# 1 Binary systems and hexadecimal

In this chapter you will learn about:

- the binary system
- measurement of computer memories
- the hexadecimal system
- how to convert numbers between different number base systems

## 1.1 Introduction

As you progress through this book you will begin to realise how complex computer systems really are. By the time you reach Chapter 12 you should have a better understanding of the fundamentals behind computers themselves and the software that controls them.

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

## 1.2 The binary system

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

| 10 000 | 1000 | 100 | 10 | 1 |
|---|---|---|---|---|
| $(10^4)$ | $(10^3)$ | $(10^2)$ | $(10^1)$ | $(10^0)$ |

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

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

A typical binary number would be:

1 1 1 0 1 1 1 0

## 1.2.1 Converting from binary to denary

It is fairly straightforward to change a binary number into a denary number. Each time a 1 appears in a column, the column value is added to the total. For example, the binary number above is:

128 + 64 + 32 + 8 + 4 + 2 = 238 (denary)

The 0 values are simply ignored.

### 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 1 1 1 0 0 0 0
**i** 0 1 1 1 0 0 0 0
**j** 1 1 1 0 1 1 1 0

## 1.2.2 Converting from denary to binary

The reverse operation, converting from denary to binary, is slightly more complex. There are two basic ways of doing this. The first method is 'trial and error' and the second method is more methodical and involves repetitive division.

### Method 1

Consider the conversion of the denary number, 107, into binary. This method involves placing 1s in the appropriate position so that the total equates to 107:

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

### Method 2

This method involves successive division by 2. The remainders are then read from BOTTOM to TOP to give the binary value. Again using 107, we get:

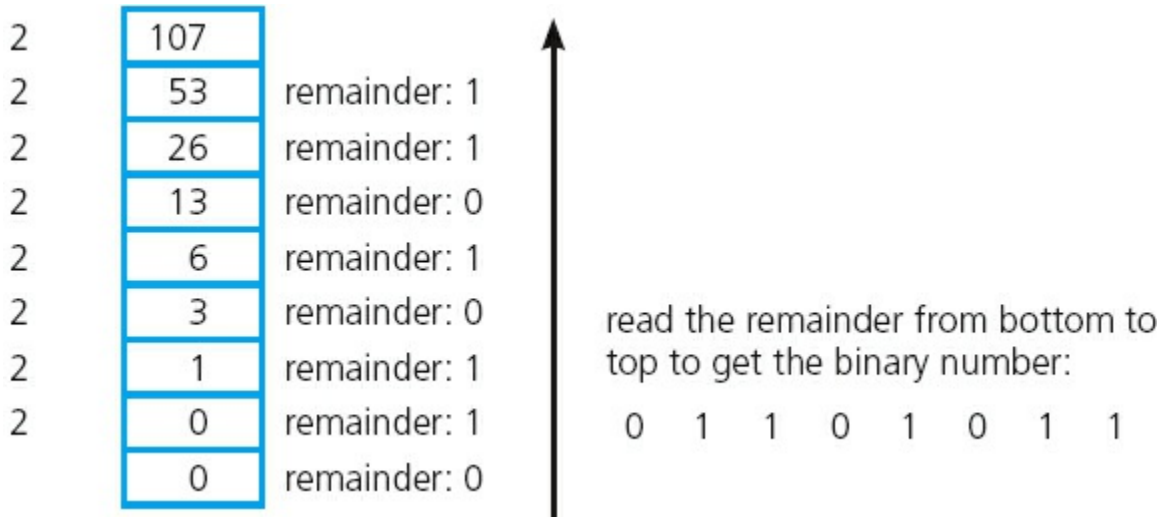


**Figure 1.1**

## Activity 1.2

Convert the following denary numbers into binary (using both methods):

**a** 4 1
**b** 6 7
**c** 8 6
**d** 1 0 0
**e** 1 1 1
**f** 1 2 7
**g** 1 4 4
**h** 1 8 9
**i** 2 0 0
**j** 2 5 5

# 1.3 Measurement of the size of computer memories

A **binary digit** is commonly referred to as a **BIT**; 8 bits are usually referred to as a **BYTE**.

The byte is the smallest unit of memory in a computer. Some computers use larger bytes but they are always multiples of 8 (e.g. 16-bit systems and 32-bit systems). One byte of memory wouldn't allow you to store very much information; therefore memory size is measured in the following multiples:

**Table 1.1**

| Name of memory size | Number of bits | Equivalent denary value |
|---|---|---|
| 1 kilobyte (1 KB) | $2^{10}$ | 1 024 bytes |

| | | |
|---|---|---|
| 1 megabyte (1 MB) | $2^{20}$ | 1 048 576 bytes |
| 1 gigabyte (1 GB) | $2^{30}$ | 1 073 741 824 bytes |
| 1 terabyte (1 TB) | $2^{40}$ | 1 099 511 627 776 bytes |
| 1 petabyte (1 PB) | $2^{50}$ | 1 125 899 906 842 624 bytes |

(Note: 1024 × 1024 = 1048576 and so on.)

To give some idea of the scale of these numbers, a typical data transfer rate using the internet is 32 megabits (i.e. 4 MB) per second (so a 40 MB file would take 10 seconds to transfer). Most hard disk systems in computers are 1 or 2 TB in size (so a 2 TB memory could store over half a million 4 MB photos, for example).

It should be pointed out here that there is some confusion in the naming of memory sizes. The IEC convention is now adopted by some organisations. Manufacturers of storage devices often use the denary system to measure storage size. For example,

1 kilobyte = 1000 byte

1 megabyte = 1000000 bytes

1 gigabyte = 1000000000 bytes

1 terabyte = 1000000000000 bytes and so on.

The IEC convention for computer internal memories (including RAM) becomes:

1 kibibyte (1 KiB) = 1024 bytes

1 mebibyte (1 MiB) = 1048576 bytes

1 gibibyte (1 GiB) = 1073741824 bytes

1 tebibyte (1 TiB) = 1099511627776 bytes and so on.

However, the IEC terms are not universally used and this textbook will use the more conventional terms shown in Table 1.1. This also ties up with the Cambridge International Examinations computer science syllabus which uses the same terminology as in Table 1.1.

## 1.4 Example use of binary

This section gives an example of a use of the binary system. We will introduce the idea of computer **REGISTERS**; this subject is covered in more depth in Chapter 4. A register is a group of bits; it is often depicted as follows:


**Figure 1.2**

When computers (or microprocessors) are used to control devices (such as robots), registers are used as part of the control system. The following example describes how registers can be used in controlling a simple device.

A robot vacuum cleaner has three wheels, A, B and C. A rotates on a spindle to allow for direction changes (as well as forward and backward movement); B and C are fixed to revolve around their axles to provide *only* forward and backward movement, and have an electric motor attached:

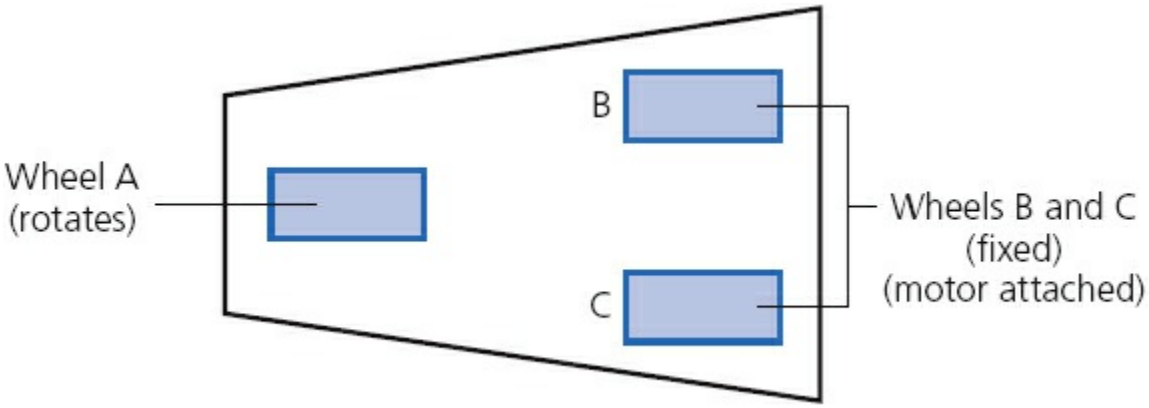


**Figure 1.3**

An 8-bit register is used to control the movement of the robot vacuum cleaner:

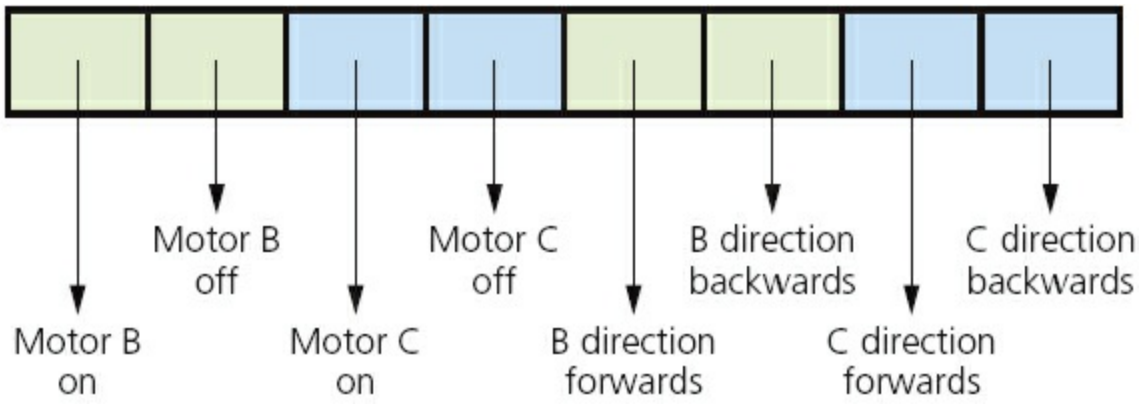


**Figure 1.4**

If the register contains 1 0 1 0 1 0 1 0 this means *'motor B is ON and motor C is ON and both motors are turning to produce FORWARDS motion'*. Effectively, the vacuum cleaner is moving forwards.

## Activity 1.3

**a** What would be the effect if the register contained the following values?

**i** 1 0 0 1 1 0 0 0

**ii** 1 0 1 0 0 1 0 1

**iii** 1 0 1 0 0 1 1 0

**b** What would the register contain if only motor C was ON and the motors were turning in a BACKWARDS direction?
**c** What would the register contain if motor B and motor C were both ON but B was turning in a backward direction and C was turning in a forward direction?
**d** What would be the effect if the register contained the following?
1 1 1 1 1 1 1 1

## 1.5 The hexadecimal system

The **HEXADECIMAL SYSTEM** is very closely related to the binary system. Hexadecimal (sometimes referred to as simply 'hex') is a base 16 system and therefore needs to use 16 different 'values' to represent each digit.

Because it is a system based on 16 different digits, the numbers 0 to 9 and the letters A to F are used to represent each hexadecimal (hex) digit. (A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15.) Using the same method as denary and binary, this gives the headings of $16^0$, $16^1$, $16^2$, $16^3$ and so on. The typical headings for a hexadecimal number with five digits would be:

| 65 536 | 4 096 | 256 | 16 | 1 |
|---|---|---|---|---|
| $(16^4)$ | $(16^3)$ | $(16^2)$ | $(16^1)$ | $(16^0)$ |

Since $16 = 2^4$ this means that FOUR binary digits are equivalent to each hexadecimal digit. Table 1.2 summarises the link between binary, hexadecimal and denary.

**Table 1.2**

| Binary value | Hexadecimal value | Denary value |
|---|---|---|
| 0 0 0 0 | 0 | 0 |
| 0 0 0 1 | 1 | 1 |
| 0 0 1 0 | 2 | 2 |
| 0 0 1 1 | 3 | 3 |
| 0 1 0 0 | 4 | 4 |
| 0 1 0 1 | 5 | 5 |
| 0 1 1 0 | 6 | 6 |
| 0 1 1 1 | 7 | 7 |
| 1 0 0 0 | 8 | 8 |
| 1 0 0 1 | 9 | 9 |
| 1 0 1 0 | A | 10 |

| | | |
|---|---|---|
| 1 0 1 1 | B | 11 |
| 1 1 0 0 | C | 12 |
| 1 1 0 1 | D | 13 |
| 1 1 1 0 | E | 14 |
| 1 1 1 1 | F | 15 |

## 1.5.1 Converting from binary to hexadecimal and from hexadecimal to binary

Converting from binary to hexadecimal is a fairly easy process. Starting from the right and moving left, split the binary number into groups of 4 bits. If the last group has less than 4 bits, then simply fill in with 0s from the left. Take each group of 4 bits and convert it into the equivalent hexadecimal digit using Table 1.2. Look at the following two examples to see how this works.

# Example 1

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

First split this up into groups of 4 bits:

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

Then, using Table 1.2, find the equivalent hexadecimal digits:

B  E  1

# Example 2

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

First split this up into groups of 4 bits:

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

The left group only contains 2 bits, so add in two 0s:

**0 0** 1 0  0 0 0 1  1 1 1 1  1 1 0 1

Now use Table 1.2 to find the equivalent hexadecimal digits:

2  1  F  D

## Activity 1.4

Convert the following binary numbers into hexadecimal:

**a** 1 1 0 0 0 0 1 1

**b** 1 1 1 1 0 1 1 1
**c** 1 0 0 1 1 1 1 1 1 1
**d** 1 0 0 1 1 1 0 1 1 1 0
**e** 0 0 0 1 1 1 1 0 0 0 0 1
**f** 1 0 0 0 1 0 0 1 1 1 1 0
**g** 0 0 1 0 0 1 1 1 1 1 1 1 0
**h** 0 1 1 1 0 1 0 0 1 1 1 0 0
**i** 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1
**j** 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0

Converting from hexadecimal to binary is also very straightforward. Using the data in Table 1.2, simply take each hexadecimal digit and write down the 4-bit code which corresponds to the digit.

# Example 3

4 5 A

Using Table 1.2, find the 4-bit code for each digit:

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

Put the groups together to form the binary number:

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

# Example 4

B F 0 8

Again just use Table 1.2:

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

Then put all the digits together:

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

## Activity 1.5

Convert the following hexadecimal numbers into binary:

**a** 6 C
**b** 5 9
**c** A A
**d** A 0 0
**e** 4 0 E
**f** B A 6
**g** 9 C C

**h** 4 0 A A
**i** D A 4 7
**j** 1 A B 0

## 1.5.2 Converting from hexadecimal to denary and from denary to hexadecimal

To convert a hexadecimal number to denary is fairly straightforward. Take each hexadecimal digit and multiply it by its value. Add the totals together to obtain the denary value.

### Example 1

4 5 A

First multiply each digit by its value:

| 256 | 16 | 1 | |
|---|---|---|---|
| $(4 \times 256 = 1024)$ | $(5 \times 16 = 80)$ | $(10 \times 1 = 10)$ | (Note: A = 10) |

Add the totals together:

denary number = 1 1 1 4

### Example 2

C 8 F

First multiply each digit by its value:

| 256 | 16 | 1 | |
|---|---|---|---|
| $(12 \times 256 = 3072)$ | $(8 \times 16 = 128)$ | $(15 \times 1 = 15)$ | (Note: C = 12 and F = 15) |

Add the totals together:

denary number = 3 2 1 5

### Activity 1.6

Convert the following hexadecimal numbers into denary:

**a** 6 B
**b** 9 C
**c** 4 A
**d** F F
**e** 1 F F
**f** A 0 1
**g** B B 4

**h** C A 8
**i** 1 2 A E
**j** A D 8 9

To convert from denary to hexadecimal is a little more difficult. As with the conversion from binary to denary, there are two very similar methods that can be used. Again, the first method is 'trial and error' and the second method is more methodical and involves repetitive division.

## Method 1

Consider the conversion of the denary number, 2004, into hexadecimal. This method involves placing hexadecimal digits in the appropriate position so that the total equates to 2004:

| 256 | 16 | 1 | |
|---|---|---|---|
| 7 | D | 4 | (Note: D = 13) |

A quick check shows that: (7 × 256) + (13 × 16) + (4 × 1) gives 2004.

## Method 2

This method involves successive division by 16. The remainders are then read from BOTTOM to TOP to give the hexadecimal value. Again using 2004, we get:

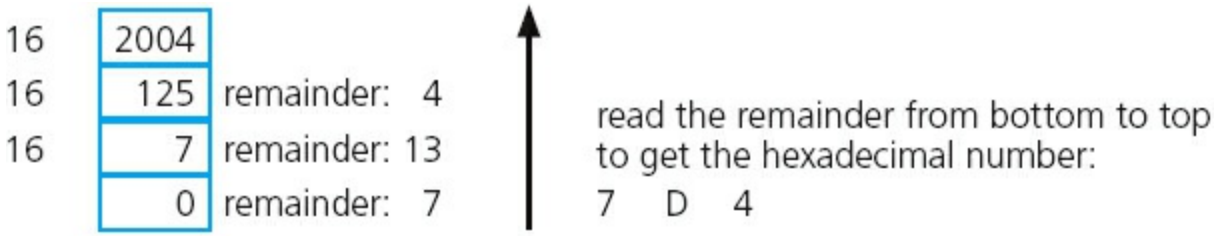


**Figure 1.5**

## Activity 1.7

Convert the following denary numbers into hexadecimal (using both methods):

**a** 9 8
**b** 2 2 7
**c** 4 9 0
**d** 5 1 1
**e** 8 2 6
**f** 1 0 0 0
**g** 2 6 3 4

h 3 7 4 3
i 4 0 0 7
j 5 0 0 0

## 1.6 Use of the hexadecimal system

This section reviews five uses of the hexadecimal system. The information in this chapter gives the reader sufficient grounding in each topic at this level. Further material can be found by searching the internet, but be careful that you don't go off at a tangent.

### 1.6.1 Memory dumps

Since it is much easier to work with: B 5 A 4 1 A F C

rather than: 1 0 1 1 | 1 0 0 1 | 1 0 1 0 | 0 1 0 0 | 0 0 0 1 | 1 0 1 0 | 1 1 1 1 | 1 1 0 0

hexadecimal is often used when developing new software or when trying to trace errors in programs. The contents of part of the computer memory can hold the key to help solve many problems. When the memory contents are output to a printer or monitor, this is known as a **MEMORY DUMP**:

```
00990F60   54 68 69 73 20 69 73 20 61 6E 20 65 78 61 6D 70 6C 65 20 6F 66
00990F77   61 20 6D 65 6D 6F 72 79 20 64 75 6D 70 20 66 72 6F 6D 20 20 61
00990E8E   74 79 70 69 63 61 6C 20 20 63 6F 6D 70 75 74 65 72 20 20 6D 85
00990EA5   6D 6F 72 79 20 73 68 6F 77 69 6E 67 20 74 68 65 20 20 63 6F 6E
00990EBC   74 65 6E 74 73 20 6F 66 20 61 20 6E 75 6D 62 65 72 20 20 6F 66
00990ED3   6C 6F 63 61 74 69 6F 6E 73 20 20 69 6E 20 20 68 65 78 20 20 20
00990EEA   6E 6F 74 61 74 69 6F 6E 20 20 00 00 00 00 00 00 00 00 00 00 00
```

**Figure 1.6**

A program developer can look at each of the hexadecimal codes (as shown in Figure 1.6) and determine where the error lies. The value on the far left shows the memory location so that it is possible to find out exactly where in memory the fault occurs. This is clearly much more manageable using hexadecimal rather than using binary. It's a very powerful fault-tracing tool, but requires considerable knowledge of computer architecture in order to interpret the results.

### 1.6.2 HyperText Mark-up Language (HTML)

**HYPERTEXT MARK-UP LANGUAGE (HTML)** is used when writing and developing web pages. HTML isn't a programming language but is simply a mark-up language. A mark-up language is used in the processing, definition and presentation of text (for example, specifying the colour of the text).

HTML uses <tags> which are used to bracket a piece of code; for example, <td> starts a standard cell in an HTML table, and </td> ends it. Whatever is between the two tags has been defined. Here is a short section of HTML code:

```
<tr>
    <td><h3>Small car</h3>
      <h3>Used car sales</h3>
      <h2>Cars from $500</h2>
      <br><h2>Cash sales only</h2></td></br>
</tr>
<table border="1">
  <colgroup>
  <col span="2" style="background-color:red">
  <col style="background-color:yellow">
  </colgroup>
```

HTML code is often used to represent colours of text on the computer screen. The values change to represent different colours. The different intensity of the three primary colours (red, green and blue) is determined by its hexadecimal value. For example:

- # FF 00 00 represents primary colour **red**
- # 00 FF 00 represents primary colour **green**
- # 00 00 FF represents primary colour **blue**
- # FF 00 FF represents **fuchsia**
- # FF 80 00 represents **orange**
- # B1 89 04 represents **tan**

and so on producing almost any colour the user wants. There are many websites available that allow a user to find the HTML code for the colour needed.

## Activity 1.8

Using the internet, find the HTML codes for a number of colours.
Try entering HTML code into the computer and see how the colours and font types can be changed to good effect.
Make use of websites, such as www.html.am/ to produce your own web pages.

With a little practice, you can import/embed images into your own design of web page using freely available software.
Remember this is not a programming language. It is simply a mark-up language, so very little programming skill is required to use HTML.

## 1.6.3 Media Access Control (MAC)

A **Media Access Control (MAC) address** refers to a number which uniquely identifies a device on the internet. The MAC address refers to the network interface card (NIC) which is part of the device. The MAC address is rarely changed so that a particular device can always be identified no matter where it is.

A MAC address is usually made up of 48 bits which are shown as six groups of hexadecimal digits (although 64-bit addresses are also known):

NN – NN – NN – DD – DD – DD
or
NN:NN:NN:DD:DD:DD

where the first half (NN – NN – NN) is the identity number of the manufacturer of the device and the second half (DD – DD – DD) is the serial number of the device. For example: 00 – 1C – B3 – 4F – 25 – FE is the MAC address of a device produced by the Apple Corporation (code: 001CB3) with a serial number of 4F25FE. Sometimes lower case hexadecimal letters are used in the MAC address: 00-1c-b3-4f-25-fe. Other manufacturer identity numbers include:

- 00 – 14 – 22 which identifies devices made by Dell
- 00 – 40 – 96 which identifies devices made by Cisco
- 00 – A0 – C9 which identifies devices made by Intel, and so on.

### Types of MAC address

It should be pointed out that there are two types of MAC address: the **Universally Administered MAC Address (UAA)** and the **Locally Administered MAC Address (LAA)**.

The UAA is by far the most common type of MAC address and this is the one set by the manufacturer at the factory. It is rare for a user to want to change this MAC address.

However, there are some occasions when a user or an organisation wishes to change their MAC address. This is a relatively easy task to carry out but it will cause big problems if the changed address isn't unique.

There are a few reasons why the MAC address needs to be changed using LAA:

- Certain software used on mainframe systems needs all the MAC addresses of devices to fall into a strict format; because of this, it may be necessary to change the MAC address of some devices to ensure they follow the correct format.
- It may be necessary to bypass a MAC address filter on a router or a firewall; only

MAC addresses with a certain format are allowed through, otherwise the devices will be blocked.

- To get past certain types of network restrictions it may be necessary to emulate unrestricted MAC addresses; hence it may require the MAC address to be changed on certain devices connected to the network.

## 1.6.4 Web addresses

Each character used on a keyboard has what is known as an **ASCII CODE (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)**. These codes can be represented using hexadecimal values or decimal values. Figure 1.7 shows part of an ASCII table.

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|
| 32 | 20 | <SPACE> | 64 | 40 | @ | 96 | 60 | ` |
| 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | <DELETE> |

**Figure 1.7**

A good example of the use of ASCII codes is the representation of a web address (or URL, which stands for uniform resource locator) such as www.hodder.co.uk which becomes (using hexadecimal values):

%77 %77 %77 %2E %68 %6F %64 %64 %65 %72 %2E %63 %6F %2E %75 %6B

w w w . h o d d e r . c o . u k

(Note: the % sign is used to denote that hexadecimal is being used.)

## Activity 1.9

Using the ASCII code table (Figure 1.7) convert the following URLs into the

equivalent hexadecimal:

**a** www.cie.org.uk
**b** www.cie.org.uk/computer_science
**c** https://www.hodder.co.uk
**d** www.HodderEducation.co.uk
**e** http://www.ucles.ac.uk/computing.htm

Sometimes the hexadecimal addresses are used in the address of files or web pages as a security feature. It takes longer to type in the URL using the hexadecimal codes, but it has the advantage that you are unlikely to fall into the trap of copying and pasting a 'fake' website address.

## 1.6.5 Assembly code and machine code

Computer memory can be referred to directly using machine code or assembly code. This can have many advantages to program developers or when carrying out troubleshooting.

Machine code and assembly code are covered in much more detail in Chapter 7; here we are simply interested in how hexadecimal fits into the picture.

Using hexadecimal makes it much easier, faster and less error prone to write code compared to binary. Using true machine code (which uses binary) is very cumbersome and it takes a long time to key in the values. It is also very easy to mistype the digits in a 'sea of 1s and 0s'. Here is a simple example:

STO FFA4 (assembly code)

A5E4 FFA4 (machine code using hexadecimal values)

1010 0101 1110 0100 1111 1111 1010 0100 (machine code using binary)

Machine code and assembly code are examples of low-level languages and are used by software developers when producing, for example, computer games. As you will find in Chapter 7, although they look cumbersome, they have many advantages at the development stage of software writing (especially when trying to locate errors in the code).