

Digital Logic & Design (Theory)

Lecture No.1

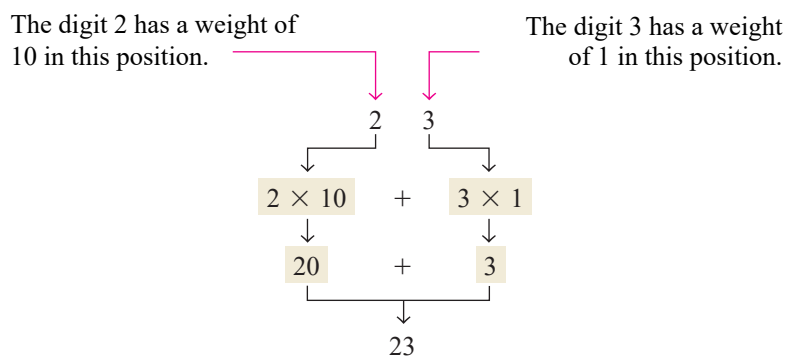
LECTURE OUTLINE

- 2-1 Decimal Numbers
- 2-2 Binary Numbers
- 2-3 Decimal-to-Binary Conversion
- 2-4 Binary Arithmetic
- 2-5 Complements of Binary Numbers
- 2-6 Signed Numbers

2-1 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. In this section, the structure of decimal numbers is reviewed. This review will help you more easily understand the structure of the binary number system, which is important in computers and digital electronics.

In the **decimal** number system each of the ten digits, 0 through 9, represents a certain quantity. As you know, the ten symbols (**digits**) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity. You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you use two or more digits, and the position of each digit within the number tells you the magnitude it represents. If, for example, you wish to express the quantity twenty-three, you use (by their respective positions in the number) the digit 2 to represent the quantity twenty and the digit 3 to represent the quantity three, as illustrated below.




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 $10^0 = 1$.

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

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with 10^{-1} .

$$10^2 \ 10^1 \ 10^0 \cdot 10^{-1} \ 10^{-2} \ 10^{-3} \ \dots$$



The value of a decimal number is the sum of the digits after each digit has been multiplied by its weight, as Examples 2-1 and 2-2 illustrate.

EXAMPLE 2-1

Express the decimal number 47 as a sum of the values of each digit.

Solution

The digit 4 has a weight of 10, which is 10^1 , as indicated by its position. The digit 7 has a weight of 1, which is 10^0 , as indicated by its position.

$$\begin{aligned} 47 &= (4 \times 10^1) + (7 \times 10^0) \\ &= (4 \times 10) + (7 \times 1) = \mathbf{40 + 7} \end{aligned}$$

EXAMPLE 2-2

Express the decimal number 568.23 as a sum of the values of each digit.

Solution

The whole number digit 5 has a weight of 100, which is 10^2 , the digit 6 has a weight of 10, which is 10^1 , the digit 8 has a weight of 1, which is 10^0 , the fractional digit 2 has a weight of 0.1, which is 10^{-1} , and the fractional digit 3 has a weight of 0.01, which is 10^{-2} .

$$\begin{aligned} 568.23 &= (5 \times 10^2) + (6 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2}) \\ &= (5 \times 100) + (6 \times 10) + (8 \times 1) + (2 \times 0.1) + (3 \times 0.01) \\ &= \mathbf{500 + 60 + 8 + 0.2 + 0.03} \end{aligned}$$

2-2 Binary Numbers

The binary number system is another way to represent quantities. It is less complicated than the decimal system because the binary system has only two digits. The decimal system with its ten digits is a base-ten system; 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 powers of two.

Counting in Binary

To learn to count in the binary system, first look at how you count in the decimal system. You start at zero and count up to nine before you run out of digits. You then start another digit position (to the left) and continue counting 10 through 99. At this point you have exhausted all two-digit combinations, so a third digit position is needed to count from 100 through 999.

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.

A binary count of zero through fifteen is shown in Table 2–1. Notice the patterns with which the 1s and 0s alternate in each column.

As you have seen in Table 2–1, four bits are required to count from zero to 15. In general, with n bits you can count up to a number equal to $2^n - 1$.

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

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

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 is a weighted number. The right-most bit is the **LSB** (least significant bit) in a binary whole number and 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 is the **MSB** (most significant bit); its weight depends on the size of the binary number.

Fractional numbers can also be represented in binary by placing bits to the right of the binary point, just as fractional decimal digits are placed to the right of the decimal point. The left-most bit is the MSB in a binary fractional number and has a weight of $2^{-1} = 0.5$. The fractional weights decrease from left to right by a negative power of two for each bit.

The weight structure of a binary number is

$$2^{n-1} \dots 2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} \dots 2^{-n}$$


 Binary point

where n is the number of bits from the binary point. Thus, all the bits to the left of the binary point have weights that are positive powers of two, as previously discussed for whole numbers. All bits to the right of the binary point have weights that are negative powers of two, or fractional weights.

The powers of two and their equivalent decimal weights for an 8-bit binary whole number and a 6-bit binary fractional number are shown in Table 2-2. Notice that the weight doubles for each positive power of two and that the weight is halved for each negative power of two. You can easily extend the table by doubling the weight of the most significant positive power of two and halving the weight of the least significant negative power of two; for example, $2^9 = 512$ and $2^{-7} = 0.0078125$.

TABLE 2-2
Binary weights.

Positive Powers of Two (Whole Numbers)									Negative Powers of Two (Fractional Number)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64
									0.5	0.25	0.125	0.625	0.03125	0.015625

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.

EXAMPLE 2-3

Convert the binary whole number 1101101 to decimal.

Solution

Determine the weight of each bit that is a 1, and then find the sum of the weights to get the decimal number.

$$\begin{aligned}
 \text{Weight: } & 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
 \text{Binary number: } & 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\
 1101101 = & 2^6 + 2^5 + 2^3 + 2^2 + 2^0 \\
 = & 64 + 32 + 8 + 4 + 1 = \mathbf{109}
 \end{aligned}$$

EXAMPLE 2-4

Convert the fractional binary number 0.1011 to decimal.

Solution

Determine the weight of each bit that is a 1, and then sum the weights to get the decimal fraction.

$$\begin{aligned}
 \text{Weight: } & 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \\
 \text{Binary number: } & 0 \ . \ 1 \ 0 \ 1 \ 1 \\
 0.1011 = & 2^{-1} + 2^{-3} + 2^{-4} \\
 = & 0.5 + 0.125 + 0.0625 = \mathbf{0.6875}
 \end{aligned}$$

2-3 Decimal-to-Binary Conversion

In Section 2-2 you learned how to convert a binary number to the equivalent decimal number. Now you will learn two ways of converting from a decimal number to a binary number.

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. An easy way to remember binary weights is that the lowest is 1, which is 2^0 , and that by doubling any weight, you get the next higher weight; thus, a list of seven binary weights would be 64, 32, 16, 8, 4, 2, 1 as you learned in the last section. The decimal number 9, for example, can be expressed as the sum of binary weights as follows:

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

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.

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 0 & 1 \end{array} \quad \text{Binary number for decimal 9}$$

EXAMPLE 2-5

Convert the following decimal numbers to binary:

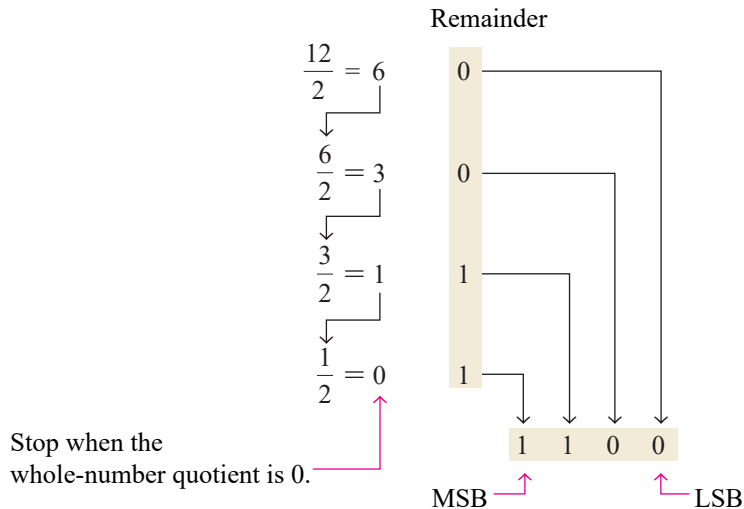
- (a) 12 (b) 25
(c) 58 (d) 82

Solution

- (a) $12 = 8 + 4 = 2^3 + 2^2 \longrightarrow 1100$
 (b) $25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \longrightarrow 11001$
 (c) $58 = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \longrightarrow 111010$
 (d) $82 = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \longrightarrow 1010010$

Repeated Division-by-2 Method

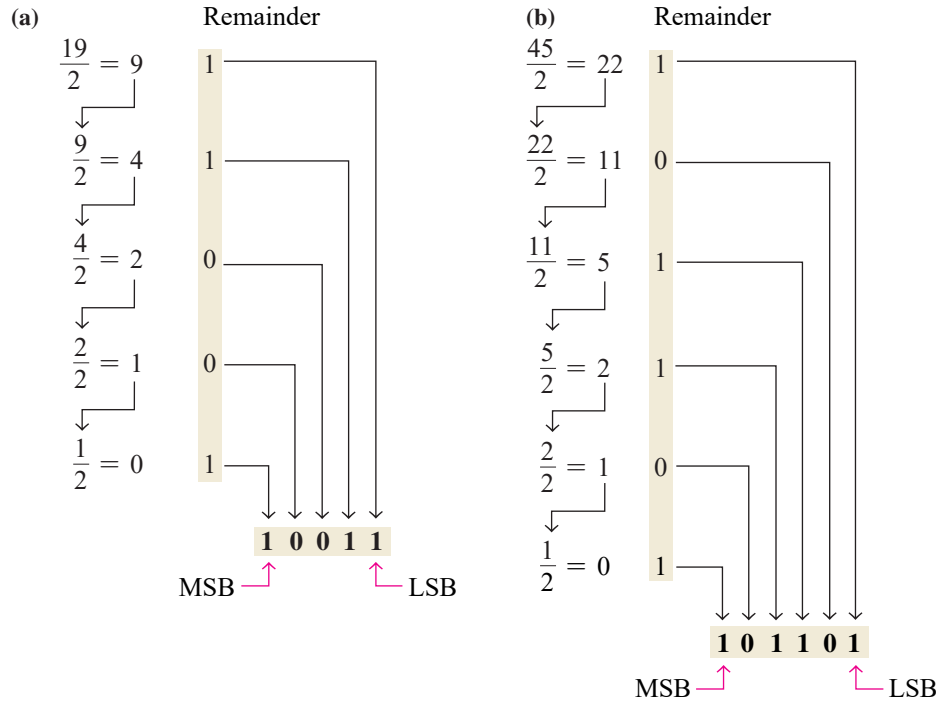
A systematic method of converting whole numbers from decimal to binary is the *repeated division-by-2* process. For example, to convert the decimal number 12 to binary, begin by dividing 12 by 2. Then divide each resulting quotient by 2 until there is a 0 whole-number quotient. The **remainders** generated by each division form the binary number. The first remainder to be produced is the LSB (least significant bit) in the binary number, and the last remainder to be produced is the MSB (most significant bit). This procedure is illustrated as follows for converting the decimal number 12 to binary.



EXAMPLE 2-6

Convert the following decimal numbers to binary:

- (a) 19 (b) 45

Solution**Converting Decimal Fractions to Binary**

Examples 2-5 and 2-6 demonstrated whole-number conversions. Now let's look at fractional conversions. An easy way to remember fractional binary weights is that the most significant weight is 0.5, which is 2^{-1} , and that by halving any weight, you get the next lower weight; thus a list of four fractional binary weights would be 0.5, 0.25, 0.125, 0.0625.

Sum-of-Weights

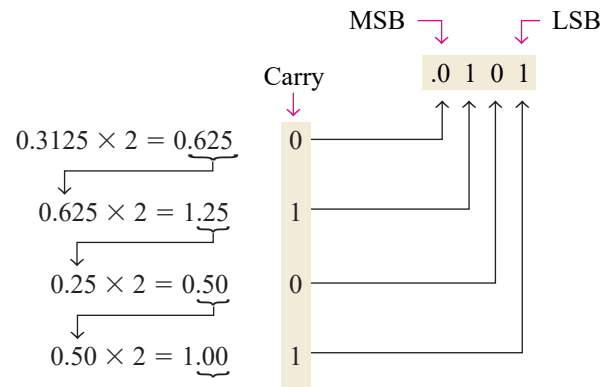
The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

$$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$$

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

As you have seen, decimal whole numbers can be converted to binary by repeated division by 2. Decimal fractions can be converted to binary by repeated multiplication by 2. For example, to convert the decimal fraction 0.3125 to binary, begin by multiplying 0.3125 by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero or until the desired number of decimal places is reached. The carry digits, or **carries**, generated by the multiplications produce the binary number. The first carry produced is the MSB, and the last carry is the LSB. This procedure is illustrated as follows:



Continue to the desired number of decimal places or stop when the fractional part is all zeros.

2-4 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. This section provides an introduction that will be expanded in later sections.

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). 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 $11 + 1$:

$$\begin{array}{r}
 \text{Carry} \quad \text{Carry} \\
 \begin{array}{r}
 1 \leftarrow 1 \leftarrow \\
 0 \quad 1 \quad 1 \\
 + 0 \quad 0 \quad 1 \\
 \hline
 1 \quad 0 \quad 0
 \end{array}
 \end{array}$$

In the right column, $1 + 1 = 0$ with a carry of 1 to the next column to the left. In the middle column, $1 + 1 + 0 = 0$ with a carry of 1 to the next column to the left. In the left column, $1 + 0 + 0 = 1$.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

Carry bits			
	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

EXAMPLE 2-7

Add the following binary numbers:

- (a) $11 + 11$ (b) $100 + 10$
 (c) $111 + 11$ (d) $110 + 100$

Solution

The equivalent decimal addition is also shown for reference.

(a)	11	3	(b)	100	4
	$+ 11$	$+ 3$		$+ 10$	$+ 2$
	110	6		110	6
(c)	111	7	(d)	110	6
	$+ 11$	$+ 3$		$+ 100$	$+ 4$
	1010	10		1010	10

Binary Subtraction

The four basic rules for subtracting bits are as follows:

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

$$10 - 1 = 1 \quad 0 - 1 \text{ with a borrow of } 1$$

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied. Examples 2-8 and 2-9 illustrate binary subtraction; the equivalent decimal subtractions are also shown.

EXAMPLE 2-8

Perform the following binary subtractions:

- (a) $11 - 01$ (b) $11 - 10$

Solution

(a)	11	3	(b)	11	3
	$- 01$	$- 1$		$- 10$	$- 2$
	10	2		01	1

No borrows were required in this example. The binary number 01 is the same as 1.

EXAMPLE 2-9

Subtract 011 from 101.

Solution

$$\begin{array}{r} 101 \quad 5 \\ -011 \quad -3 \\ \hline 010 \quad 2 \end{array}$$

Let's examine exactly what was done to subtract the two binary numbers since a borrow is required. Begin with the right column.

Left column:

When a 1 is borrowed,
a 0 is left, so $0 - 0 = 0$.

$$\begin{array}{r} 0 \\ 101 \\ -011 \\ \hline 010 \end{array}$$

Middle column:

Borrow 1 from next column
to the left, making a 10 in
this column, then $10 - 1 =$
1.

Right column:

$$1 - 1 = 0$$

Binary Multiplication

The four basic rules for multiplying bits are as follows:

$$\begin{array}{l} 0 \times 0 = 0 \\ 0 \times 1 = 0 \\ 1 \times 0 = 0 \\ 1 \times 1 = 1 \end{array}$$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive partial product left one place, and then adding all the partial products. Example 2–10 illustrates the procedure; the equivalent decimal multiplications are shown for reference.

EXAMPLE 2-10

Perform the following binary multiplications:

(a) 11×11 (b) 101×111

Solution

(a)	$\begin{array}{r} 11 \\ \times 11 \\ \hline 11 \\ +11 \\ \hline 1001 \end{array}$	(b)	$\begin{array}{r} 111 \\ \times 101 \\ \hline 111 \\ 000 \\ +111 \\ \hline 100011 \end{array}$
-----	---	-----	--

Binary Division

Division in binary follows the same procedure as division in decimal, as Example 2–11 illustrates. The equivalent decimal divisions are also given.

EXAMPLE 2-11

Perform the following binary divisions:

(a) $110 \div 11$ (b) $110 \div 10$

Solution

(a)	$\begin{array}{r} 10 \\ 11 \overline{)110} \\ \underline{11} \\ 000 \end{array}$	(b)	$\begin{array}{r} 11 \\ 10 \overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array}$
-----	--	-----	---

2-5 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

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Figure 2-2 for an 8-bit binary number.

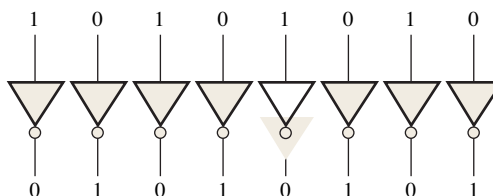


FIGURE 2-2 Example of inverters used to obtain the 1's complement of a binary number.

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.

$$2's \text{ complement} = (1's \text{ complement}) + 1$$

EXAMPLE 2-12

Find the 2's complement of 10110010.

Solution

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	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.

EXAMPLE 2-13

Find the 2's complement of 10111000 using the alternative method.

Solution

10111000	Binary number
01001000	2's complement
1's complements of original bits	

The 2's complement of a negative binary number can be realized using inverters and an adder, as indicated in Figure 2-3. This illustrates how an 8-bit number can be converted to its 2's complement by first inverting each bit (taking the 1's complement) and then adding 1 to the 1's complement with an adder circuit.

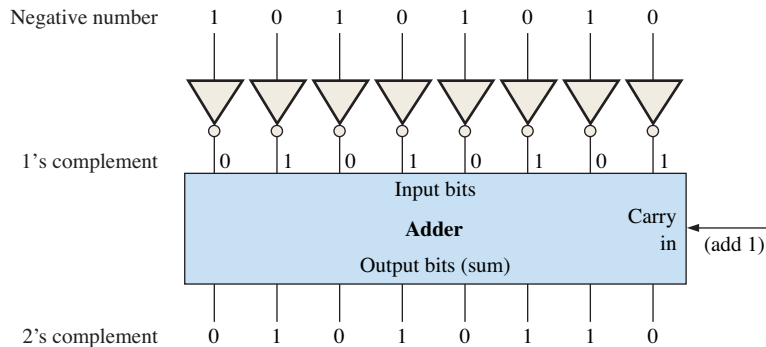


FIGURE 2-3 Example of obtaining the 2's complement of a negative binary number.

To convert from a 1's or 2's complement back to the true (uncomplemented) binary form, use the same two procedures described previously. To go from the 1's complement back to true binary, reverse all the bits. To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

2-6 Signed Numbers

Digital systems, such as the computer, must be able to handle both positive and negative numbers. A signed binary number consists of both sign and magnitude information. The sign indicates whether a number is positive or negative, and the magnitude is the value of the number. There are three forms in which signed integer (whole) numbers can be represented in binary: sign-magnitude, 1's complement, and 2's complement. Of these, the 2's complement is the most important and the sign-magnitude is the least used. Noninteger and very large or small numbers can be expressed in floating-point format.

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 left-most bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in 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

$$\begin{array}{c} 00011001 \\ \text{Sign bit} \uparrow \quad \uparrow \text{Magnitude bits} \end{array}$$

The decimal number -25 is expressed as

$$10011001$$

Notice that the only difference between +25 and -25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

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.

EXAMPLE 2-14

Express the decimal number -39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution

First, write the 8-bit number for $+39$.

$$00100111$$

In the *sign-magnitude form*, -39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

$$10100111$$

In the *1's complement form*, -39 is produced by taking the 1's complement of $+39$ (00100111).

$$11011000$$

In the *2's complement form*, -39 is produced by taking the 2's complement of $+39$ (00100111) as follows:

$$\begin{array}{r} 11011000 \quad 1's \text{ complement} \\ + \quad \quad 1 \\ \hline 11011001 \quad 2's \text{ complement} \end{array}$$

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.

EXAMPLE 2-15

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101.

Solution

The seven magnitude bits and their powers-of-two weights are as follows:

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is **-21**.

1's Complement

Decimal values of positive numbers in the 1'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. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

EXAMPLE 2-16

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 (b) 11101000

Solution

(a) The bits and their powers-of-two weights for the positive number are as follows:

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = \mathbf{+23}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of -2^7 or -128 .

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = \mathbf{-23}$$

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.

From these examples, you can see why the 2's complement form is preferred for representing signed integer numbers: To convert to decimal, it simply requires a summation of weights regardless of whether the number is positive or negative. The 1's complement system requires adding 1 to the summation of weights for negative numbers but not for positive numbers. Also, the 1's complement form is generally not used because two representations of zero (00000000 or 11111111) are possible.

Range of Signed Integer Numbers

We have used 8-bit numbers for illustration because the 8-bit grouping is common in most computers and has been given the special name **byte**. With one byte or eight bits, you can represent 256 different numbers. With two bytes or sixteen bits, you can represent 65,536 different numbers. With four bytes or 32 bits, you can represent 4.295×10^9 different numbers. The formula for finding the number of different combinations of n bits is

$$\text{Total combinations} = 2^n$$

For 2's complement signed numbers, the range of values for n -bit numbers is

$$\text{Range} = -(2^{n-1}) \text{ to } +(2^{n-1} - 1)$$

where in each case there is one sign bit and $n - 1$ magnitude bits. For example, with four bits you can represent numbers in 2's complement ranging from $-(2^3) = -8$ to $2^3 - 1 = +7$. Similarly, with eight bits you can go from -128 to $+127$, with sixteen bits you can go from $-32,768$ to $+32,767$, and so on. There is one less positive number than there are negative numbers because zero is represented as a positive number (all zeros).

EXAMPLE 2-17

Determine the decimal values of the signed binary numbers expressed in 2's complement:

- (a) 01010110 (b) 10101010

Solution

- (a) The bits and their powers-of-two weights for the positive number are as follows:

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

- (b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = \mathbf{-86}$$

Floating-Point Numbers

To represent very large **integer** (whole) numbers, many bits are required. There is also a problem when numbers with both integer and fractional parts, such as 23.5618, need to be represented. The floating-point number system, based on scientific notation, is capable of representing very large and very small numbers without an increase in the number of bits and also for representing numbers that have both integer and fractional components.

A **floating-point number** (also known as a *real number*) consists of two parts plus a sign. The **mantissa** is the part of a floating-point number that represents the magnitude of the number and is between 0 and 1. The **exponent** is the part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

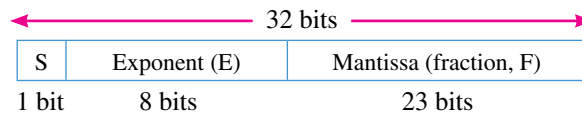
A decimal example will be helpful in understanding the basic concept of floating-point numbers. Let's consider a decimal number which, in integer form, is 241,506,800. The mantissa is .2415068 and the exponent is 9. When the integer is expressed as a floating-point number, it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten. The floating-point number is written as

$$0.2415068 \times 10^9$$

For binary floating-point numbers, the format is defined by ANSI/IEEE Standard 754-1985 in three forms: *single-precision*, *double-precision*, and *extended-precision*. These all have the same basic formats except for the number of bits. Single-precision floating-point numbers have 32 bits, double-precision numbers have 64 bits, and extended-precision numbers have 80 bits. We will restrict our discussion to the single-precision floating-point format.

Single-Precision Floating-Point Binary Numbers

In the standard format for a single-precision binary number, the sign bit (S) is the left-most bit, the exponent (E) includes the next eight bits, and the mantissa or fractional part (F) includes the remaining 23 bits, as shown next.



In the mantissa or fractional part, the binary point is understood to be to the left of the 23 bits. Effectively, there are 24 bits in the mantissa because in any binary number the left-most (most significant) bit is always a 1. Therefore, this 1 is understood to be there although it does not occupy an actual bit position.

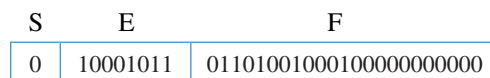
The eight bits in the exponent represent a *biased exponent*, which is obtained by adding 127 to the actual exponent. The purpose of the bias is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$.

To illustrate how a binary number is expressed in floating-point format, let's use 1011010010001 as an example. First, it can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of two.

$$1011010010001 = 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the sign bit (S) is 0. The exponent, 12, is expressed as a biased exponent by adding it to 127 ($12 + 127 = 139$). The biased exponent (E) is expressed as the binary number 10001011. The mantissa is the fractional part (F) of the binary number, .011010010001. Because there is always a 1 to the left of the binary point

in the power-of-two expression, it is not included in the mantissa. The complete floating-point number is



Next, let's see how to evaluate a binary number that is already in floating-point format. The general approach to determining the value of a floating-point number is expressed by the following formula:

$$\text{Number} = (-1)^S(1 + F)(2^{E-127})$$

To illustrate, let's consider the following floating-point binary number:

S	E	F
1	10010001	100011100010000000000000

The sign bit is 1. The biased exponent is 10010001 = 145. Applying the formula, we get

$$\begin{aligned} \text{Number} &= (-1)^1 (1.10001110001)(2^{145-127}) \\ &= (-1)(1.10001110001)(2^{18}) = -11000111000100000000 \end{aligned}$$

This floating-point binary number is equivalent to $-407,688$ in decimal. Since the exponent can be any number between -126 and $+128$, extremely large and small numbers can be expressed. A 32-bit floating-point number can replace a binary integer number having 129 bits. Because the exponent determines the position of the binary point, numbers containing both integer and fractional parts can be represented.

There are two exceptions to the format for floating-point numbers: The number 0.0 is represented by all 0s, and infinity is represented by all 1s in the exponent and all 0s in the mantissa.

EXAMPLE 2-18

Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

Solution

Convert the decimal number to binary.

$$3.248 \times 10^4 = 32480 = 111111011100000_2 = 1.11111011100000 \times 2^{14}$$

The MSB will not occupy a bit position because it is always a 1. Therefore, the mantissa is the fractional 23-bit binary number 11111011100000000000000 and the biased exponent is

$$14 + 127 = 141 = 10001101_2$$

The complete floating-point number is

0	10001101	11111011100000000000000
---	----------	-------------------------