1 Data representation

In this chapter you will learn about:

- ★ number systems
  - how and why computers use binary to represent data
  - the denary, binary and hexadecimal number systems
  - converting numbers between denary, binary and hexadecimal
  - how and why hexadecimal is used for data representation
  - how to add two positive 8-bit numbers
  - overflow when performing binary addition
  - logical binary shifts on positive 8-bit integers
  - two's complement notation to represent positive and negative binary numbers
- ★ text, sound and images
  - how and why a computer represents text
  - the use of character sets including ASCII and Unicode
  - how and why a computer represents sound
  - sound sample rate and sample resolution
  - how and why a computer represents an image
  - the effects of the resolution and colour depth on images
- ★ data storage and compression
  - how data storage is measured
  - calculating the file size of an image and sound file
  - the purpose of and need for data compression
  - lossy and lossless compression.

This chapter considers the three key number systems used in computer science, namely binary, denary and hexadecimal. It also discusses how these number systems are used to measure the size of computer memories and storage devices, together with how sound and images can be represented digitally.

1.1 Number systems

1.1.1 Binary represents data

As you progress through this book you will begin to realise how complex computer systems really are. By the time you reach Chapter 10 you should have a better understanding of the fundamentals behind computers themselves and the software that controls them.

You will learn that any form of data needs to be converted into a binary format so that it can be processed by the computer.

However, no matter how complex the system, the basic building block in all computers is the binary number system. This system is chosen because it only consists of 1s and 0s. Since computers contain millions and millions of tiny 'switches', which must be in the ON or OFF position, they can be represented by the binary system. A switch in the ON position is represented by 1; a switch in the OFF position is represented by 0.

Switches used in a computer make use of logic gates (see Chapter 10) and are used to store and process data.

1.1.2 Binary, denary and hexadecimal systems

The binary system

We are all familiar with the denary number system which counts in multiples of 10. This gives us the well-known headings of units, 10s, 100s, 1000s, and so on:

(10^4)	(10^3)	(10^2)	(10^1)	(10^0)
10 000	**1000**	**100**	**10**	**1**
2	5	1	7	7

Denary uses ten separate digits, 0-9, to represent all values. Denary is known as a base 10 number system.

The **binary number system** is a base 2 number system. It is based on the number 2. Thus, only the two 'values' 0 and 1 can be used in this system to represent all values. Using the same method as denary, this gives the headings 2^0, 2^1, 2^2, 2^3, and so on. The typical headings for a binary number with eight digits would be:

(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	1	1	0	1	1	1	0

A typical binary number would be: 11101110.

Converting from binary to denary

The conversion from binary to denary is a relatively straightforward process. Each time a 1-value appears in a binary number column, the column value (heading) is added to a total. This is best shown by three examples which use 8-bit, 12-bit and 16-bit binary numbers:

Example 1

Convert the binary number, 11101110, into a denary number.

128	64	32	16	8	4	2	1
1	1	1	0	1	1	1	0

The equivalent denary number is 128 + 64 + 32 + 8 + 4 + 2 = 238

Example 2

Convert the following binary number, 011110001011, into a denary number.

2048	1024	512	256	128	64	32	16	8	4	2	1
0	1	1	1	1	0	0	0	1	0	1	1

The equivalent denary number is 1024 + 512 + 256 + 128 + 8 + 2 + 1 = 1931

Example 3

Convert the following binary number, 0011000111100110, into a denary number.

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	1	1	1	1	0	0	1	1	0

As with the two examples above, to convert this number to denary, each time a 1 appears in a column the column value is added to the total:

8192 + 4096 + 256 + 128 + 64 + 32 + 4 + 2 = 12 774

The same method can be used for a binary number of any size.

Activity 1.1

Convert the following binary numbers into denary:

a 0 0 1 1 0 0 1 1
b 0 1 1 1 1 1 1 1
c 1 0 0 1 1 0 0 1
d 0 1 1 1 0 1 0 0
e 1 1 1 1 1 1 1 1
f 0 0 0 0 1 1 1 1
g 1 0 0 0 1 1 1 1
h 1 0 1 1 0 0 1 1
i 0 1 1 1 0 0 0 0
j 1 1 1 0 1 1 1 0
k 0001 1110 0111
l 0101 0101 0100
m 1111 0000 1111
n 0111 1100 1000
o 0111 1111 1111
p 0111 1100 1111 0000
q 0011 1111 0000 1101
r 1100 0011 0011 1111
s 1000 1000 1000 1000
t 0111 1111 1111 1111

Converting from denary to binary

The conversion from denary numbers to binary numbers can be done in two different ways. The first method involves successive subtraction of powers of 2 (that is, 128, 64, 32, 16, and so on); whilst the second method involves successive division by 2 until the value "0" is reached. This is best shown by two examples:

Example 1

Consider the conversion of the denary number, 142, into binary:

Method 1

The denary number 142 is made up of 128 + 8 + 4 + 2 (that is, 142 – 128 = 14; 14 – 8 = 6; 6 – 4 = 2; 2 – 2 = 0; in each stage, subtract the largest possible power of 2 and keep doing this until the value 0 is reached. This will give us the following 8-bit binary number:

128	64	32	16	8	4	2	1
1	0	0	0	1	1	1	0

Method 2

This method involves successive division by 2. Start with the denary number, 142, and divide it by 2. Write the result of the division including the remainder (even if it is 0) under the 142 (that is, 142 ÷ 2 = 71 remainder 0); then divide again by 2 (that is, 71 ÷ 2 = 35 remainder 1) and keep dividing until the result is zero. Finally write down all the remainders in reverse order:

2	142				read the remainders from bottom to top to get the binary number:
2	71	remainder:	0		1 0 0 0 1 1 1 0
2	35	remainder:	1		
2	17	remainder:	1		
2	8	remainder:	1		
2	4	remainder:	0		
2	2	remainder:	0		
2	1	remainder:	0		
	0	remainder:	1		

▲ **Figure 1.1**

We end up with an 8-bit binary number which is the same as that found by Method 1.

Example 2

Consider the conversion of the denary number, 59, into binary:

Method 1

The denary number 59 is made up of 32 + 16 + 8 + 2 + 1 (that is, 59 – 32 = 27; 27 – 16 = 11; 11 – 8 = 3; 3 – 2 = 1; 1 – 1 = 0; in each stage, subtract the largest possible power of 2 and keep doing this until the value 0 is reached. This will give us the following 8-bit binary number:

128	64	32	16	8	4	2	1
0	0	1	1	1	0	1	1

Method 2

This method involves successive division by 2. Start with the denary number, 59, and divide it by 2. Write the result of the division including the remainder (even if it is 0) under the 59 (that is, 59 ÷ 2 = 29 remainder 1); then divide again by 2 (that is, 29 ÷ 2 = 14 remainder 1) and keep dividing until the result is zero. Finally write down all the remainders in reverse order:

2	59				write the remainders from bottom to top to get the binary number:
2	29	remainder:	1		1 1 1 0 1 1
2	14	remainder:	1		
2	7	remainder:	0		
2	3	remainder:	1		
2	1	remainder:	1		
	0	remainder:	1		

▲ **Figure 1.1b**

If we want to show this as an 8-bit binary number (as shown in Method 1), we now simply add two 0's from the left-hand side to give the result: 0 0 1 1 1 0 1 1. The two results from both methods clearly agree.

Both the above examples use 8-bit binary numbers. This third example shows how the method can still be used for any size of binary number; in this case a 16-bit binary number.

Example 3

Consider the conversion of the denary number, 35 000, into a 16-bit binary number:

Method 1

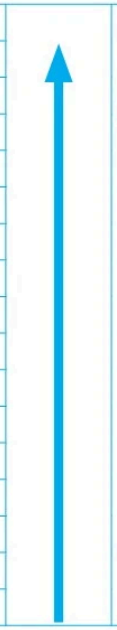
The denary number 35 000 is made up of 32 768 + 2048 + 128 + 32 + 16 + 8 (that is, 35000 – 32768 = 2232; 2232 – 2048 = 184; 184 – 128 = 56; 56 – 32 = 24; 24 – 16 = 8; 8 – 8 = 0; in each stage, subtract the largest possible power of 2 and keep doing this until the value 0 is reached. This will give us the following 16-bit binary number:

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0

Method 2

This method involves successive division by 2. Start with the denary number, 35000, and divide it by 2. Write the result of the division including the remainder (even if it is 0) under the 35000 (that is, 35000 ÷ 2 = 17500 remainder 0); then divide again by 2 (that is, 17500 ÷ 2 = 8750 remainder 0) and keep dividing until the result is zero. Finally write down all the remainders in reverse order:

2	35000		
2	17500	remainder:	0
2	8750	remainder:	0
2	4375	remainder:	0
2	2187	remainder:	1
2	1093	remainder:	1
2	546	remainder:	1
2	273	remainder:	0
2	136	remainder:	1
2	68	remainder	0
2	34	remainder	0
2	17	remainder	0
2	8	remainder	1
2	4	remainder	0
2	2	remainder	0
2	1	remainder	0
	0	remainder	1



read the remainder from bottom to top to get the binary number:

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

▲ **Figure 1.1c**