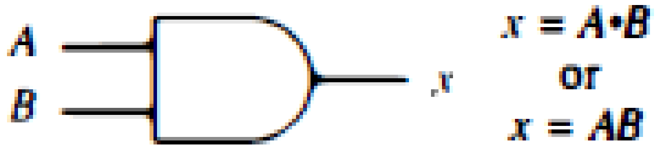
Logic Gates

- Binary logic deals with binary variables and with operations that assume a logical meaning.
- The manipulation of binary information is done by logic circuits called gates .
- Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied.
- A variety of logic gates are commonly used in digital computer systems.

- Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression.
- The input-output relationship of the binary variables for each gate can be represented in tabular form by a truth table.

AND

- The AND gate produces the AND logic function: that is, the output is 1 if input A and input B are both equal to 1; otherwise, the output is 0.



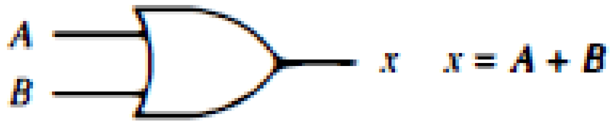
Graphic Symbol

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

Truth Table

OR

- The OR gate produces the inclusive-OR function; that is, the output is 1 if input A or input B or both inputs are l; otherwise, the output is 0.



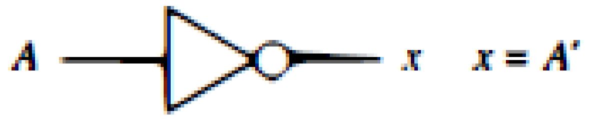
Graphic Symbol

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

Truth Table

INVERTER

- The inverter circuit inverts the logic sense of a binary signal. It produces the NOT, or complement, function.
- The algebraic symbol used for the logic complement is either a prime or a bar over the variable symbol.



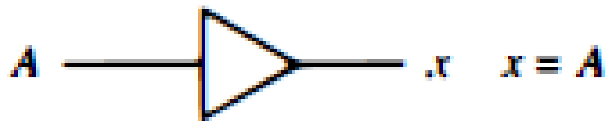
Graphic Symbol

A	x
0	1
1	0

Truth Table

BUFFER

- A triangle symbol by itself designates a buffer circuit.
- A buffer does not produce any particular logic function since the binary value of the output is the same as the binary value of the input.



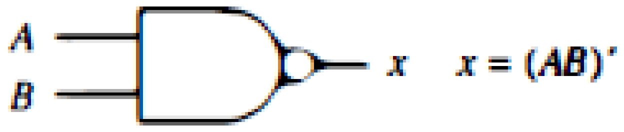
Graphic Symbol

A	x
0	0
1	1

Truth Table

NAND

- The NAND function is the complement of the AND function.
- It consists of an AND graphic symbol followed by a small circle.



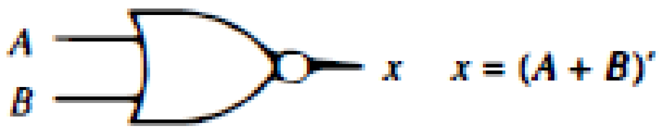
Graphic Symbol

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

Truth Table

NOR

❖ The NOR gate is the complement of the OR gate and uses an OR graphic symbol followed by a small circle.



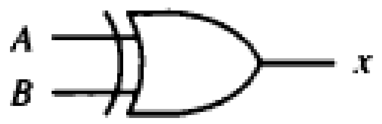
Graphic Symbol

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

Truth Table

Exclusive OR(XOR)

- The exclusive-OR gate has a graphic symbol similar to the OR gate except for the additional curved line on the input side. The output of this gate is I if any input is 1 but excludes the combination when both inputs are 1.



$x = A \oplus B$

or

$x = A'B + AB'$

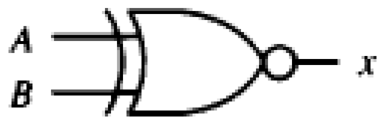
Graphic Symbol

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

Truth Table

Exclusive NOR (XNOR)

- The exclusive-NOR is the complement of the exclusive-OR.



$x = (A \oplus B)'$

or

$x = A'B' + AB$

Graphic Symbol

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

Truth Table

Boolean Algebra

- Boolean algebra is an algebra that deals with binary variables and logic operations.
- The variables are designated by letters such as A, B, x, and y.
- The three basic logic operations are AND, OR, and complement.
- Consider, for example, the Boolean function.

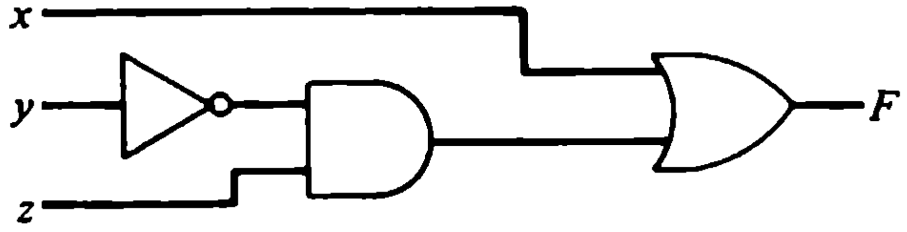
$$F = x + y'z$$

Truth Table

- For $F = x + y'z$

x	y	z	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

Truth Table



Logic Diagram

Boolean Algebra Function

❖ The purpose of Boolean algebra is to facilitate the analysis and design of digital circuits. It provides a convenient tool to:

1. Express in algebraic form a truth table relationship between binary variables.
2. Express in algebraic form the input-output relationship of logic diagrams.
3. Find simpler circuits for the same function.

Boolean Expression

- A Boolean function specified by a truth table can be expressed algebraically in many different ways.
- By manipulating a Boolean expression according to Boolean algebra rules, one may obtain a simpler expression that will require fewer gates.
- Table 1-1 lists the most basic identities of Boolean algebra.

Rules for Boolean Expressions

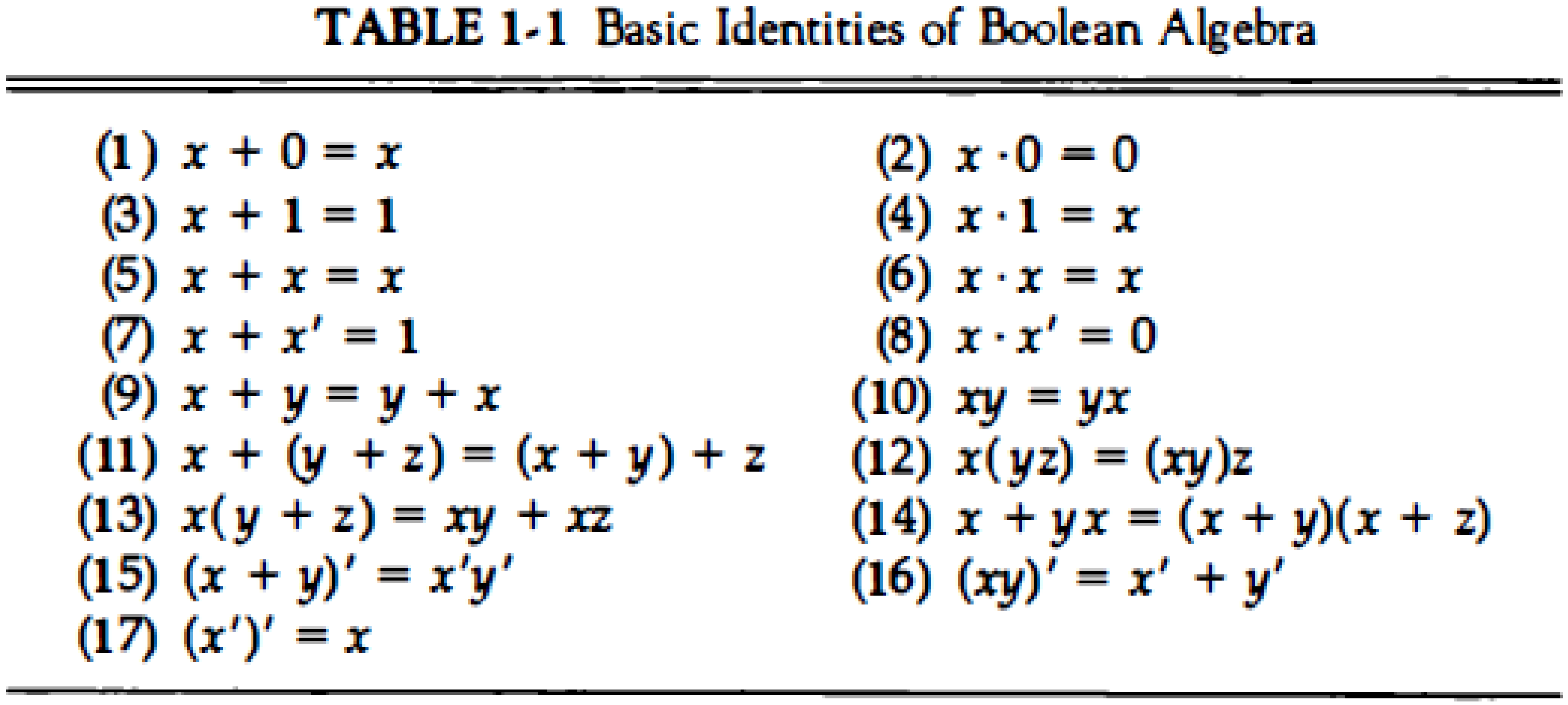


TABLE 1-1 Basic Identities of Boolean Algebra

(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) $(x')' = x$	

Rules for Boolean Expressions

- Identity 14 does not apply in ordinary algebra but is very useful in manipulating Boolean expressions.
- Identities 15 and 16 are called DeMorgan's theorems.
- DeMorgan' s theorem is very important in dealing with NOR and NAND gates. It states that a NOR gate that performs the $(x + y)'$ function is equivalent to the function $x'y'$ Similarly, a NAND function can be expressed by either $(xy)'$ or $(x' + y')$.
- The last identity states that if a variable is complemented twice, one obtains the original value of the variable.

Example

$$AB' + C'D + AB' + C'D$$

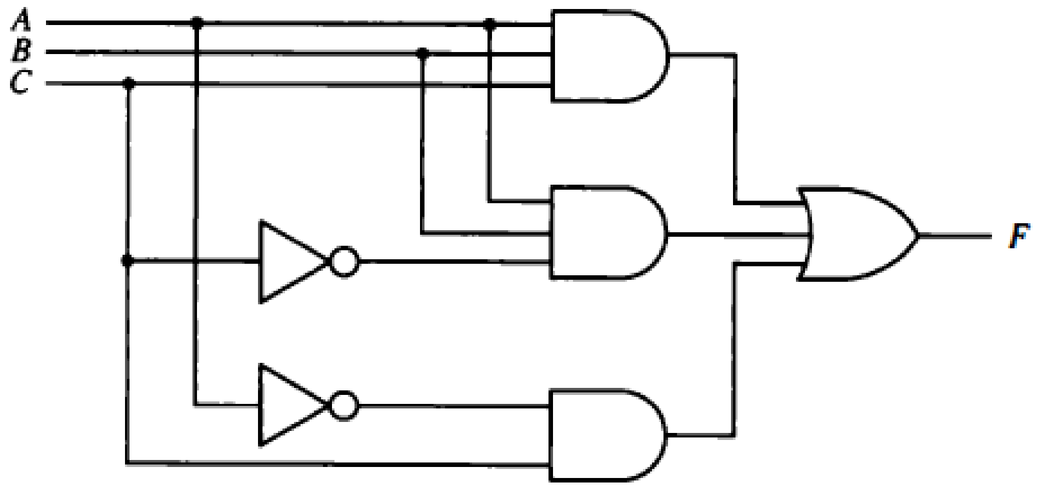
- By letting $x = AB' + C'D$ the expression can be written as $x + x$. From *identity 5* in Table 1-1 we find that $x + x = x$. Thus the expression can be reduced to only two terms:

$$AB' + C'D + A'B + C'D = AB' + C'D$$

Boolean Expressions Example

- To see how Boolean algebra manipulation is used to simplify digital circuits, consider the logic diagram.

$$F = ABC + ABC' + A'C$$



(a) $F = ABC + ABC' + A'C$

Boolean Expressions Example

- The expression can be simplified using Boolean algebra.

$$F = ABC + ABC' + A'C$$

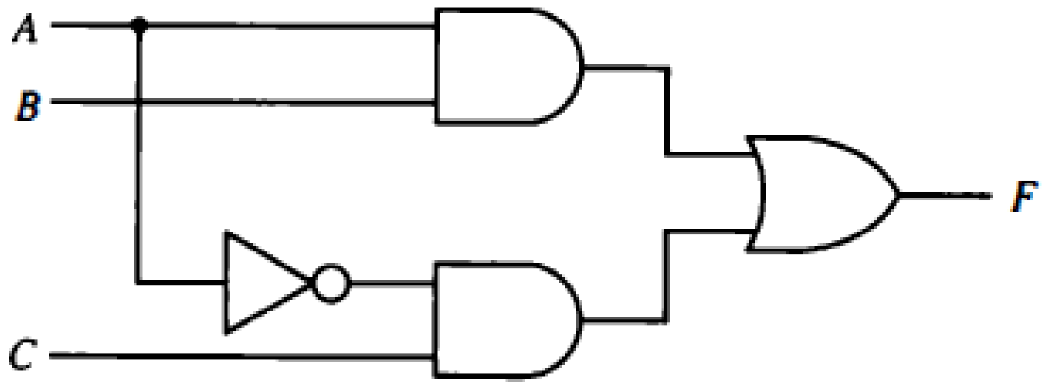
$$= AB(C + C') + A'C$$

$$= AB + A'C$$

- Note that ***(C +C)' = 1*** by identity 7 and ***AB ·1 = AB*** by identity 4.

Boolean Expressions Example

❖ The logic diagram of the simplified expression



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

Complement of a Function

- When the function is expressed in algebraic form, the complement of the function can be derived by means of DeMorgan's theorem. The general form of DeMorgan's theorem can be expressed as follows:

$$(x_1 + x_2 + x_3 + \cdots + x_n)' = x_1' x_2' x_3' \cdots x_n'$$

$$(x_1 x_2 x_3 \cdots x_n)' = x_1' + x_2' + x_3' + \cdots + x_n'$$

Complement of a Function

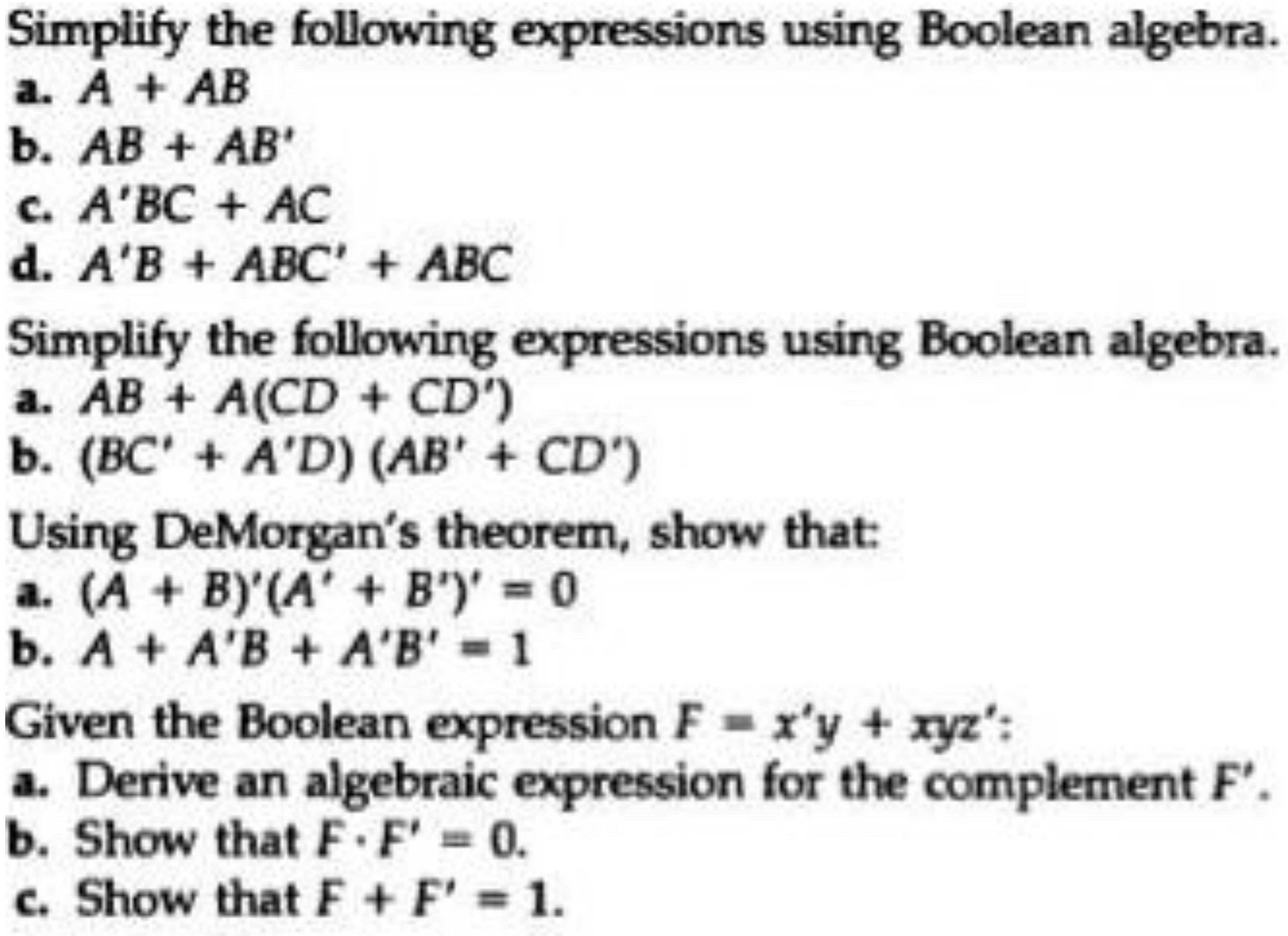
- As an example, consider the following expression and its complement:

$$F = AB + C'D' + B'D$$

- The complement expression is obtained by interchanging AND and OR operations and complementing each individual variable. Note that the complement of ***C ' is C*** .

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

Exercise



Simplify the following expressions using Boolean algebra.

a. $A + AB$

b. $AB + AB'$

c. $A'BC + AC$

d. $A'B + ABC' + ABC$

Simplify the following expressions using Boolean algebra.

a. $AB + A(CD + CD')$

b. $(BC' + A'D)(AB' + CD')$

Using DeMorgan's theorem, show that:

a. $(A + B)'(A' + B')' = 0$

b. $A + A'B + A'B' = 1$

Given the Boolean expression $F = x'y + xyz'$:

a. Derive an algebraic expression for the complement F'.

b. Show that $F \cdot F' = 0$.

c. Show that $F + F' = 1$.

Map Simplification

- The map method provides a simple, straightforward procedure for simplifying Boolean expressions.
- This method may be regarded as a pictorial arrangement of the truth table which allows an easy interpretation for choosing the minimum number of terms needed to express the function algebraically.
- The map method is also known as the Karnaugh map or K-map.

K-map

- Each combination of the variables in a truth table is called a ***minterm***.
- When expressed in a truth table a function of n variables will have 2^n ***minterms***, equivalent to the 2^n binary numbers obtained from n bits.
- The information contained in a truth table may be expressed in compact form by listing the decimal equivalent of those ***minterms*** that produce a 1 for the function.

Example for K-map

$$F = x + y'z$$

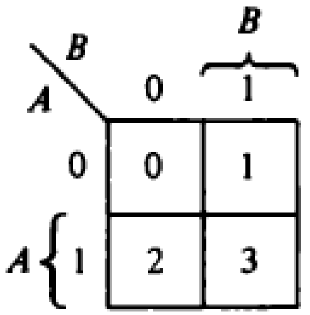
- The truth table of expression can be expressed as follows

$$F(x, y, z) = \sum (1, 4, 5, 6, 7)$$

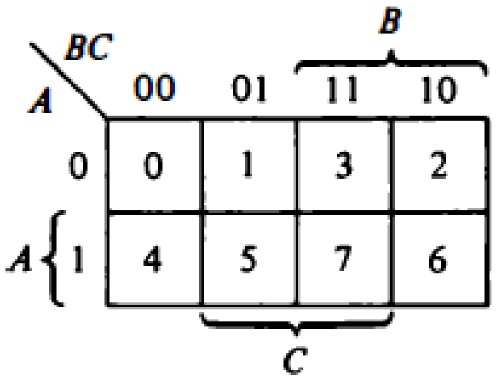
- The letters in parentheses list the binary variables in the order that they appear in the truth table. The symbol $\sum$ stands for the sum of the min terms that follow in parentheses.
- The min terms that produce 1 for the function are listed in their decimal equivalent.

- The map is a diagram made up of squares, with each square representing one minterm. The squares corresponding to minterms that produce 1 for the function are marked by a 1 and the others are marked by a 0 or are left empty.

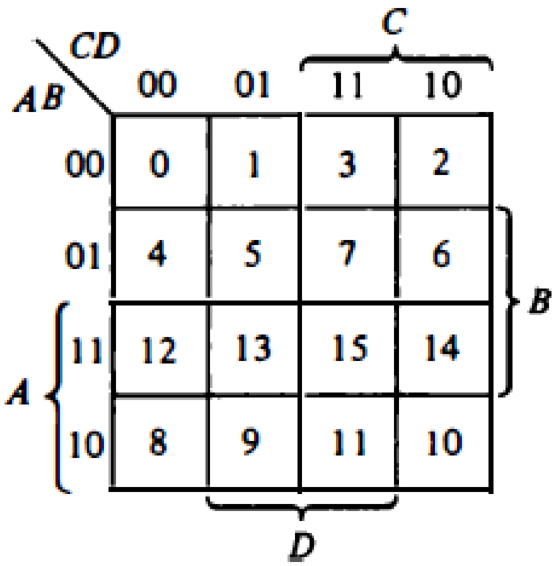
Maps for Two, Three and Four Variables



(a) Two-variable map



(b) Three-variable map

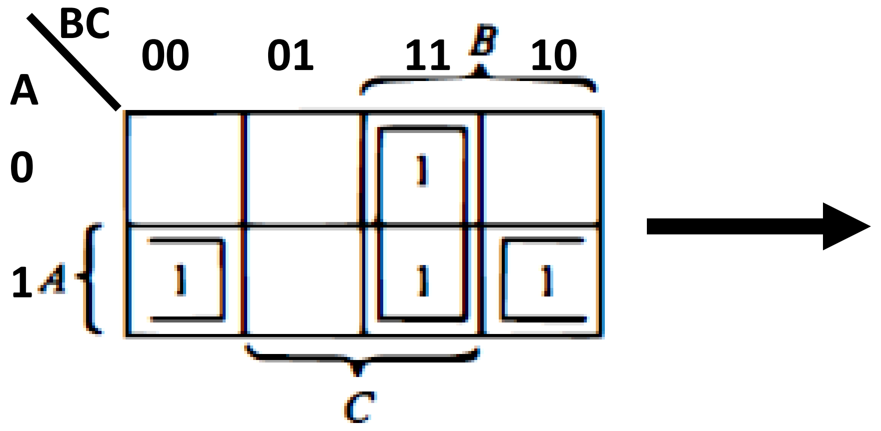


(c) Four-variable map

K-map Example

- In the first example we will simplify the Boolean function

$$F(A,B,C) = \sum(3,4,6,7)$$



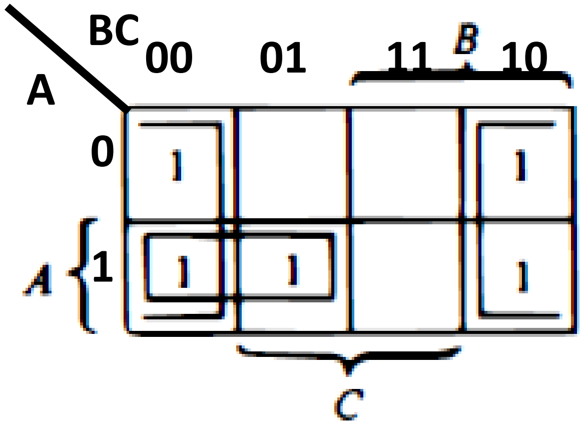
$$F = BC + AC'$$

K-map Example

- Two adjacent squares are combined in the third column. This column belongs to both **B** and **C** and produces the term **BC**.
- The remaining two squares with 1's in the two comers of the second row are adjacent and belong to row **A** and the two columns of **C'**, so they produce the term **AC'**.

K-map Example-2

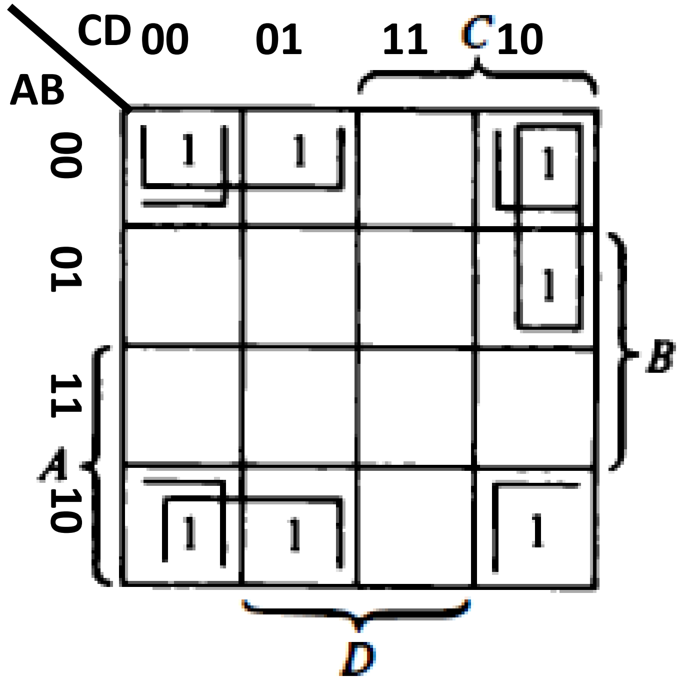
$$F(A, B, C) = \sum (0, 2, 4, 5, 6)$$



$$\longrightarrow \quad F = C' + AB'$$

K-map Example-3

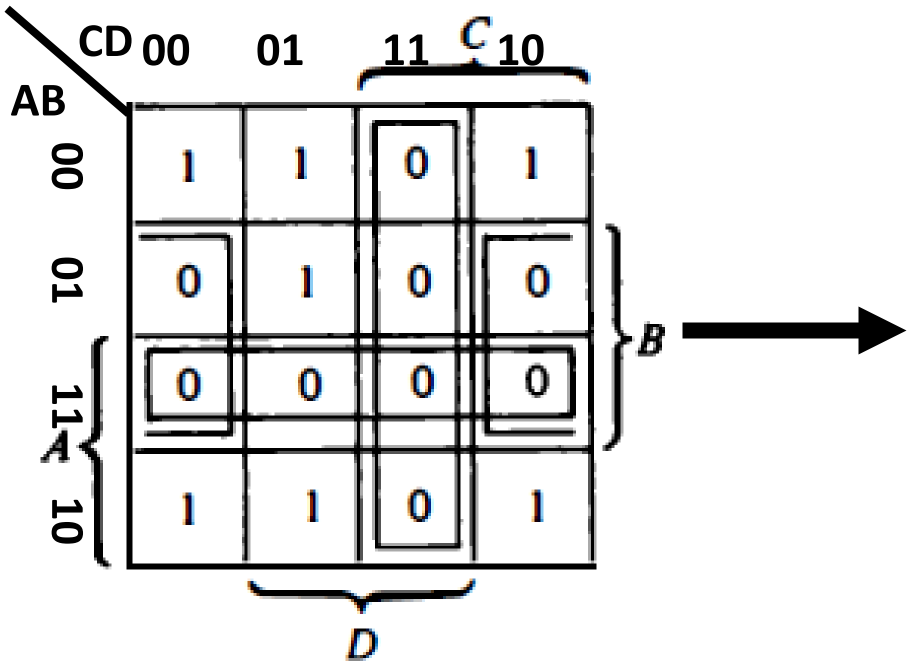
$$F(A,B,C,D) = \sum (0,1,2,6,8,9,10)$$



$$\longrightarrow \quad F = B'D' + B'C' + A'CD'$$

K-map Example-4

$$F(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$



Sum of Products

$$F = B'D' + B'C' + A'C'D$$

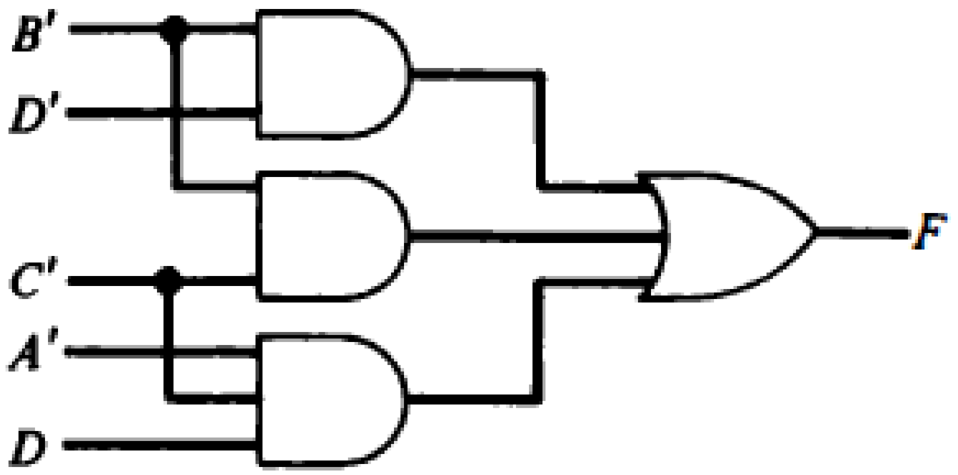
Product of Sum

$$F' = AB + CD + BD'$$

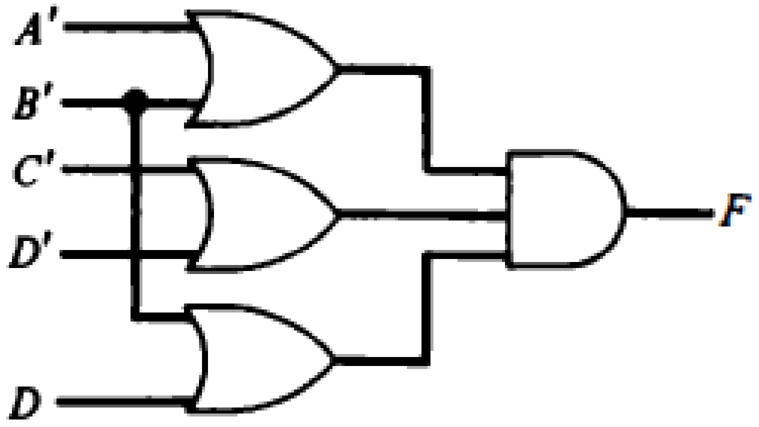
$$F = (A' + B')(C' + D')(B' + D)$$

Implementation

Logic diagrams with AND and OR gates.



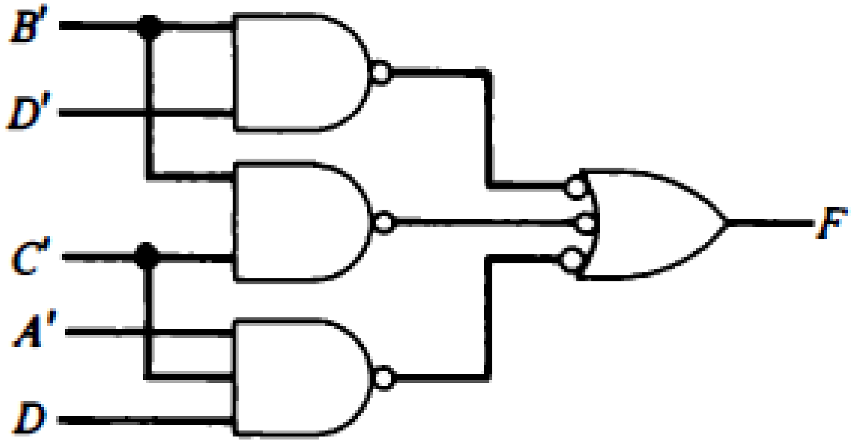
(a) Sum of products:
$F = B'D' + B'C' + A'C'D$



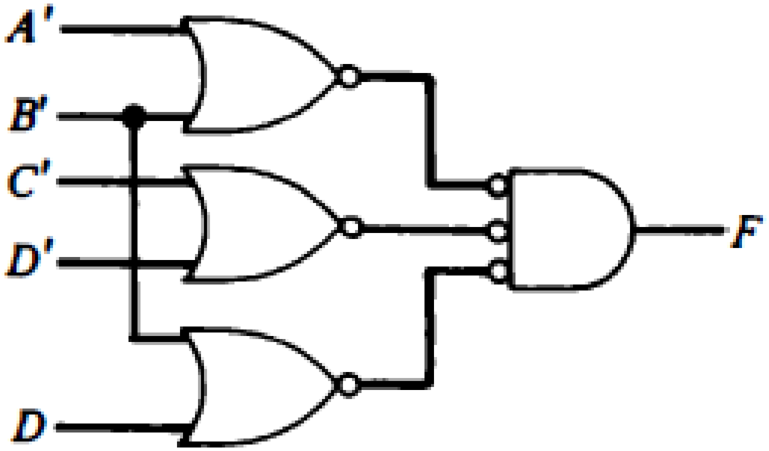
(b) Product of sums:
$F = (A' + B')(C' + D')(B' + D)$

Implementation

Logic diagrams with NAND or NOR gates.



(a) With NAND gates



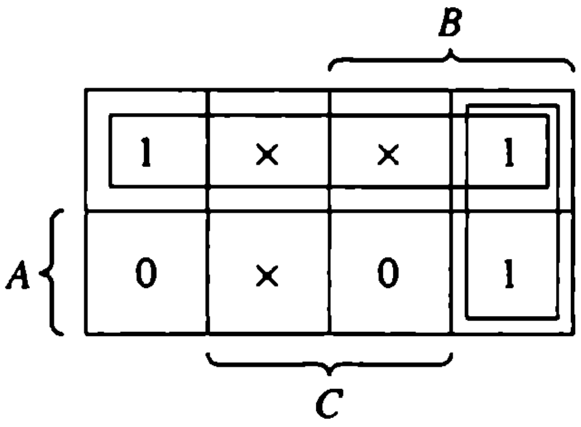
(b) With NOR gates

Don't-Care Conditions

- Min terms that don't-care conditions may produce either 0 or 1 for the function are said to be don't-care conditions and are marked with an x in the map.
- The x 's may be assumed to be either 0 or 1, whichever gives the simplest expression. In addition, an x need not be used at all if it does not contribute to the simplification of the function.

$$F(A,B,C) = \Sigma(0,2,6)$$
$$d(A,B,C) = \Sigma(1,3,5)$$

- The minterms listed with F produce a 1 for the function.
- The don't-care minterms listed with d may produce either a 0 or a 1 for the function.



simplified expression for F would have been

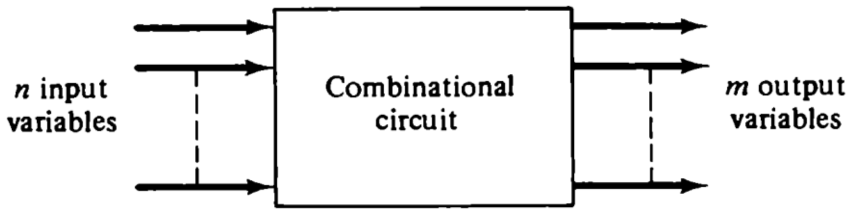
$$F = A'C' + BC'$$

- The function is determined completely once the x 's are assigned to the 1's or O's in the map. Thus the expression.

$$F = A' + BC'$$

Combinational Circuits

- A combinational circuit is a connected arrangement of logic gates with a set of inputs and outputs.
- The n binary input variables come from an external source, the m binary output variables go to an external destination, and in between there is an interconnection of logic gates.
- A combinational circuit can be described by a truth table showing the binary relationship between the n input variables and the m output variables.



Combinational Circuits

The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram. The procedure involves the following steps:

1. The problem is stated.

2. The input and output variables are assigned letter symbols.

3. The truth table that defines the relationship between inputs and outputs

is derived.

4. The simplified Boolean functions for each output are obtained.

5. The logic diagram is drawn.

Half-Adder

- The most basic digital arithmetic circuit is the addition of two binary digits.
- A combinational circuit that performs the arithmetic addition of two bits is called a half-adder.
- The input variables of a half-adder are called the augend and addend bits.
- The output variables the sum and carry.
- It is necessary to specify two output variables because the sum of 1 + 1 is binary 10, which has two digits.

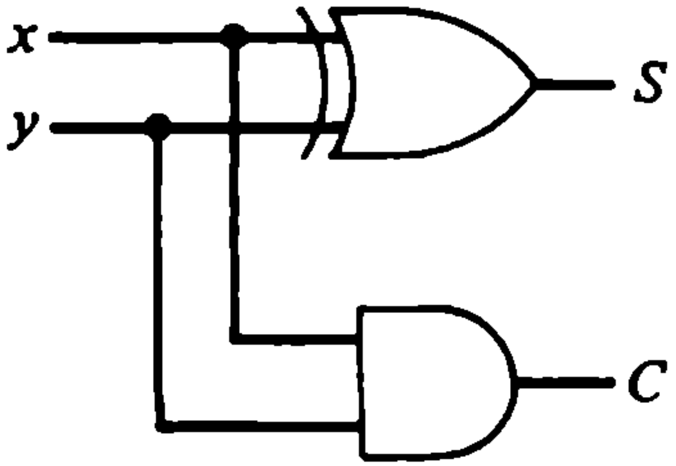
Half-Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

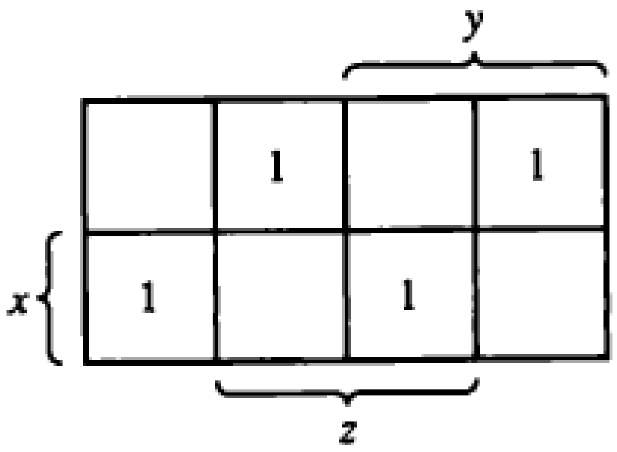
Truth Table

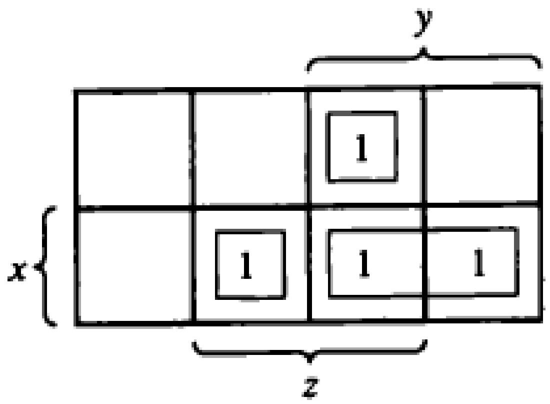
$S = x'y + xy' = x \oplus y$

$C = xy$



Logic Diagram for Half Adder





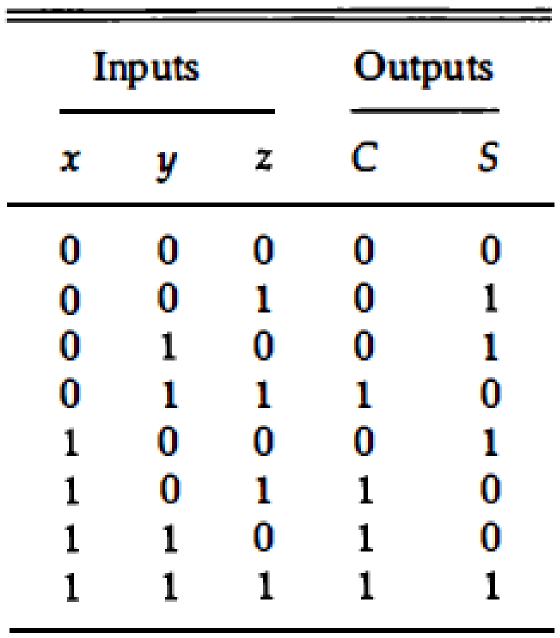
$S = x'y'z + x'yz' + xy'z' + xyz$
$= x \oplus y \oplus z$

$C = xy + xz + yz$
$= xy + (x'y + xy')z$

Full Adder

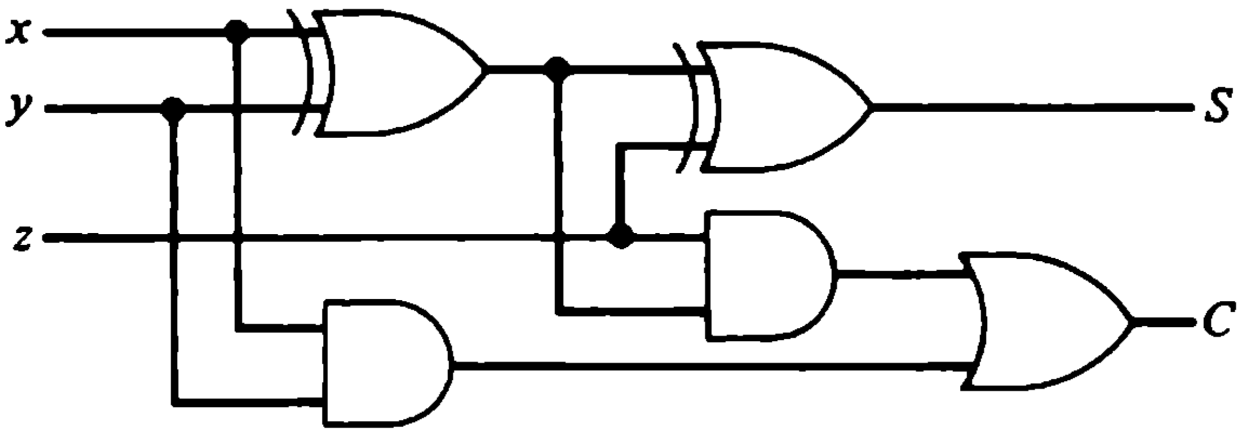
- A full-adder is a combinational circuit that forms the arithmetic sum of three input bits.

Truth Table

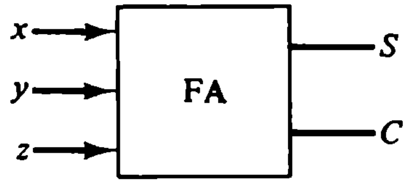


Inputs			Outputs	
x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Logic Diagram of Full Adder



(a) Logic diagram



(b) Block diagram

Sequential Circuit

- The digital circuits considered thus far have been combinational, where the outputs at any given time are entirely dependent on the inputs that are present at that time.
- Most systems encountered in practice also include storage elements, which require that the system be described in terms of sequential circuits.
- The most common type of sequential circuit is the synchronous type.
- Synchronization is achieved by a timing device called a clock pulse generator that produces a periodic train of clock pulses.

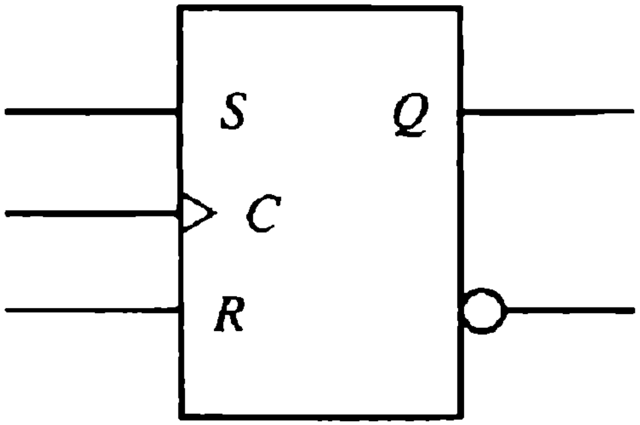
Flip Flop

- The storage elements employed in clocked sequential circuits are called flip-flops.
- A flip-flop is a binary cell capable of storing one bit of information.
- It has two outputs, one for the normal value and one for the complement value of the bit stored in it.
- A flip-flop maintains a binary state until directed by a clock pulse to switch states.

SR Flip-Flop

- It has three inputs, labeled S (for set), R (for reset), and C (for clock).
- It has an output Q and sometimes the flip-flop has a complemented output, which is indicated with a small circle at the other output terminal.
- There is an arrowhead-shaped symbol in front of the letter C to designate a dynamic input.
- The flip-flop responds to a positive transition (from 0 to 1) of the input clock signal .

S-R Flip Flop



(a) Graphic symbol

S	R	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

(b) Characteristic table

S-R Flip Flop Operation

- If there is no signal at the clock input C, the output of the circuit cannot change irrespective of the values at inputs S and R.
- Only when the clock signal changes from 0 to 1 can the output be affected according to the values in inputs S and R.
- If S = 1 and R = 0 when C changes from 0 to 1, output Q is set to 1 .
- If S = 0 and R = 1 when C changes from 0 to 1, output Q is cleared to 0.
- If both S and R are 0 during the clock transition, the output does not change .
- When both S and R are equal to 1, the output is unpredictable and may go to either 0 or 1.

S-R Flip Flop Operation

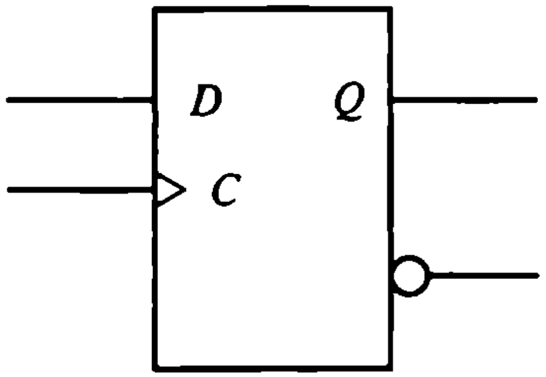
- Q(t) is the binary state of the Q output at a given time (referred to as present state).
- Q(t + 1) is the binary state of the Q output after the occurrence of a clock transition (referred to as next state).
- If S = R = 0, a clock transition produces no change of state [i. e . , Q(t + 1) = Q(t)] .
- If S = 0 and R = 1 , the flip-flop goes to the 0 (clear) state . If S = 1 and R = 0, the flip-flop goes to the 1 (set) state . The SR flip-flop should not be pulsed when S = R = 1 since it produces an indeterminate next state . This indeterminate condition makes the SR flip-flop difficult to manage and therefore it is seldom used in practice .

D Flip-Flop

- D flip-flop by inserting an inverter between S and R and assigning the symbol D to the single input.
- The D input is sampled during the occurrence of a clock transition from 0 to 1. If D = 1, the output of the flip-flop goes to the 1 state, but if D = 0, the output of the flip-flop goes to the 0 state .
- From the characteristic table we note that the next state Q(t + 1) is determined from the D input. The relationship can be expressed by a characteristic equation:

$$Q(t+1) = D$$

D Flip-Flop



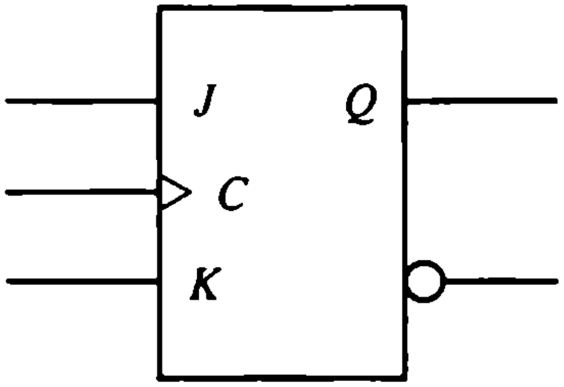
(a) Graphic symbol

D	$Q(t+1)$	
0	0	Clear to 0
1	1	Set to 1

(b) Characteristic table

JK Flip-Flop

- A JK flip-flop is a refinement of the SR flip-flop in that the indeterminate condition of the SR type is defined in the JK type.
- JK flip-flop has a complement condition Q(t + 1) = Q ' (t) when both J and K are equal to 1 .



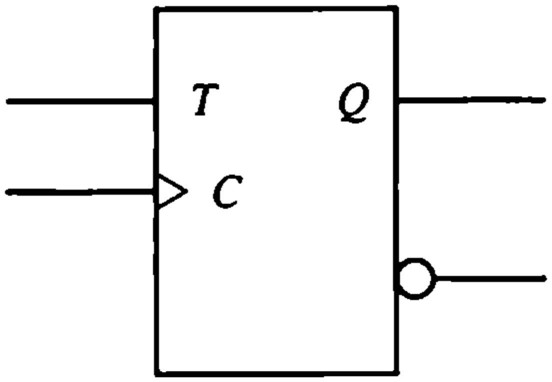
(a) Graphic symbol

J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$Q'(t)$	Complement

(b) Characteristic table

T Flip-Flop

- The T flip-flop therefore has only two conditions. When T = 0 (J = K = 0) a clock transition does not change the state of the flip-flop. When T = 1 (J = K = 1) a clock transition complements the state of the flip-flop.



(a) Graphic symbol

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

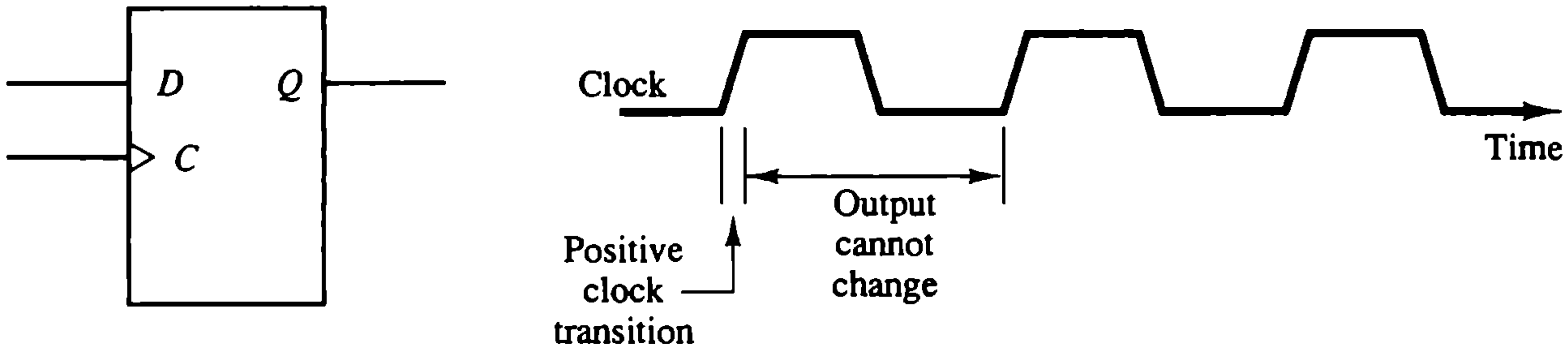
(b) Characteristic table

Edge Triggered Flip Flops

- The most common type of flip-flop used to synchronize the state change during a clock pulse transition is the edge-triggered flip-flop. In this type of flip-flop, output transitions occur at a specific level of the clock pulse.
- Some edge-triggered flip-flops cause a transition on the rising edge of the clock signal (positive-edge transition), and others cause a transition on the falling edge (negative-edge transition).

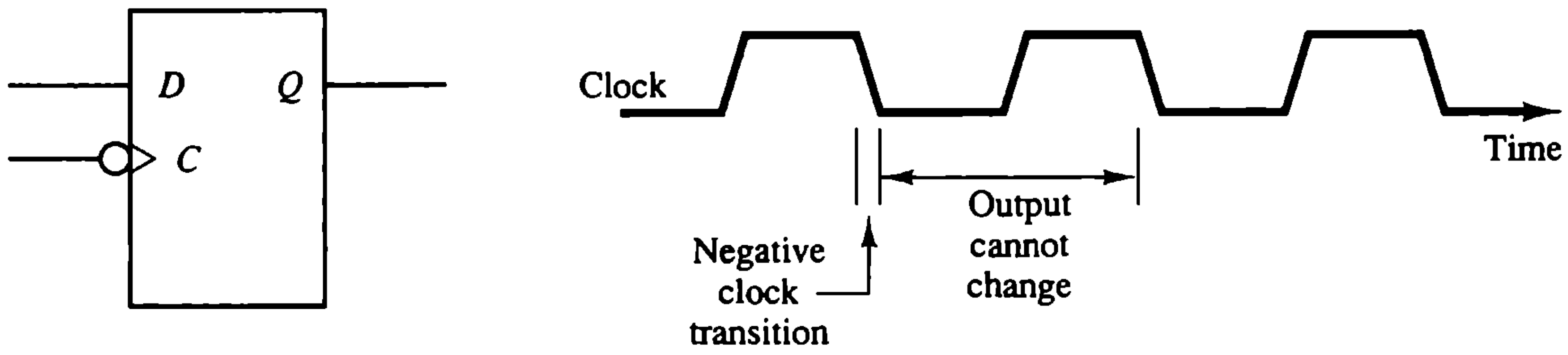
Positive-Edge-Triggered

- Figure shows the clock pulse signal in a positive-edge-triggered D flip-flop. The value in the D input is transferred to the Q output when the clock makes a positive transition.



(a) Positive-edge-triggered *D* flip-flop.

Negative-Edge Triggered



(b) Negative-edge-triggered *D* flip-flop.

- ❖ In this case the flip-flop responds to a transition from the 1 level to the 0 level of the clock signal.

Master-Slave flip-flop

- This type of circuit consists of two flip-flops .
- The first is the master, which responds to the positive level of the clock, and the second is the slave, which responds to the negative level of the clock.
- The result is that the output changes during the 1-to-0 transition of the clock signal.
- The trend is away from the use of master-slave flip-flops and toward edge-triggered flip-flops.

Excitation Table

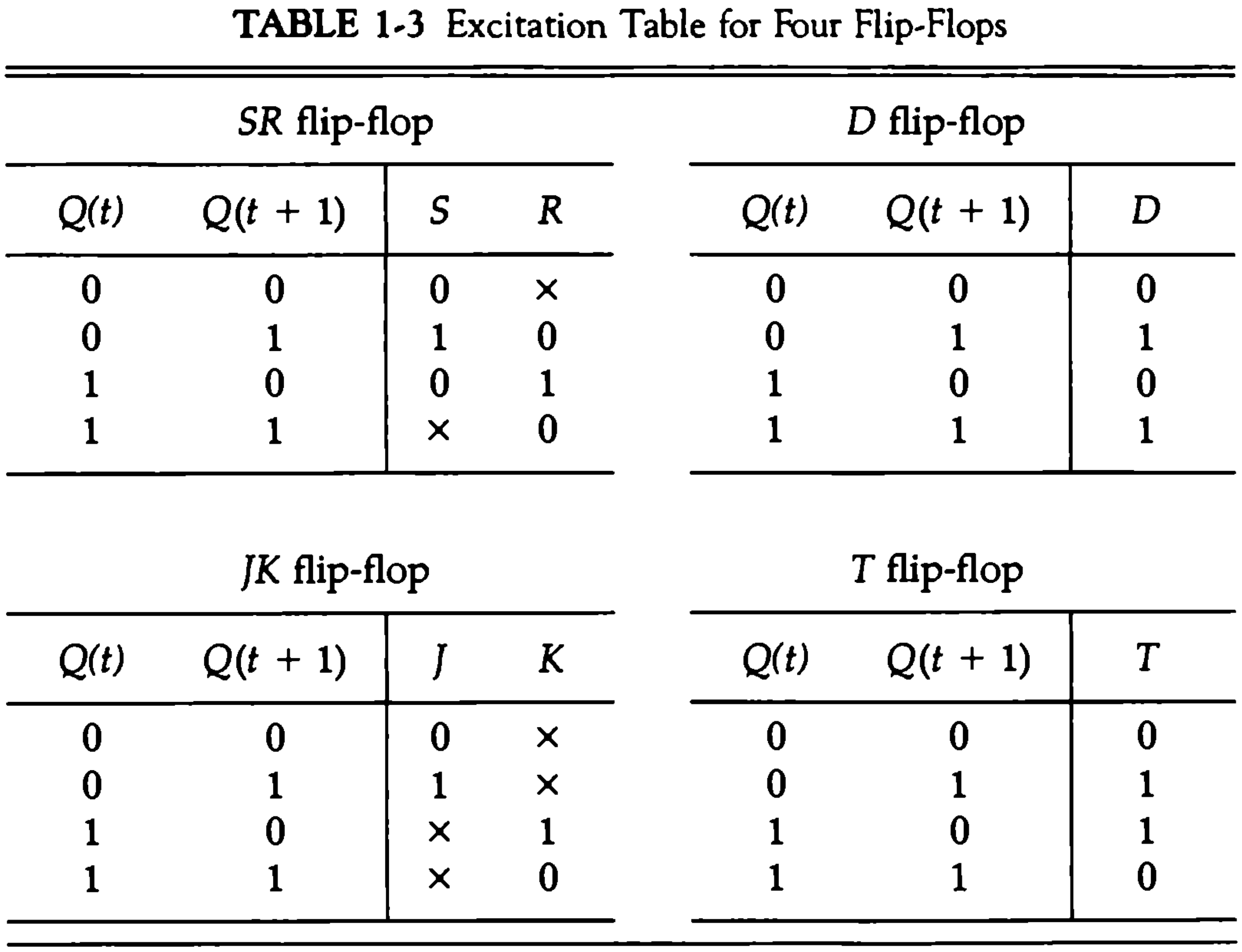


TABLE 1-3 Excitation Table for Four Flip-Flops

SR flip-flop

$Q(t)$	$Q(t+1)$	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

D flip-flop

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

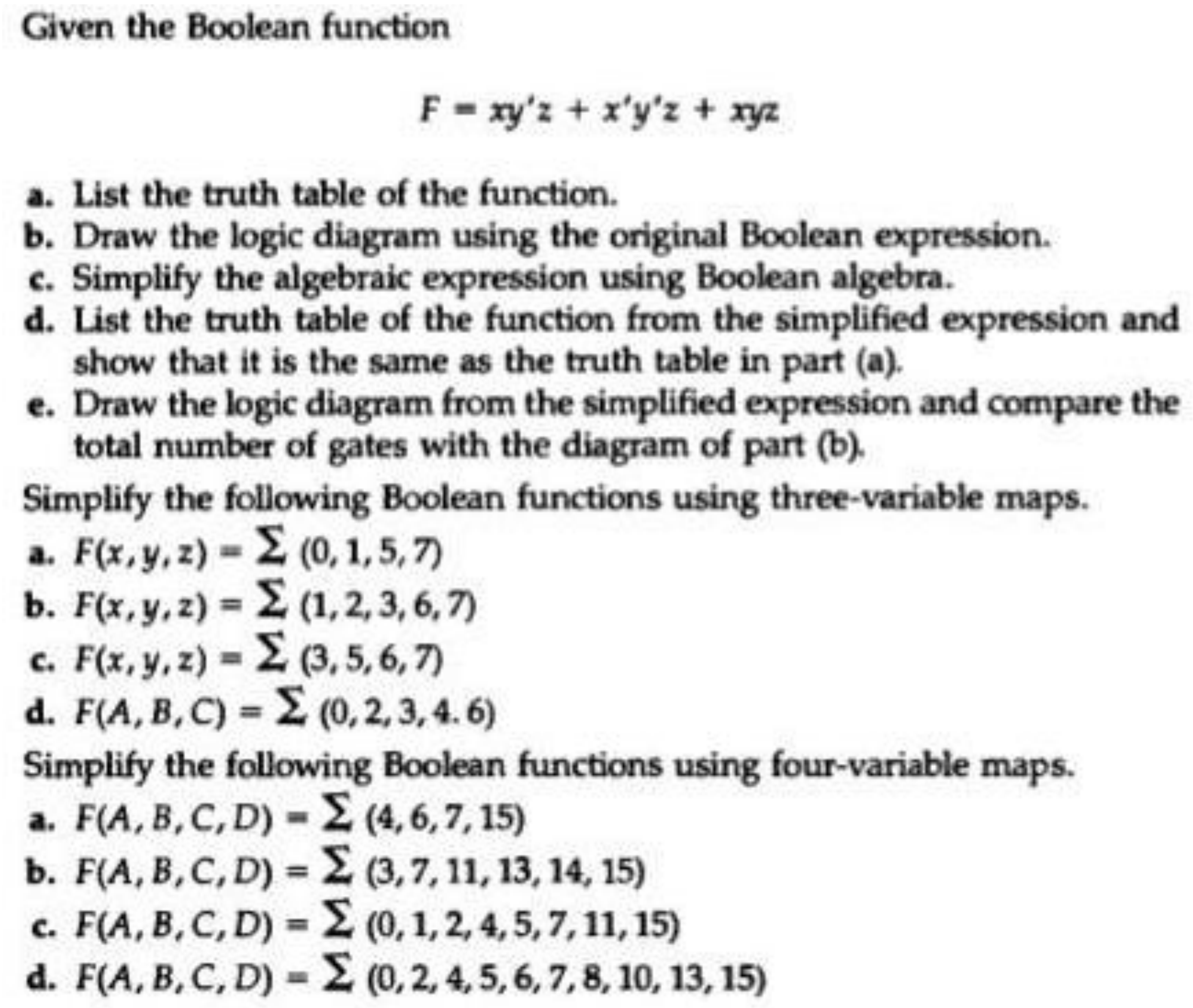
JK flip-flop

$Q(t)$	$Q(t+1)$	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

T flip-flop

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

Exercise



Given the Boolean function

$$F = xy'z + x'y'z + xyz$$

a. List the truth table of the function.
b. Draw the logic diagram using the original Boolean expression.
c. Simplify the algebraic expression using Boolean algebra.
d. List the truth table of the function from the simplified expression and show that it is the same as the truth table in part (a).
e. Draw the logic diagram from the simplified expression and compare the total number of gates with the diagram of part (b).

Simplify the following Boolean functions using three-variable maps.

a. $F(x, y, z) = \sum (0, 1, 5, 7)$
b. $F(x, y, z) = \sum (1, 2, 3, 6, 7)$
c. $F(x, y, z) = \sum (3, 5, 6, 7)$
d. $F(A, B, C) = \sum (0, 2, 3, 4. 6)$

Simplify the following Boolean functions using four-variable maps.

a. $F(A, B, C, D) = \sum (4, 6, 7, 15)$
b. $F(A, B, C, D) = \sum (3, 7, 11, 13, 14, 15)$
c. $F(A, B, C, D) = \sum (0, 1, 2, 4, 5, 7, 11, 15)$
d. $F(A, B, C, D) = \sum (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

Exercise

Simplify the following expressions in (1) sum-of-products form and (2) product-of-sums form.

a. $x'z' + y'z' + yz' + xy$

b. $AC' + B'D + A'CD + ABCD$

Simplify the following Boolean function in sum-of-products form by means of a four-variable map. Draw the logic diagram with (a) AND-OR gates; (b) NAND gates.

$$F(A, B, C, D) = \sum (0, 2, 8, 9, 10, 11, 14, 15)$$

Simplify the following Boolean function in product-of-sums form by means of a four-variable map. Draw the logic diagram with (a) OR-AND gates; (b) NOR gates.

$$F(w, x, y, z) = \sum (2, 3, 4, 5, 6, 7, 11, 14, 15)$$

Simplify the Boolean function *F* together with the don't-care conditions *d* in (1) sum-of-products form and (2) product-of-sums form.

$$F(w, x, y, z) = \sum (0, 1, 2, 3, 7, 8, 10)$$

$$d(w, x, y, z) = \sum (5, 6, 11, 15)$$

Simplify the following Boolean functions using three-variable maps.

$$F(x, y, z) = \sum (3, 5, 6, 7)$$
$$F(A, B, C) = \sum (0, 2, 3, 4. 6)$$
$$F(A, B, C, D) = \sum (4, 6, 7, 15)$$
$$F(A, B, C, D) = \sum (3, 7, 11, 13, 14, 15)$$

Simplify the Boolean function *F* together with the don't-care conditions *d* in (1) sum-of-products form and (2) product-of-sums form.

$$F(w, x, y, z) = \sum (0, 1, 2, 3, 7, 8, 10)$$
$$d(w, x, y, z) = \sum (5, 6, 11, 15)$$

Design a 2-bit count-down counter. This is a sequential circuit with two flip-flops and one input x. When $x = 0$, the state of the flip-flops does not change. When $x = 1$, the state sequence is 11, 10, 01, 00, 11, and repeat.

Chapter-2 (Digital Components)

- Integrated Circuits
- Decoders
- Multiplexers
- Registers
- Shift Registers
- Binary Counters
- Memory Unit

Integrated Circuits

IC

- Digital circuits are constructed with integrated circuits. An integrated circuit (abbreviated IC) is a small silicon semiconductor crystal. called a chip, containing the electronic components for the digital gates.

SSI (Small Scale integration)

- Small Scale integration (SSI) devices contain several independent gates in a single package. The inputs and outputs of the gates are connected directly to the pins in the package.

MSI(Medium-scale integration)

- Medium-scale integration (MSI) devices have a complexity of approximately 10 to 200 gates in a single package. They usually perform specific elementary digital functions such as decoders, adders, and registers.

Large-scale integration (LSI)

- Large-scale integration (LSI) devices contain between 200 and a few thousand gates in a single package. They include digital systems, such as processors, memory chips, and programmable modules.

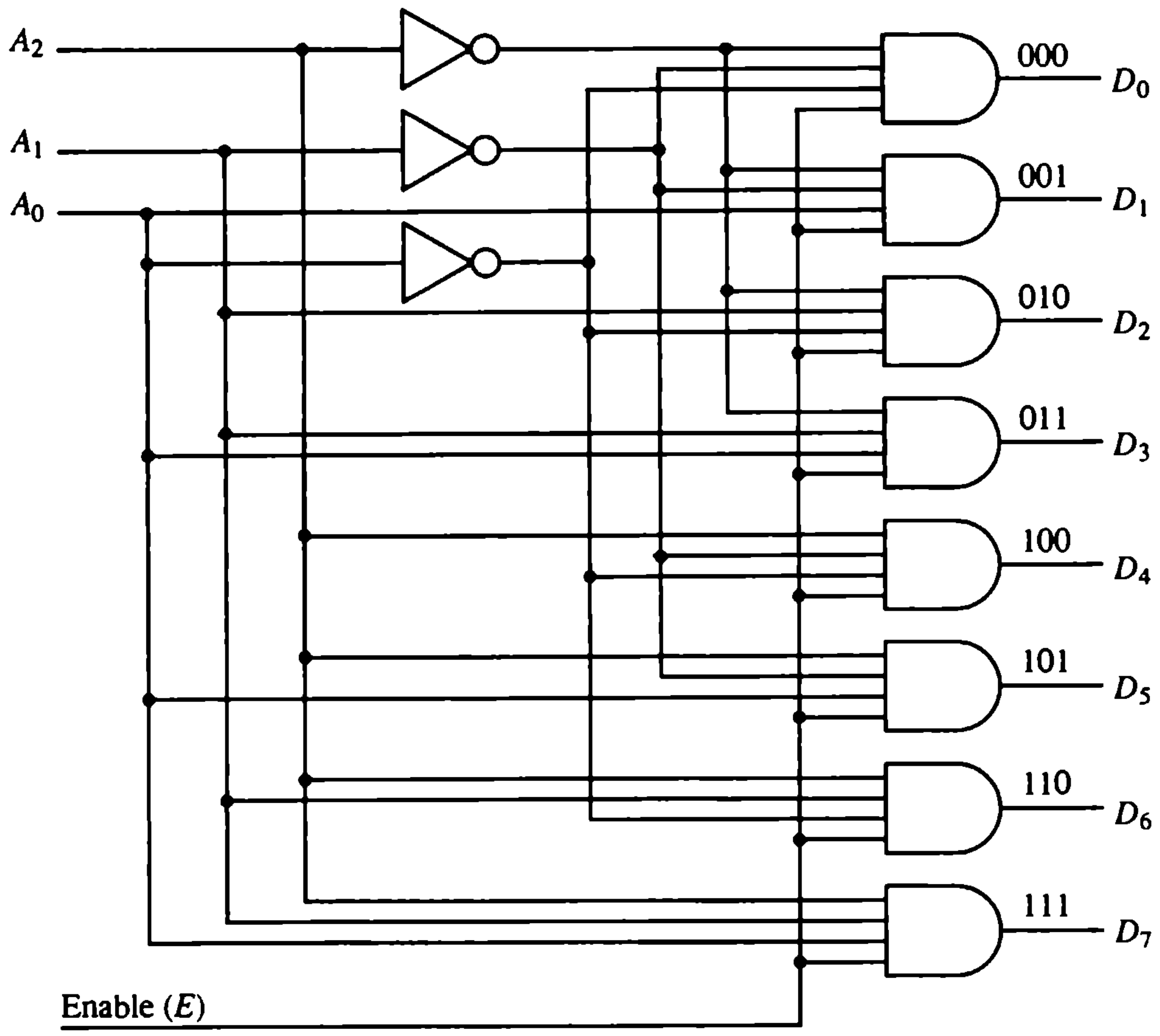
Very-large-scale integration (VLSI)

- Very-large-scale integration (VLSI) devices contain thousands of gates TTL within a single package. Examples are large memory arrays and complex microcomputer chips.
- Many different logic families of integrated circuits have been introduced commercially. The following are the most popular.
- TTL----Transistor-transistor logic
- ECL---Emitter-coupled logic
- MOS ----Metal-oxide semiconductor
- CMOS--Complementary metal-oxide semiconductor

Decoders

- A decoder is a combinational circuit that converts binary information from the n coded inputs to a maximum of 2^nunique outputs.
- If the n-bit coded information has unused bit combinations, the decoder may have less than 2^n outputs.
- A decoder has *n* inputs and *m* outputs and is also referred to as an *n x m* decoder.
- The logic diagram of a 3-to-8-line decoder is shown in Figure.

3-to-8-Line Decoder



Truth Table for 3-to-8-Line Decoder

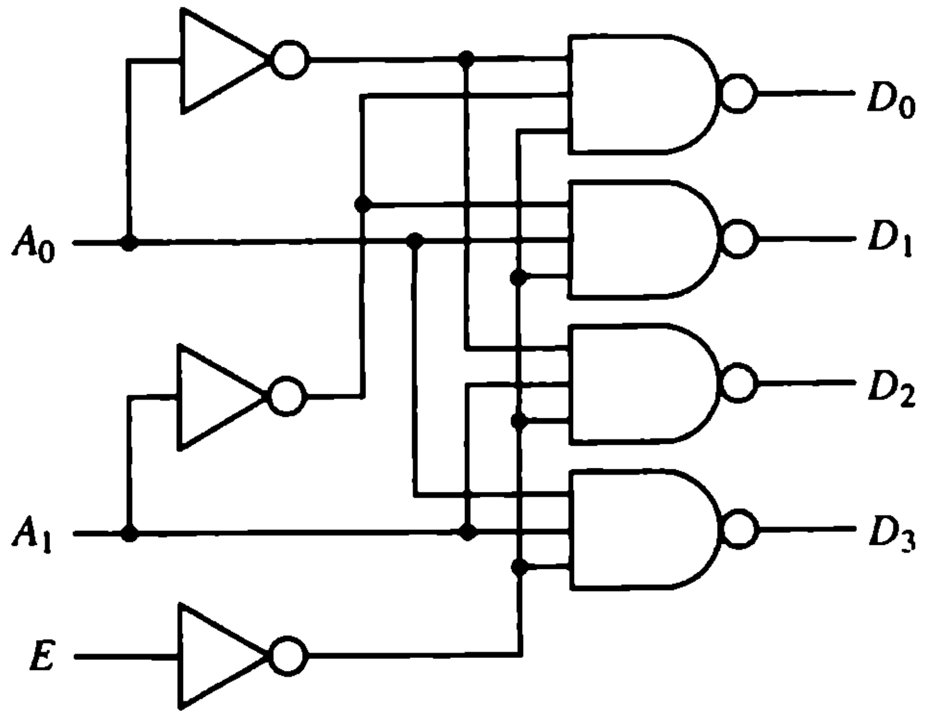
❖ The decoder is enabled when E is equal to 1 and disabled when E is equal to 0.

Enable	Inputs			Outputs							
E	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

NAND Gate Decoder

- The decoder is enabled when E is equal to 0.
- The output whose value is equal to 0 represents the equivalent binary number in inputs A_1 and A_0.
- The circuit is disabled when E is equal to 1.

2-to-4-Line Decoder with NAND



(a) Logic diagram

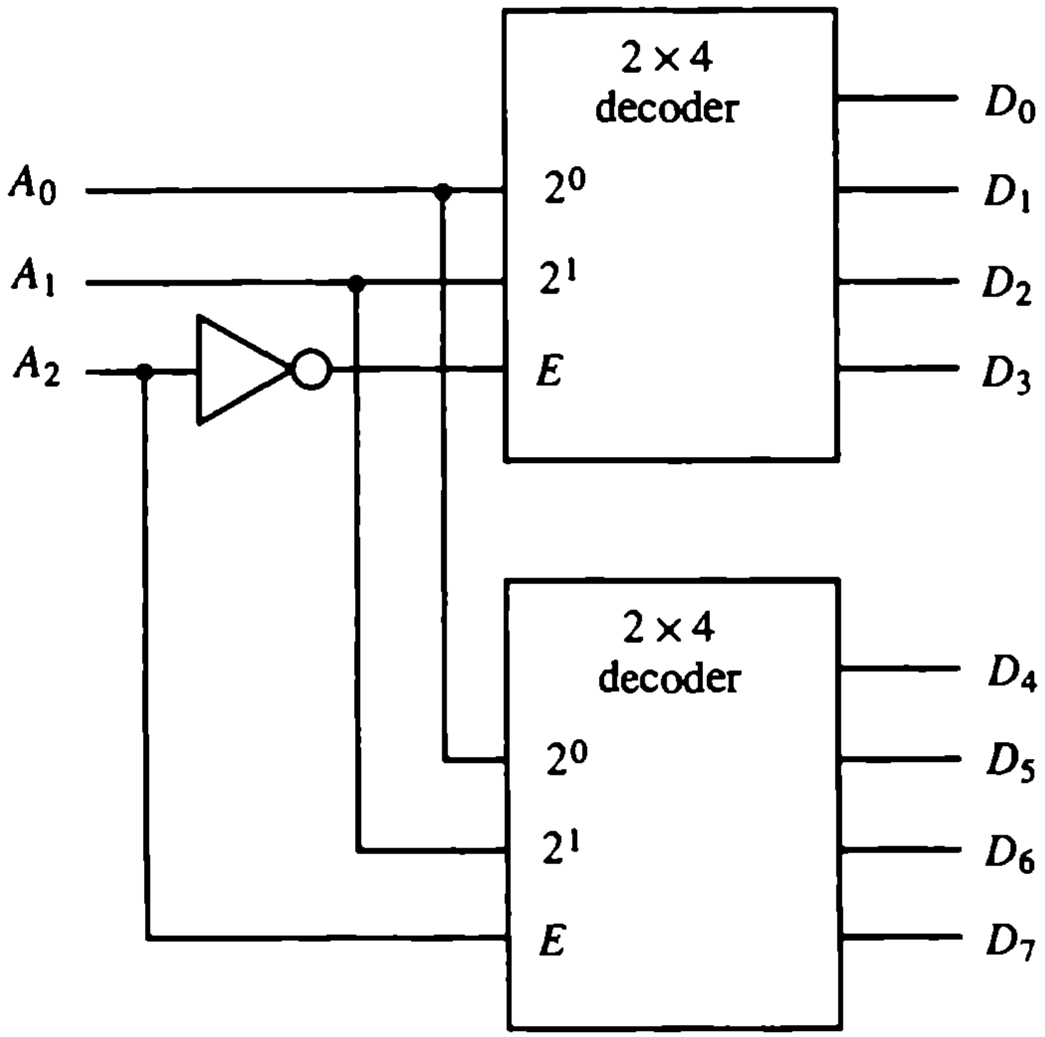
E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	×	×	1	1	1	1

(b) Truth table

Decoder Expansion

- It is possible to combine two or more decoders with enable inputs to form a larger decoder.
- Thus if a 6-to-64-line decoder is needed, it is possible to construct it with four 4-to-16-line decoders.
- Two 2-to-4-line decoders are combined to achieve a 3-to-8-line decoder.
- When $A_2 = 0$, the upper decoder is enabled and the lower is disabled. The lower decoder outputs become inactive with all outputs at 0. The outputs of the upper decoder generate outputs Do through D3, depending on the values of A1 and A0 (while $A_2 = 0$). When $A_2 = 1$, the lower decoder is enabled and the upper is disabled. The lower decoder output generates the binary equivalent D4 through D, since these binary numbers have a 1 in the A_2, position.

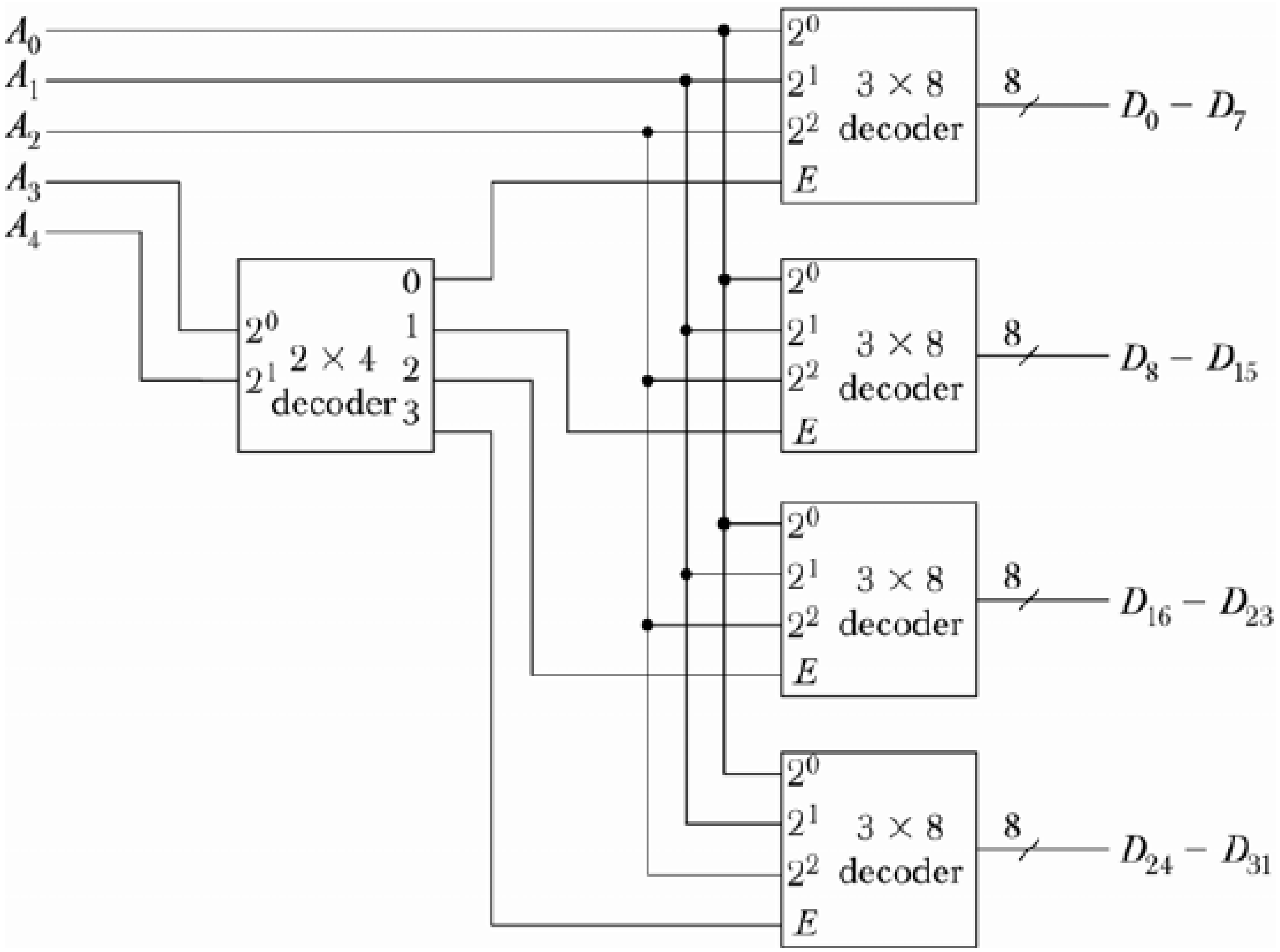
3×8 *DECODER WITH TWO* 2×4



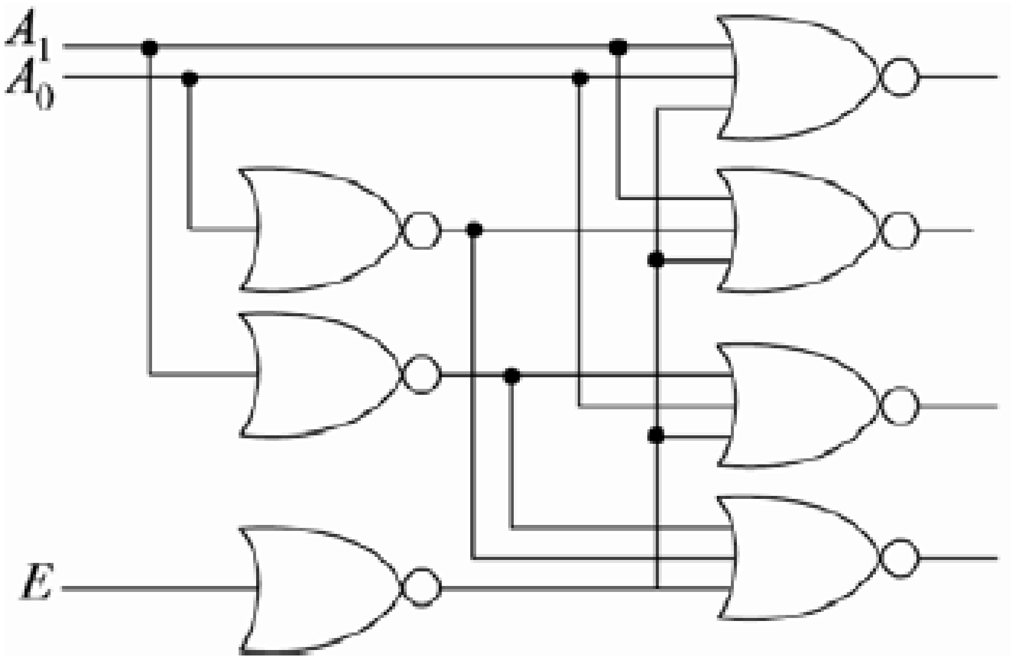
Problem

- Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and one 2-to-4 line decoder. Use block diagrams.
- Draw the logic diagram of a 2-to-4-line decoder with only NOR gates. Include an enable input.
- Implement the function f(w1, w2, w3) = (0, 1, 3, 4, 6, 7) by using a 3-to-8 binary decoder and an OR gate.

- Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and one 2-to-4 line decoder. Use block diagrams.
- Draw the logic diagram of a 2-to-4-line decoder with only NOR gates. Include an enable input.



A_0
A_1
A_2
A_3
A_4
2^0
2^1
2×4 decoder
0
1
2
3
2^0
2^1
2^2
E
3×8 decoder
8
$D_0 - D_7$
$D_8 - D_{15}$
$D_{16} - D_{23}$
$D_{24} - D_{31}$



$D_0 = (A_1 + A_0 + E')' = A'_1 A'_0 E$

$D_1 = (A_1 + A'_0 + E')' = A'_1 A_0 E$

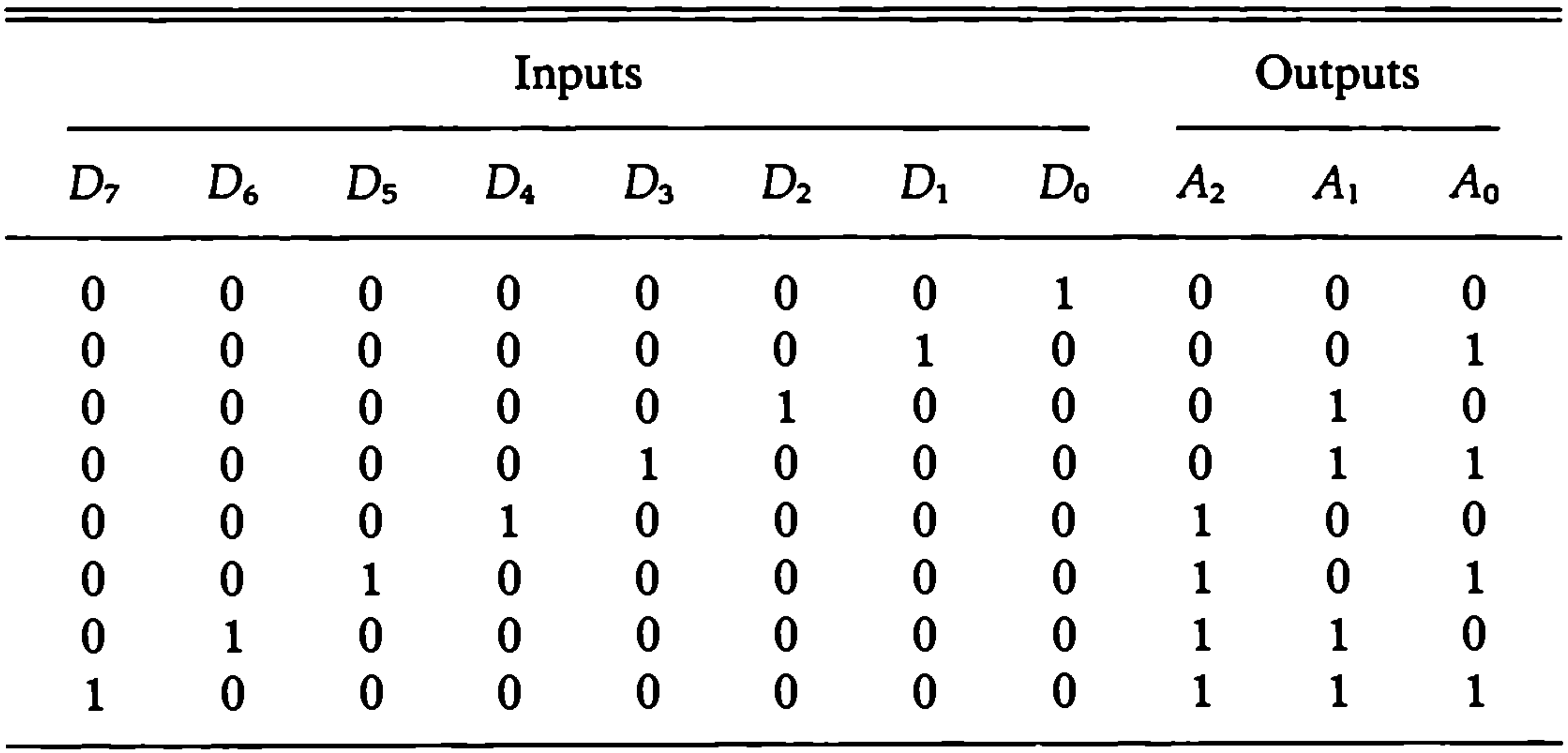
$D_2 = (A'_1 + A_0 + E')' = A_1 A'_0 E$

$D_3 = (A'_1 + A'_0 + E')' = A_1 A_0 E$

Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or less) input lines and n output lines.
- The output lines generate the binary code corresponding to the input value.
- An example of an encoder is the octal-to-binary encoder, whose truth table is given in Table.

Octal-to-Binary Encoder Truth Table



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Encoder

- The encoder can be implemented with OR gates.
- Output Ao = 1 if the input octal digit is 1 or 3 or 5 or 7.

$$A_0 = D_1 + D_3 + D_5 + D_7$$

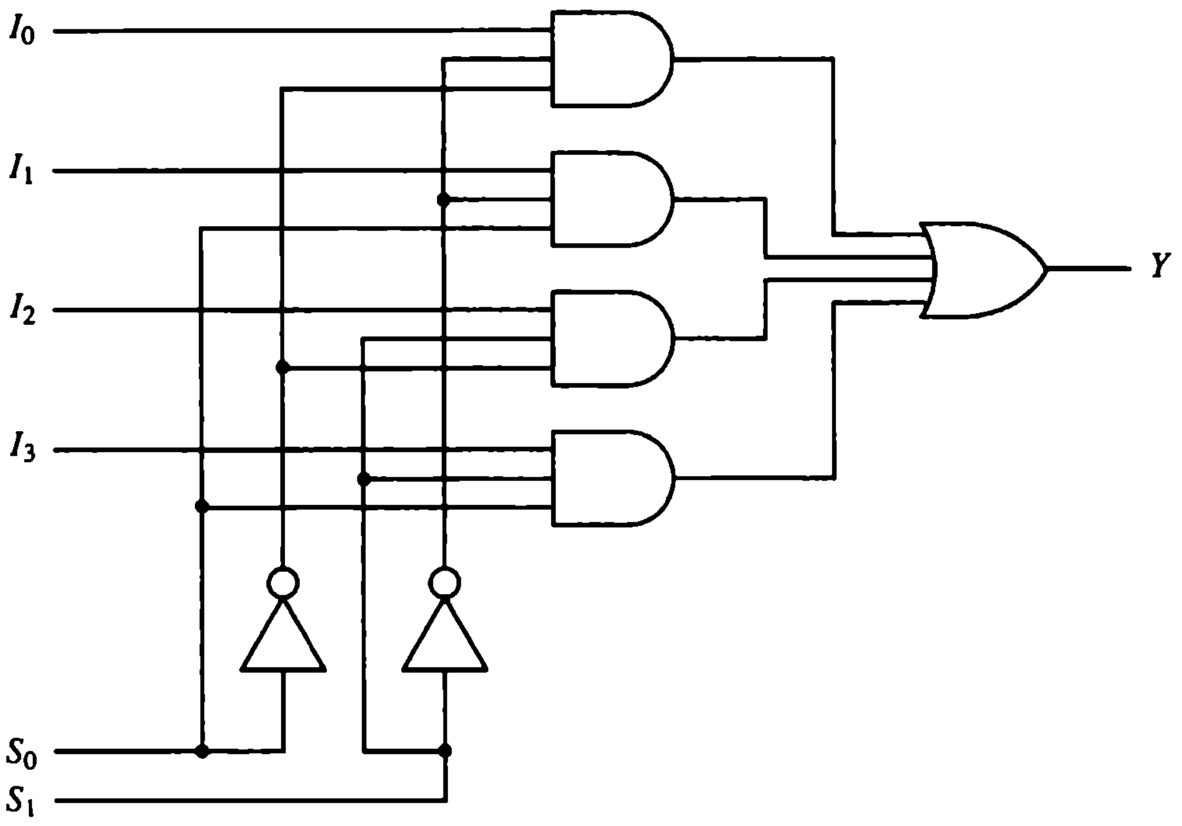
$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Multiplexers

- A multiplexer is a combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single output line.
- The selection of a particular input data line for the output is determined by a set of selection inputs.
- A 2^n-to-1 multiplexer has 2^n input data lines and n input selection lines whose bit combinations determine which input data are selected for the output.

4-to-1 Line Multiplexer



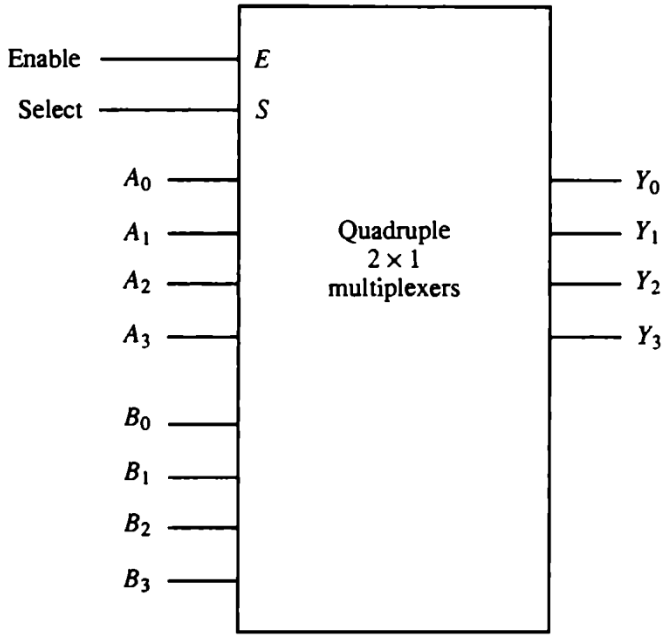
Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

4-to-1 Line Multiplexer

- To demonstrate the circuit operation, consider the case when $S_0S_1 = 10.$
- The AND gate associated with input I_2, has two of its inputs equal to 1.
- The third input of the gate is connected to I_2.
- The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0.
- The OR gate output is now equal to the value of I_2 thus providing a path from the selected input to the output.

Quadruple 2-to- 1 line multiplexers.

- In some cases two or more multiplexers are enclosed within a single integrated circuit package. The selection and the enable inputs in multiple-unit construction are usually common to all multiplexers.



(a) Block diagram

E	S	Y
0	×	All 0's
1	0	A
1	1	B

(b) Function table

Problems

- Construct a 16-to-1 line multiplexer with two 8-to-1 line multiplexers and one 2·to-1 line multiplexer. Use block diagrams for the three multiplexers.

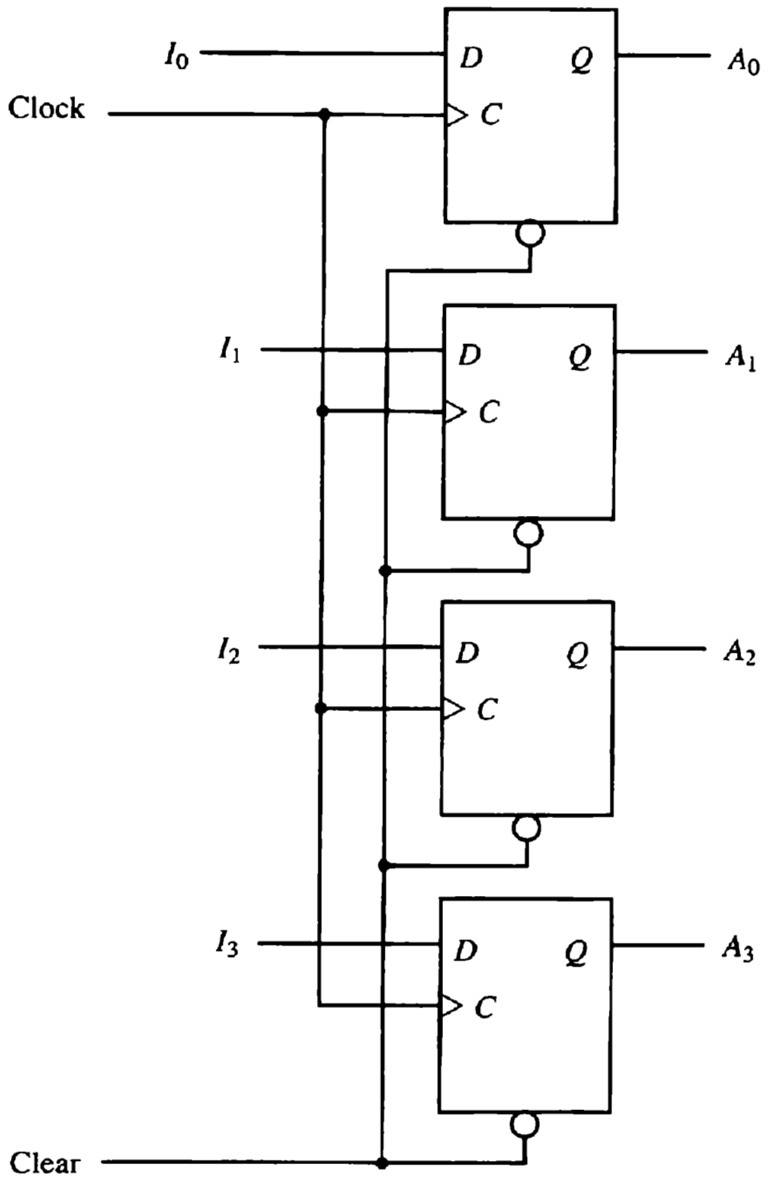
Registers

- A register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits.
- In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.
- Register consists of a group of flip-flops and gates that effect their transition. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

Registers

- The simplest register is one that consists only of flip-flops, with no external gates.
- Figure shows such a register constructed with four D flip-flops.
- The four outputs can be sampled at any time to obtain the binary information stored in the register. The clear input goes to a special terminal in each flip-flop. When this input goes to 0, all flip-flops are reset asynchronously.
- The clear input is useful for clearing the register to all O's prior to its clocked operation.

4-bit Register



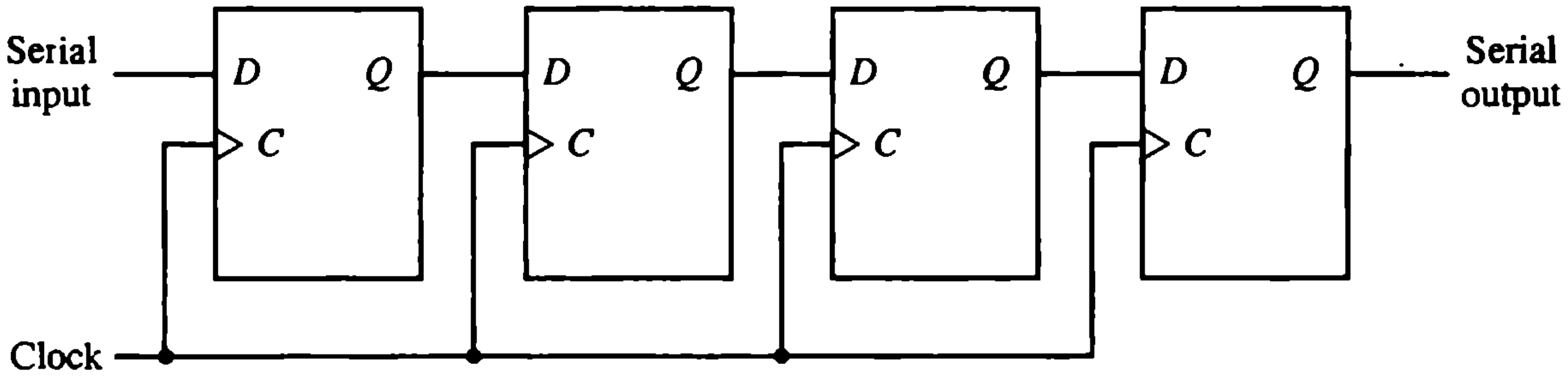
Register load

- The transfer of new information into a register is referred to as loading the register. If all the bits of the register are loaded simultaneously with a common clock pulse transition, we say that the loading is done in parallel.
- Note that the clock pulses are applied to the C inputs at all times. The load input determines whether the next pulse will accept new information or leave the information in the register intact. The transfer of information from the inputs into the register is done simultaneously with all four bits during a single pulse transition.

Shift Registers

- A register capable of shifting its binary information in one or both directions is called a shift register.
- The simplest possible shift register is one that uses only flip-flops, as shown in Fig.
- The output of a given flip-flop is connected to the D input of the flip-flop at its right. The clock is common to all flip-flops. The serial input determines what goes into the leftmost position during the shift. The serial output is taken from the output of the rightmost flip-flop.

4-bit shift register



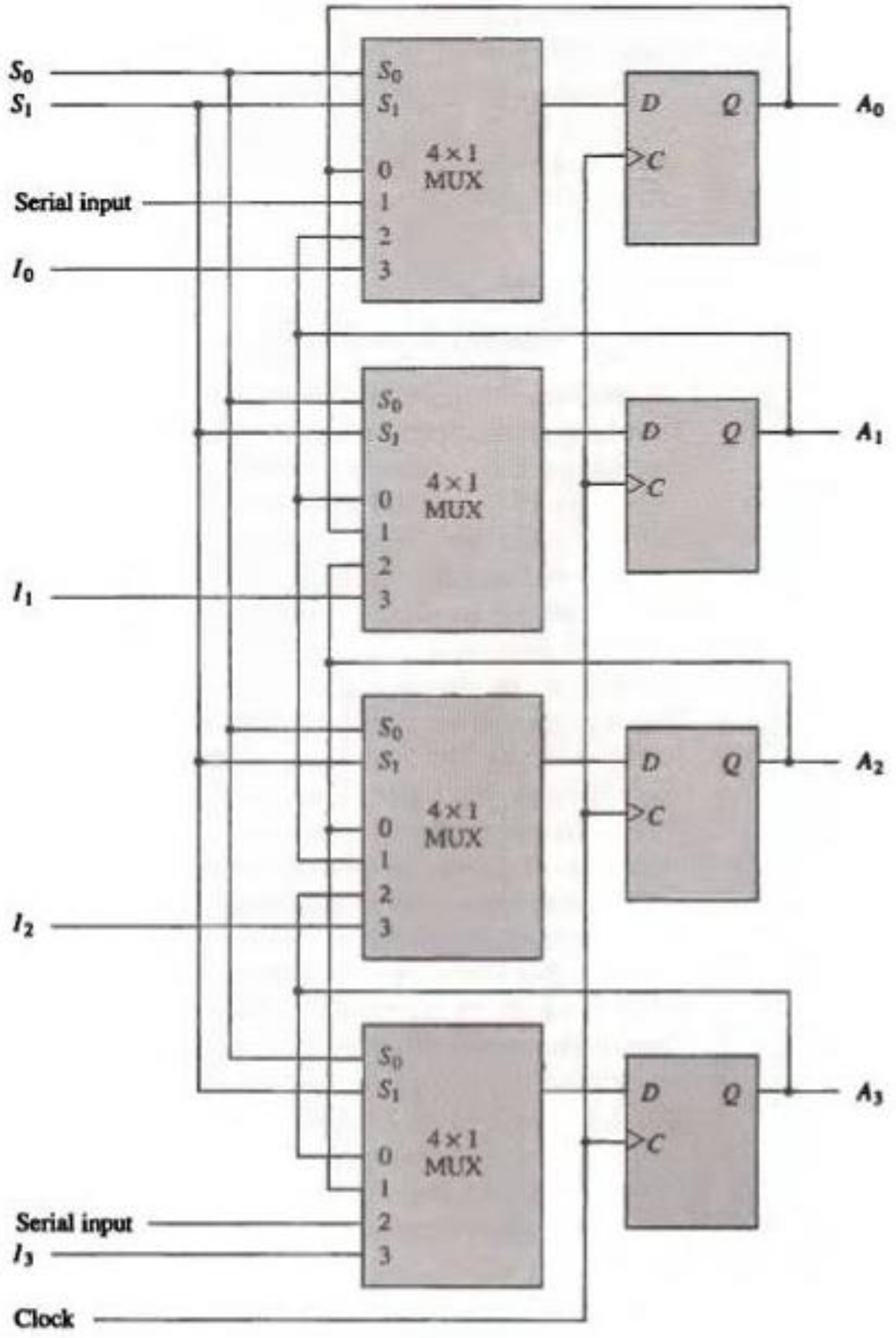
Bidirectional Shift Register with Parallel Load

- A register capable of shifting in one direction only is called a unldirectional shift register. A register that can shift in both directions is called a bidirectional shift register.

The most general shift register has all the capabilities listed below.

1. An input for clock pulses to synchronize all operations.
2. A shift-right operation and a serial input line associated with the shift-right.
3. A shift-left operation and a serial input line associated with the shift-left.
4. A parallel load operation and n input lines associated with the parallel transfer.
5. n parallel output lines.
6. A control state that leaves the information in the register unchanged even though clock pulses are applied continuously.

4-bit bidirectional shift register



Mode control		
S_1	S_0	Register operation
0	0	No change
0	1	Shift right (down)
1	0	Shift left (up)
1	1	Parallel load

- When the mode control $S_1S_0 = 00$, data input 0 of each multiplexer is selected.
- When $S_1S_0 = 00$, the terminal marked 1 in each multiplexer has a path to the D input of the corresponding flip-flop.
- When $S_1S_0 = 00$ a shift-left operation results, with the other serial input data going into flip-flop.
- When $S_1S_0 = 11$, the binary information from each input 10 through I, is transferred into the corresponding flip-flop.