Chapter 2

1. DIGITAL LOGIC

A manufactured article's condition may change or be changed, and then the article may be classified either as Discrete-State or as a Continuous-State System.

System: Describes the article as a whole.

Continuous-State System: When the variable condition, i.e., the state of the system, is able to take any value between certain limits.

For example, the sound output from a radio receiver may be adjusted by the listener to any level from inaudible to the maximum output possible.

Similarly, a car driver can select any engine speed required by use of the accelerator control.

In contrast, to the wide range of states (conditions) possible in a Continuous-State System, a conventional electric light switch allows only on or off settings of the light; similarly, a car driver can use the gear lever to select one of the small number of gear rations available.

Discrete-State System: Only a finite, usually fixed, number of different states are allowed.

Both the lighting system with an on-off switch and the car gearbox are examples of Discrete-State System.

Two-State Systems

Discrete-State Systems which have only two possible states e.g., simple on-off light switch are Logic Systems, Logic Circuits or Logic networks which are synonymously referred to as Logic function or Logic gate.

Modern computers use digital circuitry with voltages all at one of two values called logic 0 and logic 1 representing Off/False/No state and On/True/Yes state respectively.

Logic gates are the most basic and the important component of any digital system including computers. It is a piece of hardware or an electronic circuit that can be used to implement the most basic logic expressions (referred t as the Boolean expression). Three basic logic functions are the OR-Gate, the AND-Gate and the NOT-Gate.

Commonly used Logic Gates

All digital devices can be boiled down to the three elementary Boolean functions commonly known as AND, OR and NOT.

These three elementary functions can be further combined to produce other functions: NAND (NOT of AND), NOR (NOT of OR) and the more complicated Ex-OR and Ex-NOR functions.

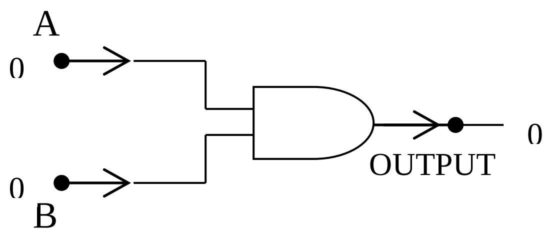
Truth Table

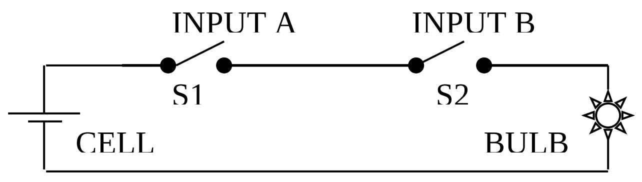
A truth table is a table representing the results of logical operations on all possible combinations of logical values. A truth table for a logical circuit shows the outputs for all possible combinations of inputs.

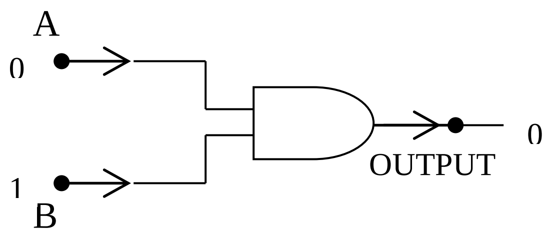
1. AND Gate

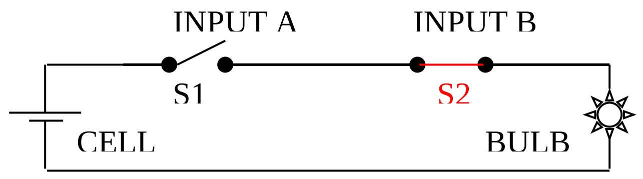
AND gate is a logic circuit having two or more than two inputs and one output. The output of an AND gate is logic "1" only when all of its inputs are in logic "1" state. In all other cases, the output is logic "0".

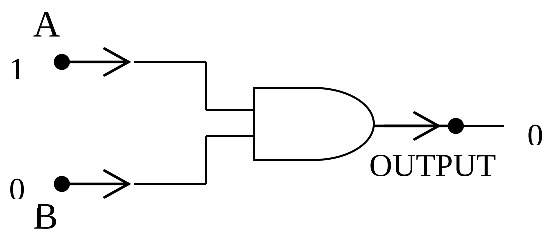
Logic Diagram **Electrical Diagram**

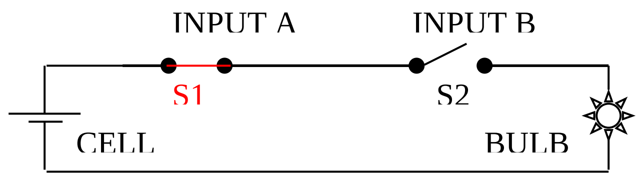


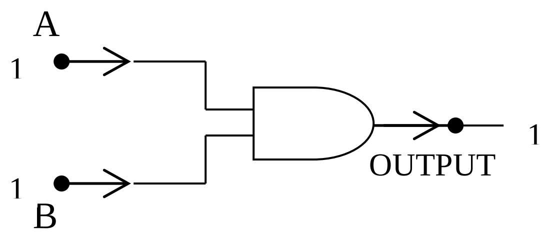


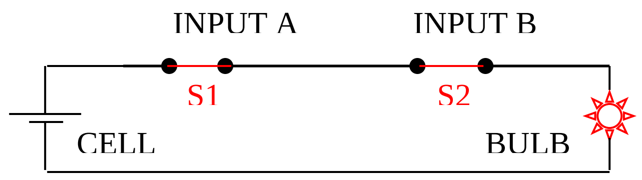












In above electrical analogue of an AND gate if one of the switches is ON or “1” the circuit remains open, hence the voltage in the output is “0” (i.e., bulb does not glow). Both the switches must be closed in order to give an output voltage (i.e., to glow the bulb).

Truth Table for AND Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

Input A	**Input B**	**Output A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

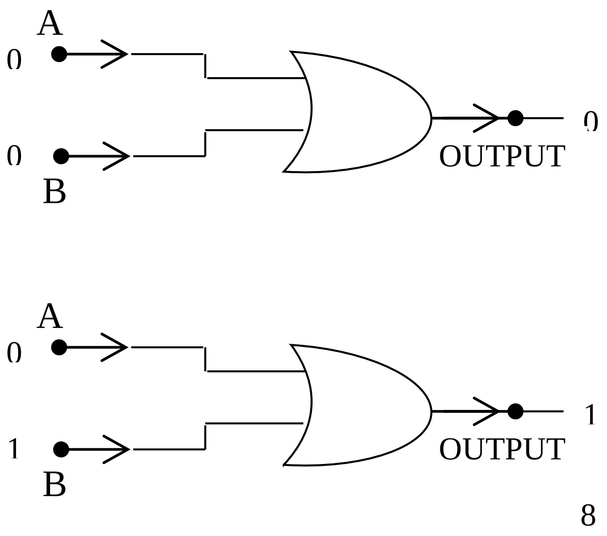
Switch S1	**Switch S2**	**Bulb Glow**
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

Operation Symbol for AND gate is represented by a dot “.”, and Boolean Expression for AND function is f = A•B.

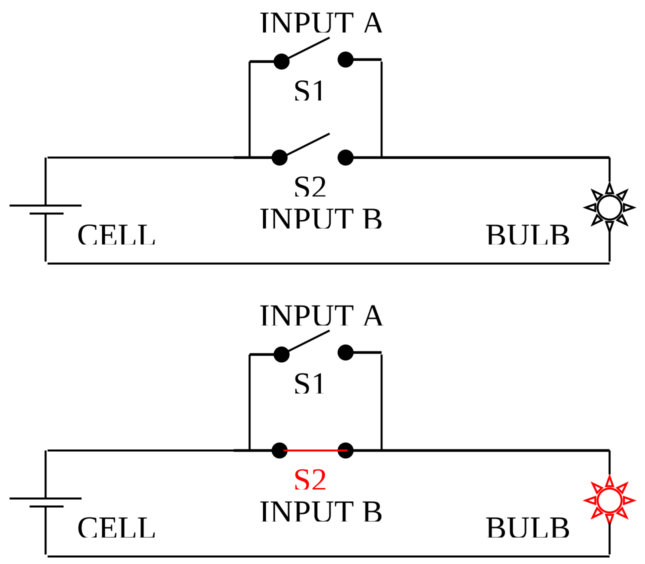
2. OR Gate

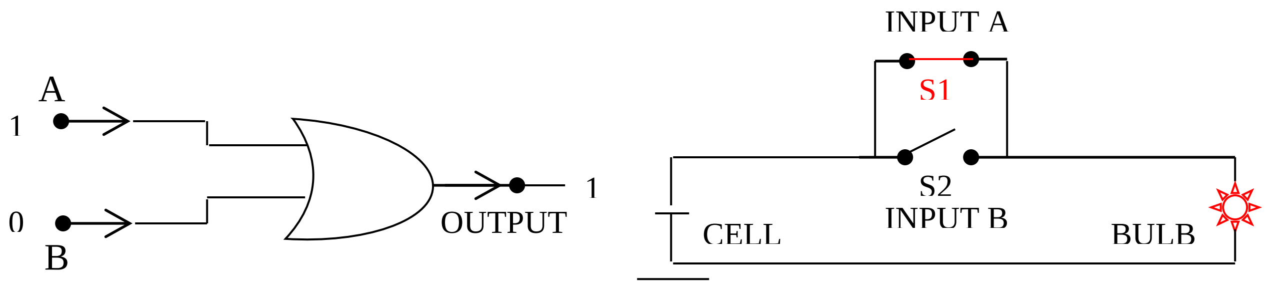
The OR gate is a logic circuit with two or more than two inputs and one output. The output of an OR gate is logic “1” when any one of its inputs are at logic “1” state. Only when both the inputs are at logic “0” the output of an OR gate is logic “0”.

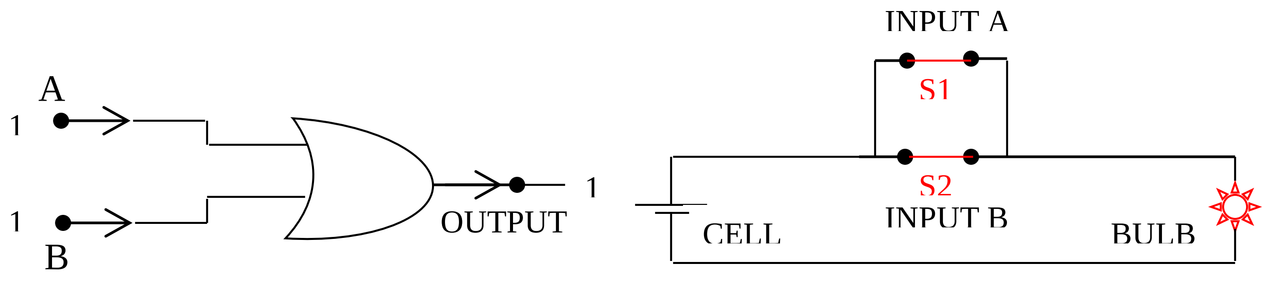
Logic Diagram



Electrical Diagram







In above electrical analogue of an OR gate if one of the switches or both the switches are closed or ON or "1" the circuit remains closed, hence the voltage in the output is "1" (i.e., bulb glows). In the case both the switches are open the output voltage is "0" (i.e., bulb does not glow).

Truth Table for OR Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

Input A	**Input B**	**Output A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

Switch S1	**Switch S2**	**Bulb Glow**
OFF	OFF	OFF
OFF	ON	ON
ON	OFF	ON
ON	ON	ON

Operation Symbol for OR gate is represented by a dot "+", and Boolean Expression for OR function is f = A + B.

3. NOT Gate (or Inverter)

NOT gate is a logic circuit with one input one output logic gate whose output is always the complement of the input. E.g., logic "0" at the input produces logic "1" at the output and vice versa. It is also known as a Complementing Circuit or an Inverting Circuit or a Negating Circuit.

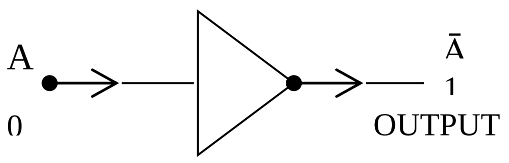
If: $A=0$

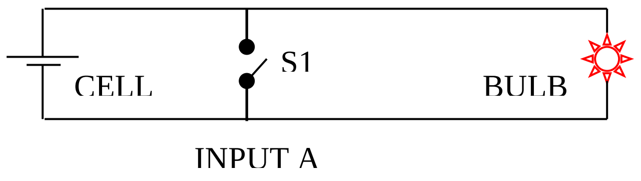
Then: $\bar{A}=1$

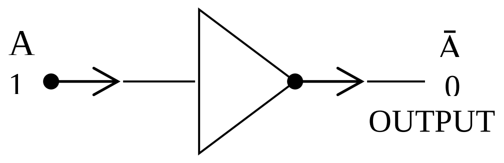
If: $A=1$

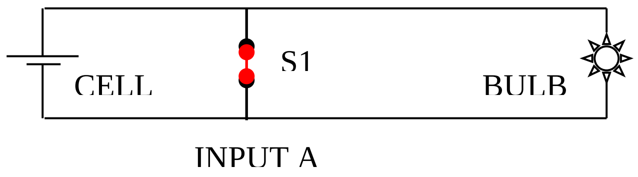
Then: $\bar{A}=0$

Logic Diagram **Electrical Diagram**









Truth Table for NOT Gate (with two inputs)

No. of inputs = 1, hence input combinations = $2^1 = 2$

Input A	Output NOT A
0	1
1	0

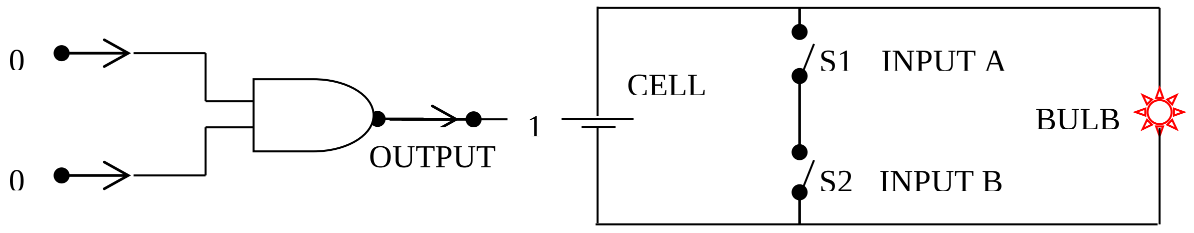
Switch S1	Bulb Glow
OFF	ON
ON	OFF

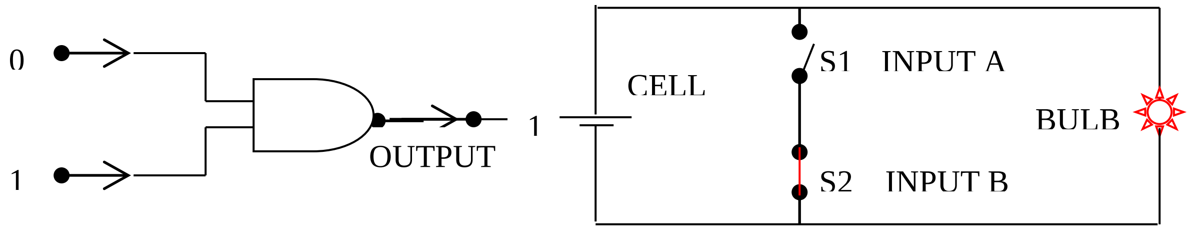
Operation Symbol for NOT gate is represented by a bar " ¯ " , and Boolean Expression for NOT function is $f = \bar{A}$.

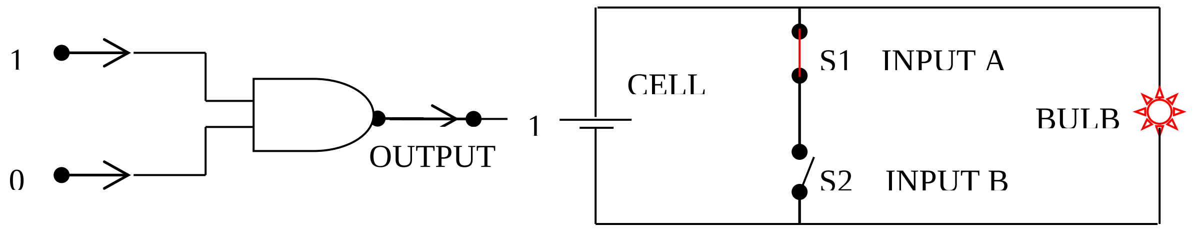
4. NAND Gate

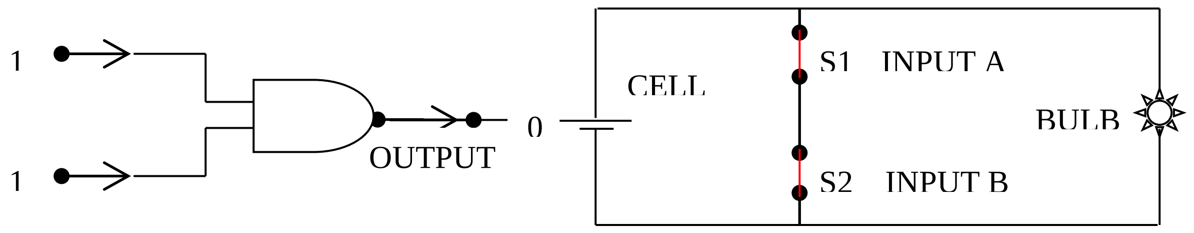
NAND gate is obtained by complementing the output of an AND gate. It stands for NOT – AND. The truth table of NAND gate is obtained from the truth table of an AND gate by complementing the output entries. The output of a NAND gate is logic "0" when all its inputs are logic "1". For all other cases the output is logic "1".

Logic Diagram **Electrical Diagram**

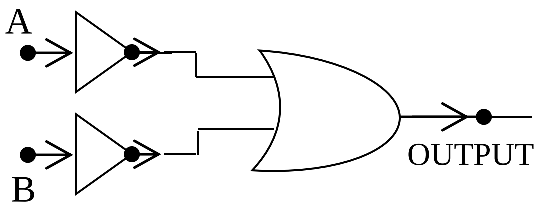








The NAND logic diagram can also be constructed with the combination of NOT and OR logic circuits as below.



Truth Table for NAND Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

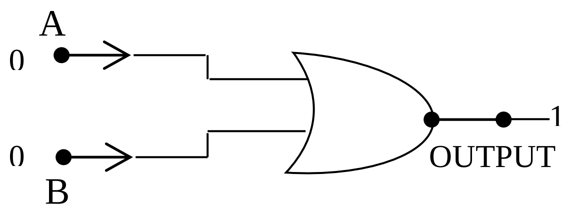
Input A	**Input B**	**Output A NAND B**	**Switch S1**	**Switch S2**	**Bulb Glow**
0	0	1	OFF	OFF	ON
0	1	1	OFF	ON	ON
1	0	1	ON	OFF	ON
1	1	0	ON	ON	OFF

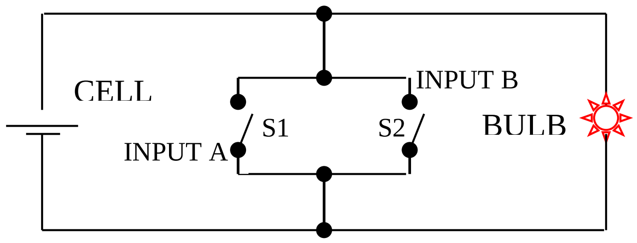
Boolean Expression for NAND function is f = $\overline{A \bullet B}$.

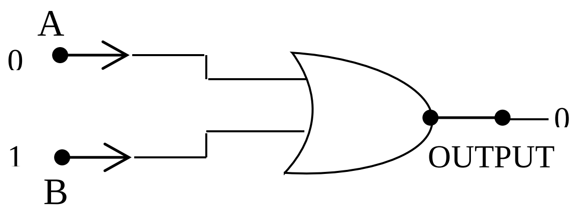
5. NOR Gate

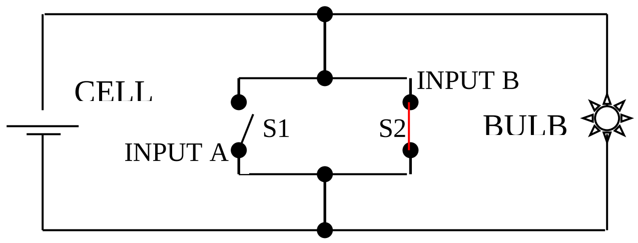
NOR gate is obtained by complementing the output of an OR gate. It stands for NOT – OR. The truth table of NOR gate is obtained from the truth table of an OR gate by complementing the output entries. The output of a NOR gate is logic “1” when all its inputs are logic “0”. For all other cases the output is logic “0”.

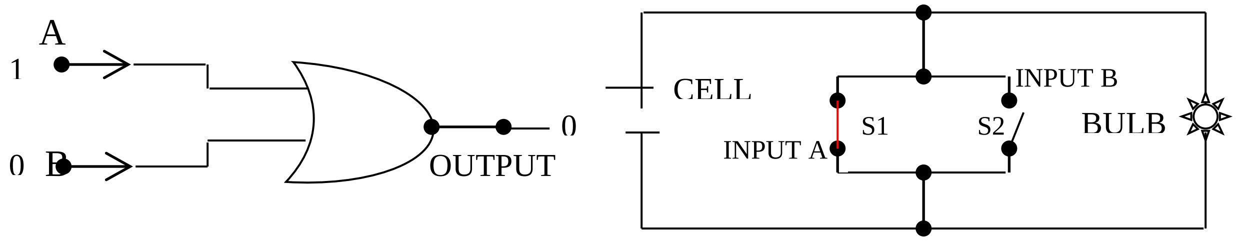
Logic Diagram **Electrical Diagram**

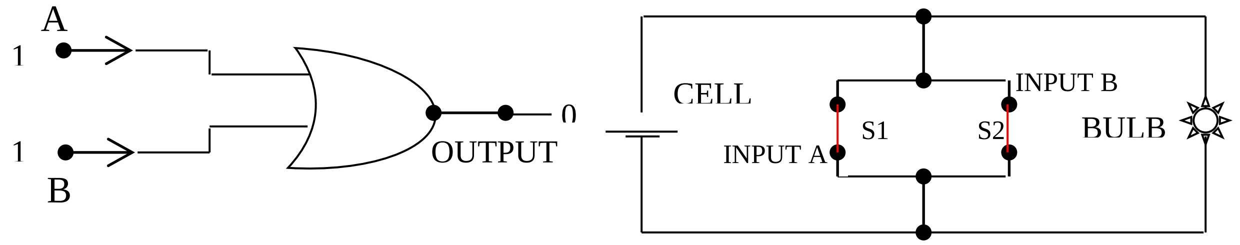




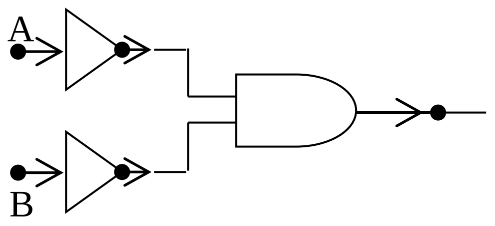








The NOR logic diagram can also be constructed with the combination of NOT and AND logic circuits as below.



Truth Table for NOR Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

Input A	**Input B**	**Output A NOR B**
0	0	1
0	1	0
1	0	0
1	1	0

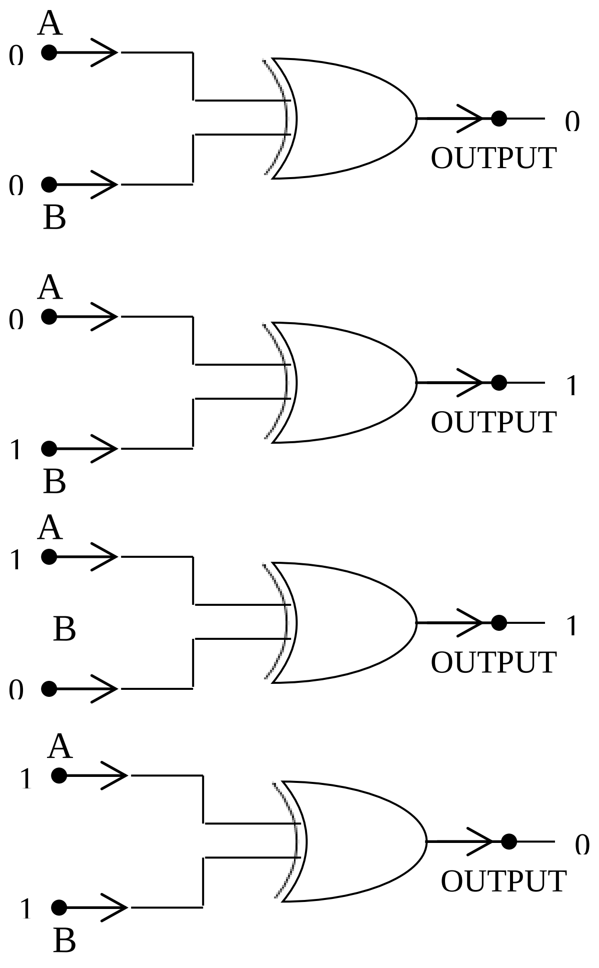
Switch S1	**Switch S2**	**Bulb Glow**
OFF	OFF	ON
OFF	ON	OFF
ON	OFF	OFF
ON	ON	OFF

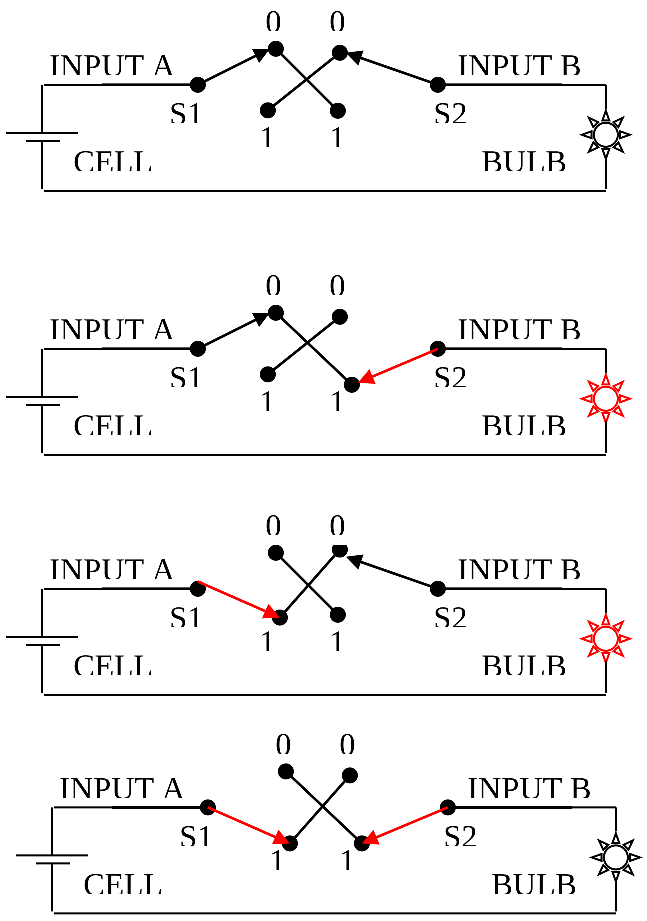
Boolean Expression for NOR function is f = $\overline{A+B}$.

6. Exclusive OR Gate (XOR)

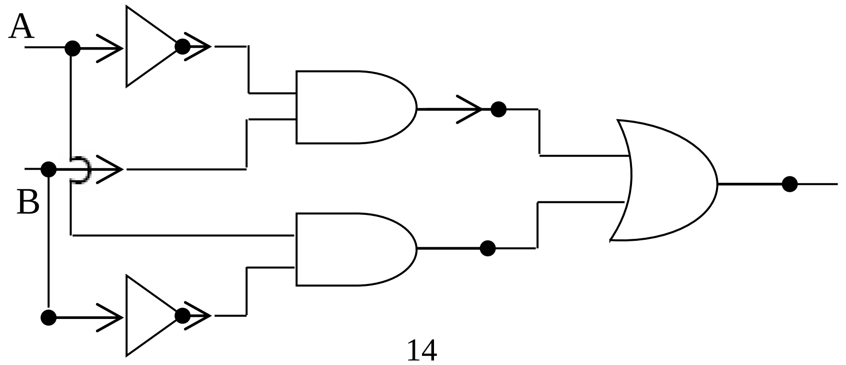
Exclusive OR gate is configured with two or more inputs. If we compare the truth table for a 2 input OR and a 2 input Exclusive OR there is only one difference. The output of an Exclusive OR gate when both its inputs are logic “1”, is logic “0” instead of logic “1” as in the case of an OR gate.

Logic Diagram **Electrical Diagram**





The XOR logic diagram can also be constructed with the combination of NOT, AND and OR logic circuits as below.



Truth Table for XOR Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

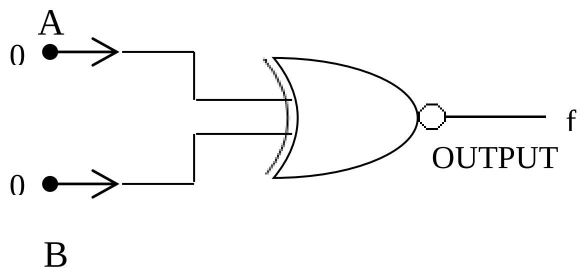
Input A	Input B	Output A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Switch S1	Switch S2	Bulb Glow
OFF	OFF	OFF
OFF	ON	ON
ON	OFF	ON
ON	ON	OFF

Operation Symbol for XOR gate is represented by an encircled plus $\oplus$, and Boolean Expression for XOR function is $f = A \oplus B$.

7. XNOR Gate

XNOR is obtained by complementing the output of XOR gate. It stands for NOT – XOR. The truth table of XNOR gate is obtained from the truth table of an XOR gate by complementing the output entries. The output of a XNOR gate is logic "1" when all its inputs are equal or logic "1" when the inputs are unequal.



Truth Table for XNOR Gate (with two inputs)

No. of inputs = 2, hence input combinations = $2^2 = 4$

Input A	Input B	Output A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Switch S1	Switch S2	Bulb Glow
OFF	OFF	ON
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

Boolean Expression for XNOR function is $f = \overline{A \oplus B}$.

8. Boolean Algebra

British mathematician and logician George Boole developed Boolean algebra in 1854. In Boolean algebra, logical propositions are denoted by symbols and abstract mathematical operators that correspond to the laws of logic can be applied. Boolean algebra is of major importance in the study of pure mathematics and in the design of modern computers.

In simple terms Boolean algebra is a system of mathematical logic which differs from both the conventional algebra and the binary arithmetic. For example, In Boolean algebra

$1 + 1 = 1$

Where + is the logical operation OR and not the simple addition.

In Boolean algebra the variables permitted to have two values "true" or "false" usually written as 1 and 0 respectively. The Boolean operations on the variables are limited to three basic logic operations AND, OR and NOT.

Two states of Boolean variables i.e., true and false may be represented by ON and OFF states of electronic switching circuits respectively. Hence, Boolean algebra is of practical significance in implementing electronic logic circuits used in digital equipments.

In the world of Boolean algebra, there are only two possible values for any quantity and for any arithmetic operation: 1 or 0. There is no such thing as "2" within the scope of Boolean values.

9. Boolean algebraic identities

In mathematics, an identity is a statement true for all possible values of its variable or variables. The algebraic identity of $x + 0 = x$ tells us that anything (x) added to zero equals the original "anything," no matter what value that "anything" (x) may be. Like ordinary algebra, Boolean algebra has its own unique identities based on the bivalent states of Boolean variables.

The first Boolean identity is that the sum of anything and zero is the same as the original "anything." This identity is no different from its real-number algebraic equivalent:

$A + 0 = A$

No matter what the value of A, the output will always be the same: when A=1, the output will also be 1; when A=0, the output will also be 0.

The next identity is most definitely *different* from any seen in normal algebra. Here we discover that the sum of anything and one is one:

$A + 1 = 1$

No matter what the value of A, the sum of A and 1 will always be 1. In a sense, the "1" signal *overrides* the effect of A on the logic circuit, leaving the output fixed at a logic level of 1.

Next, we examine the effect of adding A and A together, which is the same as connecting both inputs of an OR gate to each other and activating them with the same signal:

$$A + A = A$$

In real-number algebra, the sum of two identical variables is twice the original variable's value ($x + x = 2x$), but remember that there is no concept of "2" in the world of Boolean math, only 1 and 0, so we cannot say that $A + A = 2A$. Thus, when we add a Boolean quantity to itself, the sum is equal to the original quantity: $0+0 = 0$, and $1+1= 1$.

Introducing the uniquely Boolean concept of complementation into an additive identity, we find an interesting effect. Since there must be one "1" value between any variable and its complement, and since the sum of any Boolean quantity and 1 is 1, the sum of a

$$A + \bar{A} = 1$$

Just as there are four Boolean additive identities ($A + 0$, $A + 1$, $A + A$, and $A + \bar{A}$), so there are also four multiplicative identities: Ax0, Ax1, AxA, and AxA'. Of these, the first two are no different from their equivalent expressions in regular algebra:

$$0\,A = 0$$
$$1\,A = A$$

The third multiplicative identity expresses the result of a Boolean quantity multiplied by itself. In normal algebra, the product of a variable and itself is the *square* of that variable ($3 \times 3 = 3^2 = 9$). However, the concept of "square" implies a quantity of 2, which has no meaning in Boolean algebra, so we cannot say that $A \times A = A^2$. Instead, we find that the product of a Boolean quantity and itself is the original quantity, since $0 \times 0 = 0$ and $1 \times 1 = 1$:

$$A\,A = A$$

The fourth multiplicative identity has no equivalent in regular algebra because it uses the complement of a variable, a concept unique to Boolean mathematics. Since there must be one "0" value between any variable and its complement, and since the product of any Boolean quantity and 0 is 0, the product of a variable and its complement must be 0:

$$A\,\bar{A} = 0$$

To summarize, then, we have four basic Boolean identities for addition and four for multiplication:

Basic Boolean algebraic identities

Additive	Multiplicative
$A + 0 = A$	$0A = 0$
$A + 1 = 1$	$1A = A$
$A + A = A$	$AA = A$
$A + \overline{A} = 1$	$A\overline{A} = 0$

Another identity having to do with complementation is that of the *double complement*: a variable inverted twice. Complementing a variable twice (or any even number of times) results in the original Boolean value. This is analogous to negating (multiplying by -1) in real-number algebra: an even number of negations cancel to leave the original value:

$A = A$

10. Boolean Laws

10.1 AND Laws

The Boolean expression with two inputs for AND function is: f = A AND B = A•B which implies that f is True only if A and B both are true.

10.2 OR Laws

The Boolean expression with two inputs for OR function is: f = A OR B = A + B which implies that f is True if any one of A or B or both are true.

10.3 NOT Laws

The Boolean expression with one input for NOT function is: f = NOT A = $\bar{A}$ which implies that f is True only if A is false.

10.4 Commutative Laws

The commutative laws tell us we can reverse the order of variables that are either added together or multiplied together without changing the truth of the expression. It is expressed as:

$A + B = B + A$	(Commutative property of addition)
$A•B = B•A$	(Commutative property of multiplication)

10.5 Associative Laws

The associative laws tell us we can associate groups of added or multiplied variables together with parentheses without altering the truth of the equations. It is expressed as:

$A + (B + C) = (A + B) + C$ (Associative property of addition)

$A (BC) = (AB) C$ (Associative property of multiplication)

10.6 Distributive Laws

The distributive property illustrates how to expand a Boolean expression formed by the product of a sum, and in reverse shows us how terms may be factored out of Boolean sums-of-products. It is expressed as:

$A (B + C) = AB + AC$

10.7 Idempotent Laws

The effect of adding A and A together, which is the same as connecting both inputs of an OR gate to each other and activating them with the same signal:

$A + A = A$

In real-number algebra, the sum of two identical variables is twice the original variable's value ($x + x = 2x$), but remember that there is no concept of "2" in the world of Boolean math, only 1 and 0, so we cannot say that $A + A = 2A$. Thus, when we add a Boolean quantity to it, the sum is equal to the original quantity: $0 + 0 = 0$, and $1 + 1 = 1$.

It can be extended to any number of input i.e.,

$A + A + A + A \ \dots\dots\dots\dots\dots\dots + A = A$

The result of a Boolean quantity multiplied by it is itself. In normal algebra, the product of a variable and itself is the *square* of that variable ($3 \times 3 = 3^2 = 9$). However, the concept of "square" implies a quantity of 2, which has no meaning in Boolean algebra, so we cannot say that $A \times A = A^2$. Instead, we find that the product of a Boolean quantity and itself is the original quantity, since $0 \times 0 = 0$ and $1 \times 1 = 1$:

$AA = A$

It can be extended to any number of input i.e,

$A \cdot A \cdot A \cdot A \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots A = A$

10.8 Absorption Laws

The effect of Boolean quantity A multiplied with the addition of the same Boolean quantity A and another quantity B together, is the Boolean quantity A itself.

$A \cdot (A + B) = A$

We can prove it as follows:

L.H.S.	$= A \cdot (A + B)$	
	$= A \cdot A + A \cdot B$	(Distributive Law)
	$= A + A \cdot B$	(Idempotent Law)
	$= A (1 + B)$	(Distributive Law)
	$= A.\ 1$	$(A + 1 = 1$ then, $1 + B = 1)$
	$= A$	
	$=$ R.H.S.	

11. DeMorgan's Theorem

A mathematician named DeMorgan developed a pair of important rules regarding group complementation in Boolean algebra. By *group* complementation, we are referring to the complement of a group of terms, represented by a long bar over more than one variable.

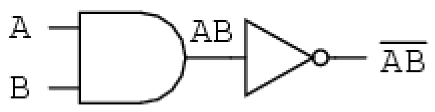
You should recall from the logic gates that inverting all inputs to a gate reverses that gate's essential function from AND to OR, or vice versa, and also inverts the output. So, an OR gate with all inputs inverted (a Negative-OR gate) behaves the same as a NAND gate, and an AND gate with all inputs inverted (a Negative-AND gate) behaves the same as a NOR gate. DeMorgan's theorems state the same equivalence in "backward" form: that inverting the output of any gate results in the same function as the opposite type of gate (AND vs. OR) with inverted inputs:

DeMorgan's first theorem states that the complement of a sum equals the product on the complements i.e.,

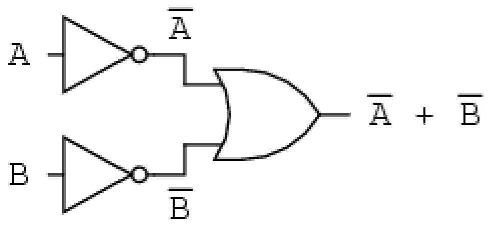
$\overline{A + B} = \overline{A} \cdot \overline{B}$

DeMorgan's second theorem states that, complements of a product equals sum of complements i.e.,

$\overline{A \cdot B} = \overline{A} + \overline{B}$



. . . is equivalent to . . .



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

A long bar extending over the term AB acts as a grouping symbol, and as such is entirely different from the product of A and B independently inverted. In other words, (AB)' is not equal to A'B'. Because the "prime" symbol (') cannot be stretched over two variables like a bar can, we are forced to use parentheses to make it apply to the whole term AB in the previous sentence. A bar, however, acts as its own grouping symbol when stretched over more than one variable. This has profound impact on how Boolean expressions are evaluated and reduced, as we shall see.

In general, if X1, X2, X3, X4 … … Xn are binary variables.

DeMorgan’s first theorem can be generalized as :

$$\overline{X1 + X2 + X3 + X4 \ldots \ldots + Xn} = \overline{X1} \,.\, \overline{X2} \,.\, \overline{X3} \,.\, \overline{X4} \ldots \ldots \overline{Xn}$$

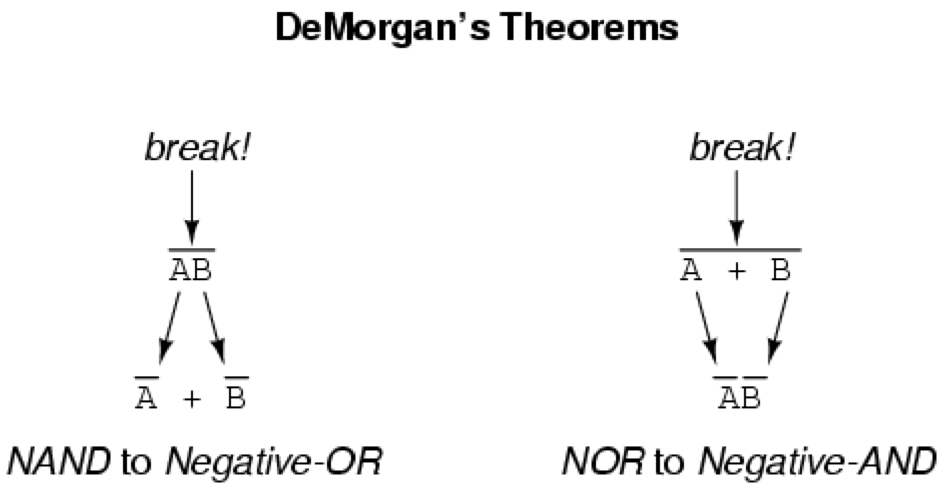
DeMorgan’s second theorem can be generalized as :

$$\overline{X1 \,.\, X2 \,.\, X3 \,.\, X4 \ldots \ldots \ldots \,.\, Xn} = \overline{X1} + \overline{X2} + \overline{X3} + \overline{X4} \ldots \ldots + \overline{Xn}$$

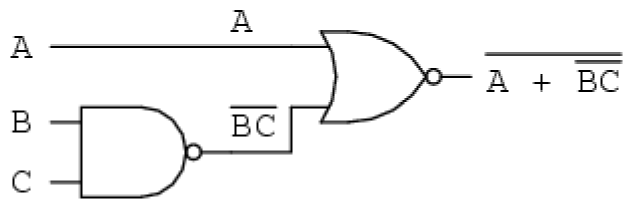
The following truth table proves the two DeMorgan’s theorems. The truth table is very simple to understand.

A	B	$\overline{A + B}$ NOR	$\overline{A . B}$ NAND	$\overline{A}$	$\overline{B}$	$\overline{A} . \overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1	1	1
0	1	0	1	1	0	0	1
1	0	0	1	0	1	0	1
1	1	0	0	0	0	0	0

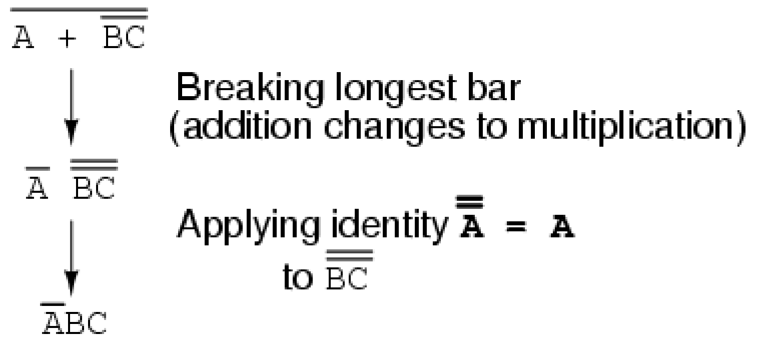
DeMorgan's theorem may be thought of in terms of *breaking* a long bar symbol. When a long bar is broken, the operation directly underneath the break changes from addition to multiplication, or vice versa, and the broken bar pieces remain over the individual variables. To illustrate:



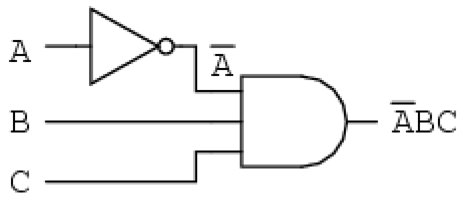
When multiple "layers" of bars exist in an expression, you may only break *one bar at a time*, and it is generally easier to begin simplification by breaking the longest (uppermost) bar first. To illustrate, let's take the expression (A + (BC)')' and reduce it using DeMorgan's Theorems:



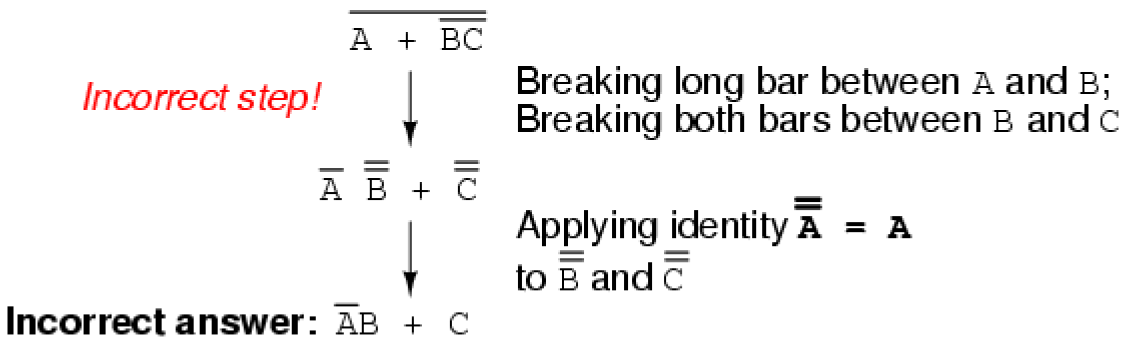
Following the advice of breaking the longest (uppermost) bar first, We will begin by breaking the bar covering the entire expression as a first step:


$\overline{A + \overline{BC}}$
Breaking longest bar
(addition changes to multiplication)
$\overline{A}\ \overline{\overline{BC}}$
Applying identity $\overline{\overline{A}} = A$
to $\overline{\overline{BC}}$
$\overline{A}BC$

As a result, the original circuit is reduced to a three-input AND gate with the A input inverted:

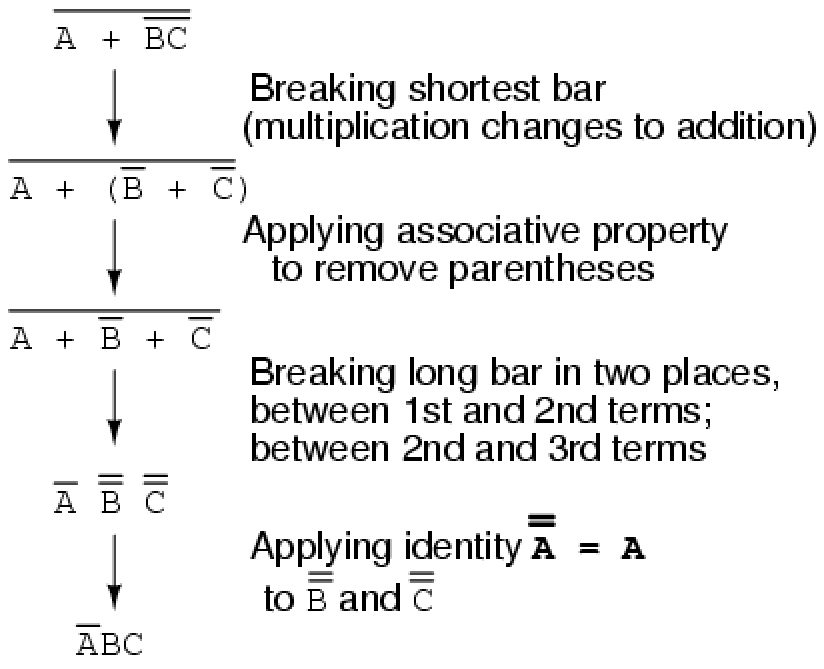


You should *never* break more than one bar in a single step, as illustrated here:


$\overline{A + \overline{BC}}$
Incorrect step!
Breaking long bar between A and B; Breaking both bars between B and C
$\overline{A}\,\overline{\overline{B}} + \overline{\overline{C}}$
Applying identity $\overline{\overline{A}} = A$ to $\overline{\overline{B}}$ and $\overline{\overline{C}}$
Incorrect answer: $\overline{A}B + C$

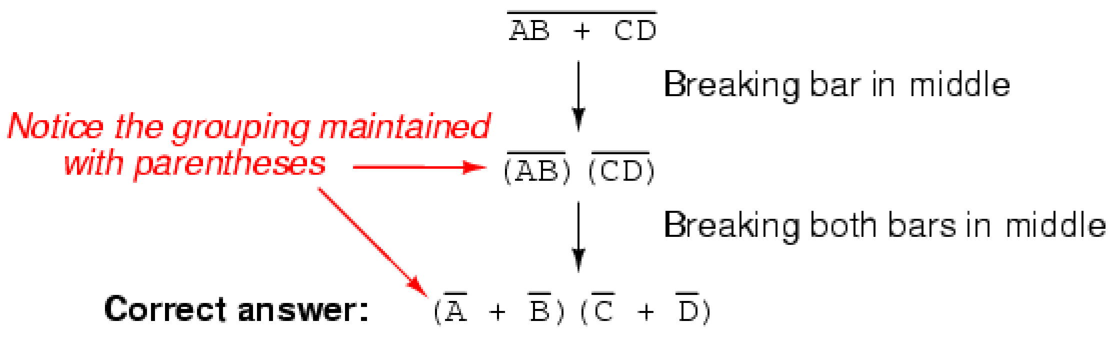
As tempting as it may be to conserve steps and break more than one bar at a time, it often leads to an incorrect result, so don't do it!

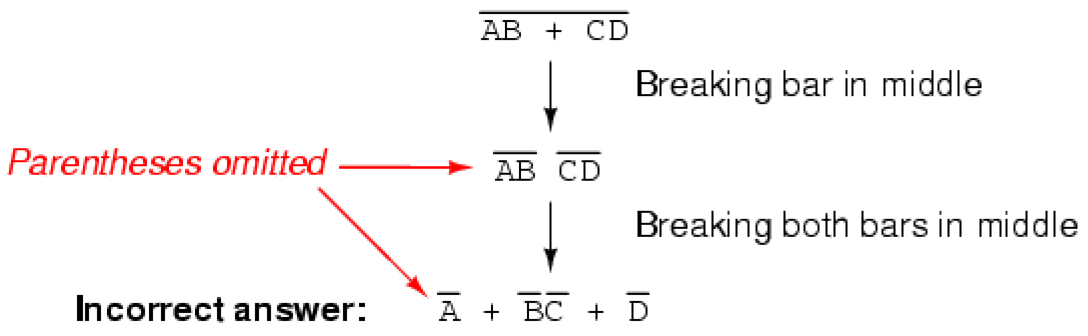
It is possible to properly reduce this expression by breaking the short bar first, rather than the long bar first:


$\overline{A + \overline{BC}}$
Breaking shortest bar (multiplication changes to addition)
$\overline{A + (\overline{B} + \overline{C})}$
Applying associative property to remove parentheses
$\overline{A + \overline{B} + \overline{C}}$
Breaking long bar in two places, between 1st and 2nd terms; between 2nd and 3rd terms
$\overline{A}\,\overline{\overline{B}}\,\overline{\overline{C}}$
Applying identity $\overline{\overline{A}} = A$ to $\overline{\overline{B}}$ and $\overline{\overline{C}}$
$\overline{A}BC$

The end result is the same, but more steps are required compared to using the first method, where the longest bar was broken first. Note how in the third step we broke the long bar in two places. This is a legitimate mathematical operation, and not the same as breaking two bars in one step! The prohibition against breaking more than one bar in one step is *not* a prohibition against breaking a bar in more than one place. Breaking in more than one *place* in a single step is okay; breaking more than one *bar* in a single step is not.

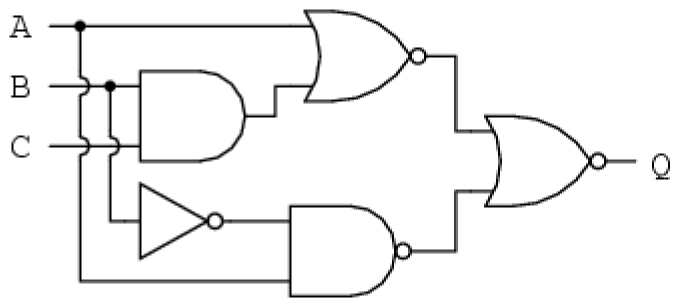
You might be wondering why parentheses were placed around the sub-expression B' + C', considering the fact that I just removed them in the next step. I did this to emphasize an important but easily neglected aspect of DeMorgan's theorem. Since a long bar functions as a grouping symbol, the variables formerly grouped by a broken bar must remain grouped lest proper precedence (order of operation) be lost. In this example, it really wouldn't matter if I forgot to put parentheses in after breaking the short bar, but in other cases it might. Consider this example, starting with a different expression:


$\overline{AB + CD}$
Breaking bar in middle
Notice the grouping maintained with parentheses
$\overline{(AB)}\ \overline{(CD)}$
Breaking both bars in middle
Correct answer:
$(\overline{A} + \overline{B})(\overline{C} + \overline{D})$

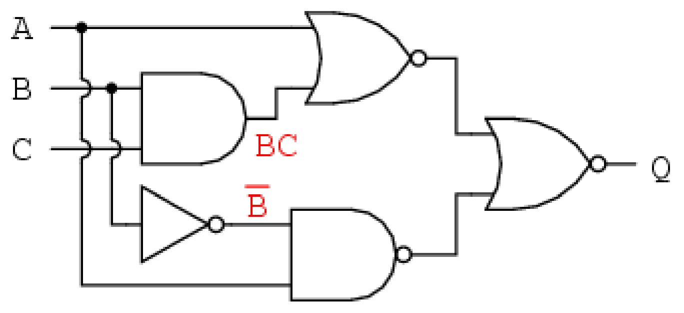

$\overline{AB + CD}$
Breaking bar in middle
Parentheses omitted
$\overline{AB}\ \overline{CD}$
Breaking both bars in middle
Incorrect answer:
$\overline{A} + \overline{B}\overline{C} + \overline{D}$

As you can see, maintaining the grouping implied by the complementation bars for this expression is crucial to obtaining the correct answer.

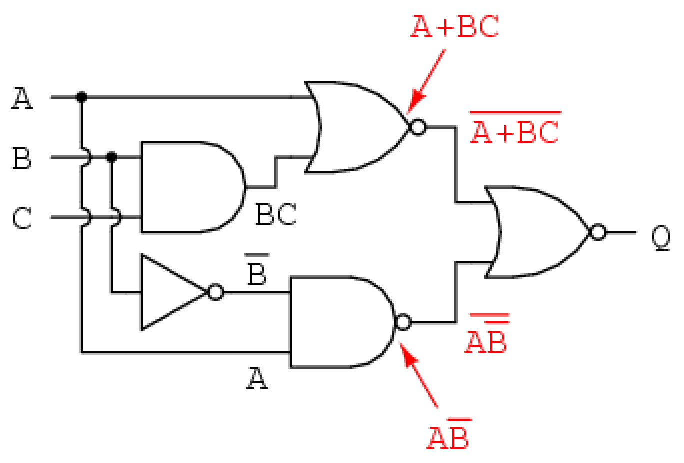
Let's apply the principles of DeMorgan's theorems to the simplification of a gate circuit:



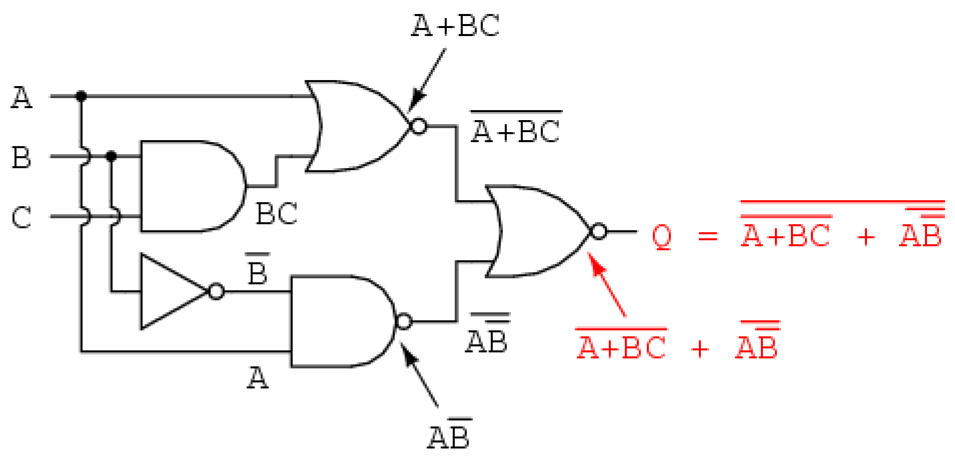
As always, our first step in simplifying this circuit must be to generate an equivalent Boolean expression. We can do this by placing a sub-expression label at the output of each gate, as the inputs become known. Here's the first step in this process:



Next, we can label the outputs of the first NOR gate and the NAND gate. When dealing with inverted-output gates, I find it easier to write an expression for the gate's output without the final inversion, with an arrow pointing to just before the inversion bubble. Then, at the wire leading out of the gate (after the bubble), I write the full, complemented expression. This helps ensure I don't forget a complementing bar in the sub-expression, by forcing myself to split the expression-writing task into two steps:



Finally, we write an expression (or pair of expressions) for the last NOR gate:


A+BC
A
B
C
$\overline{A+BC}$
BC
$\overline{B}$
A
$\overline{A\overline{B}}$
$A\overline{B}$
$Q = \overline{\overline{A+BC} + \overline{A\overline{B}}}$
$\overline{A+BC} + \overline{A\overline{B}}$

Now, we reduce this expression using the identities, properties, rules, and theorems (DeMorgan's) of Boolean algebra:

$$\overline{\overline{A + BC} + \overline{A\overline{B}}}$$

$\downarrow$ Breaking longest bar

$$(\overline{\overline{A + BC}})\,(\overline{\overline{A\overline{B}}})$$

$\downarrow$ Applying identity $\overline{\overline{A}} = A$ wherever double bars of equal length are found

$$(A + BC)(A\overline{B})$$

$\downarrow$ Distributive property

$$AA\overline{B} + BCA\overline{B}$$

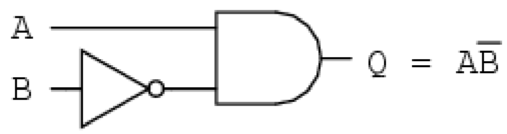
$\downarrow$ Applying identity $AA = A$ to left term; applying identity $A\overline{A} = 0$ to B and $\overline{B}$ in right term

$$A\overline{B} + 0$$

$\downarrow$ Applying identity $A + 0 = A$

$$A\overline{B}$$

The equivalent gate circuit for this much-simplified expression is as follows:



12. Karnaugh Maps

Karnaugh Maps are used for Logic Simplification. We saw that any logic function can be obtained directly from the truth table. The obvious method is to give the logic function in terms of each row in the truth table for which the output function f is a 1. Each row is the "product" of the input variables (A if the entry is a 1 and ~A if the entry is a 0).

Example of Logic Simplification:

```
 -----------
  AB | S  T
 -----------
  00  | 1  1       S = ~A*~B + ~A*B + A*~B     (5 gates)
  01  | 1  1         = ~A*(~B+B) + ~B*(~A+A)
  10  | 1  0         = ~A*1 + ~B*1
  11  | 0  1         = ~A + ~B                  (1 gate)
 -----------
```

In the above example, the direct method gives us

S = ~A*~B + ~A*B + A*~B

The above logic function is correct, but it requires 5 gates (AND,OR) to implement, not counting NOT gates. We can simplify it to S = ~A + ~B as shown above and it will take only 1 gate to implement.

Applying this simplification process to T, we have:

```
  T = ~A*~B + ~A*B + A*B          (5 gates)
    = ~A*(~B+B) + B*(~A+A)
    = ~A + B                       (1 gate)
```

The above simplification process can also be applied to functions of more inputs, but it will be tedious. There is better way to do it.

One technique developed by computer scientists and engineers to help the logic simplification process is to use something called Karnaugh Maps. Karnaugh Maps is a way to represent the entries of the truth table in a special way so that the simplification process is made easy.

For two variables, the Karnaugh Map (called K-Map for short) is drawn as shown below. Each cell in the K-Map represents a row in the truth table and is labelled by its corresponding binary code for the input variables.

```
  A \B   0     1
      +------+------+
  0   | 00   | 01   |          Top Row:       00+01 = 0X  Logic Function: ~A
      |      |      |          Bottom Row:  10+11 = 1X   Logic Function:  A
      +------+------+          Left Column:  00+10 = X0  Logic Function: ~B
      | 10   | 11   |          Right Column:01+11 = X1   Logic Function:  B
  1   |      |      |          ("X" to indicate variable that was "dropped")
      +------+------+
```

In the Karnaugh Map above, if we group together the two cells of the top row, namely the ones labeled 00 and 01, it is equivalent to saying

~A~B + ~AB = ~A*(~B+B) = ~A*1 = ~A.

Equivalently, we represent that as

(00+01) = 0X

Where we use the "X" to indicate that the variable B has "dropped off".
Similarly, combining any other two adjacent cells will give us:

```
  eg: 10+11 = 1X      or     A~B + AB  =  A
      00+10 = X0      or    ~A~B + A~B = ~B
      01+11 = X1      or    ~AB  + AB  =  B
```

To use K-Maps to aid in logic simplification, we first fill the K-Map: for each row with a 1 in the truth table, we label the corresponding cell in the K-Map with a "Y". Of course, we also label all the cells with a "N" to correspond to those rows in the truth table with a "0". We illustrate for the function T above.

```
   \    K-Map for function T
  A \B   0      1
     +------+------+
   0 |  00   |  01  |    To simplify,
     |  Y    |  Y   |     we can combine 00 and 01 --> 0X   (~A)
     +------+------+     we can combine 01 and 11 --> X1   ( B)
     |  10   |  11  |    So, the function is simplified to
   1 |  N    |  Y   |     T = ~A + B
     +------+------+
```

After filling up the K-map, we have completely described the function T (look at all the squares labelled with "Y"'s.)

To simplify, we now look for some adjacent cells with a 1 to combine.

Thus,
  we can combine 00 and 01 --> 0X   (~A)
  we can combine 01 and 11 --> X1   (B)

T = ~A + B.

The advantage of K-maps is that simplification of the logic circuits or functions can be done by looking for neighboring squares to combine.

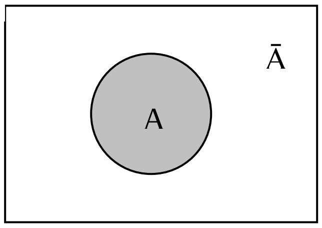
This is a lot easier than looking at the Boolean functions!!

13. Venn Diagram

This diagram is named after the British logician John Venn. It is a diagram representing a set or sets in mathematics and the logical relationships between them. The sets are drawn as circles. An area overlap between two circles (sets) contains elements that are common to both sets, and thus represents a third set. Circles that do not overlap represents sets with no elements in common (disjoint sets).

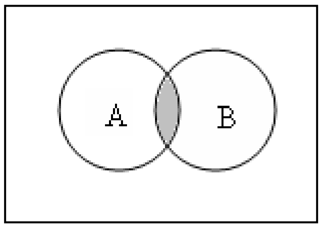
Venn diagrams are diagrams in which AREAS represent OPERATIONS or PROPOSITIONS. For example, the area within the rectangle represents the proposition under consideration. In these examples a letter within a circle refers to the whole circle.

a



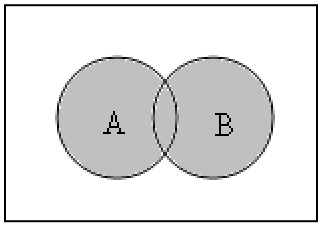
The Shaded area is A

b

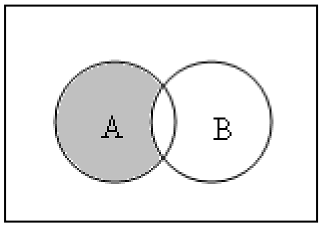


A ^ B Also written as A • B

c

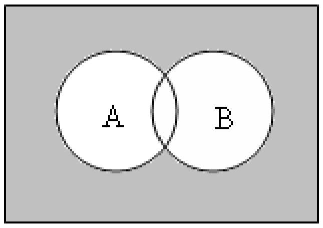


A v B Also written as A + B

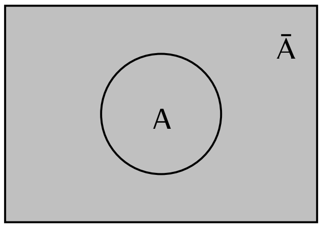


A ^ B Also written as A • B

e



A v B Also written as A + B



A • Ā Also written as A + Ā

If two operations have the same Venn diagram they are equivalent. This fact is used to show equivalence.

Exercises

1. What is Logic Gate? Explain giving example.
2. What is Truth table? Explain with examples.
3. Briefly explain about NOT, OR, AND, NOR, XOR and NAND operations.
4. What is Boolean algebra? Describe in brief how it differs from conventional algebra.
5. What is Karnaugh Map? For what purpose it is used? Describe with examples.
6. Describe DeMorgan’s Theorms with examples.
7. What is Venn diagram? Explain with examples.
8. Use Truth table to find which of the following relations are true.
    a. A . (A + B) = A + AB
    b. A = Ā . B = A . B
    c. A . (Ā + B) = A . B
    d. B . (A + 1) = B . (B + B)

Experiment no: 1

Familiarization with AND, OR and INVENTOR Gates.

OBJECTIVES:

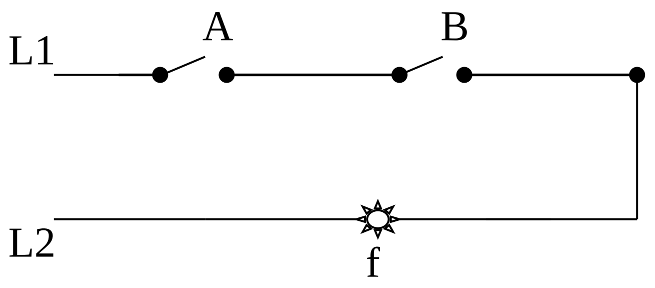
1. Investigate the relationship between the inputs and outputs of AND gate.
2. Building a 4 input AND from three 2 input AND gates.
3. Investigate the relationship between the inputs and outputs of OR gate.
4. Building a 3 input OR gate from two 2 input OR gates
5. Examine the input and output of inverter.

OBJECTIVE NO: 1

Related Theory:

In our everyday processes, we often use logic. We draw logical conclusions from known facts. When using logic, a statement must be either true or false. There are no shades of opinion. Logic systems are therefore, essentially switching circuits. In the first electrical logic systems relay operated contacts were used. These switches routed electrical current through the circuit in the desired manner.

Take, for example, the logic function AND. This means that the conclusion (called f) is truly if both statement A and statement B are true. If either A or B is false, then f is false. To accomplish this electromechanically, a lamp is used to indicate f. True means that the lamp is lighted. False means the lamp is off. The circuit looks like this:



If we let 1= closed and 0= open for A and B; 1 = lighted and 0 = off for f, we can construct a truth table as follows:

A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

When using integrated circuits logic gates, we take a slightly different approach. The 1's and 0's represent a voltage (or no voltage) appearing a specific points of a circuit. Positive logic means that 5volts = 1 and 0 volts = 0. Negative logic means that 5 volts = 0 and 0 volts = 1. All of the circuits in this manual will define conditions in terms of positive logic.

EQUIPMENT REQUIRED:

Power supply, switch module, Led module, Quad 2 – input AND (7408) .

PROCEDURE:

Step 1. Connect the circuit of figure 1-1.

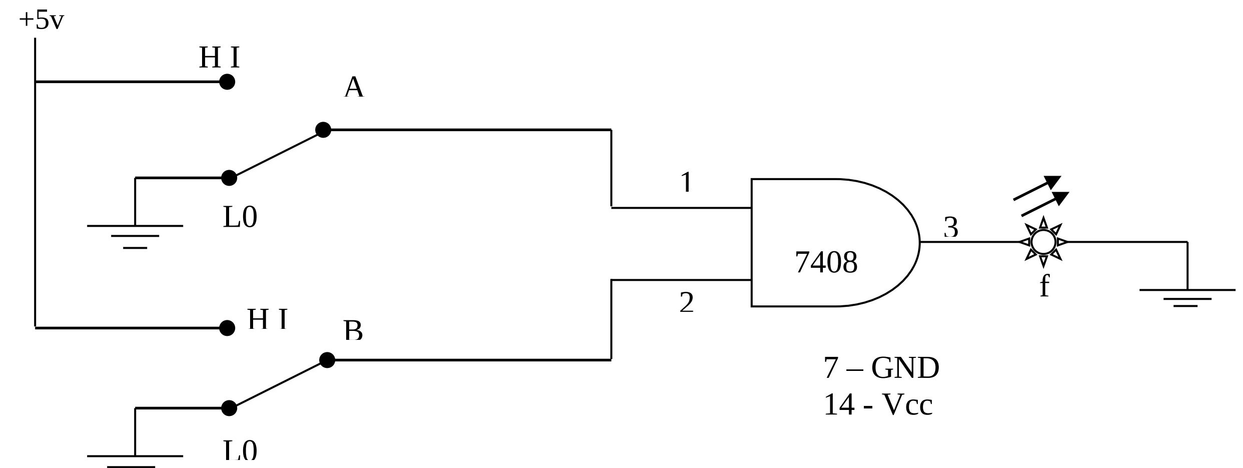


Fig. 1-1

Note that the output must have over 2.4 volts to be logic 1: less than 0.4 volts for logic 0.

Step 2. With Switch A in the LO position, put switch B in the LO position.
Step 3. Note the condition of the light emitting Diode f, and record in table 1.1

A	B	f

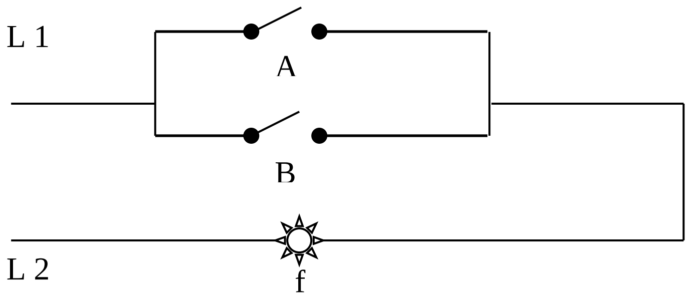
Table 1.1

Step 4. Move Switch B to the H1 position . Repeat Step 3.
Step 5. Move Switch A to the H1 position and put Switch B in the L0 position. Repeat Step 3.
Step 6. Repeat Step 4.

In your report, describe the operation of the AND gate in the words and compare the results you recorded in table 1-1 with expected results. How many H I – LO combinations are possible with two inputs?

OBJECTIVE NO: 3

RELATED THEORY:
Very often, when a task is to be performed, there is more than one way of obtaining the same result. Suppose you had a light f, which could be turned on from two switches in parallel.



Your logic statement would read: If switch A is closed or Switch B is closed, lamp f will be lighted. There will be a logic 1 at the output of the OR gate if there is a logic 1 at either input (or both inputs). There will be logic 0 at the output only when there is logic 0 at both inputs.

EQUIPMENT REQUIRED:
Power supply, Switch module, LED module, Quad 2 – input OR (7432).

PROCEDURE:
Step 1. Connect the circuit of figure 1–3

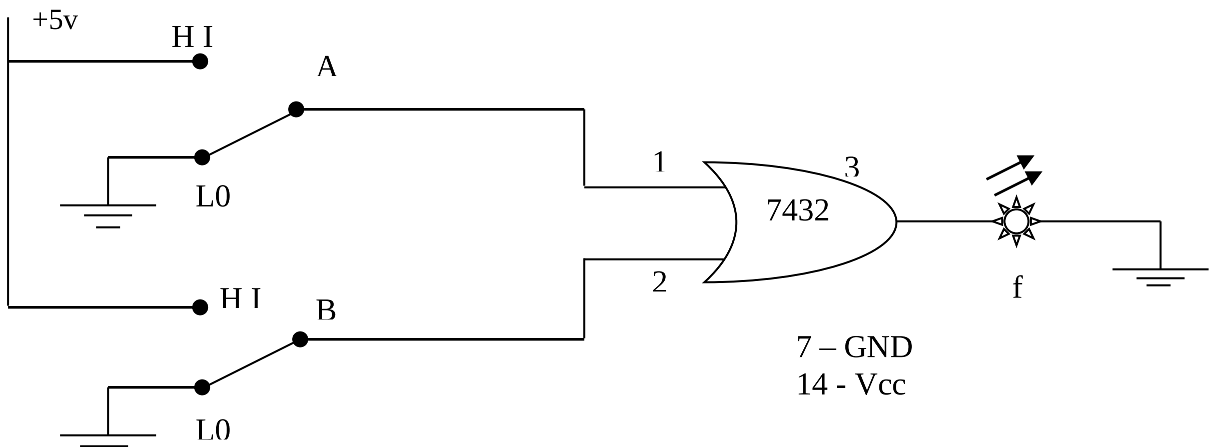


Figure 1- 2

Step 2. Observe the condition of LED for all possible combinations of switch A & B and note down on the table 1-2

A	B	fe	f

Table 1-2

In your report, describe the operation of the OR gate in words and compare the results you recorded in table 1-3 with the expected results (Fe). Assume you defined this system in terms of negative logic (H I = 0 and LO = 1; lighted = 0 and off = 1). Construct a negative logic truth table and compare it with Table 1-1 and draw the conclusion.

<u>OBJECTIVE NO: 5</u>

RELATED THEORY:

The logic function known as NOT is perhaps the simplest and, at the same time, most difficult of the logic concept.

The simple part is that NOT applies to two statements that cannot both be true at the same time. It "A" is true, then "X" must be false. What makes the NOT function difficult is the concept of complementary signals. To illustrate, let's take the complementary statement: "The switch is <u>not</u> closed". To indicate that this is a complementary statement, we put a bar or ' above the symbol" A. (This is read A – not). Now, if logic 1 means A, a logic 0 means A'.

An inverter is simply a gate whose output is the inversion of its input. If logic 1 is present at the input, logic 0 is present at the output. Also, if logic 0 is present at the input, logic 1 is present at the output.

EQUIPMENT REQUIRED:

Power supply, Switch module, LED module, Hex Inverter (7404).

PROCEDURE:

Step 1. Connect the circuit of figure 1–3

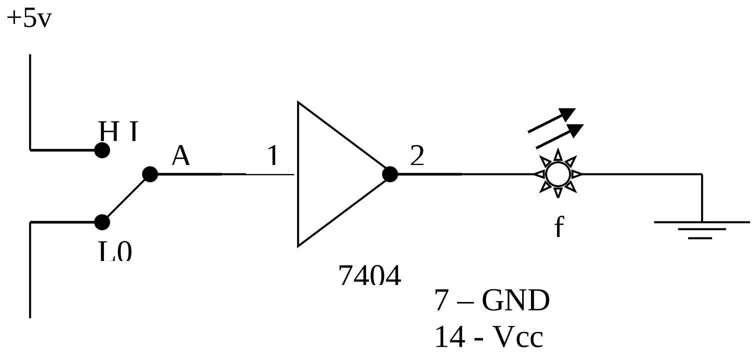


Figure 1-3

Step 2. Note the condition of the Light – Emitting Diode, f, for both H1 & L0 condition (1 = lighted, 0 = off) in Table 1-3.

Step 3. Add another inverter as shown in Figure 1–4 and note the output in table 1–4.

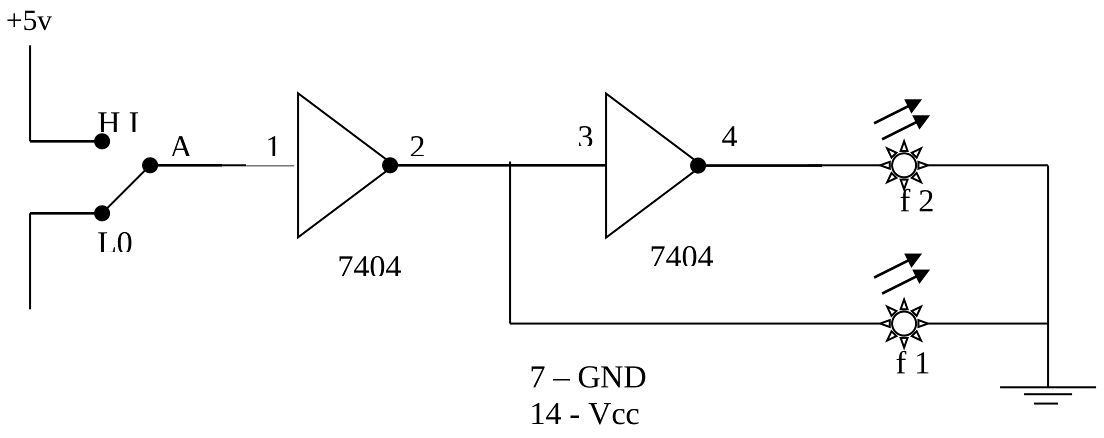


Figure 1-4

Step 4. Add a third inverter as shown in Figure 1–5 and note the output in table 1–5.

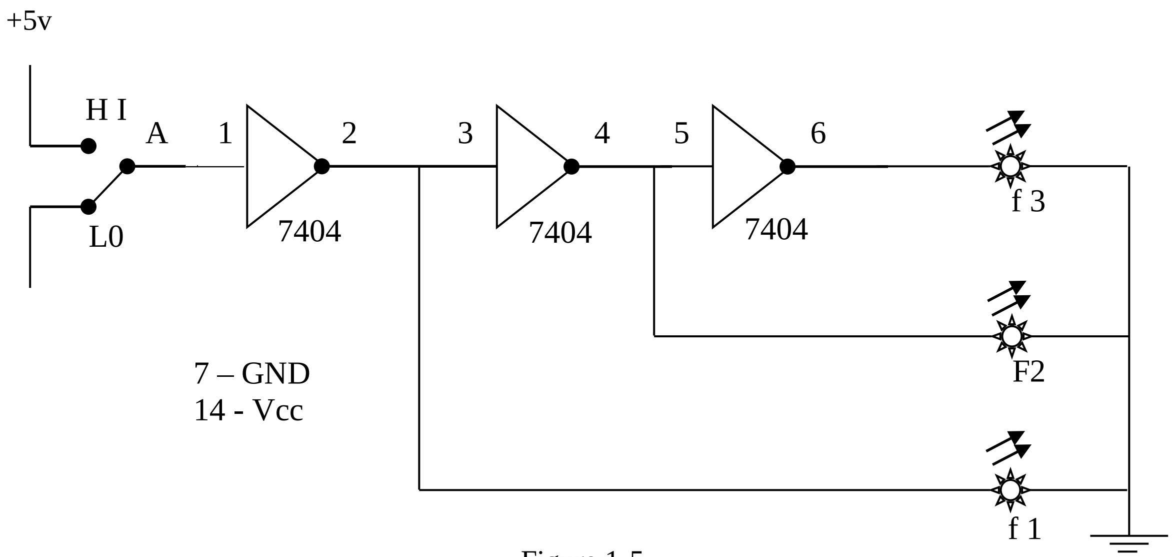


Figure 1-5

In your report, discuss the effect on a logic 1 passing through an odd number of inverters (1, 3, 5, etc.) and passing through an even number of inverters (2, 4, etc.)

A	f

Table 1-3

A	f1	f2

Table 1-4

A	f1	f2	f3

Table 1-5

Experiment no: 2

Verification of deMorgan’s Laws and familiarization with NAND and NOR gates.

OBJECTIVES:

1. Verify deMorgan’s first law.
2. Verify deMorgan’s second law.
3. Investigate the operation of NAND gate.
4. Investigate the operation of NOR gate.

OBJECTIVE NO: 1

Related Theory:

Thus far, we have examined the three basic logic gates, namely, AND, OR and the inverter. Because of the nature of logic (If a statement is not true, then it must be false). We can (in theory, at least) reduce the number of basic gates to two, one of which is the inverter.

For example, we saw that, in order for a logic 1 to be present at the output of an AND gate, logic 1’s had to be at all inputs. For an OR gate, a logic 0 was present at the output only when all inputs were at logic 0.

Now, let us invert the output of the AND. The new statement is: A logic 0 output occurs only when all inputs are logic 1. Similarly, let us invert the inputs of the OR. The new statement is: A logic 0 output occurs only when all inputs are logic 1.

EQUIPMENT REQUIRED:

Power supply, Switch module, LED module, Quad 2-input AND (7408), Quad 2-input OR (7432), Hex Inverter (7404).

PROCEDURE:

Step 1. Connect the circuit of figure 2–1.

Step 2. This is a 2 – input AND gate with an inverted output. Operate the switches (H1 = 1; L0 = 0) and complete the truth table of Table 2-1.

Step 3. Connect the circuit of Figure 2-2.

Step 4. This is a 2 – input OR gate with inverted inputs. Operate the switches (H1 = 1; L0 = 0) and complete the truth table of Table 2.2.

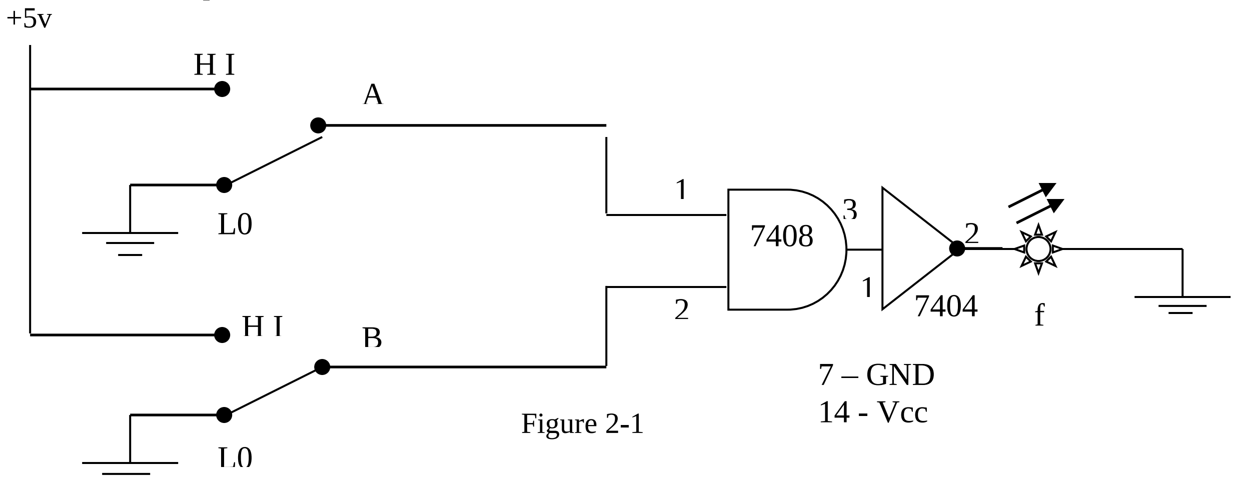


Figure 2-1

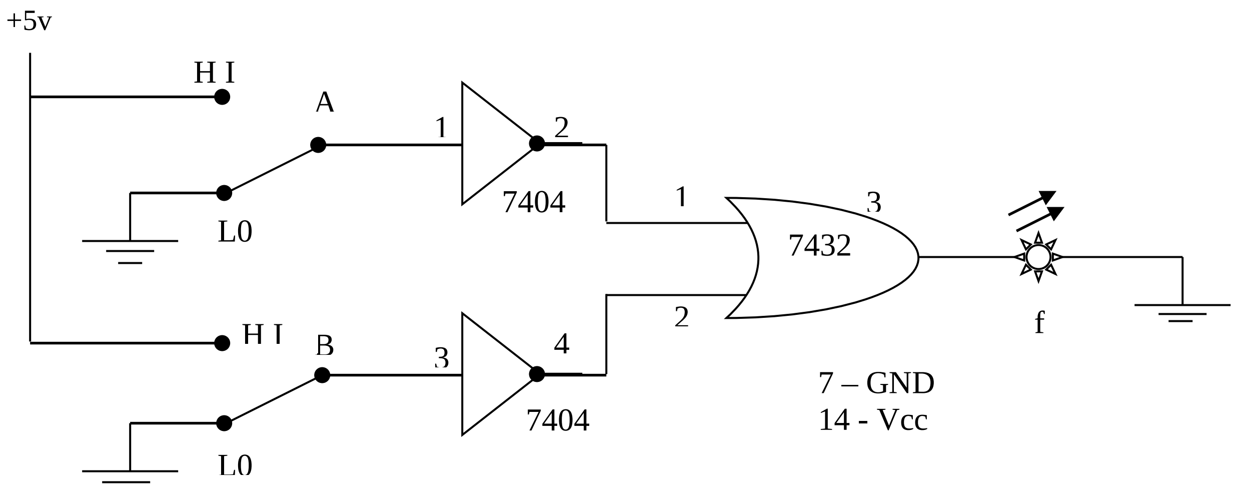


Figure 2-2

A	B	f

Table 2-1

A	B	f

Table 2-2

In your report, describe how you could make an AND gate using OR's and inverters.

OBJECTIVE NO: 2

Verify deMorgan's second law.

Related Theory:

The logic statement for the OR gate is: A logic 0 is present at the output only when all of the inputs are a logic 0.

The logic statement for the AND gate is: A logic 1 is present at the output only when all of the inputs are logic 1.

If we invert the output of the OR, we have: A logic 1 is present at the output only when all of the inputs are logic 0.
If we invert the output of the AND, we have: A logic 1 is present at the output only when all of the inputs are logic 0.

EQUIPMENT REQUIRED:
Power supply, Switch module, LED module, Quad 2-input AND (7408), Quad 2-input OR (7432), Hex Inverter (7404).

PROCEDURE:
Step 1. Connect the circuit of figure 2–3.
Step 2. Operate Switches A and B (H1 = 1; L0 = 0) and complete the truth table of Table 2-3 for the condition of the LED, f(1 = lighted; 0 = off).

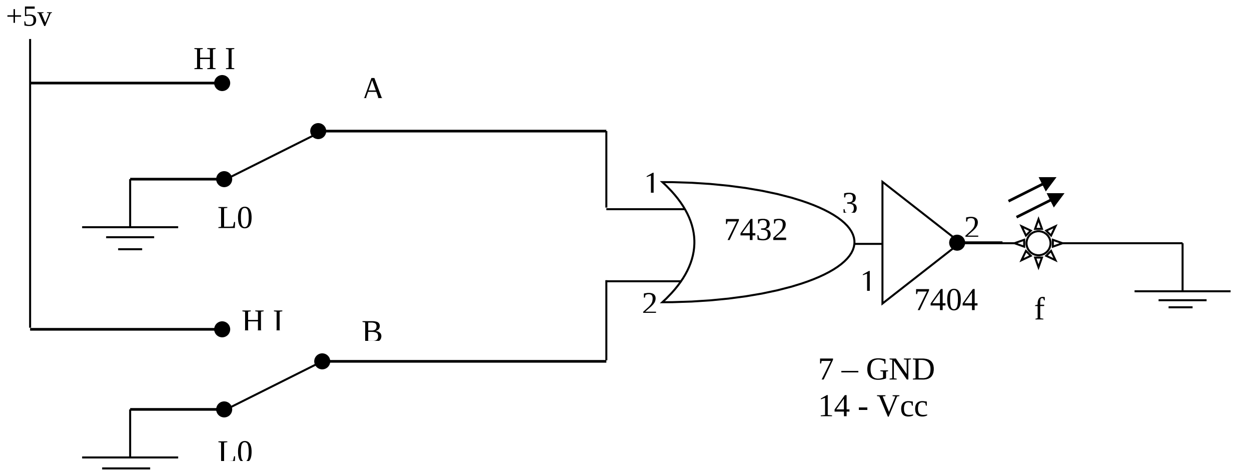


Figure 2-3

Step 3. Connect the circuit of figure 2–4.

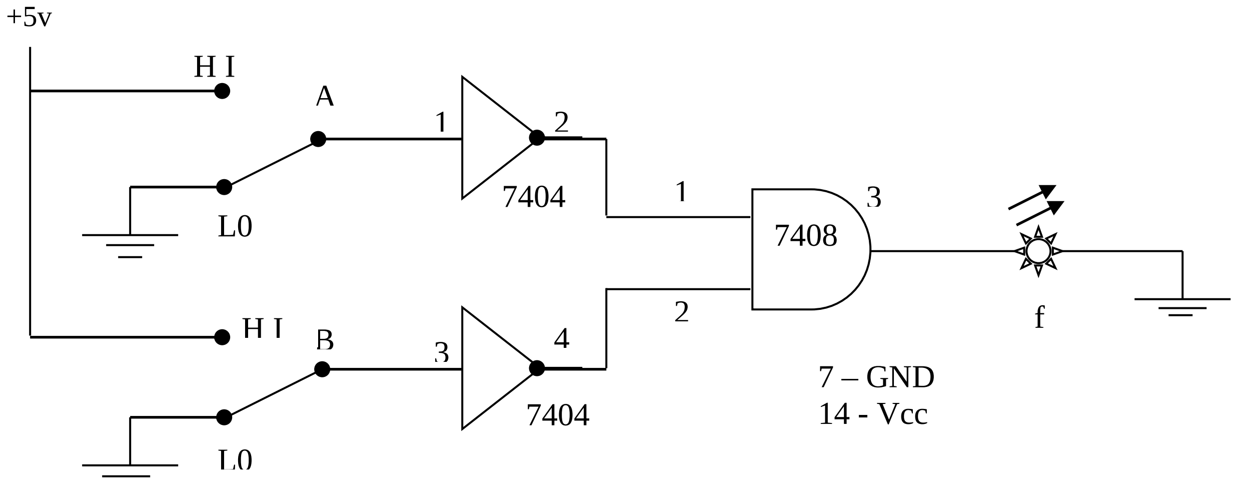


Figure 2-4

Step 4. Repeat Step for Table 2-4.

In your report, show how you could make an OR gate using AND gates and inverters.

A	B	f

Table 2-3

A	B	f

Table 2-4

<u>**OBJECTIVE NO: 3**</u>

Investigate the operation of NAND gate.

Related Theory:

The NAND gate is simply an AND with a built-in inverter on the output. Therefore, a logic 0 is present at the output only when a logic 1 appears at all inputs. A logic 1 is present at the output if nay input has a logic 0. A NAND should always be thought of as AND – NOT.

The NAND is perhaps the most popular gate. You will note that its symbol is very similar to the AND except for the circle (called a bubble) at the output. The bubble shows that the normal AND output is inverted.

The NAND has many uses in digital circuits. It can also be used in combination with an inverter to produce an AND. Several NAND's can be combined to produce an OR gate. They are also used in memory (Flip-flop) circuits.

EQUIPMENT REQUIRED:

Power supply, Switch module, LED module, Quad 2-input NAND (7400), Hex Inverter (7404).

PROCEDURE:

Step 1. Connect the circuit of figure 2–5.

Step 2. Operate Switches A and B (H1 = 1; L0 = 0) and complete the truth table for the condition of LED f. (1 = lighted; 0 = Off) in Table 2-5.

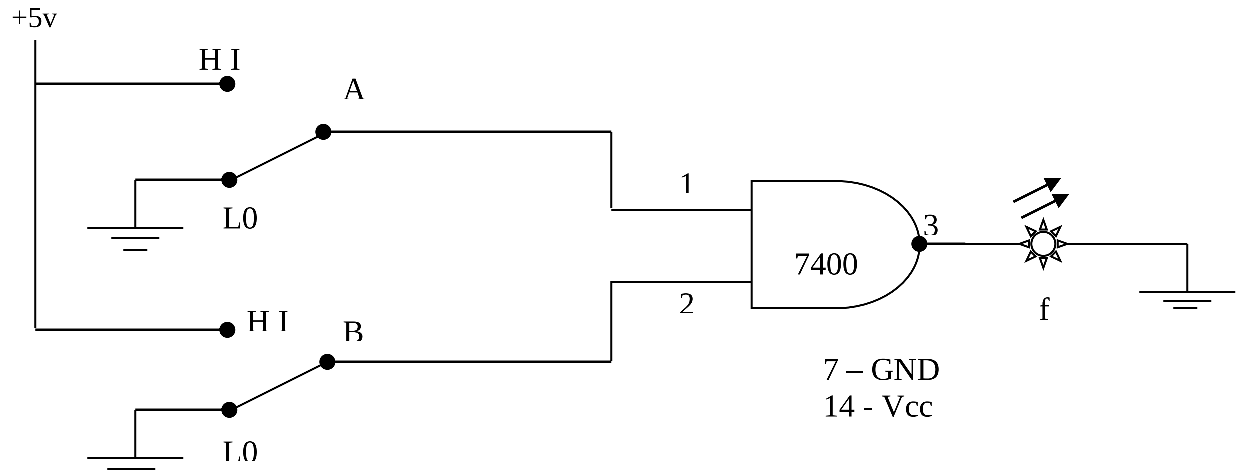


Figure 2-5

Step 3: Connect the circuit of figure 2-6.

Step 4: Repeat Step 2 for Table 2-6.

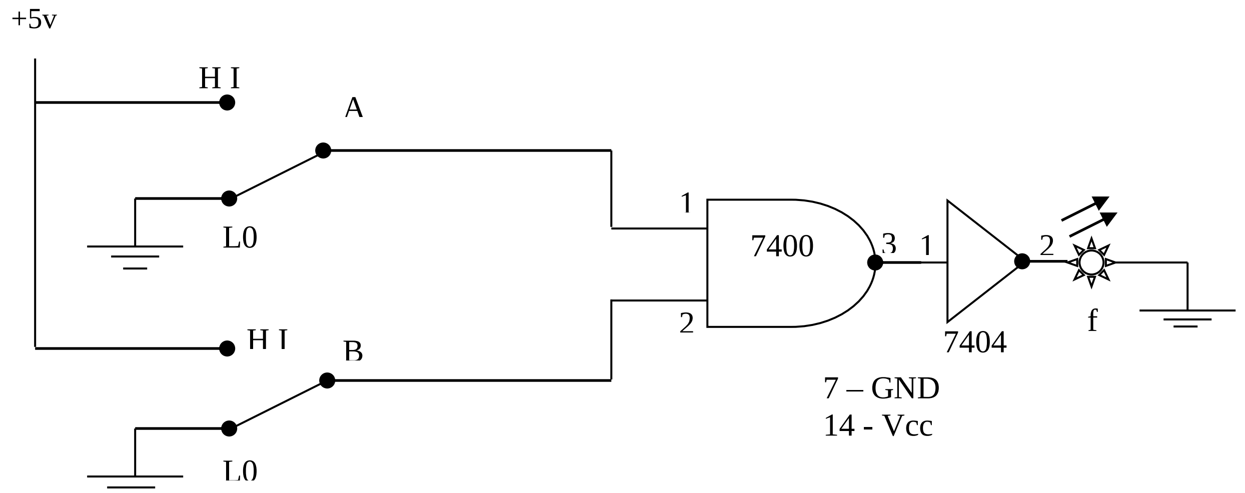


Figure 2-6

Step 5: Connect the circuit of figure 2-7.

Step 6: Repeat Step 2 for Table 2-7.

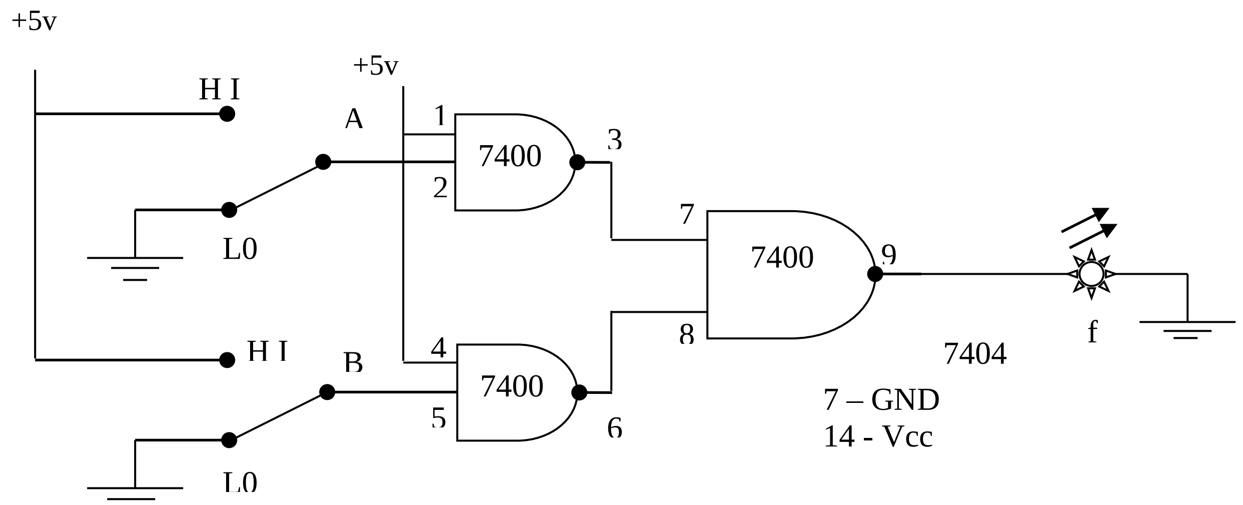


Figure 2-6

In your report, describe how you could make an AND gate using NAND gates only. Compare truth table 2.6 and 2.7 with truth table of other gates.

NAND

A	B	f

Table 2-5

AND

A	B	f

Table 2-6

OR

A	B	f

Table 2-7

OBJECTIVE NO: 4

Investigate the operation of NOR gate.

Related Theory:

Like the NAND, the NOR gate is extremely popular. Also like the NAND, the NOR gate is derived from other gates. It is OR gate with a built-in inverter on the output. It should be thought of as an OR – NOT.

A logic 1 is present at the output of a NOR gate only when all inputs are logic 1. A logic 1 at any one or more inputs will produce a logic 0 at the output.

All you have t do to use a NOR as an OR is to add an inverter at its output. If you wanted t use a NOR as a NOT, you could tie the unused input or inputs to ground (logic 0). Then when input A is logic 0 (all inputs are logic 0), the output is logic 1. If input A goes to logic 1, the output goes to logic 0. Three NOR gates can be combined to produce an AND gate. NOR's can also be used in memory (Flip-flop) circuits.

EQUIPMENT REQUIRED:
Power supply, Switch module, LED module, Quad 2-input NOR (7402), Hex Inverter (7404).

PROCEDURE:
Step 1. Connect the circuit of figure 2–8.

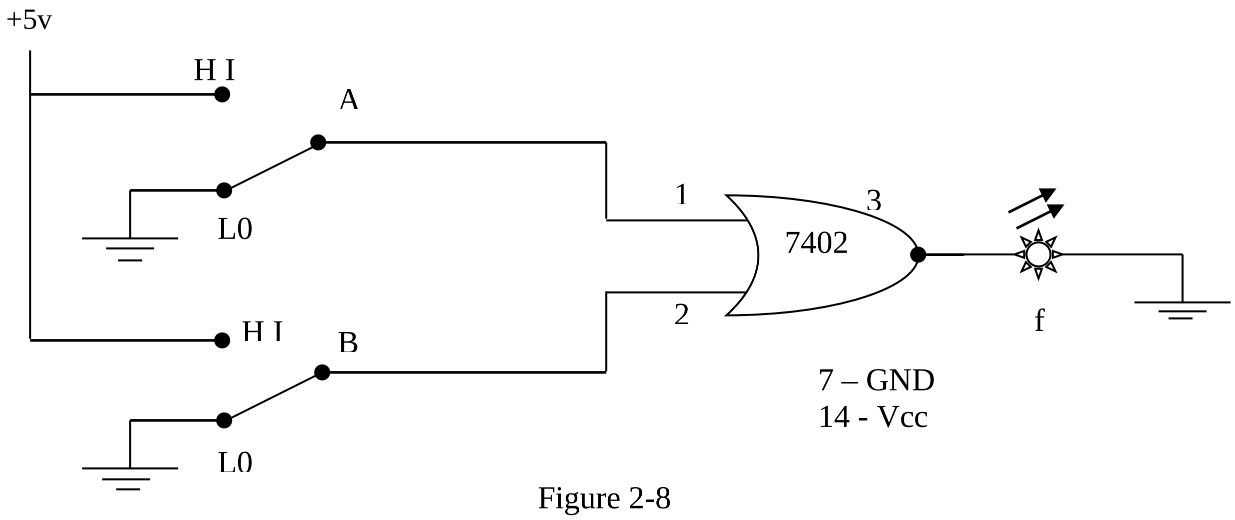


Figure 2-8

Step 2. Operate Switches A and B (H1 = 1; L0 = 0) and complete the truth table for the condition of LED f (1 = lighted; 0 = off) in Table 2-8.
Step 3. Connect the OR circuit of Figure 2-9.

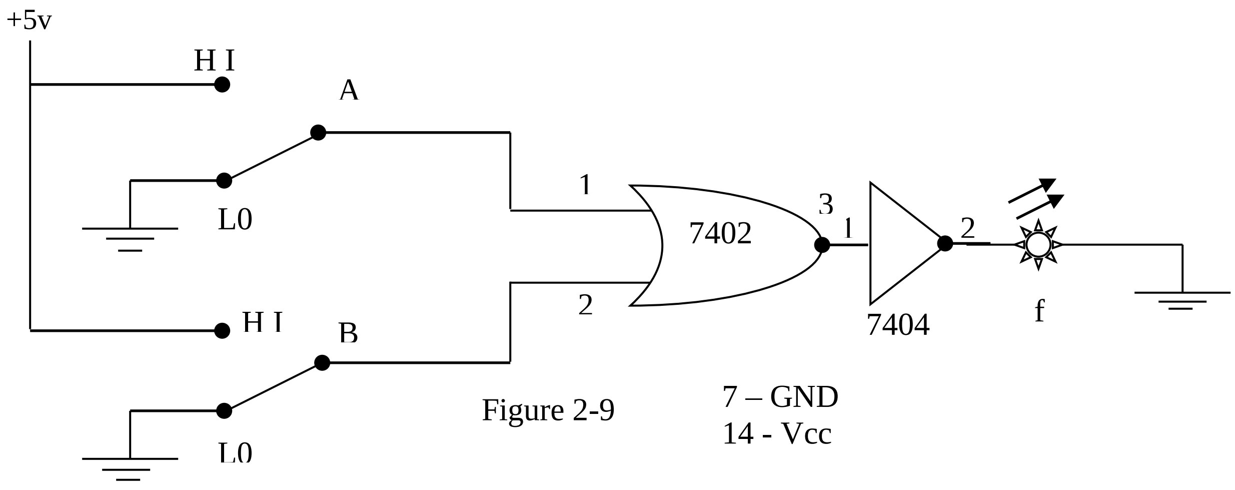


Figure 2-9

Step 4. Repeat Step 2 for Table 2-9.
Step 5. Connect the AND circuit of Figure 2-10.
Step 6. Repeat Step 2 for Table 2-10.

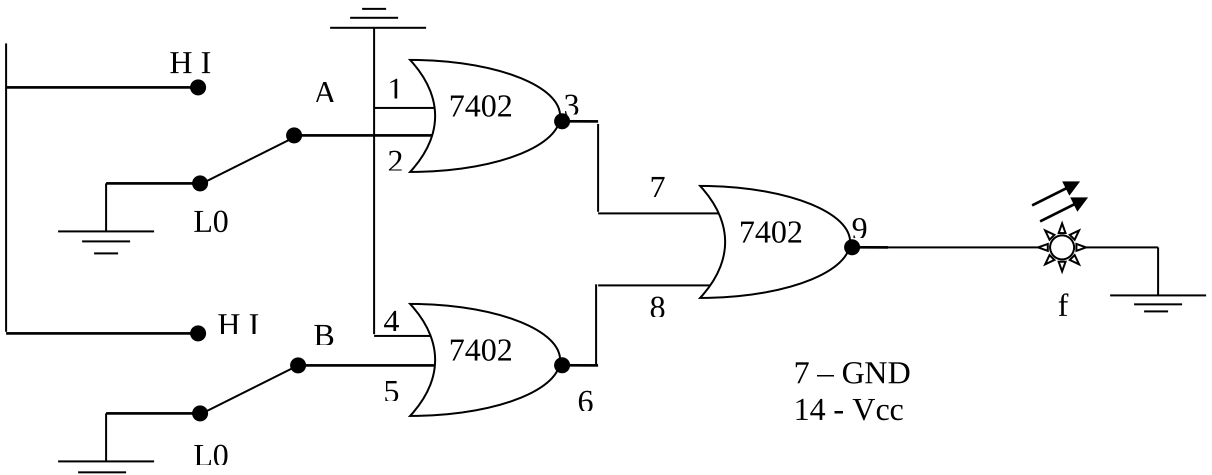


Figure 2-10

NOR

A	B	f

Table 2-8

OR

A	B	f

Table 2-9

AND

A	B	f

Table 2-10