smoothly from one value to the next, like gradually turning a light dimmer switch up or down. The diagram below shows an analogue signal that changes with time.

Devices exist which convert from analogue signals to digital signals and vice-versa. Although most signals that we can perceive in the real world are analogue (sound levels in speech, light levels in vision, etc.) more and more signals are being stored in digital format (audio CDs, DVDs, digital audio cassettes etc.) because of one major advantages it offers: Digital signals are potentially immortal!

The problem with all signals is that they acquire *noise* - that horrible crackling that seems to appear from nowhere - as they get older. Noise isn't just something that you can hear - the fuzz that appears on old video recordings also qualifies as noise. In general, noise is any unwanted change to a signal that tends to corrupt it. Digital signals can easily be recognized even among all that noise. They can be put through an electronic device that "cleans them up" and restores the original digital signal in perfect condition. It takes a great deal of noise to be added to a digital signal for a 0 to be mistaken for a 1 or vice-versa!

Although it is possible for an analogue signal to be cleaned up to a certain extent, you can never get back a perfect copy of the original signal. Because of this analogue signals inevitably build up noise as they get older, and this is the reason old records sound scratchy and that old video recording are unwatchable.

One example that is often quoted in the war between analogue and digital is the cleaning of the Sistine Chapel in Venice. During the 1980s the famous painting by Michaelangelo on the ceiling of the Sistine Chapel was cleaned and centuries of grime was removed. However the restorers could not be certain that they are putting the colours back to exactly that Michaelangelo had intended. If the ceiling had been coped in to the form of a digital signal in the early sixteenth century when it was created, the restorers would have know exactly what the original colours should have been.

1.2 NUMBER SYSTEM

A nybble (also spelled nibble) is a binary number consisting of four bits. A byte, or octet is a binary number consisting of eight bits. There are other systems, which we will look at briefly.

- Decimal
- Binary
- Octal
- Hexadecimal

1.2.1 Decimal System

The decimal system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Using these symbols as digits of a number, we can express any quantity. The decimal system is also called the base-10 system because it has 10 digits. Even though the decimal system has only 10 symbols, any number of any magnitude can be expressed by using our system of positional weighting.

Table 1.1 Decimal System

10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}
1000	100	10	•	0.1	0.01	0.001
Most Significant Digit			Decimal point			Least Significant Digit

Examples

- 3.14_{10}
- 52_{10}
- 1024_{10}
- 64000_{10}

1.2.2 Binary System

In the binary system, there are only two symbols or possible digit values are 0 and 1. This base-2 system can be used to represent any quantity that can be represented in decimal or other base system.

Table 1.2 Binary System

2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
4	2	1	•	0.5	0.25	0.125
Most Significant Digit			Binary point			Least Significant Digit

In digital systems, the information that is being processed is usually presented in binary form. Binary quantities can be represented by any device that has only two operating states or possible conditions. For example, A switch is only open or closed. We arbitrarily (as we define them) let an open switch represent binary 0 and a closed switch represent binary 1. Thus we can represent any binary number by using series of switches. The binary number system is the most important one in digital systems, but several others are also important. The decimal system is important because it is universally used to represent quantities outside a digital system. This means that there will be situations where decimal values have to be converted to binary values before they are entered into the digital systems. In additional to binary and decimal, two other number systems find wide-spread applications in digital systems. The octal (base-8) and hexadecimal (base-16) number systems are both used for the same purpose to provide an efficient means for representing large binary system.

1.2.3 Octal System

The octal number system has a base of eight, meaning that it has eight possible digits: 0, 1, 2, 3, 4, 5, 6, 7.

Table 1.3 Octal System

8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
64	8	1	•	1/8	1/64	1/512
Most Significant Digit			octal point			Least Significant Digit

Octal to Decimal Conversion

- $237_8 = 2 \times (8^2) + 3 \times (8^1) + 7 \times (8^0) = 159_{10}$
- $24.6_8 = 2 \times (8^1) + 4 \times (8^0) + 6 \times (8^{-1}) = 20.75_{10}$
- $11.1_8 = 1 \times (8^1) + 1 \times (8^0) + 1 \times (8^{-1}) = 9.125_{10}$
- $12.3_8 = 1 \times (8^1) + 2 \times (8^0) + 3 \times (8^{-1}) = 10.375_{10}$

1.2.4 Hexadecimal System

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters *A*, *B*, *C*, *D*, *E*, and *F* as the 16 digit symbols.

Table 1.4 Hexadecimal system

16^2	16^1	16^0		16^{-1}	16^{-2}	16^{-3}
256	16	1	•	1/16	1/256	1/2096
Most Significant Digit			Hexadecimal point			Least Significant Digit

Hexadecimal to Decimal Conversion

- $24.6_{16} = 2 \times (16^1) + 4 \times (16^0) + 6 \times (16^{-1}) = 36.375_{10}$
- $11.1_{16} = 1 \times (16^1) + 1 \times (16^0) + 1 \times (16^{-1}) = 17.0625_{10}$
- $12.3_{16} = 1 \times (16^1) + 2 \times (16^0) + 3 \times (16^{-1}) = 18.1875_{10}$

1.3 CODE CONVERSION

Converting from one code form to another code form is called code conversion, like converting from binary to decimal or converting from hexadecimal to decimal.

1.3.1 Binary-To-Decimal Conversion

Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.

Example

1. *Convert binary to decimal* 11011_2

$$= 2^4 + 2^3 + 0^1 + 2^1 + 2^0$$

$$= 16 + 8 + 0 + 2 + 1 = 27_{10}$$

2. *Convert binary to decimal* 10110101_2

$$= 2^7 + 0^6 + 2^5 + 2^4 + 0^3 + 2^2 + 0^1 + 2^0$$

$$= 128 + 0 + 32 + 16 + 0 + 4 + 0 + 1 = 181_{10}$$

It should be noticed that the method is to find the weights (i.e., powers of 2) for each bit position that contains a 1, and then add them up.

1.3.2 Decimal-To-Binary Conversion

There are 2 methods:

- Reverse of Binary-To-Decimal Method
- Repeat Division

1. Reverse of Binary-To-Decimal Method

Example: Convert decimal to binary 45_{10}

$$= 32 + 0 + 8 + 4 + 0 + 1$$

$$= 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 = 101101_2$$

2. Repeat Division-Convert decimal to binary

This method uses repeated division by 2.

Example: Convert decimal to binary 25_{10}

Division	Remainder	Binary
25/2	= 12 + remainder of 1	1 (Least Significant Bit)
12/2	= 6 + remainder of 0	0
6/2	= 3 + remainder of 0	0
3/2	= 1 + remainder of 1	1
1/2	= 0 + remainder of 1	1 (Most Significant Bit)
Result	25_{10}	$= 11001_2$

The Flow chart for repeated-division method is as follows

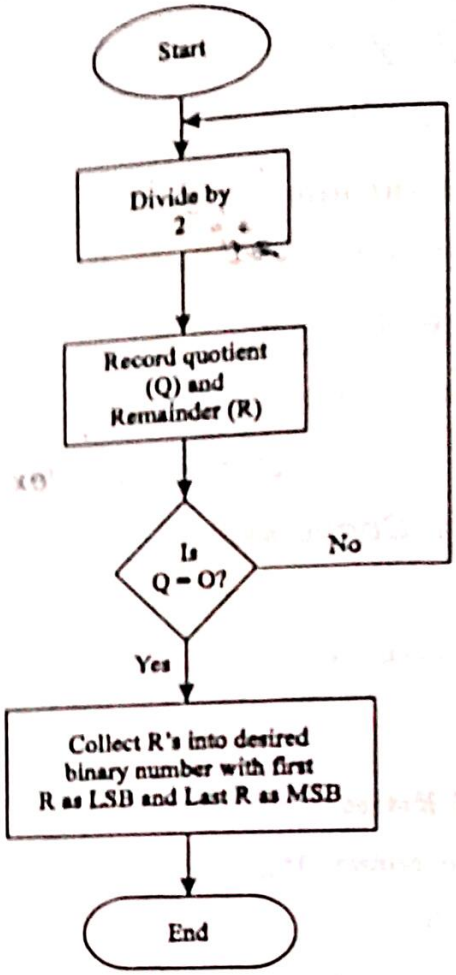


Fig. 1.1 Flowchart for repeated-division method

1.3.3 Binary-To-Octal / Octal-To-Binary Conversion

Octal Digit	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

Each Octal digit is represented by three binary digits.

Example

1. **For binary to decimal conversion,**

   $100111010_2 = (100)\,(111)\,(010)_2 = 472_8$

2. **Repeat Division-Convert decimal to octal**

   This method uses repeated division by 8.

Example

Convert 177_{10} to octal and binary.

Division	Remainder	Octal/Binary
177/8	= 12 + remainder of 1	1 (Least Significant Bit)
22/8	= 2 + remainder of 6	6
2/8	= 0 + remainder of 2	2 (Most Significant Bit)
Result	Octal	$= 261_8$
	Binary	010110001_2

1.3.4 Hexadecimal to Decimal/Decimal to Hexadecimal Conversion

Example

For hexadecimal to decimal conversion,

$2AF_{16} = 2 \times (16^2) + 10 \times (16^1) + 15 \times (16^0) = 687_{10}$

Repeat Division - Convert decimal to hexadecimal

This method uses repeated division by 16

Division	Remainder	Hexadecimal
378/16	= 23 + remainder of 10	A (Least Significant Bit)
23/16	= 1 + remainder of 7	7
1/16	= 0 + remainder of 1	1 (Most Significant Bit)
Result		$= 17A_{16}$
Binary		$(000101111010)_2$

1.3.5 Binary-To-Hexadecimal/Hexadecimal-To-Binary Conversion

Hexadecimal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal Digit	**8**	**9**	***A***	***B***	***C***	***D***	***E***	***F***
Binary Equivalent	1000	1001	1010	1011	1100	1101	1110	1111

Each Hexadecimal digit is represented by **four** bits of binary digit.

Example

For binary to hexadecimal conversion,

$1011\ 0010\ 1111_2 = (1011)\ (0010)\ (1111)_2 = B2F_{16}$

1.3.6. Octal-To-Hexadecimal/Hexadecimal-To-Octal Conversion

Procedure

- Convert Octal (Hexadecimal) to Binary first.
- Regroup the binary number by three bits per group starting from LSB if Octal is required.
- Regroup the binary number by four bits per group starting from LSB if Hexadecimal is required

Example

1. Convert $5A8_{16}$ to Octal.

$5A8_{16} = 0101\ 1010\ 1000_2$

$= 010\ 110\ 101\ 000$ (Binary)

$= 2\ 6\ 5\ 0_8$ (Octal)

2. Convert 2650_8 to hexadecimal

$2650_8 = 010\ 110\ 101\ 000_2$

$= 0101\ 1010\ 1000$

$= 5A8_{16}$

EXAMPLE 1

Convert Binary to Decimal (a) $1101 \cdot 10$ *(b)* $10101 \cdot 111$

(a) $1101 \cdot 10 = 2^3 \times 1 + 2^2 \times 1 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2}$

$= 8 + 4 + 0 + 1 + 0.5 = (13.5)_{10}$

(b) $10101 \cdot 111 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$

$= 16 + 0 + 4 + 0 + 1 + 0.5 + 0.25 + 0.125 = (21.875)_{10}$

EXAMPLE 2

Convert Octal to Decimal (a) $375 \cdot 25$ *(b)* $3725 \cdot 5672$

(a) $375 \cdot 20 = 3 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 + 2 \times 8^{-1} + 5 \times 2^{-2}$

$= (253 \cdot 328)_{10}$

(b) $3725 \cdot 5672 = 3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 + 5 \times 8^{-1} + 6 \times 8^{-2} + 7 \times 8^{-3} + 2 \times 8^{-4}$

$= (2005 \cdot 73)_{10}$

The second sum is the largest possible sum of two hex digits; the carry to the next position is 1. This shows that the sum of two hex digits will never produce a carry larger than 1. The second sum can be calculated as follows:

$$\begin{aligned} FH + FH &= 15_{10} + 15_{10} \\ &= 30_{10} \\ &= 16_{10} + 14_{10} \\ &= 10H + EH \\ &= 1EH \end{aligned}$$

The third sum shows that if there is a carry from a previous sum, the carry to the next bit will still be 1.

Hex Subtraction

There are two ways to subtract hex numbers. The first reverses the addition process in the previous section. The second is a complement form of subtraction.

The second subtraction method is a complement method, where, as in 2's complement subtractions, we add a negative number to subtract a positive number.

Calculate the 15's complement of a hex number by subtracting it from a number having the same number of digits, all Fs. Calculate the 16's complement by adding 1 to this number. This is the negated value of the number.

6.5 Numeric and Alphanumeric Codes

BCD Codes

BCD stands for **binary-coded decimal.** As the name implies, BCD is a system of writing decimal numbers with binary digits. There is more than one way to do this, as BCD is a *code*, not a positional number system. That is, the various positions of the bits do not necessarily represent increasing powers of a specified number base and are used to represent a number, (usually) not to mathematically manipulate a number.

Two commonly used BCD codes are 8421 code, where the bits for *each decimal digit* are weighted, and Excess-3 code, where each decimal digit is represented by a binary number that is 3 larger than the true binary value of the digit.

TABLE 6.2 *Decimal Digits and Their 8421 BCD Equivalents*

Decimal Digit	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

8421 Code

The most straightforward BCD code is the **8421 code,** also called Natural BCD. Each decimal digit is represented by its 4-bit true binary value. When we talk about BCD code, this is usually what we mean.

This code is called 8421 because these are the positional weights of each digit. Table 6.2 shows the decimal digits and their BCD equivalents.

8421 BCD is not a positional number system, because each decimal digit is encoded separately as a 4-bit number.

Excess-3 Code

Excess-3 code is a type of BCD code that is generated by adding 11_2 (3_{10}) to the 8421 BCD codes. Table 6.3 shows the Excess-3 codes and their 8421 and decimal equivalents.

The advantage of this code is that it is **self-complementing.** If the bits of the Excess-3 digit are inverted, they yield the **9's complement** of the decimal equivalent.

We can generate the 9's complement of an n-digit number by subtracting it from a number made up of n 9's. Thus, the 9's complement of 632 is 999 – 632 = 367.

The Excess-3 equivalent of 632 is 1001 0110 0101. If we invert all the bits, we get 0110 1001 1010. The decimal equivalent of this Excess-3 number is 367, the 9's complement of 632.

This property is useful for performing decimal arithmetic digitally.

TABLE 6.3 *Decimal Digits and Their 8421 and Excess-3 Equivalents*

Decimal Digit	8421	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Gray Code

Table 6.4 shows a 4-bit **Gray code** compared to decimal and binary values. Any two adjacent Gray codes differ by exactly one bit.

Gray code can be extended indefinitely if you understand the relationship between the binary and Gray digits. Let us name the binary digits $b_3b_2b_1b_0$, with b_3 as the most significant bit, and the Gray code digits $g_3g_2g_1g_0$ for a 4-bit code. The Gray code bits are defined as follows:

$$g_3 = b_3$$
$$g_2 = b_3 \oplus b_2$$
$$g_1 = b_2 \oplus b_1$$
$$g_0 = b_1 \oplus b_0$$

TABLE 6.4 *4-Bit Gray Code*

Decimal	True Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

For an n-bit code, the MSBs are the same in Gray and binary ($g_n = b_n$). The other Gray digits are generated by the Exclusive OR function of the binary digits in the same position and the next most significant position.

Another way to generate a Gray code sequence is to recognize the inherent symmetry in the code. For example, a 2-bit Gray code sequence is given by:

00
01
11
10

(Note that this is the same sequence as the numbering of cells in a Karnaugh map.) To generate a 3-bit Gray code, write the 2-bit sequence, then write it again in reverse order.

00
01
11
10
10
11
01
00

Add an MSB of 0 to the first four codes and an MSB of 1 to the last four codes. The sequence followed by the last two bits of all codes is symmetrical about the center of the sequence.

000
001
011
010
110
111
101
100

We can apply a similar process to generate a 4-bit Gray code. Write the 3-bit sequence, then again in reverse order. Add an MSB of 0 to the first half of the table and an MSB of 1 to the second half. This procedure yields the code inTable 6.4.

We will see further applications of Gray code in a later chapter.

ASCII Code

Digital systems and computers could operate perfectly well using only binary numbers. However, if there is any need for a human operator to understand the input and output data of a digital system, it is necessary to have a system of communication that is understandable to both a human operator and the digital circuit.

A code that represents letters (alphabetic characters) and numbers (numeric characters) as binary numbers is called an **alphanumeric code.** The most commonly used alphanumeric code is **ASCII** ("askey"), which stands for American Standard Code for Information Interchange. ASCII code represents letters, numbers, and other printable characters in 7 bits. In addition, ASCII has a repertoire of "control characters," codes that are used to send control instructions to and from devices such as video display terminals, printers, and modems.

Table 6.5 shows the ASCII code in both binary and hexadecimal forms. The code for any character consists of the bits in the column heading, then those in the row heading. For example, the ASCII code for "A" is 1000001_2 or 41H. The code for "a" is 1100001_2 or 61H. The codes for capital (uppercase) and lowercase letters differ only by the second most significant bit, for all letters. Thus, we can make an alphabetic **case shift,** like using the Shift key on a typewriter or computer keyboard, by switching just one bit.

Numeric characters are listed in column 3, with the least significant digit of the ASCII code being the same as the represented number value. For example, the numeric character "0" is equivalent to 30H in ASCII. The character "9" is represented as 39H.

The codes in columns 0 and 1 are control characters. They cannot be displayed on any kind of output device, such as a printer or video monitor, although they may be used to control the device. For instance, if the codes 0AH (Line Feed) and 0DH (Carriage Return) are sent to a printer, the paper will advance by one line and the print head will return to the beginning of the line.

The displayable characters begin at 20H ("space") and continue to 7EH ("tilde"). Spaces are considered ASCII characters.

6.6 Binary Adders and Subtractors

Half and Full Adders

There are only three possible sums of two 1-bit binary numbers:

$$0 + 0 = 00$$
$$0 + 1 = 01$$
$$1 + 1 = 10$$

TABLE 6.5 *ASCII Code*

LSBs \ MSBs	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	NUL	DLE	SP	0	@	P	`	p
0001 (1)	SOH	DC1	!	1	A	Q	a	q
0010 (2)	STX	DC2	"	2	B	R	b	r
0011 (3)	ETX	DC3	#	3	C	S	c	s
0100 (4)	EOT	DC4	$	4	D	T	d	t
0101 (5)	ENQ	NAK	%	5	E	U	e	u
0110 (6)	ACK	SYN	&	6	F	V	f	v
0111 (7)	BEL	ETB	'	7	G	W	g	w
1000 (8)	BS	CAN	(	8	H	X	h	x
1001 (9)	HT	EM	)	9	I	Y	i	y
1010 (A)	LF	SUB	*	:	J	Z	j	z
1011 (B)	VT	ESC	+	;	K	[	k	{
1100 (C)	FF	FS	,	<	L	\	l	|
1101 (D)	CR	GS	-	=	M	]	m	}
1110 (E)	SO	RS	.	>	N	^	n	~
1111 (F)	SI	US	/	?	O	—	o	DEL

Control Characters:

NUL—NULL
SOH—Start of Header
STX—Start Text
ETX—End Text
EOT—End of Transmission
ENQ—Enquiry
ACK—Acknowledge
BEL—Bell
BS—Backspace
HT—Horizontal Tabulation
LF—Line Feed
VT—Vertical Tabulation
FF—Form Feed
CR—Carriage Return
SO—Shift Out
SI—Shift In
SP—Space
DLE—Data Link Escape
DC1—Device Control 1
DC2—Device Control 2
DC3—Device Control 3
DC4—Device Control 4
NAK—No Acknowledgment
SYN—Synchronous Idle
ETB—End of Transmission Block
CAN—Cancel
EM—End of Medium
SUB—Substitute
ESC—Escape
FS—Form Separator
GS—Group Separator
RS—Record Separator
US—Unit Separator
DEL—Delete

We can build a simple combinational logic circuit to produce these sums. Let us designate the bits on the left side of these equalities as inputs to the circuit and the bits on the right side as outputs. Let us call the LSB of the output the sum bit, symbolized by Σ, and the MSB of the output the carry bit, designated C_{OUT}.

Figure 6.1 shows the logic symbol of the circuit, which is called a **half adder.** Its truth table is given in Table 6.6. Because addition is subject to the commutative property $(A + B = B + A)$, the second and third lines of the truth table are the same.

Logics 1 and 0 are generally represented by voltage levels. In a positive logic system, the most positive voltage level (HIGH) represents the logical 1 state and the most negative voltage level (LOW) represents the logical '0' state.

In a negative logic system, the most positive (HIGH) voltage level represents logical 0 state and the most negative (LOW) voltage level represents the logical '1' state.

The logic gates are the building blocks of hardware which are available in the form of various IC families. Each gate has a distinct logic symbol and its operation can be described by means of an algebraic function.

6.2.1 OR Gate

The OR gate performs logical addition. The OR gate as two or more inputs and only one output. A HIGH (1) on the output is produced when any of the inputs is HIGH(1). The output is LOW(0) only when all the inputs are LOW(0).

If A and B are the input variables of an OR gate then its output Y is,

$$Y = A + B$$

a) Logical Symbol

Figure 6.1 shows logical symbol of OR gate .

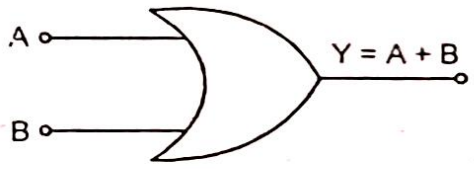


Figure 6.1

b) Truth Table

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

Circuit Diagram

Figure 6.2 shows circuit diagram of OR gate.

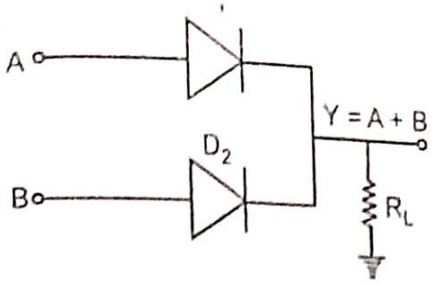


Figure 6.2

Electrical Equivalent

Figure 6.3 shows electrical equivalent of OR gate.

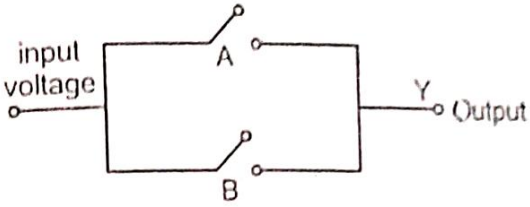


Figure 6.3

If A = 0, B = 0 both diodes will not conduct and hence Y = 0.

If A = 1 & B = 0, diode D_1 conducts and Y = 1.

If A = 0 & B = 1, diode D_2 conducts, hence Y = 1.

If A = 1 & B = 1, both diodes conducts and hence Y = 1.

6.2 AND Gate

The AND gate performs logical multiplication. The output of an AND gate is HIGH only when all the inputs are HIGH. Even if any one of the inputs is LOW, the output will be LOW.

If A and B are the input variables of an AND gate then the output Y = A.B.

Logic Symbol

Figure 6.4 shows logical symbol of AND gate.

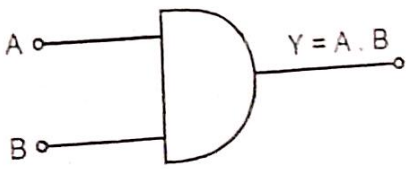


Figure 6.4

b) Truth Table

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

c) Circuit Diagram

Figure 6.5 shows circuit diagram of AND gate.

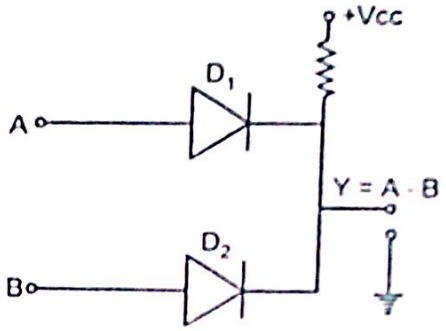


Figure 6.5

d) Electrical Equivalent

Figure 6.6 shows electrical equivalent of AND gate.

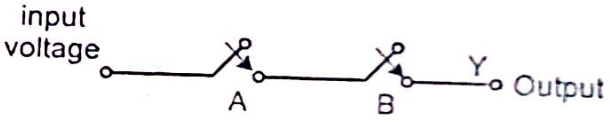


Figure 6.6

If A = 0 & B = 0 the diodes conduce & hence the output voltage is ideally zero (i.e) Y = 0.

If A = 0 & B = 0, upper diode (D_1) is forward biased (ON) and it pulls the output down to a low voltage. (i.e) Y = 0, with B = 1, the diode D_2 goes into reverse bias(OFF).

If A = 1 & B = 0, Y = 0.

If A = 1 & B = 1, both diodes are reverse biased and there is no current through diodes and resistor R_L. This pulls up the output Y to the supply voltage.

$\therefore$ Y = 1.

6.2.3 NOT Gate (Inverter)

The NOT gate performs the basic logical function called inversion or complementation. It has one input and one output. When a HIGH level is applied LOW level appears at its output and vice versa.

a) Logic Symbol

Figure 6.7 shows logical symbol of NOT gate.

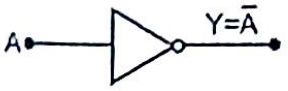


Figure 6.7

b) Truth Table

Input A	*Output* Y = $\bar{A}$
0	1
1	0

c) Circuit Diagram

Figure 6.8 shows circuit diagram of NOT gate.

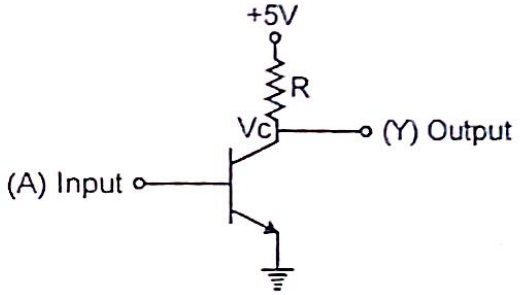


Figure 6.8

When the input is HIGH, the transistor is in the ON state and the output ($V_C = V_{CE(SAT)}$) is low. If the input is LOW, the transistor is in the off state and the output $V_C = V_{CC}$ is HIGH.

6.2.4 NAND Gate

NAND is a contraction of the NOT & AND gates. It has two or more inputs and one output.

If A & B are inputs, output

$$Y = \overline{A.B}$$

When all the inputs are HIGH, the output is LOW. If any one or both inputs are LOW, then the output is HIGH.

The small circle or bubble in the logic symbol represents the operation of inversion.

a) Logic Symbol

Figure 6.9 shows logical symbol of NAND gate.

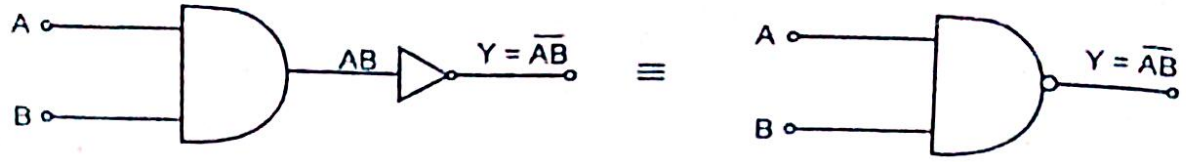


Figure 6.9

b) Truth Table

Inputs		*Output*
A	B	$Y = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

6.2.5 NOR Gate

NOR is a contraction of NOT-OR gates. If A and B are two inputs, the output $Y = \overline{A + B}$.

The output is HIGH only when all the inputs are LOW. If any one or both the inputs are HIGH, then the output is LOW.

a) Logic Symbol

Figure 6.10 shows logical symbol of NOR gate.

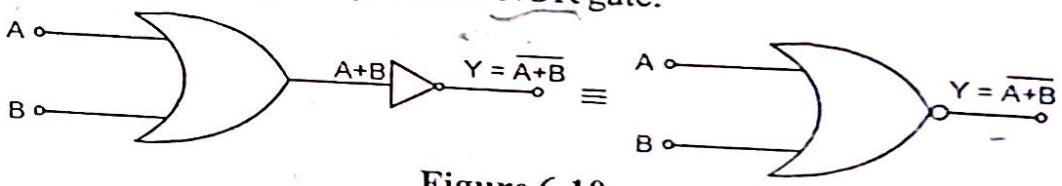


Figure 6.10

b) Truth Table

Inputs		*Output*
A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

6.6 Exclusive - OR (EX-OR) Gate

If A and B are two inputs, the output

$$Y = \overline{A}.B + A.\overline{B} \Rightarrow A \oplus B$$

The output of a two-input EX-OR gate assumes a HIGH state if one and only one input assumes a HIGH state. (i.e) the output is HIGH, if either input A or input B is HIGH but not HIGH exclusively, and low when both are 1 or 0 simultaneously.

Logic Symbol

Figure 6.11 shows logical symbol of EX-OR gate.

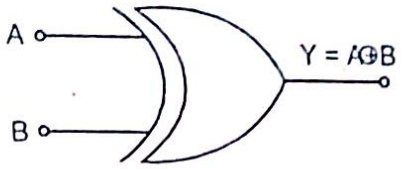


Figure 6.11

Truth Table

Inputs		*Output*
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

The EX-OR gate responds with a HIGH output only when an odd number of inputs are HIGH. When there is an even number of HIGH inputs, the output will always be LOW.

6.7 Exclusive NOR (EX-NOR) Gate

The exclusive-NOR gate is an EX-OR gate followed by an inverter.

The output of EX-NOR gate assumes a HIGH state if both the inputs assume the same logic state or have an even number of 1's and its output is LOW, when the input have odd number of 1's.

If A and B are two inputs, the output

$$Y = \overline{A \oplus B} \Rightarrow AB + \overline{A}\,\overline{B}$$

a) Logic Symbol

Figure 6.12 shows logical symbol of EX-NOR gate.

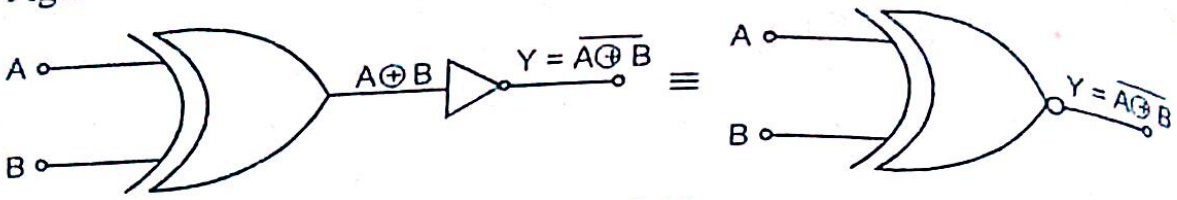


Figure 6.12

b) Truth Table

Inputs		*Output*
A	B	$Y = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

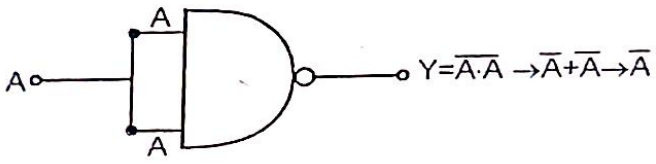
The output of EX-NOR gate is 1 if both the inputs are similar (i.e) both are 0 or 1, otherwise, its output is 0.

6.3 Universal Gates

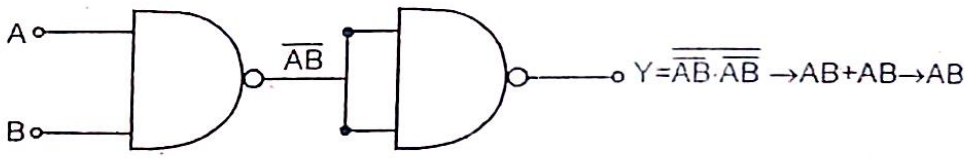
NAND and NOR gates are called universal gates or universal building blocks because both can be used to implement any gate like AND, OR and NOT gates or any combination of these basic gates.

Realisation of Various Logic Gates using NAND Gates

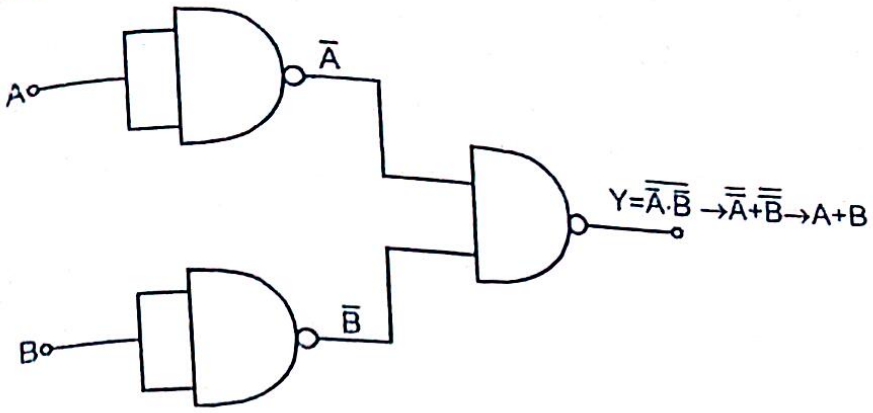
a) NOT Gate



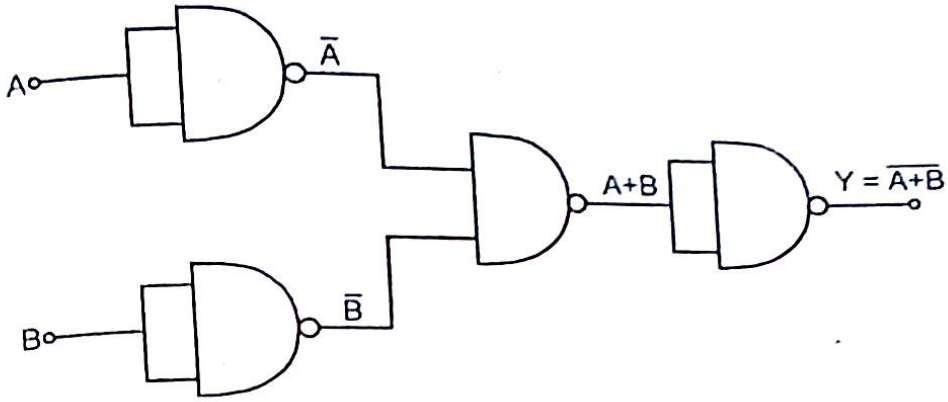
b) AND Gate



c) OR Gate

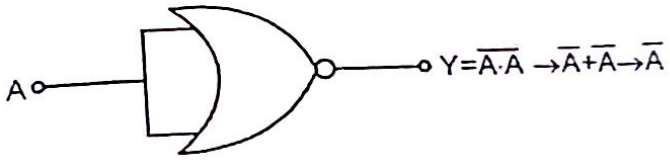


d) NOR Gate

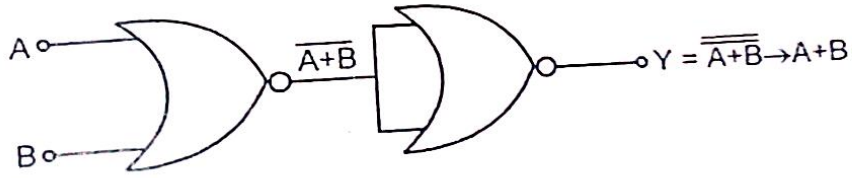


Realisation of Various Logic Gates using NOR Gates

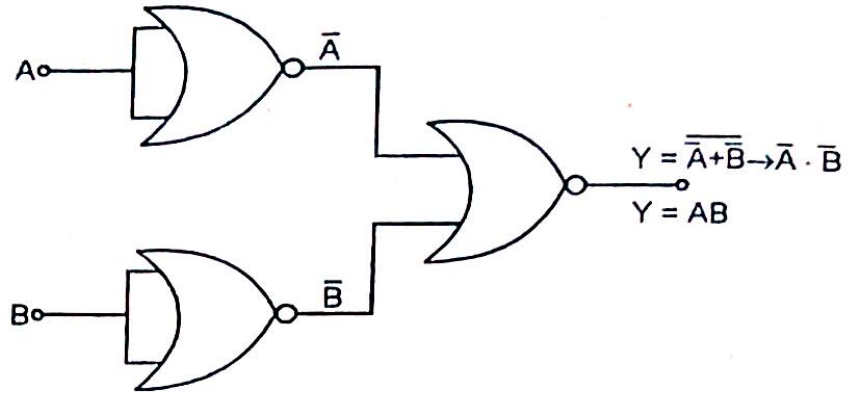
a) NOT Gate



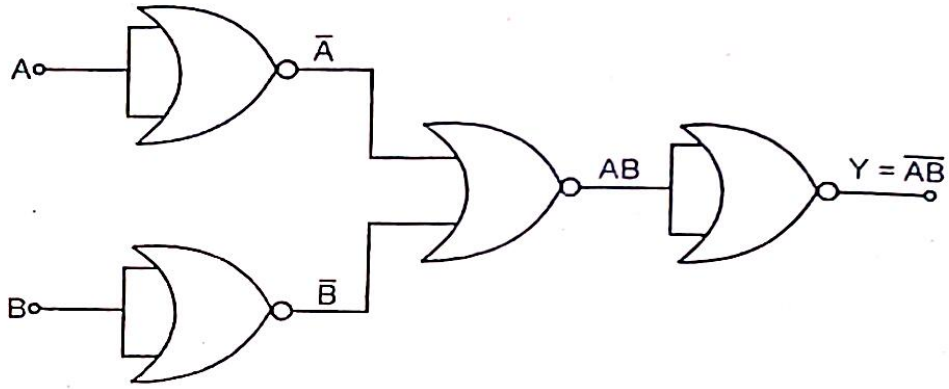
b) OR Gate



c) AND Gate



d) NAND Gate



6.4 Basic Laws of Boolean Algebra

a) Basic rules of Boolean addition are,

$0 + 0 = 0$

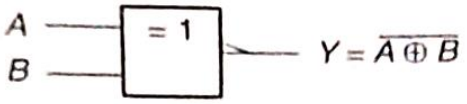
$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 1$

Boolean addition is same as the logical OR.

a. Distinctive-shape



b. Rectangular-outline

FIGURE 2.17 Exclusive NOR Gate

TABLE 2.10 Exclusive NOR Function Truth Table

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

property is similar to that of the Exclusive OR gate, only opposite in sense. Many of the applications that make use of this property can use either the XOR or the XNOR gate.

2.3 DeMorgan's Theorems and Gate Equivalence

Recall the description of a 2-input NAND gate: "**All inputs HIGH make output LOW.**" This condition is satisfied in the last line of the 2-input NAND truth table, repeated in Table 2.11. We could also describe the gate function by saying, "**At least one input LOW makes output HIGH.**" This condition is satisfied by the first three lines of Table 2.11.

TABLE 2.11 NAND Truth Table

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

The gates in Figure 2.18 represent positive and negative forms of a NAND gate. Figure 2.19 shows the logic equivalents of these gates. In the first case, we combine the inputs in an AND function, then invert the result. In the second case, we invert the input variables, then combine the inverted inputs in an OR function.

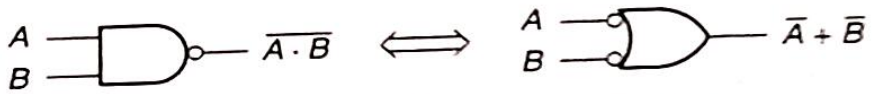
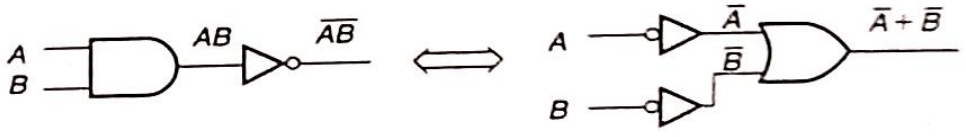


FIGURE 2.18 NAND Gate and DeMorgan Equivalent (positive and negative NAND)



a. AND then Invert

b. Invert then OR

FIGURE 2.19 Logic Equivalents of Positive and Negative NAND Gates

The Boolean function for the AND-shaped gate is given by:

$$Y = \overline{A \cdot B}$$

The Boolean expression for the OR-shaped gate is:

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

The gates shown in Figure 2.18 are called **DeMorgan equivalent forms.** Both gates have the same truth table, but represent different aspects or ways of looking at the NAND function. We can extend this observation to state that *any* gate (except XOR and XNOR) has two equivalent forms, one AND, one OR.

A gate can be categorized by examining three attributes: *shape*, *input*, and *output*. A question arises from each attribute:

1. What is its shape (AND/OR)?
   AND: *all*
   OR: *at least one*
2. What active level is at the gate inputs (HIGH/LOW)?
3. What active level is at the gate output (HIGH/LOW)?

The answers to these questions characterize any gate and allow us to write a descriptive sentence and a truth table for that gate. The DeMorgan equivalent forms of the gate will yield opposite answers to each of these questions.

Thus the gates in Figure 2.18 have the following complementary attributes:

	Basic Gate	DeMorgan Equivalent
Boolean Expression	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
Shape	AND	OR
Input Active Level	HIGH	LOW
Output Active Level	LOW	HIGH

The gates in Figures 2.18 and 2.46 yield the following algebraic equivalencies:

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

These equivalencies are known as **DeMorgan's theorems.** (You can remember how to use DeMorgan's theorems by a simple rhyme: "Break the line and change the sign.")

We will look at DeMorgan's Theorems more in the next chapter, exploring how we can use these mathematically. For now, we will use these when it is to our advantage to change the shape of the gate in a circuit.

It is tempting to compare the first gate in Figure 2.18 and the second in Figure 2.46 and declare them equivalent. Both gates are AND-shaped, both have inversions. However, the comparison is false. The gates have different truth tables, as we have found in Tables 2.11 and 2.13. Therefore they have different logic functions and are not equivalent. The same is true of the OR-shaped gates in Figures 2.18 and 2.46. The gates may look similar, but because they have different truth tables, they have different logic functions and are therefore not equivalent.

The confusion arises when, after changing the logic input and output levels, you forget to change the shape of the gate. This is a common, but serious, error. These inequalities can be expressed as follows:

$$\overline{A \cdot B} \neq \overline{A} \cdot \overline{B}$$
$$\overline{A + B} \neq \overline{A} + \overline{B}$$

As previously stated, any AND- or OR-shaped gate can be represented in its DeMorgan equivalent form. All we need to do is analyze a gate for its shape, input, and output, then *change everything.*

2.4 Enable and Inhibit Properties of Logic Gates

In Chapter 1, we saw that a **digital signal** is just a string of bits (0s and 1s) generated over time. A major task of digital circuitry is the direction and control of such signals. Logic gates can be used to **enable** (pass) or **inhibit** (block) these signals. (The word "gate" gives a clue to this function; the gate can "open" to allow a signal through or "close" to block its passage.)