Digital Systems
Section 2

Chapter (2)

Mohammed Khalil

Boolean Algebra → Mathematics of Logic

	Regular	Boolean
Values	Real numbers (e.g., 1, 2, 3, ...)	Binary values (0, 1)
Operations	Addition (+), Multiplication (×) Subtraction (-), Division (÷)	AND (•), OR (+), NOT (′)
Purpose	Solving numerical equations	Simplifying logic circuits
Deal with	Formulas	Truth Tables & Gates

- Boolean expressions are made up of **Variables**, **Constants** and **Operators**
  - **Operators**: AND, OR and NOT
  - **Literals** each instance of a variable or constant

$$(x+y)'\cdot(x'+y')$$

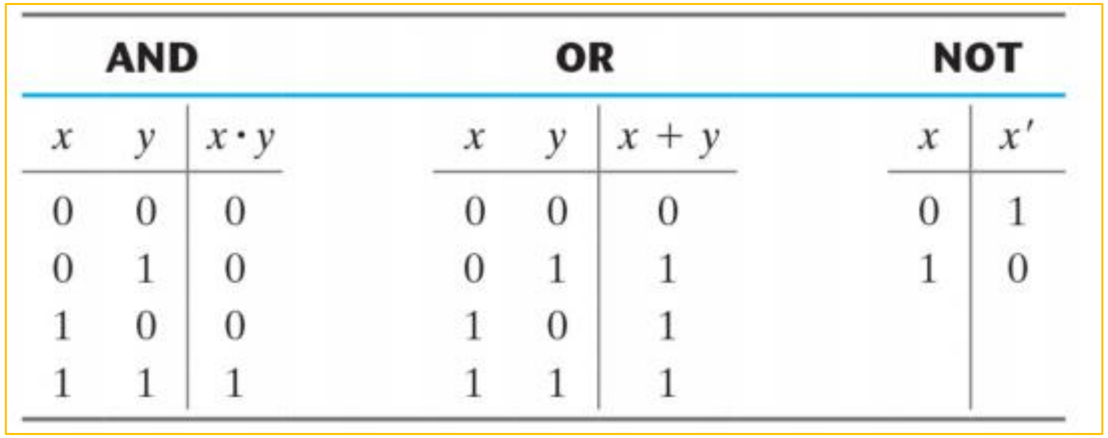
Boolean Algebra: Algebraic structure defined by a set of elements, B={0,1}, with two binary operators (+ and •) and a **unary** operator (’) satisfying the following **Postulates**:

- There exit at least two elements x , y ∊ B such that x≠y
- For x ∊ B there is x’ ∊ B . (**Complement or Inverse**)
- The structure is closed with respect to (+ and •). (**Closure**)
- **0** is the **identity** element for (+), and **1** is the identity element for (•).
- The structure is **commutative** with respect to (+ and •).
- (•) is **distributive** over +, and + is distributive over (•).

Boolean Operators

- Logical **operators** operate on binary values and binary variables
- Three basic logical operations; AND (•), OR (+), NOT (′).

Truth Table



AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

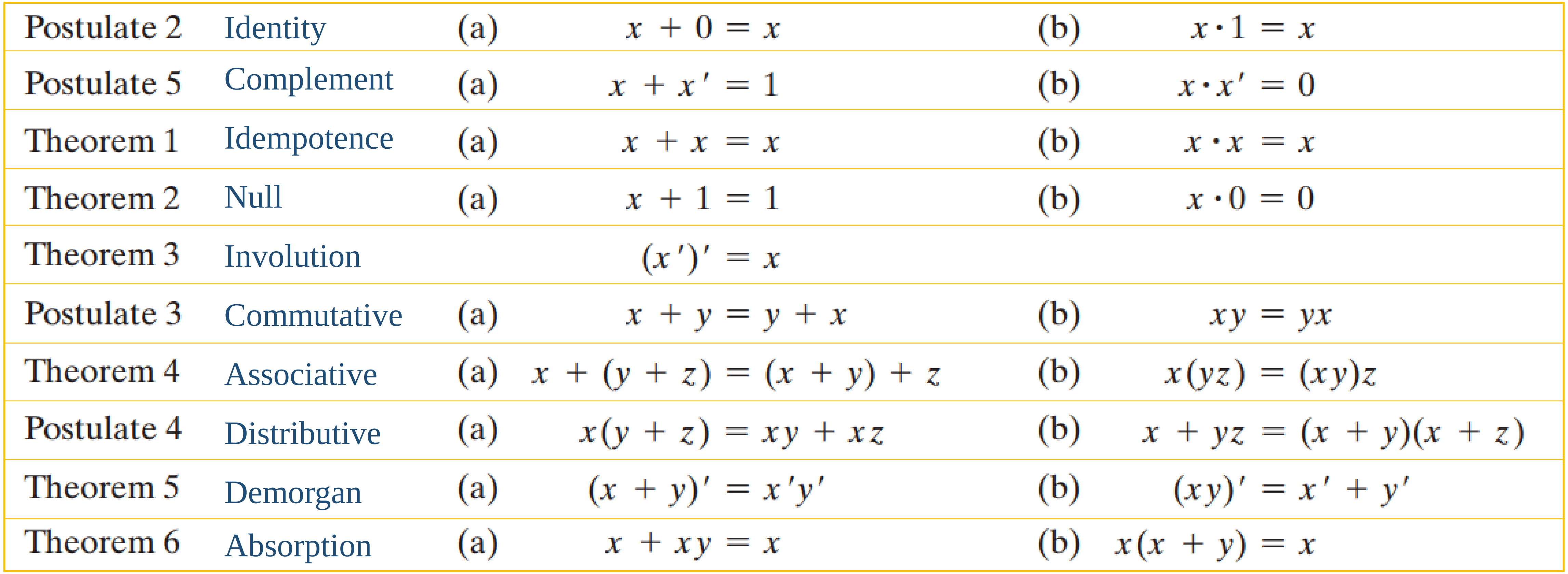
Sometimes, the dot symbol (•) is **not written** when the meaning is clear

NOT Operator Notation: $(\overline{A})$ $(\sim A)$ (A')

Postulates and Theorems

- The **Postulates** are basic axioms of the algebraic structure and **need no proof**
- The **Theorems** **must** be proven from the postulates

Dual



Postulate 2	Identity	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	Complement	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	Idempotence	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	Null	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3	Involution		$(x')' = x$		
Postulate 3	Commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4	Associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4	Distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5	Demorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6	Absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Duality Principle

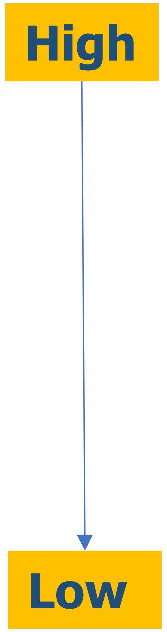
- The **dual** of an algebraic expression is obtained by **interchanging** + and • and **interchanging** 0’s and 1’s
- Every algebraic expression deducible from the postulates of Boolean algebra remains **valid** if the operators and identity elements are **interchanged**
- If two Boolean Expressions are equal, the duals are equal

Examples:

Dual of $(x + y + z)$ → $(x \bullet y \bullet z)$
Dual of $(x + 0 = x)$ → $(x \bullet 1 = x)$
Dual of $(x + 1 = 1)$ → $(x \bullet 0 = 0)$

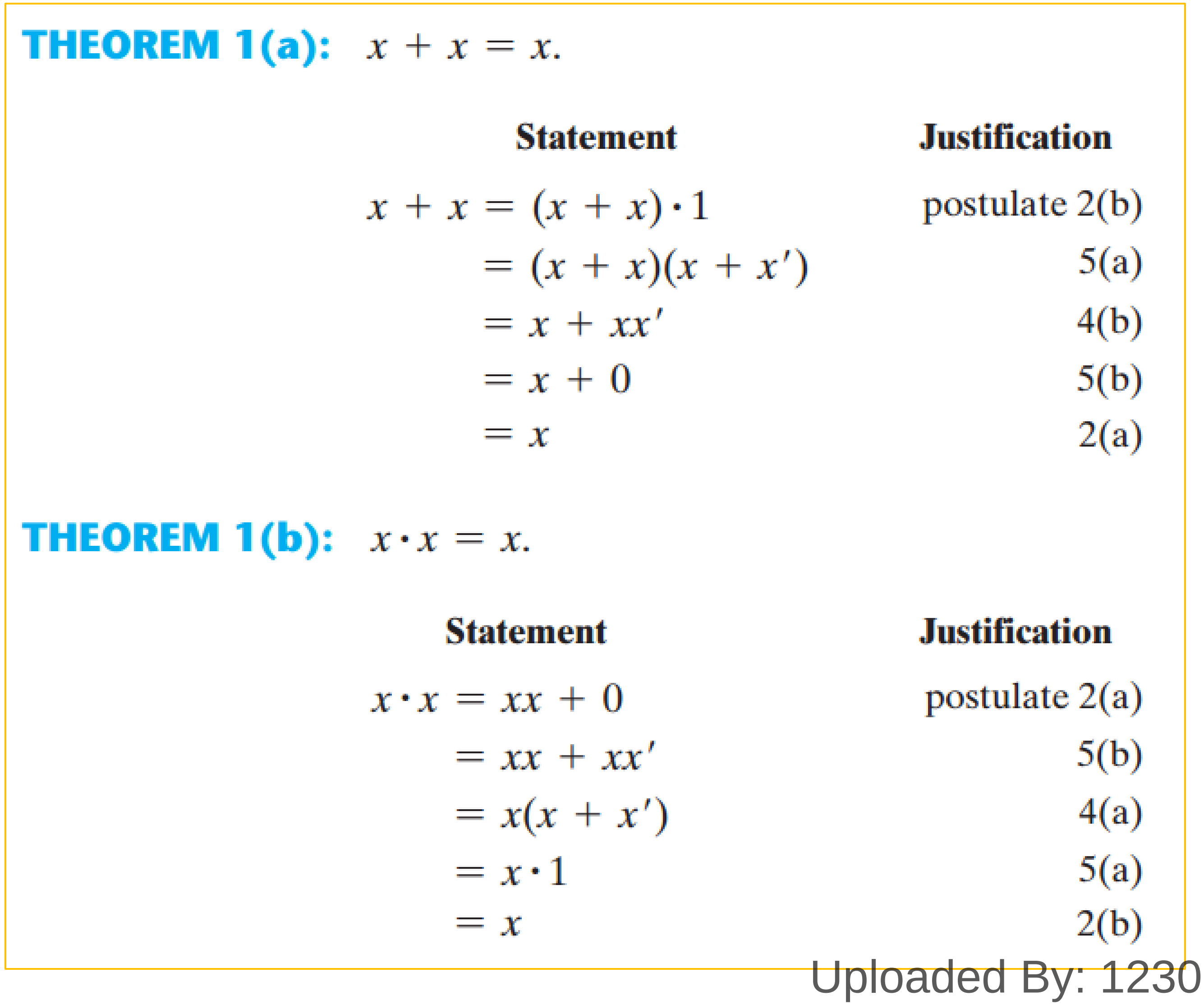
Operator Precedence

1) Parentheses
2) NOT
3) AND
4) OR



Proof of Theorems

- Proof Using **Postulates** & Other Proven Theorems
- Proof Using **Truth Tables**



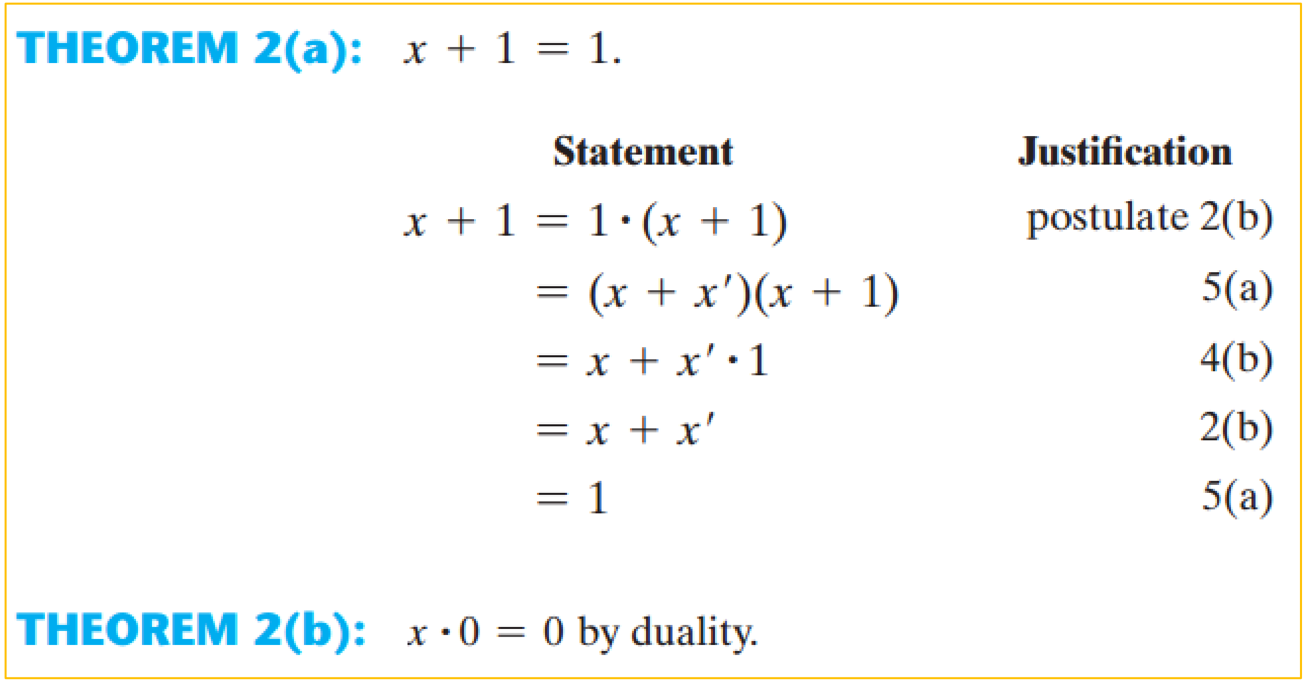
THEOREM 1(a): $x + x = x$.

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

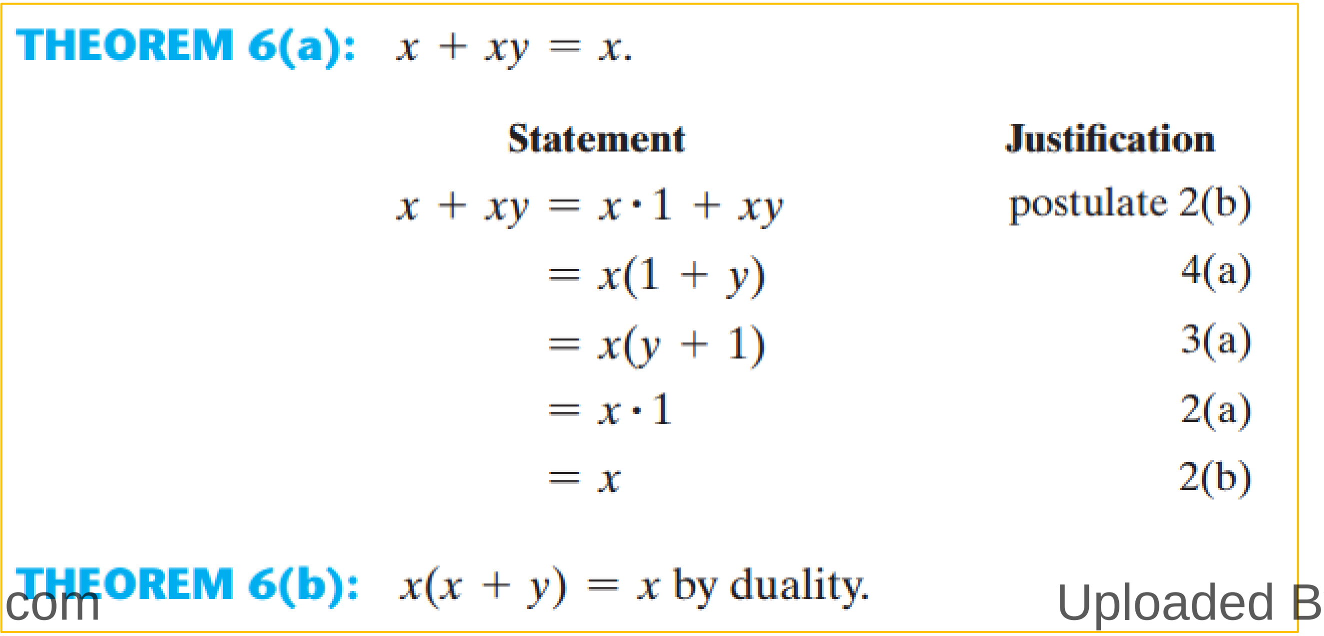
Proof of Theorems



THEOREM 2(a): $x + 1 = 1.$

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.



THEOREM 6(a): $x + xy = x.$

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

Other Useful Theorems

Consensus Theorem $AB + \overline{A}C + BC = AB + \overline{A}C$

$$\begin{aligned}
&= AB + \overline{A}C + BC \\
&= AB + \overline{A}C + 1 \cdot BC, && \text{Identity element} \\
&= AB + \overline{A}C + (A + \overline{A}) \cdot BC, && \text{Complement} \\
&= AB + \overline{A}C + ABC + \overline{A}BC, && \text{Distributive} \\
&= AB + ABC + \overline{A}C + \overline{A}BC, && \text{Commutative} \\
&= AB(1 + C) + \overline{A}C(1 + B), && \text{Distributive} \\
&= AB \cdot 1 + \overline{A}C \cdot 1, && \text{Null} \\
&= AB + \overline{A}C, && \text{Identity element}
\end{aligned}$$

Minimization Theorem $xy + \overline{x}y = y$ $(x + y)(\overline{x} + y) = y$

Simplification Theorem $x + \overline{x}y = x + y$ $x(\overline{x} + y) = xy$

Boolean Functions

- Boolean algebra deals with binary variables and logic operations
- A Boolean **Function** is an algebraic expression consisting of binary **variables**, the **constants** (0, and 1), and logic **operation** symbols
- A Boolean **Function** always equals/evaluated to either **0 or 1**
- A Boolean **Function** is **uniquely** represented by a **truth table** that maps each possible combination of the input variables to the corresponding output literal
- Boolean **Function** can be implemented (**NOT Uniquely**) by a Boolean **Equation** and the corresponding **logic diagram**
- Simplest Functions use the **smallest** number of the **smallest** gates and therefore give the *most economical* and *efficient* circuit implementations

Several such implementations map to the **same** function

Requires:
Optimization/Minimization/
Manipulations Techniques

Examples:

$F_1 = x + y'z$	$F_5 = xy + x(wz + wz')$
$F_2 = x'y'z + xyz + x'yz + xy'z$	$F_6 = (yz' + x'w)(xy' + zw')$
$F_3 = x'yz + xz$	$F_7 = (x' + z')(x + y' + z')$
$F_4 = (x + y)'(x' + y')$	$F_8 = x + xz$

Truth Table Representation

- Any Boolean **Function** can be represented in a **Truth Table**
  - The number of rows in a truth table is **2^n** where
    n is the number of variables in the function/expression
  - The binary combinations of the variables are obtained
    by counting from **0 to 2^n– 1**

$F_1 = xyz'$ $\qquad F_2 = x + y'z$ $\qquad F_3 = x'y'z + x'yz + xy'$ $\qquad F_4 = xy' + x'z$

Input Variables (n=3) **Output Functions**

0

2^n - 1

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Complement of a Function (Inverse)

- The complement of a Boolean function may be obtained by either one of two methods:
  1) Repetitive application of **DeMorgan's** theorem.
  2) Taking the dual of the function and **complementing** each **literal**.

Examples:

$$F=\overline{x}\cdot y\cdot\overline{z}+\overline{x}\cdot\overline{y}\cdot z$$

Method 1

$$\overline{F}=\overline{(\overline{x}\cdot y\cdot\overline{z}+\overline{x}\cdot\overline{y}\cdot z)}$$
$$\overline{F}=\overline{(\overline{x}\cdot y\cdot\overline{z})}\cdot\overline{(\overline{x}\cdot\overline{y}\cdot z)}$$
$$\overline{F}=(x+\overline{y}+z)\cdot(x+y+\overline{z})$$

Method 2

$$F^D=(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+z)$$
$$\overline{F}=(x+\overline{y}+z)\cdot(x+y+\overline{z})$$

$F_1 = x'yz' + x'y'z.$

The dual of F_1 is $(x' + y + z')(x' + y' + z)$.

Complement each literal: $(x + y' + z)(x + y + z') = F_1'$.

$F_2 = x(y'z' + yz).$

The dual of F_2 is $x + (y' + z')(y + z)$.

Complement each literal: $x' + (y + z)(y' + z') = F_2'$.

Be Careful: (XY)′ ≠ X′Y′

Algebraic Manipulation (Simplification)

- Reducing the number of **terms**, the number of **literals**, or **both** → a simpler logic circuit can be used to implement the Boolean function (Less Gates & Less Inputs)

- Reduction of the number of terms and/or number of literals is done by algebraic **manipulation**.

- Basic theorems and postulates of Boolean algebra are applied to perform the **manipulation.**

Examples:

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

xy

$x + \overline{x} \cdot y =$

$x+y$

$(x + y)(x + \overline{y}) =$

x

$x \cdot y + \overline{x} \cdot z + y \cdot z =$

$xy + \overline{x}z$

Algebraic Manipulation (Simplification)

More Examples:

$$
\begin{aligned}
F_2 &= x'y'z + xyz + x'yz + xy'z \\
&= x'z(y+y') + xz(y+y') \\
&= x'z + xz \\
&= (x'+x)z \\
&= z
\end{aligned}
$$

Factoring Out Common Terms

$$
\begin{aligned}
F_3 &= x'yz + xz \\
&= (x'y + x)z \\
&= (x+x')(x+y)z \\
&= (x+y)z
\end{aligned}
$$

$$
\begin{aligned}
xy + x'z + yz &= xy + x'z + yz(x+x') \\
&= xy + x'z + xyz + x'yz \\
&= xy(1+z) + x'z(1+y) \\
&= xy + x'z
\end{aligned}
$$

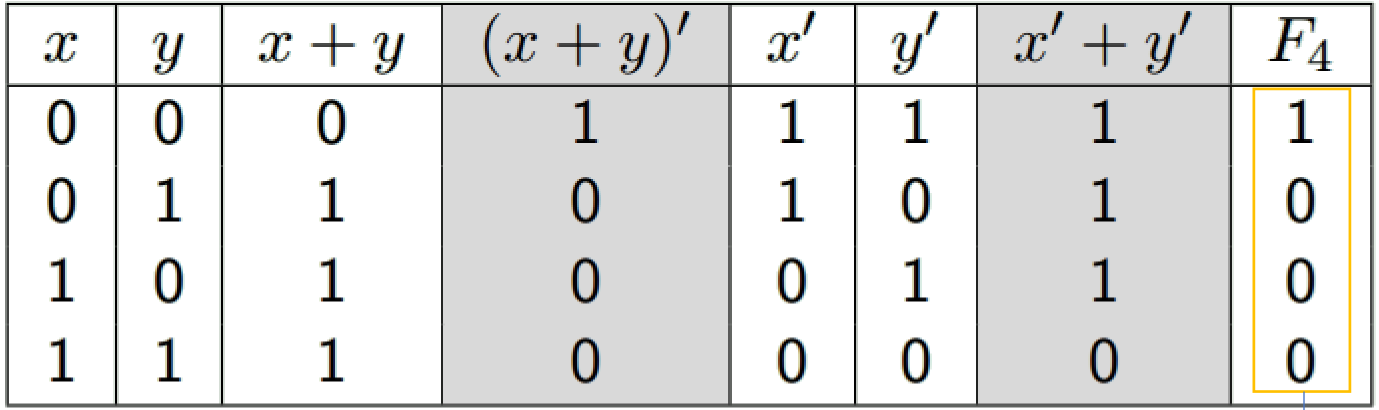
Multiply (AND) By 1

Similarly, we could Add (OR) By 0 $x \cdot x'$

More Examples:

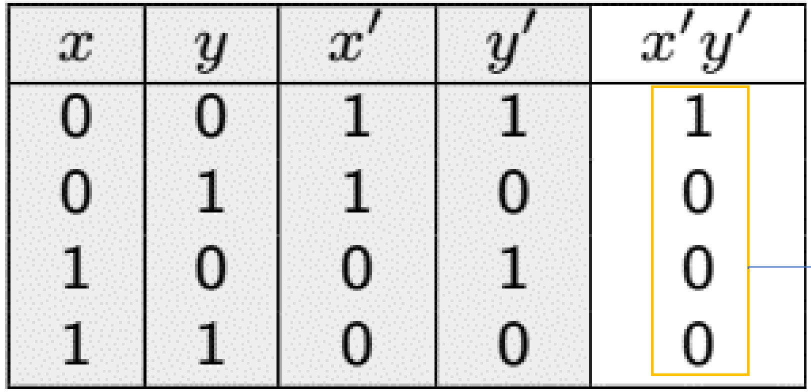
$$
\begin{aligned}
F_4 &= (x+y)'(x'+y') \\
&= x'y'(x'+y') \\
&= x'y'x' + x'y'y' \\
&= x'y' + x'y' \\
&= x'y'
\end{aligned}
$$

F4 Truth Table (Original)



x	y	$x+y$	$(x+y)'$	x'	y'	$x'+y'$	F_4
0	0	0	1	1	1	1	1
0	1	1	0	1	0	1	0
1	0	1	0	0	1	1	0
1	1	1	0	0	0	0	0

F4 Truth Table (Simplified)



x	y	x'	y'	$x'y'$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Equivalent

Remember, A Boolean Function may have **multiple** expression/logical forms, but can have **only** one **Truth Table representation.**

More Examples:

$$\overline{\overline{A\overline{BC}}+C+D} = A(\overline{C+D})$$

$$\begin{aligned}
\text{L.H.S.} &= \overline{\overline{A\overline{BC}}+C+D} \\
&= A\overline{BC}\,\overline{C}\,\overline{D} \qquad \text{DeMorgan's theorem} \\
&= A(\overline{B}+\overline{C})\,\overline{C}\,\overline{D} \qquad \text{DeMorgan's theorem} \\
&= A\overline{B}\,\overline{C}\,\overline{D}+A\,\overline{C}\,\overline{D} \\
&= A\,\overline{C}\,\overline{D}(\overline{B}+1) \\
&= A\,\overline{C}\,\overline{D} \\
&= A\,(\overline{C+D}) \qquad \text{DeMorgan's theorem} \\
&= \text{R.H.S.}
\end{aligned}$$

Algebraic Manipulation (Simplification)

Consensus

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

More Examples:

a’c’ + ad + bc’d = a’c’ + ad

a’c’ + ad + bc’d = a’c’ + a d + bc’d + c’d (by **consensus** between a’c’ + ad)
= a’c’ + ad + c’d (by **absorption** of bc’d in c’d)
= a’c’ + a d (by **consensus** between a’c’ + ad)

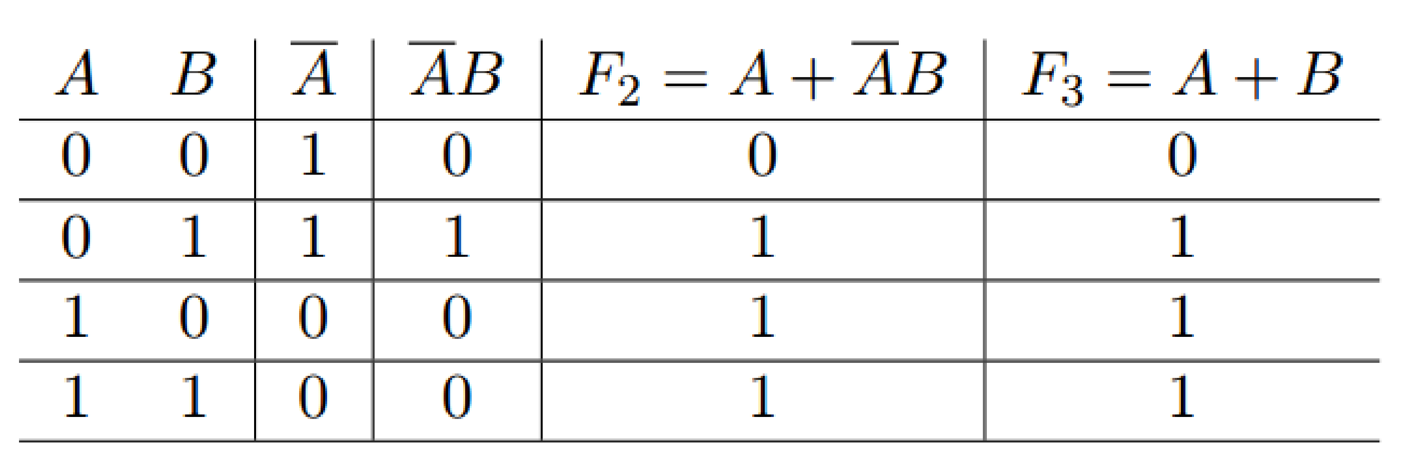
(a’ [c’ + d] + c [b’ + d’] + c’d’)’ = ad (b + c’)

= (a + c d’) (c’ + b d) (c +d) (by **DeMrogan’s** Law)
= (a c’ + a b d) (c + d) (by **distributive** law)
= (a c’ d + a b c d + a b d) (by **distributive** law)
= a c’ d + a b d (by **absorption** of abcd in abd)
= a d (c’ + b) (by **distributive** law)

More Examples:

Using Truth Table, Prove that F_2 & F_3 are equivalent.

$$F_2(A,B) = A + \overline{A}B \text{ and } F_3(A,B) = A + B$$



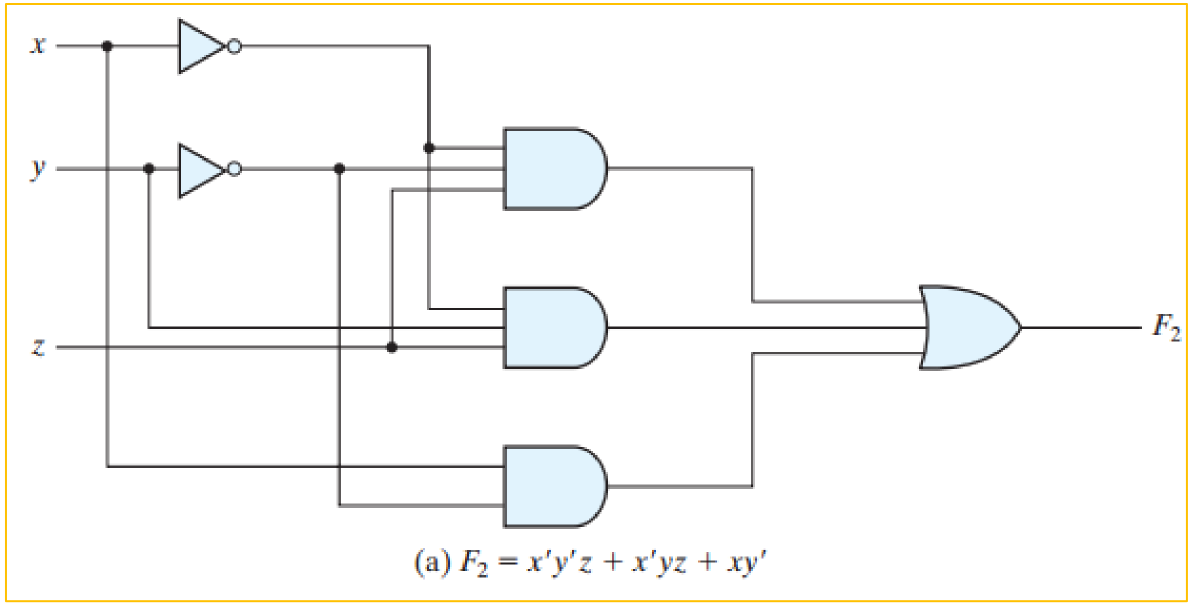
A	B	$\overline{A}$	$\overline{A}B$	$F_2 = A + \overline{A}B$	$F_3 = A + B$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

The last two columns are **identical**. This proves that F2 = F3

Logic Diagrams/Circuits

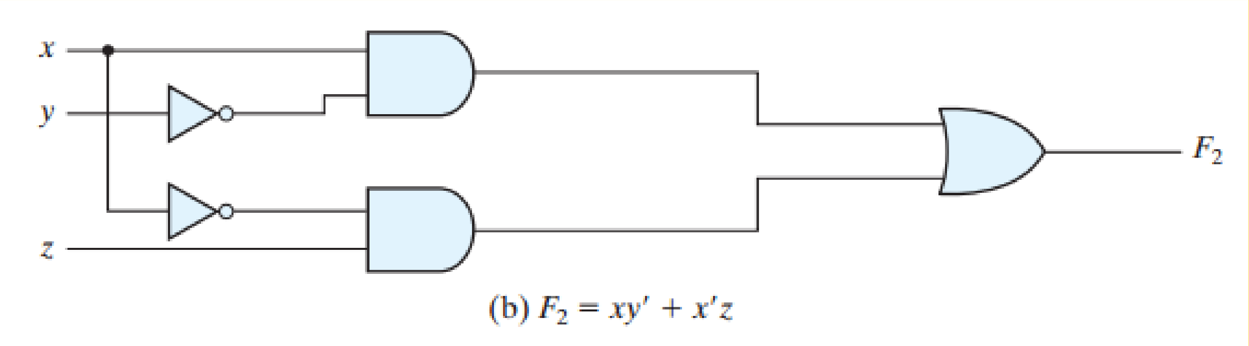
- A Boolean function can be transformed into a **circuit** diagram consisting of **logic gates** connected in a particular structure.

Original Expression



(a) $F_2 = x'y'z + x'yz + xy'$

Simplified Expression



(b) $F_2 = xy' + x'z$

General Observations

- If we have a Boolean function, then there is only **one** way for it to be represented as a truth table
- A Boolean function can be simplified or even modified to **another** expression
- The particular expression used to represent a Boolean function dictates the **structure** of its equivalent logic circuit – and, therefore, the **number** of gates in it
- Conversely, the interconnection of logic gates will determine the logic expression
- We can use the rules of Boolean **algebra** to **simplify** expressions and thus save cost!

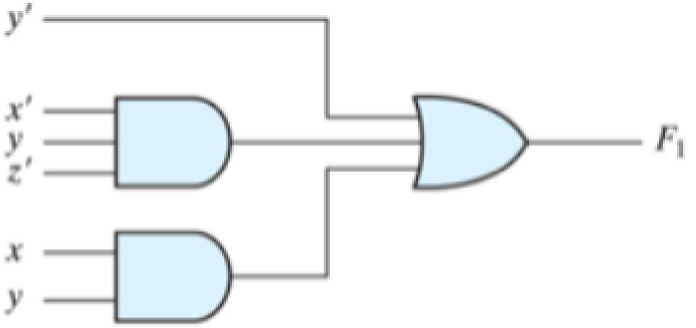
Forms of Functions

Standard forms:

- **Two** Level Implementation (least amount of delay, but may result of too many gate inputs)
- **Terms** of a function may contain one, two, or **any** number of literals/variables
- Two **Types**:
  1) Sum of Products (SOP)
  2) Product of Sums (POS)

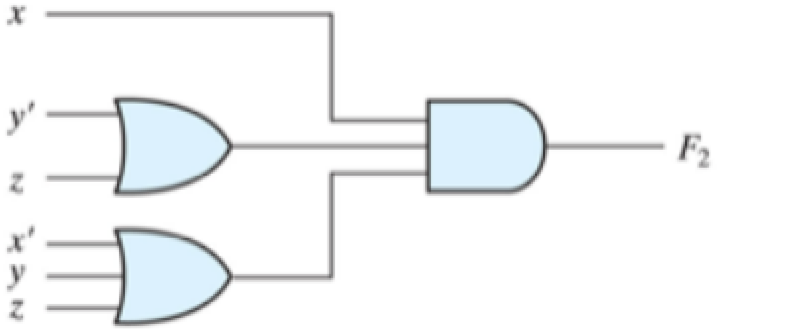
Sum of products

$$F_1 = y' + xy + x'yz'$$



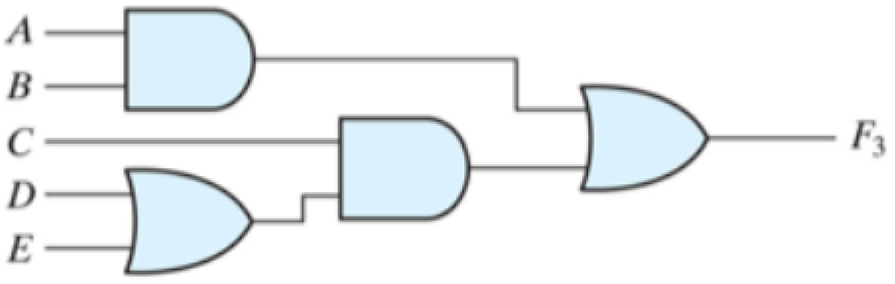
Product of sums

$$F_2 = x(y' + z)(x' + y + z')$$



Nonstandard form:

$F_3 = AB + C (D + E)$



Can be converted to Standard Form

CANONICAL FORM

- A binary variable may appear either in its
  1) **Normal (Unprimed)** form (x)
  2) **Complement** (**Primed**) form (x')

- For **n** variables, there are:
  1) **2^n combinations of (AND)**.
     - Example: for variables x and y , we have $x\,y$, $x'\,y$, $x\,y'$, and $x'\,y'$ (n=2)
  2) **2^n combinations of (OR)**.
     - Example: for variables x and y , we have $x+y$, $x'+y$, $x+y'$, and $x'+y'$ (n=2)

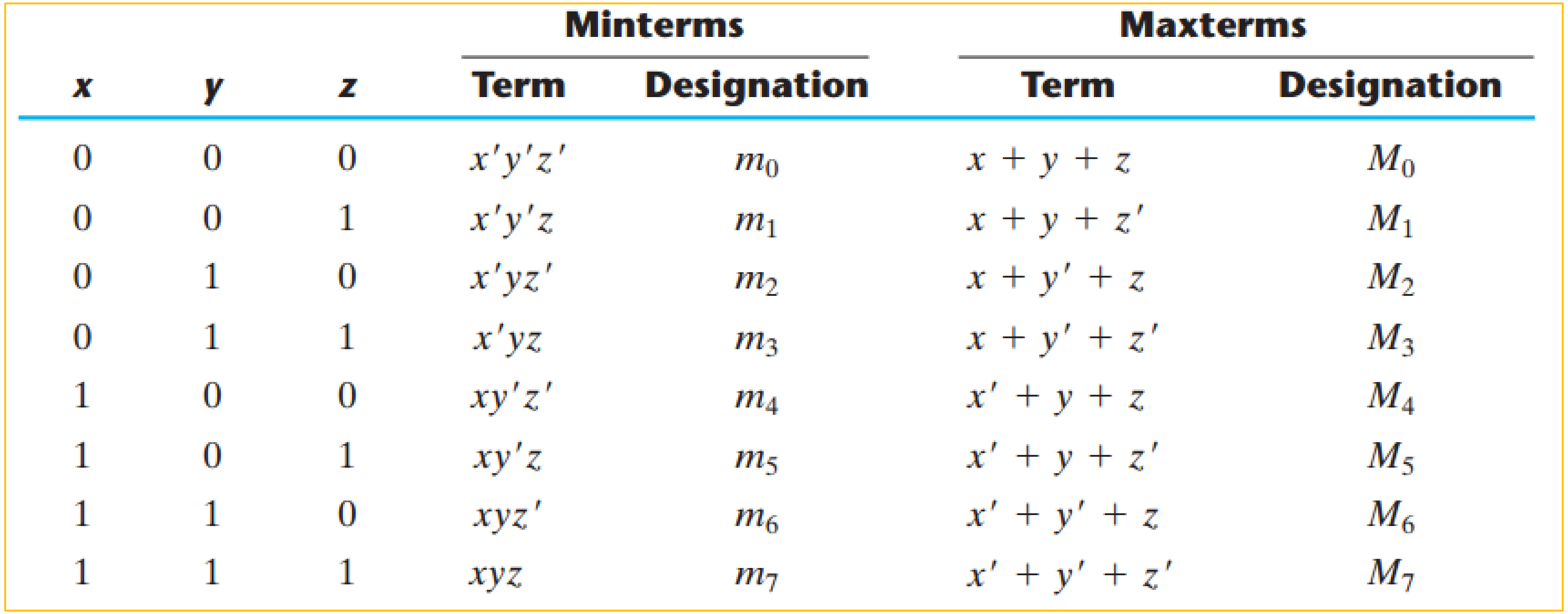
- Each of the AND terms is called **minterm** or **standard product**. Notation: m_j

- Each of the OR terms is called **maxterm** or **standard sum** Notation: M_j

- Each **maxterm** is the **complement** of its corresponding **minterm** (vise versa)

CANONICAL AND STANDARD FORMS

Minterms and Maxterms for Three Binary Variables (n=3)



			Minterms		Maxterms	
x	**y**	**z**	**Term**	**Designation**	**Term**	**Designation**
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Each variable → **primed** if the corresponding bit is a **0** , **unprimed** if the corresponding bit is a **1**

CANONICAL FORM

- **Canonical form**:
  - ✓ Expressing a Boolean function using **sum of minterms** or **product of maxterms**
  - ✓ **All** variables should be present and should be listed in the **same** order

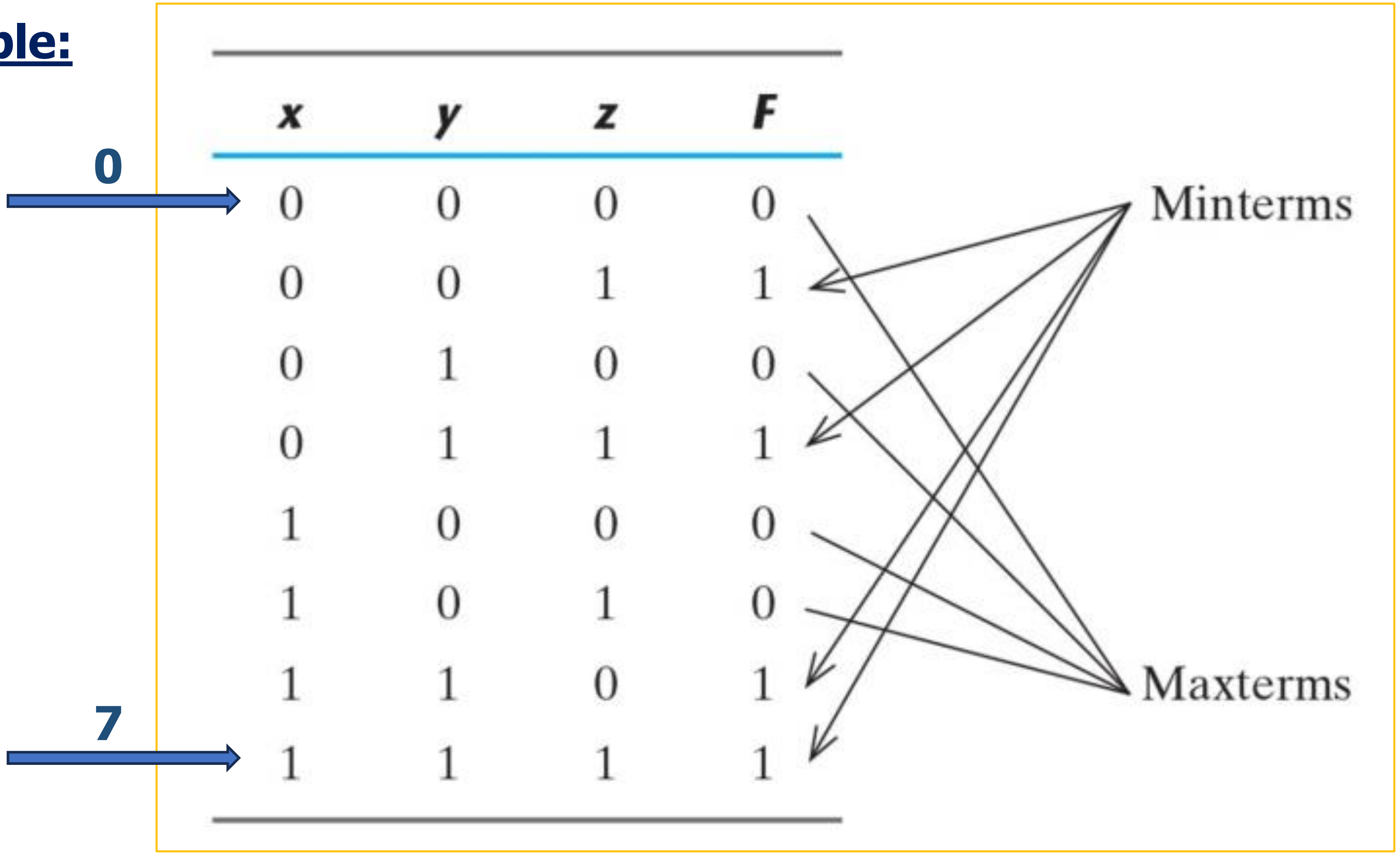
- **Minterms** whose sum defines the Boolean function are those which give **1's** in the truth table

- **Maxterms** whose product defines the Boolean function are those which give **0's** in the truth table

- Maxterm $\mathbf{M_j}$ is the **complement** of minterm $\mathbf{m_j}$

CANONICAL FORM

Example:



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$F = xy + x'z$$

$$F(x, y, z) = x'y'z + x'yz + xyz' + xyz = \Sigma(1, 3, 6, 7)$$

$$F(x, y, z) = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z') = \Pi(0, 2, 4, 5)$$

More Example:

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

$$f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

$$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ = M_0 M_1 M_2 M_4$$

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$f_1 = \sum(1,4,7) \rightarrow f_1' = \sum(0,2,3,5,6) \rightarrow f_1 = \prod(0,2,3,5,6) \rightarrow f_1' = \prod(1,4,7)$$

From any one, we can directly derive the other three

Remember

$$m_j' = M_j$$

→

Hence (e.g.)

$$(m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3$$

More Example:

Represent **F_4** as
(a) sum-of-minterms (**canonical**) form
(b) sum-of-product (**standard**) form with **minimum** number of literals

$$F_4(x,y,z) = \Pi(2,3,6,7)$$

SoM

$$\begin{aligned} F_4(x,y,z) &= \Pi(2,3,6,7) \\ &= \Sigma(0,1,4,5) \\ &= \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}\,z + x\,\overline{y}\,\overline{z} + x\,\overline{y}\,z \end{aligned}$$

SoP - min

$$\begin{aligned} F_4(x,y,z) &= \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}\,z + x\,\overline{y}\,\overline{z} + x\,\overline{y}\,z \\ &= \overline{x}\,\overline{y}\,(\overline{z} + z) + x\,\overline{y}\,(\overline{z} + z) \\ &= \overline{x}\,\overline{y} + x\,\overline{y} \\ &= (\overline{x} + x)\overline{y} \\ &= \overline{y} \end{aligned}$$

Express $F = A + B'C$ as a sum of minterms

⇨ each term should have all variables.

- 1^{st} term missing B & C.

$$= A(B + B') = AB + AB'$$
$$= AB(C + C') + AB'(C + C')$$
$$= ABC + ABC' + AB'C + AB'C'$$

Multiply (AND) By 1

- 2^{nd} term missing A.

$$= B'C(A + A') = AB'C + A'B'C$$
$$F = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$
$$= m_1 + m_4 + m_5 + m_6 + m_7 = \sum(1, 4, 5, 6, 7)$$

Express $F = A + B'C$ as a product of maxterms

➡ convert to OR terms $(A + B')(A + C)$.

- 1^{st} term missing C, $\Rightarrow$ add CC'.

$A + B' = A + B' + CC'$

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

- 2^{nd} term missing B, $\Rightarrow$ add BB'.

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

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

$= m_2 + m_3 + m_0 = \prod(0, 2, 3)$

Add (OR) By 0

Conversion to Canonical Forms

Express the following Boolean expressions in both **canonical** forms

F_1 (A, B, C, D) = (B + D) (A + C)

$$
\begin{aligned}
F_1 &= (AA' + B + CC' + D)\ (A + BB' + C + DD') \\
&= (A+B+C+D)\ (A+B+C'+D)\ (A'+B+C+D)\ (A'+B+C'+D)\ (A+B'+C+D)\ (A+B+C+D')\ (A+B'+C+D') \\
&= \prod (0,1,2,4,5,8,10) \\
&= \sum (3,6,7,9,11,12,13,14,15)
\end{aligned}
$$

Add (OR) By 0

F_2 (A, B, C, D) = A′C (B′ + D)

$$
\begin{aligned}
F_2 &= A'B'C + A'CD \\
&= A'B'C(D+D') + A'(B+B')CD \\
&= A'B'CD' + A'B'CD + A'BCD \\
&= \sum (2,3,7) \\
&= \prod (0,1,\ 4,5,6,8,9,10,11,12,13,14,15)
\end{aligned}
$$

Multiply (AND) By 1

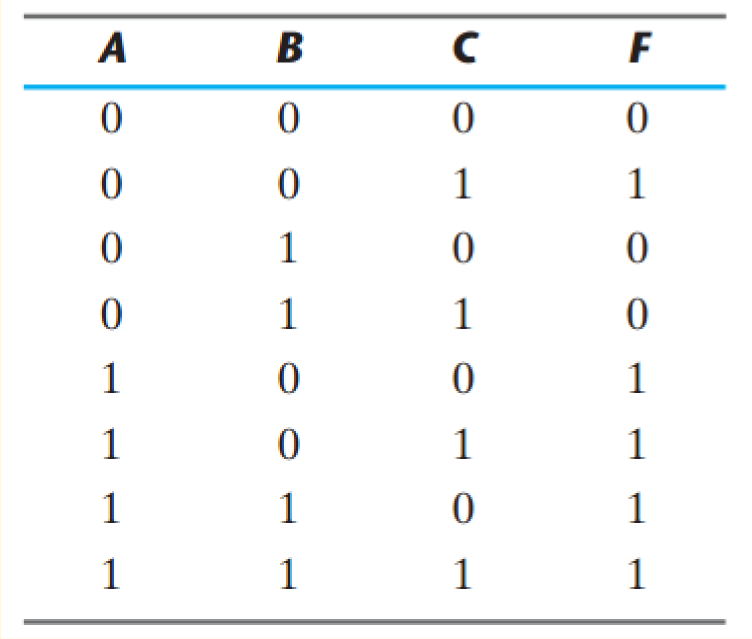
Conversion to Canonical Forms

- An alternative procedure for converting to **Canonical Form**:
  1) Obtain the **Truth Table** of the function **directly** from the **algebraic expression**
  2) Read the minterms/maxterms from the **Truth Table**

Example: $F = A + B'C$

- Derive the Truth Table **directly** from the algebraic expression by:
  1) Listing the eight binary combinations under variables A, B, and C
  2) **Inserting 1's** under F for those combinations for which A=1 **and** BC=01.

- From the truth table, we can **read** the five minterms of the function to be 1, 4, 5, 6, and 7.

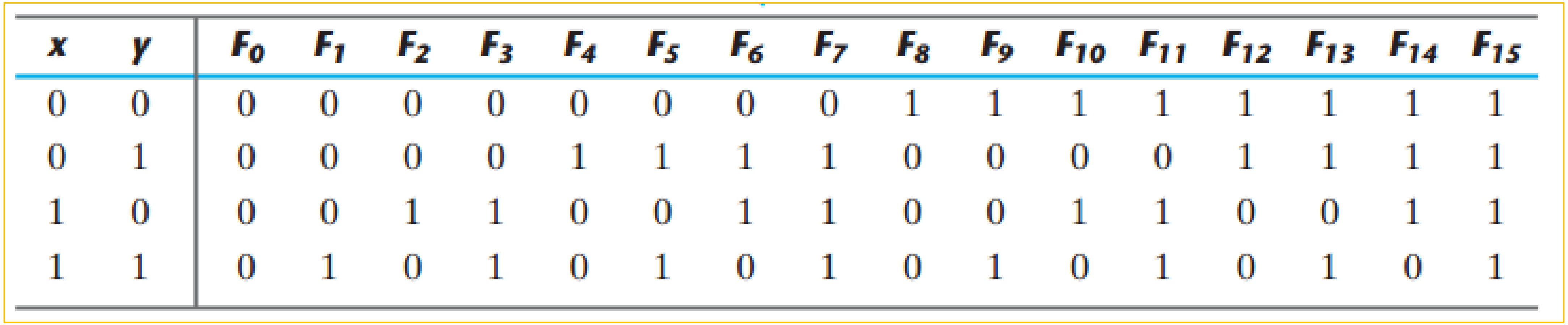


A	*B*	*C*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

OTHER LOGIC OPERATIONS

- For **2** binary inputs we considered so far only the **AND** & **OR** functions
- How many **different** functions can we have for 2 binary inputs (x, y)? **16**
- **In general, for n variables → 2^{2n} Functions (2n is the Number of Rows)**

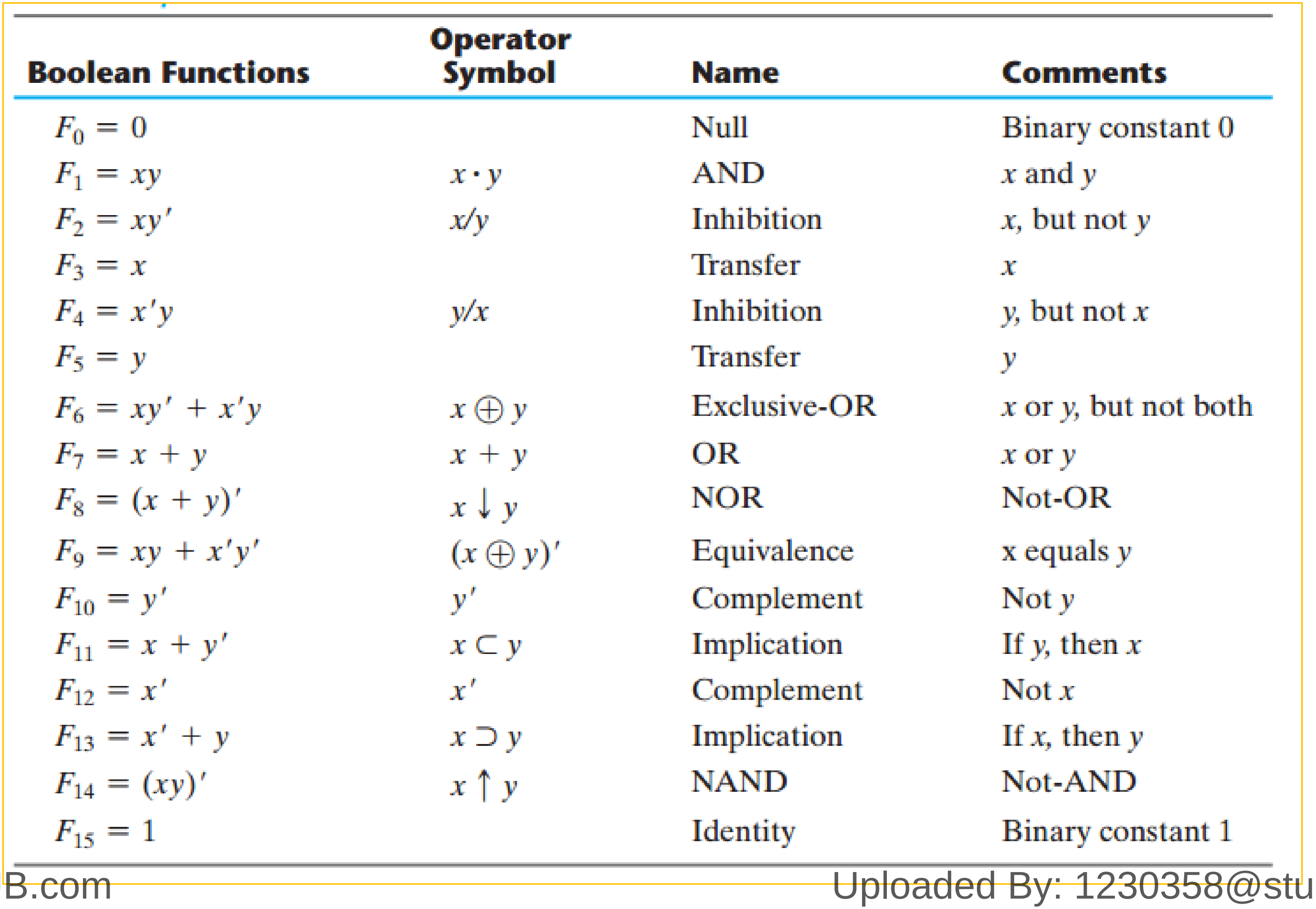
Truth Tables for the 16 Functions of Two Binary Variables



x	*y*	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- ✓ **Two** functions that produce a **constant** 0 or 1.
- ✓ **Four** functions with **unary** operations: complement and transfer.
- ✓ **Ten** functions with **binary** operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

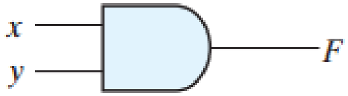
Boolean Expressions for the 16 Functions of Two Variables



Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Digital LOGIC Gates

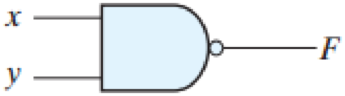
AND



$F = x \cdot y$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

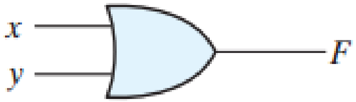
NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

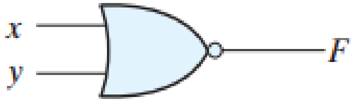
OR



$F = x + y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

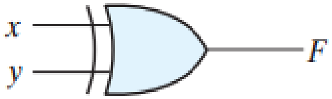
NOR



$F = (x + y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

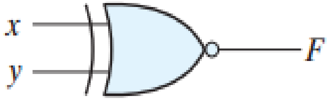
Exclusive-OR
(XOR)



$$F = xy' + x'y = x \oplus y$$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

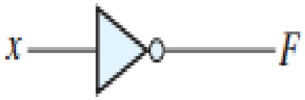
Exclusive-NOR
or
equivalence



$$F = xy + x'y' = (x \oplus y)'$$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

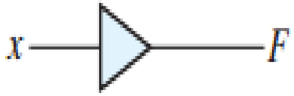
Inverter



$F = x'$

x	F
0	1
1	0

Buffer



$F = x$

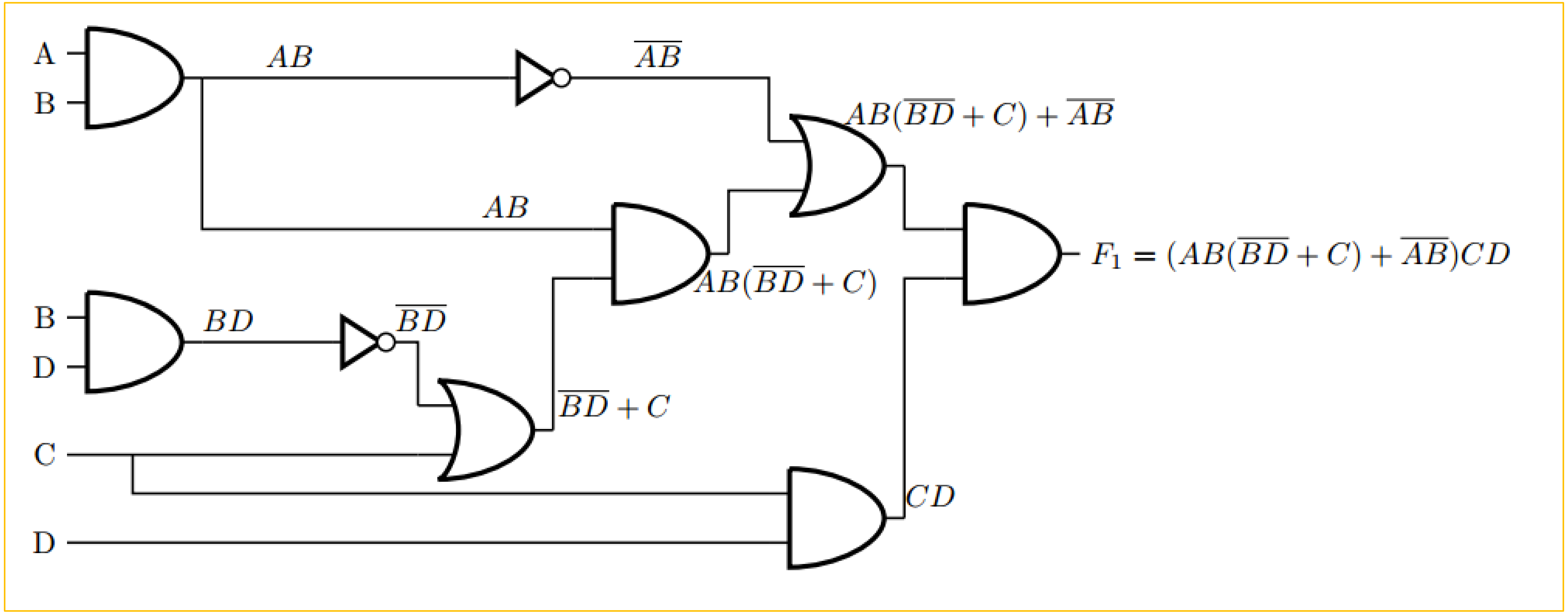
x	F
0	0
1	1

Digital LOGIC Gates

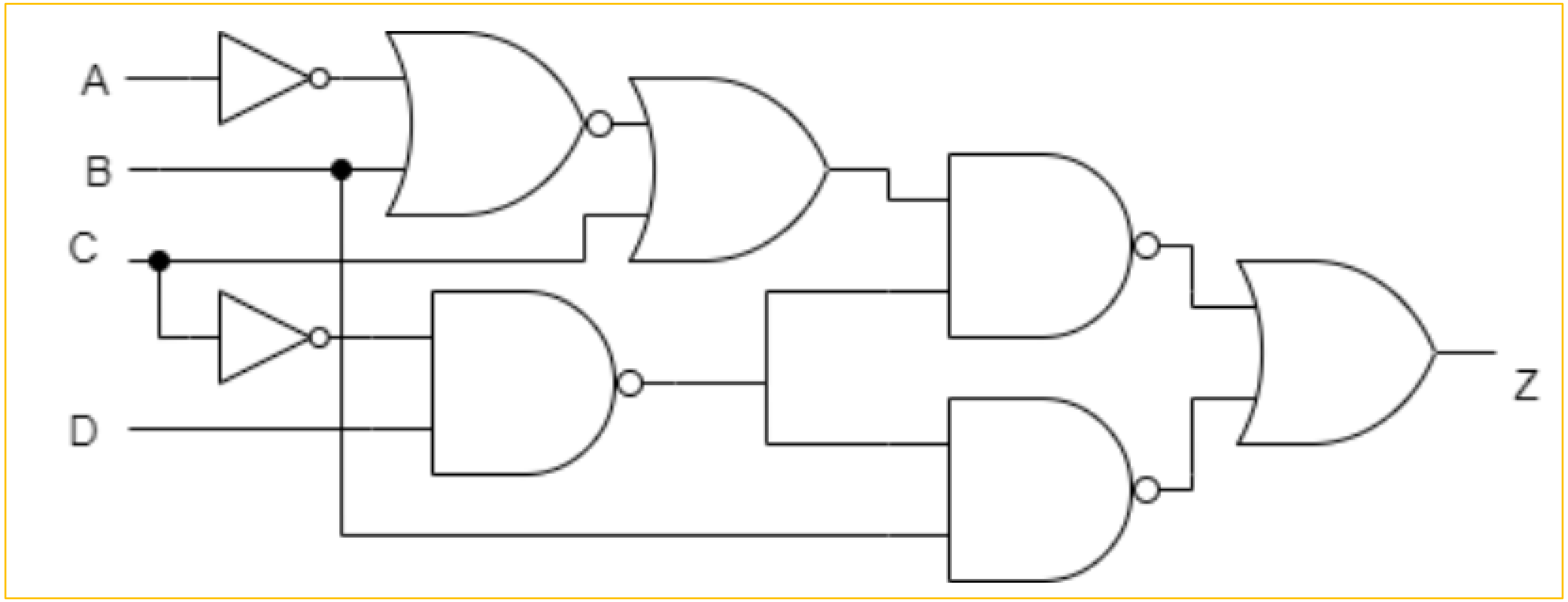
- **NOR** function is the complement of **OR** function (**not-OR**)
- **NAND** function is the complement of **AND** function (**not-AND**)
- **XOR** (exclusive OR) is similar to **OR** but excludes the combination of (x=1 and y=1)
- **Equivalence** is 1 when x and y are **equal**
- **Equivalence** is complement of **XOR**, therefore also called exclusive NOR (**XNOR**)

- **NOR** function is the complement of **OR** function (**not-OR**)
- **NAND** function is the complement of **AND** function (**not-AND**)
- **XOR** (exclusive OR) is similar to **OR** but excludes the combination of (x=1 and y=1)
- **Equivalence** is 1 when x and y are **equal**
- **Equivalence** is complement of **XOR**, therefore also called exclusive NOR (**XNOR**)

Example:


A
B
AB
$\overline{AB}$
$AB(\overline{BD}+C)+\overline{AB}$
AB
$F_1 = (AB(\overline{BD}+C)+\overline{AB})CD$
$AB(\overline{BD}+C)$
B
D
BD
$\overline{BD}$
$\overline{BD}+C$
C
CD
D

Extra Example:



$$
\begin{aligned}
Z &= [[(A' + B)' + C](C'D)']'+[(C'D)'.B]' \\
&= [[(A' + B)'+ C]'+(C'D)] + [(C'D)+B'] \\
&= [(A' + B).C']+[(C'D) + (C'D)+B'] \\
&= A'C' + BC' + C'D + B' \\
&= A'C' + C'D + BC' + B' \\
&= A'C' + C'D + C' + B' \\
&= C' (A'+D+1)+ B' \\
&= C' + B'
\end{aligned}
$$

$$
\boxed{x + yz = (x + y)(x + z)}
$$

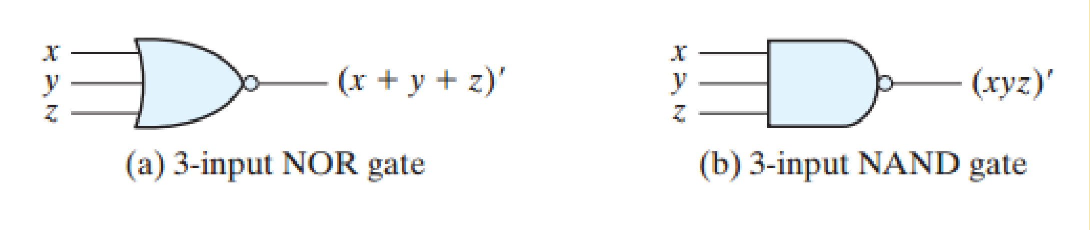
Extension of Gates to Multiple Inputs

- **All** gates except inverter and buffer — can be **extended** to have **more** than two inputs
- **NAND** and **NOR** are **commutative** but **not associative**

$$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$$

- We define the **multiple** NOR / NAND gate as a **complemented** OR / AND gate.

$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

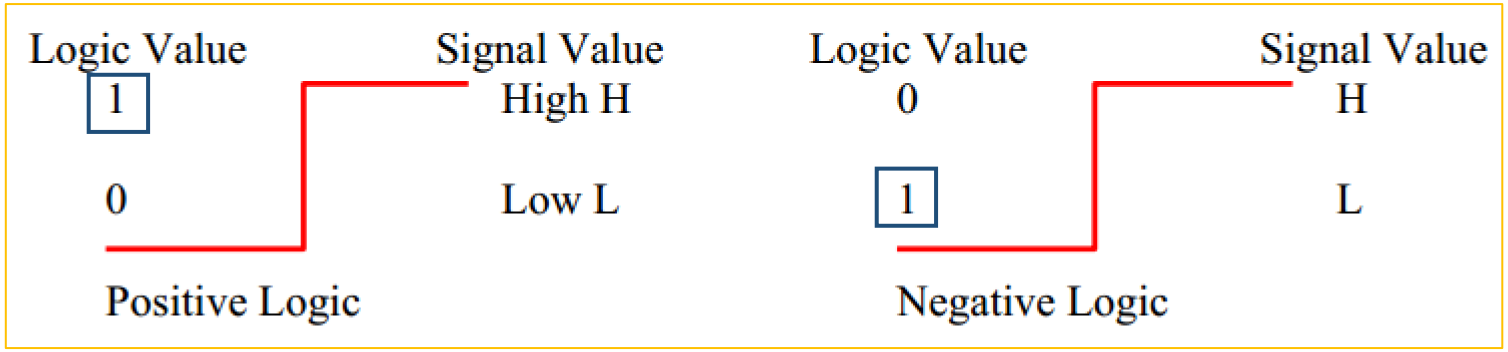


(a) 3-input NOR gate

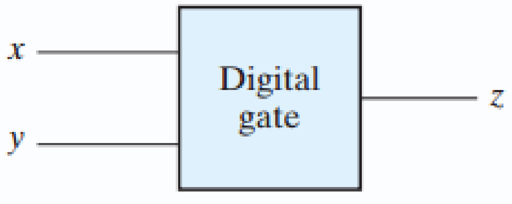
(b) 3-input NAND gate

Positive and Negative Logic

- Choosing the **high-level H** to represent **logic 1** defines a ***positive logic*** system
- Choosing the **low-level L** to represent **logic 1** defines a ***negative logic*** system

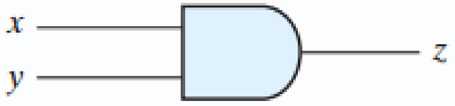


x	*y*	*z*
L	*L*	*L*
L	*H*	*L*
H	*L*	*L*
H	*H*	*H*



H = 1 **Positive logic AND gate**

x	*y*	*z*
0	0	0
0	1	0
1	0	0
1	1	1



L = 1 **Negative logic OR gate**

x	*y*	*z*
1	1	1
1	0	1
0	1	1
0	0	0