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:

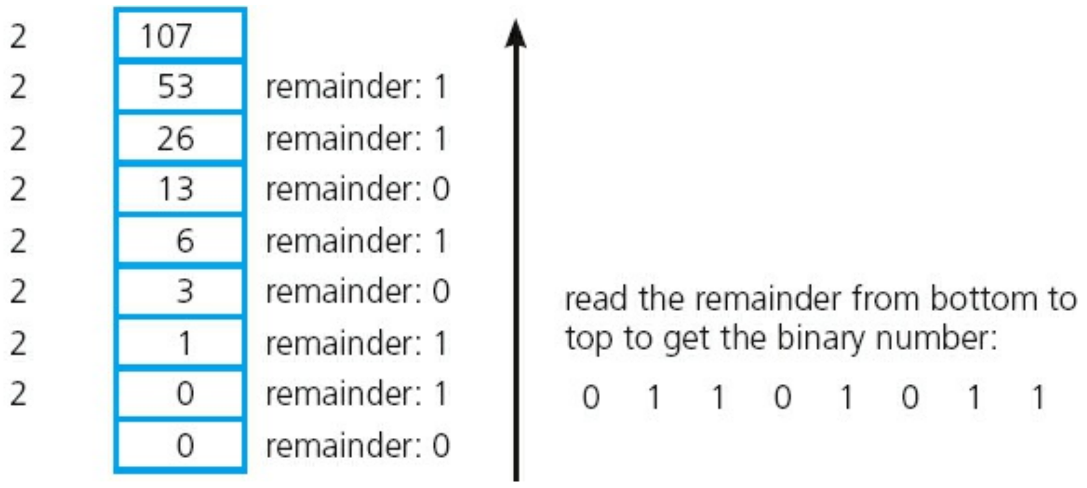


Figure 1.1

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:

1	0	0	1	0	1	1	1

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:

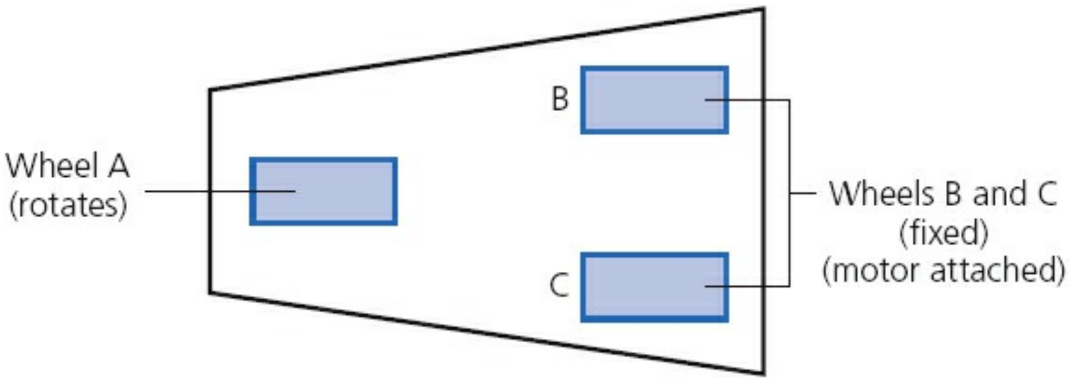


Figure 1.3

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

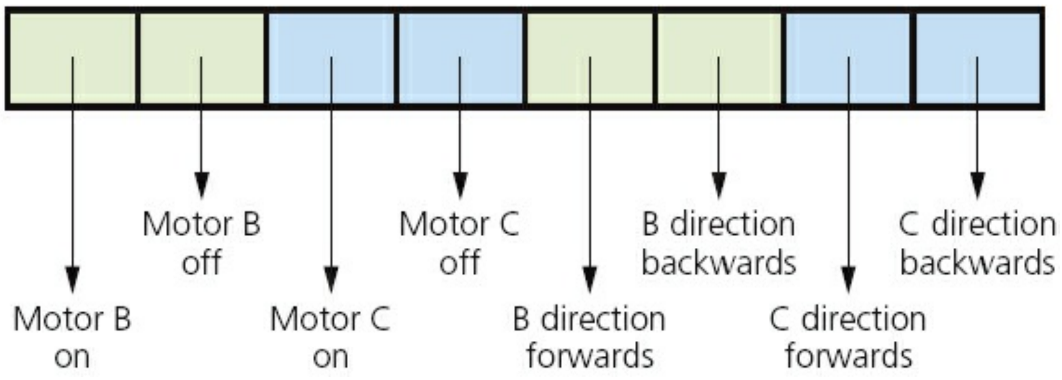


Figure 1.4

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 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:

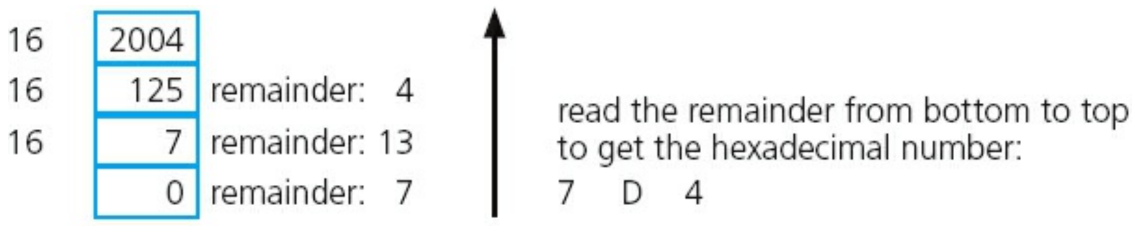

16 2004
16 125 remainder: 4
16 7 remainder: 13
0 remainder: 7
read the remainder from bottom to top
to get the hexadecimal number:
7 D 4

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

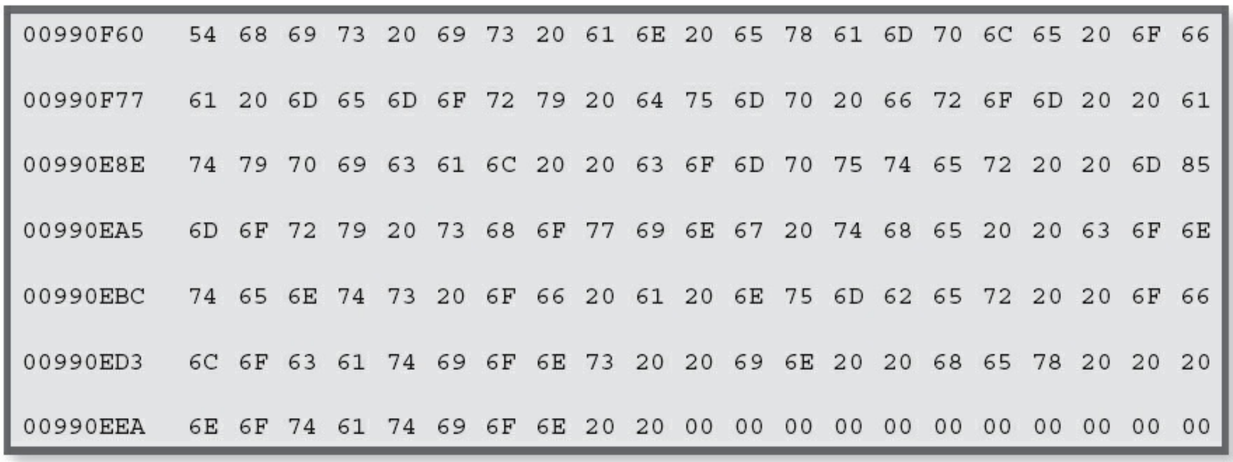
This section reviews five uses of the hexadecimal system. The information in this chapter gives the reader sufficient grounding in each topic at this level. Further material can be found by searching the internet, but be careful that you don't go off at a tangent.

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).

2 Communication and internet technologies

In this chapter you will learn about:

- serial and parallel transmission
- error checking after transmission
- web browsers and internet service providers
- http and HTML

2.1 Introduction

When data is sent from one device to another, it is important to consider how that data is transmitted. It is also important to ensure that the data hasn't been changed in any way.

The internet has now become an integral part of all of our lives. This chapter will consider some of the important technologies going on in the background which support the internet.

2.2 Data transmission

Data transmission can be either over a short distance (for example, from computer to printer) or over longer distances (for example, over a telephone network). Essentially, three factors need to be considered when transmitting data (each factor has to be agreed by both sender and receiver for this to work without error):

- the direction of the data transmission (i.e. in one direction only or in both directions)
- the method of transmission (how many bits are sent at the same time)
- the method of synchronisation between the two devices.

2.2.1 Simplex, half-duplex and full-duplex

simplex data transmission is in *one direction* only (i.e. from sender to receiver). Example: data being sent from a computer to a printer.

Half-duplex data transmission is in *both directions* but *not* at the same time (i.e. data can be sent from 'A' to 'B' or from 'B' to 'A' along the same line, but not at the same time). Example: a phone conversation between two people where only one person speaks at a time.

Full-duplex data transmission is in *both directions simultaneously* (i.e. data can be sent from 'A' to 'B' and from 'B' to 'A' along the same line, *both at the same time*). Example: broadband connection on a phone line.

2.2.2 Serial and parallel data transmission

Serial data transmission is when data is sent, *one bit at a time*, over *a single wire or channel* (bits are sent one after the other in a single stream).

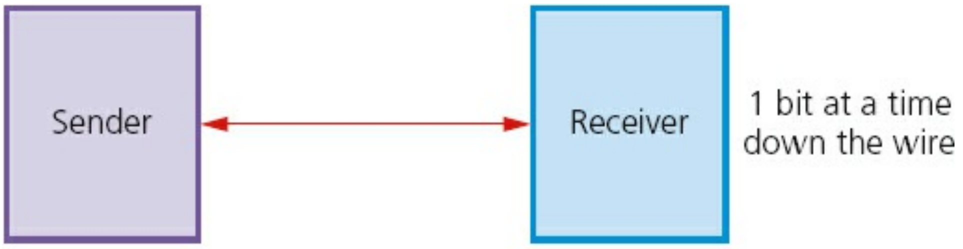


Figure 2.1

(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

This method of data transmission works well over long distances. However, data is transmitted at a slower rate than parallel data transmission. Since only one wire or channel is used, there is no problem of data arriving at its destination out of synchronisation.

An example of its use is sending data from a computer to a modem for transmission over a telephone line.

Parallel data transmission is when *several bits of data (usually 1 byte)* are sent down *several wires or channels at the same time*; one wire or channel is used to transmit each bit.

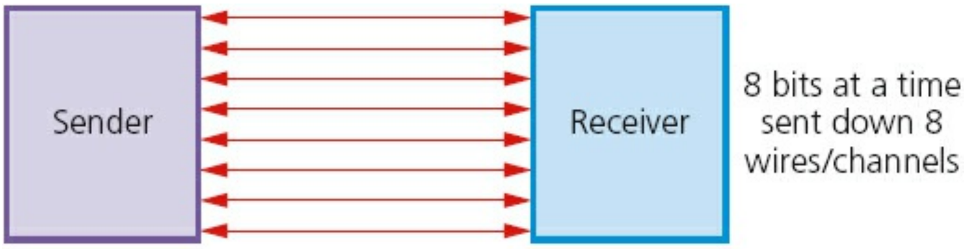


Figure 2.2

(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

This method of data transmission works very well over short distances (over longer distances, the bits can become 'skewed' – this means they will no longer be synchronised). It is, however, a faster method of data transmission than serial.

An example of its use is when sending data to a printer from a computer using a ribbon connector.

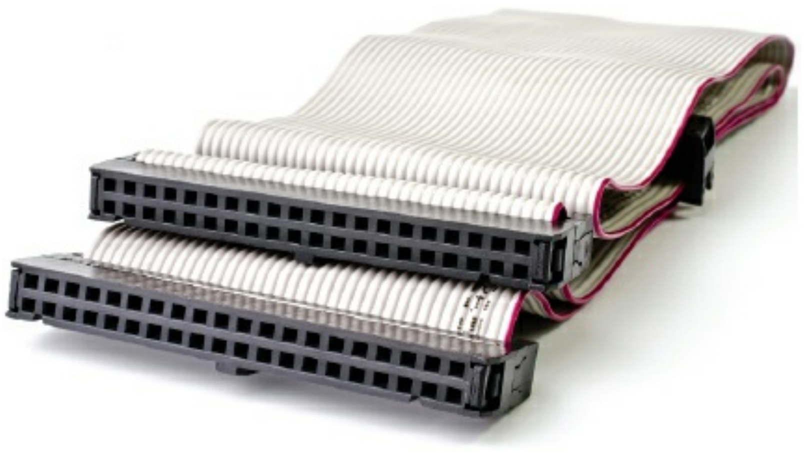


Figure 2.3 Ribbon connector

Activity 2.1

Describe what is meant by:

a serial, half-duplex data transmission
b parallel, full-duplex data transmission
c serial, simplex data transmission.

A common use for serial data transmission is discussed in Section 2.2.4 (Universal Serial Bus (USB)).

Parallel data transmission is used in the internal electronics of the computer system. The pathways between the CPU and the memory all use this method of data transmission. Integrated circuits, buses and other internal components all use parallel data transmission because of the need for high speed data transfer. The use of 8-bit, 16-bit, 32-bit and 64-bit buses, for example, allow much faster data transmission rates than could be achieved with single channel serial data transfer. An internal clock is used to ensure the correct timing of data transfer; it is essentially synchronous in nature (see Section 2.2.3) and the short distances between components mean that none of the issues described earlier have any real impact on the accuracy of the data.

Chapter 4 covers the internal architecture of computer systems (including the role of buses) and this should be read in conjunction with the information given above.

2.2.3 Asynchronous and synchronous data transmission

ASYNCHRONOUS DATA TRANSMISSION refers to data being transmitted in an agreed bit pattern. Data bits (1s and 0s) are grouped together and sent with **CONTROL BITS**:

start bit	1	0	1	1	1	1	0	0	1	1	1	0	1	0	1	0	stop bit
control bit																	control bit

Figure 2.4

This means that the receiver of the data knows when the data starts and when it ends. This prevents data becoming mixed up; without these control bits, it would be impossible to separate groups of data as they arrived.

SYNCHRONOUS DATA TRANSMISSION is a continuous stream of data (unlike asynchronous data which is sent in discrete groups). The data is accompanied by timing signals generated by an internal clock. This ensures that the sender and receiver are synchronised with each other.

The receiver counts how many bits (1s and 0s) were sent and then reassembles them into bytes of data. The timing must be very accurate here since there are no control bits sent in this type of data transmission. However, it is a faster data transfer method than asynchronous and is therefore used where this is an important issue (for example, in network communications).

2.2.4 Universal Serial Bus (USB)

The **UNIVERSAL SERIAL BUS (USB)** is an asynchronous serial data transmission method. It has quickly become the standard method for transferring data between a computer and a number of devices. Essentially, the USB cable consists of:

- a four-wire shielded cable
- two of the wires are used for power and the earth
- two of the wires are used in the data transmission.

When a device is plugged into a computer using one of the USB ports:

- the computer automatically detects that a device is present (this is due to a small change in the voltage level on the data signal wires in the cable)
- the device is automatically recognised, and the appropriate **DEVICE DRIVER** is loaded up so that computer and device can communicate effectively
- if a new device is detected, the computer will look for the device driver which matches the device; if this is not available, the user is prompted to download the appropriate software.

Figure 2.5 USB cable

Even though the USB system has become the industrial standard, there are still a number of benefits (✓) and drawbacks (✗) to using this system:

Table 2.1

✓	✗
Devices plugged into the computer are automatically detected; device drivers are automatically uploaded	–
The connectors can only fit one way; this prevents incorrect connections being made	The maximum cable length is presently about 5 metres
This has become the industry standard; this means that considerable support is available to users	–
Several different data transmission rates are supported	The present transmission rate is limited to less than 500 megabits per second
Newer USB standards are backward compatible with older USB standards	The older USB standard (e.g. 1.1) may not be supported in the near future

2.3 Error-checking methods

Following data transmission, there is always the risk that the data has been corrupted or changed in some way. This can occur whether data is being transmitted over short distances or over long distances.

Checking for errors is important since computers aren't able to check that text is correct; they can only recognise whether a word is in their built-in dictionary or not. Look at the following text:

> Can you raed tihs?
>
> 'I cnduo't bvleiee taht I culod aulaclty uesdtannrd waht I was rdnaieg. Unisg the icndeblire pweor of the hmuan mnid, aocdcrnig to rseecrah at Cmabridge Uinervtisy, it dseno't mttaer in waht oderr the lterets in a wrod are, the olny irpoamtnt tihng is taht the frsit and lsat ltteer be in the rhgit pclae. The rset can be a taotl mses and you can sitll raed it whoutit a pboerlm.
>
> Tihs is bucseae the huamn mnid deos not raed ervey ltteer by istlef, but the wrod as a wlohe.
>
> Aaznmig, huh? Yeah and I awlyas tghhuot slelinpg was ipmorantt! See if yuor fdreins can raed tihs too'
>
> *(From an unknown source at Cambridge University)*

Whilst you probably had little problem understanding this text, a computer would be unable to make any sense of it.

This is why error checking is such an important part of computer technology. This section considers a number of ways that can be used to check for errors so that you don't end up with text as shown in the example above!

A number of methods exist which can detect errors and, in some cases, actually correct the error. The methods covered in this section are:

- parity checking
- automatic repeat request (ARQ)
- checksum
- echo checking.

2.3.1 Parity checking

PARITY CHECKING is one method used to check whether data has been changed or corrupted following transmission from one device or medium to another device or medium.

A byte of data, for example, is allocated a **PARITY BIT**. This is allocated before transmission takes place. Systems that use **EVEN PARITY** have an even number of 1-bits; systems that use **ODD PARITY** have an odd number of 1-bits.
Consider the following byte:

	1	1	0	1	1	0	0

parity bit

Figure 2.6

If this byte is using even parity, then the parity bit needs to be 0 since there is already an even number of 1-bits (in this case, 4).

If odd parity is being used, then the parity bit needs to be 1 to make the number of 1-bits odd.

Therefore, the byte just before transmission would be:

either (even parity)

0	1	1	0	1	1	0	0

parity bit

Figure 2.7

or (odd parity)

1	1	1	0	1	1	0	0

parity bit

Figure 2.8

Before data is transferred, an agreement is made between sender and receiver regarding which of the two types of parity are used. This is an example of a **PROTOCOL**.

Activity 2.2

Find the parity bits for each of the following bytes:

a 1 1 0 1 1 0 1	even parity used
b 0 0 0 1 1 1 1	even parity used
c 0 1 1 1 0 0 0	even parity used
d 1 1 1 0 1 0 0	odd parity used
e 1 0 1 1 0 1 1	odd parity used

If a byte has been transmitted from ‘A’ to ‘B’, and even parity is used, an error would be flagged if the byte now had an odd number of 1-bits at the receiver’s end.

Example 1

sender’s byte:

0	1	0	1	1	1	0	0

receiver’s byte:

0	1	0	0	1	1	0	0

Figure 2.9

In this case, the receiver’s byte has three 1-bits, which means it now has odd parity whilst the byte from the sender had even parity (four 1-bits). This clearly means an error has occurred during the transmission of the data.

The error is detected by the computer recalculating the parity of the byte sent. If even parity has been agreed between sender and receiver, then a change of parity in the received byte indicates that a transmission error has occurred.

Activity 2.3

Which of the following bytes have an error following data transmission?

a	1 1 1 0 1 1 0 1	even parity used
b	0 1 0 0 1 1 1 1	even parity used
c	0 0 1 1 1 0 0 0	even parity used
d	1 1 1 1 0 1 0 0	odd parity used
e	1 1 0 1 1 0 1 1	odd parity used

In each case where an error occurs, can you work out which bit is incorrect?

Naturally, *any* of the bits in Example 1 could have been changed leading to a transmission error. Therefore, even though an error has been flagged, it is impossible to know *exactly* which bit is in error. (Your last answer in Activity 2.3 should have been ‘NO’ since there isn’t enough information to determine which bit has been changed.)

One of the ways around this problem is to use **PARITY BLOCKS**. In this method, a block of data is sent and the number of 1-bits are totalled horizontally and vertically (in other words, a parity check is done in both horizontal and vertical directions). As Example 2 shows, this method not only identifies that an error has occurred but also indicates where the error is.

Example 2

In this example, nine bytes of data have been transmitted. Agreement has been made that even parity will be used. Another byte, known as the **PARITY BYTE**, has also been sent. This byte consists entirely of the parity bits produced by the vertical parity check. The parity byte also indicates the end of the block of data.

The following table shows how the data arrived at the receiving end:

Table 2.2

	parity bit	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
byte 1	1	1	1	1	0	1	1	0
byte 2	1	0	0	1	0	1	0	1
byte 3	0	1	1	1	1	1	1	0
byte 4	1	0	0	0	0	0	1	0
byte 5	0	1	1	0	1	0	0	1
byte 6	1	0	0	0	1	0	0	0
byte 7	1	0	1	0	1	1	1	1
byte 8	0	0	0	1	1	0	1	0
byte 9	0	0	0	1	0	0	1	0
parity byte	1	1	0	1	0	0	0	1

A careful study of Table 2.2 shows the following:

- byte 8 (row 8) has incorrect parity (there are three 1-bits)
- bit 5 (column 5) also has incorrect parity (there are five 1-bits).

First of all, the table shows that an error has occurred following data transmission.

Secondly, at the intersection of row 8 and column 5, the position of the incorrect bit value (which caused the error) can be found.

This means that byte 8 should have the value:

0	0	0	1	0	0	1	0

Figure 2.10

which would also correct column 5 giving an even vertical parity (now has four 1-bits).

This byte could therefore be corrected automatically as shown above, or an error message could be relayed back to the sender asking them to retransmit the block of data.

One final point: if two of the bits change value following data transmission, it may be impossible to locate the error using the above method. For example, using Example 1 again:

0	1	0	1	1	1	0	0

Figure 2.11

This byte could reach the destination as:

0	1	1	1	1	1	0	1

Figure 2.12

or:

0	1	0	1	0	0	0	0

Figure 2.13

or:

0	1	0	1	0	1	1	0

Figure 2.14

All three are clearly incorrect; but they have retained even parity so this wouldn't have triggered an error message at the receiving end. Clearly we need to look at other methods to complement parity when it comes to error checking transmitted data.

Activity 2.4

The following block of data was received after transmission from a remote computer; odd parity being used by both sender and receiver. One of the bits has been changed during the transmission stage. Locate where this error is and suggest a corrected byte value.

Table 2.3

	parity bit	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
byte 1	0	1	1	0	0	0	1	0
byte 2	1	0	1	1	1	1	1	1
byte 3	1	0	0	1	1	0	0	0
byte 4	0	1	1	0	1	0	1	0
byte 5	1	1	1	0	0	1	1	0
byte 6	1	0	0	0	0	1	0	1
byte 7	0	1	1	1	0	0	0	0
byte 8	0	0	0	0	0	0	0	1
byte 9	0	1	1	1	1	0	1	0
parity byte	1	0	1	1	1	1	0	0

2.3.2 Automatic Repeat Request (ARQ)

Automatic Repeat ReQuest (ARQ) is another method used to check whether data has been correctly transmitted.

It uses an **acknowledgement** (a message sent by the receiver indicating that data has been received correctly) and **timeout** (this is the time allowed to elapse before an acknowledgement is received).

If an acknowledgement isn't sent back to the sender before timeout occurs, then the message is automatically resent.

2.3.3 Checksum

Checksum is another way to check if data has been changed or corrupted following data transmission. Data is sent in blocks and an additional value, the checksum, is also sent at the end of the block of data.

To explain how this works, we will assume the checksum of a block of data is 1 byte in length. This gives a maximum value of $2^8 - 1$ (i.e. 255). The value 0000 0000 is ignored in this calculation. Example 3 explains how a checksum is generated.

Example 3

If the sum of all the bytes in the transmitted block of data is <= 255, then the checksum is this value.

However, if the sum of all the bytes in the data block > 255, then the checksum is found using the simple algorithm in Figure 2.15.

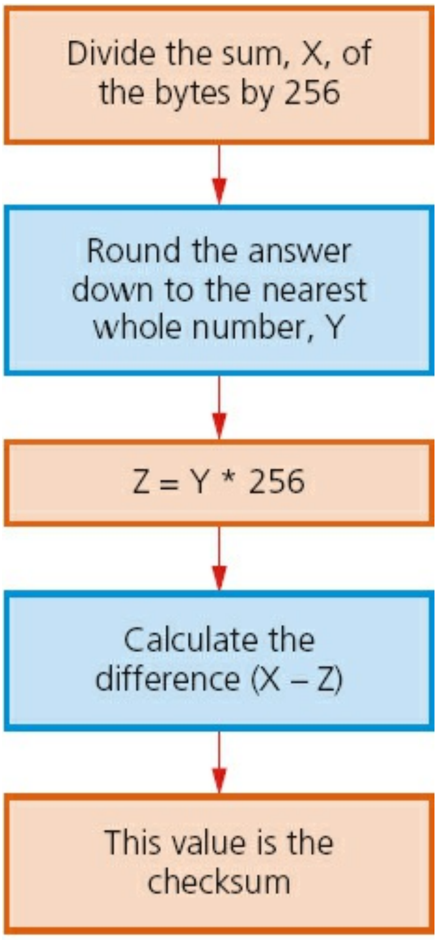

Divide the sum, X, of the bytes by 256
Round the answer down to the nearest whole number, Y
Z = Y * 256
Calculate the difference (X – Z)
This value is the checksum

Figure 2.15

Suppose the value of X is 1185, then tracing through the algorithm, we get:

X = 1185

1 1185/256 = 4.629
2 Rounding down to nearest whole number gives Y = 4
3 Multiplying by 256 gives Z = Y * 256 = 1024
4 The difference (X – Z) gives the checksum: (1185 – 1024) = 161
5 This gives the checksum = 161

When a block of data is about to be transmitted, the checksum for the bytes is first of all calculated. This value is then transmitted with the block of data. At the receiving end, the checksum is recalculated from the block of data received. This calculated value is then compared to the checksum transmitted. If they are the same value, then the data was transmitted without any errors; if the values are different, then a request is sent for the data to be retransmitted.

Activity 2.5

Calculate the checksum for blocks of data with the following byte sums:

a 148
b 905
c 1450
d 4095

2.3.4 Echo check

With **ECHO CHECK**, when data is sent to another device, this data is sent back again to the sender. The sender compares the two sets of data to check if any errors occurred during the transmission process.

As you will have no doubt worked out, this isn't very reliable. If the two sets of data are different, it isn't known whether the error occurred when sending the data in the first place, or if the error occurred when sending the data back for checking!

However, if no errors occurred then it is another way to check that the data was transmitted correctly.

2.4 Internet technologies

The internet is a world-wide system of computer networks and computers. All computers attached to the internet can communicate with each other providing a number of rules and protocols are adhered to.

2.4.1 Internet Service Provider (ISP)

Each user makes use of an **INTERNET SERVICE PROVIDER (ISP)**; these are companies that provide the user with access to the internet. A monthly fee is usually charged for this service. The ISP will set up a user account which will contain a username and a password; most ISPs also give the user an email address.

Before ISPs became common in the 1990s, internet access was usually limited to users who were part of a university or a government agency.

2.4.2 Internet Protocol (IP) Address

Each device on the internet is given a unique address known as the **INTERNET PROTOCOL (IP) ADDRESS**. This is a 32-bit number which is usually written in the form:

109.108.158.1

A home computer is given an IP address when it connects to the internet. This is assigned by the ISP and is unique for that particular internet session. The only IP addresses that remain fairly unchanged are web servers.

An IP address can be used instead of typing in the full URL. For example:

http://109.108.158.1

would take you straight to the device corresponding to this address.

IP addresses and MAC addresses

You will recall the term **MEDIA ACCESS CONTROL (MAC) ADDRESS** from Chapter 1. This is a unique number that identifies a device connected to the internet. So what is the difference between an IP address and a MAC address? The IP address gives the location of a device on the internet, whereas the MAC address identifies the device connected to the internet.

You can think of the IP address as the address of the house you live in (it will have some unique way of identifying it, such as a post or zip code). Using this example, the MAC address can be thought of as a way of uniquely identifying each person living in that house. It is possible to move house (so your IP address will change) but the same people will be living in the new house (so their MAC addresses will remain unchanged).

2.4.3 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 structure and presentation

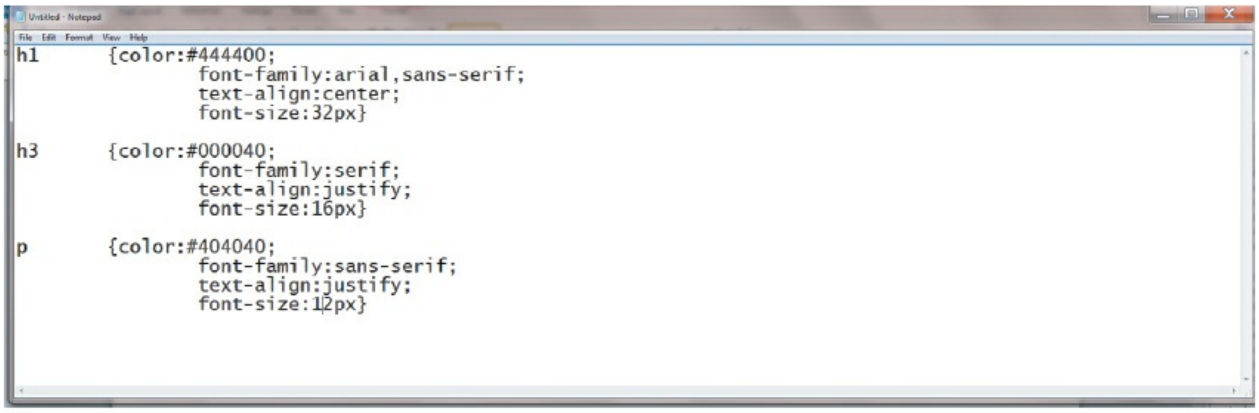
When writing HTML code, it is very important to realise that there is a difference between the structure and the presentation.

STRUCTURE is the essential part of the HTML document; it includes the semantics (meaning) and structural mark-up of the document.

PRESENTATION is the style of the document; i.e. how the document will look (or even sound if it includes multimedia elements).

These two features must be kept separate throughout the designing of a web page. At the end of the design process, the author should have an HTML document (which contains the structure and the actual content) and a separate **CSS** (**CASCADING STYLE SHEET**) file. The css file will contain everything to control the actual presentation of the web page.

Some of the `<tags>` used to create a css file have been shown already in the HTML example shown above. The following section shows an example of how these `<tags>` can be used to create a stylesheet called `example2.css`. This is then used in a web page document. The tags (h1, h3 and p) all define how the document will look when this css file (stylesheet) is attached.

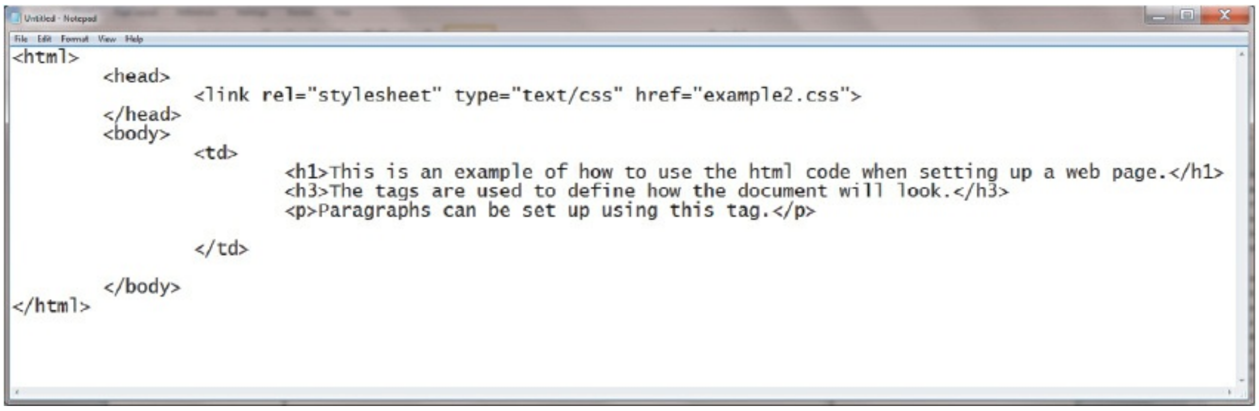


```
h1      {color:#444400;
                font-family:arial,sans-serif;
                text-align:center;
                font-size:32px}

h3      {color:#000040;
                font-family:serif;
                text-align:justify;
                font-size:16px}

p       {color:#404040;
                font-family:sans-serif;
                text-align:justify;
                font-size:12px}
```

Figure 2.16 This shows how the css file (example2.css) is created for use in the document below; h1, h3 and p have all been defined, so when the file is attached below, a web browser knows how to display this web page



```
<html>
        <head>
                <link rel="stylesheet" type="text/css" href="example2.css">
        </head>
        <body>
                <td>
                        <h1>This is an example of how to use the html code when setting up a web page.</h1>
                        <h3>The tags are used to define how the document will look.</h3>
                        <p>Paragraphs can be set up using this tag.</p>

                </td>

        </body>
</html>
```

Figure 2.17

2.4.4 Hypertext transfer protocol (http)

HYPERTEXT TRANSFER PROTOCOL (HTTP) is a set of rules that must be obeyed when transferring files across the internet. When some form of security (e.g. SSL certification or encryption – see Chapter 8) is used, then this changes to http**s** (you also often see the padlock sign in the status bar). The letter **s** after http refers to **http over secure**. It is slower to use https than http; https is usually only adopted where sensitive or private data is being transferred across the internet.

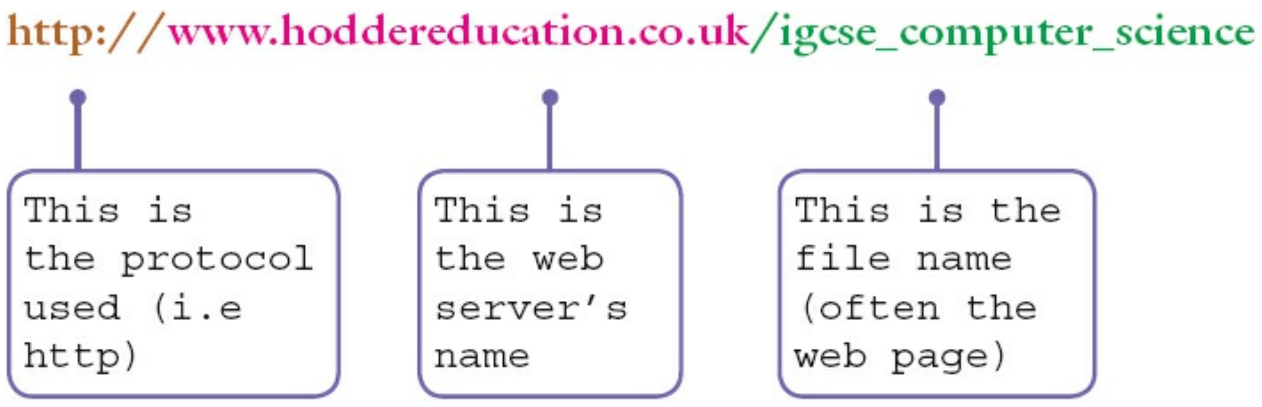
2.4.5 Web browsers

A **WEB BROWSER** is software which allows a user to display a web page on their computer screen. Web browsers interpret or translate the HTML code from websites and show the result of the translation. This can often be in the form of videos, images or sound. Most web browsers share the following features:

- they have a HOME page
- they have the ability to store a user's favourite websites/pages
- they keep a history of the websites visited by the user
- they give the ability to go backward and forward to websites opened.

Users can either click on a link, such as www.hoddereducation.co.uk/igcse or they can type in the uniform resource locator (URL) manually.

The web browser will break up the URL into three parts:


http://www.hoddereducation.co.uk/igcse_computer_science
This is the protocol used (i.e http)
This is the web server's name
This is the file name (often the web page)

The web browser translates the web server name into an IP address (see Section 2.4.2) which is part of the URL. The HTML code is returned and is shown as a correctly formatted page on the computer screen. It is also possible that cookies may be sent from the web browser to the web server when the code is executed.

(Note: please refer to Section 8.3 for more information about the use of cookies.)