CHAPTER 2

Bases and number representation

2.1 Real numbers and the decimal number system

Numbers can be represented in various ways. For example, the number 'four' can be written in the usual notation as 4, in Roman numerals as IV, or even just as four tally marks, ||||. Similarly, 'two-and-a-half' can be written as the mixed number $2\frac{1}{2}$, as the improper fraction $\frac{5}{2}$ or as the decimal number 2.5. In particular, numbers can be represented using systems similar to the familiar decimal system but using a base other than 10. In this chapter, we investigate the representation of numbers using different number bases, paying particular attention to the number systems used in computing.

Before we look at other number bases, it is helpful to recall what types of numbers there are, and how they are represented in the decimal system:

- The *natural* numbers (also called the positive integers) are the numbers 1, 2, 3, 4, ...
- The *integers*, or whole numbers, include zero and the negative whole numbers as well as the natural numbers: ..., –3, –2, –1, 0, 1, 2, 3, ...
- The *rational* numbers are all the numbers that can be expressed in the form m/n where m and n are integers and n is not zero. Note that this includes all of the integers, as well as all 'fractions', both proper (for example, $\frac{1}{2}$) and improper (for example, $\frac{5}{3}$). Any rational number can also be written as a terminating or recurring decimal, for example $\frac{1}{4} = 0.25$, $\frac{1}{6} = 0.16666\ldots$

One of the more surprising mathematical facts is that some of the numbers that we would like to be able to define and use are not rational numbers. For example, although $\sqrt{2}$ can be approximated by rational numbers to any degree of accuracy we choose, it is impossible to find integers m and n such that $m / n = \sqrt{2}$ exactly. We express this fact by saying that $\sqrt{2}$ is an *irrational* number. (A proof that $\sqrt{2}$ is irrational will have to wait until we have studied proof techniques in Chapter 4.) All roots (square roots, cube roots and so on) of the natural numbers are irrational numbers, except those roots that are themselves natural

numbers such as $\sqrt{4}$. The numbers π and e (the base of natural logarithms) are also irrational. The decimal representation of an irrational number does not terminate or recur, for example $\sqrt{2} = 1.4142135\ldots$ and $\pi = 3.1415926\ldots$.

All of the numbers we have introduced, both rational and irrational numbers, are called *real* numbers. For example, 3, $\frac{7}{5}$, −4.38 and $2-3\sqrt{5}$ are all real numbers. Unless you have studied complex numbers elsewhere, all of the numbers you have encountered in your studies are real numbers.

A real number is written in the decimal system as a string of digits, preceded by a minus sign if the number is negative. The representation may also include a decimal point. (If no decimal point is written, there is an implied decimal point after the last digit.) Some real numbers, such as $\frac{1}{6}$ and $\sqrt{2}$, have non-terminating decimal representations; they cannot be represented exactly in this system using a finite number of digits.

The decimal system is an example of a *positional* number system, because each digit has a *place value* that depends on its position in relation to the decimal point. The digit immediately to the left of the point is the units (or ones) digit ($1 = 10^0$). To the left of the units digit is the tens digit ($10 = 10^1$), then the hundreds digit ($100 = 10^2$), the thousands digit ($1000 = 10^3$), and so on. Similarly, to the right of the decimal point are the tenths digit ($\frac{1}{10} = 10^{-1}$), the hundredths digit ($\frac{1}{100} = 10^{-2}$), and so on. The value of the number is obtained by multiplying each digit by its place value and adding the results.

For example, the decimal number 2386.75 can be expanded in the following way:

$$2386.75 = 2 \times 10^3 + 3 \times 10^2 + 8 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 5 \times 10^{-2}$$

The decimal system is said to use a *base* of 10 because the place values are powers of 10.

2.2 The binary number system

Most of what we have said so far is probably already familiar to you. Because we use the decimal system all the time, we rarely give it a second thought, and it is easy to forget that there is no mathematical reason for using powers of 10 as the place values. The choice of 10 as the base presumably arose because people in ancient times used their 10 fingers for counting, but in fact any natural number greater than 1 can be used as the base of a positional number system.[1]

A familiar example of something like a non-decimal number system is the subdivision of an hour into 60 minutes and a minute into 60 seconds,

1 Actually, the situation is more general than this – negative numbers, and even complex numbers, can be used as bases. An interesting discussion of the different number systems that can be constructed appears in Chapter 4 of *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., by D.E. Knuth (Addison-Wesley, 1981).

giving a kind of base 60 number system. For example, a time of 2 hours 26 minutes and 35 seconds can be expressed in seconds as follows:

$$2\text{ h }26\text{ m }35\text{ s} = 2 \times 60^2 + 26 \times 60^1 + 35 \times 60^0 \text{ seconds}$$

Notice the similarity of this expression to the expansion of a decimal number into powers of 10. (The analogy is not perfect, because there are no names for 60^3 seconds, 60^4 seconds, and so on.)

It turns out, for reasons that will be explained later in this chapter, that the bases 2, 8 and 16, corresponding to what are known as the binary, octal and hexadecimal systems respectively, are particularly useful in computing.

The *binary* system is the positional number system that uses 2 as the base. Whereas the decimal system uses the 10 decimal digits 0, 1, 2, ..., 9, the binary system uses the 2 binary digits (or *bits*) 0 and 1. A positive number written in the binary system appears as a string of zeros and ones, and may also include a point (don't call it a decimal point!); for example:

$$1101.01_2$$

The subscript 2 denotes the base. The base should always be indicated in this way in any work involving number systems with bases other than 10.

The place values of the digits in a binary number are powers of 2. Starting at the point and moving to the left, we have the units digit ($1 = 2^0$), the twos digit ($2 = 2^1$), the fours digit ($4 = 2^2$), the eights digit, and so on. To the right of the point are the halves digit, the quarters digit, the eighths digit, and so on. A binary number can be evaluated (and hence converted to decimal) by writing it in expanded form:

$$\begin{aligned} 1101.01_2 &= 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} \\ &= 8 + 4 + 1 + 0.25 \\ &= 13.25 \end{aligned}$$

The reason that binary numbers arise so often in computing can be summarised as follows. In the digital computing systems in common use, the devices that store data or information of any kind (including permanent storage media such as a disk, tape or CD, and temporary media such as random access memory (RAM) in a microchip) consist of a large number of memory elements, each of which can be in one of two states (such as magnetised or unmagnetised, on or off). Similarly, when data is transmitted inside a computer or through a network, it is usually coded as a stream of signal elements that take one of two forms, such as the presence or absence of an electric current, or two alternating currents with different frequencies. (An exception is the modems used for data communication across telephone networks, which use a set of 256 frequencies.) The manipulation of data to perform arithmetic and other computations takes place in the digital circuitry etched on a microchip, which also operates using currents of just two kinds. In short, the data handled by a digital computer is stored, transmitted and manipulated as a

stream of information 'elements' of two types, which can be denoted by the symbols 0 and 1. Thus the binary number system is the most natural way of representing numbers in a digital computer.

Table 2.1 shows the integers (whole numbers) from 0 to 20 in their binary and decimal representations.

Table 2.1

Binary	*Decimal*	*Binary*	*Decimal*
0	0		
1	1	1011	11
10	2	1100	12
11	3	1101	13
100	4	1110	14
101	5	1111	15
110	6	10000	16
111	7	10001	17
1000	8	10010	18
1001	9	10011	19
1010	10	10100	20

2.3 Conversion from decimal to binary

How can a number be converted from the decimal to the binary system? It should come as no surprise that the answer to this question will take the form of an algorithm. Different processes are used for converting the integer and fractional parts of a number, so we will consider the two cases separately, beginning with the integer part.

Examination of Table 2.1 reveals the following pattern:

- Odd numbers have a binary representation ending in 1.
- Even numbers have a binary representation ending in 0.

These two statements can be summarised as follows:

> *The last digit in the binary representation is the remainder after dividing the number by 2.*

The first step in carrying out the conversion to binary is therefore to divide the decimal number by 2, to obtain a (whole number) *quotient* and a *remainder*. We will use the following (Pascal-style) notation to denote these operations:

n div 2 is the quotient when n is divided by 2

n mod 2 is the remainder when n is divided by 2

For example:

12 div 2 = 6, 12 mod 2 = 0

13 div 2 = 6, 13 mod 2 = 1

In order to see how to proceed further, we need to make another observation:

The binary number obtained by removing the rightmost bit from the binary representation of n is the binary representation of n div 2.

Therefore, in order to obtain the second bit from the right-hand end, we must perform another division by 2, starting with the quotient from the first division, and obtaining a new quotient and remainder. This process is repeated, the remainders at each step forming the digits of the answer from right to left, until a quotient of zero is obtained.

Here is the entire process written as an algorithm:

1. Input n {n must be a natural number}
2. **Repeat**
   2.1. Output n mod 2
   2.2. $n \leftarrow n$ div 2
   until $n = 0$

The outputs must be read in reverse order to yield the correct answer.

Table 2.2 shows a trace of the algorithm with $n = 6$.

Table 2.2

Step	n	Output
1	6	–
2.1	6	0
2.2	3	–
2.1	3	1
2.2	1	–
2.1	1	1
2.2	0	–

The answer, reading the output column from the bottom up, is 110_2.

In practice, it is more convenient to set out decimal-to-binary conversions in the manner shown in the next example.

Example 2.3.1 Convert the decimal number 25 to its binary representation.

Solution The number to be converted (25) forms the first entry in a column, with the new base (2) written to the left. At each step, the last entry in the column is divided by 2, the quotient is written below the number being divided, and the remainder is written in another column to the right.

2	25	
	12	1
	6	0
	3	0
	1	1
	0	1

(Leaving out the last step, 1 div 2 = 0 and 1 mod 2 = 1, is a common mistake. The algorithm does not terminate until the quotient is zero.)

Reading the remainder column from the bottom up, we obtain the answer: $25_{10} = 11001_2$.

We now turn to the problem of converting a decimal fraction to binary. For example, suppose we want to find the binary representation of 0.375_{10}. Because 0.375 is less than 0.5, the first bit to the right of the point in the binary representation (the halves bit, which has a place value of $\frac{1}{2}$) will be 0. Expressed another way, the halves bit is 0 because 2×0.375 is less than 1. If, on the other hand, we had started with a number greater than or equal to $\frac{1}{2}$, say 0.875, the halves bit would be 1, and $2 \times 0.875 \geq 1$. We can sum up this observation in the following rule:

The halves bit in the binary representation of n is the integer part of 2n.

Some new notation will be needed in what follows:

$\lfloor n \rfloor$ denotes the integer part of n

frac(n) denotes the fractional part of n

For example, $\lfloor 2.7 \rfloor = 2$ and frac(2.7) = 0.7.

The observation we have made suggests that the following process involving repeated multiplication by 2 can be used to convert a decimal fraction to its binary representation.

1. Input n
2. **Repeat**
   2.1. $m \leftarrow 2n$
   2.2. Output $\lfloor m \rfloor$
   2.3. $n \leftarrow$ frac(m)
   until $n = 0$

There is one problem with this process – what happens if the condition $n = 0$ is never satisfied? This difficulty can certainly arise; we already

know that the decimal representation of a fraction need not terminate (recall $\frac{1}{3} = 0.33333...$), and the same can happen with the binary representation. We conclude that this sequence of steps is not an algorithm, since it fails the requirement that an algorithm must terminate after a finite number of steps.

One way to avoid the difficulty is to specify on input the number of digits we want in the answer, and to output just that number of digits of the binary representation. Here is the process with the necessary changes made:

1. Input n, *digits*
2. $i \leftarrow 0$
3. **Repeat**
   3.1. $i \leftarrow i + 1$
   3.2. $m \leftarrow 2n$
   3.3. Output $\lfloor m \rfloor$
   3.4. $n \leftarrow \text{frac}(m)$
   until $n = 0$ or $i =$ *digits*

In practice, the calculations are usually set out as shown in the next example.

Example 2.3.2 Convert the decimal fraction 0.32_{10} to its binary representation, with 5 digits after the point.

Solution The number to be converted forms the first entry in a column, with the new base (2) written to the right. At each step, the last entry in the column is multiplied by 2, the fractional part is written below the number being multiplied, and the integer part is written in another column to the left. For convenience, the point in front of the fractional part is omitted.

	32	2
0	64	
1	28	
0	56	
1	12	
0	24	

The left column (read from the top down) gives the answer: $0.32_{10} = 0.01010..._2$. (Note that the answer is truncated to 5 digits after the point; it is not rounded off.)

If a number has both an integer part and a fractional part, simply convert each part separately and combine the results. For example, now that we have shown that $25_{10} = 11001_2$ and $0.32_{10} = 0.01010..._2$, it follows without any further work that $25.32_{10} = 11001.01010..._2$.

2.4 The octal and hexadecimal systems

The techniques described in the previous section can be generalised to bases other than 2. In particular, the bases 8 and 16 are often used in computing, for a reason that will be explained shortly.

In the base 8 or *octal* system, numbers are written using the 8 octal digits 0, 1, 2, 3, 4, 5, 6, 7. The place value of each digit is a power of 8.

For example:

$$\begin{aligned} 374.2_8 &= 3\times8^2 + 7\times8^1 + 4\times8^0 + 2\times8^{-1} \\ &= 252.25_{10} \end{aligned}$$

In the base 16 or *hexadecimal* system (often simply called 'hex'), 16 digits are needed. New symbols are required to denote the digits with values 10, 11, 12, 13, 14 and 15, and there is an established convention that the first six upper-case letters are used for this purpose. The 16 hexadecimal digits are therefore 0, 1, 2, ..., 9, A, B, C, D, E, F. The place value of each digit is a power of 16.

For example:

$$\begin{aligned} \text{E9C.8}_{16} &= 14\times16^2 + 9\times16^1 + 12\times16^0 + 8\times16^{-1} \\ &= 3740.5_{10} \end{aligned}$$

Converting a decimal number to its octal or hexadecimal representation is similar to converting from decimal to binary; the only difference is that 8 or 16 is used in place of 2 as the divisor or multiplier.

Example 2.4.1 Convert 275.4375_{10} to octal.

Solution

Convert the integer part:

8	275	
	34	3
	4	2
	0	4

$275_{10} = 423_8$

Convert the fractional part:

	4375	8
3	5000	
4	0	

$0.4375_{10} = 0.34_8$

Combine the results to obtain the answer:

$275.4375_{10} = 423.34_8$

Example 2.4.2 Convert 985.78125_{10} to hexadecimal.

Solution Convert the integer part:

16	985	
	61	9
	3	13
	0	3

$985_{10} = 3D9_{16}$

Convert the fractional part:

	78125	16
12	50000	
8	0	

$0.78125_{10} = 0.C8_{16}$

Combine the results to obtain the answer:

$985.78125_{10} = 3D9.C8_{16}$

We have seen why the binary system is important in computing; it is the system used internally by the computer itself. But why are the octal and hexadecimal systems useful?

The main drawback of using the binary system as a general purpose number system is that even moderately large integers have many digits in their binary representation (typically more than three times as many as in the decimal representation). The advantage of larger bases is that we can write numbers using fewer digits. However, the decimal system is inconvenient if we often have to convert between it and the binary system, because, as we have seen, the conversion can involve a substantial amount of calculation.

By using the octal and hexadecimal systems, we avoid the problem of large numbers of digits, while gaining an important advantage over the decimal system – there are simple algorithms for converting between binary and octal, and between binary and hexadecimal. For this reason, bases 2 and 8 are described as *related bases*. Similarly, bases 2 and 16 are related.

To convert a number from binary to octal, group the bits into sets of 3 on either side of the point. Each group of 3 bits corresponds to 1 digit in the octal representation.

Example 2.4.3 Convert 10100011.10111_2 to octal.

Solution

$$\underbrace{10}_{2}\,\underbrace{100}_{4}\,\underbrace{011}_{3}.\underbrace{101}_{5}\,\underbrace{110}_{6}$$

$10100011.10111_2 = 243.56_8$

Converting from octal to binary is equally easy – replace each octal digit by its 3-bit binary representation. (If an octal digit is less than 4, its 3-bit binary representation will begin with one or more 'leading' zeros – don't leave them out!)

Example 2.4.4 Convert 514.7_8 to binary.

Solution

$$\overbrace{101}^{5}\,\overbrace{001}^{1}\,\overbrace{100}^{4}.\overbrace{111}^{7}$$

$$514.7_8 = 101001100.111_2$$

Conversion between binary and hexadecimal is similar, except that each hexadecimal digit corresponds to *four* bits.

Example 2.4.5 Convert 10111101001.110001_2 to hexadecimal.

Solution

$$\underbrace{101}_{5}\,\underbrace{1110}_{E}\,\underbrace{1001}_{9}\,.\,\underbrace{1100}_{C}\,\underbrace{0100}_{4}$$

$$10111101001.110001_2 = 5E9.C4_{16}$$

Example 2.4.6 Convert $B2.5D6_{16}$ to binary.

Solution

$$\overbrace{1011}^{B}\,\overbrace{0010}^{2}\,.\,\overbrace{0101}^{5}\,\overbrace{1101}^{D}\,\overbrace{0110}^{6}$$

$$B2.5D6_{16} = 10110010.01011101011_2$$

The hexadecimal system is commonly used in computing to represent the contents of part of the memory or a binary file in human-readable form, since each byte (consisting of 8 bits) can be represented by 2 hexadecimal digits.

2.5 Arithmetic in non-decimal bases

The rules for adding, subtracting, multiplying and dividing numbers by hand in bases other than 10 are the same as the rules you learnt in primary school for the decimal system; it is only the tables that are different. In this section, we look at how binary arithmetic with natural numbers can be performed by hand. In Chapter 3, we will see how computers perform arithmetic in the binary system.

The addition table in the binary system is shown in Table 2.3.

Table 2.3

+	0	1
0	0	1
1	1	10

Numbers are added in the usual column-by-column fashion. Sometimes there will be a 1 to 'carry' to the next column.

For example:

$$\begin{array}{r} 11011_2 \\ 1110_2 \\ \hline 101001_2 \end{array}$$

In this example, the second column from the right gives 1 + 1 = 10, so there is a 1 to carry to the third column. Similarly, there is a carry from the third to the fourth column, from the fourth to the fifth column, and from the fifth to the sixth column.

This is essentially the method used by a computer to add natural numbers.

Subtraction can also be done by the usual method, as shown in the example below. (This is not the way most computers perform subtraction. We will see in Chapter 3 how subtraction is done on a computer.)

$$\begin{array}{r} 11011_2 \\ -\ 1110_2 \\ \hline 1101_2 \end{array}$$

The multiplication table in the binary system is shown in Table 2.4.

Table 2.4

×	0	1
0	0	0
1	0	1

A 'long multiplication' in binary is carried out without really doing any multiplications at all. (All you ever need to multiply by is 0 or 1.) Here is an example; notice that the row corresponding to multiplying by the fours bit of the multiplier is omitted because that bit is zero:

$$
\begin{array}{rr}
 & 11010_2 \\
\times & 1011_2 \\
\hline
 & 11010_2 \\
 & 11010_2 \\
 & 11010_2 \\
\hline
 & 100011110_2
\end{array}
$$

Long division is also straightforward, with the proviso that the fractional part of the result may be non-terminating, in which case you need to decide how many digits are required in the answer. At each step of the division, the divisor 'goes into' the number either once (if it is less than or equal to the number), or not at all (if it is greater than the number). For example:

$$
\begin{array}{rr}
101_2 \overline{)11101_2} & 1 \\
\underline{101_2} & \\
100_2 & 0 \\
\underline{0_2} & \\
1001_2 & 1 \\
\underline{101_2} & \\
100_2 &
\end{array}
$$

In this example, the integer quotient 101_2 has been calculated, leaving a remainder of 100_2.

Some calculators have the facility for converting between the bases 10, 2, 8 and 16, and for performing arithmetic in those bases. Usually only unsigned (positive) integers can be used in these calculations. Many computer algebra software packages provide more flexible facilities for working with numbers in different bases.

EXERCISES

1 Write the decimal number 394.27_{10} in expanded form.

2 Convert the following binary numbers to decimal by first writing them in expanded form:

(a) 1100101_2 (b) 1010111.1011_2

3 Convert the following numbers from decimal to binary:

(a) 826_{10} (b) 0.34375_{10}

(c) 1604.1875_{10} (d) -471.25_{10}

4 Convert the following numbers from decimal to binary, with 5 digits after the point:

(a) 0.2_{10} (b) 13.47_{10}

5 What is the effect on the value of a natural number if:

(a) 0 is appended to its binary representation?

(b) 1 is appended to its binary representation?

6 Convert the following octal and hexadecimal numbers to decimal:

(a) 4715_8 (b) 603.25_8

(c) $C6E_{16}$ (d) $2FA.8_{16}$

7 An efficient computational method for converting a natural number from a non-decimal base to decimal, known as Horner's method, is illustrated in the following example:

$$6253_8 = ((6 \times 8 + 2) \times 8 + 5) \times 8 + 3 = 3243_{10}$$

In words: multiply the first digit by the base, add the second digit, multiply by the base, add the third digit, multiply by the base, and so on. The final step is to add the last digit.

Use Horner's method to convert the following numbers to decimal:

(a) 7216_8 (b) 543517_8

(c) $8CB2_{16}$ (d) $E490DF_{16}$

8 Write Horner's method as an algorithm in pseudocode. Assume that the base of the number to be converted is input first, followed by the number itself, regarded as a list of digits in that base.

9 Convert the following decimal numbers to octal:

(a) 3842_{10} (b) 291.9375_{10}

10 Convert the following decimal numbers to hexadecimal:

(a) 29803_{10} (b) 6962.578125_{10}

11 Convert the following binary numbers to octal and hexadecimal:

(a) 1110100110_2 (b) 11000101.00111_2

12 Convert the following octal and hexadecimal numbers to binary:

(a) 247_8 (b) 31.63_8

(c) $93B_{16}$ (d) $AD.1C_{16}$

13 The contents of one byte (consisting of 8 bits) in a memory register of a computer can be represented by 2 hexadecimal digits. A *left shift* is an operation in which the 8 bits are moved one position to the left, the leftmost bit is lost, and a 0 is inserted in the rightmost position. A *right shift* is defined in a similar manner.

For each of the following bytes, find the result (in hexadecimal) of a left shift and a right shift:

(a) 3A (b) E7

14 Perform the following calculations in binary arithmetic:

(a) $1101101_2 + 1011110_2$

(b) $1001101_2 + 101011_2$

(c) $1110011_2 - 101101_2$

(d) $1100010_2 - 1010111_2$

(e) $10011_2 \times 1101_2$

(f) $11010_2 \times 10101_2$

(g) $110110_2 \div 1001_2$

(h) $10110_2 \div 11_2$ (3 digits after the point)

15 In the number system known as *balanced ternary*, each number (positive, zero or negative) is represented by a string of digits chosen from 1, 0, $\bar{1}$, where $\bar{1}$ denotes the number –1 regarded as a digit. The place values of the digits correspond to powers of 3; for example, $10\bar{1}1 = 1\times3^3 + 0\times3^2 + (-1)\times3^1 + 1\times3^0 = 25$.

(a) Write down the balanced ternary representations of the integers from –5 to 5.

(b) For any positive number n, what is the relationship between the ternary representations of n and $-n$? Justify your answer.