

# Chapter 3: Describing Logic Circuits

Dr. Aloke Kumar Saha

Professor

Department of CSE, UAP

## Chapter 3 Objectives

- *Selected areas covered in this chapter:*
  - Operation of truth tables for **AND**, **NAND**, **OR**, and **NOR** gates, and the **NOT** (INVERTER) circuit.
  - Boolean expression for logic gates.
  - DeMorgan's theorems to simplify logic expressions.
  - Universal gates (**NAND** or **NOR**) to implement a circuit represented by a Boolean expression.
  - Concepts of active-LOW & active-HIGH logic signals.
  - Describing and measuring propagation delay time.
  - Differences between an HDL and a computer programming language.

## 3-1 Boolean Constants and Variables

- 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**.

## 3-2 Truth Tables

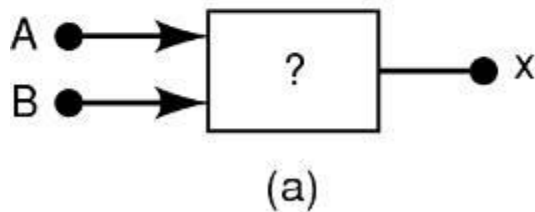
- A truth table describes the relationship between the input and output of a logic circuit.
- The number of entries corresponds to the number of inputs.
  - A 2-input table would have  $2^2 = 4$  entries.
  - A 3-input table would have  $2^3 = 8$  entries.

## 3-2 Truth Tables

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

Diagram illustrating a 2-input logic function. Inputs A and B are shown with arrows pointing into a box labeled 'Output'. The output is labeled 'x'.

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



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)

### 3-3 OR Operation With OR Gates

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

$$X = A + B \text{ — Read as “} X \text{ equals } A \text{ OR } B \text{”}$$

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:

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

In the Boolean expression  $x = 1 + 1 + 1 = 1 \dots$   
*x is true (1) when A is true (1) **OR** B is true (1) **OR** C is true (1)*

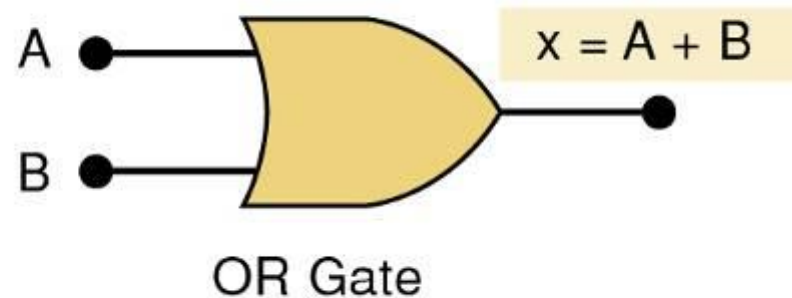
### 3-3 OR Operation With OR Gates

- 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

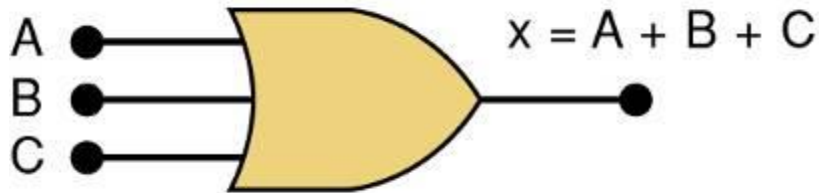
A	B	$x = A + B$
0	0	0
0	1	1
1	0	1
1	1	1



### 3-3 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 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



### 3-4 AND Operations with AND gates

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

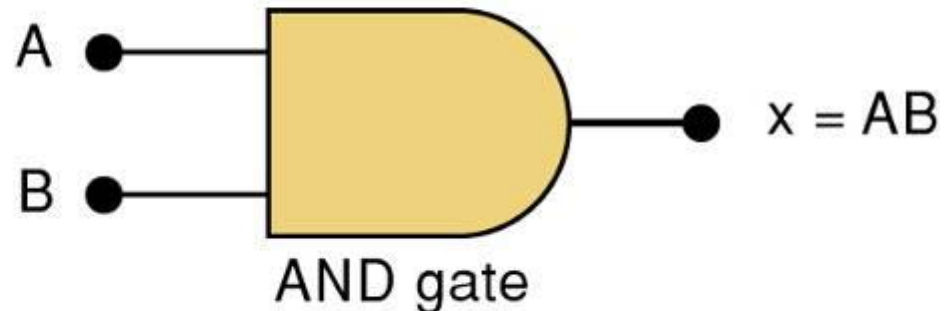
$$X = A \cdot B \cdot C \text{ — Read as “} X \text{ equals } A \text{ AND } B \text{ AND } C\text{”}$$

The  $\cdot$  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**

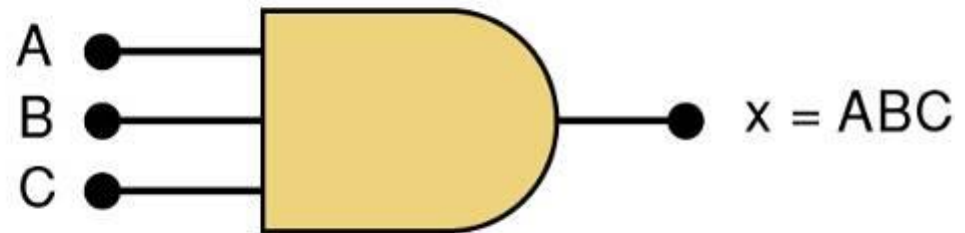


**Gate symbol.**

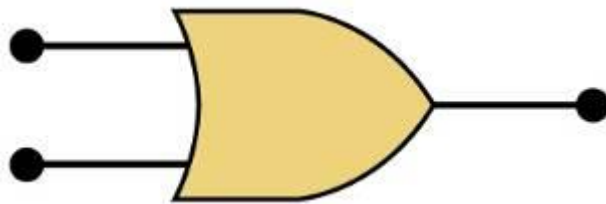
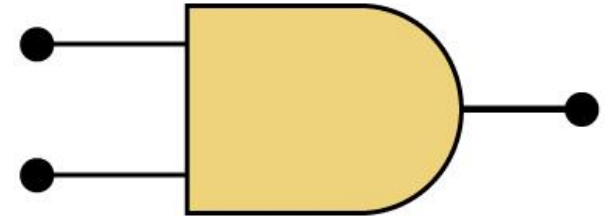
### 3-4 AND Operations with AND gates

**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



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.

## 3-5 NOT Operation

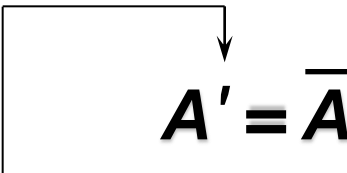
- The Boolean expression for the **NOT** operation:


$$X = \overline{A} \text{ — Read as: “} X \text{ equals NOT } A \text{”}$$

The overbar represents the **NOT** operation.

“*X* equals the *inverse* of *A*”

“*X* equals the *complement* of *A*”


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

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

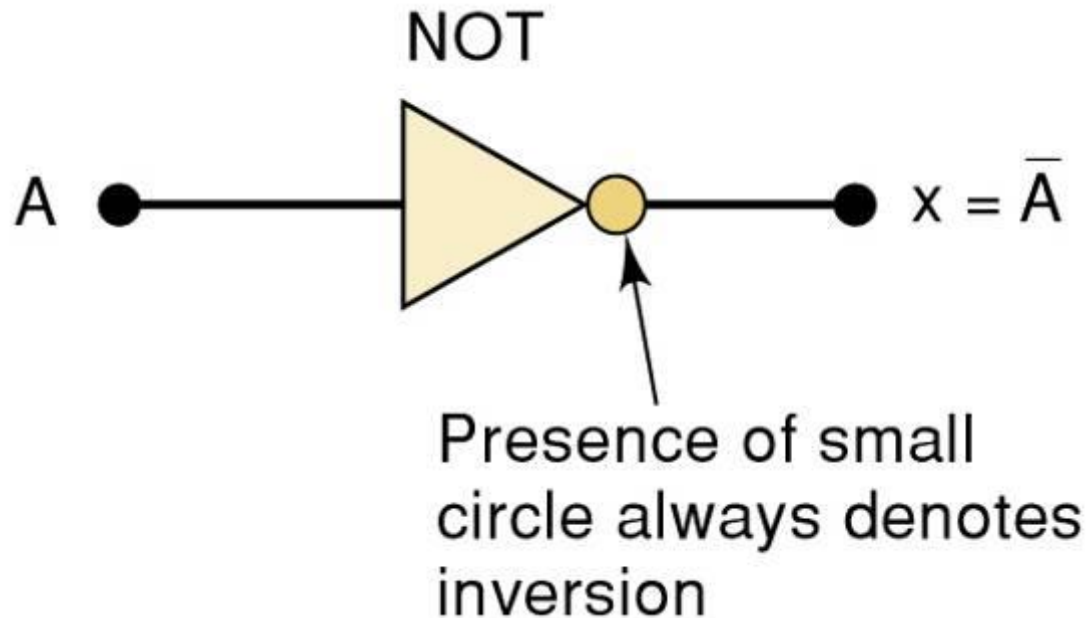
NOT

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

**NOT Truth Table**

## 3-5 NOT Operation

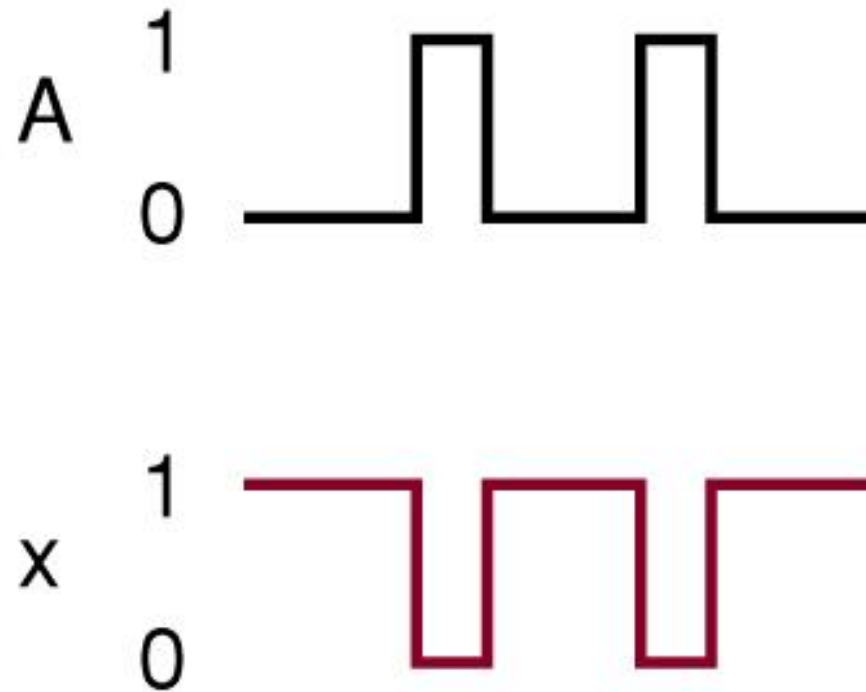
**A NOT circuit—commonly called an INVERTER.**



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

## 3-5 NOT Operation

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



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

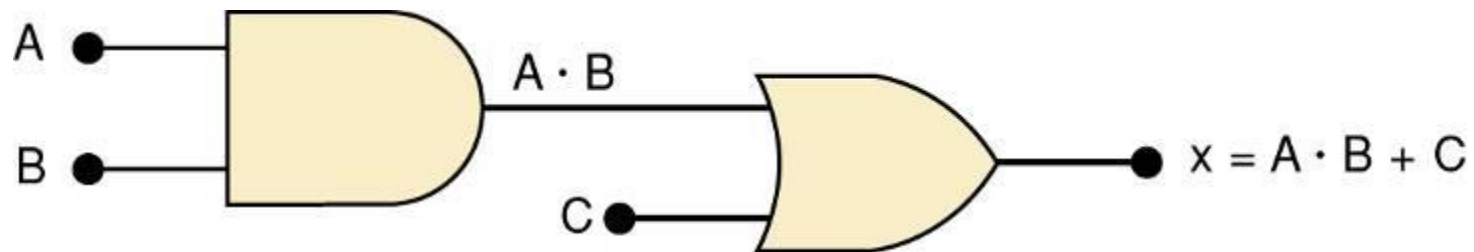
## Summarized rules for OR, AND and NOT

<i>OR</i>	<i>AND</i>	<i>NOT</i>
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\overline{0} = 1$
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\overline{1} = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	

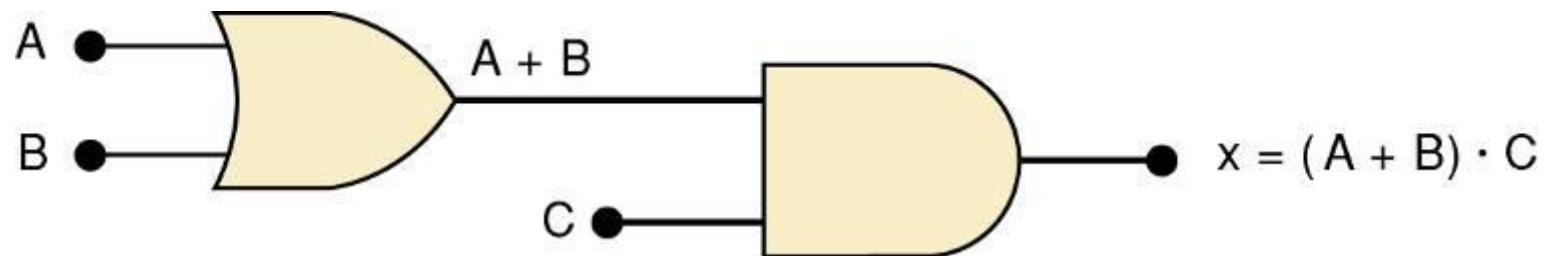
**These three basic Boolean operations  
can describe any logic circuit.**

### 3-6 Describing Logic Circuits Algebraically

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



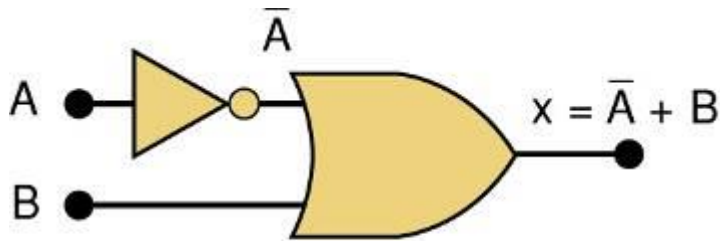
- Unless there is a parenthesis in the expression.



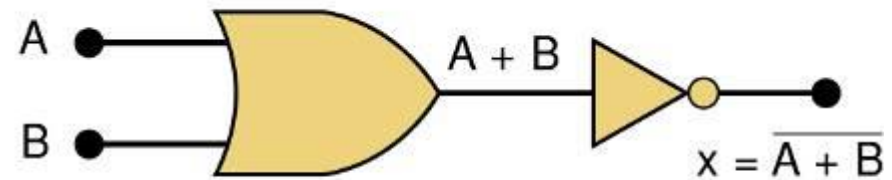


### 3-6 Describing Logic Circuits Algebraically

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



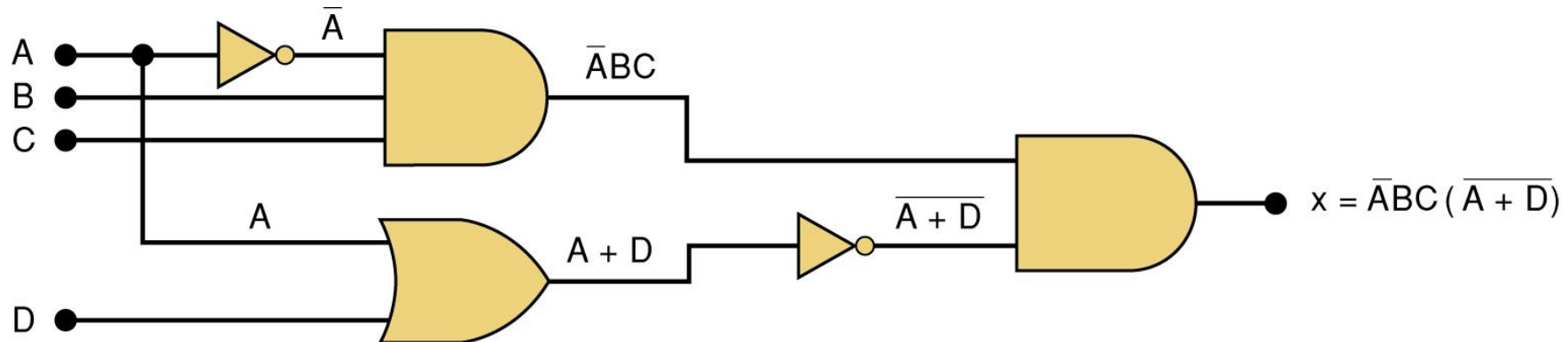
(a)



(b)

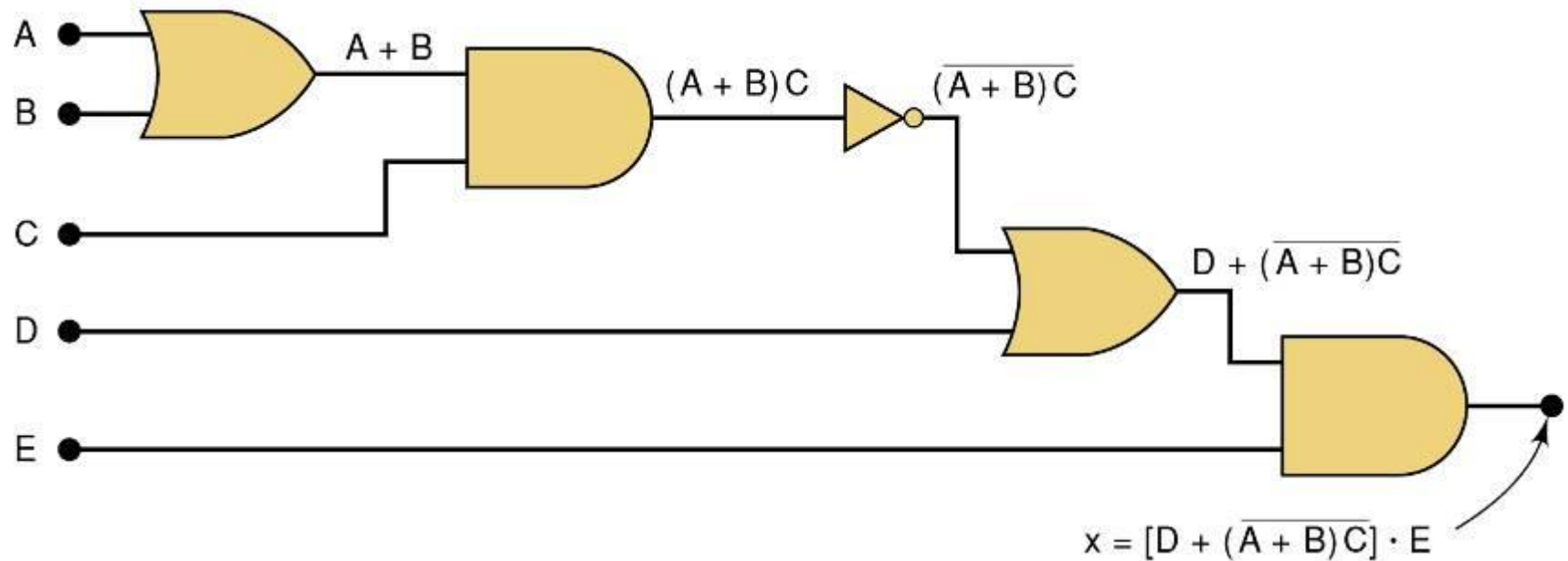
## 3-6 Describing Logic Circuits Algebraically

- Further examples...



## 3-6 Describing Logic Circuits Algebraically

- Further examples...

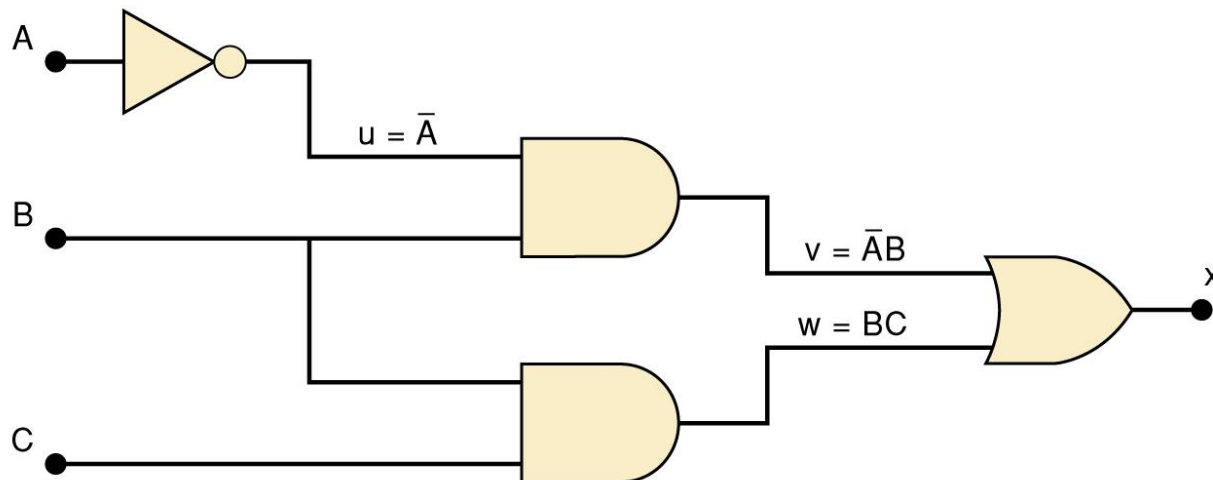


## 3-7 Evaluating Logic Circuit Outputs

- 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.

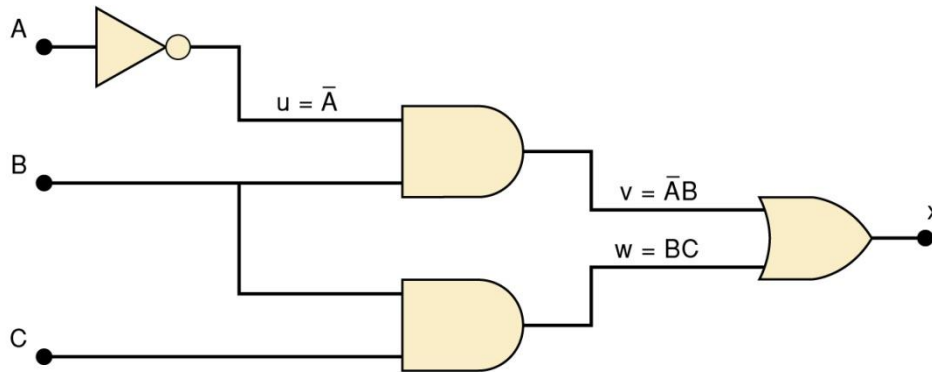
## 3-7 Evaluating Logic Circuit Outputs

- 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.



## 3-7 Evaluating Logic Circuit Outputs

- The first step after listing all input combinations is to create a column in the truth table for each intermediate signal (node).

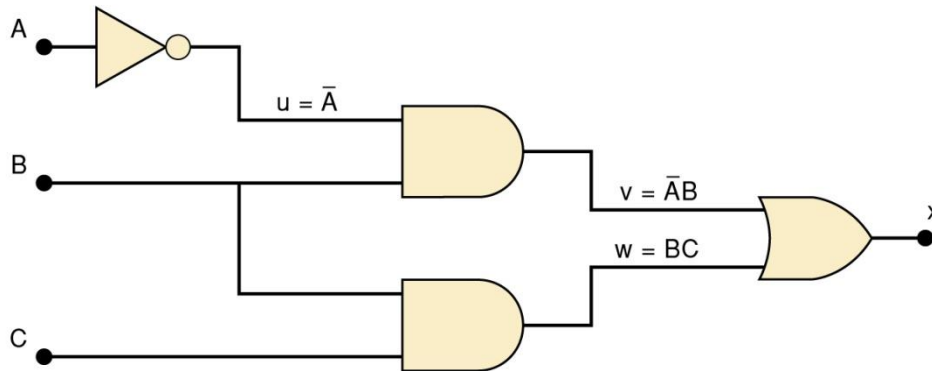


A	B	C	$u = \bar{A}$	$v = \bar{A}B$	$w = BC$	$x = v + w$
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	0			
1	0	1	0			
1	1	0	0			
1	1	1	0			

Node  $u$  has been filled as the complement of  $A$

### 3-7 Evaluating Logic Circuit Outputs

- The next step is to fill in the values for column  $v$ .

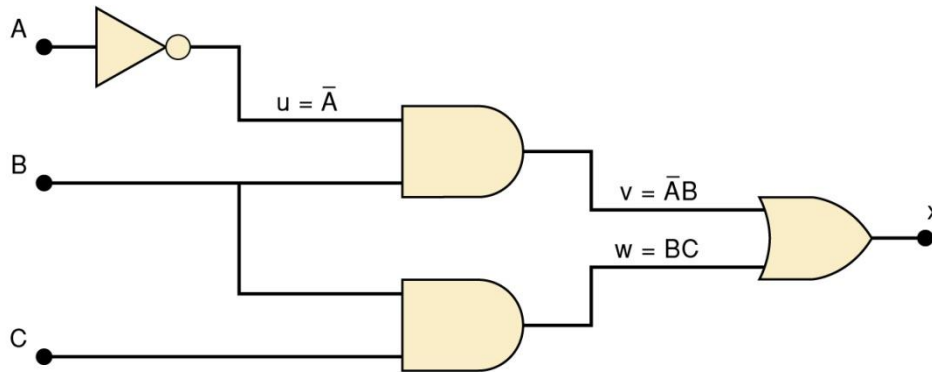


A	B	C	$u = \bar{A}$	$v = \bar{A}B$	$w = BC$	$x = v + w$
0	0	0	1	0		
0	0	1	1	0		
0	1	0	1	1		
0	1	1	1	1		
1	0	0	0	0		
1	0	1	0	0		
1	1	0	0	0		
1	1	1	0	0		

$v = \bar{A}B$  — Node  $v$  should be HIGH when A (node  $u$ ) is HIGH **AND** B is HIGH

### 3-7 Evaluating Logic Circuit Outputs

- The third step is to predict the values at node  $w$  which is the logical product of  $BC$ .



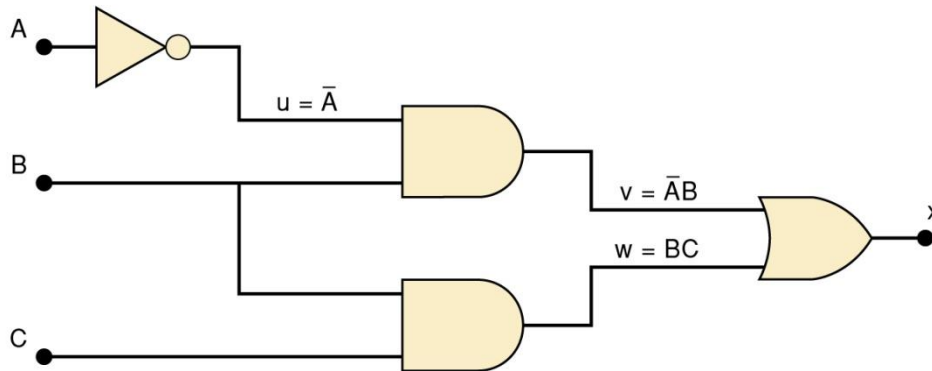
A	B	C	$u = \bar{A}$	$v = \bar{A}B$	$w = BC$	$x = v + w$
0	0	0	1	0	0	
0	0	1	1	0	0	
0	1	0	1	1	0	
0	1	1	1	1	1	
1	0	0	0	0	0	
1	0	1	0	0	0	
1	1	0	0	0	0	
1	1	1	0	0	1	

This column is HIGH whenever  $B$  is HIGH **AND**  $C$  is HIGH



### 3-7 Evaluating Logic Circuit Outputs

- The final step is to logically combine columns  $v$  and  $w$  to predict the output  $x$ .



A	B	C	$\bar{u} = \bar{A}$	$\bar{v} = \bar{A}B$	$w = BC$	$x = v + w$
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

Since  $x = v + w$ , the  $x$  output will be HIGH when  $v$  **OR**  $w$  is HIGH

## ● 3-7 Evaluating Logic Circuit Outputs

- Output logic levels can be determined directly from a circuit diagram.
  - Output of each gate is noted until final output is found.
    - Technicians frequently use this method.

### 3-7 Evaluating Logic Circuit Outputs

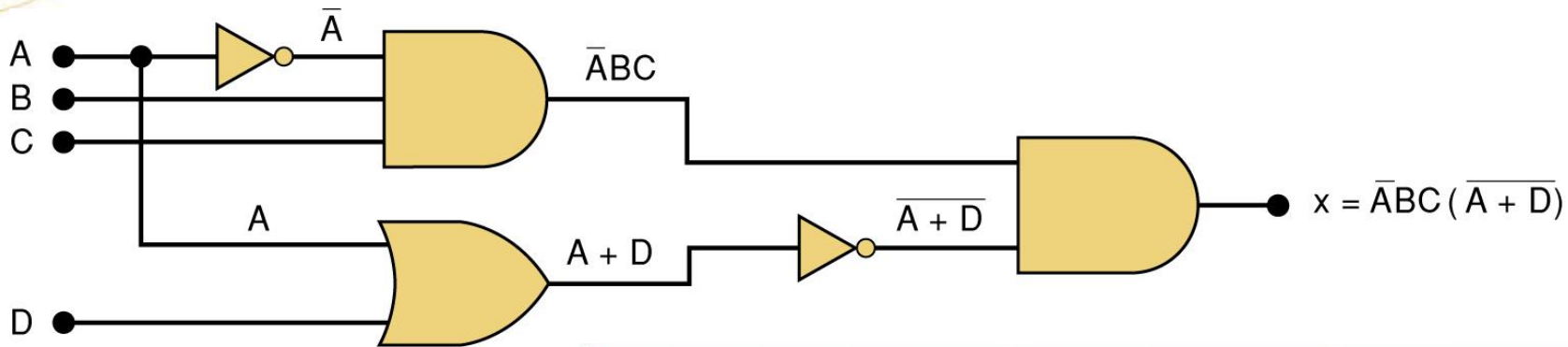


Table of logic state at each node of the circuit shown.

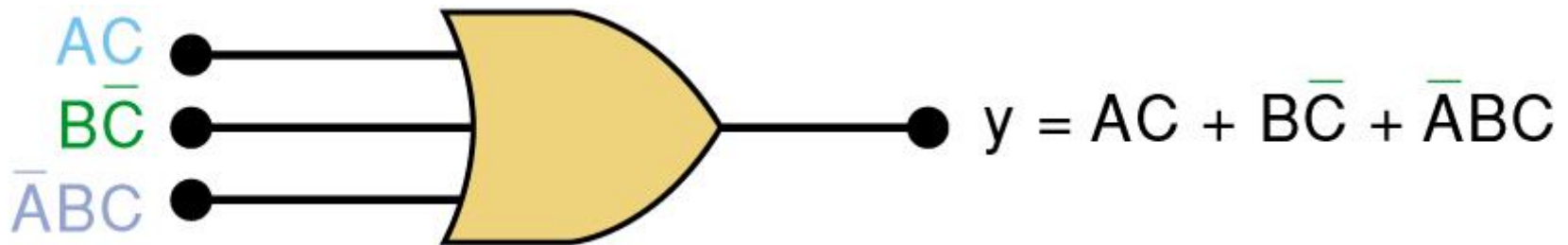
A	B	C	D	t = $\bar{A}BC$	u = A + D	v = $\bar{A} + \bar{D}$	x = tv
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	1
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	0	0	1	0	0
1	1	1	1	0	1	0	0

## 3-8 Implementing Circuits From Boolean Expressions

- It is important to be able to draw a logic circuit from a Boolean expression.
  - The expression  $X = A \cdot B \cdot C$ , could be drawn as a three input **AND** gate.
  - A circuit defined by  $X = A + \bar{B}$ , would use a two-input **OR** gate with an INVERTER on one of the inputs.

## 3-8 Implementing Circuits From Boolean Expressions

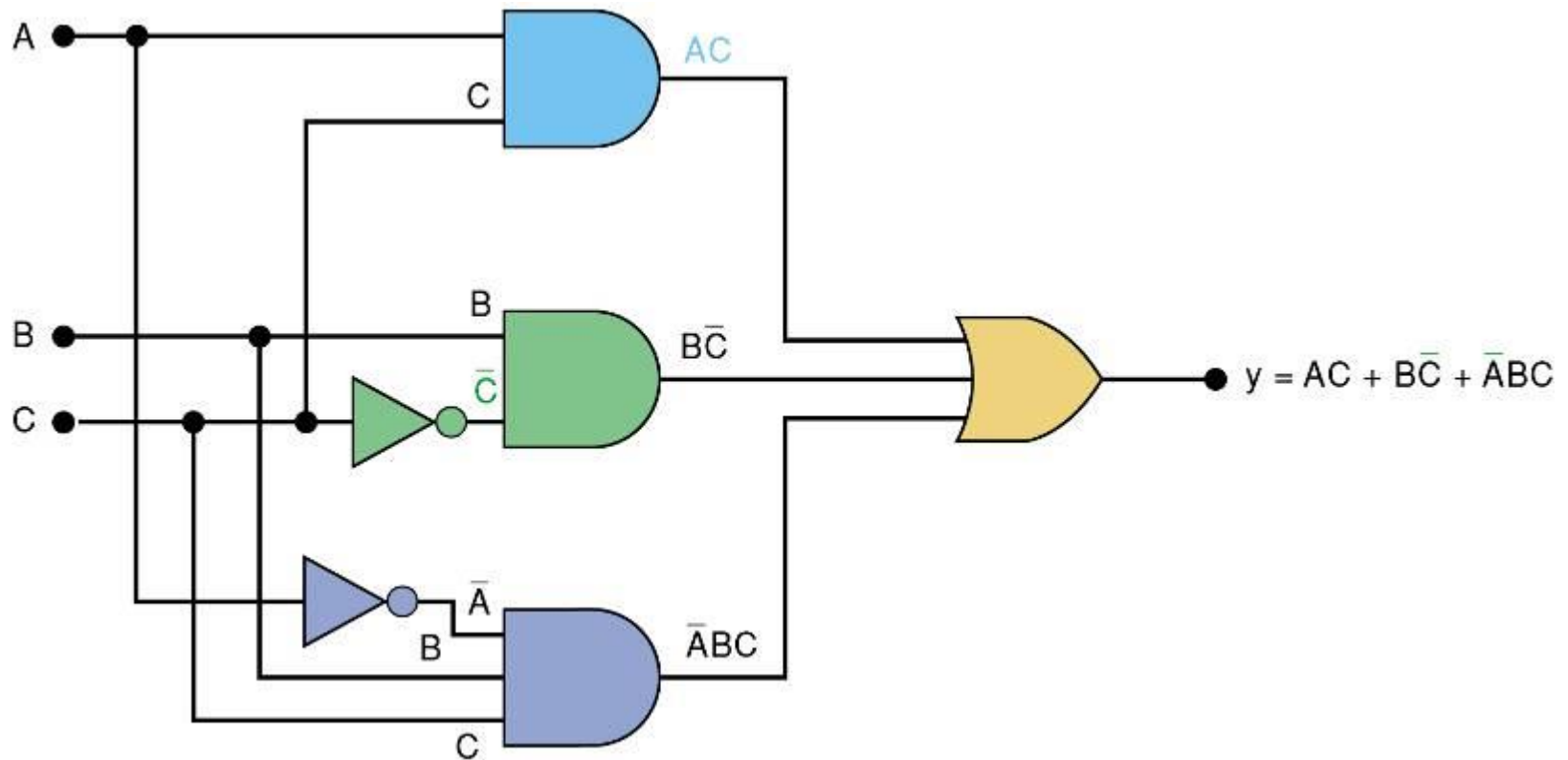
A circuit with output  $y = AC + B\bar{C} + \bar{A}BC$  contains three terms which are **OR**ed together.



...and requires a three-input **OR** gate.

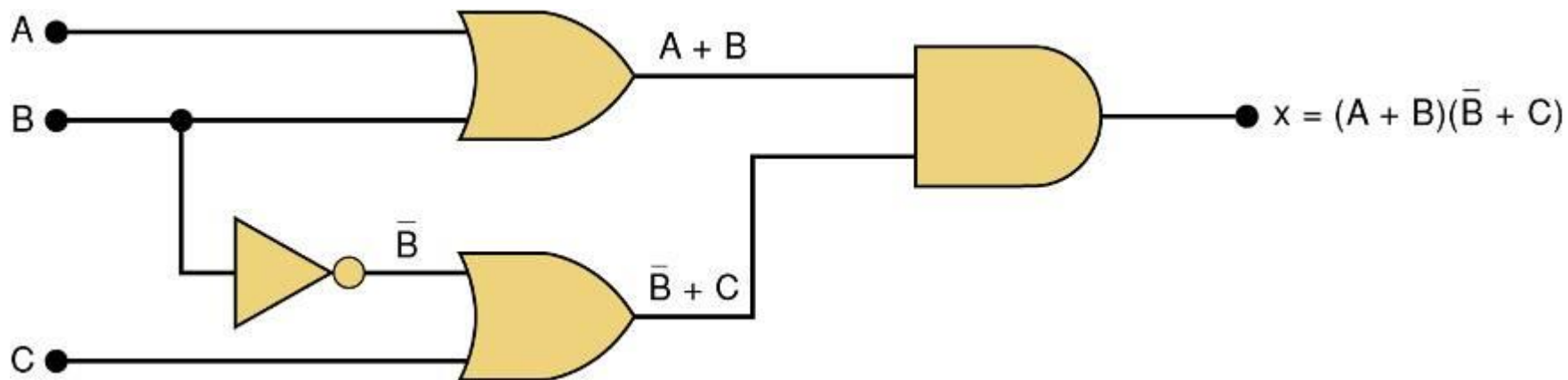
## 3-8 Implementing Circuits From Boolean Expressions

- Each **OR** gate input is an **AND** product term,
  - An **AND** gate with appropriate inputs can be used to generate each of these terms.



## 3-8 Implementing Circuits From Boolean Expressions

Circuit diagram to implement  $x = (A + B)(\bar{B} + C)$



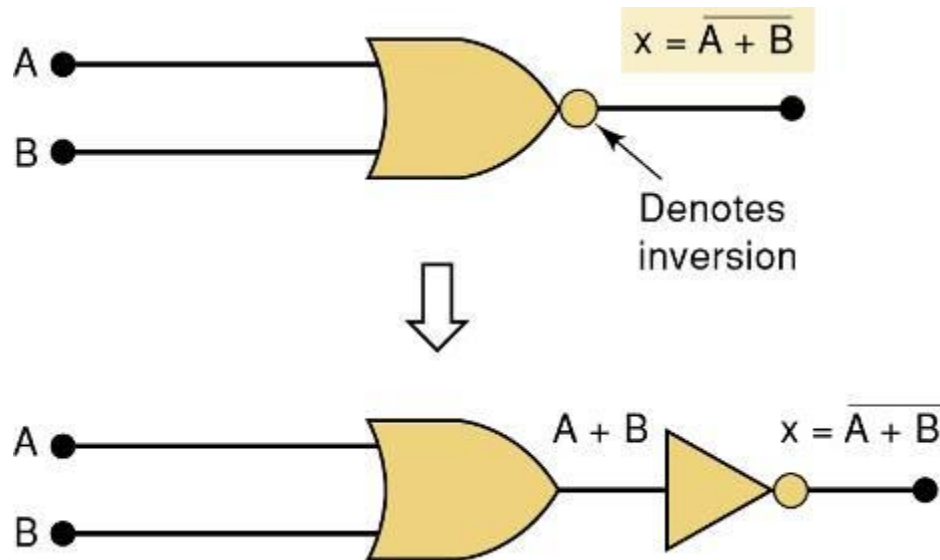
## 3-9 NOR Gates and NAND Gates

- 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.



## 3-9 NOR Gates and NAND Gates

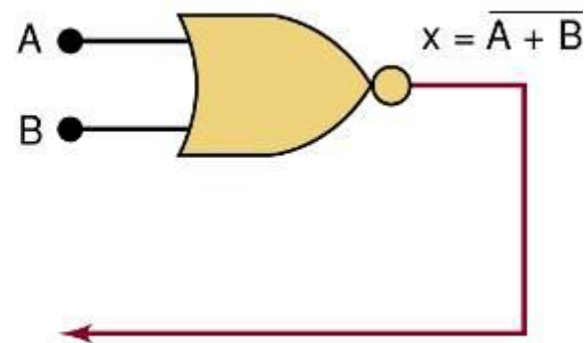
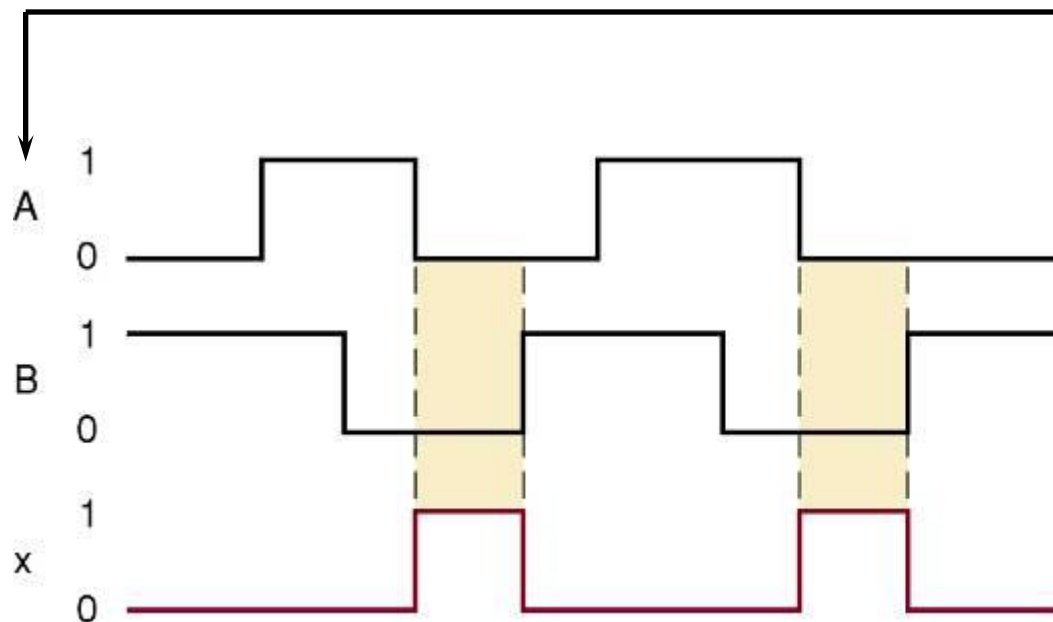
- The **NOR** gate is an inverted **OR** gate.
  - An inversion “bubble” is placed at the output of the **OR** gate, making the Boolean output expression  $x = \overline{A + B}$



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

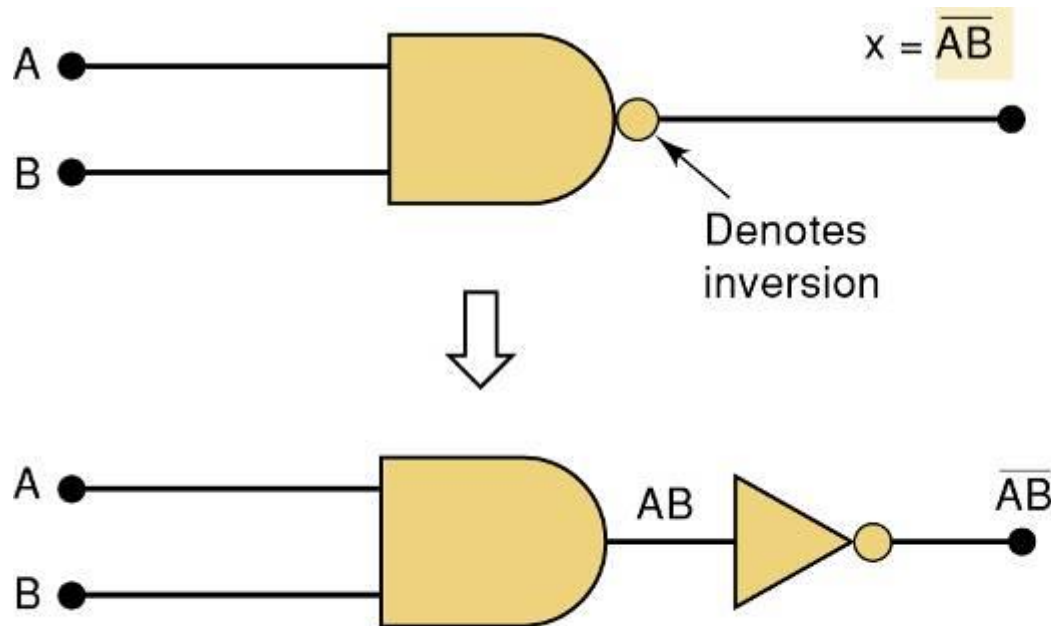
### 3-9 NOR Gates and NAND Gates

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



### 3-9 NOR Gates and NAND Gates

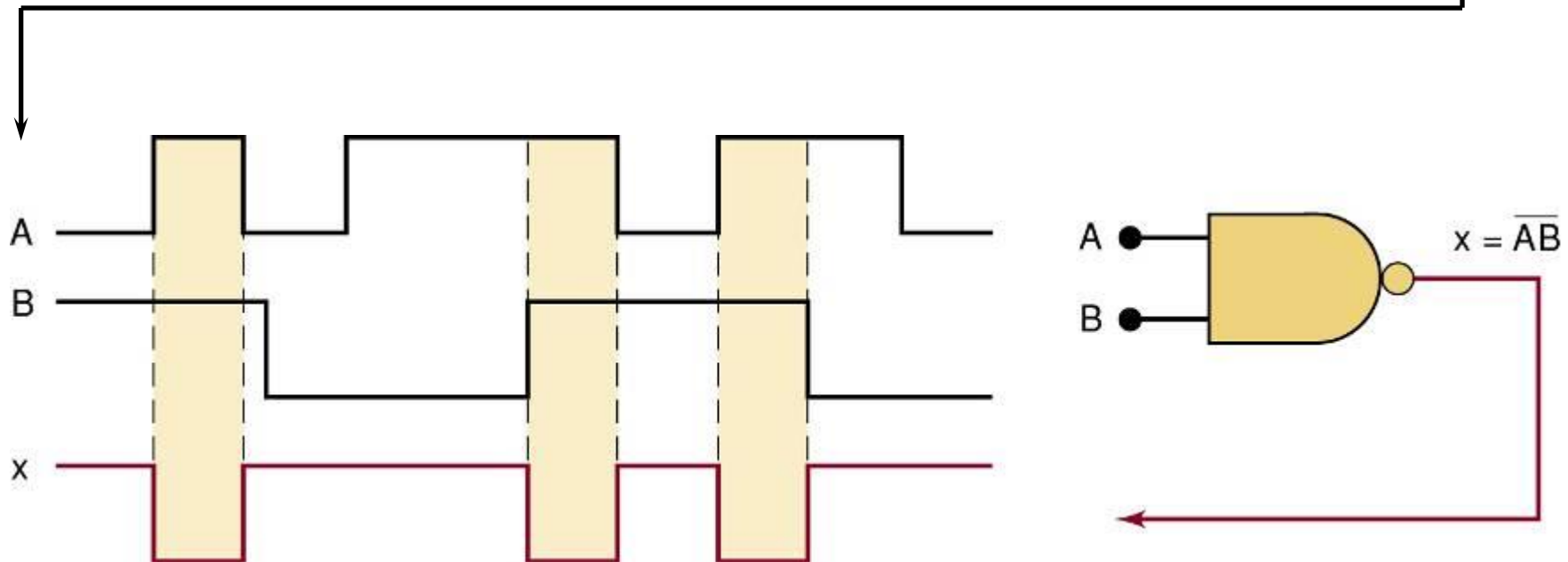
- The **NAND** gate is an inverted **AND** gate.
  - An inversion “bubble” is placed at the output of the **AND** gate, making the Boolean output expression  $x = \overline{AB}$



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

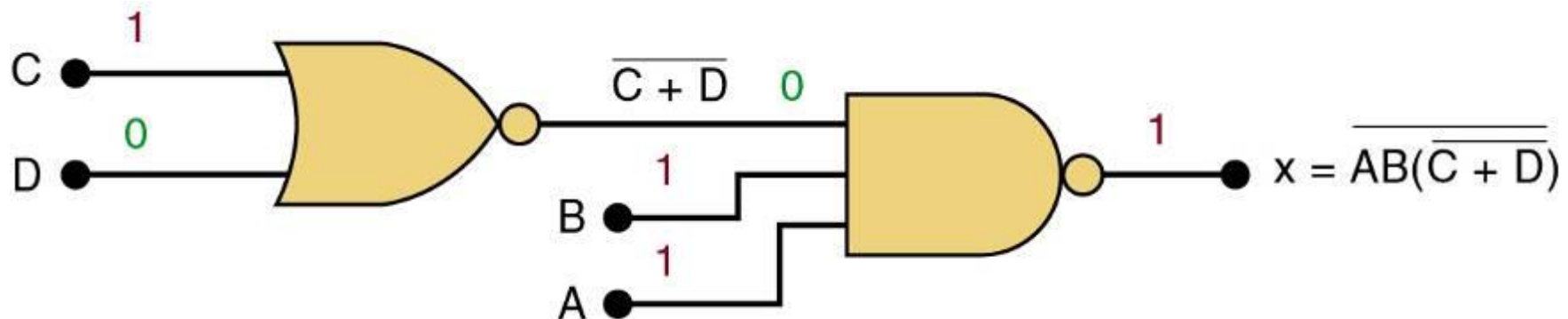
### 3-9 NOR Gates and NAND Gates

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



### 3-9 NOR Gates and NAND Gates

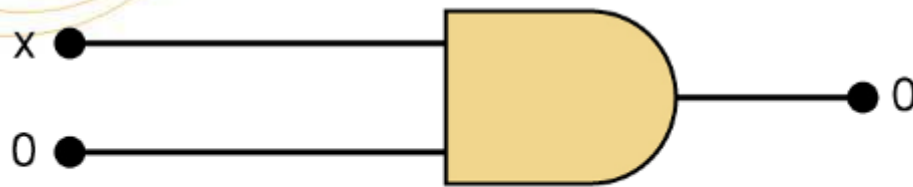
Logic circuit with the expression  $x = \overline{AB} \cdot (\overline{C + D})$  using only **NOR** and **NAND** gates.



## 3-10 Boolean Theorems

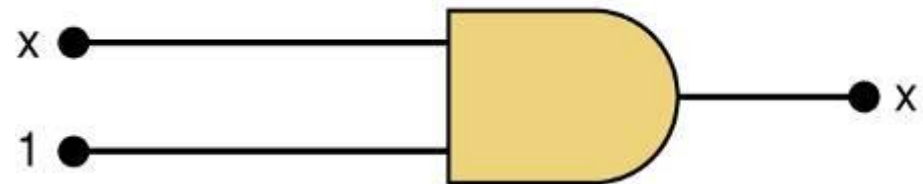
The theorems or laws that follow may represent an expression containing more than one variable.

## 3-10 Boolean Theorems

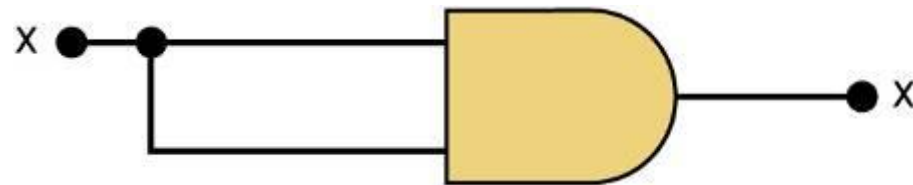


(1)  $x \cdot 0 = 0$

Theorem (2) is also obvious by comparison with ordinary multiplication.

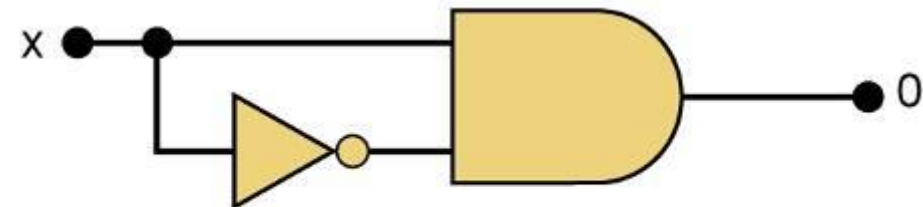


(2)  $x \cdot 1 = x$



(3)  $x \cdot x = x$

Theorem (4) can be proved in the same manner.



(4)  $x \cdot \bar{x} = 0$

Theorem (1) states that if any variable is ANDed with 0, the result must be 0.

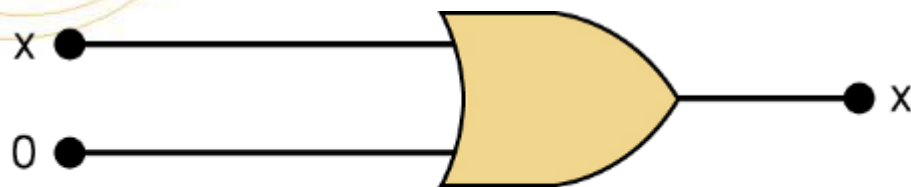
Prove Theorem (3) by trying each case.

If  $x = 0$ , then  $0 \cdot 0 = 0$

If  $x = 1$ , then  $1 \cdot 1 = 1$

Thus,  $x \cdot x = x$

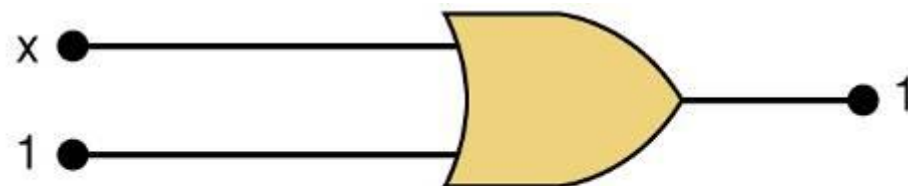
## 3-10 Boolean Theorems



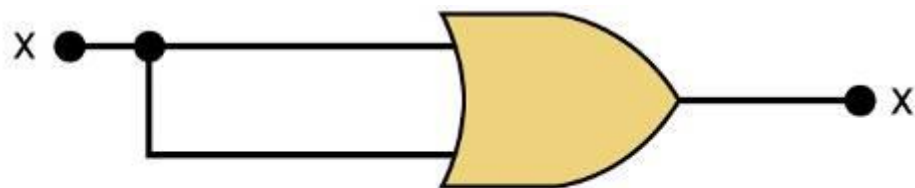
(5)  $x + 0 = x$

Theorem (5) is straightforward, as 0 *added* to anything does not affect value, either in regular addition or in OR addition.

Theorem (6) states that if any variable is ORed with 1, the is always 1.  
Check values:  $0 + 1 = 1$  and  $1 + 1 = 1$ .



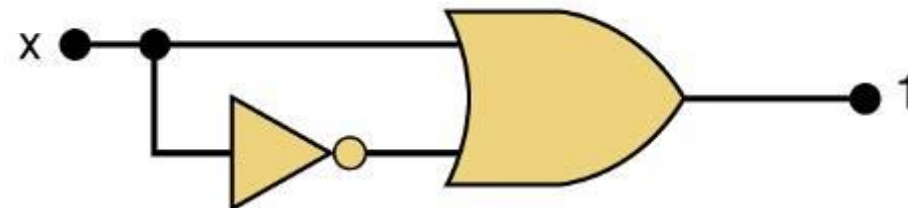
(6)  $x + 1 = 1$



(7)  $x + x = x$

Theorem (7) can be proved by checking for both values of x:  
 $0 + 0 = 0$  and  $1 + 1 = 1$ .

Theorem (8) can be proved similarly.



(8)  $x + \bar{x} = 1$



### Multivariable Theorems

#### Commutative laws

$$(9) \quad x + y = y + x$$

$$(10) \quad x \cdot y = y \cdot x$$

#### Associative laws

$$(11) \quad x + (y + z) = (x + y) + z = x + y + z$$

$$(12) \quad x(yz) = (xy)z = xyz$$

#### Distributive law

$$(13a) \quad x(y + z) = xy + xz$$

$$(13b) \quad (w + x)(y + z) = wy + xy + wz + xz$$

## Multivariable Theorems

Theorems (14) and (15) do not have counterparts in ordinary algebra. Each can be proved by trying all possible cases for  $x$  and  $y$ .

$$(14) \quad x + \underline{xy} = x$$

$$(15a) \quad \underline{x} + \underline{xy} = \underline{x} + y$$

$$(15b) \quad \underline{x} + xy = \underline{x} + y$$

Analysis table & factoring  
for Theorem (14)

$$\begin{aligned}
 x + xy &= x(1 + y) \\
 &= x \cdot 1 && \text{[using theorem (6)]} \\
 &= x && \text{[using theorem (2)]}
 \end{aligned}$$

x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

## 3-11 DeMorgan's Theorems

- **DeMorgan's theorems** are extremely useful in simplifying expressions in which a product or sum of variables is inverted.

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

Theorem (16) says inverting the OR sum of two variables is the same as inverting each variable individually, then ANDing the inverted variables.

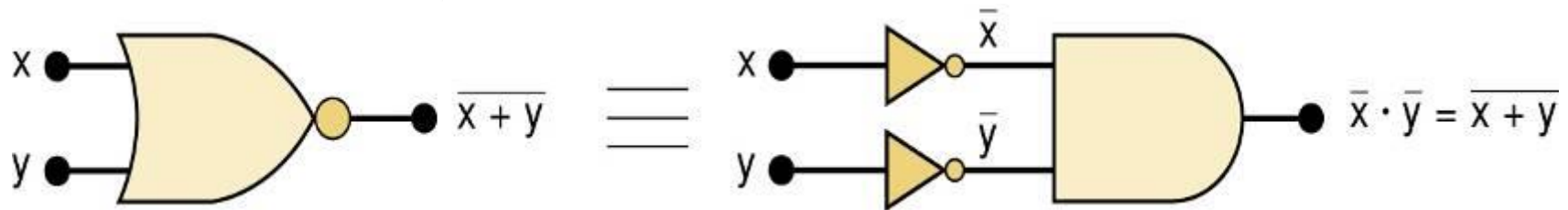
$$(17) \quad \overline{(x \cdot y)} = \bar{x} + \bar{y}$$

Theorem (17) says inverting the AND product of two variables is the same as inverting each variable individually and then ORing them.

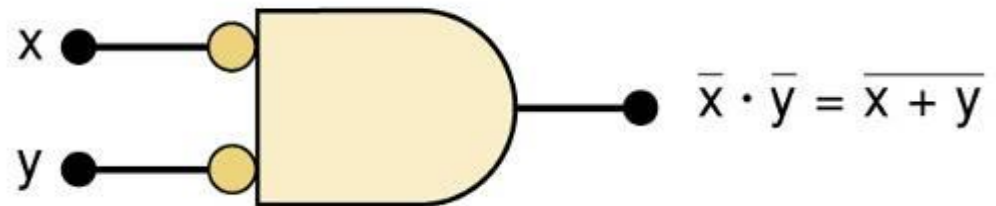
Each of DeMorgan's theorems can readily be proven by checking for all possible combinations of  $x$  and  $y$ .

### Equivalent circuits implied by Theorem (16)

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

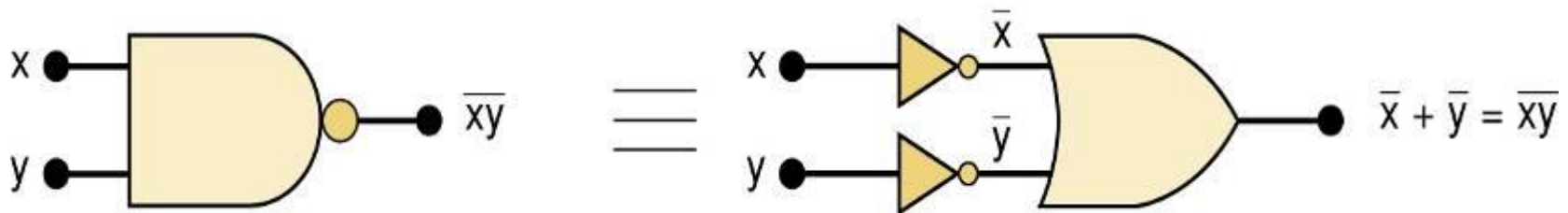


The alternative symbol  
for the NOR function.

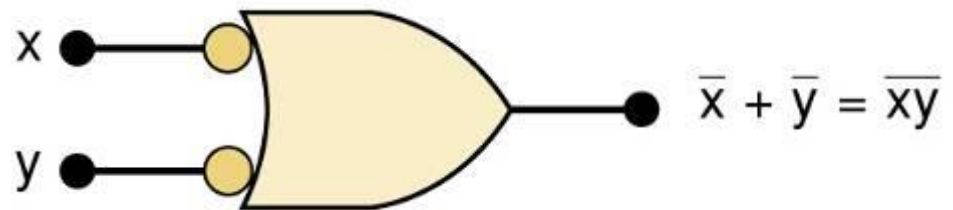


### Equivalent circuits implied by Theorem (17)

$$(17) \quad \overline{(x \cdot y)} = \bar{x} + \bar{y}$$



The alternative symbol for the NAND function.

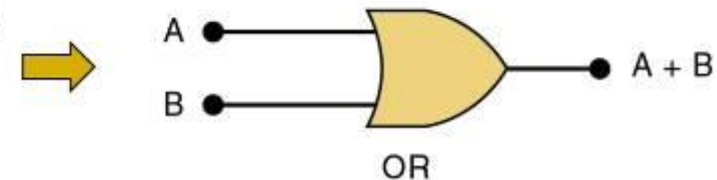
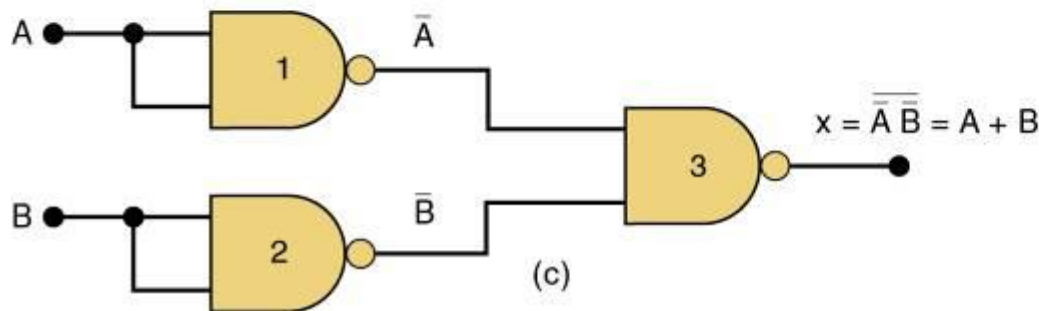
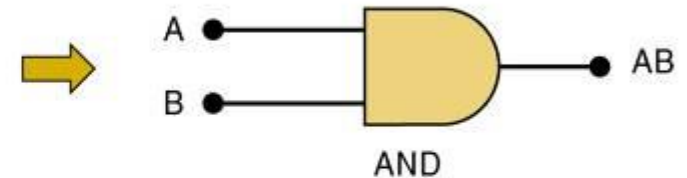
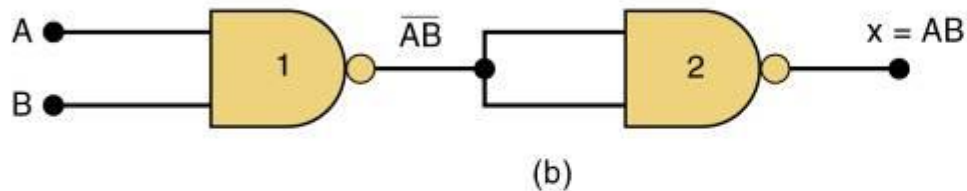
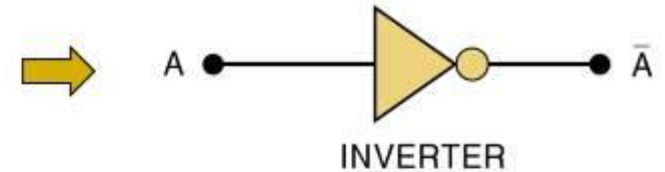
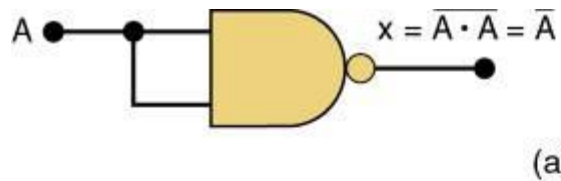


## 3-12 Universality of **NAND** and **NOR** Gates

- **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.

## 3-12 Universality of NAND and NOR Gates

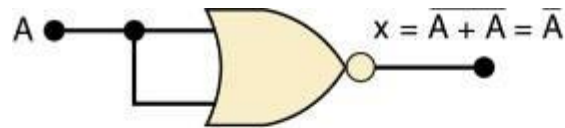
How combinations of NANDs or NORs are used to create the three logic functions.



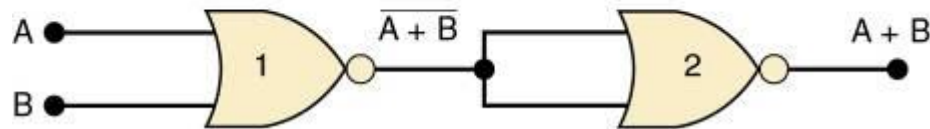
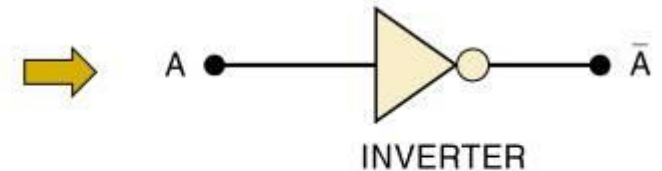
It is possible, however, to implement any logic expression using **only NAND gates** and no other type of gate, as shown.

## 3-12 Universality of NAND and NOR Gates

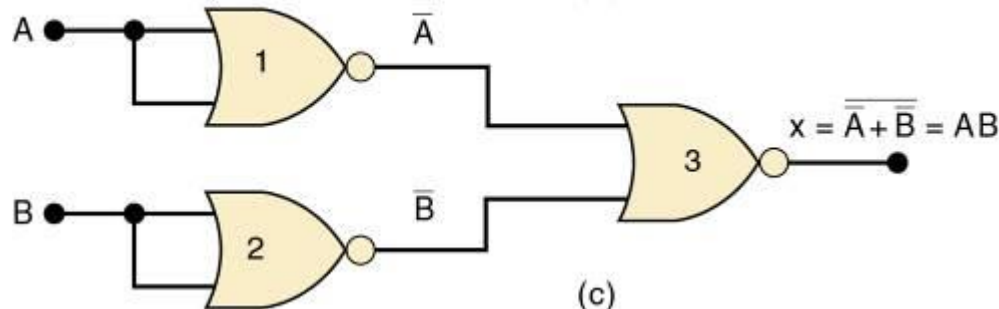
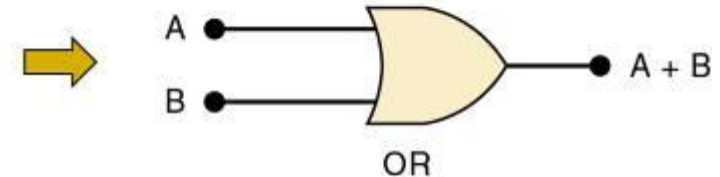
How combinations of NANDs or NORs are used to create the three logic functions.



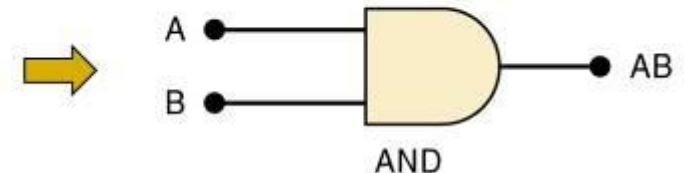
(a)



(b)



(c)



**NOR gates can be arranged to implement any of the Boolean operations, as shown.**

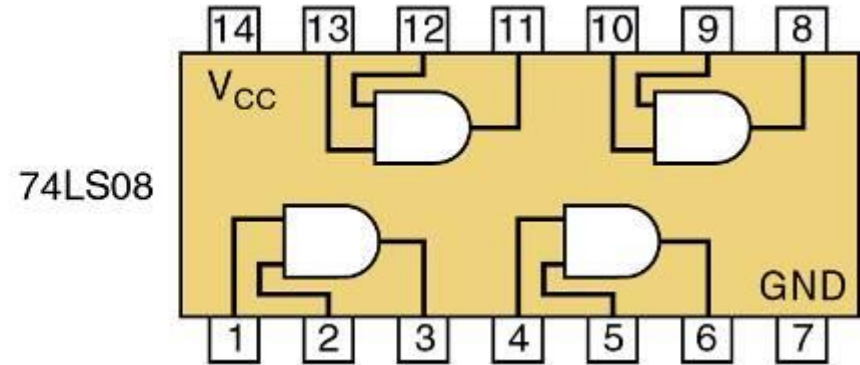
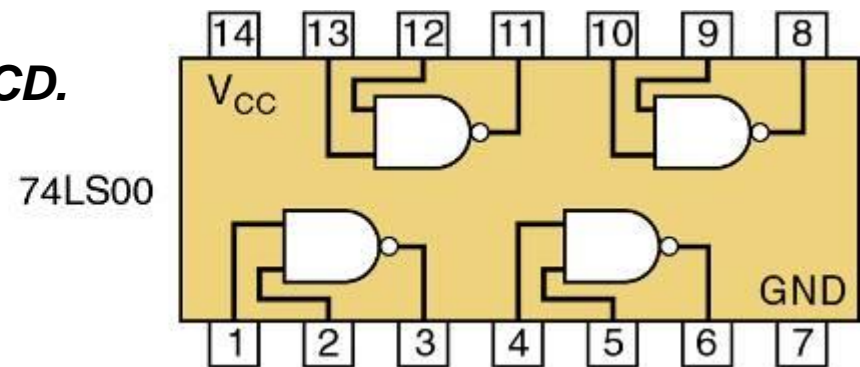
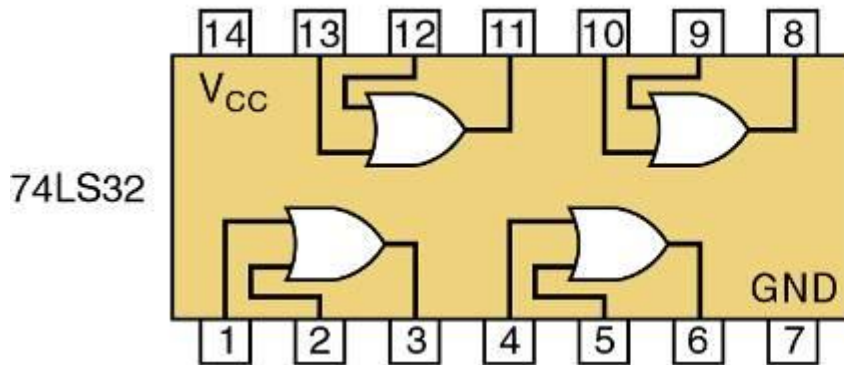


## 3-12 Universality of NAND and NOR Gates

A logic circuit to generate a signal  $x$ , that will go HIGH whenever conditions  $A$  and  $B$  exist simultaneously, or whenever conditions  $C$  and  $D$  exist simultaneously.

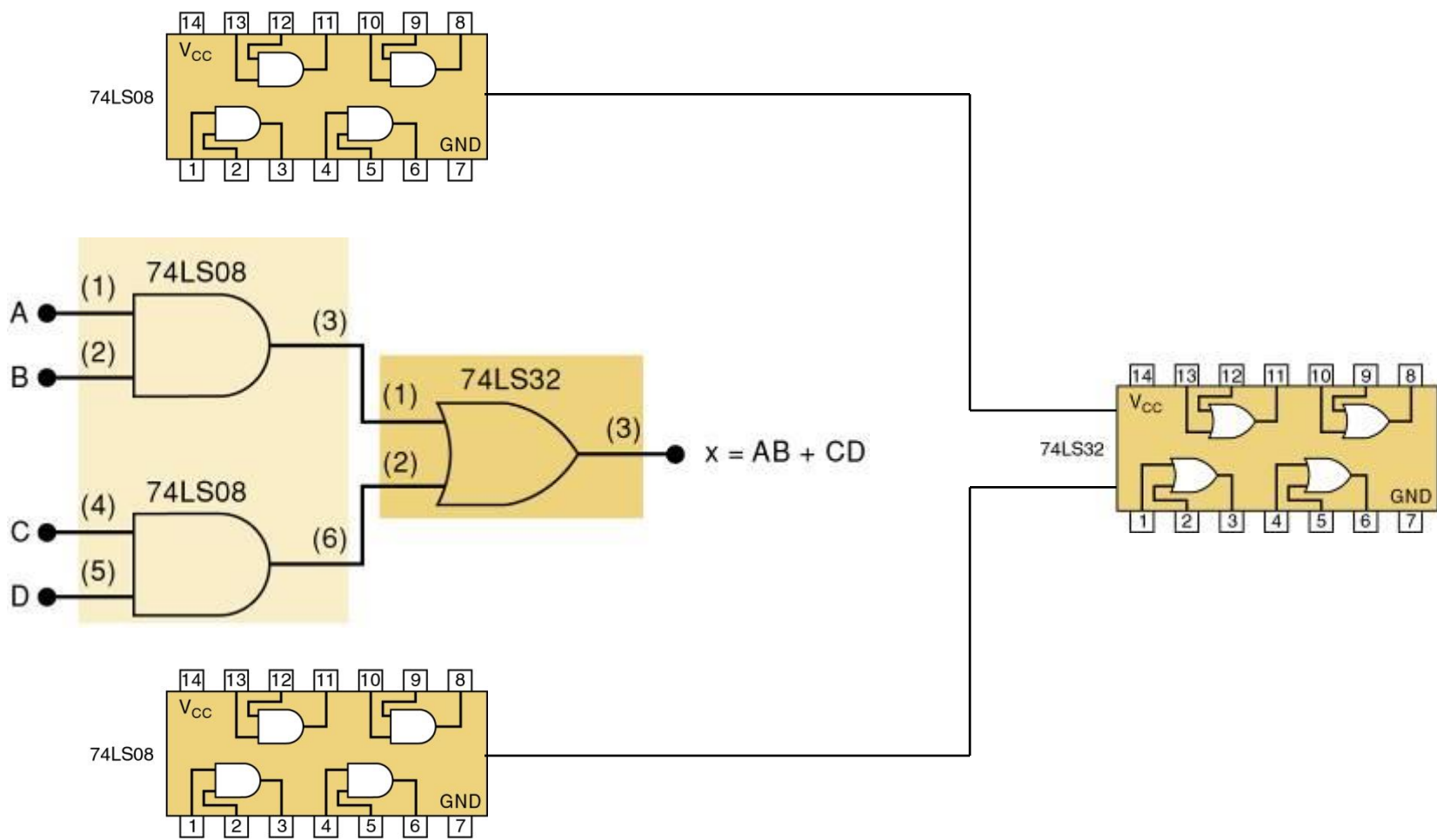
The logic expression will be  $x = AB + CD$ .

Each of the TTL ICs shown here will fulfill the function. Each IC is a *quad*, with *four* identical gates on one chip

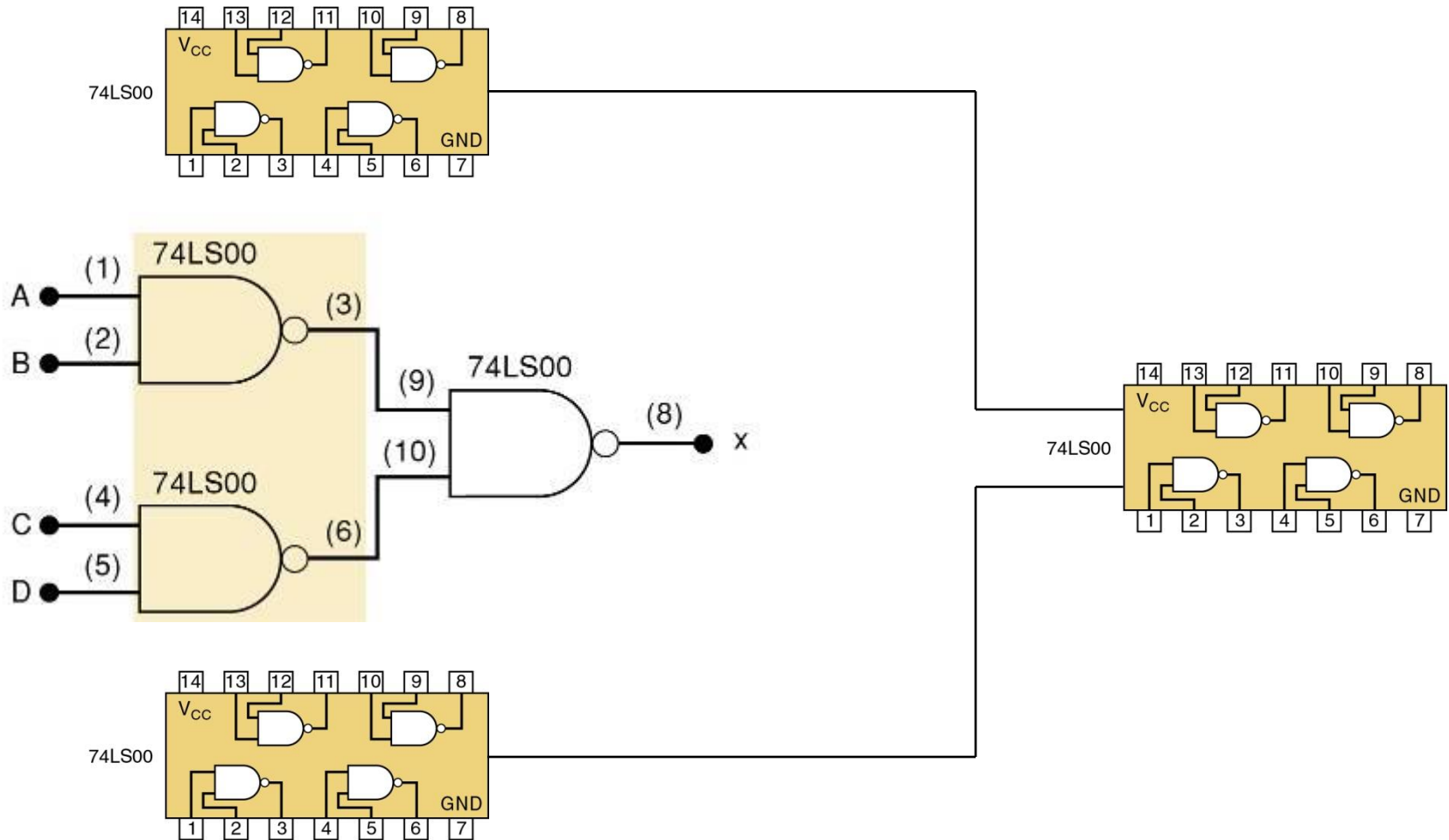


# 3-12 Universality of NAND and NOR Gates

## Possible Implementations # 1



## Possible Implementations #2



## 3-13 Alternate Logic-Gate Representations

- To convert a standard symbol to an alternate:
  - Invert each input and output in standard symbols.
    - Add an inversion bubble where there are none.
    - Remove bubbles where they exist.



## 3-13 Alternate Logic-Gate Representations

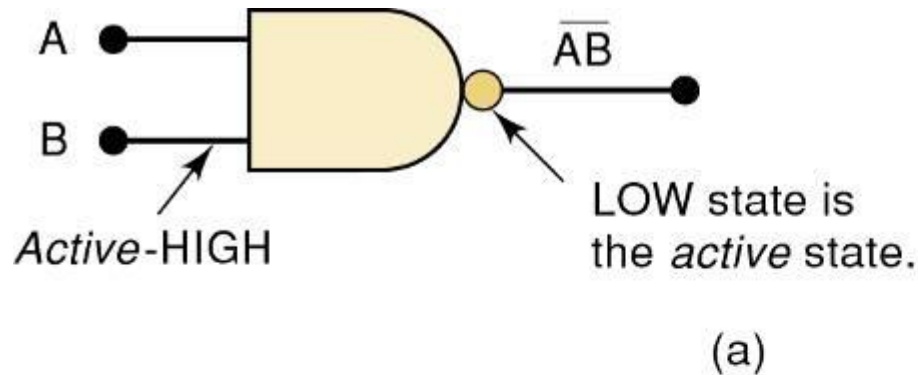
- Points regarding logic symbol equivalences:
  - The equivalences can be extended to gates with *any* number of inputs.
  - None of the standard symbols have bubbles on their inputs, and all the alternate symbols do.
  - Standard & alternate symbols for each gate represent the same physical circuit.
  - **NAND** and **NOR** gates are inverting gates.
    - Both the standard and the alternate symbols for each will have a bubble on *either* the input or the output.
  - **AND** and **OR** gates are *noninverting* gates.
    - The alternate symbols for each will have bubbles on *both* inputs and output.

## ● 3-13 Alternate Logic-Gate Representations

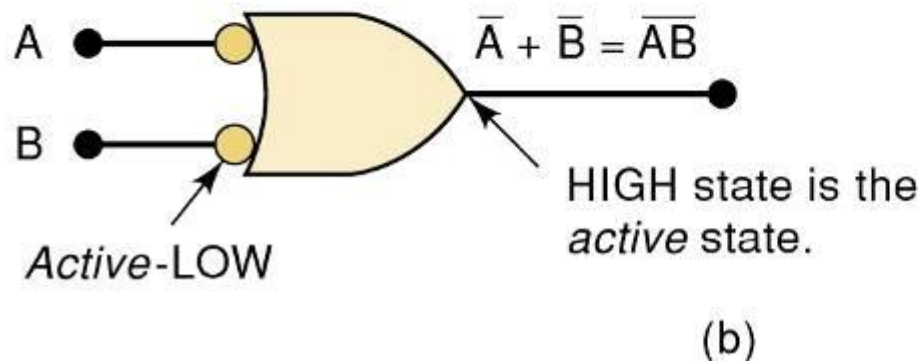
- Active-HIGH – an input/output has *no* inversion bubble.
- Active-LOW – an input or output has an inversion bubble.

## 3-13 Alternate Logic-Gate Representations

Interpretation of the two **NAND** gate symbols.



Output goes LOW only when *all* inputs are HIGH.

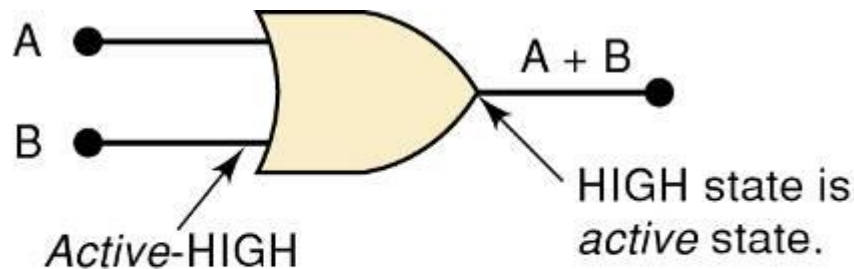


Output is HIGH when *any* input is LOW.

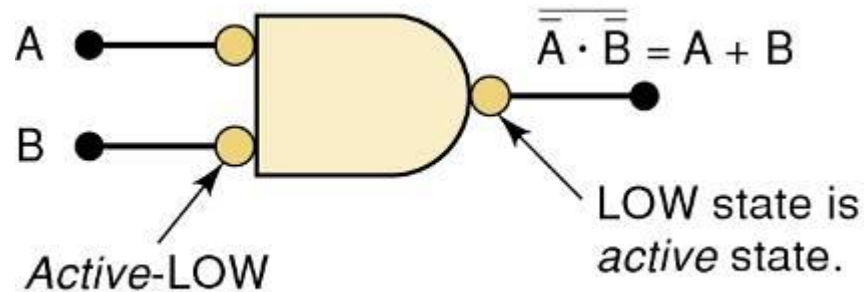


## 3-13 Alternate Logic-Gate Representations

Interpretation of the two **OR** gate symbols.



Output goes HIGH when *any* input is HIGH.

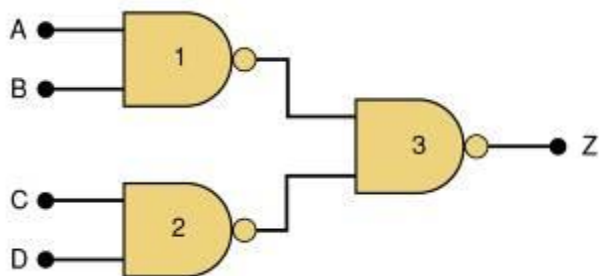


Output goes LOW only when *all* inputs are LOW.



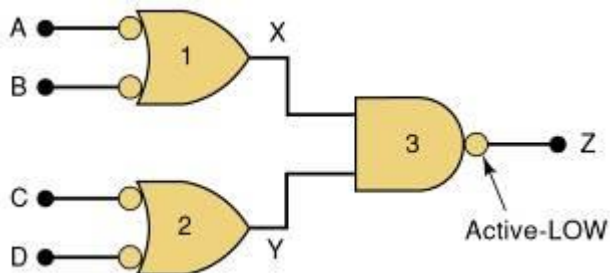
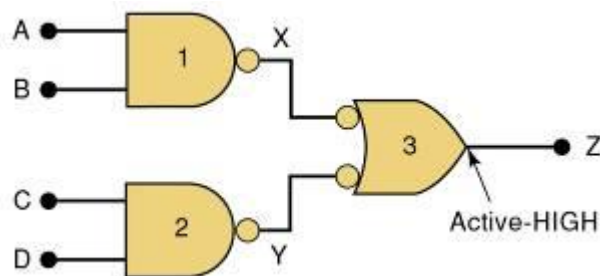
## 3-14 Which Gate Representation to Use

Proper use of alternate gate symbols in the circuit diagram can make circuit operation much clearer.



Original circuit using standard **NAND** symbols.

Equivalent representation where output Z is active-HIGH.



Equivalent representation where output Z is active-LOW.

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
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	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

### Boolean Theorems:

- $x*0=0$ ;  $x*1=x$ ;  $x*x=x$ ;  $x*x'=0$
- $x+0=x$ ;  $x+1=1$ ;  $x+x=x$ ;  $x+x'=1$
- $x+y=y+x$ ;  $x*y=y*x$
- $x+(y+z)=(x+y)+z=x+y+z$ ;  $x(yz)=(xy)z=xyz$
- $x(y+z)=xy+xz$ ;  $(w+x)(y+z)=wy+wz+xy+xz$
- $x+xy=x$ ;  $x+x'y=x+y$ ;  $x'+xy=x'+y$
- $(x+y)'=x'y'$ ;  $(xy)'=x'+y'$