

Review of Number Systems

Binary, Octal, and Hexadecimal Numbers and Two's Complement

Topic 1: Binary, Octal, and Hexadecimal Numbers

The number system we generally use in our everyday lives is a decimal place value system; that is, it is based on powers of ten: 1, 10, 100, 1000, etc. In the field of computer science, however, it is often useful to represent numbers in binary, octal, or hexadecimal notation.

Table 1 below compares how the numbers from 0 through 24 are expressed in each of these number systems. Notice that, in decimal notation, we use ten different digits to express numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. In binary notation, we use just two different digits: 0 and 1. In octal notation, we use eight digits: 0, 1, 2, 3, 4, 5, 6, and 7; and in hexadecimal notation, we must come up with sixteen different digits to use: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Table 1

Counting to 24

Decimal	Binary	Octal	Hexadecimal
0	00000	0	0
1	00001	1	1
2	00010	2	2
3	00011	3	3
4	00100	4	4
5	00101	5	5
6	00110	6	6
7	00111	7	7
8	01000	10	8
9	01001	11	9
10	01010	12	A
11	01011	13	B
12	01100	14	C
13	01101	15	D
14	01110	16	E
15	01111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18

Table 2 below shows the powers of 2, 8, 10, and 16 upon which the place values within the binary, octal, decimal, and hexadecimal number system are based.

Table 2

Powers					
Bases	0	1	2	3	4
Binary	1	2	4	8	16
Octal	1	8	64	512	4096
Decimal	1	10	100	1000	10,000
Hexadecimal	1	16	256	4096	65,536

Example 1: Convert the binary number $10101101111111_{(2)}$ first to an octal number and then to a decimal number. (The subscripts are used to make clear which base system the number is being expressed in.)

First, group the digits of the binary number into sets of three, working from right to left. When there are not enough digits to complete a set of three, insert 0s on the left as in this example. When each of these sets of three digits is converted to its decimal equivalent, the result is an octal number.

$$\begin{array}{r}
 10\ 101\ 101\ 111\ 111_{(2)} \rightarrow 010\quad 101\quad 101\quad 111\quad 111_{(2)} \\
 \rightarrow 2\quad 5\quad 5\quad 7\quad 7_{(8)}
 \end{array}$$

Note that we evaluate the grouping $111_{(2)}$ as follows:

$$(1 * 2^2) + (1 * 2^1) + (1 * 2^0) = 4 + 2 + 1 = 7$$

We evaluate the grouping $101_{(2)}$ as follows:

$$(1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 4 + 0 + 1 = 5$$

We evaluate the grouping $010_{(2)}$ as follows:

$$(0 * 2^2) + (1 * 2^1) + (0 * 2^0) = 0 + 2 + 0 = 2$$

The octal number $25,577_{(8)}$ means:

$$\begin{array}{r}
 (2 * 8^4) + (5 * 8^3) + (5 * 8^2) + (7 * 8^1) + (7 * 8^0) = \\
 (2 * 4096) + (5 * 512) + (5 * 64) + (7 * 8) + (7 * 1) = \\
 11135_{(10)}
 \end{array}$$

Of course, the decimal number $11135_{(10)}$ means:

$$(1 * 10^4) + (1 * 10^3) + (1 * 10^2) + (3 * 10^1) + (5 * 10^0)$$

Thus, we see that the same number can be represented in three different number bases as follows:

$$10101101111111_{(2)} = 25,577_{(8)} = 11135_{(10)}$$

Example 2: Convert the binary number $10101101111111_{(2)}$ to a hexadecimal number.

This time, we first group the digits of the binary number into sets of four, working from right to left. Again, because there are not enough digits to complete a set of four, we insert 0s on the left in the final grouping. When each of these sets of four digits is converted to its decimal equivalent, the result is a hexadecimal number.

$$\begin{array}{rcccc} 10\ 1011\ 0111\ 1111_{(2)} & \rightarrow & 0010 & 1011 & 0111 & 1111_{(2)} \\ & & \rightarrow & 2 & 11 & 7 & 15 \\ & & \rightarrow & 2 & \mathbf{B} & 7 & \mathbf{F}_{(16)} \end{array}$$

Referring to Table 1 and Table 2, if necessary, we see that the hexadecimal number $2B7F_{(16)}$ means:

$$\begin{array}{rcl} (2 * 16^3) + (11 * 16^2) + (7 * 16^1) + (15 * 16^0) & = & \\ (2 * 4096) + (11 * 256) + (7 * 16) + (15 * 1) & = & \\ 11135_{(10)} & & \end{array}$$

Example 3: Convert the decimal number 41 to binary representation.

This is the reverse operation from what we did in Example 1. One way to accomplish this task is by dividing 41 by 2, noting the remainder, dividing the quotient by 2, noting the remainder, etc., until we finally achieve a quotient of 0:

	Remainder
$41 \div 2 = 20, R 1$	1
$20 \div 2 = 10, R 0$	0
$10 \div 2 = 5, R 0$	0
$5 \div 2 = 2, R 1$	1
$2 \div 2 = 1, R 0$	0
$1 \div 2 = 0, R 1$	1

Writing the remainders left-to-right, in the reverse of the order in which they were obtained will now form the binary representation:

$101001_{(2)}$

Another method that can be used to convert the number 41 from decimal representation to binary is to first divide 41 by the largest possible power of 2 and note the quotient; then divide the remainder by the next largest power of 2 and note the quotient; etc. Continue this process until the remainder is 0. The highest power of 2 that can be divided into 41 is $2^5 = 32$, so the process is as follows:

	Quotient
$41 \div 32 = 1, R 9$	1
$9 \div 16 = 0, R 9$	0
$9 \div 8 = 1, R 1$	1
$1 \div 4 = 0, R 1$	0
$1 \div 2 = 0, R 1$	0
$1 \div 1 = 1, R 0$	1

Using this method, we will write the digits obtained as quotients from left-to-right, in the order in which they were obtained:

$101001_{(2)}$

As a check, notice that: $101001_{(2)} =$

$$\begin{aligned} (1 * 2^5) + (0 * 2^4) + (1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (1 * 2^0) &= \\ (1 * 32) + (0 * 16) + (1 * 8) + (0 * 4) + (0 * 2) + (1 * 1) &= \\ 41 & \end{aligned}$$

Example 4: Convert 163 from decimal representation to octal representation.

We will adapt the first method shown in Example 3 to make this conversion. This time, however, we will repeatedly divide by 8.

	Remainder
$163 \div 8 = 20, R 3$	3
$20 \div 8 = 2, R 4$	4
$2 \div 8 = 0, R 2$	2

$$163_{(10)} = 243_{(8)}$$

Note that the second method illustrated in Example 3 could also have been adapted and used in this problem.

Example 5: Convert 315 from decimal representation to hexadecimal representation.

	Remainder
$315 \div 16 = 19, R 11$	11
$19 \div 16 = 1, R 3$	3
$1 \div 16 = 0, R 1$	1

$$315_{(10)} = 13B_{(16)}$$

Table 3 below can be used to add binary numbers:

Table 3

+	0	1
0	0	1
1	1	10

Example 6: The three problems below illustrate addition of binary numbers.

$$\begin{array}{r} \text{a)} \quad 1\ 0\ 0\ 1 \\ + \quad 1 \\ \hline 1\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} \text{b)} \quad 1\ 0\ 1\ 1 \\ + 1\ 0\ 1 \\ \hline 1\ 0\ 0\ 0\ 0 \end{array}$$

$$\begin{array}{r} \text{c)} \quad 1\ 0\ 0\ 1\ 1 \\ + 1\ 0\ 0\ 1 \\ \hline 1\ 1\ 1\ 0\ 0 \end{array}$$

Continue your study of this section now by proceeding to the "Practice Exercises," #s 1-37.

Topic 2: Two's Complement Representation

Two's complement notation is used to represent negative and positive binary numbers. Each number is represented in a fixed number of bits (e.g., eight). The high order bit (HOB), the bit furthest to the left, is 0 for positive numbers and 1 for negative numbers. The remaining bits are used to represent the absolute values of the numbers.

Coding of values using two's complement notation is done as follows:

Nonnegative values are represented by the bit pattern that corresponds to their base two representations (with 0 in the HOB).

Coding a negative value consists of first coding the corresponding positive value, then negating it by forming its complement, and then adding one.

The complement of a bit pattern is formed by changing the 0s in the pattern to 1s and the 1s to 0s.

Thus, to code -4 in a four-bit two's complement system, we would:

Write the pattern 0100, which is the binary representation for 4;

Complement it to obtain 1011; and

Add one to get 1100.

Example 7: Represent -5 in binary notation, using eight bits.

First consider how to represent positive 5 in binary notation: 00000101.

Now complement that pattern by reversing the 0s and 1s: 11111010.

Finally, add 1: $11111010 + 1 = 11111011$. This pattern represents -5 in a fixed eight-bit two's complement system.

To decode two's complement representations, we must first determine whether or not the value in question is negative. If the HOB is 0, the value is nonnegative and is obtained by reading the bit pattern as if it were a binary number. If, however, the HOB is 1, we negate the pattern by means of the complement and increment process, decode this positive value, and place a negative sign in front of the result.

Example 8: Decode the bit pattern 1010.

Because the HOB is 1, the value represented is negative. Therefore, we negate the pattern by complementing it to get 0101 and then increment it to get 0110. This result is the binary representation of 6; thus the original two's complement pattern 1010 represents the value -6.

Table 1 below illustrates how the values from -8 to 7 are represented in a four-bit system. Notice that the largest value that can be represented in four-bit two's complement notation is 7. Notice, too, that -8 is the smallest value that can be represented in such a system.

Table 1 -- Two's Complement Notation Using Patterns of Length Four

Bit Pattern	Value Represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Adding values represented in two's complement notation is the same process as regular binary addition except that all bit patterns, including the answer, are the same fixed length. This means that when adding in a two's complement system, any extra bit generated on the left of the answer by a final carry must be truncated. Thus, adding 0101 and 0010 would produce 0111, and adding 0111 and 1011 would result in 0010 (0111 + 1011 = 10010, which would be truncated to 0010).

Example 9: Find the following sums in a fixed four-bit two's complement system:

$$\begin{array}{r}
 \text{a)} \quad 3 \quad \rightarrow \quad 0011 \\
 +2 \quad \rightarrow \quad +0010 \\
 \hline
 \quad \quad \quad 0101 \rightarrow 5
 \end{array}$$

$$\begin{array}{r} \text{b)} \quad (-3) \\ \underline{+(-2)} \end{array} \rightarrow \begin{array}{r} 1101 \\ \underline{+1110} \\ 1011 \end{array} \rightarrow -5$$

$$\begin{array}{r}
 \text{c) } \quad 7 \\
 \underline{+(-5)} \rightarrow \quad 0111 \\
 \quad \quad \quad \quad \quad \quad \underline{+1011} \\
 \quad \quad \quad \quad \quad \quad 0010 \rightarrow \quad 2
 \end{array}$$

The operation of subtraction can be defined in terms of addition: $A - B = A + (-B)$. The advantage of two's complement notation is that addition of any combination of signed numbers (within the size limitations imposed by the fixed bit length) can be accomplished using the same process. Rather than needing to know how to add and subtract, a machine using two's complement notation needs only to know how to add.

Example 10: Find the difference $7 - 5$.

The subtraction problem $7 - 5$ is the same as the addition problem $7 + (-5)$. Thus,

$$\begin{array}{r}
 7 \\
 \underline{-5} \rightarrow \quad 0111 \\
 \quad \quad \quad \quad \quad \quad \underline{-0101} \rightarrow \quad 0111 \\
 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \underline{+1011} \\
 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad 0010 \rightarrow 2
 \end{array}$$

When using two's complement notation, a problem of overflow can occur. For example, in a four-bit two's complement system, the value 9 cannot be expressed. Thus, we could not obtain the correct answer to the problem $5 + 4$. In fact, the result would appear as -7 .

Example 11: Find the sum $5 + 4$ in four-bit two's complement notation.

$$\begin{array}{r}
 5 \\
 \underline{+4} \rightarrow \quad 0101 \\
 \quad \quad \quad \quad \quad \quad \underline{+0100} \\
 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad 1001
 \end{array}$$

Since a 1 appears in the HOB, the answer string represents a negative number. We decode it as follows:

Complement the string to obtain: 0110.

Add one to get: 0111.

This result is the binary representation of 7, so we interpret 1001 to represent -7 . This, of course, is incorrect.

A similar problem would arise if we were restricted to five bits and tried to represent the value 17. Try it.

Example 12: Find the following sums in a fixed eight-bit two's complement system:

$$\begin{array}{r}
 117 \\
 \underline{+88} \rightarrow \quad 01110101 \\
 \quad \quad \quad \quad \quad \quad \underline{+01011000}
 \end{array}$$

11001101

We recognize a problem here. The 1 in the HOB of the sum indicates a negative value; however, we know that the sum of two positive values is positive. $177 + 88 = 205$, and 205 is beyond the range of values that can be represented in eight-bit two's complement notation.

$$\begin{array}{r} -100 \qquad 10011100 \\ + (-60) \rightarrow + 11000100 \\ \hline 101100000 \rightarrow 01100000 \end{array}$$

Because we are restricted to eight bits, the final 1 that is carried is truncated, and the answer is found to be 01100000. The 0 in the HOB indicates this is a positive value. We know, however, that the sum of two negative numbers is a negative number.

When using two's complement notation, the problem of overflow might occur when adding two positive values or when adding two negative values. In either case, the condition can be detected by checking the sign bit of the answer. That is, an overflow is indicated if the addition of two positive values results in the pattern for a negative value or if the sum of two negative values appears to be positive. The problem of overflow will never occur when one positive and one negative value are added.