Boolean Analysis

Introduction

- 1854: Logical algebra was published by **George Boole** → known today as "Boolean Algebra"

  - **It's a convenient way and systematic way of expressing and analyzing the operation of logic circuits.**

- 1938: **Claude Shannon** was the first to apply Boole's work to the analysis and design of logic circuits.

Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.

Boolean algebra allows only two values— 0 and 1

Logic 0 can be: *false, off, low, no, open switch*.

Logic 1 can be: *true, on, high, yes, closed switch*.

Logic 0	Logic 1
False	True
Off	On
LOW	HIGH
No	Yes
Open switch	Closed switch

The three basic logic operations: OR, AND, and NOT.

Boolean Operations & Expressions

- *Variable* – a symbol used to represent a logical quantity.
- *Complement* – the inverse of a variable and is indicated by a bar over the variable.
- *Literal* – a variable or the complement of a variable.
- Constant: A constant has fix value 1 or 0 in the equations.
- *Operation Sign: (+)* for addition, (.) for multiplications and (-) bar over the binary number means inversion of that number.

Truth Tables

A truth table describes the relationship between the input and output of a logic circuit.

The number of entries corresponds to the number of inputs.

A 2-input table would have $2^2 = 4$ entries.

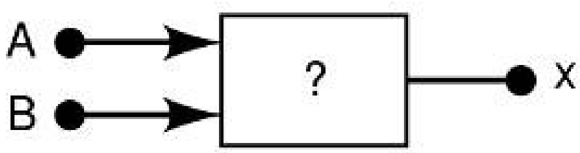
A 3-input table would have $2^3 = 8$ entries.

Examples of truth tables with 2, 3, and 4 inputs.

Output

Inputs

A	B	x
0	0	1
0	1	0
1	0	1
1	1	0



(a)

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(c)

OR Operation With OR Gates

- The Boolean expression for the **OR** operation is:

$$X = A + B$$ — Read as "*X* equals *A* OR *B*"

The + sign does *not* stand for ordinary addition—it stands for the OR operation

- The OR operation is similar to addition, but when A = 1 and B = 1, the OR operation produces:

$$1 + 1 = 1 \textit{ not } 1 + 1 = 2$$

In the Boolean expression x = 1 + 1 + 1 = 1…

x is true (1) when A is true (1) OR B is true (1) OR C is true (1)

OR Operation With OR Gates

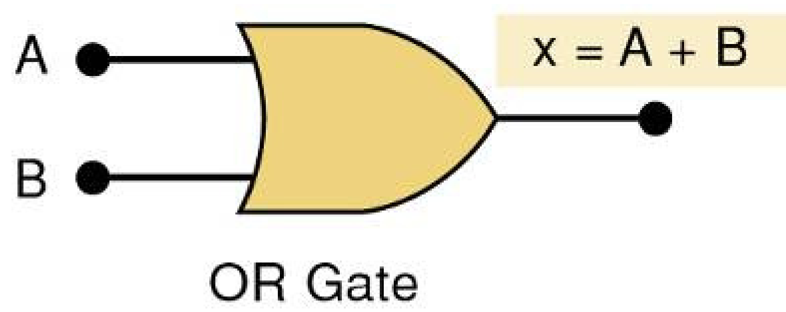
- An OR gate is a circuit with two or more inputs, whose output is equal to the OR combination of the inputs.

Truth table/circuit symbol for a two input OR gate.

OR

A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1

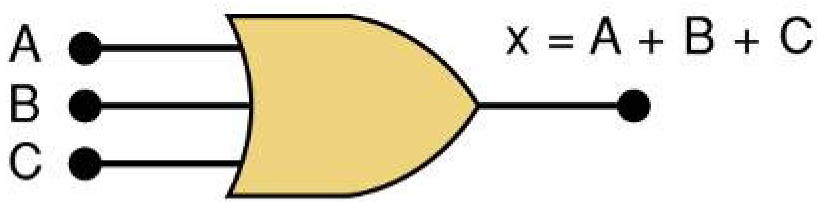
(a)



(b)

- An OR gate is a circuit with two or more inputs, whose output is equal to the OR combination of the inputs.

Truth table/circuit symbol for a three input OR gate.



A	B	C	$x = A + B + C$
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

Boolean Postulates

P1: A = 0 or A = 1
P2: 0 . 0 = 0
P3: 1 + 1 = 1
P4: 0 + 0 = 0
P5: 1 . 1 = 1
P6: 1 . 0 = 0 . 1 = 0
P7: 1 + 0 = 0 + 1 = 1

OR Gate Identity

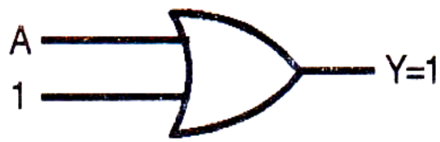
1. Identity

A+0 = A

A	0	Y(o/p)
0	0	0
1	0	1

2. Identity

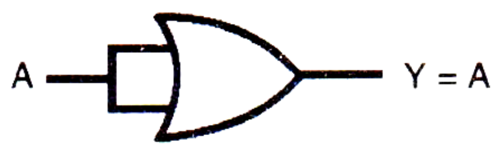
A + 1 = 1



A	1	Y
0	1	1
1	1	1

3. Identity

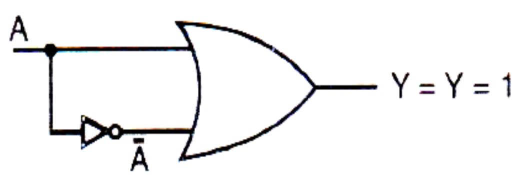
A + A = A



A	A	Y
0	0	0
1	1	1

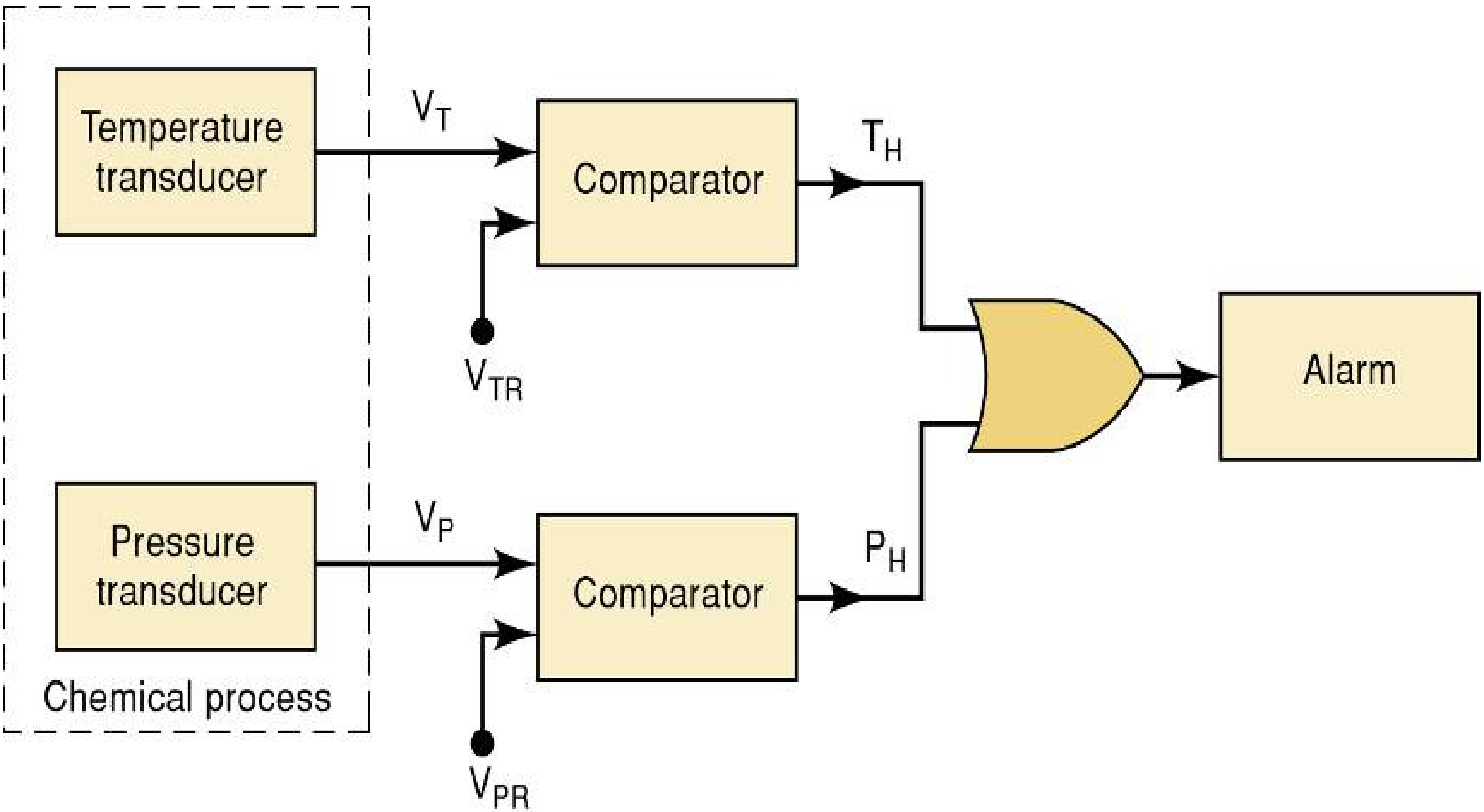
4. Identity

$A + \bar{A} = 1$



A	$\bar{A}$	Y
0	1	1
1	0	1

Example of the use of an OR gate in an alarm system.



AND Operations with AND gates

- The **AND** operation is similar to multiplication:

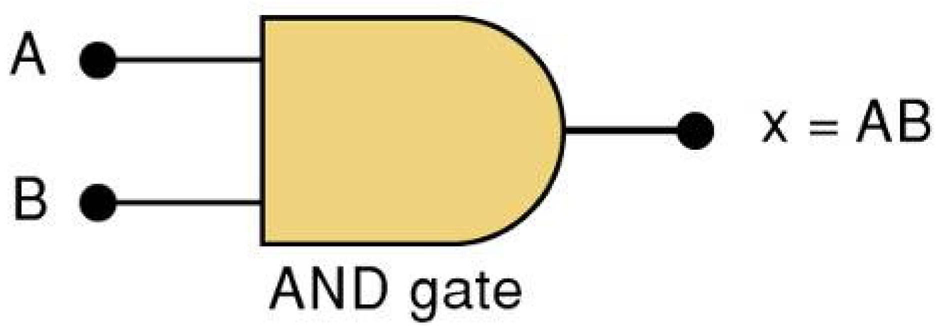
$$X = A \bullet B \bullet C$$ — Read as "*X* equals *A* AND *B* AND *C*"

The • sign does *not* stand for ordinary multiplication—it stands for the AND operation.

x is true (1) when A AND B AND C are true (1)

AND

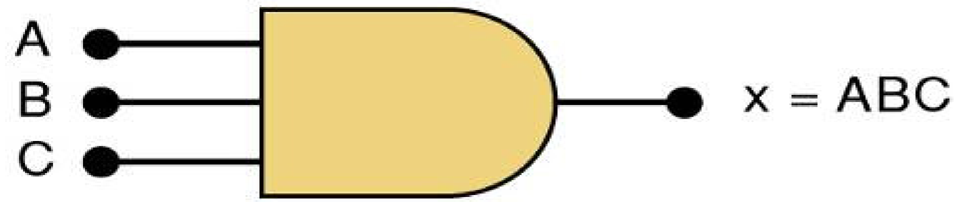
A	B	$x = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



Truth table — Gate symbol.

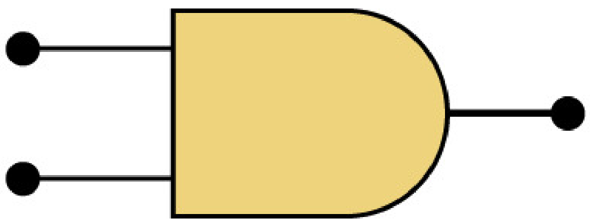
Truth table/circuit symbol for a three input AND gate.

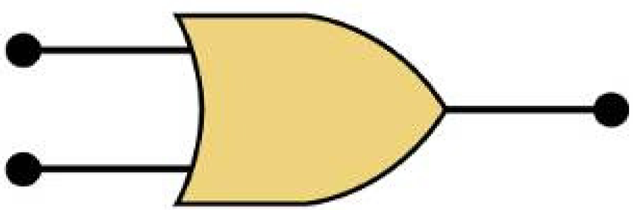
A	B	C	x = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



AND / OR

The AND symbol on a logic-circuit diagram tells you output will go HIGH *only* when *all* inputs are HIGH.



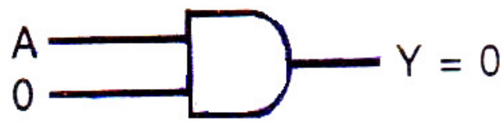


The OR symbol means the output will go HIGH when *any* input is HIGH.

AND Gate Identity

5. Identity

$A.0 = 0$



A	0	Y
0	0	0
1	0	0

6. Identity

$A.1 = A$

A	1	Y
0	1	0
1	1	1

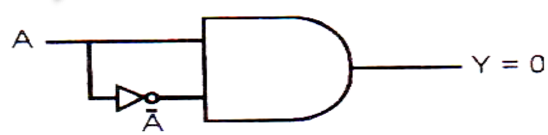
7. Identity

$A.A = A$

A	A	Y
0	0	0
1	1	1

8. Identity

$A.\bar{A} = 0$



A	$\bar{A}$	Y
0	1	0
1	0	0

NOT Operation

- The Boolean expression for the **NOT** operation:

$X = \overline{A}$ — Read as: "*X* equals NOT *A*"

"*X* equals the *inverse* of *A*"

"*X* equals the *complement* of *A*"

The overbar represents the NOT operation.

$A' = \overline{A}$

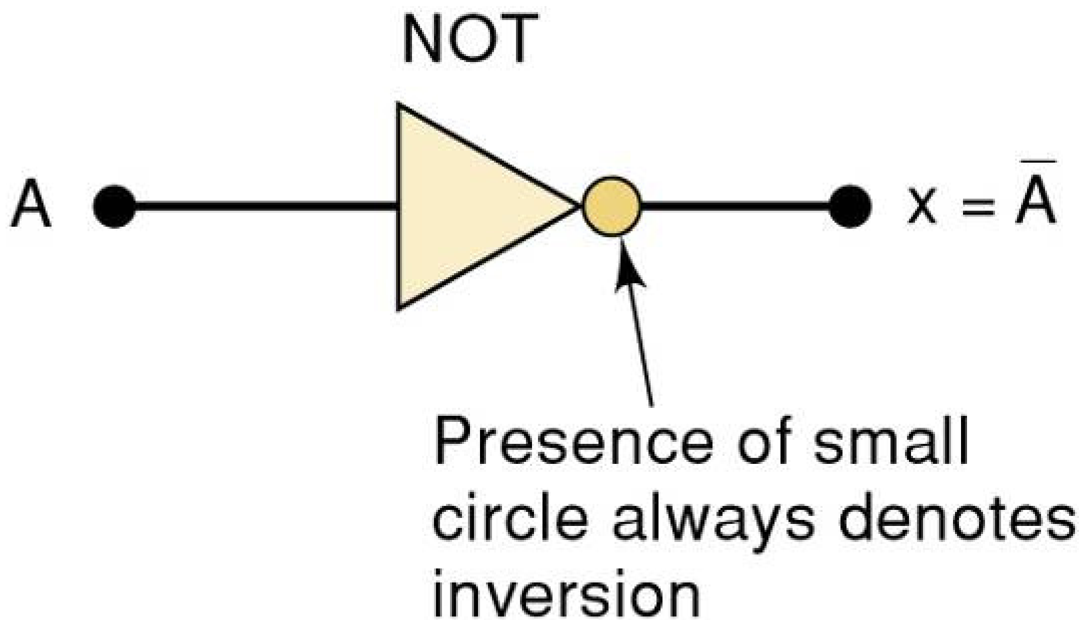
Another indicator for inversion is the prime symbol (').

NOT

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

NOT Truth Table

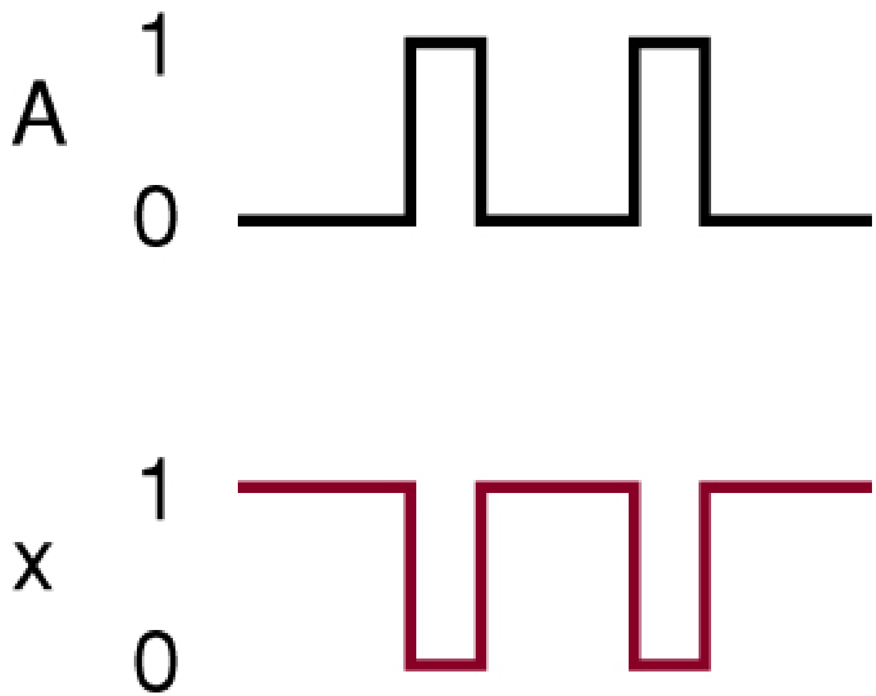
A NOT circuit—commonly called an INVERTER.



This circuit *always* has only a single input, and the out-put logic level is always *opposite* to the logic level of this input.

NOT Operation

The INVERTER inverts (*complements*) the input signal at all points on the waveform.

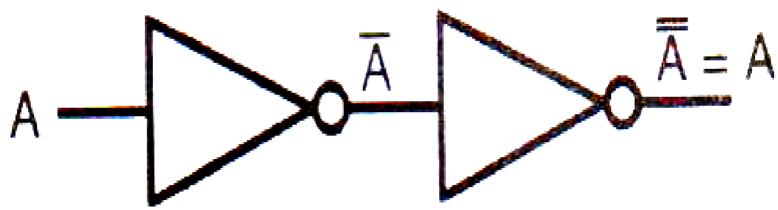


Whenever the input = 0, output = 1, and vice versa.

NOT Gate Identity

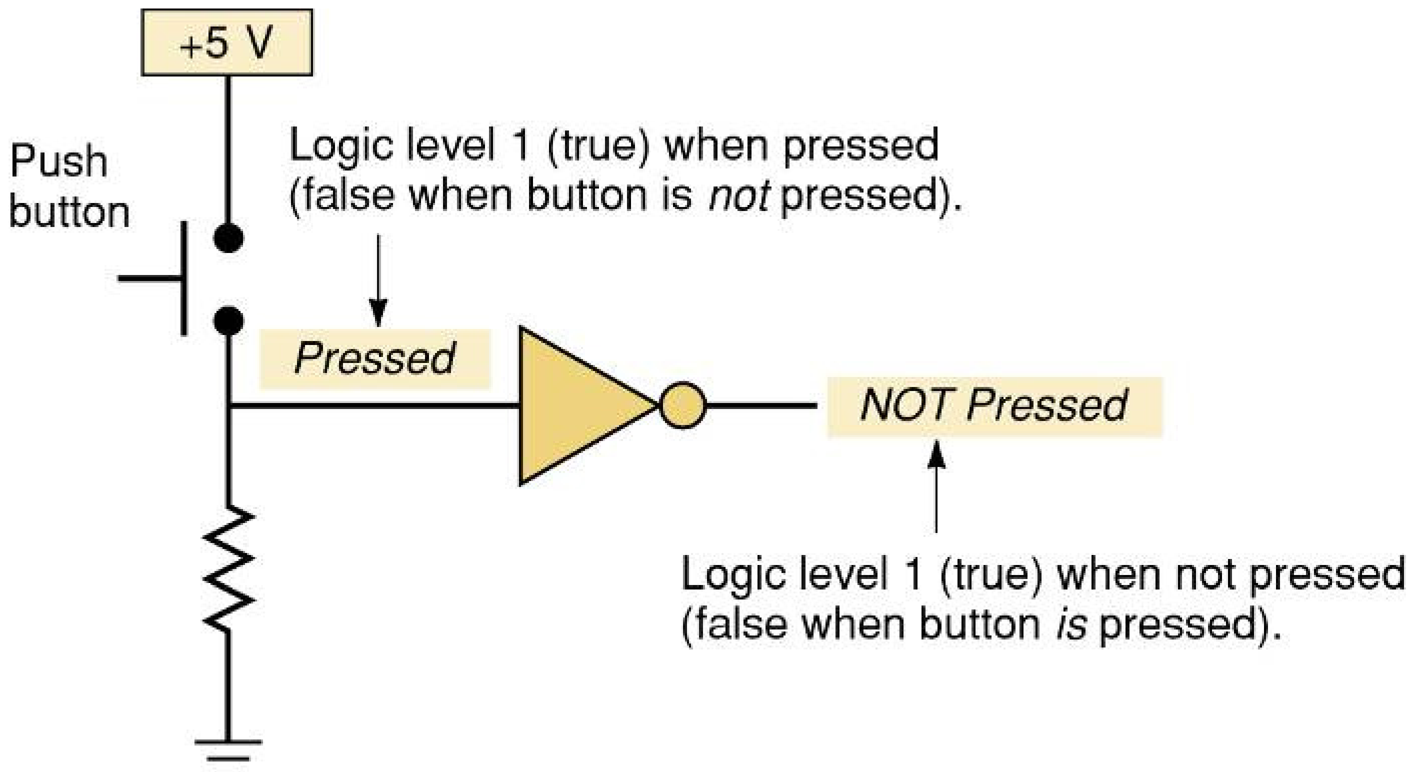
9. Identity

$$\bar{\bar{A}} = A$$



A	$\bar{A}$	$\bar{\bar{A}}$	Y
0	1	0	0
1	0	1	1

Typical application of the NOT gate.



This circuit provides an expression that is true when the button is not pressed.

Boolean Operations

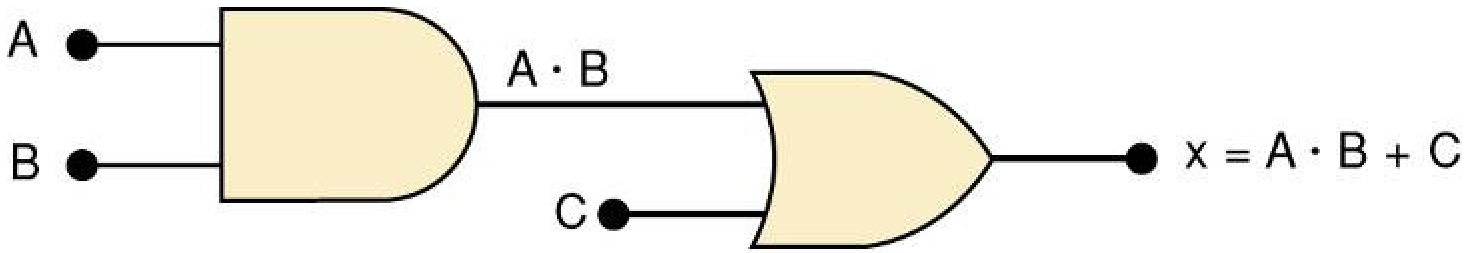
Summarized rules for OR, AND and NOT

OR	*AND*	*NOT*
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\overline{0} = 1$
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\overline{1} = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	

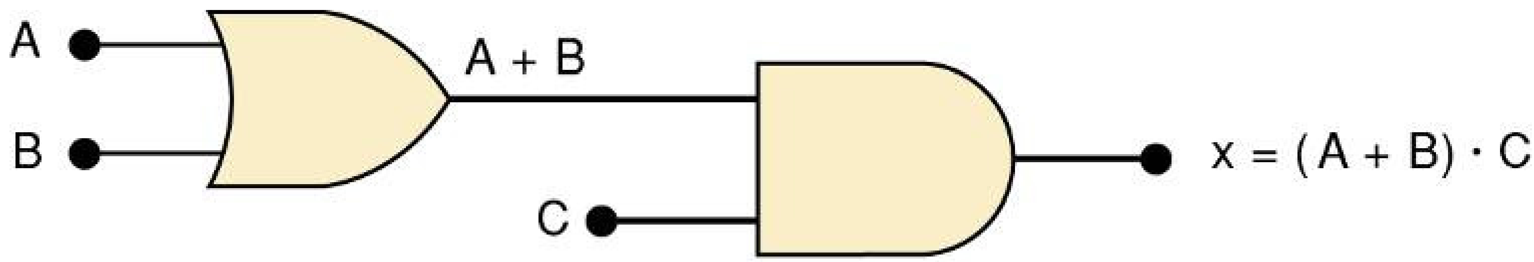
These three basic Boolean operations can describe any logic circuit.

Describing Logic Circuits Algebraically

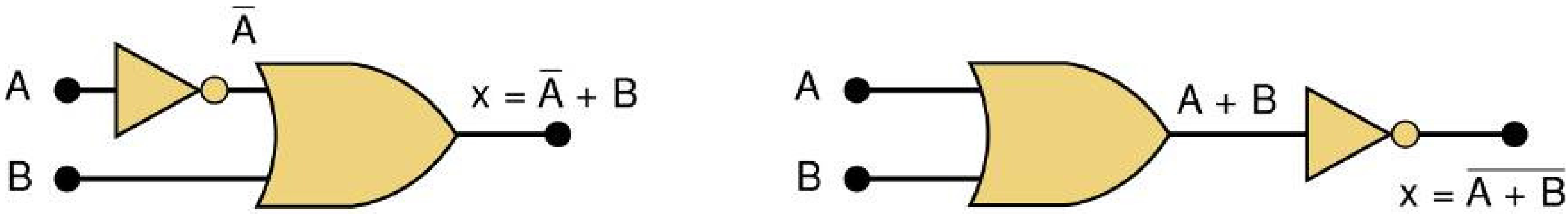
- If an expression contains both **AND** and **OR** gates, the **AND** operation will be performed first.



- Unless there is a parenthesis in the expression.

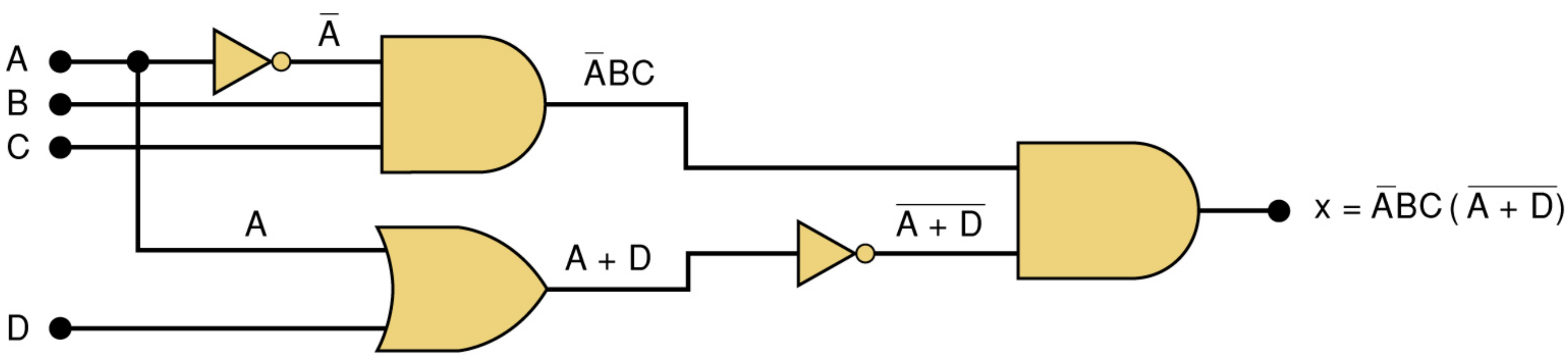


- Whenever an INVERTER is present, output is equivalent to input, with a bar over it.
  - Input *A* through an inverter equals $\overline{A}$.



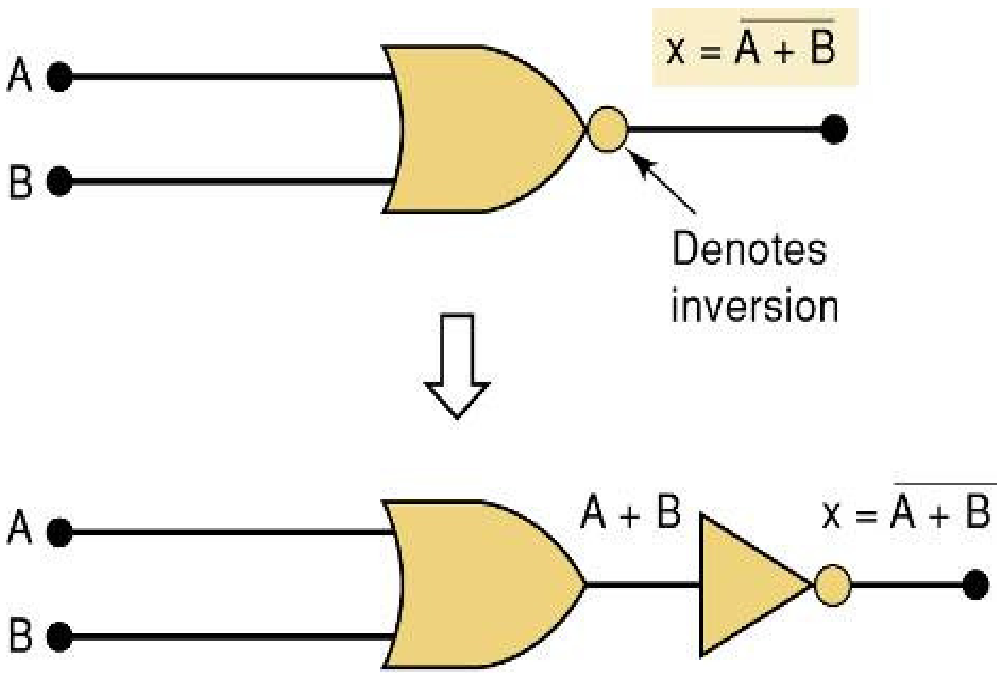
(a) (b)

- Further examples…



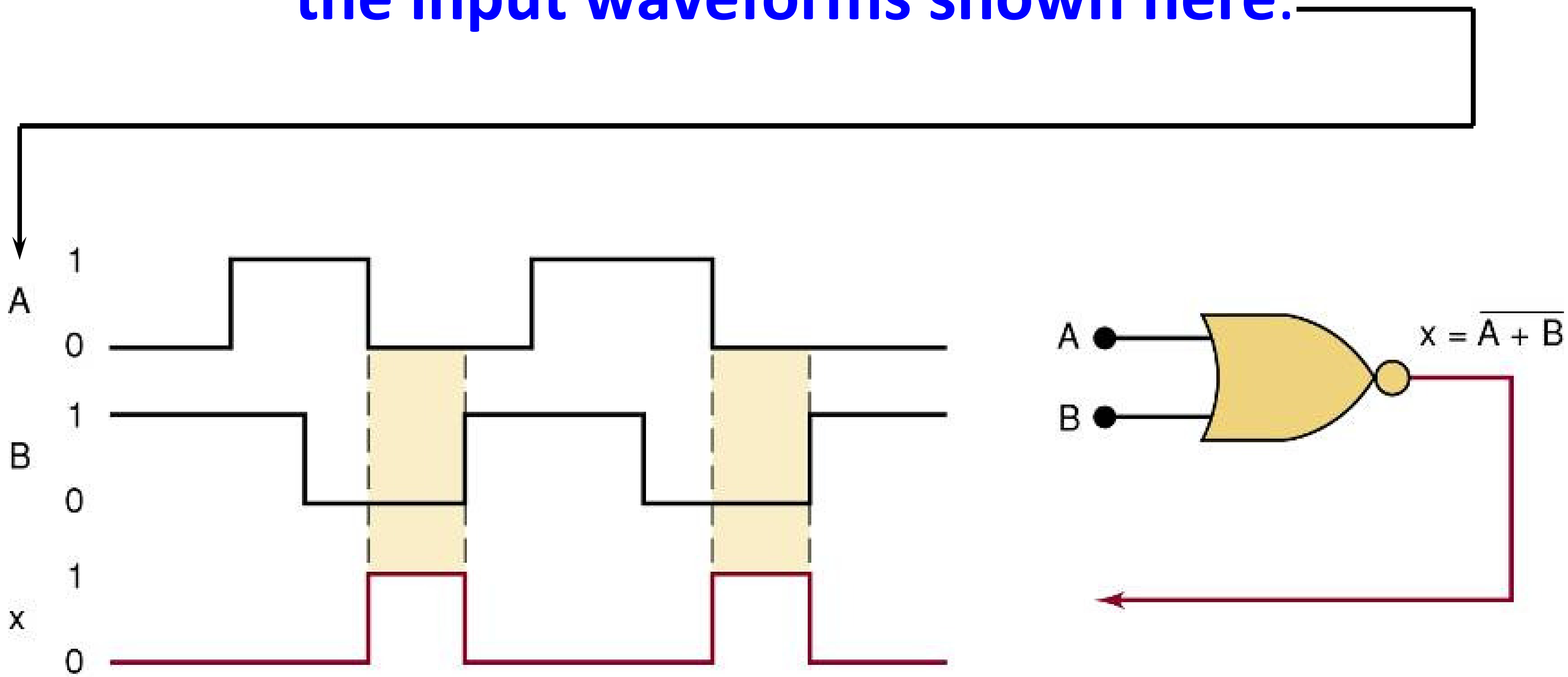
NOR Gates and NAND Gates

- The **NOR** gate is an inverted **OR** gate.
  - An inversion “bubble” is placed at the output of the OR gate, making the Boolean output expression $x = \overline{A + B}$



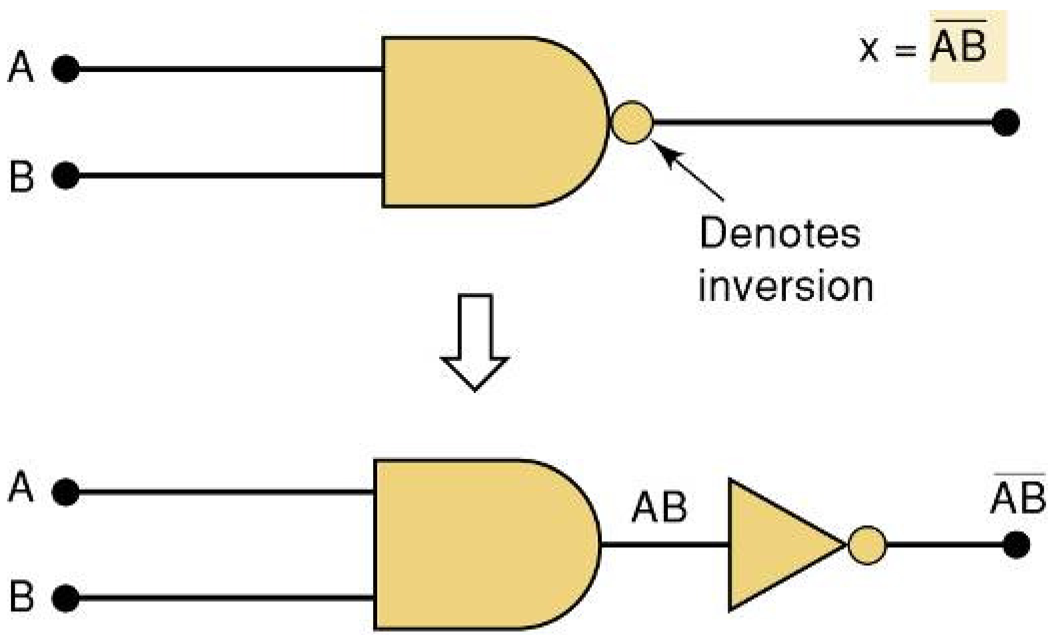
A	B	OR: $A + B$	NOR: $\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Output waveform of a NOR gate for the input waveforms shown here.



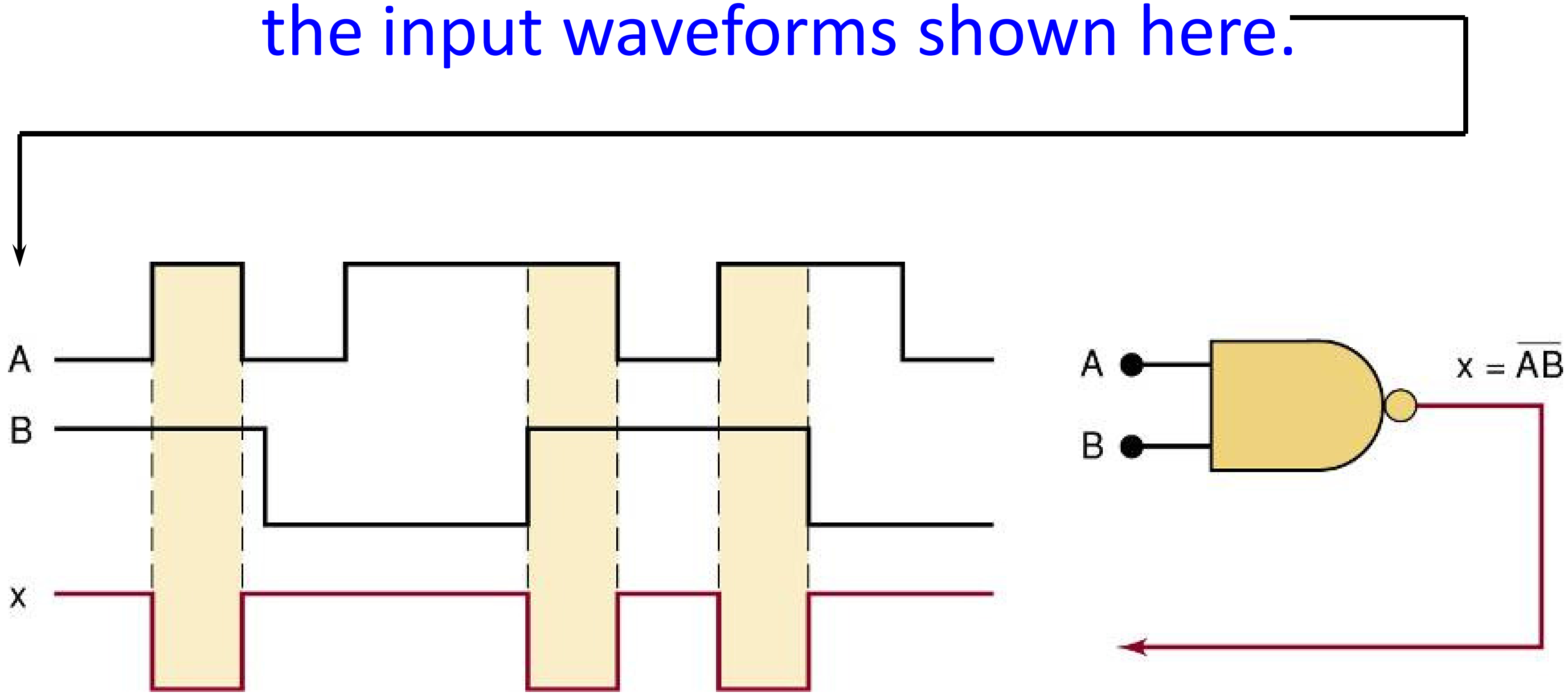
AND Gates and NAND Gates

- The NAND gate is an inverted AND gate.
  - An inversion “bubble” is placed at the output of the AND gate, making the Boolean output expression $x = \overline{AB}$



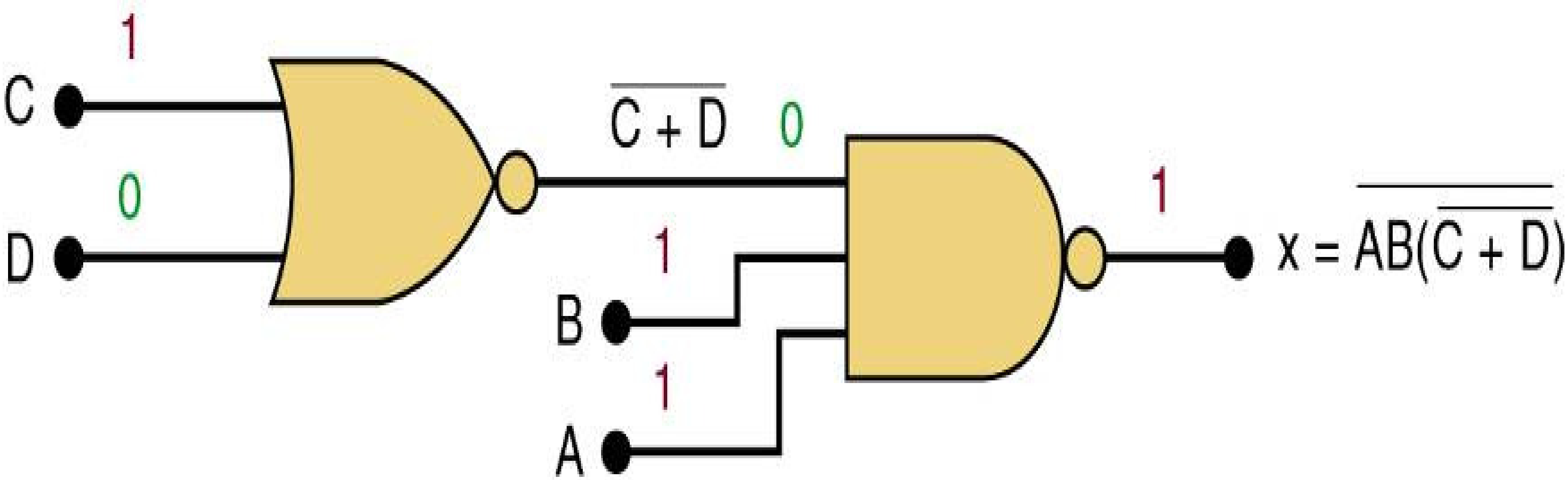
		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Output waveform of a **NAND** gate for the input waveforms shown here.



NOR Gates and NAND Gates

Logic circuit with the expression $x = \overline{AB \bullet (\overline{C + D})}$ using only NOR and NAND gates.



Evaluating Logic Circuit Outputs

- **Rules for evaluating a Boolean expression:**
  - Perform all inversions of single terms.
  - Perform all operations within parenthesis.
  - Perform **AND** operation before an **OR** operation unless parenthesis indicate otherwise.
  - If an expression has a bar over it, perform operations inside the expression, and then invert the result.
- **The best way to analyze a circuit made up of multiple logic gates is to use a truth table.**
  - It allows you to analyze one gate or logic combination at a time.
  - It allows you to easily double-check your work.
  - When you are done, you have a table of tremendous benefit in troubleshooting the logic circuit.
- Combine basic **AND**, **OR**, and **NOT** operations. Simplifying the writing of Boolean expressions
- Output of **NAND** and **NOR** gates may be found by determining the output of an **AND** or **OR** gate, and inverting it.
- The truth tables for **NOR** and **NAND** gates show the complement of truth tables for **OR** and **AND** gates.

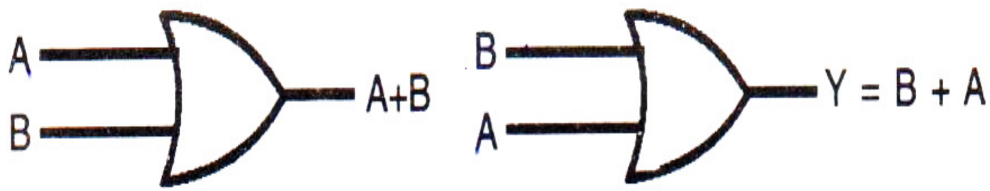
Laws & Rules of Boolean Algebra

- The basic laws of Boolean algebra:
  - The **commutative** laws
  - The **associative** laws
  - The **distributive** laws

Commutative Laws

Commutative Law of Addition

$$A + B = B + A$$

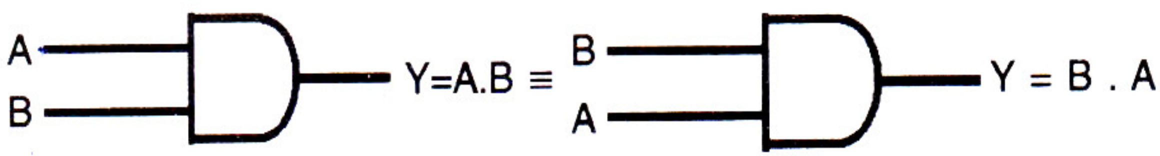


A	B	A+B	B+A
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Commutative Law of Multiplications

$$A . B = B . A$$

The equation can be realized using AND tages.

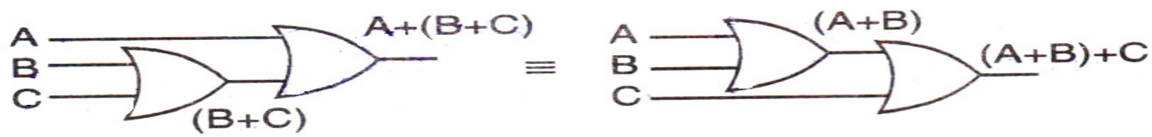


A	B	A.B	B.A
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Associative Laws

Associative Law of Addition

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

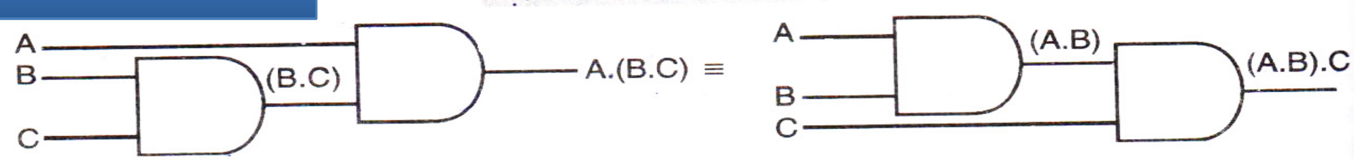


Varification of the law by perfect induction method.

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

Associative Law of Multiplications

$$A . (B.C) = (A.B) . C$$



Varification of the low by perfect induction method.

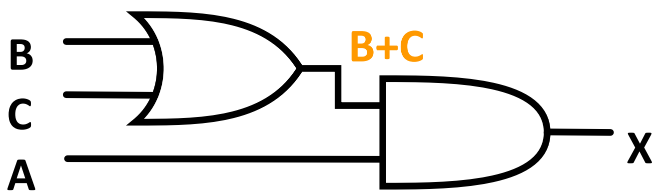
A	B	C	(B.C)	A.B	A.(B.C)	(A.B).C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

Distributive Law

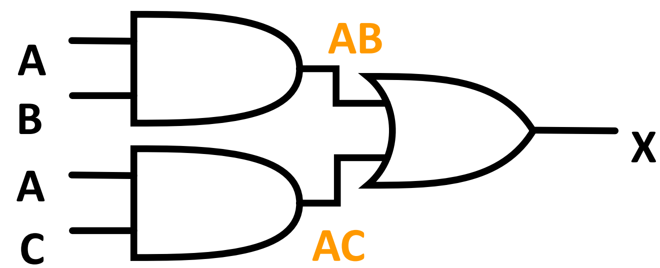
The *distributive law* is written for 3 variables as follows:

(A)

A(B+C)=AB+AC



≡

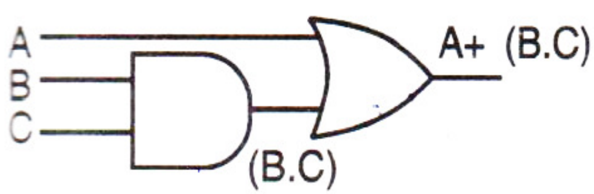


X=A(B+C)

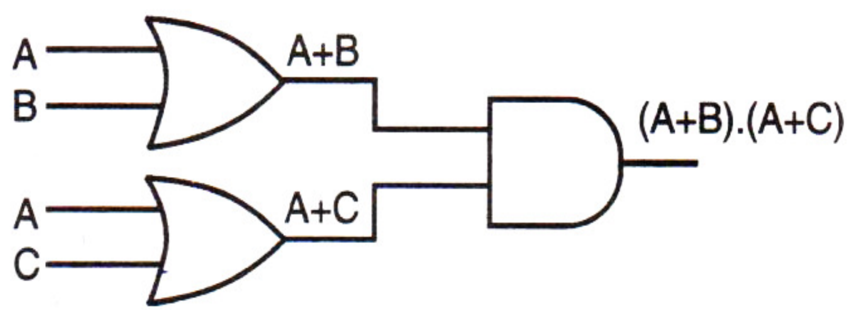
X=AB+AC

(B)

A+(B.C)=(A+B).(A+C)



≡



Duality

The dual of any statement in a Boolean algebra is the statement obtained by interchanging (+) and (.), and simultaneously inter-changing the elements 0 and 1 in the statement.

$$(+) \leftrightarrow (.)$$

$$1 \leftrightarrow 0$$

The AND and OR gate identity are dual , the table illustratrates the duality property of these identities.

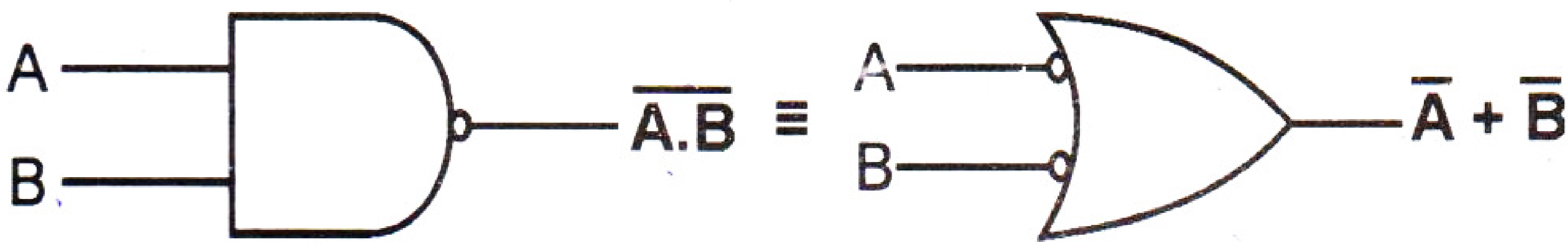
Additive	Multiplicative
$A+0 = A$	$A.1=A$
$A+1= 1$	$A.0=0$
$A+A = A$	$A.A=A$
$A+\overline{A} = 1$	$A.\overline{A}=0$
$A+B = B+A$	$A.B = B.A$
$A+(B+C) = (A+B)+C$	$A.(B.C) = (A.B).C$
$A.(B+C) = A.B + A.C$	$A+(B.C)=(A+B).(A+C)$
$A+(\overline{A}.B) = A+B$	$A.(\overline{A}+B) = A.B$

DeMorgan's Theorem

Augustus De Morgan (1806–1871) a mathematician developed a pair of important rules regarding group complementation in Boolean algebra. The group complementation here means complimenting a group of terms, represented by a long bar over more than one variable.

I **Rule:** When the AND product of two or more variables are inverted, this is the same as inverting each variable individually and then ORing these inverted variables.

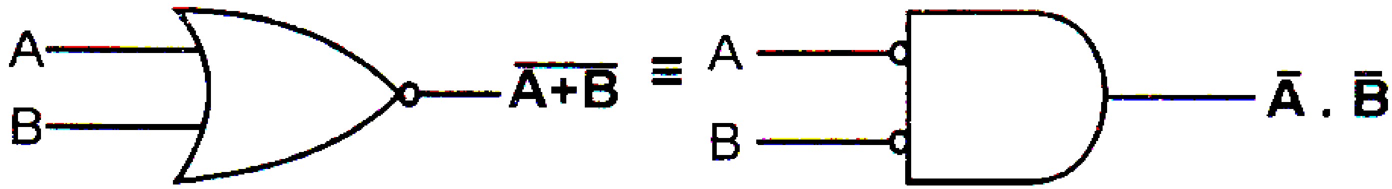
$$\overline{AB} = \overline{A} + \overline{B}$$



From the equation we can say that the output of a NAND gate is equivalent to an bubbled input OR gate

II Rule: When the OR sum of two or more variables is inverted, this is the same as inverting each variable individually and then ANDing them.

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



How to de-morganize a Boolean Expression ?

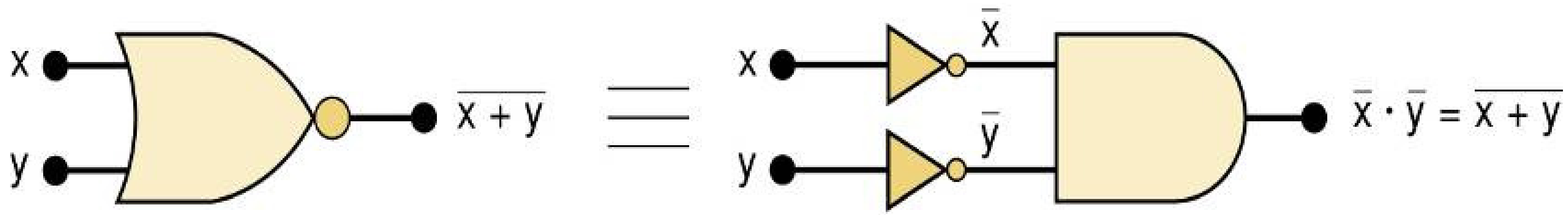
Following procedure is adopted for de-morganizing the Boolean expression:

1. Break the expression using rule I and II.
2. When multiple bars exist in an expression, break one bar at a time, and it is easier to begin simplification by breaking the longest (uppermost) bar first.
3. One should never break more than one bar in a single step otherwise it may not give correct answer.

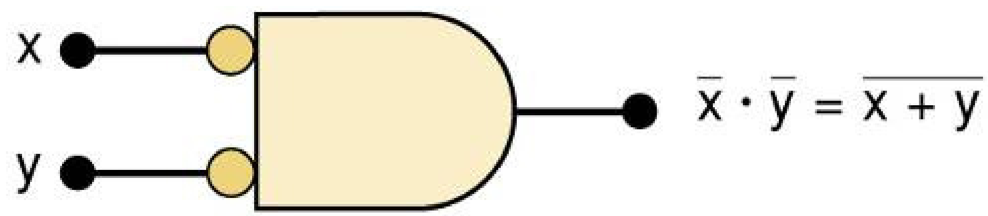
DeMorgan's Theorems

Equivalent circuits implied by Theorem

$$(16) \quad (\overline{x + y}) = \overline{x} \cdot \overline{y}$$



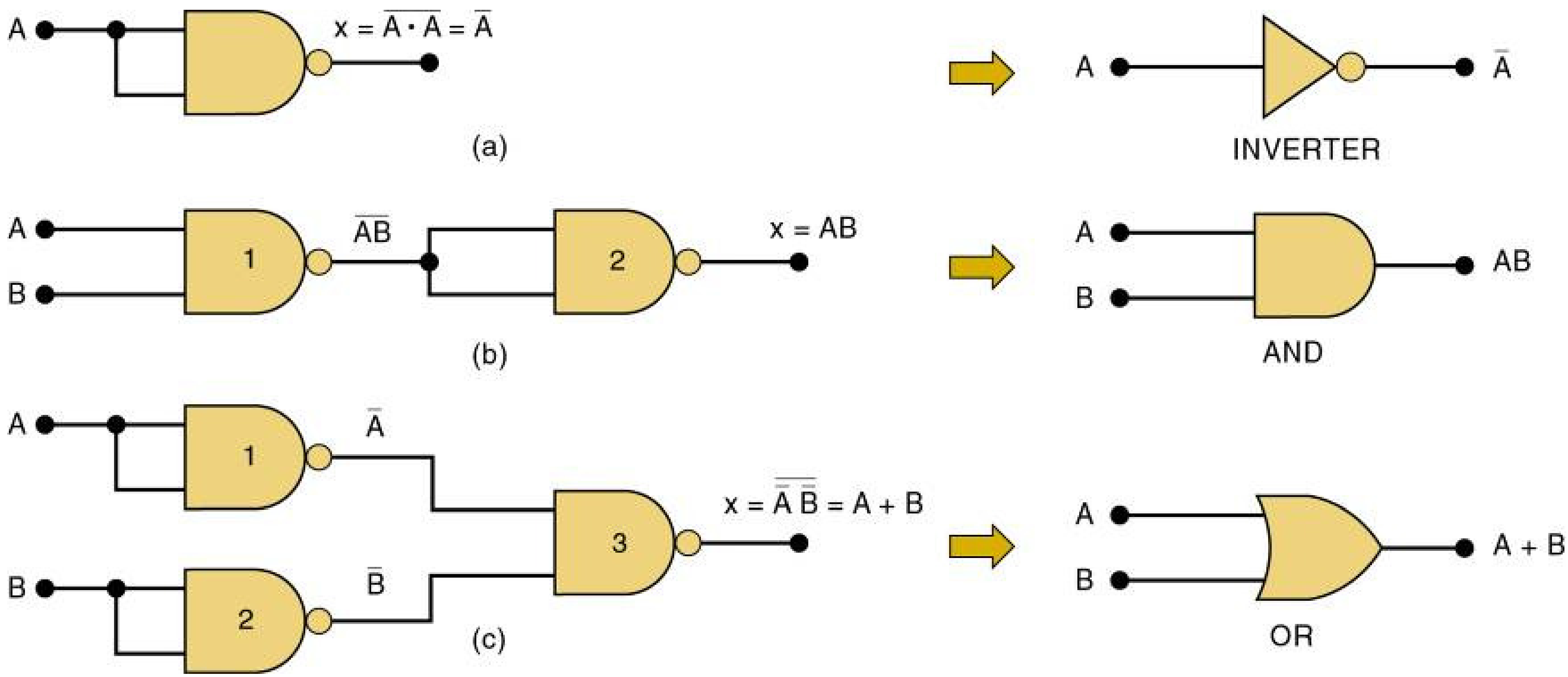
The alternative symbol
for the NOR function.



Universality of NAND and NOR Gates

- **NAND** or **NOR** gates can be used to create the three basic logic expressions.
  - **OR**, **AND**, and INVERT.
    - Provides flexibility—very useful in logic circuit design.

How combinations of NANDs or NORs are used to create the three logic functions.



It is possible, however, to implement any logic expression using *only* NAND gates and no other type of gate, as shown.