INTRODUCTION TO DIGITAL SYSTEM DESIGN

Computers and Electricity

Gate

A device that performs a basic operation on electrical signals

Circuits

Gates combined to perform more complicated tasks

How do we describe the behavior of gates and circuits?

Boolean expressions

Uses Boolean algebra, a mathematical notation for expressing two-valued logic

Logic diagrams

A graphical representation of a circuit; each gate has its own symbol

Truth tables

A table showing all possible input value and the associated output values

Gates

Seven types of gates

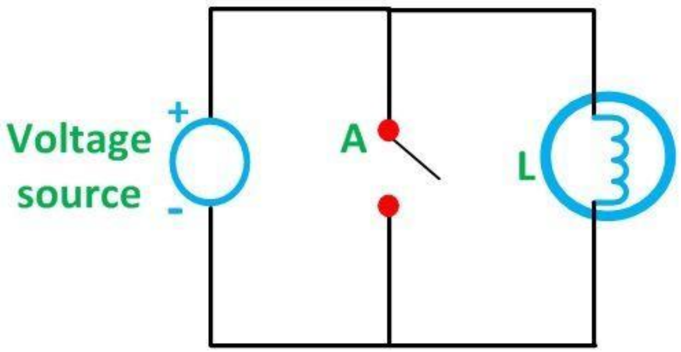
- NOT
- AND
- OR
- XOR
- NAND
- NOR
- XNOR

Typically, logic diagrams are black and white with gates distinguished only by their shape

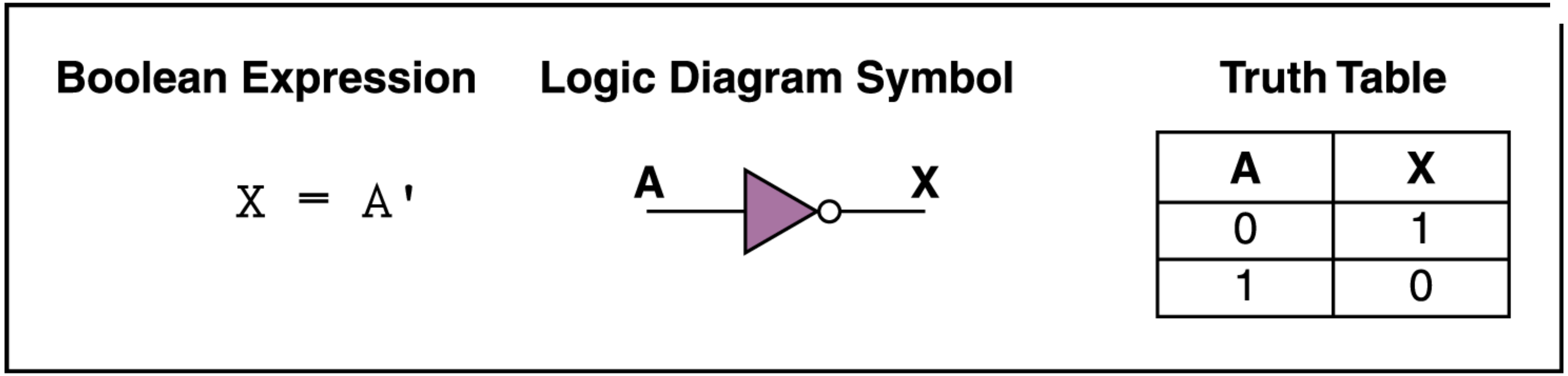
We use color for emphasis (and fun)

NOT Gate

A NOT gate accepts one input signal (0 or 1) and returns the opposite signal as output



Circuit Globe



Boolean Expression

$X = A'$

Logic Diagram Symbol

Truth Table

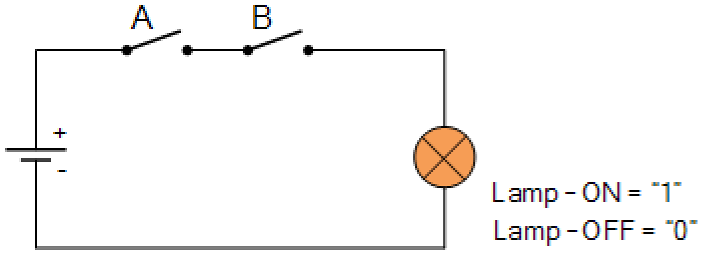
A	X
0	1
1	0

Figure 4.1 Various representations of a NOT gate

AND Gate

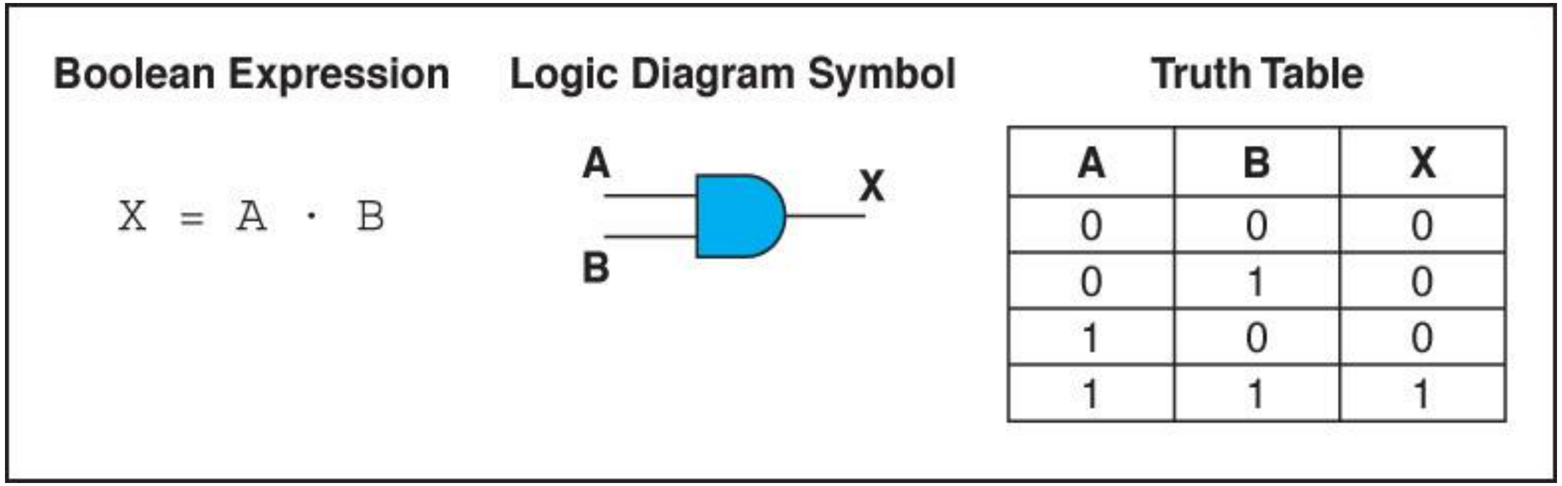
An AND gate accepts two input signals

If both are 1, the output is 1; otherwise, the output is 0



Lamp - ON = "1"
Lamp - OFF = "0"

Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"



Boolean Expression

$X = A \cdot B$

Logic Diagram Symbol

Truth Table

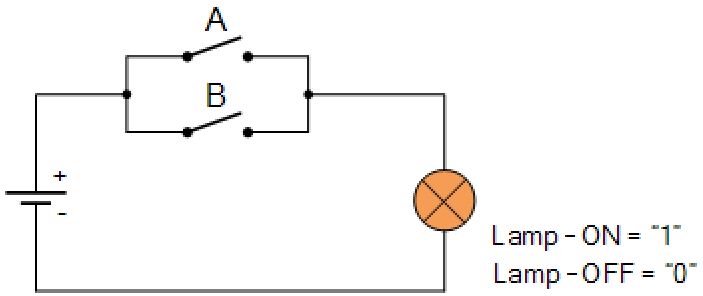
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2 Various representations of an AND gate

OR Gate

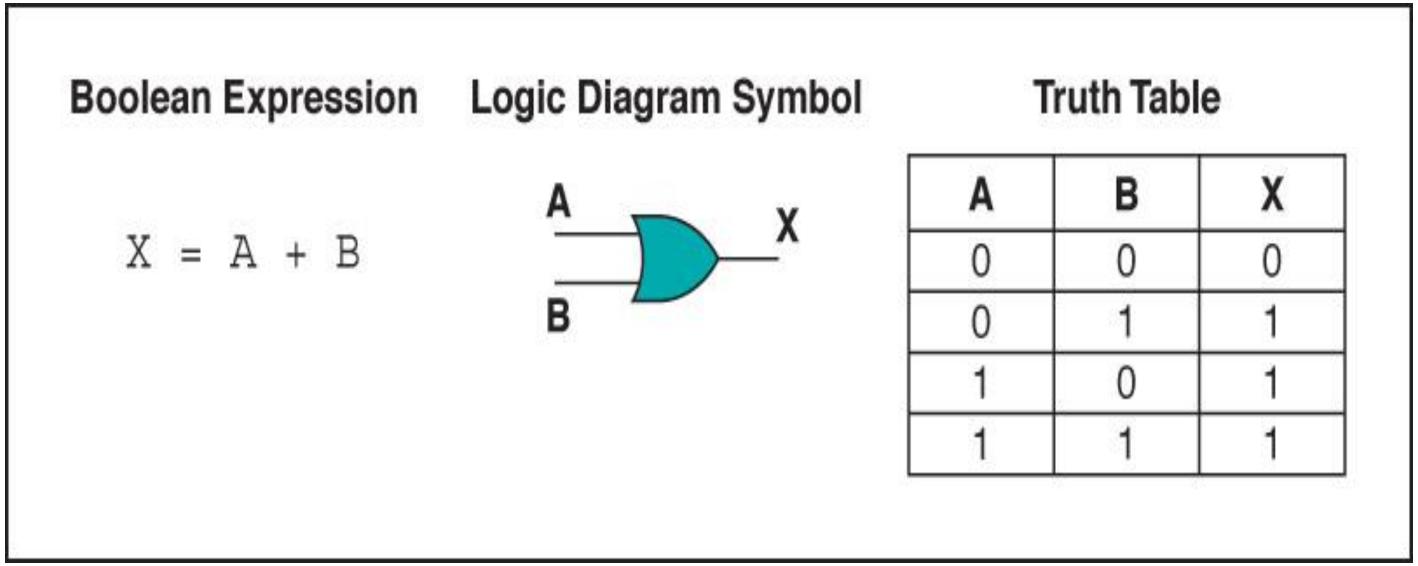
An OR gate accepts two input signals

If both are 0, the output is 0; otherwise, the output is 1



Lamp - ON = "1"
Lamp - OFF = "0"

Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"



Boolean Expression

$X = A + B$

Logic Diagram Symbol

Truth Table

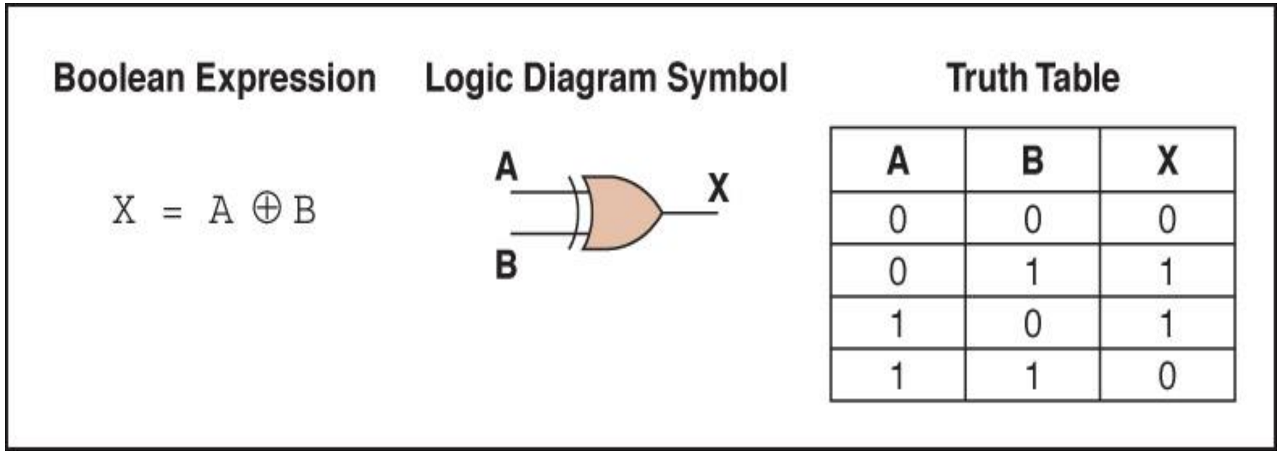
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3. Various representations of a OR gate

XOR Gate

An XOR gate accepts two input signals

If both are the same, the output is 0; otherwise, the output is 1



Boolean Expression

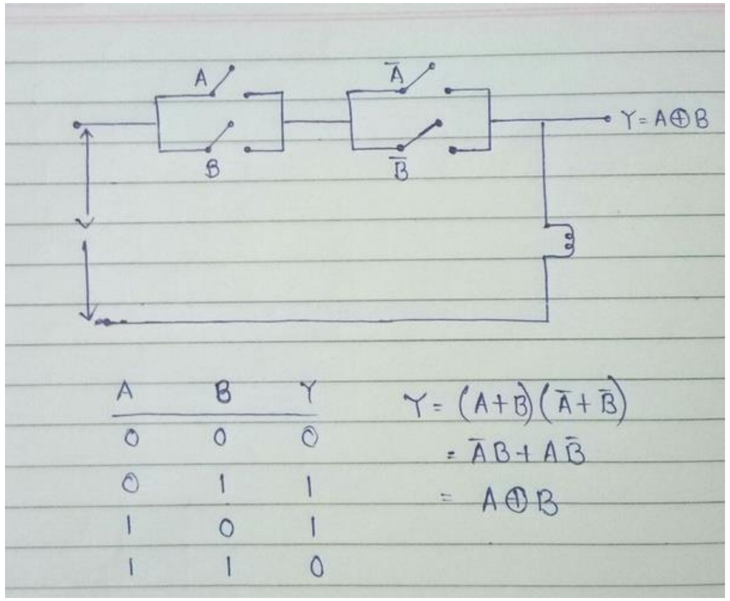
$X = A \oplus B$

Logic Diagram Symbol

Truth Table

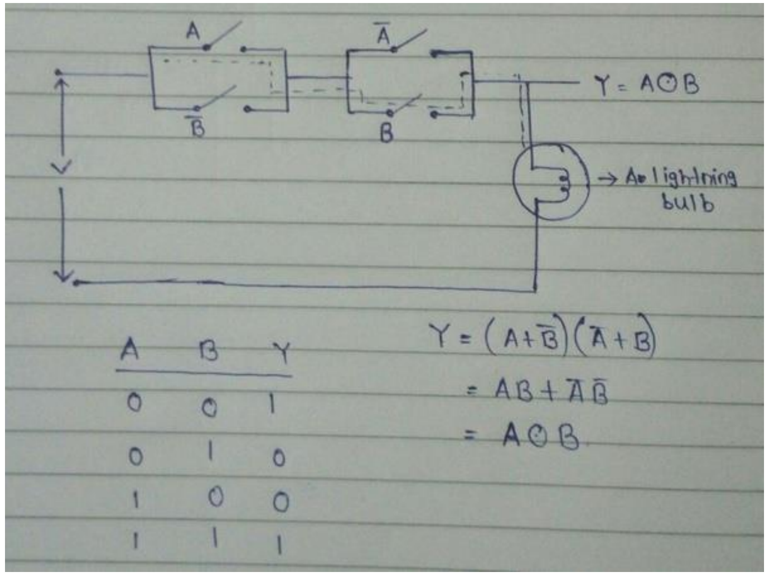
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4. Various representations of an XOR gate



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = (A+B)(\bar{A}+\bar{B})$$
$$= \bar{A}B + A\bar{B}$$
$$= A \oplus B$$



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

$$Y = (A+\bar{B})(\bar{A}+B)$$
$$= AB + \bar{A}\bar{B}$$
$$= A \odot B$$

XOR Gate

Note the difference between the XOR gate and the OR gate; they differ only in one input situation

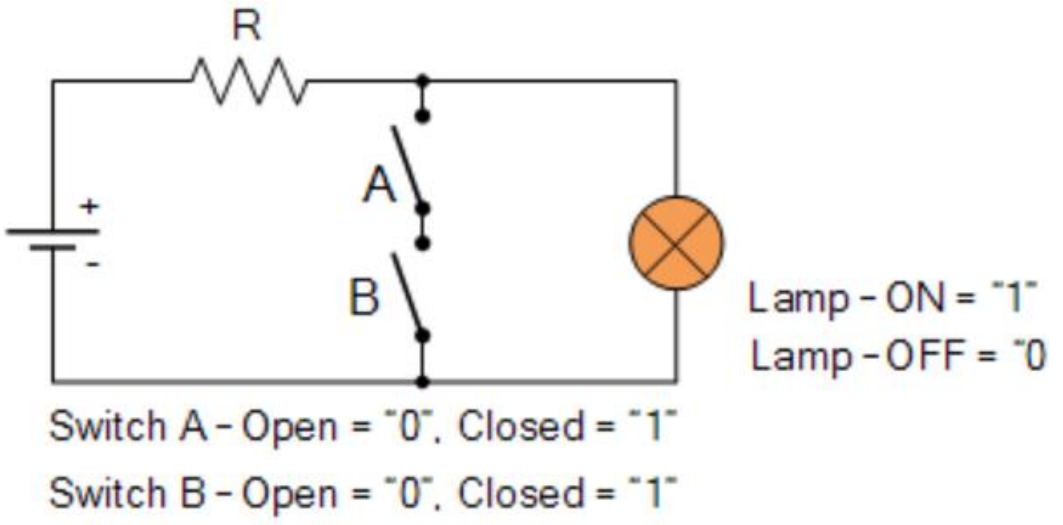
When both input signals are 1, the OR gate produces a 1 and the XOR produces a 0

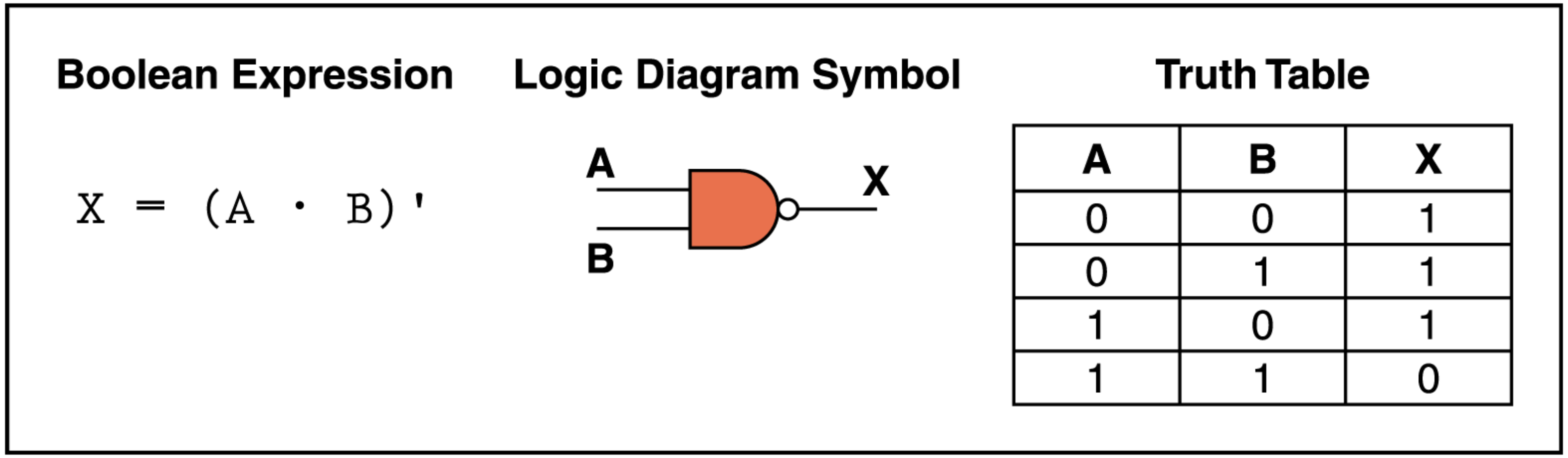
XOR is called the *exclusive OR*

NAND Gate

The NAND gate accepts two input signals

If both are 1, the output is 0; otherwise, the output is 1


R
A
B
+
-
Lamp - ON = "1"
Lamp - OFF = "0
Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"



Boolean Expression

$X = (A \cdot B)'$

Logic Diagram Symbol

Truth Table

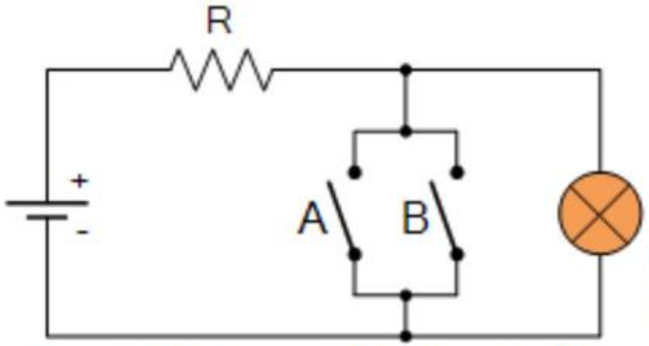
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Figure 5. Various representations of a NAND gate

NOR Gate

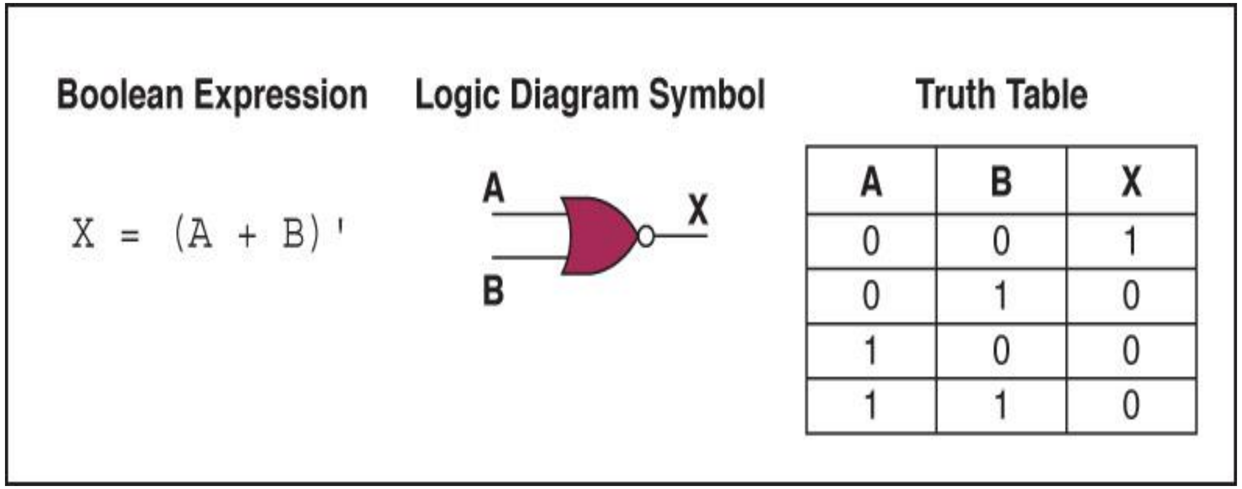
The NOR gate accepts two input signals

If both are 0, the output is 1; otherwise, the output is 0



Lamp - ON = "1"
Lamp - OFF = "0

Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"



Boolean Expression

$X = (A + B)'$

Logic Diagram Symbol

Truth Table

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Figure 6 Various representations of a NOR gate

Review of Gate Processing

A NOT gate inverts its single input

An AND gate produces 1 if both input values are 1

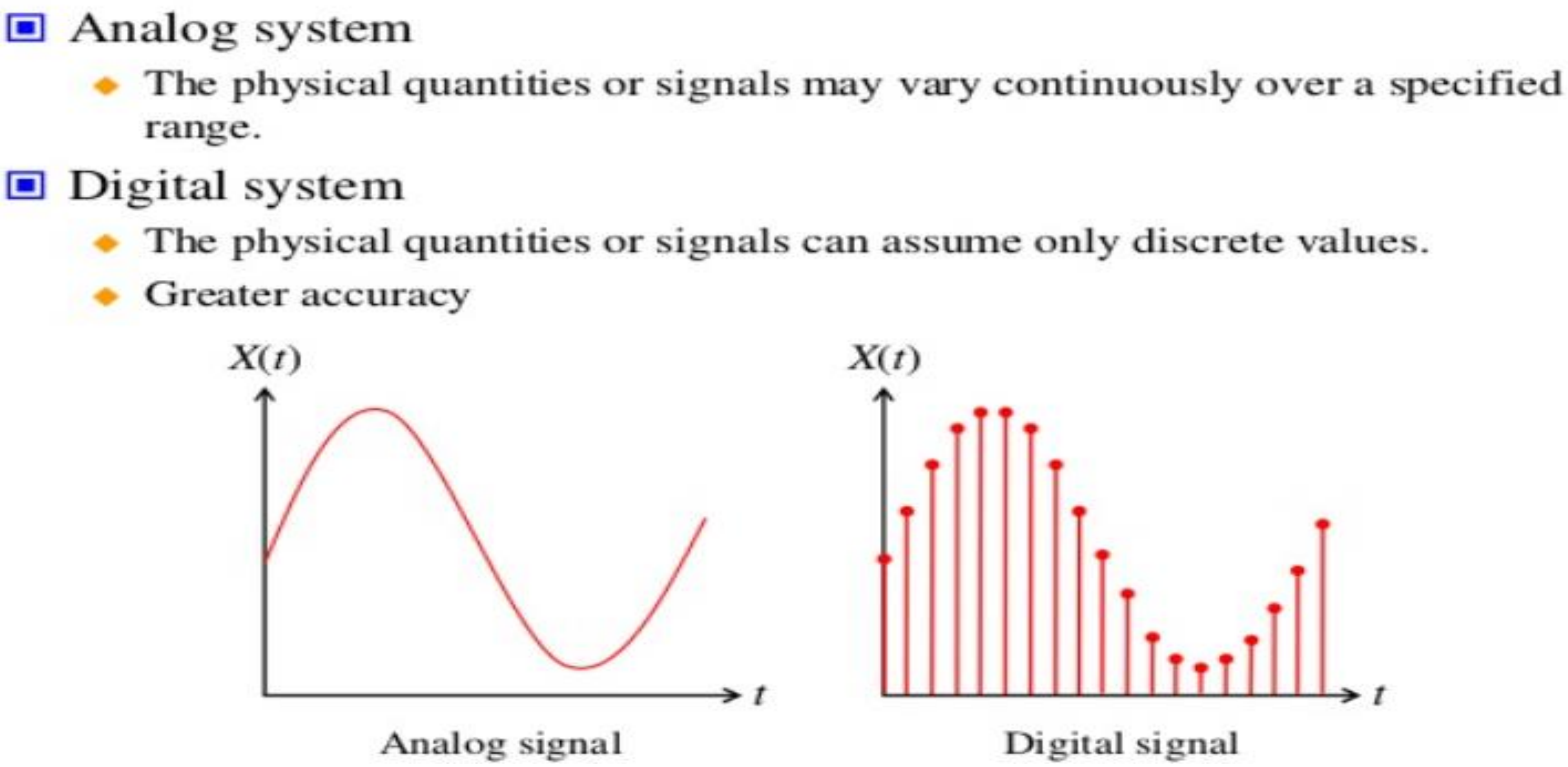
An OR gate produces 0 if both input values are 0

An XOR gate produces 0 if input values are the same

A NAND gate produces 0 if both inputs are 1

A NOR gate produces a 1 if both inputs are 0

Analog vs Digital system

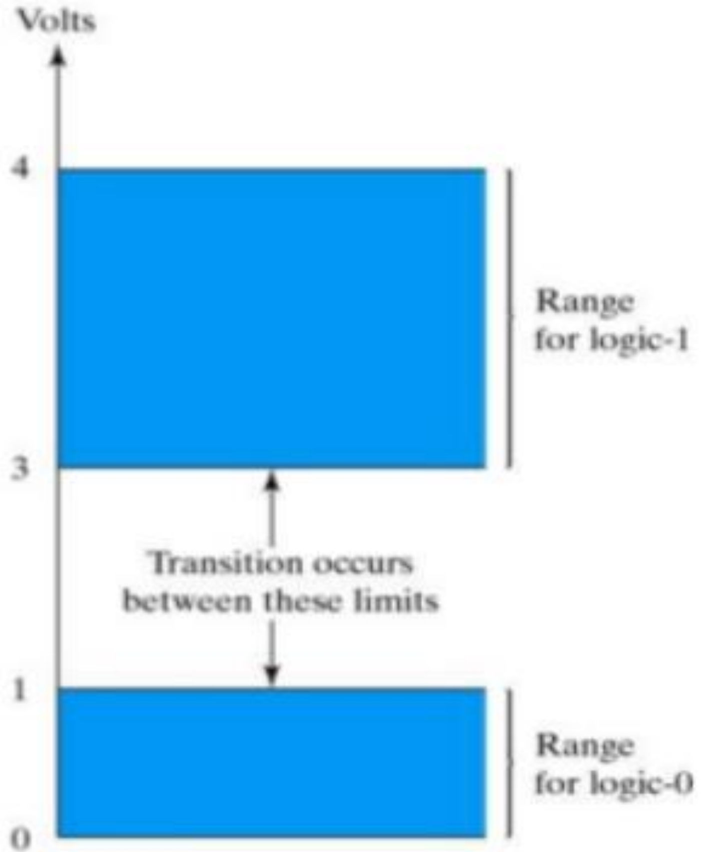


- ▣ Analog system
  - The physical quantities or signals may vary continuously over a specified range.
- ▣ Digital system
  - The physical quantities or signals can assume only discrete values.
  - Greater accuracy

Binary information is represented in digital computer by physical quantities called signal.

Logic Signals

- Binary ‘0’ is represented by a “*low*” voltage (range of voltages)
- Binary ‘1’ is represented by a “*high*” voltage (range of voltages)
- The “voltage ranges” guard against noise



Example of binary signals

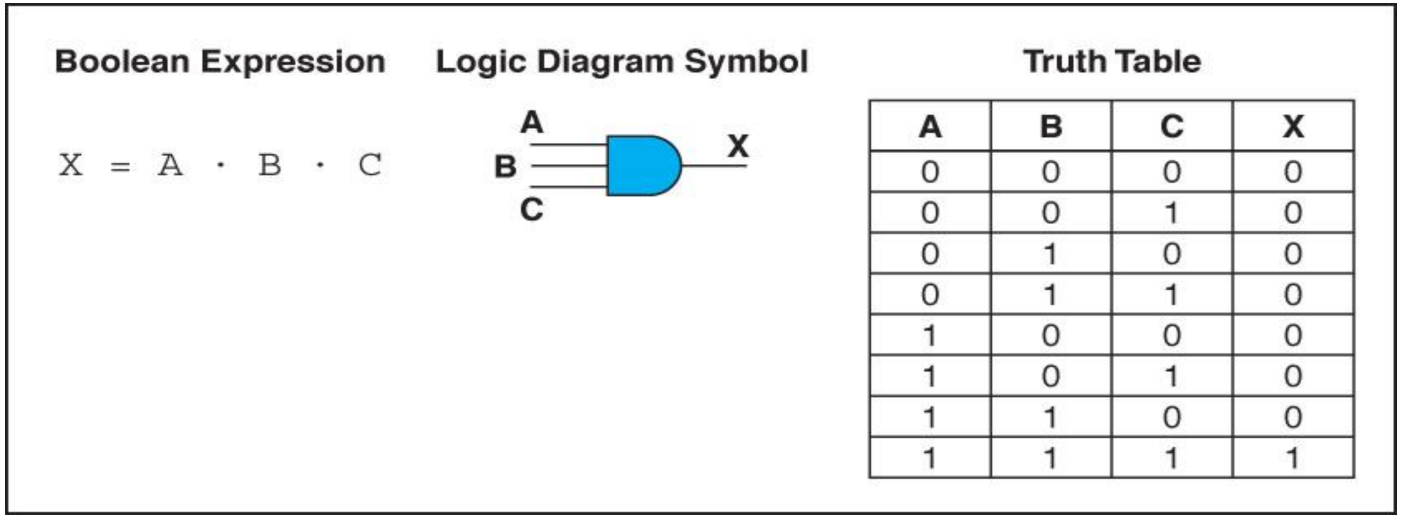
Digital Computer

- Digital computer is a Digital system that perform various computational task.
- It use Binary number system to represent the information, i.e. 0 and 1.
- Binary Digits 0 or 1 is called a bit.
- Information is represented in digital computer is a group of bits.
- Gates: Manipulation of binary information is done by logic circuit called gates.
- Boolean Algebra: It is an algebra that deals with binary variables and logic operations.

Gates with More Inputs

Gates can be designed to accept three or more input values

A three-input AND gate, for example, produces an output of 1 only if all input values are 1



Boolean Expression

$X = A \cdot B \cdot C$

Logic Diagram Symbol

Truth Table

A	B	C	X
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

Figure 7 Various representations of a three-input AND gate

Constructing Gates

Transistor

A device that acts either as a wire that conducts electricity or as a resistor that blocks the flow of electricity, depending on the voltage level of an input signal

A transistor has no moving parts, yet acts like a switch

It is made of a semiconductor material, which is neither a particularly good conductor of electricity nor a particularly good insulator

Constructing Gates

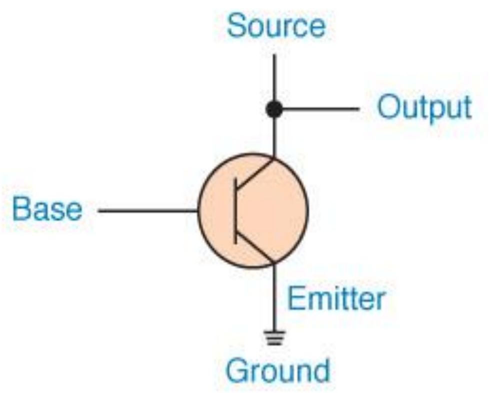


Figure 8 The connections of a transistor

A transistor has three terminals

- A source
- A base
- An emitter, typically connected to a ground wire

If the electrical signal is grounded, it is allowed to flow through an alternative route to the ground (literally) where it can do no harm

Constructing Gates

The easiest gates to create are the NOT, NAND, and NOR gates

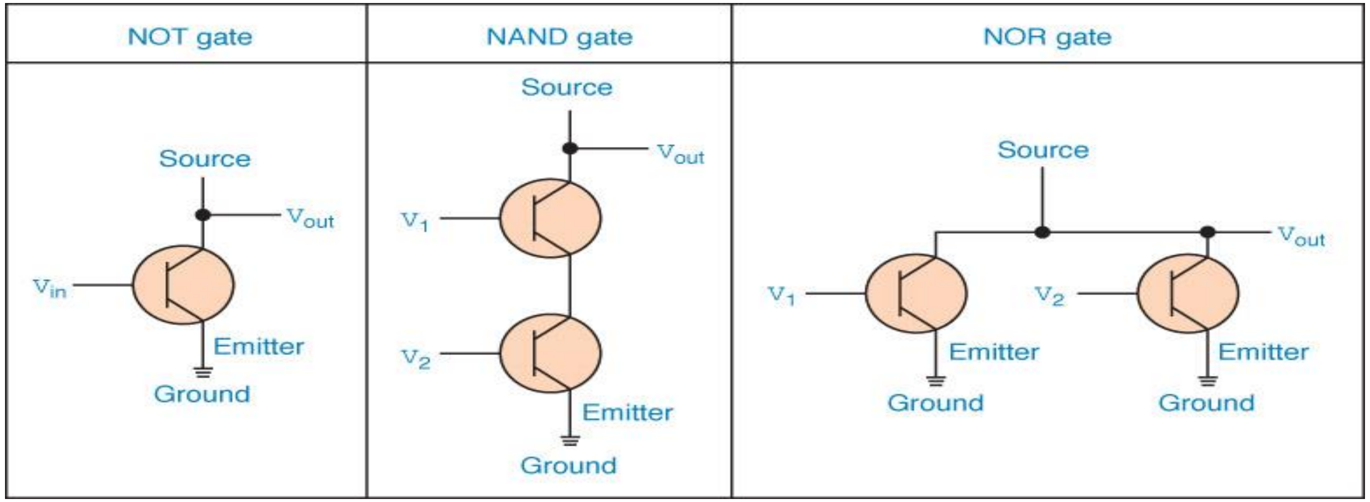


Figure 9 Constructing gates using transistors

Circuits

Combinational circuit

The input values explicitly determine the output

Sequential circuit

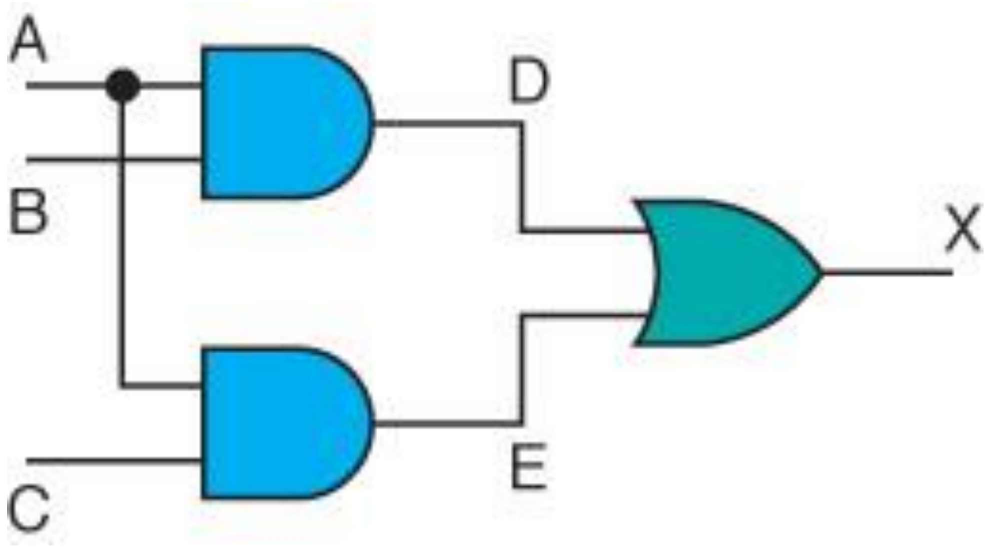
The output is a function of the input values and the existing state of the circuit

We describe the circuit operations using

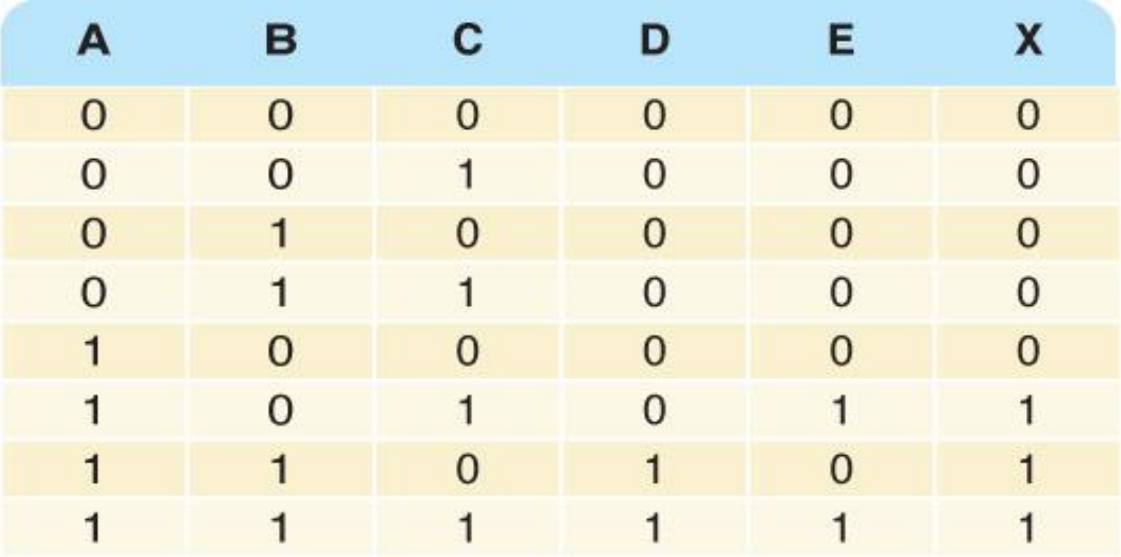
- Boolean expressions
- Logic diagrams
- Truth tables

Combinational Circuits

Gates are combined into circuits by using the output of one gate as the input for another



Combinational Circuits



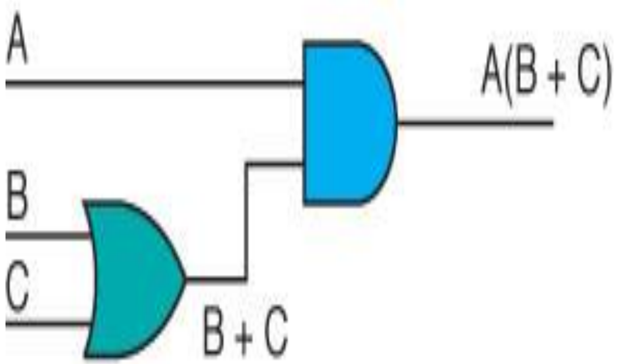
A	B	C	D	E	X
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

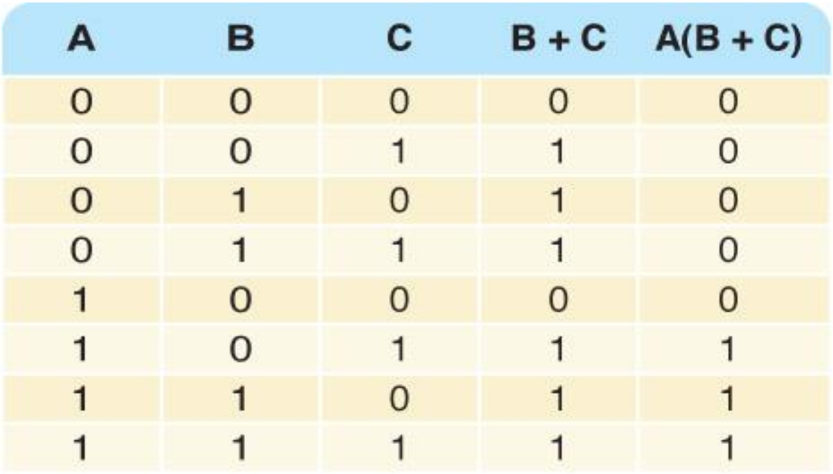
Three inputs require eight rows to describe all possible input combinations

This same circuit using a Boolean expression is $(AB + AC)$

Combinational Circuits

Consider the following Boolean expression $A(B + C)$





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

Does this truth table look familiar?

Compare it with previous table

Combinational Circuits

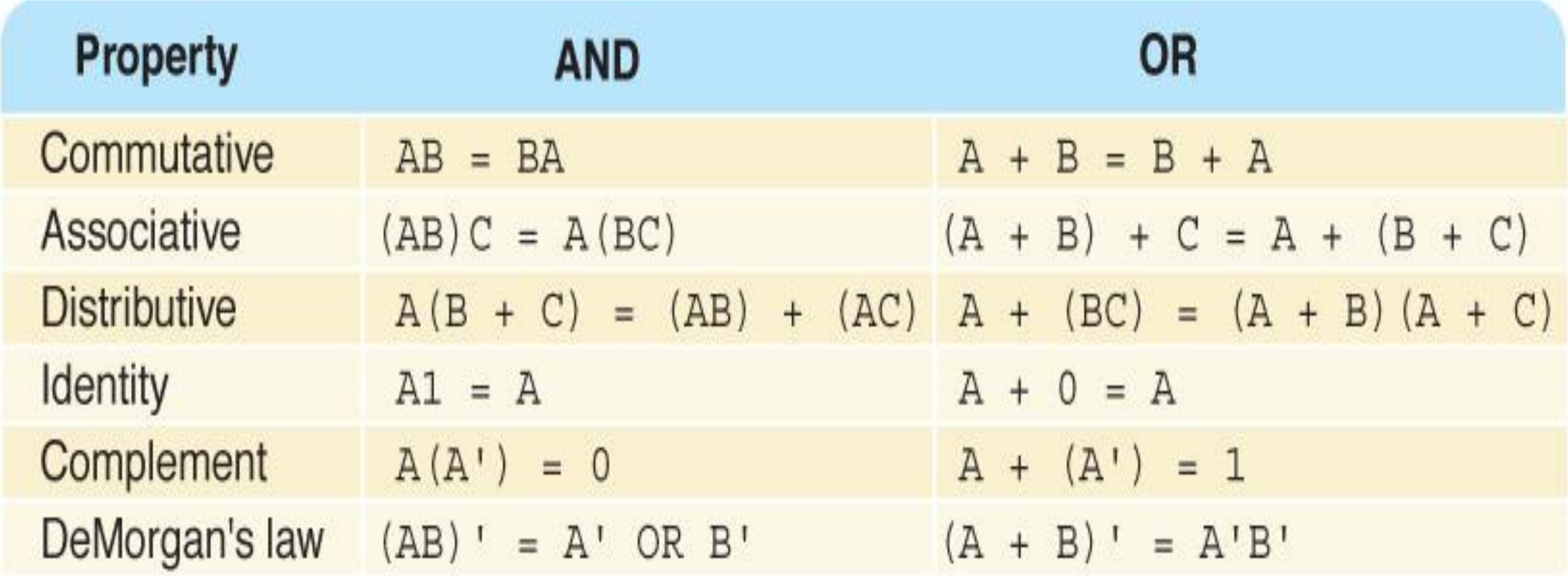
Circuit equivalence

Two circuits that produce the same output for identical input

Boolean algebra allows us to apply provable mathematical principles to help design circuits

A(B + C) = AB + BC (distributive law) so circuits must be equivalent

Properties of Boolean Algebra

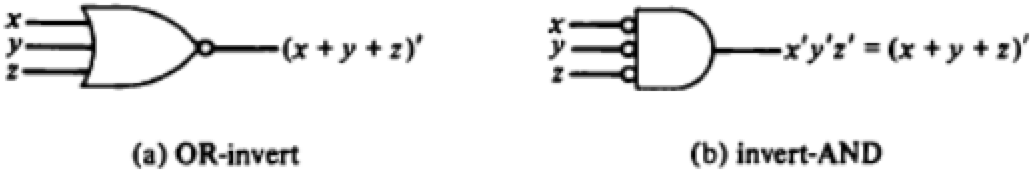


Property	AND	OR
Commutative	AB = BA	A + B = B + A
Associative	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive	A(B + C) = (AB) + (AC)	A + (BC) = (A + B)(A + C)
Identity	A1 = A	A + 0 = A
Complement	A(A') = 0	A + (A') = 1
DeMorgan's law	(AB)' = A' OR B'	(A + B)' = A'B'

TABLE 1-1 Basic Identities of Boolean Algebra

(1) $x + 0 = x$	(2) $x \cdot 0 = 0$
(3) $x + 1 = 1$	(4) $x \cdot 1 = x$
(5) $x + x = x$	(6) $x \cdot x = x$
(7) $x + x' = 1$	(8) $x \cdot x' = 0$
(9) $x + y = y + x$	(10) $xy = yx$
(11) $x + (y + z) = (x + y) + z$	(12) $x(yz) = (xy)z$
(13) $x(y + z) = xy + xz$	(14) $x + yx = (x + y)(x + z)$
(15) $(x + y)' = x'y'$	(16) $(xy)' = x' + y'$
(17) $(x')' = x$	

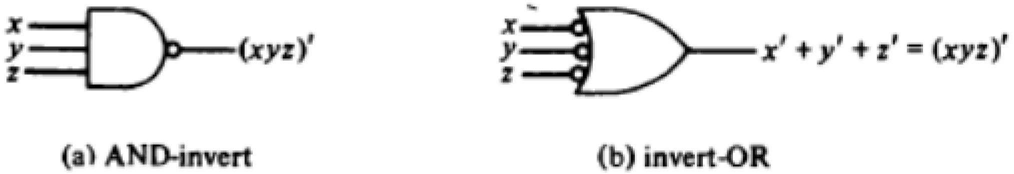
Figure 1-4 Two graphic symbols for NOR gate.



(a) OR-invert

(b) invert-AND

Figure 1-5 Two graphic symbols for NAND gate.

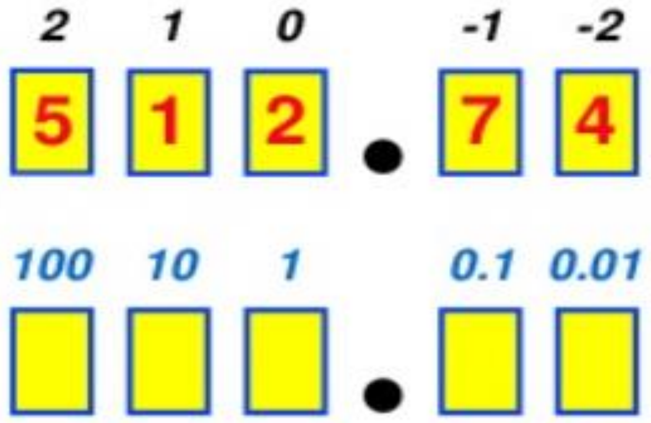


(a) AND-invert

(b) invert-OR

Decimal Number System

- ★ **Base (also called radix) = 10**
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- ★ **Digit Position**
  - Integer & fraction
- ★ **Digit Weight**
  - Weight = (*Base*) $^{\textit{Position}}$
- ★ **Magnitude**
  - Sum of "*Digit* x *Weight*"
- ★ **Formal Notation**



2 1 0 . -1 -2

5 1 2 . 7 4

100 10 1 . 0.1 0.01

500 10 2 0.7 0.04

$d_2{}^*B^2+d_1{}^*B^1+d_0{}^*B^0+d_{-1}{}^*B^{-1}+d_{-2}{}^*B^{-2}$

$(512.74)_{10}$

Octal Number System

- ★ **Base = 8**
  - **8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }**
- ★ **Weights**
  - **Weight = (*Base*)** $^{\textit{Position}}$
- ★ **Magnitude**
  - **Sum of "*Digit* x *Weight*"**
- ★ **Formal Notation**

64	*8*	*1*		*1/8*	*1/64*
5	1	2	●	7	4
2	*1*	*0*		*-1*	*-2*

$5*8^2+1*8^1+2*8^0+7*8^{-1}+4*8^{-2}$

$=(330.9375)_{10}$

$(512.74)_8$

Binary Number System

★ **Base = 2**

- **2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"**

★ **Weights**

- **Weight = (*Base*)** ^{Position}

$$\begin{array}{cccccc} 4 & 2 & 1 & & 1/2 & 1/4 \\ \boxed{1} & \boxed{0} & \boxed{1} & \bullet & \boxed{0} & \boxed{1} \\ 2 & 1 & 0 & & -1 & -2 \end{array}$$

★ **Magnitude**

- **Sum of "*Bit* x *Weight*"**

$$1*2^2+0*2^1+1*2^0+0*2^{-1}+1*2^{-2}$$

$$=(5.25)_{10}$$

★ **Formal Notation**

$$(101.01)_2$$

★ **Groups of bits**

4 bits = *Nibble* $\boxed{1\ 0\ 1\ 1}$

8 bits = *Byte* $\boxed{1\ 1\ 0\ 0\ 0\ 1\ 0\ 1}$

Hexadecimal Number System

★ **Base = 16**

- **16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }**

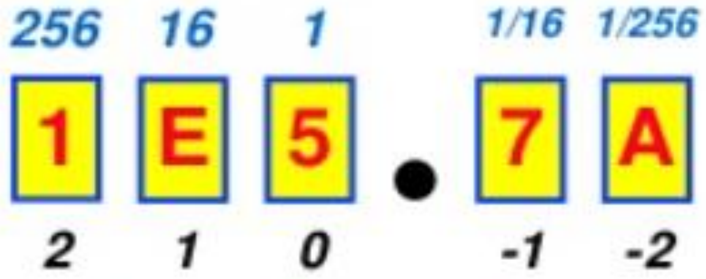
★ **Weights**

- **Weight = (*Base*)** $^{\textit{Position}}$

★ **Magnitude**

- **Sum of "*Digit* x *Weight*"**

★ **Formal Notation**



256 16 1 1/16 1/256

1 E 5 . 7 A

2 1 0 -1 -2

$$1 * 16^2 + 14 * 16^1 + 5 * 16^0 + 7 * 16^{-1} + 10 * 16^{-2}$$

$$=(485.4765625)_{10}$$

$$(1E5.7A)_{16}$$

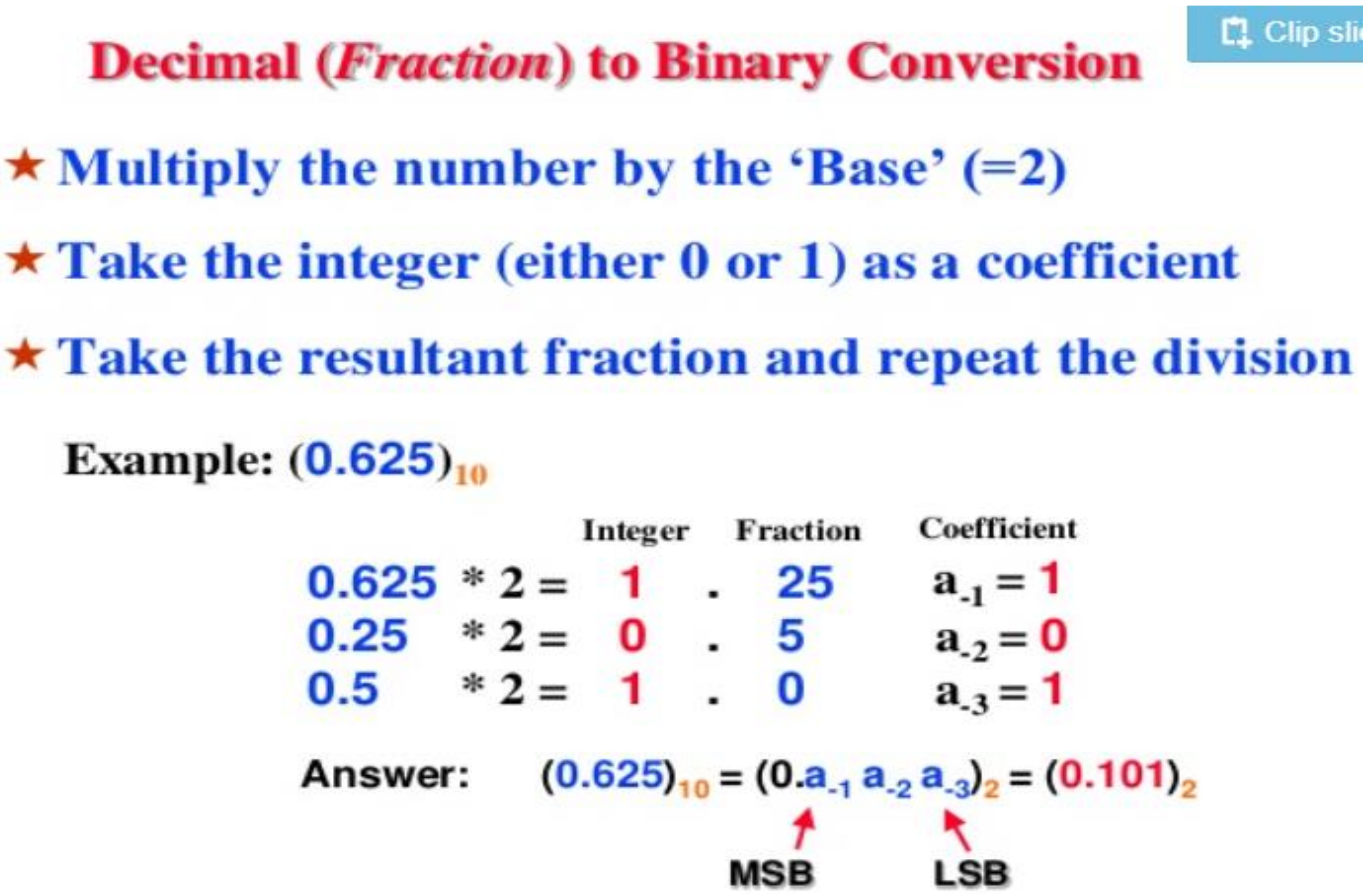
Decimal (*Integer*) to Binary Conversion

- ★ Divide the number by the 'Base' (=2)
- ★ Take the remainder (either 0 or 1) as a coefficient
- ★ Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
$13 / 2 =$	6	1	$a_0 = 1$
$6 / 2 =$	3	0	$a_1 = 0$
$3 / 2 =$	1	1	$a_2 = 1$
$1 / 2 =$	0	1	$a_3 = 1$

Answer: $(13)_{10} = (\underset{\text{MSB}}{\underset{\nearrow}{a_3}}\, a_2\, a_1\, \underset{\text{LSB}}{\underset{\nwarrow}{a_0}})_2 = (1101)_2$



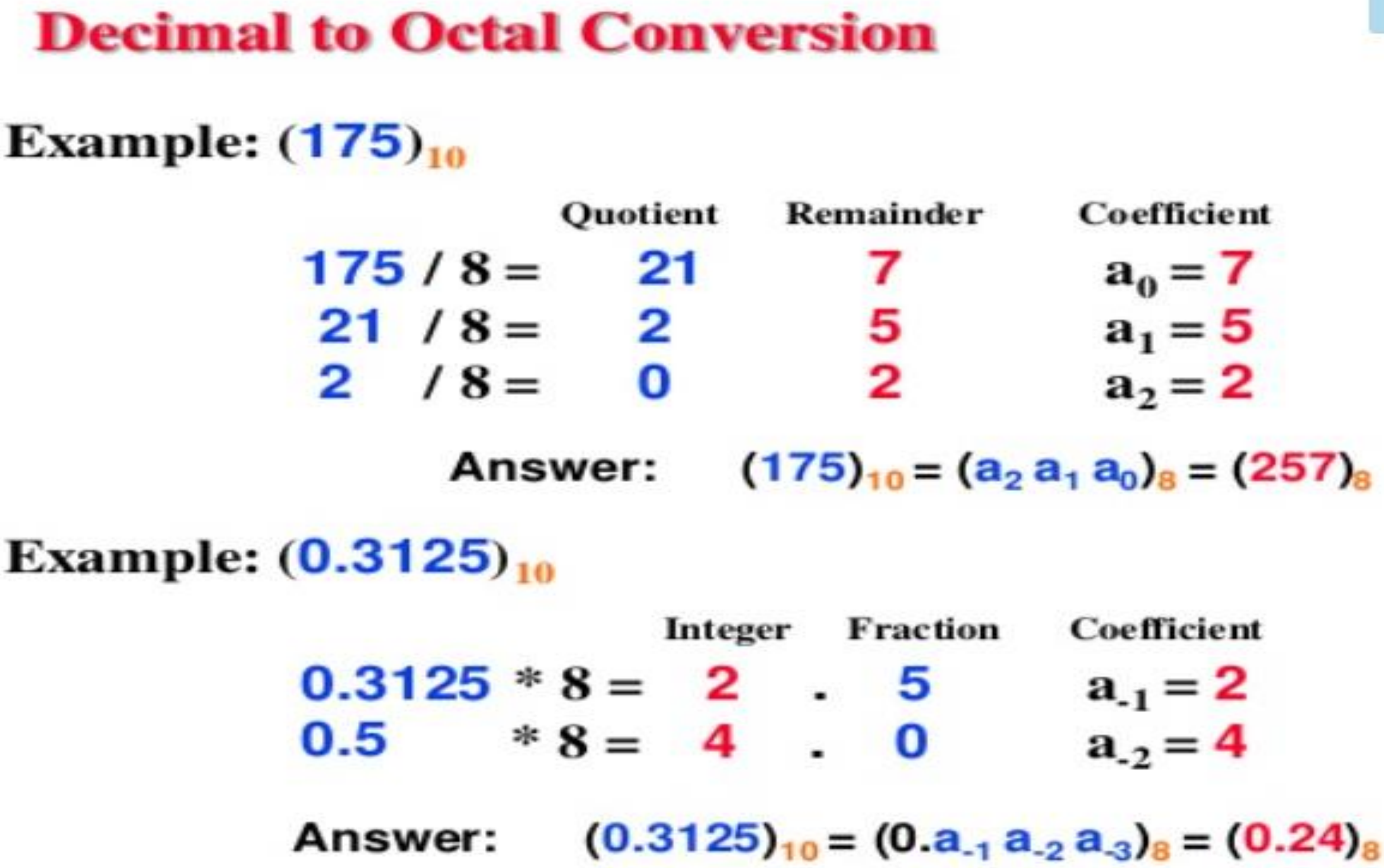
Decimal (*Fraction*) to Binary Conversion

- ★ Multiply the number by the 'Base' (=2)
- ★ Take the integer (either 0 or 1) as a coefficient
- ★ Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.\underset{\substack{\uparrow \\ \text{MSB}}}{a_{-1}}\ a_{-2}\ \underset{\substack{\nwarrow \\ \text{LSB}}}{a_{-3}})_2 = (0.101)_2$



Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
$175\ /\ 8 =$	21	7	$a_0 = 7$
$21\ /\ 8 =$	2	5	$a_1 = 5$
$2\ /\ 8 =$	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\ a_1\ a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

	Integer		Fraction	Coefficient
$0.3125 * 8 =$	2	.	5	$a_{-1} = 2$
$0.5 * 8 =$	4	.	0	$a_{-2} = 4$

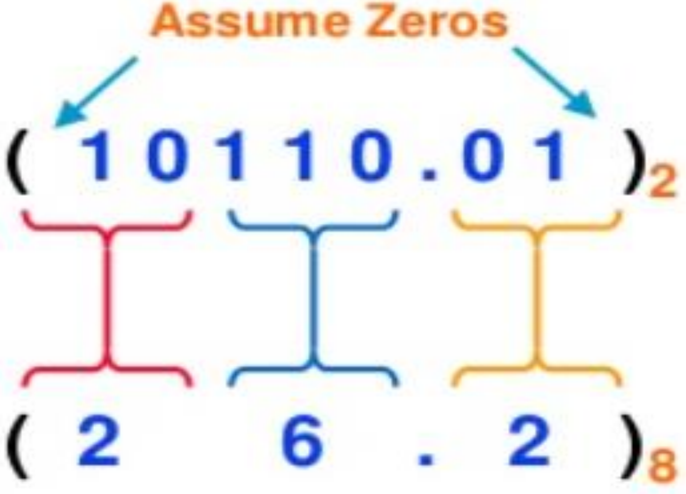
Answer: $(0.3125)_{10} = (0.a_{-1}\ a_{-2}\ a_{-3})_8 = (0.24)_8$

Binary – Octal Conversion

★ $8 = 2^3$

★ Each group of 3 bits represents an octal digit

Example:



Assume Zeros

$(\ 1\ 0\ 1\ 1\ 0\ .\ 0\ 1\ )_2$

$(\ 2\ \ 6\ .\ 2\ )_8$

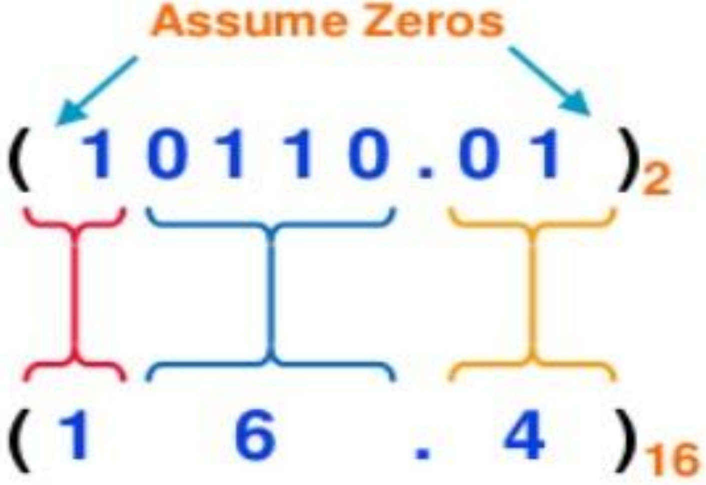
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

Binary – Hexadecimal Conversion

- ★ $16 = 2^4$
- ★ Each group of 4 bits represents a hexadecimal digit

Example:



Assume Zeros

$(10110.01)_2$

$(1\ 6\ .\ 4)_{16}$

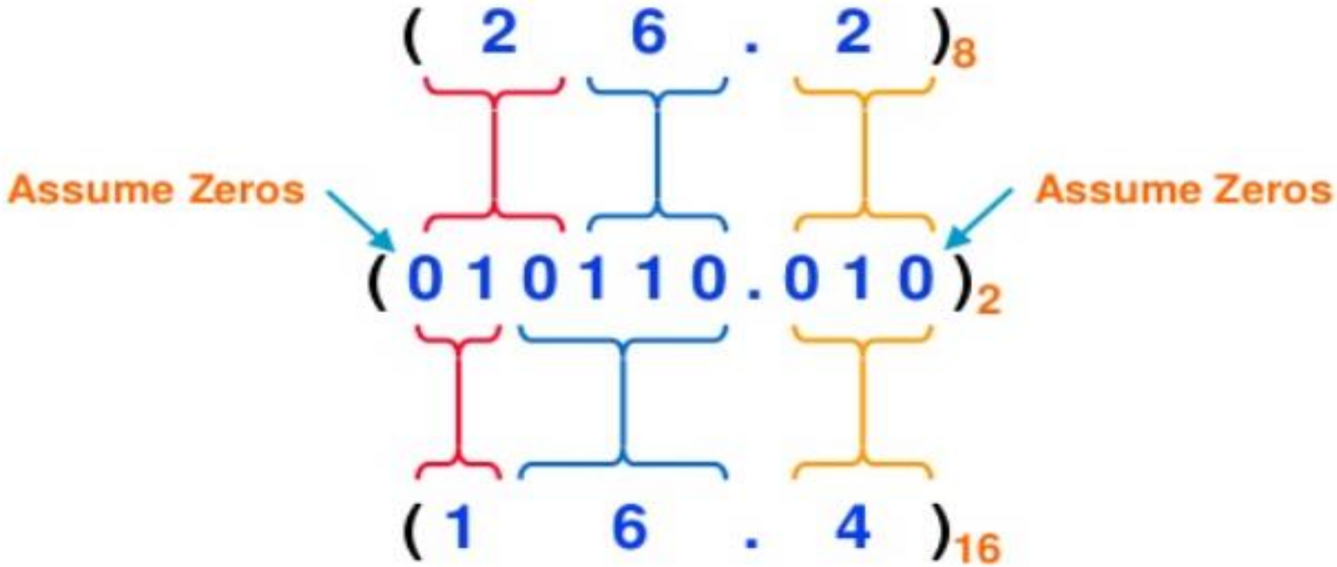
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Octal − Hexadecimal Conversion

★ **Convert to Binary as an intermediate step**

Example:



$(2\ 6\ .\ 2)_8$

$(0\,1\,0\,1\,1\,0\,.\,0\,1\,0)_2$

$(1\ 6\ .\ 4)_{16}$

Works both ways (*Octal* to *Hex* & *Hex* to *Octal*)

Decimal, Binary, Octal and Hexadecimal

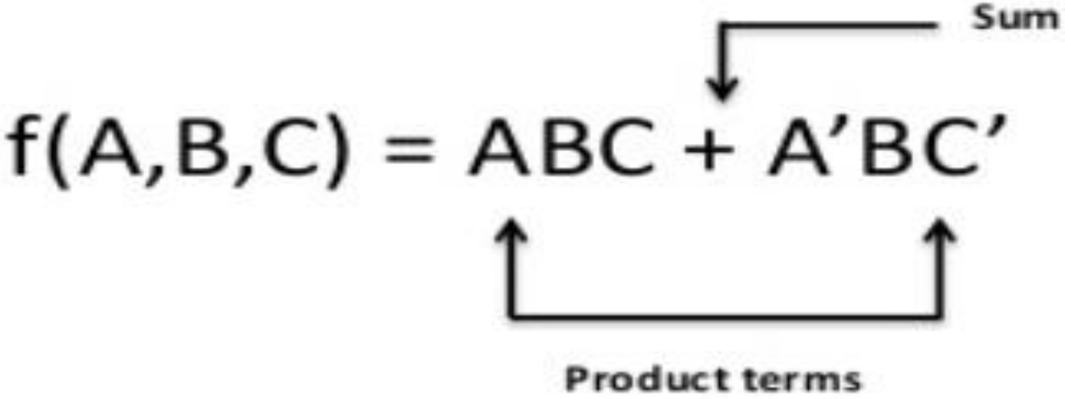
Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Content:

- Sum of product (SOP)
- Product of sum (POS)
- Standard SOP and POS Forms
- Convert SOP to standard SOP
- Convert POS to Standard POS
- Minterms and Maxterms

Sum of Product

- The sum-of-products (**SOP**) form is a method (or form) of simplifying the Boolean expressions of logic gates.
- Sum and product derived from the symbolic representations of the OR and AND functions.
- OR (+) , AND (.) , addition and multiplication.



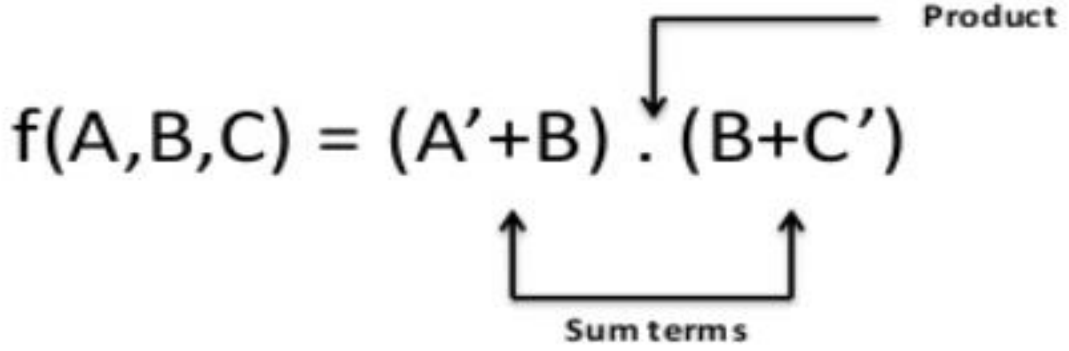
Sum

$$f(A,B,C) = ABC + A'BC'$$

Product terms

Product of Sum

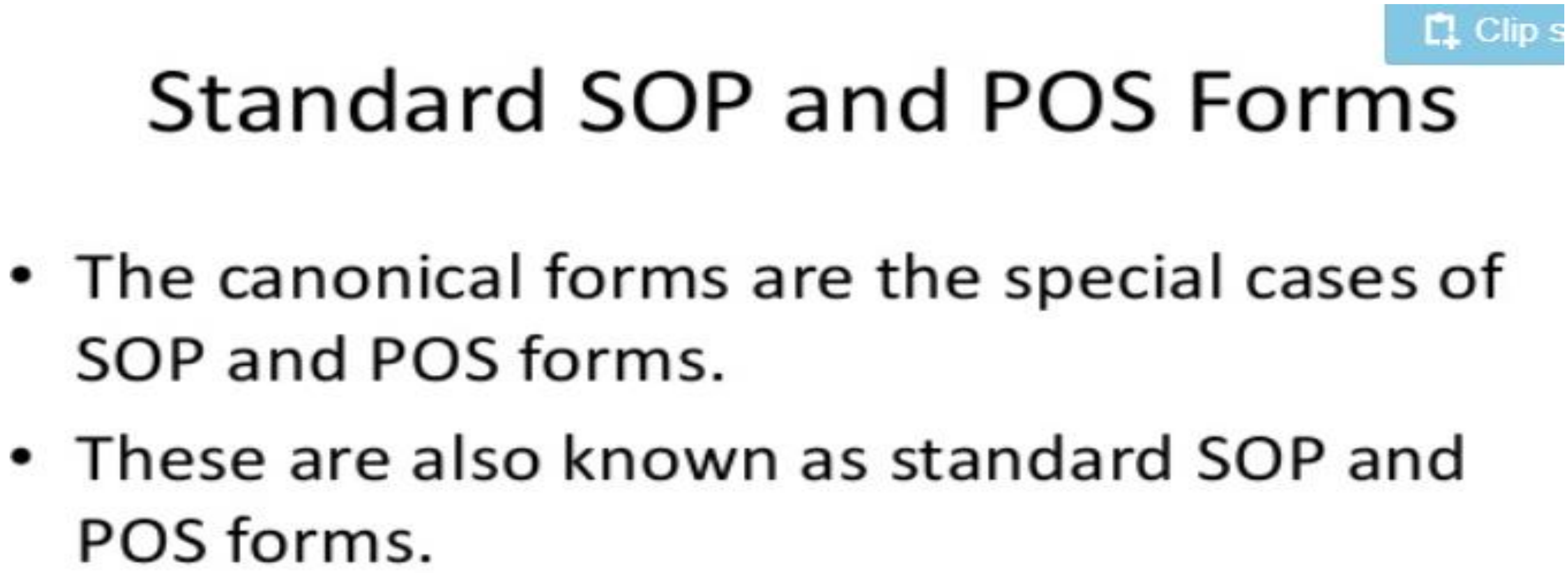
- When two or more sum terms are multiplied by a Boolean **AND** operation.
- Sum terms are defined by using OR operation and the product term is defined by using AND operation.



$$f(A,B,C) = (A'+B)\ .\ (B+C')$$

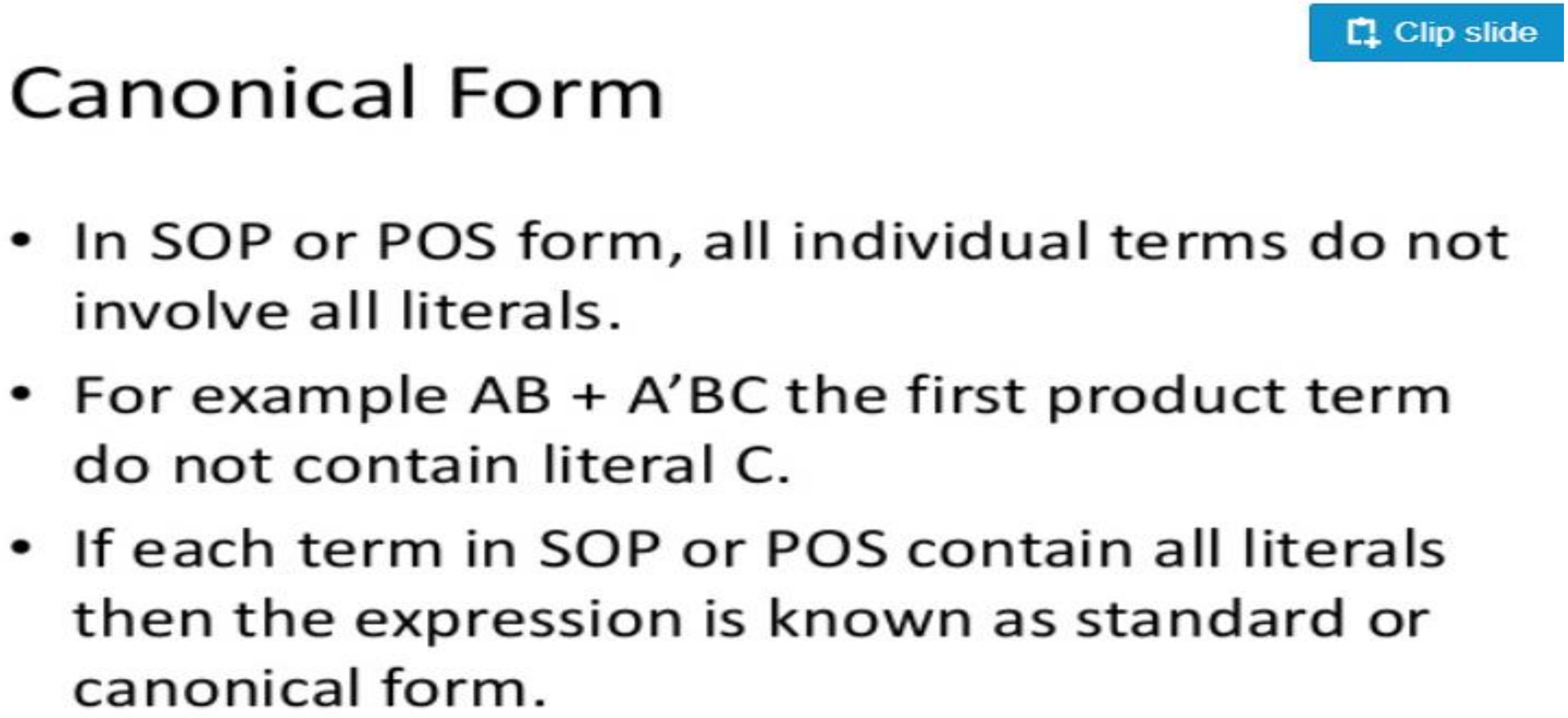
Product

Sum terms



Standard SOP and POS Forms

- The canonical forms are the special cases of SOP and POS forms.
- These are also known as standard SOP and POS forms.



Canonical Form

- In SOP or POS form, all individual terms do not involve all literals.
- For example AB + A’BC the first product term do not contain literal C.
- If each term in SOP or POS contain all literals then the expression is known as standard or canonical form.

Canonical Form

- Each individual term in the POS form is called **Maxterm**.
- Each individual term in the SOP form is called **Minterm**.
- In Minterm, we look for the functions where the output results is “1”.
- while in Maxterm we look for function where the output results is “0”.
- We perform **Sum of minterm** also known as Sum of products (SOP) .
- We perform **Product of Maxterm** also known as Product of sum (POS).

Convert SOP to standard SOP form

Step 1: Find the missing literal in each product term if any.

Step 2: And each product term having missing literals with terms form by ORing the literal and its complement.

Step 3: Expends the term by applying, distributive law and reorder the literals.

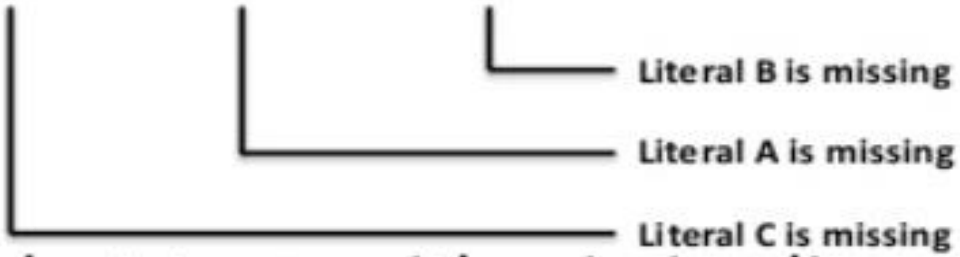
Step 4: Reduce the repeated product terms. Because A + A = A (Theorem 1a).

Example:

$$f\,(A,B,C) = AB + BC + AC$$

Step 1: Find the missing literals in each product term.

$$f\,(A,B,C) = AB + BC + AC$$



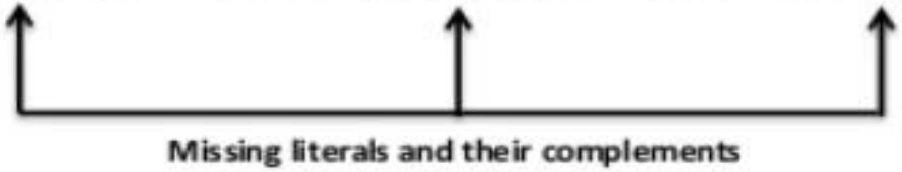
Literal B is missing

Literal A is missing

Literal C is missing

Step 2: AND the product term with missing literal + its complement.

$$f\,(A,B,C) = AB\,.\,(C+C') + BC\,.\,(A+A') + AC\,.\,(B+B')$$



Missing literals and their complements

Step 3: Expends the term and reorder the literals.

$f\ (A,B,C) = AB\ .\ (C+C') + BC\ .\ (A+A') + AC\ .\ (B+B')$

Expand & Reorder:

$$ABC + ABC' + ABC + A'BC + ABC + AB'C$$

Step 4: Omit repeated product terms.

$$f(A,B,C)=ABC + ABC' + \overset{\downarrow}{ABC} + A'BC + \overset{\downarrow}{ABC} + AB'C$$

$$f(A,B,C)= ABC + ABC' + A'BC + AB'C$$

Convert POS to standard POS form

Step 1: Find the missing literal in each sum term if any.

Step 2: OR each sum term having missing literals with terms form by ANDing the literal and its complement.

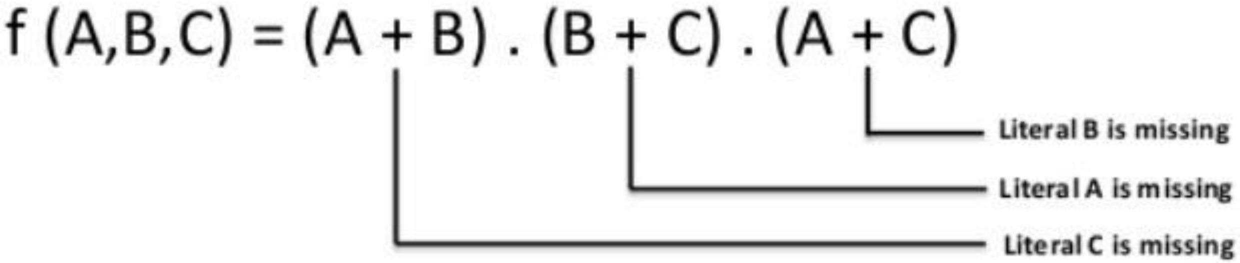
Step 3: Expends the term by applying, distributive law and reorder the literals.

Step 4: Reduce the repeated product terms. Because $A + A = A$ (Theorem 1a).

Example:

$$f\,(A,B,C) = (A + B)\,.\,(B + C)\,.\,(A + C)$$

Step 1: Find the missing literals in each sum term.



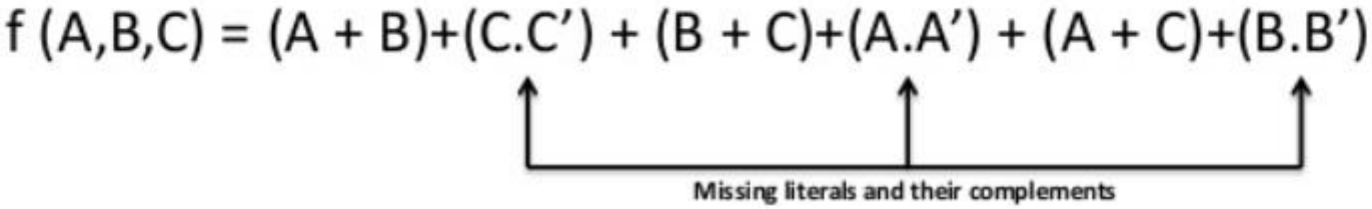
$$f\,(A,B,C) = (A + B)\,.\,(B + C)\,.\,(A + C)$$

Literal B is missing

Literal A is missing

Literal C is missing

Step 2: OR the sum term with missing literal . its complement.



$$f\,(A,B,C) = (A + B)+(C.C') + (B + C)+(A.A') + (A + C)+(B.B')$$

Missing literals and their complements

Step 3: Expends the term and reorder the literals.

$f\ (A,B,C) = (A + B)+(C.C') + (B + C)+(A.A') + (A +C)+(B.B')$

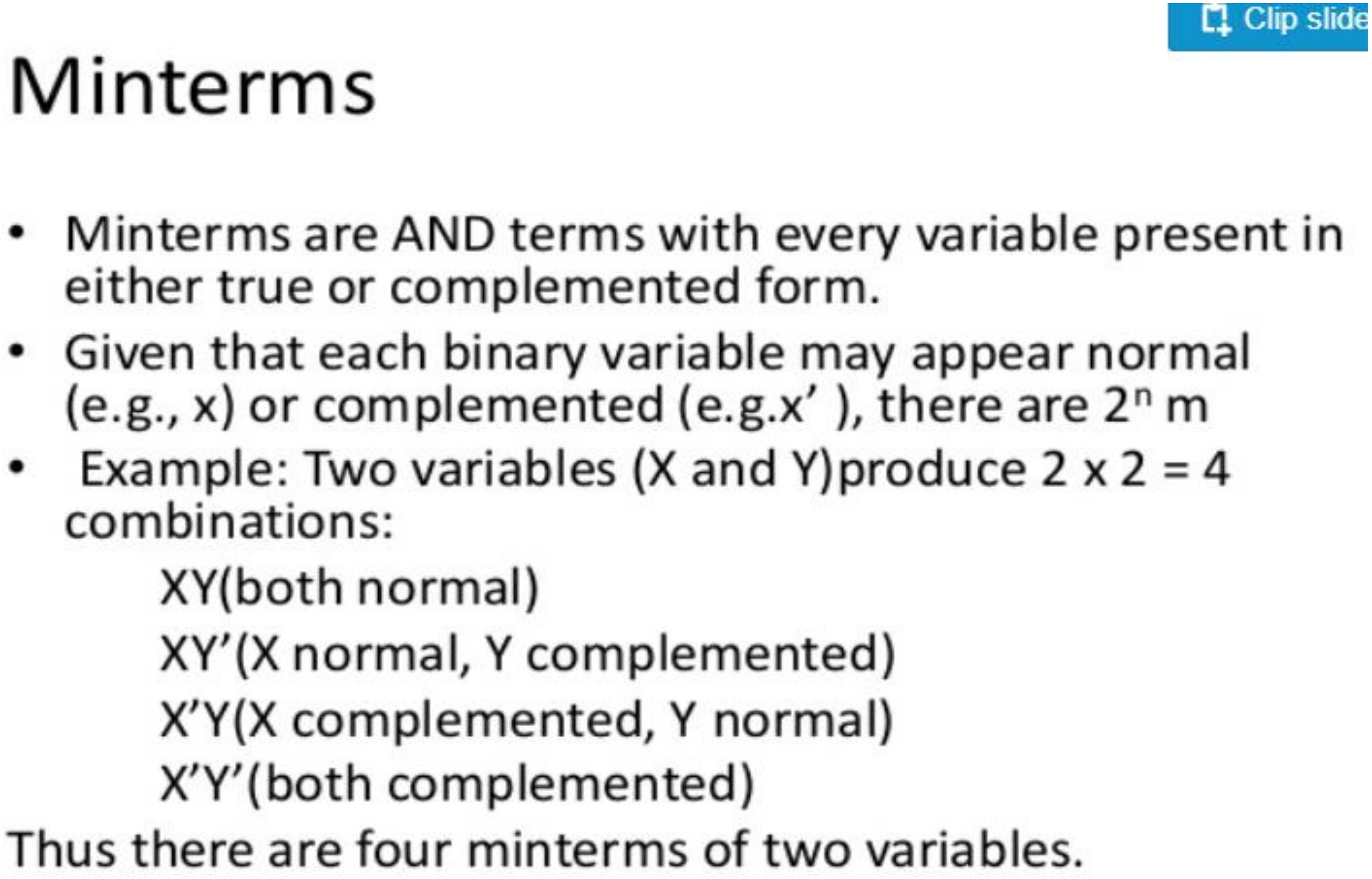
Expand & Reorder:

$f(A,B,C)=(A+B+C).(A+B+C').(A+B+C).(A'+B+C).(A+B+C).(A+B'+C)$

Step 4: Omit repeated sum terms.

$$f(A,B,C)=(A+B+C).(A+B+C').(A+B+C).(A'+B+C).(A+B+C).(A+B'+C)$$

$$f(A,B,C)=(A+B+C).(A+B+C').(A'+B+C).(A+B'+C)$$



Minterms

- Minterms are AND terms with every variable present in either true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g.x'), there are 2^n m
- Example: Two variables (X and Y)produce 2 x 2 = 4 combinations:

  XY(both normal)

  XY'(X normal, Y complemented)

  X'Y(X complemented, Y normal)

  X'Y'(both complemented)

Thus there are four minterms of two variables.

Maxterms

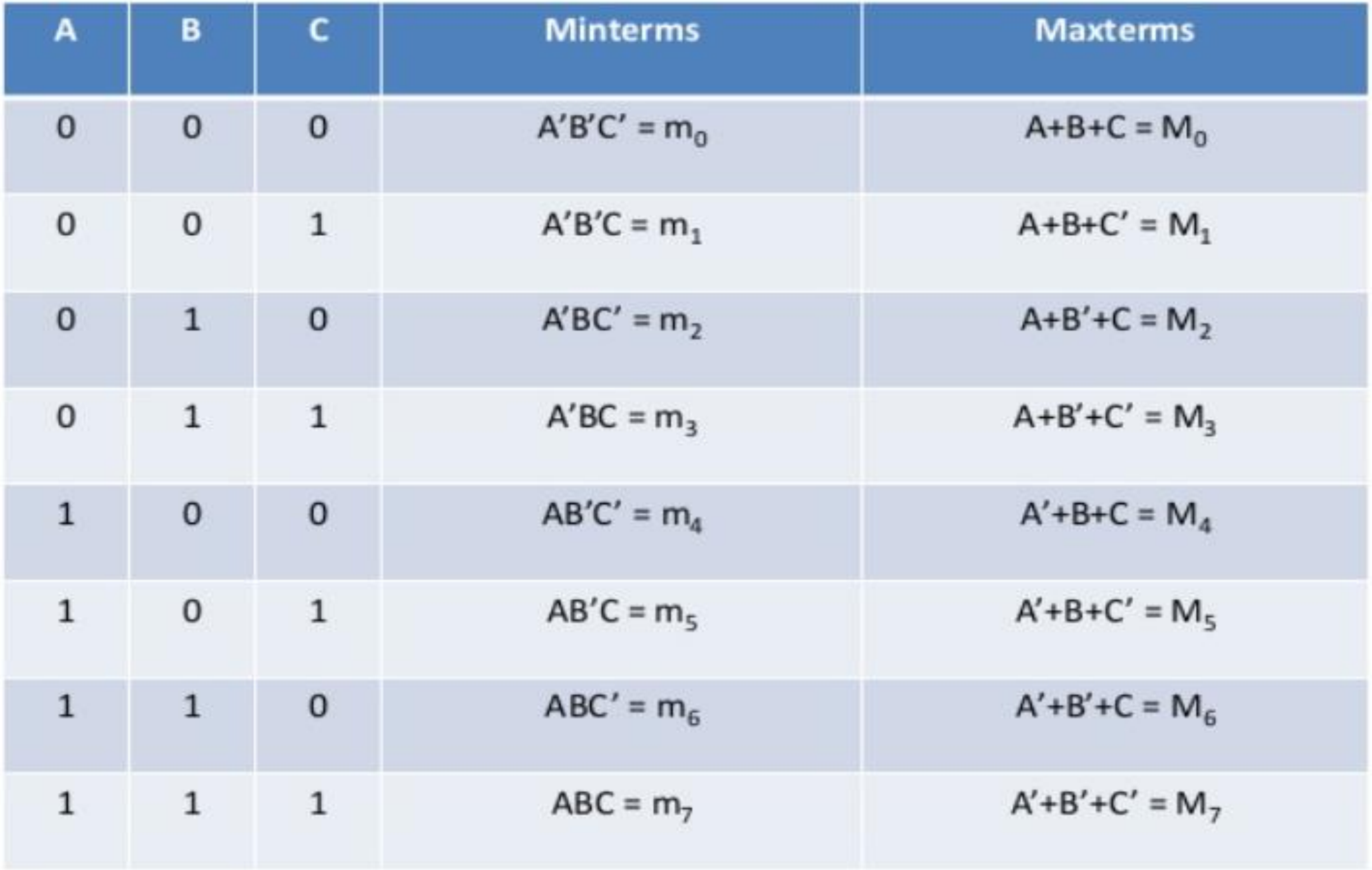
- Maxterms are OR terms with every variable in true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g. x’), there are 2^n maxterms for n variables.
- Example: Two variables (X and Y) produce 2x2=4 combinations:

    X+Y(both normal)

    X+Y’(x normal, y complemented)

    X’+Y(x complemented, y normal)

    X’+Y’(both complemented)



A	B	C	Minterms	Maxterms
0	0	0	A’B’C’ = m_0	A+B+C = M_0
0	0	1	A’B’C = m_1	A+B+C’ = M_1
0	1	0	A’BC’ = m_2	A+B’+C = M_2
0	1	1	A’BC = m_3	A+B’+C’ = M_3
1	0	0	AB’C’ = m_4	A’+B+C = M_4
1	0	1	AB’C = m_5	A’+B+C’ = M_5
1	1	0	ABC’ = m_6	A’+B’+C = M_6
1	1	1	ABC = m_7	A’+B’+C’ = M_7

Minterms:

1. $f(A,B,C) = A'B'C' + A'BC' + A'BC + ABC$

   $= m_0 + m_2 + m_3 + m_7$

   $= \Sigma m(0,2,3,7)$

2. $f(A,B,C)= A'B'C + A'BC + AB'C + ABC$

   $= m_1 + m_3 + m_5 + m_7$

   $= \Sigma m(1,3,5,7)$

3. $f(A,B,C)= A'B'C' + A'BC' + A'BC + ABC'$

   $= m_0 + m_2 + m_3 + m_6$

   $= \Sigma m(0,2,3,6)$

Maxterms:

1. $f(A,B,C) = (A+B+C).(A+B'+C).(A+B'+C')+(A'+B'+C')$

   $= M_0 + M_2 + M_3 + M_7$

   $= \Pi M(0,2,3,7)$

2. $f(A,B,C)= (A+B+C').(A+B'+C').(A+B'+C').(A'+B'+C')$

   $= M_1 + M_3 + M_5 + M_7$

   $= \Pi M\ (1,3,5,7)$

3. $f(A,B,C)= (A+B+C).(A+B'+C).(A+B'+C').(A'+B'+C)$

   $= M_0 + M_2 + M_3 + M_6$

   $= \Pi M\ (0,2,3,6)$

Adders

At the digital logic level, addition is performed in binary

Addition operations are carried out by special circuits called, appropriately, **adders**

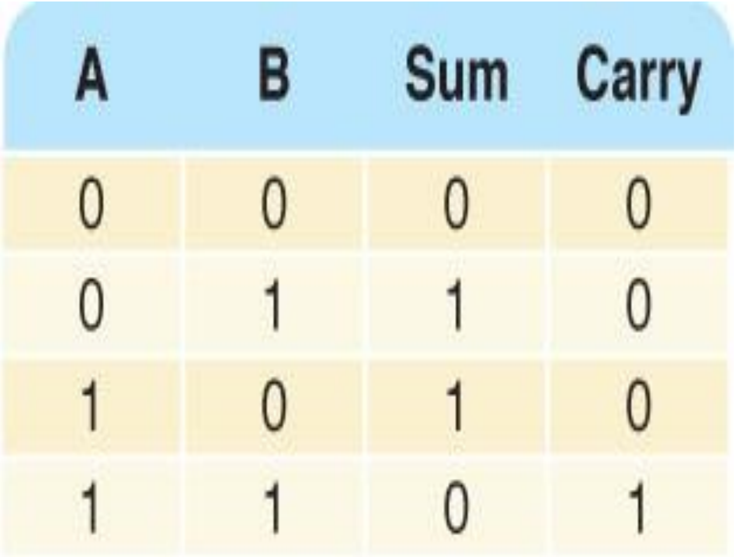
Adders

The result of adding two binary digits could produce a *carry value*

Recall that 1 + 1 = 10 in base two

Half adder

A circuit that computes the sum of two bits and produces the correct carry bit



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth table

m1= A’B

m2= AB’

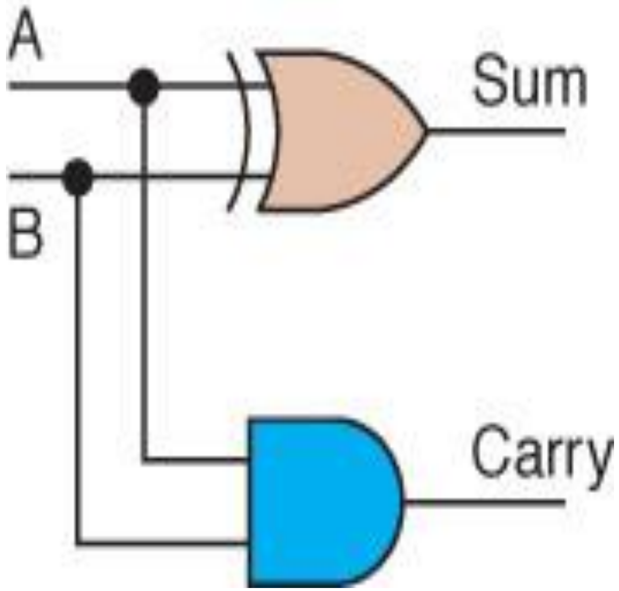
m3= AB

Sum= m1 + m2

= A’B+AB’

Carry= m3 = AB

Adders



Circuit diagram representing a half adder

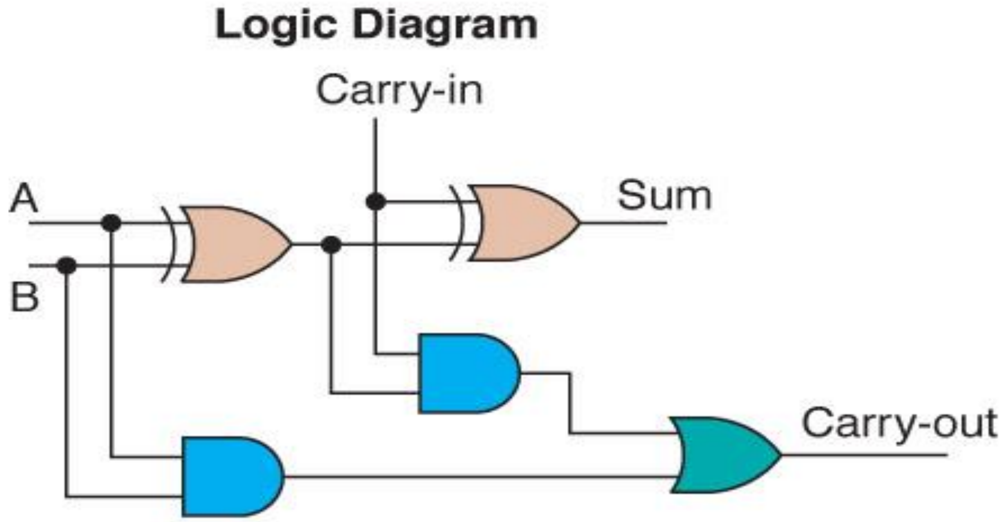
Boolean expressions

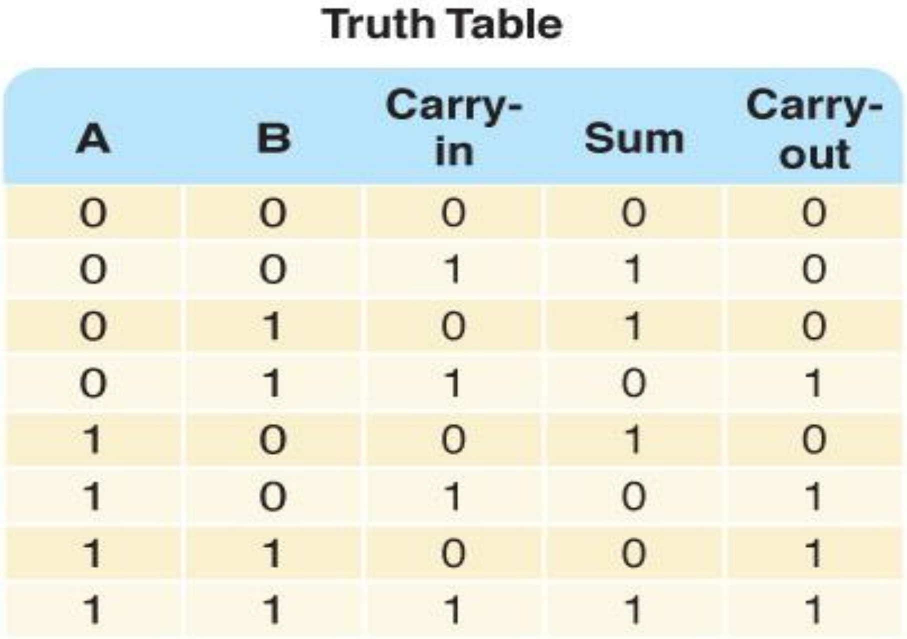
$\text{sum} = A \oplus B$
$\text{carry} = AB$

Adders

Full adder

A circuit that takes the carry-in value into account




Truth Table

A	B	Carry-in	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	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

$$SUM = A \oplus B \oplus C$$

CARRY= AB+BC+AC

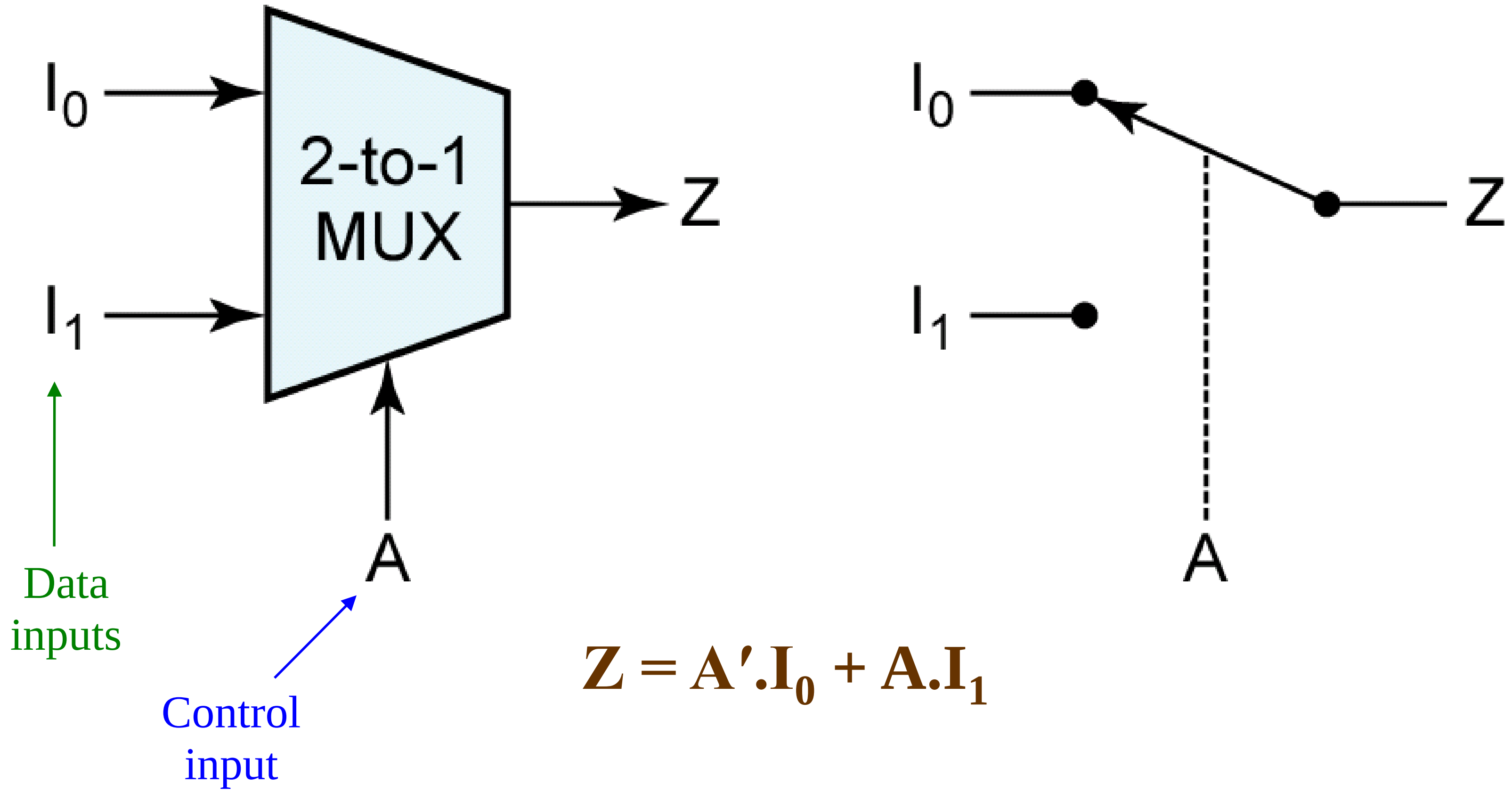
Figure 10 A full adder

Multiplexers

Multiplexers

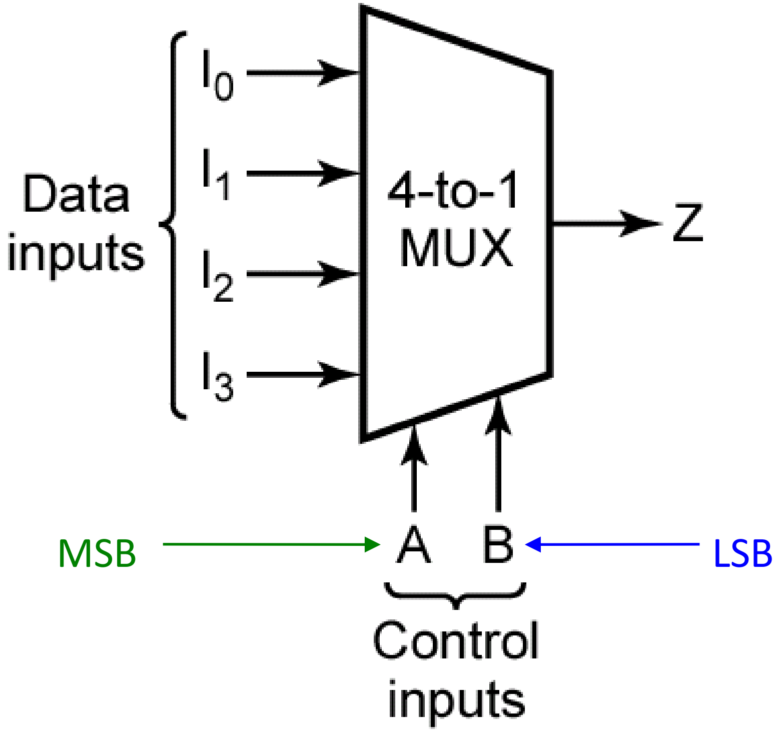
- A multiplexer has
  - N control inputs
  - 2^N data inputs
  - 1 output
- A multiplexer routes (or connects) the selected data input to the output.
  - The value of the control inputs determines the data input that is selected.

Multiplexers



$$Z = A'.I_0 + A.I_1$$

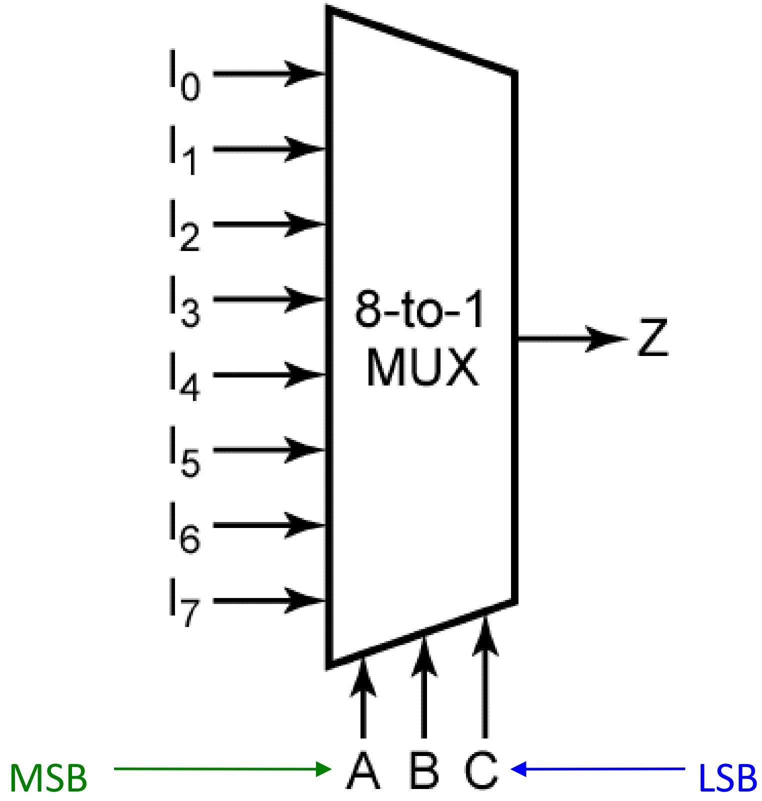
Multiplexers



A	B	**F**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$\mathbf{Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3}$$

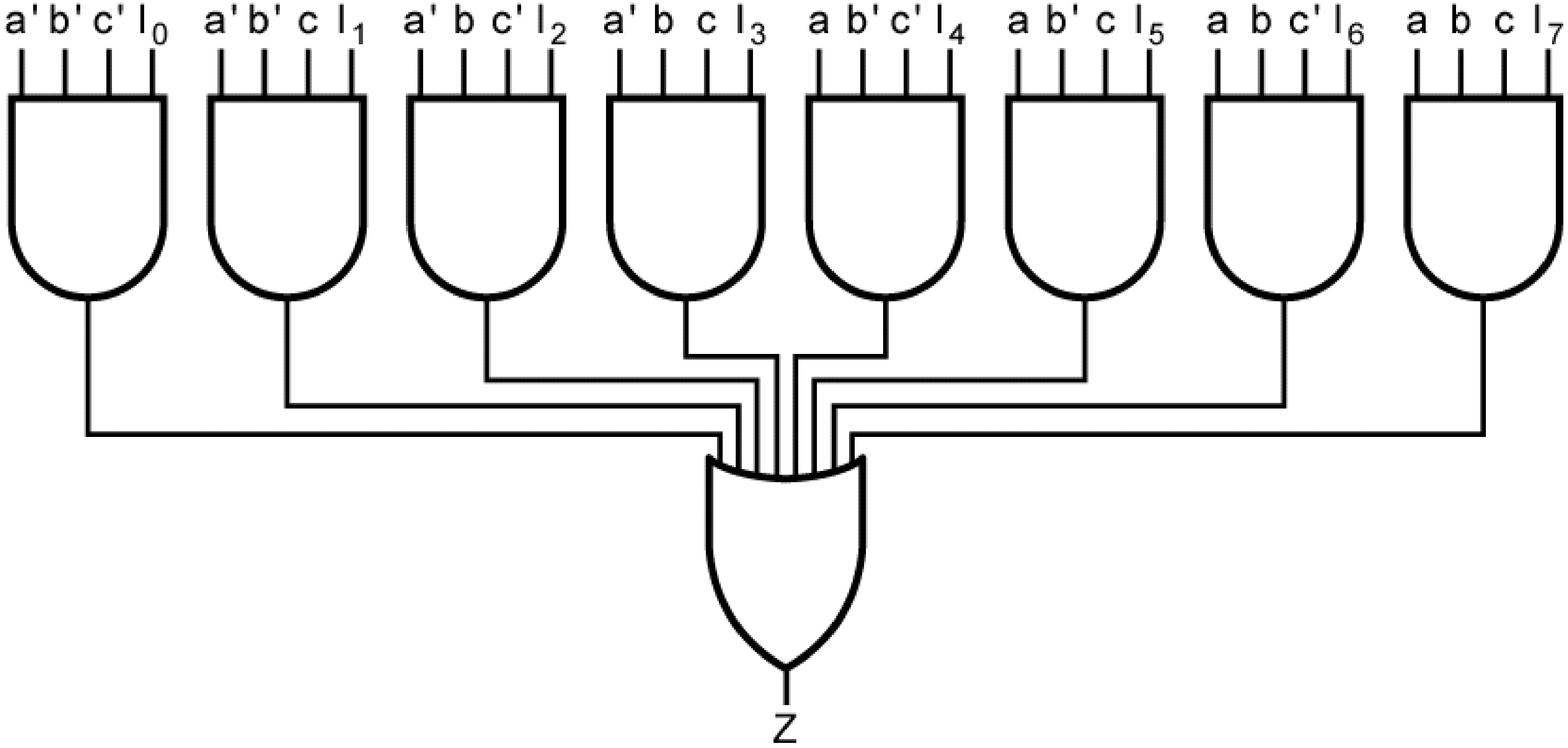
Multiplexers



A	B	C	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$$Z = A'.B'.C'.I_0 + A'.B'.C.I_1 + A'.B.C'.I_2 + A'.B.C.I_3 +$$
$$A.B'.C'.I_0 + A.B'.C.I_1 + A'.B.C'.I_2 + A.B.C.I_3$$

Multiplexers



Multiplexers

Exercise:

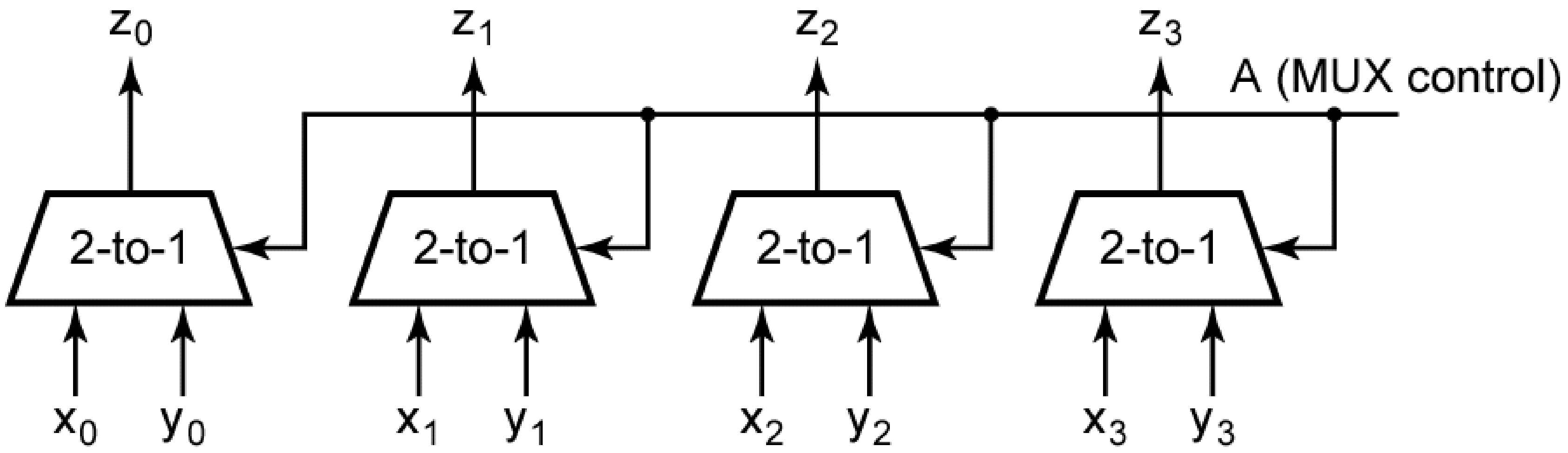
Design an 8-to-1 multiplexer using
4-to-1 and 2-to-1 multiplexers only.

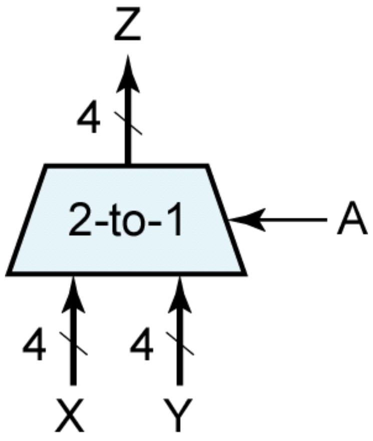
Multiplexers

Exercise:

Design a 16-to-1 multiplexer using
4-to-1 multiplexers only.

Multiplexer (Bus)



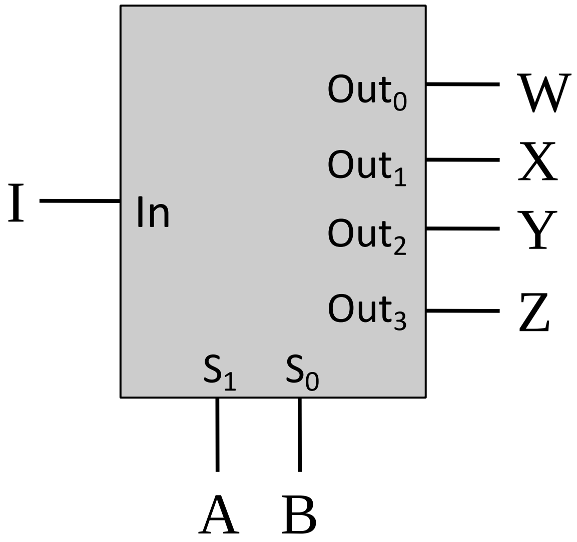


Demultiplexers

Demultiplexers

- A demultiplexer has
  - N control inputs
  - 1 data input
  - 2^N outputs
- A demultiplexer routes (or connects) the data input to the selected output.
  - The value of the control inputs determines the output that is selected.
- A demultiplexer performs the opposite function of a multiplexer.

Demultiplexers



W = A'.B'.I

X = A.B'.I

Y = A'.B.I

Z = A.B.I

A	B	**W**	**X**	**Y**	**Z**
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

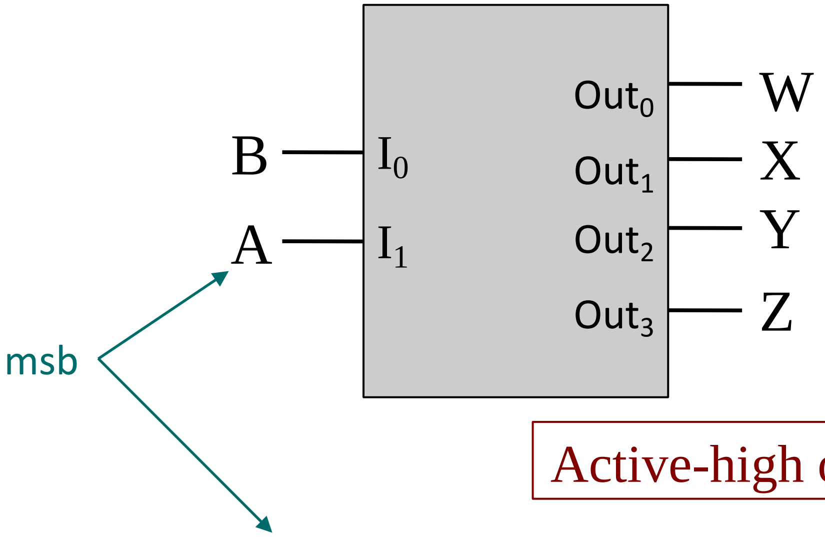
Decoders

Decoders

- A decoder has
  - N inputs
  - 2^N outputs
- A decoder selects one of 2^N outputs by decoding the binary value on the N inputs.
- The decoder generates all of the minterms of the N input variables.
  - Exactly one output will be active for each combination of the inputs.

What does “active” mean?

Decoders



W = A'.B'

X = A.B'

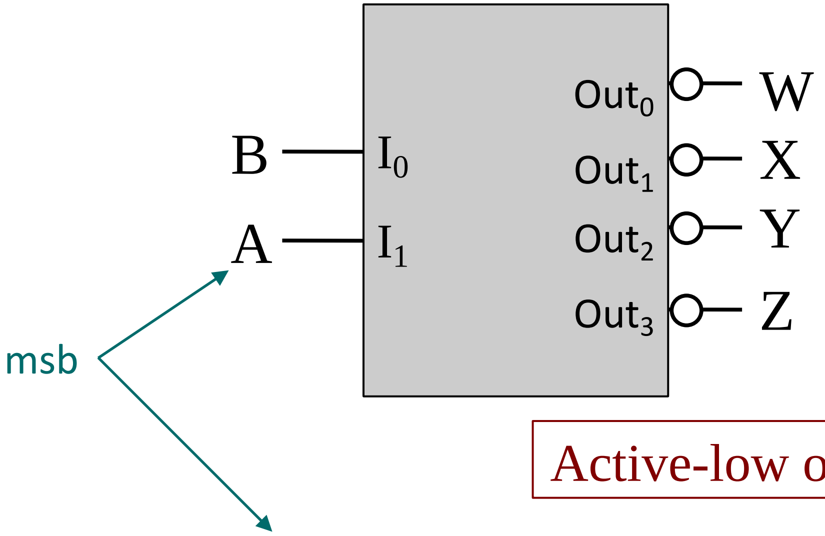
Y = A'.B

Z = A.B

Active-high outputs

A	B	**W**	**X**	**Y**	**Z**
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoders



W = (A'.B')'

X = (A.B')'

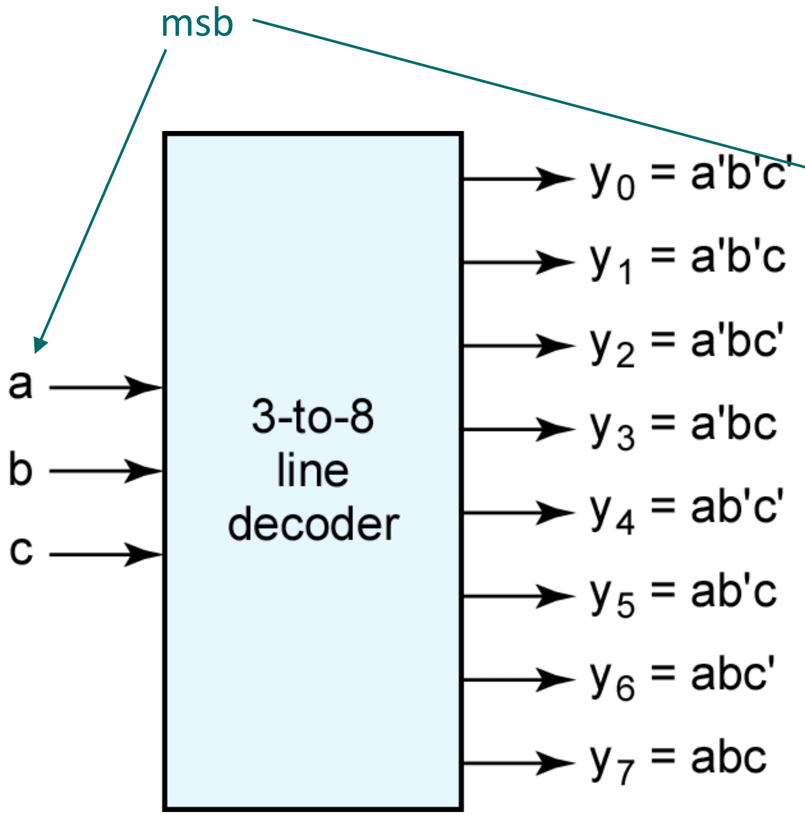
Y = (A'.B)'

Z = (A.B)'

Active-low outputs

A	B	W	X	Y	Z
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Decoders



msb

a

b

c

3-to-8 line decoder

$y_0 = a'b'c'$

$y_1 = a'b'c$

$y_2 = a'bc'$

$y_3 = a'bc$

$y_4 = ab'c'$

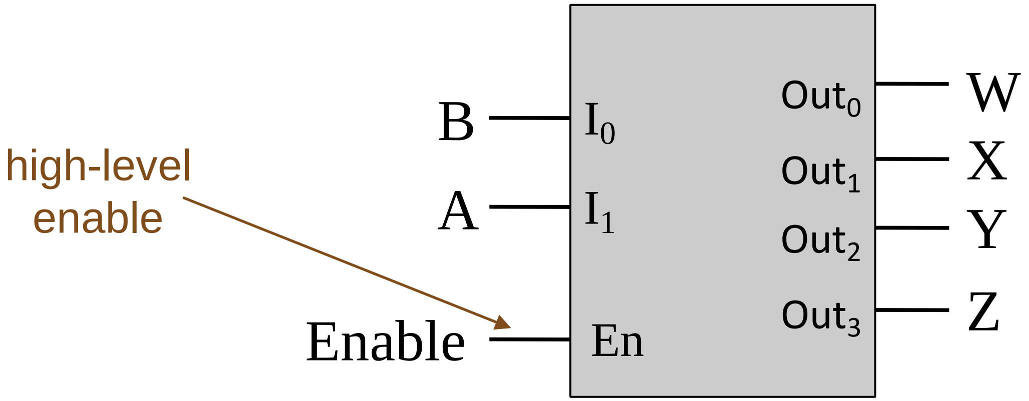
$y_5 = ab'c$

$y_6 = abc'$

$y_7 = abc$

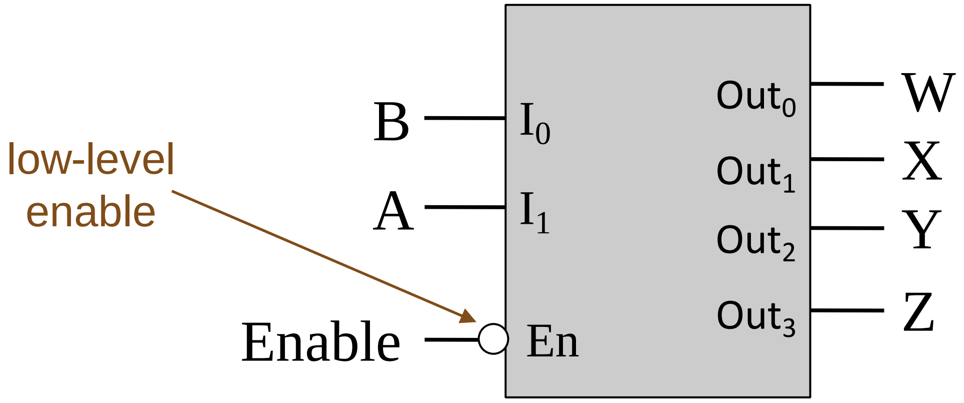
a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder with Enable



	En	A	B	**W**	**X**	**Y**	**Z**
enabled	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	0	1
disabled	0	x	x	0	0	0	0

Decoder with Enable



En	A	B	**W**	**X**	**Y**	**Z**
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	x	x	0	0	0	0

enabled

disabled

Decoders

Exercise:

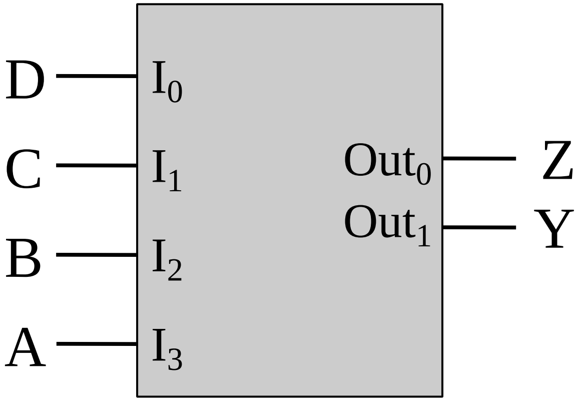
Design a 4-to-16 decoder using
2-to-4 decoders only.

Encoders

Encoders

- An encoder has
  - 2^N inputs
  - N outputs
- An encoder outputs the binary value of the selected (or active) input.
- An encoder performs the inverse operation of a decoder.
- Issues
  - What if more than one input is active?
  - What if no inputs are active?

Encoders



A	B	C	D	**Y**	**Z**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

FLIP FLOPS

DEFINITION: Flip flop is a basic memory element in a digital computer. It is used to store One bit of information with a 0 or a 1.

TYPES OF FLIP FLOPS:

✓R S flip flop (Reset & Set).

✓D flip flop (Delay / Data).

✓J K flip flop.

✓T flip flop (Toggle).

✓Master Slave J K flip flop.

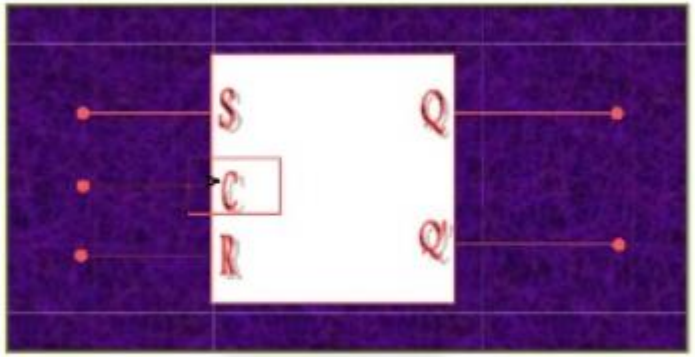
CHARACTERISTICS :

✓Flip-flop is a bi stable device ,It has only two states 0 and 1.

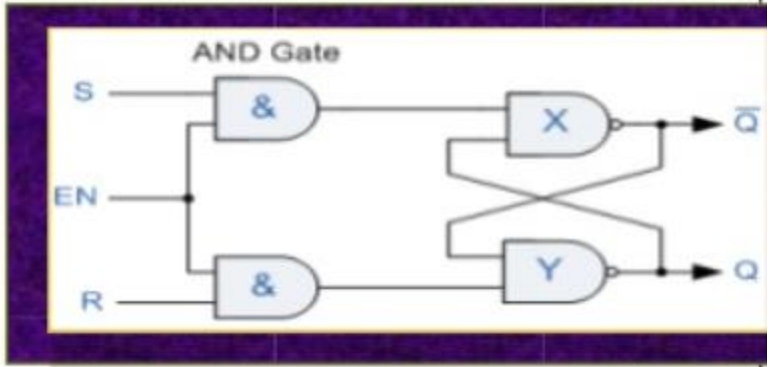
✓Flip-flop has two outputs, One output is complement of other.

RS - FLIP FLOP

Logic Symbol :



Logic Circuit :



Truth table :

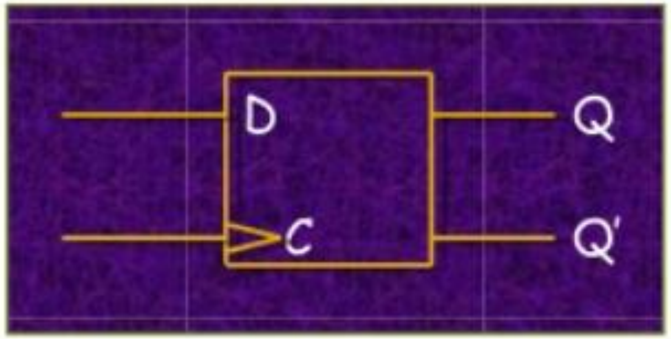
S	R	Q(t+1)	
0	0	Q(t)	No Change
0	1	0	Clear or Reset to 0
1	0	1	Set to 1
1	1	?	Indeterminate or Forbidden state

Working of RS flip flop

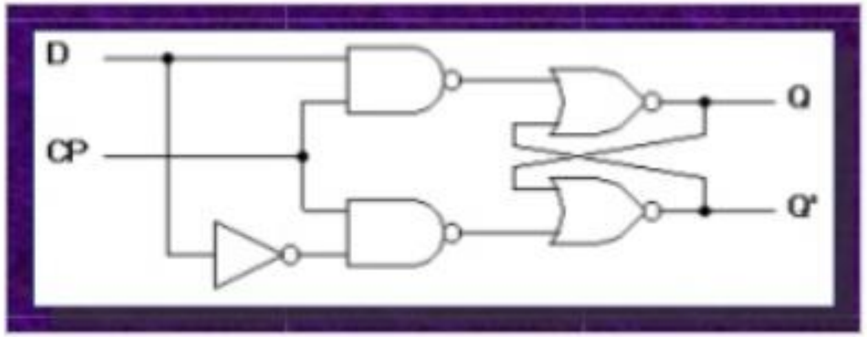
- If both S and R are 0 during transition, the output does not change.
- If S= 1 and R= 0, the out put Q is set to 1.
- If S= 0 and R=1, the output is cleared or reset to 0.
- If both S and R are 1, the output is unpredictable. This condition makes the RS flip flop difficult to manage and therefore is forbidden in practice.

D - FLIP FLOP

Logic diagram:



Logic diagram with NAND gates



TRUTH TABLE FOR D FLIPFLOP :

D	Q(t+1)		STATE	
0	0		Clear to 0	
1	1		Clear to 1	

Working of D – Flip flop:

The D input goes directly into the S input and the complement of the D input goes to the R input.

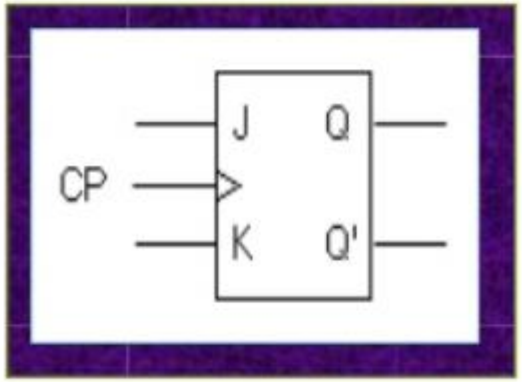
- If it is 1, the flip-flop is switched to the set state (unless it was already set).
- If it is 0, the flip-flop switches to the clear state.

Applications:

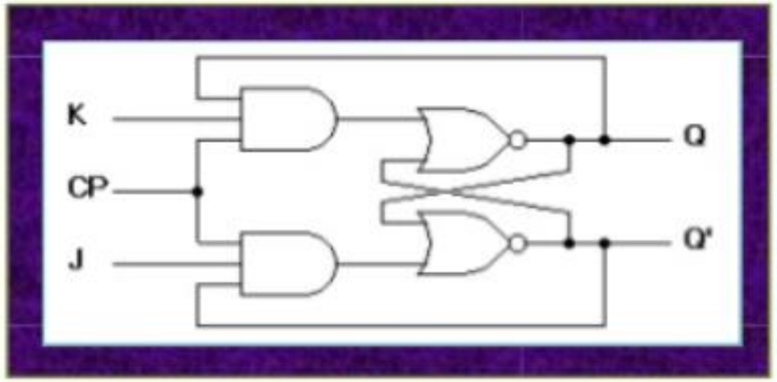
- Registers as storage devices.
- Used as a Buffer.
- In Digital system.

JK - FLIP FLOP

LOGIC DIAGRAM:



LOGIC DIAGRAM USING NAND GATES:



TRUTH TABLE:

J	K	Q(t+1)	STATE
0	0	Q(t)	No change
0	1	0	reset
1	0	1	set
1	1	Q'(t)	complement

Working of JK – Flip Flop :

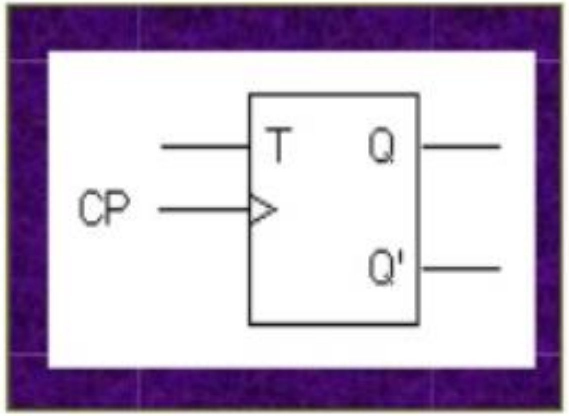
- When j and k both are 0, the data inputs have no effect on the outputs.
- When j=0 and k=1, the flip flop is reset or cleared to 0.
- When j=1 and k=0. the flip flop is set to 1.
- When j and k are 1, if the state of flip flop was 0,applying a clock with 1and flip flop state was 1, it changes to 0.this on off state is TOGGLING.
- Racing condition: Toggling between 0 to 1 and 1 to 0 alternatively for one clock cycle.

APPLICATIONS:

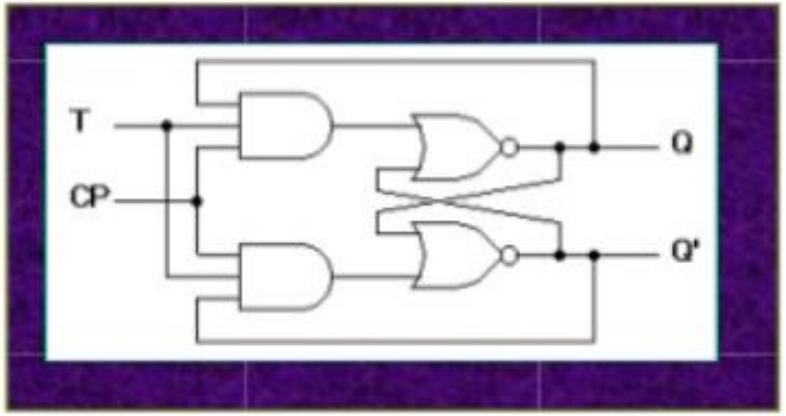
- Counters
- Frequency Dividers

T - FLIP FLOP

LOGIC DIAGRAM:



LOGIC DIAGRAM USING NAND GATES:



TRUTH TABLE:

T	Q(t+1)	STATE
0	Q(t)	No Change
1	Q’(t)	Complement

T - FLIP FLOP

- The T - flip flop is also known as the TOGGLE - flip flop. The toggle mode of JK flip flop is used as T - Flip flop.

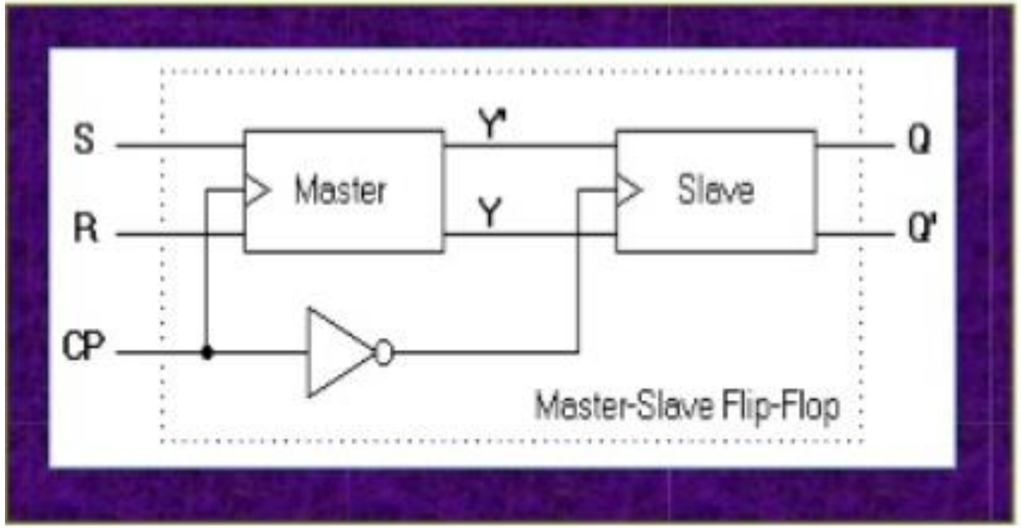
- This Flip flop can be obtained from a JK flip flop when inputs J and K are connected to provide a single input designated by T.

- The T flip-flop is a single input version of the JK flip - flop. The T flip-flop is obtained from the JK type if both inputs are tied together. The output of the T flip-flop "toggles" with each clock pulse..

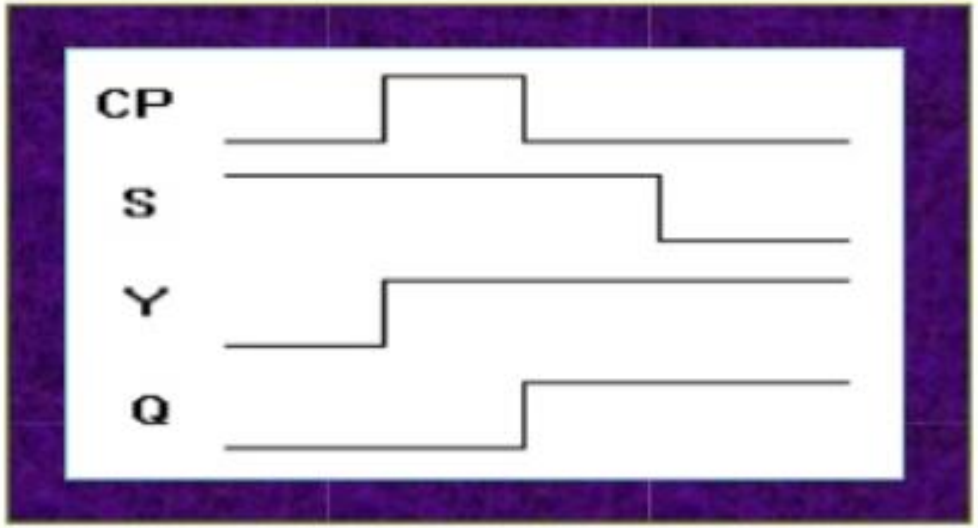
- When t=0, (j=0, k=0) the clock transition does not change.

- When t=1, (j=1, k=1) the clock transition complements the state.

LOGIC DIAGRAM:



TIMING RELATIONSHIP:



MASTER SLAVE JK - FLIP FLOP

- ✓ It is a Combination of 2 flip flops. Where, the first is the Master that responds to the positive edge of the clock and the second is the slave which responds to the negative level.
- ✓ The output changes only during the negative edge transition of the clock.
- ✓ Output triggers only for negative edge triggering.
- ✓ This helps in avoiding the racing condition in JK flip flop where it can be triggered only once within a clock cycle.
- ✓ Master/Slave triggering has become obsolete. Newer flip flops use negative edge triggering.

APPLICATIONS OF FLIP FLOPS:

Flip flops have a wide variety of applications. They are:

✓ REGISTERS

✓ FREQUENCY DIVIDERS

✓ DIGITAL COUNTERS

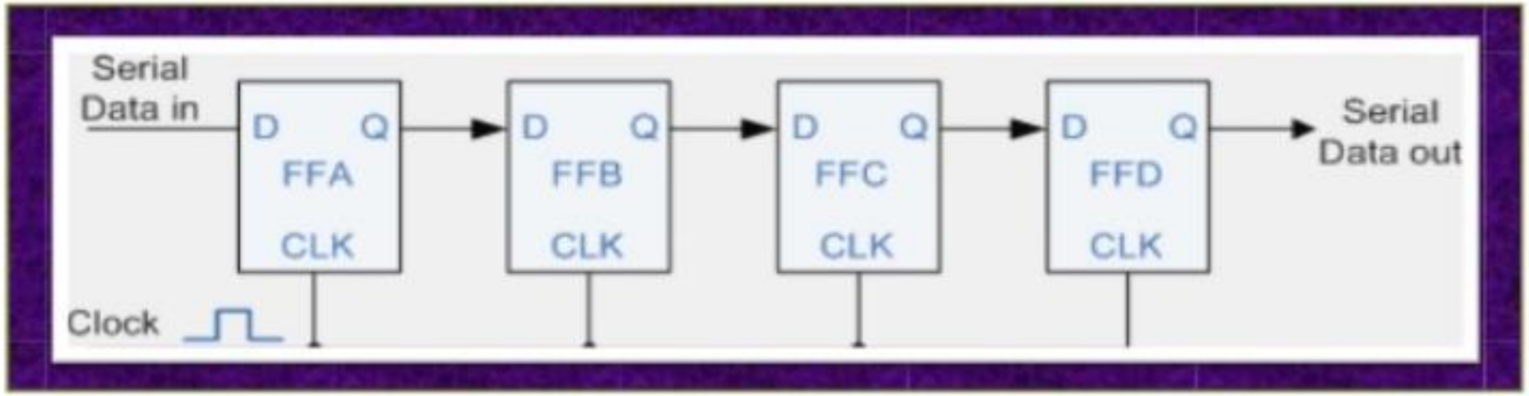
REGISTERS

- ✓ Collection of a group of flip flops and logic gates which affect their transition.
- ✓ Basic function is to hold information.
- ✓ It may have combinational gates that perform certain data processing tasks in addition to flip flops.
- ✓ It consists of a group of flip flops and gates that effect their transition.

SHIFT REGISTERS

- ✓ A REGISTER capable of shifting binary information in one or both directions.
- ✓ Consists of a series of flip flops cascaded together with the output of one flip flop connected to the input of the other flip flop.
- ✓ All flip flops receive common clock pulses that initiate the shift from one stage to the next.
- ✓ They are generally provided with a Clear or Reset connection so that they can be "SET" or "RESET" as required.
- ✓ Controlled with certain clock pulses by inhibiting the clock from the input of the register if shift is not required.
- ✓ It is also able to provide extra circuits to control the shift operation through the D inputs of the flip flops rather than the clock input.

4 BIT SHIFT REGISTERS



- ✓ Common clock pulse.

Clk(011)	d0	d1	d2	d3
0	0	0	0	0
1, 1	1	0	0	0
2,1	1	1	0	0
3,0	0	1	1	0

CHARACTERISTICS

- ✓ Provide necessary input and output terminals for parallel data transfer.
- ✓ An input for clock pulses to synchronize all operations.
- ✓ A control state that leaves the information in the register unchanged even through clock pulses are applied continuously.

Shift registers are of two types depending upon the direction of shift. They are:

- ✓ Unidirectional: Capable of shifting only in one direction
- ✓ Bidirectional: Capable of shifting in both directions

APPLICATIONS

- ✓ Used to interface digital systems.
- ✓ Can be used as simple delay circuits.
- ✓ They can be used to convert serial data to parallel data commonly in microprocessor based system.
- ✓ It can also be used for multiplication by shifting left and division by shifting right.

Integrated Circuits

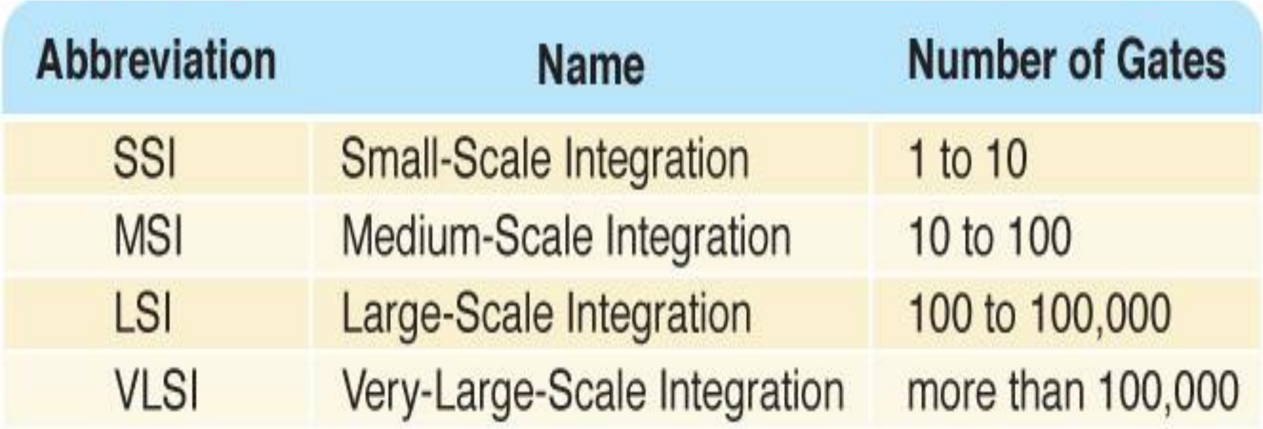
Integrated circuit (also called a *chip*)

A piece of silicon on which multiple gates have been embedded

Silicon pieces are mounted on a plastic or ceramic package with pins along the edges that can be soldered onto circuit boards or inserted into appropriate sockets

Integrated Circuits

Integrated circuits (IC) are classified by the number of gates contained in them



Abbreviation	Name	Number of Gates
SSI	Small-Scale Integration	1 to 10
MSI	Medium-Scale Integration	10 to 100
LSI	Large-Scale Integration	100 to 100,000
VLSI	Very-Large-Scale Integration	more than 100,000

Integrated Circuits

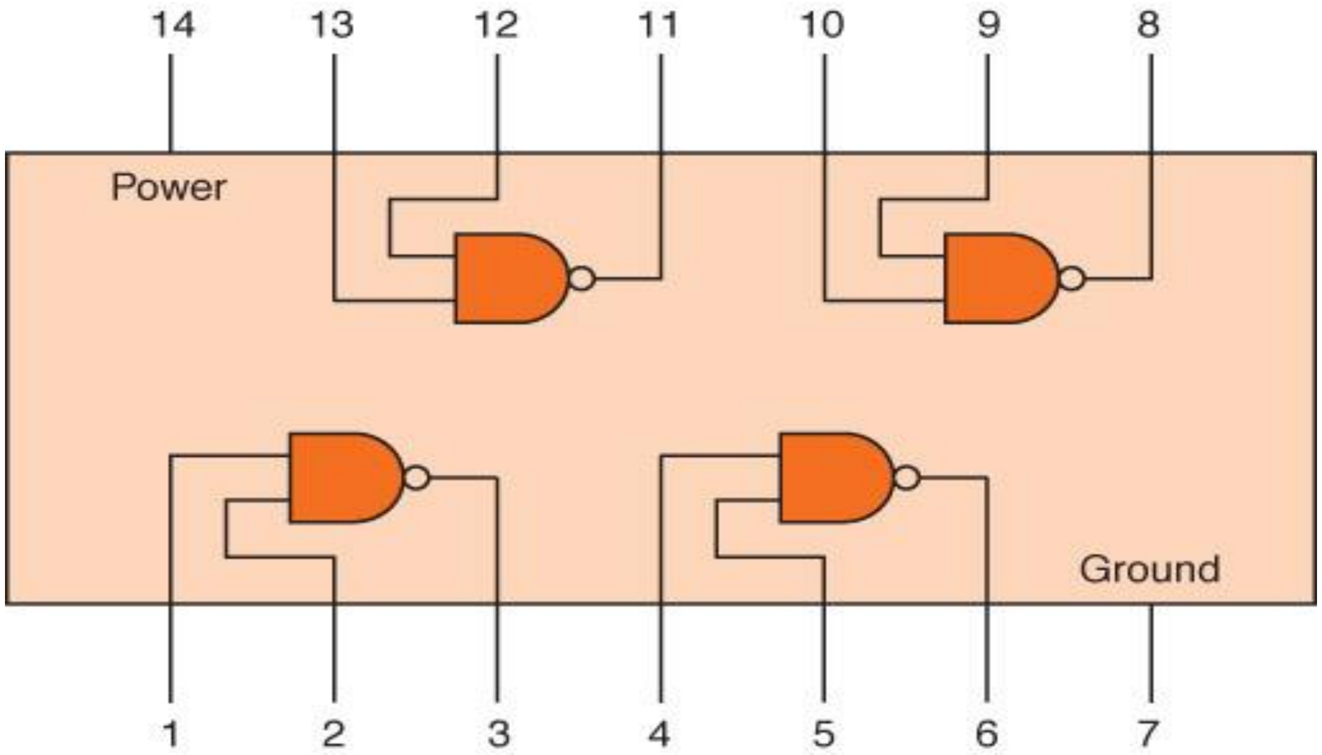


Figure 13 An SSI chip contains independent NAND gates