Logic Design

Number Systems

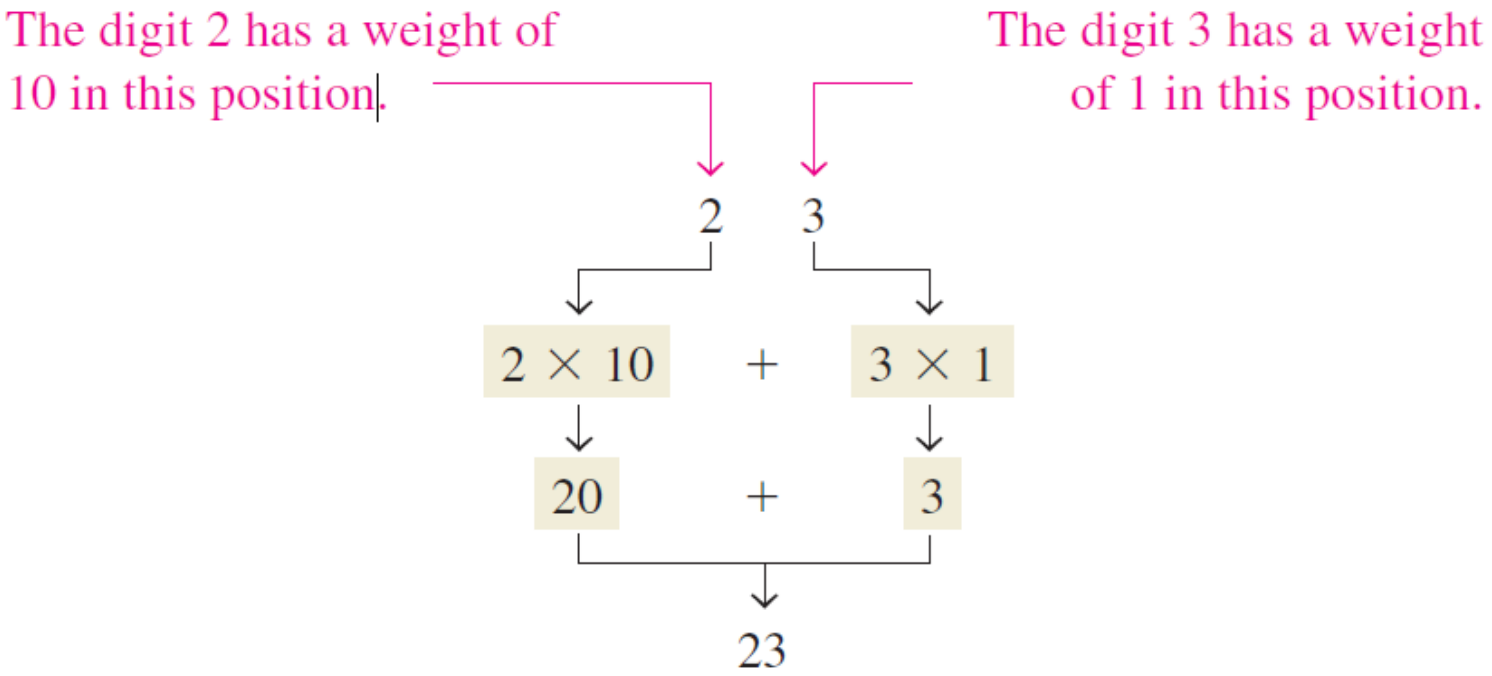
Decimal Numbers

You are familiar with the decimal number system because you use decimal numbers every day. Although decimal numbers are commonplace, their weighted structure is often not understood.

The decimal number system has ten digits.

In the decimal number system each of the ten digits, 0 through 9. When we write decimal (**base 10**) numbers, we use a positional notation; each digit is multiplied by an appropriate power of 10 depending on its position in the number.

example


The digit 2 has a weight of 10 in this position.
The digit 3 has a weight of 1 in this position.
2
3
2 × 10
+
3 × 1
20
+
3
23

The decimal number system has a **base** of 10

The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a weight. The **weights** for whole numbers are positive powers of ten that increase from right to left, beginning with $10^0 = 1$.

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$$

Binary Numbers

The binary number system is another way to represent quantities. the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in a binary number are based on the powers of the number two.

Counting in Binary

A comparable situation occurs when you count in binary, except that you have only two digits, called bits. Begin counting: 0, 1. At this point, you have used both digits, so include another digit position and continue: 10, 11. You have now exhausted all combinations of two digits, so a third position is required. With three digit positions, you can continue to count: 100, 101, 110, and 111. Now you need a fourth digit position to continue, and so on.

TABLE 2–1

Decimal Number	Binary Number			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

The value of a bit is determined by its position in the number

In general, with n bits you can count up to a number equal to $2^n - 1$.

$$\text{Largest decimal number} = 2^n - 1$$

For example, with five bits (n = 5) you can count from zero to thirty-one.

$$2^5 - 1 = 32 - 1 = 31$$

With six bits (n = 6) you can count from zero to sixty-three.

$$2^6 - 1 = 64 - 1 = 63$$

The Weighting Structure of Binary Numbers

A binary number has a different associated “weight”, a bit increases from right to left in a binary number. The right-most bit has a weight of $2^0 = 1$. The weights increase from right to left by a power of two for each bit. The left-most bit's weight depends on the size of the binary number.

..... 2^4 2^3 2^2 2^1 2^0

.....16 8 4 2 1

Binary-to-Decimal Conversion

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

Add the weights of all 1s in a binary number to get the decimal value.

Decimal-to-Binary Conversion

Now you will learn two ways of converting from a decimal number to a binary number.

- Sum-of-Weights Method
- Repeated Division-by-2 Method

Sum-of-Weights Method

One way to find the binary number that is equivalent to a given decimal number is to determine the set of binary weights whose sum is equal to the decimal number.

for example

$$9 = 8 + 1 \text{ or } 9 = 2^3 + 2^0$$

Placing 1s in the appropriate weight positions, 2^3 and 2^0, and 0s in the 2^2 and 2^1 positions determines the binary number for decimal 9.

2^3 2^2 2^1 2^0

1 0 0 1 Binary number for decimal 9

Sum-of-Weights (Decimal Fractions)

The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

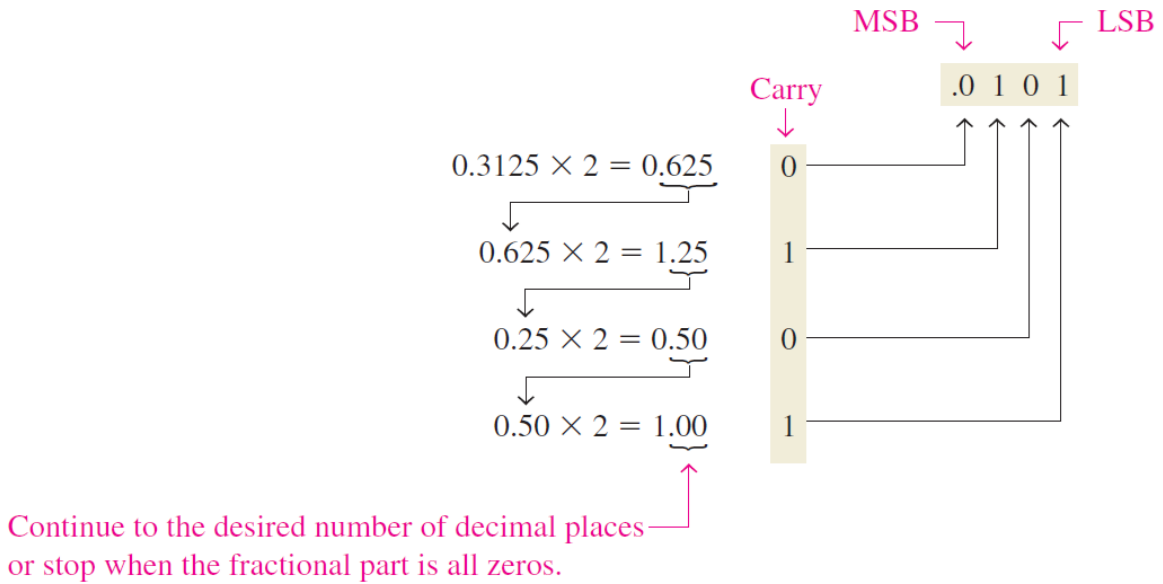
$$0.625 = 0.5 + 0.125$$

$$= \quad 2^{-1} + \quad 2^{-3} \qquad\qquad = 0.101$$

There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position, and a 1 in the 2^{-3} position.

Repeated Multiplication by 2

Decimal fractions can be converted to binary by repeated multiplication by 2.



H.W. convert number from 39.25_{10} to binary

Octal Numbers

the octal number system provides a convenient way to express binary numbers and codes.

The octal number system is composed of eight digits, which are

0, 1, 2, 3, 4, 5, 6, 7

To count above 7, begin another column and start over:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21,

The octal number system has a base of 8.

Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used. To distinguish octal numbers from decimal numbers or hexadecimal numbers, we will use the subscript 8 to indicate an octal number.

Weight: 8^3 8^2 8^1 8^0

Octal-to-Decimal Conversion

The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for 2374_8.

Weight: $8^3\ 8^2\ 8^1\ 8^0$

Octal number: 2 3 7 4

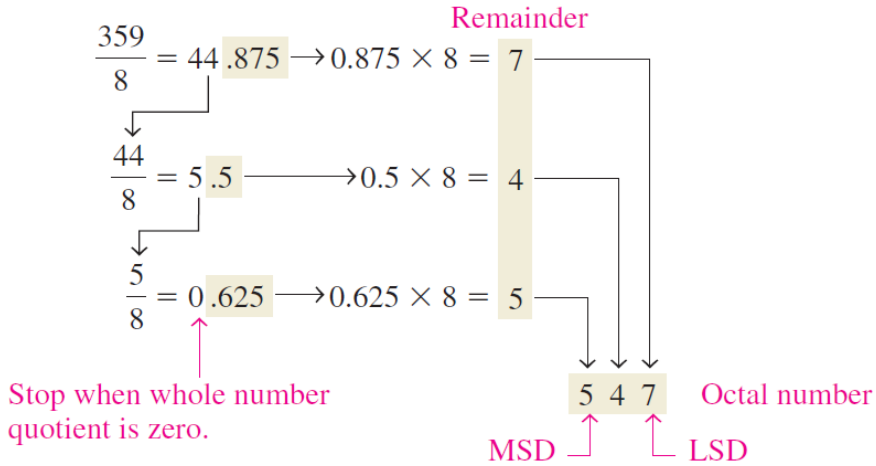
$2374_8 = (2 * 8^3) + (3 * 8^2) + (7 * 8^1) + (4 * 8^0)$

$= (2 * 512) + (3 * 64) + (7 * 8) + (4 * 1)$

$= 1024 + 192 + 56 + 4 = 1276_{10}$

Decimal-to-Octal Conversion

A method of converting a decimal number to an octal number is the repeated division by- 8 method, which is similar to the method used in the conversion of decimal numbers to binary



H.W. convert number 624_{10} to Octal

Octal-to-Binary Conversion

octal is a convenient way to represent binary numbers Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary.

Octal/binary conversion.

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

To convert an octal number to a binary number, simply replace each octal digit with the appropriate three bits.

Binary-to-Octal Conversion

The procedure is as follows:

- Start with the right-most group of three bits and, moving from right to left, convert each 3-bit group to the equivalent octal digit

- If there are not three bits available for the left-most group, add either one or two zeros to make a complete group

Hexadecimal Numbers

The hexadecimal number system has sixteen characters; it is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal

The hexadecimal number system has a base of sixteen

that is, it is composed of 16 numeric and alphabetic characters.The hexadecimal number system consists of digits 0–9 and letters A–F. Most digital systems process binary data in groups that are multiples of four bits

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

Counting in Hexadecimal

How do you count in hexadecimal once you get to F? Simply start over with another column and continue as follows:

0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E , F ,

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F,

20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F,

30, 31, ……

Binary-to-Hexadecimal Conversion

Converting a binary number to hexadecimal is a straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit, and replace each 4-bit group with the equivalent hexadecimal symbol.

Hexadecimal-to-Binary Conversion

Hexadecimal is a convenient way to represent binary numbers.

Conversion between hexadecimal and binary is direct and easy.

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

Since conversion is so easy, the hexadecimal system is widely used for representing binary numbers in programming, printouts, and displays.

Hexadecimal-to-Decimal Conversion

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal.

Hexadecimal-to-Decimal Conversion (Another way)

Another way to convert a hexadecimal number to its decimal equivalent is to multiply the decimal value of each hexadecimal digit by its weight and then take the sum of these products. The weights of a hexadecimal number are increasing powers of 16 (from right to left). For a 4-digit hexadecimal number, the weights are

16^3	16^2	16^1	16^0
4096	256	16	1

Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions.

The first remainder was produced in the least significant digit (LSD). Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number.

Note that when a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

Hexadecimal-to-Octal Conversion

convert number from $(AB3E.87D)_{16}$ to Octal

$(AB3E.87D)_{16} = (1010101100111110.100001111101)_2$

001 010 101 001 111 110 . 100 001 111 101

1 2 5 4 7 6 . 4 1 7 5

$(AB3E.87D)_{16} = (125476.4175)_8$

Octal-to-Hexadecimal Conversion

convert number from 25.342_8 to Hexadecimal

$(25.342)_8 = (010101.011100010)_2$

0001 0101 . 0111 0001

1 2 . 7 1

$(25.342)_8 = (12.71)_{16}$

Binary Arithmetic

Binary arithmetic is essential in all digital computers and in many other types of digital systems. To understand digital systems, you must know the basics of binary addition, subtraction, multiplication, and division.

Binary Addition

The four basic rules for adding binary digits (bits) are as follows:

0 + 0 = 0 Sum of 0 with a carry of 0

0 + 1 = 1 Sum of 1 with a carry of 0

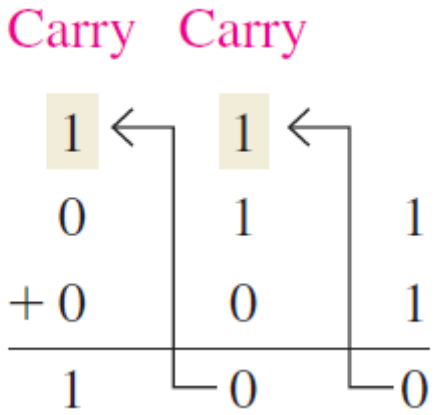
1 + 0 = 1 Sum of 1 with a carry of 0

1 + 1 = 10 Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10).

In binary 1 + 1 = 10, not 2.

When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following addition of 11 + 1:



When there is a carry of 1 This situation is illustrated as follows:

1 + 0 + 0 = 01 Sum of 1 with a carry of 0

1 + 1 + 0 = 10 Sum of 0 with a carry of 1

1 + 0 + 1 = 10 Sum of 0 with a carry of 1

1 + 1 + 1 = 11 Sum of 1 with a carry of 1

Binary Subtraction

The four basic rules for subtracting bits are as follows:

0 - 0 = 0

1 - 1 = 0

1 - 0 = 1

10 - 1 = 1 0 - 1 with a borrow of 1

In binary 10 - 1 = 1, not 9.

Binary Multiplication

The four basic rules for multiplying bits are as follows:

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

EXAMPLE

Perform the following binary divisions: (a) 110 / 11 (b) 110 / 10

$$\textbf{(a)}\quad \begin{array}{r} \mathbf{10} \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

$$\textbf{(b)}\quad \begin{array}{r} \mathbf{11} \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

H.W. Divide 1100 by 100.

Octal Addition

7	6	5	4	3	2	1	0	+
7	6	5	4	3	2	1	0	0
10	7	6	5	4	3	2	1	1
11	10	7	6	5	4	3	2	2
12	11	10	7	6	5	4	3	3
13	12	11	10	7	6	5	4	4
14	13	12	11	10	7	6	5	5
15	14	13	12	11	10	7	6	6
16	15	14	13	12	11	10	7	7

Subtraction in Octal System

The subtraction process in the octal system can be summarized as follows

If the first number is greater than or equal to the seconde , then the process of subtracting is exactly the same in decimal numbers

But if the first number is less than the subtraction, then 1 is borrowed from the next column . This one is expressed as octal 8. It is added to the column from which the subtraction is intended in the number subtracted from it, then the usual subtraction is done in the decimal system

Hexadecimal Addition

Addition can be done directly with hexadecimal numbers by remembering that the hexadecimal digits 0 through 9 are equivalent to decimal digits 0 through 9 and that hexadecimal digits A through F are equivalent to decimal numbers 10 through 15.

Hexadecimal Subtraction

As you have learned, the 2’s complement allows you to subtract by adding binary numbers. Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2’s complement of a binary number.

There are three ways to get the 2’s complement of a hexadecimal number. Method 1 is the most common and easiest to use. Methods 2 and 3 are alternate methods.

EXAMPLE

Subtract the following hexadecimal numbers: (a) 84_{16} - $2A_{16}$ (b) $C3_{16}$ - $0B_{16}$

84_{16}	$C3_{16}$
- $2A_{16}$	$0B_{16}$
$5A_{16}$	$B8_{16}$

H.W. Subtract 173_{16} from BCD_{16}.

One's and Two's Complements of Binary Numbers

Complements of Binary Numbers

The 1’s complement and the 2’s complement of a binary number are important because they permit the representation of negative numbers. The method of 2’s complement arithmetic is commonly used in computers to handle negative numbers.

Finding the 1’s Complement

The 1’s complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0 Binary number

0 1 0 0 1 1 0 1 1’s complement

Finding the 2’s Complement

The 2’s complement of a binary number is found by adding 1 to the LSB of the 1’s complement.

2’s complement = (1’s complement) + 1

Change all bits to the left of the least significant 1 to get 2’s complement

Change all bits to the left of the least significant 1 to get 2’s complement

An alternative method of finding the 2’s complement of a binary number is as follows:

1. Start at the right with the LSB and write the bits as they are up to and including the first 1.

2. Take the 1’s complements of the remaining bits.

Representation of Negative Numbers (Signed Numbers)

In mathematics, when dealing with negative numbers we do use a -ve sign in front of the number to show that the number is negative in value and different from a positive unsigned value, and the same is true with signed binary numbers. However, in computer hardware, numbers are represented only as sequences of bits, without extra symbols

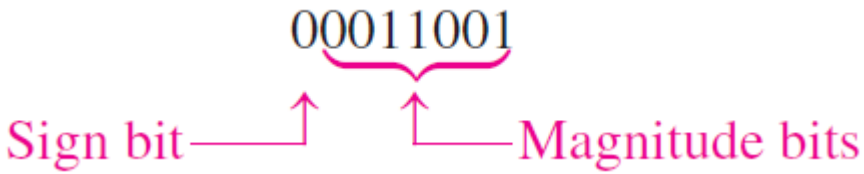
The Sign Bit

The left-most bit in a signed binary number is the sign bit, which tells you whether the number is positive or negative.

A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number.

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the leftmost bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in the true (uncomplemented) binary for both positive and negative numbers. For example, the decimal number +25 is expressed as an 8-bit signed binary number using the sign-magnitude form as



1’s Complement Form

Positive numbers in 1’s complement form are represented the same way as the positive sign-magnitude numbers. Negative numbers, however, are the 1’s complements of the corresponding positive numbers. For example, using eight bits, the decimal number -25 is expressed as the 1’s complement of +25 (00011001) as 11100110 In the 1’s complement form, a negative number is the 1’s complement of the corresponding positive number.

2’s Complement Form

Positive numbers in 2’s complement form are represented the same way as in the sign-magnitude and 1’s complement forms. Negative numbers are the 2’s complements of the corresponding positive numbers. Again, using eight bits, let’s take decimal number -25 and express it as the 2’s complement of +25 (00011001). Inverting each bit and adding 1, you get -25 = 11100111 In the 2’s complement form, a negative number is the 2’s complement of the corresponding positive number.

The Decimal Value of Signed Numbers

Sign-Magnitude Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit.

2’s Complement

Decimal values of positive and negative numbers in the 2’s complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value.

Arithmetic Operations with Signed Numbers

In this section, you will learn how signed numbers are added, subtracted, multiplied, and divided.

Because the 2’s complement form for representing signed numbers is the most widely used in computers and microprocessor-based systems, the coverage in this section is limited to 2’s complement arithmetic. The processes covered can be extended to the other forms if necessary.

Addition

There are four cases that can occur when two signed binary numbers are added.

1. Both numbers positive

2. Positive number with magnitude larger than negative number

3. Negative number with magnitude larger than positive number

4. Both numbers negative

Let’s take one case at a time using 8-bit signed numbers as examples.

Both numbers positive:

00000111	7
+ 00000100	4
00001011	11

The sum is positive and is therefore in true (uncomplemented) binary.

The sum is positive and is therefore in true (uncomplemented) binary.

Positive number with magnitude larger than negative number:

00001111 15

+ 11111010 -6

1 00001001 9

The final carry bit is discarded. The sum is positive and therefore in true (uncomplemented) binary.

Negative number with magnitude larger than positive number:

00010000	16
+ 11101000	-24
11111000	-8

The sum is negative and therefore in 2’s complement form.

Both numbers are negative:

11111011 -5

+ 11110111 -9

1 11110010 -14

The final carry bit is discarded. The sum is negative and therefore in 2’s complement form.

Subtraction

Subtraction is a special case of addition. For example, subtracting +6 (the subtrahend) from +9 (the minuend) is equivalent to adding 26 to +9. Basically, the subtraction operation changes the sign of the subtrahend and adds it to the minuend. The result of subtraction is called the difference.

The sign of a positive or negative binary number is changed by taking its 2’s complement.

Multiplication

The numbers in multiplication are the multiplicand, the multiplier, and the product. These are illustrated in the following decimal multiplication:

8 Multiplicand

* 3 Multiplier

24 Product

Division

The numbers in a division are the dividend, the divisor, and the quotient. These are illustrated in the following standard division format.

dividend = quotient

divisor

Division

The division operation in computers is accomplished using subtraction. Since subtraction is done with an adder, division can also be accomplished with an adder. The result of a division is called the quotient; the quotient is the number of times that the divisor will go into the dividend. This means that the divisor can be subtracted from the dividend a number of times equal to the quotient, as illustrated by dividing 21 by 7.

21 Dividend

- 7 1st subtraction of divisor

14 1st partial remainder

- 7 2nd subtraction of divisor

7 2nd partial remainder

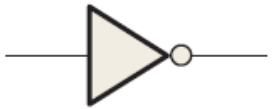
- 7 3rd subtraction of divisor

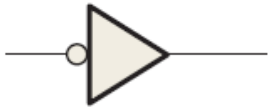
0 Zero remainder

Logic Gates

The Inverter

The inverter performs the operation called inversion or complementation. The inverter changes one logic level to the opposite level. In terms of bits, it changes a 1 to a 0 and a 0 to a 1.





The negation indicator is a “bubble” (O) that indicates inversion or complementation when it appears on the input or output of any logic element. Generally, inputs are on the left of a logic symbol and the output is on the right.

Inverter Truth Table

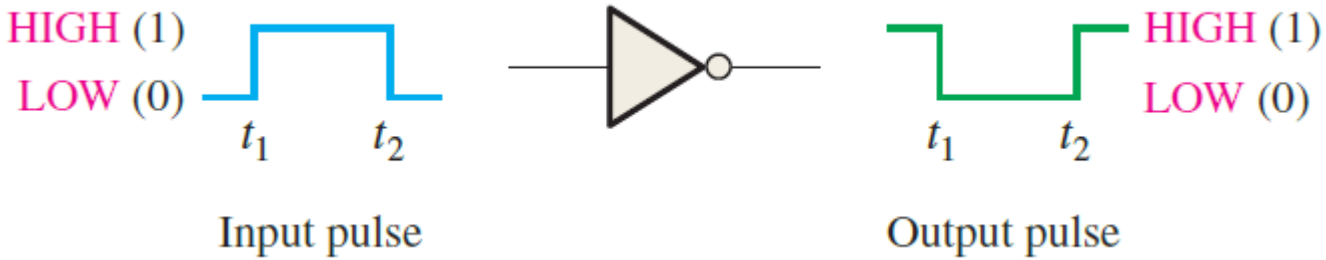
When a HIGH level is applied to an inverter input, a LOW level will appear on its output. When a LOW level is applied to its input, a HIGH will appear on its output. This operation is summarized in the next Table , which shows the output for each possible input in terms of levels and corresponding bits. A table such as this is called a truth table

Inverter truth table.	
Input	Output
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

Inverter Operation

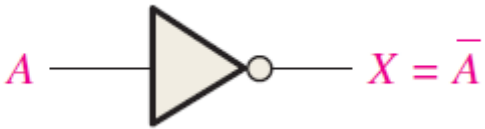
the next Figure shows the output of an inverter for a pulse input, where t1 and t2 indicate the corresponding points on the input and output pulse waveforms.

When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.



Logic Expression for an Inverter

In Boolean algebra, which is the mathematics of logic circuits a variable is generally designated by one or two letters although there can be more. The complement of a variable is designated by a bar over the letter. A variable can take on a value of either 1 or 0. If a given variable is 1, its complement is 0 and vice versa.



The AND Gate

The AND gate is one of the basic gates that can be combined to form any logic function. An AND gate can have two or more inputs and performs what is known as logical multiplication.

The AND gate is composed of two or more inputs and a single output, as indicated by the standard logic symbols shown in the next Figure. Inputs are on the left, and the output is on the right in each symbol. Gates with two inputs are shown; however, an AND gate can have any number of inputs greater than one.

Operation of an AND Gate

For a 2-input AND gate, output X is HIGH only when inputs A and B are HIGH; X is LOW when either A or B is LOW, or when both A and B are LOW.

AND Gate Truth Table

The logical operation of a gate can be expressed with a truth table that lists all input combinations with the corresponding outputs, as illustrated in the next Table for a 2-input AND gate. The truth table can be expanded to any number of inputs.

Truth table for a 2-input AND gate.		
Input A	b	Output x
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Truth Table

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

$N = 2^n$

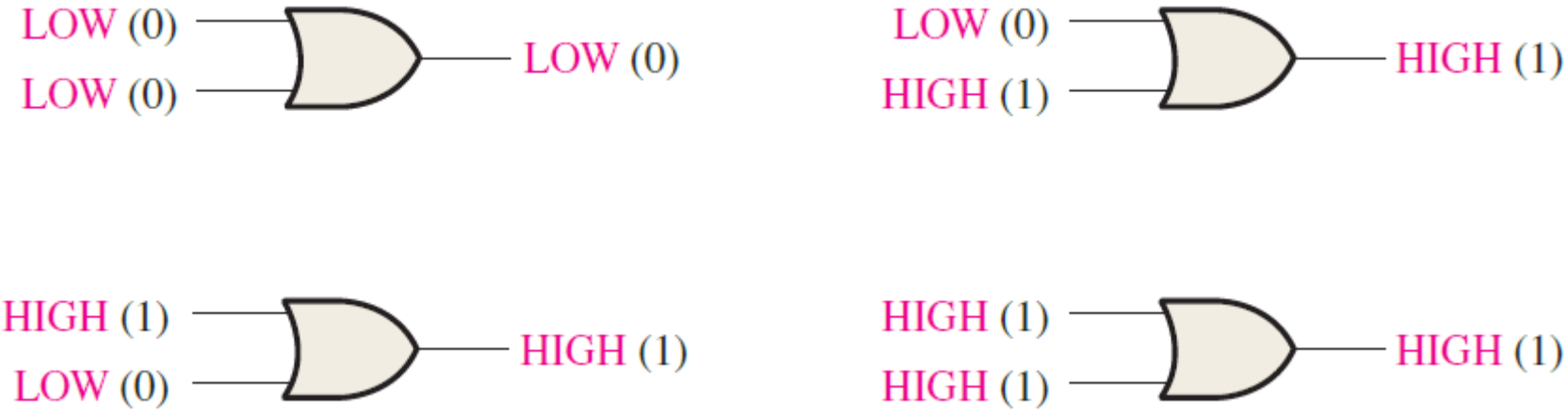
where N is the number of possible input combinations and n is the number of input variables.

The OR Gate

The OR gate is another of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is known as logical addition.

Operation of an OR Gate

For a 2-input OR gate, output X is HIGH when either input A or input B is HIGH, or when both A and B are HIGH; X is LOW only when both A and B are LOW.



OR Gate Truth Table

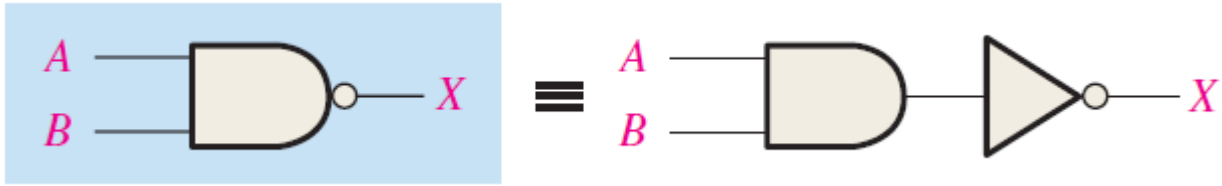
The operation of a 2-input OR gate is described in Table . This truth table can be expanded for any number of inputs; but regardless of the number of inputs, the output is HIGH when one or more of the inputs are HIGH.

Truth table for a 2-input OR gate.		
Input A	b	Output x
0	0	0
0	1	1
1	0	1
1	1	1

The NAND Gate

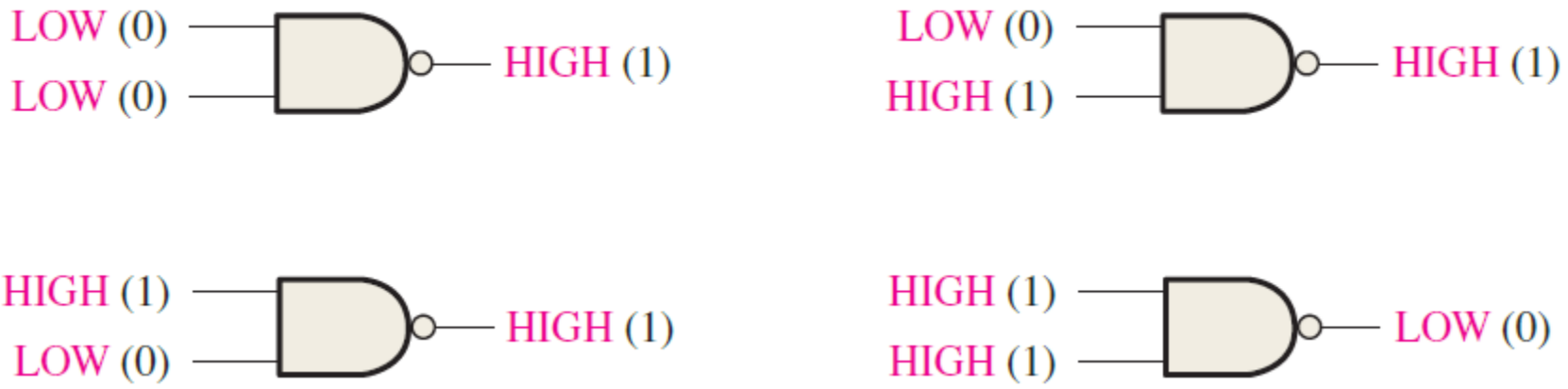
The NAND gate is a popular logic element because it can be used as a universal gate; that is, NAND gates can be used in combination to perform the AND, OR, and inverter operations.

The term NAND is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output.



Operation of a NAND Gate

For a 2-input NAND gate, output X is LOW only when inputs A and B are HIGH; X is HIGH when either A or B is LOW, or when both A and B are LOW.

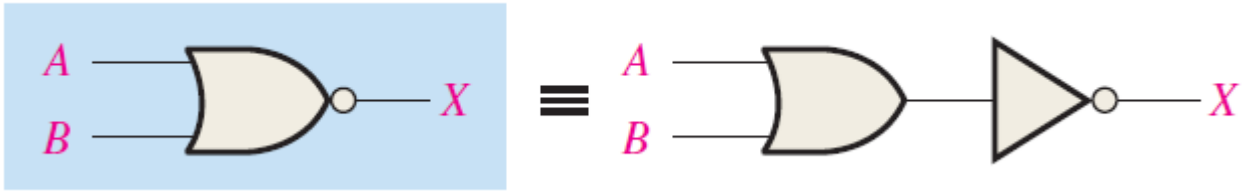


NAND Gate Truth Table

Truth table for a 2-input NAND gate.		
Input A	b	Output x
0	0	1
0	1	1
1	0	1
1	1	0

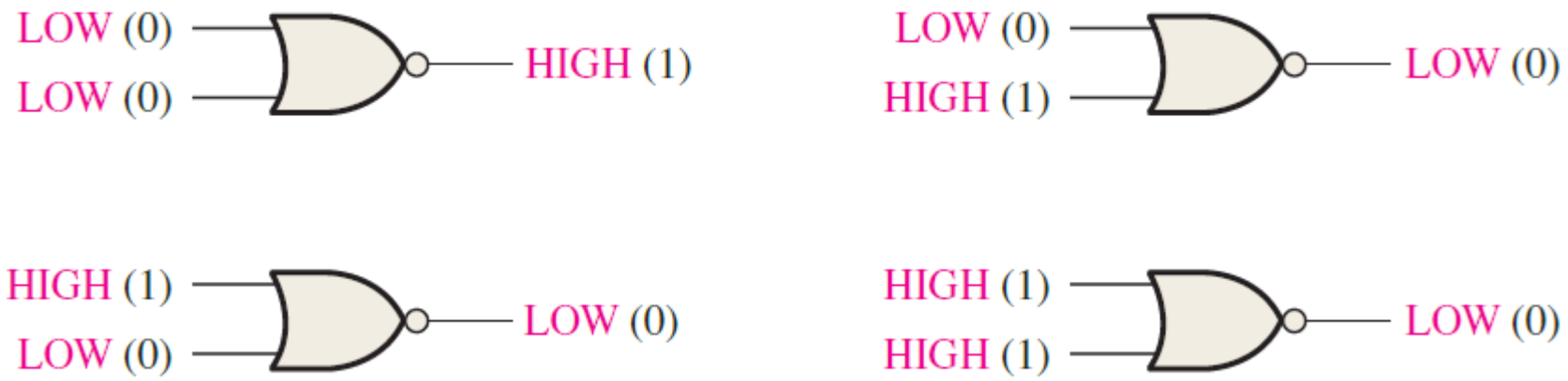
The NOR Gate

The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations.



Operation of a NOR Gate

For a 2-input NOR gate, output X is LOW when either input A or input B is HIGH, or when both A and B are HIGH; X is HIGH only when both A and B are LOW.



NOR Gate Truth Table

Table is the truth table for a 2-input NOR gate.

Truth table for a 2-input NOR gate.		
Input A	b	Output x
0	0	1
0	1	0
1	0	0
1	1	0

Logic Expressions for a NOR Gate

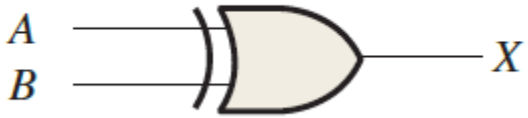
The Boolean expression for the output of a 2-input NOR gate can be written as

X = A + B

A B	A B X
0 0	0 + 0 = 0 = 1
0 1	0 + 1 = 1 = 0
1 0	1 + 0 = 1 = 0
1 1	1 + 1 = 1 = 0

The Exclusive-OR Gate

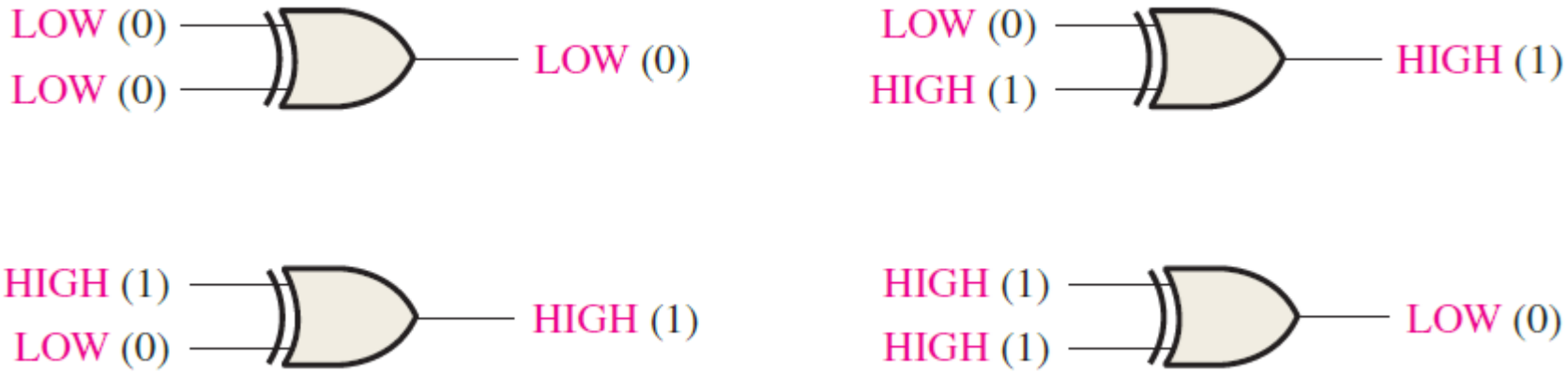
Exclusive-OR gates connected to form an adder circuit allow a processor to perform addition, subtraction, multiplication, and division in its Arithmetic Logic Unit (ALU). An exclusive-OR gate combines basic AND, OR, and NOT logic.



Standard symbols for an exclusive-OR (XOR for short) gate are shown in the next Figure. The XOR gate has only two inputs. The exclusive-OR gate performs modulo-2 addition. The output of an exclusive-OR gate is HIGH only when the two

Operation of an exclusive-OR Gate

For an exclusive-OR gate, output X is HIGH when input A is LOW and input B is HIGH, or when input A is HIGH and input B is LOW; X is LOW when A and B are both HIGH or both LOW.



exclusive-OR Gate Truth Table

The operation of an XOR gate is summarized in the truth table

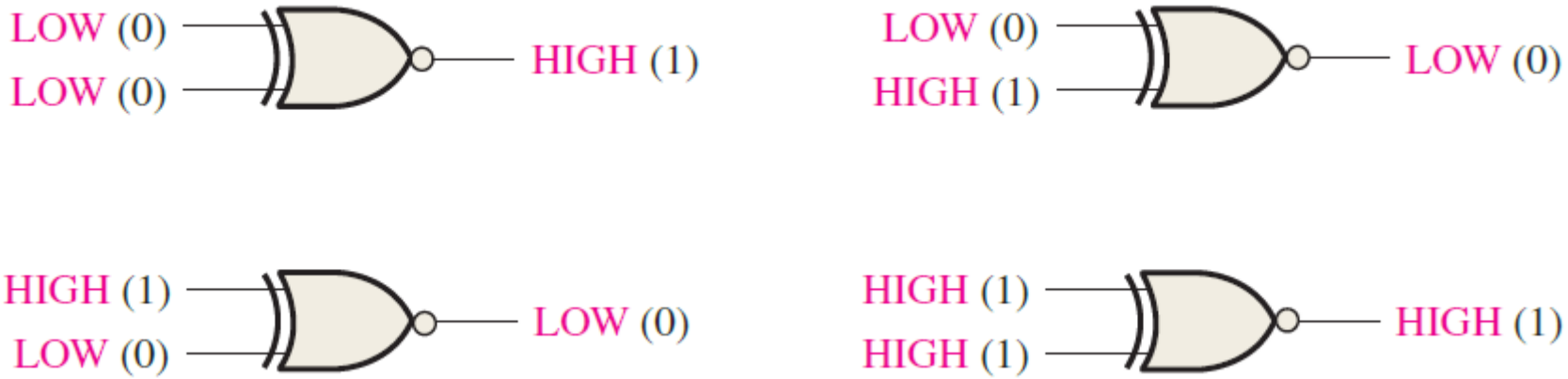
Truth table for a 2-input exclusive-OR gate.		
Input A	b	Output x
0	0	0
0	1	1
1	0	1
1	1	0

The Exclusive-NOR Gate

Standard symbols for an exclusive-NOR (XNOR) gate are shown in the next Figure. Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate.

Operation of an exclusive-NOR Gate

For an exclusive-NOR gate, output X is LOW when input A is LOW and input B is HIGH, or when A is HIGH and B is LOW; X is HIGH when A and B are both HIGH or both LOW.



exclusive-NOR Gate Truth Table

The operation of an XNOR gate is summarized in Table. Notice that the output is HIGH when the same level is on both inputs.

Truth table for a 2-input exclusive-NOR gate.		
Input A	b	Output x
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

Boolean Operations and Expressions

Boolean algebra is the mathematics of digital logic. A basic knowledge of Boolean algebra is indispensable to the study and analysis of logic circuits

A variable is a symbol (usually an italic uppercase letter or word) used to represent an action, a condition, or data.

The complement is the inverse of a variable and is indicated by a bar over the variable (overbar).

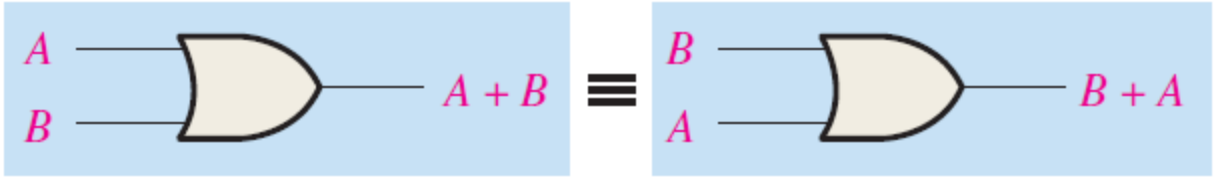
Laws of Boolean Algebra

The basic laws of Boolean algebra—the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law—are the same as in ordinary algebra. Each of the laws is illustrated with two or three variables, but the number of variables is not limited to this.

Commutative Laws

The commutative law of addition for two variables is written as

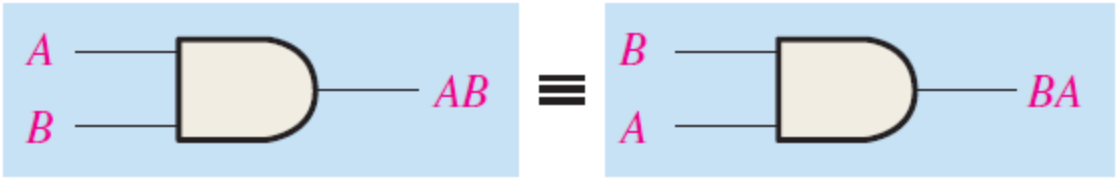
A + B = B + A



Commutative Laws

The commutative law of multiplication for two variables is

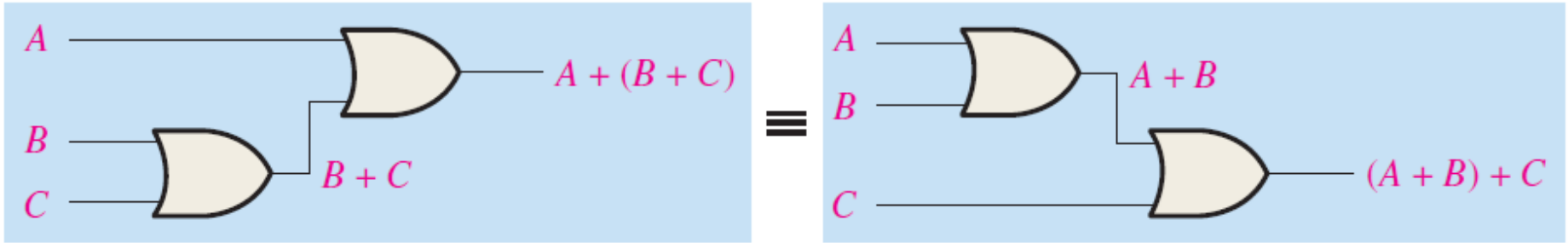
AB = BA



Associative Laws

The associative law of addition is written as follows for three variables:

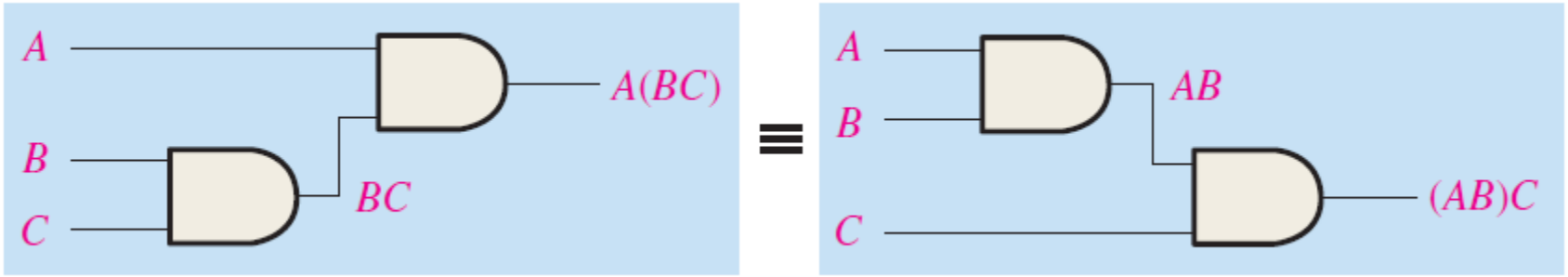
A + (B + C) = (A + B) + C



Associative Laws

The associative law of multiplication is written as follows for three variables:

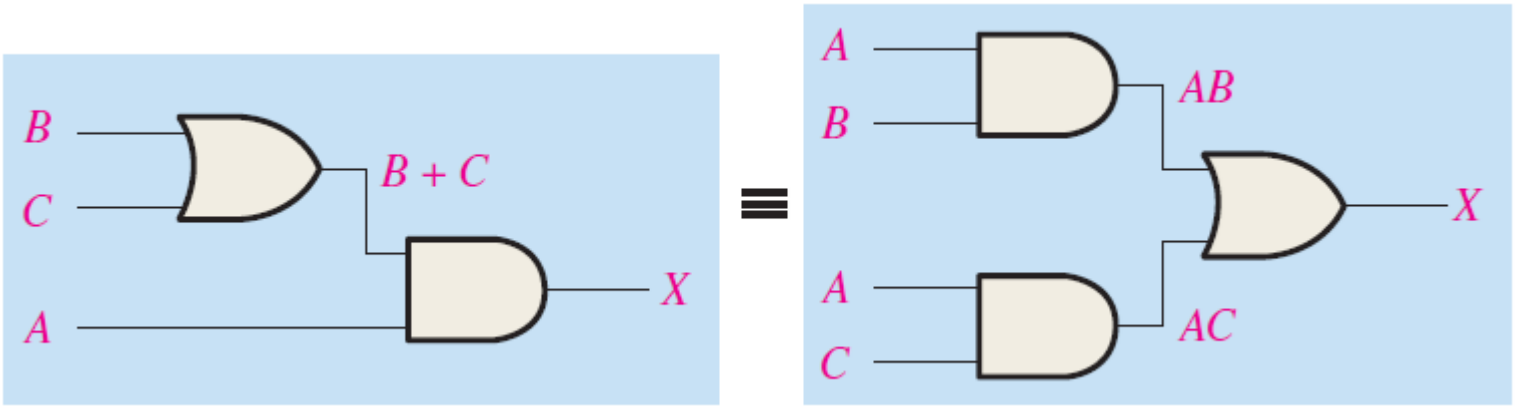
A(BC) = (AB)C



Distributive Law

The distributive law is written for three variables as follows:

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



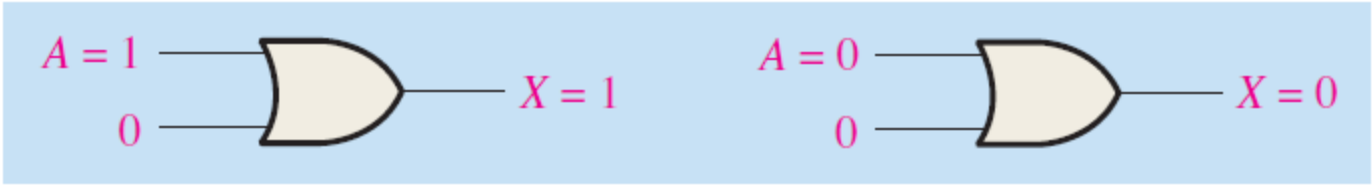
Laws and Rules of Boolean Algebra

Basic rules of Boolean algebra.

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A * 0 = 0$
4. $A * 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$

7. $A * A = A$
8. $A * A = 0$
9. $A = A$
10. $A + AB = A$
11. $A + AB = A + B$
12. $(A + B)(A + C) = A + BC$

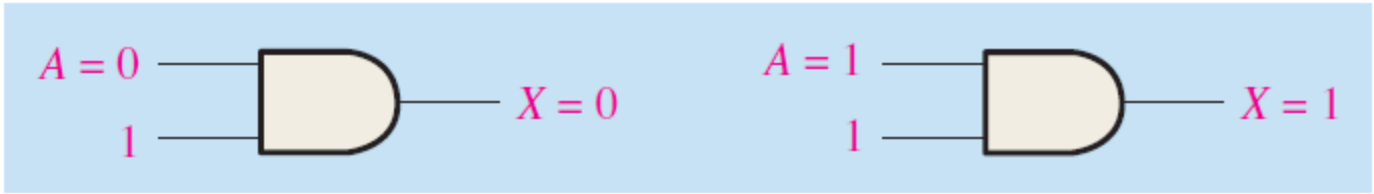
Rule 1: A + 0 = A



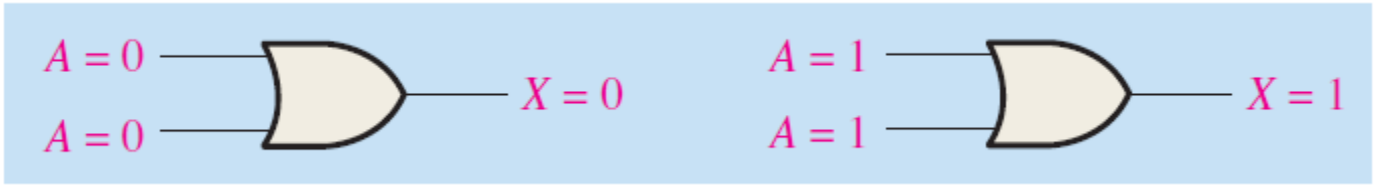
Rule 2: A + 1 = 1

Rule 3. A * 0 = 0

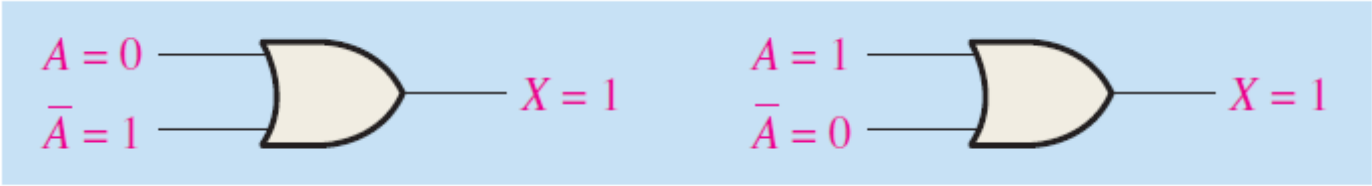
Rule 4. A * 1 = A



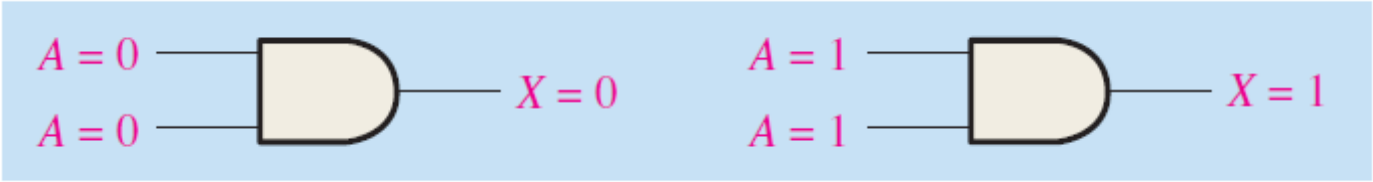
Rule .5 A + A = A



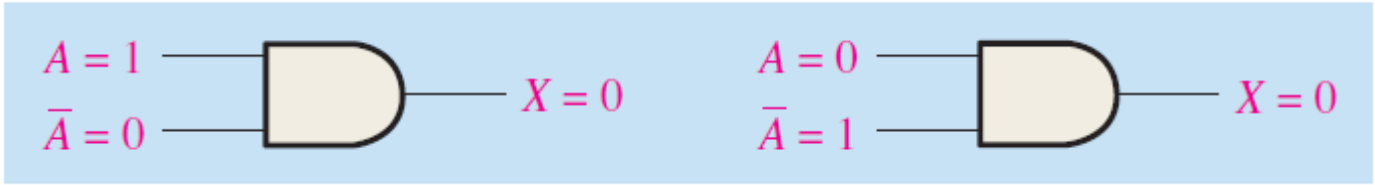
Rule 6. $A + \overline{A} = 1$



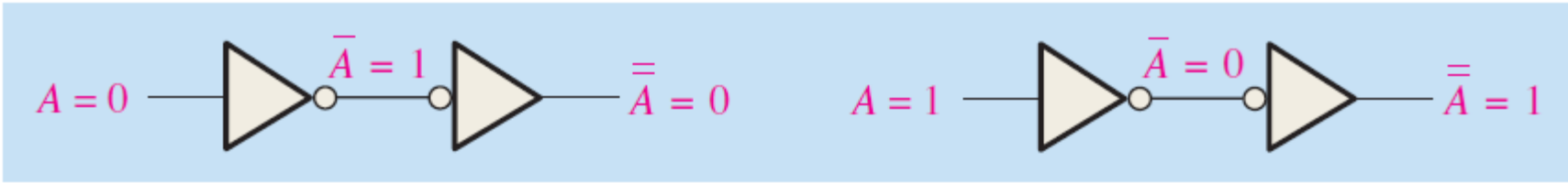
Rule 7. A * A = A



Rule 8. $A * \overline{A} = 0$



Rule 9. $\overline{\overline{A}} = A$



Rule 10: A + AB = A

A + AB = A * 1 + AB = A(1 + B) Factoring (distributive law)

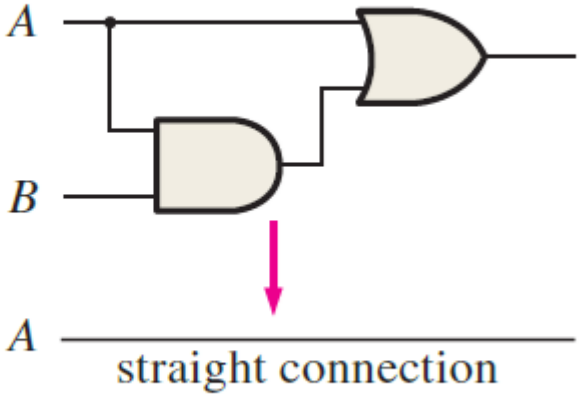
= A * 1 Rule 2: (1 + B) = 1

= A Rule 4: A * 1 = A

Rule 10: A + AB = A

A	*B*	*AB*	*A* + *AB*
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



Rule 11: $A + \overline{A}B = A + B$

$A + \overline{A}B = (A + AB) + \overline{A}B$ Rule 10: $A = A + AB$

$= (AA + AB) + \overline{A}B$ Rule 7: $A = AA$

$= AA + AB + A\overline{A} + \overline{A}B$ Rule 8: adding $A\overline{A} = 0$

$= (A + \overline{A})(A + B)$ Factoring

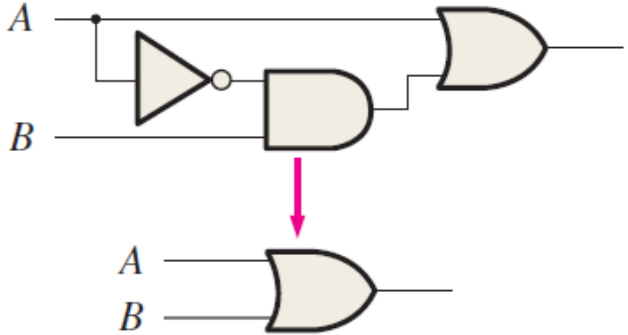
$= 1 * (A + B)$ Rule 6: $A + \overline{A} = 1$

$= A + B$ Rule 4: drop the 1

Rule 11: $A + \overline{A}B = A + B$

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

equal



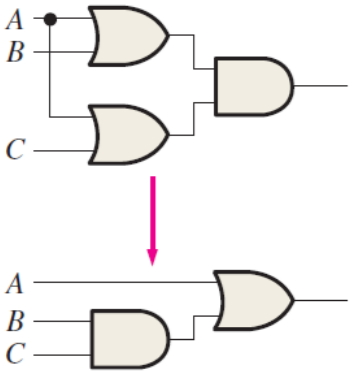
Rule 12: (A + B)(A + C) = A + BC

(A + B)(A + C)	= AA + AC + AB + BC	Distributive law
	= A + AC + AB + BC	Rule 7: AA = A
	= A(1 + C) + AB + BC	Factoring (distributive law)
	= A * 1 + AB + BC	Rule 2: 1 + C = 1
	= A(1 + B) + BC	Factoring (distributive law)
	= A * 1 + BC	Rule 2: 1 + B = 1
	= A + BC	Rule 4: A * 1 = A

Rule 12: (A + B)(A + C) = A + BC

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



DeMorgan’s Theorems

DeMorgan’s Theorems

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms, DeMorgan’s theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

DeMorgan’s first theorem is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables.

Stated another way,

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

$\overline{XY} = \overline{X} + \overline{Y}$

DeMorgan’s second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

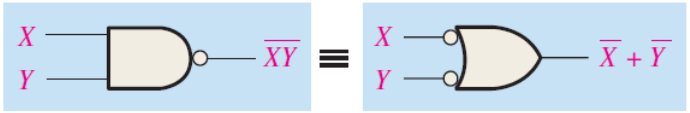
Stated another way,

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables.

The formula for expressing this theorem for two variables is

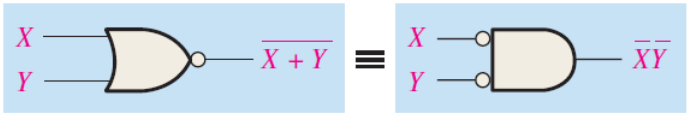
$X + Y = \overline{X}\,\overline{Y}$

Gate equivalencies and the corresponding truth tables for DeMorgan's theorems



NAND Negative-OR

Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0



NOR Negative-AND

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Applying DeMorgan's Theorems

$$\overline{A + B\overline{C}} + \overline{D(E + \overline{F})}$$

Step 1: Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable. Let $\overline{A + B\overline{C}} = X$ and $\overline{D(E + \overline{F})} = Y$.

Step 2: Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

$$\overline{(A + B\overline{C}) + (D(E + \overline{F}))} = \overline{(A + B\overline{C})}\,\overline{(D(E + \overline{F}))}$$

Step 3: Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

$$(A + B\overline{C})\,\overline{(D(E + \overline{F}))} = (A + B\overline{C})\,\overline{(D(E + \overline{F}))}$$

Step 4: Apply DeMorgan's theorem to the second term.

$$(A + B\overline{C})\,\overline{(D(E + \overline{F}))} = (A + B\overline{C})(\overline{D} + \overline{(E + \overline{F})})$$

Step 5: Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the $E + \overline{F}$ part of the term.

$$(A + B\overline{C})(\overline{D} + \overline{E + \overline{F}}) = (A + B\overline{C})(\overline{D} + E + \overline{F})$$

Logic Simplification Using Boolean Algebra

Logic Simplification Using Boolean Algebra

A logic expression can be reduced to its simplest form or changed to a more convenient form to implement the expression most efficiently using Boolean algebra.

EXAMPLE

Using Boolean algebra, simplify this expression: $AB + A(B + C) + B(B + C)$

Step 1: Apply the distributive law to the 2nd and 3rd $AB + AB + AC + BB + BC$

Step 2: Apply rule 7 ($BB = B$) to the fourth term. $AB + AB + AC + B + BC$

Step 3: Apply rule 5 ($AB + AB = AB$) to the first two terms. $AB + AC + B + BC$

Step 4: Apply rule 10 ($B + BC = B$) to the last two terms. $AB + AC + B$

Step 5: Apply rule 10 ($AB + B = B$) to the first and third terms. $B + AC$

EXAMPLE $\overline{[AB(C + BD) + A\,B]}C$

Step 1: Apply the distributive law to the terms within the brackets.

$$(A\overline{B}C + A\overline{B}BD + \overline{A}\,\overline{B})C$$

Step 2: Apply rule 8 ($\overline{B}B = 0$) to the second term within the parentheses.

$$(A\overline{B}C + A \cdot 0 \cdot D + \overline{A}\,\overline{B})C$$

Step 3: Apply rule 3 ($A \cdot 0 \cdot D = 0$) to the second term within the parentheses.

$$(A\overline{B}C + 0 + \overline{A}\,\overline{B})C$$

Step 4: Apply rule 1 (drop the 0) within the parentheses.

$$(A\overline{B}C + \overline{A}\,\overline{B})C$$

Simplify the following Boolean expression:

$$\overline{AB + AC} + \overline{A}\,\overline{B}C$$

Solution

Step 1: Apply DeMorgan's theorem to the first term.

$$(\overline{AB})(\overline{AC}) + \overline{A}\,\overline{B}C$$

Step 2: Apply DeMorgan's theorem to each term in parentheses.

$$(\overline{A} + \overline{B})(\overline{A} + \overline{C}) + \overline{A}\,\overline{B}C$$

Step 3: Apply the distributive law to the two terms in parentheses.

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

Step 4: Apply rule 7 ($\overline{A}\,\overline{A} = \overline{A}$) to the first term, and apply rule 10 [$\overline{A}\,\overline{B} + \overline{A}\,\overline{B}C = \overline{A}\,\overline{B}(1 + C) = \overline{A}\,\overline{B}$] to the third and last terms.

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

Step 5: Apply rule 10 [$\overline{A} + \overline{A}\,\overline{C} = \overline{A}(1 + \overline{C}) = \overline{A}$] to the first and second terms.

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

Step 6: Apply rule 10 [$\overline{A} + \overline{A}\,\overline{B} = \overline{A}(1 + \overline{B}) = \overline{A}$] to the first and second terms.

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

Standard Forms of Boolean Expressions

Standard Forms of Boolean Expressions

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

The Sum-of-Products (SOP) Form

A product term was defined as a term consisting of the product (Boolean multiplication) of literals (variables or their complements). When two or more product terms are summed by Boolean addition, the resulting expression is a sum of products (SOP). Some examples are

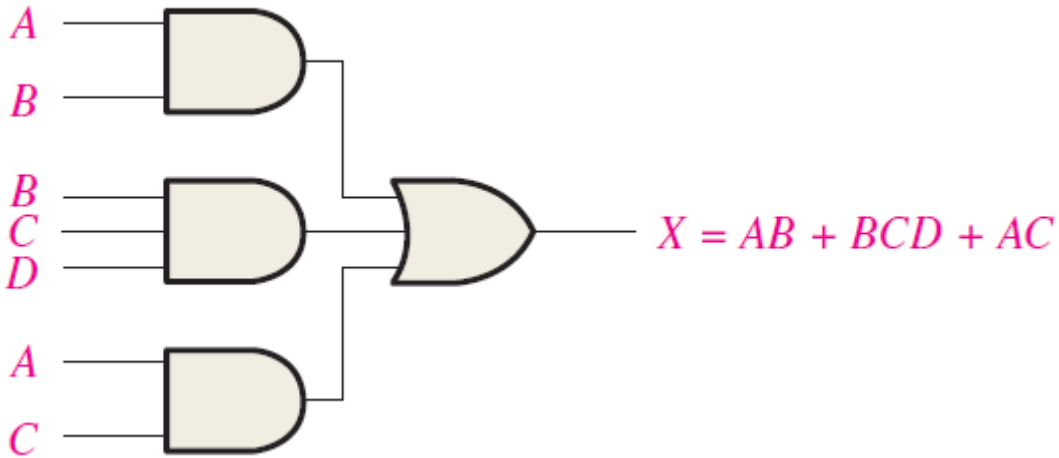
$AB + A\bar{B}C$

$\bar{A}BC + C\bar{D}E + BCD$

$AB + ABC + AC$

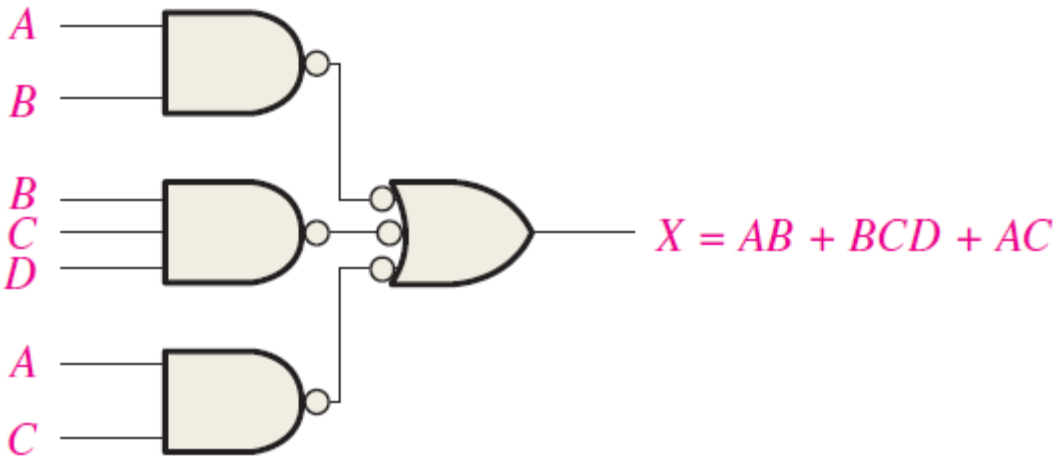
AND/OR Implementation of an SOP Expression

Implementing an SOP expression simply requires ORing the outputs of two or more AND gates. A product term is produced by an AND operation, and the sum (addition) of two or more product terms is produced by an OR operation.



NAND/NAND Implementation of an SOP Expression

NAND gates can be used to implement an SOP expression. By using only NAND gates, an AND/OR function can be accomplished, as illustrated in Figure



Conversion of a General Expression to SOP Form

Any logic expression can be changed into SOP form by applying Boolean algebra techniques.

For example, the expression A(B + CD) can be converted to SOP form by applying the distributive law:

$A(B + CD) = AB + ACD$

The Standard SOP Form

the expression ABC + ABD + ABCD has a domain made up of the variables A, B, C, and D. However, notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or D is missing from the first term and C or C is missing from the second term.

A standard SOP expression is one in which all the variables in the domain appear in each product term in the expression. For example, ABCD + A BCD + ABC D is a standard SOP expression.

Converting Product Terms to Standard SOP

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into a standard form using Boolean algebra rule 6 (A + A = 1)

Step 1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement.

Step 2: Repeat Step 1 until all resulting product terms contain all variables in the domain in either complimented or uncomplemented form.

The Product-of-Sums (POS) Form

A sum term was defined as a term consisting of the sum (Boolean addition) of literals (variables or their complements).

Some examples are

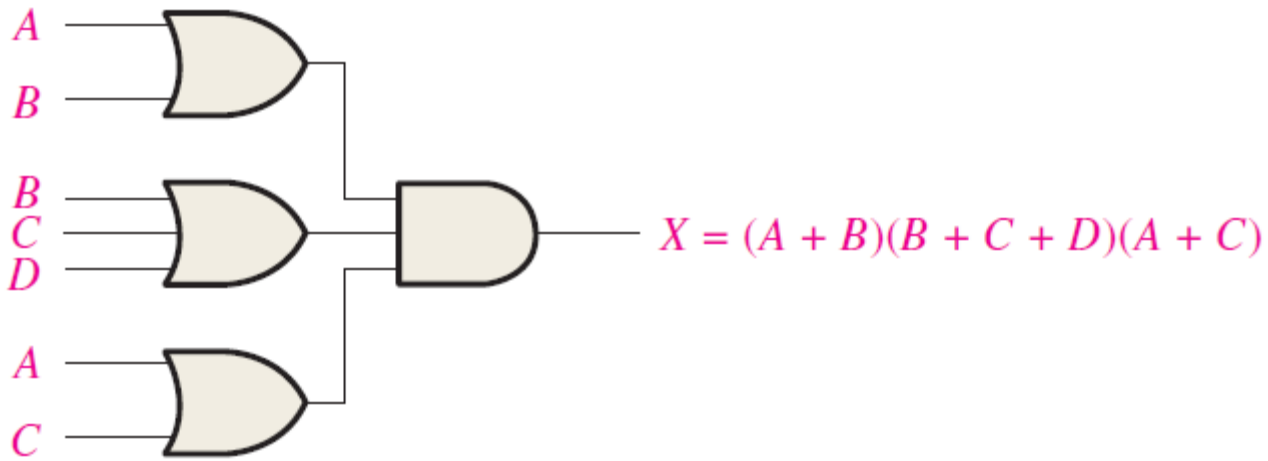
$(A + B)(A + \overline{B} + C)$

$(\overline{A} + \overline{B} + \overline{C})(C + \overline{D} + E)(\overline{B} + C + D)$

$(A + B)(A + \overline{B} + C)(\overline{A} + C)$

Implementation of a POS Expression

Implementing a POS expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation, and the product of two or more sum terms is produced by an AND operation.



The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example, the expression

$$(A + \overline{B} + C)(A + B + \overline{D})(A + \overline{B} + \overline{C} + D)$$

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression. For example,

$$(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + C + D)(A + B + \overline{C} + D)$$

Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements.

Step 1: Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms. As you know, you can add 0 to anything without changing its value.

Step 2: Apply rule 12 : $A + BC = (A + B)(A + C)$

Step 3: Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or uncomplemented form

EXAMPLE

Convert the following Boolean expression into standard POS form:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time. The first term, $A + \overline{B} + C$, is missing variable D or $\overline{D}$, so add $D\overline{D}$ and apply rule 12 as follows:

$$A + \overline{B} + C = A + \overline{B} + C + D\overline{D} = (A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})$$

The second term, $\overline{B} + C + \overline{D}$, is missing variable A or $\overline{A}$, so add $A\overline{A}$ and apply rule 12 as follows:

$$\overline{B} + C + \overline{D} = \overline{B} + C + \overline{D} + A\overline{A} = (A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})$$

The third term, $A + \overline{B} + \overline{C} + D$, is already in standard form. The standard POS form of the original expression is as follows:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D) =$$
$$(A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Related Problem

Convert the expression $(A + \overline{B})(B + C)$ to standard POS form.

The Karnaugh Map

Karnaugh map

A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression. As you have seen, the effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them. The Karnaugh map, on the other hand, provides a “cookbook” method for simplification. Other simplification techniques include the Quine-McCluskey method and the Espresso algorithm.

Karnaugh map

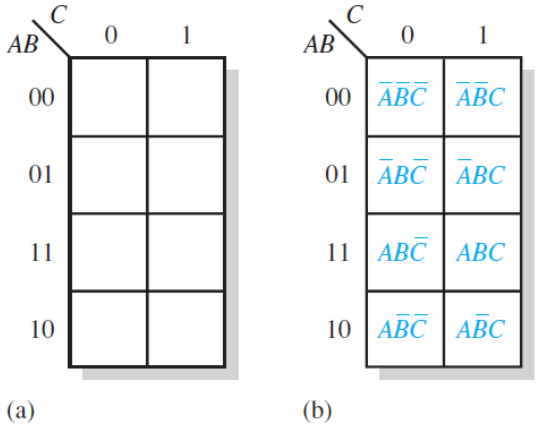
A Karnaugh map is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells.

The number of cells in a Karnaugh map

The number of cells in a Karnaugh map, as well as the number of rows in a truth table, is equal to the total number of possible input variable combinations. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.

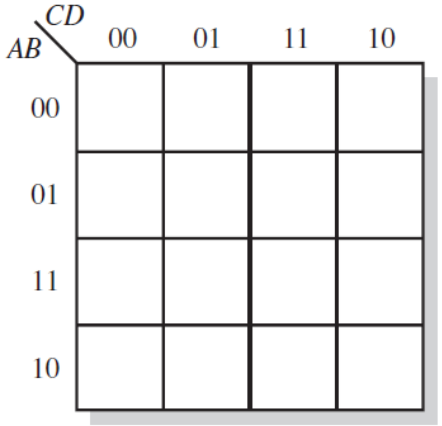
The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells. In this case, A, B, and C are used for the variables although other letters could be used. Binary values of A and B are along the left side (notice the sequence) and the values of C are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C at the top in the same column.

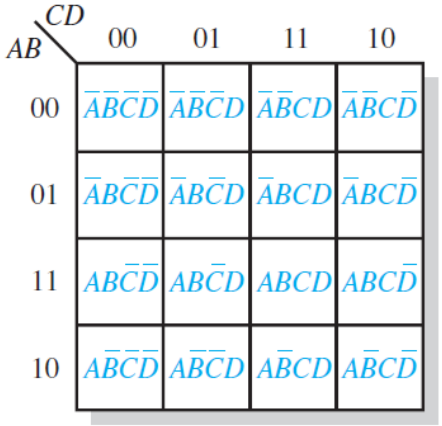


The 4-Variable Karnaugh Map

The 4-variable Karnaugh map is an array of sixteen cells, as shown in Figure (a). Binary values of A and B are along the left side and the values of C and D are across the top. The value of a given cell is the binary values of A and B at the left in the same row combined with the binary values of C and D at the top in the same column.



(a)



(b)

Cell Adjacency

Physically, each cell is adjacent to the cells that are immediately next to it on any of its four sides. A cell is not adjacent to the cells that diagonally touch any of its corners. Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column. This is called “wrap-around” adjacency because you can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder.

The figure illustrates the cell adjacencies with a 4-variable map, although the same rules for adjacency apply to Karnaugh maps with any number of cells.

Karnaugh Map SOP Minimization

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term. Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression. In this section, Karnaugh maps with up to four variables are covered.

Mapping a Standard SOP Expression

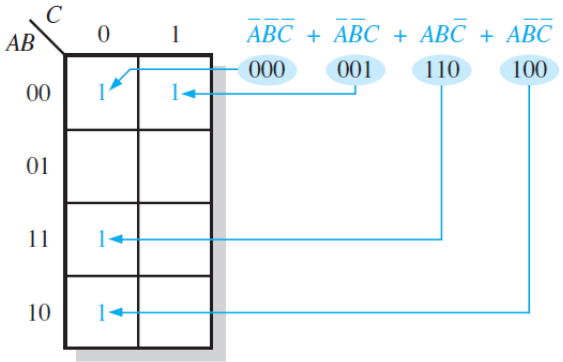
For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term ABC, a 1 goes in the 101 cell on a 3-variable map When an SOP expression is completely mapped, there will be a number of 1s on the Karnaugh map equal to the number of product terms in the standard SOP expression.

Steps

The following steps and the illustration in Figure show the mapping process.

Step 1: Determine the binary value of each product term in the standard SOP expression.

Step 2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.



Karnaugh Map Simplification of SOP Expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called minimization. After an SOP expression has been mapped, a minimum SOP expression is obtained by grouping the 1s and determining the minimum SOP expression from the map.

Grouping the 1s

You can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8, or 16 cells, which are all powers of two. In the case of a 3-variable map, $2^3 = 8$ cells is the maximum group.

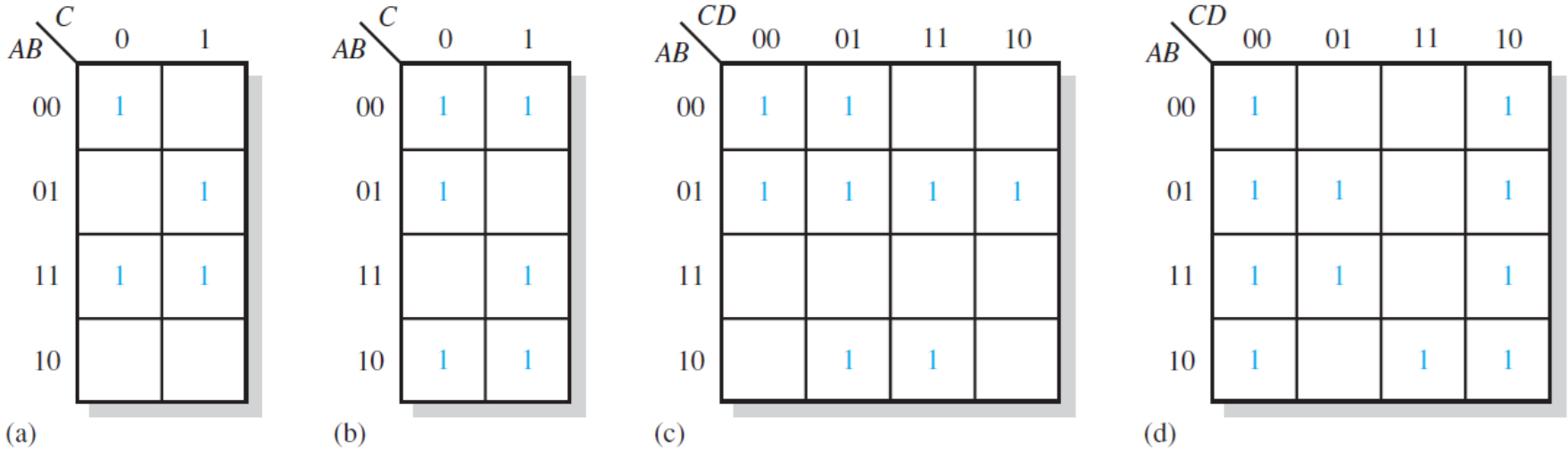
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.

3. Always include the largest possible number of 1s in a group in accordance with rule 1.

4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s

Group the 1s in each of the Karnaugh maps in Figure

Group the 1s in each of the Karnaugh maps in Figure 4–33.



Determining the Minimum SOP Expression from the Map

When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins.

The following rules are applied to find the minimum product terms and the minimum SOP expression:

1. Group the cells that have 1s. Each group of cells containing 1s creates one product term composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. Variables that occur both uncomplemented and complemented within the group are eliminated. These are called contradictory variables.

Determining the Minimum SOP Expression from the Map

2. Determine the minimum product term for each group.

(a) For a 3-variable map:

(1) A 1-cell group yields a 3-variable product term

(2) A 2-cell group yields a 2-variable product term

(3) A 4-cell group yields a 1-variable term

(4) An 8-cell group yields a value of 1 for the expression

Determining the Minimum SOP Expression from the Map

(b) For a 4-variable map:

(1) A 1-cell group yields a 4-variable product term

(2) A 2-cell group yields a 3-variable product term

(3) A 4-cell group yields a 2-variable product term

(4) An 8-cell group yields a 1-variable term

(5) A 16-cell group yields a value of 1 for the expression

3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

Karnaugh Map POS Minimization

In the last section, you studied the minimization of an SOP expression using a Karnaugh map. In this section, we focus on POS expressions. The approaches are much the same except that with POS expressions, 0s representing the standard sum terms are placed on the Karnaugh map instead of 1s.

steps

For a POS expression in standard form, a 0 is placed on the Karnaugh map for each sum term in the expression. Each 0 is placed in a cell corresponding to the value of a sum term.

The following steps and the illustration in Figure show the mapping process.

Step 1: Determine the binary value of each sum term in the standard POS expression. This is the binary value that makes the term equal to 0.

Step 2: As each sum term is evaluated, place a 0 on the Karnaugh map in the corresponding cell.

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms.

EXAMPLE

Use a Karnaugh map to minimize the following standard POS expression:

$$(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

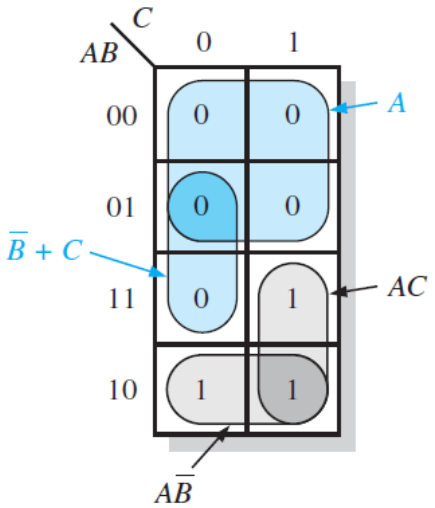
Also, derive the equivalent SOP expression.

Solution

The combinations of binary values of the expression are

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

Map the standard POS expression and group the cells as shown in Figure 4–45.



Use a Karnaugh map to minimize the following POS expression:

$$(B + C + D)(A + B + \overline{C} + D)(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + D)$$

Solution

The first term must be expanded into $\overline{A} + B + C + D$ and $A + B + C + D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in Figure 4–46. The sum term for each group is shown and the resulting minimum POS expression is

$$(C + D)(A + B + D)(\overline{A} + B + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.