INTRODUCTION TO NUMBER SYSTEMS

BINARY NUMBERS

A decimal number such as 7392 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. To be more exact, 7392 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 coefficients and from their position deduce the necessary powers of 10. In general, a number with a decimal point is represented by a series of coefficients as follows:

$$a_5a_4a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3}$$

The a_j coefficients are one of the ten 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.

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

The decimal number system is said to be of *base*, or *radix*, 10 because it uses ten digits and the coefficients are multiplied by powers of 10. The *binary* system is a different number system. The coefficients of the binary numbers system have two possible values: 0 and 1. Each coefficient a_j is multiplied by 2^j. For example, the decimal equivalent of the binary number 11010.11 is 26.75, as shown from the multiplication of the coefficients by powers of 2:

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

In general, a number expressed in base-r system has coefficients multiplied by powers of r:

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

The coefficients a_j range in value from 0 to $r - 1$. To distinguish between numbers of different bases, we enclose the coefficients in parentheses and write a subscript equal to the base used (except sometimes for decimal numbers, where the content makes it obvious that it is decimal). An example of a base-5 number is

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

Note that coefficient values for base 5 can be only 0, 1, 2, 3, and 4.

It is customary to borrow the needed r digits for the coefficients from the decimal system when the base of the number is less than 10. The letters of the alphabet are used to supplement the ten decimal digits when the base of the number is greater than 10. For example, in the *hexadecimal* (base 16) number system, the first ten digits are borrowed from the decimal system. The letters A, B, C, D, E, and F are used for digits 10, 11, 12, 13, 14, and 15, respectively. An example of a hexadecimal number is

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16 + 15 = (46687)_{10}$$

The first 16 numbers in the decimal, binary, octal, and hexadecimal systems are listed in Table 1-1.

TABLE 1-1
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

Arithmetic operations with numbers in base *r* follow the same rules as for decimal numbers. When other than the familiar base 10 is used, one must be careful to use only the *r* allowable digits. Examples of addition, subtraction, and multiplication of two binary numbers are as follows:

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	+100111	subtrahend:	−100111	multiplier:	× 101
sum:	1010100	difference:	000110		1011
					0000
					1011
				product:	110111

The sum of two binary numbers is calculated by the same rules as in decimal, except that the digits of the sum in any significant position can be only 0 or 1. Any carry obtained in a given significant position is used by the pair of digits one significant position higher. The subtraction is slightly more complicated. The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.) Multiplication is very simple. The multiplier digits are always 1 or 0. Therefore, the partial products are equal either to the multiplicand or to 0.

NUMBER BASE CONVERSIONS

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

$$(1010.011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10.375)_{10}$$

The binary number has four 1's and the decimal equivalent is found from the sum of four powers of 2. Similarly, a number expressed in base r can be converted to its decimal equivalent by multiplying each coefficient with the corresponding power of r and adding. The following is an example of octal-to-decimal conversion:

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

The conversion from decimal to binary or to any other base-r system is more convenient if the number is separated into an *integer part* and a *fraction part* and the conversion of each part done separately. The conversion of an *integer* from decimal to binary is best explained by example.

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. The quotient is again divided by 2 to give a new quotient and remainder. This process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer quotient		Remainder	Coefficient
$\frac{41}{2} =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$\frac{20}{2} =$	10	+	0	$a_1 = 0$
$\frac{10}{2} =$	5	+	0	$a_2 = 0$
$\frac{5}{2} =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$\frac{2}{2} =$	1	+	0	$a_4 = 0$
$\frac{1}{2} =$	0	+	$\frac{1}{2}$	$a_5 = 1$

answer: $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$

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

$\uparrow$ 101001 = answer ■

The conversion from decimal integers to any base-*r* system is similar to the example, except that division is done by *r* instead of 2.

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:

$$\begin{array}{r|l} 153 & \\ 19 & 1 \\ 2 & 3 \\ 0 & 2 \;\; \uparrow\!\rule{1em}{0.4pt} \quad = (231)_8 \end{array} \qquad \blacksquare$$

Which implies that:

$$(153)_{10} = (231)_8$$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers are accumulated instead of remainders. Again, the method is best explained by example.

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. The new fraction is multiplied by 2 to give a new integer and a new fraction. This process is continued until the fraction becomes 0 or until the number of digits have 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	$a_{-4} = 1$

Answer: $(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$ ■

To convert a decimal fraction to a number expressed in base r, a similar procedure is used. Multiplication is by r instead of 2, and the coefficients found from the integers may range in value from 0 to $r - 1$ instead of 0 and 1.

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:

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

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and fraction separately and then combining the two answers. Using the results of Examples 1-1 and 1-3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$
$$(153.513)_{10} = (231.406517)_8$$

OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important part in digital computers. 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:

$$(\,\underbrace{10}_{2}\;\;\underbrace{110}_{6}\;\;\underbrace{001}_{1}\;\;\underbrace{101}_{5}\;\;\underbrace{011}_{3}\;\;.\;\;\underbrace{111}_{7}\;\;\underbrace{100}_{4}\;\;\underbrace{000}_{0}\;\;\underbrace{110}_{6}\,)_2 = (26153.7460)_8$$

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

$$(\,\underbrace{10}_{2}\;\;\underbrace{1100}_{C}\;\;\underbrace{0110}_{6}\;\;\underbrace{1011}_{B}\;\;.\;\;\underbrace{1111}_{F}\;\;\underbrace{0010}_{2}\,)_2 = (2C6B.F2)_{16}$$

The corresponding hexadecimal (or octal) digit for each group of binary digits is easily remembered after studying the values listed in Table 1-1.

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

$$(673.124)_8 = (\ \underbrace{110}_{6}\ \ \underbrace{111}_{7}\ \ \underbrace{011}_{3}\ \ .\ \ \underbrace{001}_{1}\ \ \underbrace{010}_{2}\ \ \underbrace{100}_{4}\ )_2$$

$$(306.\mathrm{D})_{16} = (\ \underbrace{0011}_{3}\ \ \underbrace{0000}_{0}\ \ \underbrace{0110}_{6}\ \ .\ \ \underbrace{1101}_{\mathrm{D}}\ )_2$$

Binary numbers are difficult to work with because they require three or four times as many digits as their decimal equivalent. For example, the binary number 111111111111 is equivalent to decimal 4095. However, digital computers use binary numbers and it is sometimes necessary for the human operator or user to communicate directly with the machine by means of binary numbers. One scheme that retains the binary system in the computer but reduces the number of digits the human must consider utilizes the relationship between the binary number system and the octal or hexadecimal system. By this method, the human thinks in terms of octal or hexadecimal numbers and performs the required conversion by inspection when direct communication with the machine is necessary. Thus the binary number 111111111111 has 12 digits and is expressed in octal as 7777 (four digits) or in hexadecimal as FFF (three 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. When the human communicates with the machine (through console switches or indicator lights or by means of programs written in *machine language*), the conversion from octal or hexadecimal to binary and vice versa is done by inspection by the human user.

COMPLEMENTS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. 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

Given a number N in base r having n digits, the $(r - 1)$'s complement of N is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$. Now, 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. Some numerical examples follow.

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. 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

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and 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 $r^n - N = [(r^n - 1) - N] + 1$. 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^n 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.

The 10's complement of 012398 is 987602.

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.

The 2's complement of 1101100 is 0010100.

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.

In the previous definitions, it was assumed that the numbers do not have a radix point. If the original number N contains a radix point, the point should be removed temporarily in order to form the r's or $(r - 1)$'s complement. The radix point is then restored to the complemented number in the same relative position. It is also worth mentioning that the complement of the complement restores the number to its original value. The r's complement of N is $r^n - N$. The complement of the complement is $r^n - (r^n - N) = N$, giving back the original number.

Subtraction with Complements

The direct method of subtraction taught in elementary schools uses the borrow concept. In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit. This seems to be easiest when people perform subtraction with paper and pencil. When subtraction is implemented with digital hardware, this method is found to be 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. This performs $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry, r^n, which is 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.

The following examples illustrate the procedure.

The subtraction of two n-digit unsigned numbers (M – N) in base (r-1) can be done as follows:

1. Add the minuend M to the (r – 1)'s complement of the subtrahend N.
2. Inspect the result obtained in step 1 for an end carry.
   a) If an end carry occurs, add 1 to the least significant digit (end – around carry).
   b) If an end around carry does not occur, take the (r – 1)'s complement of the number obtained in step 1 and place a negative sign in the front.

Example 1-5 Using 10's complement, subtract 72532 − 3250.

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

■

Note that M has 5 digits and N has only 4 digits. Both numbers must have the same number of digits; so we can 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 the result is positive.

Example 1-6 Using 10's complement, subtract 3250 − 72532.

$$
\begin{aligned}
M &= 03250 \\
\text{10's complement of } N &= +\ \underline{27468} \\
\text{Sum} &= 30718
\end{aligned}
$$

There is no end carry.

Answer: −(10's complement of 30718) = −69282 ■

Note that since 3250 < 72532, the result is negative. Since we are dealing with unsigned numbers, there is really no way to get an unsigned result for this case. When subtracting with complements, the negative answer is recognized from the absence of the end carry and the complemented result. When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined before.

Example 1-7 Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ using 2's complements.

(a)

$$\begin{array}{rr}
X = & 1010100 \\
\text{2's complement of } Y = & +\ \underline{0111101} \\
\text{Sum} = & 10010001 \\
\text{Discard end carry } 2^7 = & -\underline{10000000} \\
\text{Answer: } X - Y = & 0010001
\end{array}$$

(b)

$$\begin{array}{rr}
Y = & 1000011 \\
\text{2's complement of } X = & +\ \underline{0101100} \\
\text{Sum} = & 1101111
\end{array}$$

There is no end carry.

$$\text{Answer: } Y - X = -(\text{2's complement of } 1101111) = -0010001$$ ■

Subtraction of unsigned numbers can be done also 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 1 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 using 1's complement.

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

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

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

$$\begin{array}{rr}
Y = & 1000011 \\
\text{1's complement of } X = & +\;\underline{0101011} \\
\text{Sum} = & 1101110
\end{array}$$

There is no end carry.

Answer: $Y - X = -(\text{1's complement of } 1101110) = -0010001$

Note that the negative result is obtained by taking the 1's complement of the sum since this is the type of complement used. The procedure with end-around carry is also applicable for subtracting unsigned decimal numbers with 9's complement.

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 having a common property. If S is a set, and x and y are certain objects, then $x \in S$ denotes that x is a member of the set S, and $y \notin S$ denotes that y is not an element of S. A set with a denumerable number of elements is specified by braces: $A = \{1, 2, 3, 4\}$, i.e., 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 $a, b \in S$, whereas the rule finds $c \notin S$.

The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems, and properties of the system. The most common postulates used to formulate various algebraic structures are:

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

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

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

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

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

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

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

Example: The element 0 is an identity element with respect to operation + on the set of integers I = {. . . , −3, −2, −1, 0, 1, 2, 3, . . .} since

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

The set of natural numbers N has no identity element since 0 is excluded from the set.

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 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

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

Boolean algebra is an algebraic structure defined on a set of elements B together with two binary operators $+$ and $\cdot$ provided the following (Huntington) postulates are satisfied:

1. **(a)** Closure with respect to the operator $+$.
   (b) Closure with respect to the operator $\cdot$.
2. **(a)** An identity element with respect to $+$, designated by 0: $x + 0 = 0 + x = x$.
   (b) An identity element with respect to $\cdot$, designated by 1: $x \cdot 1 = 1 \cdot x = x$.
3. **(a)** Commutative with respect to $+$: $x + y = y + x$.
   (b) Commutative with respect to $\cdot$: $x \cdot y = y \cdot x$.
4. **(a)** $\cdot$ is distributive over $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (b) $+$ is distributive over $\cdot$: $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
6. There exists at least two elements $x, y \in B$ such that $x \neq y$.

Two-Valued Boolean Algebra

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rules for the two binary operators $+$ and $\cdot$ as shown in the following operator tables (the rule for the complement operator is for verification of postulate 5):

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

These rules are exactly the same as the AND, OR, and NOT operations, respectively, defined in Table 1-6. We must now show that the Huntington postulates are valid for the set $B = \{0, 1\}$ and the two binary operators defined before.

1. *Closure* is obvious from the tables since the result of each operation is either 1 or 0 and $1, 0 \in B$.
2. From the tables we see that
   (a) $0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$
   (b) $1 \cdot 1 = 1 \qquad 1 \cdot 0 = 0 \cdot 1 = 0$
   which establishes the two *identity elements* 0 for + and 1 for · as defined by postulate 2.

3. The *commutative* laws are obvious from the symmetry of the binary operator tables.
4. **(a)** The *distributive* law $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ can be shown to hold true from the operator tables by forming a truth table of all possible values of x, y, and z. For each combination, we derive $x \cdot (y + z)$ and show that the value is the same as $(x \cdot y) + (x \cdot z)$.

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

(b) The *distributive* law of + over · can be shown to hold true by means of a truth table similar to the one above.

5. From the complement table it is easily shown that
(a) $x + x' = 1$, since $0 + 0' = 0 + 1 = 1$ and $1 + 1' = 1 + 0 = 1$.
(b) $x \cdot x' = 0$, since $0 \cdot 0' = 0 \cdot 1 = 0$ and $1 \cdot 1' = 1 \cdot 0 = 0$, which verifies postulate 5.

6. Postulate 6 is satisfied because the two-valued Boolean algebra has two distinct elements, 1 and 0, with $1 \neq 0$.

2-3 BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA

Duality

The Huntington postulates have been listed in pairs and designated by part (a) and part (b). One part may be obtained from the other if the binary operators and the identity elements are interchanged. This important property of Boolean algebra is called the *duality principle*. 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 Theorems

Table 2-1 lists six theorems of Boolean algebra and four of its postulates. The notation is simplified by omitting the $\cdot$ whenever this does not lead to confusion. The theorems and postulates listed are the most basic relationships in Boolean algebra. The reader is advised to become familiar with them as soon as possible. The theorems, like the postulates, are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. The proofs of the theorems with one variable are presented below. At the right is listed the number of the postulate that justifies each step of the proof.

TABLE 2-1
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): $x + x = x$.

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

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

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

Note that theorem 1(b) is the dual of theorem 1(a) and that each step of the proof in part (b) is the dual of part (a). Any dual theorem can be similarly derived from the proof of its corresponding pair.

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

$x + 1 = 1 \cdot (x + 1)$	by 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 defines the complement of x. The complement of x' is x and is also $(x')'$. Therefore, since the complement is unique, we have that $(x')' = x$.

The theorems involving two or three variables may be proven algebraically from the postulates and the theorems that have already been proven. Take, for example, the absorption theorem.

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

$$\begin{array}{rlll} x + xy & = x \cdot 1 + xy & \text{by postulate:} & 2(b) \\ & = x(1 + y) & & 4(a) \\ & = x(y + 1) & & 3(a) \\ & = x \cdot 1 & & 2(a) \\ & = x & & 2(b) \end{array}$$

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

The theorems of Boolean algebra can be shown to hold true by means of truth tables. In truth tables, both sides of the relation are checked to yield identical results for all possible combinations of variables involved. The following truth table verifies the first absorption theorem.

		=	
x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

The algebraic proofs of the associative law and DeMorgan's theorem are long and will not be shown here. However, their validity is easily shown with truth tables. For example, the truth table for the first DeMorgan's theorem $(x + y)' = x'y'$ is shown below.

x	y	$x + y$	$(x + y)'$	x'	y'	$x'y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

2-4 BOOLEAN FUNCTIONS

A binary variable can take the value of 0 or 1. A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, and unary operator NOT, parentheses, and an equal sign. For a given value of the variables, the function can be either 0 or 1. Consider, for example, the Boolean function

$$F_1 = xyz'$$

The function F_1 is equal to 1 if $x = 1$ *and* $y = 1$ *and* $z' = 1$; otherwise $F_1 = 0$. The above is an example of a Boolean function represented as an algebraic expression. A Boolean function may also be represented in a truth table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the n binary variables, and a column showing the combinations for which the function is equal to 1 or 0. As shown in Table 2-2, there are eight possible distinct combinations for assigning bits to three variables. The column labeled F_1 contains either a 0 or a 1 for each of these combinations. The table shows that the function F_1 is equal to 1 only when $x = 1$, $y = 1$, and $z = 0$. It is equal to 0 otherwise. (Note that the statement $z' = 1$ is equivalent to saying that $z = 0$.) Consider now the function

TABLE 2-2
Truth Tables for $F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$, and $F_4 = xy' + x'z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

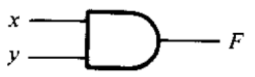
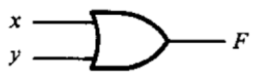
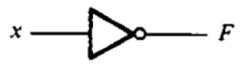
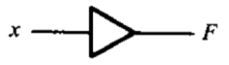
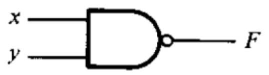
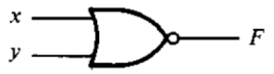
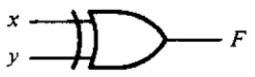
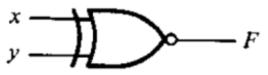
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= x \odot y$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

FIGURE 2-5
Digital logic gates

1.5 Definition of Boolean Algebra

Boolean algebra provides the necessary tools to calculate and interpret information presented in binary form. Boolean algebra is an *algebraic system* (a set of elements to which a set of operations is associated), defined by:

- The set of values $\{0,1\}$;
- The operations *OR*, *AND*, and *NOT*;
- The equivalence operator "=", along with the properties reflexive, symmetric, and transitive.

The three operations are defined as follows:

Operation:	OR (logical sum)	AND (logical product)	NOT (negation)
Algebraic symbols:	$X + Y$	$X \cdot Y = XY$	$\overline{X}$
	$X \vee Y$	$X \wedge Y$	$!X$
	$X \cup Y$	$X \cap Y$	$-X$
	X or Y	X and Y	not(X)

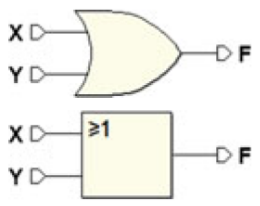
Truth table:

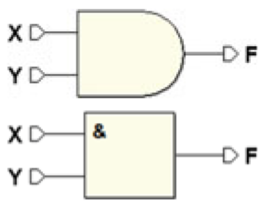
X	Y	$X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

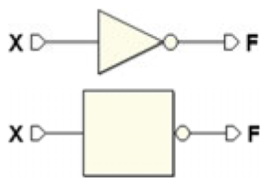
X	Y	$X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

X	$\overline{X}$
0	1
1	0

Circuit diagram symbols:







1.6 The Fundamental Properties of Boolean Algebra

Conventions

- $X, Y, Z, X_1, X_2, X_3, \ldots, X_n$, are considered Boolean variables.
- The parentheses establish the calculation priorities as in regular algebra.
- AND is prioritized over OR (e.g., $X + YZ = X + (YZ)$).

This is also analogous to regular algebra. All the properties can be demonstrated through Perfect Induction, that is, by verifying the validity of each combination of values assumed by the variables that make up the expression.

Example: $X \cdot 0 = 0$ is verified through the truth table:

X	0	$X \cdot 0$
0	0	0
1	0	0

Duality Principle

If a given expression is valid, its dual expression is also valid. The dual expression is obtained by switching the OR with the AND and the 0 constants with the 1 constants from the original expression. For example:

$$\begin{array}{rl} & X + 1 = 1 \\ \text{(dual:)} & X \cdot 0 = 0 \end{array}$$

$$\begin{array}{rl} & X + 0 = X \\ \text{(dual:)} & X \cdot 1 = X \end{array}$$

Idempotent Law

$$\begin{array}{rl} & X + X = X \\ \text{(dual:)} & X \cdot X = X \end{array}$$

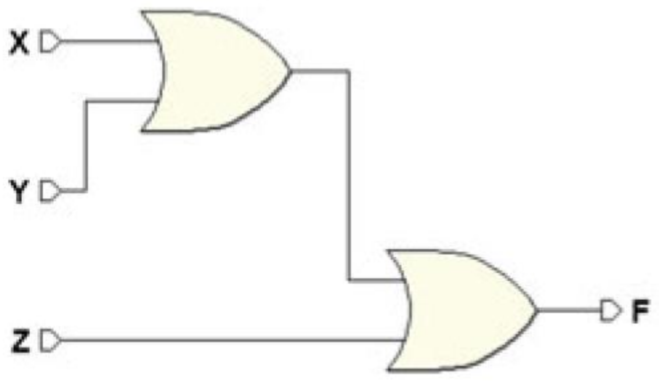
Commutative Law

$$\begin{array}{rl} & X + Y = Y + X \\ \text{(dual:)} & X \cdot Y = Y \cdot X \end{array}$$

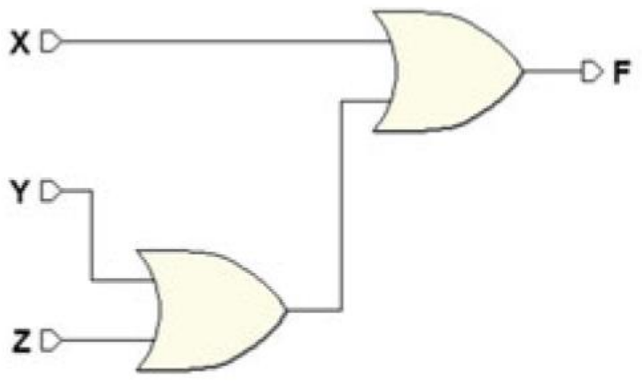
Associative Law

$$\begin{array}{rl} & (X + Y) + Z = X + (Y + Z) = X + Y + Z \\ \text{(dual:)} & (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z. \end{array}$$

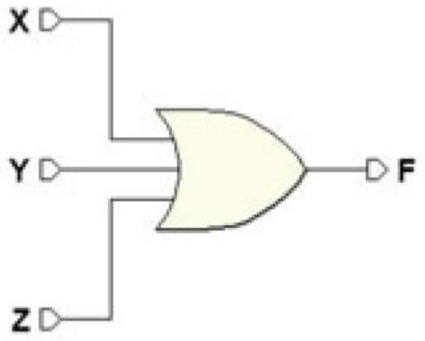
The associative law makes it possible to extend fundamental operations to more than two variables. The circuit symbols for the first expression are:



$(X+Y)+Z$



$X+(Y+Z)$



$X+Y+Z$

As there is no distinction between the first and second circuits, it makes sense to generally define an OR of three or more inputs. The same holds true for the AND, so it is really the property of Associativity that allows us to make sense of OR and AND gates with more than two inputs.

We can redefine the OR and AND operations with n inputs:

- An OR with n inputs gives a 0 as output only if all the n inputs are 0, otherwise it gives a 1 as output.
- An AND with n inputs gives a 1 as output only if all the n inputs are 1, otherwise it gives a 0 as output.

Distributivity

Factoring law	$(X+Y)\cdot(X+Z) = X+(Y\cdot Z)$
Distributive law (dual:)	$(X\cdot Y)+(X\cdot Z) = X\cdot(Y+Z)$

Proof of the factoring law:

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

It would also be possible to demonstrate this law through Perfect Induction (i.e., verifying all the possible combinations for X, Y, Z):

X	Y	Z	$Y\cdot Z$	$X+Y\cdot Z$	$X+Y$	$X+Z$	$(X+Y)(X+Z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

It is clear that columns $X + Y \cdot Z$ and $(X + Y)(X + Z)$ are equal.

Complementation

$$\begin{aligned} & X + \overline{X} = 1 \\ \text{(dual:)} \quad & X \cdot \overline{X} = 0 \end{aligned}$$

Absorption

First form:

$$\begin{aligned} & X + X \cdot Y = X \\ \text{(dual:)} \quad & X \cdot (X + Y) = X \end{aligned}$$

Second form:

$$\begin{aligned} & X + (\overline{X} \cdot Y) = X + Y \\ \text{(dual:)} \quad & X \cdot (\overline{X} + Y) = X \cdot Y \end{aligned}$$

Proof:

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

Logic Adjacency

$$Y\,X + Y\,\overline{X} = Y$$
$$\text{(dual:)}\quad (Y + X)\cdot(Y + \overline{X}) = Y$$

Proof:

$$Y\,X + Y\,\overline{X} = Y\cdot(X + \overline{X}) = Y\cdot 1 = Y$$
$$(Y + X)\cdot(Y + \overline{X}) = Y + (X\cdot\overline{X}) = Y + 0 = Y$$

Consensus

$$X\cdot Y + Y\cdot Z + Z\cdot\overline{X} = X\cdot Y + Z\cdot\overline{X}$$
$$\text{(dual:)}\quad (X + Y)(Y + Z)(Z + \overline{X}) = (X + Y)(Z + \overline{X})$$

Proof:

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

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

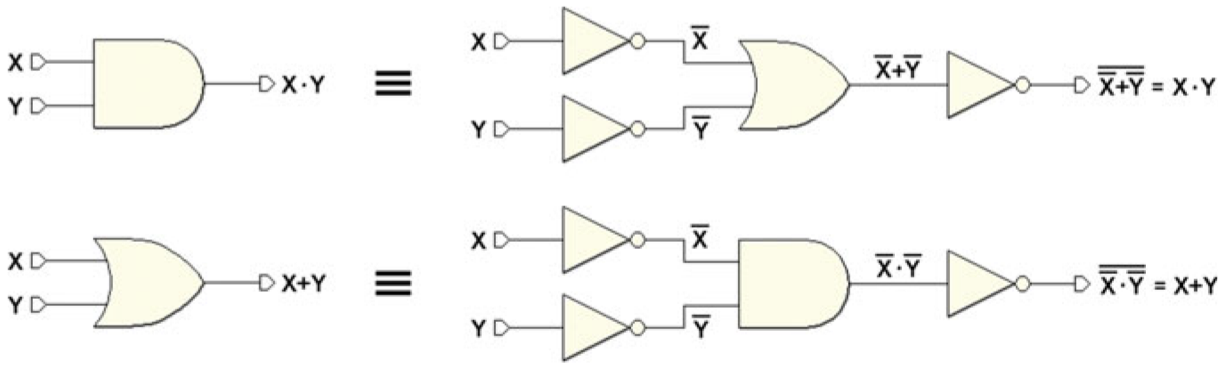
Involution

Also known as *Double Complement law*: $\overline{\overline{X}} = X$.

Duality or De Morgan's Theorem

A logical product of two variables can be substituted by the negation of their logical sum. Dual: a logical sum of two variables can be substituted by the negation of their logical product:

$$
\begin{aligned}
X \cdot Y &= \overline{\overline{X} + \overline{Y}} \\
\text{(dual:)} \quad X + Y &= \overline{\overline{X} \cdot \overline{Y}}
\end{aligned}
$$



This theorem is important: it allows us to obtain an AND through an OR gate and vice versa. The theorem tells us that either one of the two functions is superfluous according to the definition of Boolean algebra.

Generalized De Morgan's Theorem

The theorem applies to any number of variables:

$$
\begin{aligned}
X_1 \cdot X_2 \cdot \ldots \cdot X_n &= \overline{\overline{X_1} + \overline{X_2} + \ldots + \overline{X_n}} \\
\text{(dual :)} \quad X_1 + X_2 + \ldots + X_n &= \overline{\overline{X_1} \cdot \overline{X_2} \cdot \ldots \cdot \overline{X_n}}.
\end{aligned}
$$

1.7 Other Operations

In this paragraph, we define other operations in Boolean algebra: NAND, NOR, and EXOR.

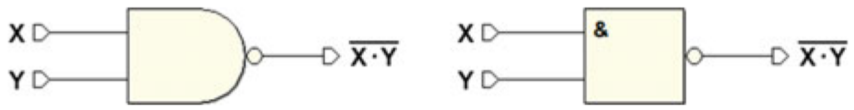
NAND

The NAND operation is equivalent to an AND whose output is negated:

$$X \text{ nand } Y = \overline{(X \cdot Y)}$$

X	Y	$(X \text{ nand } Y)$
0	0	1
0	1	1
1	0	1
1	1	0

Circuital symbols:



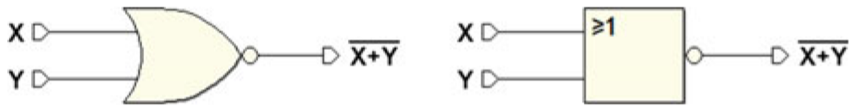
NOR

The NOR operation is equivalent to an OR whose output is negated:

$$X \text{ nor } Y = \overline{(X + Y)}$$

X	Y	$(X \text{ nor } Y)$
0	0	1
0	1	0
1	0	0
1	1	0

Circuital symbols:



NAND and NOR are *commutative* but <u>not</u> *associative*.

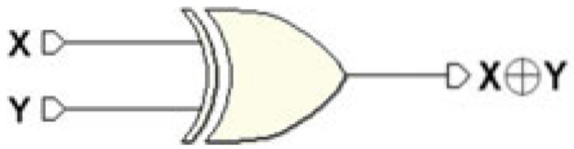
XOR (Exclusive OR)

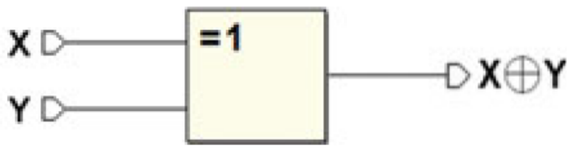
The XOR operation is said “anticoincidence” (it provides 1 when the inputs are different):

$$X \oplus Y = X \text{ xor } Y = X\,\overline{Y} + \overline{X}\,Y$$

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Circuital symbols:





1.8 Functionally Complete Operation Sets

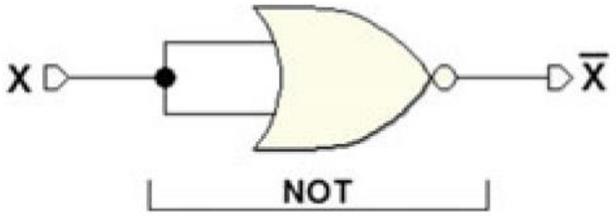
We have seen that Boolean algebra is based on a set of two elements {0, 1} and a set of operations: OR, AND, NOT. Also, De Morgan's Theorem shows that one of the two AND or OR operations can be considered superfluous and the sets of {OR, NOT} or {AND, NOT} are a sufficient basis to construct all of Boolean algebra. Let's broaden the subject by discussing other sets of operations that allows to construct Boolean algebra (named, for this reason, *Functionally Complete Operation Sets*):

1. {AND, OR, NOT}
2. {NOR}
3. {NAND}
4. {OR, NOT}
5. {AND, NOT}
6. {EXOR, AND}
6. {EXOR, OR}

Note: in practice, only {NOR} and {NAND} sets are used.

{NOR} Set

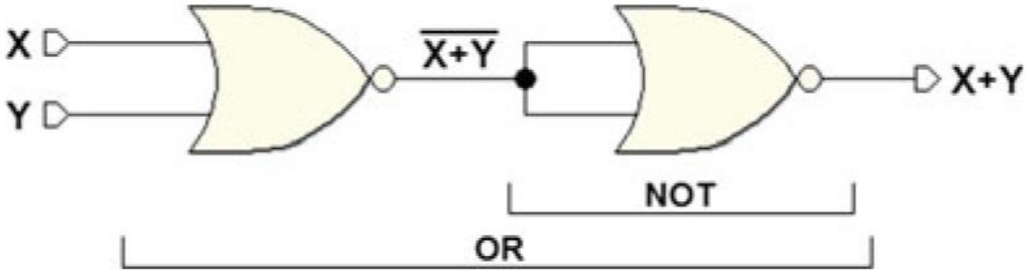
We can obtain OR and NOT from NOR gates. If we connect a NOR as in the figure below, we obtain a NOT. Given that the X and Y inputs are connected together, we obtain the following from the NOR table:



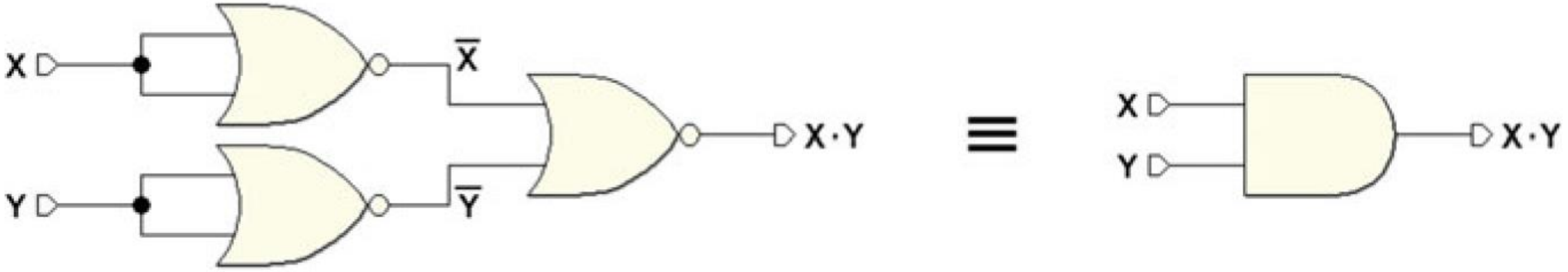
In fact:

X	Y	X nor Y
0	0	1
1	1	0

However, we obtain the OR gate by negating the NOR output with a NOT:

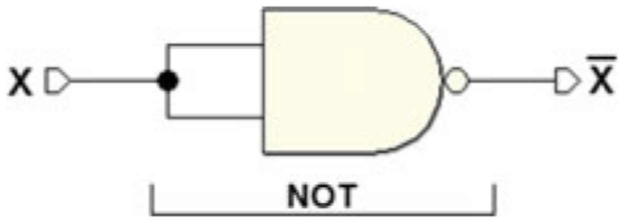


To obtain the AND, we apply De Morgan: $X \cdot Y = \overline{\overline{X} + \overline{Y}}$. We have:



{NAND} Set

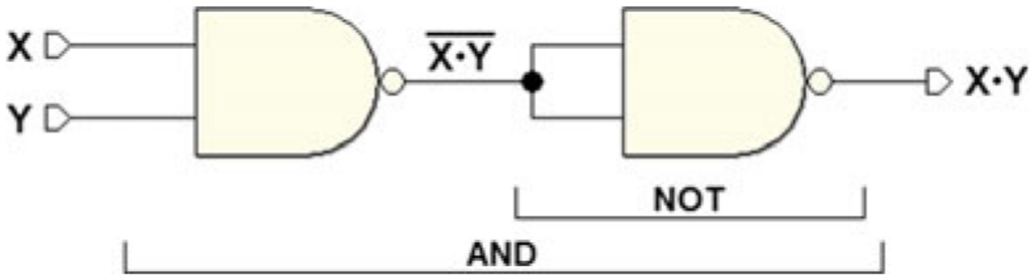
Similar to the above, the NOT is obtained as follows, taking into account the two lines of the NAND table where the two X and Y inputs are equal:



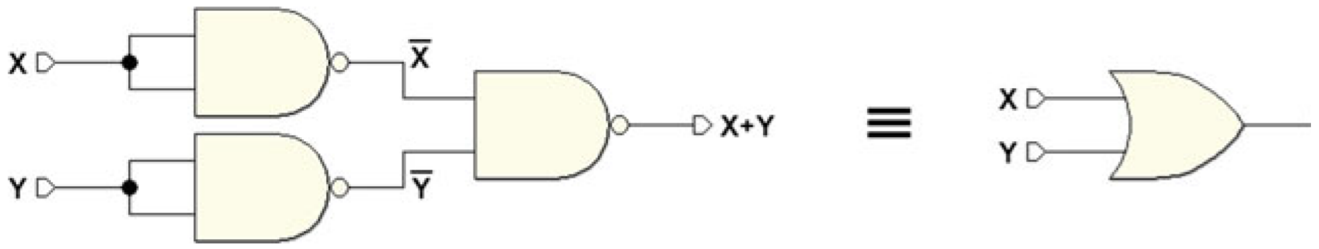
In fact:

X	Y	X nand Y
0	0	1
1	1	0

Therefore, to obtain the AND, it is sufficient to connect the NAND to a NOT made with a NAND.

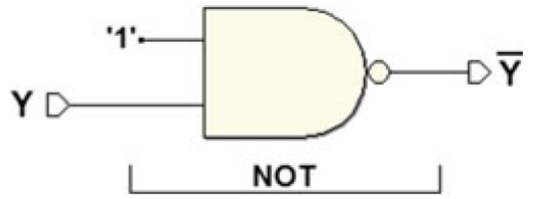


Finally, by De Morgan, we obtain the OR:



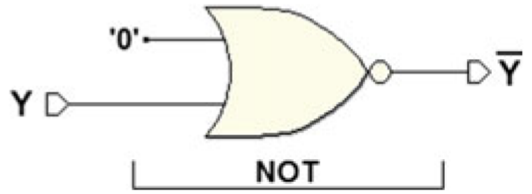
Take note: there is another way to obtain the NOT by the NAND. By conne
one of the inputs to the constant $X = 1$:

X	Y	X nand Y
0	0	1
1	1	0

we obtain: 

Similarly, if we posit $X = 0$ for the NOR we get:

X	Y	X nor Y
0	0	1
0	1	0

we obtain: 

{OR, NOT} Set

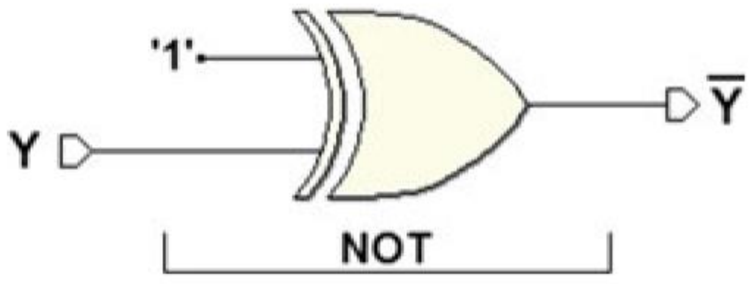
The AND is obtained by De Morgan's Theorem.

{AND, NOT} Set

The OR is obtained by De Morgan's Theorem.

{XOR, AND} Set

The NOT is obtained by the XOR as follows:



From the XOR truth table, we get:

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

positing $X = 1$:

X	Y	$X \oplus Y$
1	0	1
1	1	0

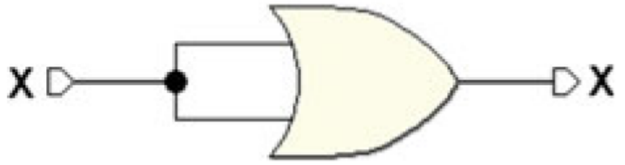
Note: If we change the constant $X = 1$ in 0, we get the identity. Therefore, we obtain an inverting/identity function, "programmable" by the input X.

{XOR, OR} Set

The NOT is obtained through the XOR and the AND by using De Morgan's Theorem.

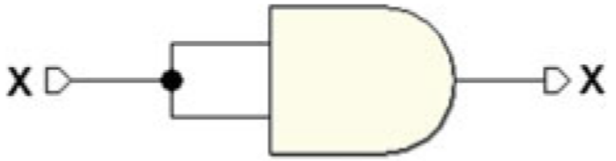
Identity

Identity can be obtained in the following ways:

from **OR**: 

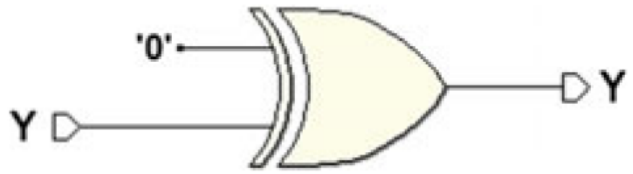
In fact:

X	Y	$X + Y$
0	0	0
1	1	1

from **AND**: 

In fact:

X	Y	$X \cdot Y$
0	0	0
1	1	1

from **XOR**: 

In fact:

X	Y	$X \oplus Y$
0	0	0
0	1	1

DECODERS AND ENCODERS

Discrete quantities of information are represented in digital systems with binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information. A *decoder* is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit decoded information has unused or don't-care combinations, the decoder output will have fewer than 2^n outputs.

The decoders presented here are called n-to-m-line decoders, where $m \leq 2^n$. Their purpose is to generate the 2^n (or fewer) minterms of n input variables. The name *decoder* is also used in conjunction with some code converters such as a BCD-to-seven-segment decoder.

As an example, consider the 3-to-8-line decoder circuit of Fig. 5-8. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. A particular application of this decoder would be a binary-to-octal conversion. The input variables may represent a binary number, and the outputs will then represent the eight digits in the octal number

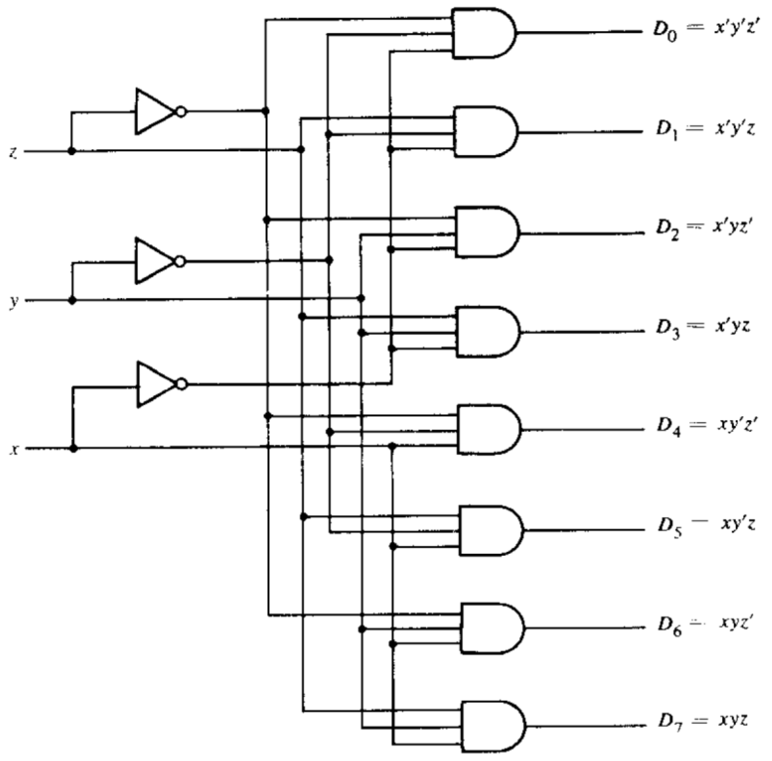


FIGURE 5-8
A 3-to-8 line decoder

TABLE 5-2
Truth Table of a 3-to-8-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder whose truth table is given in Table 5-3. It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time; otherwise the circuit has no meaning.

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following output Boolean functions:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

TABLE 5-3
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

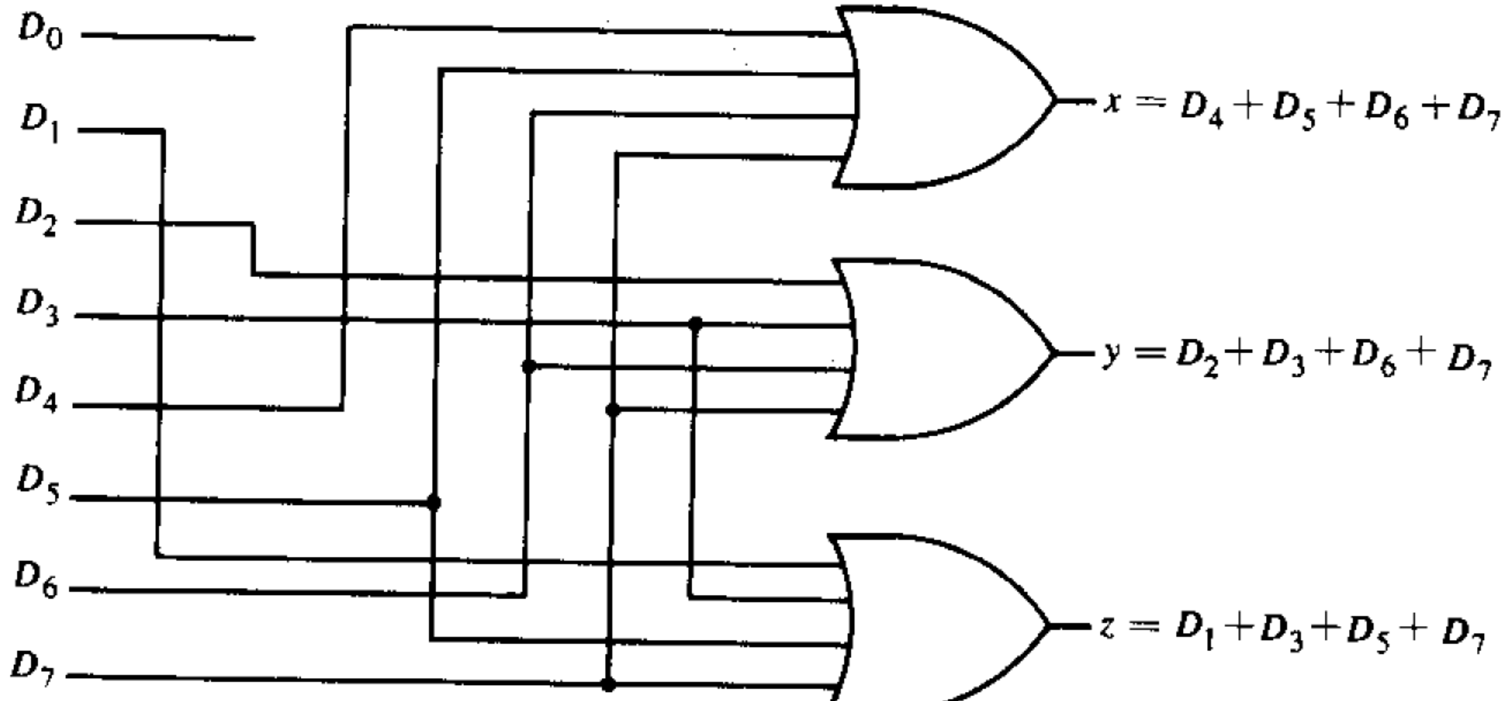
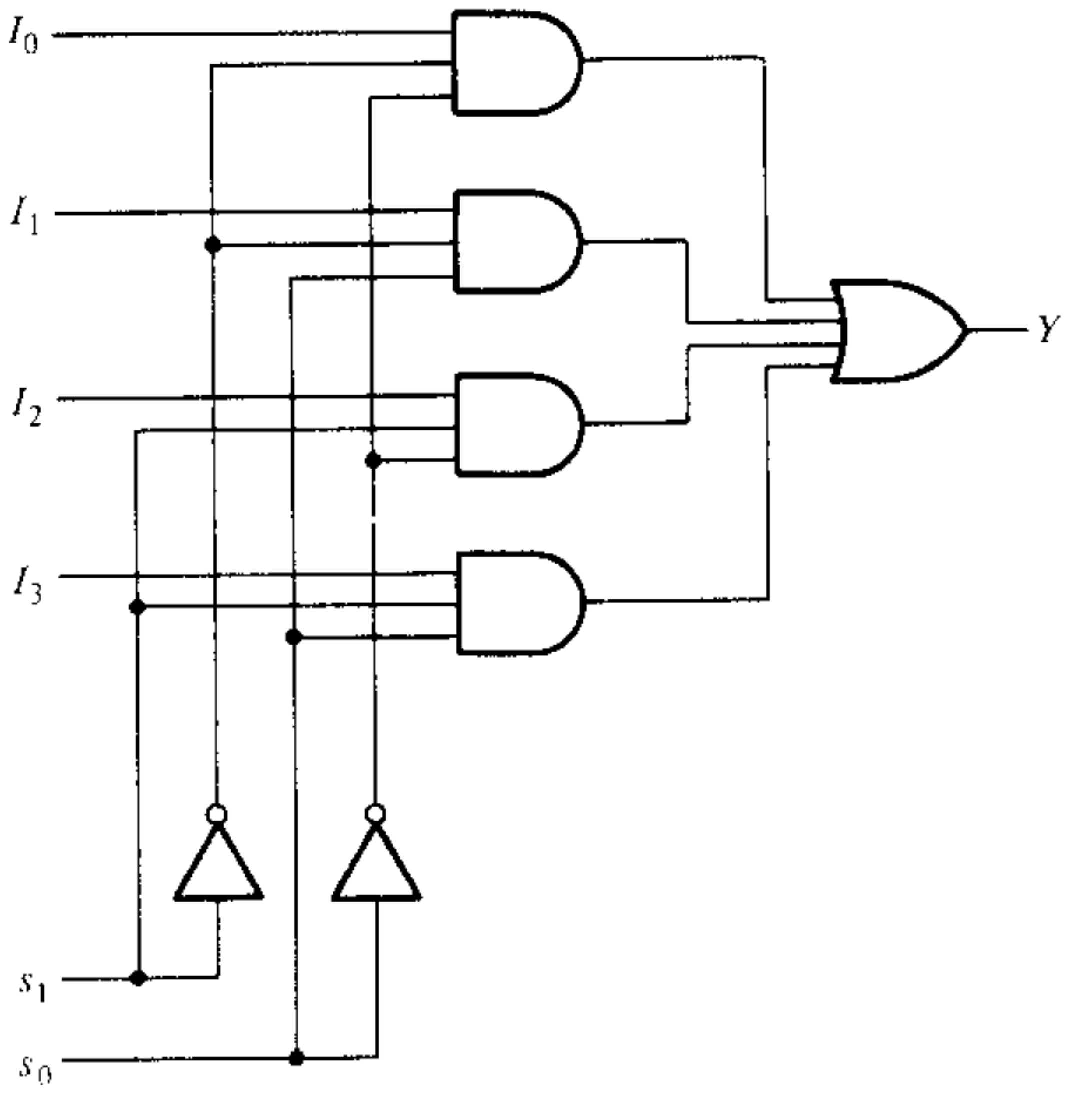

D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

FIGURE 5-13
Octal-to-binary encoder

MULTIPLEXERS

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A *digital multiplexer* is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

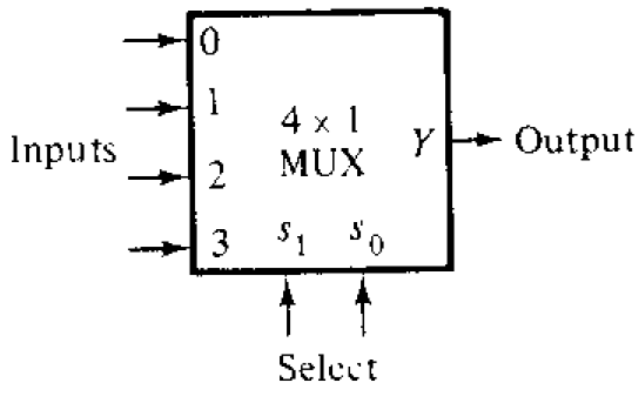
A 4-to-1-line multiplexer is shown in Fig. 5-16. Each of the four input lines, I_0 to I_3, is applied to one input of an AND gate. Selection lines s_1 and s_0 are decoded to select a particular AND gate. The function table, Fig. 5-16(b), lists the input-to-output path for each possible bit combination of the selection lines. When this MSI function is used in the design of a digital system, it is represented in block diagram form, as shown in Fig. 5-16(c). To demonstrate the circuit operation, consider the case when $s_1 s_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output. A multiplexer is also called a *data selector*, since it selects one of many inputs and steers the binary information to the output line.



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table



(c) Block diagram

FIGURE 5-16

A 4-to-1-line multiplexer