# A

# BINARY NUMBERS

The arithmetic used by computers differs in some ways from the arithmetic used by people. The most important difference is that computers perform operations on numbers whose precision is finite and fixed. Another difference is that most computers use the binary rather than the decimal system for representing numbers. These topics are the subject of this appendix.

## A.1 FINITE-PRECISION NUMBERS

While doing arithmetic, one usually gives little thought to the question of how many decimal digits it takes to represent a number. Physicists can calculate that there are $10^{78}$ electrons in the universe without being bothered by the fact that it requires 79 decimal digits to write that number out in full. Someone calculating the value of a function with pencil and paper who needs the answer to six significant digits simply keeps intermediate results to seven, or eight, or however many are needed. The problem of the paper not being wide enough for seven-digit numbers never arises.

With computers, matters are quite different. On most computers, the amount of memory available for storing a number is fixed at the time that the computer is designed. With a certain amount of effort, the programmer can represent numbers two, or three, or even many times larger than this fixed amount, but doing so does not change the nature of this difficulty. The finite nature of the computer forces

us to deal only with numbers that can be represented in a fixed number of digits. We call such numbers **finite-precision numbers**.

In order to study properties of finite-precision numbers, let us examine the set of positive integers representable by three decimal digits, with no decimal point and no sign. This set has exactly 1000 members: 000, 001, 002, 003, ..., 999. With this restriction, it is impossible to express certain kinds of numbers, such as

1. Numbers larger than 999.
2. Negative numbers.
3. Fractions.
4. Irrational numbers.
5. Complex numbers.

One important property of arithmetic on the set of all integers is **closure** with respect to the operations of addition, subtraction, and multiplication. In other words, for every pair of integers $i$ and $j$, $i + j$, $i - j$, and $i \times j$ are also integers. The set of integers is not closed with respect to division, because there exist values of $i$ and $j$ for which $i/j$ is not expressible as an integer (e.g., 7/2 and 1/0).

Finite-precision numbers are not closed with respect to any of these four basic operations, as shown below, using three-digit decimal numbers as an example:

| | |
|---|---|
| 600 + 600 = 1200 | (too large) |
| 003 – 005 = –2 | (negative) |
| 050 × 050 = 2500 | (too large) |
| 007 / 002 = 3.5 | (not an integer) |

The violations can be divided into two mutually exclusive classes: operations whose result is larger than the largest number in the set (overflow error) or smaller than the smallest number in the set (underflow error), and operations whose result is neither too large nor too small but is simply not a member of the set. Of the four violations above, the first three are examples of the former, and the fourth is an example of the latter.

Because computers have finite memories and therefore must of necessity perform arithmetic on finite-precision numbers, the results of certain calculations will be, from the point of view of classical mathematics, just plain wrong. A calculating device that gives the wrong answer even though it is in perfect working condition may appear strange at first, but the error is a logical consequence of its finite nature. Some computers have special hardware that detects overflow errors.

The algebra of finite-precision numbers is different from normal algebra. As an example, consider the associative law:

$$a + (b - c) = (a + b) - c$$

Let us evaluate both sides for $a = 700$, $b = 400$, $c = 300$. To compute the left-hand side, first calculate $(b - c)$, which is 100, and then add this amount to $a$, yielding 800. To compute the right-hand side, first calculate $(a + b)$, which gives an overflow in the finite arithmetic of three-digit integers. The result may depend on the machine being used but it will not be 1100. Subtracting 300 from some number other than 1100 will not yield 800. The associative law does not hold. The order of operations is important.



As another example, consider the distributive law:

$$a \times (b - c) = a \times b - a \times c$$

Let us evaluate both sides for $a = 5$, $b = 210$, $c = 195$. The left-hand side is $5 \times 15$, which yields 75. The right-hand side is not 75 because $a \times b$ overflows.

Judging from these examples, one might conclude that although computers are general-purpose devices, their finite nature renders them especially unsuitable for doing arithmetic. This conclusion is, of course, not true, but it does serve to illustrate the importance of understanding how computers work and what limitations they have.

## A.2 RADIX NUMBER SYSTEMS

An ordinary decimal number with which everyone is familiar consists of a string of decimal digits and, possibly, a decimal point. The general form and its usual interpretation are shown in Fig. A-1. The choice of 10 as the base for exponentiation, called the **radix**, is made because we are using decimal, or base 10, numbers. When dealing with computers, it is frequently convenient to use radices other than 10. The most important radices are 2, 8, and 16. The number systems based on these radices are called **binary**, **octal**, and **hexadecimal**, respectively.

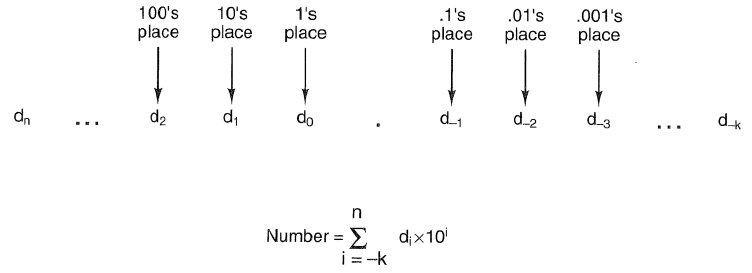


**Figure A-1.** The general form of a decimal number.

A radix $k$ number system requires $k$ different symbols to represent the digits 0 to $k - 1$. Decimal numbers are built up from the 10 decimal digits

0 1 2 3 4 5 6 7 8 9

In contrast, binary numbers do not use these ten digits. They are all constructed exclusively from the two binary digits

0 1

Octal numbers are built up from the eight octal digits

0 1 2 3 4 5 6 7

For hexadecimal numbers, 16 digits are needed. Thus six new symbols are required. It is conventional to use the uppercase letters A through F for the six digits following 9. Hexadecimal numbers are then built up from the digits

0 1 2 3 4 5 6 7 8 9 A B C D E F

The expression "binary digit" meaning a 1 or a 0 is usually referred to as a **bit**. Figure A-2 shows the decimal number 2001 expressed in binary, octal, decimal, and hexadecimal form. The number 7B9 is obviously hexadecimal, because the symbol B can only occur in hexadecimal numbers. However, the number 111 might be in any of the four number systems discussed. To avoid ambiguity, people use a subscript of 2, 8, 10, or 16 to indicate the radix when it is not obvious from the context.

Binary 1 1 1 1 1 0 1 0 0 0 1

$1\times2^{10}+1\times2^{9}+1\times2^{8}+1\times2^{7}+1\times2^{6}+0\times2^{5}+1\times2^{4}+0\times2^{3}+0\times2^{2}+0\times2^{1}+1\times2^{0}$

1024 + 512 + 256 + 128 + 64 + 0 +16 + 0 + 0 + 0 + 1

Octal 3 7 2 1

$3\times8^{3}+7\times8^{2}+2\times8^{1}+1\times8^{0}$

1536 + 448 + 16 + 1

Decimal 2 0 0 1

$2\times10^{3}+0\times10^{2}+0\times10^{1}+1\times10^{0}$

2000 + 0 + 0 + 1

Hexadecimal 7 D 1

$7\times16^{2}+13\times16^{1}+1\times16^{0}$

1792 + 208 + 1

**Figure A-2.** The number 2001 in binary, octal, and hexadecimal.

As an example of binary, octal, decimal, and hexadecimal notation, consider Fig. A-3, which shows a collection of nonnegative integers expressed in each of these four different systems. Perhaps some archaeologist thousands of years from now will discover this table and regard it as the Rosetta Stone to late twentieth century and early twenty-first century number systems.



| Decimal | Binary | Octal | Hex |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 3 | 3 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |
| 16 | 10000 | 20 | 10 |
| 20 | 10100 | 24 | 14 |
| 30 | 11110 | 36 | 1E |
| 40 | 101000 | 50 | 28 |
| 50 | 110010 | 62 | 32 |
| 60 | 111100 | 74 | 3C |
| 70 | 1000110 | 106 | 46 |
| 80 | 1010000 | 120 | 50 |
| 90 | 1011010 | 132 | 5A |
| 100 | 11001000 | 144 | 64 |
| 1000 | 1111101000 | 1750 | 3E8 |
| 2989 | 101110101101 | 5655 | BAD |

**Figure A-3.** Decimal numbers and their binary, octal, and hexadecimal equivalents.

## A.3 CONVERSION FROM ONE RADIX TO ANOTHER

Conversion between octal or hexadecimal numbers and binary numbers is easy. To convert a binary number to octal, divide it into groups of 3 bits, with the 3 bits immediately to the left (or right) of the decimal point (often called a binary point) forming one group, the 3 bits immediately to their left, another group, and so on. Each group of 3 bits can be directly converted to a single octal digit, 0 to 7, according to the conversion given in the first lines of Fig. A-3. It may be necessary to add one or two leading or trailing zeros to fill out a group to 3 full bits. Conversion from octal to binary is equally trivial. Each octal digit is simply replaced by the equivalent 3-bit binary number. Conversion from hexadecimal to

binary is essentially the same as octal-to-binary except that each hexadecimal digit corresponds to a group of 4 bits instead of 3 bits. Figure A-4 gives some examples of conversions.

**Example 1**

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hexadecimal | | 1 | | 9 | | 4 | | 8 | . | B | | 6 | | | | | |
| Binary | | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | . | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| Octal | | 1 | | 4 | | 5 | | 1 | | 0 | . | 5 | | 5 | | 4 | | | | | | | | | | | |

**Example 2**

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hexadecimal | | 7 | | B | | A | | 3 | . | B | | C | | 4 | | | | | | | | | | | | | |
| Binary | | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | . | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| Octal | | 7 | | 5 | | 6 | | 4 | | 3 | . | 5 | | 7 | | 0 | | 4 | | | | | | | | | |

**Figure A-4.** Examples of octal-to-binary and hexadecimal-to-binary conversion.

Conversion of decimal numbers to binary can be done in two different ways. The first method follows directly from the definition of binary numbers. The largest power of 2 smaller than the number is subtracted from the number. The process is then repeated on the difference. Once the number has been decomposed into powers of 2, the binary number can be assembled with 1s in the bit positions corresponding to powers of 2 used in the decomposition, and 0s elsewhere.

The other method (for integers only) consists of dividing the number by 2. The quotient is written directly beneath the original number and the remainder, 0 or 1, is written next to the quotient. The quotient is then considered and the process repeated until the number 0 has been reached. The result of this process will be two columns of numbers, the quotients and the remainders. The binary number can now be read directly from the remainder column starting at the bottom. Figure A-5 gives an example of decimal-to-binary conversion.

Binary integers can also be converted to decimal in two ways. One method consists of summing up the powers of 2 corresponding to the 1 bits in the number. For example,

$$10110 = 2^4 + 2^2 + 2^1 = 16 + 4 + 2 = 22$$

In the other method, the binary number is written vertically, one bit per line, with the leftmost bit on the bottom. The bottom line is called line 1, the one above it line 2, and so on. The decimal number will be built up in a parallel column next to the binary number. Begin by writing a 1 on line 1. The entry on line $n$ consists of two times the entry on line $n-1$ plus the bit on line $n$ (either 0 or 1). The entry on the top line is the answer. Figure A-6 gives an example of this method of binary to decimal conversion.



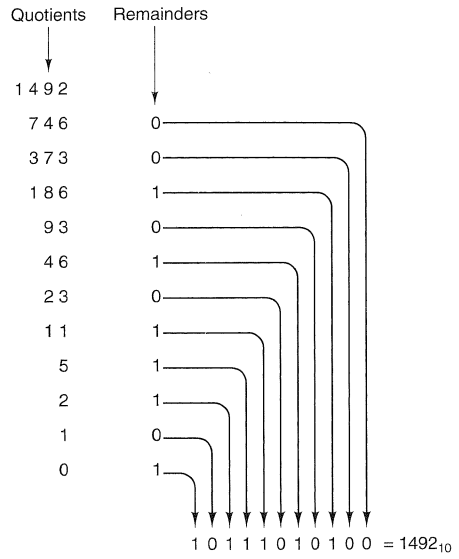


**Figure A-5.** Conversion of the decimal number 1492 to binary by successive halving, starting at the top and working downward. For example, 93 divided by 2 yields a quotient of 46 and a remainder of 1, written on the line below it.

Decimal-to-octal and decimal-to-hexadecimal conversion can be accomplished either by first converting to binary and then to the desired system or by subtracting powers of 8 or 16.

## A.4 NEGATIVE BINARY NUMBERS

Four different systems for representing negative numbers have been used in digital computers at one time or another in history. The first one is called **signed magnitude**. In this system the leftmost bit is the sign bit (0 is + and 1 is −) and the remaining bits hold the absolute magnitude of the number.

The second system, called **one's complement**, also has a sign bit with 0 used for plus and 1 for minus. To negate a number, replace each 1 by a 0 and each 0 by a 1. This holds for the sign bit as well. One's complement is obsolete.

The third system, called **two's complement**, also has a sign bit that is 0 for plus and 1 for minus. Negating a number is a two-step process. First, each 1 is

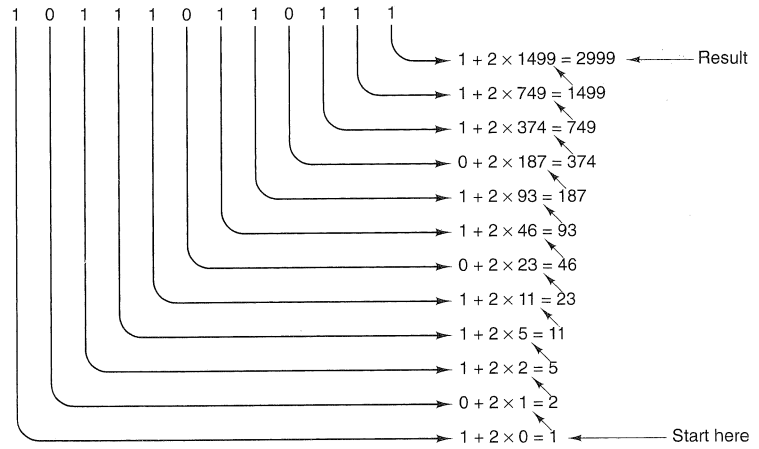


**Figure A-6.** Conversion of the binary number 101110110111 to decimal by successive doubling, starting at the bottom. Each line is formed by doubling the one below it and adding the corresponding bit. For example, 749 is twice 374 plus the 1 bit on the same line as 749.

replaced by a 0 and each 0 by a 1, just as in one's complement. Second, 1 is added to the result. Binary addition is the same as decimal addition except that a carry is generated if the sum is greater than 1 rather than greater than 9. For example, converting 6 to two's complement is done in two steps:

00000110 (+6)
11111001 (–6 in one's complement)
11111010 (–6 in two's complement)

If a carry occurs from the leftmost bit, it is thrown away.

The fourth system, which for $m$-bit numbers is called **excess $2^{m-1}$**, represents a number by storing it as the sum of itself and $2^{m-1}$. For example, for 8-bit numbers, $m = 8$, the system is called excess 128 and a number is stored as its true value plus 128. Therefore, –3 becomes $-3 + 128 = 125$, and –3 is represented by the 8-bit binary number for 125 (01111101). The numbers from –128 to +127 map onto 0 to 255, all of which are expressible as an 8-bit positive integer. Interestingly enough, this system is identical to two's complement with the sign bit reversed. Figure A-7 gives examples of negative numbers in all four systems.

Both signed magnitude and one's complement have two representations for zero: a plus zero, and a minus zero. This situation is undesirable. The two's complement system does not have this problem because the two's complement of plus



| N decimal | N binary | –N signed mag. | –N 1's compl. | –N 2's compl. | –N excess 128 |
|---|---|---|---|---|---|
| 1 | 00000001 | 10000001 | 11111110 | 11111111 | 01111111 |
| 2 | 00000010 | 10000010 | 11111101 | 11111110 | 01111110 |
| 3 | 00000011 | 10000011 | 11111100 | 11111101 | 01111101 |
| 4 | 00000100 | 10000100 | 11111011 | 11111100 | 01111100 |
| 5 | 00000101 | 10000101 | 11111010 | 11111011 | 01111011 |
| 6 | 00000110 | 10000110 | 11111001 | 11111010 | 01111010 |
| 7 | 00000111 | 10000111 | 11111000 | 11111001 | 01111001 |
| 8 | 00001000 | 10001000 | 11110111 | 11111000 | 01111000 |
| 9 | 00001001 | 10001001 | 11110110 | 11110111 | 01110111 |
| 10 | 00001010 | 10001010 | 11110101 | 11110110 | 01110110 |
| 20 | 00010100 | 10010100 | 11101011 | 11101100 | 01101100 |
| 30 | 00011110 | 10011110 | 11100001 | 11100010 | 01100010 |
| 40 | 00101000 | 10101000 | 11010111 | 11011000 | 01011000 |
| 50 | 00110010 | 10110010 | 11001101 | 11001110 | 01001110 |
| 60 | 00111100 | 10111100 | 11000011 | 11000100 | 01000100 |
| 70 | 01000110 | 11000110 | 10111001 | 10111010 | 00111010 |
| 80 | 01010000 | 11010000 | 10101111 | 10110000 | 00110000 |
| 90 | 01011010 | 11011010 | 10100101 | 10100110 | 00100110 |
| 100 | 01100100 | 11100100 | 10011011 | 10011100 | 00011100 |
| 127 | 01111111 | 11111111 | 10000000 | 10000001 | 00000001 |
| 128 | Nonexistent | Nonexistent | Nonexistent | 10000000 | 00000000 |

**Figure A-7.** Negative 8-bit numbers in four systems.

zero is also plus zero. The two's complement system does, however, have a different singularity. The bit pattern consisting of a 1 followed by all 0s is its own complement. The result is to make the range of positive and negative numbers unsymmetric; there is one negative number with no positive counterpart.

The reason for these problems is not hard to find: we want an encoding system with two properties:

1. Only one representation for zero.
2. Exactly as many positive numbers as negative numbers.

The problem is that any set of numbers with as many positive as negative numbers and only one zero has an odd number of members, whereas $m$ bits allow an even number of bit patterns. There will always be either one bit pattern too many or one bit pattern too few, no matter what representation is chosen. This extra bit

pattern can be used for –0 or for a large negative number, or for something else, but no matter what it is used for it will always be a nuisance.

## A.5 BINARY ARITHMETIC

The addition table for binary numbers is given in Fig. A-8.

| | | | | |
|---|---|---|---|---|
| Addend | 0 | 0 | 1 | 1 |
| Augend | +0 | +1 | +0 | +1 |
| Sum | 0 | 1 | 1 | 0 |
| Carry | 0 | 0 | 0 | 1 |

**Figure A-8.** The addition table in binary.

Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits in the addend and the augend. If a carry is generated, it is carried one position to the left, just as in decimal arithmetic. In one's complement arithmetic, a carry generated by the addition of the leftmost bits is added to the rightmost bit. This process is called an end-around carry. In two's complement arithmetic, a carry generated by the addition of the leftmost bits is merely thrown away. Examples of binary arithmetic are shown in Fig. A-9.

| Decimal | 1's complement | 2's complement |
|---|---|---|
| 10 | 00001010 | 00001010 |
| + (–3) | 11111100 | 11111101 |
| +7 | 1 00000110 | 1 00000111 |
| | carry 1 | discarded |
| | 00000111 | |

**Figure A-9.** Addition in one's complement and two's complement.

If the addend and the augend are of opposite signs, overflow error cannot occur. If they are of the same sign and the result is of the opposite sign, overflow error has occurred and the answer is wrong. In both one's and two's complement arithmetic, overflow occurs if and only if the carry into the sign bit differs from the carry out of the sign bit. Most computers preserve the carry out of the sign bit, but the carry into the sign bit is not visible from the answer. For this reason, a special overflow bit is usually provided.



## PROBLEMS

1. Convert the following numbers to binary: 1984, 4000, 8192.
2. What is 1001101001 (binary) in decimal? In octal? In hexadecimal?
3. Which of the following are valid hexadecimal numbers? BED, CAB, DEAD, DECADE, ACCEDED, BAG, DAD.
4. Express the decimal number 100 in all radices from 2 to 9.
5. How many different positive integers can be expressed in $k$ digits using radix $r$ numbers?
6. Most people can only count to 10 on their fingers; however, computer scientists can do better. If you regard each finger as one binary bit, with finger extended as 1 and finger touching palm as 0, how high can you count using both hands? With both hands and both feet? Now use both hands and both feet, with the big toe on your left foot as a sign bit for two's complement numbers. What is the range of expressible numbers?
7. Perform the following calculations on 8-bit two's complement numbers.

| | | | |
|---|---|---|---|
| 00101101 | 11111111 | 00000000 | 11110111 |
| + 01101111 | + 11111111 | – 11111111 | – 11110111 |

8. Repeat the calculation of the preceding problem but now in one's complement.
9. Consider the following addition problems for 3-bit binary numbers in two's complement. For each sum, state
   a. Whether the sign bit of the result is 1.
   b. Whether the low-order 3 bits are 0.
   c. Whether an overflow occurred.

| | | | | |
|---|---|---|---|---|
| 000 | 000 | 111 | 100 | 100 |
| + 001 | + 111 | + 110 | + 111 | + 100 |

10. Signed decimal numbers consisting of $n$ digits can be represented in $n + 1$ digits without a sign. Positive numbers have 0 as the leftmost digit. Negative numbers are formed by subtracting each digit from 9. Thus the negative of 014725 is 985274. Such numbers are called nine's complement numbers and are analogous to one's complement binary numbers. Express the following as three-digit nine's complement numbers: 6, –2, 100, –14, –1, 0.
11. Determine the rule for addition of nine's complement numbers and then perform the following additions.

| | | | |
|---|---|---|---|
| 0001 | 0001 | 9997 | 9241 |
| + 9999 | + 9998 | + 9996 | + 0802 |

12. Ten's complement is analogous to two's complement. A ten's complement negative number is formed by adding 1 to the corresponding nine's complement number, ignoring the carry. What is the rule for ten's complement addition?