

SWITCHING THEORY AND LOGIC DESIGN

UNIT I - syllabus

NUMBER SYSTEM & BOOLEAN ALGEBRA

Digital systems, Binary Numbers, Number base conversions, Complements of numbers, Signed Binary numbers, Binary codes. Boolean algebra - Basic definition, Basic theorems and properties, Boolean Functions, Canonical & Standard forms, other logic operations & Digital logic gates.

UNIT I- DIGITAL SYSTEMS AND BINARY NUMBERS

DIGITAL SYSTEMS

Digital systems have such a prominent role in everyday life that we refer to the present technological period as the digital age. Digital systems are used in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises.

The most striking property of the digital computer is its generality. It can follow a sequence of instructions, called a program that operates on given data. The user can specify and change the program or the data according to the specific need. Because of this flexibility, general-purpose digital computers can perform a variety of information-processing tasks that range over a wide spectrum of applications.

One characteristic of digital systems is their ability to represent and manipulate discrete elements of information. Discrete elements of information are represented in a digital system by physical quantities called signals. Electrical signals such as voltages and currents are the most common. The signals in most present-day electronic digital systems use just two discrete values and are therefore said to be binary. A binary digit, called a bit. It has two values: 0 and 1.

Discrete quantities of information either emerge from the nature of the data being processed or may be quantized from a continuous process. This process is called Analog to Digital conversion.

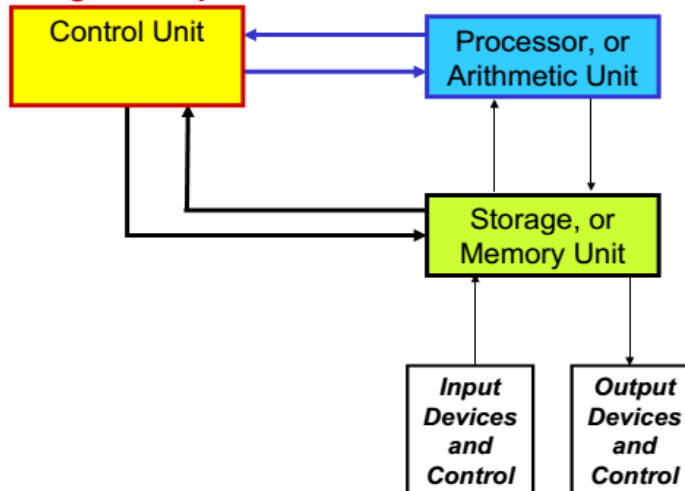
DIGITAL SYSTEMS: manipulate discrete elements of information (finite sets)

- E.g. the 10 decimal digits, the 26 letters of the alphabet, 64 squares of chess board)

The general-purpose digital computer is the best-known example of a digital system. The major parts of a computer are a memory unit, a central processing unit, and input-output units. The memory unit stores programs as well as input, output, and intermediate data. The central processing unit performs arithmetic and other data-processing operations as specified by the program. The program and data prepared by a user are transferred into memory by means of an input device such as a keyboard. An output device, such as a printer, receives the results of the computations, and the printed results are presented to the user.

There are fundamental reasons that commercial products are made with digital circuits. Like a digital computer, most digital devices are programmable. By changing the program in a programmable device, the same underlying hardware can be used for many different applications

Block diagram of a digital computer



BINARY NUMBERS

A decimal number such as 7,392 represents a quantity equal to 7 thousands, plus 3 hundreds, plus 9 tens, plus 2 units. The thousands, hundreds, etc., are powers of 10 implied by the position of the coefficients (symbols) in the number. To be more exact, 7,392 is a shorthand notation for what should be written as

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

However, the convention is to write only the numeric coefficients and, from their position, deduce the necessary powers of 10 with powers increasing from right to left. In general, a number with a decimal point is represented by a series of coefficients:

$$a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3}$$

The coefficients a_j are any of the 10 digits (0, 1, 2, ..., 9), and the subscript value j gives the place value and, hence, the power of 10 by which the coefficient must be multiplied. Thus, the preceding decimal number can be expressed as

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$$

with $a_3 = 7$, $a_2 = 3$, $a_1 = 9$, and $a_0 = 2$.

The decimal number system is said to be of base, or radix, 10 because it uses 10 digits and the coefficients are multiplied by powers of 10. The binary system is a different number system. The coefficients of the binary number system have only two possible values: 0 and 1. Each coefficient a_j is multiplied by a power of the radix, e.g., 2^j and the results are added to obtain the decimal equivalent of the number.

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

There are many different number systems. In general, a number expressed in a base- r system has coefficients multiplied by powers of r :

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \dots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \dots + a_{-m} \cdot r^{-m}$$

An example of an octal number is 127.4. To determine its equivalent decimal value, we expand the number in a power series with a base of 8:

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

For example, in the hexadecimal(base-16) number system, the first 10 digits are borrowed from the decimal system. The letters A, B, C, D, E, and F are used for the digits 10, 11, 12, 13, 14, and 15, respectively. An example of a hexadecimal number is

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46,687)_{10}$$

Examples of addition, subtraction, and multiplication of two binary numbers are as follows:

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	<u>+100111</u>	subtrahend:	<u>-100111</u>	multiplier:	<u>× 101</u>
sum:	1010100	difference:	000110		<u>1011</u>
				partial product:	<u>0000</u>
					<u>1011</u>
				product:	110111

Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

NUMBER-BASE CONVERSIONS

The conversion of a number in base r to decimal is done by expanding the number in a power series and adding all the terms as shown previously. We now present a general procedure for the reverse operation of converting a decimal number to a number in base r . The conversion of a decimal integer to a number in base r is done by dividing the number and all successive quotients by r and accumulating the remainders.

BINARY TO DECIMAL:

A binary number can be converted to a decimal by forming the sum of the powers of 2 of those coefficients whose value is 1.

$$\begin{aligned} (1010.011)_2 &= 2^3 + 2^1 + 2^{-2} + 2^{-3} \\ \text{Example:} &= (10.375)_{10} \end{aligned}$$

OCTAL TO DECIMAL

$$\begin{aligned} (630.4)_8 &= 6 \times 8^2 + 3 \times 8 + 4 \times 8^{-1} \\ &= (408.5)_{10} \end{aligned}$$

DECIMAL TO BINARY

$$\begin{array}{r|l} (160)_{10} & 2 \quad 160 \\ \hline & 2 \quad 80 \quad 0 \\ \hline & 2 \quad 40 \quad 0 \\ \hline & 2 \quad 20 \quad 0 \\ \hline & 2 \quad 10 \quad 0 \\ \hline & 2 \quad 5 \quad 0 \\ \hline & 2 \quad 2 \quad 1 \\ \hline & 1 \quad 1 \quad 0 \end{array} = (10100000)_2$$

$$(41)_{10} = (\quad)_2$$

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder
41	
20	1
10	0
5	0
2	1
1	0
0	1

101001 = answer

Conversion from decimal integers to any base- r system is similar to this example, except that division is done by r instead of 2.

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

DECIMAL TO OCTAL

Find the Octal equivalent for Decimal 214

8	214	6	<div style="display: flex; align-items: center;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg); font-weight: bold; margin-right: 5px;">Remainder</div> <div style="border-left: 1px solid black; border-right: 1px solid black; height: 100px; margin: 0 5px;"></div> <div style="font-weight: bold; margin-left: 5px;">LSD</div> </div> <div style="margin-top: 10px; display: flex; align-items: center;"> <div style="border-left: 1px solid black; border-right: 1px solid black; height: 100px; margin: 0 5px;"></div> <div style="font-weight: bold; margin-left: 5px;">MSD</div> </div>
8	26	2	
8	3	3	
	0		
Divisor			
	Quotient		

MSD - most significant digit

LSD - least significant digit

Therefore, the Octal equivalent for decimal 214 is 326

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

153	
19	1
2	3
0	$2 = (231)_8$

Convert $(0.513)_{10}$ to octal.

$$0.513 \times 8 = 4.104$$

$$0.104 \times 8 = 0.832$$

$$0.832 \times 8 = 6.656$$

$$0.656 \times 8 = 5.248$$

$$0.248 \times 8 = 1.984$$

$$0.984 \times 8 = 7.872$$

The answer, to seven significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517 \dots)_8$$

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers. Using the results of Examples 1.1 and 1.3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important role in digital computers, because shorter patterns of hex characters are easier to recognize than long patterns of 1's and 0's. Since $2^3=8$ and $2^4=16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group.

The following example illustrates the procedure:

$$\begin{array}{cccccccccccc} (10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110) & _2 & = & (26153.7406) & _8 \\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 & & & & \end{array}$$

Conversion from binary to hexadecimal is similar, except that the binary number is Divided into groups of four digits:

$$\begin{array}{cccccccc} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010) & _2 & = & (2C6B.F2) & _{16} \\ 2 & C & 6 & B & & F & 2 & & & & \end{array}$$

Conversion from octal or hexadecimal to binary is done by reversing the preceding procedure. Each octal digit is converted to its three-digit binary equivalent. Similarly, each hexadecimal digit is converted to its four-digit binary equivalent. The procedure is illustrated in the following examples:

$$(673.124)_8 = \begin{matrix} (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2 \\ 6 & 7 & 3 & & 1 & 2 & 4 \end{matrix}$$

and

$$(306.D)_{16} = \begin{matrix} (0011 & 0000 & 0110 & \cdot & 1101)_2 \\ 3 & 0 & 6 & & D \end{matrix}$$

Thus, the binary number 111111111111 has 12 digits and is expressed in octal as 7777 (4 digits) or in hexadecimal as FFF (3 digits). During communication between people (about binary numbers in the computer), the octal or hexadecimal representation is more desirable because it can be expressed more compactly with a third or a quarter of the number of digits required for the equivalent binary number. Thus, most computer manuals use either octal or hexadecimal numbers to specify binary quantities ..

COMPLEMENTS OF NUMBERS

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations. There are two types of complements for each base-r system: the radix complement and the diminished radix complement. The first is referred to as the r's complement and the second as the (r-1)'s complement. When the value of the base r is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

DIMINISHED RADIX COMPLEMENT--- (r-1) COMPLEMENT

Given a number N in base r having n digits, the (r-1)'s complement of N, i.e., its diminished radix complement, is defined as $(r^n-1)-N$. For decimal numbers, $r=10$ and $r-1=9$, so the 9's complement of N is $(10^n-1)-N$. In this case, 10^n represents a number that consists of a single 1 followed by n 0's. 10^n-1 is a number represented by n 9's. For example, if $n=4$, we have $10^4=10,000$ and $10^4-1=9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. Here are some numerical examples:

The 9's complement of 546700 is $999999 - 546700 = 453299$.

The 9's complement of 012398 is $999999 - 012398 = 987601$.

For binary numbers, $r=2$ and $r-1=1$, so the 1's complement of N is $(2^n-1)-N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's. 2^n-1 is a binary number represented by n 1's. For example, if $n=4$, we have $2^4=(10000)_2$ and $2^4-1=(1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can have either $1-0=1$ or $1-1=0$, which causes the bit to change from 0 to 1 or from 1 to 0, respectively. Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's. The following are some numerical examples:

The 1's complement of 1011000 is 0100111.

The 1's complement of 0101101 is 1010010.

The (r-1)'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

RADIX COMPLEMENT (r's Complement)

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N=0$. Comparing with the $(r-1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r-1)$'s complement, since $r^n - N = [(r-1) - N] + 1$. Thus, the 10's complement of decimal 2389 is $7610 + 1 = 7611$ and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's-complement value. Since 10 is a number represented by a 1 followed by n 0's, $10^n - N$, which is the 10's complement of N, can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9. Thus,

the 10's complement of 012398 is 987602 and

the 10's complement of 246700 is 753300

The 10's complement of the first number is obtained by subtracting 8 from 10 in the least significant position and subtracting all other digits from 9. The 10's complement of the second number is obtained by leaving the two least significant 0's unchanged, subtracting 7 from 10, and subtracting the other three digits from 9.

Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged and replacing 1's with 0's and 0's with 1's in all other higher significant digits. For example,

the 2's complement of 1101100 is 0010100

and the 2's complement of 0110111 is 1001001

The 2's complement of the first number is obtained by leaving the two least significant 0's and the first 1 unchanged and then replacing 1's with 0's and 0's with 1's in the other four most significant digits. The 2's complement of the second number is obtained by leaving the least significant 1 unchanged and complementing all other digits.

SUBTRACTION WITH COMPLEMENTS

when subtraction is implemented with digital hardware, the borrow method is less efficient than the method that uses complements.

The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically,

$$M + (r^n - N) = M - N + r^n.$$

2. If $M \geq N$, the sum will produce an end carry r^n , which can be discarded; what is left is the result $M - N$.

3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

EXAMPLE 1.5

Using 10's complement, subtract $72532 - 3250$.

$$\begin{array}{r}
 M = \quad 72532 \\
 10\text{'s complement of } N = + \quad 96750 \\
 \text{Sum} = \quad 169282 \\
 \text{Discard end carry } 10^5 = - \quad 100000 \\
 \text{Answer} = \quad 69282
 \end{array}$$

Note that M has five digits and N has only four digits. Both numbers must have the same number of digits, so we write N as 03250. Taking the 10's complement of N produces a 9 in the most significant position. The occurrence of the end carry signifies that $M \geq N$ and that the result is therefore positive.

EXAMPLE 1.6

Using 10's complement, subtract $3250 - 72532$.

$$\begin{array}{r}
 M = \quad 03250 \\
 10\text{'s complement of } N = + \quad 27468 \\
 \text{Sum} = \quad 30718
 \end{array}$$

There is no end carry. Therefore, the answer is $-(10\text{'s complement of } 30718) = -69282$. Note that since $3250 < 72532$, the result is negative

EXAMPLE 1.7

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ by using 2's complements.

$$\begin{array}{r}
 \text{(a)} \quad X = \quad 1010100 \\
 2\text{'s complement of } Y = + \quad 0111101 \\
 \text{Sum} = \quad 10010001 \\
 \text{Discard end carry } 2^7 = - \quad 10000000 \\
 \text{Answer: } X - Y = \quad 0010001 \\
 \\
 \text{(b)} \quad Y = \quad 1000011 \\
 2\text{'s complement of } X = + \quad 0101100 \\
 \text{Sum} = \quad 1101111
 \end{array}$$

There is no end carry. Therefore, the answer is $Y - X = -(2\text{'s complement of } 1101111) = -0010001$.

Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less than the r 's complement. Because of this, the result of adding the minuend to the complement of the subtrahend produces a sum that is one less than the correct difference when an end carry occurs. Removing the end carry and adding 1 to the sum is referred to as an end-around carry.

EXAMPLE 1.8

Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$\begin{array}{r} X = 1010100 \\ 1\text{'s complement of } Y = + 0111100 \\ \hline \text{Sum} = 10010000 \\ \text{End-around carry} = + \quad \quad \quad 1 \\ \hline \text{Answer: } X - Y = 0010001 \end{array}$$

(b) $Y - X = 1000011 - 1010100$

$$\begin{array}{r} Y = 1000011 \\ 1\text{'s complement of } X = + 0101011 \\ \hline \text{Sum} = 1101110 \end{array}$$

There is no end carry. Therefore, the answer is $Y - X = -(1\text{'s complement of } 1101110) = -0010001$.

SIGNED BINARY NUMBERS

Positive integers (including zero) can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with binary digits. The convention is to make the sign bit 0 for positive and 1 for negative.

The user determines whether the number is signed or unsigned. If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number. If the binary number is assumed to be unsigned, then the leftmost bit is the most significant bit of the number. For example, the string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary) because the leftmost bit is 0. The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of -9 when considered as a signed number.

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

As an example, consider the number 9, represented in binary with eight bits. +9 is represented with a sign bit of 0 in the leftmost position, followed by the binary equivalent of 9, which gives 00001001. Note that all eight bits must have a value; therefore, 0's are inserted following the sign bit up to the first 1. Although there is only one way to represent +9, there are three different ways to represent -9 with eight bits:

signed-magnitude representation: 10001001

signed-1's-complement representation: 11110110

signed-2's-complement representation: 11110111

ARITHMETIC ADDITION

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are the same, we add the two magnitudes and give the sum the common sign. If the signs are different, we subtract the smaller magnitude from the larger and give the difference the sign of the larger magnitude. For example, $(+25)+(-37)=-37-25=-62$ is done by subtracting the smaller magnitude, 25, from the larger magnitude, 37, and appending the sign of 37 to the result.

The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded.

Numerical examples for addition follow:

$$\begin{array}{r} + 6 \quad 00000110 \\ +13 \quad 00001101 \\ \hline +19 \quad 00010011 \end{array} \qquad \begin{array}{r} - 6 \quad 11111010 \\ +13 \quad 00001101 \\ \hline + 7 \quad 00000111 \end{array}$$
$$\begin{array}{r} + 6 \quad 00000110 \\ -13 \quad 11110011 \\ \hline - 7 \quad 11111001 \end{array} \qquad \begin{array}{r} - 6 \quad 11111010 \\ -13 \quad 11110011 \\ \hline -19 \quad 11101101 \end{array}$$

ARITHMETIC SUBTRACTION

Subtraction of two signed binary numbers when negative numbers are in 2's-complement form is simple and can be stated as follows:

Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign-bit position is discarded.

To see this, consider the subtraction $(-6)-(-13)=+7$. In binary with eight bits, this operation is written as $(11111010-11110011)$. The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13) , giving $(+13)$. In binary, this is $11111010+00001101=100000111$. Removing the end carry, we obtain the correct answer: $00000111 (+7)$.

BINARY CODES

An n-bit binary code is a group of n bits that assumes up to 2^n distinct combinations of 1's and 0's, with each combination representing one element of the set that is being coded. A set of four elements can be coded with two bits, with each element assigned one of the following bit combinations: 00, 01, 10, 11. A set of eight elements requires a three-bit code and a set of 16 elements requires a four-bit code. The bit combination of an n-bit code is determined from the count in binary from 0 to 2^n-1 .

WEIGHTED CODES

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. Eg. BCD, 8421, 84-2-1.

BINARY-CODED DECIMAL CODE

The code most commonly used for the decimal digits is the straight binary assignment listed in Table 1.4 . This scheme is called **binary-coded decimal** and is commonly referred to as BCD. Other decimal codes are possible and a few of them are presented later in this section. Table 1.4 gives the four-bit code for one decimal digit. A number with k decimal digits will require 4kbits in BCD. Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit. A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.

$$(185)_{10} = (0001\ 1000\ 0101)_{\text{BCD}} = (10111001)_2$$

Table 1.4
Binary-Coded Decimal (BCD)

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

BCD ADDITION

1. Convert the given decimal to its equivalent BCD code
2. The given number are to be added using the rule of binary.
3. The result of addition is less than 9, which is valid for BCD numbers.
4. If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is $(0110)_2$ to the result of addition. Then the resultant that we would get will be a valid binary coded number.

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

In each case, the two BCD digits are added as if they were two binary numbers. If the binary sum is greater than or equal to 1010, we add 0110 to obtain the correct BCD sum and a carry. In the second example, the binary sum produces an invalid BCD digit. The addition of 0110 produces the correct BCD sum, 0010 (i.e., the number 2), and a carry. In the third example, the binary sum produces a carry. This condition occurs when the sum is greater than or equal to 16. Although the other four bits are less than 1001, the binary sum requires a correction because of the carry. Adding 0110, we obtain the required BCD sum 0111 (i.e., the number 7) and a BCD carry.

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

BCD SUBTRACTION:

- At first the decimal equivalent of the given Binary Coded Decimal (BCD) codes are found out.
- Then the 9's complement of the subtrahend is done and then that result is added to the number from which the subtraction is to be done.
- If there is any carry bit then the carry bit may be added to the result of the subtraction.
- If carry bit is not generated then the result is negative, to obtain the answer find the 9's complement of the sum and place -ve sign in front.

Eg 1. 8-3

9's complement of 3 is 6 → 0110 ADD

$$\begin{array}{r}
 8 \rightarrow 1000 \\
 \hline
 1110 \text{ (14>9)} \\
 \hline
 0110 \text{ ADD 6} \\
 \hline
 10100 \\
 \text{1 ADD CARRY} \\
 \hline
 0101 \text{ (5)}
 \end{array}$$

Eg . 2. 24-68

$$24 \rightarrow 0010 \ 0100$$

9's complement of 68 is 31 → 0011 0001 (ADD)

$$0101 \ 0101 \text{ (55) with no carry}$$

9's complement of 55 is 44 → -0100 0100

NON WEIGHTED CODES

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: excess-3 code, gray code

EXCESS THREE (XS-3)CODE:

It is a non-weighted BCD code .Each binary code word is the corresponding 8421 code word plus 0011(3).It is a sequential code & therefore it can be used for arithmetic operations. It is a self-complementing .

Excess-3 Addition:

Add the xs-3 no.s by adding the 4 bit groups in each column starting from the LSD. If there is no carry starting from the addition of any of the 4-bit groups , subtract 0011 from the sum term of those groups (because when 2 decimal digits are added in xs-3 & there is no carry , result in xs-6). If there is a carry out, add 0011 to the sum term of those groups(because when there is a carry, the invalid states are skipped and the result is normal binary).

$$\begin{array}{r}
 \text{EX: } 37 \quad \quad 0110 \quad \quad 1010 \\
 +28 \quad \quad +0101 \quad \quad 1011 \\
 \hline
 65 \quad \quad 1011 \quad (1)0101 \quad \text{carry generated} \\
 \quad \quad +1 \quad \quad \leftarrow \quad \quad \text{propagate carry} \\
 \hline
 \quad \quad 1100 \quad \quad 0101 \quad \quad \text{add 0011 to correct 0101 \& } \\
 \quad \quad -0011 \quad \quad +0011 \quad \quad \text{subtract 0011 to correct 1100} \\
 \hline
 \quad \quad 1001 \quad \quad 1000 \quad \quad =65_{10}
 \end{array}$$

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, -2, -1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111

REFLECTIVE CODE

A code is said to be reflective when code for 9 is complement for the code for 0, and so is for 8 and 1

codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is

not.

THE GRAY CODE (REFLECTIVE –CODE):

Gray code is a non-weighted code & is not suitable for arithmetic operations. It is not a BCD code. It is a cyclic code because successive code words in this code differ in one bit position only i.e, it is a unit distance code. Popular of the unit distance code. It is also a reflective code i.e both reflective & unit distance.

Table 1.6
Gray Code

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

BINARY CODE TO GRAY CODE

Let Binary code be $b_3 b_2 b_1 b_0$. Then the respective Gray Code can be obtained is as follows

$$\begin{array}{l} \text{Binary code : } b_3 \oplus b_2 \oplus b_1 \oplus b_0 \\ \downarrow \\ \text{Gray code : } g_3 \quad g_2 \quad g_1 \quad g_0 \\ \quad (b_3) \quad (b_3 \oplus b_2) \quad (b_2 \oplus b_1) \quad (b_1 \oplus b_0) \end{array}$$

$$g_3 = b_3$$

$$g_2 = b_3 \oplus b_2$$

$$g_1 = b_2 \oplus b_1$$

$$g_0 = b_1 \oplus b_0$$

Example:

Binary Code: $b_3 \ b_2 \ b_1 \ b_0 = 1 \ 1 \ 1 \ 0$

$$g_3 = b_3 = 1$$

$$g_2 = b_3 \oplus b_2 = 1 \oplus 1 = 0$$

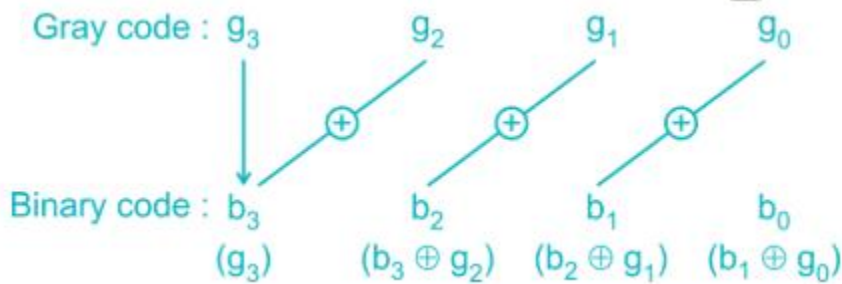
$$g_1 = b_2 \oplus b_1 = 1 \oplus 1 = 0$$

$$g_0 = b_1 \oplus b_0 = 1 \oplus 0 = 1$$

\therefore Final Gray code: 1 0 0 1



GRAY TO BINARY



Gray Code: $g_3 \ g_2 \ g_1 \ g_0 = 1 \ 0 \ 0 \ 1$ then Binary Code: $b_3 \ b_2 \ b_1 \ b_0$

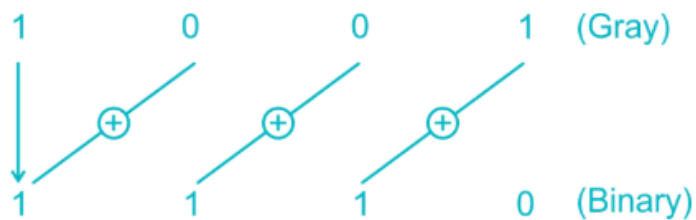
$$b_3 = g_3 = 1$$

$$b_2 = b_3 \oplus g_2 = 1 \oplus 0 = 1$$

$$b_1 = b_2 \oplus g_1 = 1 \oplus 0 = 1$$

$$b_0 = b_1 \oplus g_0 = 1 \oplus 1 = 0$$

\therefore Final Binary Code: 1 1 1 0



ERROR – DETECTING CODES: When binary data is transmitted & processed, it is susceptible to noise

that can alter or distort its contents. The 1's may get changed to 0's & 1's .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals. Popular modern alphanumeric codes are ASCII code & EBCDIC code.

ASCII Character Code

Many applications of digital computers require the handling not only of numbers, but also of other characters or symbols, such as the letters of the alphabet. The standard binary code for the alphanumeric characters is the American Standard Code for Information Interchange (ASCII), which uses seven bits to code 128 characters.

SELF-COMPLEMENT CODE:

XS -3 is a self complement code because the 1s complement of an xs-3 is equal to the xs-3 code for 9's complement of respective decimal no.

Eg. $(2)_{10}$

XS -3 code for 2 is 0101

1's comp. of XS-3 for 2 is 1010

9's comp. of 2 is 7

XS -3 code for 7 is 1010

Decimal	8 4 2 1 CODE BCD CODE	XS 3
0	0000 +0011	0011
1	0001 +0011	0100
2	0010 +0011	0101
3	0011 +0011	0110
4	0100 +0011	0111
5	0101 +0011	1000
6	0110 +0011	1001
7	0111 +0011	1010
8	1000 +0011	1011
9	1001 +0011	1100

$9 = 0'$
 $8 = 1'$

LEARN

BOOLEAN ALGEBRA AND LOGIC GATES

BASIC DEFINITIONS

Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects, usually having a common property. If S is a set, and x and y are certain objects, then the notation $x \in S$ means that x is a member of the set S and $y \notin S$ means that y is not an element of S . A set with a denumerable number of elements is specified by braces: $A = \{1, 2, 3, 4\}$ indicates that the elements of set A are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns, to each pair of elements from S , a unique element from S . As an example, consider the relation $a * b = c$. We say that $*$ is a binary operator if it specifies a rule for finding c from the pair (a, b) and also if $a, b, c \in S$. However, $*$ is not a binary operator if $a, b \in S$, and if $c \notin S$.

1. *Closure.* A set S is closed with respect to a binary operator if, for every pair of elements of S , the binary operator specifies a rule for obtaining a unique element of S . For example, the set of natural numbers $N = \{1, 2, 3, 4, \dots\}$ is closed with respect to the binary operator $+$ by the rules of arithmetic addition, since, for any $a, b \in N$, there is a unique $c \in N$ such that $a + b = c$. The set of natural numbers is *not* closed with respect to the binary operator $-$ by the rules of arithmetic subtraction, because $2 - 3 = -1$ and $2, 3 \in N$, but $(-1) \notin N$.

2. *Associative law.* A binary operator $*$ on a set S is said to be associative whenever

$$(x * y) * z = x * (y * z) \text{ for all } x, y, z, \in S$$

3. *Commutative law.* A binary operator $*$ on a set S is said to be commutative whenever

$$x * y = y * x \text{ for all } x, y \in S$$

4. *Identity element.* A set S is said to have an identity element with respect to a binary operation $*$ on S if there exists an element $e \in S$ with the property that

$$e * x = x * e = x \text{ for every } x \in S$$

Example: The element 0 is an identity element with respect to the binary operator $+$ on the set of integers $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$, since

$$x + 0 = 0 + x = x \text{ for any } x \in I$$

5. *Inverse.* A set S having the identity element e with respect to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that

$$x * y = e$$

Example: In the set of integers, I , and the operator $+$, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.

6. *Distributive law.* If $*$ and \cdot are two binary operators on a set S , $*$ is said to be distributive over \cdot whenever

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA

DUALITY

It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. In a two-valued Boolean algebra, the identity elements and the elements of the set B are the same: 1 and 0. The duality principle has many applications. If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

BASIC THEOREM

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

THEOREM 1(a): $x + x = x$.

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

THEOREM 2(a): $x + 1 = 1$.

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

THEOREM 3: $(x')' = x$. From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which together define the complement of x . The complement of x' is x and is also $(x')'$.



THEOREM 6(a): $x + xy = x$.

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

BOOLEAN FUNCTIONS

Boolean algebra is an algebra that deals with binary variables and logic operations. A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols. For a given value of the binary variables, the function can be equal to either 1 or 0.

$$F1 = x + y'z$$

The function F_1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F_1 is equal to 0 otherwise. The complement operation dictates that when $y' = 1$, $y = 0$. Therefore, $F_1 = 1$ if $x = 1$ or if $y = 0$ and $z = 1$.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n , where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Table 2.2
Truth Tables for F_1 and F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

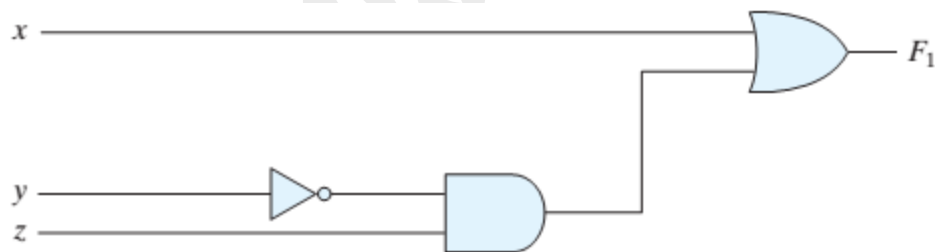


FIGURE 2.1
Gate implementation of $F_1 = x + y'z$

$$F_2 = x'y'z + x'yz + xy'$$

A schematic of an implementation of this function with logic gates is shown in Fig. 2.2(a). Input variables x and y are complemented with inverters to obtain x' and y' . The three terms in the expression are implemented with three AND gates. The OR gate forms the logical OR of the three terms. The truth table for F_2 is listed in Table 2.2. The function is equal to 1 when $xyz = 001$ or 011 or when $xy = 10$ (irrespective of the value of z) and is equal to 0 otherwise. This set of conditions produces four 1's and four 0's for F_2 .

Now consider the possible simplification of the function by applying some of the identities of Boolean algebra:

$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$

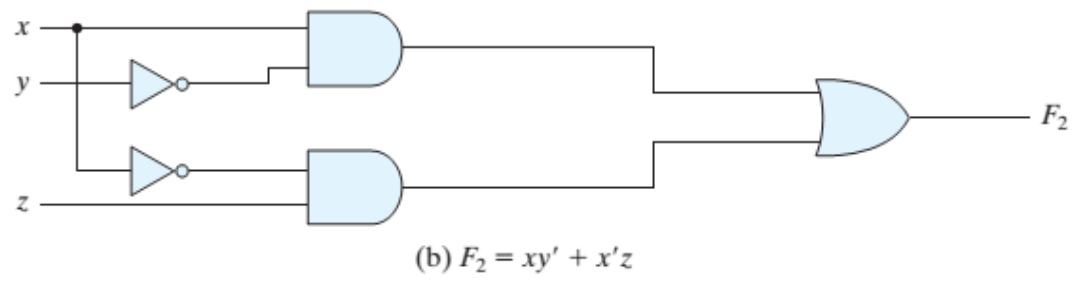
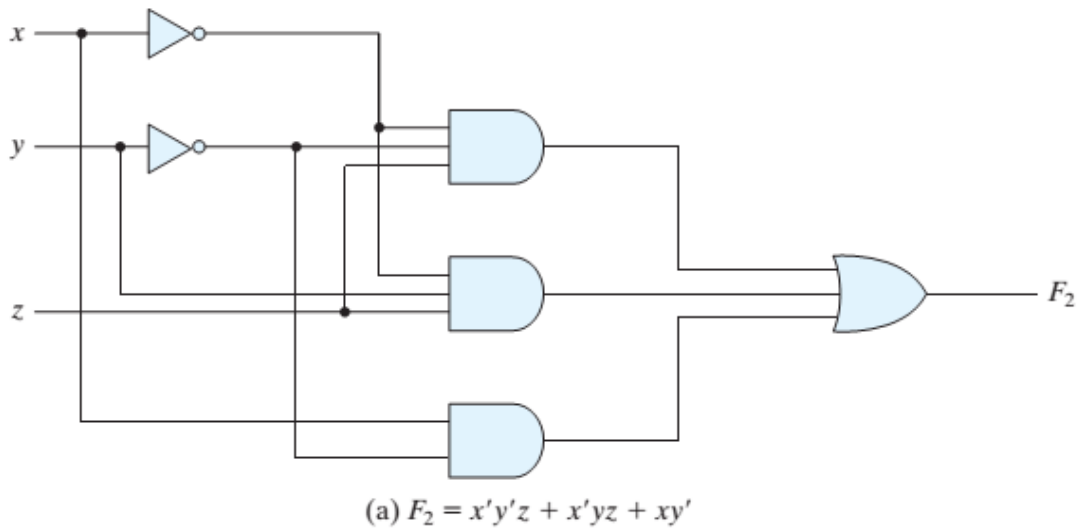


FIGURE 2.2
Implementation of Boolean function F_2 with gates

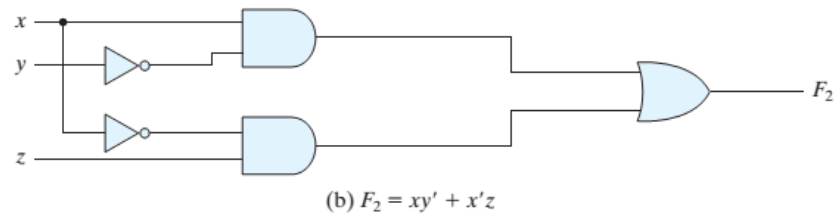
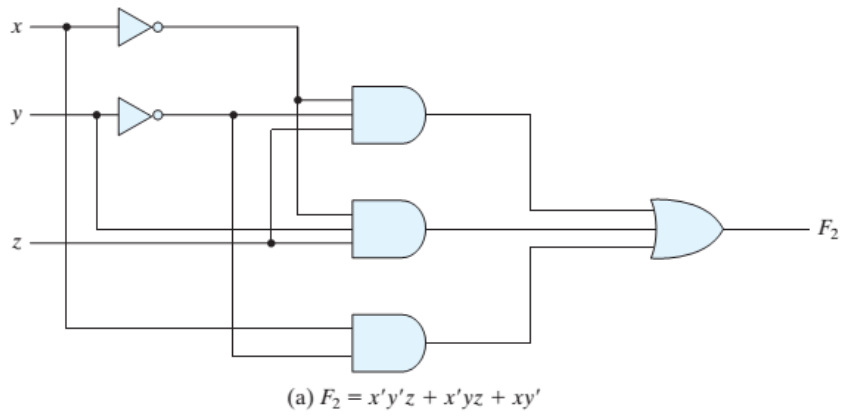


FIGURE 2.2
Implementation of Boolean function F_2 with gates

ALGEBRAIC MANIPULATION

When a Boolean expression is implemented with logic gates, each term requires a gate and each variable within the term designates an input to the gate. We define a literal to be a single variable within a term, in complemented or uncomplemented form. The function of fig. 2.2 (a) has three terms and eight literals, and the one in fig. 2.2 (b) has two terms and four literals. By reducing the number of terms, the number of literals, or both in a boolean expression, it is often possible to obtain a simpler circuit. The manipulation of Boolean algebra consists mostly of reducing an expression for the purpose of obtaining a simpler circuit.

EXAMPLE 2.1

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $xy + x'z + yz = xy + x'z + yz(x + x')$
 $= xy + x'z + xyz + x'yz$
 $= xy(1 + z) + x'z(1 + y)$
 $= xy + x'z.$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z),$ by duality from function 4.

The fourth function illustrates the fact that an increase in the number of literals sometimes leads to a simpler final expression. Function 5 is not minimized directly, but can be derived from the dual of the steps used to derive function 4. Functions 4 and 5 are together known as the consensus theorem.

COMPLEMENT OF A FUNCTION

The complement of a function f is f' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of f . The complement of a function may be derived algebraically through demorgan's theorems, listed in table 2.1 for two variables. Demorgan's theorems can be extended to three or more variables. The three-variable form of the first Demorgan's theorem is derived as follows, from postulates and theorems listed in table 2.1 :

$$\begin{aligned}(A + B + C)' &= (A + x)' && \text{let } B + C = x \\ &= A'x' && \text{by theorem 5(a) (DeMorgan)} \\ &= A'(B + C)' && \text{substitute } B + C = x \\ &= A'(B'C') && \text{by theorem 5(a) (DeMorgan)} \\ &= A'B'C' && \text{by theorem 4(b) (associative)}\end{aligned}$$

EXAMPLE 2.2

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

$$\begin{aligned} F_1' &= (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z') \\ F_2' &= [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)' \\ &= x' + (y + z)(y' + z') \\ &= x' + yz' + y'z \end{aligned}$$

EXAMPLE 2.3

Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

- $F_1 = x'yz' + x'y'z$.
The dual of F_1 is $(x' + y + z')(x' + y' + z)$.
Complement each literal: $(x + y' + z)(x + y + z') = F_1'$.
- $F_2 = x(y'z' + yz)$.
The dual of F_2 is $x + (y' + z')(y + z)$.
Complement each literal: $x' + (y + z)(y' + z') = F_2'$.

CANONICAL AND STANDARD FORMS

MINTERMS AND MAXTERMS

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: $x'y'$, $x'y$, xy' , and xy . Each of these four terms is called a **MINTERM**, or **A STANDARD PRODUCT**. In a similar manner, n variables can be combined to form 2^n minterms. The 2^n different minterms may be determined.

In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called **MAXTERMS**, or **STANDARD SUMS**. The eight maxterms for three variables, together with their symbolic designations.

Table 2.3
Minterms and Maxterms for Three Binary Variables

x	y	z	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

A boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the or of all those terms.

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Similarly, it may be easily verified that

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Now consider the complement of a boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms. The complement of f_1 is read as

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of f_1' , we obtain the function f_1 :

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Similarly, it is possible to read the expression for f_2 from the table:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

SUM OF MINTERMS

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.

EXAMPLE 2.4

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A , B , and C . The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C' + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table. Consider the Boolean function given in Example 2.4:

$$F = A + B'C$$

Table 2.5
Truth Table for $F = A + B'C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

PRODUCT OF MAXTERMS

Each of the 2^{2n} functions of n binary variables can be also expressed as a product of maxterms. To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law, $x+yz=(x+y)(x+z)$. Then any missing variable in each OR term is ORed with xx' .

EXAMPLE 2.5

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x , y , and z . Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

CONVERSION BETWEEN CANONICAL FORMS

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function.

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m_j' = M_j$$

That is, the **maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa.**

A Boolean function can be converted from an algebraic expression to a product of maxterms by means of a truth table and the canonical conversion procedure. Consider, for example, the Boolean expression

$$F = xy + x'z$$

First, we derive the truth table of the function, as shown in Table 2.6. The 1's under F in the table are determined from the combination of the variables for which $xy=11$ or $xz=01$. The minterms of the function are read from the truth table to be 1, 3, 6, and 7. The function expressed as a sum of minterms is

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

Since there is a total of eight minterms or maxterms in a function of three variables, we determine the missing terms to be 0, 2, 4, and 5. The function expressed as a product of maxterms is

Table 2.6
Truth Table for $F = xy + x'z$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

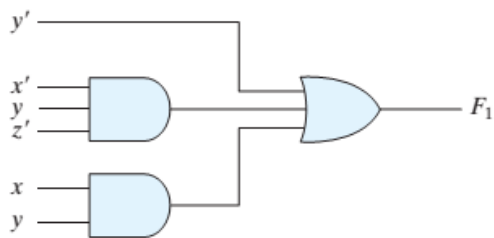
$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

STANDARD FORMS

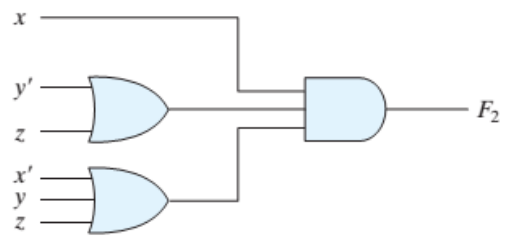
There are two types of standard forms: the sum of products and products of sums. The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$$F1 = Y' + XY + X'YZ'$$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.



(a) Sum of Products



(b) Product of Sums

A product of sums is a boolean expression containing or terms, called sum terms. Each term may have any number of literals. The product denotes the AND ing of these Terms. An example of a function expressed as a product of sums is

$$F_2 = X(Y' + Z)(X' + Y + Z')$$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation. This standard type of expression results in a two-level structure of gates.

A boolean function may be expressed in a **NONSTANDARD FORM**. For example, the function

$$F_3 = AB + C(D + E)$$

It can be changed to a standard form by using the distributive law to remove the parentheses:

$$F_3 = AB + C(D + E) = AB + CD + CE$$









OTHER LOGIC OPERATIONS

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x , but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y , but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y , but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y , then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x , then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

DIGITAL LOGIC GATES

Since Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates. Still, the possibility of constructing gates for the other logic operations is of practical interest.

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table border="1"> <thead> <tr> <th>x</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table border="1"> <thead> <tr> <th>x</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

POSITIVE AND NEGATIVE LOGIC

The binary signal at the inputs and outputs of any gate has one of two values, except during transition. One signal value represents logic 1 and the other logic 0. Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value, as shown in Fig. The higher signal level is designated by *H* and the lower signal level by *L*. Choosing the high-level *H* to represent logic 1 defines a positive logic system. Choosing the low-level *L* to represent logic 1 defines a negative logic system.

