**EXPERIMENT No. 1**

**Title:** To study the different logic gates. A logic gate is a circuit with one or more inputs & one single point

**Estimated time to complete this experiment:** 2 Hrs

**Objective:** In this experiment, student should understand the different IC’s available for logic operations and be able to verify the truth table.

**CO to be achieved:** CO1

**Expected Outcome of Experiment**: Students will be able to understand various logic operations and digital integrated circuits for the purpose.

**Books/ Journals/ Websites referred:**

1. R. P Jain: Modern digital design, forth edition, tata mcgrawhill

2. Morris Mano, digital design, Pearson education, Asia2002.

3. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.

4. John M. Yarbrough, digital logic: applications and design, Thomson brooks/ cole, 2004

5. www.alldatasheet.com,

6. www.datasheetcatalog.com.

7. en.wikipedia.org/wiki/7400\_series

<http://vlabs.iitb.ac.in/vlabs-dev/labs/digital-electronics/experimentlist.html>

**Pre Lab/ Prior Concepts:** Concept about various logic operations

**Historical Profile:**

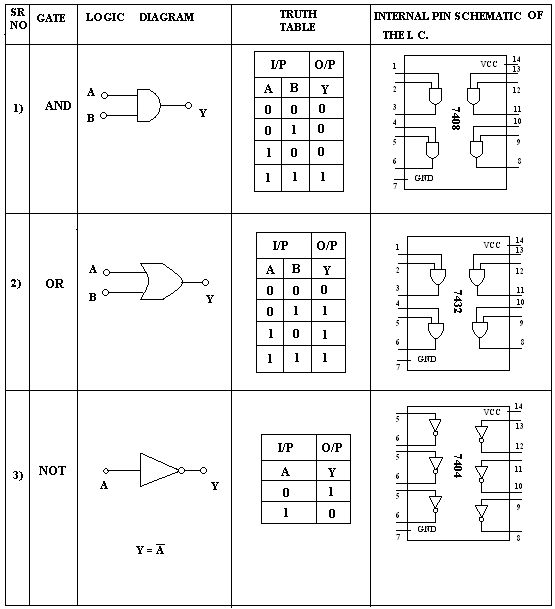
In a 1886 letter, [Charles Sanders Peirce](http://en.wikipedia.org/wiki/Charles_Sanders_Peirce) described how logical operations could be carried out by electrical switching circuits. Starting in 1898, [Nikola Tesla](http://en.wikipedia.org/wiki/Nikola_Tesla) filed for [patents](http://en.wikipedia.org/wiki/Patent) of devices containing electro-mechanical logic gate circuits (see [List of Tesla patents](http://en.wikipedia.org/wiki/List_of_Tesla_patents)). Eventually, [vacuum tubes](http://en.wikipedia.org/wiki/Vacuum_tube) replaced relays for logic operations. [Lee De Forest](http://en.wikipedia.org/wiki/Lee_De_Forest)'s modification, in 1907, of the [Fleming valve](http://en.wikipedia.org/wiki/Vacuum_tube) can be used as AND logic gate. [Ludwig Wittgenstein](http://en.wikipedia.org/wiki/Ludwig_Wittgenstein) introduced a version of the 16-row [truth table](http://en.wikipedia.org/wiki/Truth_table), which is [shown above](http://en.wikipedia.org/wiki/Logic_gate#Logic_gates), as proposition 5.101 of [Tractatus Logico-Philosophicus](http://en.wikipedia.org/wiki/Tractatus_Logico-Philosophicus) (1921). [Claude E. Shannon](http://en.wikipedia.org/wiki/Claude_E._Shannon) introduced the use of Boolean algebra in the analysis and design of switching circuits in 1937. [Walther Bothe](http://en.wikipedia.org/wiki/Walther_Bothe), inventor of the [coincidence circuit](http://en.wikipedia.org/wiki/Coincidence_circuit), got part of the 1954 [Nobel Prize](http://en.wikipedia.org/wiki/Nobel_Prize) in physics, for the first modern electronic AND gate in 1924. Active research is taking place in [molecular logic gates](http://en.wikipedia.org/wiki/Molecular_logic_gate).

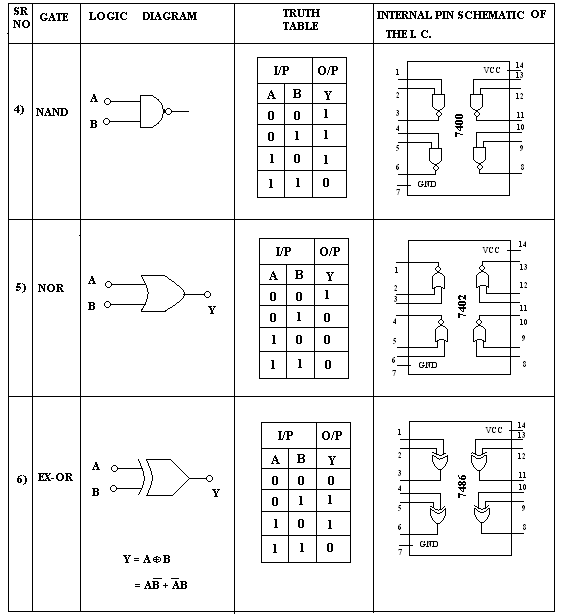
**New Concepts to be learned:** Working and Integrated circuits available for various logic operations.

**Requirements:**

 TTL IC no. 7408, 7432, 7404, 7400,7402,7486, 74266 patch cords & trainer kit

**Circuit Diagram:**





**Procedure:**

1. To study the logic gate, first select an appropriate IC for it.
2. Lock the selected IC in the slot provided in the trainer kit.
3. Make the connections as shown in the pin diagram of the concerned gate.
4. Give inputs as per the truth table & observe their output.

**Observations: Truth Tables**

**AND gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**OR gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **1** |

**NAND gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**NOR gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **0** |

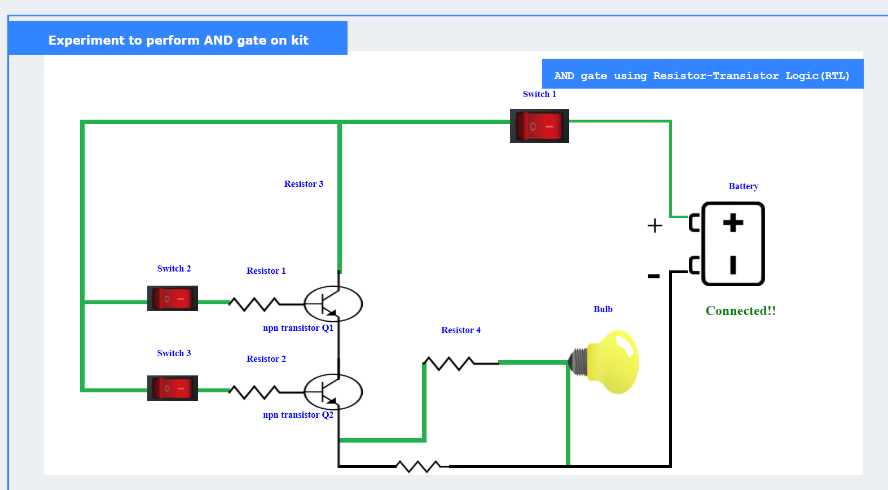
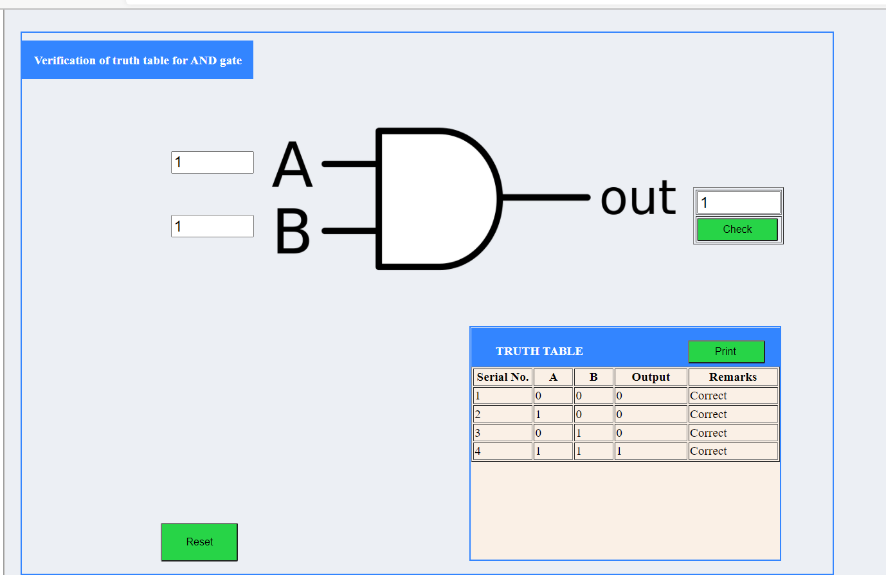
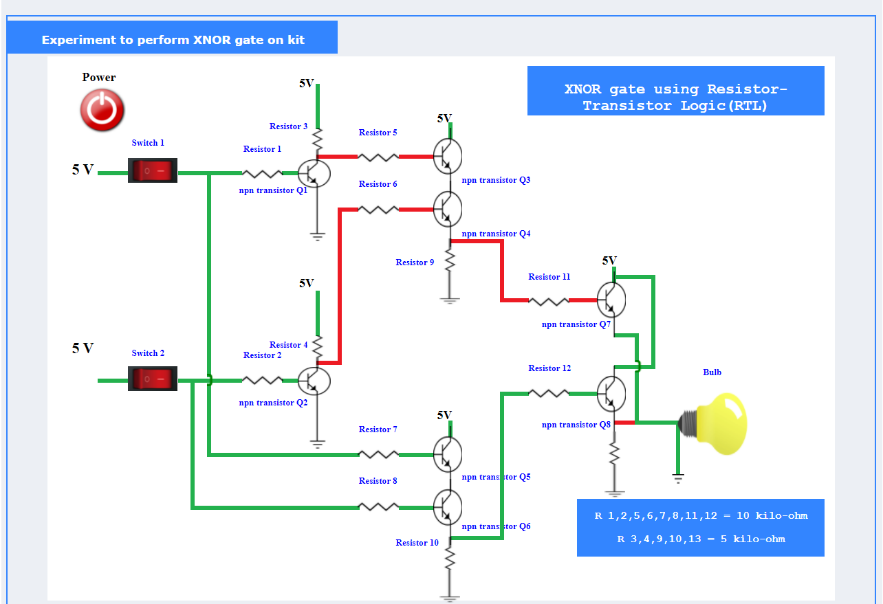
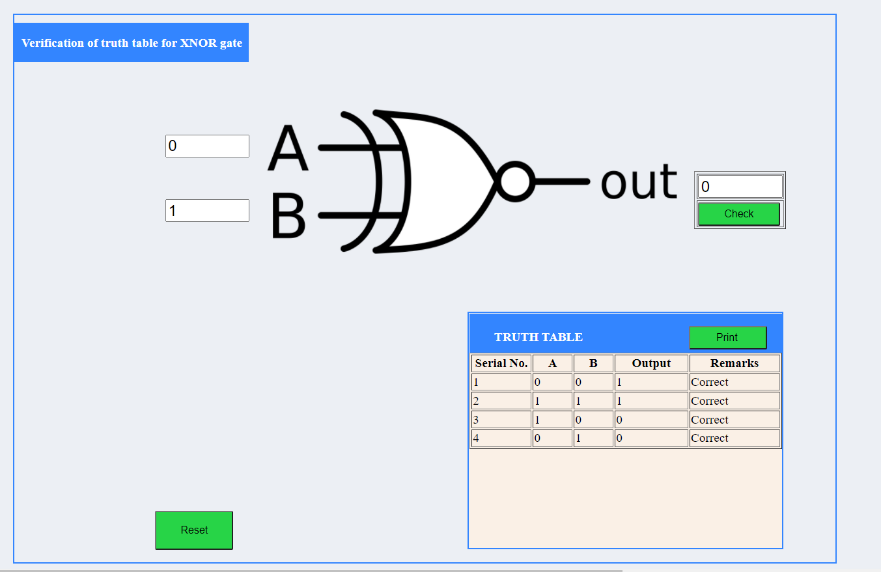
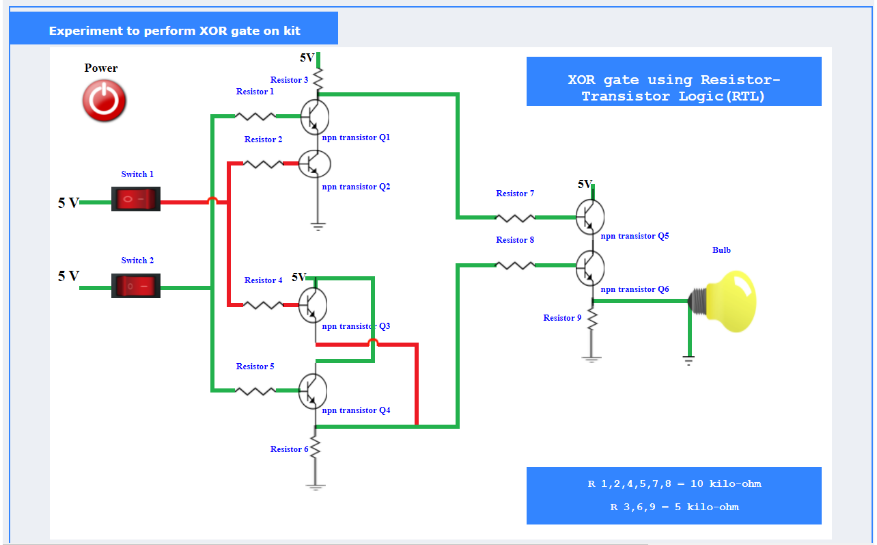
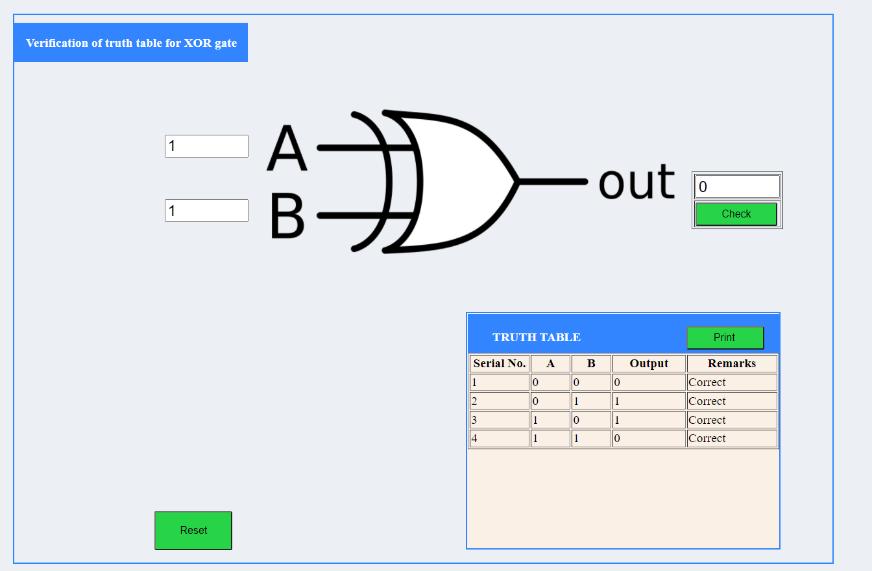
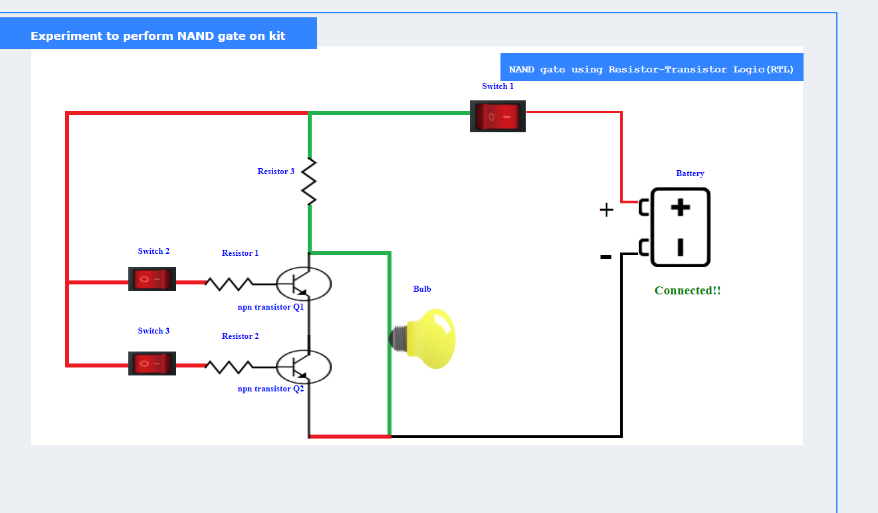
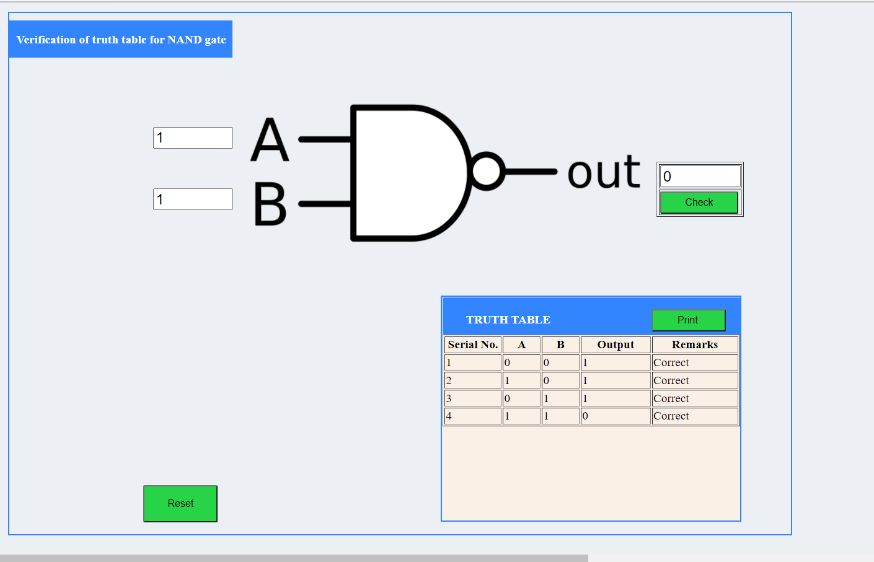
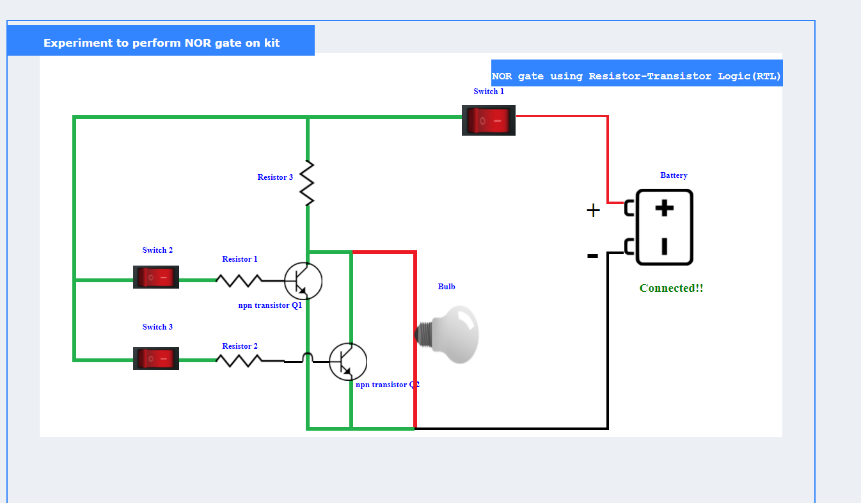
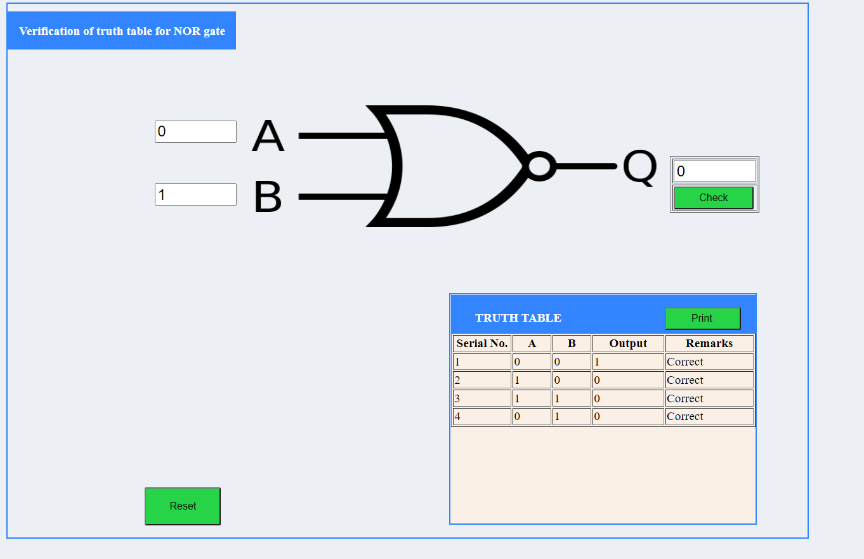
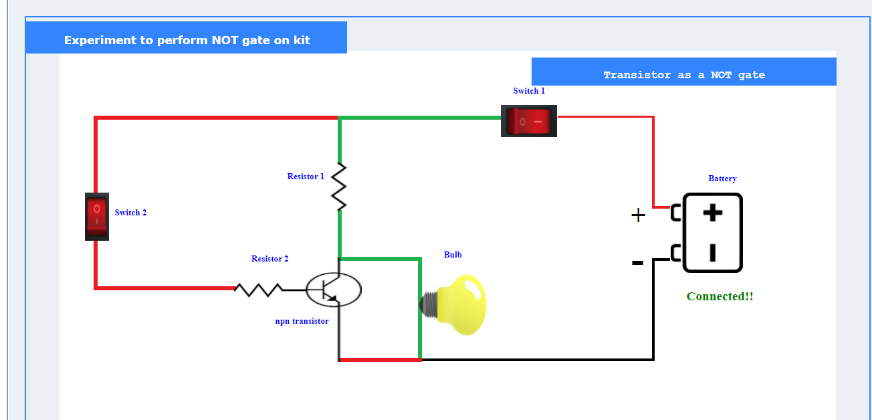
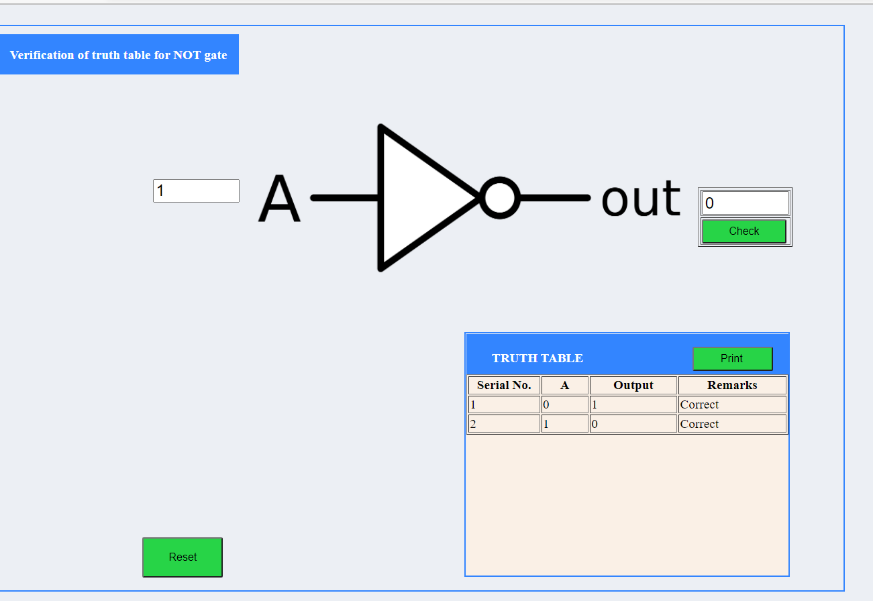
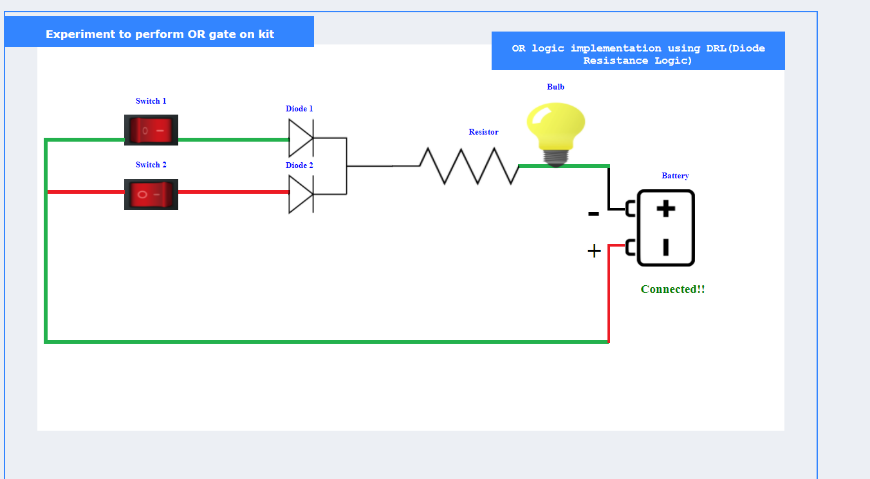
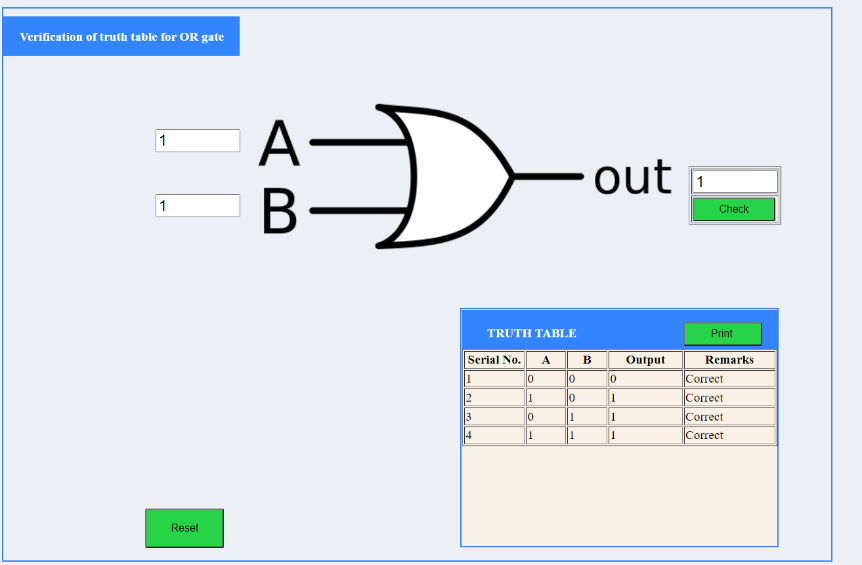
**EXOR Gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**NOT gate**

|  |  |
| --- | --- |
| **A** | **Y** |
| **0** | **1** |
| **1** | **0** |

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conclusion:** All fundamental logic gates & their outputs are studied successfully.

**Real Life Application:** Design of digital electronic circuit.

**Post Lab Questions:**

1. What is meant by positive logic/negative logic?

Positive logic is defined as a high voltage level representing a logic 1 and a low voltage level representing a logic 0. Negative logic is the reverse, i.e., a low voltage level represents a logic 1 and a high voltage level represents a logic 0.

1. Where do we use XOR gates?

It is used in simple digital addition circuits which calculate the sum and carry of two (half-adder) or three (full-adder) bit numbers. XOR gates are also used to determine the parity of a binary number, i.e., if the total number of 1's in the number is odd or even.

1. How do you make 3 input AND gate using two input AND gates?

Start by building the 2-input AND block from the last experiment, but plug the output of that into the input of another AND. Then add an Input Block to the second AND's second input.

1. Give one application of OR gate.

Alarm circuit for car door system

**Viva Questions:**

* In electronic circuit, what is a logic gate?

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output.

* Why is the name logic gate?

The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named

* What does logic state 0/1 corresponds to?

This is commonly translated as a binary 1 or binary 0. A binary 1 is also referred to as a HIGH signal and a binary 0 is referred to as a LOW signal.

* What is the truth table of AND (OR/ NAND/ NOR/EX-OR etc) gates?

As shown above in the experiment.

**EXPERIMENT No. 2**

**Title** To study the various gate obtained from a common universal gate NAND/NOR gate

**Estimated time to complete this experiment:** 2 Hrs

**Objective:** In this experiment, student should understand the use of NAND/ NOR gate as a universal gate

**CO to be achieved:** CO1

**Expected Outcome of Experiment**: Students will be able to understand the use of NAND/NOR gate as a universal gate and how to realize various logic operations using NAND/NOR logic.

**Books/ Journals/ Websites referred:**

1. R. P Jain: Modern digital design, forth edition, tata mcgrawhill

2. Morris Mano, digital design, Pearson education, Asia2002.

3. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.

4. John M. Yarbrough, digital logic: applications and design, Thomson brooks/ cole, 2004

5. www.alldatasheet.com,

6. www.datasheetcatalog.com.

7. en.wikipedia.org/wiki/7400\_series

**Pre Lab/ Prior Concepts:** Concept about universal NOR logic and De Morgan’s Theorem

**Historical Profile:**



The law is named after [Augustus De Morgan](http://en.wikipedia.org/wiki/Augustus_De_Morgan) (1806–1871)who introduced a formal version of the laws to classical [propositional logic](http://en.wikipedia.org/wiki/Propositional_logic). De Morgan's formulation was influenced by algebraization of logic undertaken by [George Boole](http://en.wikipedia.org/wiki/George_Boole), which later cemented De Morgan's claim to the find. Although a similar observation was made by [Aristotle](http://en.wikipedia.org/wiki/Aristotle) and was known to Greek and Medieval logicians(in the 14th century, [William of Ockham](http://en.wikipedia.org/wiki/William_of_Ockham) wrote down the words that would result by reading the laws out),De Morgan is given credit for stating the laws formally and incorporating them in to the language of logic. De Morgan's Laws can be proved easily, and may even seem trivial. Nonetheless, these laws are helpful in making valid inferences in proofs and deductive arguments.

[Charles Sanders Peirce](http://en.wikipedia.org/wiki/Charles_Sanders_Peirce) (winter of 1880–81) showed that [NOR gates](http://en.wikipedia.org/wiki/NOR_gates) alone (or alternatively [NAND gates](http://en.wikipedia.org/wiki/NAND_gates) alone) can be used to reproduce the functions of all the other logic gates, but his work on it was unpublished until 1933. The first published proof was by [Henry M. Sheffer](http://en.wikipedia.org/wiki/Henry_M._Sheffer) in 1913, so the NAND logical operation is sometimes called [Sheffer stroke](http://en.wikipedia.org/wiki/Sheffer_stroke); the [logical NOR](http://en.wikipedia.org/wiki/Logical_NOR) is sometimes called *Peirce's arrow*.Consequently, these gates are sometimes called *universal logic gates*.

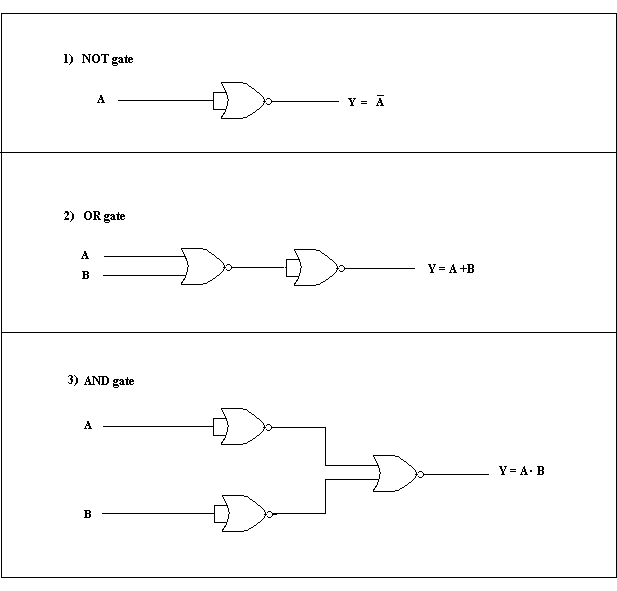
**New Concepts to be learned:** Realizing various logic operations using NOR logic. Converting AOI logic to NOR logic

**Requirements:**

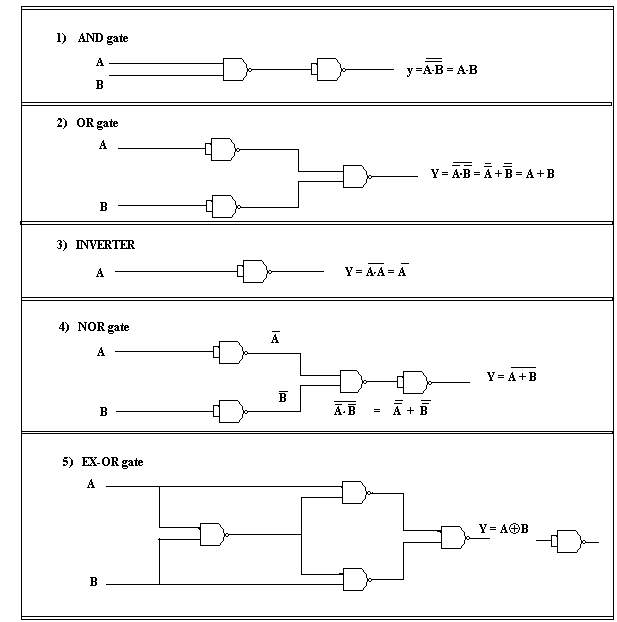
TTL IC No.7400(NAND gate),7402 (NOR gate) , patch cords & trainer kit.

**Circuit Diagram:**

**NOR Logic**



**NAND Logic**



**Procedure:**

1. To study the logic gate, first select an appropriate IC for it.
2. Lock the selected IC in the slot provided in the trainer kit.
3. Make the connections as shown in the pin diagram of the concerned gate.
4. Give inputs as per the truth table & observe their output

**Observations: Truth Tables**

**AND gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **1** |

**OR gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **1** |

**NAND gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**NOR gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |
| **1** | **1** | **0** |

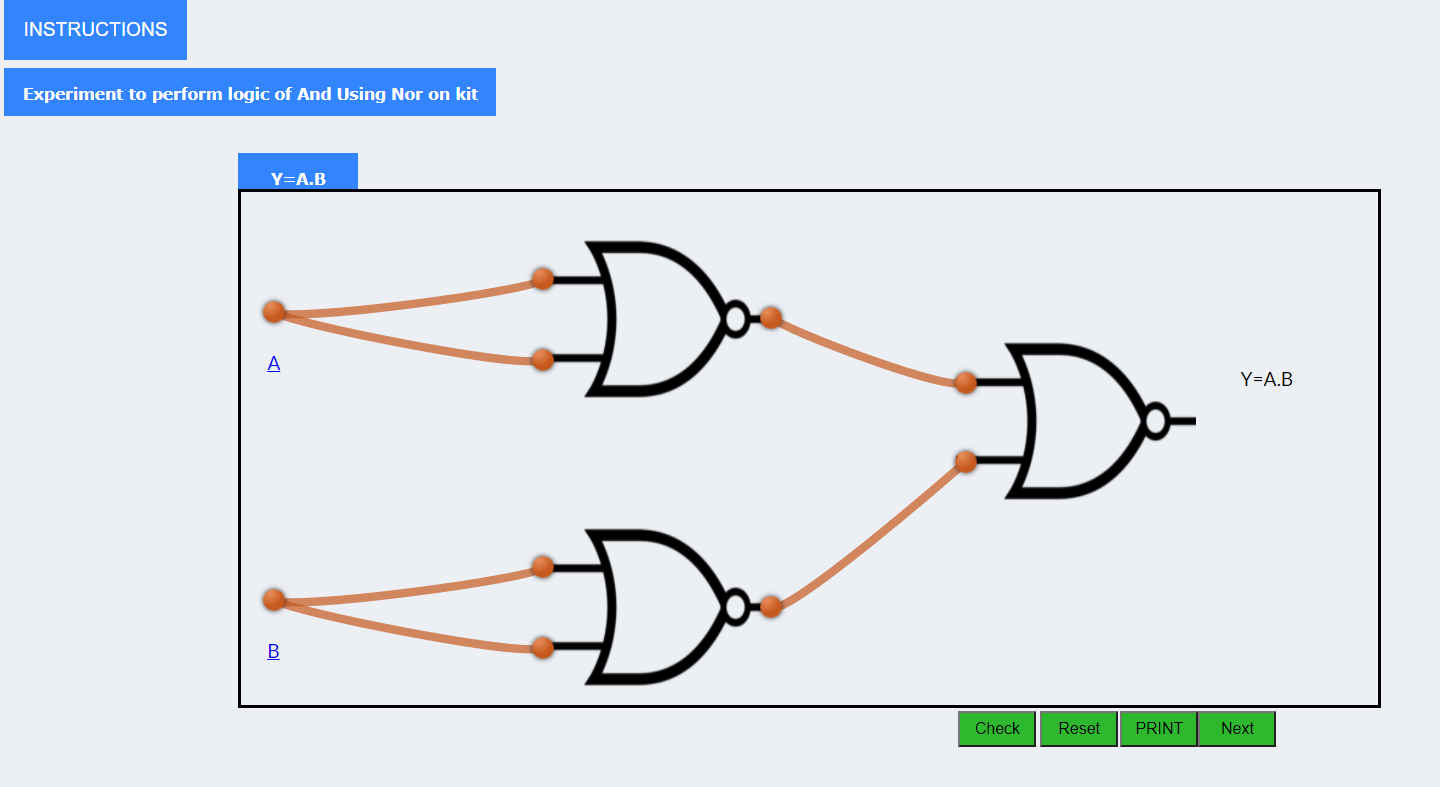
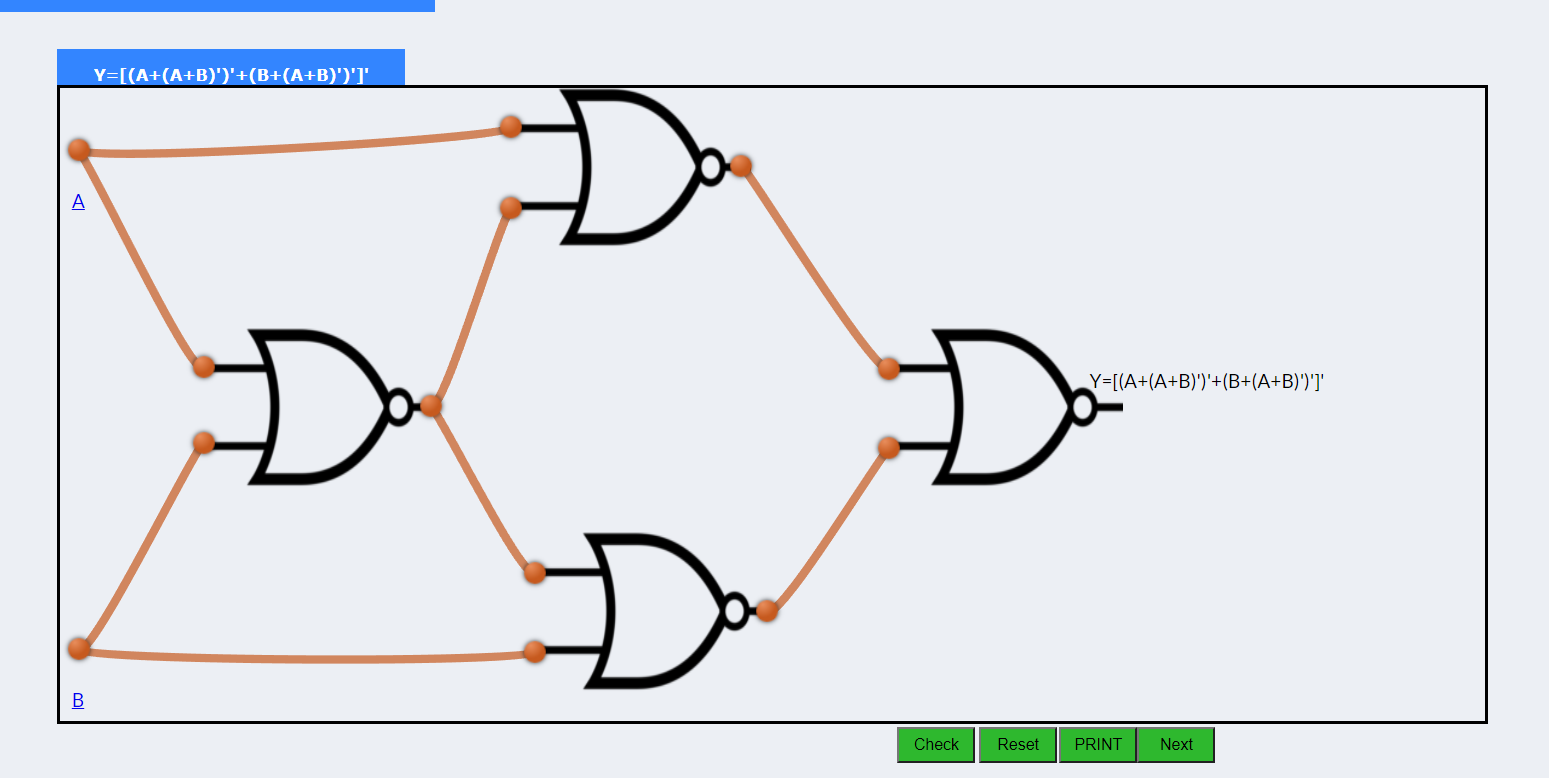
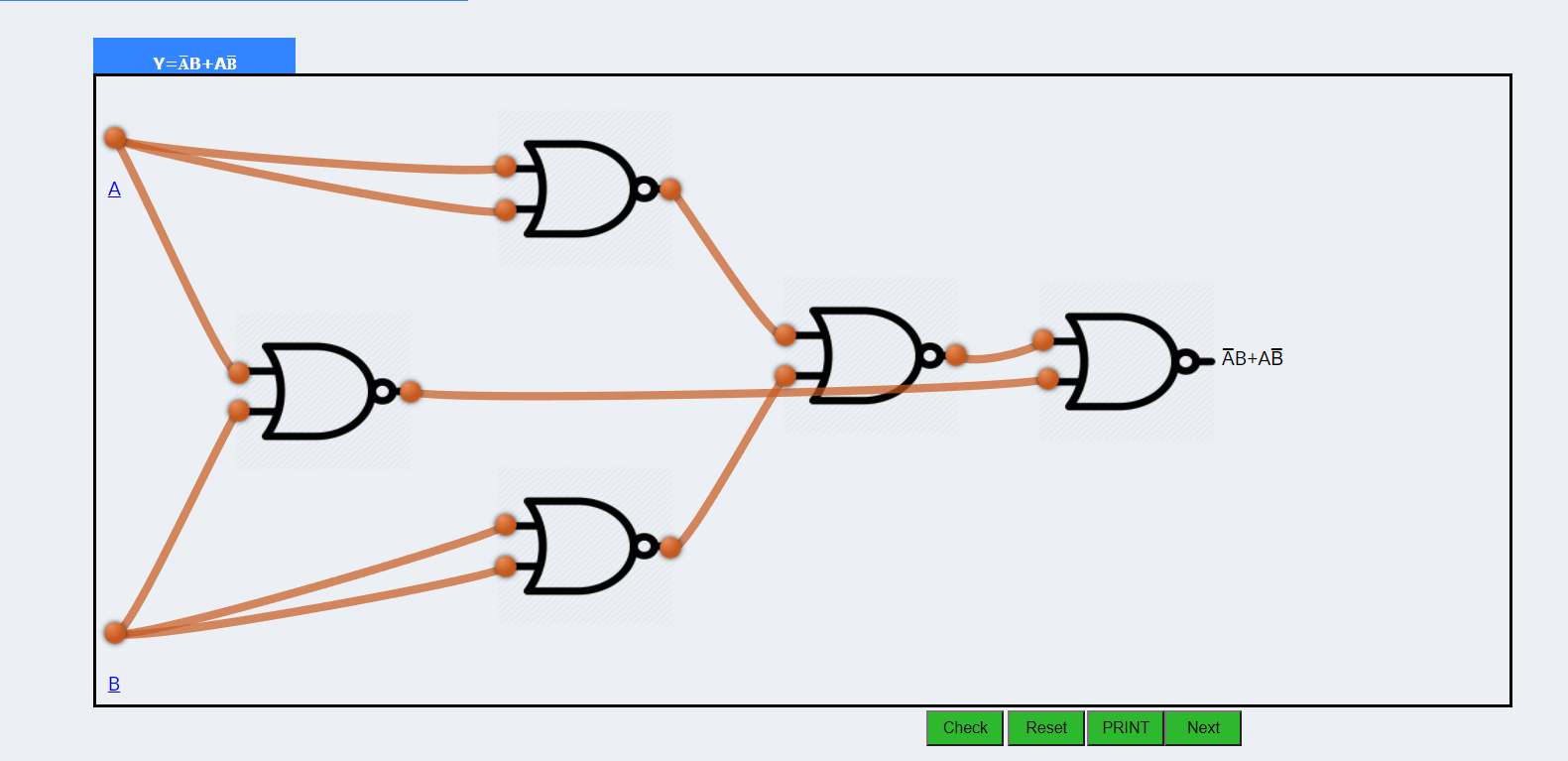
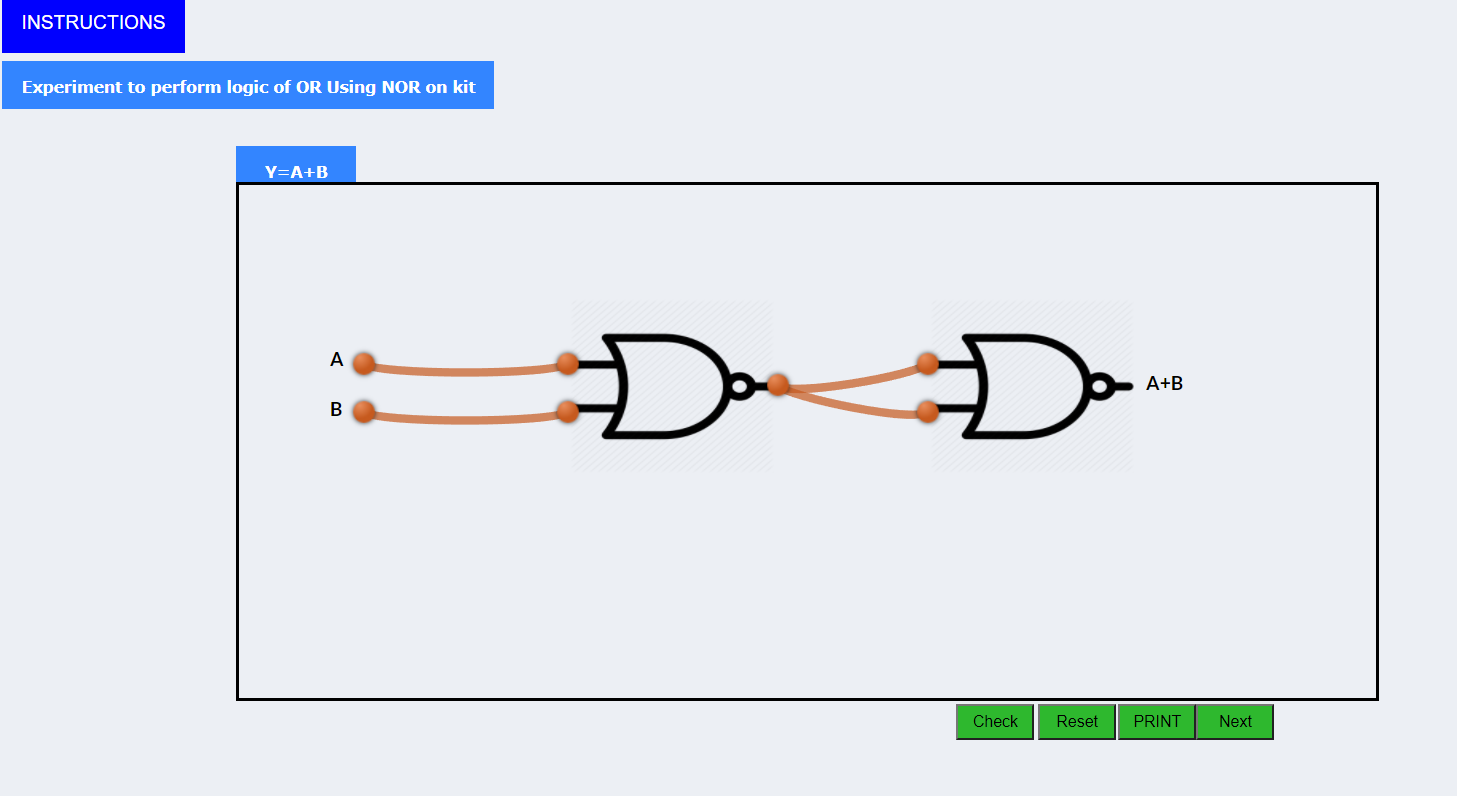
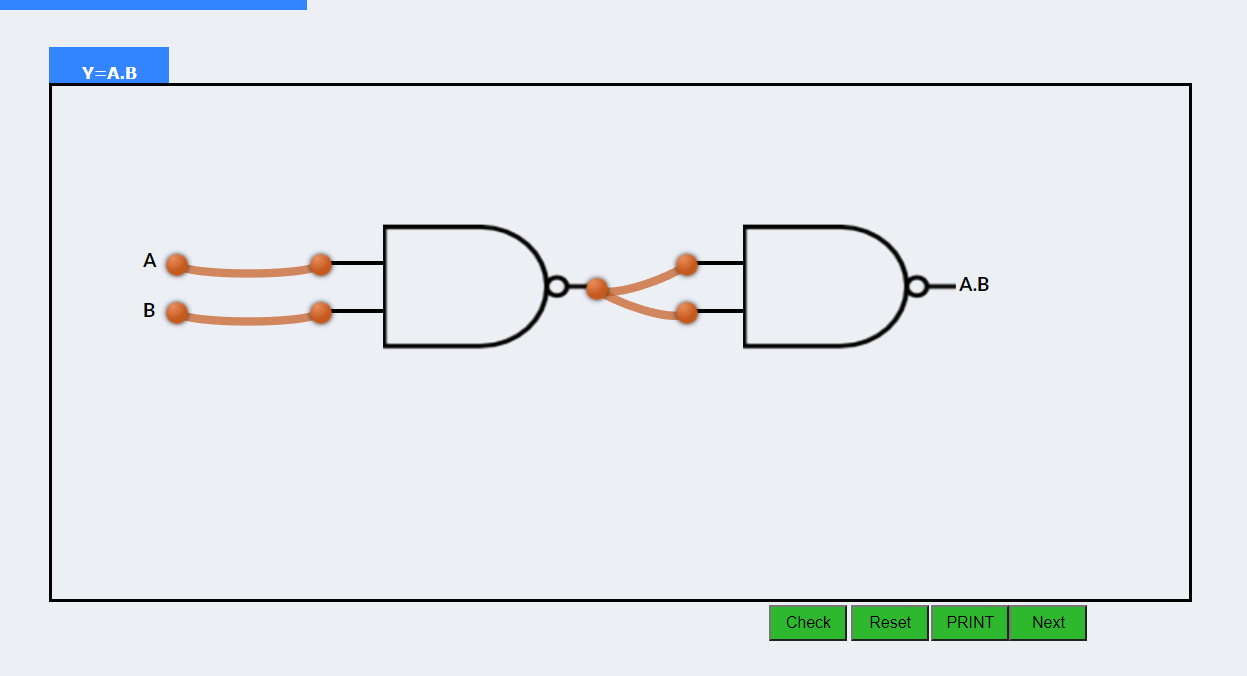
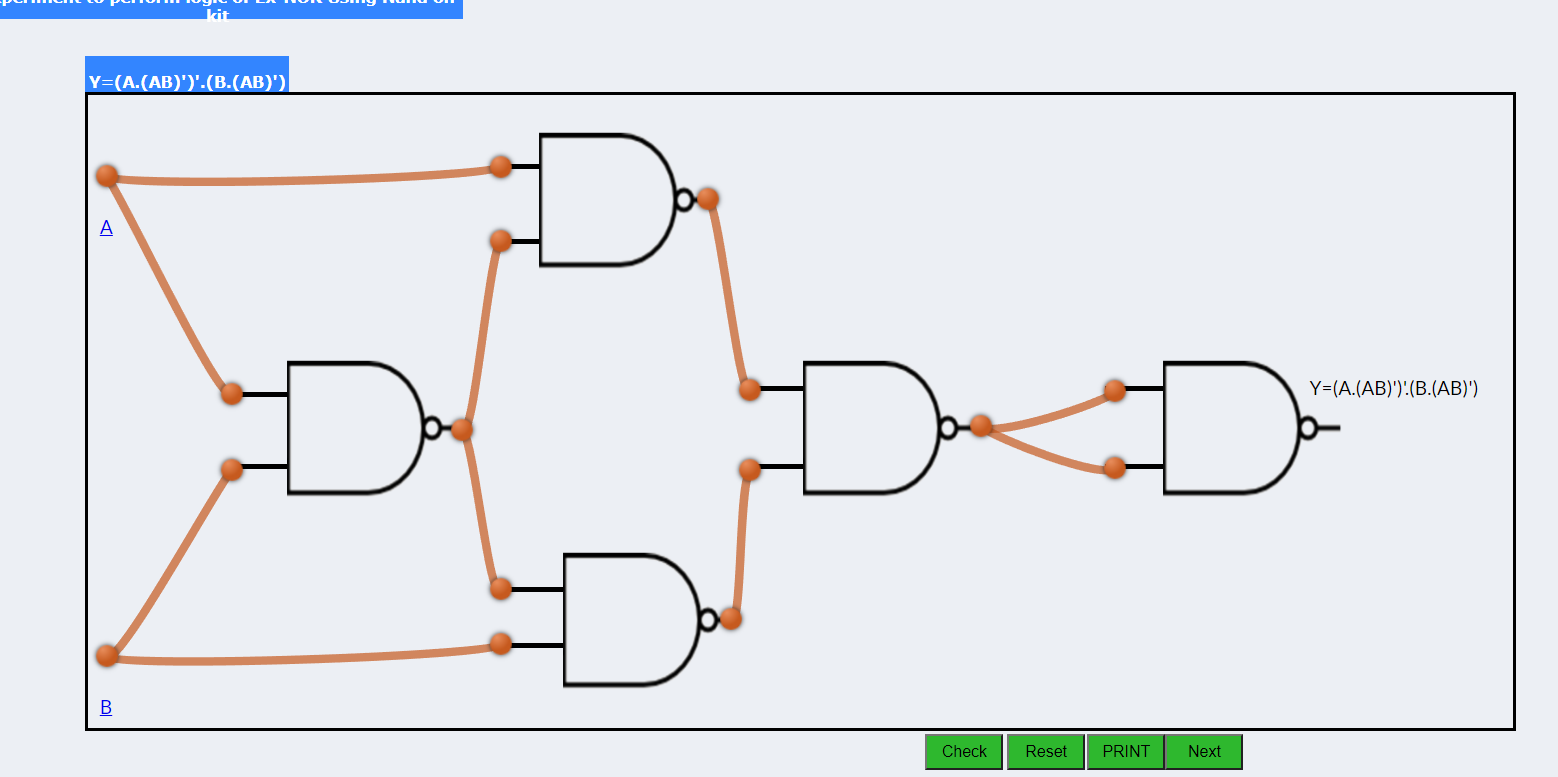
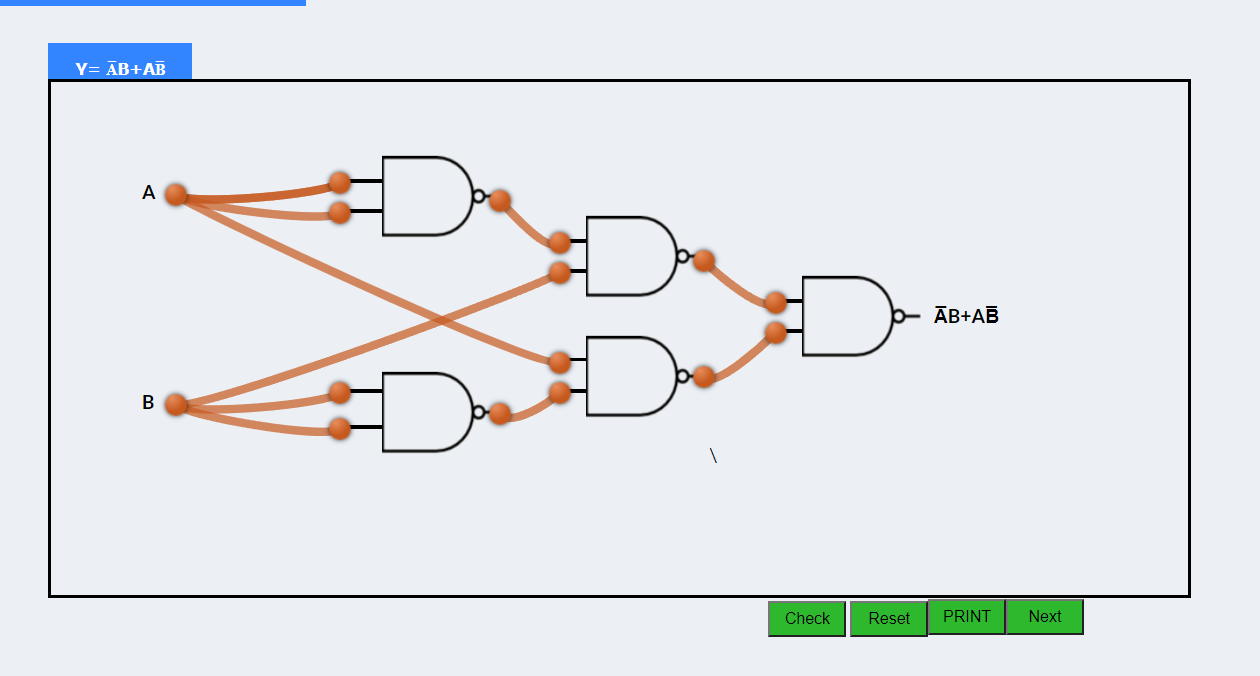
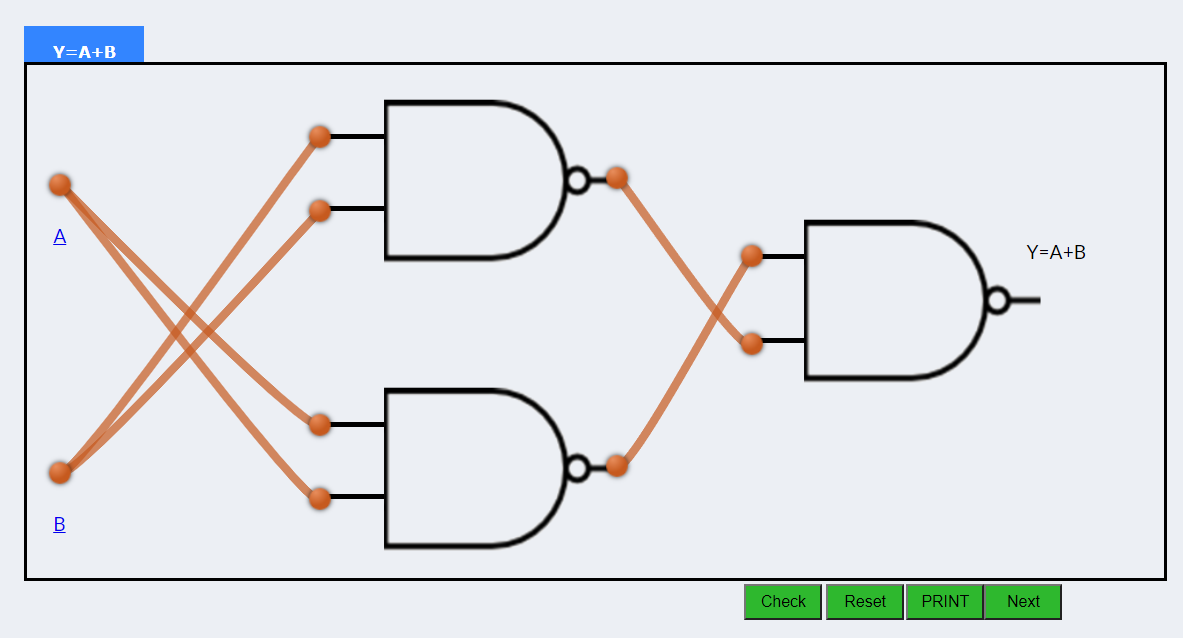
**EXOR Gate**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y** |
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**NOT gate**

|  |  |
| --- | --- |
| **A** | **Y** |
| **0** | **1** |
| **1** | **0** |

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conclusion:** Here we have designed & studied all gates using universal NAND /NOR gate. So using a single IC package we can design any logic circuit..

**Real Life Application:** Design of digital electronic circuit.

**Post Lab Questions:**

1. Why is NOR gate known as a Universal gate?

The NAND & NOR gates are called universal gates because they perform all the logical operations of basis gates like AND, OR, NOT.

1. Which statement is true

All basic logic gate function can be obtained from

1. Combination of AND and OR gates.
2. Combination of NOT and OR gates.
3. Using only NOR gates

Correct Statement-1

**Viva Questions:**

1. How is de Morgan’s Theorem applicable to conversion of other logic to NOR logic?

DeMorgan's Second theorem proves that when two (or more) input variables are OR'ed and negated, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function is a negative-AND function proving that A+B = A.

1. What is advantage of having universal logic?

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

1. What is output of NOR gate when all inputs are 0?

o/p=1

**EXPERIMENT No. 3**

**Title** : Design of adder- subtractor using gates and IC 7483

**Estimated time to complete this experiment:** 2hr

**Objective:** In this experiment, student should understand Design of full adder/full

subtractor using gates and IC 7483

**CO to be achieved:** CO1,CO3

**Expected Outcome of Experiment**: Students will be able to Design of full

adder/full subtractor using gates and IC 7483

**Books/ Journals/ Websites referred:**

1. R. P Jain: Modern digital design, forth edition, tata mcgrawhill

2. Morris Mano, digital design, Pearson education, Asia2002.

3. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.

4. John M. Yarbrough, digital logic: applications and design, Thomson brooks/ cole, 2004

5. www.alldatasheet.com,

6. www.datasheetcatalog.com.

7. en.wikipedia.org/wiki/7400\_series

**Pre Lab/ Prior Concepts:** Combinational design practices

**Historical Profile:**

It was not until the beginning of the 19th Century that the calculator became a popular

device. This happened in 1820 when Charle s Xavier Thomas de Colmar (1785-1870), of France, made the Arithmometer ( French Patent No 1420, November 18 , 1820 ), a

machine based on Leibniz’s design which was capable of performing the four operations in a simple and reliable way. Because of its unidirectional drum, division and subtraction required setting a lever. The *Arithmometer* was also known as the Thomas Machine . The Thomas Machine was very successful and one hundred years later, during the first half of the 20th Century, the machine was still sold. Thomas received France's Chevalier of the Legion of Honor for his machine. About 1,500 machines were made by the Compagnie d'Assurance Le Soleil, founded by Thomas, and other contractors, between 1820 and 1930. Between 1850 and 1887, many attempts were made to develop a calculating machine that would use keys as means to enter data. In Europe, key-driven machines were made by V. Schilt (1851), F. Arzberger (1866), A. Stettner (1882), Bagge (1882), d’Azevedo (1884), Petetin (1885) and Maq Meyer (1886). In the United States: D.D. Parmelee (1850), T . Hil l (1857), G.W. Chapin (1870), W. Robjohn (1872), D. Carrol (1876), Borland & Hoffman (1878), M. Bouchet (1883), C.G. Spalding (1884), A. Stark (1884), L.W. Swem (1885), P.T. Lindholm (1886) and B.F. Smith (1887) [7].

**New Concepts to be learned:**

Designing combinational circuits using K map and

gates and IC 7483

**Requirements:**

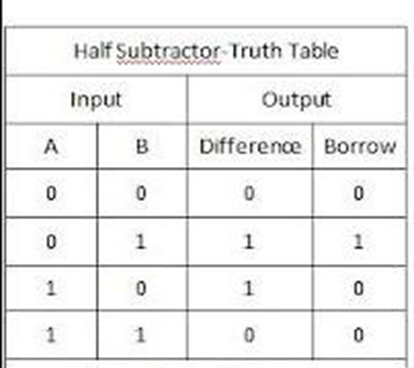
TTL IC 7486, IC 7400, IC 7483Trainer kit, wires, breadboard etc.

**Circuit Diagram:**

**Half Adder using gates**



**Half Subtractor using gates**



**Full Adder using gates**



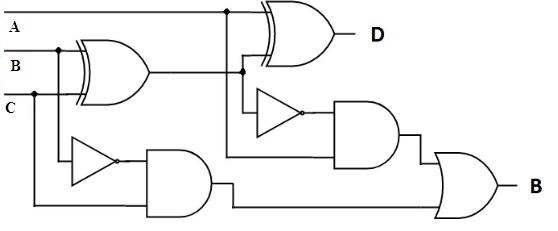
**Full Adder using two Half adders**



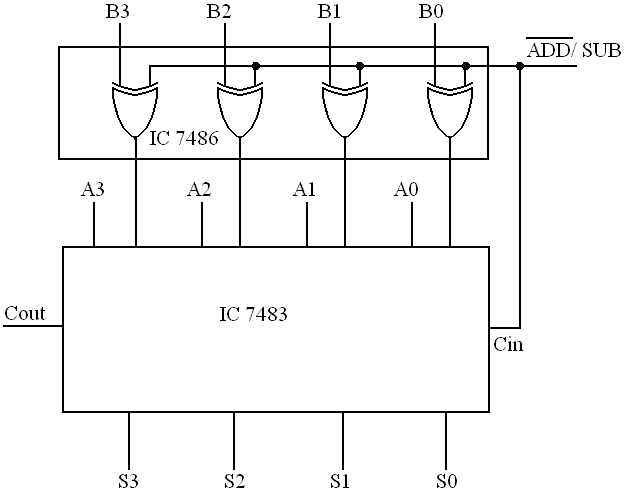
**Full Subtractor using gates**



**Full Subtractor using two Half Subtractors**

****

**Full Adder /Subtractor using IC 7483**



**Procedure:**

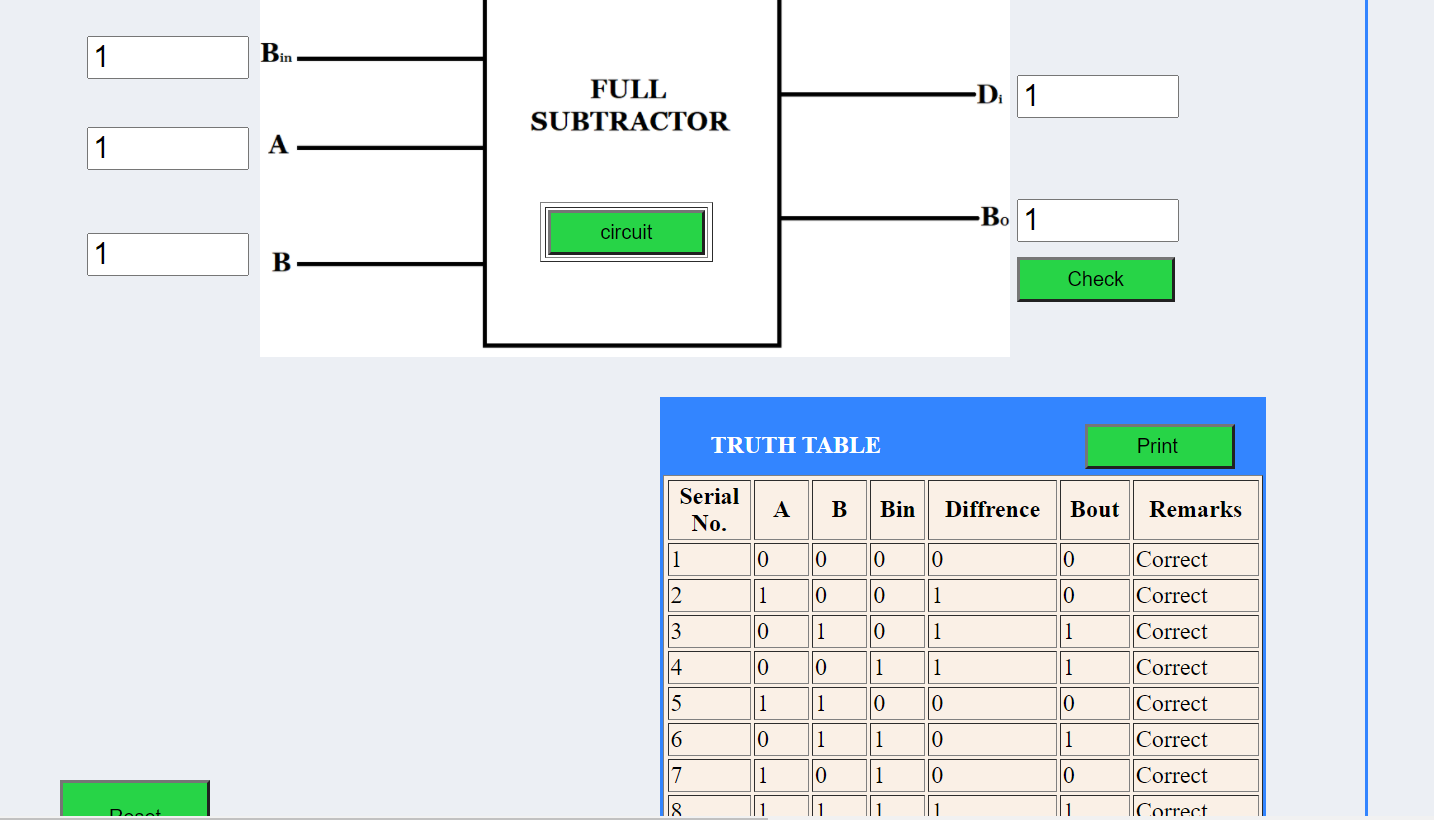
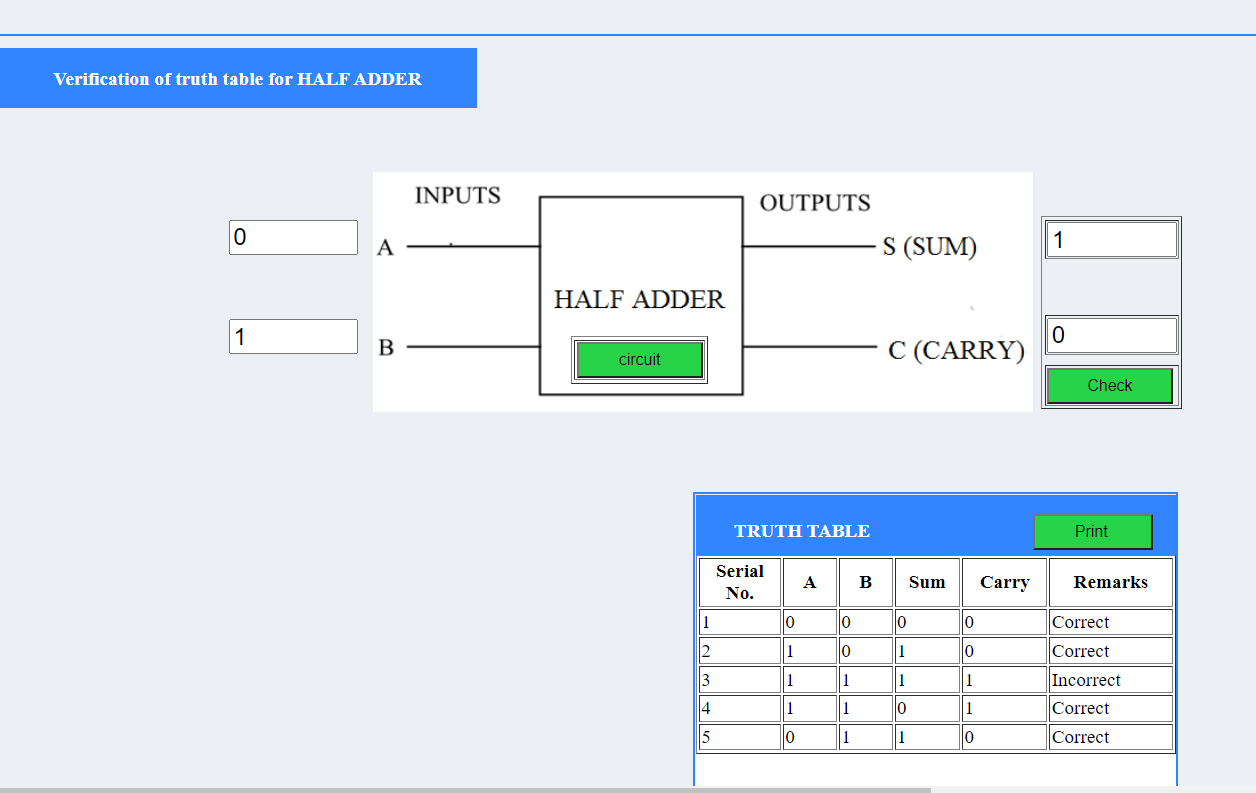
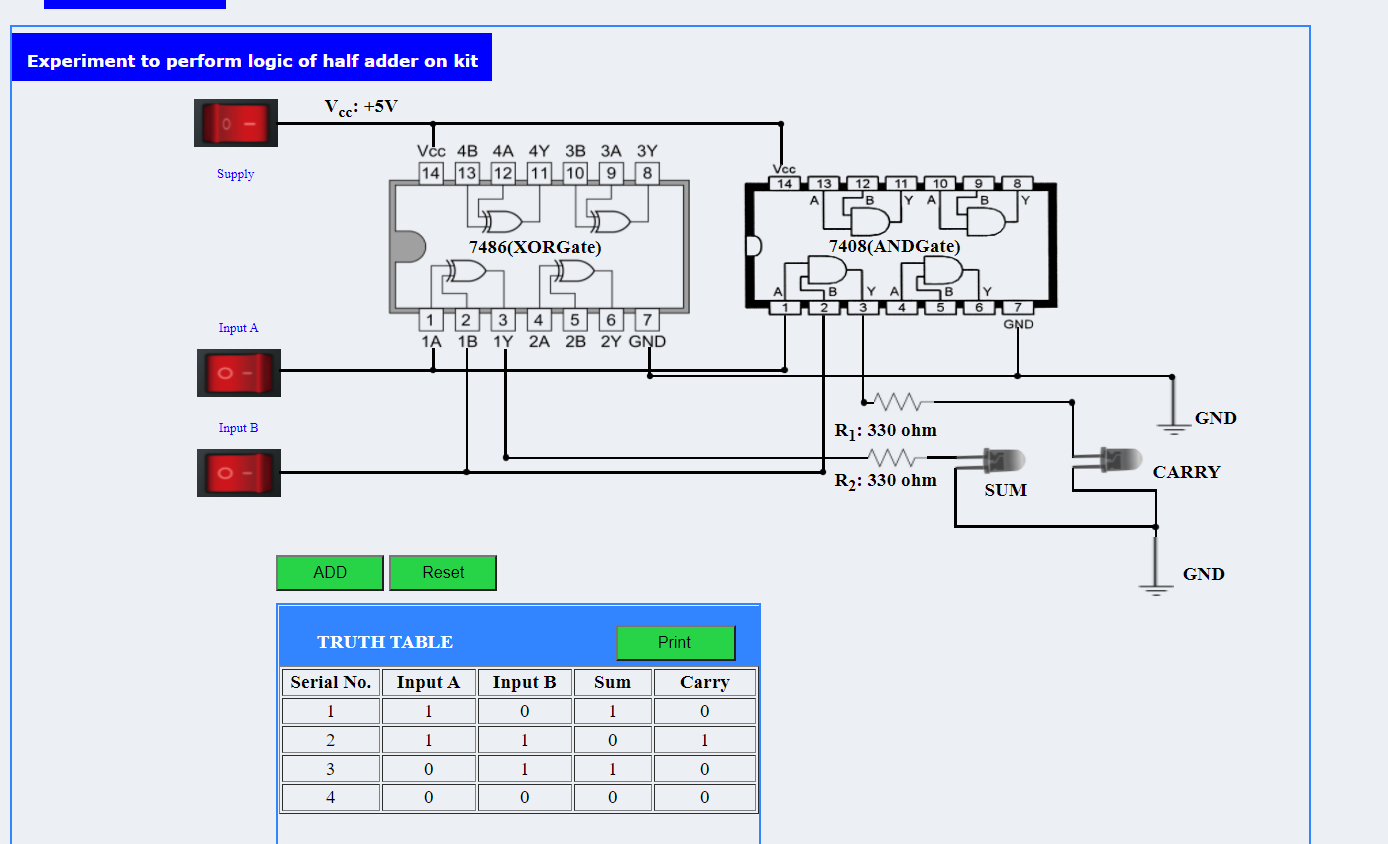
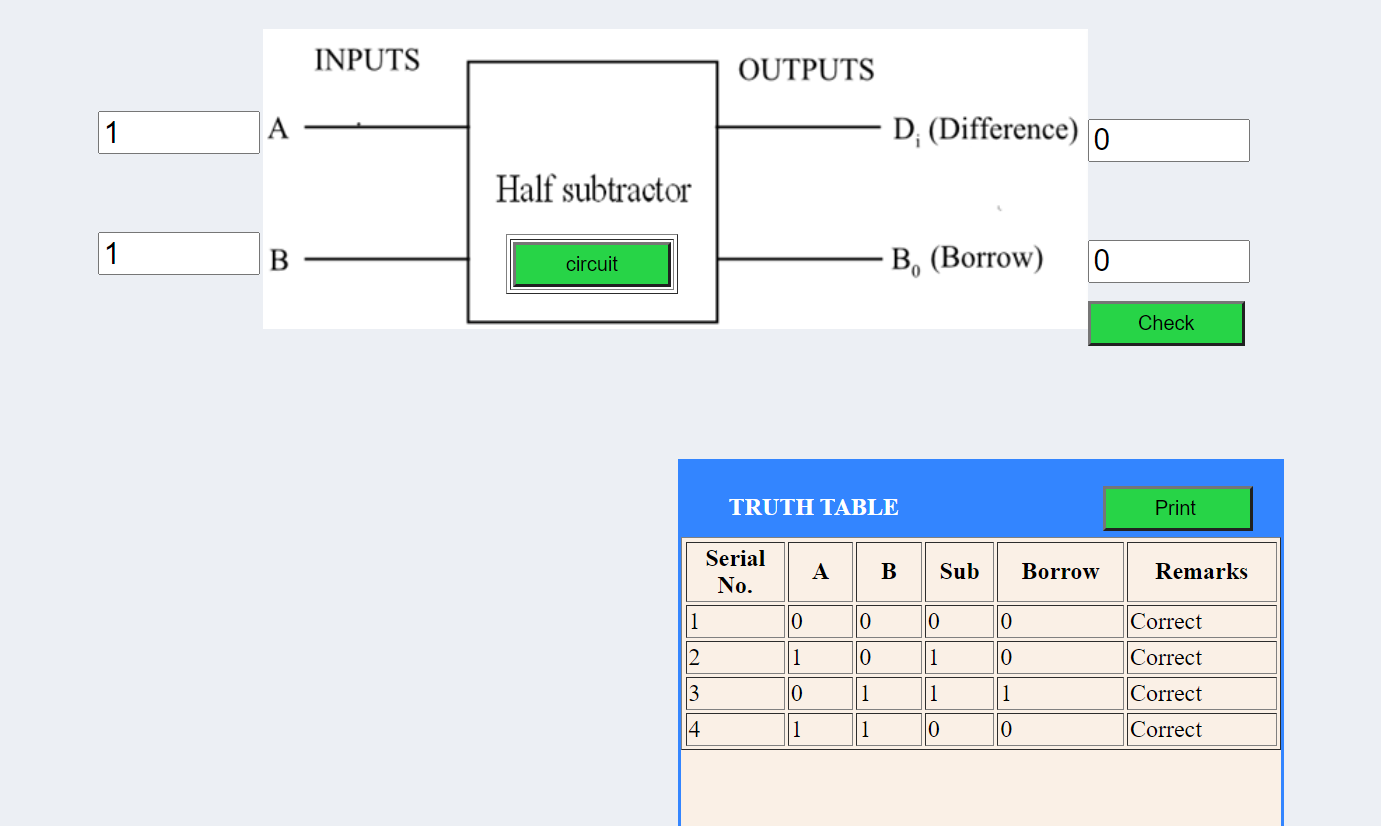
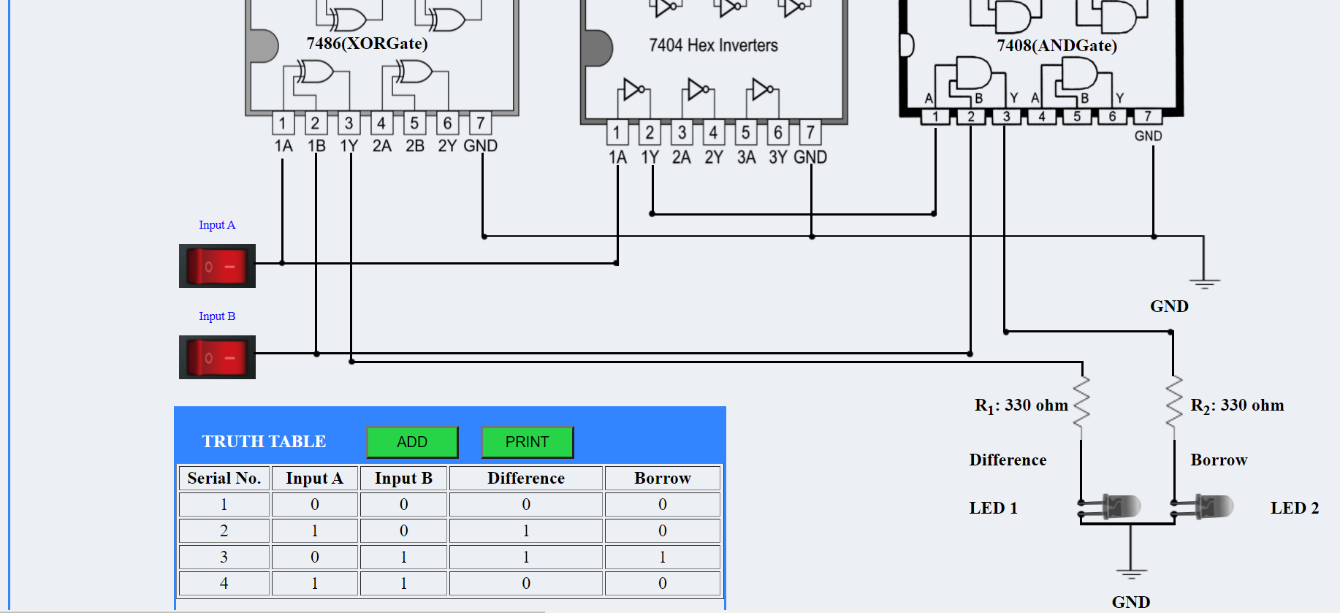
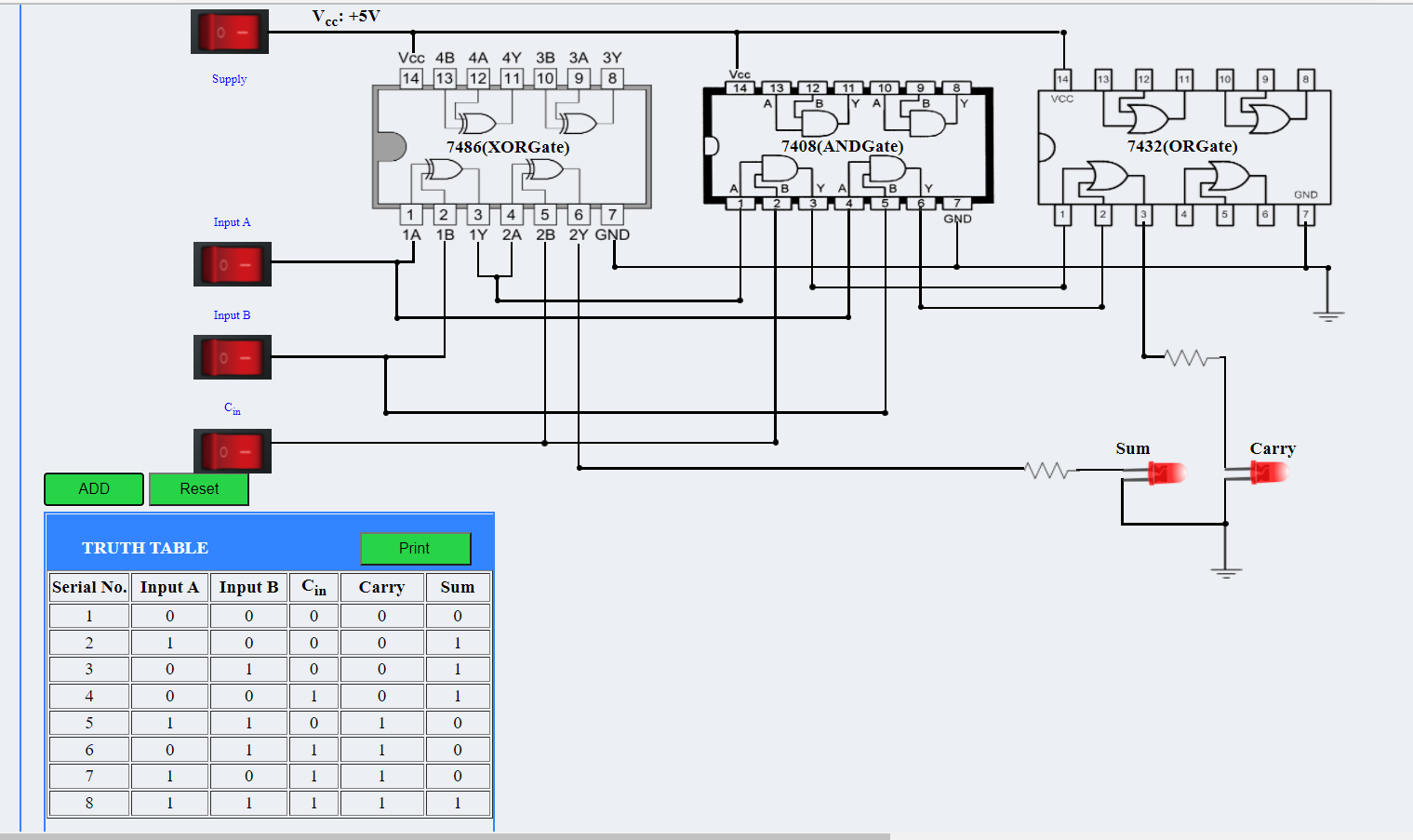
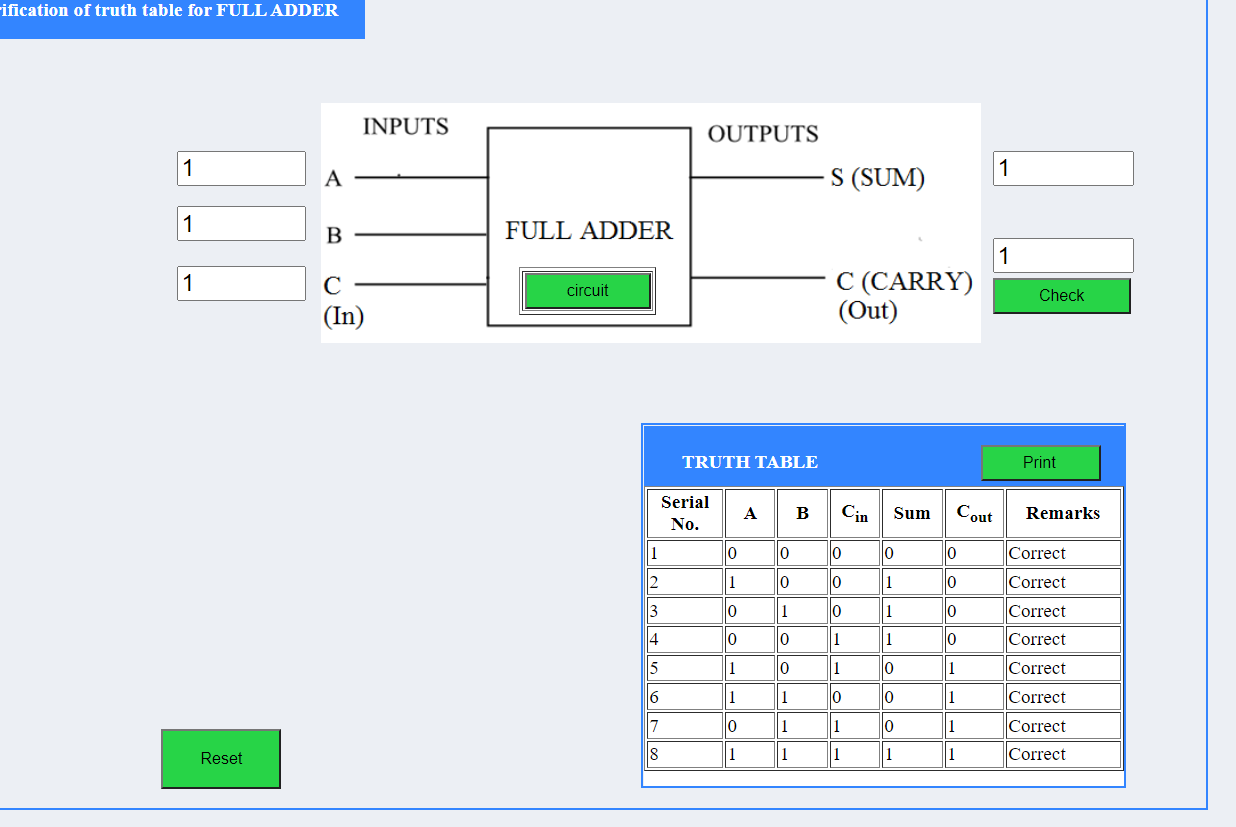
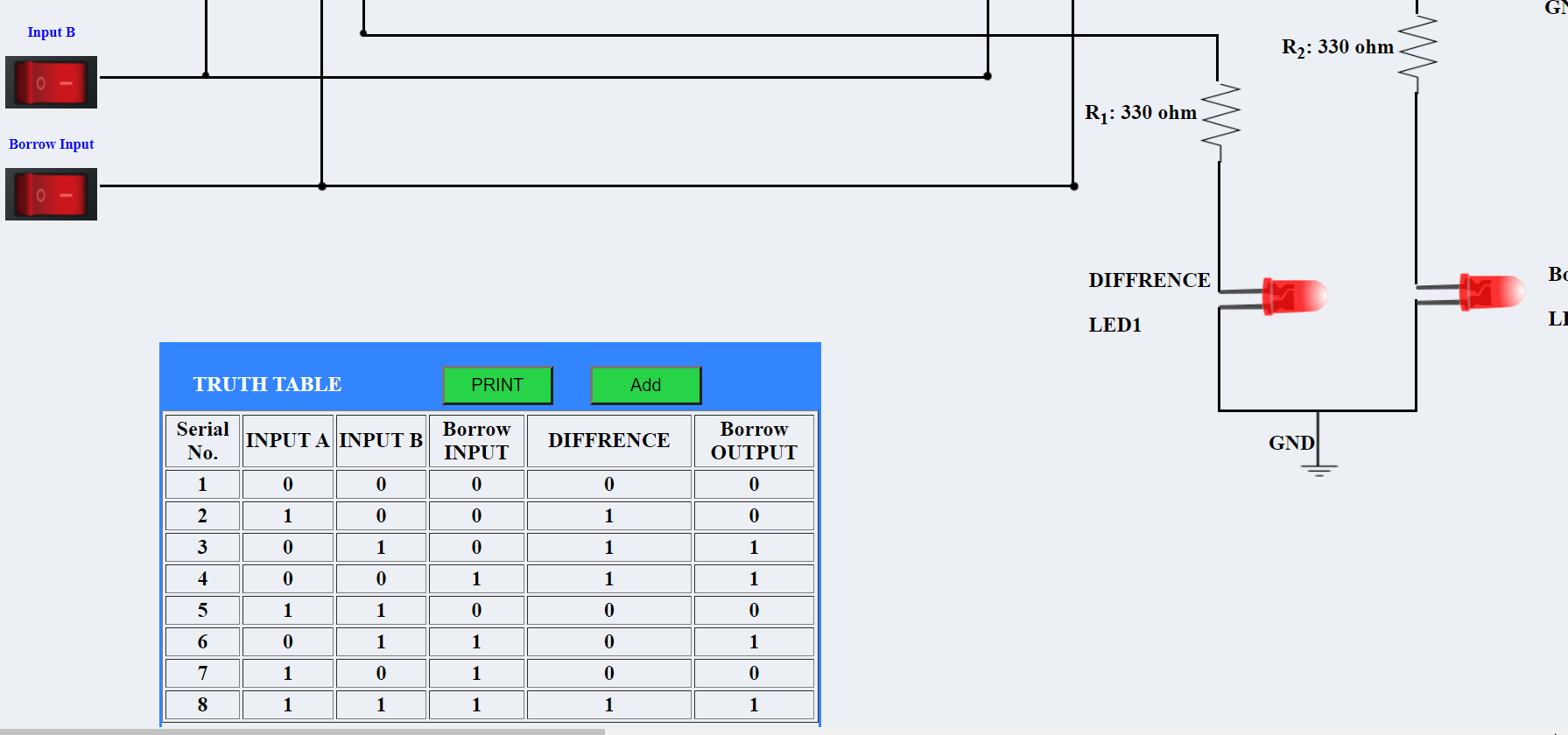
1) Make connections as shown in circuit diagram.

2) Give the i/p accordingly to verify the o/p

3) Make i/p M = 0 & verify the truth table for full adder.

4) Make i/p M = 1 & verify the truth table for full subtractor

**Output:**

****

**Conclusion:**

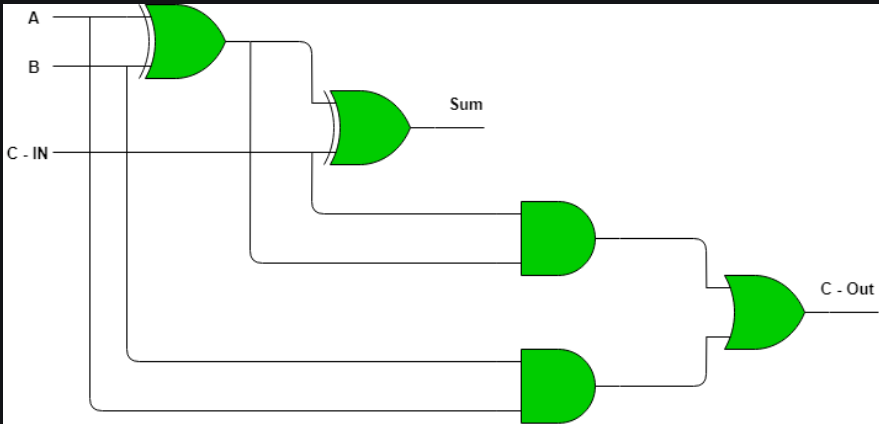
Full Adder and subtractor can be designed by using K Map and can be implemented using Gates

**Real Life Application:**

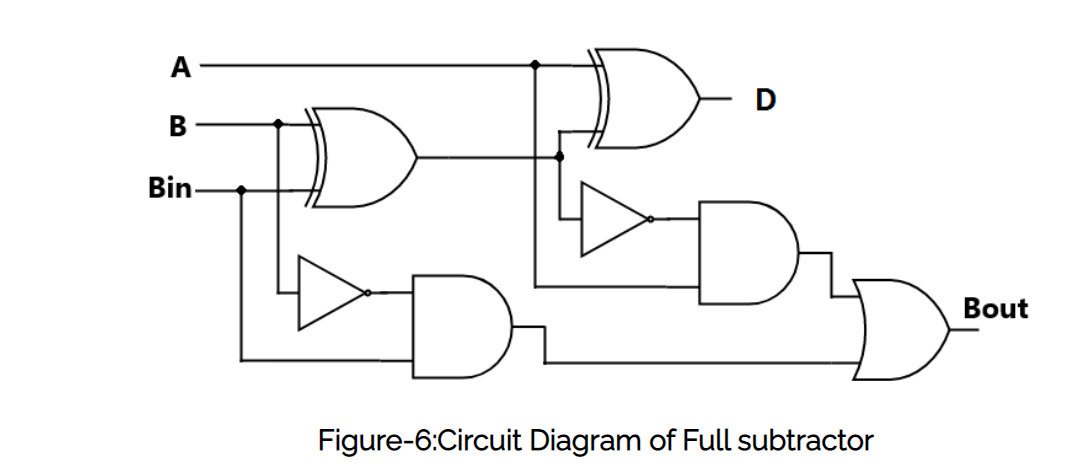
An **adder** or **summer** is a digita l circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic unit (s), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar.

**Post Lab Questions:**

1 Design full adder using gates



2 Design full subtractor using gates



3 What is carry look ahead adder?

A carry-lookahead adder or fast adder is a type of electronics adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits

**Viva Questions:**

1. What are serial adders?

The serial binary adder or bit-serial adder is a digital circuit that performs binary addition bit by bit. There are two single-bit outputs for the sum and carry out. The carry-in signal is the previously calculated carry-out signal. The addition is performed by adding each bit, lowest to highest, one per clock cycle.

2. Which adder is fastest adder?

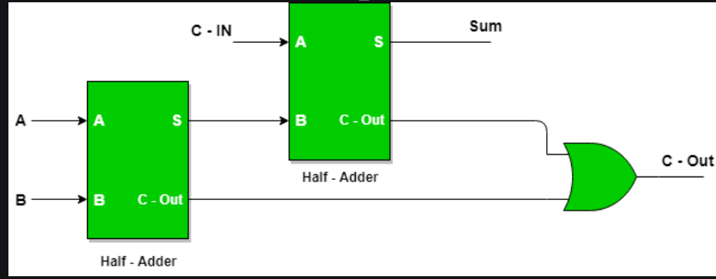
The carry lookahead adder

The speed of compute becomes the most considerable condition for a designer. The carry lookahead adder is the highest speed adder nowadays.

3. Where do we use serial adder?

A serial adder is used to add two binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers 4. How will you make Full adder using half adders?

4. How will you make Full adder using half adders?



5. Suggest an arrangement for adding more than two bits.

To add multiple binary bits together, we must include a possible carry over from the lower order of magnitude, and send it as an input carry to the next higher order of magnitude bit.

6. Suggest a circuit for subtraction using complement addition

Subtraction is done by adding the ten's complement of the subtrahend, which is the nines' complement plus 1. The result of this addition used when it is clear that the difference will be positive, otherwise the ten's complement of the addition's result is used with it marked as negative

**EXPERIMENT No. 4**

**Part A**

**Title:** To design Binary to Gray code converter

**Estimated time to complete this experiment:** 2 hours

**Objective:** To make the student understand the concept of Gray code, advantage of using unit distance code, code conversion.

**CO to be achieved: CO1,CO2,CO3**

**Expected Outcome of Experiment:** Knowledge about code conversions.

**Books/ Journals/ Websites referred:**

Books:

1. R. P Jain: Modern digital design, fourth edition, Tata mc grawhill,Morris Mano, digital design, Pearson education, Asia 2002.
2. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.
3. John M. Yarbrough, digital logic: applications and design,

Thomson brooks/ cole, 2004

Websites:

1. www.all**datasheet**.com,
2. [www.**datasheet**catalog.com](http://www.datasheetcatalog.com).
3. en.wikipedia.org/wiki/**7400**\_series

**Pre Lab/ Prior Concepts:** binary codes, operation of ex-or gate.

**New Concepts to be learned:** code conversions from binary to gray, use of these codes in K map.

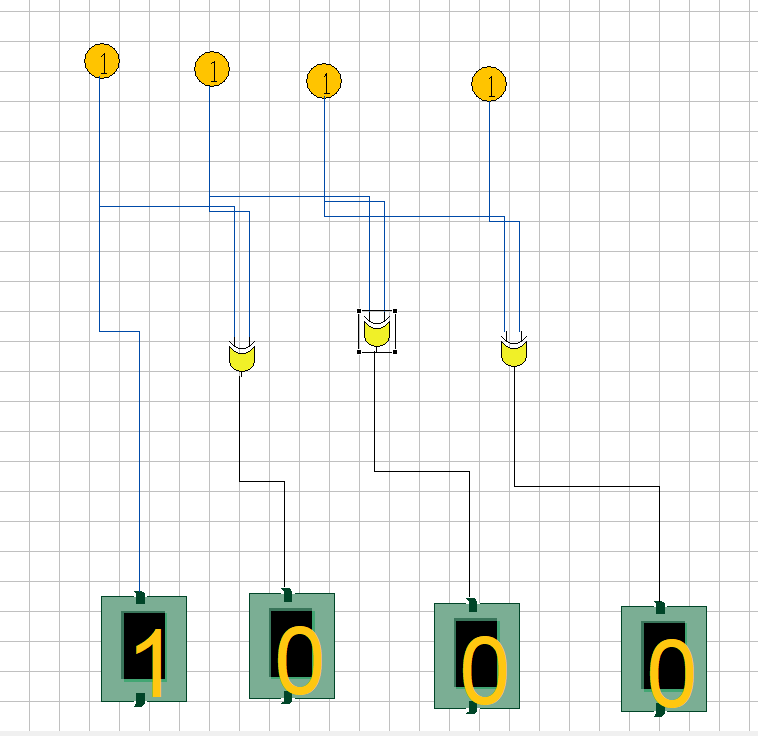
**Requirements:** IC’s 7486, patch cords & trainer kit

|  |
| --- |
| **Theory:** gray code is a non weighted code. It is not an arithmetic code. It has a special feature that only one bit changes each time a decimal is incremented. As only one bit changes at a time, it is known as unit distance code.  **Procedure**:  1) Connect the power supply & switch ON the circuit  2) Check the supply LED glows.  3)Use IC of EX-OR gate i.e.7486, to create combination of logic binary to gray code converter, with no. 7-pin ground & no. 14-pin Vcc, and according to the design makes the connection of breadboard using wires.  4)Give different logic inputs to the gates & check the according to the truth table.  5)If the output is high, the LED, will glow otherwise, it will not glow. |
| |  |  | | --- | --- | |  |  | |

**Results:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
| **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** |

**Output:**



**Conclusion:** Gray code is a useful code in which a decimal number is represented in binary form in such a way so that each gray-code number differs from the preceding and the succeeding number by a single bit.

**Real Life Application:** Gray codes are used in rotory encoders

.

**Post Lab Questions:**

1. What is the advantage of using gray code in K Map?

In k-map the same simplification of Boolean expression is done by pairing adjacent 1's for reducing variables. so if gray code is used in K-map the adjacent cells can be easily paired as they differ by only one bit and final expression can be obtained.

1. Convert the full decimal numbers into gray code :
2. 32- 00110000
3. 45 - 00111011
4. 67- 01100010
5. 27- 00010110
6. 205- 10101011
7. Which of the following codes are weighted
8. BCD
9. Gray
10. Excess 3

BCD code is weighted

1. The most suitable gate for comparing two bits is –
2. AND
3. OR
4. NAND
5. EX-OR

AND Gate

**Part B**

**Title:** Todesign Gray to Binary code converter.

**Estimated time to complete this experiment:** 2 Hrs

**Objective:** Concept of Gray code, advantage of using unit distance code, code conversion

**CO to be achieved: CO1,CO2,CO3**

**Expected Outcome of Experiment** : Knowledge about code conversions.

**Books/ Journals/ Websites referred:**

1. R. P Jain: Modern digital design, forth edition, tata mcgrawhill

2. Morris Mano, digital design, Pearson education, Asia2002.

3. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.

4. John M. Yarbrough, digital logic: applications and design, Thomson brooks/ cole, 2004

5. www.alldatasheet.com,

6. www.datasheetcatalog.com.

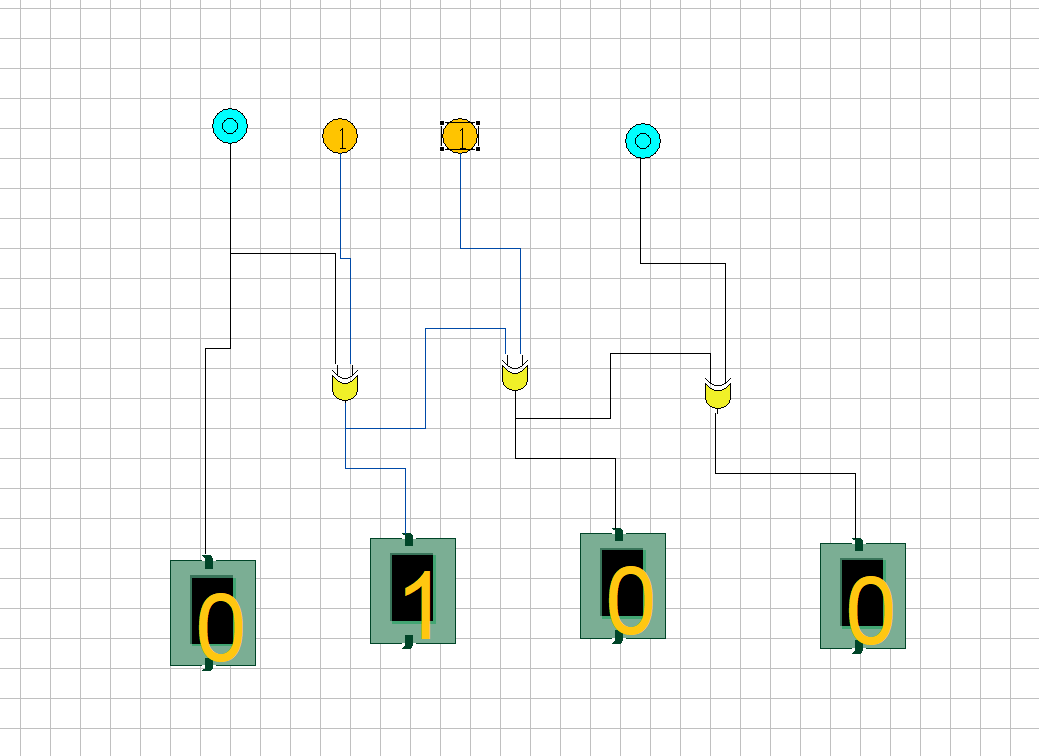
7. en.wikipedia.org/wiki/7400\_series

**Pre Lab/ Prior Concepts:** concept of Gray codes, Binary to gray code conversion and vice versa

|  |  |  |
| --- | --- | --- |
| **Procedure** | : | 1. Connect the power supply & switch ON the circuit 2. Check the supply LED glows. 3. Use IC of EX-OR gate i.e.7486, to create combination of logic binary code converter, with no. 7-pin ground & no. 14-pin Vcc, and according to the design makes the connection of breadboard using wires. 4. Give different logic inputs to the gates & check the according to the truth table. 5. If the output is high, the LED, will glow otherwise, it will not glow. |
| Diagrams | : |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **G3** | **G2** | **G1** | **G0** | **B3** | **B2** | **B1** | B0 |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **0** | **1** | **0** | **1** |
| **0** | **1** | **0** | **1** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** | **1** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** | **1** | **0** | **1** |
| **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** |

**Output:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conclusion:** Binary code is a 4-bit code & used to count the numbers from 0 to 15. Various arithmetic operations can be performed in this form they are weighted codes. With the help of truth table & K- map reduction, we can convert the Binary codes to Gray code using gates

**Real Life Application:** Applications of grey codes are used in rotary encoders, in K Maps

**Post Lab Questions:**

1. What is code converter?

The Code converter is used to convert one type of binary code to another. There are different types of binary codes like BCD code, gray code, excess-3 code, etc. Different codes are used for different types of digital applications.

1. How do you convert from Gray to Binary using IC 7486?

The Gray Code or RBC (reflected binary code), or cyclic code is a series of binary number systems. The main reason to call this reflected binary code is the initial N/2 values are in reverse order as compare with the last N/2 values. In this kind of code, the two successive values are changed through a single bit of binary digits. These codes are mainly used in the common series of binary numbers generated by hardware

**Viva Questions:**

1. What are the advantages of Gray code?

Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next. So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.

1. What is the difference between weighted and non-weighted codes?

In weighted codes, each digit is a assigned a specific weight according to its position. NON- WEIGHTED CODE - The Non - Weighted Code are not positionally weighted. In other words, codes that are not assigned with any weight to each digit position

**Experiment No: 5**

**Title:** Ripple Carry Adder

**Estimated time to complete this experiment:** 2 hours

**Objective:** To understand the operation of a ripple carry adder, specifically how the carry ripples through the adder.

1. Examining the behaviour of the working module to understand how the carry ripples through the adder stages
2. To design a ripple carry adder using full adders to mimic the behaviour of the working module
3. The adder will add two 4 bit numbers

**Books/ Journals/ Websites referred:**

Books:

1. Digital Logic and Computer Design - M. Morris Mano. Pearson Education - Prentice Hall.
2. Digital Principles Foundation of Circuit Design and Application - Arun Kumar Singh. New Age Publishers.
3. The Art of Electronics - Paul Horowitz and Winfield Hill (1989). Cambridge University Press
4. Modern Dictionary of Electronics - Rudolf F. Graf (1999). Newnes

Web Sites:

* [http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Comb/ adder.html](http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Comb/adder.html)
* <http://en.wikipedia.org/wiki/Adder_(electronics)>
* [NPTEL (e-learning courses from IITs and IISC)](http://nptel.iitm.ac.in/courses.php?disciplineId=106)

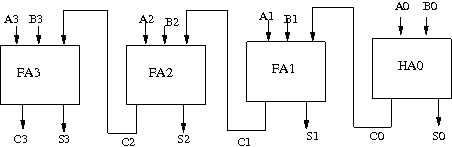
**Requirements:** Virtual simulator.

**Components:** The components needed to create 4 bit ripple carry adder is listed here -

* 4 full-adders
* Wires for the connections
* LED display to obtain the output

Optionally we can create the same using the following components also-

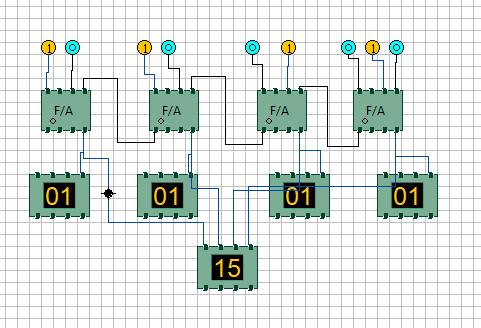
* 3 full-adders
* 1 half adder
* Wires fro the connections.
* LED display to obtain the output

**Circuit Diagram:**

**Procedure:**

1. Start the simulator as directed.
2. To design the circuit we need 3 full adder, 1 half adder, 8 Bit switch(to give input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1, For full adder input is in pin-5,6,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 3 full adders(from the Adder drawer in the pallet), 8 Bit switches, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet ,if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect 4 bit switches to the 4 terminals of a digital display and another set of 4 bit switches to the 4 terminals of another digital display. connect the pin-1 of the full adder which will give the final carry output. connect the sum(pin-4) of all the adders to the terminals of the third digital display(according to the circuit diagram shown in screenshot). After the connection is over click the selection tool in the palate.
7. To see the circuit working, click on the Selection tool in the pallet then give input by double clicking on the bit switch, (let it be 0011(3) and 0111(7)) you will see the output on the output(10) digital display as sum and 0 as carry in bit display.

**Output**:



**Conclusion:**  This experiment provides information about how by using Full Adder ICs in a cascading connection we can implement addition of 2 n-bit numbers(Here 4).

**Real Life Application:**

1. Incrementer.
2. BCD adder.
3. Binary Arithmetic.

**Post Lab Questions:**

1. **What is the gate delay in a 32-bit ripple carry adder?**

65

Each full adder requires three levels of logic. In a 32-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is 31 \* 2(for carry propagation) + 3(for sum) = 65 gate delays.

1. **What is the disadvantage of ripple-carry adder?**

Ripple Carry Adder does not allow to use all the full adders simultaneously.

Each full adder has to necessarily wait until the carry bit becomes available from its adjacent full adder.This increases the propagation time.Due to this reason, ripple carry adder becomes extremely slow.This is considered to be the biggest disadvantage of using ripple carry adder.

1. **What is the major difference between half-adders and full-adders?**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Half Adder | Full Adder |
| 1. | Half Adder is combinational logic circuit which adds two 1-bit digits. The half adder produces a sum of the two inputs. | Full adder is combinational logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the three inputs and carry value. |
| 2. | Previous carry is not used. | Previous carry is used. |
| 3. | In Half adder there are two input bits (A, B). | In full adder there are three input bits (A, B, C-in). |
| 4. | Logical Expression for half adder is: S=a⊕b; C=a\*b. | Logical Expression for Full adder is:  S=a⊕b⊕Cin; Cout=(a\*b)+(Cin\*(a⊕b)). |
| 5. | It consists of one EX-OR gate and one AND gate. | It consists of two EX-OR, two AND gate and one OR gate. |
| 6 | It is used in Calculators, computers, digital measuring devices etc. | It is used in Multiple bit addition, digital processors etc. |

**Experiment No: 6**

**Title:** Carry Look Ahead Adder

**Estimated time to complete this experiment:** 2 hours

**Objective:**

1. Understanding behaviour of carry lookahead adder from module designed by the student as part of the experiment
2. understanding the concept of reducing computation time with respect of ripple carry adder by using carry generate and propagate functions
3. the adder will add two 4 bit numbers

**Books/ Journals/ Websites referred:**

Books:

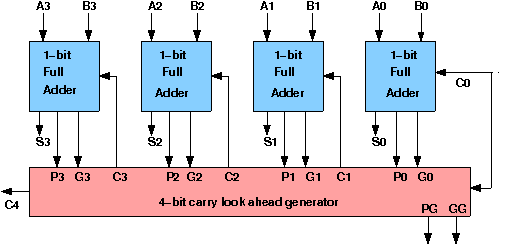
1. Digital Logic and Computer Design - M. Morris Mano. Pearson Education - Prentice Hall.
2. Digital Principles Foundation of Circuit Design and Application - Arun Kumar Singh. New Age Publishers.
3. The Art of Electronics - Paul Horowitz and Winfield Hill (1989). Cambridge University Press
4. Modern Dictionary of Electronics - Rudolf F. Graf (1999). Newnes

Web Sites:

* [http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Comb/ adder.html](http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Comb/adder.html)
* <http://en.wikipedia.org/wiki/Adder_(electronics)>
* [NPTEL (e-learning courses from IITs and IISC)](http://nptel.iitm.ac.in/courses.php?disciplineId=106)

**Requirements:** Virtual simulator.

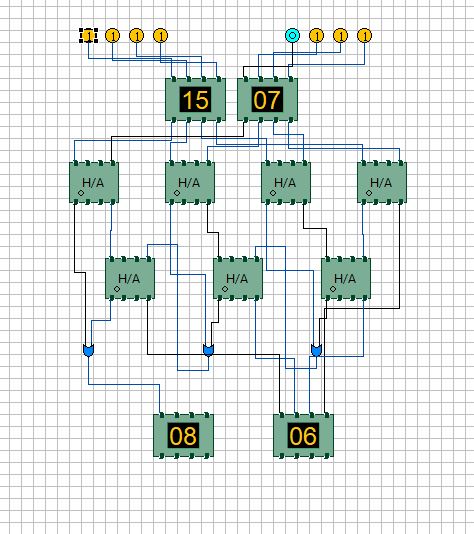
* **Components :-** 7 half-adders: 4 to create the look adder circuit, and 3 to evaluate Si and Pi · Ci
* 3 OR gates to generate the next level carry Ci+1
* Wires to connect
* LED display to obtain the output

**Circuit Diagram:**

**Procedure:**

1. Start the simulator as directed.
2. To design the circuit we need 7 half adder, 3 OR gate, 1 V+(to give 1 as input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 6 more full adders(from the Adder drawer in the pallet), 3 OR gates(from Logic Gates drawer in the pallet), 1 V+, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet, if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Pallet, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect V+ to the upper input terminals of 2 digital displays according to you input. connect the OR gates according to the diagram shown in the screenshot connect the pin-1 of the half adder which will give the final carry output. connect the sum(pin-4) of those adders to the terminals of the third digital display which will give output sum. after athe connection is over click the selection tool in the pallet.
7. See the output, in the screenshot diagram we have given the value 0011(3) and 0111(7) so get 10 as sum and 0 as carry.you can also use many bit switches instead of V+ to give input and by double clicking those bit switches can give different values and check the result.

**Output**:



**Conclusion:**  The drawbacks of ripple carry adder are avoided using carry look ahead adder which generates all the carry bits from a single carry bit i.e. C0.

**Real Life Application:**

1.Incrementer.

2.BCD adder.

3.Binary Arithmetic.

**Post Lab Questions:**

1. **What is the reason for using look ahead carry adder?**

Unlike Ripple Carry Adder, the Carry-look ahead adder does not need to wait for the previous adder circuit to generate the carry. It can generate the carry for all the adders simultaneously. This reduces the propagation delay.

**Experiment No. 7**

**Title:** Design Flip-Flops using gates and flip-flops and conversions.

.

**Estimated time to complete this experiment:** 2 hours

**Objective:** Flip Flops working and their conversions

**CO to be achieved:** CO5

**Expected Outcome of Experiment:**  knowledge about flip-flops & its conversions.

**Books/ Journals/ Websites referred:**

Books:

1. R. P Jain: Modern digital design, fourth edition, Tata mc grawhill, Morris Mano, digital design, Pearson education, Asia 2002.
2. John f. Wakerley, digital design principles and practices third edition updated Pearson education, Singapore, 2002.
3. John M. Yarbrough, digital logic: applications and design,

Thomson brooks/ Cole, 2004

Websites:

1. www.alldatasheet.com,
2. [www.**datasheet**catalog.com](http://www.datasheetcatalog.com).
3. en.wikipedia.org/wiki/7400\_series

**Pre Lab/ Prior Concepts:** knowledge about flip-flops & its conversions

**Historical Profile:**

The first electronic flip-flop was invented in 1918 by [William Eccles](http://en.wikipedia.org/wiki/William_Eccles) and [F. W. Jordan](http://en.wikipedia.org/wiki/F._W._Jordan).It was initially called the *Eccles–Jordan trigger circuit* and consisted of two active elements ([vacuum tubes](http://en.wikipedia.org/wiki/Vacuum_tube)). Such circuits and their transistorized versions were common in computers even after the introduction of [integrated circuits](http://en.wikipedia.org/wiki/Integrated_circuit), though flip-flops made from [logic gates](http://en.wikipedia.org/wiki/Logic_gate) are also common now. Early flip-flops were known variously as trigger circuits or [multivibrators](http://en.wikipedia.org/wiki/Multivibrator).

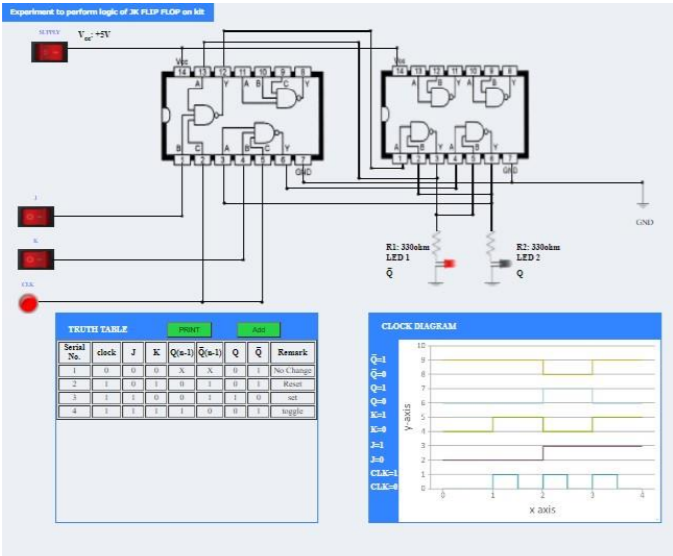
According to P. L. Lindley, a [JPL](http://en.wikipedia.org/wiki/JPL) engineer, the flip-flop types discussed below (RS, D, T, JK) were first discussed in a 1954 [UCLA](http://en.wikipedia.org/wiki/UCLA) course on computer design by Montgomery Phister, and then appeared in his book *Logical Design of Digital Computers.*[[8]](http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29#cite_note-8)[[9]](http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29#cite_note-9) Lindley was at the time working at Hughes Aircraft under Eldred Nelson, who had coined the term JK for a flip-flop which changed states when both inputs were on (a logical "one"). The other names were coined by Phister. They differ slightly from some of the definitions given below. Lindley explains that he heard the story of the JK flip-flop from Eldred Nelson, who is responsible for coining the term while working at [Hughes Aircraft](http://en.wikipedia.org/wiki/Hughes_Aircraft). Flip-flops in use at Hughes at the time were all of the type that came to be known as J-K. In designing a logical system, Nelson assigned letters to flip-flop inputs as follows: #1: A & B, #2: C & D, #3: E & F, #4: G & H, #5: J & K. Nelson used the notations "*j*-input" and "*k*-input" in a patent application filed in 1953.

**New Concepts to be learned:** Obtaining a Flip flop from another Flip Flop

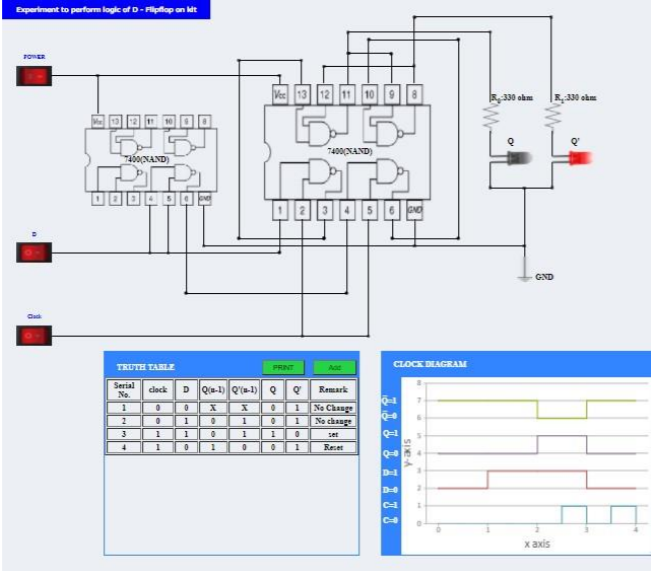
**Requirements:** Logic circuit trainer, bread board,Single strand wire,IC-7476 (JK flip-flop), IC-7474 (D flip-flop), IC-7400, IC-7404

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Procedure**:   1. Connect the circuit for SR Flip Flop as shown in the diagram Verify the truth table of SR flip flop 2. Convert SR Flip flop to D flip flop as shown in the diagram Verify the Truth table 3. Connect the diagram of JK Flip flop using Nand gates Verify the truth table 4. Convert JK Flip flop to T flip flop Verify the truth table   **Diagram:**      Sr Flip Flop:   |  |  |  |  | | --- | --- | --- | --- | | INPUT | |  | OUTPUT | | Sn | Rn |  | Qn+1 | | 0 | 0 |  | Qn | | 0 | 1 |  | 1 | | 1 | 0 |  | 0 | | 1 | 1 |  | ? |   JK Flip Flop   |  |  |  | | --- | --- | --- | | **INPUT** | | **OUTPUT** | | **Jn** | **Kn.** | **Qn+1** | | **0** | **0** | **Qn** | | **0** | **1** | **1** | | **1** | **0** | **0** | | **1** | **1** | **Qn** |   D Flip Flop   |  |  | | --- | --- | | D | Q n+1 | | 0 | 0 | | 1 | 1 |   T Flip Flop   |  |  | | --- | --- | | T | Q n+1 | | 0 | Q n | | 1 | Q n \* |   **Output:**  **SR Flip Flop:** |

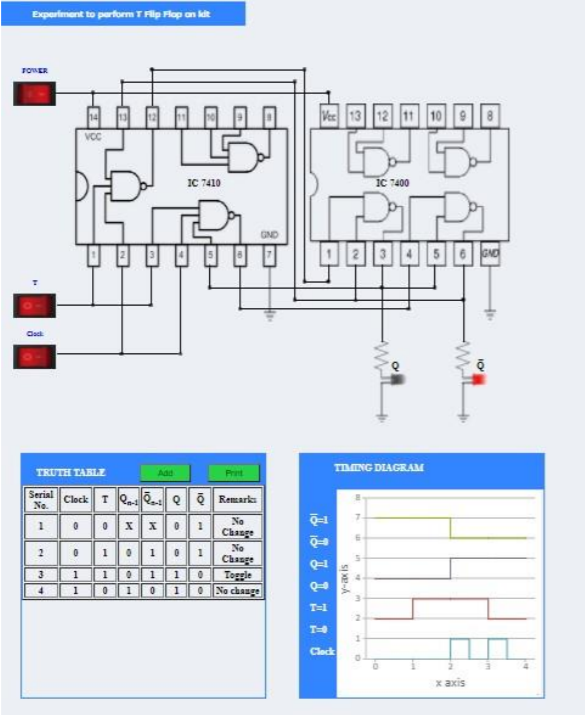
**JK Flip Flop:**



**D Flip Flop:**



**T Flip Flop:**

****

**Result/Conclusion:**

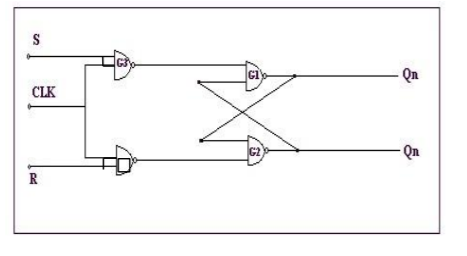
Working and analysis of various flip flops is done and the results are verified experimentally. Conversion of one flip flop to other is performed and its results are verified

**Real Life Application:**

1. Flip flops are used to store a bit of dsata
2. They are used in memories
3. They are used in counters and registers

**Post Lab Questions:**

1. What is a latch? What is the difference between latch and Flip Flop?



Latches and flip flops both are basically the bistable elements. A latch has got an enable input. As long as it is active, the latch output will keep changing according to the changes in its input. In other words, latch is a level triggered flip flop.

But a flip flop is a sequential circuit with generally samples its inputs and changes its outputs only at particular instance of time and not continuously.

The flip flops are therefore said to be at sensitive or edge triggered rather than being level triggered like latches.

1. Explain clocking operations of the following types of Flip Flop
   * 1. Positive edge triggered

When a flip flop is required to respond at a low to high transition state is known as positive edge triggering.

* + 1. Negative edge triggered

When a flip flop is required to respond at a high to low transition state is known as negative edge triggering.

* + 1. Level triggered

Edge triggering is a type of triggering that allows a circuit to become active at the positive edge or the negative edge of the clock signal.

**Experiment No. : 08**

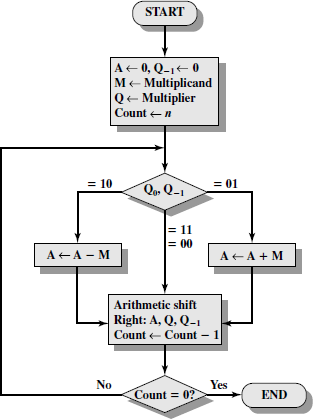
**Name of the Experiment:** Implementation of Booth’s Algorithm.

**Aim:** To implement Booth’s Algorithm.

**Theory:**

Booth's algorithm is a multiplication algorithm that multiplies two Signed binary numbers in 2's compliment notation. It gives a procedure for **multiplying binary integers** in signed 2’s complement representation **in efficient way**, i.e., less number of additions / subtractions required. It operates on the fact that strings of 0’s in the multiplier require no addition but just shifting and a string of 1’s in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1 ) to 2^m.

**Flowchart:**



**Program:**

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

void binary();

void sum(int num[]); void arithmetic\_shift();

int comparison[5] = {1, 0, 0, 0, 0};

int first\_number[5] = {0}, second\_number[5] = {0}, anumcp[5] = {0};

int compare\_num1[5] = {0}, compare\_num2[5] = {0}, product[5] = {0}, result[5] = {0}; int num1 = 0, num2 = 0, num3 = 0;

int m = 0, n = 0;

int main()

{

int count, x = 0;

printf("Enter Two Numbers to Multiply (Less Than 16)\n"); do

{

printf("Enter A:\t"); scanf("%d", &num1); printf("Enter B:\t"); scanf("%d", &num2);

}while(num1 >=16 || num2 >=16);

printf("\nExpected Product of %d \* %d = %d", num1, num2, num1 \* num2);

binary();

printf("\n\nBinary Equivalents\n"); printf("\nA:\t");

for(count = 4; count >= 0; count--)

{

printf("%d", first\_number[count]);

}

printf("\nB:\t");

for(count = 4; count >= 0; count--)

{

printf("%d", second\_number[count]);

}

printf("\nB'+ 1 = ");

for(count = 4; count >= 0; count--)

{

printf("%d", compare\_num2[count]);

}

printf("\n");

for(count = 0; count < 5; count++)

{

if(first\_number[count] == x)

{

printf("\n-->"); arithmetic\_shift();

x = first\_number[count];

}

else if(first\_number[count] == 1 && x == 0)

{

printf("\n-->");

printf("\nSUB B: "); sum(compare\_num2); arithmetic\_shift();

x = first\_number[count];

}

else

{

printf("\n-->");

printf("\nADD B: "); sum(second\_number); arithmetic\_shift();

x = first\_number[count];

}

}

printf("\nProduct:\t");

for(count = 4; count >= 0; count--)

{

printf("%d", product[count]);

}

for(count = 4; count >= 0; count--)

{

printf("%d", anumcp[count]);

}

printf("\n"); return 0;

}

void binary()

{

m = fabs(num1); n = fabs(num2);

int r2, remainder, count, temp; for(count = 0; count < 5; count++)

{

remainder = m % 2;

m = m / 2; r2 = n % 2;

n = n / 2;

first\_number[count] = remainder; anumcp[count] = remainder; second\_number[count] = r2;

if(r2 == 0)

{

compare\_num2[count] = 1;

}

if(remainder == 0)

{

compare\_num1[count] =1;

}

}

num3 = 0;

for(count = 0; count < 5; count++)

{

result[count] = comparison[count]+ compare\_num2[count] +

num3; if(result[count] >= 2)

{

num3 = 1;

}

else

{

num3 = 0;

}

result[count] = result[count] % 2;

}

for(count = 4; count >= 0; count--)

{

compare\_num2[count] = result[count];

}

if(num1 < 0)

{

num3 = 0;

for(count = 4; count >= 0; count--)

{

result[count] = 0;

}

for(count = 0; count < 5; count++)

{

result[count] = comparison[count] + compare\_num1[count] +

num3;

if(result[count] >= 2)

{

num3 = 1;

}

else

{

num3 = 0;

}

result[count] = result[count] % 2;

}

for(count = 4; count >= 0; count--)

{

first\_number[count] = result[count]; anumcp[count] = result[count];

}

}

if(num2 < 0)

{

for(count = 0; count < 5; count++)

{

temp = second\_number[count]; second\_number[count] = compare\_num2[count]; compare\_num2[count] = temp;

}

}

}

void sum(int num[])

{

int count; num3 = 0;

for(count = 0; count < 5; count++)

{

result[count] = product[count] + num[count] + num3; if(result[count] >= 2)

{

num3 = 1;

}

else

{

num3 = 0;

}

result[count] = result[count] % 2;

}

for(count = 4; count >= 0; count--)

{

product[count] = result[count];

printf("%d", product[count]);

}

printf(":");

for(count = 4; count >= 0; count--)

{

printf("%d", anumcp[count]);

}

}

void arithmetic\_shift()

{

int x = product[4], y = product[0], count; for(count = 1; count < 5 ; count++)

{

product[count - 1] = product[count];

}

product[4] = x;

for(count = 1; count < 5; count++)

{

anumcp[count - 1] = anumcp[count];

}

anumcp[4] = y; printf("\nArithmetic Shift"); for(count = 4; count >= 0; count--)

{

printf("%d", product[count]);

}

printf(":");

for(count = 4; count >= 0; count--)

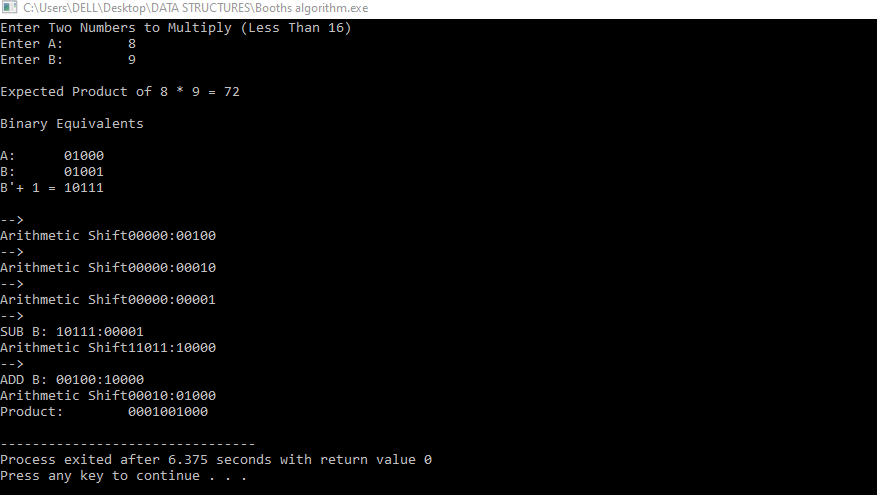
{

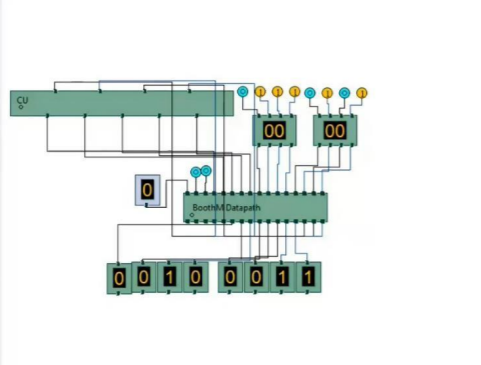
printf("%d", anumcp[count]);

}

}

**Result:**



****

**Experiment No. : 09**

**Name of the Experiment:** Implementation of Restoring Division Algorithm.

**Aim:** To implement Restoring Division Algorithm.

**Theory:**

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring, non-restoring, non- performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

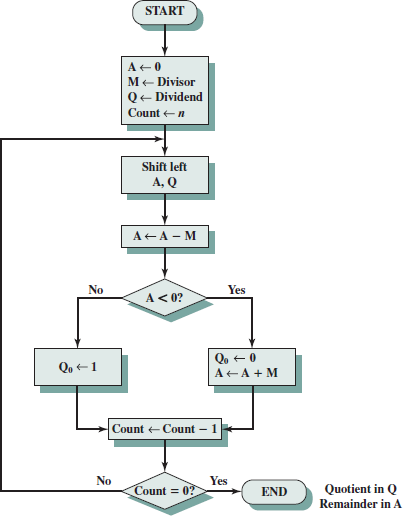
In this experiment, will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.

Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.

Let’s pick the step involved:

* **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
* **Step-2:** Then the content of register A and Q is shifted right as if they are a single unit
* **Step-3:** Then content of register M is subtracted from A and result is stored in A
* **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
* **Step-5:** The value of counter n is decremented
* **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
* **Step-7:** Finally, the register Q contain the quotient and A contain remainder.

**Flowchart:**



**Program:**

/\*Program for implementing 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]);

}

}

int main(){ int i;

printf("\*\*\*RESTORING DIVISION ALGORITHM\*\*\*");

printf("\n\nEnter two numbers to multiply(Both 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--){

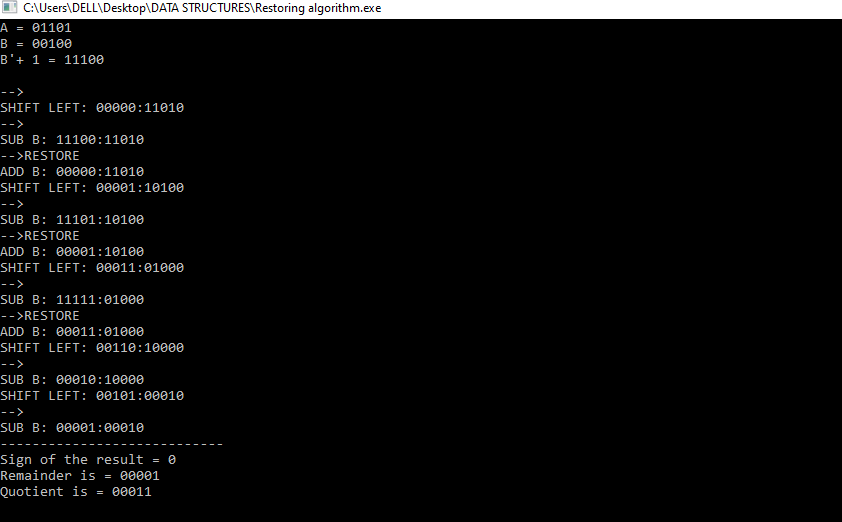
printf("%d",anumcp[i]);

}

getch(); return 0;

}

**Result:**



**Conclusion:** Thus, We successfully implemented Restoring Division Algorithm.

**Experiment No. : 10**

**Name of the Experiment:** Implementation of Non-Restoring Division Algorithm.

**Aim:** To implement Non-Restoring Division Algorithm.

**Theory:**

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring, non-restoring, non- performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

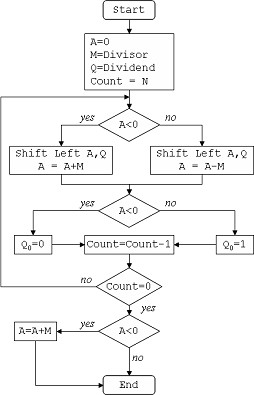
In this experiment, will be performing non restoring algorithm for unsigned integer. It is less complex than the restoring one because simpler operation are involved i.e. addition and subtraction, also now restoring step is performed. In the method, rely on the sign bit of the register which initially contain zero named as

A.

Let’s pick the step involved:

* + **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
  + **Step-2:** Check the sign bit of register A
  + **Step-3:** If it is 1 shift left content of AQ and perform A = A+M, otherwise shift left AQ and perform A = A-M (means add 2’s complement of M to A and store it to A)
  + **Step-4:** Again the sign bit of register A
  + **Step-5:** If sign bit is 1 Q[0] become 0 otherwise Q[0] become 1 (Q[0] means least significant bit of register Q)
  + **Step-6:** Decrements value of N by 1
  + **Step-7:** If N is not equal to zero go to **Step 2** otherwise go to next step
  + **Step-8:** If sign bit of A is 1 then perform A = A+M
  + **Step-9:** Register Q contain quotient and A contain remainder.

**Flowchart:**



**Program:**

/\*Program for implementing Non Restoring Division algorithm.\*/

#include<stdio.h> #include<malloc.h>

int \*a,\*q,\*m,\*mc,\*c,n,d; int powr(int x,int y)

{

int s=1,i; for(i=0;i<y;i++) s=s\*x;

return s;

}

void print(int arr[],int n)

{

int i; for(i=0;i<n;i++) printf("%d ",arr[i]);

}

void bin(int n, int arr[]){ int r, i = 0;

do{

r = n % 2; n /= 2; arr[i] = r; i++;

}while(n > 0);

}

void set(int array[], int x){ int i,tmp[20]={0};

for(i = x -1; i >=0; i--)

tmp[x-1-i]=array[i]; for(i=0;i<x;i++) array[i]=tmp[i];

}

int len(int x)

{

int i=0; while(powr(2,i)<=x) i++; return ++i;

}

void addBinary(int a1[], int a2[])

{

int bi[2]={0},ca[20]={0};

int t=len(n),tmp=0;

int \*su=(int\*)malloc(sizeof(int)\*len(n)); while(t-->0)

{

tmp=a1[t]+a2[t]+ca[t]; bin(tmp,bi);

su[t]=bi[0];

ca[t-1]=bi[1];

bi[0]=0;bi[1]=0;

}

for(t=0;t<len(n);t++) a1[t]=su[t];

free(su);

}

void twoCom(int arr[]){ int i;

int \*one=(int\*)malloc(sizeof(int)\*len(n));

for(i=0;i<len(n)-1;i++) one[i]=0;

one[i]=1;

for(i = 0; i < len(n); i++){

arr[i]=1-arr[i];

}

addBinary(arr, one); free(one);

}

void ls(int alen,int blen)

{

int i=0; for(i=0;i<alen-1;i++) a[i]=a[i+1];

a[i]=q[0]; for(i=0;i<blen-1;i++) q[i]=q[i+1];

q[i]=-1;

}

void printaq()

{

print(a,len(n));

printf("\t");

print(q,len(n)-1);

printf("\t");

printf("\n");

}

int main()

{

int i,cnt=0;

printf("Enter The Numerator/Denominator: "); scanf("%d/%d",&n,&d); q=(int\*)malloc(sizeof(int)\*len(n)-1);

bin (n,q); m=(int\*)malloc(sizeof(int)\*(len(n))); bin(d,m); a=(int\*)malloc(sizeof(int)\*(len(n))); for(i=0;i<len(n);i++)

a[i]=0;

mc=(int\*)malloc(sizeof(int)\*(len(n))); bin(d,mc);

set(q,len(n)-1);

set(m,len(n));

set(mc,len(n)); twoCom(mc); cnt=len(n)-1;

printf("\t A\t\t Q\t\t M\t Count\n"); printf("\t \t \t\t \n"); while(cnt>0)

{

printf("\t");

print(a,len(n));

printf("\t");

print(q,len(n)-1);

printf("\t");

print(m,len(n)); printf("\t%d\n",cnt); if(a[0]==1)

{

ls(len(n),len(n)-1); printf("LSHIFT\t"); printaq(); addBinary(a,m); printf("A=A+M\t"); printaq();

}

else

{

ls(len(n),len(n)-1); printf("LSHIFT\t"); printaq(); addBinary(a,mc); printf("A=A-M\t"); printaq();

} if(a[0]==1)

{

q[len(n)-2]=0; addBinary(a,m);

}

else

q[len(n)-2]=1; printf("A=A+M\t"); printaq();

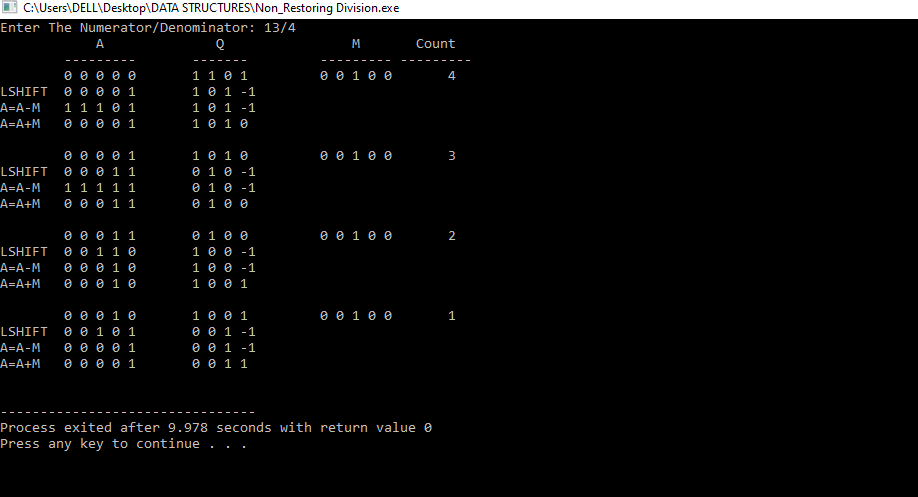
cnt-=1; printf("\n");

}

return 0;

}

**Result:**



**Conclusion:** Thus, We successfully implemented Non-Restoring Division Algorithm.