

Digital Logic & Design (Theory)

Lecture No.2

LECTURE OUTLINE

- 2-7 Arithmetic Operations with Signed Numbers
- 2-8 Hexadecimal Numbers
- 2-9 Octal Numbers

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.

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:

$$\begin{array}{r} 00001111 \quad 7 \\ + 00001000 \quad + 4 \\ \hline 00010111 \quad 11 \end{array}$$

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

Positive number with magnitude larger than negative number:

$$\begin{array}{r} 00001111 \quad 15 \\ + 11111010 \quad + -6 \\ \hline 10001001 \quad 9 \end{array}$$

Discard carry \rightarrow 1

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

Negative number with magnitude larger than positive number:

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

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

Both numbers negative:

$$\begin{array}{r}
 11111011 \quad -5 \\
 + 11110111 \quad + -9 \\
 \hline
 1 \quad 11110010 \quad -14
 \end{array}$$

Discard carry \longrightarrow 1

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.

$$\begin{array}{r}
 01111101 \quad 125 \\
 + 00111010 \quad + 58 \\
 \hline
 10110111 \quad 183
 \end{array}$$

Sign incorrect \longrightarrow (points to the 1 in the sign bit of the result)

Magnitude incorrect \longrightarrow (points to the 0110111 in the magnitude of the result)

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.

$$\begin{array}{r}
 68 \quad 01000100 \\
 + 27 \quad + 00011011 \quad \text{Add 1st two numbers} \\
 \hline
 95 \quad 01011111 \quad \text{1st sum} \\
 + 14 \quad + 00001110 \quad \text{Add 3rd number} \\
 \hline
 109 \quad 01101101 \quad \text{2nd sum} \\
 + 18 \quad + 00010010 \quad \text{Add 4th number} \\
 \hline
 127 \quad \mathbf{01111111} \quad \text{Final sum}
 \end{array}$$

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, $8 - 3 = 8 + (-3) = 5$.

$$\begin{array}{r} 00001000 \quad \text{Minuend (+8)} \\ + 11111101 \quad \text{2's complement of subtrahend (-3)} \\ \hline \text{Discard carry } \longrightarrow \mathbf{1\ 0000101} \quad \text{Difference (+5)} \end{array}$$

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

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

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

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

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

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

Multiplication

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

$$\begin{array}{r} 8 \quad \text{Multiplicand} \\ \times 3 \quad \text{Multiplier} \\ \hline 24 \quad \text{Product} \end{array}$$

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:

$$\begin{array}{r} 01001101 \quad \text{1st time} \\ + 01001101 \quad \text{2nd time} \\ \hline 10011010 \quad \text{Partial sum} \\ + 01001101 \quad \text{3rd time} \\ \hline 11100111 \quad \text{Partial sum} \\ + 01001101 \quad \text{4th time} \\ \hline \mathbf{100110100} \quad \text{Product} \end{array}$$

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.

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.

$$\begin{array}{r} 239 \quad \text{Multiplicand} \\ \times 123 \quad \text{Multiplier} \\ \hline 717 \quad \text{1st partial product (3} \times 239) \\ 478 \quad \text{2nd partial product (2} \times 239) \\ + 239 \quad \text{3rd partial product (1} \times 239) \\ \hline 29,397 \quad \text{Final product} \end{array}$$

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
\times 0111011	Multiplier
<u>1010011</u>	1st partial product
+ <u>1010011</u>	2nd partial product
11111001	Sum of 1st and 2nd
+ <u>0000000</u>	3rd partial product
011111001	Sum
+ <u>1010011</u>	4th partial product
1110010001	Sum
+ <u>1010011</u>	5th partial product
100011000001	Sum
+ <u>1010011</u>	6th partial product
1001100100001	Sum
+ <u>0000000</u>	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 011001101111

Attach the sign bit \downarrow

1 011001101111

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
<u>0</u>	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).

$$\begin{array}{r} 01100100 \quad \text{Dividend} \\ + 11100111 \quad \text{2's complement of divisor} \\ \hline 01001011 \quad \text{Positive 1st partial remainder} \end{array}$$

Add 1 to quotient: 00000000 + 00000001 = 00000001.

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

$$\begin{array}{r} 01001011 \quad \text{1st partial remainder} \\ + 11100111 \quad \text{2's complement of divisor} \\ \hline 00110010 \quad \text{Positive 2nd partial remainder} \end{array}$$

Add 1 to quotient: 00000001 + 00000001 = 00000010.

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

$$\begin{array}{r} 00110010 \quad \text{2nd partial remainder} \\ + 11100111 \quad \text{2's complement of divisor} \\ \hline 00011001 \quad \text{Positive 3rd partial remainder} \end{array}$$

Add 1 to quotient: 00000010 + 00000001 = 00000011.

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

$$\begin{array}{r} 00011001 \quad \text{3rd partial remainder} \\ + 11100111 \quad \text{2's complement of divisor} \\ \hline 00000000 \quad \text{Zero remainder} \end{array}$$

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

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.

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) 11111000101101001

Solution

(a) $\overbrace{1100}^C \overbrace{1010}^A \overbrace{0101}^5 \overbrace{0111}^7} = CA57_{16}$ (b) $\overbrace{0011}^3 \overbrace{1111}^F \overbrace{1000}^1 \overbrace{1010}^6 \overbrace{1001}^9} = 3F169_{16}$

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

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

$$\begin{array}{ccc}
 \text{(a)} & \begin{array}{cccc} 1 & 0 & A & 4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \underbrace{0001} & \underbrace{0100} & \underbrace{1001} & \underbrace{0100} \\ 1000010100100 \end{array} & \text{(b)} & \begin{array}{cccc} C & F & 8 & E \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \underbrace{1100} & \underbrace{1111} & \underbrace{1000} & \underbrace{1110} \\ 1100111110001110 \end{array} & \text{(c)} & \begin{array}{cccc} 9 & 7 & 4 & 2 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \underbrace{1001} & \underbrace{0111} & \underbrace{1010} & \underbrace{0001} \\ 1001011110100010 \end{array}
 \end{array}$$

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

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.

$$\begin{array}{l}
 \text{(a)} \quad \begin{array}{cc} 1 & C \\ \downarrow & \downarrow \\ \underbrace{0001} & \underbrace{1100} \\ 00011100 = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = 28_{10} \end{array} \\
 \text{(b)} \quad \begin{array}{ccc} A & 8 & 5 \\ \downarrow & \downarrow & \downarrow \\ \underbrace{1010} & \underbrace{1000} & \underbrace{0101} \\ 101010000101 = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = 2693_{10} \end{array}
 \end{array}$$

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

$$\begin{array}{cccc}
 16^3 & 16^2 & 16^1 & 16^0 \\
 4096 & 256 & 16 & 1
 \end{array}$$

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

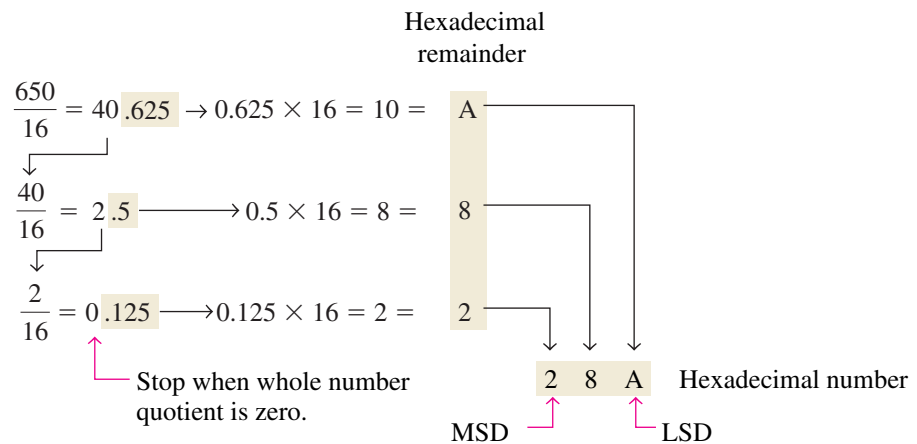
(b) $B2F8_{16} = (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1)$
 $= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1)$
 $= 45,056 + 512 + 240 + 8 = 45,816_{10}$

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

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

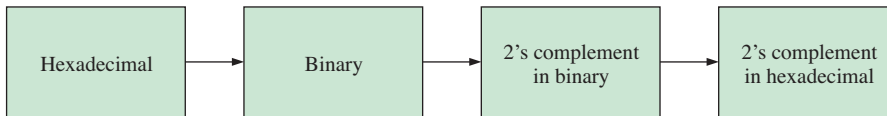
$\begin{array}{r} \text{(a)} \quad 23_{16} \\ + 16_{16} \\ \hline 39_{16} \end{array}$	right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$ left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$
$\begin{array}{r} \text{(b)} \quad 58_{16} \\ + 22_{16} \\ \hline 7A_{16} \end{array}$	right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$ left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$
$\begin{array}{r} \text{(c)} \quad 2B_{16} \\ + 84_{16} \\ \hline AF_{16} \end{array}$	right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$ left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$
$\begin{array}{r} \text{(d)} \quad DF_{16} \\ + AC_{16} \\ \hline 18B_{16} \end{array}$	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

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.



Example:

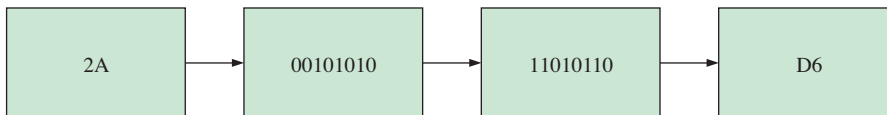
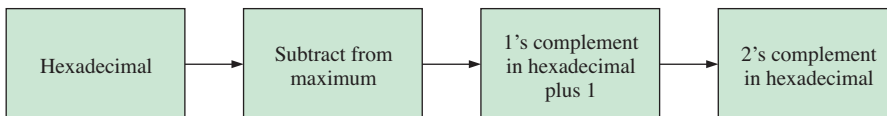


FIGURE 2-4 Getting the 2's complement of a hexadecimal number, Method 1.

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



Example:

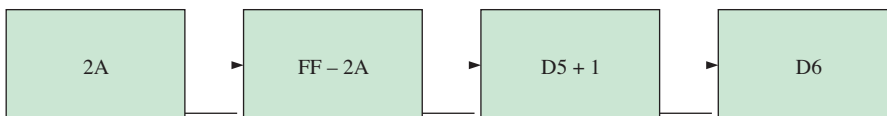


FIGURE 2-5 Getting the 2's complement of a hexadecimal number, Method 2.

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.

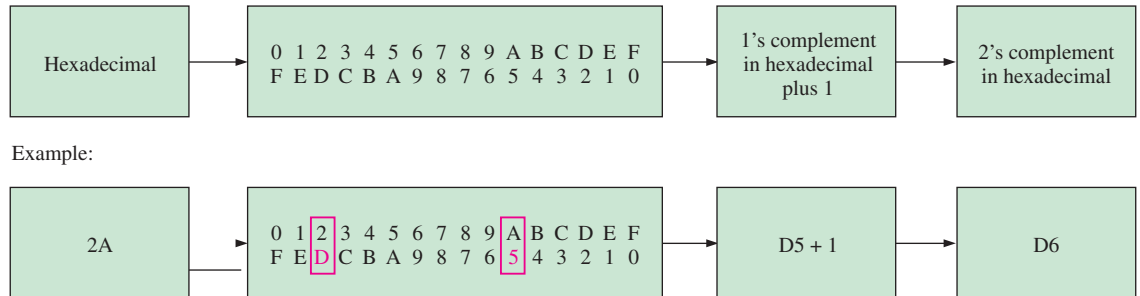


FIGURE 2-6 Getting the 2's complement of a hexadecimal number, Method 3.

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)

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

The difference is $5A_{16}$.

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

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

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

The difference is $B8_{16}$.

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.

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:

$$10, 11, 12, 13, 14, 15, 16, 17, 20, 21, \dots$$

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.

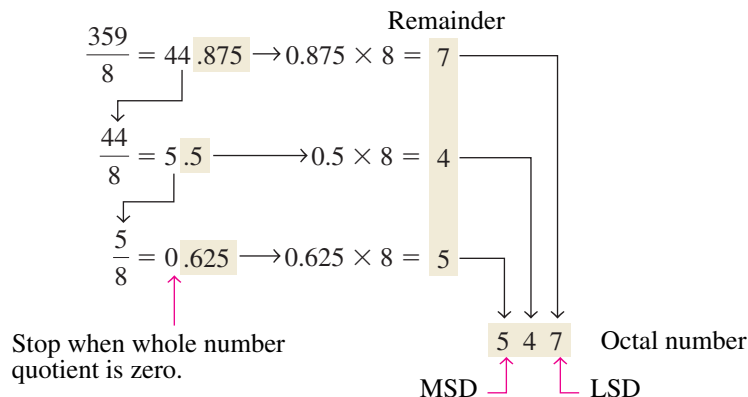
Octal-to-Decimal Conversion

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 .

$$\begin{array}{r}
 \text{Weight: } 8^3 \ 8^2 \ 8^1 \ 8^0 \\
 \text{Octal number: } 2 \ 3 \ 7 \ 4 \\
 2374_8 = (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\
 = (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\
 = 1024 + 192 + 56 + 4 = 1276_{10}
 \end{array}$$

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).



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

$$\begin{array}{cccc}
 \text{(a)} & \begin{array}{c} 1 \quad 3 \\ \downarrow \downarrow \\ \underbrace{001011} \end{array} & \text{(b)} & \begin{array}{c} 2 \quad 5 \\ \downarrow \downarrow \\ \underbrace{010101} \end{array} & \text{(c)} & \begin{array}{c} 1 \quad 4 \quad 0 \\ \downarrow \downarrow \downarrow \\ \underbrace{001100000} \end{array} & \text{(d)} & \begin{array}{c} 7 \quad 5 \quad 2 \quad 6 \\ \downarrow \downarrow \downarrow \downarrow \\ \underbrace{11110101110} \end{array}
 \end{array}$$

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

$$\begin{array}{cc}
 \text{(a)} & \begin{array}{c} 110101 \\ \downarrow \downarrow \\ 6 \quad 5 = 65_8 \end{array} & \text{(b)} & \begin{array}{c} 101111001 \\ \downarrow \downarrow \downarrow \\ 5 \quad 7 \quad 1 = 571_8 \end{array} \\
 \text{(c)} & \begin{array}{c} 100110011010 \\ \downarrow \downarrow \downarrow \downarrow \\ 4 \quad 6 \quad 3 \quad 2 = 4632_8 \end{array} & \text{(d)} & \begin{array}{c} 011010000100 \\ \downarrow \downarrow \downarrow \downarrow \\ 3 \quad 2 \quad 0 \quad 4 = 3204_8 \end{array}
 \end{array}$$