

Number Representations

We usually use ten *digits* (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers. If we do not allow leading zeros, every non-negative integer has a unique representation as a finite sequence of digits. This way of representing numbers is called the *decimal* or *base 10* system. This system seems natural to us, partly because we have grown up with it and partly because we have ten fingers. The word “digits” comes from the Latin word for finger, *digitus*. Computers don’t usually have fingers; they have *bits* (0, 1) from the words “Binary” and “digIT.” Numbers on computers are represented with just 0 and 1, using the *binary* or *base 2* system.

Before we look at the binary system let’s remember and formalize how the decimal system works. If $d_0, d_1, \dots, d_{n-1}, d_n$ are digits, then

$$\begin{aligned} d_n d_{n-1} \cdots d_2 d_1 d_0 &= d_n \cdot 10^n + d_{n-1} \cdot 10^{n-1} + \cdots + d_1 \cdot 10^1 + d_0 \cdot 10^0 \\ &= \sum_{k=0}^n d_k \cdot 10^k \end{aligned}$$

For example, $60325 = 6 \cdot 10^4 + 0 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0 = 60000 + 0000 + 300 + 20 + 5$.

The following theorem tells us that we can use any integer b as a base for a number representation system.

Theorem 1 *Let b be an integer greater than 1. Then if n is a positive integer, n can be expressed uniquely in the form*

$$n = a_k \cdot b^k + a_{k-1} \cdot b^{k-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0$$

where k is a nonnegative integer, a_0, a_1, \dots, a_k are nonnegative integers less than b and $a_k \neq 0$.

We say b is the *base of expansion* of n and we write $n = (a_k a_{k-1} \cdots a_1 a_0)_b$. For “short” numbers, typically those with three digits or less, we often eliminate the parentheses (e.g., 101_2). When the base is understood, we do not write it as a subscript.

Examples

$$201_3 = 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 2 \cdot 9 + 0 \cdot 3 + 1 \cdot 1 = 19_{10}$$

$$201_5 = 2 \cdot 5^2 + 0 \cdot 5^1 + 1 \cdot 5^0 = 2 \cdot 25 + 0 \cdot 5 + 1 \cdot 1 = 51_{10}$$

We use the decimal (base 10) representation of integers in our everyday lives but as computer scientists, we will also use binary (base 2) and hexadecimal (base 16) representations. The octal (base 8) representation is rarely used these days but is included below for historical reasons.

1 Binary Representation

In the binary representation, the base is 2 and the integers, $a_k, a_{k-1}, \dots, a_1, a_0$ must be non-negative and less than 2. The only choices are 0 and 1 but we are still able to express any positive integer as indicated in the theorem.

Examples

$$\begin{aligned}(100101)_2 &= 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\ &= 37_{10}\end{aligned}$$

$$\begin{aligned}(11010111)_2 &= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 \\ &= 215_{10}\end{aligned}$$

The decimal numbers 0 through 15 written in their binary or base 2 representation are:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

Essentially all digital data is stored in binary. A *byte* is an 8-bit binary number with leading zeros allowed. There are 256 different bytes and they represent the integers from 0 (00000000) to 255 (11111111). A *word* is a basic unit of storage whose size depends on the particular computer. Words are commonly composed of four or eight bytes, 32 or 64 bits respectively.

Bytes are commonly used to represent characters. *ASCII* uses the lower 7-bits of a byte, 0 to 127, to represent letters and special characters; *ISO Latin-1* and *Mac-Roman* (now obsolete) use values above 127 for accented letters and additional special characters. *Unicode* is an international standard intended to encode all characters in all languages as well as mathematical and other specialized characters. *UTF-32*, also called *UCS-4*, uses four bytes to encode Unicode characters.

2 Hexadecimal Representation

The word *hexadecimal* combines the Greek *hexa* for six with the English word decimal [fr. L. *decimus* tenth]. “Hexadecimal” is too long to say all the time so we usually just say *hex*. We need 16 *hex-digits* to represent integers in base 16. We use the ordinary decimal digits 0, . . . , 9 and the letters A, B, C, D, E, and F (or a, b, c, d, e, and f) to represent 10, 11, 12, 13, 14, and 15 respectively.

Examples

$$\begin{aligned}A2_{16} &= 10 \cdot 16^1 + 2 \cdot 16^0 \\ &= 162_{10}\end{aligned}$$

$$\begin{aligned}(30AC92)_{16} &= 3 \cdot 16^5 + 0 \cdot 16^4 + 10 \cdot 16^3 + 12 \cdot 16^2 + 9 \cdot 16^1 + 2 \cdot 16^0 \\ &= 3 \cdot 1048576 + 0 \cdot 65536 + 10 \cdot 4096 + 12 \cdot 256 + 9 \cdot 16 + 2 \cdot 1 = (3189906)_{10}\end{aligned}$$

The binary representations of numbers can be quite long and difficult for humans to read. Hexadecimal numbers are particularly useful for representing patterns of binary values (bit-masks), machine addresses, or any particular bytes or words. Each hex-digit corresponds to four bits which is half a byte or a *nibble*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

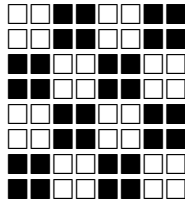
A byte can be represented by two hex-digits instead of 8 bits, and a 32-bit word can be written with only 8 hex-digits. It is easier for a human to correctly copy 8 hex-digits than 32 bits. To convert a binary integer to hex, each four-bit cluster corresponds to a single hex-digit. If the number of bits in the binary integer is not a multiple of four, add zeros to the left, e.g., $11011 = 00011011$.

Examples. There are spaces between the nibbles in the numbers below so you can see the correspondence with the hex-digits.

$$(1101\ 0011)_2 = D3_{16}$$

$$(0101\ 1110\ 1001\ 1111)_2 = (5E9F)_{16}$$

$$(1100\ 0001\ 0000\ 1010\ 0111\ 1110\ 1011\ 0101)_2 = (C10A7EB5)_{16}$$

The pattern  can be represented in binary by

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0

or by 33, 33, CC, CC, 33, 33, CC, CC in hex.

3 Octal Representation

(The) Octal system used to be widespread back when many computers used 6-bit bytes, as a 6-bit byte can be conveniently written as a two-digit octal number. Since nowadays a byte is almost always 8-bits long the octal system lost most of its appeal to the hexadecimal system. *The Free On-line Dictionary of Computing (2003-OCT-10)*

Examples

$$\begin{aligned} 723_8 &= 7 \cdot 8^2 + 2 \cdot 8^1 + 3 \cdot 8^0 \\ &= 7 \cdot 64 + 2 \cdot 8 + 3 = 448 + 16 + 3 \\ &= 467_{10} \end{aligned}$$

$$\begin{aligned} (2045)_8 &= 2 \cdot 8^3 + 0 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0 = 2 \cdot 512 + 0 \cdot 64 + 4 \cdot 8 + 5 \\ &= 1024 + 0 + 32 + 5 \\ &= (1061)_{10} \end{aligned}$$

To convert a binary integer to octal, each three-bit cluster corresponds to a single octal-digit. If the number of bits in the binary integer is not a multiple of three, add zeros to the left, e.g., $11011 = 011011$. There are spaces between to separate the three-bit clusters in the numbers below so you can see the correspondence with the octal-digits.

Examples

$$(010\ 011)_2 = 23_8$$

$$(111\ 010\ 011\ 111)_2 = (7237)_8$$

$$(100\ 000\ 100\ 001\ 010\ 111\ 111\ 010\ 110\ 101)_2 = (4041277265)_8$$

4 Converting Between Decimal and Binary

In the examples above, we converted numbers given in binary, hex, and octal representations to their decimal equivalents by multiplying each bit, digit, hex-digit, or octal-digit by the appropriate power of the base, using base-10 arithmetic, and adding up the pieces. We were really just evaluating a polynomial at the base 2, 16, or 8. Recall that:

$$\begin{aligned}n &= (a_k a_{k-1} \cdots a_1 a_0)_b \\ &= a_k \cdot b^k + a_{k-1} \cdot b^{k-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0\end{aligned}$$

$$\begin{aligned}(1101)_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 8 + 4 + 0 + 1 \\ &= 13_{10}\end{aligned}$$

$$\begin{aligned}(1101)_8 &= 1 \cdot 8^3 + 1 \cdot 8^2 + 0 \cdot 8^1 + 1 \cdot 8^0 \\ &= 512 + 64 + 0 + 1 \\ &= 577_{10}\end{aligned}$$

$$\begin{aligned}(1101)_{16} &= 1 \cdot 16^3 + 1 \cdot 16^2 + 0 \cdot 16^1 + 1 \cdot 16^0 \\ &= 4096 + 256 + 0 + 1 \\ &= (4353)_{10}\end{aligned}$$

How do you convert a decimal integer to its binary equivalent? For small integers N , you can easily do this by find the largest power of 2 less than N , say 2^k , and then finding the binary representation of $N - 2^k$. Just remember to use “place-keeper” zeros for missing powers of 2. For example, $39 = 32 + 7$ so we need a 1 in the $2^5 = 32$ place. The remaining 7 is too small for any 16s or 8s so we put a 0 in each of those places and then 1s in the 4, 2, and 1 places.

$$39_{10} = (100111)_2$$

The division algorithm provides a way to convert an integer to any base: just divide n by b . There are integers q_0 and r_0 such that $n = q_0 \cdot b + r_0$ where $0 \leq r_0 < b$. The remainder r_0 is the base- b digit that goes in the 1s place, the rightmost digit. Now divide q_0 by b . We have $q_0 = q_1 \cdot b + r_1$ where $0 \leq r_1 < b$. The remainder r_1 is the base- b digit that is second from the right, and so on.

$$\begin{aligned}39 &= 19 \cdot 2 + 1 \\ 19 &= 9 \cdot 2 + 1 \\ 9 &= 4 \cdot 2 + 1 \\ 4 &= 2 \cdot 2 + 0 \\ 2 &= 1 \cdot 2 + 0 \\ 1 &= 0 \cdot 2 + 1\end{aligned}$$

Therefore, once again, we see that $39_{10} = (100111)_2$.