in pin details.
4. Connect pin no.14 to +5V & Pin no. 7 to Gnd.
5. Give supply to the Trainer Kit.
6. Verify the Truth Table & observe the Output



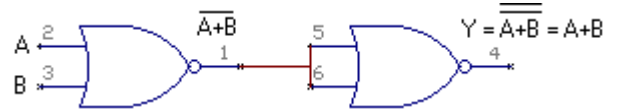
**Verification of Basic Gates using Universal Gates.**

Function	Using NAND Gates	Using NOR Gates																																				
1. NOT Gate	 <table border="1" data-bbox="598 627 853 862"> <thead> <tr> <th colspan="2">Input</th> <th>Output</th> </tr> <tr> <th>A</th> <th>t</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td></td> <td>1</td> </tr> <tr> <td>1</td> <td></td> <td>0</td> </tr> </tbody> </table>	Input		Output	A	t	Y	0		1	1		0	 <table border="1" data-bbox="1236 627 1492 862"> <thead> <tr> <th colspan="2">Input</th> <th>Output</th> </tr> <tr> <th>A</th> <th>t</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td></td> <td>1</td> </tr> <tr> <td>1</td> <td></td> <td>0</td> </tr> </tbody> </table>	Input		Output	A	t	Y	0		1	1		0												
Input		Output																																				
A	t	Y																																				
0		1																																				
1		0																																				
Input		Output																																				
A	t	Y																																				
0		1																																				
1		0																																				
2. AND Gate	 <table border="1" data-bbox="574 1456 869 1859"> <thead> <tr> <th colspan="2">Input</th> <th>Output</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	Input		Output	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	 <table border="1" data-bbox="1260 1456 1556 1859"> <thead> <tr> <th colspan="2">Input</th> <th>Output</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	Input		Output	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
Input		Output																																				
A	B	Y																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
Input		Output																																				
A	B	Y																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				

3. OR Gate

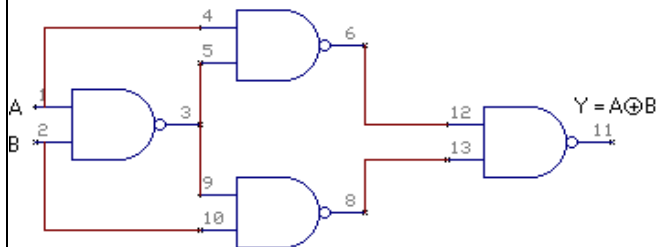


Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

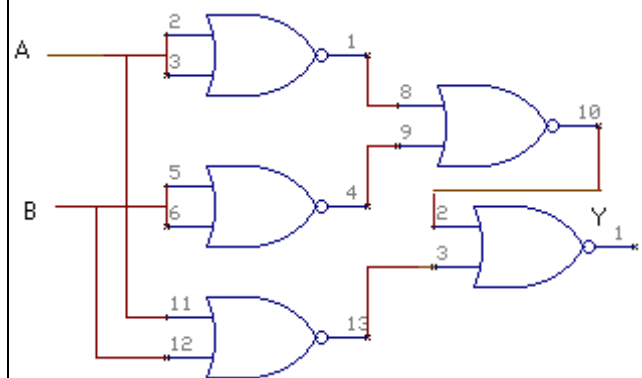


Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

4. Ex-OR Gate



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

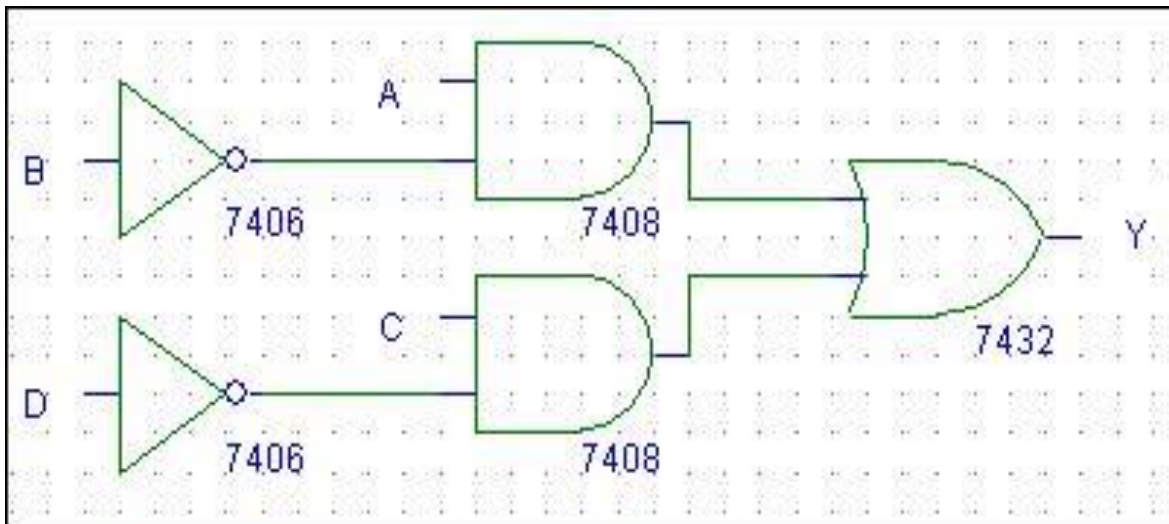
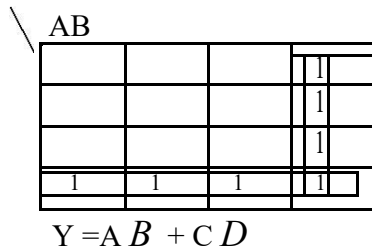


Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

**C). REALIZATION OF A BOOLEAN EXPRESSIONS.**

Realization of Boolean expression:

1)  $Y = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABC\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$



**TRUTH TABLE**

INPUTS				OUTPUT
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

## EXPERIMENT 2

### Design and Implement

#### A) Half / Full Adder Half / Full Subtractor using Logic Gates

#### Components Required:

Serial No.	Component Description	IC No.	Required No.
1.	AND Gate	7408	1
2.	Ex-OR Gate	7486	1
3.	NOT Gate	7404	1
4.	OR Gate	7432	1
1	NAND Gate	7400	3
2	NOR Gate	7402	3
3	Patch Chords	----	30
4	IC Trainer Kit	----	1

#### Procedure:

1. Check all the components and patch chords whether they are in good condition.
2. Develop the Truth Table.
3. Write the Boolean Expressions.
4. Make connection as shown in circuit diagram.
5. Connect pin no.14 to +5V & Pin no. 7 to Gnd of each IC.
6. Give supply to the Trainer Kit.
7. Provide the input data to the circuit via switches.
8. Verify the truth table sequence and observe the output.

1. HALF ADDER

**Truth Table  
Circuit  
Diagram**

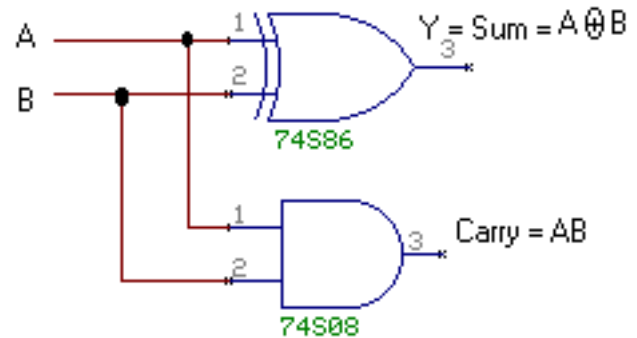
Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Using Basic Gates.

$$\text{Sum} = A\bar{B} + \bar{A}B = A \oplus B$$

$$\text{Carry} = AB$$

**Design**



2. FULL ADDER

**Design**

Inputs			Outputs	
A	B	C	S	C
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

**Using Basic Gates**

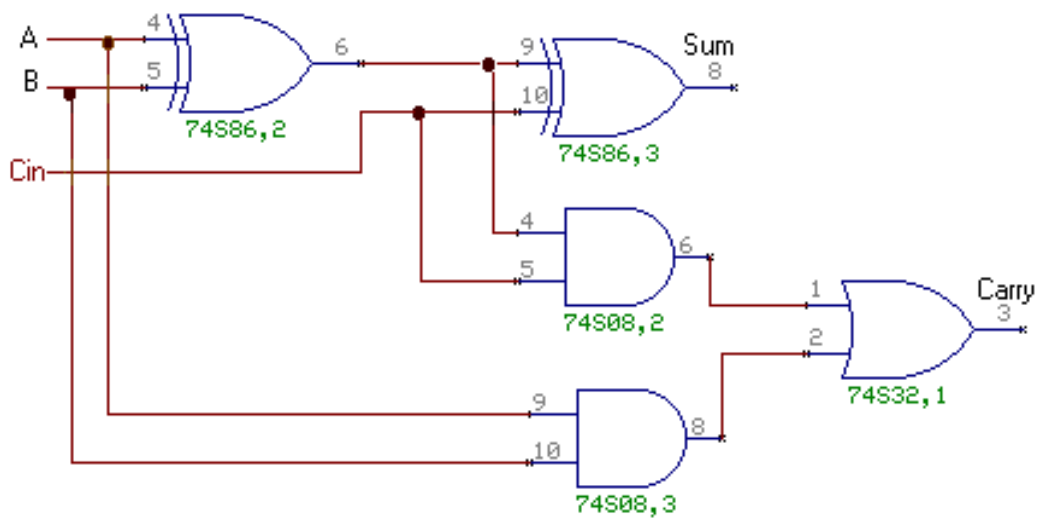
$$\begin{aligned} \text{Sum} &= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} \\ &= \bar{A}(\bar{B}C_{in} + B\bar{C}_{in}) + A(\bar{B}\bar{C}_{in} + BC_{in}) \\ &= \bar{A}(B \oplus C_{in}) + A(B \oplus C_{in}) \end{aligned}$$

Let  $x = B \oplus C_{in}$

$$\begin{aligned} \text{Sum} &= \bar{A}x + Ax \\ &= A \oplus x \end{aligned}$$

Now,  $\text{Sum} = A \oplus B \oplus C_{in}$

$$\begin{aligned} \text{Carry} &= \bar{A}BC_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}C_{in} + ABC_{in} \\ &= C_{in}(\bar{A}B + AB) + AB(\bar{C}_{in} + C_{in}) \\ &= C_{in}(A \oplus B) + AB \\ &= C_{in}(A \oplus B) \cdot \overline{AB} \\ &= (C_{in} + x) \cdot (A + B) \\ &= (C_{in} + x) + (\overline{A + B}) \end{aligned}$$



### 3. HALF SUBTRACTOR

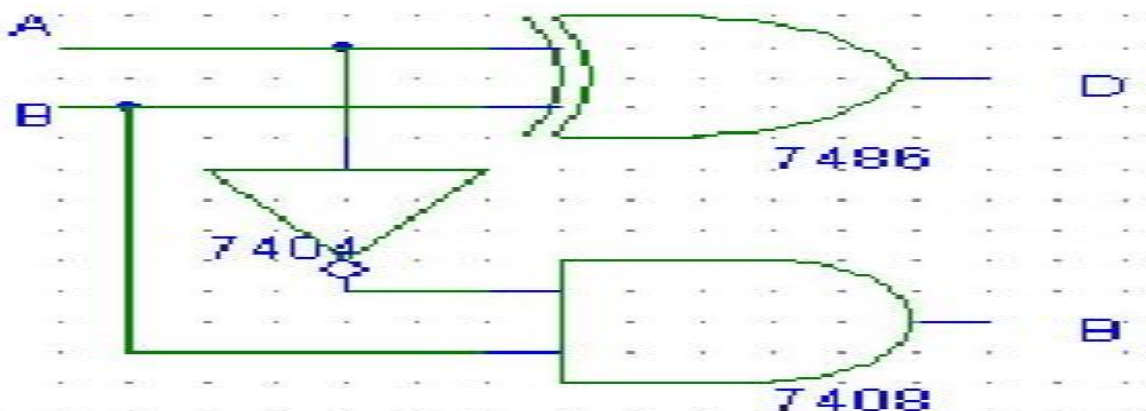
#### TRUTH TABLE

INPUTS		OUTPUTS	
A	B	D	Br
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

#### BOOLEAN EXPRESSIONS:

$$D = A \oplus B$$

$$Br = \bar{A} B$$



4. FULL SUBTRACTOR

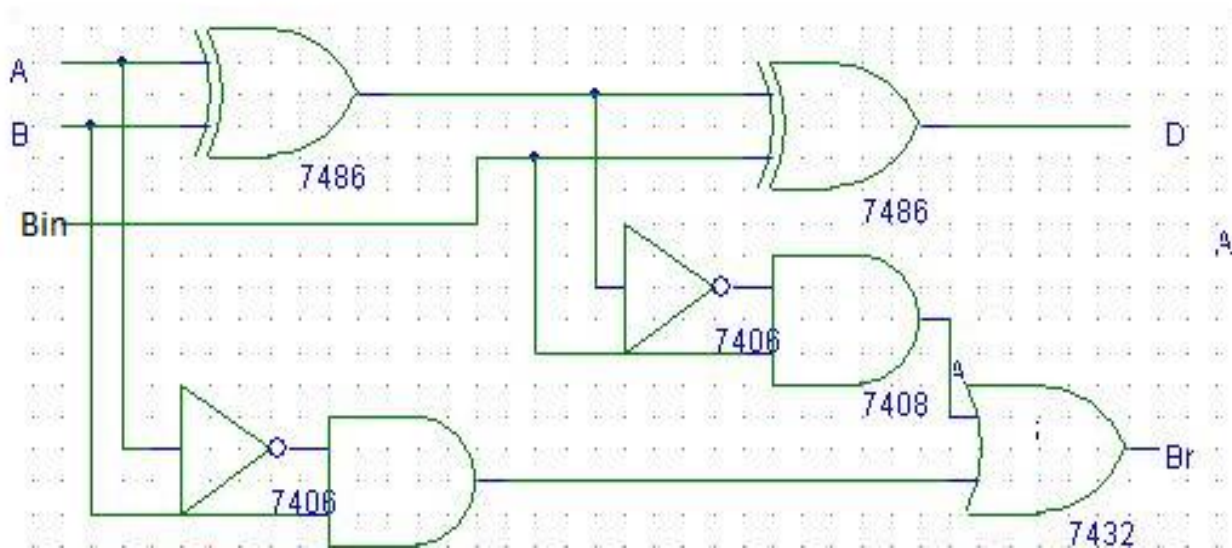
TRUTH TABLE

INPUTS			OUTPUTS	
A	B	Bin	D	Br
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

BOOLEAN EXPRESSIONS:

$$D = A \oplus B \oplus Bin$$

$$Br = \bar{A} B + B Bin + \bar{A} Bin$$



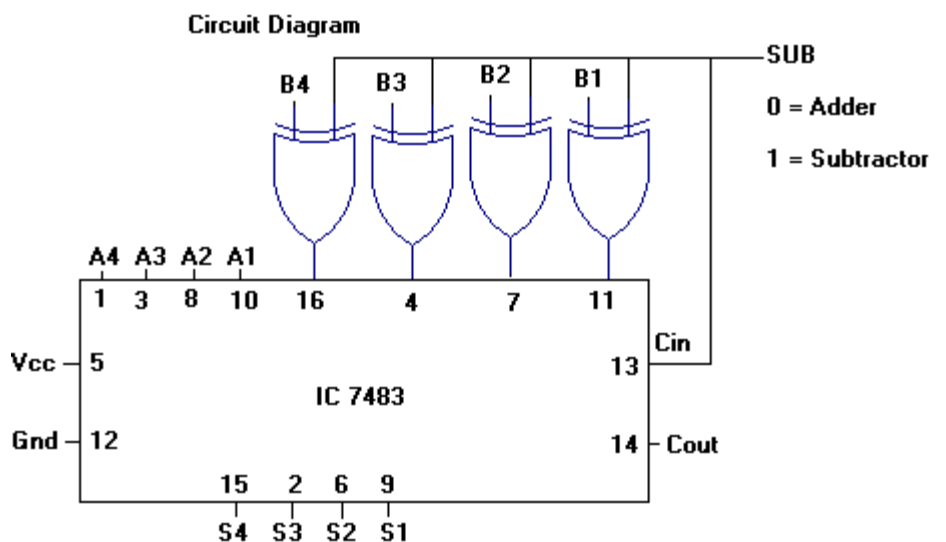
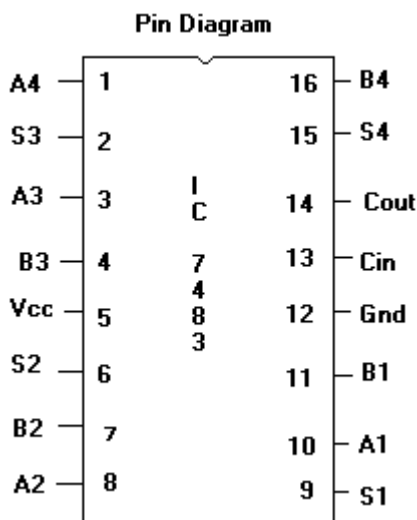
**B). Design and realize the 4-bit Adder / Subtractor Circuit using IC 7483.**

**Components Required:**

Serial No.	Component Description	IC No.	Required No.
1	4-bit Parallel Adder Chip	7483	1
2	Ex-OR Gate	7486	1
3	Patch Chords	----	15
4	IC Trainer Kit	----	1

**Procedure:**

1. Check all the components IC Packages using multimeter and digital IC Tester.
2. Make the circuit connection as shown in the diagram starting from the source and gnd pin connections.
3. Setup Adder/ Subtractor circuit. Make Sub = 0 and verify whether it works as a nibble adder.
4. To function as subtractor, make Sub = 1 and verify whether the circuit works as a 4-bit Subtractor of not. Try out some examples.
5. Give supply to the Trainer Kit.
6. Provide the input data to the circuit via switches.
7. Verify the truth table sequence and observe the output.





**4-bit Adder.**

Sl No	A4	A3	A2	A1	B4	B3	B2	B1	Cout	S4	S3	S2	S1

**4-bit Subtractor.**

Sl No	A4	A3	A2	A1	B4	B3	B2	B1	Cout	S4	S3	S2	S1

**EXPERIMENT 3****A). Design and realize BCD to Excess-3 Code converter and vice - versa.****Components Required:**

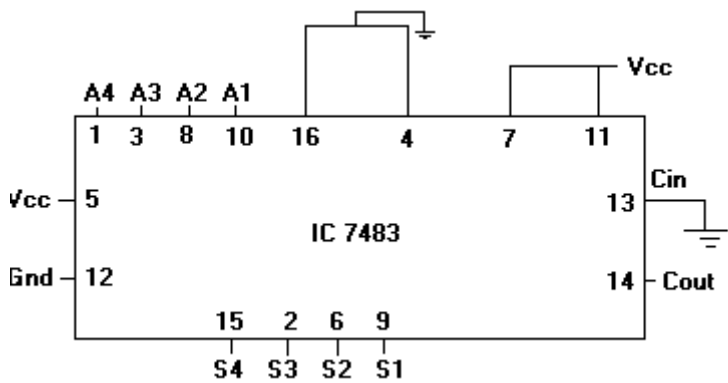
Serial No.	Component Description	IC No.	Required No.
1	4-bit Parallel Adder Chip	7483	1
2	Patch Chords	----	15
3	IC Trainer Kit	----	1

**Procedure:**

1. Verify all the components and patchchords whether they are in good condition.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit.
4. For different Excess-3 code inputs, verify the corresponding BCD data outputs using truth table.
5. And also for different BCD data inputs, verify the corresponding Excess-3 Code inputs using truth table.

**1. BCD to Excess-3 Code Converter**

**Truth Table**

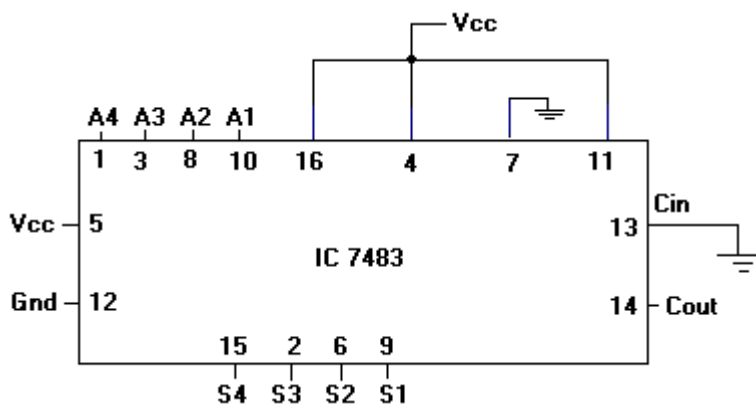


Input BCD Code				Output Excess-3 Code			
A4	A3	A2	A1	S4	S3	S2	S1

2.

**2. Excess-3 Code to BCD Code Converter**

**Truth Table**



Input Excess-3Code				Output BCD Code			
A4	A3	A2	A1	S4	S3	S2	S1

**B). Design and realize Binary to Gray Code converter and Viceversa.**

**Components Required:**

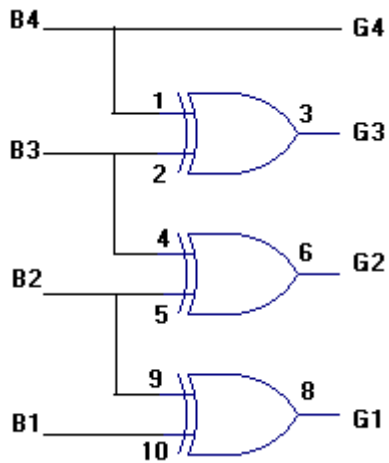
Serial No.	Component Description	IC No.	Required No.
1	Ex-OR Gates	7486	1
2	Patch Chords	----	15
3	IC Trainer Kit	----	1

**Procedure:**

1. Verify all the components and patch chords whether they are in good condition.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit.
4. For different binary data inputs, verify the corresponding Gray code outputs with the help of truth table.
5. And also for different Gray code inputs, verify the corresponding Binary Code output with the help of truth table.

**Binary to Gray Code Converter.**

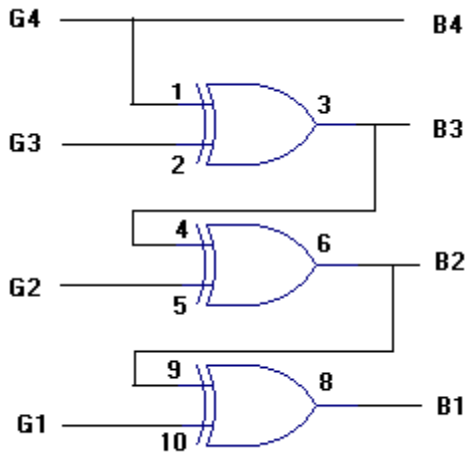
**Truth Table**



Binary Code				Gray Code			
B4	B3	B2	B1	G4	G3	G2	G1
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

**Gray Code to Binary Code Converter.**

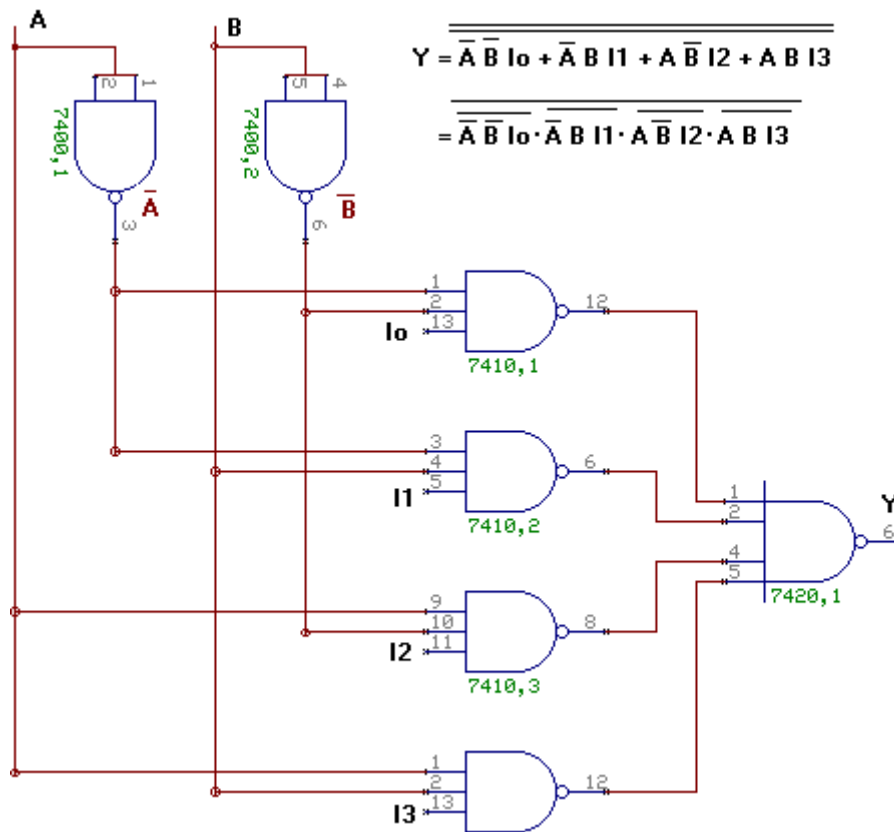
**Truth Table.**



Gray Code				Binary Code			
G4	G3	G2	G1	B4	B3	B2	B1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	0
1	1	1	1	1	0	0	0

## EXPERIMENT 4

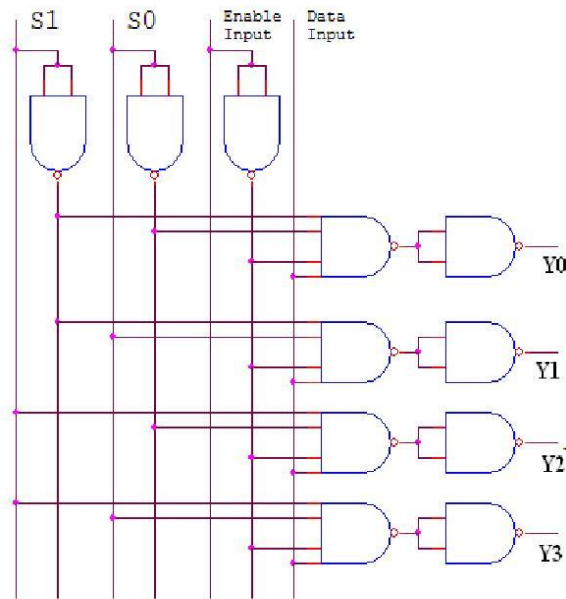
### A). 4:1 Multiplexer using gates



#### Truth Table:

Select Input		Output
A	B	Y
0	0	I <sub>0</sub>
0	1	I <sub>1</sub>
1	0	I <sub>2</sub>
1	1	I <sub>3</sub>

**B). DE-MUX USING GATES**



Enable Inputs	Data Input	Select Inputs		Outputs			
		S1	S0	Y3	Y2	Y1	Y0
1	0	X	X	X	X	X	X
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	0	0



### C). Priority encoder and 3:8 Decoder using IC74138

- 1) To setup a circuit of Decimal-to-BCD Encoder using IC 74147
- 2) To Setup a Circuit of Octal-to-Binary Encoder using IC 74148.

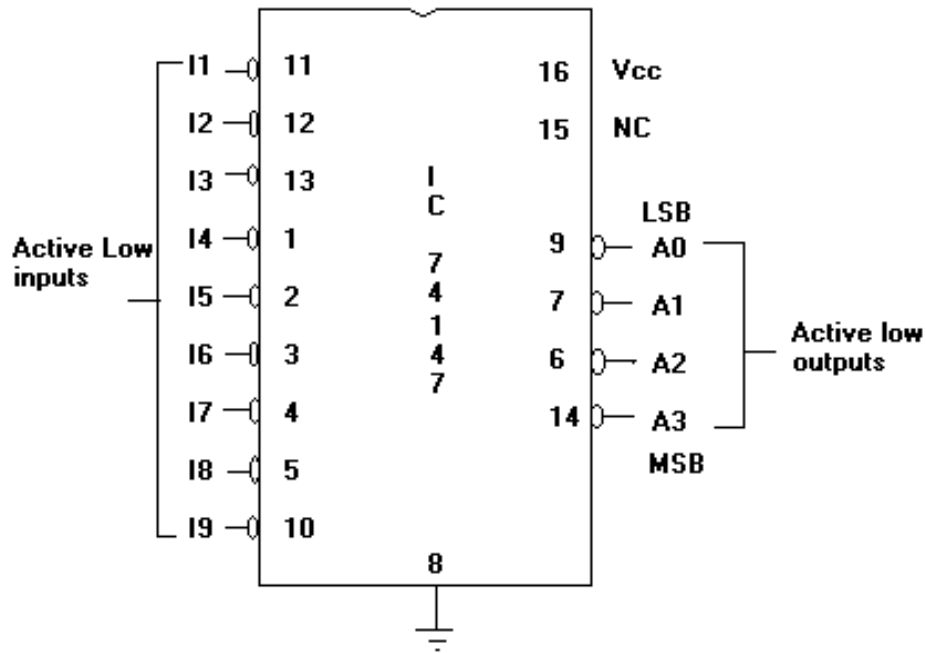
#### Components Required:

Serial No.	Component Description	IC No.	Required No.
1	10-line to 4-line Priority Encoder	7447	1
2	8-line 3-line priority encoder	LT 542	1
3	Patch Chords	----	20
4	IC Trainer Kit	----	1

#### Procedure:

1. Verify all the components and patch chords whether they are in good condition.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit & feed active low input bit combinations.
4. Verify the truth table sequence and observe the outputs.

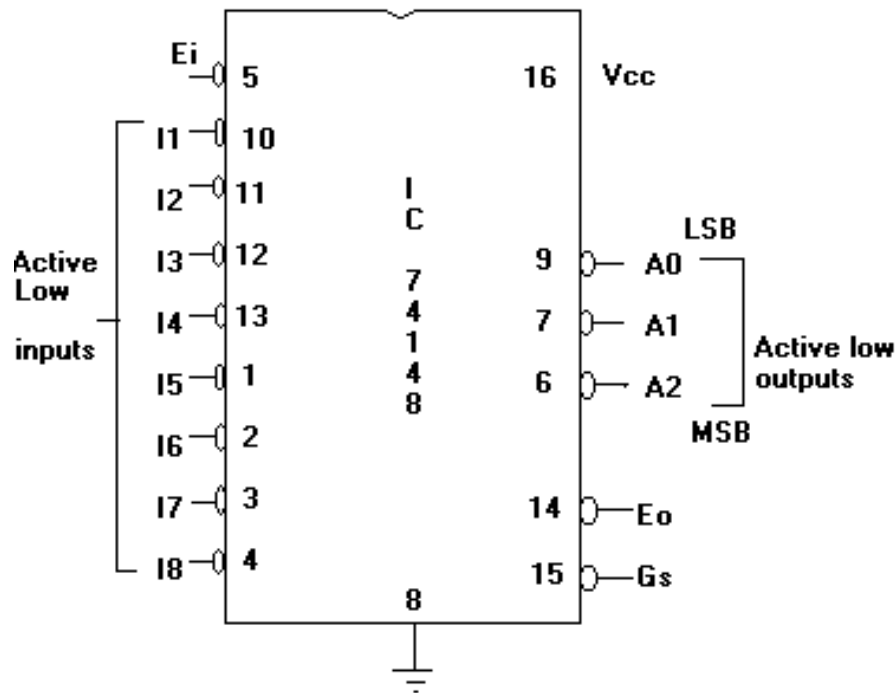
**Decimal to BCD Priority Encoder.**



**Truth Table**

Inputs									Outputs			
I1	I2	I3	I4	I5	I6	I7	I8	I9	A3	A2	A1	A0
x	x	x	x	x	x	x	x	0	0	1	1	0
x	x	x	x	x	x	x	0	1	0	1	1	1
x	x	x	x	x	x	0	1	1	1	0	0	0
x	x	x	x	x	0	1	1	1	1	0	0	1
x	x	x	0	1	1	1	1	1	1	0	1	0
x	x	0	1	1	1	1	1	1	1	1	0	0
x	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1

**Octal to Binary Priority Encoder.**



**Truth Table**

Inputs									Outputs		
EI	I0	I1	I2	I3	I4	I5	I6	I7	A2	A1	A0
1	X	x	x	x	x	x	x	x	1	1	1
0	X	x	x	x	x	x	x	x	1	1	1
0	X	x	x	x	x	x	x	0	0	0	0
0	X	x	x	x	x	x	0	1	0	0	1
0	X	x	x	x	x	0	1	1	0	1	0
0	X	x	x	x	0	1	1	1	0	1	1
0	X	x	x	0	1	1	1	1	1	0	0
0	X	x	0	1	1	1	1	1	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1

D).One/Two bit Comparator.

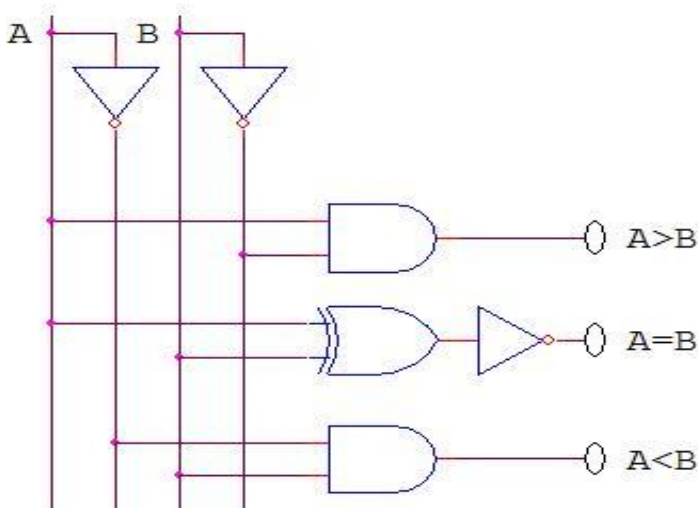
TRUTH TABLE

INPUT S		OUTPUT S		
A	B	A > B	A = B	A < B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

$$A > B = A \bar{B}$$

$$A < B = \bar{A} B$$

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

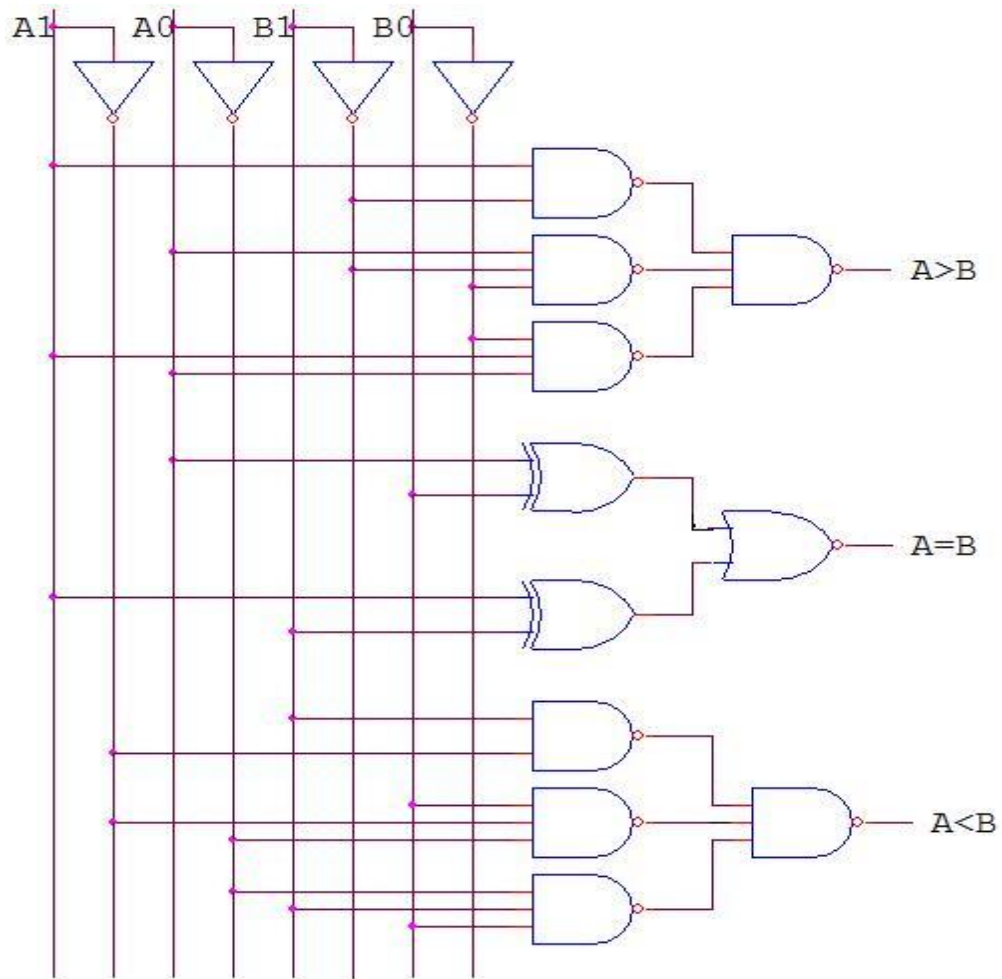


## 2- BIT COMPARATOR

$$(A>B) = A_1 B_1 + A_0 B_1 B_0 + B_0 A_1 A_0$$

$$(A=B) = (A_0 \oplus B_0) (A_1 \oplus B_1)$$

$$(A<B) = B_1 A_1 + B_0 A_1 A_0 + A_0 B_1 B_0$$



## TRUTH TABLE

INPUTS				OUTPUTS		
A <sub>1</sub>	A <sub>0</sub>	B <sub>1</sub>	B <sub>0</sub>	A > B	A = B	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 5**

.To realize the following flip-flops using NAND Gates (a) T type (b) JK Master slave (c) D type

Master Slave JK Flip-Flop:

Preset	Clear	J	K	Clock	$Q_{n+1}$	$\overline{Q_{n+1}}$	
0	1	X	X	X	1	0	Set
1	0	X	X	X	0	1	Reset
1	1	0	0	$\downarrow$	$Q_n$	$\overline{Q_n}$	No Change
1	1	0	1	$\downarrow$	0	1	Reset
1	1	1	0	$\downarrow$	1	0	Set
1	1	1	1	$\downarrow$	$\overline{Q_n}$	$Q_n$	Toggle

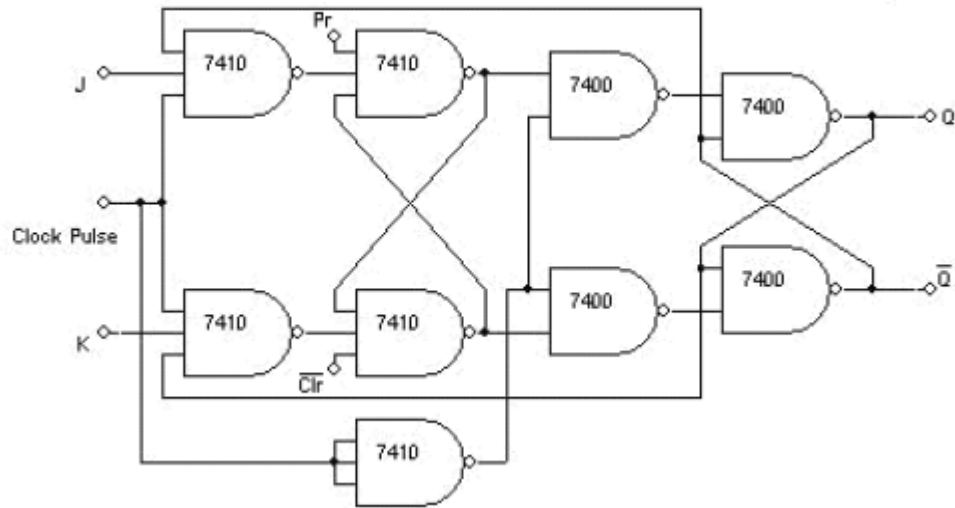
D Flip-Flop:-

Preset	Clear	D	Clock	$Q_{n+1}$	$\overline{Q_{n+1}}$
1	1	0	$\downarrow$	0	1
1	1	1	$\downarrow$	1	0

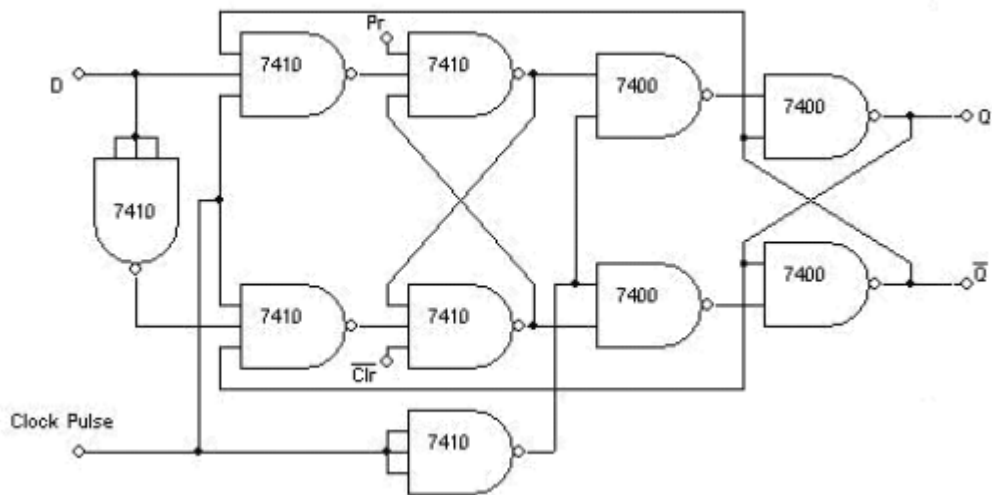
T Flip-Flop:-

Preset	Clear	T	Clock	$Q_{n+1}$	$\overline{Q_{n+1}}$
1	1	0	$\downarrow$	$Q_n$	$\overline{Q_n}$
1	1	1	$\downarrow$	$\overline{Q_n}$	$Q_n$

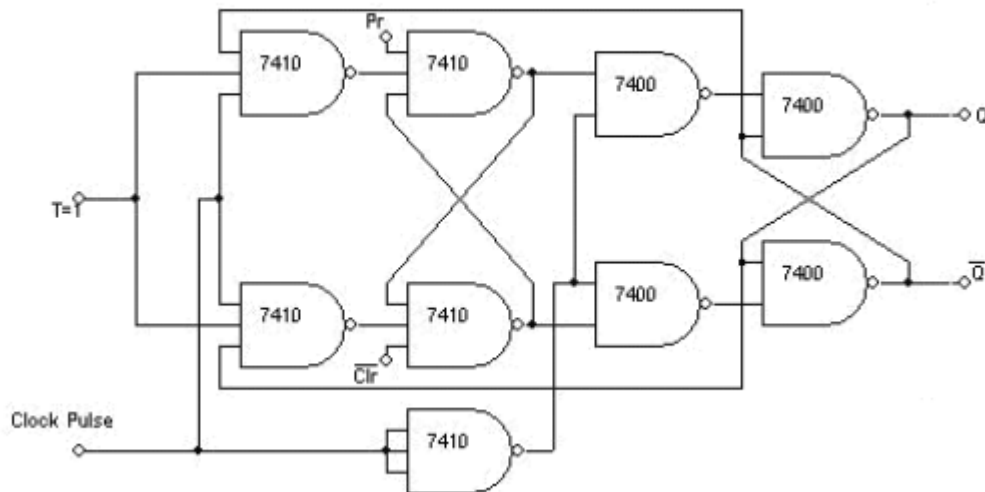
**Circuit Diagram: - (Master Slave JK Flip-Flop)**



**D Flip-Flop:-**



**T Flip-Flop:-**





### Experiment 6

To realize the 3-bit counters as a sequential circuit and Mod-N Counter design (7476, 7490, 74192, 74193)

**Procedure:**

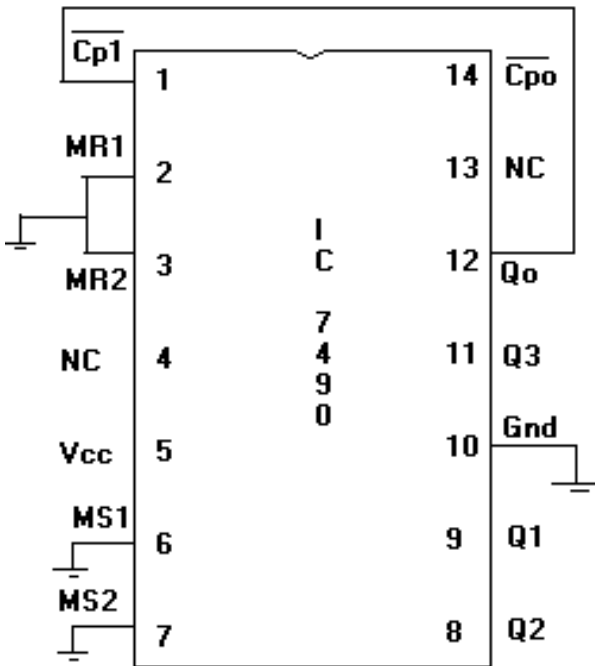
1. Verify all the components and patch chords whether they are in good condition.
2. Make connections as shown in the circuit diagram.
3. Give supply to the trainer kit
4. Apply the manual pulses and verify the truth table.

**Pin Diagram**

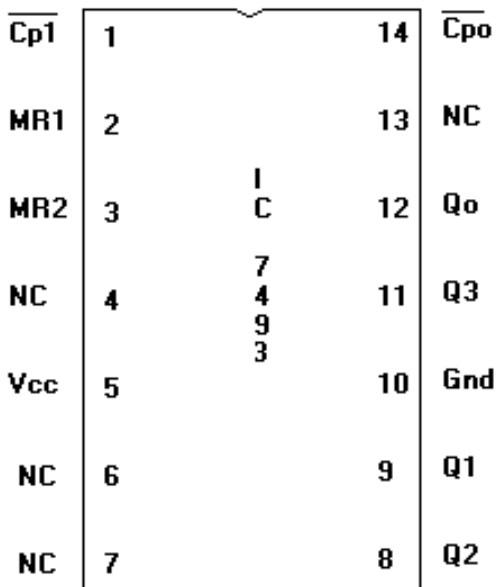
$\overline{Cp1}$	1		14	$\overline{Cpo}$
MR1	2		13	NC
MR2	3	I C	12	Qo
NC	4	7 4 9 0	11	Q3
Vcc	5		10	Gnd
MS1	6		9	Q1
MS2	7		8	Q2

Truth Table

Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1

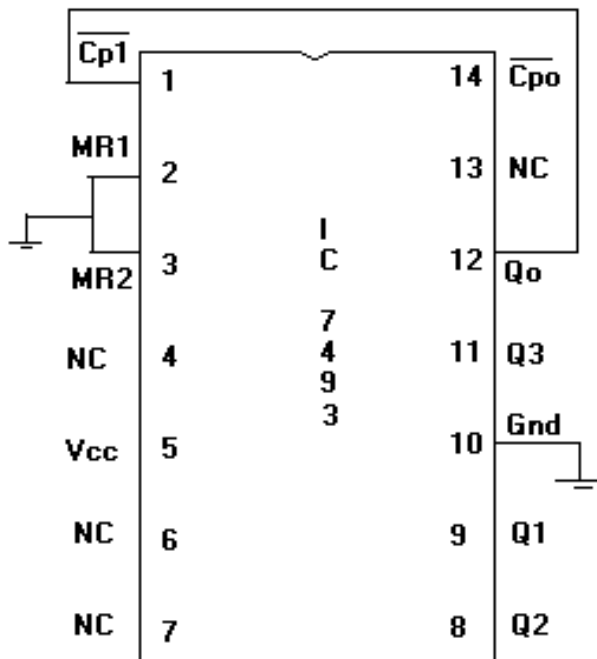


Pin diagram



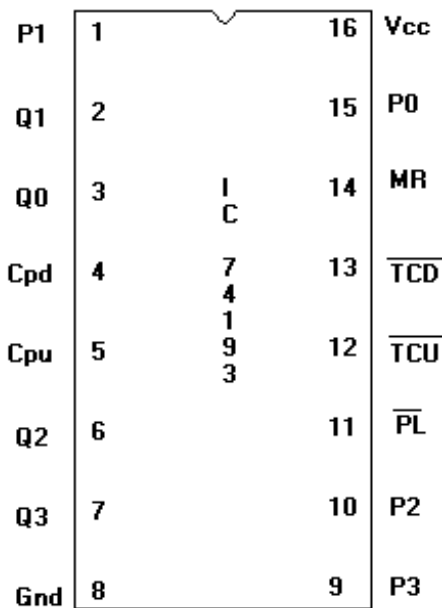
IC 7493 as a 4-bit binary Counter

Truth Table

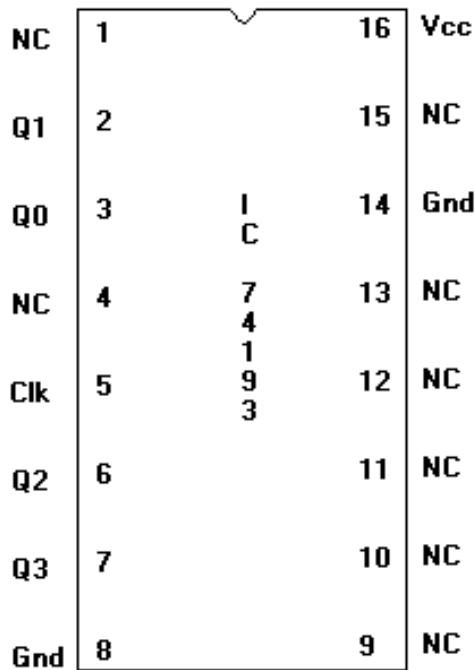


Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Pin Diagram



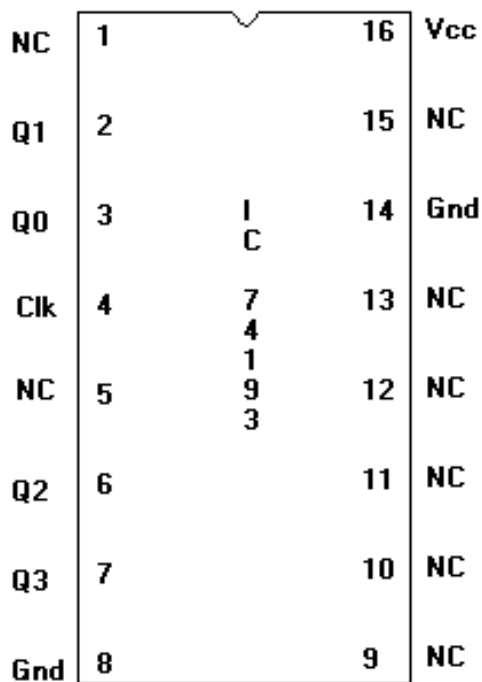
74193 as Mod-16 Up Counter



Truth Table

Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	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

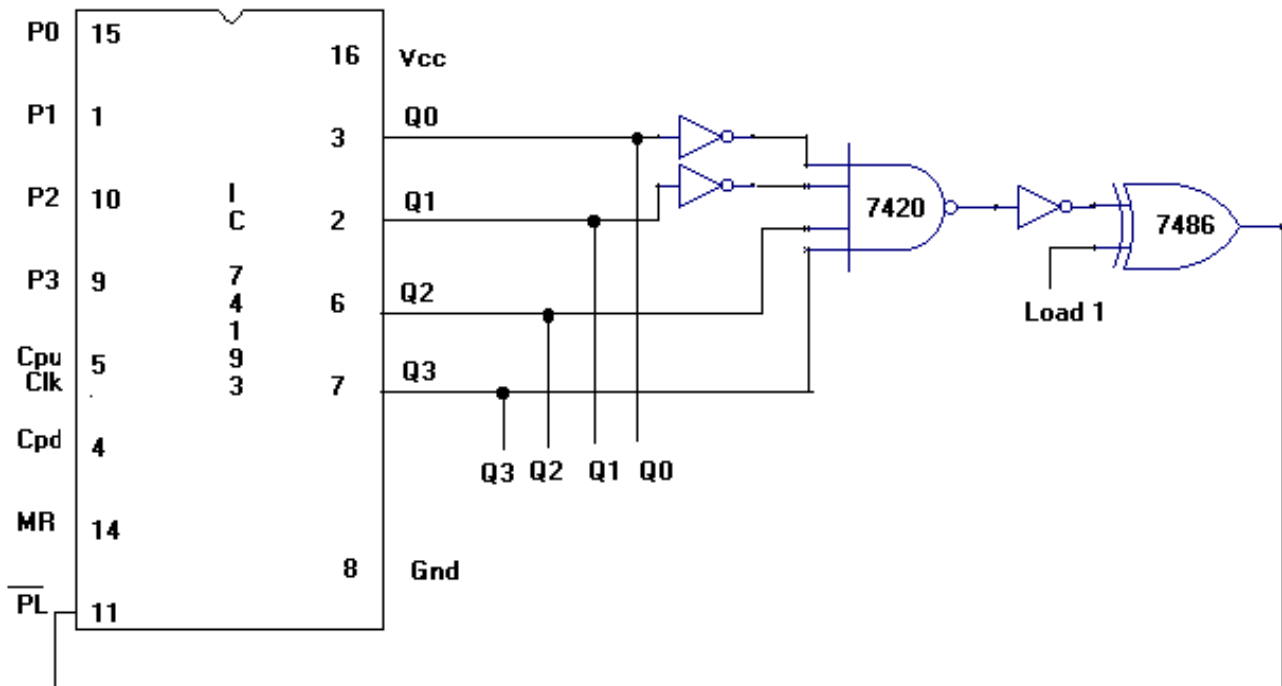
74193 as Mod-16 Down Counter



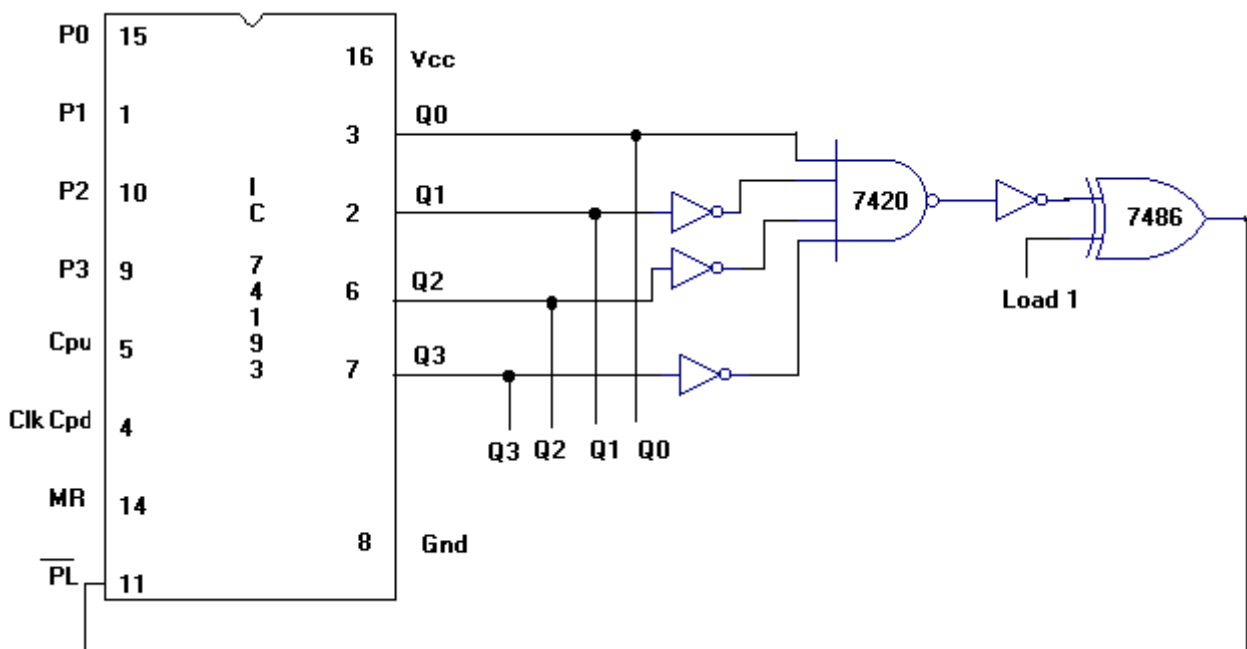
Truth Table

Q3	Q2	Q1	Q0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

**Design Up Counter For Preset Value 0010 and N=10 using IC 74193**



**Design Down Counter For Preset Value 1011 and N=10 using IC 74193**



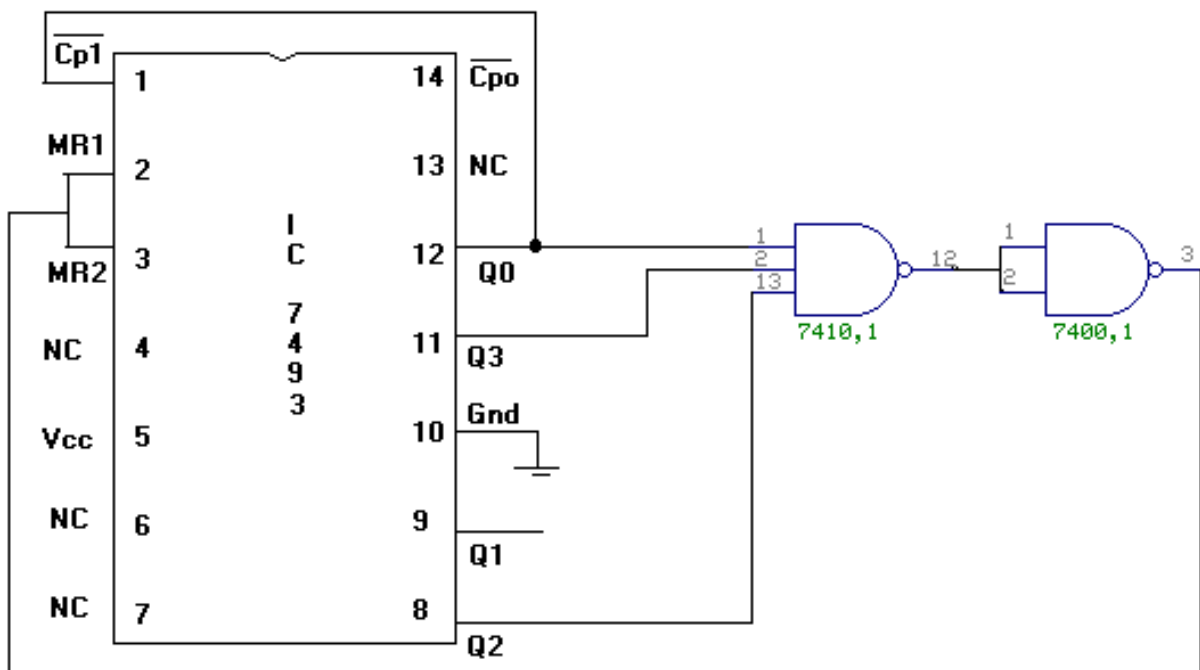
Truth Table For UP Counter

No. of Counts	Q3	Q2	Q1	Q0
1	0	0	1	0
2	0	0	1	1
3	0	1	0	0
4	0	1	0	1
5	0	1	1	0
6	0	1	1	1
7	1	0	0	0
8	1	0	0	1
9	1	0	1	0
10	1	0	1	1
11	Reset			

Truth Table For Down Counter

No. of Counts	Q3	Q2	Q1	Q0
1	1	0	1	1
2	1	0	1	0
3	1	0	0	1
4	1	0	0	0
5	0	1	1	1
6	0	1	1	0
7	0	1	0	1
8	0	1	0	0
9	0	0	1	1
10	0	0	1	0
11	Reset			

Design Mod-13 Up Counter Using IC 7493

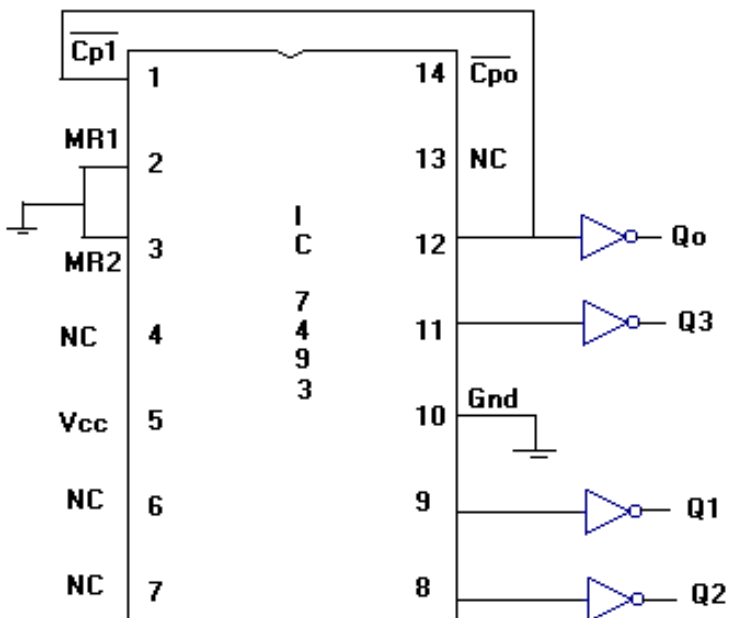


Truth Table

Clk Pulse	Q3	Q2	Q1	Q0
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	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1

**Design Mod-16 Down Counter using IC 7493**

**Truth Table.**



Q3	Q2	Q1	Q0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

# **HDL PROGRAMMING**



## Experiment 7

### Adder/Subtractor – Full/half using Verilog data flow description

```
module HA(a,b,s,c);  
Input a,b;  
Output s,c;  
assign s=a^b;  
assign c=a&b;  
endmodule
```

```
module HSub(a,b,d,b);  
Input a,b;  
Output d,b;  
assign s=a^b;  
assign c=(~a)&b;  
endmodule
```

```
module FA(a,b,cin,s,cout);  
Input a,b,cin;  
Output s,cout;  
assign s=a^b^cin;  
assign cout=((a&b)|(a&cin)|(b&cin));  
endmodule
```

```
module FSub(a,b,bin,d,bout);  
Input a,b,bin;  
Output d,bout;  
assign d=a^b^bin;  
assign bout=((~a)&(b&bin)|(b&bin));  
endmodule
```

## Experiment 8

. Code converters using Verilog Behavioral description a) Gray to binary and vice versa b) Binary to excess3 and vice versa

### a) Gray to binary and vice versa

```
module g2b_vconv(g, b);  
  input [3:0] g;  
  output [3:0] b;  
  
  assign b[3] = g[3];  
  assign b[2] = g[3] ^ g[2];  
  assign b[1] = b[2] ^ g[1];  
  assign b[0] = b[1] ^ g[0];  
  
endmodule
```

### Binary to Gray

```
module b2g_vconv(b, g);  
  input [3:0] b;  
  output [3:0] g;  
  
  assign g[3] = b[3];  
  assign g[2] = b[3] ^ b[2];  
  assign g[1] = b[2] ^ b[1];  
  assign g[0] = b[1] ^ b[0];  
  
endmodule
```

**Binary to excess3 and vice versa**

```
module binary2ex3(b3,b2,b1,b0,e3,e2,e1,e0);  
input b3,b2,b1,b0;  
output e3,e2,e1,e0;  
assign e3=b3|b0&b2|b2&b1;  
assign e2=(~b1)&(~b0)&(b2)|(~b2)&(b0)|(~b2)&(b1);  
assign e1=(~b1)&(~b0)|b1&b2;  
assign e0=(~b1)&(~b0)|b1&(~b0);  
endmodule
```

## Experiment 9

Multiplexers/decoders/encoder using Verilog Behavioral description

- 8:1 mux, 3:8 decoder, 8:3 encoder, Priority encoder

-2-bit Comparator using behavioral description

### 8:3 encoder

```
module vencoder(i, f, valid);
  input [7:0] i;
  output [2:0] f;
  output valid;
  reg [2:0] f;

  always @ (i)
  begin
    if (i[7] == 1) assign f = 111;
    else if (i[6] == 1) assign f = 110;
    else if (i[5] == 1) assign f = 101;
    else if (i[4] == 1) assign f = 100;
    else if (i[3] == 1) assign f = 011;
    else if (i[2] == 1) assign f = 010;
    else if (i[1] == 1) assign f = 001;
    else if (i[0] == 1) assign f = 000;
  end
  assign valid = i[7] | i[6] | i[5] | i[4] | i[4] | i[3] | i[2] | i[1] | i[0];
endmodule
```

### 8:1 mux

```
module 8x1 mux(I,sel,y);
  input[7:0]I;
  input[3:0]sel;
  output y;
  reg y;
  always @ (sel)
  begin
    case(sel)
      3'b000:begin y=i[0]; end
```

```
3'b001:begin y=i[1]; end

3'b010:begin y=i[2]; end

3'b011:begin y=i[3]; end

3'b100:begin y=i[4]; end

3'b101:begin y=i[5]; end

3'b110:begin y=i[6]; end

3'b111:begin y=i[7]; end
endcase
end
endmodule
```

### 2-bit Comparator

```
module comp_2bitv(x,y, xgty,xlty, xeqy);
  input [1:0] x,y;
  output xeqy,xgty,xlty;

  assign xgty = (x[1] & ~ y[1]) |
               (x[0] & ~ y[1] & ~ y[0]) |
               (x[0] & x[1] & ~ y[0]);
  assign xlty = (y[1] & ~ x[1] ) |
               (~ x[0] & y[0] & y[1]) |
               (~ x[0] & ~ x[1] & y[0]);
  assign xeqy = ~ (xgty | xlty);
endmodule
```

### 3:8 decoder

```
module decoder(d,x,y,z);
output [7:0] d;
input x,y,z;
assign d[0] = ~x & ~y & ~z;
assign d[1] = ~x & ~y & z;
assign d[2] = ~x & y & ~z;
assign d[3] = ~x & y & z;
assign d[4] = x & ~y & ~z;
assign d[5] = x & ~y & z;
assign d[6] = x & y & ~z;
assign d[7] = x & y & z;
endmodule
```

### Priority encoder

```
module priority (sel, code);
input [7:0] sel;
output [2:0] code;
reg [2:0] code;
always @(sel)
begin
    if (sel[0]) code <= 3'b000;
    else if (sel[1]) code <= 3'b001;
    else if (sel[2]) code <= 3'b010;
    else if (sel[3]) code <= 3'b011;
    else if (sel[4]) code <= 3'b100;
    else if (sel[5]) code <= 3'b101;
    else if (sel[6]) code <= 3'b110;
    else if (sel[7]) code <= 3'b111;
    else
code <= 3'bxxx;
end
endmodule
```

## Experiment 10

. Flip-flops using Verilog Behavioral description a) JK type b) SR type c) T type and d) D type

### a) JK type

```
module JKFlipFlop(J,K, CLK,Q,Qbar);
    input J;
    input K;
    input clk;
    output Q;
    output Qbar ;
    reg Q,Qbar;

    always@(posedge clk)
    begin
        case({J,K})
            2'b0_0:Q<=Q;
            2'b0_1:Q<=1'b0;
            2'b1_0:Q<=1'b1;
            2'b1_1:Q<=Qbar;
        endcase
    end
endmodule
```

### b) SR type

```
module SR_FlipFlop(S,R,clk,Q,Qbar);
    input S,R ;
    input clk;
    output Q;
    output Qbar;
    reg Q,Qbar;
```

```
always@(posedge clk)
begin
    case({S,R})
        2'b0_0:Q<=Q;
        2'b0_1:Q<=1'b0;
        2'b1_0:Q<=1'b1;
        2'b1_1:Q<=1'bz;
    endcase
end
endmodule
```

**c) T type**

```
module tffmod(t, clk, q);
input t;
input clk;
output q;
reg q;
initial q<=1'b0;
always @(posedge clk)
q<=q^t;
endmodule
```



**d) D type**

```
module d_flip_flop ( din ,clk ,reset ,dout );  
output dout ;  
reg dout ;  
input din ;  
wire din ;  
input clk ;  
wire clk ;  
input reset ;  
wire reset ;  
  
always @ (posedge (clk)) begin  
if (reset)  
dout <= 0;  
else  
dout <= din ;  
end  
  
endmodule
```

## Experiment 11

### Counter up/down (BCD and binary) , sequential counters using Verilog Behavioral description

#### Binary Up/Down Counter

```
module UpDownCounterBinary(clk,enable,reset,mode,count,tc);
input clk,enable,reset,mode;
output reg [3:0]count;
output reg tc;
always @(posedge clk)
begin
    if(enable)
    begin
        if(reset)
        begin
            count=0;
            tc=0;
        end
        else
        begin
            if(mode==0)
            begin
                count=count+1;
                if(count==15)
                tc=1;
            else
                tc=0;
            end
            else
            begin
                count=count-1;
                if(count==0)
                tc=1;
            else
                tc=0;
            end
        end
    end
end
endmodule
```

**BCD Up/Down Counter**

```
module UpDownCounterBCD(clk,enable,reset,mode,count,tc);
input clk,enable,reset,mode;
output reg [3:0]count;
output reg tc;
always @(posedge clk)
begin
    if(enable)
    begin
        if(reset)
        begin
            count=0;
            tc=0;
        end
        else
        begin
            if(mode==0)
            begin
                count=count+1;
                if(count==9)
                tc=1;
            else
                tc=0;
            end
            else
            begin
                count=count-1;
                if(count==0)
                tc=1;
            else
                tc=0;
            end
        end
    end
end
endmodule
```

## Experiment 12

**Interface experiments: (a) Stepper motor (b) Relay (c) Waveform generation using DAC**

### (a) Stepper motor

```
module stepper_motor_full_step ( start ,clk ,dout );
output [3:0] dout ;
reg [3:0] dout ;
input start ;
wire start ;
input clk ;
wire clk ;
reg [1:0] m ;
initial m = 0;

always @ (posedge (clk)) begin
if (start)
m <= m + 1;
end

always @ (m) begin
case (m)
0 : dout = 8;
1 : dout = 4;
2 : dout = 2;
default : dout = 1;
endcase
end

endmodule
```

**(b) Waveform generation using DAC****Square Wave**

```
module square_wave(clk,rst,dac_out);
    input clk;
    input rst;
    output reg [0:7] dac_out;
    reg [7:0] temp;
    reg [7:0] counter;

    always @(posedge clk)
    begin
        temp <= temp + 1'b1;
    end

    always @(posedge temp[3])
    begin
        if (rst)
            begin
                counter <=0;
            end
        else
            counter<=counter + 1'b1;
    end

    always @(posedge temp[3])

    begin
        if (counter<=127)
            dac_out=8'd1;
        else
            dac_out=8'd0;
    end
endmodule
```

### Triangular Wave

```
module pila(clk,res,out2);
    input  clk,res;
    output [0:7]out2;
    reg [0:7]out2;
    always @(posedge clk)
    begin
    if (res)
    begin
    if(out2<=8'b11111111)
    out2=out2+1;
    else if(out2>=8'b00000000)
    out2=out2-1;
    else out2=8'b00000000;
    end
    else out2=8'b00000000;
    end
endmodule
```

```
module testbench;
    reg clk,res;
    wire [0:7]out2;
    pila Sevo(clk,res,out2);
    always #2 clk=~clk;
    initial
    begin
    clk=0;res=0;
    #2 res=1;
    end
    initial #5000 $finish;
endmodule
```

### Ramp Wave

```
module Ramp(Ramp,En,Clk,Rst);
output reg [7:0] Ramp;
input En,Clk,Rst;
always@(posedge Clk or posedge Rst)
begin
if (Rst)
Ramp<=0;
else begin
if (En)
begin
Ramp <= Ramp+1'b1;
end
end
end
endmodule
```