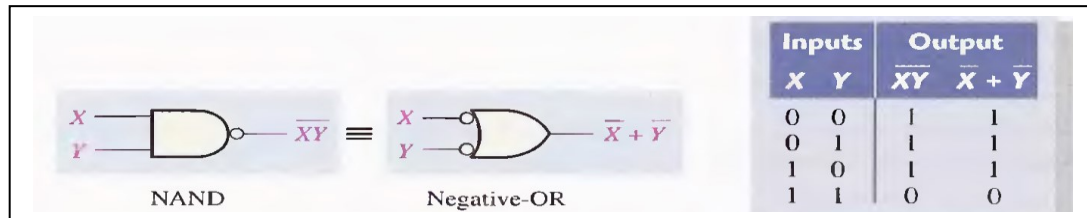


DEMORGAN'S THEOREMS نظريه ديموركان**One of DeMorgan's الأولى** theorems stated as follows:

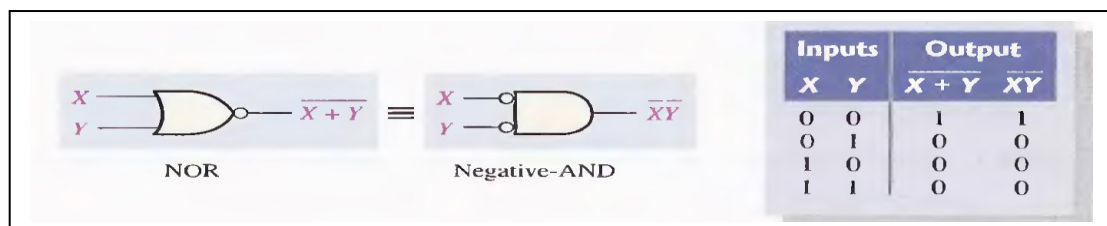
The complement of a product of variables is equal to the sum of the complements of the variables.

$$\overline{XY} = \overline{X} + \overline{Y}$$

**DeMorgan's second الثانية** theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

$$\overline{X + Y} = \overline{X} \overline{Y}$$



Example:

Apply DeMorgan's theorems to each of the following expressions:

(a) $\overline{(A + B + C)D}$ (b) $\overline{ABC + DEF}$ (c) $\overline{AB + CD + EF}$

Solution (a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

$$\overline{A + B + C} + \overline{D} = \overline{A} \overline{B} \overline{C} + \overline{D}$$

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X} \overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms \overline{ABC} and \overline{DEF} .

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $\overline{AB} = X$, $\overline{CD} = Y$, and $EF = Z$. The expression $\overline{AB + CD + EF}$ is of the form $\overline{X + Y + Z} = \overline{X} \overline{Y} \overline{Z}$ and can be rewritten as

$$\overline{AB + CD + EF} = (\overline{AB})(\overline{CD})(\overline{EF})$$

Next, apply DeMorgan's theorem to each of the terms \overline{AB} , \overline{CD} , and \overline{EF} .

$$(\overline{AB})(\overline{CD})(\overline{EF}) = (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{E} + \overline{F})$$

SIMPLIFICATION USING BOOLEAN ALGEBRA تبسيط معادلات الجبر البولي

A simplified Boolean expression uses the fewest gates possible to implement a given expression. Simplification means fewer gates for the same function

Example:

Using Boolean algebra techniques, simplify this expression:

$$AB + A(B + C) + B(B + C)$$

Solution The following is not necessarily the only approach.

Step 1: Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2: Apply rule 7 ($BB = B$) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3: Apply rule 5 ($AB + AB = AB$) to the first two terms.

$$AB + AC + B + BC$$

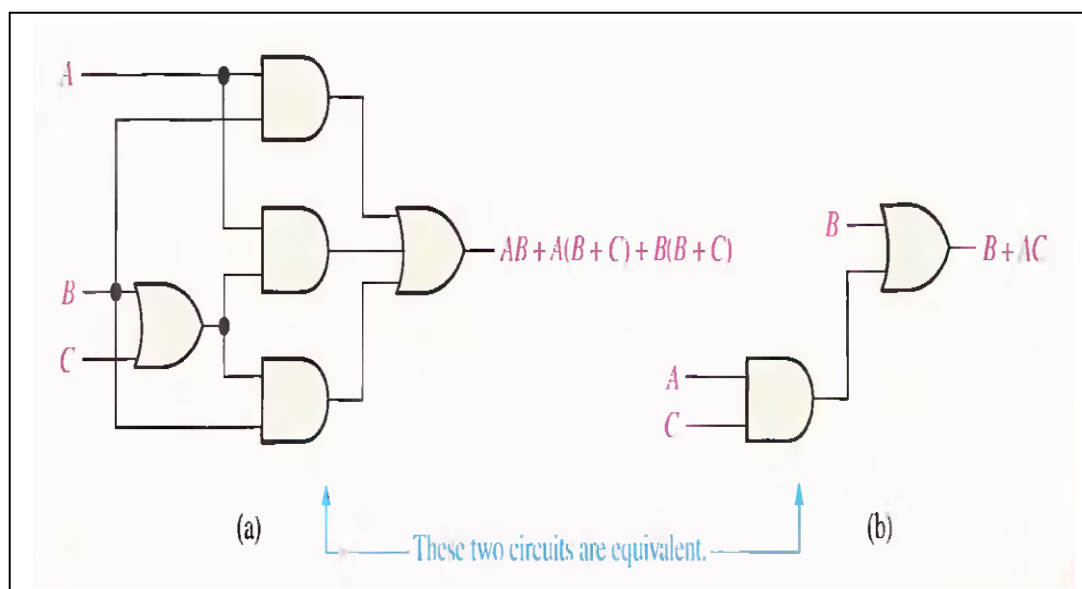
Step 4: Apply rule 10 ($B + BC = B$) to the last two terms.

$$AB + AC + B$$

Step 5: Apply rule 10 ($AB + B = B$) to the first and third terms.

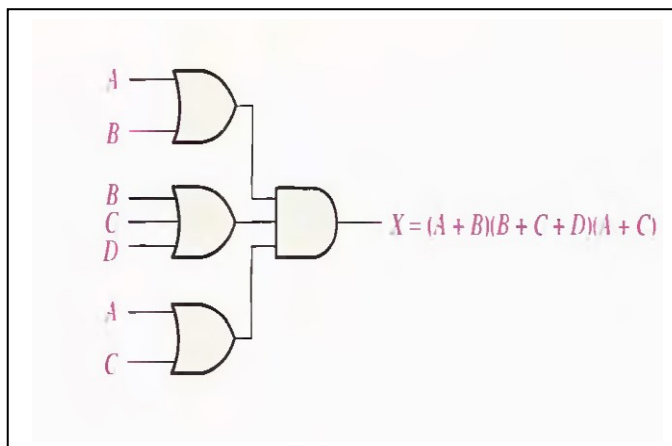
$$B + AC$$

At this point the expression is simplified as much as possible. Once you gain experience in applying Boolean algebra, you can often combine many individual steps.



Product-of-Sums (POS) Form

When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS). Implementing a POS expression simply requires ANDing the outputs of two or more OR gates.



A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to 0.

Example

Determine the binary values of the variables for which the following standard POS expression is equal to 0:

$$(A + B + C + D)(A + \bar{B} + \bar{C} + D)(\bar{A} + \bar{B} + \bar{C} + \bar{D})$$

Solution The term $A + B + C + D$ is equal to 0 when $A = 0$, $B = 0$, $C = 0$, and $D = 0$.

$$A + B + C + D = 0 + 0 + 0 + 0 = 0$$

The term $A + \bar{B} + \bar{C} + D$ is equal to 0 when $A = 0$, $B = 1$, $C = 1$, and $D = 0$.

$$A + \bar{B} + \bar{C} + D = 0 + \bar{1} + \bar{1} + 0 = 0 + 0 + 0 + 0 = 0$$

The term $\bar{A} + \bar{B} + \bar{C} + \bar{D}$ is equal to 0 when $A = 1$, $B = 1$, $C = 1$, and $D = 1$.

$$\bar{A} + \bar{B} + \bar{C} + \bar{D} = \bar{1} + \bar{1} + \bar{1} + \bar{1} = 0 + 0 + 0 + 0 = 0$$

The POS expression equals 0 when any of the three sum terms equals 0.

BOOLEAN EXPRESSIONS AND TRUTH TABLES

تحويل جمع المضاريب الى جدول حقيقه Converting SOP Expressions to Truth Table Format

The first step in constructing a truth table is to list all possible combinations of binary values of the variables in the expression. Next, convert the SOP expression to standard form if it is not already. Finally, place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining binary values.

Example

Develop a truth table for the standard SOP expression $\overline{A}\overline{B}C + \overline{A}B\overline{C} + ABC$.

Solution There are three variables in the domain, so there are eight possible combinations of binary values of the variables as listed in the left three columns of Table 4–6. The binary values that make the product terms in the expressions equal to 1 are $\overline{A}\overline{B}C$: 001; $\overline{A}B\overline{C}$: 100; and ABC : 111. For each of these binary values, place a 1 in the output column as shown in the table. For each of the remaining binary combinations, place a 0 in the output column.

INPUTS			OUTPUT	PRODUCT TERM
A	B	C	X	
0	0	0	0	
0	0	1	1	$\overline{A}\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$\overline{A}B\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

تحويل ضرب المجاميع الى جدول حقيقه Converting POS Expressions to Truth Table Format

To construct a truth table from a POS expression, list all the possible combinations of binary values of the variables just as was done for the SOP expression. Next, convert the POS expression to standard form if it is not already. Finally, place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values.

Determine the truth table for the following standard POS expression:

$$(A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$$

Solution There are three variables in the domain and the eight possible binary values are listed in the left three columns of Table 4-7. The binary values that make the sum terms in the expression equal to 0 are $A + B + C$: 000; $A + \bar{B} + C$: 010; $A + \bar{B} + \bar{C}$: 011; $\bar{A} + B + \bar{C}$: 101; and $\bar{A} + \bar{B} + C$: 110. For each of these binary values, place a 0 in the output column as shown in the table. For each of the remaining binary combinations, place a 1 in the output column.

INPUTS			OUTPUT	SUM TERM
A	B	C	X	
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \bar{B} + C)$
0	1	1	0	$(A + \bar{B} + \bar{C})$
1	0	0	1	
1	0	1	0	$(\bar{A} + B + \bar{C})$
1	1	0	0	$(\bar{A} + \bar{B} + C)$
1	1	1	1	

THE KARNAUGH MAP **كارنوف ماب**

The purpose of a Karnaugh map is to simplify a Boolean expression.

The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.

3-Variable Karnaugh Map

		C	
		0	1
AB	00	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
	01	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
	11	$\bar{A}B\bar{C}$	$\bar{A}BC$
	10	$\bar{A}B\bar{C}$	$\bar{A}BC$

4-Variable Karnaugh Map

		CD			
		00	01	11	10
AB	00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}CD$
	01	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}CD$
	11	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BC\bar{D}$	$\bar{A}BCD$
	10	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BC\bar{D}$	$\bar{A}BCD$

Karnaugh Map Simplification of SOP Expressions **كارنوف لتبسيط جمع المضاريب**
Determining the Minimum SOP Expression from the Map

1. Group the cells that have 1 s. Each group of cells containing 1 s creates one product term composed of all variables that occur in only one form (either un complemented or complemented) within the group. Variables that occur both un complemented and complemented within the group are eliminated. These called contradictory variables.

2. Determine the minimum product term for each group.

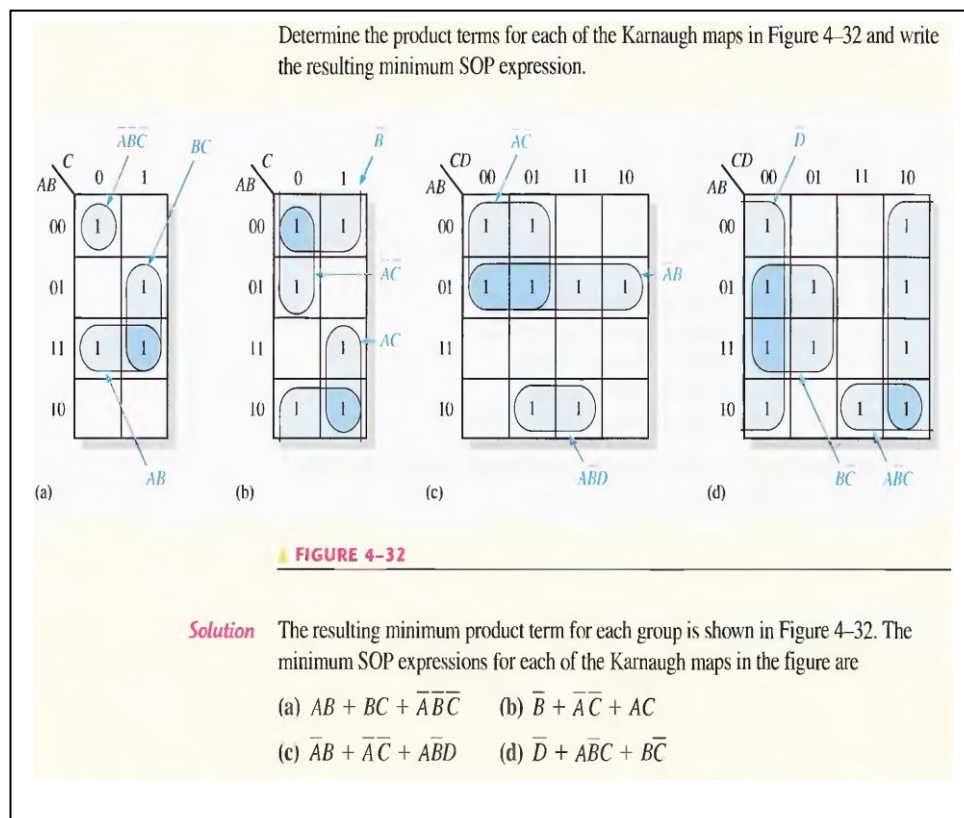
A. For a 3-variable map:

- (1) 1-cell group yields a 3-variable product term
- (2) 2-cell group yields a 2-variable product term
- (3) 4-cell group yields a 1-variable term
- (4) 8-cell group yields a value of 1 for the expression

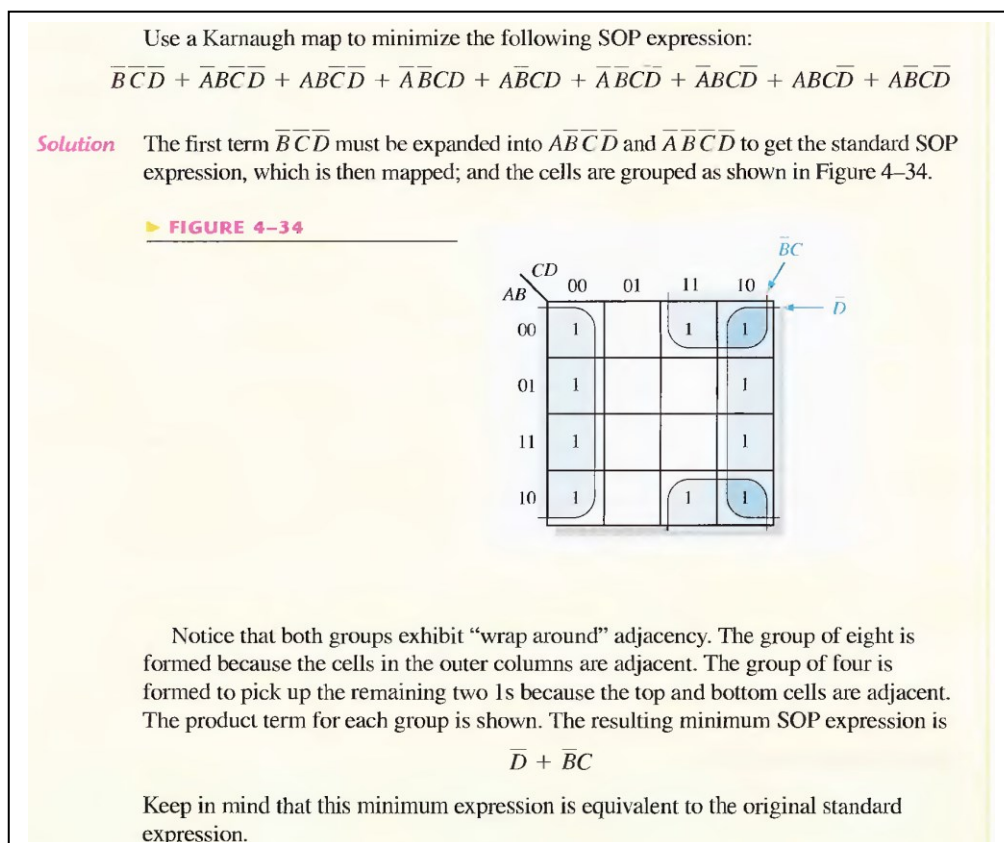
B. For a 4-variable map:

- (1) 1-cell group yields a 4-variable product term
 - (2) 2-cell group yields a 3-variable product term
 - (3) 4-cell group yields a 2-variable product term
 - (4) 8-cell group yields a 1-variable term
 - (5) 16-cell group yields a value of 1 for the expression
3. When all the minimum product terms derived from the Karnaugh map, they summed to form the minimum SOP expression.

Example:



Example:



KARNAUGH MAP POS MINIMIZATION كارنوف لتبسيط ضرب المجاميع**Karnaugh Map Simplification of POS Expressions**

The process for minimizing a POS expression is the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms.

Example

Use a Karnaugh map to minimize the following POS expression:

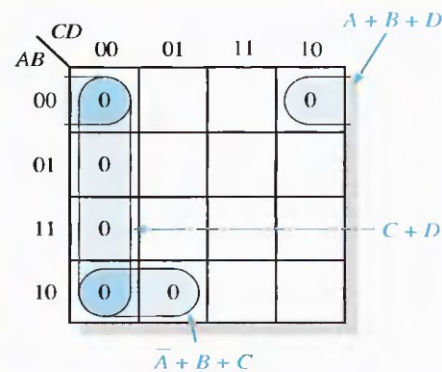
$$(B + C + D)(A + B + \bar{C} + D)(\bar{A} + B + C + \bar{D})(A + \bar{B} + C + D)(\bar{A} + \bar{B} + C + D)$$

Solution The first term must be expanded into $\bar{A} + B + C + D$ and $A + B + C + D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in Figure 4-40. The sum term for each group is shown and the resulting minimum POS expression is

$$(C + D)(A + B + D)(\bar{A} + B + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

► **FIGURE 4-40**



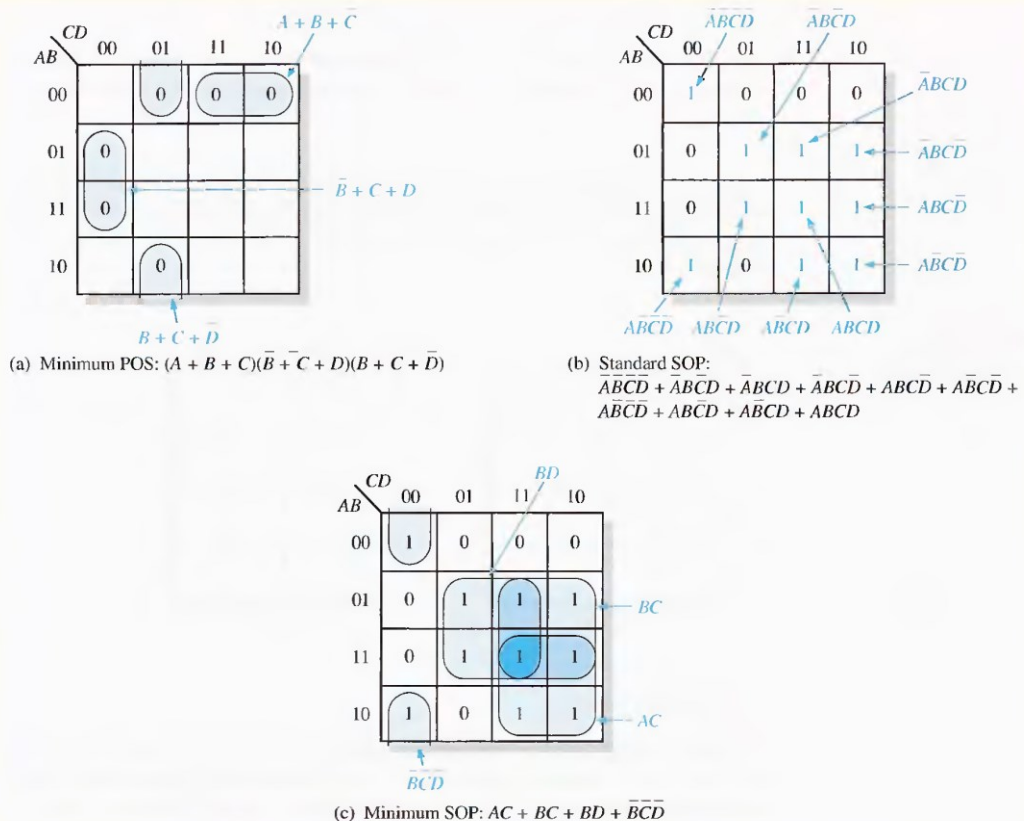
Converting Between POS and SOP Using the Karnaugh Map

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

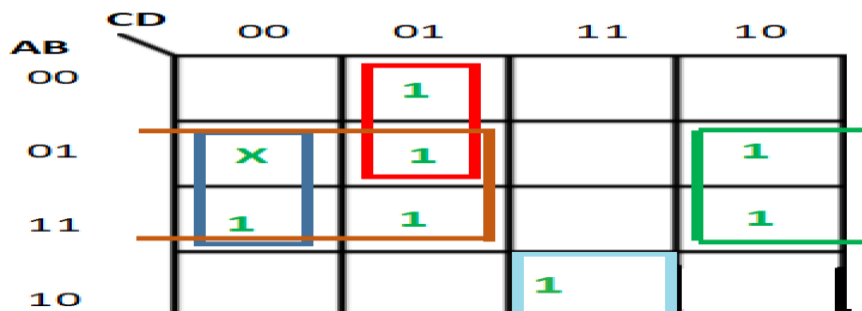
$$(\bar{A} + \bar{B} + C + D)(A + \bar{B} + C + D)(A + B + C + \bar{D})$$

$$(A + B + \bar{C} + \bar{D})(\bar{A} + B + C + \bar{D})(A + B + \bar{C} + D)$$

Solution The 0s for the standard POS expression are mapped and grouped to obtain the minimum POS expression in Figure 4-41(a). In Figure 4-41(b), 1s are added to the cells that do not contain 0s. From each cell containing a 1, a standard product term is obtained as indicated. These product terms form the standard SOP expression. In Figure 4-41(c), the 1s are grouped and a minimum SOP expression is obtained.



Don't care Karnaugh condition

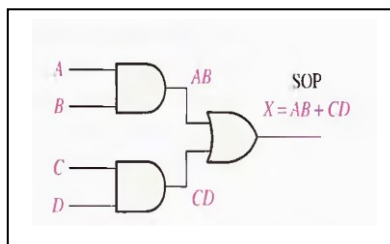


Chapter 4

COMBINATIONAL LOGIC CIRCUITS

- **AND-OR Logic**

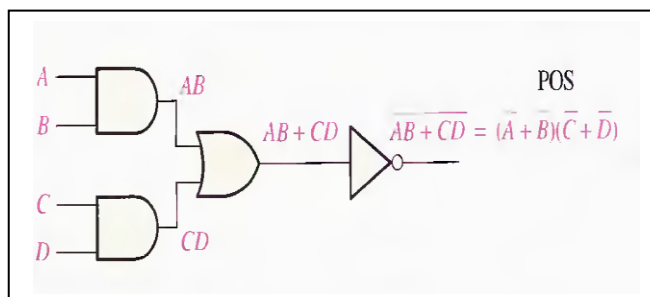
AND-OR circuit consisting of two (2-input) AND gates and one 2-input OR gate;



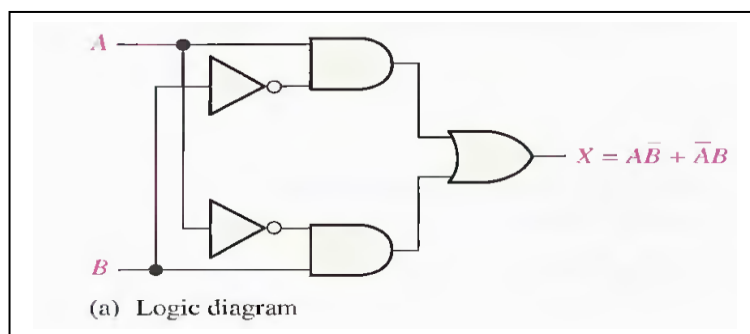
- **AND-OR-Invert Logic**

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR-Invert circuit.

$$X = (\bar{A} + \bar{B})(\bar{C} + \bar{D}) = (\overline{AB})(\overline{CD}) = \overline{(AB)(CD)} = \overline{AB + CD} = \overline{AB} + \overline{CD}$$

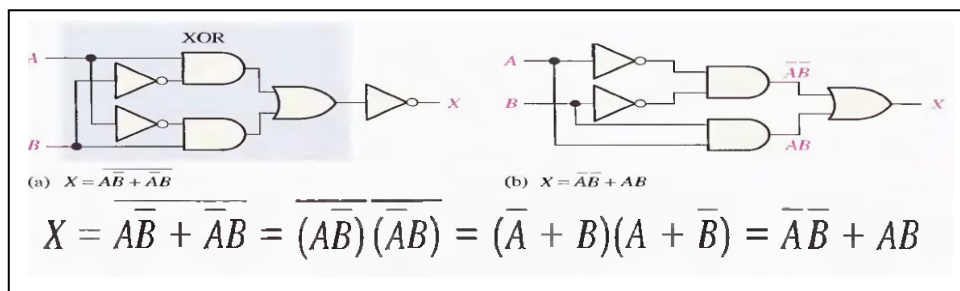


- **Exclusive-OR logic** we can use AND-OR to represent X-OR



$$X = A \oplus B$$

- **Exclusive-NOR Logic**



Example

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

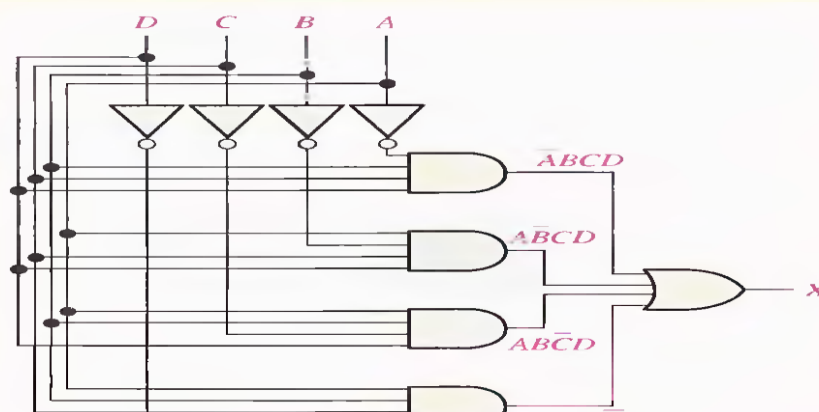
Solution Out of sixteen possible combinations of four variables, the combinations in which there are exactly three 1s are listed in Table 5-5, along with the corresponding product term for each.

A	B	C	D	PRODUCT TERM
0	1	1	1	$\bar{A}BCD$
1	0	1	1	$A\bar{B}CD$
1	1	0	1	$AB\bar{C}D$
1	1	1	0	$ABCD\bar{D}$

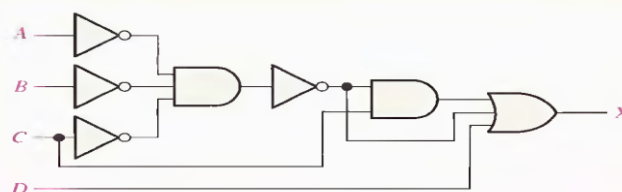
The product terms are ORed to get the following expression:

$$X = \bar{A}BCD + A\bar{B}CD + AB\bar{C}D + ABC\bar{D}$$

This expression is implemented in Figure 5-11 with AND-OR logic.



Reduce the combinational logic circuit in Figure 5-12 to a minimum form.



Solution The expression for the output of the circuit is

$$X = (\bar{A}\bar{B}\bar{C})C + \bar{A}\bar{B}C + A\bar{B}C$$

Applying DeMorgan's theorem and Boolean algebra,

$$\begin{aligned} X &= (\bar{A} + \bar{B} + \bar{C})C + \bar{A}\bar{B}C + A\bar{B}C \\ &= AC + BC + CC + A\bar{B}C + A\bar{B}C \\ &= AC + BC + C + A\bar{B}C + A\bar{B}C \\ &= C(A + B + 1) + A\bar{B}C + A\bar{B}C \\ X &= A + B + C + D \end{aligned}$$

The simplified circuit is a 4-input OR gate as shown in Figure 5-13.

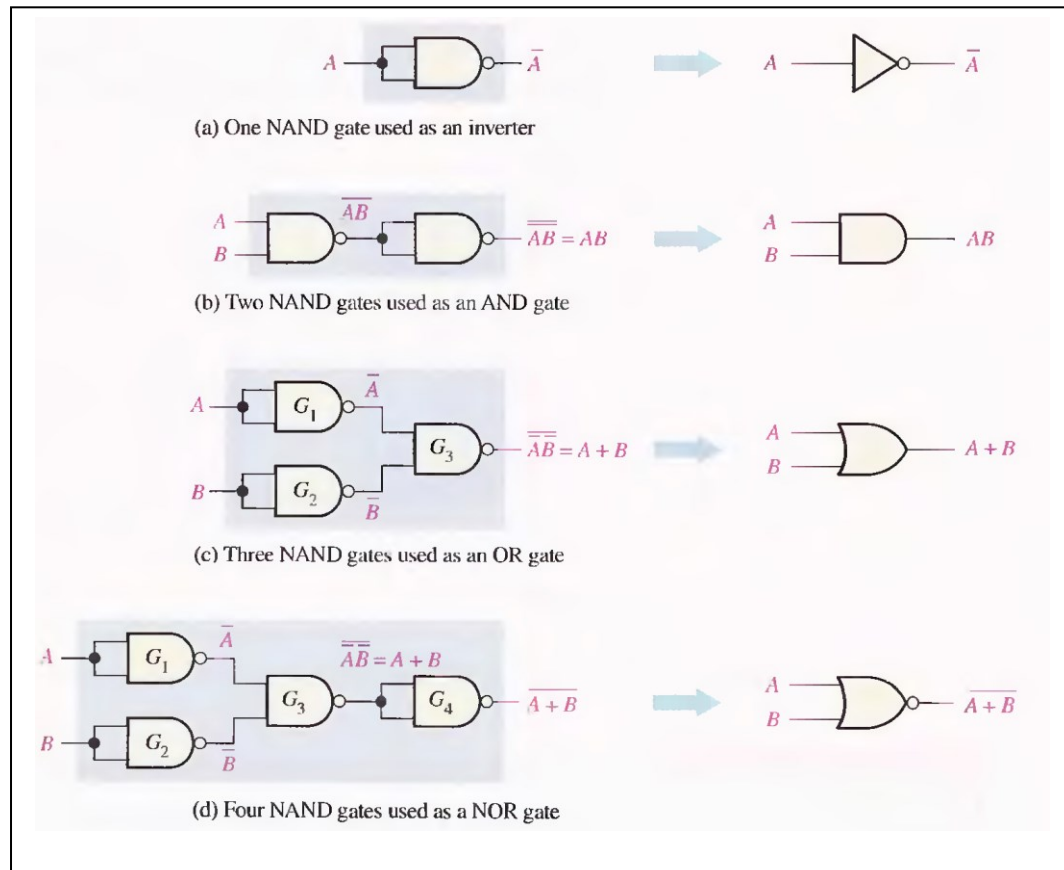
FIGURE 5-13



THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

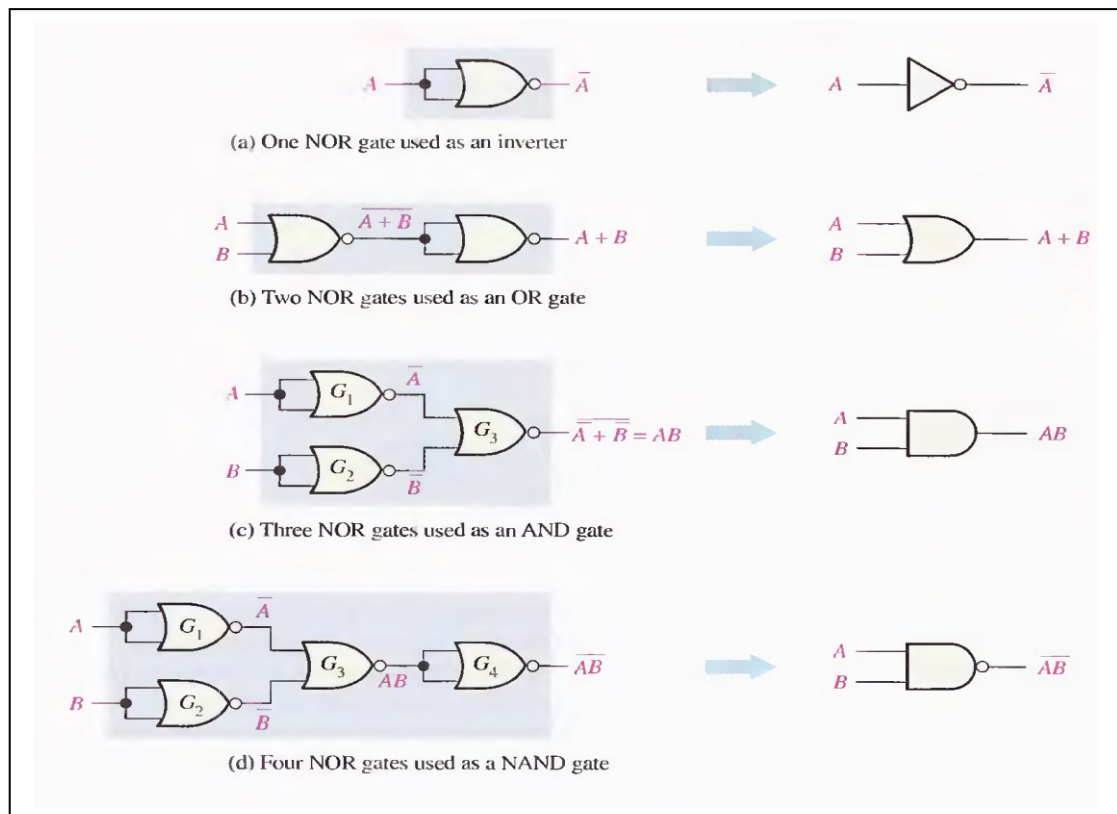
- The NAND Gate as a Universal Logic Element**

The NAND gate is a universal gate because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input,



The NOR Gate as a Universal Logic Element

Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR and NAND functions. As shown below:

**COMBINATIONAL LOGIC USING NAND AND NOR GATES**

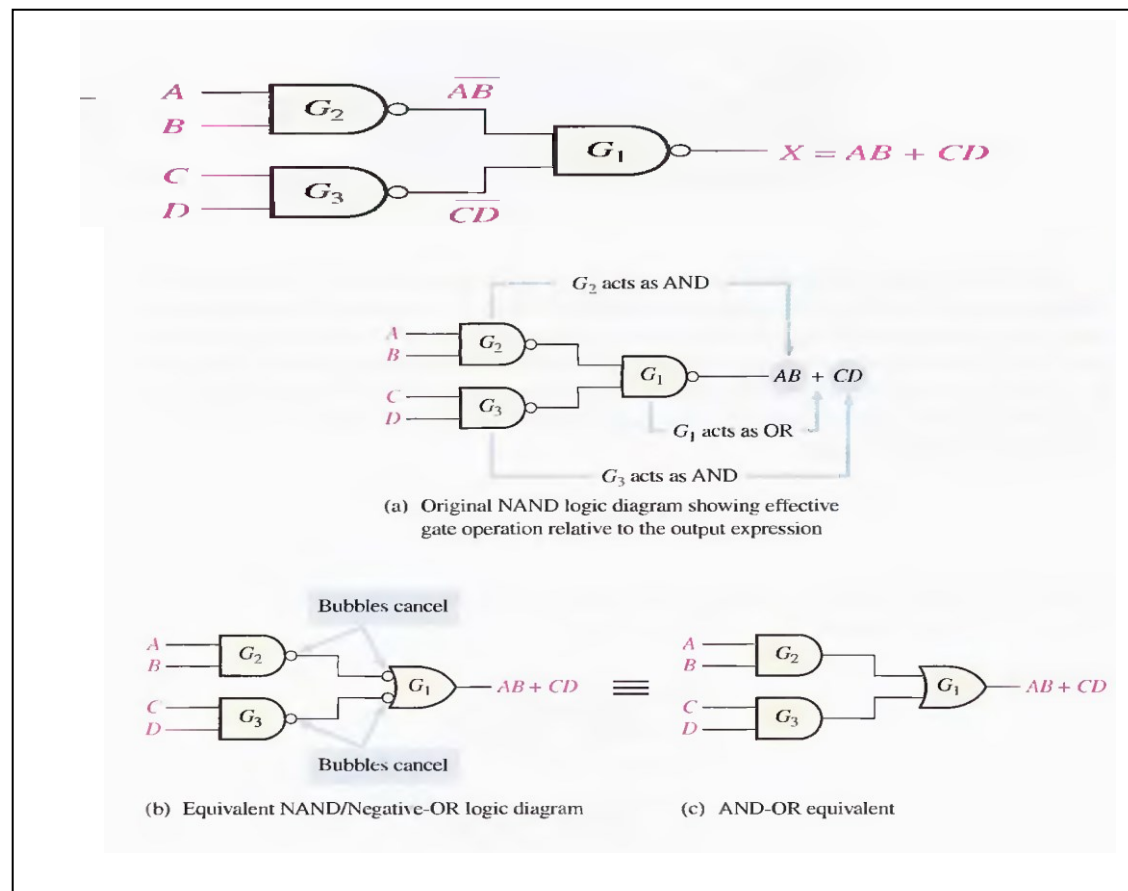
- NAND Logic**

NAND gate can function as either a NAND or a negative-OR because, by DeMorgan's theorem,

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

NAND $\xrightarrow{\quad}$ $\xrightarrow{\quad}$ negative-OR

Example:

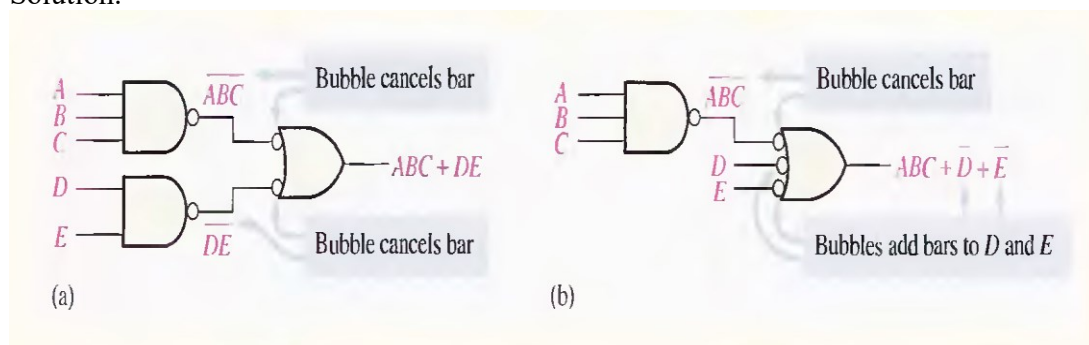


Example:

Implement each expression with NAND logic using appropriate dual symbols:

(a) $ABC + DE$ (b) $ABC + \bar{D} + \bar{E}$

Solution:

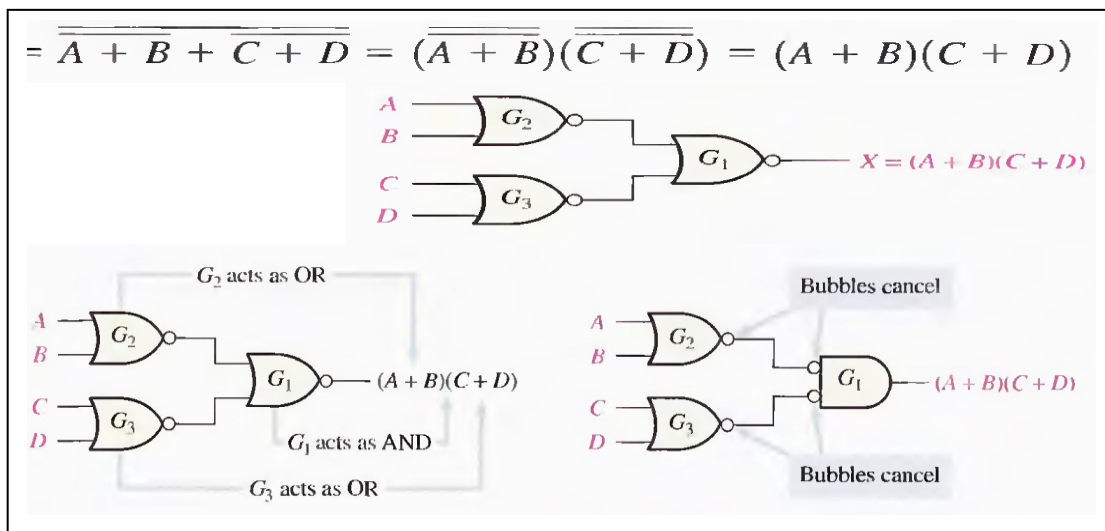


• NOR Logic

A NOR gate can function as either a NOR or a negative-AND, as shown by DeMorgan's theorem.

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

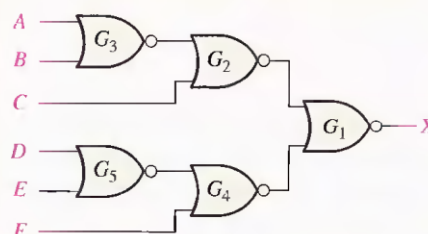
NOR ↑ ↑ negative-AND



Example

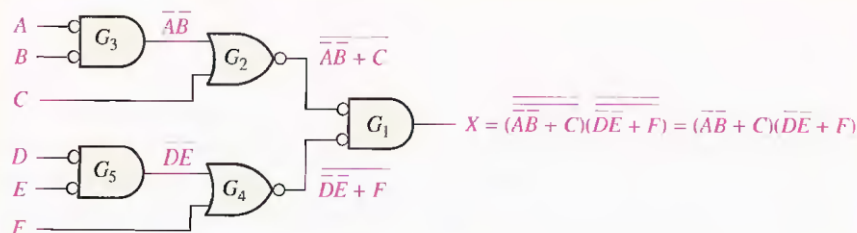
Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in Figure 5–27.

► **FIGURE 5–27**



Solution Redraw the logic diagram with the equivalent negative-AND symbols as shown in Figure 5–28. Writing the expression for X directly from the indicated operation of each gate,

$$X = (\overline{AB} + C)(\overline{DE} + F)$$

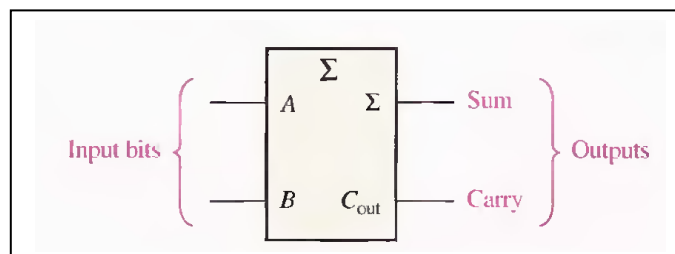


CHAPYER FIVE BASIC ADDERS

The Half-Adder

The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a **sum** bit and a **carry** bit.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10



Truth table for half adder

From the operation of the half-adder as stated in Table

The carry output expression

Notice that the output Carry

(C_{out}) is a 1 only when both A and B are 1 s: therefore, (C_{out}) can be expressed as the AND of the input variables.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

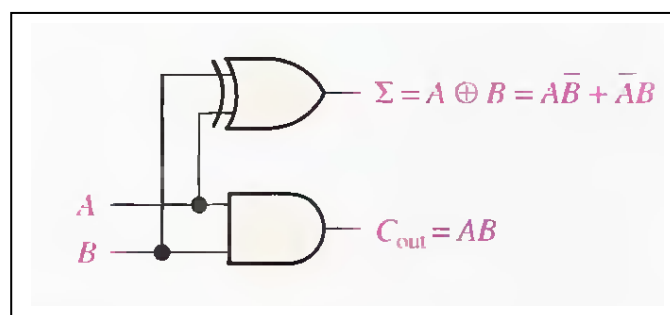
Σ = sum
 C_{out} = output carry
 A and B = input variables (operands)

$$C_{out} = AB$$

Now observe that the sum output (Σ) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

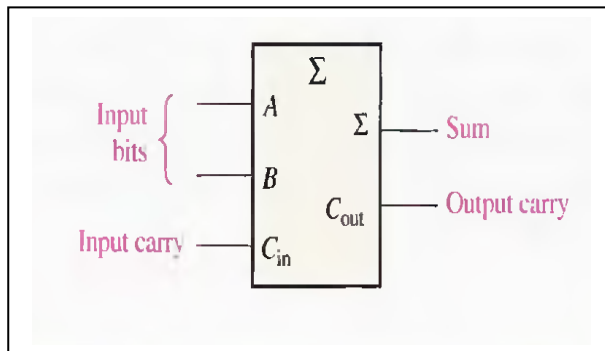
$$\Sigma = A \oplus B$$

Half-adder logic diagram.



The Full-Adder

The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.



A	B	C _{in}	C _{out}	Σ
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

C_{in} = input carry, sometimes designated as CI

C_{out} = output carry, sometimes designated as CO

Σ = sum

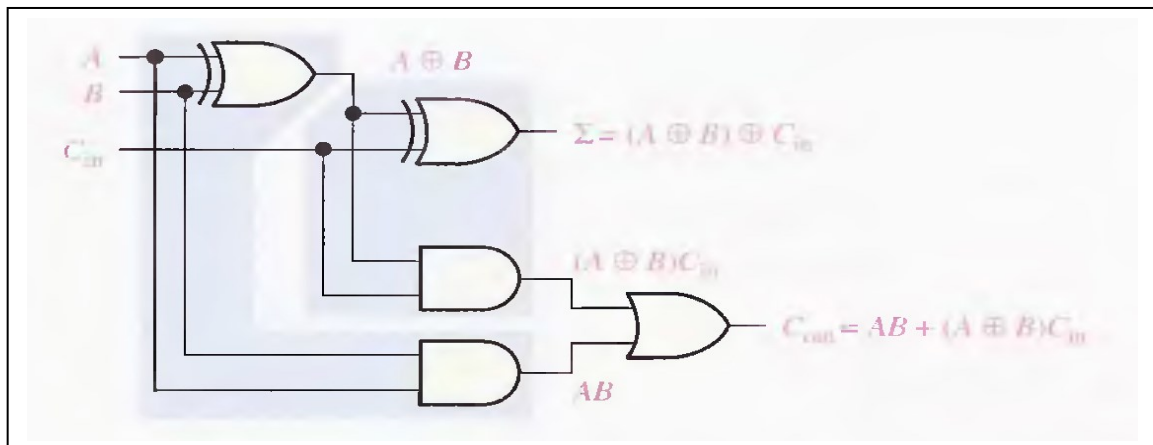
A and B = input variables (operands)

Full-Adder Logic

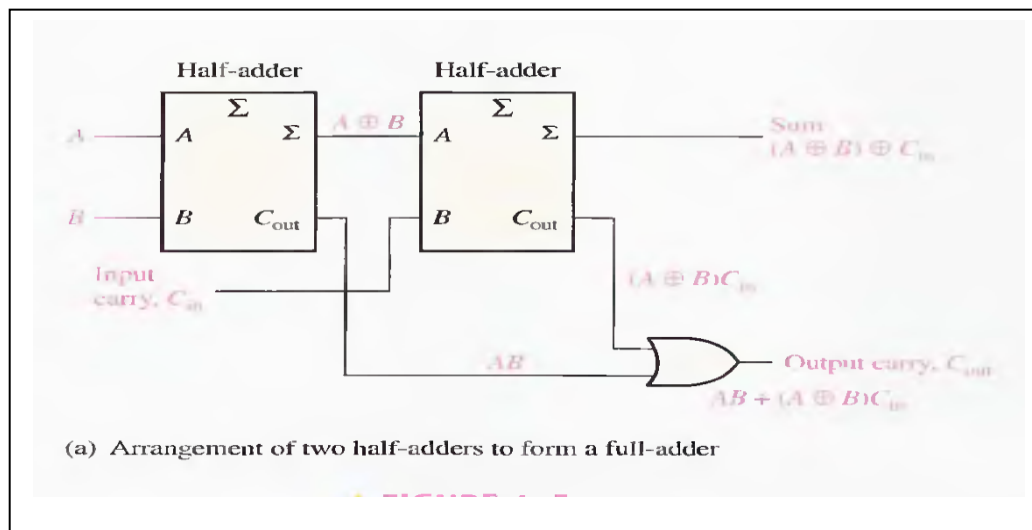
Full-Adder Logic The full-adder must add the two input bits and the input carry from the half-adder.

$$\Sigma = (A \oplus B) \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$



There are two half-adders, connected as shown in the block diagram with their output carries ORed.



Example:

For each of the three full-adders in Figure 6–6, determine the outputs for the inputs shown.

(a)

(b)

(c)

▲ FIGURE 6–6

Solution

(a) The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.
 $1 + 0 + 0 = 1$ with no carry
 Therefore, $\Sigma = 1$ and $C_{out} = 0$.

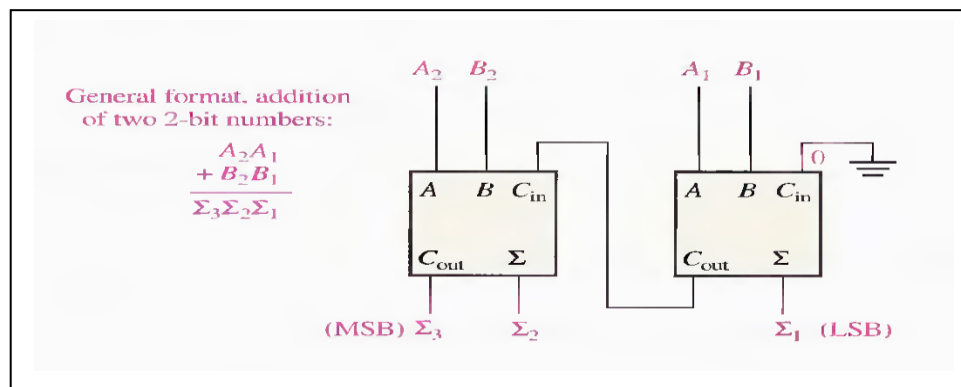
(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.
 $1 + 1 + 0 = 0$ with a carry of 1
 Therefore, $\Sigma = 0$ and $C_{out} = 1$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.
 $1 + 0 + 1 = 0$ with a carry of 1
 Therefore, $\Sigma = 0$ and $C_{out} = 1$.

PARALLEL BINARY ADDERS

Two or more full-adders have connected to form parallel binary adders.

To add two binary numbers, a full-adder is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure below for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.



In Figure above the least significant bits (LSB) of the two numbers are represented by A_1 and B_1 . The next higher-order bits are represented by A_2 and B_2 . The three sum bits are Σ_1, Σ_2 and Σ_3 . Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum, Σ_3 .

Example

Determine the sum generated by the 3-bit parallel adder in Figure 6–8 and show the intermediate carries when the binary numbers 101 and 011 are being added.

► **FIGURE 6–8**

Solution The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in blue in Figure 6–8.

Four-Bit Parallel Adders

A group of four bits is called a nibble. A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure

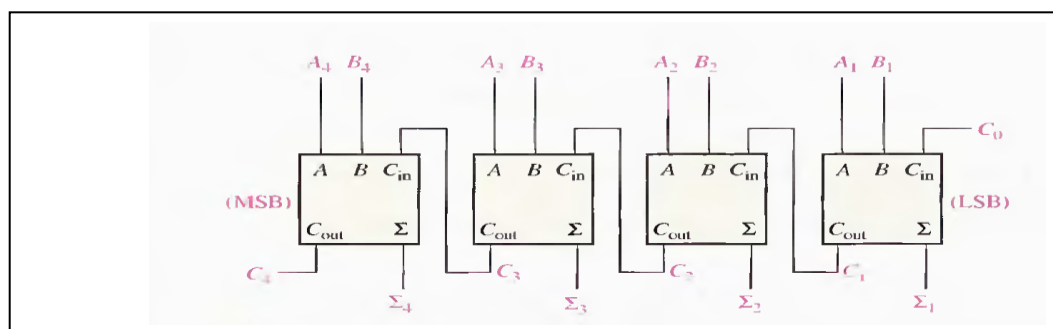
**Truth Table for a 4-Bit Parallel Adder**

Table 6-3 is the truth table for a 4-bit adder. On some data sheets, truth tables may be called function tables or functional truth tables. The subscript n represents the adder bit and can be 1, 2, 3, or 4 for the 4-bit adder. C_{n-1} is the carry from the previous adder. Carries C_1 , C_2 and C_3 are generated internally. C_n is an external carry input and C_4 is an output.

C_{n-1}	A_n	B_n	Σ_n	C_n
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

Example:

Use the 4-bit parallel adder truth table (Table 6-3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

$$A_4A_3A_2A_1 = 1100 \quad \text{and} \quad B_4B_3B_2B_1 = 1100$$

Solution For $n = 1$: $A_1 = 0$, $B_1 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_1 = 0 \quad \text{and} \quad C_1 = 0$$

For $n = 2$: $A_2 = 0$, $B_2 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_2 = 0 \quad \text{and} \quad C_2 = 0$$

For $n = 3$: $A_3 = 1$, $B_3 = 1$, and $C_{n-1} = 0$. From the 4th row of the table,

$$\Sigma_3 = 0 \quad \text{and} \quad C_3 = 1$$

For $n = 4$: $A_4 = 1$, $B_4 = 1$, and $C_{n-1} = 1$. From the last row of the table,

$$\Sigma_4 = 1 \quad \text{and} \quad C_4 = 1$$

C_4 becomes the output carry; the sum of 1100 and 1100 is 11000.

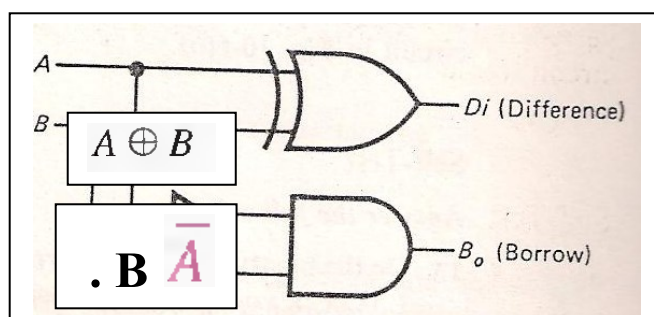
Half Subtractor

A subtracted from B the output is D_i (Difference), and if B greater than A we need to borrow and labeled (B_o)

The truth table and block diagram of half subtractor as shown below

TRUTH TABLE			
INPUTS		OUTPUTS	
A	B	D_i	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
$A - B$		Difference	Borrow out

Logic Diagram half subtractor.



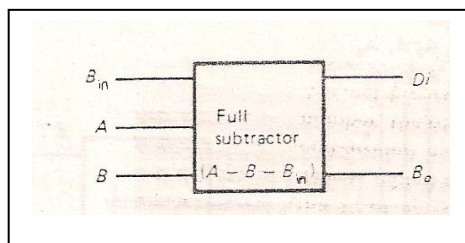
The Boolean expression for half subtractor

$$D_i =$$

$$B_o =$$

Full subtractor

Full subtractor we have Barrow in (B_{in}) the truth table as shown below

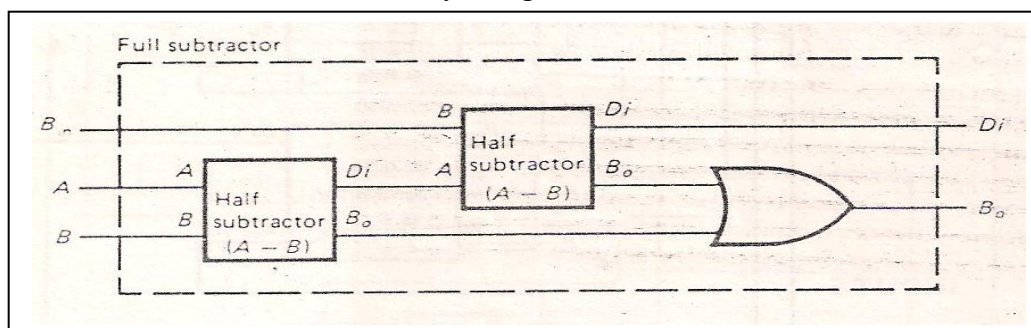


Block symbol

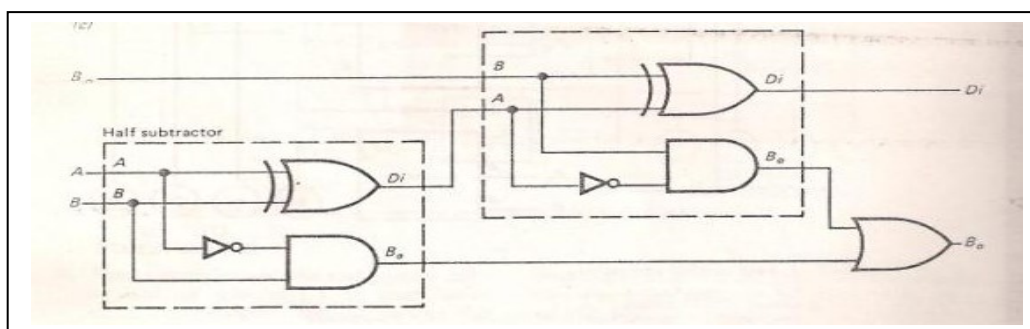
TRUTH TABLE

INPUTS			OUTPUTS	
A	B	B_{in}	D_i	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
$A - B - B_{in}$			Difference	Borrow out

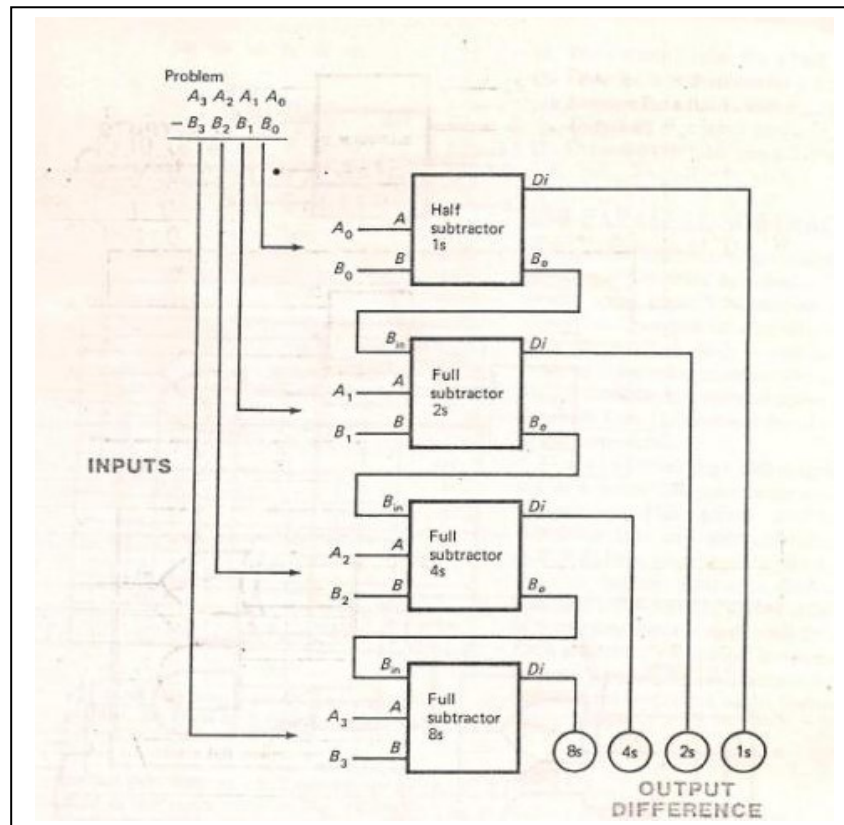
We can construct full subtractor by using half subtractor as shown below



Logic diagram as shown

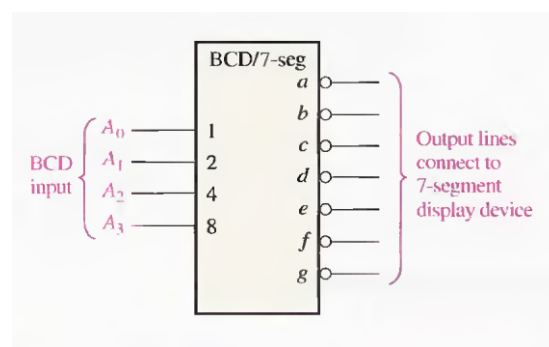


4 bit parallel subtractor: the form bellow 4 bit parallel subtractor that can be subtract binary number $B_3B_2B_1B_0$ from binary number $A_3A_2A_1A_0$. Notice that the top subtractor (half subtractor) subtract the LSBs(1s place). The B_o of the 1s subtractor is tied to next subtractor as B_{in}



The BCD-to-7-Segment Decoder

The BCD-to-7-segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in Figure 6-34.

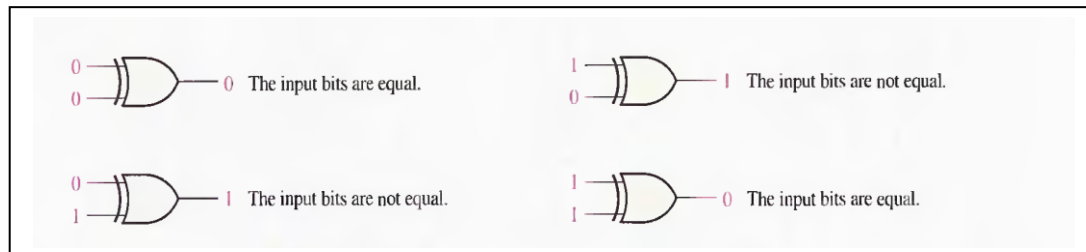


COMPARATORS

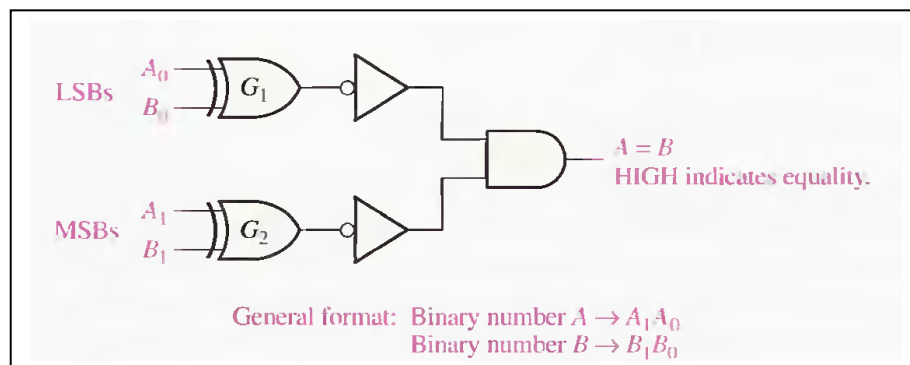
The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities.

Equality

Exclusive-OR gate can be used as a basic comparator because its output is a 1 if the two input bits are not equal and a 0 if the input bits are equal.



In order to compare binary numbers containing two bits each, an additional exclusive OR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G_1 . In addition, the two most significant bits (MSBs) are compared by gate G_2 , as shown in Figure below. If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-OR gate is a 0. If the corresponding sets of bits are not equal, a 1 occurs on that exclusive-OR gate output. In order to produce a single output indicating an equality or inequality of two numbers, two inverters and an AND gate can be used,



Example:

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6-21, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 (b) 11 and 10

(a) $A_0 = 0$, $B_0 = 0$ → 1
 $A_1 = 1$, $B_1 = 1$ → 1
 Output: 1 → equal

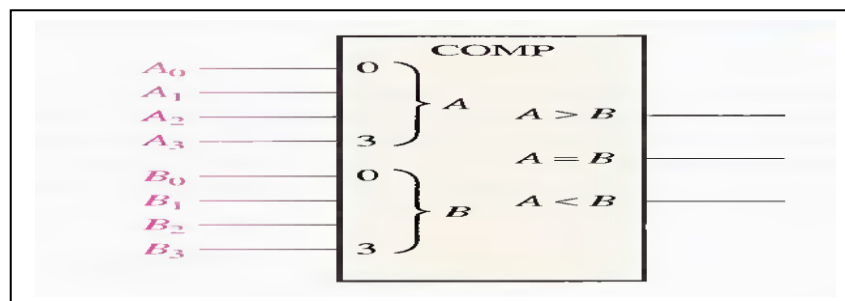
(b) $A_0 = 1$, $B_0 = 0$ → 0
 $A_1 = 1$, $B_1 = 1$ → 1
 Output: 0 → not equal

▲ FIGURE 6-21

Solution (a) The output is 1 for inputs 10 and 10, as shown in Figure 6-21(a).
 (b) The output is 0 for inputs 11 and 10, as shown in Figure 6-21(b).

Inequality

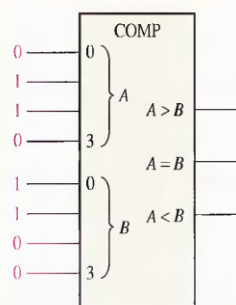
In addition to the equality output, many IC comparators provide additional outputs that indicate which of the two binary numbers being compared is the larger. That is, there is an output that indicates when number A is greater than number B ($A > B$) and an output that indicates when number A is less than number B ($A < B$), as shown in the logic symbol for a 4-bit comparator.



Example:

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–23.

► **FIGURE 6–23**



Solution The number on the A inputs is 0110 and the number on the B inputs is 0011. The $A > B$ output is HIGH and the other outputs are LOW.

CODE CONVERTERS

BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number have represented by a binary number.
2. All of the binary representations of the weights of bits that are 1 s in the BCD number have added.
3. The result of this addition is the binary equivalent of the BCD number.

The binary numbers representing the weights of the BCD bits have summed to produce the total binary number.

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

Example:

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

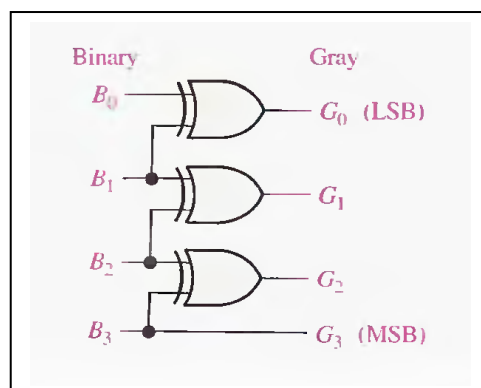
Solution Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

80	40	20	10	8	4	2	1		
0	0	1	0	0	1	1	1		
								0000001	1
								0000010	2
								0000100	4
								+ 0010100	20
								<u>0011011</u>	Binary number for decimal 27

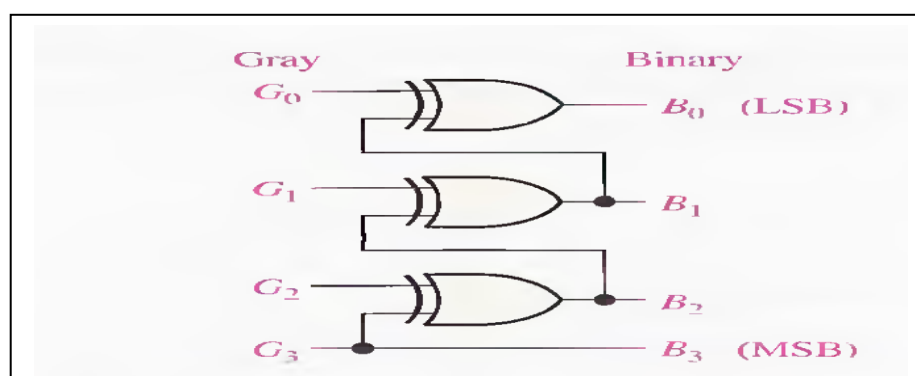
80	40	20	10	8	4	2	1		
1	0	0	1	1	0	0	0		
								0001000	8
								0001010	10
								+ 1010000	80
								<u>1100010</u>	Binary number for decimal 98

Binary-to-Gray and Gray-to-Binary Conversion

Figure bellow shows a 4-bit binary-to-Gray code converter.



In addition, Figure bellow illustrates a 4-bit Gray-to-binary converter.



Example:

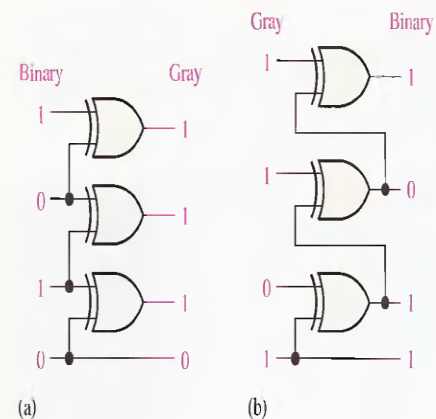
(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.

(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution (a) 0101_2 is 0111 Gray. See Figure 6-45(a).

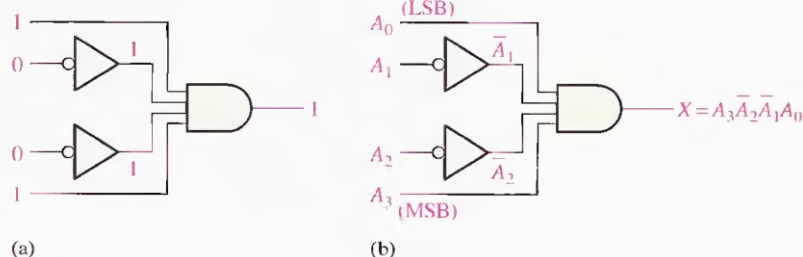
(b) 1011 Gray is 1101_2 . See Figure 6-45(b).

► FIGURE 6-45



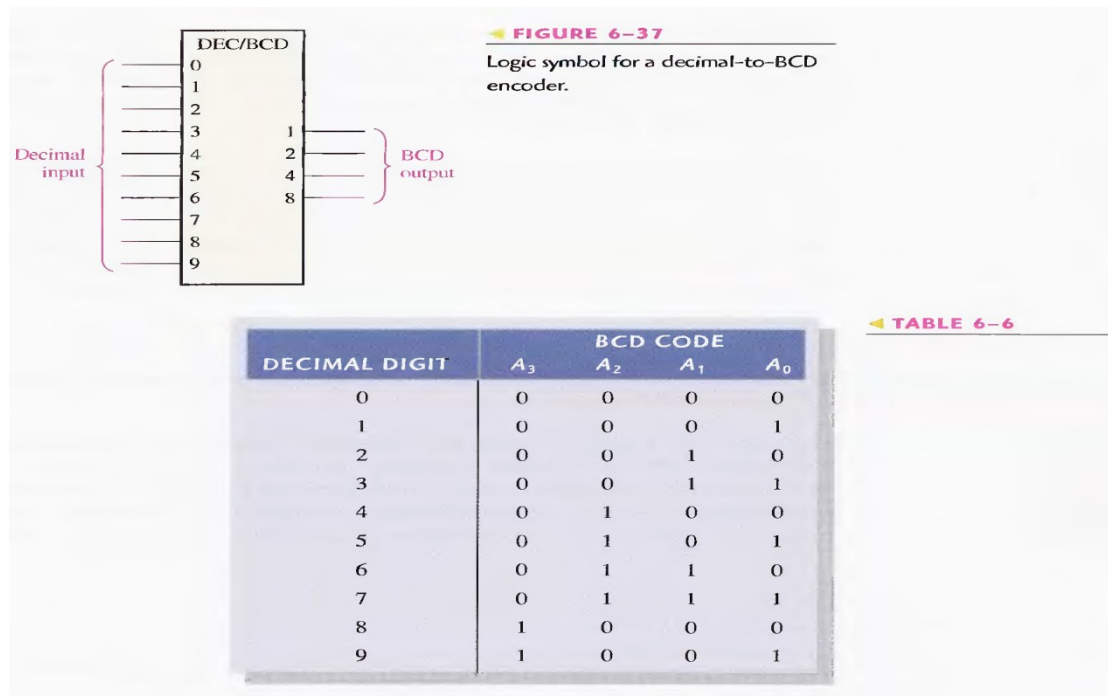
The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH. Therefore, you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle bits (the 0s), as shown in Figure 6-26.



ENCODER

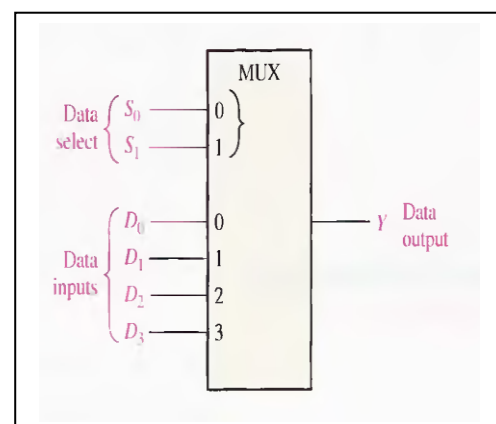
An **encoder** is a combinational logic circuit that essentially performs a “reverse” decoder function. An encoder accepts an active level on one of its inputs representing digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called *encoding*.



MULTIPLEXERS (DATA SELECTORS)

Logic symbol for a 4-input multiplexer (MUX) is shown in Figure below. Notice that there are two data-select lines because with two select bits. Any one of the four data input lines have selected.

2-bit code on the data-select (S) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 ($S_1 = 0$ and $S_0 = 0$) is applied to the data-select lines, the data on input D_0 appear on the data-output line. If a binary 1 ($S_1 = 0$ and $S_0 = 1$) is applied to the data-select lines, the data on input D_1 appear on the data output. If a binary 2 ($S_1 = 1$ and $S_0 = 0$) is applied, the data on D_2 appear on the output. If a binary 3 ($S_1 = 1$ and $S_0 = 1$) is applied, the data on D_3



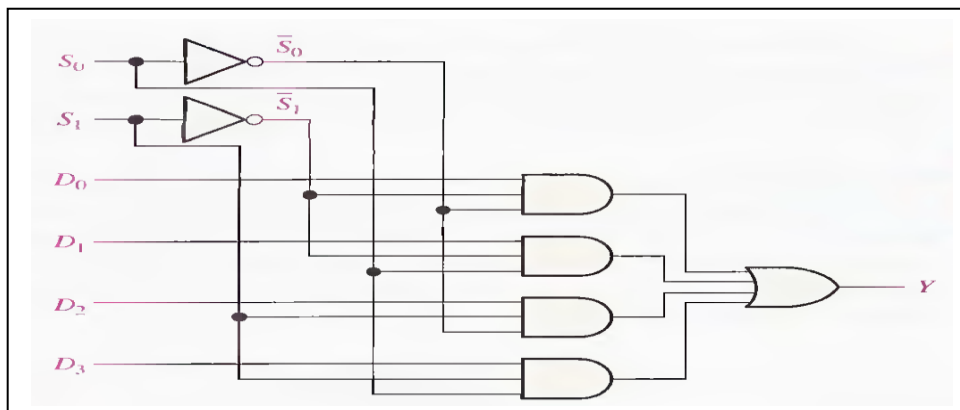
DATA-SELECT INPUTS		INPUT SELECTED
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0 \bar{S}_1 \bar{S}_0$.
 The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1 \bar{S}_1 S_0$.
 The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2 S_1 \bar{S}_0$.
 The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3 S_1 S_0$.

When these terms are OR, the total expression for the data output is

$$Y = D_0 \bar{S}_1 \bar{S}_0 + D_1 \bar{S}_1 S_0 + D_2 S_1 \bar{S}_0 + D_3 S_1 S_0$$

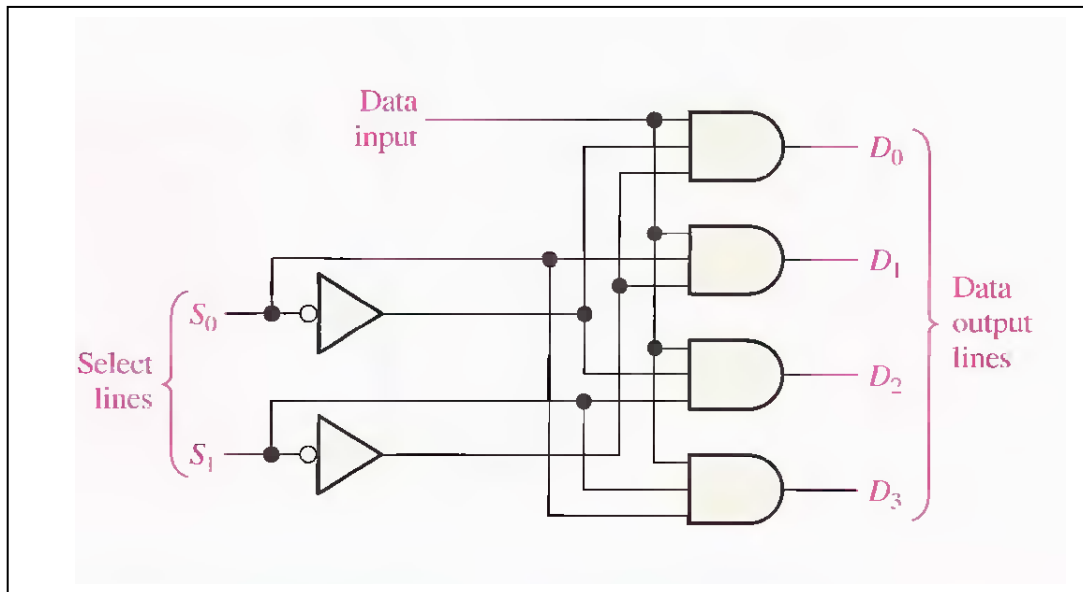
The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of S_1 and S_2



DEMULTIPLEXERS

A demultiplexer (DEMUX), basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer has known as a data distributor.

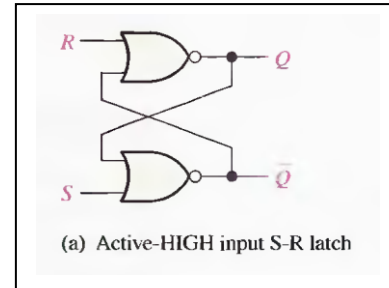
In Figure shows a 1-line-to-4-line demultiplexer (DEMUX) circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.



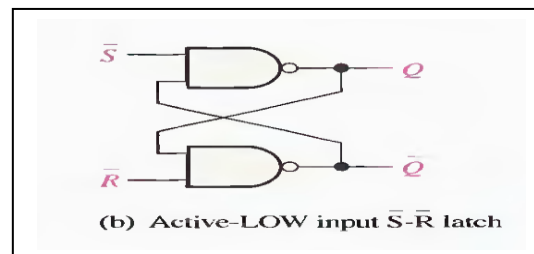
CHAPTER SIX/ Flip Flop

The S-R (SET-RESET) Latch

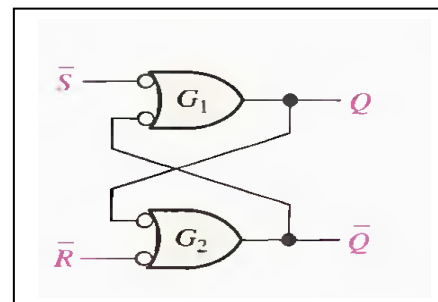
An active-**HIGH** input S-R (SET-RESET) latch is formed with two cross-coupled NOR gates, as shown in Figure



An active-**LOW** $\overline{S}-\overline{R}$ input latch has formed with two cross-coupled NAND gates, as shown in figure



We start explain the operation by using Negative OR gates as shown in figure



two inputs, \overline{S} and \overline{R} , and two outputs, Q and \overline{Q} . Let's start by assuming that both inputs and the Q output are HIGH. Since the Q output is connected back to an input of gate G_2 , and the \overline{R} input is HIGH, the output of G_2 must be LOW. This LOW output is coupled back to an input of gate G_1 , ensuring that its output is HIGH.

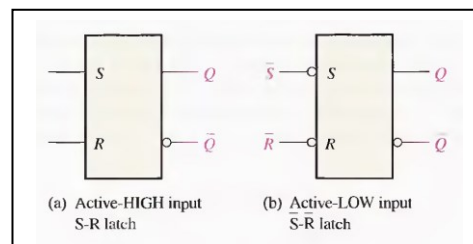
When the Q output is HIGH, the latch is in the **SET** state. It will remain in this state indefinitely until a LOW is temporarily applied to the \overline{R} input. With a LOW on the \overline{R} input and a HIGH on \overline{S} , the output of gate G_2 is forced HIGH. This HIGH on the \overline{Q} output is coupled back to an input of G_1 , and since the \overline{S} input is HIGH, the output of G_1 goes LOW. This

LOW on the Q output is then coupled back to an input of G_2 , ensuring that the \overline{Q} output remains HIGH even when the LOW on the \overline{R} input is removed. When the Q output is LOW, the latch is in the **RESET** state. Now the latch remains indefinitely in the RESET state until a LOW is applied to the \overline{S} input.

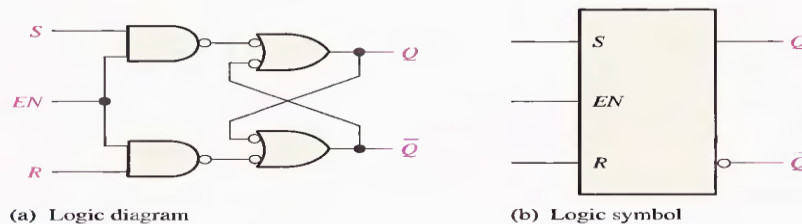
The truth table as shown in figure below

INPUTS		OUTPUTS		COMMENTS
\bar{S}	\bar{R}	Q	\bar{Q}	
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

Logic symbols for both the active-HIGH input and the active-LOW input latches are shown in figure



A gated latch requires an enable input, EN (G is also used to designate an enable input). The logic diagram and logic symbol for a gated S-R latch are shown in Figure 7–8. The S and R inputs control the state to which the latch will go when a HIGH level is applied to the EN input. The latch will not change until EN is HIGH; but as long as it remains HIGH, the output is controlled by the state of the S and R inputs. In this circuit, the invalid state occurs when both S and R are simultaneously HIGH.

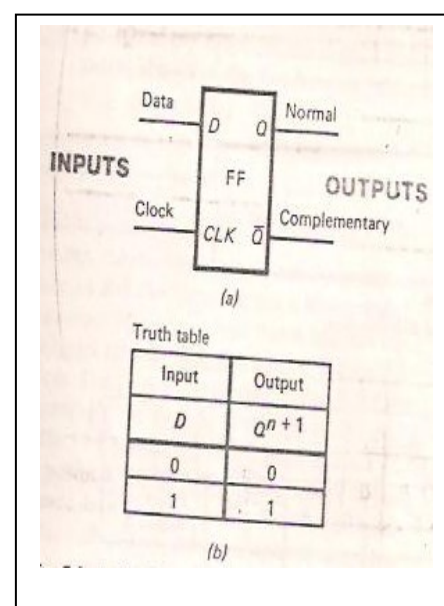


D Flip Flop:

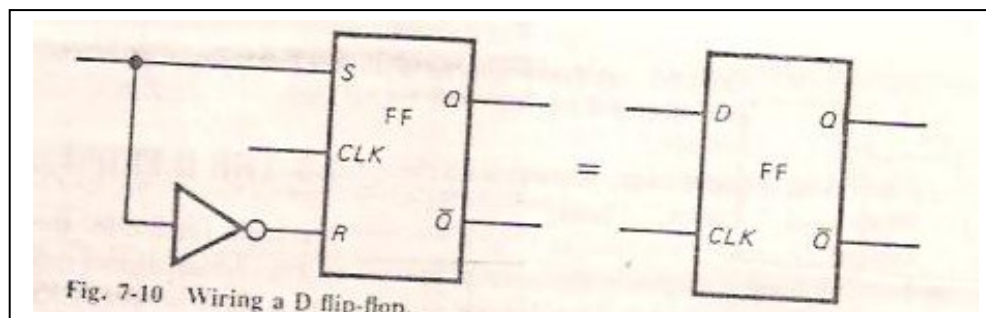
Logic symbol for D- flip-flop has shown in figure
It has only one data input (D) and clock input (CLK)

the output are labeled Q and \bar{Q} the D flip-flop is often called delay flip flop because the data (0 or 1) at input D is delayed one clock pulse from getting to the output Q

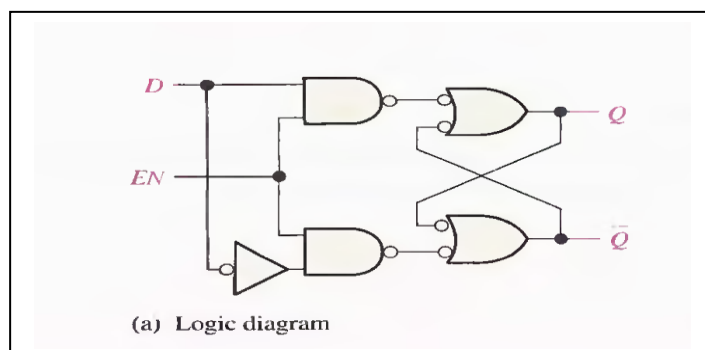
From truth table Notice that output Q follows input D after one clock pulse (Q^{n+1})



We can make D flip flop from a clocked R-S flip flop by adding inverter as shown



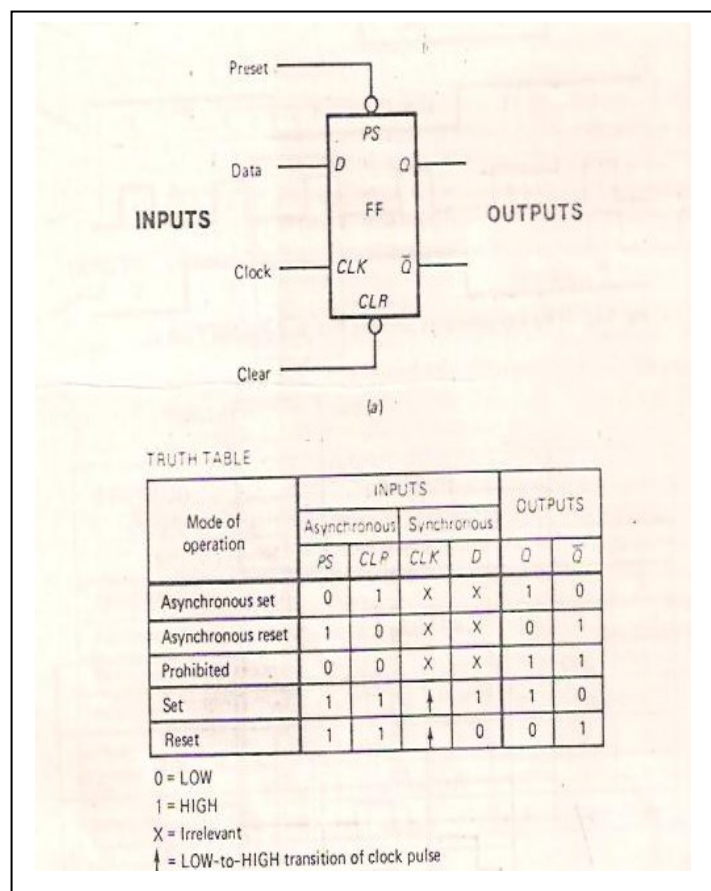
Logic diagram of D flip flop As shown below



Always we use the D flip flop contained in IC as shown in figure

We have two extra input
[PS(preset) and CLR (clear)]
PS input sets o/p Q=1 when enabled
by logic 0
The CLR i/p clear o/p to 0 when
enabled by logic 0
The PS and CLR override the D and
CLK I/P
As shown in truth table

When we have the PS and CLR the
flip flop operate as Asynchronous (not
synchronous)
If the flip flop disable the PS and CLR
therefore in synchronous operation
and can be set and reset by D and
CLK input this can be see from the
last two line in truth table.



J-K Flip Flop

Most widely used and universal Flip Flop

The I/P label J and K are data input

CLK is the clock input

Q and \bar{Q} are the normal and complementary o/p

From truth table we see

If J and K =0 the flip flop in hold mode the data input not effect on output then the output in hold on last data present line 1 in truth table

Line 2 and 3 is the reset and set condition for Q output

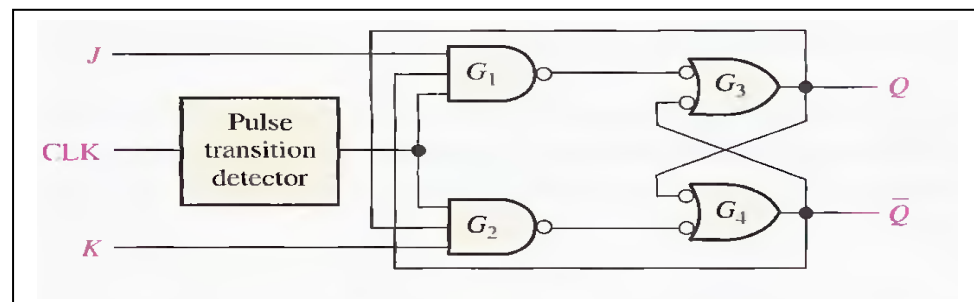
Line 4 toggle position of J-K flip flop when both data J=K=1 therefore the Q O/P will repeat clock pulses causes turn off-on-off-on and so on

(a)

(b)

Mode of operation	INPUTS			OUTPUTS		
	CLK	J	K	Q	\bar{Q}	Effect on output Q
Hold		0	0	No change	No change	No change – disable
Reset		0	1	0	1	Reset or cleared to 0
Set		1	0	1	0	Set to 1
Toggle		1	1	Toggle	Toggle	Changes to opposite state

The logic diagram of J-K flip-flop as shown



The commercial logic symbols for J_K flip flop as shown

We see asynchronous input (preset and clear) and the synchronous is the clock input

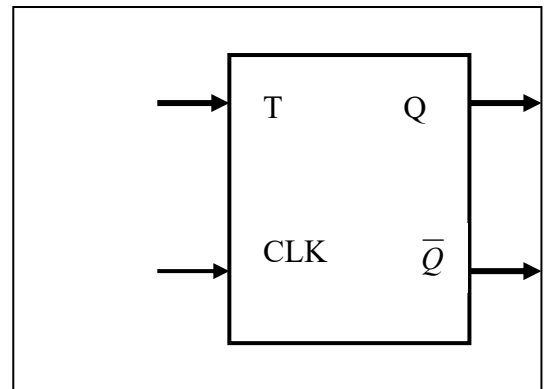
(a)

Mode of operation	INPUTS					OUTPUTS	
	Asynchronous		Synchronous			Q	\bar{Q}
	PS	CLR	CLK	J	K	Q	\bar{Q}
Asynchronous set	0	1	X	X	X	1	0
Asynchronous reset	1	0	X	X	X	0	1
Prohibited	0	0	X	X	X	1	1
Hold	1	1		0	0	No change	No change
Reset	1	1		0	1	0	1
Set	1	1		1	0	1	0
Toggle	1	1		1	1	Opposite state	Opposite state



0 = LOW
1 = HIGH

T –Flip Flop

The logic symbols for T- flip flop



The output is toggle if the input data $T = 1$ with clock (reverse of previous state) as shown in truth table

CK	T	$Q(t+1)$	وضع التشغيل
	0	$Q(t)$	No Change
	1	$\bar{Q}(t)$	Toggle

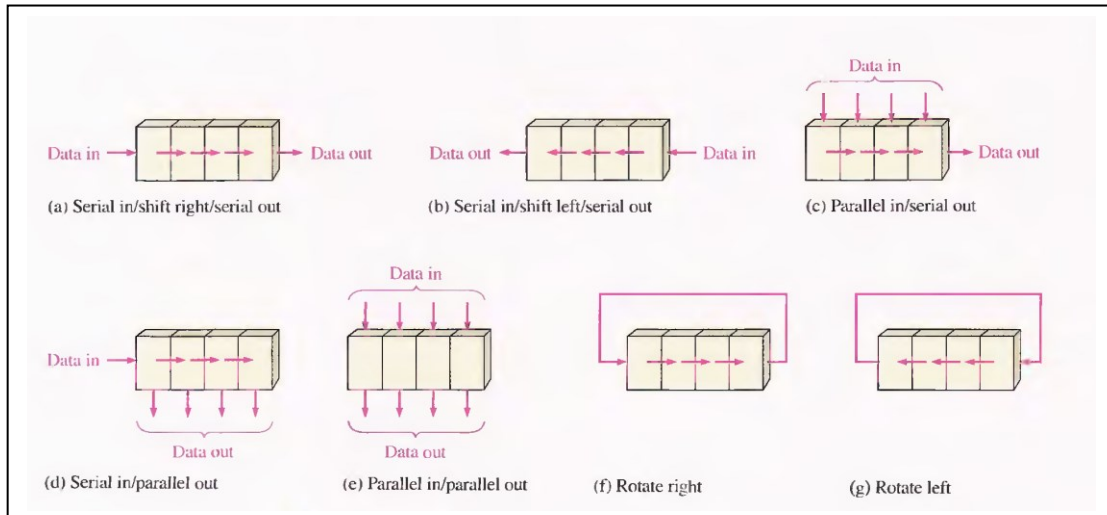
Also can be used the J-K flip flop to represent T flip flop by short J and $K = 1$

CHAPTER SEVEN

SHIFT REGISTER

BASIC SHIFT REGISTER FUNCTIONS

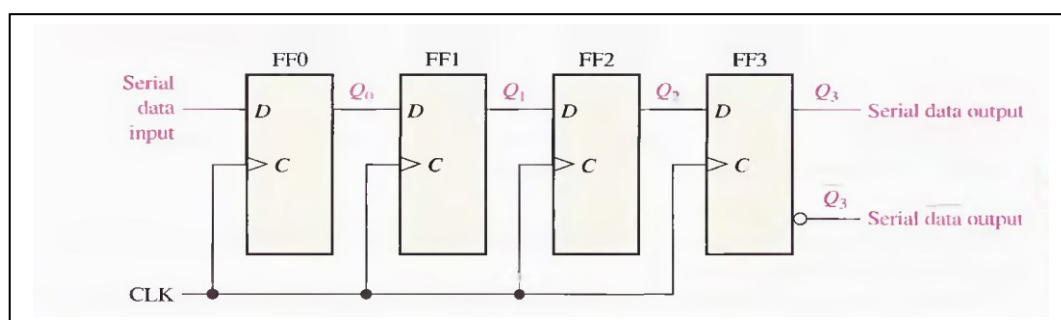
A register is a digital circuit with two basic functions: data storage and data movement. We have many type of shift register as shown



Serial IN/Serial OUT SHIFT REGISTERS

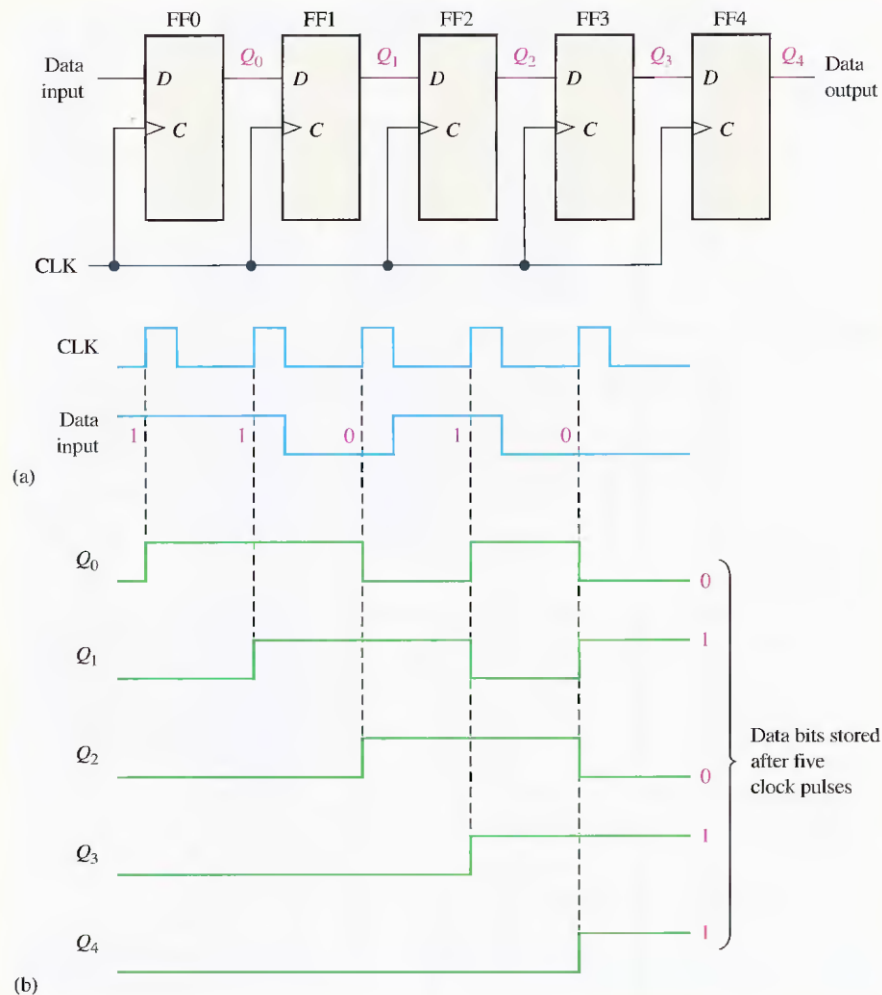
The serial in/serial out shift register accepts data serially-that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

Let first look at the serial entry of data into a typical shift register. Figure below shows a 4-bit device implemented with D flip-flops. With four stages, this register can store up to four bits of data.



Example:

Show the states of the 5-bit register in Figure 9–6(a) for the specified data input and clock waveforms. Assume that the register is initially cleared (all 0s).

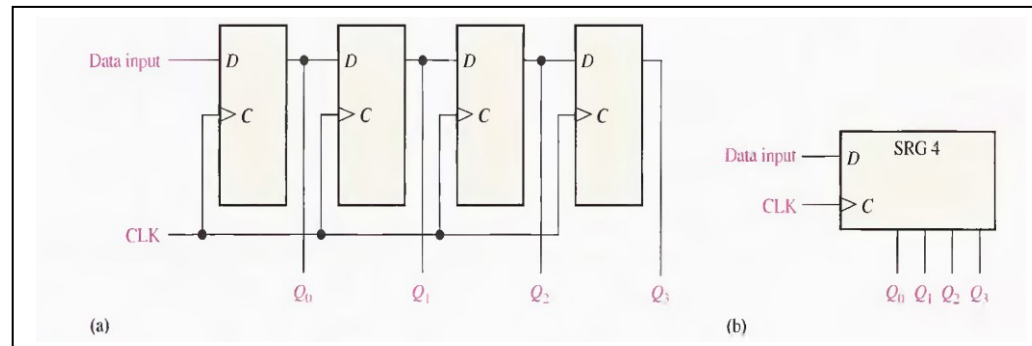


Solution The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4Q_3Q_2Q_1Q_0 = 11010$ after five clock pulses. See Figure 9–6(b).

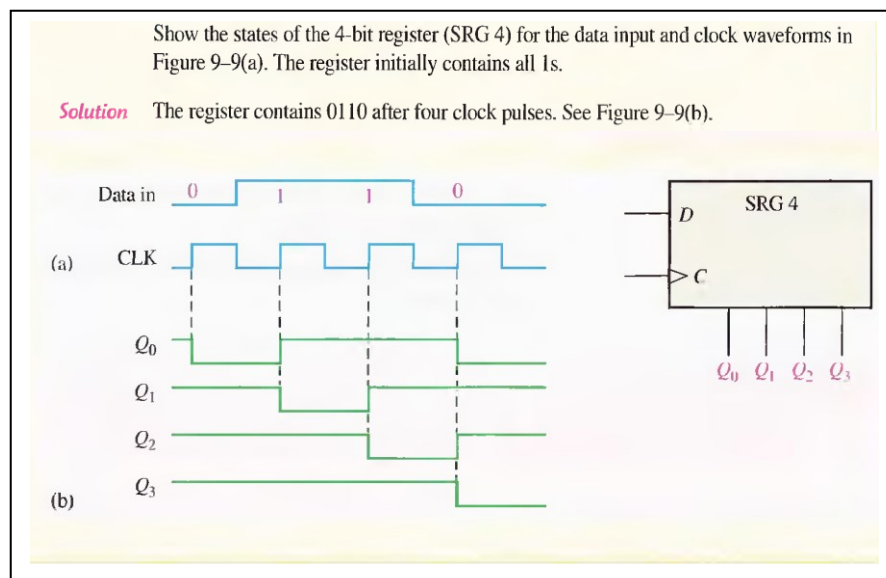
SERIAL IN/PARALLEL OUT SHIFT REGISTERS

in the parallel output register. the output of each stage is available. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial

Figure below shows a 4-bit serial in/parallel out shift register and its logic block symbol



Example :

**PARALLEL IN/SERIAL OUT SHIFT REGISTERS**

For parallel data, multiple bits have transferred at one time.

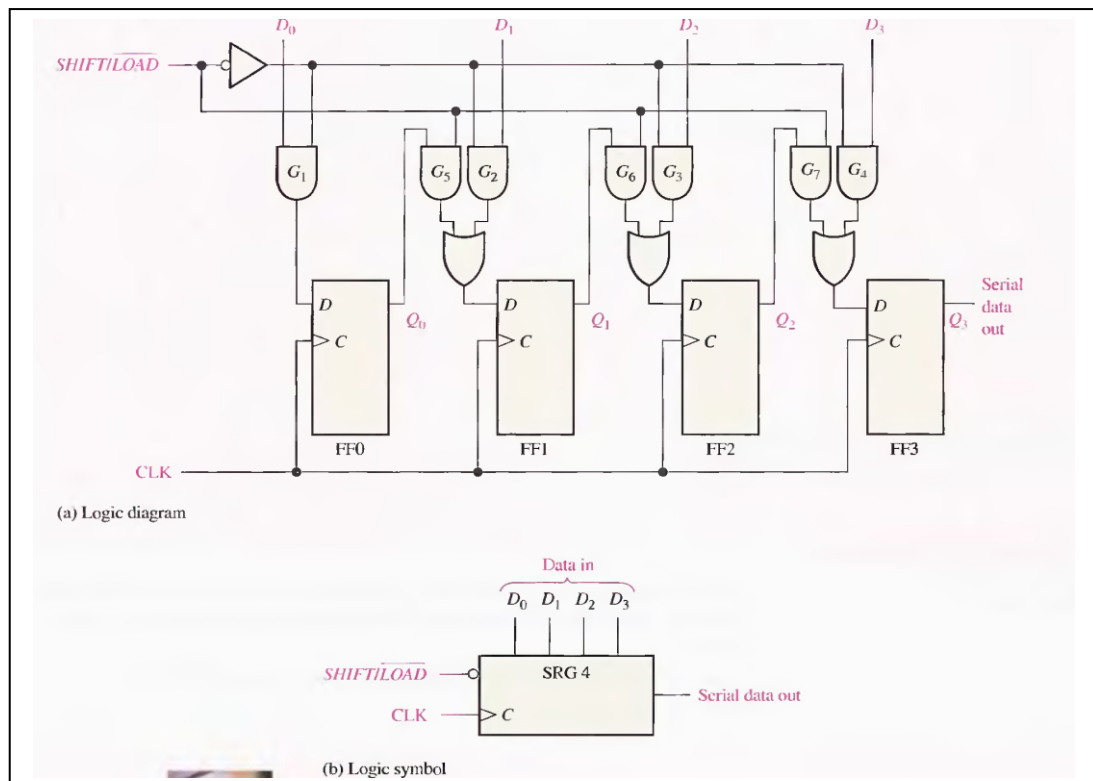
Figure below illustrates a 4-bit parallel in/serial out shift register and a typical logic symbol.

Notice that there are four data-input lines, D_0 , D_1 , D_2 , and D_3 , and $\overline{\text{SHIFT}} / \text{LOAD}$ input, which allows four bits of data to load in parallel into the register.

When $\overline{\text{SHIFT}} / \text{LOAD}$ is LOW, gates G 1 through G 4 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will set and those with $D = 0$ will reset. Thereby storing all four bits simultaneously.

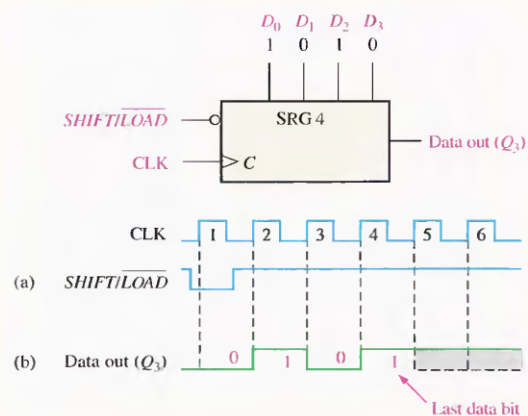
When $\overline{\text{SHIFT}} / \text{LOAD}$ is HIGH, gates G_1 , through G_4 are disabled and gates G_5 ; through G_7 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the $\overline{\text{SHIFT}} / \text{LOAD}$ input.

Notice that FF0 has a single AND to disable the parallel input, D_0 . It does not require an AND/OR arrangement because there is no serial data in.



Example

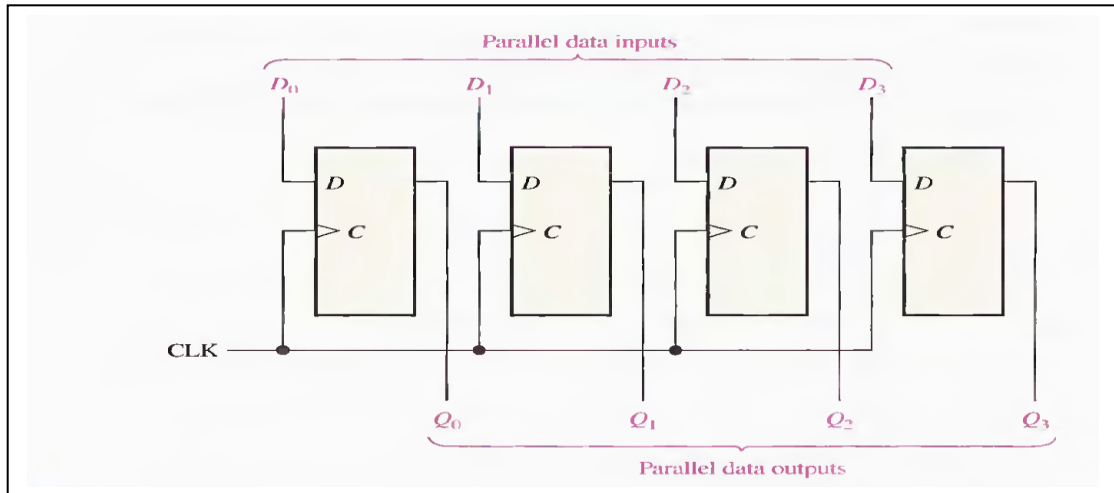
Show the data-output waveform for a 4-bit register with the parallel input data and the clock and *SHIFT/LOAD* waveforms given in Figure 9–13(a). Refer to Figure 9–12(a) for the logic diagram.



Solution On clock pulse 1, the parallel data ($D_0D_1D_2D_3 = 1010$) are loaded into the register, making Q_3 a 0. On clock pulse 2 the 1 from Q_2 is shifted onto Q_3 ; on clock pulse 3 the 0 is shifted onto Q_3 ; on clock pulse 4 the last data bit (1) is shifted onto Q_3 ; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D input remains a 1). See Figure 9–13(b).

PARALLEL IN/PARALLEL OUT SHIFT REGISTERS

The parallel in/parallel out register, immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs.



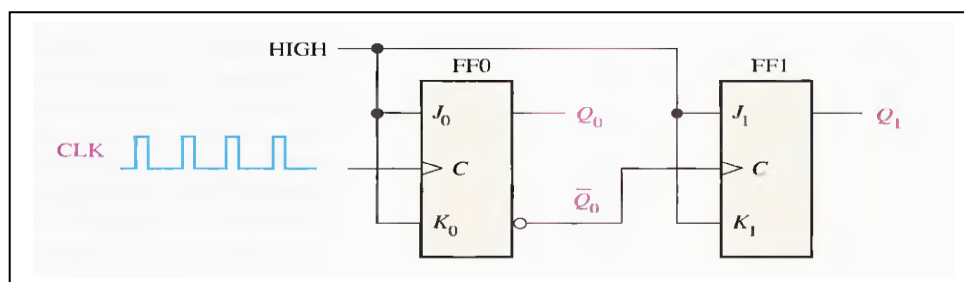
CHAPTER EIGHT Counters

ASYNCHRONOUS COUNTER OPERATION

The term asynchronous refers to events that do not have a fixed time relationship with each other and, generally, do not occur at the same time. An asynchronous counter is one in which the flip-flops (FF) within the counter do not change states at exactly the same time because they do not have a common clock pulse. A counter can have 2^n states, where n is the number of flip-flops.

A 2-Bit Asynchronous Binary Counter

Figure below shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) has applied to the clock input (C) of only the first flip-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the Q_0 output of FF0. FF0 changes state at the positive-going edge of each clock pulse. But FF1 changes only when triggered by a positive-going transition of the Q_0 output of FF0.



The binary state sequence of 2 bit as shown in table below

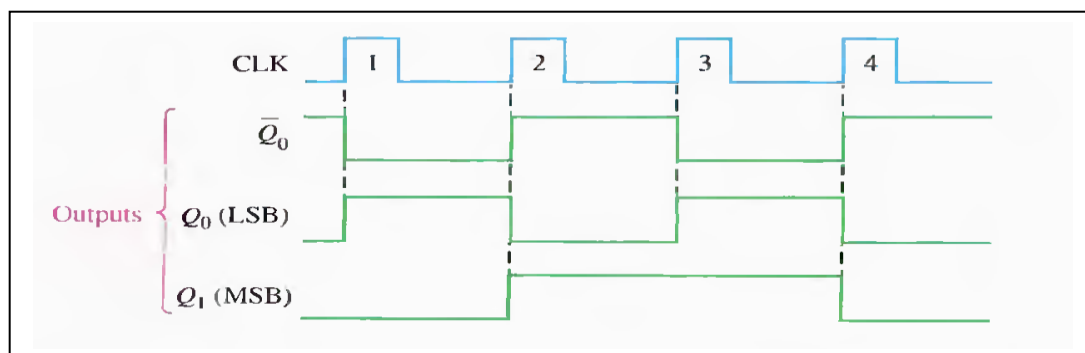
CLOCK PULSE	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Since it goes through a binary sequence, the counter in Figure 8-1 is a binary counter. It actually counts the number of clock pulses up to three, and on the fourth pulse it recycles to its original state ($Q_0 = 0$, $Q_1 = 0$). The term recycle has commonly applied to counter operation; it refers to the transition of the counter from its final state back to its original state.

The Timing Diagram

The positive-going edge of CLK1 (clock pulse]) causes the Q_0 output of FF0 to go HIGH, At the same time the \bar{Q}_0 output goes low. But it has no effect on FF1 because a positive -going transition must occur to trigger the flip-flop. After the

leading edge of CLK1, $Q_0 = 1$ and $\bar{Q}_0 = 0$. The positive-going edge of CLK2 causes Q_0 to go LOW. Output \bar{Q}_0 goes HIGH and triggers FF1, causing Q_1 to go HIGH. After the leading edge of CLK2, $Q_0 = 0$ and $Q_1 = 1$. The positive-going edge of CLK3 causes Q_0 to go HIGH again. Output \bar{Q}_0 goes LOW and has no effect on FF1. Thus, after the leading edge of CLK3, $Q_0 = 1$ and $Q_1 = 1$. The positive-going edge of CLK4 causes Q_0 to go LOW, while \bar{Q}_0 goes HIGH and triggers FF1, causing Q_1 to go LOW. After the leading edge of CLK4, $Q_0 = 0$ and $Q_1 = 0$. The counter has now recycled to its original state (both flip-flops are RESET).

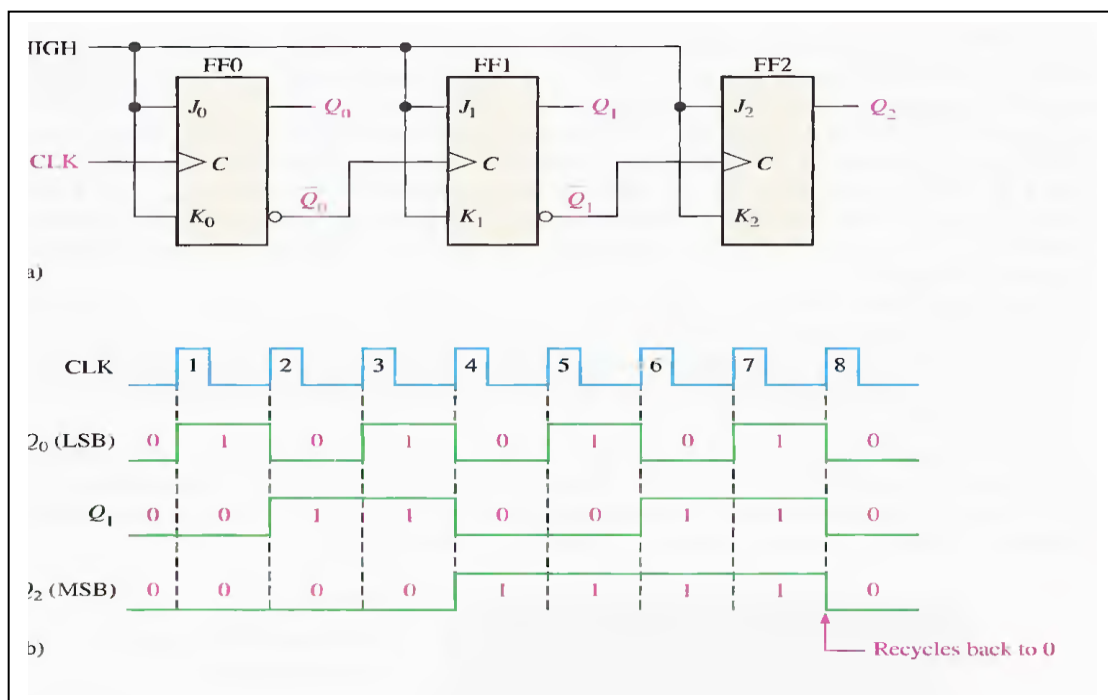
**A 3-Bit Asynchronous Binary Counter**

The state sequence for a 3-bit binary counter has listed in table below:

basic operation is the same as that of the 2-bit counter except that the 3-bit counter has eight states, due to its three flip-flops.

CLOCK PULSE	Q_2	Q_1	Q_0
Initially	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
8 (recycles)	0	0	0

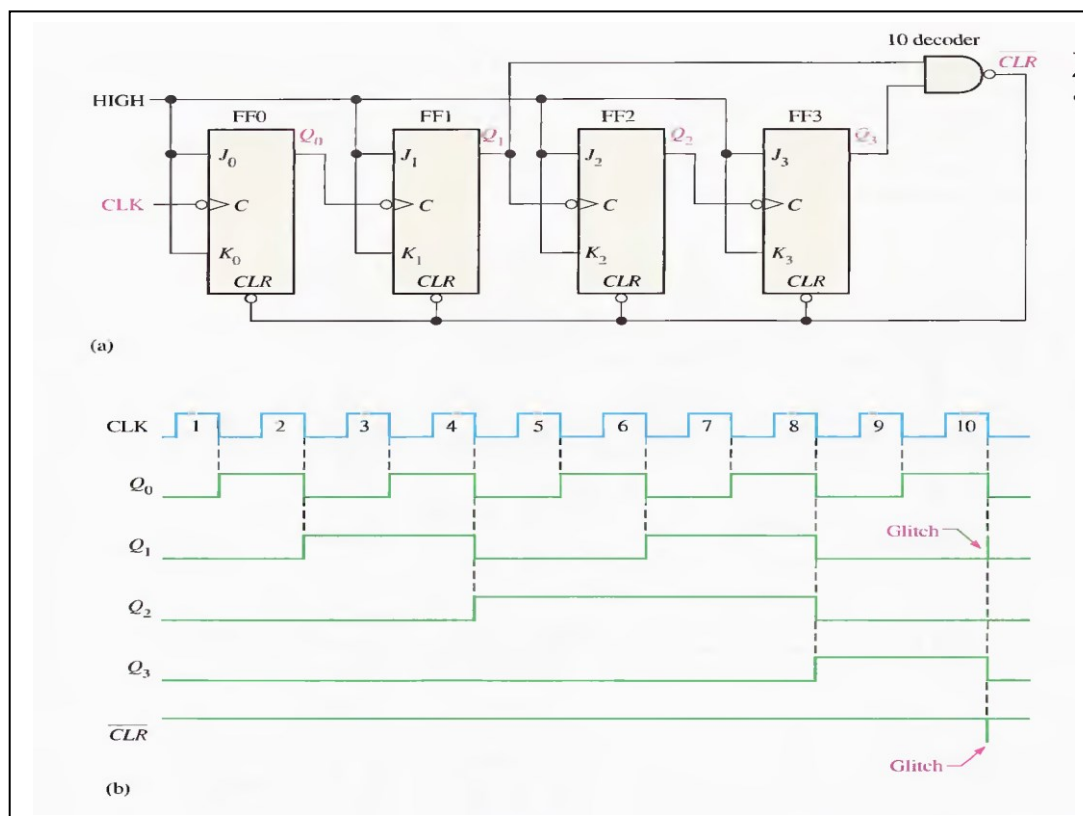
Notice that the counter progresses through a binary count of zero through seven and then recycles to the zero state. This counter can be easily expanded for higher count, by connecting additional toggle flip-flops.



Asynchronous Decade Counters

One common modulus for counters with truncated sequences is ten (called MOD10).

Counters with ten states in their sequence have called decade counters. A decade counter with a count sequence of zero (0000) through nine (1001) is a BCD decade counter because its ten-state sequence produces the BCD code. To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. For example, the BCD decade counter must recycle back to the 0000 state after the 1001 state. One way to make the counter recycle after the count of nine (1001) is to decode count ten



(1010) with a NAND gate and connect the output of the NAND gate to the clear (\overline{CLR}) inputs of the flip-flops, as shown

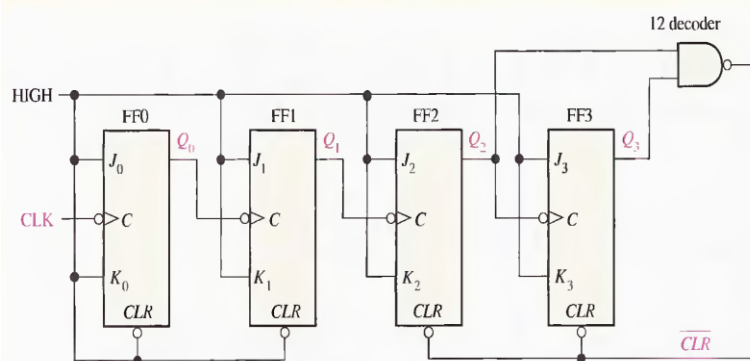
Example:

Show how an asynchronous counter can be implemented having a modulus of twelve with a straight binary sequence from 0000 through 1011.

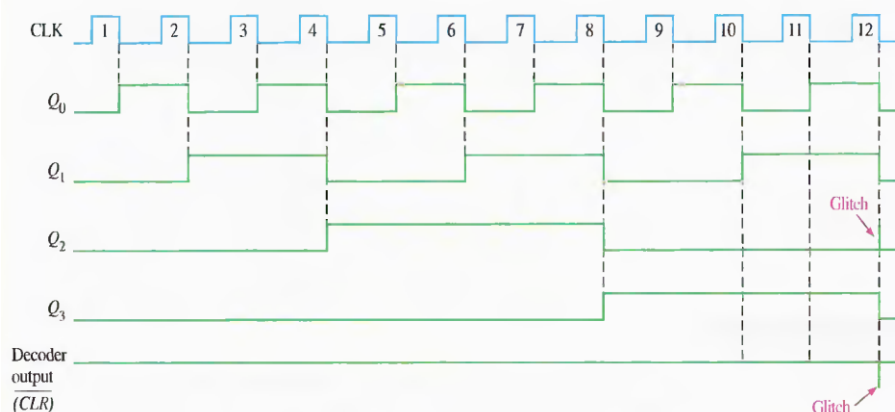
Solution Since three flip-flops can produce a maximum of eight states, four flip-flops are required to produce any modulus greater than eight but less than or equal to sixteen. When the counter gets to its last state, 1011, it must recycle back to 0000 rather than going to its normal next state of 1100, as illustrated in the following sequence chart:

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	
.	.	.	.	
.	.	.	.	
1	0	1	1	← Recycles
1	1	0	0	← Normal next state

Observe that Q_0 and Q_1 both go to 0 anyway, but Q_2 and Q_3 must be forced to 0 on the twelfth clock pulse. Figure 8-7(a) shows the modulus-12 counter. The NAND gate partially decodes count twelve (1100) and resets flip-flop 2 and flip-flop 3. Thus, on the twelfth clock pulse, the counter is forced to recycle from count eleven to count zero, as shown in the timing diagram of Figure 8-7(b). (It is in count twelve for only a few nanoseconds before it is reset by the glitch on \overline{CLR} .)



(a)



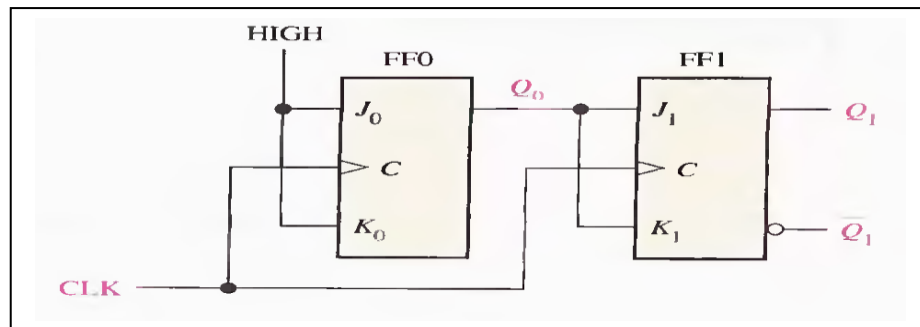
(b)

SYNCHRONOUS COUNTER OPERATION

The term **synchronous** refers to events that have a fixed time relationship with each other. A **synchronous** counter is one in which all the flip-flops in the counter have clocked at the same time by a common clock pulse.

A 2-Bit Synchronous Binary Counter

Figure below shows a 2-bit synchronous binary counter. Notice that an arrangement different from that for the asynchronous counter must be used for the J_1 and K_1 inputs of FF1 in order to achieve a binary sequence.

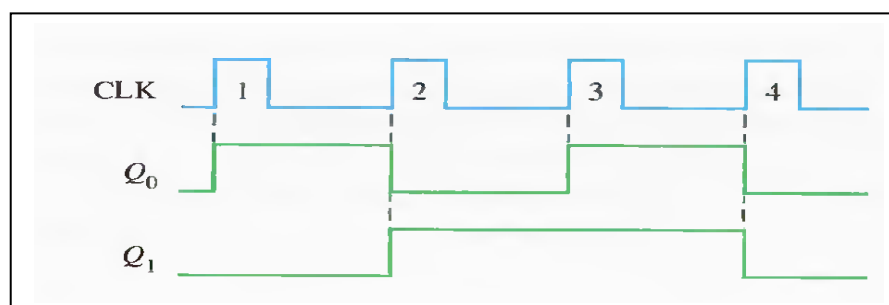


The operation of this synchronous counter is as follows:

First, assume that the counter is initially in the binary 0 state: that is. Both flip-flops are RESET. When the positive edge of the first clock pulse is applied, FF0 will toggle and Q_0 will therefore go HIGH. After CLK1, $Q_0 = 1$ and $Q_1 = 0$. When the leading edge of CLK2 occurs, FF0 will toggle and Q_0 will go LOW. Since FF1 has a HIGH ($Q_0 = 1$) on its J_1 , and K_1 inputs at the triggering edge of this clock pulse, the flip-flop toggles and Q_1 goes HIGH. Thus, after CLK2, $Q_0 = 0$ and $Q_1 = 1$ (which is a binary 2 state).

When the leading edge of CLK3 occurs. FF0 again toggles to the SET state ($Q_0 = 1$), and FF1 remains SET ($Q_1 = 1$) because its J_1 and K_1 inputs are both LOW ($Q_0 = 0$). After this triggering edge, $Q_0 = 1$ and $Q_1 = 1$ (which is a binary 3 state).

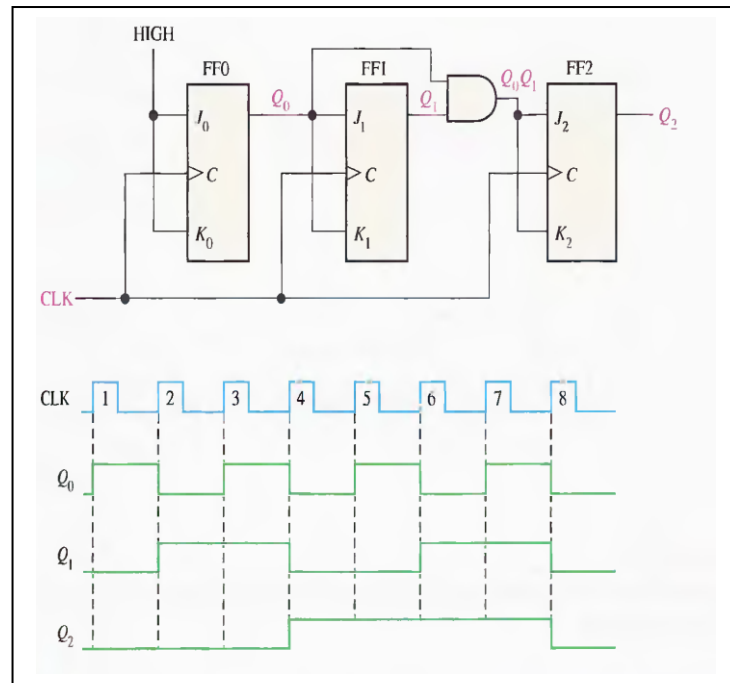
Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they both have a toggle condition on their J and K inputs.



A 3-Bit Synchronous Binary Counter

A 3-bit synchronous binary counter is shown in Figure below its timing diagram is shown in Figure You can understand this counter operation by examining its sequence of states as shown in table

CLOCK PULSE	Q_2	Q_1	Q_0
Initially	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
8 (recycles)	0	0	0



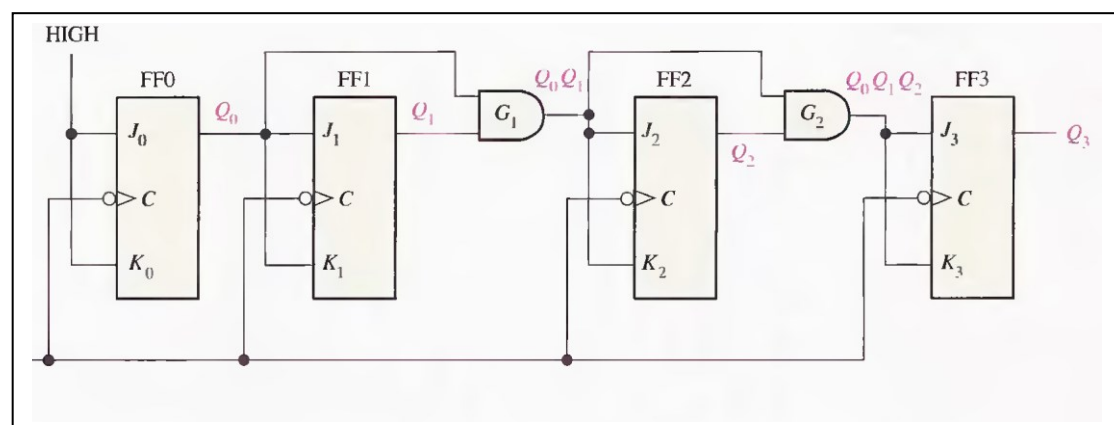
First, let's look at Q_0 . Notice that Q_0 changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state.

To produce this operation, FF0 must be held in the toggle mode by constant HIGH on its J_0 and K_0 inputs. Notice that Q_1 goes to the opposite state following each time Q_0 is a 1. This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes the counter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF1. When Q_0 is a 1 and a clock pulse occurs, FF1 is in the toggle mode and therefore changes state. The other times, when Q_0 is a 0, FF1 is in the no-change mode and remains in its present state. Next, let's see how FF2 is made to change at the proper times according to the binary sequence. Notice that both times Q_2 changes state, it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF2. Whenever both Q_0 and Q_1 are HIGH, the output of the AND gate makes the J_2 and K_2 inputs of FF2 HIGH, and FF2 toggles on the following clock pulse. At all other times, the J_2 and K_2 inputs of FF2 are held LOW by the AND gate output, and FF2 does not change state.

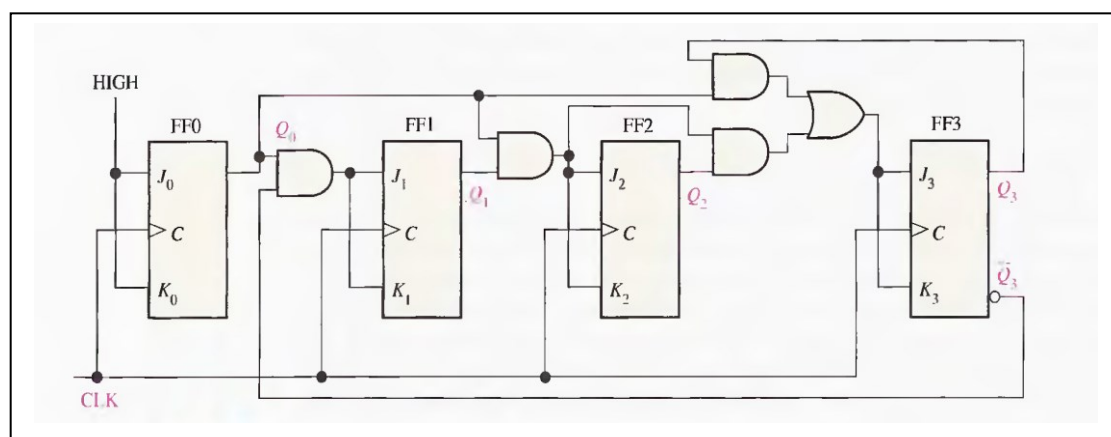
A 4-Bit Synchronous Binary Counter

Figure below shows a 4-bit synchronous binary counter,
This particular counter is implemented with negative edge-triggered flip-

flops. The reasoning behind the J and K input control for the first three flip-flops is the same as previously discussed for the 3-bit counter. The fourth stage, FF3, changes only twice in the sequence. Notice that both of these transitions occur following the times that Q_0 , Q_1 , and Q_2 are all HIGH. This condition is decoded by AND gate G_2 so that when a clock pulse occurs, FF3 will change state. For all other times the J_3 and K_3 inputs of FF3 are LOW, and it is in a no-change condition.

**A 4-Bit Synchronous Decade Counter**

As you know, a BCD decade counter exhibits a truncated binary sequence and goes from 0000 through the 1001 state. Rather than going from the 1001 state to the 1010 state, it recycles to the 0000 state. A synchronous BCD decade counter is shown in Figure below.



First, notice that FF0 (Q_0) toggles on each clock pulse, so the logic equation for its J_0 and K_0 inputs is $J_0 = K_0 = 1$

This equation is implemented by connecting J_0 and K_0 to a constant HIGH level.

Next, notice in Table below that FF1 (Q_1) changes on the next clock pulse each time $Q_0 = 1$ and $Q_3 = 0$, so the logic equation for the J_1 and K_1 inputs is

$$J_1 = K_1 = Q_0 \bar{Q}_3$$

This equation is implemented by ANDing Q_0 and $\overline{Q_3}$ and connecting the gate output to the J_1 and K_1 inputs of FF1.

Flip-flop 2 (Q_2) changes on the next clock pulse each time both $Q_0 = 1$ and $Q_1 = 1$. This requires an input logic equation as follows:

$$J_2 = K_2 = Q_0 Q_1$$

This equation is implemented by ANDing Q_0 and Q_1 and connecting the gate output to the J_2 and K_2 inputs of FF2.

Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

$$J_3 = K_3 = Q_0 Q_1 Q_2 + Q_0 Q_3$$

This function is implemented with the AND/OR logic connected to the J_3 and K_3 inputs of

FF3 as shown in the logic diagram

States of a BCD decade counter .table

CLOCK PULSE	Q_3	Q_2	Q_1	Q_0
Initially	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

UP/DOWN SYNCHRONOUS COUNTERS

In general, most up/down counters can be reversed at any point in their sequence. Table below shows the complete up/down sequence for a 3-bit binary counter. The arrows indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation. An examination of Q_0 for both the up and down sequences shows that FF0 toggles on each clock pulse. Thus, the J_0 and K_0 inputs of FF0 are

$$J_0 = K_0 = 1$$

For the up sequence, Q_1 changes state on the next clock pulse when $Q_0 = 1$. For the down sequence, Q_1 changes on the next clock pulse when $Q_0 = 0$. Thus, the J_1 and K_1 inputs of FF1 must equal 1 under the conditions expressed by the following equation:

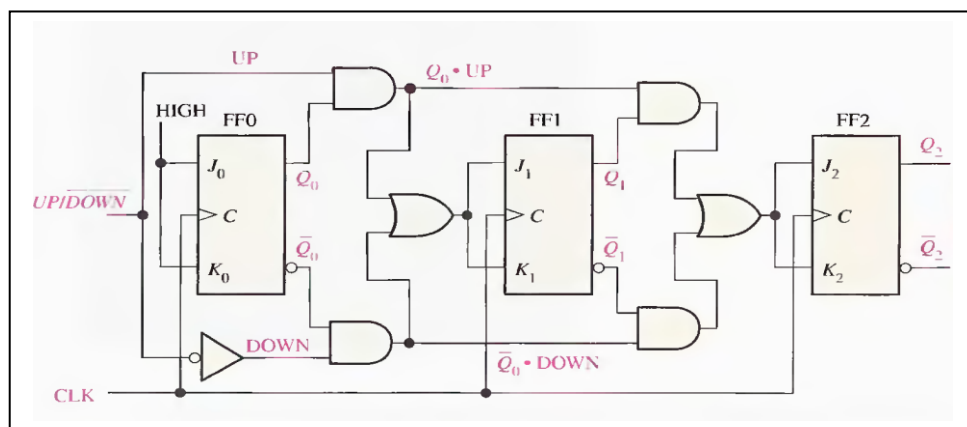
$$J_1 = K_1 = (Q_0 \cdot \text{UP}) + (\overline{Q_0} \cdot \text{DOWN})$$

For the up sequence, Q_2 changes state on the next clock pulse when $Q_0 = Q_1 = 1$. For the down sequence, Q_2 changes on the next clock pulse when $Q_0 = Q_1 = 0$. Thus, the J_2 and K_2 inputs of FF2 must equal 1 under the conditions expressed by the following equation:

$$J_2 = K_2 = (Q_0 \cdot Q_1 \cdot \text{UP}) + (\overline{Q_0} \cdot \overline{Q_1} \cdot \text{DOWN})$$

Figures below shows the logic diagram and truth table for UP/DOWN counter

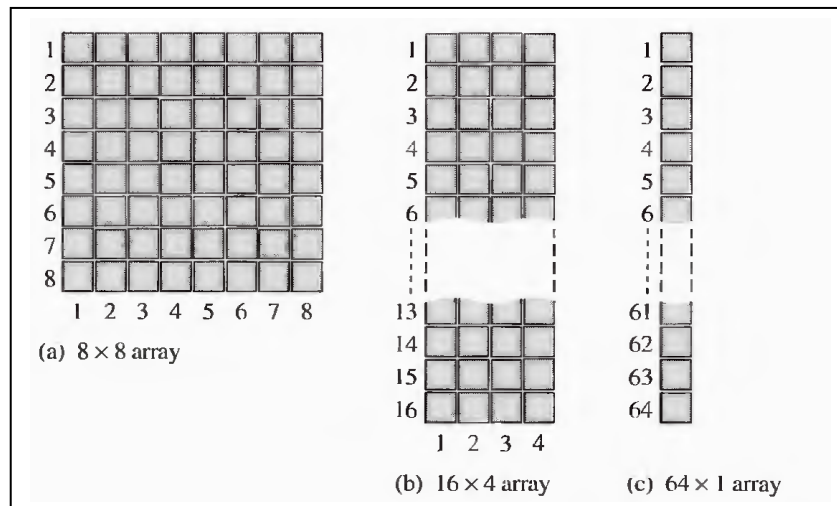
CLOCK PULSE	UP	Q_2	Q_1	Q_0	DOWN
0	↶	0	0	0	↷
1	↶	0	0	1	↷
2	↶	0	1	0	↷
3	↶	0	1	1	↷
4	↶	1	0	0	↷
5	↶	1	0	1	↷
6	↶	1	1	0	↷
7	↶	1	1	1	↷



CHAPTER NINE /Memories

The Basic Semiconductor Memory Array

Each storage element in a memory can retain either a 1 or a 0 and is called a cell. Memories made up of arrays of cells, as illustrated in Figure below Each block in the memory array represents one storage cell. and its location can be identified by specifying a row and a column.



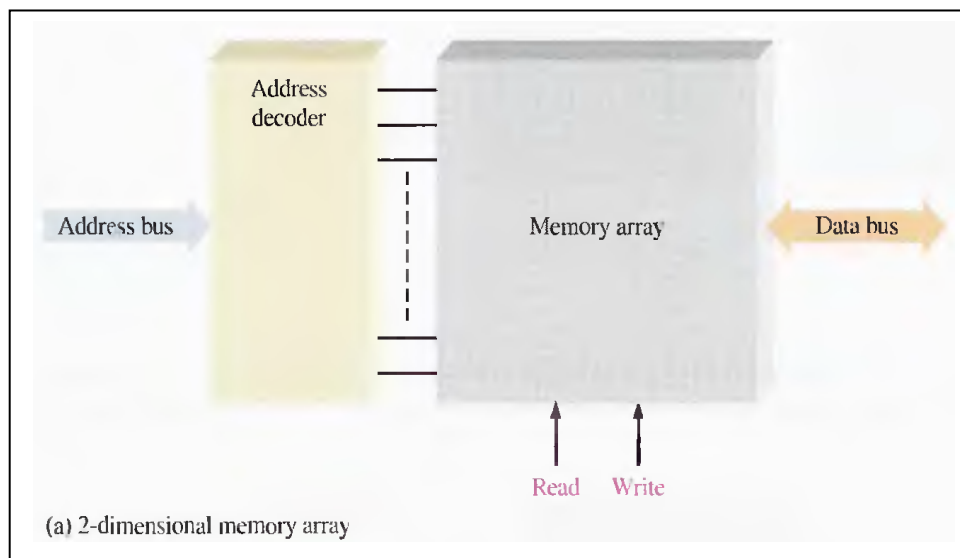
The location of a unit of data in a memory array called its address . The address of a byte specified only by the **row**.

Basic Memory Operations

Since a memory stores binary data. data must be put into the memory and data must be copied from the memory when needed. The write operation puts data into a specified address in the memory, and the read operation copies data out of a specified address in the memory. The addressing operation, which is part of both the write and the read operations, selects the specified memory address.

Data units go into the memory during a write operation and come out of the memory during a read operation on a set of lines called the data bus.

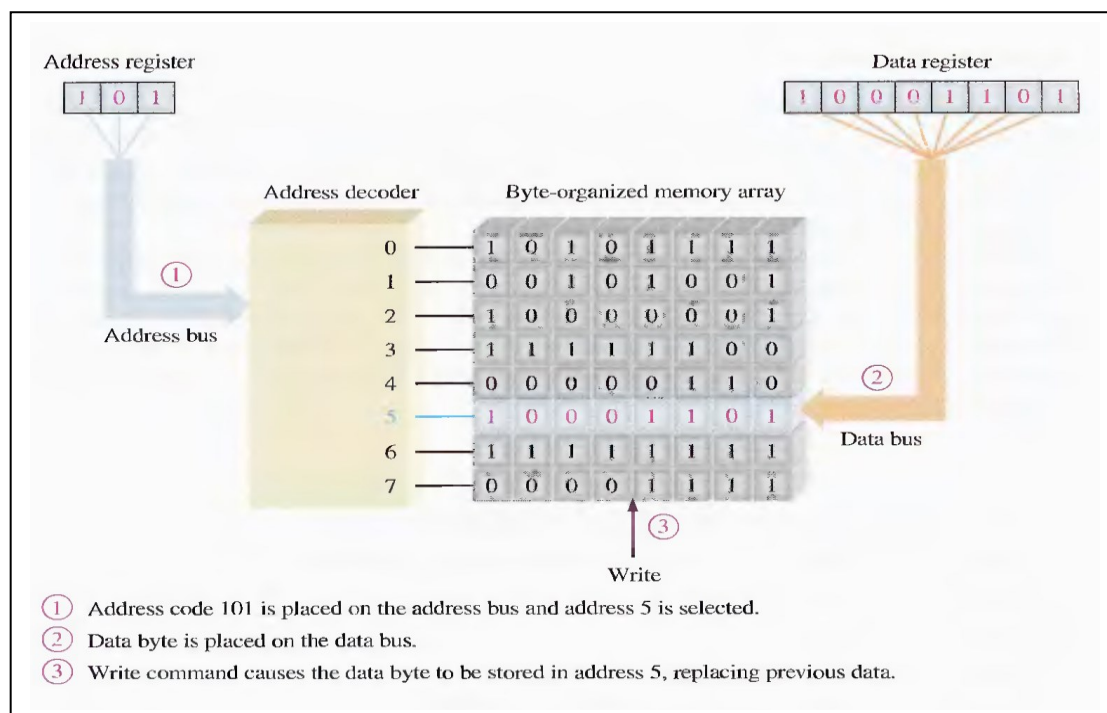
bus. As indicated in Figure below the data bus is bidirectional, which means that data can go in either direction (into the memory or out of the memory).



In this case, of byte-organized memories, the data bus has at least eight lines so that all eight bits in a selected address are transferred in parallel. For write or read operation. An address selected by placing a binary code representing the desired address on a set of lines called the address bus. The address code is decoded internally. And the appropriate address is selected.

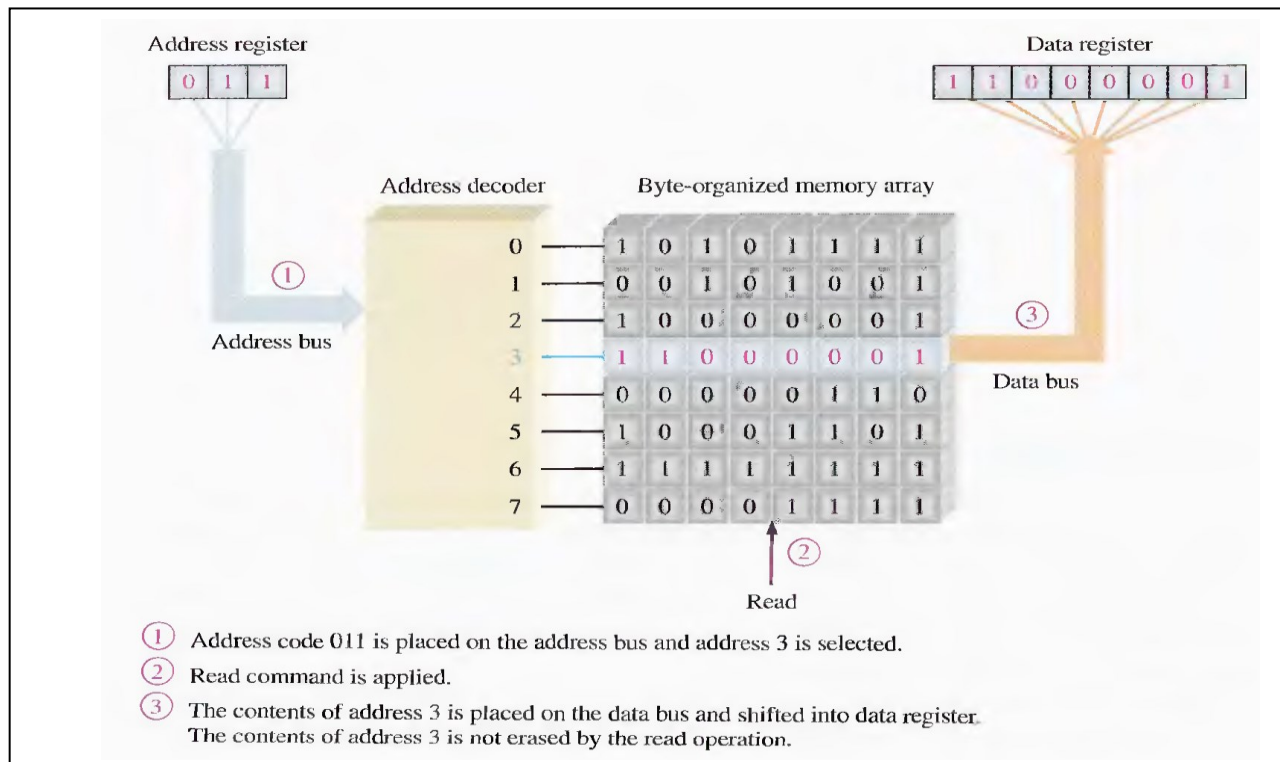
The Write Operation

A simplified write operation illustrated in Figure



Read Operation

A simplified read operation is illustrated in Figure



Bit: The smallest unit of binary data

Byte: data are handled in an 8-bit unit or in multiples of 8-bit units.

Nibbles: The byte can be split into two 4-bit units

Word: generally consists of one or more bytes.

RANDOM-ACCESS MEMORIES (RAMs)

RAM are read/write memories in which data can be written into or read from any selected address in any sequence. When a data unit is written into a given address in the RAM, the data unit previously stored at that address is replaced by the new data unit. When a data unit is read from a given address in the RAM, the data unit remains stored and is not erased by the read operation. This nondestructive read operation can be viewed as copying the content of an address while leaving the content intact. A

RAM is typically used for short-term data storage because it cannot retain stored data when power is turned off.

The RAM Family

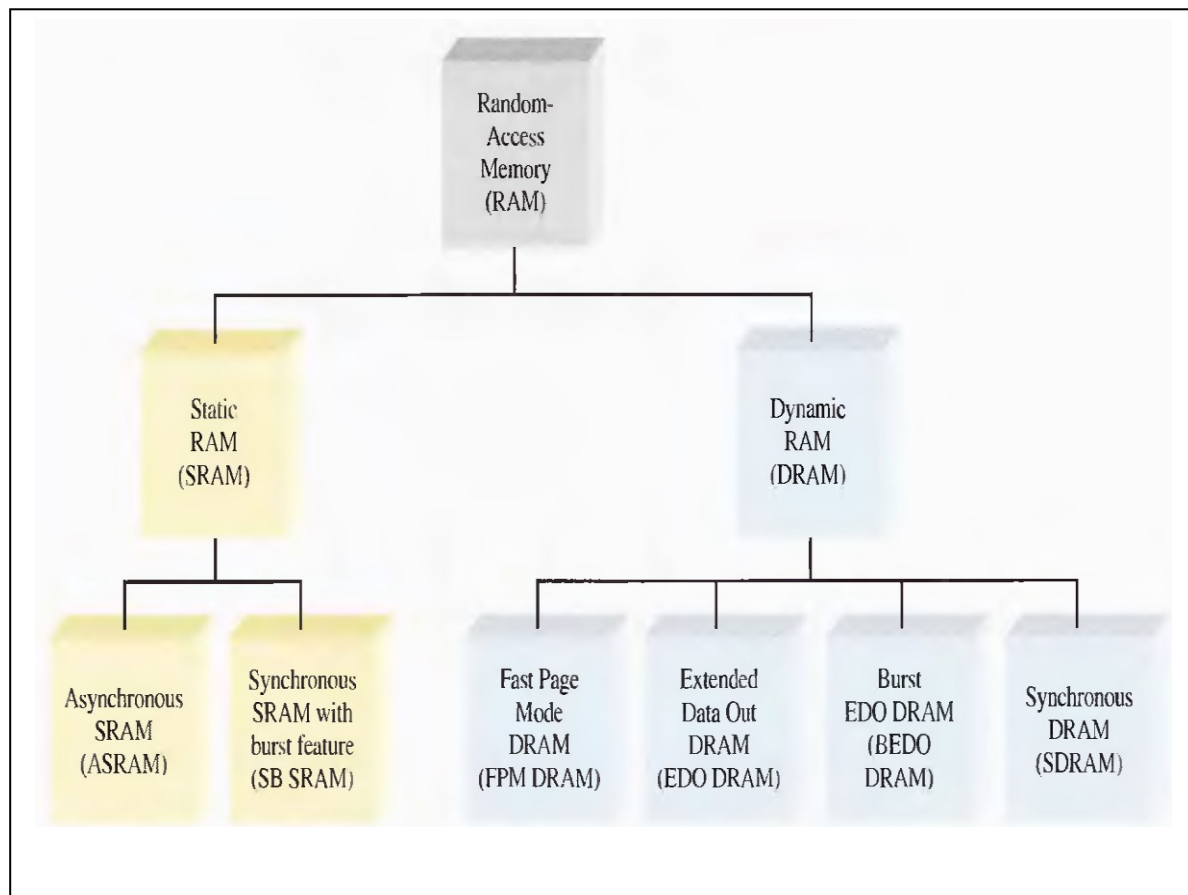
The two categories of RAM are the static RAM (**SRAM**) and the dynamic RAM (**DRAM**).

Static RAMs generally **use latches** as storage elements and can therefore store data indefinitely as long as dc power is applied.

Dynamic RAMs **use capacitors** as storage elements and cannot retain data very long without the capacitors being recharged by a process called refreshing.

Data can be read **much faster** from SRAMs than from DRAMs.

DRAMs can store much more data than SRAMs for a given physical size and cost because the DRAM cell is much simpler, and more cells can be crammed into a given chip area than in the SRAM. The type of RAM as shown in figure below



READ-ONLY MEMORIES (ROMs)

ROM contain permanently or semi permanently stored data, Which can be read from the memory but either cannot be changed at all or cannot be changed without specialized equipment. ROMs retain stored data when the power is off and are therefore nonvolatile memories.

The **mask ROM** is the type in which the data are permanently stored in the memory during the manufacturing process.

The **PROM**, or programmable ROM, is the type in which the data are electrically stored by the user with the aid of specialized equipment.

The **EPROM**, or erasable PROM.

The **UV EPROM** is electrically programmable by the user, but the stored data must be erased by exposure to ultraviolet light over a period of several minutes.

The electricallyerasable PROM (**EEPROM** or **E² PROM**) can be erased in a few milliseconds.

The types of ROM as shown in figure below

