Lecture No.2

LECTURE OUTLINE

2–5 Complements of Binary Numbers
2–6 Signed Numbers
2–7 Arithmetic Operations with Signed Numbers
2–8 Hexadecimal Numbers
2–9 Octal Numbers

LECTURE OBJECTIVES

- Determine the 1's and 2's complements of a binary number
- Express signed binary numbers in sign-magnitude, 1's complement, 2's complement, and floating-point format
- Carry out arithmetic operations with signed binary numbers
- Convert between the binary and hexadecimal number systems
- Add numbers in hexadecimal form
- Convert between the binary and octal number systems

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.

After completing this section, you should be able to

- Convert a binary number to its 1's complement
- Convert a binary number to its 2's complement using either of two methods

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.

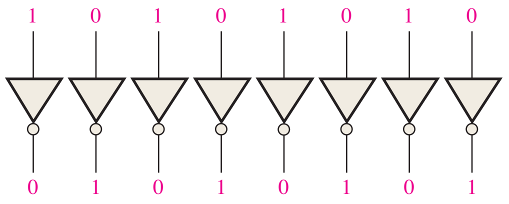


FIGURE 2–2 Example of inverters used to obtain the 1's complement of a binary number.

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.

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

EXAMPLE 2–12

Find the 2's complement of 10110010.

Solution

$$\begin{array}{rl}
10110010 & \text{Binary number} \\
01001101 & \text{1's complement} \\
+\qquad\quad 1 & \text{Add 1} \\
\hline
\mathbf{01001110} & \text{2's complement}
\end{array}$$

Related Problem

Determine the 2's complement of 11001011.

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

$$\begin{array}{rl}
10111000 & \text{Binary number} \\
\underbrace{\mathbf{0100}}_{\text{1's complements of original bits}}\underbrace{\mathbf{1000}}_{\text{These bits stay the same.}} & \text{2's complement}
\end{array}$$

Related Problem

Find the 2's complement of 11000000.

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.

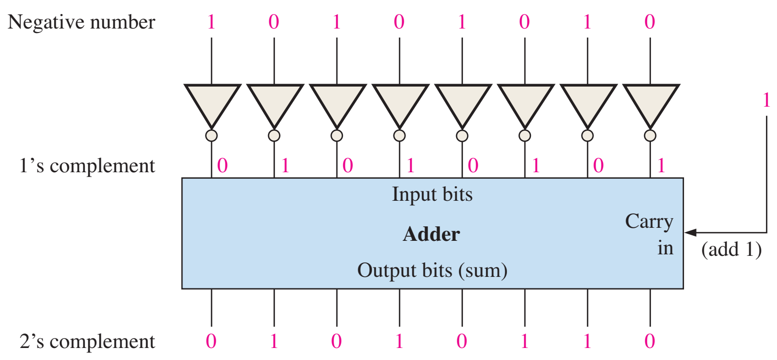


FIGURE 2–3 Example of obtaining the 2's complement of a negative binary number.

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.

SECTION 2–5 CHECKUP

1. Determine the 1's complement of each binary number:
   (a) 00011010 **(b)** 11110111 **(c)** 10001101
2. Determine the 2's complement of each binary number:
   (a) 00010110 **(b)** 11111100 **(c)** 10010001

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.

After completing this section, you should be able to

- Express positive and negative numbers in sign-magnitude
- Express positive and negative numbers in 1's complement
- Express positive and negative numbers in 2's complement
- Determine the decimal value of signed binary numbers
- Express a binary number 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

$$\underset{\text{Sign bit}\uparrow}{0}\underbrace{0011001}_{\uparrow\text{Magnitude bits}}$$

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

$$\mathbf{10100111}$$

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

$$\mathbf{11011000}$$

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

$$\begin{array}{rl} 11011000 & \text{1's complement} \\ +\qquad\quad 1 & \\ \hline \mathbf{11011001} & \text{2's complement} \end{array}$$

Related Problem

Express +19 and −19 as 8-bit numbers in sign-magnitude, 1's complement, and 2's complement.

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:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	1

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is $\mathbf{-21}$.

Related Problem

Determine the decimal value of the sign-magnitude number 01110111.

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:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

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.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Summing the weights where there are 1s,

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

Adding 1 to the result, the final decimal number is

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

Related Problem

Determine the decimal value of the 1's complement number 11101011.

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.

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:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

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 $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s,

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

Related Problem

Determine the decimal value of the 2's complement number 11010111.

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).

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.

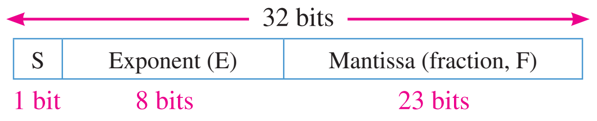
A decimal example will be helpful in understanding the basic concept of floating-point numbers. Let's consider a decimal number which, in integer form, is 241,506,800. The mantissa is .2415068 and the exponent is 9. When the integer is expressed as a floating-point number, it is normalized by moving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

S	E	F
0	10001011	01101001000100000000000

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	10001110001000000000000

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

$$\begin{aligned}\text{Number} &= (-1)^1\,(1.10001110001)(2^{145-127})\\ &= (-1)(1.10001110001)(2^{18}) = -1100011100010000000\end{aligned}$$

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**

Related Problem

Determine the binary value of the following floating-point binary number:

0 10011000 10000100010100110000000

SECTION 2–6 CHECKUP

1. Express the decimal number $+9$ as an 8-bit binary number in the sign-magnitude system.
2. Express the decimal number -33 as an 8-bit binary number in the 1's complement system.
3. Express the decimal number -46 as an 8-bit binary number in the 2's complement system.
4. List the three parts of a signed, floating-point number.

2–7 Arithmetic Operations with Signed Numbers

In the last section, you learned how signed numbers are represented in three different forms. 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.

After completing this section, you should be able to

- Add signed binary numbers
- Define *overflow*
- Explain how computers add strings of numbers
- Subtract signed binary numbers
- Multiply signed binary numbers using the direct addition method
- Multiply signed binary numbers using the partial products method
- Divide signed binary numbers

Addition

The two numbers in an addition are the **addend** and the **augend**. The result is the **sum**. 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. The equivalent decimal numbers are shown for reference.

Both numbers positive:

$$\begin{array}{r} 00000111 \\ +\ 00000100 \\ \hline 00001011 \end{array} \qquad \begin{array}{r} 7 \\ +\ 4 \\ \hline 11 \end{array}$$

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

Positive number with magnitude larger than negative number:

$$\begin{array}{rr} & 00001111 \\ & +\ 11111010 \\ \hline \text{Discard carry} \longrightarrow 1 & 00001001 \end{array} \qquad \begin{array}{r} 15 \\ +\ -6 \\ \hline 9 \end{array}$$

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

Negative number with magnitude larger than positive number:

$$\begin{array}{r} 00010000 \\ +\ 11101000 \\ \hline 11111000 \end{array} \qquad \begin{array}{r} 16 \\ +\ -24 \\ \hline -8 \end{array}$$

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

Both numbers negative:

$$\begin{array}{rr} & 11111011 \\ & +\ 11110111 \\ \hline \text{Discard carry} \longrightarrow 1 & 11110010 \end{array} \qquad \begin{array}{r} -5 \\ +\ -9 \\ \hline -14 \end{array}$$

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

In a computer, the negative numbers are stored in 2's complement form so, as you can see, the addition process is very simple: *Add the two numbers and discard any final carry bit.*

Overflow Condition

When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an **overflow** results as indicated by an incorrect sign bit. An overflow can occur only when both numbers are positive or both numbers are negative. If the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated. The following 8-bit example will illustrate this condition.

$$\begin{array}{r} 01111101 \\ +\,00111010 \\ \hline 1\underbrace{0110111} \end{array} \qquad \begin{array}{r} 125 \\ +\ \ 58 \\ \hline 183 \end{array}$$

Sign incorrect

Magnitude incorrect

In this example the sum of 183 requires eight magnitude bits. Since there are seven magnitude bits in the numbers (one bit is the sign), there is a carry into the sign bit which produces the overflow indication.

Numbers Added Two at a Time

Now let's look at the addition of a string of numbers, added two at a time. This can be accomplished by adding the first two numbers, then adding the third number to the sum of the first two, then adding the fourth number to this result, and so on. This is how computers add strings of numbers. The addition of numbers taken two at a time is illustrated in Example 2–19.

EXAMPLE 2–19

Add the signed numbers: 01000100, 00011011, 00001110, and 00010010.

Solution

The equivalent decimal additions are given for reference.

68	01000100	
+ 27	+ 00011011	Add 1st two numbers
95	01011111	1st sum
+ 14	+ 00001110	Add 3rd number
109	01101101	2nd sum
+ 18	+ 00010010	Add 4th number
127	**01111111**	Final sum

Related Problem

Add 00110011, 10111111, and 01100011. These are signed numbers.

Subtraction

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

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

For example, when you take the 2's complement of the positive number 00000100 (+4), you get 11111100, which is −4 as the following sum-of-weights evaluation shows:

$$-128 + 64 + 32 + 16 + 8 + 4 = -4$$

As another example, when you take the 2's complement of the negative number 11101101 (−19), you get 00010011, which is +19 as the following sum-of-weights evaluation shows:

$$16 + 2 + 1 = 19$$

Since subtraction is simply an addition with the sign of the subtrahend changed, the process is stated as follows:

To subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.

Example 2–20 illustrates the subtraction process.

EXAMPLE 2–20

Perform each of the following subtractions of the signed numbers:

(a) 00001000 − 00000011 **(b)** 00001100 − 11110111

(c) 11100111 − 00010011 **(d)** 10001000 − 11100010

Solution

Like in other examples, the equivalent decimal subtractions are given for reference.

(a) In this case, $8 - 3 = 8 + (-3) = 5$.

$$\begin{array}{rrl} & 00001000 & \text{Minuend } (+8) \\ & +\ 11111101 & \text{2's complement of subtrahend } (-3) \\ \hline \text{Discard carry} \longrightarrow & \mathbf{1\ \ 00000101} & \text{Difference } (+5) \end{array}$$

(b) In this case, $12 - (-9) = 12 + 9 = 21$.

$$\begin{array}{rl} 00001100 & \text{Minuend } (+12) \\ +\ 00001001 & \text{2's complement of subtrahend } (+9) \\ \hline 00010101 & \text{Difference } (+21) \end{array}$$

(c) In this case, $-25 - (+19) = -25 + (-19) = -44$.

$$\begin{array}{rrl} & 11100111 & \text{Minuend } (-25) \\ & +\ 11101101 & \text{2's complement of subtrahend } (-19) \\ \hline \text{Discard carry} & \mathbf{1\ \ 11010100} & \text{Difference } (-44) \end{array}$$

(d) In this case, $-120 - (-30) = -120 + 30 = -90$.

$$\begin{array}{rl} 10001000 & \text{Minuend } (-120) \\ +\ 00011110 & \text{2's complement of subtrahend } (+30) \\ \hline \mathbf{10100110} & \text{Difference } (-90) \end{array}$$

Related Problem

Subtract 01000111 from 01011000.

Multiplication

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

8	Multiplicand
× 3	Multiplier
24	Product

The multiplication operation in most computers is accomplished using addition. As you have already seen, subtraction is done with an adder; now let's see how multiplication is done.

Direct addition and *partial products* are two basic methods for performing multiplication using addition. In the direct addition method, you add the multiplicand a number of times equal to the multiplier. In the previous decimal example (8 × 3), three multiplicands are added: 8 + 8 + 8 = 24. The disadvantage of this approach is that it becomes very lengthy if the multiplier is a large number. For example, to multiply 350 × 75, you must add 350 to itself 75 times. Incidentally, this is why the term *times* is used to mean multiply.

When two binary numbers are multiplied, both numbers must be in true (uncomplemented) form. The direct addition method is illustrated in Example 2–21 adding two binary numbers at a time.

EXAMPLE 2–21

Multiply the signed binary numbers: 01001101 (multiplicand) and 00000100 (multiplier) using the direct addition method.

Solution

Since both numbers are positive, they are in true form, and the product will be positive. The decimal value of the multiplier is 4, so the multiplicand is added to itself four times as follows:

01001101	1st time
+ 01001101	2nd time
10011010	Partial sum
+ 01001101	3rd time
11100111	Partial sum
+ 01001101	4th time
100110100	Product

Since the sign bit of the multiplicand is 0, it has no effect on the outcome. All of the bits in the product are magnitude bits.

Related Problem

Multiply 01100001 by 00000110 using the direct addition method.

The partial products method is perhaps the more common one because it reflects the way you multiply longhand. The multiplicand is multiplied by each multiplier digit beginning with the least significant digit. The result of the multiplication of the multiplicand by a multiplier digit is called a *partial product.* Each successive partial product is moved (shifted) one place to the left and when all the partial products have been produced, they are added to get the final product. Here is a decimal example.

239	Multiplicand
× 123	Multiplier
717	1st partial product (3 × 239)
478	2nd partial product (2 × 239)
+ 239	3rd partial product (1 × 239)
29,397	Final product

The sign of the product of a multiplication depends on the signs of the multiplicand and the multiplier according to the following two rules:

- **If the signs are the same, the product is positive.**
- **If the signs are different, the product is negative.**

The basic steps in the partial products method of binary multiplication are as follows:

Step 1: Determine if the signs of the multiplicand and multiplier are the same or different. This determines what the sign of the product will be.

Step 2: Change any negative number to true (uncomplemented) form. Because most computers store negative numbers in 2's complement, a 2's complement operation is required to get the negative number into true form.

Step 3: Starting with the least significant multiplier bit, generate the partial products. When the multiplier bit is 1, the partial product is the same as the multiplicand. When the multiplier bit is 0, the partial product is zero. Shift each successive partial product one bit to the left.

Step 4: Add each successive partial product to the sum of the previous partial products to get the final product.

Step 5: If the sign bit that was determined in step 1 is negative, take the 2's complement of the product. If positive, leave the product in true form. Attach the sign bit to the product.

EXAMPLE 2–22

Multiply the signed binary numbers: 01010011 (multiplicand) and 11000101 (multiplier).

Solution

Step 1: The sign bit of the multiplicand is 0 and the sign bit of the multiplier is 1. The sign bit of the product will be 1 (negative).

Step 2: Take the 2's complement of the multiplier to put it in true form.

$$11000101 \longrightarrow 00111011$$

Step 3 and 4: The multiplication proceeds as follows. Notice that only the magnitude bits are used in these steps.

```
         1010011     Multiplicand
       × 0111011     Multiplier
         1010011     1st partial product
      + 1010011      2nd partial product
        11111001     Sum of 1st and 2nd
     + 0000000       3rd partial product
       011111001     Sum
    + 1010011        4th partial product
      1110010001     Sum
   + 1010011         5th partial product
    100011000001     Sum
  + 1010011          6th partial product
   1001100100001     Sum
 + 0000000           7th partial product
   1001100100001     Final product
```

Step 5: Since the sign of the product is a 1 as determined in step 1, take the 2's complement of the product.

$$1001100100001 \longrightarrow 0110011011111$$

Attach the sign bit

1 0110011011111

Related Problem

Verify the multiplication is correct by converting to decimal numbers and performing the multiplication.

Division

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

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

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

In this simple example, the divisor was subtracted from the dividend three times before a remainder of zero was obtained. Therefore, the quotient is 3.

The sign of the quotient depends on the signs of the dividend and the divisor according to the following two rules:

- **If the signs are the same, the quotient is positive.**
- **If the signs are different, the quotient is negative.**

When two binary numbers are divided, both numbers must be in true (uncomplemented) form. The basic steps in a division process are as follows:

Step 1: Determine if the signs of the dividend and divisor are the same or different. This determines what the sign of the quotient will be. Initialize the quotient to zero.

Step 2: Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive, go to step 3. If the partial remainder is zero or negative, the division is complete.

Step 3: Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.

Continue to subtract the divisor from the dividend and the partial remainders until there is a zero or a negative result. Count the number of times that the divisor is subtracted and you have the quotient. Example 2–23 illustrates these steps using 8-bit signed binary numbers.

EXAMPLE 2–23

Divide 01100100 by 00011001.

Solution

Step 1: The signs of both numbers are positive, so the quotient will be positive. The quotient is initially zero: 00000000.

Step 2: Subtract the divisor from the dividend using 2's complement addition (remember that final carries are discarded).

01100100	Dividend
+ 11100111	2's complement of divisor
01001011	Positive 1st partial remainder

Add 1 to quotient: 00000000 + 00000001 = 00000001.

Step 3: Subtract the divisor from the 1st partial remainder using 2's complement addition.

01001011	1st partial remainder
+ 11100111	2's complement of divisor
00110010	Positive 2nd partial remainder

Add 1 to quotient: 00000001 + 00000001 = 00000010.

Step 4: Subtract the divisor from the 2nd partial remainder using 2's complement addition.

00110010	2nd partial remainder
+ 11100111	2's complement of divisor
00011001	Positive 3rd partial remainder

Add 1 to quotient: 00000010 + 00000001 = 00000011.

Step 5: Subtract the divisor from the 3rd partial remainder using 2's complement addition.

00011001	3rd partial remainder
+ 11100111	2's complement of divisor
00000000	Zero remainder

Add 1 to quotient: 00000011 + 00000001 = **00000100** (final quotient). The process is complete.

Related Problem

Verify that the process is correct by converting to decimal numbers and performing the division.

SECTION 2–7 CHECKUP

1. List the four cases when numbers are added.
2. Add the signed numbers 00100001 and 10111100.
3. Subtract the signed numbers 00110010 from 01110111.
4. What is the sign of the product when two negative numbers are multiplied?
5. Multiply 01111111 by 00000101.
6. What is the sign of the quotient when a positive number is divided by a negative number?
7. Divide 00110000 by 00001100.

2–8 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. As you are probably aware, long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Since computers and microprocessors understand only 1s and 0s, it is necessary to use these digits when you program in "machine language." Imagine writing a sixteen bit instruction for a microprocessor system in 1s and 0s. It is much more efficient to use hexadecimal or octal; octal numbers are covered in Section 2–9. Hexadecimal is widely used in computer and microprocessor applications.

After completing this section, you should be able to

- List the hexadecimal characters
- Count in hexadecimal
- Convert from binary to hexadecimal
- Convert from hexadecimal to binary
- Convert from hexadecimal to decimal
- Convert from decimal to hexadecimal
- Add hexadecimal numbers
- Determine the 2's complement of a hexadecimal number
- Subtract hexadecimal numbers

The **hexadecimal** number system has a base of sixteen; that is, it is composed of 16 **numeric** and alphabetic **characters**. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient because each hexadecimal digit represents a 4-bit binary number (as listed in Table 2–3).

TABLE 2–3

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

Ten numeric digits and six alphabetic characters make up the hexadecimal number system. The use of letters A, B, C, D, E, and F to represent numbers may seem strange at first, but keep in mind that any number system is only a set of sequential symbols. If you understand what quantities these symbols represent, then the form of the symbols themselves is less important once you get accustomed to using them. We will use the subscript 16 to designate hexadecimal numbers to avoid confusion with decimal numbers. Sometimes you may see an "h" following a hexadecimal number.

Counting in Hexadecimal

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

. . ., 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, . . .

With two hexadecimal digits, you can count up to FF_{16}, which is decimal 255. To count beyond this, three hexadecimal digits are needed. For instance, 100_{16} is decimal 256, 101_{16} is decimal 257, and so forth. The maximum 3-digit hexadecimal number is FFF_{16}, or decimal 4095. The maximum 4-digit hexadecimal number is $FFFF_{16}$, which is decimal 65,535.

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.

EXAMPLE 2–24

Convert the following binary numbers to hexadecimal:

(a) 1100101001010111 **(b)** 111111000101101001

Solution

(a) $\underbrace{1100}_{\substack{\downarrow\\ C}}\underbrace{1010}_{\substack{\downarrow\\ A}}\underbrace{0101}_{\substack{\downarrow\\ 5}}\underbrace{0111}_{\substack{\downarrow\\ 7}} = \mathbf{CA57}_{16}$

(b) $\underbrace{0011}_{\substack{\downarrow\\ 3}}\underbrace{1111}_{\substack{\downarrow\\ F}}\underbrace{0001}_{\substack{\downarrow\\ 1}}\underbrace{0110}_{\substack{\downarrow\\ 6}}\underbrace{1001}_{\substack{\downarrow\\ 9}} = \mathbf{3F169}_{16}$

Two zeros have been added in part (b) to complete a 4-bit group at the left.

Related Problem

Convert the binary number 1001111011110011100 to hexadecimal.

Hexadecimal-to-Binary Conversion

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

EXAMPLE 2–25

Determine the binary numbers for the following hexadecimal numbers:

(a) $10A4_{16}$ **(b)** $CF8E_{16}$ **(c)** 9742_{16}

Solution

(a) $\overset{\substack{1\\ \downarrow}}{1}\overbrace{0000}^{\substack{0\\ \downarrow}}\overbrace{1010}^{\substack{A\\ \downarrow}}\overbrace{0100}^{\substack{4\\ \downarrow}}$

(b) $\overbrace{1100}^{\substack{C\\ \downarrow}}\overbrace{1111}^{\substack{F\\ \downarrow}}\overbrace{1000}^{\substack{8\\ \downarrow}}\overbrace{1110}^{\substack{E\\ \downarrow}}$

(c) $\overbrace{1001}^{\substack{9\\ \downarrow}}\overbrace{0111}^{\substack{7\\ \downarrow}}\overbrace{0100}^{\substack{4\\ \downarrow}}\overbrace{0010}^{\substack{2\\ \downarrow}}$

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

Related Problem

Convert the hexadecimal number 6BD3 to binary.

It should be clear that it is much easier to deal with a hexadecimal number than with the equivalent binary number. 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.

EXAMPLE 2–26

Convert the following hexadecimal numbers to decimal:

(a) $1C_{16}$ **(b)** $A85_{16}$

Solution

Remember, convert the hexadecimal number to binary first, then to decimal.

(a)
$$\overset{\overset{1}{\downarrow}}{\overbrace{0001}}\overset{\overset{C}{\downarrow}}{\overbrace{1100}} = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28}_{10}$$

(b)
$$\overset{\overset{A}{\downarrow}}{\overbrace{1010}}\overset{\overset{8}{\downarrow}}{\overbrace{1000}}\overset{\overset{5}{\downarrow}}{\overbrace{0101}} = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = \mathbf{2693}_{10}$$

Related Problem

Convert the hexadecimal number 6BD to decimal.

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

EXAMPLE 2–27

Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ **(b)** $B2F8_{16}$

Solution

Recall from Table 2–3 that letters A through F represent decimal numbers 10 through 15, respectively.

(a) $E5_{16} = (E \times 16) + (5 \times 1) = (14 \times 16) + (5 \times 1) = 224 + 5 = \mathbf{229}_{10}$

(b)
$$\begin{aligned} B2F8_{16} &= (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1) \\ &= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1) \\ &= 45{,}056 + 512 + 240 + 8 = \mathbf{45{,}816}_{10} \end{aligned}$$

Related Problem

Convert $60A_{16}$ to decimal.

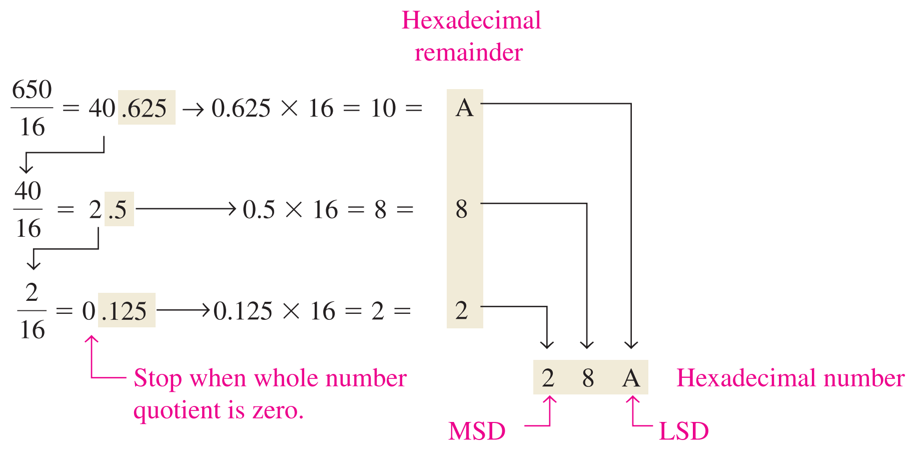
Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions. The first remainder produced is the least significant digit (LSD). Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number. This procedure is similar to repeated division by 2 for decimal-to-binary conversion that was covered in Section 2–3. Example 2–28 illustrates the procedure. Note that when a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

EXAMPLE 2–28

Convert the decimal number 650 to hexadecimal by repeated division by 16.

Solution



Related Problem

Convert decimal 2591 to hexadecimal.

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. When adding two hexadecimal numbers, use the following rules. (Decimal numbers are indicated by a subscript 10.)

1. In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal values. For instance, $5_{16} = 5_{10}$ and $C_{16} = 12_{10}$.
2. If the sum of these two digits is 15_{10} or less, bring down the corresponding hexadecimal digit.
3. If the sum of these two digits is greater than 15_{10}, bring down the amount of the sum that exceeds 16_{10} and carry a 1 to the next column.

EXAMPLE 2–29

Add the following hexadecimal numbers:

(a) $23_{16} + 16_{16}$ **(b)** $58_{16} + 22_{16}$ **(c)** $2B_{16} + 84_{16}$ **(d)** $DF_{16} + AC_{16}$

Solution

(a)
$$\begin{array}{r} 23_{16} \\ +\ 16_{16} \\ \hline \mathbf{39_{16}} \end{array}$$

right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$
left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$

(b) $\begin{array}{r} 58_{16} \\ +\,22_{16} \\ \hline \mathbf{7A_{16}} \end{array}$ right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$
left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$

(c) $\begin{array}{r} 2B_{16} \\ +\,84_{16} \\ \hline \mathbf{AF_{16}} \end{array}$ right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$
left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$

(d) $\begin{array}{r} DF_{16} \\ +\,AC_{16} \\ \hline \mathbf{18B_{16}} \end{array}$ right column: $F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10}$
$27_{10} - 16_{10} = 11_{10} = B_{16}$ with a 1 carry
left column: $D_{16} + A_{16} + 1_{16} = 13_{10} + 10_{10} + 1_{10} = 24_{10}$
$24_{10} - 16_{10} = 8_{10} = 8_{16}$ with a 1 carry

Related Problem

Add $4C_{16}$ and $3A_{16}$.

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.

Method 1: Convert the hexadecimal number to binary. Take the 2's complement of the binary number. Convert the result to hexadecimal. This is illustrated in Figure 2–4.

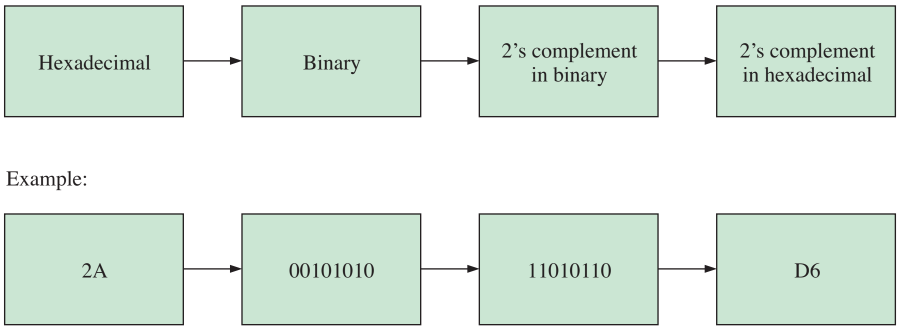


FIGURE 2–4 Getting the 2's complement of a hexadecimal number, Method 1.

Method 2: Subtract the hexadecimal number from the maximum hexadecimal number and add 1. This is illustrated in Figure 2–5.

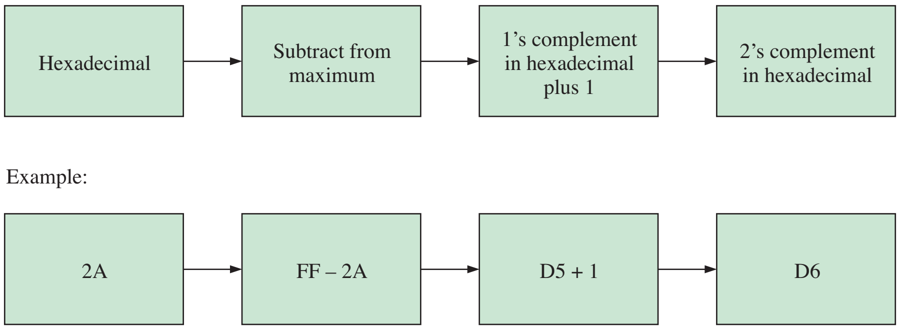

Hexadecimal
Subtract from maximum
1's complement in hexadecimal plus 1
2's complement in hexadecimal
Example:
2A
FF – 2A
D5 + 1
D6

FIGURE 2–5 Getting the 2's complement of a hexadecimal number, Method 2.

Method 3: Write the sequence of single hexadecimal digits. Write the sequence in reverse below the forward sequence. The 1's complement of each hex digit is the digit directly below it. Add 1 to the resulting number to get the 2's complement. This is illustrated in Figure 2–6.

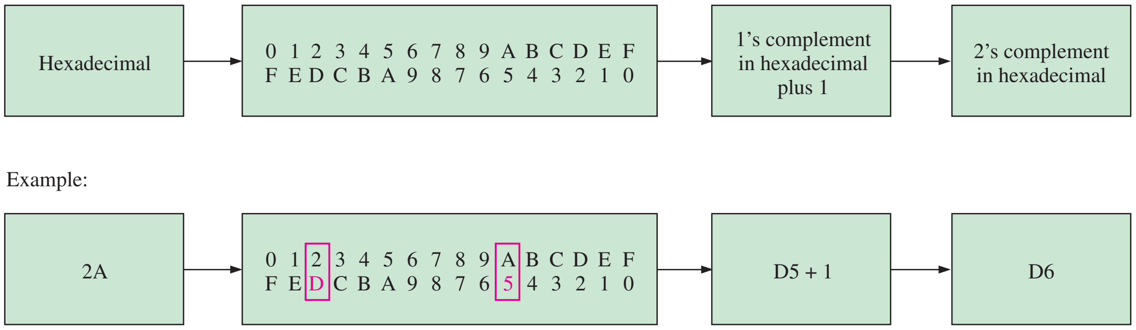

Hexadecimal
0 1 2 3 4 5 6 7 8 9 A B C D E F
F E D C B A 9 8 7 6 5 4 3 2 1 0
1's complement in hexadecimal plus 1
2's complement in hexadecimal
Example:
2A
0 1 2 3 4 5 6 7 8 9 A B C D E F
F E D C B A 9 8 7 6 5 4 3 2 1 0
D5 + 1
D6

FIGURE 2–6 Getting the 2's complement of a hexadecimal number, Method 3.

EXAMPLE 2–30

Subtract the following hexadecimal numbers:

(a) $84_{16} - 2A_{16}$ **(b)** $C3_{16} - 0B_{16}$

Solution

(a) $2A_{16} = 00101010$

2's complement of $2A_{16} = 11010110 = D6_{16}$ (using Method 1)

$$\begin{array}{rl} 84_{16} & \\ +\ D6_{16} & \text{Add} \\ \hline \cancel{1}5A_{16} & \text{Drop carry, as in 2's complement addition} \end{array}$$

The difference is $\mathbf{5A_{16}}$.

(b) $0B_{16} = 00001011$

2's complement of $0B_{16} = 11110101 = F5_{16}$ (using Method 1)

$$\begin{array}{rl} C3_{16} & \\ +\ F5_{16} & \text{Add} \\ \hline \cancel{1}B8_{16} & \text{Drop carry} \end{array}$$

The difference is $\mathbf{B8}_{16}$.

Related Problem

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

SECTION 2–8 CHECKUP

1. Convert the following binary numbers to hexadecimal:
   (a) 10110011 **(b)** 110011101000
2. Convert the following hexadecimal numbers to binary:
   (a) 57_{16} **(b)** $3A5_{16}$ **(c)** $F80B_{16}$
3. Convert $9B30_{16}$ to decimal.
4. Convert the decimal number 573 to hexadecimal.

5. Add the following hexadecimal numbers directly:

(a) $18_{16} + 34_{16}$ **(b)** $3F_{16} + 2A_{16}$

6. Subtract the following hexadecimal numbers:

(a) $75_{16} - 21_{16}$ **(b)** $94_{16} - 5C_{16}$

2–9 Octal Numbers

Like the hexadecimal number system, the octal number system provides a convenient way to express binary numbers and codes. However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes.

After completing this section, you should be able to

- Write the digits of the octal number system
- Convert from octal to decimal
- Convert from decimal to octal
- Convert from octal to binary
- Convert from binary to octal

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, . . .

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. For instance, 15_8 in octal is equivalent to 13_{10} in decimal and D in hexadecimal. Sometimes you may see an “o” or a “Q” following an octal number.

Octal-to-Decimal Conversion

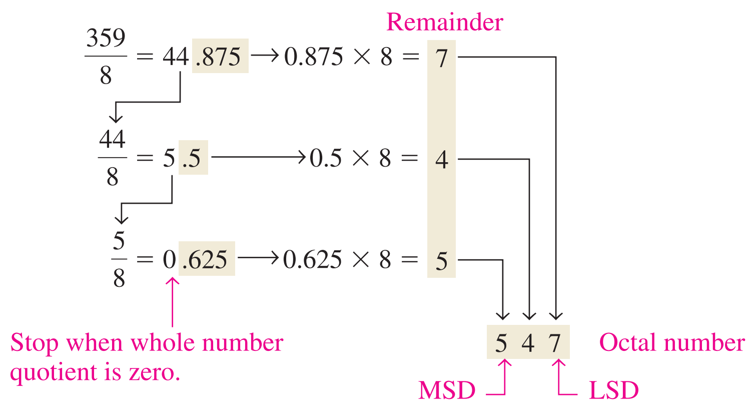
Since the octal number system has a base of eight, each successive digit position is an increasing power of eight, beginning in the right-most column with 8^0. The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for 2374_8.

$$\begin{array}{rl}\text{Weight:} & 8^3\ 8^2\ 8^1\ 8^0\\ \text{Octal number:} & 2\ \ \ 3\ \ \ 7\ \ \ 4\end{array}$$

$$\begin{aligned}2374_8 &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0)\\ &= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1)\\ &= 1024 + 192 + 56 + 4 = 1276_{10}\end{aligned}$$

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 or to hexadecimal. To show how it works, let’s convert the decimal number 359 to octal. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the least significant digit (LSD).



Remainder
$\frac{359}{8} = 44.875 \rightarrow 0.875 \times 8 = 7$
$\frac{44}{8} = 5.5 \rightarrow 0.5 \times 8 = 4$
$\frac{5}{8} = 0.625 \rightarrow 0.625 \times 8 = 5$
Stop when whole number quotient is zero.
5 4 7
Octal number
MSD
LSD

Octal-to-Binary Conversion

Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary. Each octal digit is represented by three bits as shown in Table 2–4.

TABLE 2–4

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.

EXAMPLE 2–31

Convert each of the following octal numbers to binary:

(a) 13_8 **(b)** 25_8 **(c)** 140_8 **(d)** 7526_8

Solution

(a) $\begin{array}{cc} 1 & 3 \\ \downarrow & \downarrow \\ \overbrace{\mathbf{001}} & \overbrace{\mathbf{011}} \end{array}$ **(b)** $\begin{array}{cc} 2 & 5 \\ \downarrow & \downarrow \\ \overbrace{\mathbf{010}} & \overbrace{\mathbf{101}} \end{array}$ **(c)** $\begin{array}{ccc} 1 & 4 & 0 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{\mathbf{001}} & \overbrace{\mathbf{100}} & \overbrace{\mathbf{000}} \end{array}$ **(d)** $\begin{array}{cccc} 7 & 5 & 2 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \overbrace{\mathbf{111}} & \overbrace{\mathbf{101}} & \overbrace{\mathbf{010}} & \overbrace{\mathbf{110}} \end{array}$

Related Problem

Convert each of the binary numbers to decimal and verify that each value agrees with the decimal value of the corresponding octal number.

Binary-to-Octal Conversion

Conversion of a binary number to an octal number is the reverse of the octal-to-binary 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. These leading zeros do not affect the value of the binary number.

EXAMPLE 2–32

Convert each of the following binary numbers to octal:

(a) 110101 **(b)** 101111001 **(c)** 100110011010 **(d)** 11010000100

Solution

(a) $\underbrace{110}_{\substack{\downarrow \\ 6}}\underbrace{101}_{\substack{\downarrow \\ 5}} = \mathbf{65_8}$

(b) $\underbrace{101}_{\substack{\downarrow \\ 5}}\underbrace{111}_{\substack{\downarrow \\ 7}}\underbrace{001}_{\substack{\downarrow \\ 1}} = \mathbf{571_8}$

(c) $\underbrace{100}_{\substack{\downarrow \\ 4}}\underbrace{110}_{\substack{\downarrow \\ 6}}\underbrace{011}_{\substack{\downarrow \\ 3}}\underbrace{010}_{\substack{\downarrow \\ 2}} = \mathbf{4632_8}$

(d) $\underbrace{011}_{\substack{\downarrow \\ 3}}\underbrace{010}_{\substack{\downarrow \\ 2}}\underbrace{000}_{\substack{\downarrow \\ 0}}\underbrace{100}_{\substack{\downarrow \\ 4}} = \mathbf{3204_8}$

Related Problem

Convert the binary number 1010101000111110010 to octal.

SECTION 2–9 CHECKUP

1. Convert the following octal numbers to decimal:

(a) 73_8 **(b)** 125_8

2. Convert the following decimal numbers to octal:

(a) 98_{10} **(b)** 163_{10}

3. Convert the following octal numbers to binary:

(a) 46_8 **(b)** 723_8 **(c)** 5624_8

4. Convert the following binary numbers to octal:

(a) 110101111 **(b)** 1001100010 **(c)** 10111111001