# NUMBER SYSTEMS AND ARITHMETIC

## 1.1 INTRODUCTION

In order to understand digital methods one must first understand what makes them differ from analog methods. The term digital refers to a process that is accomplished using discrete units or a system of counting using discrete units. It is India that gave us the ingenious method of expressing all numbers by means of ten different symbols, each symbol having an absolute value and also has place value. Today the decimal system for counting has been so widely used throughout the world, that we rarely consider the possibilities of some other number system. But in digital methods we make use of other number systems also. In fact a little used but very simple system, the binary number system has proved to be most useful number system in digital electronics. In this chapter we shall introduce four number systems which are used in computers and other digital systems.

## 1.2 DECIMAL NUMBER SYSTEM

The decimal number system has ten separate symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. These are called *arabic numerals*. Though we call it a decimal (10's) system yet it does not have a symbol for "ten" (10). In fact ten or any other number above ten is expressed as a combination of these symbols.

The *base* or *radix* of a number system is defined as the number of different digits which can occur in each position in the number system. As the decimal system has 10 different digits (0, 1, .....9) and any one of them may be used in each position in a number. Hence the decimal number system has a base or radix of 10. The decimal system is an example of a system which employes positional notations. The following are three important characteristics of a number system having positional notation (place value):

1. The base or radix is equal to the number of digits.
2. The largest digit is one less than the base.
3. Each digit is multiplied by the base or radix raised to an appropriate power depending upon the digit position to get its place value.

As stated above though the decimal system has only 10 symbols or digits, any number of any magnitude can be expressed by using place value. For example, consider a number 8927. This number can be broken down as follows:

$$8927 = 8000 + 900 + 20 + 7$$

$$= 8 \times 10^3 + 9 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$$

In the above example 8 is positioned four places to the left of the decimal point, it has much greater place value than 7. If 8 would have been next to the decimal point, it would have the place value only 8 instead of 8000. Thus the position of the digit with reference to the decimal point determines its weight or value. This can be further illustrated by the following example 8235.46.

Which is a shorthand notation for

$$(8 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (5 \times 10^0) + (4 \times 10^{-1}) + (6 \times 10^{-2})$$

The principle of positional value can be extended to any number system. In general, a number in a system having base or radix $r$ can be written as

$$a_n\ a_{n-1}\ a_{n-2}\ \ldots\ldots\ldots a_0\ .\ a_{-1}\ a_{-2}\ \ldots\ldots\ldots a_{-m}$$

This will be interpreted as

$$Y = a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots\ldots$$
$$a_0 r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} \ldots a_{-m} \times r^{-m} \ldots (1.1)$$

Where $Y$ is the value of the entire number, $a_n$ is the value of the $n$th digit from the point and $r$ is the radix. The symbols $a_n$, $a_{n-1}$, $a_{-m}$ used in the above representation should be one of the symbols allowed in system from r digits. In the above representation $a_n$ is called the *most significant digit* and $a_{-m}$ (the last digit of the number) is called the *least significant digit* of the number. To distinguish a number system from other system the radix of the system is written as a suffix to the numbers. Say $8235.46_{10}$ indicates that this number is in decimal system.

## 1.3 BINARY NUMBER SYSTEM

In digital systems and Instruments, the number system used has a radix 2 and is called *binary system*. Binary means two. The binary number system uses only two digits, 0 and 1. Modern digital systems and computers do not work with decimal numbers. Instead, they work with binary numbers.

In digital systems the principal circuit elements are transistors which are similar to those used in television and radio sets. To achieve greater reliability designers use these devices so that they are essentially in one of the two states, fully conducting or non-conducting. We can have a simple analogy between this type of circuit and an electric bulb. At any instant the bulb (or transistor) can be either *on* (conducting) or *off* (non-conducting). A transistor, after being in use for a very long period, will be in conducting or non-conducting state only, and can not have any other state. A slight change in their characteristics will not affect their performance, if the transistor is used in binary state (on or off). In fact electronic devices are most reliable when they are designed for binary (two state–on or off) operation.

The two symbols, 0 and 1 used in binary number system are called *bit*, a shortend form for binary digit. The first question which comes to mind in dealing with binary system is how do we represent numbers which are greater than 1 ? To understand it let us consider an uncommon odometer (speedometer) whose wheels have only two digits 0 and 1. In such an odometer when each wheel turns, it displays 0, then 1, and back to zero and this cycle may be repeated a number of times. As each wheel has only two digits, it can be called a binary odometer.

Let us consider such an odometer be fitted in a car. Initially it will indicate 0000 and after doing 1 km it will indicate 0001. Now when the second km is done the unit wheel or first wheel indicating unit is forced to change to zero i.e. it is reset and carry so the number indicated becomes 0010. After doing 3 km the first wheel is set and the number indicated will be 0011. After doing 4th km, the unit wheel resets and carries, the second wheel resets and carries, and the third wheel advances by 1. The same type of positional notation is used in the binary number system as in the decimal system. In decimal system the powers of 10 are used while in the binary system powers of 2 are used as the radix or base is 2 in binary system. A number in binary system can be written as a group of 0's and 1's. For example, number $1011 . 1101_2$

$$= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$
$$+ (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$$
$$= 8 + 0 + 2 + 1 + 0.5 + 0.25 + 0 + 0.0625$$
$$= 11.8125_{10}$$

The subscripts on the above numbers indicate the radix or base of the number and are generally omitted when the base of the number is understood. Table 1.1 lists the first 32 binary numbers along with their decimal equivalent.

**TABLE 1.1**

| Decimal | Binary | Decimal | Binary |
|---|---|---|---|
| 1 | 1 | 17 | 10001 |
| 2 | 10 | 18 | 10010 |
| 3 | 11 | 19 | 10011 |
| 4 | 100 | 20 | 10100 |
| 5 | 101 | 21 | 10101 |
| 6 | 110 | 22 | 10110 |
| 7 | 111 | 23 | 10111 |
| 8 | 1000 | 24 | 11000 |
| 9 | 1001 | 25 | 11001 |
| 10 | 1010 | 26 | 11010 |
| 11 | 1011 | 27 | 11011 |
| 12 | 1100 | 28 | 11100 |
| 13 | 1101 | 29 | 11101 |
| 14 | 1110 | 30 | 11110 |
| 15 | 1111 | 31 | 11111 |
| 16 | 10000 | 32 | 100000 |

In general, the value of a binary number $a_1 2^{n-1} + a_2 2^{n-2} + a_3 2^{n-3} + \ldots\ldots a_n$ can be represented as $a_1\ a_2\ a_3\ a_4\ \ldots\ldots a_n$, where $a$'s can be 0 or 1 and $n$ is the number of digits to the left of the binary (radix) point. The fractional numbers are formed in the same general way as in the decimal system (as explained in the above example).

## 1.4 DECIMAL TO BINARY CONVERSION

Following methods can be used for converting a decimal number into a binary number.

### 1.4.1 The Double-Dabble Method

Using this method integer numbers are converted to the desired base using successive division by the base or radix. A number (say $N$) can be written as

$$N = N_i + N_f$$

where $N_i$ is the integer part and $N_f$ is the fractional part. For example, in number $11.8125_{10}$ the integer part is $11_{10}$. This can be converted to the base 2 by the following process of successive division and writing down each quotient and its remainder. The remainders give the binary number and the quotient is successively divided by 2.

| *Successive division* | | *Remainder* | | |
|---|---|---|---|---|
| 2 | 11 | | | |
| 2 | 5 | 1 | LSB | (Least significant bit or last digit) |
| 2 | 2 | 1 | | |
| 2 | 1 | 0 | | |
| 2 | 0 | 1 | MSB | (Most Significant Bit or first digit) |

Here $11_{10}$ is divided by 2, we get 5 as quotient and 1 as *first remainder.* Now the quotient 5 is again divided by 2 which gives quotient 2 and *second remainder is* 1. Again this quotient is divided by 2, which gives 1 as quotient and 0 as *third remainder.* Now 1 is again divided by 2 which gives 0 as quotient and *fourth remainder* as 1.

In the above division the decimal number is repeatedly divided by 2 till the quotient becomes zero and the remainder after each division is used to indicate the coefficient of the binary number to be formed. It may be noted that the binary so derived is written from the bottom up i.e. the last remainder is the *most significant bit* (MSB) and the first remainder is the *least significant bit* (LSB). Thus in the above example the integer $11_{10} = 1011_2$.

For changing a fractional number to the desired base, successive multiplications by the radix or base are performed. For example, $0.8125_{10}$ can be converted into binary as shown in Table 1.2.

From above example we find if we multiply a fraction by 2, the integer part of the product is the most significant bit of the equivalent binary fraction.

The fractional part of the product is multiplied again by 2 to obtain the next significant bit. The procedure is continued till the fractional part of the product becomes zero or the desired number of binary places are obtained. In the present example the fractional part becomes zero on 4th multiplication. i.e., fractional part $(0.8125_{10})$ becomes $0.1101_2$ and $11.8125_{10}$ is equal to $1011.1101_2$.

### 1.4.2 Stream lined Double-Dabble Method

In this method we do not write down 2 before each division as we know each time we are to divide by 2. For example, if we wish to convert $15_{10}$ to binary we may do the division in the following way:

**TABLE 1.2**

| *Base* | | *Fractional number* | | *Product* | *Fractional part of the product* | *Overflow* | |
|---|---|---|---|---|---|---|---|
| 2 | × | 0.8125 | = | 1.6250 | 0.6250 | 1 | MSB |
| 2 | × | 0.6250 | = | 1.2500 | 0.2500 | 1 | |
| 2 | × | 0.2500 | = | 0.5000 | 0.5000 | 0 | |
| 2 | × | 0.5000 | = | 1.000 | 0.0000 | 1 | LSB |
| 2 | × | | | | | | |

| Successive division | Remainders | |
|---|---|---|
| 15 | | |
| 7 | 1 | LSB |
| 3 | 1 | |
| 1 | 1 | |
| 0 | 1 | MSB |

One of the most obvious method of converting a decimal number into a binary number is to obtain the largest power of 2 from table (giving powers of 2) which does not exceed the original number and record this. Now subtract this number obtained (with the help of the table) from the original number. This process is repeated for the remainder and continued till the remainder becomes zero. By adding the binary numbers obtained from the table the result is obtained. For example, let us convert $53_{10}$ to binary. The largest number (as power of 2) less than 53 is 32 *i.e.*

| | |
|---|---|
| 32 is the largest number without exceeding 53 | 32 = 1 0 0 0 0 0 |
| 53 – 32 = 21 | |
| 16 is the largest number without exceeding 21 | 16 = 1 0 0 0 0 |
| 21 – 16 = 5 | |
| 4 is the largest number without exceeding 5 | 4 = 1 0 0 |
| 5 – 4 = 1 | |
| One is the largest number | 1 = 1 |
| Adding the binary numbers | $53_{10}$ = 1 1 0 1 0 $1_2$ |

This method requires a table of power of 2 and its use is restricted to small numbers where these powers have been memorized.

## 1.5 BINARY TO DECIMAL CONVERSION

As humans have 10 fingers and use the decimal system, for realising a binary electronic system which deals in 1's and 0's one must be capable of converting a binary number into a decimal number. The binary system is positionally weighted as decimal system and each position represents a particular value of 2". Hence the conversion of a binary number into decimal number is identical to that of breaking a decimal number into its weighted values and can be performed by using equation (1). The $a$'s will be 0's or 1's, $r$ will be 2 and $n$'s will be various powers of 2 depending on the position of the bit (digit) with reference to the binary point (in binary radix point or binary point is used and not decimal point). For example, for converting $11110_2$ to decimal we have

$$Y = a_4 \times r^4 + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0$$

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

$$= 16 + 8 + 4 + 2 + 0$$

Hence $11110_2 = 30_{10}$

From above it is clear that for converting a binary number into a decimal number break it into parts which is equivalent to splitting its decimal equivalent into units ($2^0$), two's ($2^1$), four's ($2^2$), eight's ($2^3$) and so on. In fact each digit in a binary at its position has a value or weight. The last digit at the right has a value of 1 and is the least significant bit. The second position from the right has a value 2, the next 4, then 8, 16, 32, 64 etc. Table 1.3 shows digit positions of a binary number along with equivalent decimal weights.

From above it is clear for finding decimal equivalent for a binary number we should proceed as follows:

(1) In a binary number where bit 1 is in a digit position, add the value or weight of the position.

(2) Where bit 0 is in a digit position ignore the weight of that position.

The binary to decimal conversion can be stream line as follows:

TABLE 1.3

| *Digit position in binary from binary point* | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| *Equivalent decimal weight or Value in the powers of 2* | 64 or $2^6$ | 32 or $2^5$ | 16 or $2^4$ | 8 or $2^3$ | 4 or $2^2$ | 2 or $2^1$ | 1 or $2^0$ |

(1) Write the binary number.

(2) Just below the binary number digits write their place value such as 1, 2, 4, 8, 16, 32, 64..... starting from right to left.

(3) In case if zero appears in a digit position, ignore or cross out the decimal value for the position or take it zero.

(4) To get the decimal equivalent add the remaining weights.

For example, the conversion of 1010 to its decimal equivalent can be obtained as follows by using above method:

**1st Step** (Write the binary number) 1 0 1 0

**2nd Step** (Write place value below each binary digit) 8 4 2 1

**3rd Step** (Ignore the decimal value for bits having a 0) 8 – 2 –

**4th Step** (Add the remaining weights) $8 + 2 = 10_{10}$

Hence the decimal equivalent of binary number $1010_2$ is $10_{10}$. The binary fractions are also converted into decimal equivalent using the same method i.e. by using the weights of digit positions to the right of the binary point. Table 1.4 shows digit positions of a binary fraction number along with equivalent decimal weights.

For example, the conversion of 0.1011 can be performed as follows:

| | |
|---|---|
| Binary number | 0 . 1 0 1 1 |
| Place value of the digit position | $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$ |
| Equivalent decimal value | 0.5000 + 0 + 0.1250 + 0.0625 = 0.6875. |

The conversion of a mixed number (having an integer and a fraction part) from binary to decimal number can be obtained by handling integer and fraction part as shown above.

**Example 1.** *Convert binary number 111.101 to a decimal number.*

**Solution.**

| | |
|---|---|
| Binary Number | 1 1 1 . 1 0 1 |
| Place value of the digit position | 4 2 1 . $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ |

Equivalent decimal value $(4 + 2 + 1) + (1/2 + 0 + 1/8)$

$= 7 + (0.5 + 0 + 0.125)$

$= 7.625_{10}$

**Example 2.** *Convert $110111_2$ to decimal.*

**Solution.**

$$110111_2 = (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

$$= 32 + 16 + 0 + 4 + 2 + 1$$

$$= 55_{10}$$

**Example 3.** *Convert $300_{10}$ into binary.*

**Solution.**

| Successive division | | Remainders | |
|---|---|---|---|
| 2 | 300 | | |
| 2 | 150 | 0 | → LSB |
| 2 | 75 | 0 | |
| 2 | 37 | 1 | |
| 2 | 18 | 1 | Read from bottom to top |
| 2 | 9 | 0 | |
| 2 | 4 | 1 | |
| 2 | 2 | 0 | |
| 2 | 1 | 0 | |
| | 0 | 1 | → MSB |

**TABLE 1.4**

| *Digit position in binary after binary point* | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| *Equivalent decimal weight or* | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | $\frac{1}{64}$ | $\frac{1}{128}$ |
| *Value in powers of two* | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ | $2^{-6}$ | $2^{-7}$ |

Reading the remainders from the bottom to the top we have

$300_{10} = 100101100_2$.

The same result can also be obtained by using the table of power of 2 as follows :

256 is the largest number without exceeding 300

$\rightarrow 256 = 100000000$

$300 - 256 = 44$

32 is the largest number without exceeding 44

$\rightarrow 32 = 100000$

$44 - 32 = 12$

8 is the largest number without exceeding 12

$\rightarrow 8 = 1000$

$12 - 8 = 4$

4 is the largest number without exceeding 4

$\rightarrow 4 = 100$

Adding the binary numbers we have

$300_{10} = 100101100_2$

**Example 4.** *Convert $19.6_{10}$ to a binary number.*

**Solution.** Take 19 as integer part and 0.6 as fractional part independently and convert them into binary and then add.

| Successive division | | Remainder | |
|---|---|---|---|
| 2 | 19 | | |
| 2 | 9 | 1 | LSB |
| 2 | 4 | 1 | |
| 2 | 2 | 0 | |
| 2 | 1 | 0 | |
| | 0 | 1 | MSB |

Hence $19_{10} = 10011_2$. Similarly by successive multiplication fractional part $0.6_{10}$ can be converted into binary as follows :

| Base × | Fractional number | Product | Fractional part of the product | Overflow or carry |
|---|---|---|---|---|
| 2 × | 0.6 | 1.2 | 0.2 | 1 →MSB |
| 2 × | 0.2 | 0.4 | 0.4 | 0 |
| 2 × | 0.4 | 0.8 | 0.8 | 0 |
| 2 × | 0.8 | 1.6 | 0.6 | 1 |
| 2 × | 0.6 | 1.2 | 0.2 | 1 → LSB |

Terminating the conversion after 5-bits we have $0.6_{10} = 0.10011_2$.

Thus the approximate equivalent of $19.6_{10}$ in binary is $10011 . 10011_2$ or 10011 . 10011.

**Example 5.** *Determine the decimal equivalent of $11101.011_2$.*

**Solution.**

Binary numbers : 1 1 1 0 1 . 0 1 1

Place value of the digit position: $16\ 8\ 4\ 2\ 1 . \frac{1}{2}\ \frac{1}{4}\ \frac{1}{8}$

Equivalentdecimal value

$16 + 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125$

$= 29 + (0.25 + 0.125)$

$= 29.375_{10}$

## 1.6 BINARY ARITHMETIC

### 1.6.1 Binary Addition

Binary addition is carried out in a same manner as decimal addition. In fact binary addition is much simpler. As in binary we have only two digits we may have the following four possible cases.

**Case 1.** When zero is combined with zero, we get zero. This is represented in binary as $0 + 0 = 0$.

**Case 2.** When zero is combined with one, we get one. This is represented in binary as $0 + 1 = 1$.

**Case 3.** When one is combined with zero, we get 1. This is represented in binary as $1 + 0 = 1$.

**Case 4.** When one is combined with one, we get 2 which is represented as 10 in binary. This is represented in binary as $1 + 1 = 10$ or we can say it is a 0 with one carry. In binary 10 means 2 and not ten.

Binary addition table is shown in Table 1.5.

**TABLE 1.5 Binary Addition Table**

| | |
|---|---|
| 0 + 0 = | 0 |
| 0 + 1 = | 1 |
| 1 + 0 = | 1 |
| 1 + 1 = | 10 or 0 plus a carry over of 1 |

Binary "carry-overs" are performed in a similar way as in decimal addition. As 1 is the largest digit of the binary system, any sum greater than one requires the carry over of the digits. This is same as we do in decimal system on adding 1 to 9 which requires the carry over of the digit. For example, if we wish to add 10 binary to 10 binary it will require addition of two 1's in the second position to the left with a carry over. Since 1 + 1 = 0 + a carry over of 1, the sum of 10 and 10 will be 100.

**Example 6.** *Perform the following additions : $10111_2 + 01101_2$ .*

**Solution.**

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1 \\ 0\ 1\ 1\ 0\ 1 \\ \hline 1\ 0\ 0\ 1\ 0\ 0 \end{array}$$

carry digit

First column : 1 + 1 + 0 = 0 + carry one
Second column : 1 + 0 + 1 = 0 + carry one
Third column : 1 + 1 + 1 = 1 + carry one
Fourth column : 0 + 1 + 1 = 0 + carry one
Fifth column : 1 + 0 + 1 = 0 + carry one

**Example 7.** *Add $1001_2$ and $1110_2$.*

**Solution.**

$$\begin{array}{r} 1001 \\ 1110 \\ \hline 10111 \end{array}$$

First column : 1 + 0 = 1
Second column : 0 + 1 = 1
Third column : 0 + 1 = 1
Fourth column : 1 + 1 = 0 + carry one

The carry is taken in the fifth column in the sum.

**Example 8.** *Add $100101_2$ and $1101111_2$ .*

**Solution.**

$$\begin{array}{r} 100101 \\ 1101111 \\ \hline 10010100 \end{array}$$

We can confirm this by converting these numbers into decimal and adding ($100101_2 = 37_{10}$) $11001111_2 = 111_{10}$ and $10010100_2 = 148_{10}$). Now the sum of $37_{10} + 111_{10} = 148_{10}$, which confirms.

### 1.6.2 Binary Subtraction

The binary subtraction is performed in the same way as is done in decimal system. As binary system has only two digits we require frequent borrowing operations. Table 1.6 shows the binary subtraction.

**TABLE 1.6 Binary subtraction**

| | |
|---|---|
| 0 – 0 = 0 | |
| 1 – 0 = 1 | |
| 1 – 1 = 0 | |
| 0 – 1 = 1 | (one is borrowed and 0 becomes 10) |

The subtraction of large numbers in binary is carried by subtracting column by column and borrowing from the adjacent left column whenever required.

**Example 9.** *Subtract $11_2$ from $1000_2$.*

**Solution.**

| *Binary subtraction* | *Equivalent decimal subtraction* |
|---|---|
| 1000 | 8 |
| – 0011 | –3 |
| 0101 | 5 |

**Example 10.** *Subtract $111001_2$ from $1110001_2$.*

**Solution.**

| *Binary subtraction* | *Equivalent decimal subtraction* |
|---|---|
| 1110001 | $113_{10}$ |
| – 0111001 | $-57_{10}$ |
| 0111000 | $56_{10}$ |

### 1.6.3 Binary Multiplication

Binary multiplication is much simpler than decimal multiplication. This is due to the fact in binary we have only two digits hence the multipli- cation table is very small and requires only four multiplications

instead of 100 required for decimal multiplication. The binary multiplication table is shown in Table 1.7.

**TABLE 1.7 Binary Multiplication Table**

| |
|---|
| 0 × 0 = 0 |
| 1 × 0 = 0 |
| 0 × 0 = 0 |
| 1 × 1 = 1 |

Binary multiplication can be done by two methods called paper method and the computer method and both these methods make use of multiplication table. The paper method is identical to the method used to multiply the decimal numbers on paper.

**Example 11.** *Multiply $10111_2$ by $101_2$.*

**Solution.**

*Binary multiplication*

$$\begin{array}{r} 10111 \\ \times\ 101 \\ \hline 10111 \\ 00000 \\ 10111 \\ \hline 1110011 \end{array}$$

*Equivalent decimal multiplication*

$$\begin{array}{r} 23_{10} \\ \times 5_{10} \\ \hline 115_{10} \end{array}$$

In paper method one can use as many bits as one needs; while in digital systems or computers only a fixed number of bits are available. Hence the paper method can not be used in digital systems or computers as one can add only two numbers at a time along with carries. The other method is called computer method and is used in digital systems or computers and is discussed later.

### 1.6.4 Binary Division

Binary division is very simple and similar to decimal system division. The division by zero is meaningless. The binary division table is shown below in Table 1.8.

**Table 1.8 Binary Division Table**

| |
|---|
| 0 ÷ 1 = 0 |
| 1 ÷ 1 = 1 |

Following examples of division along with their decimal equivalent will make binary division clear.

**Example 12.** *Divide $11000_2$ by $1000_2$.*

**Solution.**

*Binary Division*

$$\begin{array}{r} 11 \\ 1000\ \overline{)\ 11000} \\ 1000 \\ \hline 1000 \\ 1000 \\ \hline 0000 \end{array}$$

*Equivalent decimal division*

$$\begin{array}{r} 3 \\ 8\ \overline{)\ 24} \\ 24 \\ \hline 0 \end{array}$$

**Example 13.** *Divide $1110100_2$ by $111010_2$.*

**Solution.**

*Binary Division*

$$\begin{array}{r} 10 \\ 111010\ \overline{)\ 1110100} \\ 111010 \\ \hline 0000000 \end{array}$$

*Equivalent decimal division*

$$\begin{array}{r} 2 \\ 58\ \overline{)\ 116} \\ 116 \\ \hline 0 \end{array}$$

**Example 14.** *Divide $1111000_2$ by $100_2$.*

**Solution.**

*Binary division*

$$\begin{array}{r} 11110 \\ 100\ \overline{)\ 1111000} \\ 100 \\ \hline 111 \\ 100 \\ \hline 110 \\ 100 \\ \hline 100 \\ 100 \\ \hline 000 \\ 000 \\ \hline \times \end{array}$$

*Equivalent decimal division*

$$\begin{array}{r} 30 \\ 4\ \overline{)\ 120} \\ 12 \\ \hline 00 \\ 00 \\ \hline \times \end{array}$$

## 1.7 HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system is another number system which is also used in digital systems and computers. In fact in using binary number system we are faced with the problem while expressing very large numbers as it requires a long sequence of 0 and 1's. Hexadecimal number system is used for expressing binary numbers concisely and by and large it is the most commonly

used number system. This number system is formed from a binary number by grouping bits in groups of four bits each, starting from the binary point.

Hexadecimal number system has a base of 16 which requires sixteen digits. The digits used are 0, 1 to 9 and A through F. The numbers from 10 to 15 are represented by A through F respectively. As in this system numbers and letters are used this system is also called alphanumeric number system. A counting sequence for this system along with equivalent binary and decimal numbers is shown in table 1.9.

**TABLE 1.9 Hexadecimal Counting**

| *Decimal* | *Binary* | *Hexadecimal* |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 10000 | 10 |
| 17 | 10001 | 11 |
| 18 | 10010 | 12 |
| 19 | 10011 | 13 |
| 20 | 10100 | 14 |
| 21 | 10101 | 15 |
| 22 | 10110 | 16 |
| 23 | 10111 | 17 |
| 24 | 11000 | 18 |
| 25 | 11001 | 19 |
| 26 | 11010 | 1A |
| 27 | 11011 | 1B |
| 28 | 11100 | 1C |
| 29 | 11101 | 1D |
| 30 | 11110 | 1E |
| 31 | 11111 | 1F |
| 32 | 100000 | 20 |

From above table we see that in hexadecimal system starting from 0 after reaching 9 we continue counting as follows:

A B C D E F 20

For counting beyond 16th digit (*i.e.* F) we form two digital combinations. As in decimal system after 9 there is a combination of two digits to represent the number.

In hexadecimal system after 16th digit (*i.e.* F) there is a carry. Hence after F we have zero at first place and 1 at the second place *i.e.* the number after F is 10 and it stands for 16. Now for counting further the first place changes from 0 to 9 and then through A to F *i.e.* 11, 12, 13, .... 19, 1A, 1B....1F. After 1F for next count there is a reset and carry and the number after 1F (1F stands for 31) we have 20 (20 stands for 32). In fact after every fifteen counts from a hexadecimal number having 0 at a digit position we have a reset and carry. For example, for 32, 48, 64, 80,.. decimal numbers corresponding hexadecimal numbers are 20, 30, 40, 50. After 3FFF, 4FFF and 5FFF hexadecimal numbers we will have 4000, 5000 and 6000 respectively.

## 1.8 CONVERSION OF HEXADECIMAL NUMBERS

### 1.8.1 Hexadecimal to Binary Conversion

For converting a hexadecimal number into a binary we convert each hexadecimal digit to its 4 bit (binary) equivalent. The result is obtained by placing the equivalent binary numbers in the same sequences. For this Table 1.9 can be used. We note from this table that each hexadecimal number is represented by four bits.

**Example 15.** *Convert $9AFC_{16}$ to Binary.*

**Solution.**

| *hexadecimal number* | 9 | A | F | C |
|---|---|---|---|---|
| *equivalent binary* | 1001 | 1010 | 1111 | 1100 |

Thus the equivalent binary number is $1001\ 1010\ 1111\ 1100_2$.

**Example 16.** *Convert $C5E2F8_{16}$ to Binary.*

**Solution.**

| *hexadecimal number* | C | 5 | E | 2 | F | 8 |
|---|---|---|---|---|---|---|
| *equivalent binary* | 1100 | 0101 | 1110 | 0010 | 1111 | 1000 |

Thus the equivalent binary number is $1100\ 0101\ 1110\ 0010\ 1111\ 1000_2$.

### 1.8.2 Binary to Hexadecimal Conversion

For converting a binary number into a hexadecimal number we make use of the table.

First we make groups of 4 bits starting from the binary point and if in the last group we do not have 4 bits we assume the remaining bits to be zero. For each 4 bit group we find equivalent hexadecimal digit and to get the result place these hexadecimal digits in the same order.

**Example 17.** *Convert $110011_2$ into hexadecimal number.*

**Solution.** As the digits are only 6 in the binary we assume this number as 0011 0011 and make two groups of 4 bits from right and proceed as follows:

| *Binary group (or 4 bits)* | 0011 | 0011 |
|---|---|---|
| *Equivalent hexadecimal number* | 3 | 3 |

Thus the hexadecimal equivalent of $110011_2$ is $33_{16}$.

**Example 18.** *Convert $1111\ 1010\ 1100\ 1110_2$ in hexadecimal.*

**Solution.**

| *Binary groups (or 4 bits)* | 1111 | 1010 | 1100 | 1110 |
|---|---|---|---|---|
| *Equivalent hexadecimal number* | F | A | C | E |

Thus the hexadecimal equivalent of $1111\ 1010\ 1100\ 1110_2$ is $FACE_{16}$.

### 1.8.3 Hexadecimal to Decimal Conversion

Hexadecimal number system like decimal and binary is a positionally weighted system having radix or base as 16. Hence in place of 10 (decimal system) or 2 (in binary system) we have 16. In hexadecimal numbers each position represents a particular value of $16^n$. Equation (1) holds good having $r$ as 16.

**Example 19.** *Convert $3C9A_{16}$ into decimal.*

**Solution.** We know the value of the entire number is given as

$$Y = a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots a_0 r^0.$$

Substituting $r = 16, n = 4$ (number of digits in 3C9A), and $a_3 = 3, a_2 = C\ (= 12), a_1 = 9$ and $a_0 = A\ (= 10)$ we have

$$\begin{aligned} Y &= a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 \\ &= 3 \times (16)^3 + 12 \times (16)^2 + 9 \times (16)^1 + 10 \times (16)^0 \\ &= 3 \times (4096) + 12 \times (256) + 9 \times (16) + 10 \times (1) \\ &= 12288 + 3072 + 144 + 10 \\ &= 15514_{10}. \end{aligned}$$

The equivalent decimal number of $3C9A_{16}$ is $15514_{10}$.

**Example 20.** *Convert $1F8E6_{16}$ into decimal.*

**Solution.** Making use of the following relation.

$$Y = a_4 \times r^4 + a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0$$

and substituting $a_4 = 1, a_3 = 15, a_2 = 8, a_1 = 14, a_0 = 6$ and $r = 16$ we have for the equivalent decimal number.

$$\begin{aligned} Y &= 1 \times (16)^4 + 15 \times (16)^3 + 8 \times (16)^2 + 14 \times (16)^1 \\ &\quad + 6 \times (16)^0 \\ &= 65536 + 15 \times (4096) + 8 \times (256) + 14 \times 16 \\ &\quad + 6 \times 1 \\ &= 65536 + 61440 + 2048 + 224 + 6 \\ &= 129254_{10}. \end{aligned}$$

Hence the decimal equivalent of $1F8E6_{16}$ is $129254_{10}$.

**Example 21.** *Convert $4AB\ .\ 2A_{16}$ into decimal.*

**Solution.** Taking integer part

$$\begin{aligned} 4AB &= 4 \times (16)^2 + 10 \times (16)^1 + 11 \times (16)^0 \\ &= 4 \times 256 + 10 \times 16 + 11 \times 1 \\ &= 1024 + 160 + 11 \\ &= 1195. \end{aligned}$$

Taking fraction part

$$0.2A_{16} = 2 \times (16)^{-1} + 10 \times (16)^{-2}$$

$= 0.125 + 0.03906$

$= 0.16406.$

Hence the equivalent decimal number of $4AB.2A_{16}$ is 1195.16406.

**Example 22.** *Represent the following in a 16-bit resister (a) $(356)_{10}$ (b) $(356)_{BCD}$.*

**Solution.**

(a) Determine binary equivalent of $(356)_{10}$, i.e.

| | | | |
|---|---|---|---|
| 2 | 356 | | |
| 2 | 178 | 0 | → LSB |
| 2 | 89 | 0 | |
| 2 | 44 | 1 | |
| 2 | 22 | 0 | |
| 2 | 11 | 0 | |
| 2 | 5 | 1 | |
| 2 | 2 | 1 | |
| 2 | 1 | 0 | |
| 2 | 0 | 1 | → MSB |

Hence $(356)_{10} = 0000\ 00010110\ 0100_2$.

(b) BCD code for $(356)_{10}$ is

| 3 | 5 | 6 |
|---|---|---|
| 0000 | 0111 | 0101 | 0110 |

$(356)_{BCD} = 0000\ 0011\ 0101\ 0110$

**Example 23.** *Convert the following hexadecimal numbers to their equivalent binary numbers. (a) $(2DE.CA6)_{16}$ (b) $(1F1.99A)_{16}$.*

**Solution.** (a)

| | | | | | | |
|---|---|---|---|---|---|---|
| | (2 | D | E | . | C | A | 6 )16 |
| = | (0010 | 1101 | 1110 | . | 1100 | 1010 | 0110 )2 |
| (b) | (1 | F | 1 | . | 9 | 9 | A )16 |
| = | ( 0001 | 1111 | 0001 | . | 1001 | 1001 | 1010 )2 |

## 1.9 ARITHMETIC WITH HEXADECIMAL NUMBERS

### 1.9.1 Hexadecimal Addition

The hexadecimal numbers addition is performed in a similar manner as is done for decimal numbers. In hexadecimal addition, a carry is generated in each digit position for which the sum exceeds $15_{10}$ or F. Table 1.10 shows the hexadecimal addition table for addend 0 through F and augend 0 through F.

**TABLE 1.10 Hexadecimal Addition Table**

| Addend / Augend | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 |
| 5 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 |
| 6 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| A | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| B | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A |
| C | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B |
| D | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C |
| E | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| F | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |

**Example 24.** *Add $B2CE5_{16}$ and $AB2C3_{16}$.*

**Solution.**

| | |
|---|---|
| Addend | B 2 C E 5 |
| Augend | + A B 2 C 3 |
| | 1 5 D F A 8 |

In the above example if we consider the addition column wise we have for 1st column (5 + 3)8 without any carry. For second column $E + C (= 14 + 12) = 26_{10}$ or $10_{16}$ with one carry or A with one carry. For column 3 we have C + 2 + carry (= 12 + 2 + 1) $15_{10}$ or $F_{16}$, for 4th column B + A (= 11 + 10) $21_{10}$ or $15_{16}$. Hence the addition of $B2CE5_{16}$ and $AB2C3_{16}$ is $15DFA8_{16}$.

For carrying out this addition we can also use table 1.10.

**Example 25.** *Add $2C3A.1A_{16}$ and $1CBA.21_{16}$*

**Solution.**

| | |
|---|---|
| Addend | 2 C 3 A . 1 A |
| Augend | 1 C B A . 2 1 |
| | 4 8 F 4 . 3 B |

### 1.9.2 Hexadecimal Subtraction

For each digit position in which the minuend digit is less than the subtrahend digit a borrow is required in hexadecimal subtraction. On borro- wing, which is done from the left, it requires that sixteen units be added to the minuend digit and one unit be subtracted from the adjacent (left) minuend digit. The subtraction in hexadecimal system can also be done by making references to the addition Table 1.10. For making use of this table to subtract two hexadecimal numbers, locate the smaller of the two numbers, along the left edge of the table. Then move horizontally along the row until the higher of the two numbers is found. The result will be given by the digit found at the top of this column. For example, if we wish to find D–9, locate 9 along the left edge and move along the row until D is found. The answer will appear at the top of the column in the line of 9 having D *i.e.* the result will be 4.

**Example 26.** *Subtract $2CB39_{16}$ from $3A76F_{16}$.*

**Solution.**

| | |
|---|---|
| Minuend | 3 A 7 6 F |
| Subtrahend | – 2 C B 3 9 |
| Difference | 0 D C 3 6 |

The above subtraction is performed column wise. For column 1 we have

$F - 9 (= 15 - 9) = 6$,

for second column $6 - 3 = 3$, for third column

$7 - B$, 7 + one borrow – B

$= 7_{10} + 16_{10} - 11_{10} = 12_{10} = C_{16}$.

As the borrow is obtained from the minuend digit at 4th position *i.e.* A, we replace this digit A by A – 1 = 9. Hence for 4th column 9 – C (= 9 + borrow – C = 9 + 16 – 12 = 13) = D. The borrow in this digit results in 3 – 1 = 2 replacing 3 of column 5 by 2 we have 0. Thus

$3A76F_{16} - 2CB39_{16} = 0DC36_{16}$ or $DC36_{16}$.

**Example 27.** *Subtract FFFD9 from A39BC1.*

**Solution.** Proceeding as in above example we have

| | |
|---|---|
| Minuend | A 3 9 B C 1 |
| Subtrahend | F F F D 9 |
| | 9 3 9 B E 8 |

### 1.9.3 Hexadecimal Multiplication

The multiplication process in hexadecimal system is similar to that in decimal system. For this we can also make use of the following hexadecimal multiplication Table 1.11.

**Example 28.** *Multiply $2A4_{16}$ by $A8_{16}$.*

**Solution.**

| | *Equivalent decimal* |
|---|---|
| 2A4 | 676 |
| × A8 | × 168 |
| 1520 | 113568 |
| 1A68 | |
| 1BBA0 | |

We notice that the equivalent of $1BBA0_{16}$ is $113568_{10}$.

The above multiplication is easily performed by using hexadecimal multiplication (Table 1.11). For using this table look for the multiplicant along the left edge. Then move horizontally along the top row until the multiplyer is found. The result is given

**TABLE 1.11 Hexadecimal Multiplication Table**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 2 | 0 | 2 | 4 | 6 | 8 | A | C | E | 10 | 12 | 14 | 16 | 18 | 1A | 1C | 1E |
| 3 | 0 | 3 | 6 | 9 | C | F | 12 | 15 | 18 | 1B | 1E | 21 | 24 | 27 | 2A | 2D |
| 4 | 0 | 4 | 8 | C | 10 | 14 | 28 | 1C | 20 | 24 | 28 | 2C | 30 | 34 | 38 | 3C |
| 5 | 0 | 5 | A | F | 14 | 19 | 1E | 23 | 18 | 2D | 32 | 37 | 3C | 41 | 46 | 4B |
| 6 | 0 | 6 | C | 12 | 18 | 1E | 24 | 2A | 30 | 36 | 3C | 42 | 48 | 4E | 54 | 5A |
| 7 | 0 | 7 | E | 15 | 1C | 23 | 2A | 31 | 38 | 3F | 46 | 4D | 54 | 5B | 62 | 69 |
| 8 | 0 | 8 | 10 | 18 | 20 | 28 | 30 | 38 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| 9 | 0 | 9 | 12 | 1B | 24 | 2D | 36 | 3F | 48 | 51 | 5A | 63 | 6C | 75 | 7E | 87 |
| A | 0 | A | 14 | 1E | 28 | 32 | 3C | 46 | 50 | 5A | 64 | 6E | 78 | 82 | 8C | 96 |
| B | 0 | B | 16 | 21 | 2C | 37 | 42 | 4D | 58 | 63 | 6E | 79 | 84 | 8F | 9A | A5 |
| C | 0 | C | 18 | 24 | 30 | 3C | 48 | 54 | 60 | 6C | 78 | 84 | 90 | 9C | A8 | B4 |
| D | 0 | D | 1A | 27 | 34 | 41 | 4E | 5B | 68 | 75 | 82 | 8F | 9C | A9 | B6 | C3 |
| E | 0 | E | 1C | 2A | 38 | 46 | 54 | 62 | 70 | 7E | 8C | 9A | A8 | B6 | C4 | D2 |
| F | 0 | F | 1E | 2D | 3C | 4B | 5A | 69 | 78 | 87 | 96 | A5 | B4 | C3 | D2 | E1 |

by the number found in the row of multiplicant and in the column in which multiplyer is found. The second digit is taken as carry and added to the next place digit.

**Example 29.** *Multiply $3AB_{16}$ by $2C_{16}$ using hexadecimal multiplication table.*

**Solution.**

$$\begin{array}{r} 3AB \\ \times\ 2C \\ \hline 2C04 \\ 756 \\ \hline A164 \end{array}$$

hence the product of $3AB_{16}$ and $2C_{16}$ is $A164_{16}$.

### 1.9.4 Hexadecimal Division

The division process in hexadecimal is done in a similar way as for decimal numbers making use of hexadecimal multiplication table.

**Example 30.** *Divide $1EC89_{16}$ by $A5_{16}$.*

**Solution.**

$$\begin{array}{r} 2FC \\ A5\ )\ 1EC89 \\ 14A \\ \hline A28 \\ 9AB \\ \hline 7D9 \\ 7BC \\ \hline 1D \end{array}$$

remainder

Hence dividing $1EC89_{16}$ by $A5_{16}$, we get $2FC_{16}$ as quotient and $1D_{16}$ as remainder.

**Example 31.** *Divide $E03B_{16}$ by $89_{16}$*

$$\begin{array}{r} 1A3 \\ 89\ )\ E03B \\ 89 \\ \hline 573 \\ 55A \\ \hline 13B \\ 13B \\ \hline 0 \end{array}$$

remainder

Hence by dividing $E03B_6$ by $89_{16}$, we get $1A3_{16}$.

**Example 32.** *Perform the following arithmetic operations*

(*a*) $AC2B5_{10} + DEFA4_{16}$

(*b*) $F25CF - 9BAC4$

(*c*) $FCD5 \times 3BA8$

(*d*) $15FFD97BE \div 3AB6$

(*e*) $1BB9EFD4 \div A2C3$

**Solution.**

### 1.13.2 1's Complement Representation

The 1's complement of a binary number is obtained by changing each 0 to a 1, and each 1 to a 0. This complemented value represents the negative of the original number. In hardware or digital systems it is very easy to obtain the 1's complement by simply feeding all bits through invertors.

*In 1's complement system the positive numbers are represented in a similar way as in sign magnitude representation. However, the negative numbers in 1's complement representation are expressed in a different way.* For example, the number – 6 is represented as 1,001. The MSB (the

bit appearing before the comma) is the sign bit. To obtain this representation, as stated above, replace every 1 by a 0 and every 0 by a 1 in the binary representation of the number +6 (0, 110). This process of replacing 0 bit by a 1 and 1 bit by a 0 is called bit complementing. In general, for a binary number of $n$ bits (excluding the sign bit) its 1's complement can be obtained by the following relation.

1' complement of a binary number 0, $X$

$$= 1, (2^n - 1 - X)$$

For example, the complement of number (+12), 0, 1100 can be obtained by substituting $n = 4$ (number of bits excluding sign bit) in the above relation *i.e.*

1' complement of binary number

$$0,1100 = 1, (2^4 - 1 - 1100)$$
$$= 1, (10000 - 1 - 1100)$$
$$= 1, (1111 - 1100)$$
$$= 1,0011$$

*The same result is obtained by bit complementing* 0,1100 *i.e.* (changing 0's to 1 and 1's to zero). In table 1.16 the decimal equivalents of a few 4 bit 1's complement numbers are shown. Which demonstrates bit by bit complement property between positive and negative numbers.

**TABLE 1.16 Four-bit 1's complement numbers and equivalent Decimal numbers**

| *Decimal* | *1,s complement* | *Decimal* | *1,s complement* |
|---|---|---|---|
| +0 | 0,000 | −0 | 1,111 |
| +1 | 0,001 | −1 | 1,110 |
| +2 | 0,010 | −2 | 1,101 |
| +3 | 0,011 | −3 | 1,100 |
| +4 | 0,100 | −4 | 1,011 |
| +5 | 0,101 | −5 | 1,010 |
| +6 | 0,110 | −6 | 1,001 |
| +7 | 0,111 | −7 | 1,000 |

### 1.13.3 Addition and Subtraction in 1's Complement Notation

First we shall discuss *the addition rules for binary numbers represented in the notation of 1's complement assuming the numbers to be 4-bit.* For adding two positive numbers we add the binary

numbers including their sign bit. For example, we have for the sum of + 4 and + 9

$$+4 = 0,0100$$
$$+9 = 0,1001$$
$$+13 = 0,1101$$

However, if we add two numbers whose sum exceeds 15 we find this addition gives incorrect answer as is clear from the addition of +9 and +10

$$+9 = 0,1001$$
$$+10 = 0,1010$$
$$= 1,0011$$ which is not equal to 19.

For adding + 9 and + 10 if we use 8-bits we have

$$+9 = 0,0001001$$
$$+10 = 0,0001010$$
$$= 1,0010011$$ which is equal to 19.

For adding eight bit number the sum should not exceed 255. In general, for adding positive numbers of $n$ bit this rule will apply only if the sum does not exceed $2^n - 1$.

The main advantage of 1's complementary representation is that the subtraction can be done as addition of the 1's complement of the number. For example, the subtraction of 7 from 21 can be carried out as follows.

$$21_{10} - 7_{10} = 21 + (-7)$$

In 1's complement eight-bit form we have

$$00010101 = +21$$
$$+\ 11111000 = -\ 7$$
Sum neglecting carry $00001101 = +13$
$$+\ \boxed{1}$$
End around carry
$$00001110 = 14$$

In the above example the subtraction of 7 from 21 is done as the addition of 21 and (–7). We find on adding +21 and –7 we get 13 which is one short than the actual answer +14, if we neglect the carry generated. To get the correct answer we add the carry generated to the binary sum which we have got by neglecting the carry generated in the addition and this carry generated in the addition, would be added to the least significant place of the sum obtained neglecting the carry.

When we add a positive number with a negative number there are two possibilities either the answer will be positive or negative. If the sum of the two numbers including sign bits results in an overflow *i.e.* there is end around carry beyond the sign bit the answer is positive and the result is obtained by adding carry to the sum obtained and is the right answer with the right sign as was the case in the above example.

However, if on adding a positive and a negative number there is no overflow *i.e.* if no carry is generated the result is negative and result obtained is correct as it is. This will be clear from the following example:

**Example 50.** *Add –9 and +4 using 1's complement notation.*

**Solution.** In 1's complement notation we have

$$-9 = 1,0110$$
$$+4 = 0,0100$$
Sum $$-5 = 1,1010$$

To check the result find the 1's complement from the sum (as it is negative number) which is 00101 or +5 hence 1,1010 is –5 and is correct answer.

If we add two negative numbers and there is a carry generated, this carry is added to the LSB and the result so obtained is the right answer with the right sign.

**Example 51.** *Add –9 and –5.*

**Solution.**

$$-9 = 1,0110$$
$$-5 = 1,1010$$
$$\boxed{1}\ 1,0000$$
Add carry generated 1
$$-14 \qquad 1,0001$$

Again as stated earlier this rule holds good only if the sum of the two numbers does not exceeds 15 for 4 bit operand and 255 for 8 bit operand.

Consider the addition of 8 and – 8

$$-8 = 1,0111$$
$$+8 = 0,1000$$
$$-0 = 1,1111$$

From this we find that the one's complement of zero. Hence there are two possible representation for zero in one's complement representation system *i.e.* + 0 is 0,0000 and – 0 is 1,1111.

Because of dual representation for zero and the necessity of the end around carry, 1's complement arithmetic is not used for performing subtraction in digital systems. For this we use 2's complement representation.

### 1.13.4 2's Complement Representation

The 2's complement of a binary number is a binary number which is obtained by adding 1 to the 1's complement of the number *i.e.*

2's complement = 1's complement + 1

For example, the 2's complement of 1010 can be obtained by first finding its 1's complement which is 0101 and then adding 1 to it i.e. 0101 + 1(= 0110) hence the 2's complement of 1010 is 0110. Table 1.17 shows few 4-bit 2's complement numbers with their decimal equivalents.

**TABLE 1.17 4-Bits 2's complement numbers with decimal equivalents**

| Decimal | 2's complement | Decimal | 2,s complement |
|---|---|---|---|
| + 0 | 0,000 | – 1 | 1,111 |
| + 1 | 0,001 | – 2 | 1,110 |
| + 2 | 0,010 | – 3 | 1,101 |
| + 3 | 0,011 | – 4 | 1,100 |
| + 4 | 0,100 | – 5 | 1,011 |
| + 5 | 0,101 | – 6 | 1,010 |
| + 6 | 0,110 | – 7 | 1,001 |
| + 7 | 0,111 | – 8 | 1,000 |

In another method for a number of $n$ bits say $x$, the 2's complement is given by the following relation

2's complement of $x = (2^n - x)$

*There is another method of obtaining 2's complement of a number in which we scan the number from right to left and complement all the bits appearing after the first appearance of a 1 i.e.* starting from LSB and moving from right to left, copy each bit including first 1 bit encountered, then complement the remaining bits.

From table 1.17 we observe some very interesting characteristics of 2's complement numbers which can be listed as follows :

1. There is only one zero.
2. The 2's complement of zero is zero.
3. A negative number can be converted into a positive number by finding its 2's complement.
4. The MSB or the left most bit indicates the sign. If it is 1 the number is negative and if 0 the number is positive. This bit can not be used for expressing quantity.
5. There are 7 positive integers, eight negative and one zero making total number of states 16. For a binary word of $n$ bits in 2's complement representation, there will always be $2^{n-1}-1$ positive integers $2^{n-1}$ negative integers and one zero for a total $2^n$ unique states.
6. The 1's in a positive number and 0's in a negative number contains significant information.

**Example 52.** *Express – 6 in 2's complement form.*

**Solution. First method.**

Binary equivalent of the positive number *i.e.* 6 = 00000110
1's complement of the number = 11111001
Add 1 + 1
2's complement of the number = 11111010

**2nd method.**

Considering the word length $n$ (= 8)
$2^n$ $(= 2^8)$ = 1000 0000
Subtract $x$ (= 6) = 0000 0110
Result 1111 1010

**3rd method.**

Original number
Copy upto first 1 = 00000110
Complementing remaining bits = 10
= 11111010

If we compare the results obtained by all the three methods we find the result is same in every case.

**Example 53.** *Express – 18 in 2's complement form.*

**Solution.**

**First method**

Positive of the number in binary = 00010010

1's complement of the number = 11101101

Add 1 = + 1

2's complement of the number = 11101110

**2nd method.**

Considering word length = 8(= $n$)

$2^n = 2^8$ = 100000000

Subtracting 18 −00010010

11101110

**3rd method.**

Original number 00010010

Copying upto first 1 10

Complementing remaining bit 11101110

Again we see by all the three methods we get the same result.

**Example 54.** *Express the following in 8-bit 2's complement representation.*

*(i)* $-128_{10}$ *(ii)* $-45_{10}$ *(iii)* $-69_{10}$ *(iv)* $-101_{10}$ *(v)* $-39_{10}$.

**Solution.**

(i) Original number ($128_{10}$) = 1000 0000

Copying upto first 1 = 1000 0000

there is no bit for complementing

(ii) Original number ($45_{10}$) = 0010 1101

Copying upto first 1 = 1

Complementing the remaining bits = 1101 0011

(iii) Original number ($69_{10}$) = 0100 0101

Copying upto first 1 = 1

Complementing the remaining bits = 1011 1011

(iv) Original number ($101_{10}$) = 0110 0101

Copying upto first 1 = 1

Complementing the remaining bits = 1001 1011

(v) Original number ($39_{10}$) = 0010 0111

Copying upto first 1 = 1

Complementing the remaining bits = 1101 1001

### 1.13.5 Addition and Subtraction of Numbers in 2's Complement Representation

For adding two positive numbers the situation is identical to the one described for 1's complement notation. However, when we add two numbers out of which one is positive and other negative, the result can be positive or negative. The addition of such numbers is carried out ignoring overflow if any or by ignoring the carry generated, if any. As in earlier case, the sign bit is treated as the part of the number. After performing addition the bit in the sign bit position represents the correct sign bit.

**Example 55.** *Add +9 and − 8 using 2's complement in 8 bits.*

**Solution.**

+9 = 0 0001001

−8 = 1 1111000 (+ 8 = 0 0001000)

Sum = [1] 0000001

ignore this carry

or = 0 0000001

Hence the addition of + 9 and − 8 gives us 1 which is correct.

**Example 56.** *Add −18 and +17 using 2's complement.*

**Solution.**

+ 17 = 0001 0001

− 18 = 1110 1110

Sum = 1111 1111

To check the result take 2's complement of 1111 1111 (as this is negative number) which is 0000 0001 which shows 1111 1111 is equal to − 1. Hence the result obtained is correct.

**Example 57.** *Add +128 and −130 using 2's complement.*

**Solution.**

+ 128 = 1000 0000

− 130 = 0111 1110

Sum = − 2 = 1111 1110

To check the result convert this negative number obtained by suming, into a positive number. For this find 2's complement of 1111 1110 which is 0000 0010 which is 2. Thus 1111 1110 is equal to − 2 and is the correct answer.

When we add two negative numbers in 2's complements notation an overflow bit will result and this bit or carry generated is ignored. The sign bit represents the correct sign. However, one should be careful that the sum is within the allowed range of the answer. For example, for 4-bit numbers the answer should not exceed 15. In case the sum is beyond the permitted range the sign bit will become zero indicating an error in the answer. The ease with which the addition and subtraction can be performed in 2's complement notation has made this representation most common for most of the digital system.

**Example 58.** *Add –15 and – 20.*

**Solution.**

$-15 = \quad 1111\ 0001$

$-20 = \quad 1110\ 1100$

$-35 = \boxed{1}\ 1101\ 1101$

→ ignoring carry

$= \quad 1101\ 1101$

To check the result find the 2's complement of this negative result which is 0010 0011 which is 35. Hence the result 1101 1101 (= – 35) is correct answer.

**Example 59.** Subtract $11100_2$ from $10011_2$ using 2's complement method.

**Solution.** $+ 19_{10} = 010011_2$

and $+ 28_{10} = 011100$ and

$- 28_{10} = 100100$ (in 2's complement)

Hence

$$\begin{array}{lll} 010011 & = & + 19_{10} \\ \underline{100100} & = & \underline{-28_{10}} \\ 110111 & = & -9 \end{array}$$

As this is negative number (110111) taking its 2's compliment we have 001001 ( = $9_{10}$ which is the magnitude of the difference).

## 2.1 INTRODUCTION

The Digital circuits and computers process data which consists of 0s and 1s i.e., which are in binary format because of bistable nature of digital electronic circuits. However, in practice, these circuits are required to handle data which may be numeric, alphabets or special characters. This requires the conversion of the incoming data into binary format before it can be processed. There are various possible ways of doing this and this process is called *encoding*. To achieve the reverse of it, we use *decoders*. This chapter discusses some of the more commonly used binary codes.

In general, the numbers in a digital system or computer are used in coded form and this is done to achieve

(i) To represent numeric or alpha-numeric or special characters in only binary digits i.e. 0 or 1.

(ii) To check whether a character transmitted in the coded form is correctly received if not then to correct it i.e. for detecting and correcting errors.

Though a number of codes are used but here we discuss only a few most commonly used codes.

## 2.2 WEIGHTED BINARY CODES

There are two types of binary codes in use i.e. **weighted** and **non-weighted**. In weighted codes, for each position (or bit), there is specific weight attached. For example, in binary number, each bit is assigned particular weight $2^n$ where $n$ is the bit (or position) number for $n = 0, 1, 2, 3, 4$ the weights are 1, 2, 4, 8, 16 respectively.

### 2.2.1 Binary Coded Decimal (BCD)

In earlier section, we discussed how we can covert decimal numbers into binary or *vice versa*. There is another method for representing decimal numbers using binary digits and is called *Binary Coded Decimal* representation or BCD in short. BCD is a weighted code. In weighted codes, each succes- sive digit from right to left, represents weights equal to some specified value and to get the equivalent decimal number we sum the products of these weights by the corresponding binary digit. Several weighted codes are possible (a few are shown in table 2.2 but the most common is 8421. Because 8421 BCD is the most natural amongst the other possible codes. It is often referred to as BCD without describing it and when we refer to the BCD code we mean 8421 code.

In 8421 code each decimal digit is expressed by its 4 bit binary equivalent. For example, 2735 can be expressed in its binary equivalent as follows:

| 2 | 7 | 3 | 5 |
|---|---|---|---|
| 0010 | 0111 | 0011 | 0101 |

Hence in 8421 code, 0010 0111 0011 0101 stands for 2735. Consider another example 9865 which can be coded as follows:

| 9 | 8 | 6 | 5 |
|---|---|---|---|
| 1001 | 1000 | 0110 | 0101 |

For representing the number 9865 in 8421 code, we have changed each decimal digit to its binary equivalent. Table 2.1 shows numbers in 8421 code along with equivalent decimal.

The 8421 code is similar to binary for numbers upto 9 and the weights of the groups are 8, 4, 2, 1. As it is clear from the table the largest 4 bit groups is 1001 as it corresponds to 9. The 8421 code is a mixed base code as it is binary with in the group of 4 bits and decimal from group to group. Table 2.2 shows many other 4 bit codes. The same procedure i.e. decimal

numbers greater than 9 are encoded taking one digit at a time.

**TABLE 2.1 Showing BCD number and their equivalent Binary and decimal number**

| 8421(BCD) | Binary | Decimal |
|---|---|---|
| 0000 | 0000 | 0 |
| 0001 | 0001 | 1 |
| 0010 | 0010 | 2 |
| 0011 | 0011 | 3 |
| 0100 | 0100 | 4 |
| 0101 | 0101 | 5 |
| 0110 | 0110 | 6 |
| 0111 | 0111 | 7 |
| 1000 | 1000 | 2 |
| 1001 | 1001 | 9 |
| 0001 0000 | 1010 | 10 |
| 0001 0001 | 1011 | 11 |
| 0001 0010 | 1100 | 12 |
| 0001 0011 | 1101 | 13 |
| 0001 0100 | 1110 | 14 |
| 0001 0101 | 1111 | 15 |
| .... .... | .... | .. |
| .... .... | .... | .. |
| 1001 1000 | 1100010 | 98 |
| 0001 0000 0000 | 1100100 | 100 |
| 0001 0000 0010 | 1100110 | 102 |
| .... .... .... | .... | ... |
| .... .... .... | .... | ... |
| 0101 0111 0110 | 1001000000 | 576 |
| 0101 0111 1000 | 1001000010 | 578 |

For example, decimal 567 can be encoded as follows in various 4 bit codes

| decimal number → | 5 | 6 | 7 |
|---|---|---|---|
| 5421 code → | 1000 | 1001 | 1010 |
| 6311 code → | 0111 | 1000 | 1001 |
| 4221 code → | 0111 | 1100 | 1101 |

In Table 2.2 all the codes except the last two ($8\overline{421}$) and ($7\overline{421}$) use positive weights. The last two uses positive as well as negative weights. For example, number 1011 can be decoded as follows in 8421 code.

| number | 1 | 0 | 1 | 1 |
|---|---|---|---|---|
| weight | +8 | +4 | –2 | –1 |

Sum of the product of digit and its weight

$= 1 \times 8 + 0 \times 4 + 1 \times (-2) + 1 \times (-1)$

$= 8 + 0 - 2 - 1 = 5$

**Example 1.** *Convert the following 8421 BCD numbers to their decimal equivalent*

(*a*) 0101 0100 0011 (*b*) 0011 0010 . 1001 0100.

**Solution** (*a*) The decimal equivalent of the BCD number are as follows:

BCD → 0101 0100 0011

↓ ↓ ↓

Decimal 5 4 3 = $543_{10}$

(b) BCD → 0011 0010 · 10001 0100

↓ ↓ ↓ ↓

Decimal 3 2 9 4 = $32.94_{10}$

**TABLE 2.2 Some 4 bit codes with decimal equivalent**

| Decimal | 7421 | 6311 | 5421 | 5311 | 5211 | 4221 | 3321 | 2421 | $8\overline{421}$ | $7\overline{421}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 1 | 0001 | 0001 | 0001 | 0001 | 0001 | 0001 | 0001 | 0001 | 0111 | 0111 |
| 2 | 0010 | 0011 | 0010 | 0011 | 0011 | 0010 | 0010 | 0010 | 0110 | 0110 |
| 3 | 0011 | 0100 | 0011 | 0100 | 0101 | 0011 | 0011 | 0011 | 0101 | 0101 |
| 4 | 0100 | 0101 | 0100 | 0101 | 0111 | 1000 | 0101 | 0100 | 0100 | 0100 |
| 5 | 0101 | 0111 | 1000 | 1000 | 1000 | 0111 | 1010 | 1011 | 1011 | 1010 |
| 6 | 0110 | 1000 | 1001 | 1001 | 1001 | 1100 | 1100 | 1100 | 1010 | 1001 |
| 7 | 1000 | 1001 | 1010 | 1011 | 1011 | 1101 | 1101 | 1101 | 1001 | 1000 |
| 8 | 1001 | 1011 | 1011 | 1100 | 1101 | 1110 | 1110 | 1110 | 1000 | 1111 |
| 9 | 1010 | 1100 | 1100 | 1101 | 1111 | 1111 | 1111 | 1111 | 1111 | 1110 |

**Example 2.** *Convert the following decimal numbers to their 8421 BCD equivalents :*

(a) 897 (b) 214.83

**Solution.**

(a) Decimal 8 9 7

↓ ↓ ↓

BCD → 1000 1001 0111

∴ BCD equivalent is 100010010111

(b) Decimal 2 1 4 · 8 3

↓ ↓ ↓ ↓ ↓

0010 0001 0100 · 1000 0011

BCD equivalent is 001000010100 · 10000011

**Example 3.** *Convert the following binary numbers to their 8421 BCD equivalent*

(a) 1010. 001 (b) 10110001 . 11

**Solution.**

(a) Binary →1 0 1 0 . 0 0 1

↓ ↓ ↓ ↓ ↓ ↓ ↓

Decimal →8+ 0+ 2+ 0 + 0 + 0+ 0.125 = 10.125

BCD →0001 0000 . 0001 0010 0101

(b) Binary →1 0 1 1 0 0 0 1 . 1 1

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Decimal →128+0+32+16+0+0+0+1+ · 0.5+ 0.25

= 177.75

BCD → 0001 0111 0111 . 0111 0101

**Example 4.** *What is the 4221 BCD equivalent of 98 ?*

**Solution.** Decimal → 9 8

4s2s 2s 1s 4s 2s 2s 1s

4221 BCD → 1 1 1 1 1 1 1 0

Hence, 4221 BCD equivalent of decimal 98 is 11111110

**Example 5.** *What is the 5421 of BCD equivalent of 75*

**Solution.** Decimal → 7 5

5s 4s 2s 1s 5s 4s 2s 1s

5421 BCD → 1 0 1 0 1 0 0 0

Hence, 5421 BCD equivalent of decimal 75 is 10101000.

**Example 6.** *Typically, digital thermometers use BCD to drive their three digit displays. What 12-bits are sent to the display for a temperature of 136 degrees.*

**Solution.** Convert 136 decimal to BCD Code i.e.,

$136_{10}$ = 0001 0011 0110

**Example 7.** *16 bit register contains the following*

0100 1001 0101 . 0111

*interpret the contents as*

*(a) BCD number*

*(b) two ASCII characters.*

**Solution :** (a) The BCD equivalent is (495.7)

0100 1001 0101 0111

4 9 5 7

(b) According to table 2.6 the ASCII codes are

$0100\ 100_2$ → I (ASCII Code)

0101 0111 → W (ASCII Code)

### 2.2.2 BCD Addition

Addition of BCD (8421) is performed by adding two digits of binary, starting from the least significant digit. In case if the result is an illegal (not permissible) code or if there is a carry out of one then add 0110(6), and add the resulting carry to the next most significant digit.

**Example 8.** *Add 647 and 487 in BCD code*

**Solution :** In binary, we have for this sum

0110 0100 0111

\+ 0100 1000 0010

Sum of right most group in binary 0111

0010

1001

As the carry out is zero and the sum is 1001 (=9) is a permitted code, we proceed to the next group (middle group) 0100

1000

1100

The carry out is zero, but 1100 is illegal code ($1100_2 = 12_{10}$) hence, we have to add 0110 (6) to the result 1100

0110

1 0010

↳ Carry out

We shall consider the sum as 0010 with one carry to be added to the next group. Now, consider the third group (MSB) sum

$$\begin{array}{r} 0110 \\ 0100 \\ \hline +1 \\ \hline 1011 \end{array} \quad \rightarrow \text{(one of carry from second group)}$$

The carry out is zero but 1011 is illegal code so we add 0110 i.e.,

$$\begin{array}{r} 1011 \\ 0110 \\ \hline 10001 \end{array}$$

The final sum is as follows

| | | | | |
|---|---|---|---|---|
| 647 | = | 0110 | 0100 | 0111 |
| 482 | = | 0100 | 1000 | 0010 |
| 1129 | = | 0001 | 0001 | 0010 | 1001 |

We observe that binary sum is same as decimal result.

## 2.3 NON-WEIGHTED CODES

In this Code, there is no positional weighted, i.e. each position within the binary number is not assigned a prefixed value.

There are two types of codes in non-weighted codes.

### 2.3.1 Excess-3 Code

There is another important BCD code. For converting decimal number in excess-3 code (also called XS-3). We add three to the each decimal digit before converting it into equivalent binary. The XS3 code is related to the 8421 BCD code because of its binary coded decimal nature i.e. each 4 bit group in XS-3 code is equal to a specific decimal digit. This will be clear from the following examples.

Table 2.3 shows XS-3 code along with their complements and decimal number.

**TABLE 2.3 Excess-3 code for decimal numbers**

| Excess-3 code | Complement of excess-3 code (9's complement) | Decimal | 8421 BCD |
|---|---|---|---|
| 0011 | 1100 | 0 | 0000 |
| 0100 | 1011 | 1 | 0001 |
| 0101 | 1010 | 2 | 0010 |
| 0110 | 1001 | 3 | 0011 |
| 0111 | 1000 | 4 | 0100 |
| 1000 | 0111 | 5 | 0101 |
| 1001 | 0110 | 6 | 0110 |
| 1010 | 0101 | 7 | 0111 |
| 1011 | 0100 | 8 | 1000 |
| 1100 | 0011 | 9 | 1001 |
| 0100 0011 | 1011 1100 | 10 | 0001 0000 |
| 0100 0100 | 1011 1011 | 11 | 0001 0001 |

**Example 9.** *Express 129 to an excess-3 number*

**Solution:** Before converting each group into binary, add 3, i.e.

| 1 | 2 | 9 |
|---|---|---|
| +3 | +3 | +3 |
| 4 | 5 | 12 |
| ↓ | ↓ | ↓ |
| 0100 | 0101 | 1100 |

Hence the code for $129_{10}$ in the excess-3 code is 0100 0101 1100.

**Example 10.** *Convert the XS3 number 1001 1010 to its decimal equivalent.*

**Solution.**

| | | | |
|---|---|---|---|
| XS 3 → | 1001 | 1010 | |
| ↓ | | | |
| BCD | −0011 | −0011 | ←Subtract 3 |
| ↓ | 0110 | 0111 | |
| | ↓ | ↓ | |
| Decimal | 6 | 7 | ←Convert to decimal |

Hence the decimal equivalent number is 67

### 2.3.2 Excess-3 Addition

The main problem in BCD (8421) was addition of the numbers whose sum exceeds 9. This excess-3 code has very interesting properties when used in addition. For adding in excess-3 binary number are added as usual and if there is a carry out, add 0011 and if there is no carry out from the four bit group, subtract 0011. Following examples will illustrate both these cases.

**Example 11.** *Add $6_{10}$ and $3_{10}$ in XS-3 code*

**Solution.**

| | | |
|---|---|---|
| Excess 3 for 6 | = | 1001 |
| Excess 3 for 3 | = | 0110 |
| Sum | = | 1111 |
| Since we are getting no carry subtract | | −0011 |
| Excess-3 for sum (12) | = | 1100 |

**Example 12.** *Add 36 and 39 in XS-3 form.*

**Solution**

| | A | B | |
|---|---|---|---|
| XS-3 for 36 = | 0110 | 1001 | 36 |
| XS-3 for 39 = | + 0110 | 1100 | + 39 |
| | 1101 | 0101 | 75 |
| Subtract and add 3 as per rule stated above | − 0011 | + 0011 | |
| XS-3 for 75 | 1010 | 1000 | |

In the above example in column B we add 1001 and 1100 and get 0101 with a carry of 1 for column A. In column A we add 0110 and 0110 and the carry we got from column B and get 1101 with no carry. As there is carry the result in column B is back in 8421 code. The result in column A is still in excess-6 form as this column does not produce any carry. To get the result in XS-3 we should add 3 to the sum obtained in column B as there is a carry and subtract 3 from the sum obtained in column A as there is no carry. The final answer is 1010 1000 which is XS-3 number for 75. When carry is generated, the sum is expressed automatically in 8421 code. For example, consider the addition of 6 and 5.

| | | |
|---|---|---|
| XS-3 for 6 | = | 1001 |
| XS-3 for 5 | = | 1000 |
| Sum (11) | = | 0001 0001 (in 8421 code) |

This type of codes are self complementing which means if we change all the 1's to zeros and all the zero's to 1 in the number, the resulting number is the 9's complement of the original number. For example, the decimal number 7 has the XS 3 code version as 1010. Its complement ($\overline{1010}$ = 0101) in XS-3 code 0101 represents 2 which is the 9's complement of 7. XS-3 codes are very useful in digital systems as this requires very simple electronic circuit for performing subtraction operation for such codes. The XS-3 code is very useful in arithmetic circuit as it is very easy to complement. If each bit is complemented the resulting 4-bit word will be 9s complement of the number which can be used to perform subtraction in adders.

### 2.3.3 Gray Codes

In Gray Codes, we assume as one advances from one number to next, only one bit is changed. Not only change in one bit takes place every time decimal number is incremented by one from 0 through 9 but even for the change from 9 to 0 also has only one bit change. Table 2.3 shows Gray Codes for decimal numbers from 0 to 12.

**TABLE 2.3 Gray Codes**

| *Decimal digit* | *Binary* | *Gray Codes* |
|---|---|---|
| 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 |

### 2.3.4 Binary to Gray Code Conversion

The binary numbers can be converted into Gray Code equivalent by the following method:

1. Record the most significant bit.
2. Add this bit to the next position, neglecting carry if any record the sum.
3. Continue recording sums until completed and LSB is reached The Gray code number will always have the same number of bits as the binary number.

**Example 13.** *Convert 1100 $1010_2$ into gray code.*

**Solution.** First record the most significant bit. Then add this most significant bit to the next most significant bit (1 + 1) and ignoring the carry generated record the sum, as 0 bit. Now add the second most significant bit to the third significant bit (1 + 0 = 1). Record it then add the third significant bit to the fourth

significant bit (0 + 0 = 0). Record it. In this way continue upto the LSB (1 + 0 = 1) as shown in Fig. 2.1 (a).

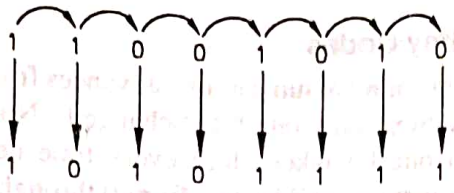



Fig. 2.1(a)

**Example 14.** *Convert binary $1001011_3$ into gray code.*

**Solution.** The conversion is shown in Fig. 2.1 (b)

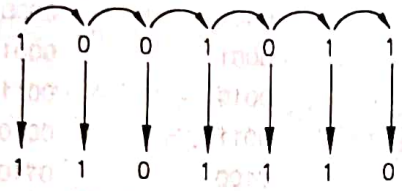



Fig. 2.1(b)

### 2.3.5 Gray Code to Binary Conversion

The gray code numbers can be converted into equivalent binary number as follows:

(i) Record the MSB.

(ii) Add the binary MSB to the next significant bit of the Gray Code.

(iii) Ignoring carries record the result.

(iv) Continue this process until you reach LSB.

**Example 15.** *Convert a Gray Code 1001 1011 to binary.*

**Solution.** First record the MSB of Gray Code. This will be the MSB of equivalent binary. Then add this MSB to the next most significant bit of the Gray Code number (1 + 0 = 1). Record it as the second most significant bit of equivalent binary. Add this second most significant bit of binary to the third most significant bit of equivalent binary. Add this third most significant bit of the Gray Code (1 + 0 = 1). Record sum as third most significant bit of equivalent binary. Add this third most significant bit of binary so obtained to the fourth most significant bit of the Gray Code (1 + 1 = 0) and ignoring the carry generated record it as the fourth most significant bit of the equivalent binary (0 in this case). Continue this process till the last between bit of equivalent binary is added to the LSB of the Gray Code as shown in Fig. 2.2.

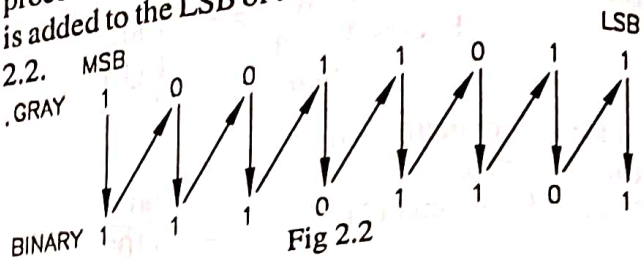



Fig 2.2

**Example 16.** *Convert a Gray Code 1110 0111 into binary.*

**Solution.** The conversion is shown in Fig. 2.3

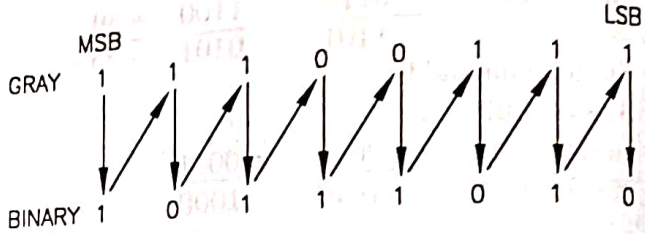



Fig. 2.3

## 2.4 ERROR DETECTING CODES

In digital systems a word consisting of group of bits is stored as a unit and moves from one unit to another. It is quite possible, when this word is transmitted say from one memory location to another or to an arithmetic unit, that an error may occur. This may happen due to any reason such as noise, by a transient, by an intermittent failure etc. This requires some method of detecting the errors. For example, it may be required to add 0100 1101 and 1100 0111. When the device transfers each of these words from the memory, it may transfer the first word as 0110 1101 (i.e. the third bit from the left of the first word is changed from 0 to 1). This will result in an error in the addition. For detecting such errors, we use a simple method of parity.

In this method, an additional bit known as parity bit is added with the numbers. For odd parity, this parity bit is set to 1 so that the sum of bits in the number is odd i.e. number of 1's in the number is odd. We can also use even parity which means the adding of the parity to the group of bits produce an even number of 1's. For example in number 011 0111 there are five 1's in this word i.e. we have odd number of 1's so we attach one extra bit having 1 and the changed number will be 01101111. In this number,

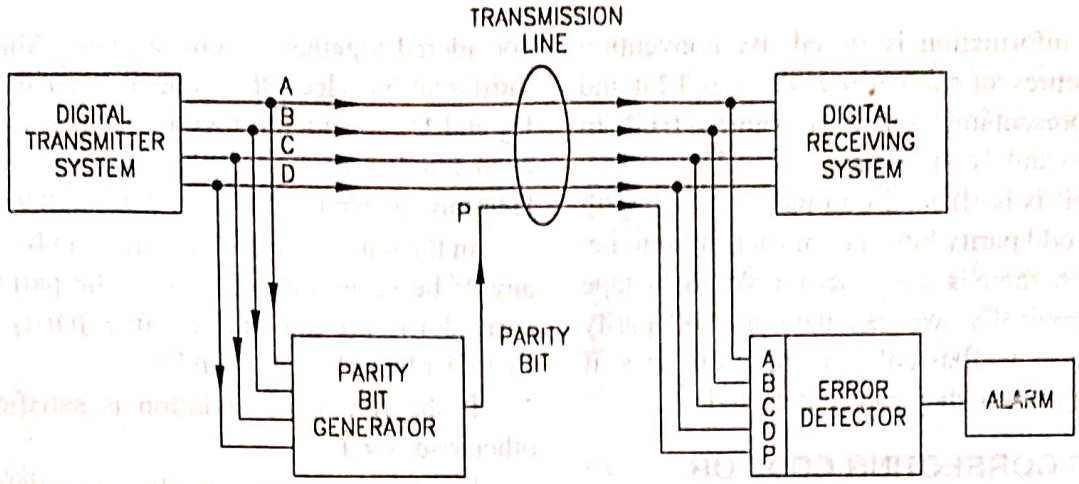



Fig. 2.4 Parity bit. Error detection in a data transmission system.

after adding parity bit we have even number of 1's. This word during transfer from one place to another can be checked for even parity. Table 2.4 shows odd as well as even parity bits for BCD numbers 0 through 9.

**TABLE 2.4 Parity Bit for BCD numbers**

| *Decimal digit* | *BCD Code (8421)* | *Added Parity bit* | |
|---|---|---|---|
| | | *even* | *odd* |
| 0 | 0000 | 0 | 1 |
| 1 | 0001 | 1 | 0 |
| 2 | 0010 | 1 | 0 |
| 3 | 0011 | 0 | 1 |
| 4 | 0100 | 1 | 0 |
| 5 | 0101 | 0 | 1 |
| 6 | 0110 | 0 | 1 |
| 7 | 0111 | 1 | 0 |
| 8 | 1000 | 1 | 0 |
| 9 | 1001 | 0 | 1 |

A digital transmission system is shown in the block diagram of Fig. 2.4. The transmitter is assumed to send a 4-bits (A, B, C, and D). This data bits are fed into a parity bit generator which sets the parity bit and this parity bit along with the four bits of the data are sent through the transmission line and routed to the receiving system. The four bit word along with the parity bit is fed to a circuit called error-detection circuit. In case if there has been some error during the transmission, this circuit activates the error alarm.

Odd as well as even parity are used and there is no preference over one another. For checking a number we check the number for its parity and this will detect if there has been a single error. The change in any bit will result in change in parity which can be detected on parity check. In this method, we can detect the error only if there is change only in one bit. If the error occurs at two places, this method can not be used. For example, if a word 0101 is transmitted as 1001 i.e. if first and second bit are interchanged (i.e. there are two errors one in first bit and second in second bit), the number of 1's remain same and the parity check can not be successfully applied. For overcoming this problem, we make use of dual parity or double-parity check.

For the storage devices such as magnetic tapes, drums, cores and paper tapes, parity checks are commonly used. Normally, the chances of having two errors are very small but in magnetic tapes it happens some time and for this to detect, we use dual parity check.

Fig. 2.5 shows a portion of magnetic tap on

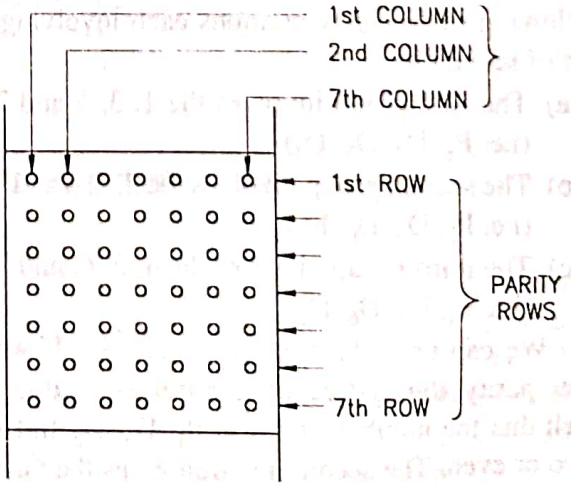



Fig. 2.5

which some information is stored. By convention dark circle (represent magnetised point) as 1 bit and light dots (represent ummagnetized point) as 0 bit. In the tape 7 rows and 7 columns are shown. In each row the seventh bit is horizontal odd parity bit. The 7th row contains odd parity bits one for each column i.e. to each column, there is one parity bit. When the tape is read electronically, we can have an odd parity check of rows as well as column and in this way, if there has been errors, they can be detected.

## 2.5 ERROR CORRECTING CODE OR HAMMING CODE

The parity check discussed above can detect an error which has crept in during the transmission of digital data i.e. parity bit indicates only that an error exists it does not tell which bit is incorrect nor it corrects the incorrect bit. To overcome this problem, another code called *Hamming Code* is used. This code detects an error and indicates which bit is in error. This incorrect bit then can be changed to its correct form. Because of this, *Hamming Code* is called self correcting code.

This code is used for correcting a single error on a massage of any length. For transmitting a four bit massage (BCD data), we use three check bits and the word format is:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | ← Bit number |
|---|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ | ←Allocated for |

In the above expression, $D$ represents the data bits and $P$ represents the parity bits or check bits. Out of $D_7$, $D_6$, $D_5$ and $D_3$ bits, $D_7$ is the most significant bit, i.e. the 4 -bit word is ($D_7$, $D_6$, $D_5$, $D_3$). The parity bit $P_1$, $P_2$ and $P_3$ are assigned values by making the following three parity relations each involving four out of seven bits.

(a) The first group involves the 1, 3, 5 and 7 bits (i.e. $P_1$, $D_3$, $D_5$, $D_7$).

(b) The second group involves the 2, 3, 6 and 7 bits (i.e. $P_2$, $D_3$, $D_6$, $D_7$).

(c) The third group involves the 4, 5, 6, and 7 bits (i.e. $P_4$, $D_5$, $D_6$, $D_7$).

We can use odd-even parity check. If we use even parity, the first equation requires a value of $P_1$ such that the number of 1's in $P_1$, $D_3$, $D_5$ and $D_7$ is zero or even. The second relation gives the value of $P_2$ such that the number of 1's in $P_2$, $D_3$, $D_6$ and $D_7$ considered together is zero or even. Similarly, the third relation selects the value of $P_4$ such that $P_4$, $D_5$, $D_6$ and $D_7$ considered together should have zero or even number of 1's. Table 2.5 shows even parity Hamming Code for numbers 0 through 9.

In the data received, the error can be detected in any of he seven bit positions. The parity check is carried for all the above three parity relations. Consider three bits, $x$, $y$ and $z$.

If the first parity relation is satisfied, $x = 0$, otherwise, $x = 1$.

If the second parity relation is satisfied, $y = 0$, otherwise, $y = 1$.

If the third parity relation is satisfied, $z = 0$, otherwise, $z = 1$.

The position of the error is given by the three binary bit x, y and z and expressed as

$$z\,y\,x$$

$x$ being the LSB and $z$ being the MSB.

**TABLE 2.5 Hamming Code for Numbers 0 through 9.**

| *Decimal number* | 7 | 6 | 5 | 4 | 3 | 2 | 1 ← *Bit No.* |
|---|---|---|---|---|---|---|---|
| | $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ ← *Bit Assigned* |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

**Example 17.** *Construct the even parity seven bit Hamming Code for a word 1011.*

**Solution.**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | ← Bit No. |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | $P_4$ | 1 | $P_2$ | $P_1$ | ← Bit |

For knowing the values of $P_1$, $P_2$ and $P_4$, we use the above three parity relations.

From first parity relation for bits 1, 3, 5 and 7 to have even parity $P_1$ should be a 1.

From second parity relation for bits 2, 3, 6 and 7 to have even parity $P_2$ should be a 0.

From third parity relation for bits 4, 5, 6 and 7 to have even parity $P_4$ should be a 0.

Hence, the final code is

| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**Example 18.** *A seven bit Hamming Code is received as 1000010. What was the code transmitted or what is the Correct Code?*

**Solution.** The code received

| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |

Applying first parity relation (for $P_1$, $D_3$, $D_5$ and $D_7$) we have $x = 1$ (as even parity is not observed).

Applying second parity relation (for $P_2$, $D_3$, $D_6$ and $D_7$) we have $y = 0$ (as even parity is observed).

Applying third parity relation(for $P_4$, $D_5$, $D_6$ and $D_7$) we have $z = 1$ (as even parity is observed)

Thus the position of the error is given by $zyx$ will be 101 i.e. the fifth data bit is having the error and it should be corrected as 1 in place of 0. The correct code is 1010010.

**Example 19.** *A seven bit Hamming Code coming out of transmission line is 0010100. Is there any error? If yes, in which data bit and what was the 4-bit data actually transmitted.*

**Solution.** The code received

| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |

Applying first parity relation for $P_1$, $D_3$, $D_5$ and $D_7$, we have $x = 0$ (as even parity is observed)

Applying second parity relation for $P_2$, $D_3$, $D_6$ and $D_7$, we have $y = 1$ (as even parity is not observed)

Applying third parity relation for $P_4$ $D_5$, $D_6$ and $D_7$ we have $z = 1$ (as even parity is not observed)

This shows that there is an error in the code received and it is at sixth bit position (i.e. $zyx = 110$). The correct bit is 1 and not 0 as transmitted. Hence, the data transmitted ($D_7$, $D_6$, $D_5$, $D_3$) was 0111 and it was received as 0011.

**Example 20.** *Encode data bits 1001 into a seven-bit even-parity Hamming Code.*

**Solution.**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | ← Bit No |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | $P_4$ | 1 | $P_2$ | $P_1$ | ← Bit |

Using first parity relation for bit 1, 3, 5, 7 we get $P_1$ as 0, using second and third parity relation for bit 2, 3, 6, 7 and 4, 5, 6, 7 respectively we get $P_2$ as 0 and $P_4$ as 1. Hence the 7-bit Hamming Code for 4-bit data 1001 is 1001100.

This seven bit Hamming Code can be used for detecting error and requires extra transmission lines and extra digital circuitry for parity bit generator, error-detector and error correcting circuits. This code can be used for more than 4-bit with the addition of more parity bits at each $2^n$ bit and holds for any length. For example, a 16-bit data requires six check bits (parity bits) at 0, 2, 4, 8, 16, 32 place value for any single bit error.

## 2.6 ALPHANUMERIC CODES

In transforming information from digital systems and to digital systems we need to process both numerals and alphabets and also special characters such as '.', '+', '$', '(', ')', '*' etc. For this, we use alphanumeric codes. These codes are used to represent alpha-numeric informations as a combination of 1's and 0's. The Devices such as punched card readers, teletypewriters, keyboards, paper tape readers, magnetic disc units, etc. use this code. For this several coding systems have been used ever since the invention of the first telegraph system by Morse in 1844. In telegraph codes, two systems – one International Morse code and American Morse code are in practical use.

The outgrowth of Morse code is code for teletypewriter. This code makes use of a five bit teletypewriter code. The telegraph codes and teletypewriter code are not very suitable for communicating with digital systems. To overcome this difficulties, following codes are used in digital systems such as computers.

## 2.7 ASCII CODE

The code known as American Standard Code for Information Interchange and written as ASCII and pronounced as "as-kee" was developed as orderly binary code applicable to alpha-numeric data. This code is used for printers and teletypewriters when interfaced with small computer systems. Basically, ASCII code is a 7 bit code but the 8th bit is usually added which is either set 0 or 1 or used as a parity bit. Table 2.6 shows this code.

**Table 2.6 ASCII Code (American Standard Code for Information Interchange)**

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

| | |
|---|---|
| ACK | ACKNOWLEDGE |
| BEL | BELL |
| BS | BACK SPACE |
| CAN | CANCEL |
| CR | CARRIAGE RETURN |
| DC1 | DEVICE CONTROL 1 |
| DC2 | DEVICE CONTROL 2 |
| DC3 | DEVICE CONTROL 3 |
| DC4 | DEVICE CONTROL 4 |
| DEL | DELETE |
| DLE | DATA LINK ESCAPE |
| EM | END OF MEDIUM |
| ENQ | ENQUIRY |
| EOT | END OF TRANSMISSION |
| ESC | ESCAPE |
| ETB | END OF TRANSMISSION BLOCK |
| ETX | END OF TEXT |
| FF | FORM FEED |
| FS | FILE SEPARATOR |
| GS | GROUP SEPARATOR |
| HT | HORIZONTAL TABULATION |
| LF | LINE FEED |
| NAK | NEGATIVE ACKNOWLEDGE |
| NUL | NULL OR ALL ZEROS |
| RS | RECORD SEPARATOR |
| SI | SHIFT IN |
| SO | SHIFT OUT |
| SOH | START OF HEADING |
| SP | SPACE |
| STX | START OF TEXT |
| SUB | SUBSTITUTE |
| SYN | SYNCHRONOUS IDLE |
| US | UNIT SEPARATOR |
| VT | VERTICAL TABULATION |

= P + K

## 3.5 LOGIC CIRCUITS OR LOGIC GATES

As the Boolean algebra is used for logical variables which have only two values 0 or 1 (true (on) or false (off)). This algebra is used extensively in digital electronics. These commonly used logic circuits consist of solid state electronic circuits which are available as IC (Integrated Circuits) and employ transistors, diodes and other electronic components.

Logic circuits are available for the following operations AND, OR, XOR and two more useful versions *i.e.* NAND and NOR operations. These circuits are also called *gates*. The logic gate is the basic building block in digital systems. As these gates operate with binary numbers, they are also called *binary logic gates*. Voltages used with logic gates will be either HIGH or LOW. In this book, HIGH voltage will mean a binary 1 and LOW voltage will mean a binary 0. As these gates are electronic circuits, they will respond to only HIGH voltages.

### 3.5.1 AND Gates

A transistor or diode in conducting state or cut off state is used to represent the two values of Boolean variable. Let us consider the electronic circuit as

shown in Fig. 3.4. The working of a transistor as a switch is explained in next chapter. Here input signal assumes a voltage of 0 or 5 volt. From Fig. 3.4, it is

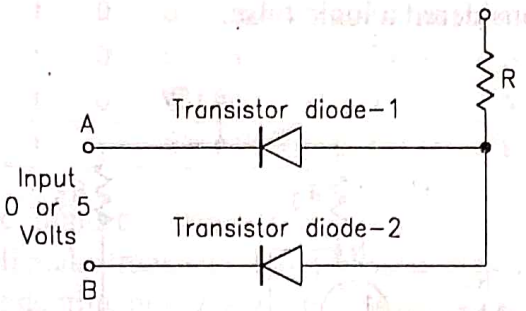



Fig. 3.4 AND gate

clear that point C will be +5 V only if both inputs to A and B are at +5 volts. If either of input to A or B is at 0 volt, the output will be 0. If we assign a value 1 to represent +5 volts (on or conducting) and a 0 to represent 0 volt (non-conducting or off) the output will be as shown in Table 3.13.

**TABLE 3.13 AND truth table**

| A | B | C |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

If we carefully look at Table. 3.13, it is seen that $C = A \cdot B$ or the above electronic circuit making use of two diodes with the particular arrangement of logic levels realizes the AND operator of Boolean algebra. The AND gate is called the "*All or nothing gate*".

In Fig. 3.5. the symbol for an AND gate is shown. This gate has a single output but may have any

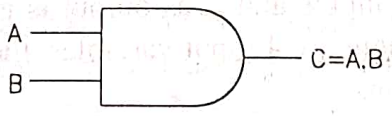



Fig. 3.5 Symbol for AND gate

number of inputs. In the electronic circuits normally available in the market, the voltage level of +5 volt is considered to be true and 0 level falls. In the above example this gate will give +5 volts at C as its output if and only if both the inputs at A and B are +5 volts. Under any other set of input conditions, the output C will be 0 V. Here it should be noted that input and output operate only at +5 volt and ground. Under ideal conditions, no other voltage (except 0 and +5 V) occurs at the input or output of the gate. However, in

practice, this is not so. Hence, in actual practice, any voltage above a certain level (Say 3 V) is considered a logic true any voltage below a certain level (say 1 V) is considered a logic false.

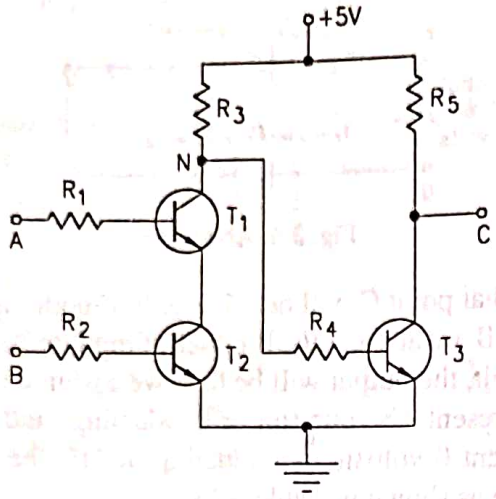



Fig. 3.6 Transistor AND gate

A transistor AND gate is shown in Fig. 3.6. It is clear from the circuit of this figure that when A and B are at +5 V, the transistors $T_1$ and $T_2$ will conduct the current and the point N will move to ground. When N is ground the transistor $T_3$ will cut off and make the point C to +5 V. In case if either of the points A or B are at ground level (0 V), either of the transistor $T_1$ or $T_2$ will be cut off and the point N will move to +5 V which will provide base current to the transistor $T_3$. Consequently the point C will become grounded.

As there can be any number of inputs to an AND gate, we have shown an AND gate with four inputs in Fig. 3.7. The truth table for this gate having A, B, C, D, as four inputs and E as output is given in Table 3.14. As there are 4 input variables, there will be 16 combinations.

From Table 3.14, it is clear that the output E of an AND gate will be 1 only if all the inputs are 1 for any other combination, the output will be 0.

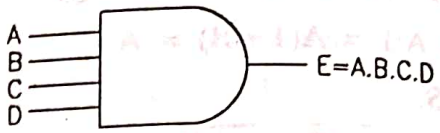



Fig. 3.7 Four Input AND gate

### 3.5.2 OR Gate

The OR gate is also called *"any or all" gate*. The OR gate is the combinational logic circuit which has only one output and may have any number of inputs. The output is 1 when any one or more than one of the input is 1 and the output is 0 only when all the inputs are 0. Fig. 3.8 shows the symbol of an OR gate as used in


Fig. 3.8 OR gate Symbol

electronics. If we assume +5 V as logical 1 and 0 V as logical 0, the OR gate may be realised as a device that gives the output as a logical 1 when any one of the or more than one of the inputs is logical 1. The output is logical 0 when all the inputs are logical 0. As stated above, the input to an OR gate can be any in number.

In Fig. 3.9. an OR gate with four inputs is shown. The truth table for such a gate will be as shown in Table 3.15.

**Table 3.14 Truth Table for a Four Input AND Gate**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input A → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Input B → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Input C → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Input D → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| E = (A.B.C.D) → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 3.15 Truth Table for a Four Input OR Gate**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input A → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Input B → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Input C → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Input D → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| E = (A +B + C + D) → | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

From this table it is clear that for an *OR gate the output is 0 only when all the inputs are zero and the output is 1 even if a single input variable is 1 or all are one.*

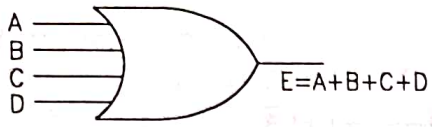



Fig. 3.9 Four input OR gate.

In Fig. 3.10, a diode OR gate is shown. It is clear from this circuit if at A +5 V input is applied, it will forward bias transistor diode-1 and the output voltage at C will be +5 V *i.e.*, if input A is 1, the output C will also be 1 irrespective of input B. Similarly, if input to B is 1, the output at C will also be 1, irrespective of input to A. The output C will be 0 only when both the transistor diodes are non-conducting *i.e.* both inputs are 0 *i.e.* A = 0 and B = 0. This shows that this circuit can realise Table 3.2 for OR gate.

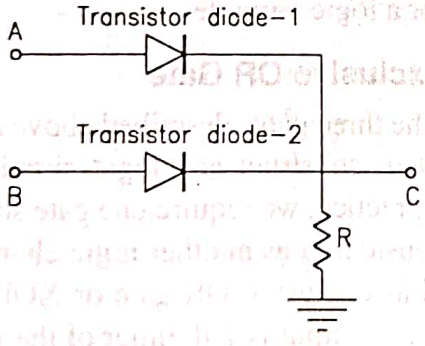



Fig. 3.10 Diode OR gate

In Fig. 3.11, a transistor OR gate is shown. If +5 V is applied to the input A, it will cause transistor T-1 to conduct and the point N will go to ground. This will result in the non-conducting of Transistor-3 and the point C will go to +5 V *i.e.* when input A is +5 V (1 or True) the output at C will also be +5 V (1 or True). If +5 V is applied to input B, it will cause the transistor-2 to conduct and this will result again to give +5 V at C *i.e.* the output will be +5 V. Now if both the inputs are grounded *i.e.* A and B are given 0 V, it will make transistor-1 and transistor-2 cut off *i.e.* non-conducting and this will cause the point N to go positive (+5 V) which results in the supply of current to the base of transistor-3. This results in output C going to ground *i.e.* if input A and B are 0, the output C will also be zero. From this, we find that such a circuit as shown in Fig. 3.11, satisfies the conditions given for OR gate above and thus can be used as OR gate.

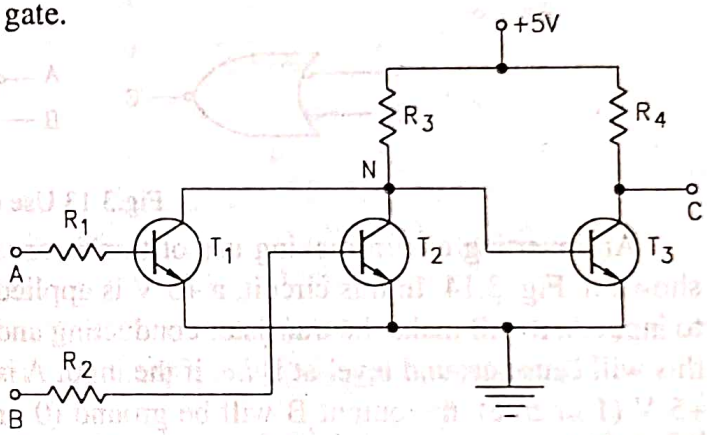



Fig. 3.11 Transistor OR Gate

### 3.5.3 The Inverter or NOT Gate

This is one of the gates which is essential in any basic set of logic functions. Its function is to give an output logic state which is different to the input state. As in logic system, only two states are possible 0 (false or off) and 1 (true or on), the NOT gate or inverter will produce an output 1 for 0 input and output 0 for input 1. Though in AND and OR gates we could have any number of inputs, but in case of inverter or NOT gate we have only one input and one output.

The logic diagrams for NOT operater as shown in Fig. 3.12 are called inverters. The symbol consists

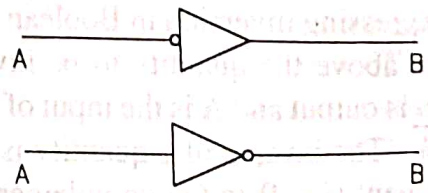



Fig. 3.12 Inverter or NOT gate symbol

of a triangle and a circle. In the figure two conventions are shown. As shown in Fig. 3.12(*a*), when a single circuit is used as inverter or for inversion alone, the triangle is included with the symbol. However, when the circuit is used for gating as well as for inversion, the circle is used in series with a gate input or output lead. Actually, the circle represents the inversion. The use of circle as inverter is shown in Fig. 3.13 for various examples along with their meaning.

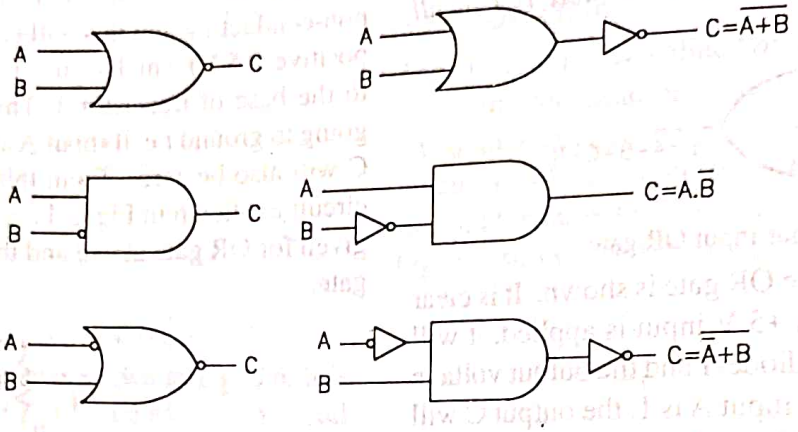



Fig.3.13 Use of circle as inverter

An inverting circuit making use of transistor is shown in Fig. 3.14. In this circuit, a +5 V is applied to input A, it will make the transistor conducting and this will cause ground level at B *i.e.* if the input A is +5 V (1 or true), the output B will be ground (0 or false). If the input A is connected to ground level, the transistor will not conduct and this will result +5 volt as output B.

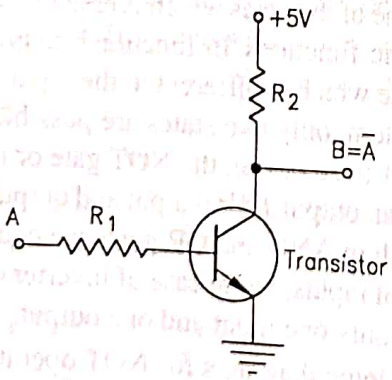



Fig. 3.14 Transistor inverter

For expressing inversion in Boolean algebra, we draw a line above the quantity to be inverted. For example, $\overline{B}$ is output and A is the input of an inverter, then $B = \overline{A}$. The inverse of a quantity is also called its complement (*i.e.* B is the complement of A). In case if we complement the complement of a quantity, we get the same quantity. If B is the complement of A and complement of B is C then C = A *i.e.*

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

The above can be proved by perfect induction. If A is 1, the complement of A, *i.e.* B will be 0. Now, C the complement of B will be equal to the complement of 0 *i.e.* 1. Similarly, if A is 0, the double complement $\overline{\overline{A}}$ (complement of complement is represented with two bars on the quantity) will be also 0. This is due to the fact that in Boolean algebra, we have only two values for a logic variable.

### 3.5.4 Exclusive OR Gate

Though the three gates described above are normally sufficient to construct any logic circuit with these gates. In practice, we require one gate so often that it can be considered as another logic element and that is named as exclusive-OR gate or XOR gate. In an OR gate, the output is 1 if either of the two inputs is one or both inputs are at 1. In exclusive OR gate, we have output as 1 when either of the input is 1, but it is not 1 when both the inputs are 1, *i.e.* it is 0 when both the inputs are 1. Unlike OR gate in XOR gate we have only two inputs.

Fig. 3.15 shows the symbol for an exclusive OR gate and Table 3.16 shows the truth table for an exclusive OR gate. From table we can say $C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$. As there are only two variables, we will have four combinations.

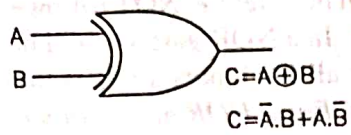



Fig. 3.15 Exclusive - OR gate symbol.

**Table 3.16 Truth Table for Exclusive OR Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| C = A ⊕ B → | 0 | 1 | 1 | 0 |

Logic symbol for a 3-input XOR gate and the corresponding truth table are shown in Fig. 3.16 and Table 3.17. Remember that an odd number of 1s generate a 1 output and even number of 1s in input gives 0 output (Table 3.17).

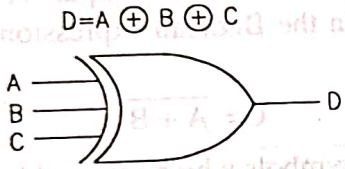



Fig. 3.16 Logic symbol for a 3-input NOR Gate.

**Table 3.17**

| INPUTS | OUTPUTS | INPUTS | OUTPUTS |
|---|---|---|---|
| C B A | D | C B A | D |
| 0 0 0 | 0 | 1 0 0 | 1 |
| 0 0 1 | 1 | 1 0 1 | 0 |
| 0 1 0 | 1 | 1 1 0 | 0 |
| 0 1 1 | 0 | 1 1 1 | 1 |

## 3.6 UNIVERSAL BUILDING BLOCKS NAND AND NOR GATES

### 3.6.1 The NAND Gate

This gate is an electronic circuit that can perform all the three logical functions, *i.e.* AND, OR and Inverter. This gate may be considered as equivalent to an AND gate followed by an inverter (*i.e.* AND NOT). However, in Boolean algebra, the convention is to represent such a gate as NOT AND *i.e.* to state NOT before the quantity to be inverted. This gate is usually called NAND.

*In NAND gate, we will have an output of logical 0 only when all the inputs are logical 1 otherwise the output is logical 1.* The output is denoted by $\overline{A \cdot B}$ when the inputs are A and B. The truth table for the NAND gate is shown in Table 3.18. For N input NAND gate $Y = \overline{X_1 \cdot X_2 \cdot X_3 \ldots X_N}$ where Y is the output and $X_1, X_2, X_3 \ldots X_N$ are N inputs.

**Table 3.18 Truth Table for Two Input NAND Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| A · B → | 0 | 0 | 0 | 1 |
| $\overline{A \cdot B}$ → | 1 | 1 | 1 | 0 |

In Fig. 3.17, a NAND gate circuit representation is illustrated. In Fig. 3.17(*a*) , this logic function,

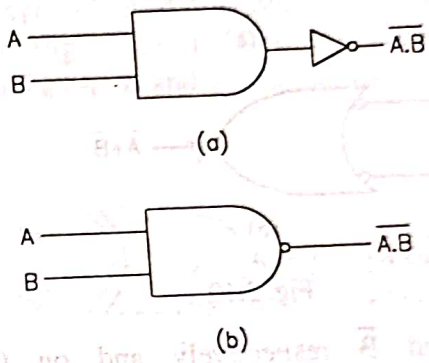



Fig. 3.17 NAND gate representation

is illustrated as an AND gate with an inverted output. The input to this gate A and B are first ANDed yielding A · B and then inverted giving $\overline{A \cdot B}$. The symbol shown in Fig. 3.17 (*b*) is reduced from an AND gate followed by an inverter. Fig. 3.18 shows

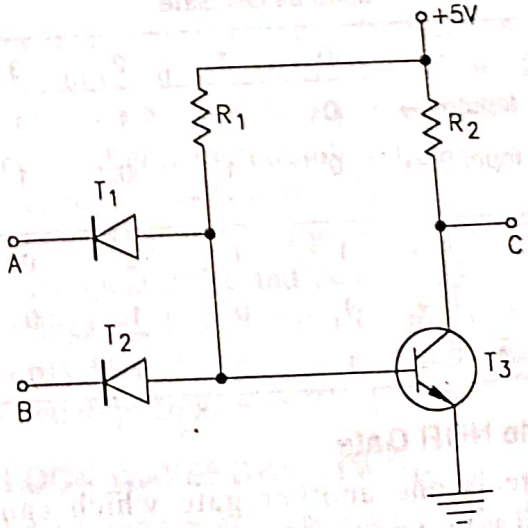



Fig. 3.18 NAND gate circuit

a NAND gate circuit. If +5 V is taken as to represent 1 and 0 V as 0 we note from this circuit that output at C will be LOW (0) only when both input A and B are at HIGH (1).

This gate can also perform the function of an OR gate as in a NAND gate the output will be 1 if A is 0 or B is 0. The symbol for this function is illustrated in Fig. 3.19(*a*). Here the input A and B are inverted

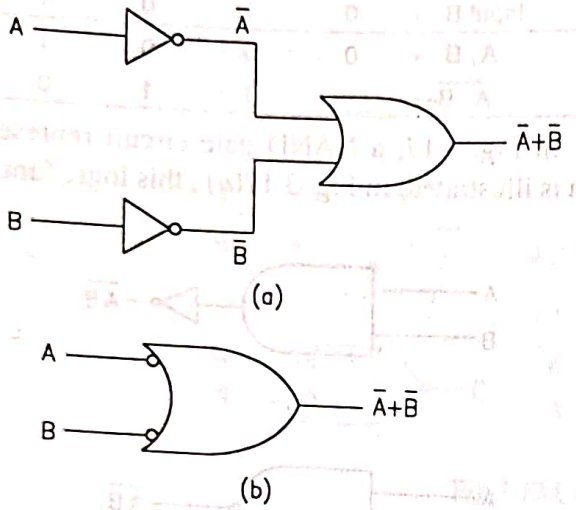



Fig. 3.19

giving $\bar{A}$ and $\bar{B}$ respectively and on ORing we get $\bar{A}+\bar{B}$ as its output. The simplified form of this representation is shown in Fig. 3.19(*b*). The circles are considered as inverters in series with the input. The truth table as shown in Table 3.19 is same as shown for NAND gate. This shows that a NAND gate can be used as an OR gate.

**Table 3.19 Truth Table for NAND Gate used as OR Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| $\bar{A}$→ | 1 | 1 | 0 | 0 |
| $\bar{B}$→ | 1 | 0 | 1 | 0 |
| $\bar{A}+\bar{B}$→ | 1 | 1 | 1 | 0 |

### 3.6.2 The NOR Gate

NOR gate is one another gate which can be considered as the universal hardware building block. This gate can be considered as equal to an OR gate followed by an inverter *i.e.* NOT followed by OR and is called NOR. In a NOR gate we will have output as low (0) unless all the inputs are 0 as in this case, the output is 1, *i.e. For a NOR gate if any of the input is 1, the output is 0 and if all the inputs are 0, only then the output is 1. The truth table for NOR gate is shown in Table 3.20.* For N input NOR gate $Y = \overline{X_1 + X_2 + X_3 + \ldots X_N}$ where $Y$ is the output and $X_1, X_2$ are N inputs.

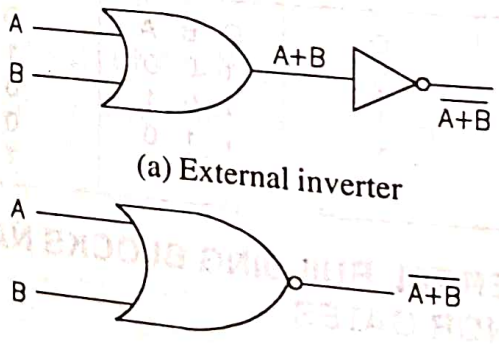
**Table 3.20 Truth Table for NOR Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| A + B→ | 0 | 1 | 1 | 1 |
| Output C = $\overline{A+B}$→ | 1 | 0 | 0 | 0 |

If the output is C for two inputs A and B for a NOR gate, then the Boolean expression for a NOR gate will be

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

The logic symbols which are used for NOR gate are shown in Figs. 3.20 and 3.21. The symbols shown in Figs. 3.20 (*a*) and (*b*) are same, but in practice, Fig. 3.20 (*b*) is generally used for representing NOR gate.



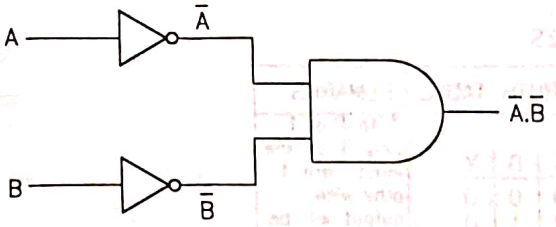

(a) External inverter



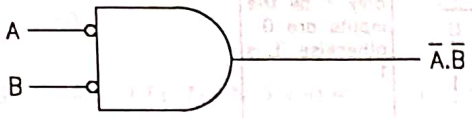

(b) Internal inverter

Fig. 3.20 NOR gate symbol used as OR function

Fig. 3.22 shows a transistor circuit for a NOR gate. It is clear from this circuit that C will be high (+5 V) or 1 only when both inputs A and B are low *i.e.* when both the transistors base are grounded the point C will be having +5 V. In other words, if A and B are 0, only then we will have output as 1. In all other cases the output will be 0. In NOR gate, we may have more




(a) as external inverter




(b) as internal inverter

Fig. 3.21 NOR gate symbol used as AND function

than one inputs but only one output. The care should be taken in evaluating $\overline{A+B+C+D}$ as it is not same as $\overline{A}+\overline{B}+\overline{C}+\overline{D}$. The inversion operation is to be performed only after OR operation is performed. A NOR gate can perform the function of OR gate as well as of AND gate. The NOR gate can be used as inverter by connecting the input leads together.

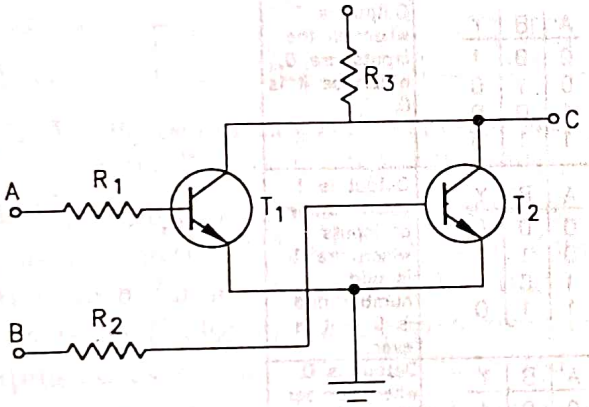



Fig. 3.22 NOR gate circuit

### 3.6.3 The Exclusive NOR Gate

This gate may be considered as NOT gate followed by exclusive OR gate. The output of an exclusive NOR gate is 1 when both the inputs (for a two input NOR gate) are 0 or when both the inputs are 1 *i.e.* when both the inputs are same whether 0 or 1. Otherwise the output will be 0. *i.e.* when the inputs are not same, the output will be 0. In fact, the output of a NOR gate is the complement of the exclusive OR gate. Table 3.21 gives the truth table for a two input NOR gate. *In* XNOR *gate output is 1 only when it has odd number of inputs as the output is 0 if the number of signals to input having 1 are even, 0 is taken as even.*

**Table 3.21 Truth Table for a NOR Gate with two Inputs**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| $A \oplus B$ → | 0 | 1 | 1 | 0 |
| Output $C = \overline{A} \oplus \overline{B}$ → | 1 | 0 | 0 | 1 |

The symbol used for exclusive-NOR gate is shown in Fig. 3.23. In Boolean algebra, it is represented as $A \cdot B$ and also sometimes as $\overline{A \oplus B}$.

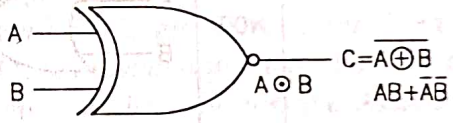



Fig. 3.23 Symbol or exclusive-NOR gate

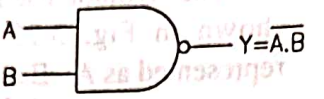
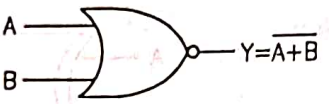
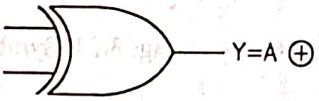
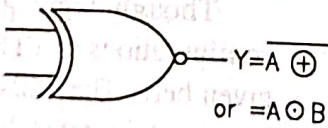
A summary of logic gates alongwith logic diagram on truth table are given in Table 3.22.

Though logic gates are available in various configurations in TTL and CMOS families a few are given here. The 7404 TTL and the 4049 CMOS are hex (six) inverter ICs. The 7400 TTL and 4011 CMOS are quad (four) two input NAND ICs , the 7402 TTL and the 4001 CMOS are quad two input ICs. For other NAND and NORs in three, four and eight input configuration consult a TTL or CMOS data manual for the availability and pin configuration of these ICs. The pin configuration for the hex inverter, the quad NOR and quad NAND ICs are shown in Figs. 3.24, 3.25 and 3.26 (see page 3.18). High speed CMOS 74 HCO4, and 74HCO2 have the same pin configuration as in TTL ICs.

## 3.7 CLOCK WAVEFORM TIMING

Special clock and timing circuits are used to produce

TABLE 3.22
SUMMERY OF LOGIC GATES

| GATE | LOGIC DIAGRAM | TRUTH TABLE | REMARKS |
|---|---|---|---|
| AND |   | A B Y<br>0 0 0<br>0 1 0<br>1 0 0<br>1 1 1 | Output is 1 only if all the inputs are 1 otherwise output will be 0 |
| OR |   | A B Y<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 1 | Output is 0 only if all the inputs are 0 otherwise it is 1. |
| NOT |   | A Y<br>0 1<br>1 0 | Output is 0 for input 1 and it is 1 for 0 input ie. it is complemented. |
| NAND |   | A B Y<br>0 0 1<br>0 1 1<br>1 0 1<br>1 1 0 | Output is 0 only when all the inputs are 1 otherwise it is 1. |
| NOR |   | A B Y<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 0 | Output is 1 when all the inputs are 0, otherwise it is 0. |
| XOR |   | A B Y<br>0 0 0<br>0 1 1<br>1 0 1<br>1 1 0 | Output is 1 when number of inputs which are 1 is odd number and is 0 if it is even. |
| XNOR |   | A B Y<br>0 0 1<br>0 1 0<br>1 0 0<br>1 1 1 | Output is 0 when number of inputs which are 1 is even otherwise it is 1. |

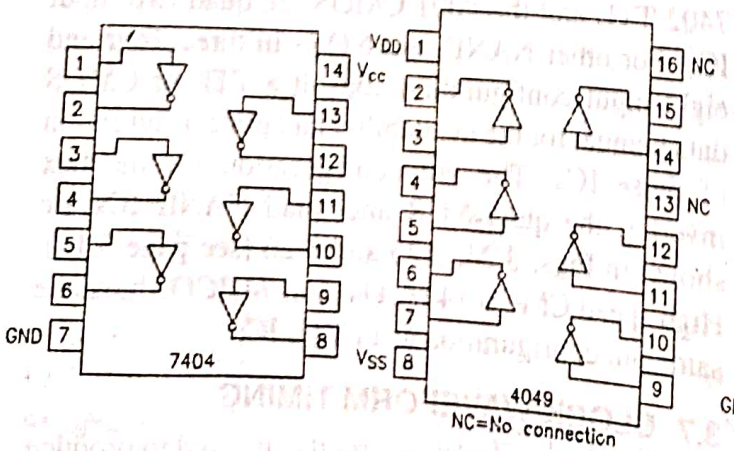



Fig. 3.24 7404 TTL and 4049 CMOS inverter pin configurations.

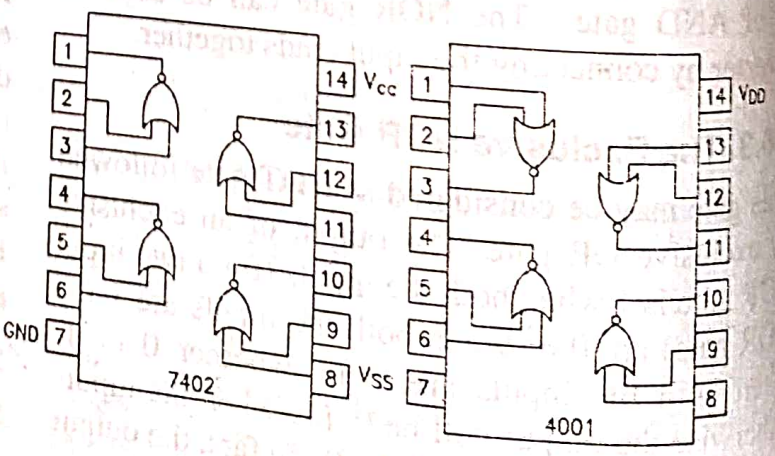



Fig. 3.25 7402 TTL NOR and 4001 CMOS NOR

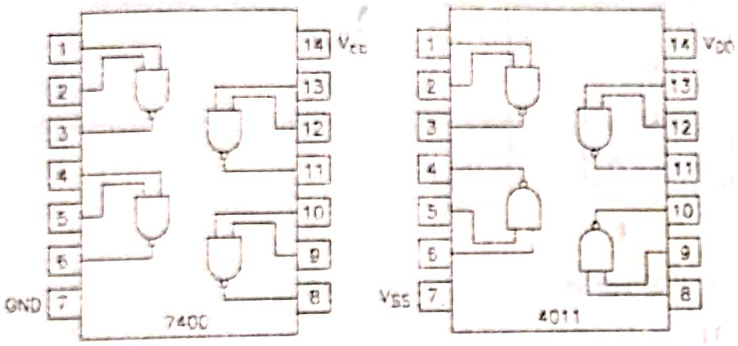



Fig. 3.26 7400 TTL NAND and 4011 CMOS NAND

clock waveforms to trigger the digital signals at precise intervals as most digital signals require precise timing. Fig. 3.27 shows a typical periodic clock waveform as it would appear on an oscilloscope displaying voltage versus time. The periodic means that the waveform is repetitive, at a specific time interval, with each successive pulse identical to the previous one.

Fig. 3.27 Periodic clock waveform as seen on an oscilloscope displaying voltages versus time.

In Fig. 3.28 eight clock pulses are shown and these are labelled 0, 1, 2, 3, 4, 5, 6 and 7. The duration of time from the falling edge of one pulse to the falling edge of the next pulse (or rising edge to rising edge) is expressed as $t_p$ and called as the period. The reciprocal of the clock period is called frequency.

being slow because each bit transmitted exists for one clock period. However this technique is used by computers to transmit data over telephone lines or from one system to another.

For example, consider Fig. 3.29 to illustrate the serial representation of the binary number 01101100. The serial representation ($S_0$) is shown with respect to a clock waveform ($C_p$), and its LSB is drawn first. Each bit from the original binary number occupies a separate clock period with the change from one bit to the next occurring at each falling edge of ($C_p$) which is drawn just for reference (Fig. 3.30).

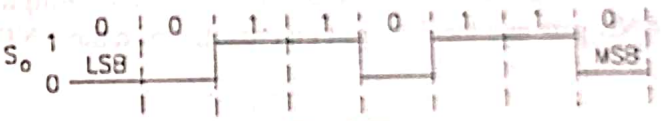



Fig. 3.29

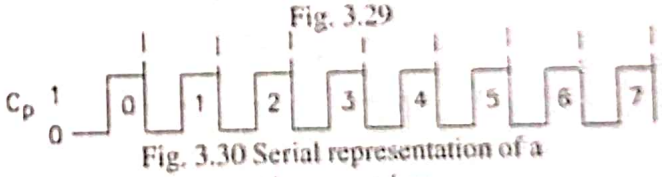



Fig. 3.30 Serial representation of a binary number.

**Example 16.** *Sketch the serial representations (least significant digit first) of the hexa decimal number 4A. Also, what is the state (1 or 0) of the serial line 1.2 ms into the transmission. Assume a clock frequency of 4 kHz.*

**Solution.** $4A_{16} = 01001010_2$ As frequency is 4 kHz so time period $t_p = 1/f = 1/4000 = 0.25$ ms. Therefore, the increment of time at each falling edge increases by 0.25 ms. As each period is 0.25 ms, 1.4 ms will occur within the period of the number 4, which on $S_0$ line is a 0 logic state. (Fig. 3.31).

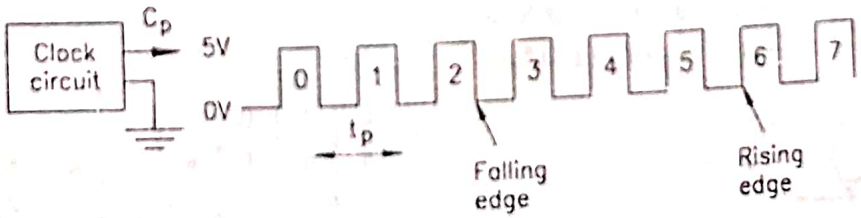



Fig. 3.28 Eight Clock pulses.

# 3

# BOOLEAN ALGEBRA AND LOGIC GATES

## 3.1 INTRODUCTION

In our day to day life, quite often, we are faced with two valued questions such as - is an action wrong or right, a conclusion is false or true, should we do an act or not? Much of our energy is used for finding the answers of such questions using our thinking and logic. Aristotle tried to evolve precise methods for getting the truth, based on a set of assumptions for such two valued or binary problems. The logic has a connection with mathematics was first proposed by De Morgan. But Boole tried to perform a mathematical analysis of logic and on the basis of his investigation of the laws of thought, Boole constructed a logical algebra. This algebra is commonly known as *Boolean algebra*. This mathematical system of logic (Boolean algebra) expressing truth functions as symbols and then manipulating these symbols to arrive at a conclusion is different from the algebra we study in school and college but is an algebra of logic.

Digital systems are basically logic machines and are capable of performing a very large number of simple logical operations in a very short time. As digital systems work in binary *i.e.* involving only two possible states, the Boolean algebra has been used extensively in digital electronics. The use of Boolean algebra simplifies the procedure of solving logical problems and also simplify any circuit to minimum components necessary to perform the function for which the circuit is designed.

## 3.2 POSTULATES OF BOOLEAN ALGEBRA

As stated above in Boolean algebra, we deal with only logic variables and these variables can have only two values or there are two constants within the boolean system : 0 and 1. *i.e.* every number is either a 0 or 1. This logical variable can be explained with the help of the following example. This variable can be used to denote the sex of the employees in an organisation. If in the personal records of the employ the sex is denoted by X, the logic variable, X = 1 can be used to denote that a person is a female while X = 0 can be used to denote that a person is a male. The basic postulate of the Boolean algebra is that any logic variable has only two states. If X is the logic variable, then

if $X = 1$ then $X \neq 0$

if $X = 0$ then $X \neq 1$

Here, X = 1 or X = 0 has no numerical significance as these denote logic states only. For example, an electric lamp may be off (may be denoted by 0) or on (denoted by 1). Sometimes these values are also expressed as true or false. Normally, Boolean algebra uses the binary digits 0 and 1 to define logical decisions.

In Boolean algebra, three Boolean operators **OR, AND,** and exclusive – **OR (XOR)** combine two binary digits to produce a single digit result. The fourth Boolean operator **NOT**, complements a binary digit. These logic operations, (**OR, AND, NOT** and **XOR**) which can be performed with two logic variables are discussed here.

### 3.2.1 The AND operator

This operator AND is used every day in our life. For example, to put a car into motion, we must start the engine AND engage the transmission. We can understand this with the following example as shown in Fig. 3.1. The lamp is connected to the battery with switch 1 and switch 2 in series.

From the circuit, it is obvious that the lamp will glow only if both the switches are put on. Let us consider three logic variables X, Y and Z representing the state of switch 1, switch 2 and lamp respectively *i.e.*

X = 0 when switch-1 is off
X = 1 when switch-1 is on
Y = 0 when switch-2 is off
Y = 1 when switch-2 is on
Z = 0 when lamp is off (not glowing)
Z = 1 when lamp is on (glowing)
Now we know that lamp will glow (Z = 1) only

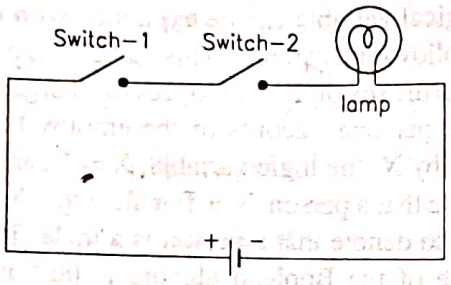



Fig. 3.1 Electrical circuit analog of AND.

when switch-1 is on (X = 1) and switch-2 is on (Y= 1) and lamp will be off (Z = 0) if switch-1 is on (X = 1) and switch-2 is off (Y = 0) also when switch-1 is off (X = 0) and switch-2 is on (Y = 1) and also when switch-1 is off (X = 0) and switch-2 is off (Y = 0).

AND operator is defined for two integers X and Y as follows. If X AND Y are both 1, the result is 1, otherwise, the result is zero. *The dot* (·), ∧ *symbol or no operator symbol at all are used to represent the AND operator.* In the above example,

$$Z = X \cdot Y$$

The equation represents the relation of the state of lamp (Z) with the on/off state of switch-1 (X) and switch-2 (Y) and represents a logic expression or an equation analogous to algebraic equation. From The above equation, if X is true (= 1) AND Y is true (= 1), then Z will be true (= 1). Under any other situation, Z will be false (= 0). The AND operation is also called logical product. There are four possible combi--nations of X and Y as shown in Table 3.1. This table is known as truth table for AND operator.

**TABLE 3.1 Truth Table for AND operator**

| X | Y | Z (= X · Y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Or

| X | Y | Z (= X · Y) |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

## 3.2.2 The OR operator

This operator OR is frequently used in our day to day life. For example, "If I had one rupee, I could buy a cup of coffee". "If I forget to bring the keys of the office, I can not get into my office". Like this, we may have a number of situations where satisfying at least one of the conditions results in a definite result. To understand the OR operator, let us consider an electric circuit as shown in Fig. 3.2. where a bulb is connected in series with two switches in parallel.

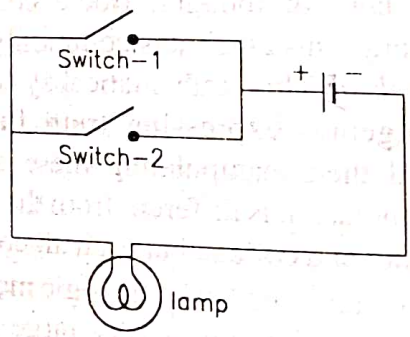



Fig. 3.2 Electrical circuit analogue of OR.

Let the state of lamp and switch-1 and switch-2 be defined in the same way as we did for AND operation. It is clear from Fig. 3.2. that the lamp will glow (Z = 1) only when switch-1 is on (X = 1) OR switch-2 is on (Y = 1). The lamp will also glow when both switches 1 and 2 are on and lamp will not glow (Z = 0) when switch-1 is off (X = 0) and switch-2 is off (Y = 0).

The OR operation is defined for two integers *i* and *j* as follows : If *i* OR *j* both equal 0, then the result is 0, otherwise, the result is one.

The symbol of plus sign '+' is used to represent OR. Sometimes, other symbols such as U and V are also used to represent OR. We shall use + sign only. This operation OR is also called as the *logical sum* or *simple sum* also sometimes it is called *inclusive OR.* The Boolean symbol for OR is same as used for representing arithmetic addition, the two operators should not be confused, though, they are very similar but they are not identical. In ordinary arithmetic 1+1 = 2 which is not the case with OR operation where 1 + 1 = 1 in Boolean algebra. There are four possible cases for combining X and Y as shown in Table 3.2. This table is known as truth table for OR operator.

**TABLE 3.2 Truth Table for OR operator**

| X | Y | Z(= X + Y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Or

| X | Y | Z(= X + Y) |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

### 3.2.3 The "Exclusive OR" operator

The above operator OR is also called inclusive OR as it includes both the cases when both the logical variables are true. The Exclusive OR differentiates between the logical variables which are identical and logical variables which are different. For example, we may say "I will go by a car OR a train". Here, the OR is exclusive because it is impossible for me to go by car and train simultaneously. The result is true (=1) when both the variables are different and is false (= 0) when both the variables are the same. The symbols used for XOR (exclusive OR) are $\oplus$ or $\veebar$. We shall use only $\oplus$. If X and Y are two logical variables and the result is Z on applying XOR operator *i.e.* $Z = X \oplus Y$. There are four possible cases. The truth table for XOR operator is given in Table 3.3.

**TABLE 3.3 Truth Table for XOR operator**

| X | Y | $Z(= X \oplus Y)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Or

| X | Y | $Z(= X \oplus Y)$ |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | False |

### 3.2.4 The NOT operator

This NOT operator changes the sense of an argument. Consider the electric circuit as shown in Fig. 3.3. in

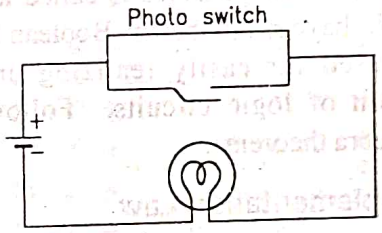



Fig. 3.3. Electrical circuit analog to NOT operator

which a lamp is connected in series with a photo-switch such that when the bulb is on, the switch will be off and when the bulb is off, the switch will be on. Let X = 1 denote that the lamp is on and Y = 1 denote the switch is on. When the lamp is on due to light, the photo switch is put off and when lamp is off, the photo switch is on. Hence, when X = 0, Y = 1 and when X = 1, Y = 0 *i.e.* X is the complement of Y. The complement is denoted by putting a bar over the logic variable *i.e.* $X = \overline{Y}$. the symbol for the NOT operator is called an *over score* or *bar*. Thus the symbolic representation of the above two situations will be

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

Table 3.4. shows the truth table for NOT operator

**TABLE 3.4 Truth Table for NOT operator**

| X | $Y = \overline{X}$ |
|---|---|
| 1 | 0 |
| 0 | 1 |

With the help of the operators defined above (OR, AND and NOT), we can derive the fundamental

postulates of Boolean algebra. The basic postulates of Boolean algebra can be summarised as follows by assigning logic values to the logic variable X.

1. $X = 1$ or $0$
2. $1 \cdot 1 = 1$
3. $1 \cdot 0 = 0 \cdot 1 = 0$
4. $0 \cdot 0 = 0$
5. $0 + 0 = 0$
6. $1 + 0 = 0 + 1 = 1$
7. $1 + 1 = 1$
8. $\overline{1} = 0$
9. $\overline{0} = 1$

## 3.3 THEOREMS OF BOOLEAN ALGEBRA

In Boolean algebra, we deal with algebraic relations between logic or Boolean variables. A fundamental rule relating Boolean variables is called a *Boolean theorem*. We have a number of Boolean theorems which are used for easily realizing and allow simplification of logic circuits. Following are Boolean algebra theorems:

### 3.3.1 Complementation Law

As explained while discussing NOT operator, the term complement means to invert, in Boolean algebra, this will mean to change 1's to 0's and 0's to 1's *i.e.*

$\overline{0} = 1$, $\overline{1} = 0$, if $A = 0$, then $\overline{A} = 1$, and if $A = 1$ then $\overline{A} = 0$

From above, it is clear for a logic variable X and $\overline{X}$ we have,

(Theorem **T-1** *a*) $X \cdot \overline{X} = 0$

(Theorem **T-1** *b*) $X + \overline{X} = 1$

The above results can be verified by using truth table for OR and AND operator. This theorem or law can be stated as follows. The multiplication of a logical variable with its complement ($X \cdot \overline{X}$) is zero and the sum of a logical variable with its complement is 1.

### 3.3.2 Commutative Law

This law allows the change in the position of the logical variables in ORing and ANDing *i.e.*

(Theorem **T-2** *a*) $X + Y = Y + X$

(Theorem **T-2** *b*) $X \cdot Y = Y \cdot X$

The above theorems can be proved with the help of the following truth tables. As there are two variables, there will be 4 combinations (0, 1, 2, 3).

**TABLE 3.5 Verification of X + Y = Y + X**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| X → | 0 | 0 | 1 | 1 |
| Y → | 0 | 1 | 0 | 1 |
| X + Y → | 0 | 1 | 1 | 1 |
| Y + X → | 0 | 1 | 1 | 1 |

**TABLE 3.6 Verification of X · Y = Y · X**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| X → | 0 | 0 | 1 | 1 |
| Y → | 0 | 1 | 0 | 1 |
| X . Y → | 0 | 0 | 0 | 1 |
| Y . X → | 0 | 0 | 0 | 1 |

From tables 3.5 and 3.6 we see L.H.S. of equation $X + Y (= Y + X)$ and $X \cdot Y (= Y \cdot X)$ are same as that R.H.S.

### 3.3.3 AND Laws

While discussing AND operator, we considered the ANDing operation of two logical variables. ANDing operation of a variable with itself or its complement is of special significance. There are following four laws or Theorems which specify the use of ANDing operation with itself:

(**T-3**) $X \cdot 0 = 0$

(**T-4**) $X \cdot 1 = X$

(**T-5**) $X \cdot X = X$ (Idempotent Law)

(**T-6**) $X \cdot \overline{X} = 0$

These theorems can be proved with the help of truth Table 3.1. for AND operator.

### 3.3.4 OR Laws

The ORing operation with the variable itself is also very useful. The following four cases are of special significance.

(**T-7**) $X + 0 = X$

(**T-8**) $X + 1 = 1$

(**T-9**) $X + X = X$ (Idempotent Law)

(**T-10**) $X + \overline{X} = 1$

The above theorems can be proved with the help of truth table for OR operator and taking bar of 0 as 1 and that of 1 as 0.

### 3.3.5 Associative Law

The associative law holds good for Boolean algebra as convention used for parenthesizing is same as that used in ordinary algebra. This law allows the grouping of variables and the two associative laws can be stated as follows:

(T-11) $X + (Y + Z) = (X + Y) + Z$ and

(T-12) $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$

The above theorems for associative law can be proved with the help of the following two truth tables. As we use 3 variables, there would be 8 combinations and for each combination it can be proved that associative law holds good.

Table 3.7 is truth table for associative law ORing X and Y gives the result shown by (X+Y). ORing Y and Z gives the result shown by (Y+ Z). Now ORing the (X+Y) with row Z, gives the result shown by row (X+Y) + Z. From this table, it is clear that the result in every case is the same as ORing row X with (Y+Z). This gives the verification of this law.

**TABLE 3.7 Verifying X + (Y + Z) = (X + Y) + Z**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| X → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Z → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| X + Y → | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Y + Z → | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| (X + Y) + Z → | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X + ( Y + Z) → | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

From truth table 3.8, we find, the result of $(X \cdot Y) \cdot Z$ in every case is similar to the results of $X \cdot (Y \cdot Z)$. This verifies the associative law for AND operator.

**TABLE 3.8 Verifying $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| X → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Z → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| X · Y → | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Y · Z → | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| (X · Y) · Z → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| X · ( Y · Z) → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

### 3.3.6 Distributive Law

The distributive law shows that we can expand expressions by multiplying terms as in ordinary algebra. We consider the following three distributive laws :

(T-13) $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$

(T-14) $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

(T-15) $X + (\overline{X} \cdot Y) = X + Y$

The perfect induction proof for theorem T-13 is given in table 3.9. as in every case, the left hand side of the identity is the same as right hand side *i.e.* $X \cdot (Y + Z)$ is same as $(X \cdot Y) + (X \cdot Z)$ for all combinations.

**TABLE 3.9 Verification of $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| X → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Z → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| (Y + Z) → | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| (X · Y) → | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| (X · Z) → | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| X · (Y + Z) → | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| (X · Y) + ( X · Z) → | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

The above Theorem(T–13) is same as we studied in ordinary algebra but the theorem (T–14) is different. This theorem can be proved in Boolean algebra as follows :

$X + (Y \cdot Z)$

$= X + Y \cdot Z$

$= X \cdot 1 + Y \cdot Z$ (As $X \cdot 1 = X$)

$= X \cdot (1 + Y) + Y \cdot Z$ (As $Y + 1 = 1$)

$= X + X \cdot Y + Y \cdot Z$ (As $X \cdot 1 = X$ and T–6)

$= X \cdot (1 + Z) + X \cdot Y + Y \cdot Z$ (As $Z + 1 = 1$)

$= X \cdot X + X \cdot Z + X \cdot Y + Y \cdot Z$ (As $X = X \cdot X$)

$= X \cdot (X + Z) + Y \cdot X + Y \cdot Z$ (As $X \cdot Y = Y \cdot X$ and T–6)

$= X \cdot (X + Z) + Y \cdot (X + Z)$ (T–6)

$= (X + Z) \cdot X + (X + Z) \cdot Y$ (As $X \cdot Y = Y \cdot X$)

$= (X + Z) \cdot (X + Y)$ (From T-6)

$= (X + Y) \cdot (X + Z)$ (As $X \cdot Y = Y \cdot X$)

$\therefore X + Y \cdot Z$

$= (X + Y) \cdot (X + Z)$

The above Theorem (T–14) can also be proved with the help of Truth Table 3.10.

**TABLE 3.10 Verification of $X + Y \cdot Z = (X + Y) \cdot (X + Z)$**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| X → | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y → | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Z → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| (X + Y) → | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| (Y · Z) → | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| (X + Z) → | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| X + (Y · Z) → | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| (X + Y) · (X + Z) → | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

From truth table 3.10, we find that, for each combination $X + (Y \cdot Z)$ is always equal to $(X + Y) \cdot (X + Z)$ which proves the Theorem(T–14). Theorem(T–15) is also different to that of ordinary algebra. This theorem can also be proved with the help of truth table or as follows:

$$\begin{aligned} X + \overline{X} \cdot Y &= X + \overline{X} \cdot Y \\ &= X \cdot 1 + \overline{X} \cdot Y \\ &= X \cdot (1 + Y) + \overline{X} \cdot Y \\ &\quad (\text{As } Y+1 = 1 \text{ and } X+Y = Y+X) \\ &= X \cdot 1 + X \cdot Y + \overline{X} \cdot Y \\ &= X + X \cdot Y + \overline{X} \cdot Y \qquad (\text{As } X \cdot 1 = X) \\ &= X + Y \cdot X + Y \cdot \overline{X} \\ &\qquad (\text{As } X \cdot Y = Y \cdot X) \\ &= X + Y \cdot (X + \overline{X}) \qquad (\text{From T - 6}) \\ &= X + Y \cdot 1 \qquad (\text{As } X + \overline{X} = 1) \\ &= X + Y \qquad (\text{As } Y \cdot 1 = Y) \end{aligned}$$

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

### 3.3.7 Absorption Law

According to this law

**(T-16)** $X + X \cdot Y = X$

**(T-17)** $X \cdot (X + Y) = X$

These two theorems also known as *Redundance law* can be proved by perfect induction as well as by adopting algebraic methods as follows:

$$\begin{aligned} X + X \cdot Y &= X + X \cdot Y \\ &= X \cdot 1 + X \cdot Y \qquad (\text{Using T-4}) \\ &= X \cdot (1 + Y) \qquad (\text{Using T-13}) \\ &= X \cdot 1 \qquad (\text{Using T-8}) \\ &= X \qquad (\text{Using T-4}) \end{aligned}$$

Hence,

$X + X \cdot Y = X$

Similarly, we can show $X \cdot (X + Y) = X$

$$\begin{aligned} X \cdot (X + Y) &= X \cdot (X + Y) \\ &= X \cdot X + X \cdot Y \qquad (\text{Using T-13}) \\ &= X + X \cdot Y \qquad (\text{Using T-5}) \\ &= X \ (\text{Using T-16 as proved above}) \end{aligned}$$

Hence,

$X \cdot (X + Y) = X$

We have the following special cases of Theorem(T–17)

**(T-18)** $X \cdot (\overline{X} + Y) = X \cdot Y$

The above theorem can be proved as follows:

$$\begin{aligned} X \cdot (\overline{X} + Y) &= X \cdot (\overline{X} + Y) \\ &= X \cdot \overline{X} + X \cdot Y \\ &= 0 + X \cdot Y \qquad (\text{Using T-1}) \\ &= X \cdot Y \end{aligned}$$

### 3.3.8 Consensus Law or Included Factor/ Term Law

This theorem can be stated as follows

**(T-19)** $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$ and

**(T-20)** $(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \cdot (\overline{X} + Z)$

These theorems can be proved as follows.

$$\begin{aligned} &X \cdot Y + \overline{X} \cdot Z + Y \cdot Z \\ &= X \cdot Y + \overline{X} \cdot Z + Y \cdot Z \cdot (X + \overline{X}) \\ &= X \cdot Y + X \cdot Y \cdot Z + \overline{X} \cdot Z + \overline{X} \cdot Y \cdot Z \\ &= X \cdot Y(1 + Z) + \overline{X} \cdot Z \cdot (1 + Y) \\ &= X \cdot Y + \overline{X} Z \end{aligned}$$

Similarly

$$\begin{aligned} &(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) \\ &= (\overline{X} \cdot Y + X \cdot Z + Y \cdot Z) \cdot (Y + Z) \\ &\qquad (\text{As } X \cdot \overline{X} = 0) \\ &= \overline{X} \cdot Y + X \cdot Y \cdot Z + Y \cdot Z + \overline{X} \cdot Y \cdot Z + X \cdot Z \\ &\qquad (\text{As } Y \cdot Y = Y) \\ &= \overline{X} \cdot Y + \overline{X} \cdot Y \cdot Z + Y \cdot Z + X \cdot Y \cdot Z + X \cdot Z \\ &= \overline{X} \cdot Y + Y \cdot Z + X \cdot Z \\ &= (X + Y) \cdot (\overline{X} + Z) \end{aligned}$$

### 3.3.9 Transposition Law

These theorems are very useful in simplifications and can be represented as follows:

**(T-21)** $\overline{X} \cdot Y + \overline{X} \cdot Z = (X + Z) \cdot (\overline{X} + Y)$

**(T-22)** $(X + Y) \cdot (\overline{X} + Z) = (X \cdot Z) + \overline{X} \cdot Y)$

These can be proved as we have proved above

theorems. Following are two forms of transposition theorem which we quite often use in practice and represents XOR and NOR operation.

$$X \cdot \overline{Y} + \overline{X} \cdot Y = (X + Y) \cdot (\overline{X} + \overline{Y}) \quad \text{and}$$

$$X \cdot Y + \overline{X} \cdot \overline{Y} = (X + \overline{Y}) \cdot (\overline{X} + Y)$$

### 3.3.10 De Morgan's Theorem

These theorems are very useful and most power- ful identities used in Boolean algebra. It states, in general, that the complement of any expression can be obtained by replacing each variable and element with its complement and changing OR operators with AND operators and AND operators to OR operators. These theorems can be expressed as follows:

(T-23) $\overline{X + Y} = \overline{X} \cdot \overline{Y}$

(T-24) $\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$

The above theorem can be proved by perfect induction. As we are using 2 variables there will be four combinations.

**TABLE 3.11 Verification or Truth Table for De Morgans Theorem**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $X \rightarrow$ | 0 | 0 | 1 | 1 |
| $Y \rightarrow$ | 0 | 1 | 0 | 1 |
| $\overline{X} \rightarrow$ | 1 | 1 | 0 | 0 |
| $\overline{Y} \rightarrow$ | 1 | 0 | 1 | 0 |
| $\overline{X} \cdot \overline{Y} \rightarrow$ | 1 | 0 | 0 | 0 |
| $X + Y \rightarrow$ | 0 | 1 | 1 | 1 |
| $\overline{X + Y} \rightarrow$ | 1 | 0 | 0 | 0 |

For performing the transformation, the following three steps should be used and this procedure is called *demorganisation*.

(i) *Complement the entire function.*

(ii) *Change all the ORs to ANDs and all ANDs to ORs.*

(iii) *Complement each of the individual variables.*

Some of the useful theorems or laws of Boolean Algebra are summarised in Table 3.12.

**TABLE 3.12 Summary of Boolean Algebra Laws**

| S. No. | Law |
|---|---|
| 1 | $\overline{0} = 1$ |
| 2 | $\overline{1} = 0$ |
| 3 | If $X = 0$, $\overline{X} = 1$ |
| 4 | If $X = 1$, $\overline{X} = 0$ |
| 5 | $\overline{\overline{X}} = X$ |
| 6 | $X \cdot 0 = 0$ |
| 7 | $X \cdot 1 = X$ |
| 8 | $X \cdot X = X$ |
| 9 | $X \cdot \overline{X} = 0$ |
| 10 | $X + 0 = X$ |
| 11 | $X + 1 = 1$ |
| 12 | $X + X = X$ |
| 13 | $X + \overline{X} = 1$ |
| 14 | $X + Y = Y + X$ |
| 15 | $X \cdot Y = Y \cdot X$ |
| 16 | $X + (Y + Z) = (X + Y) + Z$ |
| 17 | $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ |
| 18 | $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$ |
| 19 | $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$ |
| 20 | $X + (\overline{X} \cdot Y) = X + Y$ |
| 21 | $\overline{X + Y} = \overline{X} \cdot \overline{Y}$ |
| 22 | $\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$ |

### 3.3.11 Duality Theorem

Duality principle states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. Laws which obey duality principle are called duality laws. De-Morgan's theorem is a special case of duality theorem of digital functions which can be stated as follows. *For a function to be true, its dual should also be true.*

Dual of X is $\overline{X}$ i.e. of 0 is 1 and of 1 is 0, of (+) is (.) and of (.) is (+). As an example if

$X = (\overline{A} + B) \cdot (\overline{B} + \overline{C}) \cdot (C + \overline{D})$ is true

Then $\overline{X} = A \cdot \overline{B} + B \cdot C + \overline{C} \cdot D$ is also true.

## 3.4 REDUCING BOOLEAN EXPRESSIONS

The above discussed Boolean postulates and theorems are used for simplifying Boolean expressions. When a designer desires to achieve an electronic circuit making use of logic operations, he has to reduce every Boolean equation to the simplest

possible form so as to reduce the cost of the circuit to the minimum. For this reduction, the theorems stated above are used. This reduction is somewhat similar to that we do in ordinary algebra. As a general approach, we should use the following procedure for reducing a Boolean algebra functions:

(*i*) To remove the parenthesis multiply all variables necessary.

(*ii*) Collect the identical terms and using identity law, one of these can be dropped.

(*iii*) Search for the term having a variable and its negation. Omit such terms. For example, the terms such as $X \cdot \overline{X} \cdot Y (= 0 \cdot Y) = 0$ can be omitted.

(*iv*) Collect pairs of terms which are identical except for one variable. Now, if you find in one such term one variable is mixing, the larger term can be omitted. For example,

$$U \cdot X \cdot Y \cdot Z + U \cdot X \cdot Z = U \cdot X \cdot Z \cdot (Y + 1)$$
$$= U \cdot X \cdot Z \cdot 1$$
$$= U \cdot X \cdot Z$$

The above mentioned procedure can be used, in general, for most cases. Some times, we have to examine each expression carefully and for reducing we might use combinations or additions.

In order to reduce or solve Boolean expressions, it is necessary to fix the precedence of operators. Following precedence should be followed :

(1) Scan the expression from left to right

(2) First solve parenthesis *i.e.* evaluate expressions enclosed in parentheses.

(3) Now perform NOT (compliment) operations.

(4) Perform AND (·) operations after NOT operations.

(5) In the end, perform OR (+) operations.

**Example 1.** *Solve the following*

(*a*) $X \cdot Y \cdot Y \cdot Z \cdot Z \cdot Z$

(*b*) $X + X + Y + Y + Y$

(*c*) $X \cdot X \cdot Y + X \cdot Y \cdot Y + Y \cdot \overline{Y} \cdot X \cdot X$

(*d*) $X \cdot X \cdot Y \cdot \overline{Y} + \overline{X} \cdot Y \cdot Z \cdot Z$

**Solution.**

(*a*) $X \cdot Y \cdot Y \cdot Z \cdot Z \cdot Z$
$= X \cdot (Y \cdot Y) \cdot Z \cdot (Z \cdot Z)$
$= X \cdot Y \cdot Z \cdot Z$
$= X \cdot Y \cdot Z$

(*b*) $X + X + Y + Y + Y$
$= (X + X) + Y + (Y + Y)$
$= X + Y + Y$
$= X + (Y + Y)$
$= X + Y$

(*c*) $X \cdot X \cdot Y + X \cdot Y \cdot Y + Y \cdot \overline{Y} \cdot X \cdot X$
$= (X \cdot X) \cdot Y + X \cdot (Y \cdot Y) + (Y \cdot \overline{Y}) \cdot (X \cdot X)$
$= X \cdot Y + X \cdot Y + 0$
$= X \cdot Y$

(*d*) $X \cdot X \cdot \overline{Y} \cdot Y + \overline{X} \cdot Y \cdot Z \cdot Z$
$= X \cdot X \cdot Y \cdot Y + \overline{X} \cdot Y \cdot Z \cdot Z$
$= X \cdot Y + \overline{X} \cdot Y \cdot Z$
$= Y \cdot (X + \overline{X} \cdot Z)$
$= Y(X + Z)$

**Example 2.** *Solve the following*

(*a*) $X \cdot [Y + Z \cdot \overline{(X \cdot Y + X \cdot Z)}]$

(*b*) $(X \cdot Y + Z)(X \cdot Y + U)$

(*c*) $X \cdot \overline{Y} + \overline{X} \cdot Y + X \cdot Y + \overline{X \cdot Y}$

**Solution.**

(*a*) $X \cdot [Y + Z \cdot \overline{(X \cdot Y + X \cdot Z)}]$
$= X \cdot [Y + Z \cdot (\overline{X \cdot Y} \cdot \overline{X \cdot Z})]$
$= X \cdot Y + X \cdot Z \cdot \overline{X \cdot Y} \cdot \overline{(X \cdot Z)}$
$= X \cdot Y + X \cdot Z \cdot \overline{X} \cdot \overline{Z} \cdot \overline{X \cdot Y}$
$= X \cdot Y + X \cdot Z \cdot \overline{X} \cdot \overline{Z} \cdot \overline{X} \cdot \overline{Y}$
$= X \cdot Y + 0 \cdot \overline{X \cdot Y}$
$= X \cdot Y$

(*b*) For solving the expression

$$(X \cdot Y + Z)(X \cdot Y + U)$$

We use the theorem (T–14)

$$(A + B) \cdot (A + C) = A + B \cdot C.$$

Hence we have

$$(X \cdot Y + Z)(X \cdot Y + U)$$
$$= X \cdot Y + Z \cdot U$$

(*c*) $X \cdot \overline{Y} + \overline{X} \cdot Y + X \cdot Y + \overline{X} \cdot \overline{Y}$
$= X \cdot \overline{Y} + X \cdot Y + \overline{X} \cdot Y + \overline{X} \cdot \overline{Y}$
$= X \cdot (\overline{Y} + Y) + \overline{X} \cdot (Y + \overline{Y})$
$= X + \overline{X}$
$= 1$

**Example 3.** *Evaluate* $X \cdot \overline{Y} \cdot \overline{Z} + X \cdot Y$ *under the following conditions.*

(*a*) $X = 1, Y = 0, Z = 1$

(*b*) $X = 0$ $Y = 0$ and $Z = 0$

(*c*) $X = 0, Y = 1, Z = 1$

## 3.6 UNIVERSAL BUILDING BLOCKS NAND AND NOR GATES

### 3.6.1 The NAND Gate

This gate is an electronic circuit that can perform all the three logical functions, *i.e.* AND, OR and Inverter. This gate may be considered as equivalent to an AND gate followed by an inverter (*i.e.* AND NOT). However, in Boolean algebra, the convention is to represent such a gate as NOT AND *i.e.* to state NOT before the quantity to be inverted. This gate is usually called NAND.

*In NAND gate, we will have an output of logical 0 only when all the inputs are logical 1 otherwise the*

*output is logical 1.* The output is denoted by $\overline{A \cdot B}$ when the inputs are A and B. The truth table for the NAND gate is shown in Table 3.18. For N input NAND gate $Y = \overline{X_1 \cdot X_2 \cdot X_3 \ldots X_N}$ where Y is the output and $X_1, X_2, X_3 \ldots X_N$ are N inputs.

**Table 3.18 Truth Table for Two Input NAND Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| A · B → | 0 | 0 | 0 | 1 |
| $\overline{A \cdot B}$ → | 1 | 1 | 1 | 0 |

In Fig. 3.17, a NAND gate circuit representation is illustrated. In Fig. 3.17(*a*) , this logic function,

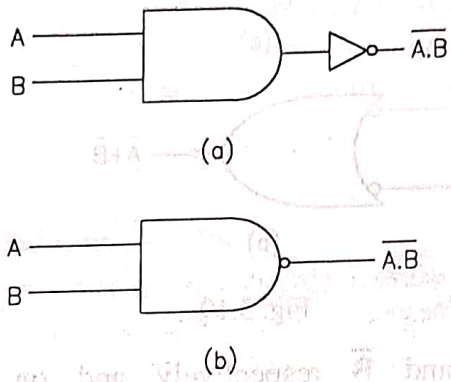



Fig. 3.17 NAND gate representation

is illustrated as an AND gate with an inverted output. The input to this gate A and B are first ANDed yielding A · B and then inverted giving $\overline{A \cdot B}$. The symbol shown in Fig. 3.17 (*b*) is reduced from an AND gate followed by an inverter. Fig. 3.18 shows

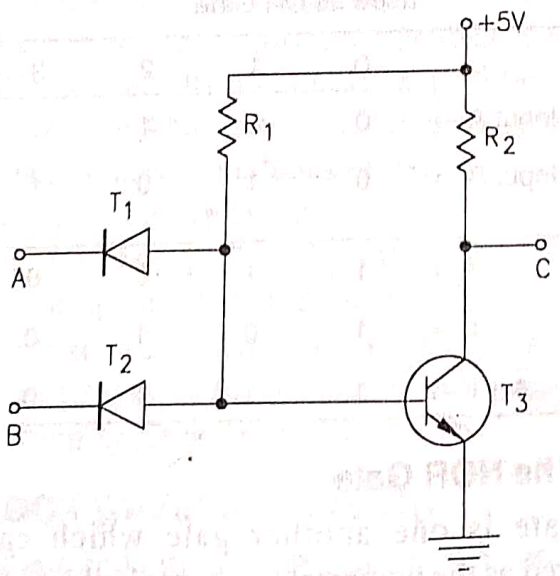



Fig. 3.18 NAND gate circuit

a NAND gate circuit. If +5 V is taken as to represent 1 and 0 V as 0 we note from this circuit that output at C will be LOW (0) only when both input A and B are at HIGH (1).

This gate can also perform the function of an OR gate as in a NAND gate the output will be 1 if A is 0 or B is 0. The symbol for this function is illustrated in Fig. 3.19(*a*). Here the input A and B are inverted

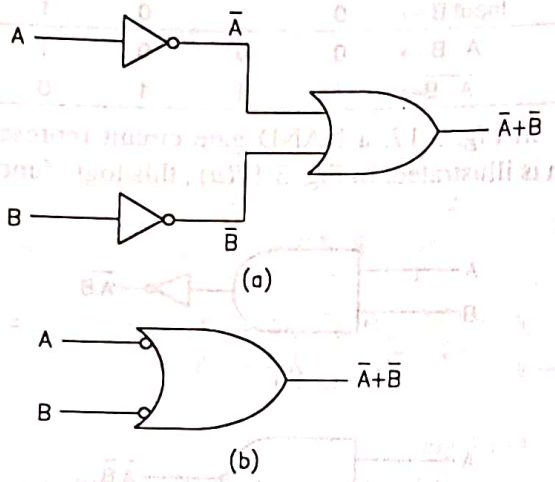



Fig. 3.19

giving $\bar{A}$ and $\bar{B}$ respectively and on ORing we get $\bar{A}+\bar{B}$ as its output. The simplified form of this representation is shown in Fig. 3.19(*b*). The circles are considered as inverters in series with the input. The truth table as shown in Table 3.19 is same as shown for NAND gate. This shows that a NAND gate can be used as an OR gate.

**Table 3.19 Truth Table for NAND Gate used as OR Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| $\bar{A}$→ | 1 | 1 | 0 | 0 |
| $\bar{B}$→ | 1 | 0 | 1 | 0 |
| $\bar{A}+\bar{B}$→ | 1 | 1 | 1 | 0 |

### 3.6.2 The NOR Gate

NOR gate is one another gate which can be considered as the universal hardware building block. This gate can be considered as equal to an OR gate followed by an inverter *i.e.* NOT followed by OR and is called NOR. In a NOR gate we will have output as low (0) unless all the inputs are 0 as in this case, the output is 1, *i.e. For a NOR gate if any of the input is 1, the output is 0 and if all the inputs are 0, only then the output is 1. The truth table for NOR gate is shown in Table 3.20. For* N input NOR gate $Y = \overline{X_1 + X_2 + X_3 + \ldots X_N}$ where $Y$ is the output and $X_1, X_2$ are N inputs.

**Table 3.20 Truth Table for NOR Gate**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Input A → | 0 | 0 | 1 | 1 |
| Input B → | 0 | 1 | 0 | 1 |
| A + B→ | 0 | 1 | 1 | 1 |
| Output C = $\overline{A+B}$→ | 1 | 0 | 0 | 0 |

If the output is C for two inputs A and B for a NOR gate, then the Boolean expression for a NOR gate will be

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

The logic symbols which are used for NOR gate are shown in Figs. 3.20 and 3.21. The symbols shown in Figs. 3.20 (*a*) and (*b*) are same, but in practice, Fig. 3.20 (*b*) is generally used for representing NOR gate.

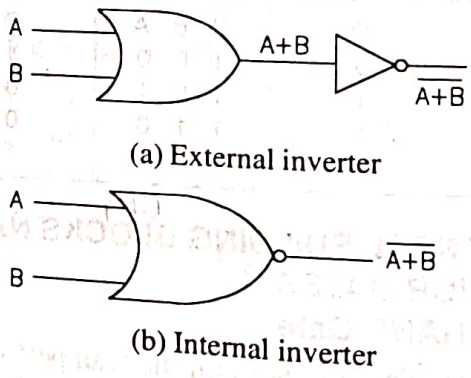
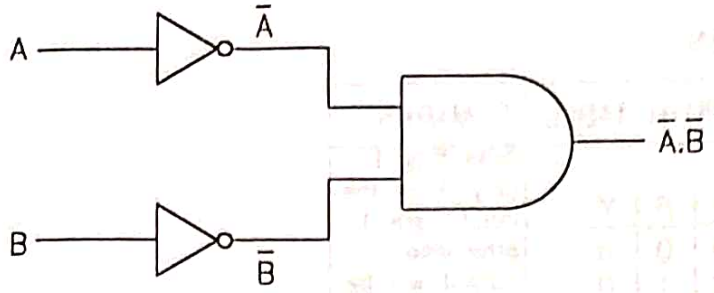
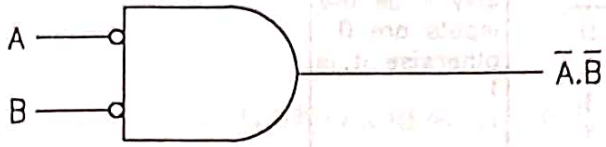



Fig. 3.20 NOR gate symbol used as OR function

Fig. 3.22 shows a transistor circuit for a NOR gate. It is clear from this circuit that C will be high (+5 V) or 1 only when both inputs A and B are low *i.e.* when both the transistors base are grounded the point C will be having +5 V. In other words, if A and B are 0, only then we will have output as 1. In all other cases the output will be 0. In NOR gate, we may have more




(a) as external inverter




(b) as internal inverter

Fig. 3.21 NOR gate symbol used as AND function

than one inputs but only one output. The care should be taken in evaluating $\overline{A + B + C + D}$ as it is not same as $\bar{A} + \bar{B} + \bar{C} + \bar{D}$ . The inversion operation is to be

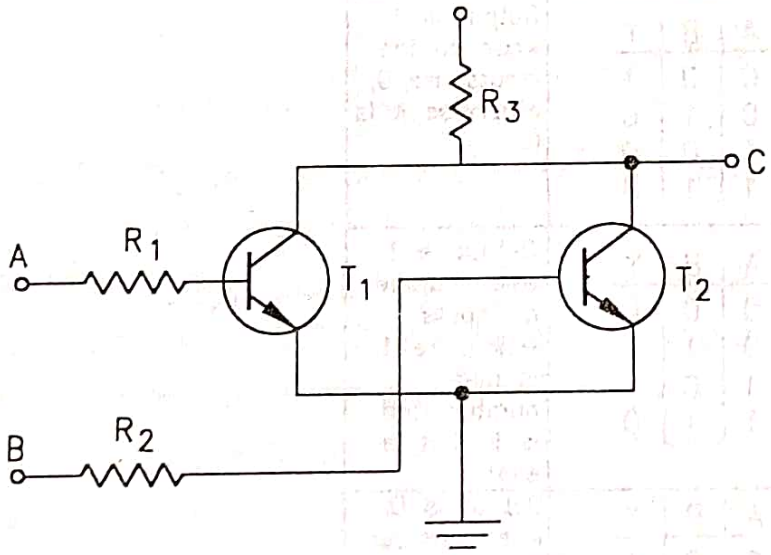



Fig. 3.22 NOR gate circuit

performed only after OR operation is performed. A NOR gate can perform the function of OR gate as well as of AND gate. The NOR gate can be used as inverter by connecting the input leads together.

## 6.2 A HALF ADDER

*A half adder* is a logic circuit that adds two binary bits. The input to the circuit are two bits say A and B and output of the circuit are carry and sum, denoted by say C and S. The sum (or S) is a 1 when A and B are different and carry (or C) is a 1 when A and B are 1's. The truth table for a half adder can be constructed using the addition table for binary numbers discussed

earlier and is as follows (Table 6.2).

**Table 6.2 Truth Table for Half Adder**

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

The Boolean expression for the half adder having above truth table can be written using Karnaugh map for S and C as

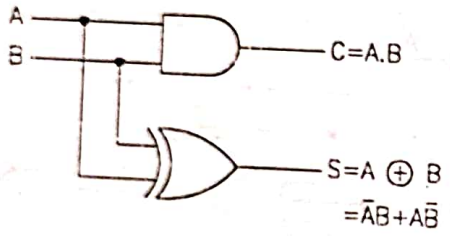
$$S = \overline{A} \cdot B + A \cdot \overline{B}$$

$$C = A \cdot B$$

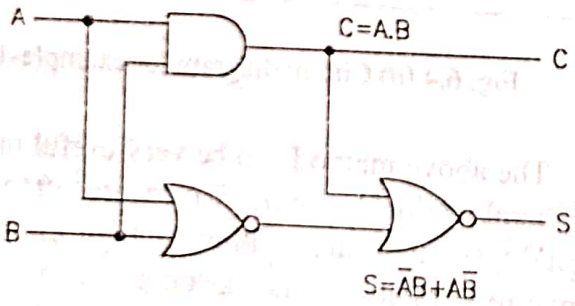
The Boolean expression for sum can be written as

$$S = A \oplus B$$

The realisation of half adder with XOR gate is shown in Fig. 6.5(a). The same function could be realized by using NAND gate also [Fig. 6.5(b)].




(a)




(b)

Fig. 6.5. Circuit diagram for Half-Adder.

## 6.3 THE FULL ADDER

When we have to add more than two binary digits we cannot use a number of half adders as a half adder has no input which can handle carries from other digits. Hence we need a circuit which takes the bits of the augend, addend and carry, adds them, and represents the sum and carry, and such a circuit is called *full adder*. A full adder circuit is capable of adding the contents of two bits of binary numbers, has the provision for handling carries as well as addend and augend bits. Therefore, a full adder has three inputs for each stage of multidigit except for the least significant bits. The two inputs correspond to the bits to be added (A and B) and one for any carry $C_0$ that have been propagated or generated from the previous stage. The block diagram for a full adder is shown in Fig. 6.6. The output will have two digits one Sum (S) and other carry (C) which is produced and added to the next stage.

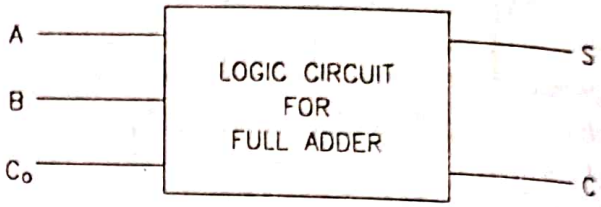



Fig. 6.6 Block diagram for Full-Adder.

**Table 6.3 Truth Table for Full Adder**

| A | B | $C_0$ | S | C |
|---|---|---|---|---|
| 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 |

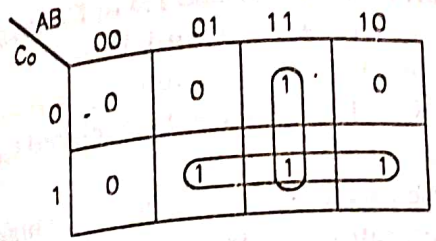
The Karnaugh map for function Sum (S) based on truth Table 6.3 is shown in Fig. 6.7 (*a*) and for carry (C) is shown in Fig. 6.7 (*b*). From these Karnaugh maps for S we have the minimal expression for S as

$$S = \overline{A}\,\overline{B}\,C_0 + \overline{A}\,B\,\overline{C}_0 + A\,\overline{B}\,\overline{C}_0 + A\,B\,C_0$$

| $C_0$ \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

(*a*) Karnaugh map for SUM(S).

Fig. 6.7. Karnaugh map for Full Adder.




(*b*) Karnaugh map for carry (C).

Fig. 6.7. Karnaugh map for Full Adder.

From Karnaugh map of Fig. 6.7 (*b*) we have the minimal expression for C as

$$C = AB + BC_0 + AC_0$$

The expression for S can be written in XOR operation as follows:

$$S = A \oplus B \oplus C_0$$

From the Karnaugh map of Fig. 6.7 and the corresponding reduced Boolean expression we observe that these functions (S and C) can be realised as shown in Fig. 6.8. A full adder may be constructed by using two half adders as shown in Fig. 6.9. As

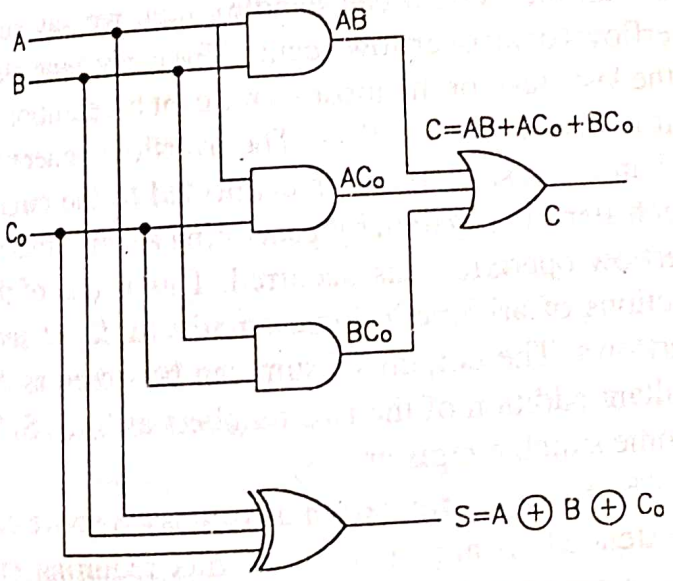



Fig. 6.8 Logic diagram for a Full Adder.

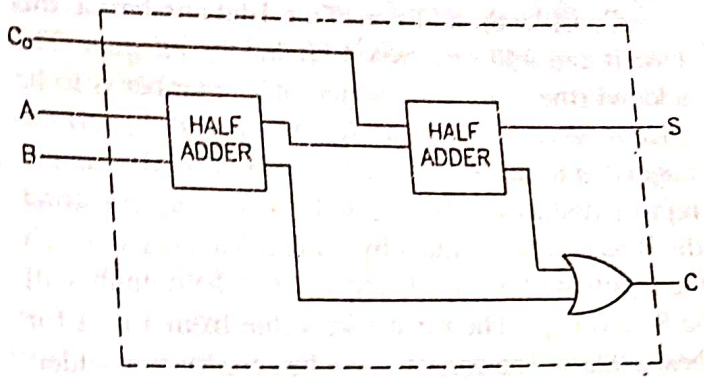



Fig. 6.9 Full Adder made of two Half Adders.

normally constructing a full adder from two half adder is not very economic so we construct full adders directly from input and output relations.

## 6.4 BINARY PARALLEL ADDER

By making use of a number of full adders we can build binary adders of any length. For example, for adding 16-bit number we may use 15 full adders and one half adder. A binary adder of any length can be represented by a block diagram as shown in Fig. 6.10.

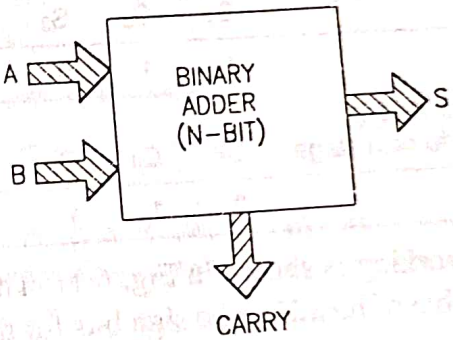



Fig. 6.10 Symbol for binary adder.

The solid arrows are standard way of representing a moving word. This figure shows that two binary numbers or words A and B are added to get a sum S and a final carry. A circuit for a parallel binary adder is shown in Fig. 6.11.

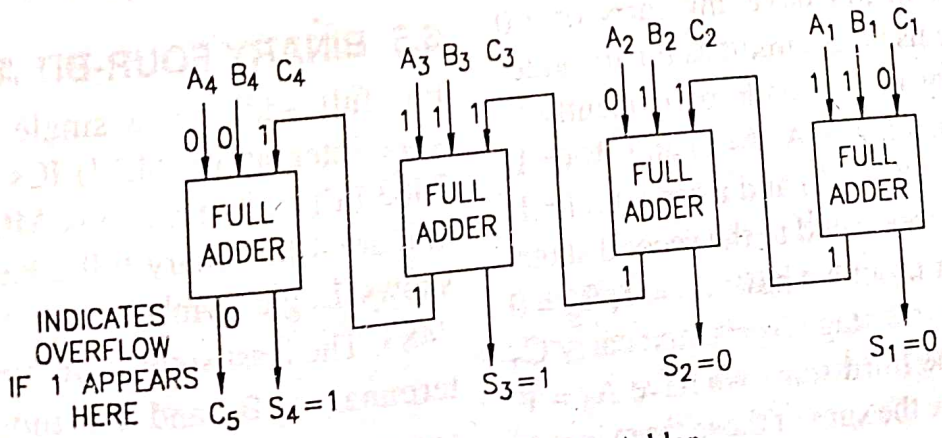



Fig. 6.11 Parallel Binary Adder

Consider the addition of the 4-bit numbers as this circuit can add any two 4-bit binary integers. The addend (the number to which other number is to be added) bits are named as $A_1$, $A_2$, $A_3$ and $A_4$ and the augend (the number which is to be added) bits are represented as $B_1$, $B_2$, $B_3$ and $B_4$. We can construct the Binary parallel adder by using full adders to which the inputs will be $A_i$, $B_i$ and $C_i$ and the outputs will be $S_i$ and $C_{i+1}$. The $i$ can take value from 1 to 4 for these adders. The carry $C_1$ will be zero for first adder. The carry $C_2$ from 1st stage is inputted to the second stage and so on.

The addition can be carried out as follows for two 4-bit binary integers say 0111 and 0101 :

| Addend bits | $A_4$ | $A_3$ | $A_2$ | $A_1$ |
|---|---|---|---|---|
| | 0 | 1 | 1 | 1 |
| Augend bits | $B_4$ | $B_3$ | $B_2$ | $B_1$ |
| | 0 | 1 | 0 | 1 |
| Carry bit from previous stage | $C_4$ | $C_3$ | $C_2$ | $C_1$ |
| | 1 | 1 | 1 | 0 |
| Sum | $S_4$ | $S_3$ | $S_2$ | $S_1$ |
| | 1 | 1 | 0 | 0 |
| Carry generated to be carried to next stage | $C_5$ | $C_4$ | $C_3$ | $C_2$ |
| | 0 | 1 | 1 | 1 |

The working is shown in Fig. 6.11. This circuit is not capable of handling the sign bits for the binary numbers to be added as it only adds the magnitudes of the numbers to be added. If we wish the sign bit also to be considered we shall have to use additional circuit and is discussed in next section. The sum obtained ($S_4\,S_3\,S_2\,S_1$) is 1100 as seen from the circuit and it can be explained as follows.

As the bit corresponding to $A_1$ and $B_1$ are least significant bits they will not have any carry or a 0 carry. In fact when we use this circuit as binary adder this line for first carry $C_1$ is made permanently 0. Hence the least significant bits $A_1$ (= 1) and $B_1$ (= 1) are added to give a sum $S_1$ (= 0) and a carry $C_2$ (= 1) is generated which is propagated to the second stage. In the second stage (for input we have $A_2 = 1$, $B_2 = 0$ and $C_2 = 1$. The sum at this stage $S_2 = 0$ and carry $C_3$ (= 1) is generated. In the third stage we have $A_3 = 1$, $B_3 = 1$ and $C_3 = 1$ . Hence the sum of these three inputs $S_3 = 1$ and carry term $C_3$ is also 1. For fourth stage the inputs are $A_4 = 0$, $B_4 = 0$ and $C_4 = 1$ (the carry generated at third stage). The sum at the fourth stage thus will be $S_4 = 1$ and the carry generated $C_5$ will be = 0.

Sometimes we may have at the last stage a carry of 1 which activates the overflow line. This situation arises when we add two binary integers whose sum cannot be expressed by the number of bits for which the binary parallel adder is designed. For example, if we add two numbers such as 1000 and 1000 the sum of these two numbers is 10000 which will require 5 bits to represent it. A four bit adder can display or handle numbers upto 15 or (four bit numbers only). Hence the addition of two numbers such as 1000 and 1000 will result in a carry of 1 at the 4th stage or the last carry will be 1 which will activate the overflow line indicating that the sum is exceeding the representable length of the system or it is a non representable number for this very system.

When we add two numbers whose sum is non-representable for the system (*i.e.* sum contains more bits than the system can handle), then we say sum overflows or an overflow occurs. The carry generated at the last stage or the most significant bit addition is 1, it indicates the overflow. The overflow generated as 1 in the last carry bit is connected to the circuit which alerts the system by generating a signal that an overflow operation has occurred. This is one of the functions of arithmetic logic circuits to detect such overflows. The outputs of sum can be stored as the resultant addition of the two numbers as $S_4\,S_3\,S_2\,S_1$ in some suitable register.

The basic configuration as discussed above can be extended to any number of bits requiring one additional full adder for each additional bit. A 32 bit binary adder will require 32 full adders.

## 6.5 BINARY FOUR-BIT ADDER ICs.

For full adder in a single package, as medium, scale-integration (MS I) ICs most popular ones are 7483 (TTL), 74HC283 (CMOS) and 4008 (CMOS) and are 4-bit binary full-adder, fast carry. Fig. 6.12 shows Logic symbols and functional diagram for 7483. The least significant binary input ($2^0$) is fed to terminal $A_1\,B_1$ and the most significant ($2^3$) are connected to $A_4\,B_4$ terminal (Leveling of the input

## 6.7 SUBTRACTOR

The subtraction of two binary numbers can be performed by taking the complement of the subtrahend (the number to be subtracted) and then adding to the minuend (the number from which the subtraction is to be done). By this method we can perform the subtraction in a same way as we did addition. The other method is to perform the subtraction in a manner analogous to hand calculations by using logic gates. In this case we generate difference

bits by subtracting the corresponding bits of subtrahend and minuend. If the minuend bit is smaller than the subtrahend bit a 1 is borrowed from the next higher significant bit or a borrow is generated to be borrowed from the next higher significant bit. This would be considered while subtracting next higher significant bits. In circuitry a borrow is generated if subtrahend bit is greater than the corresponding minuend bit and this borrow is carried as carry to the next stage as its input and taken care in the subtraction stage of the next pair of bits. A subtractor can be half or full.

## 6.8 HALF SUBTRACTOR

When we add the least significant bit, we have no carry signal from the previous stage to be inputted hence we need only half adder. For subtracting least significant bits we do not have any carry from the previous stage and we have to concentrate only on the LSB of subtrahend and minuend. Let A and B be the minuend and subtrahend bits respectively. The



difference bit be denoted by D and borrow bit by C. Hence A and B will be input to the half subtractor and C and D will be outputs. The truth table for half subtractor is shown in Table 6.5.

**Table 6.5 Truth Table for Half Subtractor**

| *A* | *B* | *C* | *D* |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |

This table is constructed by the following procedure. First we check the relative magnitudes of A and B and consider the difference between the two bits. For example, if A = 1 and B = 0 then difference A – B = 1. If A = B = 1 then A – B = 0. If A = B = 0 then also A – B = 0. However, if A = 0 and B = 1 then A – B = –1, the minus sign indicates that a borrow from the next higher position is required. A borrow from the next higher position will have different face value. For second significant bit its value will be 2 as the base of binary system is 2. For decimal system the face value of the second significant digit is tens position and a borrow will have value of 10 as the base of decimal system is 10. As we do in decimal system when we borrow from next stage we add 10 to the minuend. For example, if we wish to subtract 5 from 4 we borrow 1 and add 10 to the minuend 4 which makes it 14 and then we subtract 5 from 14 giving us 9. Similarly here in binary subtraction the borrow has a value of 2. Hence when we borrow from the next higher significant bit the borrow has a value of 2 and this should be added to the minuend bit of this stage and then subtraction should be carried out. When a borrow is required to carry out subtraction the value of C will become 1 indicating that a borrow carry has occurred and this borrow carry is carried forward to the next stage.

From the truth table 6.5 we observe that this truth table can be represented by the following Boolean expression:

$D = \overline{A}B + A\overline{B} = A \oplus B$ and

$C = \overline{A}B$

The above expression indicates that the expression for the difference D is same as that obtained for half adder. The realization of subtractor is shown in Fig. 6.15.

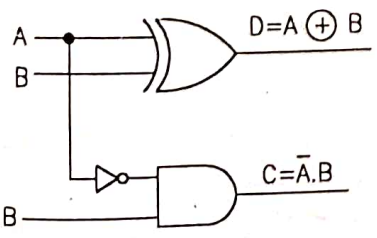



Fig. 6.15 Circuit diagram for Half subtractor

## 6.9 FULL SUBTRACTOR

A full subtractor performs the subtraction involving three bits *i.e.* it takes into account the minuend bit, subtrahend bit and the borrow from the previous stage. It has three inputs X, Y and $C_0$ corresponding to minuend bit, subtrahend bit and borrow from the subtraction process of previous stage. The outputs are the difference D and the borrow to be carried to the next stage B. The truth table for a full subtractor is shown in Table 6.6.

**Table 6.6 Truth table for full subtractor**

| X | Y | $C_0$ | B | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

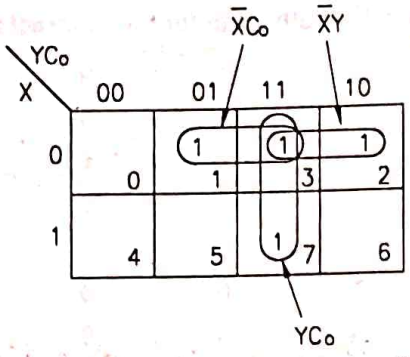
This table is constructed as follows. We subtract $Y + C_0$ from X and look for the difference. When $X = Y = C_0 = 0$, then $X - Y - C_0 = 0$ and both the difference and carry *i.e.* D = 0 and B = 0. When X = 0 and any of Y or $C_0$ is zero the sum $Y + C_0$ will be 1 and to get the difference we have to subtract from X( = 0), $Y + C_0$ (= 1) *i.e.* subtrahend is more than minuend. Hence we have to borrow 1 from the next stage which has a value 2, *i.e.* for subtraction we add 2 to minuend which is 0 and perform the subtraction $X - Y - C_0$ (= 2 – 1) which is 1. The difference D will have a value of 1 and since the borrow is there it will result in a value of 1 for B also *i.e.* when X = 0, Y = 0 and $C_0$ is 1 the value of D and B will be 1 each.

Now if X = 0, Y and $C_0$ both be also 1 the sum of Y and $C_0$ will be 2 (1 + 1) and we have to subtract 2 from 0 which will again require one borrow for carrying out subtraction. For subtracting we add 2 to minuend (X) and subtract the subtrahend ($Y + C_0$ = 2) we get difference D as 0 and since borrow has been there we have B = 1. When X = 1 and $Y = C_0 = 0$ we will have no borrow and difference will be 1 *i.e.* D = 1 and B = 0. If X = 1 and any of Y or $C_0$ is 0 we will not require a borrow and the difference will be (1 – 1) 0 *i.e.* D = 0 and B = 0. When all the three variables are 1 we will require a borrow and the difference will be (3 – 1 – 1) 1 *i.e.* D = 1 and B = 1. When all the three variables are 0 the difference and borrow will also be 0 *i.e.* B = 0 and D = 0.

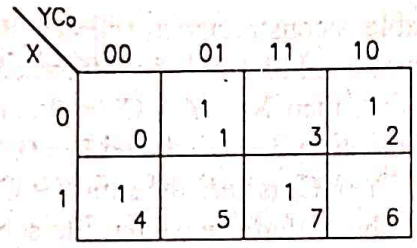
The Karnaugh map for function B and D are shown in Fig. 6.16 (*a*) and (*b*) respectively. From these Karnaugh map we have for the minimal expression for B and D as follows:

$$B = \overline{X}Y + YC_0 + \overline{X}C_0$$

and $$D = \overline{X}\,\overline{Y}C_0 + \overline{X}Y\overline{C}_0 + X\overline{Y}\,\overline{C}_0 + XYC_0$$




(*a*) Karnaugh map for borrow (B)




(*b*) Karnaugh map for difference (D)

Fig. 6.16 Karnaugh map for full adder.

From above we observe that the expression for difference D is same as that for S in case of a full adder. The realization of these expressions is shown in Fig. 6.17. The expression for B is realized in the form of $YC_0 + \overline{X}\,(Y \oplus C_0)$ and for D in the form $X \oplus Y \oplus C_0$.

The design approach of circuits consisting of discrete gates is different compared to combinational logic circuits using LSI or MSI IC. When these circuits are designed to use IC the optimising criteria is not what we have discussed so far rather it depends on the number of interconnections within the chip and other IC manufacturing criteria. It does not matter if some more gates are used for this purpose. While in discrete gate circuit we tried to optimise the number of gates required.

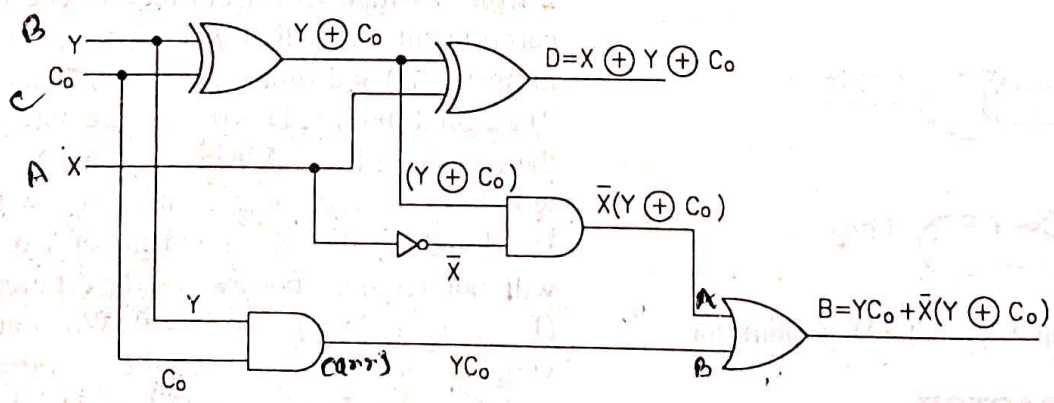



Fig. 6.17 Circuit diagram for Full Subtractors.

## 6.12 ADDER/SUBTRACTOR IN 2'S COMPLEMENT SYSTEM

For adding two signed binary numbers we require complicated arithmetic circuits and to avoid this we use 2's complement system. The negative numbers are represented in the 2's complement system and the addition of positive or negative number is carried out in the same way as we did in 1's complement system. A 2's complement of a number is formed by complementing the number and then adding 1 to the least significant bit of this number, as it has been discussed earlier in first chapter. In this system the addition requires two steps and hence it is time consuming or slow than 1's complement system. However, this system has one advantage of not requiring an end-around carry during addition. When we add two numbers in 2's complement system we come across the four situations as discussed below.

(1) When both the numbers to be added are

positive. This situation is identical to the situation discussed for 1's complement system. For example, let it be required to add decimal number 4 and 3. The addition will be carried out as follows:

| | | |
|---|---|---|
| A = 0100 | = +4 |
| +B = 0011 | = +3 |
| S = 0111 | = +7 |

(2) When one of the numbers to be added is positive and the other is negative and positive number is greater than the negative number. In this case a carry will be generated through the sign bit and as the output of the adders are correct this carry may be neglected. This will be clear by considering the addition of +7 with – 3 which will be carried out as follows:

| | | |
|---|---|---|
| A = +0111 = 0, 0111 | = +7 |
| B = –0011 = 1, 1101 | = –3 |
| + | + |
| [1] 0, 0100 | +4 |

carry to be neglected

For representing – 0011 in 2's complement form we make sign bit 1 and complement 0011 which gives 1100 and to this we add 1 *i.e.* the 2's complement representation of – 0011 will be 1,1101 (the first bit from left indicates sign). To make it more clear let us add + 4 and – 3 which will be carried out as follows:

| | | |
|---|---|---|
| +4 | +0100 | 0,0100 |
| –3 | –0011 | 1,1101 |
| + = | = + |
| +1 | +0001 | [1] 0,0001 |

carry to be neglected

(3) When one of the numbers to be added is positive and other is negative and the magnitude of the negative number is more than that of the positive number. In this case no end around carry will be generated in the sign bit and the answer will be correct as it was in the earlier case. The above will be clear from the following examples:

Let it be required to add – 4 and 3 or + 4 and – 8 which will be carried out as follows:

| Decimal | Binary | 2's Complement |
|---|---|---|
| A = –4 | = –0100 | = 1,1000 |
| B = 3 | = +0011 | = 0,0011 |
| –1 | –0001 | 1,1111 |

For obtaining the correct magnitude we should complement the magnitude bits and add 1 to the least significant bit. The complement of magnitude bits 1111 will be 0000 and on adding 1 we get 0001 which is the correct magnitude (1).

| | | |
|---|---|---|
| A = –8 | = –1000 | = 1,1000 |
| B = 4 | = +0100 | = 0,0100 |
| + | | |
| B = –4 | –0100 | 1,1100 |

The magnitude from 2's complement 1,1100 will be 0100 and the answer in signed magnitude will be 1,0100 when the number of different sign but of same magnitude are added the result will be + 0 which is correct.

(4) When both the numbers to be added are negative a carry will be generated in the sign bit as well as at the most significant bit *i.e.* the bit next to the sign bit. The carry of MSB is placed in the sign bit and the carry generated in the sign bit is neglected. For example let us consider the addition of – 4 and – 3, which can be carried out as follows:

| | | |
|---|---|---|
| A = –4 | = –0100 | = 1,1100 |
| B = –3 | = –0011 | = 1,1101 |
| S = –7 | –0111 | [1] 1,1001 |

carry to be neglected

The equivalent of 2's complement of 1,1001 will be obtained by complementing magnitude bits (1001) to 0110 and adding 1 to give 0111 and the sign bit will remain same, thus the result will be correct as 1,0111 (= –7 ) in signed magnitude binary form.

As the overflow from the sign bits is to be neglected it makes addition of positive and negative numbers very easy in 2's complement system and in most of the parallel systems addition and subtraction is performed in 2's complement system. All the negative numbers are converted in 2's complement system before addition of numbers is performed. The resultant number in 2's complement form is again changed to signed magnitude form. A block diagram for an adder or subtractor of any length is shown in Fig. 6.19 and 2's complement adder is shown in Fig. 6.20.

To perform the addition the signal ADD is held at 0 which results in a carry of 0 to the first stage and one of the inputs for all the XOR gates is zero in each case. We know that in a XOR gate having input X and Y the output will be $\overline{X}Y + X\overline{Y}$ if one of the inputs

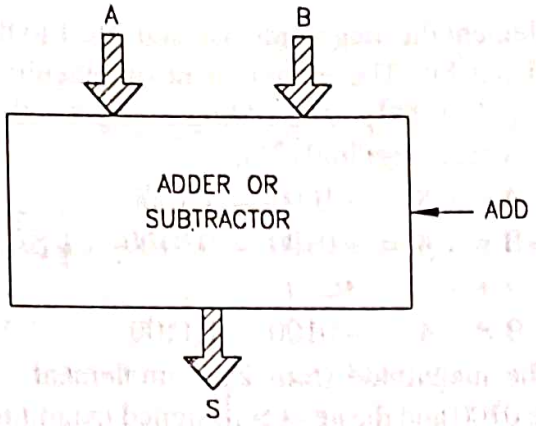



Fig. 6.19 Block diagram for adder or subtractor

significant bit will be 1 which will result in addition of a 1 to the sum of bit $A_1$ and $B_1$. Thus when ADD line is kept 1 and the input numbers are A ( = $A_4 A_3 A_2 A_1$) and B ( = $B_4 B_3 B_2 B_1$) the sum will be represented by S ( = $S_4 S_3 S_2 S_1$) we have from the circuit of Fig. 6.20 for sum

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

or $$S = A + B + 1$$

or $$S_4S_3S_2S_1 = A_4A_3A_2A_1 + \overline{B}_4\overline{B}_3\overline{B}_2\overline{B}_1 + 0001$$

Hence for performing subtraction of a number B

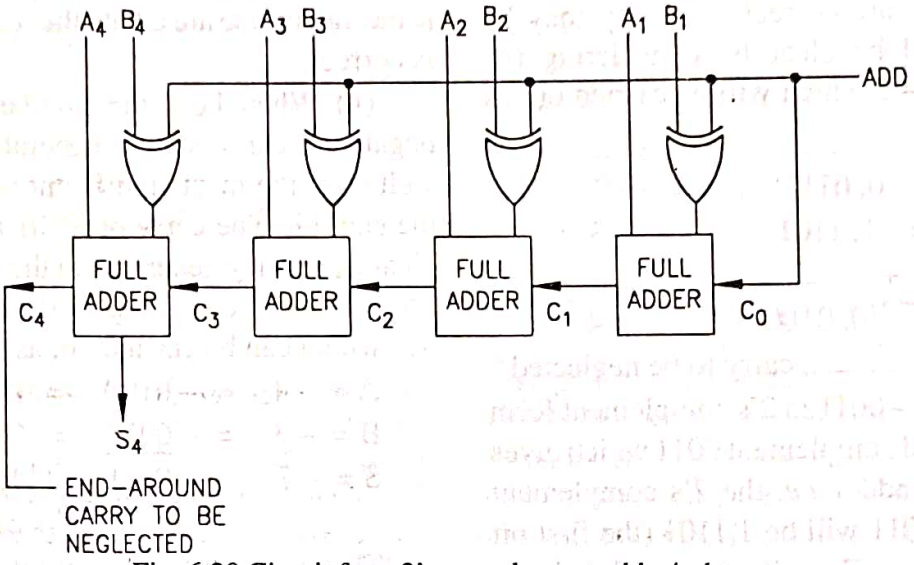



Fig. 6.20 Circuit for a 2's complement adder/subtractor

say Y is 0 the output will reduce to $X\overline{Y}$. As Y is 0 the value of $\overline{Y}$ will be 1. Hence the output will be X only i.e. $X \oplus Y = \overline{X}Y + X\overline{Y} = X\overline{Y} = X$ (when Y = 0). From above we see that the output of exclusive-OR gate will be $B_4 B_3 B_2 B_1$ as the ADD is always kept at 0. The remaining circuit functions just like a full adder described earlier. Thus this circuit can be used as an adder by keeping ADD line as 0.

For using this circuit as subtractor the ADD line is made 1. When the ADD is kept 1 one of the two outputs for the exclusive-OR gates will always be 1 and the output of the exclusive-OR gates will be the complement of the other input to the gate. In fact if X and Y are the inputs to an Exclusive OR gate the output will be = $\overline{X}Y + X\overline{Y}$ and if Y = 1 then $\overline{Y} = 0$. Hence if Y = 1 the output of the exclusive gate will be $\overline{X}$ only. Thus the output of the exclusive gates will be the complement of the bits of one of the number say B here *i.e.* the output from the exclusive OR gate will be $\overline{B}_4, \overline{B}_3, \overline{B}_2, \overline{B}_1$ (complement of the number). The carry bit to the first adder for least from A we perform addition of A with (– B). For representing B in 2's complement from first we complement all the bits of the number B by inputting each bit of this number to individual exclusive-OR gate and to this 1's complemented form a 1 is added by using logic 1 state of ADD line. Feeding the LSB of A, complemented B and carry $C_0$ as 1 we get the addition of the LSBs of A and B and the remaining adders work in the same way as was in the case of 1's complement addition. The end-around carry of the final stage is neglected.

The circuit for addition and subtraction of two signed magnitude number of 2's complement form is shown in Fig. 6.21 using AND, OR and Invertor gates. This is the complete logical circuit capable of performing addition or subtraction of any two signed 2's complement numbers. The two numbers are A and B with $A_1$ and $B_1$ being LSB. It makes use of two control signals ADD and SUBTRACT. If none of these signals is 1 i.e. both are 0 the output from all the five adders will be zero. If ADD is 1 it will perform

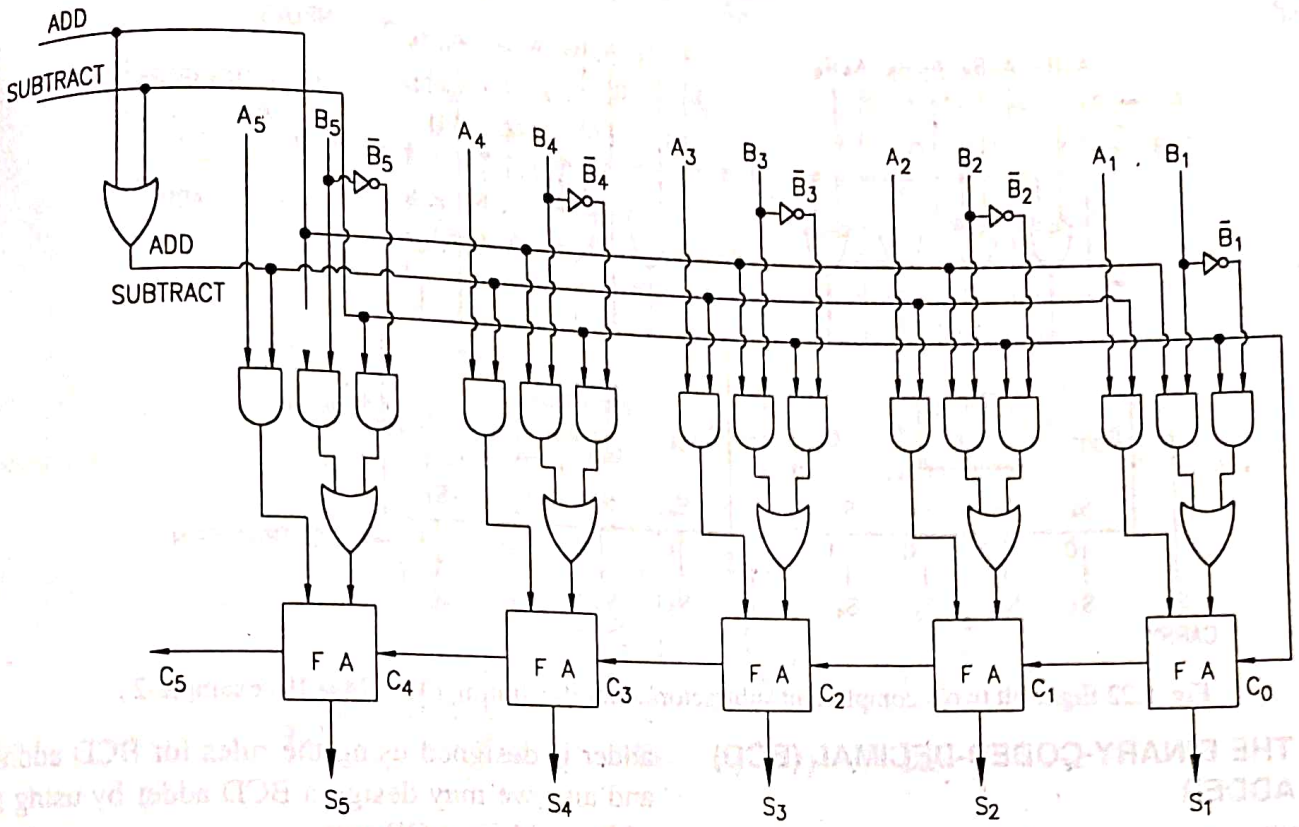



Fig. 6.21 Circuit for addition and subtraction of two signed numbers in 2's complement form.

addition, if SUBTRACT is 1 it will perform subtraction. The AND-to-OR network allows us to select either B or $\overline{B}$. For ADD signal we have all the bits of B going to the respective full adders and SUBTRACT signal makes complemented bit of B *i.e.* of $\overline{B}$ to go to the respective full adders. This configuration is mostly used for addition and subtraction of positive and negative numbers. In this circuit no care is taken of the overflow. However, we may use any circuitary available for the detection of overflow.

**Example 2.** *Design a circuit to perform two's-complement arithmetic for an eight-bit two's complement adder/subtractor making use of 4008 ICs. Show external connections and illustrate the subtraction 43–24 = 19 by labelling the input and output lines.*

**Solution.** We know that positive two's complement numbers are exactly same as regular true binary numbers and can be added using regular binary addition. Also, subtraction in two's complement arithmatic is performed by converting the number to be subtracted to a negative number in the two's-compliment form and then using regular binary addition. Hence once our numbers are in two's complement form, one can use a binary adder to get the answer whether one is adding or subtracting. What we infact need is an input switch or signal to signify addition or subtraction so that we will know whether to form a positive or a negative two's complement of the second number. Then we just use a binary adder to get the result.

To form negative two's complement of a number we make use of the controlled inverter circuit (article 6.5) and add 1 to its output. The complete circuit is shown in Fig. 6.22. The eight bit number ($A_7$ ... $A_0$) is brought directly on the A inputs into the adders. The other eight-bit number ($B_7$ ... $B_0$) comes in on the B input lines. If the subtraction operation is to be performed, the complementing switch is kept ON (position-1), causing each bit of the B number to be complemented (one's complement). At the same time, the low order $C_{IN}$ is set to 1, which results in adding 1 to the already complemented B number, thus making it a negative two's complement number.

The regular binary addition is performed by 4008s. If the complementary switch is down (0) the sum is taken. If it is up (1) the number on the B inputs is subtracted from the number on the A inputs.

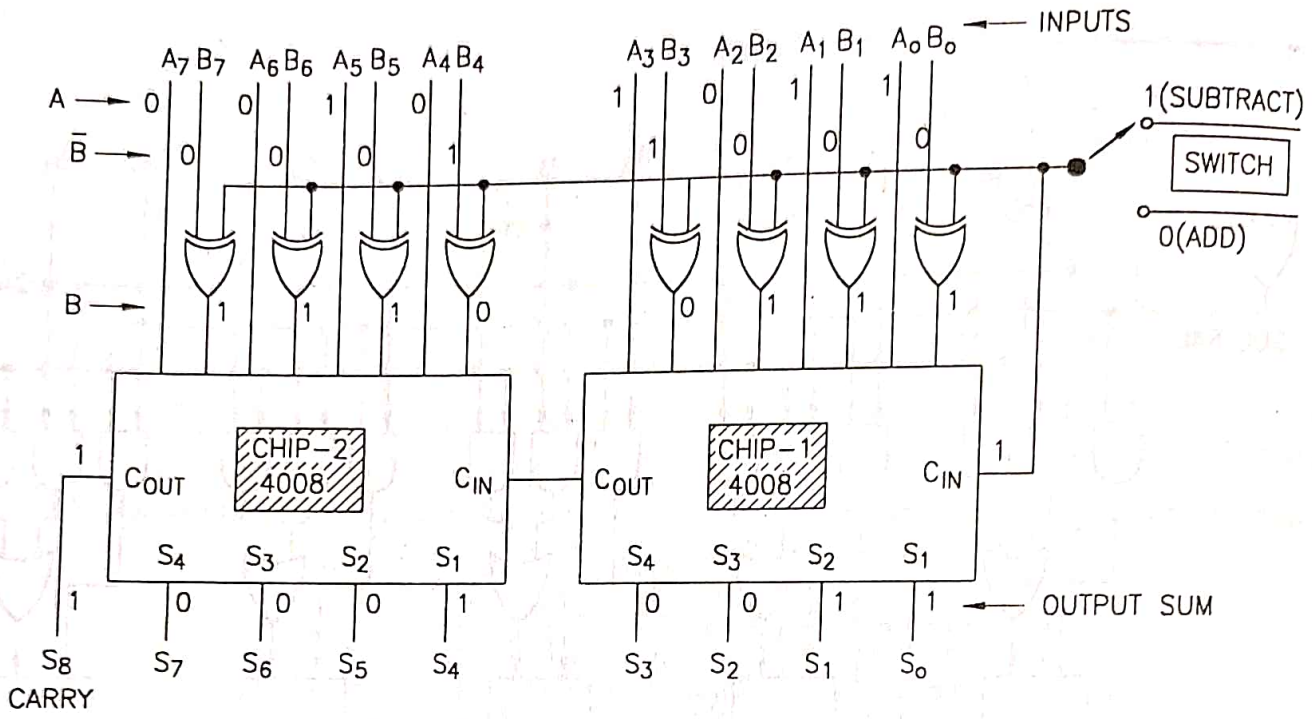



Fig. 6.22 Eight-bit two's compliment subtractor/adder illustrating (43 – 24 = 19) example-2 .

Fig. 6.51 Circuit for realizing

## 6.23.2 Eight Line to one Line Multiplexes

We can also design an eight input multiplexer using three control signals. the truth table for such a multiplexer is shown in Table 6.17. The bit selected for output is determined by the appropriate input address line $S_2$ $S_1$ $S_0$ out of the 8 input lines. The output will be available only when the Enable line is a high or 1 (accordingly $\overline{E}$ will be 0 as shown in truth table). The circuit realization of the above truth table is shown in Fig. 6.52. The circuit is enabled only when $\overline{E}$ is 0 and is disabled when $\overline{E}$ is a 1 and the output Z is a 0. The terms which do not matter or on which the output does not depend are marked as don't care terms and denoted by *d* in the truth table. There are commercially available four input, eight input and sixteen input multiplexers. From the circuit shown if a select address is 101 ($S_2 = 1$, $S_1 = 0$ and $S_0 = 1$) and $\overline{E}$ is 0 it will cause input bit $I_5$ to be steered to the output if $I_5$

**Table 6.17 Truth table for 1 to 8 multiplexer**

| Enable line | Control Line | | | Input Signals | | | | | | | | Output | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{E}$ | $S_2$ | $S_1$ | $S_0$ | $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $\overline{Z}$ | Z |
| 1 | *d* | d | *d* | *d* | *d* | *d* | d | d | *d* | d | d | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | *d* | d | d | d | *d* | *d* | d | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | *d* | d | d | d | *d* | d | d | 0 | 1 |
| 0 | 0 | 0 | 1 | *d* | 0 | d | d | d | *d* | d | d | 1 | 0 |
| 0 | 0 | 0 | 1 | *d* | 1 | d | d | d | *d* | d | d | 0 | 1 |
| 0 | 0 | 1 | 0 | *d* | d | 0 | d | d | *d* | d | d | 1 | 0 |
| 0 | 0 | 1 | 0 | *d* | d | 1 | d | d | *d* | d | d | 0 | 1 |
| 0 | 0 | 1 | 1 | *d* | d | d | 0 | d | *d* | d | d | 1 | 0 |
| 0 | 0 | 1 | 1 | *d* | d | d | 1 | d | *d* | d | d | 0 | 1 |
| 0 | 1 | 0 | 0 | *d* | d | d | d | 0 | d | d | d | 1 | 0 |
| 0 | 1 | 0 | 0 | *d* | d | d | d | 1 | d | d | d | 0 | 1 |
| 0 | 1 | 0 | 1 | *d* | d | d | d | d | 0 | *d* | d | 1 | 0 |
| 0 | 1 | 0 | 1 | *d* | d | d | d | d | 1 | *d* | d | 0 | 1 |
| 0 | 1 | 1 | 0 | *d* | d | d | d | d | *d* | 0 | d | 1 | 0 |
| 0 | 1 | 1 | 0 | *d* | d | d | d | d | *d* | 1 | d | 0 | 1 |
| 0 | 1 | 1 | 1 | *d* | d | d | d | d | *d* | d | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | *d* | d | d | d | d | *d* | d | 1 | 0 | 1 |

is 0 Z will be 0 and if $I_5$ is 1 Z will also be a 1. The logic symbol for eight input multiplexer is shown in Fig. 6.53. A multiplexer logic circuit can be used for generating random logic functions. This can be realized using AND, OR, INV, NAND and NOR gates, but it is more advantageous to use a multiplexer for this purpose. A 8 input multiplexer can be used to generate any possible function of the three control signals.

Some times it is more useful to use a dual four input multiplexer with comsmon input select logic. In this demultiplexer we have two circuits consisting of four inputs and an output along with its complement.

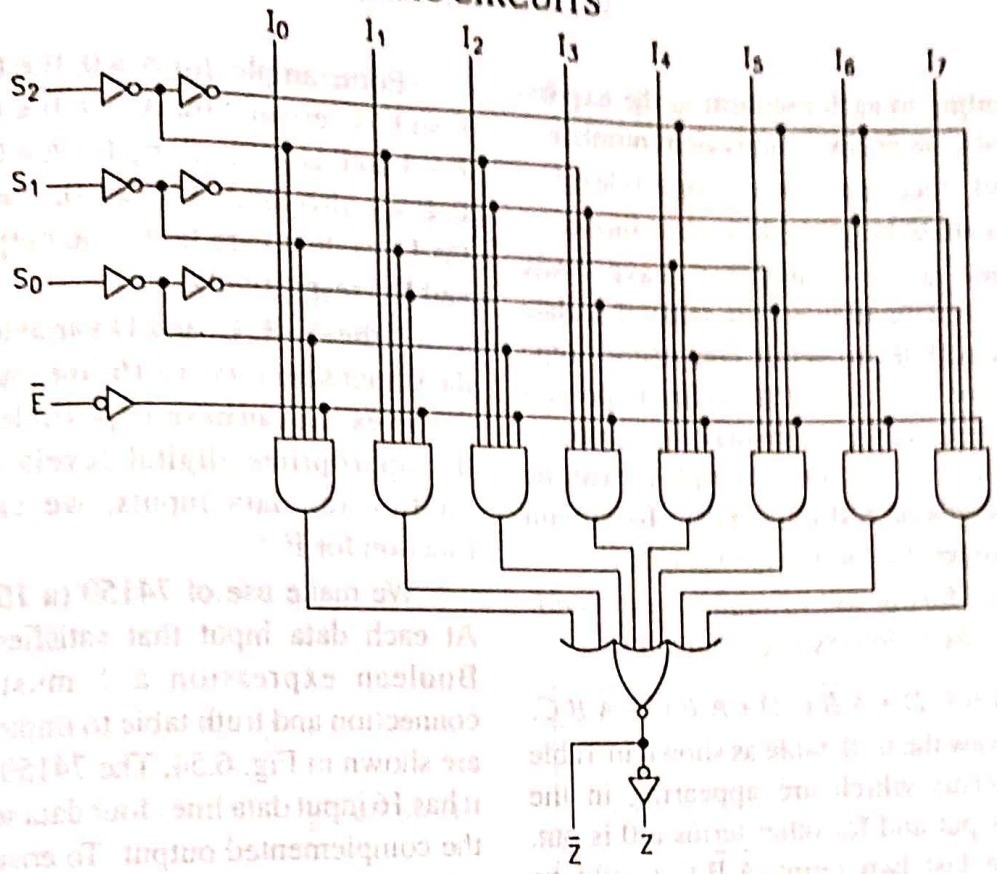



Fig. 6.52 Circuit for realization of 1 to 8 demultiplexer

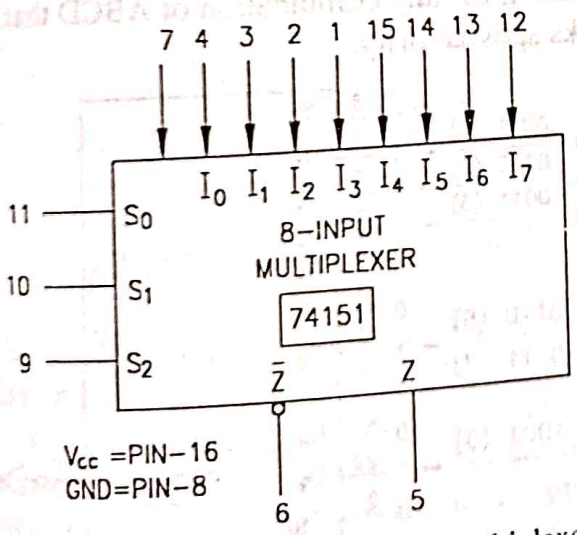



Fig. 6.53 Logic symbol for a 8-input demultiplexer

This unit can also be used to generate any two functions of three variables in addition to its use as a multiplexer.