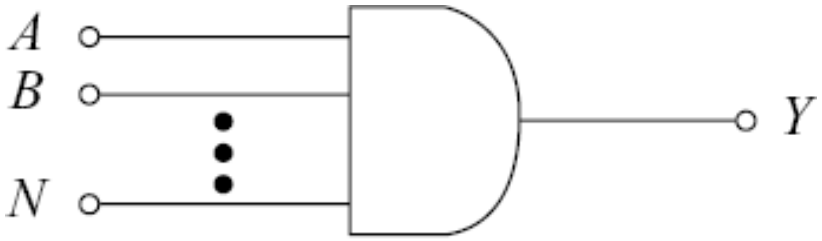
Logic Gates

LOGIC OPERATIONS & GATES

The basic operations are AND, OR, NOT, and FLIP-FLOP.

AND



$$Y = A \text{ AND } B \text{ AND } C \ldots \text{ AND } N$$
$$= A \cdot B \cdot C \cdot \ldots \cdot N$$
$$= ABC \ldots N \qquad (1.1)$$

Fig. 1.2 *The Standard Symbol for an* **AND** *Gate*

where *A*, *B*, *C*,… *N* are the input variables and *Y* is the output variable.

Equation (1.1) is known as the *Boolean equation* or the *logical equation* of the AND *gate*.

OR

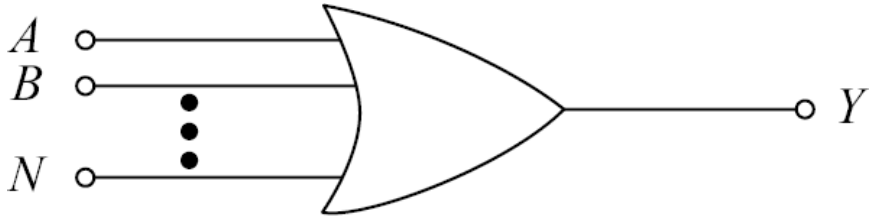


Fig. 1.5 ***The Standard Symbol for an OR Gate***

$$Y = A \text{ OR } B \text{ OR } C \ldots \text{ OR } N = A + B + C + \ldots + N \qquad (1.2)$$

Table 1.2 ***Truth Table of a 2-Input OR Gate***

Inputs		Output
A	***B***	***Y***
0	0	0
0	1	1
1	0	1
1	1	1

NOT

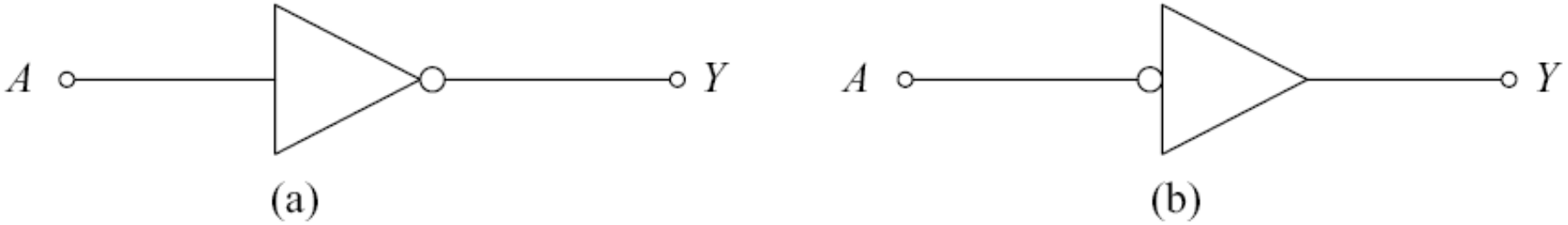


Fig. 1.7 ***The Standard Symbols for a* NOT *Gate***

$$Y = \text{NOT } A$$
$$= \overline{A}$$

"*Y* equals NOT *A*" or "*Y* equals complement of *A*". The NOT operation is also referred to as an *inversion* or *complementation*. The presence of a small circle, known as the *bubble*, always denotes inversion in digital circuits.

Table 1.3 ***Truth Table of a* NOT *Gate***

Input *A*	Output *Y*
0	1
1	0

NAND AND NOR OPERATIONS

NAND

The NOT–AND operation is known as the NAND operation.

$$Y' = AB \ldots N \qquad Y = \overline{Y'} = \overline{AB \ldots N}$$

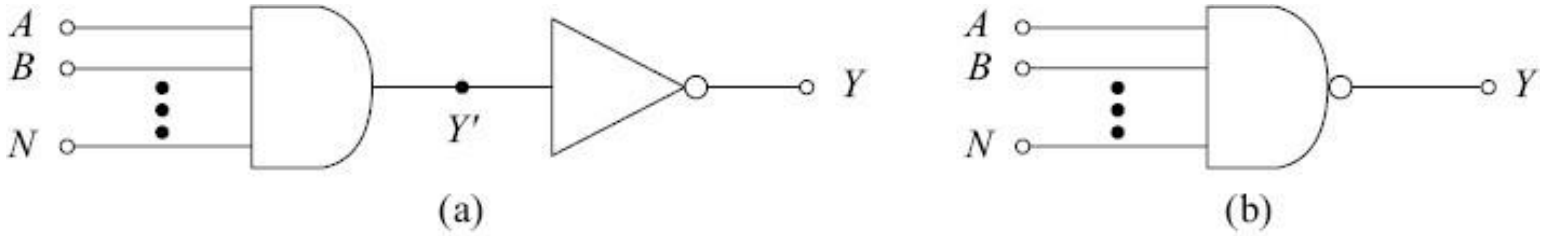


Fig. 1.10 (*a*) **NAND** *Operation as* **NOT–AND** *Operation,* (*b*) *Standard Symbol for the* **NAND** *Gate*

Table 1.4 *Truth Table of a 2-Input* **NAND** *Gate*

Inputs		Output
A	*B*	*Y*
0	0	1
0	1	1
1	0	1
1	1	0

The three basic logic operations, AND, OR, and NOT can be performed by using only NAND gates. These are given in Fig. 1.11.

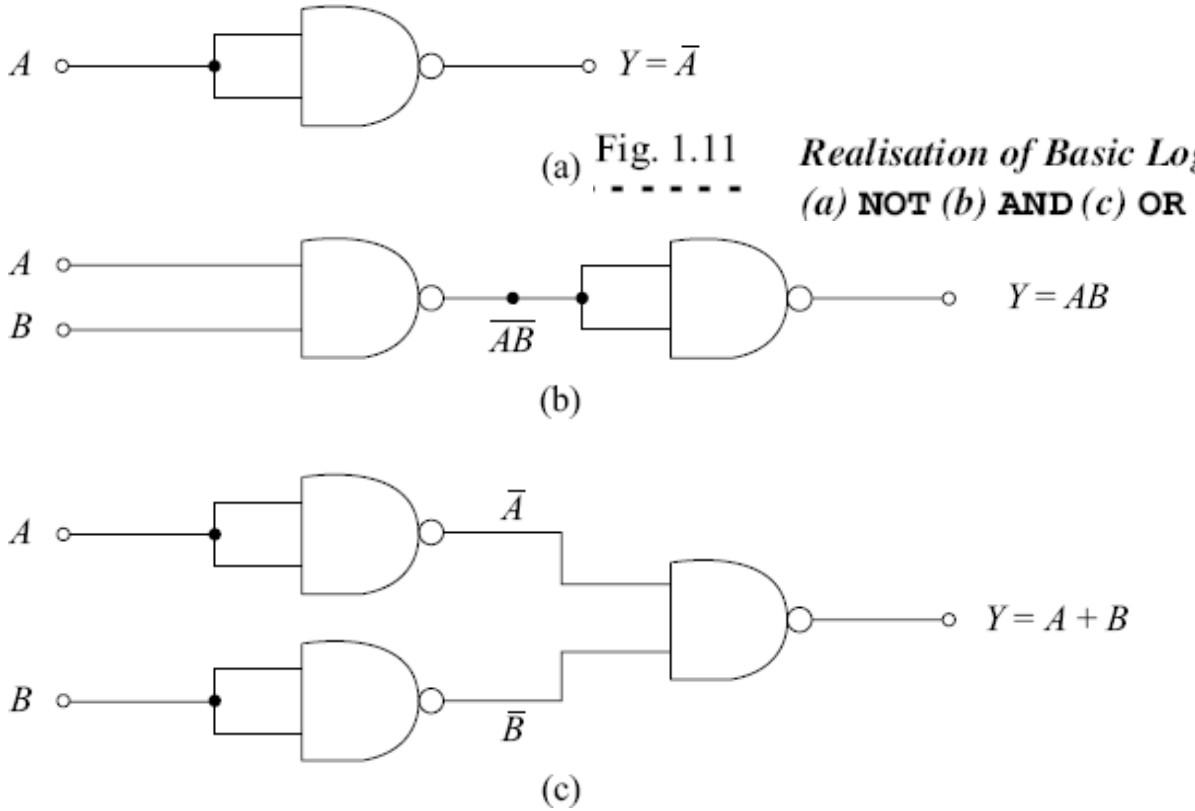


Fig. 1.11 *Realisation of Basic Logic Operations Using* **NAND** *Gates (a)* **NOT** *(b)* **AND** *(c)* **OR**

Example 1.6

If the voltage waveforms of Fig. 1.3 are applied at the inputs of a 2-input NAND gate, find the output waveform.

Solution

The output waveform will be inverse of the output waveform *Y* of Fig. 1.4.

NOR The NOT–OR operation is known as the NOR operation.

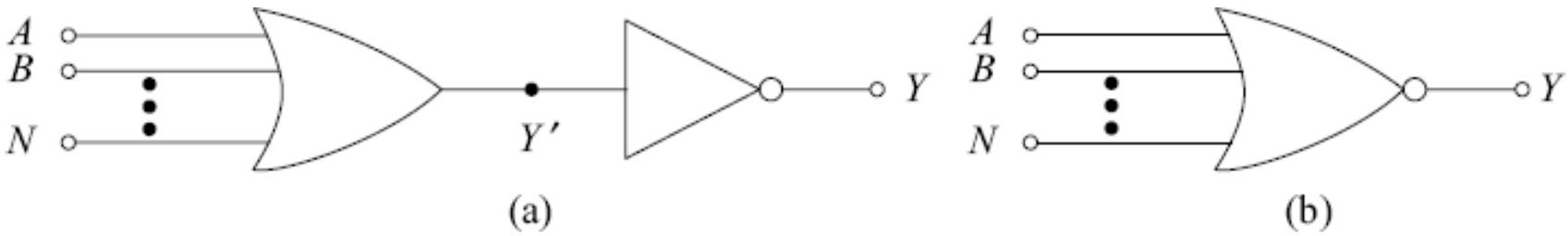
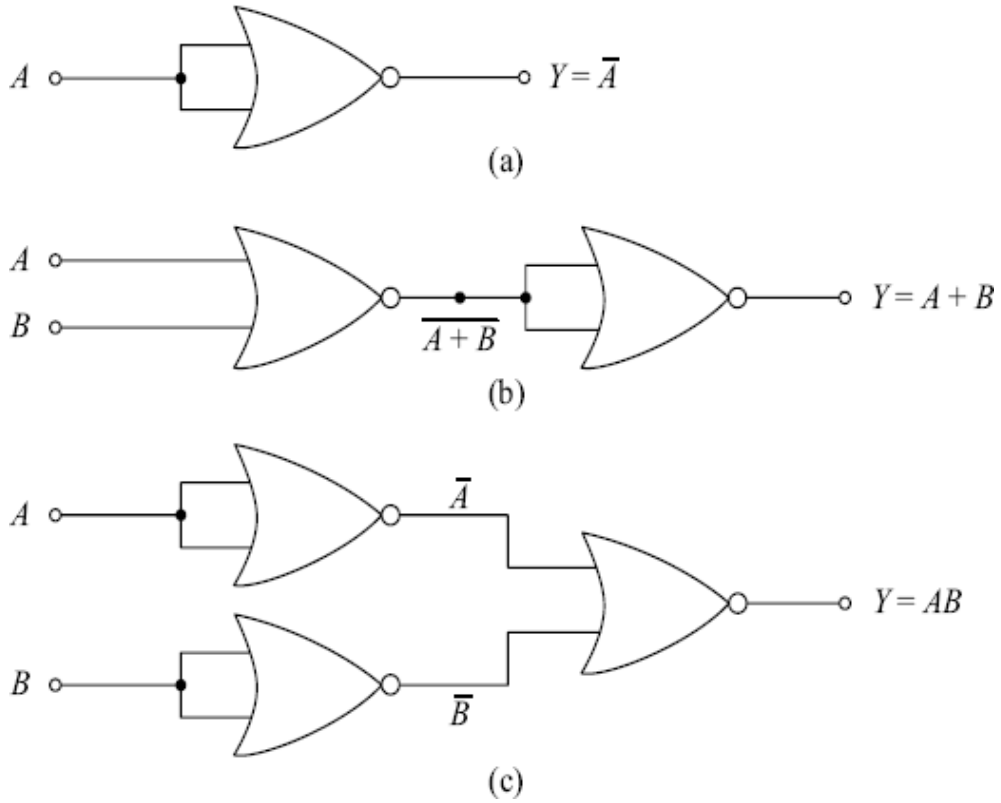


Fig. 1.13 *(a) NOR Operation as NOT–OR Operation*
(b) Standard Symbol for the NOR Gate

Table 1.5 *Truth Table of a 2-Input NOR Gate*

Inputs		Output
A	*B*	*Y*
0	0	1
0	1	0
1	0	0
1	1	0

The three basic logic operations, AND, OR, and NOT can be performed by using only the NOR gates. These are given in Fig. 1.14.



Example 1.8

If the voltage waveforms of Fig. 1.3 are applied at the inputs of a 2-input NOR gate, determine the output waveform.

Solution

The output waveform will be inverse of the output waveform of Fig. 1.6.

EXCLUSIVE-OR

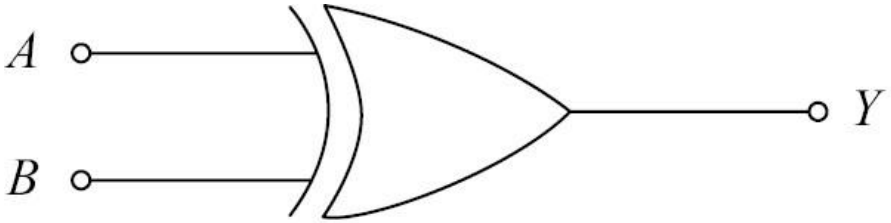


Fig. 1.15 ***Standard Symbol for* EX-OR *Gate***

$$Y = A \text{ EX-OR } B = A \oplus B$$

Table 1.6 ***Truth Table of* EX-OR *Gate***

Inputs		Output
A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE-NOR

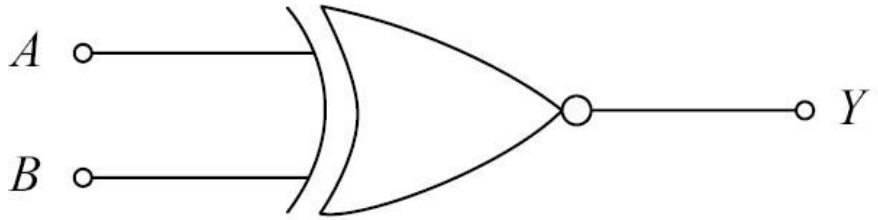


Fig. 1.17 *Standard Symbol for* **EX-NOR** *Gate*

$$Y = A \text{ EX-NOR } B = \overline{A \text{ EX-OR } B} = \overline{A \oplus B} = A \odot B$$

Table 1.7 *Truth Table of* **EX-NOR** *Gate*

Inputs		Output
A	*B*	*Y*
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebraic Theorems

Theorem No.	Theorem
1.1	$A + 0 = A$
1.2	$A \cdot 1 = A$
1.3	$A + 1 = 1$
1.4	$A \cdot 0 = 0$
1.5	$A + A = A$
1.6	$A \cdot A = A$
1.7	$A + \overline{A} = 1$
1.8	$A \cdot \overline{A} = 0$
1.9	$A \cdot (B + C) = AB + AC$
1.10	$A + BC = (A + B)\,(A + C)$
1.11	$A + AB = A$

(*Continued*)

Theorem No.	Theorem
1.12	$A\,(A + B) = A$
1.13	$A + \overline{A}B = (A + B)$
1.14	$A(\overline{A} + B) = AB$
1.15	$AB + A\overline{B} = A$
1.16	$(A + B) \cdot (A + \overline{B}) = A$
1.17	$AB + \overline{A}C = (A + C)\,(\overline{A} + B)$
1.18	$(A + B)\,(\overline{A} + C) = AC + \overline{A}B$
1.19	$AB + \overline{A}C + BC = AB + \overline{A}C$
1.20	$(A + B)\,(\overline{A} + C)\,(B + C) = (A + B)\,(\overline{A} + C)$
1.21	$\overline{A \cdot B \cdot C \cdot \ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$
1.22	$\overline{A + B + C + \ldots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \ldots$

Combinational Logic Design

Basically, Digital circuits are divided into two broad categories

1. Combinational circuits, and
2. Sequential circuits

In combinational circuits, the output at any instant of time depend upon the inputs present at that instant of time.This means there is no memory in these circuits.There are other types of circuits in which the outputs at any instant of time depend upon the present inputs as well as past inputs/outputs.This means that there are elements used to store past information.these elements are known as memory.Such circuits are known as sequential circuits.

The design requirements of combinational circuits may be specified in one of the following ways:

1. A set of statements,
2. Boolean expression,and
3. Truth table

The following methods can be used to simplify the boolean functions:

1.Algebraic method
2.Karnaugh-map technique,
3.Quine-McCluskey method, and
4.Variable entered mapping (VEM) technique

KARNAUGH MAP REPRESENTATION OF LOGIC FUNCTIONS

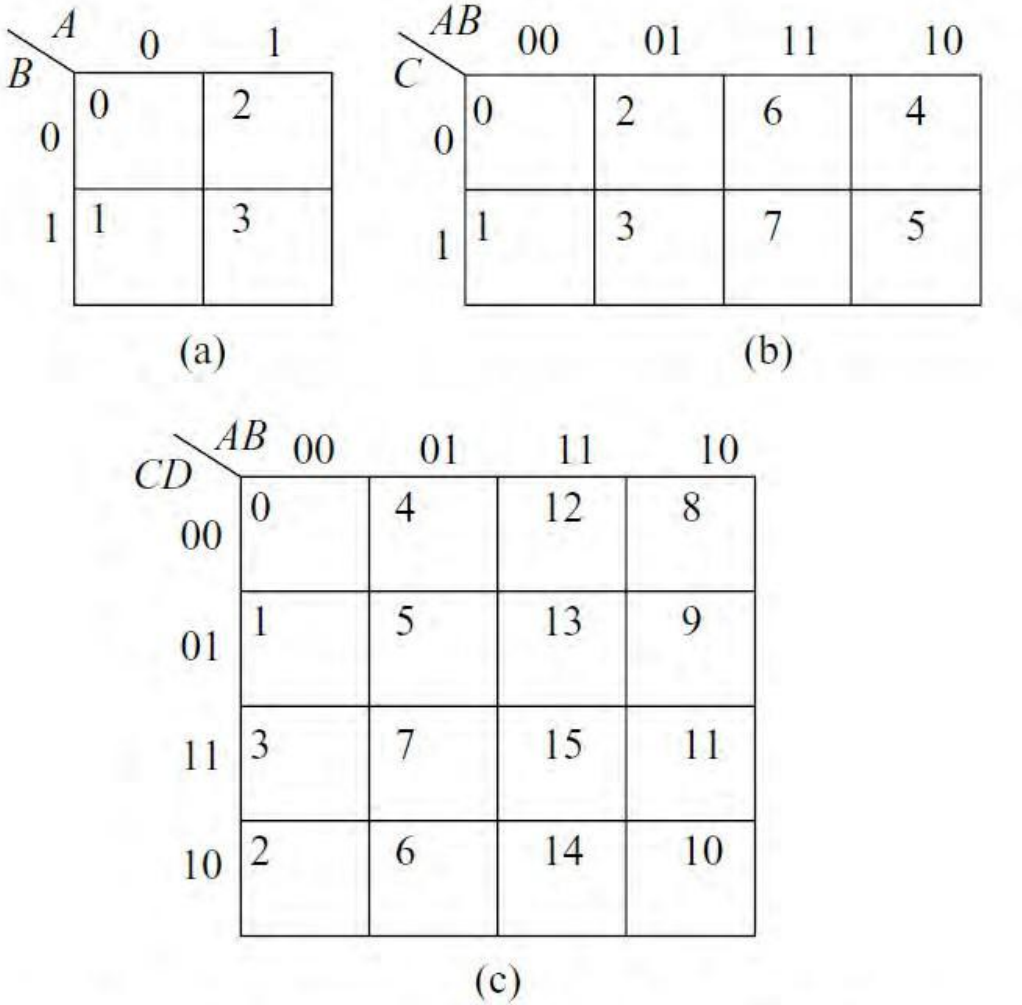


Fig. 5.5 ***Karnaugh-maps (a) Two-variable (b) Three-variable (c) Four-variable***

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

(a)

B \ A	0	1
0	$A + B$	$\overline{A} + B$
1	$A + \overline{B}$	$\overline{A} + \overline{B}$

(b)

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

(c)

C \ AB	00	01	11	10
0	$A + B + C$	$A + \overline{B} + C$	$\overline{A} + \overline{B} + C$	$\overline{A} + B + C$
1	$A + B + \overline{C}$	$A + \overline{B} + \overline{C}$	$\overline{A} + \overline{B} + \overline{C}$	$\overline{A} + B + \overline{C}$

(d)

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

(e)

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

(f)

Maxterm/Minterm Corresponding to Each Cell of K-maps

Design Examples

Arithmetic Circuits

Truth Table of Half-adder

Inputs		Outputs	
A	*B*	*S*	*C*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, we obtain the logical expressions for *S* and *C* outputs as

$S = \bar{A}B + A\bar{B} = A \oplus B$, $C = AB$

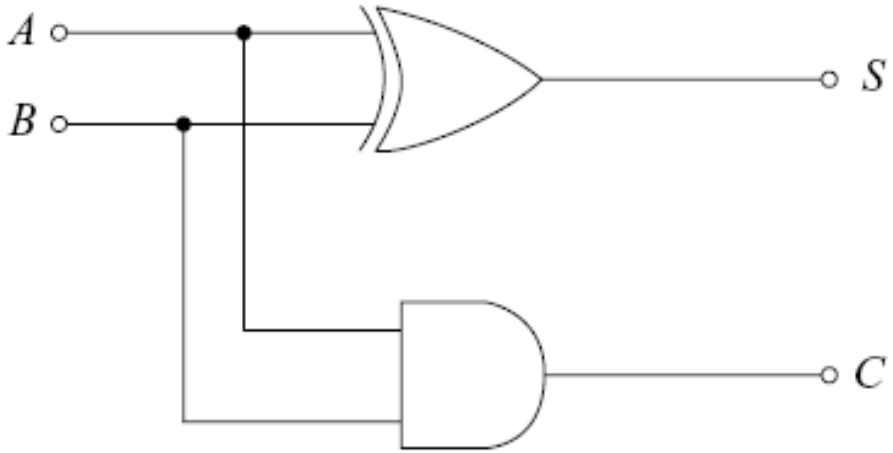


Fig. 5.19 *Realisation of an Half-adder*

Full-adder

$$S_n = \overline{A}_n\ B_n\ \overline{C}_{n-1} + \overline{A}_n\ \overline{B}_n\ C_{n-1} + A_n\ \overline{B}_n\ \overline{C}_{n-1} + A_n\ B_n\ C_{n-1}$$

$$C_n = A_n\ B_n + B_n\ C_{n-1} + A_n\ C_{n-1}$$

Truth Table of a Full-adder

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_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

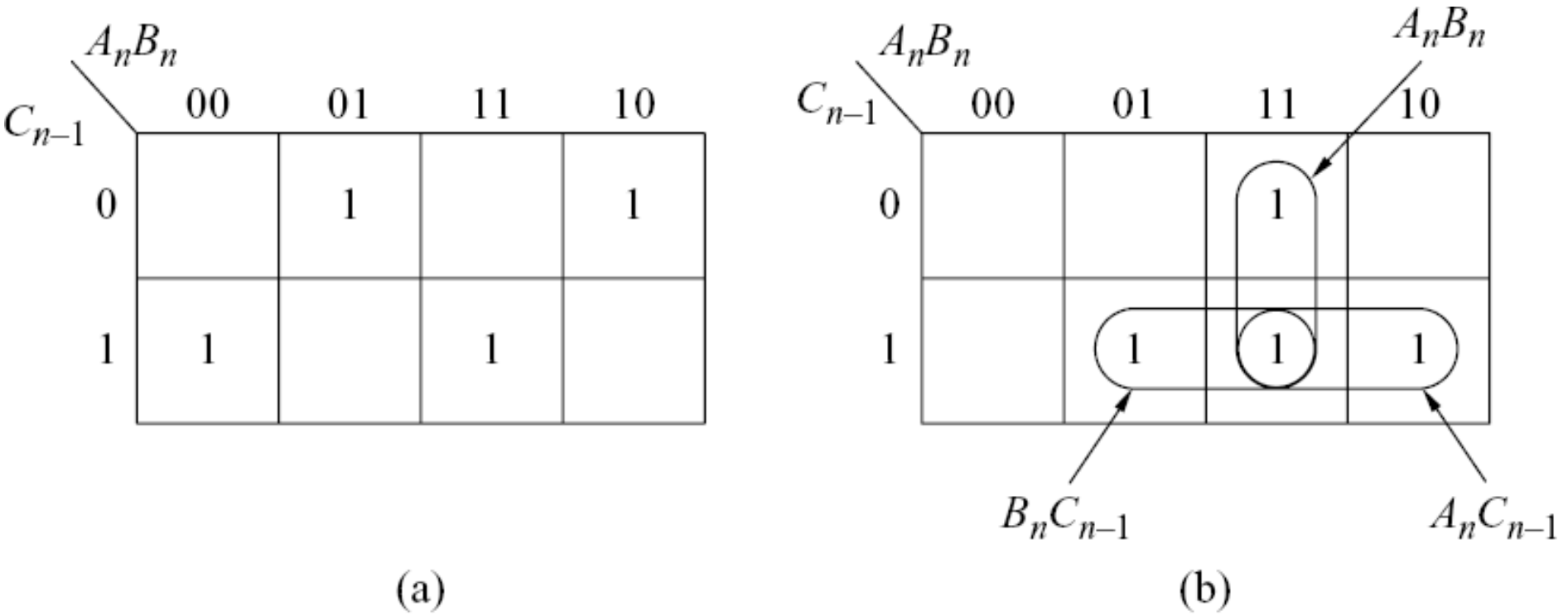


Fig. 5.20 ***K-maps for (a) S_n (b) C_n***

BCD-to-7-Segment Decoder

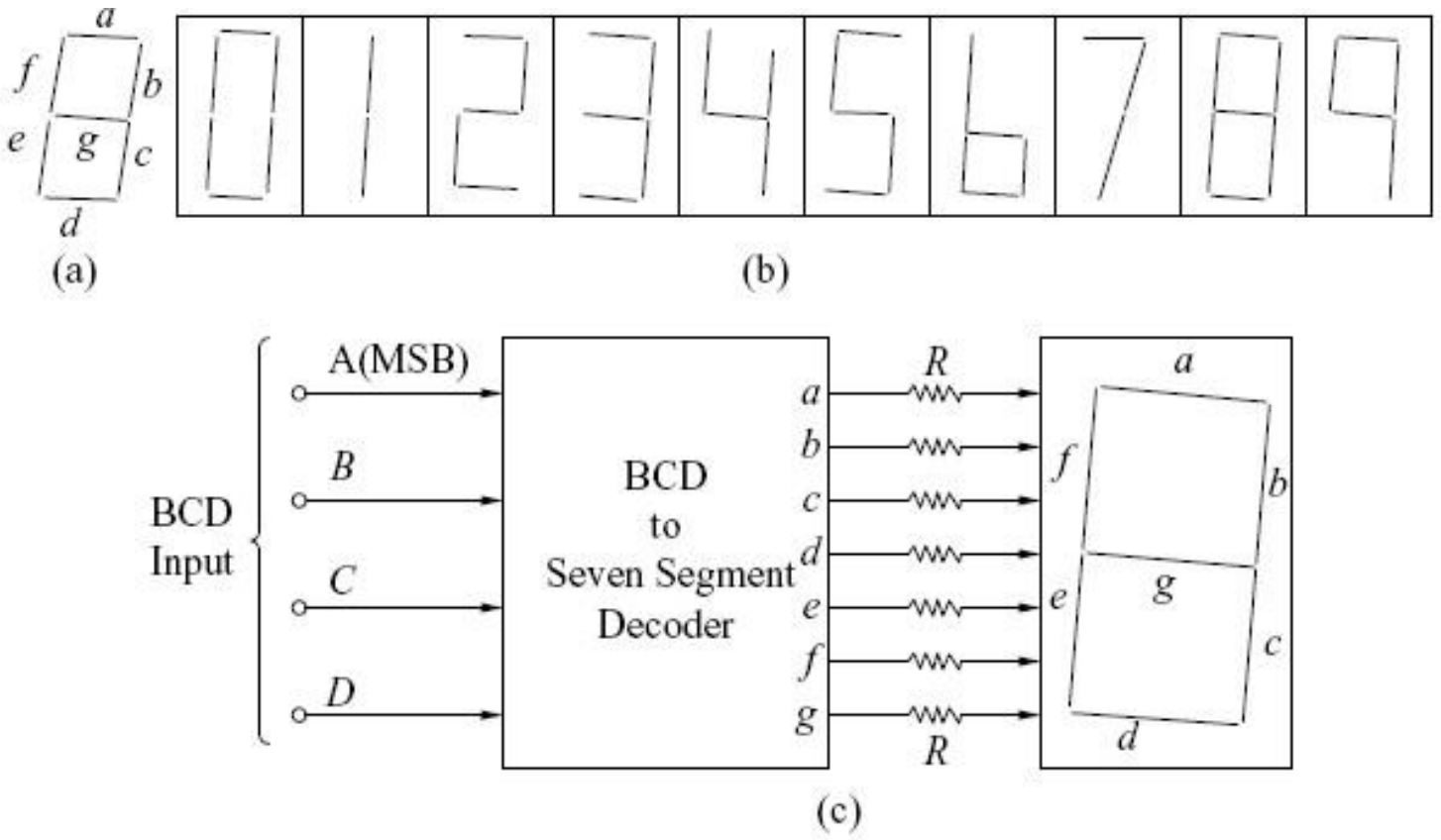


Fig. 5.24 *(a) 7-segment Display (b) Display of Numerals (c) Display System*

Truth Table of BCD-to-7 Segment Decoder

Decimal digit displayed	Inputs				Outputs						
	A	*B*	*C*	*D*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

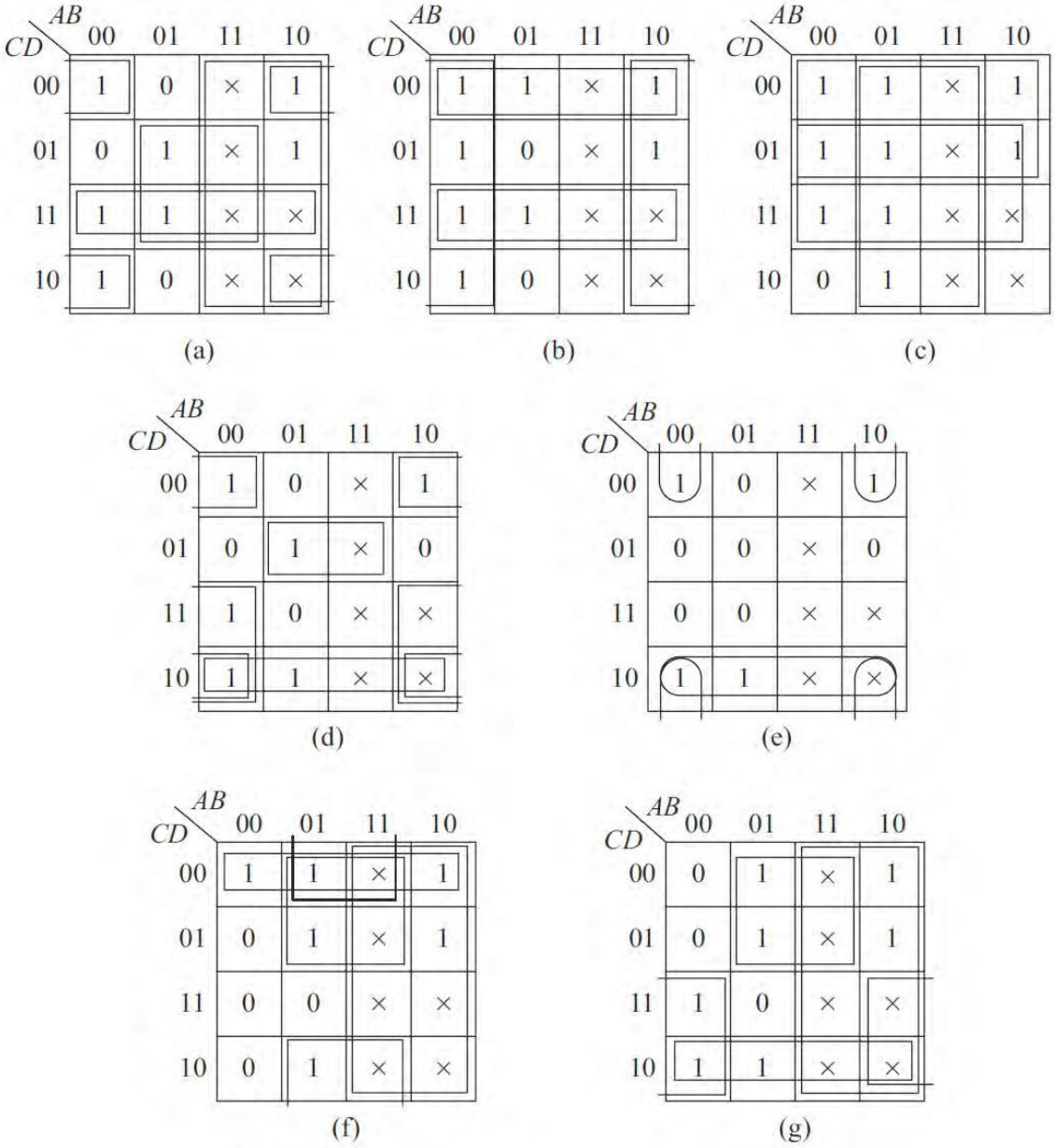


Fig. 5.25 ***K-maps of Table 5.15***

$$a = \overline{B}\overline{D} + BD + CD + A$$

$$b = \overline{B} + \overline{C}\overline{D} + CD$$

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

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

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

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

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

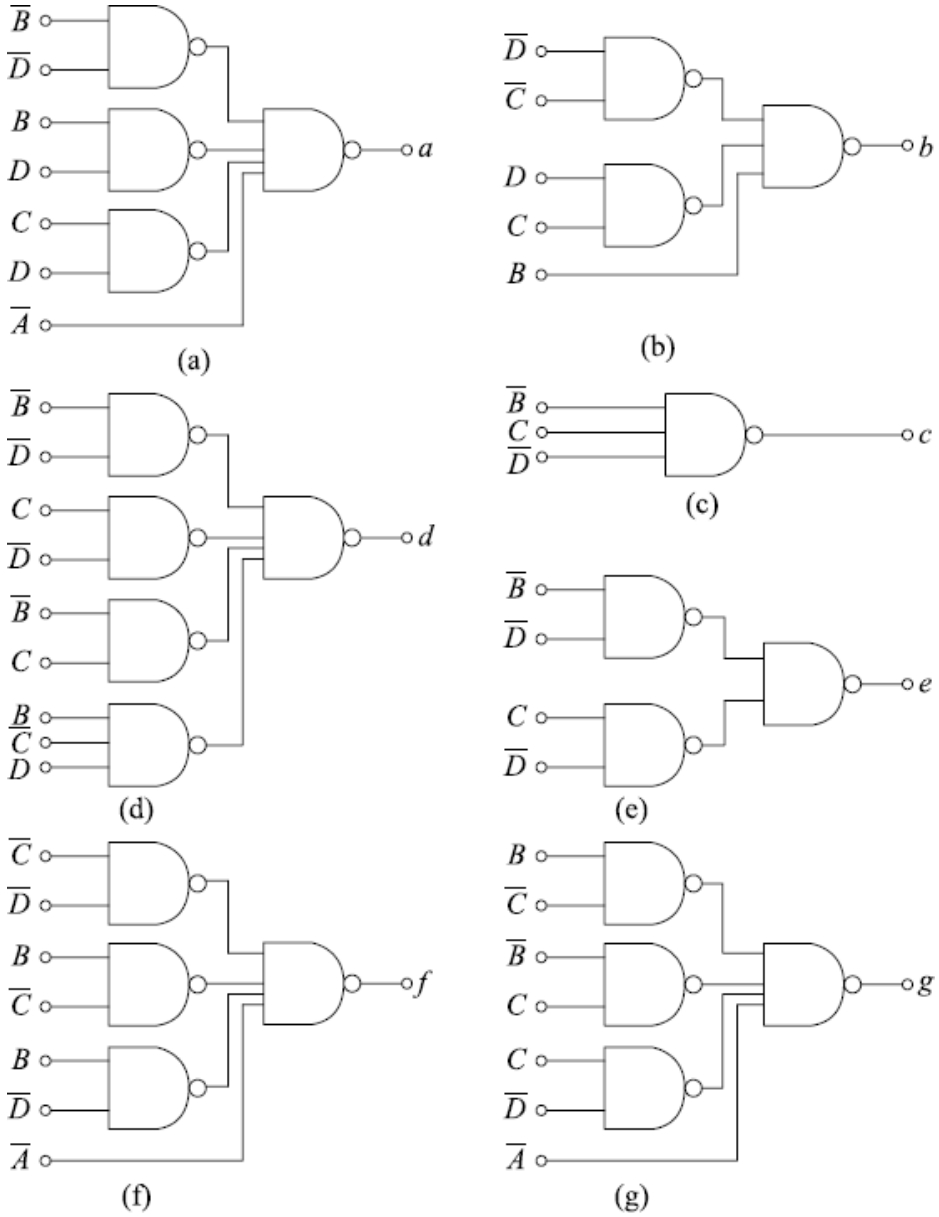


Fig. 5.26 NAND ***Gate Realisations of Eqs (5.38) to (5.44)***

Encoder

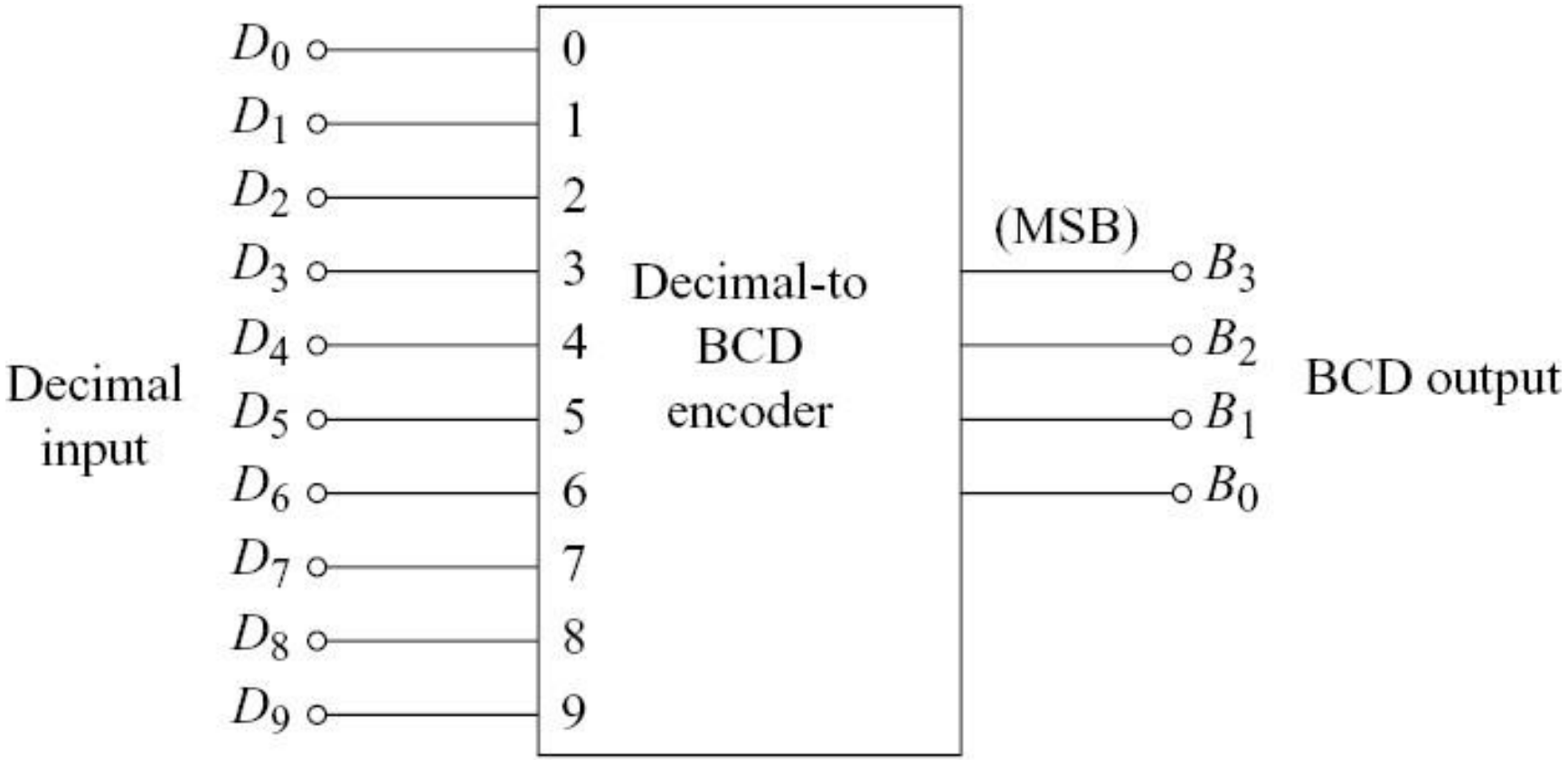


Fig. 5.27 ***Block Diagram of Decimal-to-BCD Encoder***

Truth Table of Decimal-to-BCD Encoder

Input	Output			
Decimal digit	B_3	B_2	B_1	B_0
D_0	0	0	0	0
D_1	0	0	0	1
D_2	0	0	1	0
D_3	0	0	1	1
D_4	0	1	0	0
D_5	0	1	0	1
D_6	0	1	1	0
D_7	0	1	1	1
D_8	1	0	0	0
D_9	1	0	0	1

From the truth table, we obtain the logic expressions for the outputs.

$$B_3 = D_8 + D_9$$

$$B_2 = D_4 + D_5 + D_6 + D_7$$

$$B_1 = D_2 + D_3 + D_6 + D_7$$

$$B_0 = D_1 + D_3 + D_5 + D_7 + D_9$$

Multiplexer

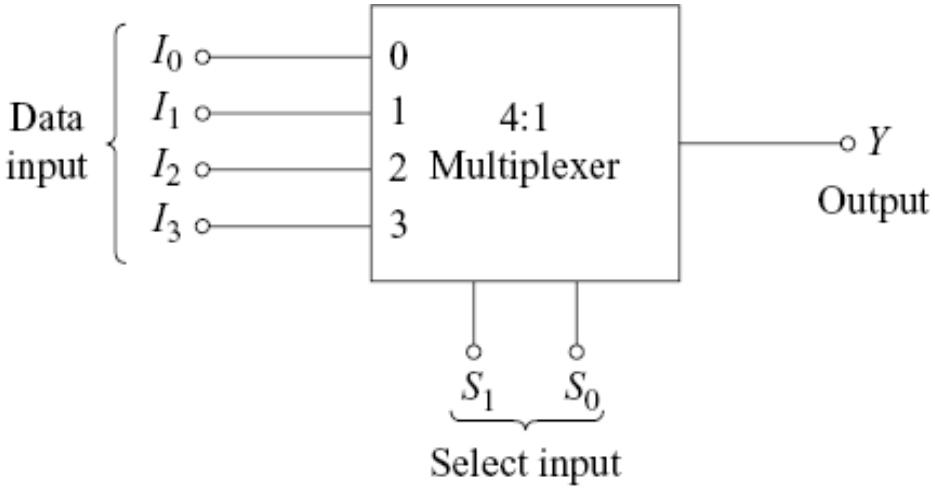


Fig. 5.29 ***Block Diagram of a 4:1 Multiplexer***

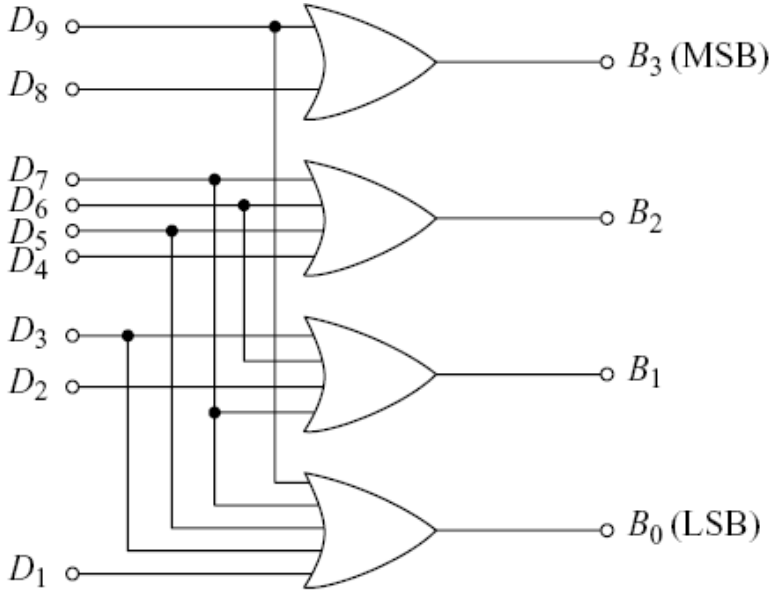


Fig. 5.28 ***Implementation of Decimal-to-BCD Encoder***

Truth Table of a 4:1 Multiplexer

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

$$Y = \overline{S}_0 \cdot \overline{S}_1 \cdot I_0 + S_0 \cdot \overline{S}_1 \cdot I_1 + \overline{S}_0 \cdot S_1 \cdot I_2 + S_0 \cdot S_1 \cdot I_3$$

Its realisation using NAND gates And inverters is shown in Fig.5.30.

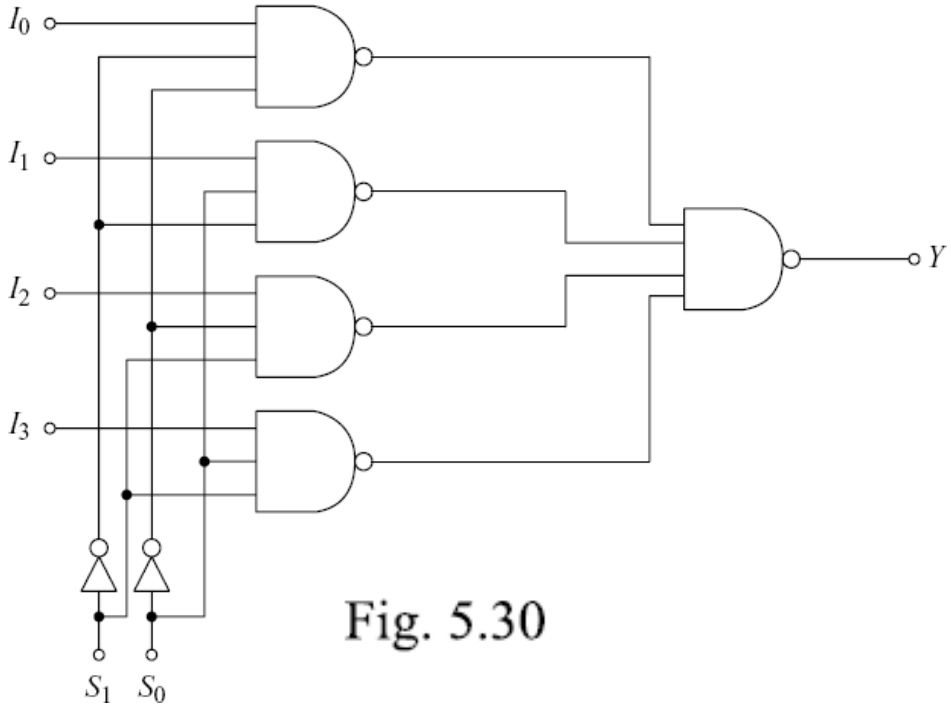


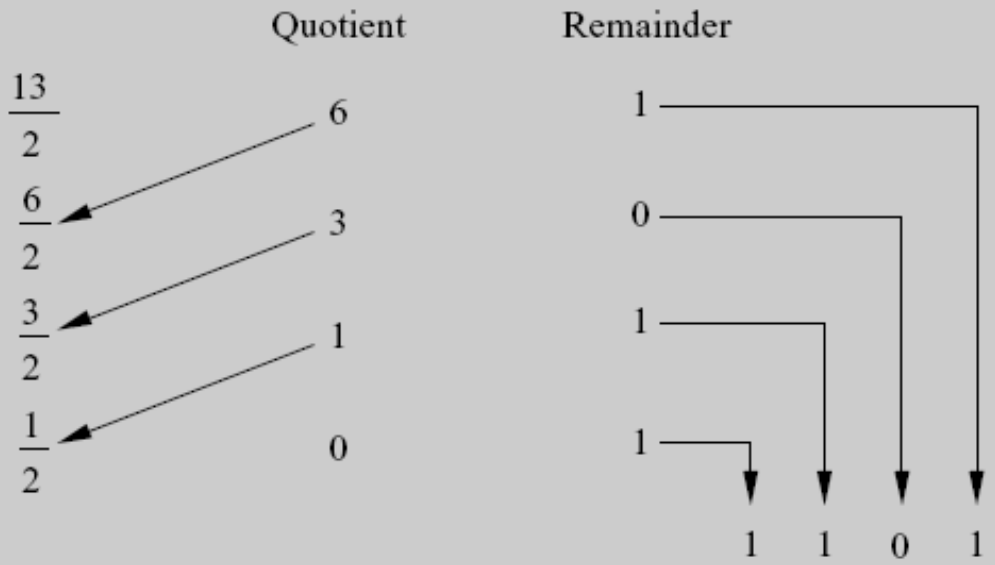
Fig. 5.30

Decimal-to-Binary Conversion

Example 2.4

Convert $(13)_{10}$ to an equivalent base-2 number.

Solution

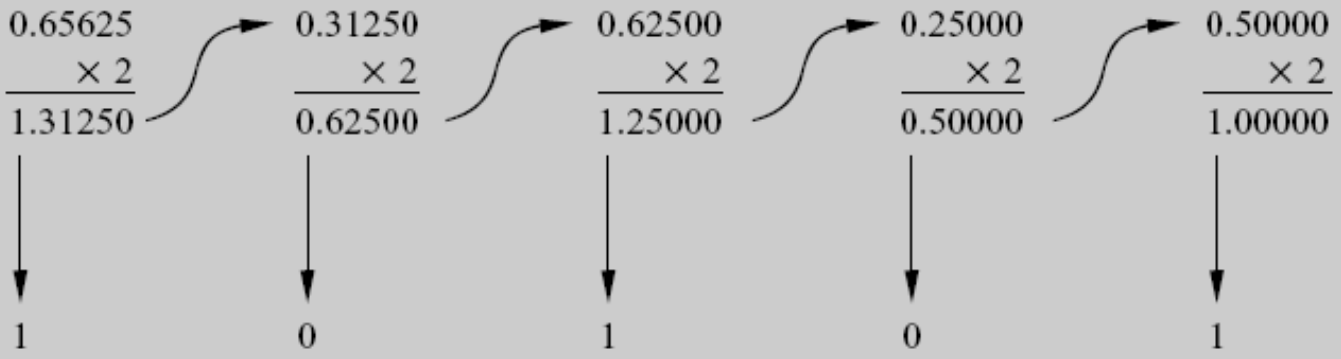


Thus, $(13)_{10} = (1101)_2$

Example 2.5

Convert $(0.65625)_{10}$ to an equivalent base-2 number.

Solution



Thus, $(0.65625)_{10} = (0.10101)_2$

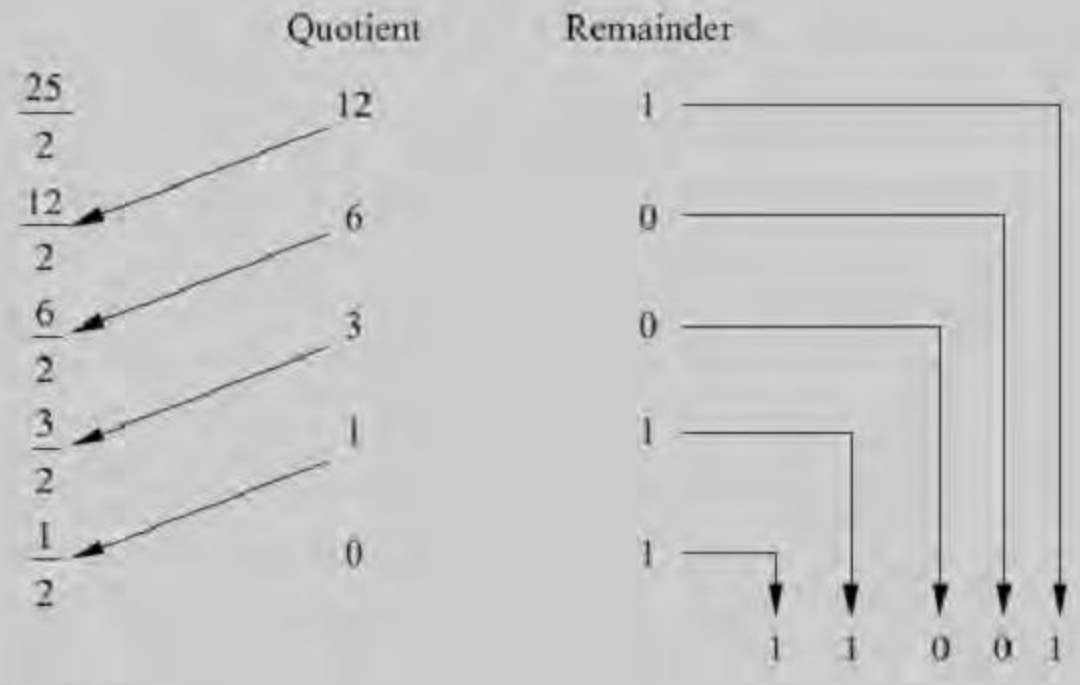
Example 2.6

Express the following decimal numbers in the binary form:

(a) 25.5 (b) 10.625 (c) 0.6875

Solution

(a) *Integer part*



Therefore, $(25)_{10} = (11001)_2$

Fractional part

$$\begin{array}{r} 0.5 \\ \times\ 2 \\ \hline 1.0 \end{array}$$

$\downarrow$

1

i.e., $(0.5)_{10} = (0.1)_2$

Therefore, $(25.5)_{10} = (11001.1)_2$

(b) *Integer part* $(10)_{10} = (1010)_2$

Fractional part

$$\begin{array}{r} 0.625 \\ \times\ 2 \\ \hline 1.250 \end{array} \quad \rightarrow \quad \begin{array}{r} 0.250 \\ \times\ 2 \\ \hline 0.500 \end{array} \quad \rightarrow \quad \begin{array}{r} 0.500 \\ \times\ 2 \\ \hline 1.000 \end{array}$$

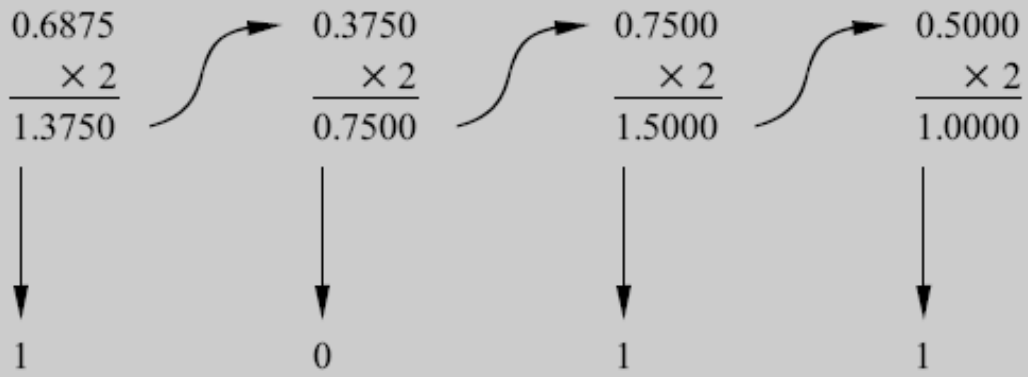
$\downarrow$ $\qquad$ $\downarrow$ $\qquad$ $\downarrow$

1 $\qquad$ 0 $\qquad$ 1

i.e., $(0.625)_{10} = (0.101)_2$

Therefore, $(10.625)_{10} = (1010.101)_2$

(c)



Therefore, $(0.6875)_{10} = (0.1011)_2$

BINARY ARITHMETIC

Arithmetic operations such as addition, subtraction, multiplication, and division are Performed similar to decimal arithmetic.

Binary Addition

Rules of Binary Addition

Augend	Addend	Sum	Carry	Result
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Example 2.13

Add the binary numbers:

(i) 1011 and 1100 (ii) 0101 and 1111

Solution

(i)

	1	0	1	1
(+)	1	1	0	0
1	0	1	1	1

↑ carry

(ii)

	(1)	(1)	(1)	← carry
	0	1	0	1
(+)	1	1	1	1
1	0	1	0	0

↑ carry

Binary Subtraction

Rules of Binary Subtraction

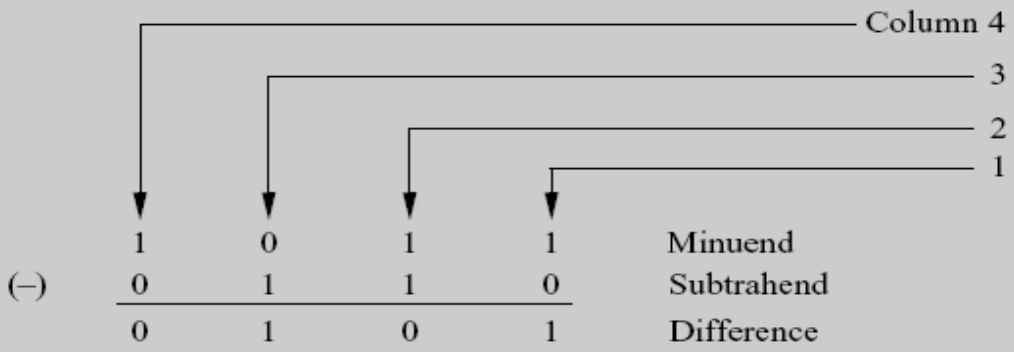
Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Example 2.15

Perform the following subtraction:

```
  1 0 1 1
– 0 1 1 0
```

Solution



Here, in columns 1 and 2, borrow = 0 and in column 3 it is 1. Therefore, in column 4 first subtract 0 from 1 and from this result obtained subtract the borrow bit.

Binary Multiplication

Binary multiplication is similar to decimal multiplication.

Example 2.16

Multiply 1001 by 1101.

Solution

1 0 0 1		Multiplicand
× 1 1 0 1		Multiplier
1 0 0 1	I	
0 0 0 0	II	Partial Products
1 0 0 1	III	
1 0 0 1	IV	
1 1 1 0 1 0 1		Final Product

In a digital circuit, the multiplication operation is performed by repeated additions of all partial products to obtain the final product.

Binary Division

Binary division is obtained using the same procedure as decimal division.

Example 2.17

Divide 1 1 1 0 1 0 1 by 1 0 0 1.

Solution

```
                    1101        ← Quotient
Divisor → 1001 ) 1110101        ← Dividend
                 1001
                 ----
                 1011
                 1001
                 ----
                 001001
                    1001
                    ----
                    0000
```

Ans: 1101

OCTAL NUMBER SYSTEM

The Number system with base(or radix) eight is known as the octal number system.

Octal-to-Decimal Conversion

Example 2.20

Convert $(6327.4051)_8$ into its equivalent decimal number.

Solution

Using the weights given in Table 2.1, we obtain

$$\begin{aligned}(6327.4051)_8 &= 6\times 8^3 + 3\times 8^2 + 2\times 8^1 + 7\times 8^0 + 4\times 8^{-1} \\ &\quad + 0\times 8^{-2} + 5\times 8^{-3} + 1\times 8^{-4} \\ &= 3072 + 192 + 16 + 7 + \frac{4}{8} + 0 + \frac{5}{512} + \frac{1}{4096} \\ &= (3287.5100098)_{10}\end{aligned}$$

Thus, $(6327.4051)_8 = (3287.5100098)_{10}$

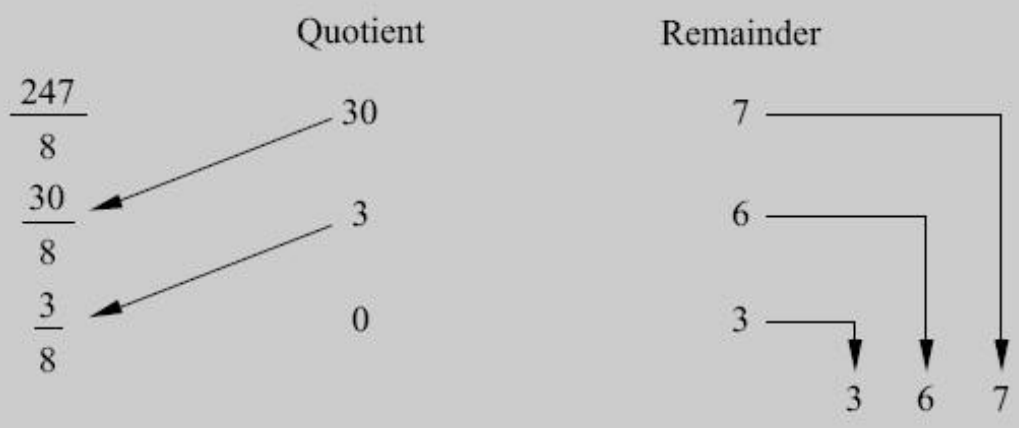
Decimal-to-Octal Conversion

Example 2.21

(a) Convert $(247)_{10}$ into octal
(b) Convert $(0.6875)_{10}$ into octal
(c) Convert $(3287.5100098)_{10}$ into octal

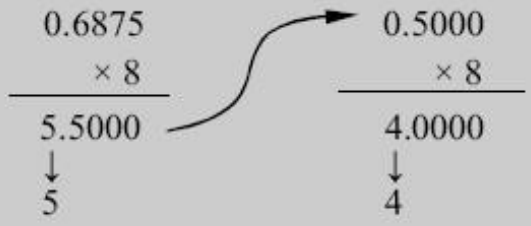
Solution

(a)



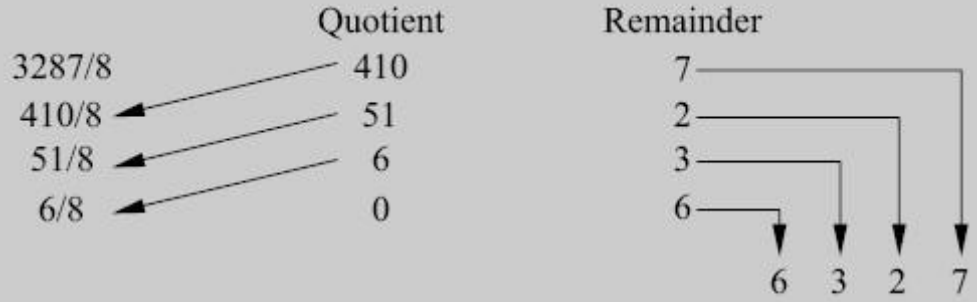
Thus, $(247)_{10} = (367)_8$

(b)



Thus, $(0.6875)_{10} = (0.54)_8$

(c) *Integer part:*



Thus, $(3287)_{10} = (6327)_8$

Fractional part:

$$0.5100098 \times 8 = 4.0800784 \rightarrow 4$$

$$0.0800784 \times 8 = 0.6406272 \rightarrow 0$$

$$0.6406272 \times 8 = 5.1250176 \rightarrow 5$$

$$0.1250176 \times 8 = 1.0001408 \rightarrow 1$$

Thus $(0.5100098)_{10} \approx (0.4051)_8$

Therefore, $(3287.5100098)_{10} = (6327.4051)_8$

Octal-to-Binary Conversion

Binary and Decimal Equivalents of Octal Numbers

Octal	Decimal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
10	8	001000
11	9	001001
12	10	001010
13	11	001011
14	12	001100
15	13	001101
16	14	001110
17	15	001111

Binary-to-Octal Conversion

Example 2.25

Convert the following binary numbers to octal numbers

(a) 11001110001.000101111001
(b) 1011011110.11001010011
(c) 111110001.10011001101

Solution

(a) 011 001 110 001.000 101 111 001 = $(3161.0571)_8$
(b) 001 011 011 110.110 010 100 110 = $(1336.6246)_8$
(c) 111 110 001.100 110 011 010 = $(761.4632)_8$

HEXADECIMAL NUMBER SYSTEM

The Number System with base (or radix) 16 is known as hexadecimal number system.

Binary and Decimal Equivalents of Hexadecimal Numbers

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadecimal-to-Decimal Conversion

Example 2.28

Obtain decimal equivalent of hexadecimal number $(3A.2F)_{16}$

Solution

Using Eq. (2.1),

$$(3A.2F)_{16} = 3 \times 16^1 + 10 \times 16^0 + 2 \times 16^{-1} + 15 \times 16^{-2}$$
$$= 48 + 10 + \frac{2}{16} + \frac{15}{16^2}$$
$$= (58.1836)_{10}$$

Decimal-to-Hexadecimal Conversion

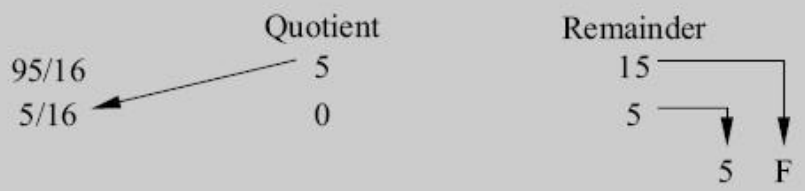
Example 2.29

Convert the following decimal numbers into hexadecimal numbers.

(a) 95.5 (b) 675.625

Solution

(a) *Integer part*



Thus, $(95)_{10} = (5F)_{16}$

Fractional part

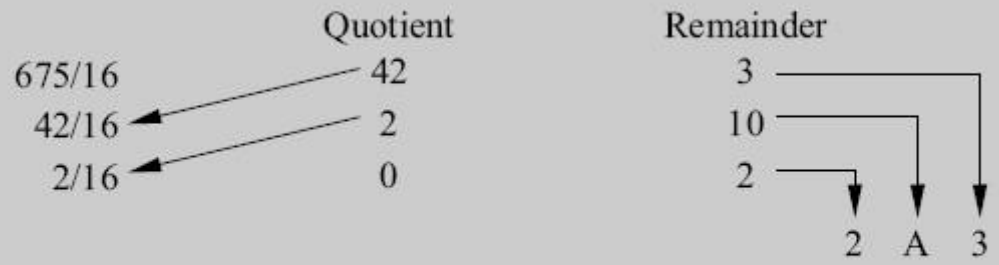
$$\begin{array}{r} 0.5 \\ \times\ 16 \\ \hline 8.0 \end{array}$$

$\downarrow$

8

Thus, $(0.5)_{10} = (0.8)_{16}$
Therefore, $(95.5)_{10} = (5F.8)_{16}$

(b) *Integer part*



Thus, $(675)_{10} = (2A3)_{16}$

Fractional part

$$\begin{array}{r} 0.625 \\ \times\ 16 \\ \hline 10.000 \end{array}$$

$\downarrow$

A

Thus, $(0.625)_{10} = (0.A)_{16}$
Therefore, $(675.625)_{10} = (2A3.A)_{16}$

Hexadecimal-to-Binary Conversion

Example 2.30

Convert $(2F9A)_{16}$ to equivalent binary number.

Solution

Using Table 2.7, find the binary equivalent of each hex digit.

$$(2F9A)_{16} = (0010\ 1111\ 1001\ 1010)_2$$
$$= (0010111110011010)_2$$

Binary-to-Hexadecimal Conversion

Example 2.31

Convert the following binary numbers to their equivalent hex numbers.

(a) 10100110101111
(b) 0.00011110101101

Solution

(a) $(10100110101111)_2 = \underbrace{0010}_{2}\ \underbrace{1001}_{9}\ \underbrace{1010}_{A}\ \underbrace{1111}_{F}$

$\therefore\ (10100110101111)_2 = (29AF)_{16}$

(b) $(0.00011110101101)_2 = \underbrace{0.0001}_{1}\ \underbrace{1110}_{E}\ \underbrace{1011}_{B}\ \underbrace{0100}_{4}$

$\therefore (0.00011110101101)_2 = (0.1EB4)_{16}$