UNIT: 1 – Representation of Information and Gates

- **Logic Gates: AND, OR, NOT, NAND, NOR, XOR, XNOR, Boolean Algebra**
- **Truth Tables, Circuit Equivalence**
- **Dr Morgan's Theorems**
- **Encoders, Decoders, Comparators**
- **Half-Adders, Full Adders, Binary Adders**

GATES

- For centuries mathematicians felt there was a connection between mathematics and logic, but no one before **George Boole** could find this missing link.
- In 1854 he invented symbolic logic, known as ***Boolean algebra.*** Each variable in Boolean algebra has either of two values: true or false.
- Boolean algebra had no practical application until 1938, when Claude Shannon used it to analyze telephone switching circuits.
- This chapter introduces the *gates,* a circuit with one or more input signals but only one output signals.
- Gates are digital (two-state) circuits because the input and output signals are either low or high voltage.
- Gates are often called *logic circuits* because they can be analyzed with Boolean algebra.

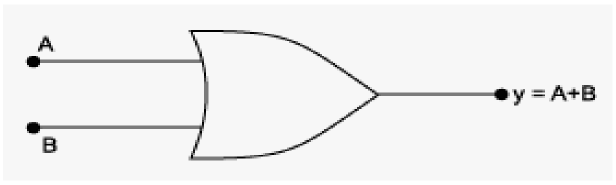
Logic Gates

A logic gate is an electronic circuit that makes logical decisions. It has one output and one or more inputs. The output signal appears only for certain combinations of input signals. Logic gates are the basic building blocks of all digital circuits. The most common logic gates used are OR, AND, NOT, NAND, and NOR gates. **The NAND and NOR gates are called universal gates**. The exclusive-OR gate is another logic gate which can be constructed using AND, OR and NOT gate.

Truth table: It is a table that shows all input and output possibilities for a logic gate. It is also called a table of combinations.

OR Gate

It has two or more inputs and one output.
The electronic symbol for a **two-input OR gate** is shown in the figure below.



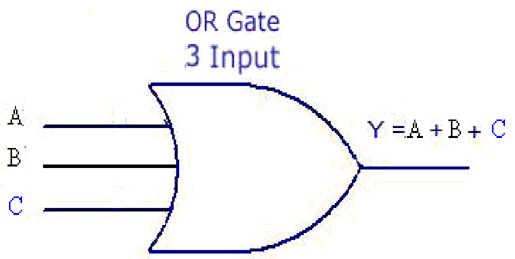
OR gate

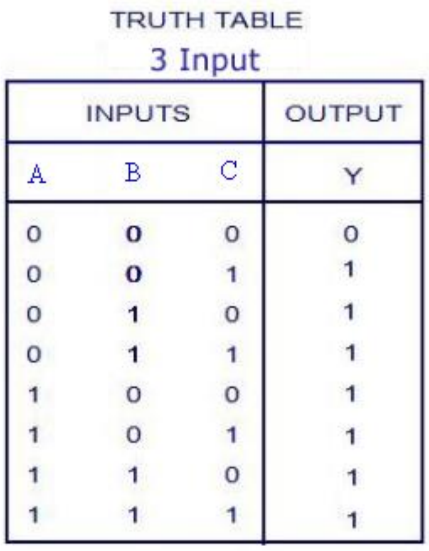
The two inputs have been marked A and B and the output as y. The OR gate has an output of 1 when either A or B or both are 1. In other words, **it is an any-or-all gate** because an output occurs when any or all the inputs are present. The output would be 0 if and only if both its inputs are 0.

The OR gate represents the Boolean equation A + B = y. The meaning of this equation is that y is true when either A is true or B is true or both are true. Alternatively, it means that y is 1 when either A or B or both are 1.

The above logic operation of the OR gate can be summarized with the help of the truth table shown below.

A	**B**	**y = A + B**
0	0	0
0	1	1
1	0	1
1	1	1





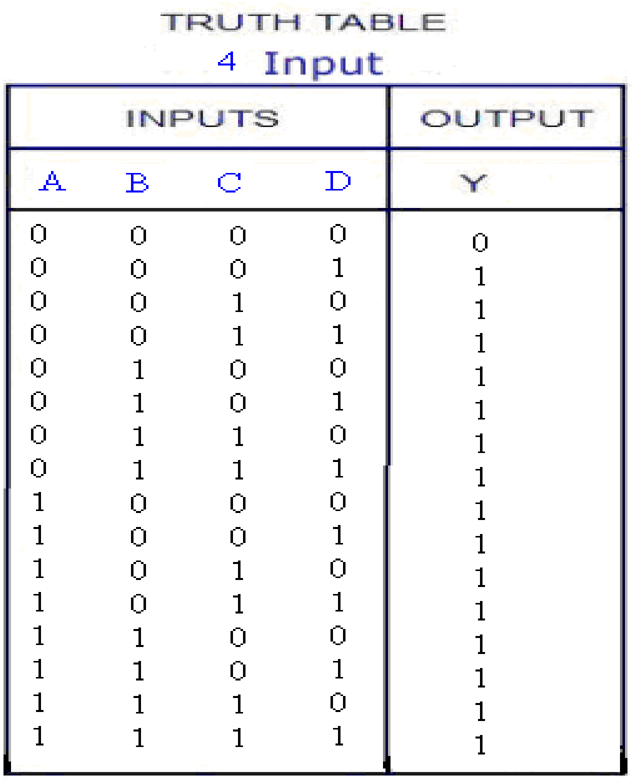
TRUTH TABLE
3 Input

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

Example 1: Show the truth table of a 4-input OR gate.

Solution:

- Let Y stands for the output bit and A, B, C, D for input bits. Then the truth table has input words of 0000, 0001, 0010… 1111, as shown in following table.
- As expected, output Y is 0 for input word 0000; Y is 1 for all other input words.
- As a check the number of input words in a truth table always equals $2^{n,}$ where n is the number of input bits. A 2-input OR gate has a truth table with 2^2 or 4 input words; and 3-input OR gate has 2^3 or 8 input words.


TRUTH TABLE

4 Input

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

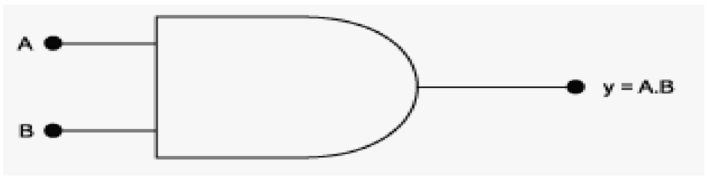
Example: How many inputs words are in the truth table of an 8-input OR gate? Which input words produce a high output?

Solution:

- The input words are 0000 0000, 0000 0001… 1111 1111.
- With the formula of the preceding example, the total number of input words is = $2^n = 2^3 = 256$.
- In any OR gate, 1 or more high inputs produce a high output. Therefore, the input words of 0000 0000 results in a low output; all other input words produce a high output.

AND Gate

- It has two or more inputs and one output.
- The electronic symbol for a two-input AND gate is shown in figure below.



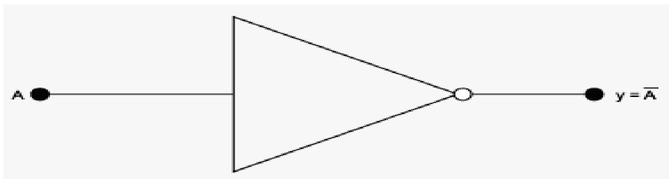
AND gate

- The AND gate gives its output only when all its inputs are present. The AND gate has an output 1 when both A and B are 1. Hence the gate is an **all-or-nothing** gate whose output occurs only when all its inputs are present.
- Its logical operations can be summarized with the help of the truth table.

A	B	y = A. B
0	0	0
0	1	0
1	0	0
1	1	1

NOT Gate

- It has only one input and only one output.
- It is a complementary operation and its symbol is an over bar. The negation or complement of 0 is 1. The symbol of NOT gate is shown below.



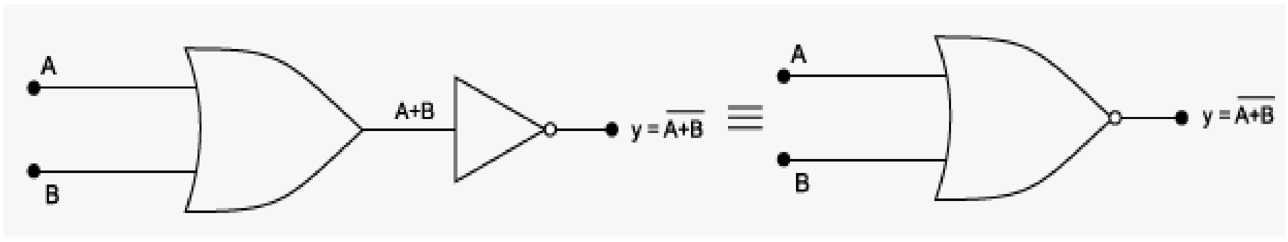
NOT gate

- The NOT gate is also called **inverter**. Its logical operations can be summarized with the help of the truth table.

A	$y = \overline{A}$
0	1
1	0

NOR Gate

- It is in fact a NOT-OR gate. It can be made out of an OR gate by connecting an inverter in its output as shown in the figure below.



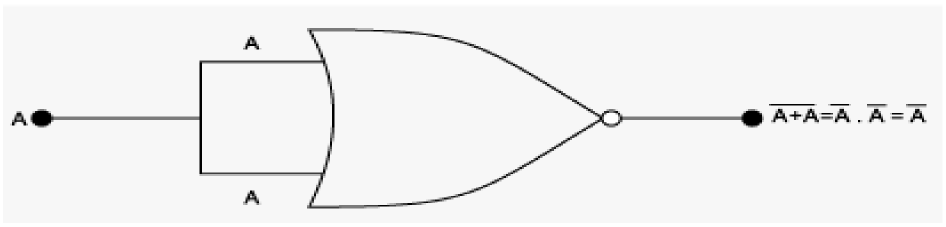
NOR gate

- The output equation is given by $y = \overline{A+B}$
- A NOR function is just the reverse of the OR function. A NOR gate will have an output of 1 only when all its inputs are 0. Obviously, if any input is 1, the output will be 0.
- Its logical operations can be summarized with the help of the truth table.

A	**B**	$y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

NOR gate is often referred to as a **universal gate** as it can be used to realize the basic logic functions OR, AND and NOT

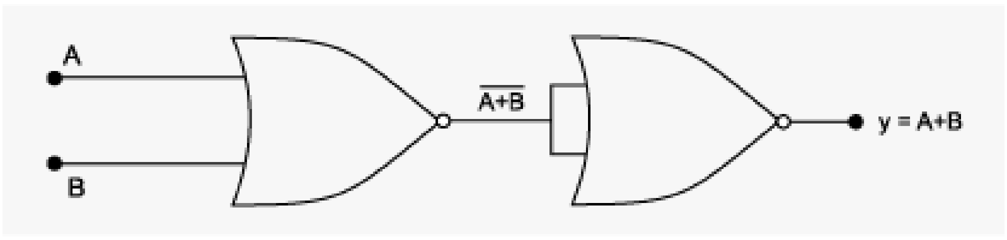
Design of NOT gate using NOR gates: The two inputs have been connected together as shown in the figure below.



NOT gate using NOR gate

The output is $\overline{A+A}=\overline{A}$ which can be proved to be equal to $\overline{A}$ with the help of De Morgan's theorem.

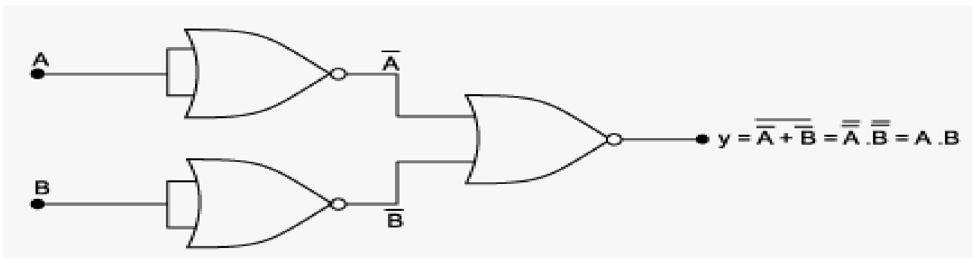
<u>Design of OR gate using NOR gates</u>: As can be seen from the figure below, the output from NOR gate is A + B.



OR gate using NOR gate

By using another inverter in the output, the final output is inverted and is given by y = A + B which is the logic function for a normal OR gate.

<u>Design of AND gate using NOR gates</u>: Here two inverters have been used, one for each input as shown in the figure below.

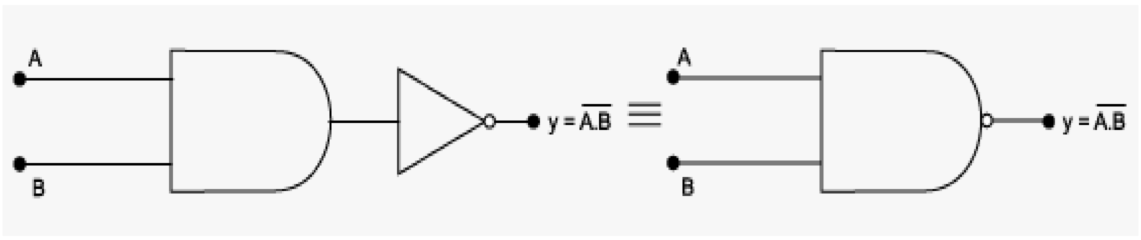

A
B
$\overline{A}$
$\overline{B}$
$y=\overline{\overline{A}+\overline{B}}=\overline{\overline{A}}\,.\overline{\overline{B}}=A\,.B$

AND using NOR gate

The inputs have, thus, been inverted before they are applied to the NOR gate. The output is equal to A. B which is same as that of AND gate.

NAND Gate

It is in fact, a NOT-AND gate. It can be obtained by connecting a NOT gate in the output of an AND gate as shown in the figure below.



NAND gate

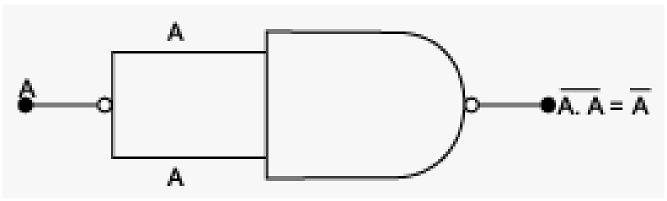
Its output can be given by the Boolean equation. $y = \overline{A.B}$

This gate gives an output of 1 if both inputs are not 1. In other words, it gives an output 1 if either A or B or both are 0.

A	B	$y = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

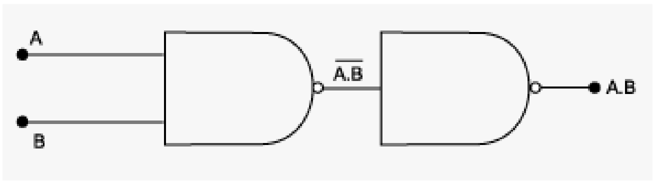
The truth table for a 2-input NAND gate is just opposite of the truth table for AND gate. It is because NAND gate performs a function that is reverse of an AND gate. NAND gate is also called the **universal gate** because it can perform all the three logic functions of an OR gate, AND gate and inverter (NOT gate) as shown below.

***Design of a NOT gate using NAND gate*:** A NOT gate can be made out of a NAND gate by connecting its two inputs together as shown in the figure below.



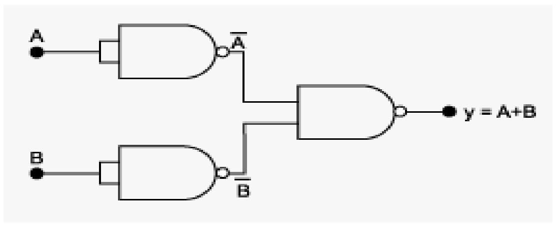
NOT gate using NAND gate

<u>Design of an AND gate using NAND gates</u>: The use of two NAND gates to produce an AND gate is shown in the figure below.



AND gate using NAND gate

<u>Design of an OR gate using NAND gate</u>: Similarly, the figure below shows how OR gate can be obtained out of three NAND gates. As seen from the figure, if the inputs A and B after being inverted by using two NOT gates (obtained from the NAND gates) are then fed to the NAND gate, the arrangement functions as the OR gate.

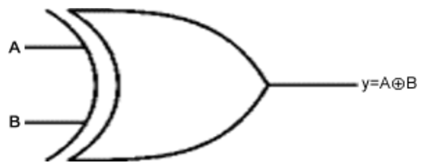


OR gate using NAND gate

X-OR Gate

The output y is true if either input A is true OR input B is true, but not when both of them are true: y = (A AND NOT B) OR (B AND NOT A)

This is like an OR gate but excluding both inputs being true. The output is true if inputs A and B are different. X-OR gates can only have 2 inputs.

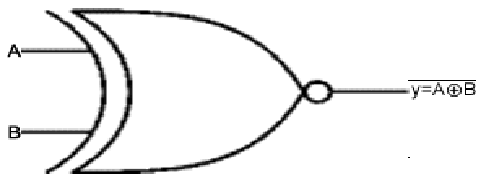


Its logical operations can be summarized with the help of the truth table.

A	B	y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

X-NOR Gate

This is an EX-OR gate with the output inverted. The output y is true if inputs A and B are the same (both true and both false): y = (A AND B) OR (NOT A AND NOT B)
EX-NOR gates can only have 2 inputs.



Its logical operations can be summarized with the help of the truth table.

A	B	$\overline{y=A\oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

Variable, complement, and literal are terms used in Boolean algebra. A variable is a symbol used to represent a logical quantity. Any single variable can have a 1 or a 0 value. The complement is the inverse of a variable and is indicated by a bar over variable (overbar). For example, the complement of the variable A is Ā. If A = 1, then Ā = 0. If A = 0, then Ā = 1. The complement of the variable A is read as "not A" or "A bar".

Boolean Addition

Boolean addition is equivalent to the OR operation. In Boolean algebra, a sum term is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are A + B, A + B, A + B + C, and A + B + C + D.

A sum term is equal to 1 when one or more of the literals in the term are 1. A sum term is equal to 0 only if each of the literals is 0.

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 1

Boolean Multiplication

Boolean multiplication is equivalent to the AND operation. In Boolean algebra, a product term is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are AB, AB, ABC, and ABCD.

A product term is equal to 1 only if each of the literals in the term is 1. A product term is equal to 0 when one or more of the literals are 0.

- $0 \cdot 0 = 0$
- $0 \cdot 1 = 0$
- $1 \cdot 0 = 0$
- $1 \cdot 1 = 1$

LAWS AND RULES OF BOOLEAN ALGEBRA

Laws of Boolean algebra

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

1. *Commutative Laws*

The commutative law of addition for two variables is written as A+B = B+A

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. Fig. (1) illustrates the commutative law as applied to the OR gate and shows that it does not matter to which input each variable is applied. (The symbol ≡ means "equivalent to.").

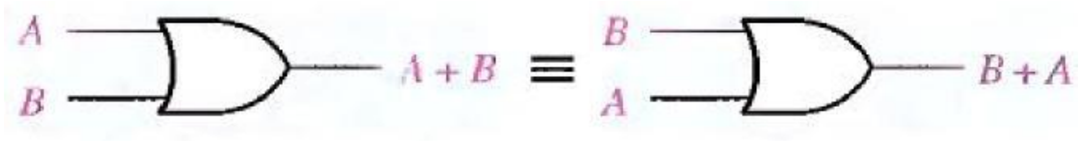


Fig. (1) Application of commutative law of addition.

The commutative law of multiplication for two variables is A · B = B ·A

This law states that the order in which the variables are ANDed makes no difference. Fig. (2), il1ustrates this law as applied to the AND gate.

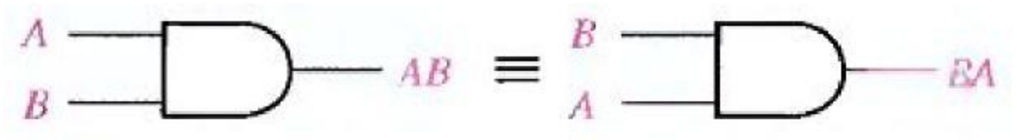


Fig. (2) Application of commutative law of multiplication

2. <u>*Associative Laws:*</u>

The associative law of addition is written as follows for three variables:

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

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Fig. (3), illustrates this law as applied to 2-input OR gates.

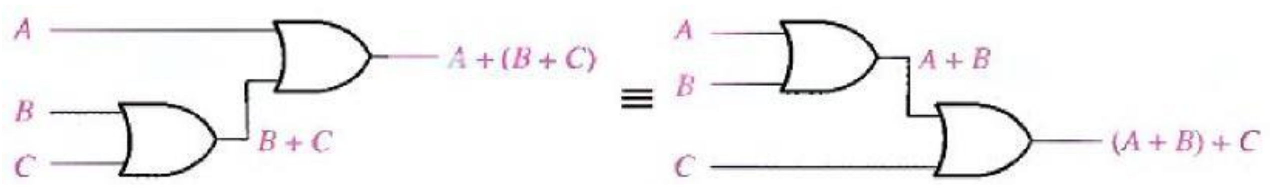


Fig. (3) Application of associative law of addition.

The associative law of multiplication is written as follows for three variables:

$$A(BC) = (AB)C$$

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Fig. (4) illustrates this law as applied to 2-input AND gates.

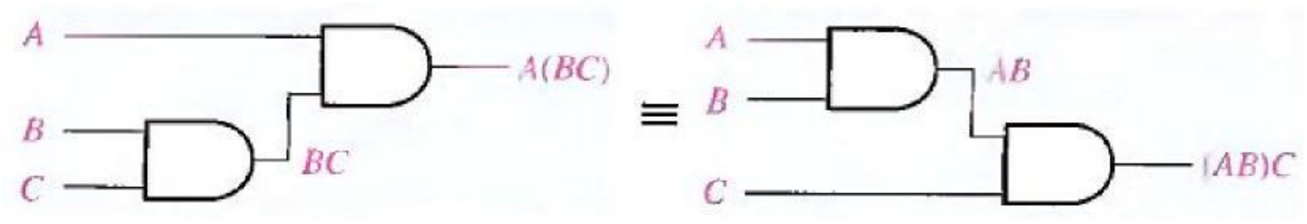


Fig. (4) Application of associative law of multiplication

3. <u>*Distributive Law*</u>

The distributive law is written for three variables as follows: A(B + C) = AB + AC

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of factoring in which the common variable A is factored out of the product terms, for example, AB + AC = A (B + C).

Fig. (5) illustrates the distributive law in terms of gate implementation.

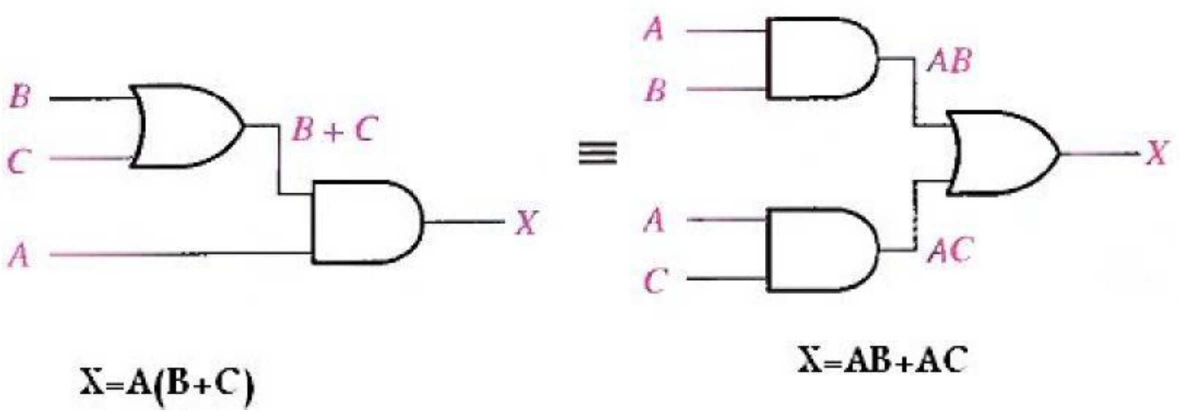
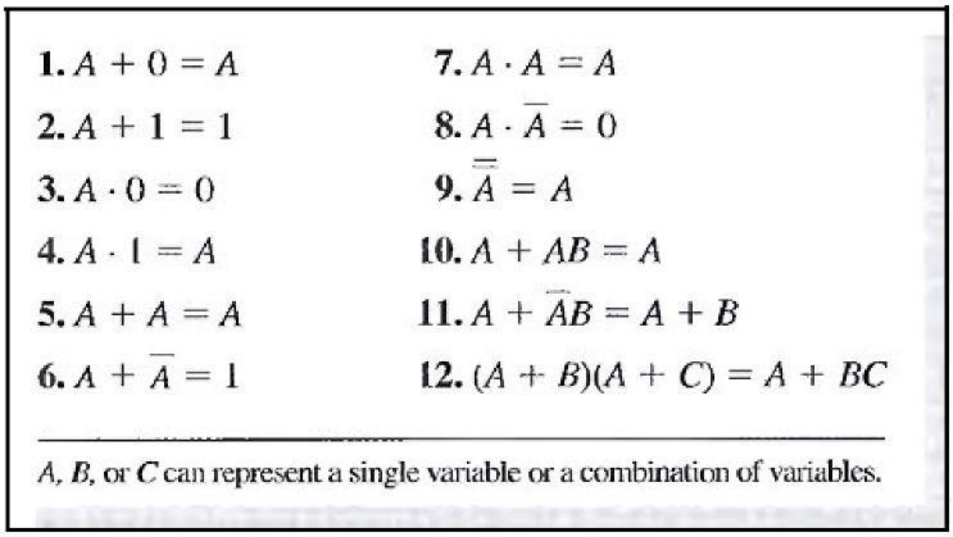


Fig. (5) Application of distributive law

Rules of Boolean Algebra

- Table lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions.
- Rules 1 through 9 will be viewed in terms of their application to logic gates.
- Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.



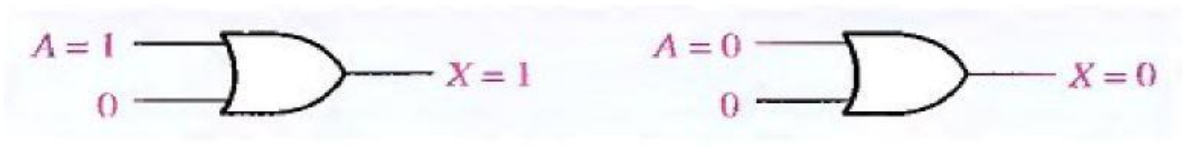
1. $A + 0 = A$	7. $A \cdot A = A$
2. $A + 1 = 1$	8. $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	9. $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	10. $A + AB = A$
5. $A + A = A$	11. $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	12. $(A + B)(A + C) = A + BC$

A, B, or *C* can represent a single variable or a combination of variables.

Basic rules of Boolean algebra

Rule 1. **A + 0 = A**

A variable ORed with 0 is always equal to the variable. If the input variable A is 1, the output variable X is 1, which is equal to A. If A is 0, the output is 0, which is also equal to A. This rule is illustrated in Fig. (6), where the lower input is fixed at 0.

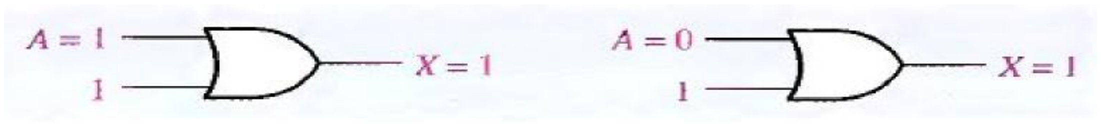


A+0=A

Rule 2. **A + 1 = 1**

A variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input.

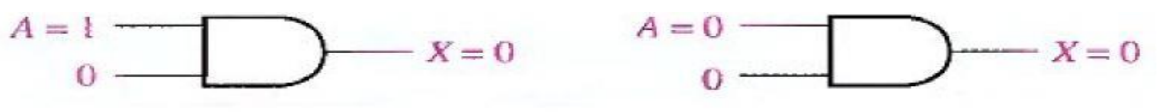
This rule is illustrated in Fig. (7), where the lower input is fixed at 1.



X=A+1=1

Rule 3. **A · 0 = 0**

A variable ANDed with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Fig.(8), where the lower input is fixed at 0.



X= A.O = O

Rule 4. **A . 1 = A**

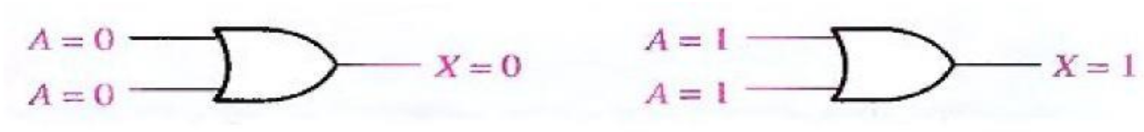
A variable ANDed with 1 is always equal to the variable. If A is 0 the output of the AND gate is 0. If A is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Fig. (9), where the lower input is fixed at 1.

A = 0
1
X = 0
A = 1
1
X = 1

X = A . 1 = A

Rule 5. **$A + A = A$**

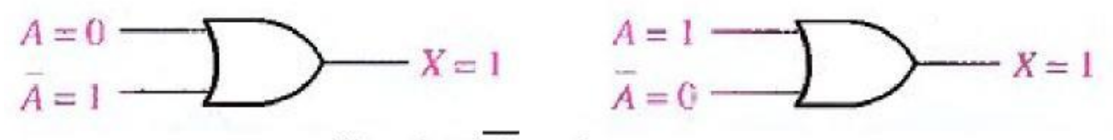
A variable ORed with itself is always equal to the variable. If A is 0, then 0 + 0 = 0; and if A is 1, then 1 + 1 = 1. This is shown in Fig.(10), where both inputs are the same variable.



$X = A + A = A$

Rule 6. **$A + \overline{A} = 1$**

A variable ORed with its complement is always equal to 1. If A is 0, then 0 + 0 = 0 + 1 = 1. If A is l, then 1 + 1 = 1+ 0 = 1. See Fig. (11), where one input is the complement of the other.

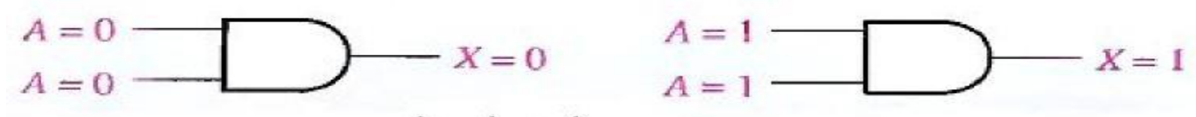


$X = A + \overline{A} = 1$

Rule 7. **$A . A = A$**

A variable ANDed with itself is always equal to the variable. If A = 0, then 0.0 = 0; and if A = 1.

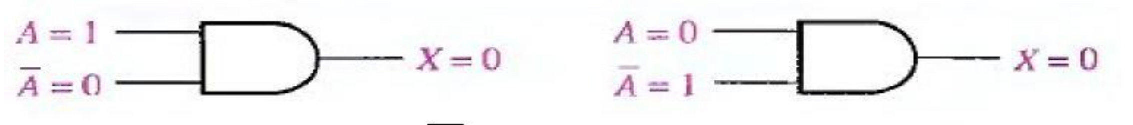
then 1.1 = 1. Fig.(12) illustrates this rule.



$A . A = A$

Rule 8. **$A \cdot \overline{A} = 0$**

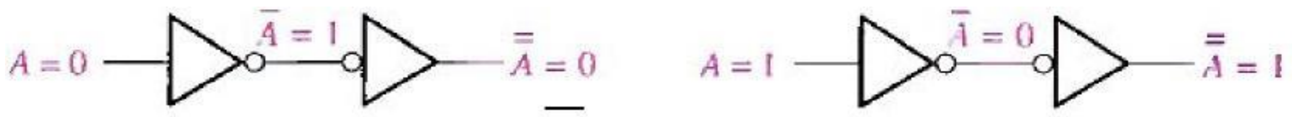
A variable ANDed with its complement is always equal to 0. Either A or A will always be 0: and when a 0 is applied to the input of an AND gate. the output will be 0 also. Fig.(13) illustrates this rule.



$A . \overline{A} = 0$

Rule 9. **$A = \overline{\overline{A}}$**

The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get A. If you then take A and complement (invert) it, you get A, which is the original variable. This rule is shown in Fig.(14) using inverters.



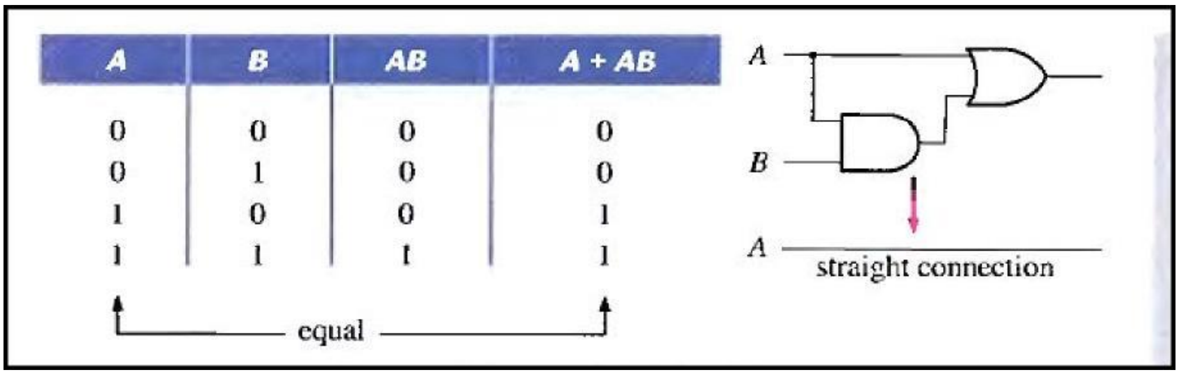
$A = \overline{\overline{A}}$

Rule 10. **A + AB = A**

This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

A + AB =	A (1 + B)	Factoring (distributive law)
=	A · 1	Rule 2: (1 + B) = 1
=	A	Rule 4: A . 1 = A

The proof is shown in Table, which shows the truth table and the resulting logic circuit simplification.



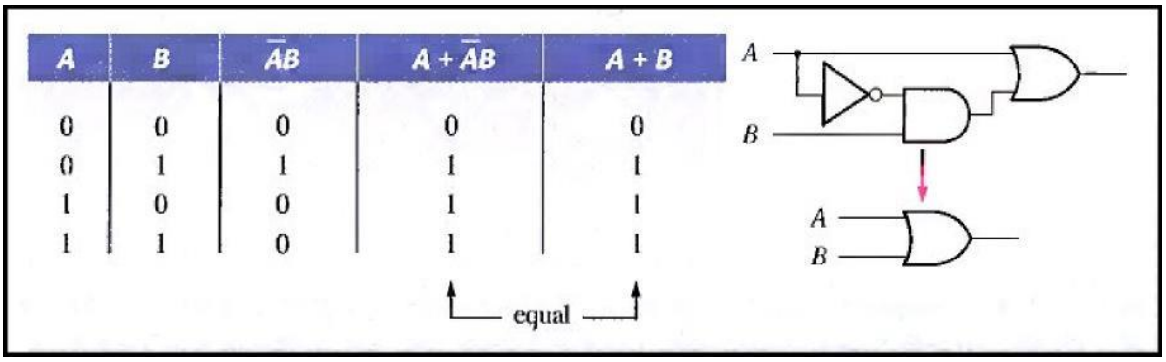
A	B	AB	A + AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Rule 11. **A +** $\overline{A}$ · B **= A + B**

This rule can be proved as follows:

A + $\overline{\mathbf{A}}$ **· B =**	(A + AB) + $\overline{A}$B	Rule 10: A = A + AB
=	(AA + AB) + $\overline{A}$B	Rule 7: A = AA
=	AA + AB + A$\overline{A}$ + $\overline{A}$B	Rule 8: adding AA = 0
=	A(A+B) + $\overline{A}$ (A+B)	Factoring
=	(A + $\overline{A}$)(A + B)	
=	1. (A + B)	Rule 6: A + $\overline{A}$ = 1
=	A + B	Rule 4: drop the 1

The proof is shown in Table, which shows the truth table and the resulting logic circuit simplification.



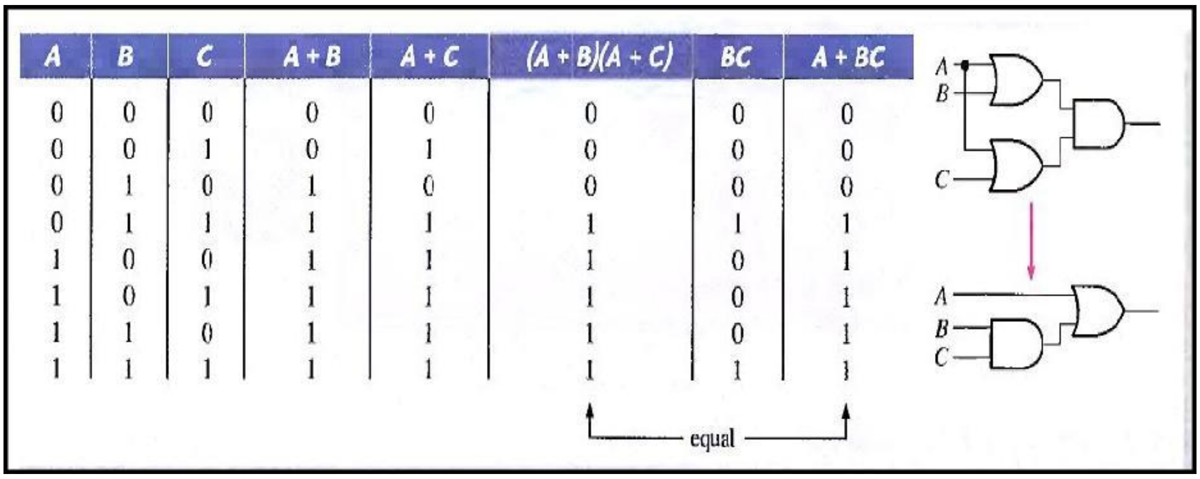
A	B	$\overline{A}B$	$A+\overline{A}B$	$A+B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

Rule 12. (A + B)(A + C) = A + BC

This rule can be proved as follows:

(A + B)(A + C) =	AA + AC + AB + BC	Distributive law
=	A + AC + AB + BC	Rule 7: AA = A
=	A (1 + C) + AB + BC	Rule 2: 1 + C = 1
=	A. 1 + AB + BC	Factoring (distributive law)
=	A (1 + B) + BC	Rule 2: 1 + B = 1
=	A. 1 + BC	Rule 4: A . 1 = A
=	A + BC	Rule 4: drop the 1

The proof is shown in Table, which shows the truth table and the resulting logic circuit simplification.



A	B	C	$A+B$	$A+C$	$(A+B)(A+C)$	BC	$A+BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal
A
B
C
A
B
C

Circuit Equivalence

Circuits with fewer and/or simpler gates (fewer inputs) are better.

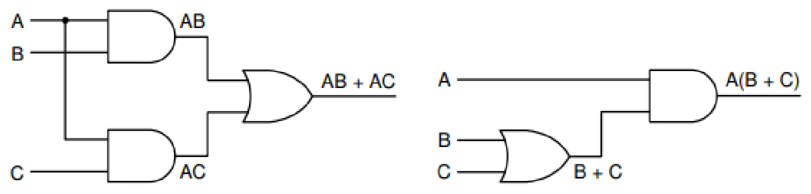
Boolean algebra can be a valuable tool for simplifying circuits.

Example:

M = AB + AC

Many of the rules of ordinary algebra also hold for Boolean algebra.

In particular, AB + AC can be factored into A(B + C) using the distributive law.



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

Two equivalence functions **(a) AB + AC** and **(b) A (B + C)**

Two functions are equivalent if and only if they have the same output for all possible inputs

Thus, AB + AC is equivalent to A (B + C).

De Morgan's Theorem

De Morgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

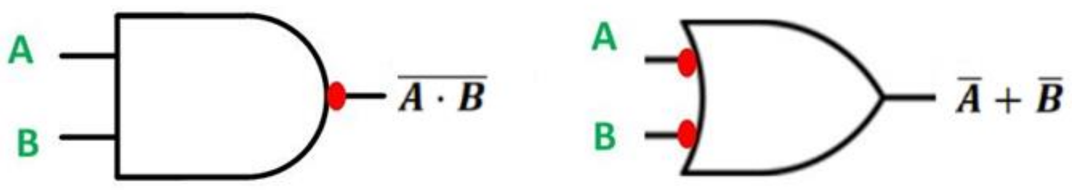
De Morgan's first theorems is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables,

Stated another way,

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables. NOR gate is equivalent to bubbled AND gate.

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



A	B	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

De Morgan's second theorem is stated as follows:

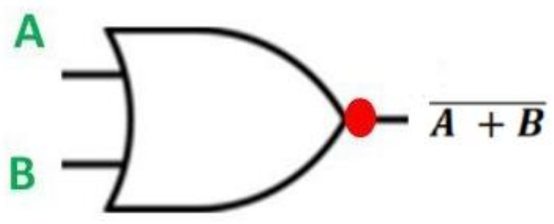
The complement of a sum of variables is equal to the product of the complements of the variables.

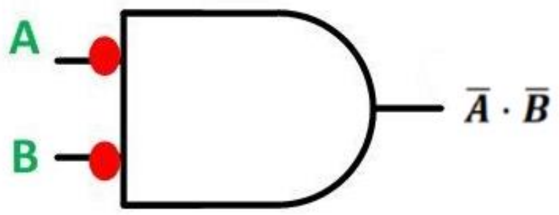
Stated another way,

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables,

The formula for expressing this theorem for two variables is

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





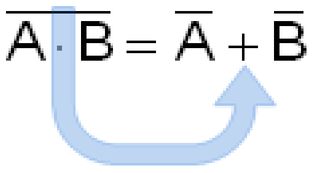
A	B	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Gate equivalencies and the corresponding truth tables that illustrate De Morgan's theorems.

DeMorgan Shortcut

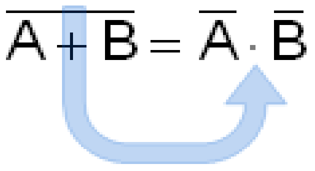
BREAK THE LINE, CHANGE THE SIGN

Break the *LINE* over the two variables,
and change the *SIGN* directly under the line.



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

For Theorem #14A, break the line, and change the AND function to an OR function. Be sure to keep the lines over the variables.



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

For Theorem #14B, break the line, and change the OR function to an AND function. Be sure to keep the lines over the variables.

Simplification using Boolean Algebra

A simplified Boolean expression uses the fewest gates possible to implement a given expression.

Example

Using Boolean algebra techniques, simplify this expression: AB + A(B + C) + B(B + C)

Solution

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

AB + AB + AC + BB + BC

Step 2: Apply rule 7 (BB = B) to the fourth term.

AB + AB + AC + B + BC

Step 3: Apply rule 5 (AB + AB = AB) to the first two terms.

AB + AC + B + BC

Step 4: Apply rule 10 (B + BC = B) to the last two terms.

AB + AC + B

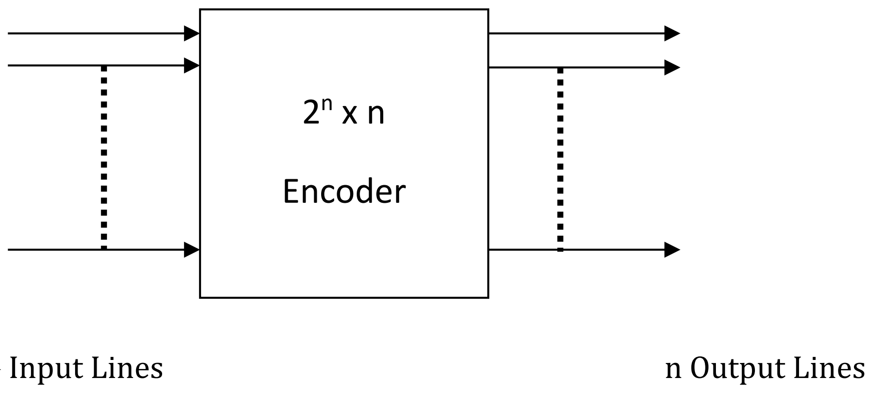
Step 5: Apply rule 10 (AB + B = B) to the first and third terms.

B+AC

At this point the expression is simplified as much as possible.

Encoder

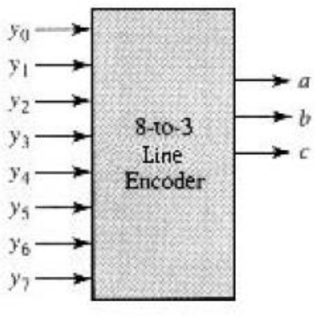
- An encoder is a device or circuit that converts information from one format or code to another, for the purpose of standardization, speed, secrecy or saving space by shrinking size.
- An encoder is a combinational circuit that has '2^n' inputs and an enable line (a sort of selection line) and n output lines.
- An encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code corresponding to the input value.
- If a device output code has fewer bits than the input code has, the device is usually called an encoder. e.g. 2^n – to – n.
- Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines.
- An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4-line configurations. The output lines of a digital encoder. Generate the binary equivalent of the input line whose value is equal to "1".



[Block diagram of Encoder]

8 x 3 Encoder (Octal to Binary Encoder)

It takes 8 inputs, one for each of the octal digits and provides 3 outputs. At any one time, only one input line has a value of 1. The figure below shows the Block diagram of an 8x3 encoder.



Truth Table

Inputs								Output		
I0	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**Y2**	**Y1**	**Y0**
1	0	0	0	0	0	0	0	**0**	**0**	**0**
0	1	0	0	0	0	0	0	**0**	**0**	**1**
0	0	1	0	0	0	0	0	**0**	**1**	**0**
0	0	0	1	0	0	0	0	**0**	**1**	**1**
0	0	0	0	1	0	0	0	**1**	**0**	**0**
0	0	0	0	0	1	0	0	**1**	**0**	**1**
0	0	0	0	0	0	1	0	**1**	**1**	**0**
0	0	0	0	0	0	0	1	**1**	**1**	**1**

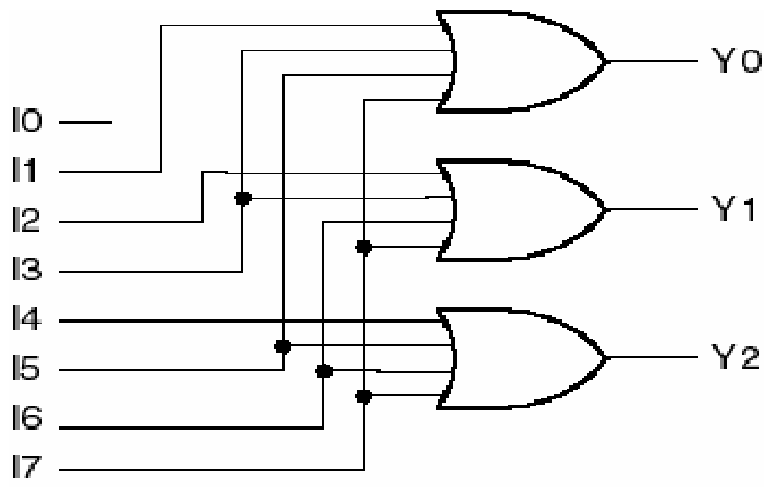
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output Y2 =1 if the input octal digit is 4 or 5 or 6 or 7. Similar conditions apply for other two outputs.
- These conditions can be expressed by the following Boolean functions. For an 8-to-3 binary encoder with inputs I0-I7 the logic expressions of the outputs Y2, Y1 and Y0 are:

$$Y2 = I4 + I5 + I6 + I7$$

$$Y1 = I2 + I3 + I6 + I7$$

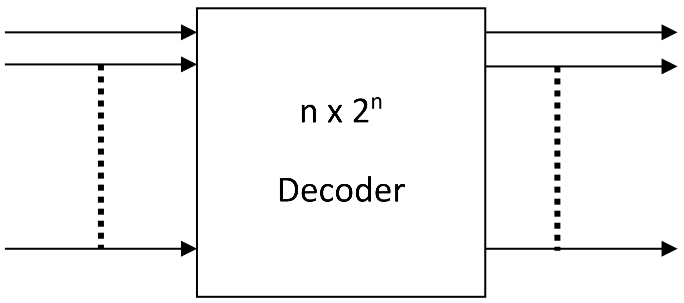
$$Y0 = I1 + I3 + I5 + I7$$

Based on the above equations, we can draw the circuit as shown below



Decoder

- Decoder converts one type of coded information to another form. It is a combinational circuit which detects the presence of discrete binary patterns at the input and produces a unique output corresponding to each of the detected codes.
- A decoder has 'n' inputs and 2^n (or 2n) output lines.
- Discrete quantities of information are represented in digital computers with binary codes. A binary code of n bits is capable of representing up to 2n distinct elements of the coded information. A decoder is a combinational circuit that converts binary information from the n coded inputs to a maximum of 2n unique outputs.
- A decoder has n inputs and m outputs and is also referred to as an n x m decoder.
- Decoders are extensively used in a computer system for addressing decoding.
- The figure below shows the Block diagram of an 3x8 Decoder.



[Block diagram of Decoder]

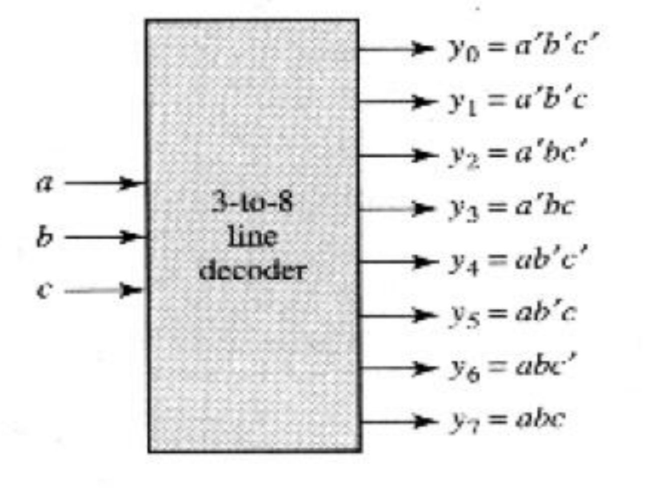
3x8 Decoder

3 x 8 decoder which decodes a 3-bit information and there is only one output line which gets the value 1 or in other words, out of 2×3 = 8 lines only 1 output line is selected. Thus, depending on selected output line the information of the 3 bits can be recognized or decoded.

The decoders presented in this section are called n-to-m-line decoders, where m <= 2^n. Their purpose is to generate the 2^n (or fewer) binary combinations of the n input variables. A decoder has n inputs and m outputs and is also referred to as an n x m decoder.

- The three data inputs. A0, A1, and A2, are decoded into eight outputs, each output representing one of the combinations of the three binary input variables. The three inverters provide the complement of the inputs, and each of the eight AND gates generate one of the binary combinations.

- A particular application of this decoder is a binary-to-octal conversion. The input variables represent a binary number and the outputs represent the eight digits of the octal number system.



- However, a 3-to-8-line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each combination of the binary code.

Truth Table

Inputs			**Output**							
A2	**A1**	**A0**	**D0**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**
0	**0**	**0**	1	0	0	0	0	0	0	0
0	**0**	**1**	0	1	0	0	0	0	0	0
0	**1**	**0**	0	0	1	0	0	0	0	0
0	**1**	**1**	0	0	0	1	0	0	0	0
1	**0**	**0**	0	0	0	0	1	0	0	0
1	**0**	**1**	0	0	0	0	0	1	0	0
1	**1**	**0**	0	0	0	0	0	0	1	0
1	**1**	**1**	0	0	0	0	0	0	0	1

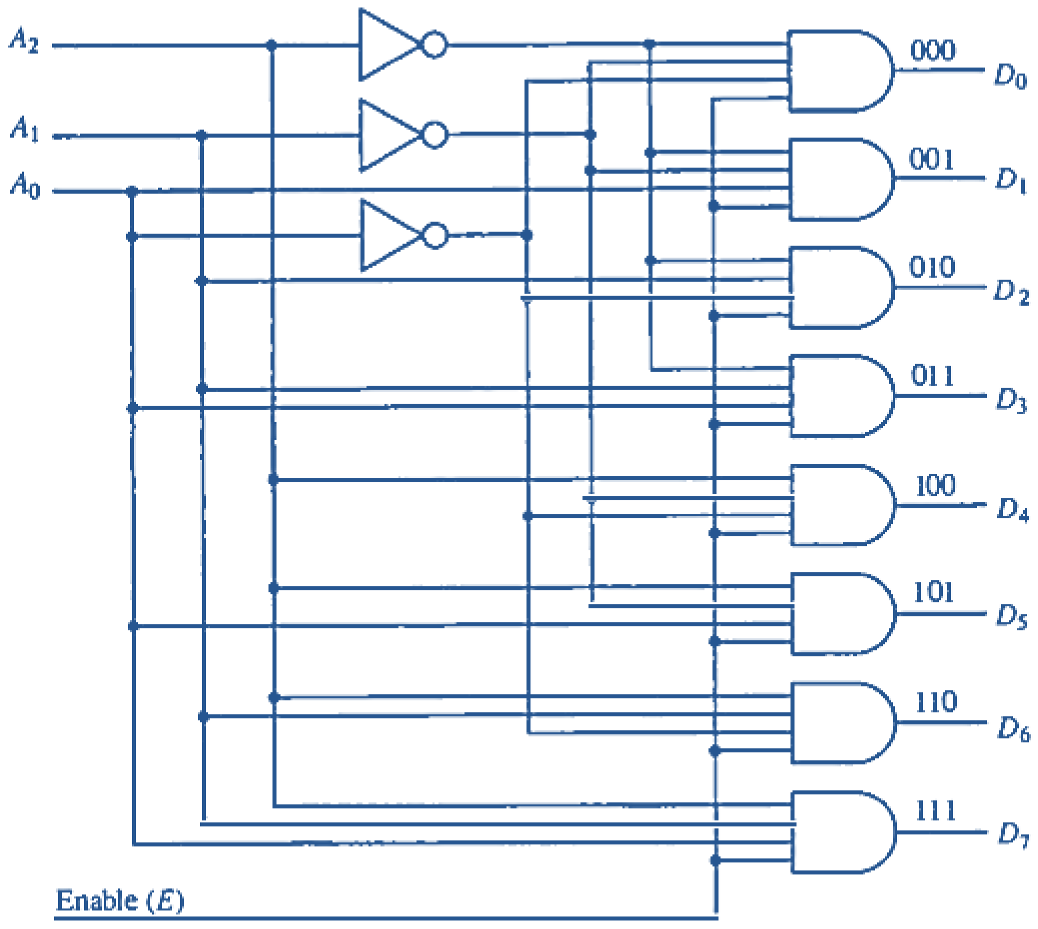
From the above truth table, we can derive the functions A2, A1, A0 as given below:

D0 = A2’A1’A0’ D1=A2’A1’A0

D2 = A2’A1A0’ D3=A2’A1A0

D4=A2A1’A0’	D5= A2A1’A0
D6=A2A1A0’	D7=A2A1A0

The logic diagram of a 3-to-8-line decoder is shown in the Figure:



Comparator

- Comparator is useful for implementing relational operations such as =, <, > and so on. The comparison of two numbers is an operation that determines if one number is greater than, less than or equal to other number.
- It is used XNOR gate and AND gate to make word comparisons. This circuit produce output 1 if word is equal A = B otherwise it produce 0.

- It is a combinational circuit that compares two numbers, A and B, and determines their relative magnitude. The outcome of the comparison is specified by three binary variables that indicates whether A>B, A = B or A <B.
- Consider two numbers, A and B, with four digit each.

A = A3A2A1A0

B = B3B2B1B0

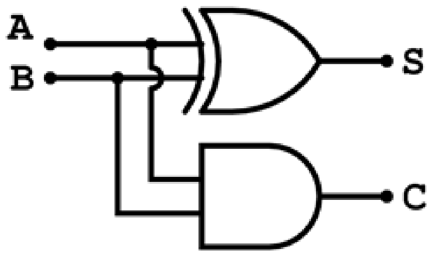
- Where each subscripted letter represents one of the digits in the number. The two numbers are equal if all pairs of significant digits are equal. i.e if $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$.
- When the numbers are binary, the digits are either 1 or 0 and the equality relation of each pair of bits can be expressed logically with an equivalence function:

$$\mathbf{Xi = A_iB_i + A'_i + B'_i}$$

- The comparison of two positive numbers A and B is performed in a bit-by-bit manner starting with the most significant bit:
  - If the most significant bits are A_n = '1' and B_n = '0' then number A is larger than B.
  - If A_n = '0' and B_n = '1' then number A is smaller than B
  - If $A_n = B_n$ then no decision can be taken about A and B based only on these two bits.
  - If the most significant bits are equal then the result of the comparison is determined by the less significant bits $A_n - 1$ and $B_n - 1$. if these bits are equal as well, the process continue with the next pair of bits. If all bits are equal then the two numbers are equal.

Half Adder

A half adder is a logical circuit that performs an addition operation on two one-bit binary numbers. The half adder outputs a sum of the two inputs and a carry value. A half adder can add two bits. It has two inputs, generally labeled A & B, and two outputs, the sum S & carry C.



A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The Boolean equations for this output are:

$$SUM = A \oplus B$$
$$CARRY = AB$$

S is the two-bit XOR of A and B, and C is the AND of A and B. Essentially the output of a half adder is the sum of two one-bit numbers, with C being the most significant of these two outputs.

The SUM output is A XOR B; the CARRY output is A AND B. Therefore. SUM is a 1 when A and B are different; CARRY is a 1 when A and S are 1s.

When A and B are 0s, the SUM is 0 with a CARRY of 0. When A is 0 and B is 1, the SUM is 1 with a CARRY of 0. When A is 1 and B is 0, the SUM equals 1 with a CARRY of 0. Finally, when A is 1 and B is 1, the SUM is 0 with a CARRY 1.

The drawback of this circuit is that in case of a multibit addition, it cannot include a carry.

Full Adder:

A full adder is a combinational circuit that performs the arithmetic sum if three input bits and produces a sum and a carry.

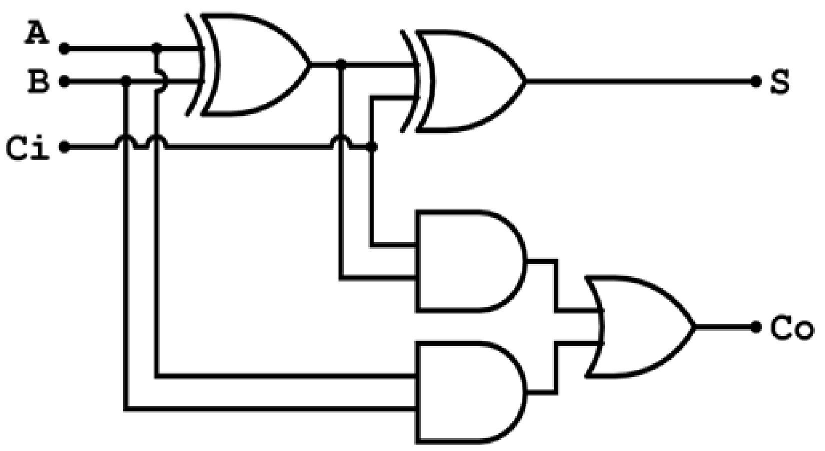
The main drawback of Half adder is it has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit operation is performed so to solve this problem, a full adder is designed.

The Boolean equations for these outputs are:

$$SUM = A \oplus B \oplus C$$
$$CARRY = AB + AC + BC$$

A full adder can be constructed from two half adders by connecting A and B to the input of one-half adder, connecting the sum from that to an input to the second adder, connecting Ci to the other input and OR the two carry outputs.

A	*B*	*C*	*CARRY*	*SUM*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



In this case, SUM equals A XOR B XOR C; CARRY equals AR OR AC OR BC. Therefore, SUM is 1 when the number of input 1s is odd; CARRY is a 1 when two or more inputs are 1s.

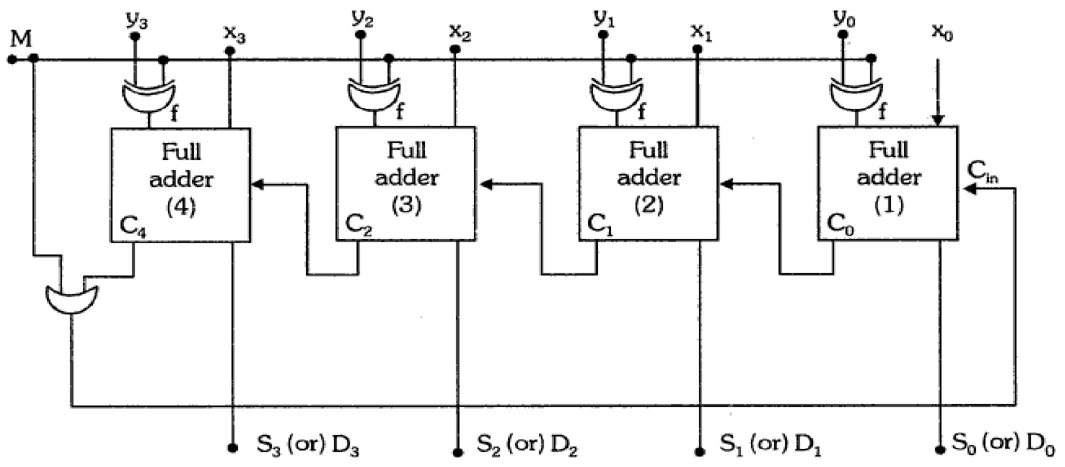
Binary Adders-Subtractor

The addition and subtraction can be combined into one circuit with one common binary adder. Consider a single system which adds and subtract. There are two types of such single systems.

1. 1's complement adder-subtractor
2. 2's complement adder-subtractor

1's Complement adder-subtractor

The single system of a 4-bit adder cum l's complement subtractor is shown in Figure. This system operates between $y_3y_2y_1y_0$ and $x_3x_2x_1x_0$.



M = 1 = Subtract ;
M = 0 = Add ;

	M	y	f
Add	0	0	0
	0	1	1
Subtract	1	0	1
	1	1	0

$$\begin{array}{l} +x_3x_2x_1x_0 \\ -y_3y_2y_1y_0 \\ \hline S_3S_2S_1S_0 \\ \text{or} \\ D_3D_2D_1D_0 \end{array}$$

Operation 1- When control input M = 0

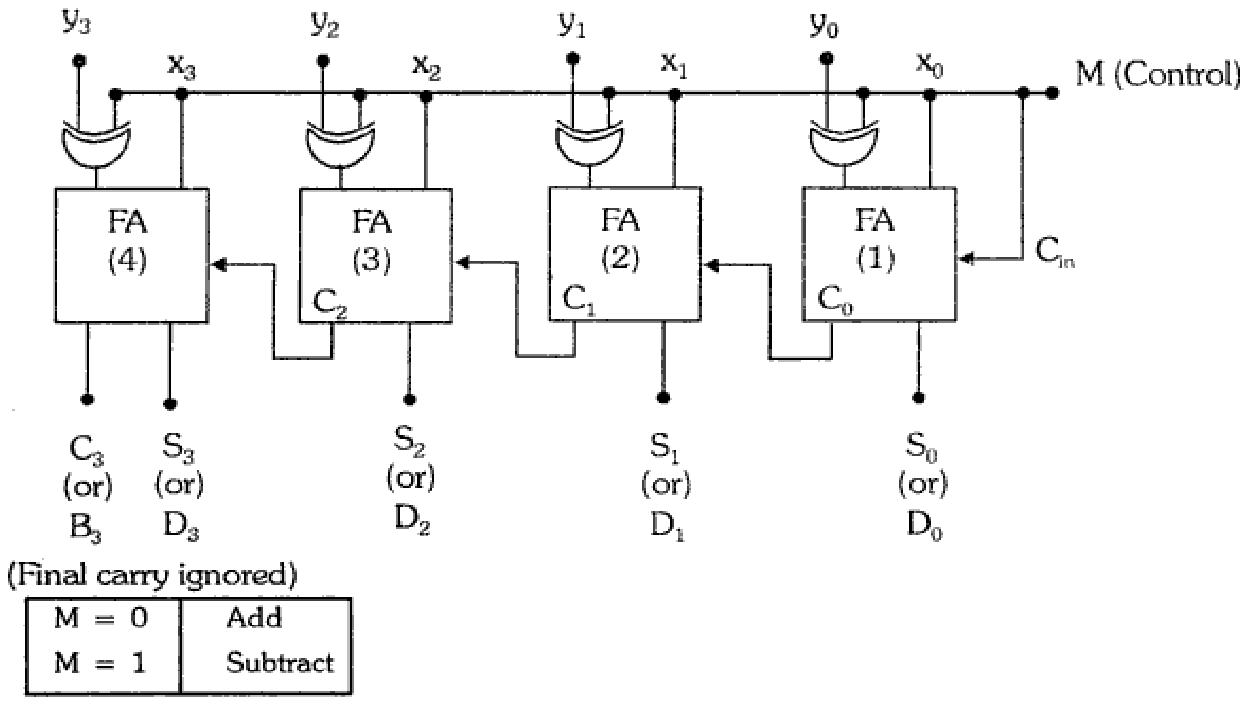
The bits y_3, y_2, y_1 and y_0 are EX-ORed with '0' and produces ($y_3 \oplus 0 = y_3$) $y_3y_2y_1$ and y_0 respectively as outputs of EX-OR gate. The full-adders receive the value of y and the input carry (end around carry) = 0. Hence, the circuit will act as an "adder", The result appears as $S_3S_2S_1S_0$ in binary form.

Operation 2 - When control input M = 1

To make the above Figure as a subtractor, the control input must be 1. This causes the EX-OR gate to act as an inverter for the 'y' inputs to the full adders. (i.e., $y \oplus 1 = \overline{Y}$). The 1 on the control input activates the AND gate so that the carry output of the full adder can take the end-around carry back to the first full adder. Thus, subtractor subtracts the binary input $y_3y_2y_1y_0$ from $x_3x_2x_1x_0$. The difference appears as $D_3D_2D_1D_0$ in binary form.

2's Complement adder-subtractor

The single system of a 4-bit adder cum 2's complement subtractor is shown in Figure. It operates between two 4-bit numbers: $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$. In case of subtraction: $y_3y_2y_1y_0$ is subtracted from $x_3x_2x_1x_0$.



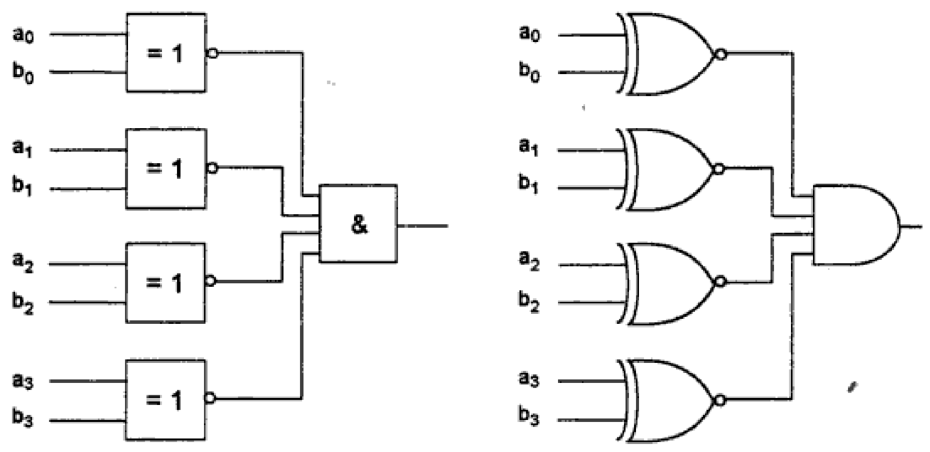
Operation 1- When control input M = 0

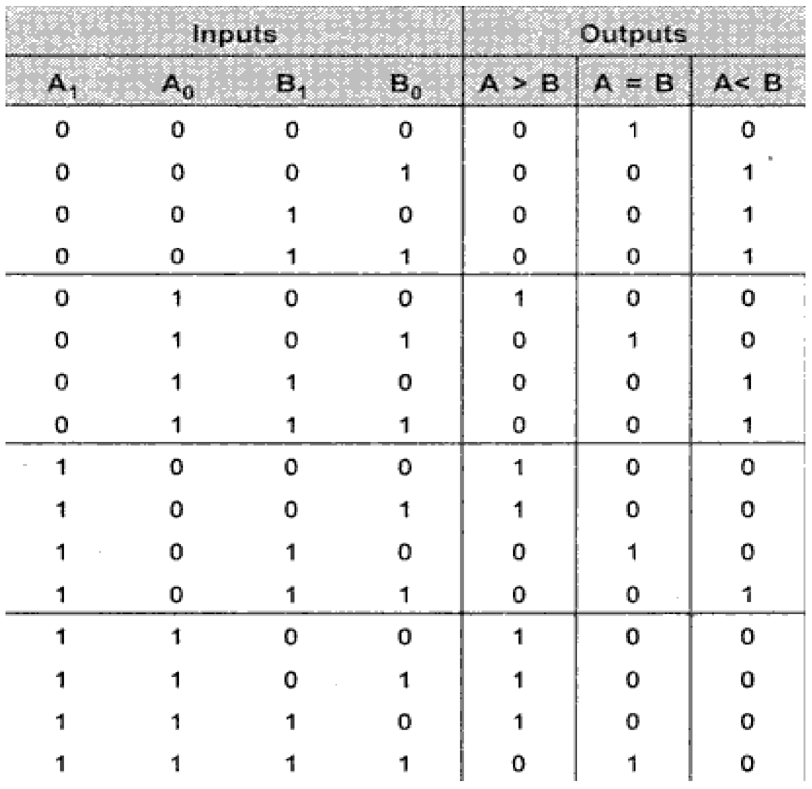
When control M = 0, the circuit will act as a 4-bit binary adder (since $y \oplus 0 = y$). Then, bits $y_3y_2y_1y_0$ pass through the XOR gates without inversion. Therefore, the unit adds the number

$x_3x_2x_1x_0$ to $y_3y_2y_1y_0$. The result appears as $S_3S_2S_1S_0$ in the binary form. It M=0, input carry= 0, and final carry output is neglected.

Operation 2 - When control input M = 1

When M = 1, this causes the XOR gate to act as an inverter for the y inputs to the full adders ($y\oplus1 = \overline{Y}$). The input to each full adder is x-y. If M = 1, the input carry = 1. The high M adds a 1 to the first full adder. This addition of 1 to the l's complement of y bits forms 2's complement of y. Then, the circuit performs the subtraction operation as per 2's complement. The result appears as $D_3D_2D_1D_0$ in binary form. The final carry output of the last full adder is not used or ignored.





Inputs				Outputs		
A_1	A_0	B_1	B_0	A > B	A = B	A< B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0