ACTIVITY 1F

Convert these 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

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

▲ **Table 1.4** Memory dump

A program developer can look at each of the hexadecimal codes (as shown in Table 1.4) and determine where the error lies. The value on the far left shows the memory location, so it is possible to find out exactly where in memory the fault occurs. Using hexadecimal is more manageable than binary. It is a powerful fault-tracing tool, but requires considerable knowledge of computer architecture to be able to interpret the results.

1.1.4 Binary-coded decimal (BCD) system

The **binary-coded decimal (BCD)** system uses a 4-bit code to represent each denary digit:

0 0 0 0 = 0	0 1 0 1 = 5
0 0 0 1 = 1	0 1 1 0 = 6
0 0 1 0 = 2	0 1 1 1 = 7
0 0 1 1 = 3	1 0 0 0 = 8
0 1 0 0 = 4	1 0 0 1 = 9

Therefore, the denary number 3 1 6 5 would be 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 in BCD format.

The 4-bit code can be stored in the computer either as half a byte or two 4-bit codes stored together to form one byte. For example, using 3 1 6 5 again …

Method 1: four single bytes

0	0	0	0	0	0	1	1	3
0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	0	6
0	0	0	0	0	1	0	1	5

Method 2: two bytes

0	0	1	1	0	0	0	1	3	1
0	1	1	0	0	1	0	1	6	5

ACTIVITY 1G

1 Convert these denary numbers into BCD format.
   a) 2 7 1 b) 5 0 0 6 c) 7 9 9 0
2 Convert these BCD numbers into denary numbers.
   a) 1 0 0 1 0 0 1 1 0 1 1 1
   b) 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 0

Uses of BCD

The most obvious use of BCD is in the representation of digits on a calculator or clock display.

Each denary digit will have a BCD equivalent value which makes it easy to convert from computer output to denary display.

As you will learn in Chapter 13, it is nearly impossible to represent decimal values exactly in computer memories which use the binary number system. Normally this doesn't cause a major issue since the differences can be dealt with. However, when it comes to accounting and representing monetary values in computers, *exact* values need to be stored to prevent significant errors from accumulating. Monetary values use a fixed-point notation, for example $1.31, so one solution is to represent each denary digit as a BCD value.

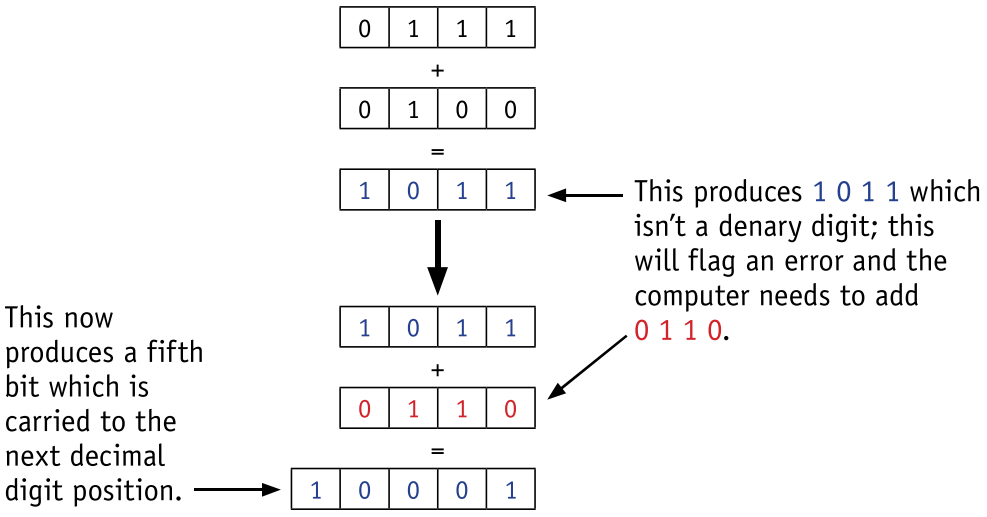
Consider adding $0.37 and $0.94 together using fixed-point decimals.

$0.37 + $0.94	0 0 0 0 0 0 0 0 . 0 0 1 1 0 1 1 1 + 0 0 0 0 0 0 0 0 . 1 0 0 1 0 1 0 0	Expected result = $1.31

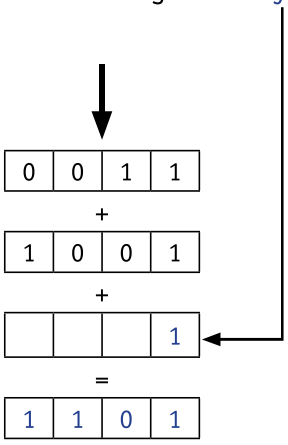
Using binary addition, this sum will produce:

0 0 0 0 0 0 0 0 . 1 1 0 0 1 0 1 1 which produces 1 1 0 0 (denary 12) and 1 0 1 1 (denary 11), which is clearly incorrect. The problem was caused by 3 + 9 = 12 and 7 + 4 = 11, as neither 12 nor 11 are single denary digits. The solution to this problem, enabling the computer to store monetary values accurately, is to add 0 1 1 0 (denary 6) whenever such a problem arises. The computer can be programmed to recognise this issue and add 0 1 1 0 at each appropriate point.

If we look at the example again, we can add .07 and .04 (the two digits in the second decimal place) first.

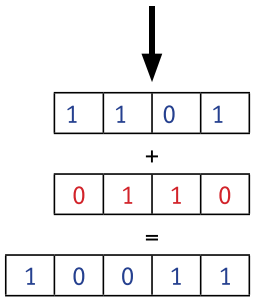


Now we will add .3 and .9 together (the two digits in the first decimal place) remembering the carry bit from the addition above:



This produces 1 1 0 1 which isn't a denary digit; this will flag an error and the computer again needs to add 0 1 1 0.

This again produces a fifth bit which is carried to the next decimal digit position.

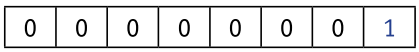


ACTIVITY 1H

Carry out these BCD additions.

a) 0.45 + 0.21
b) 0.66 + 0.51
c) 0.88 + 0.75

Adding 1 to 0 0 0 0 0 0 0 0 produces:



Final answer:

which is 1.31 in denary – the correct answer.

1.1.5 ASCII codes and Unicodes

The **ASCII code** system (American Standard Code for Information Interchange) was set up in 1963 for use in communication systems and computer systems. The newer version of the code was published in 1986. The standard ASCII code **character set** consists of 7-bit codes (0 to 127 denary or 0 to 7F in hexadecimal); this represents the letters, numbers and characters found on a standard keyboard together with 32 control codes (which use up codes 0 to 31 (denary) or 0 to 19 (hexadecimal)).

Table 1.5 shows part of the standard ASCII code table (only the control codes have been removed from the 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>

▲ **Table 1.5** Part of the ASCII code table

Notice the storage of characters with uppercase and lowercase. For example:

a	1	1	0	0	0	0	1	hex 61 (lower case)
A	1	0	0	0	0	0	1	hex 41 (upper case)
y	1	1	1	1	0	0	1	hex 79 (lower case)
Y	1	0	1	1	0	0	1	hex 59 (uppercase)

Notice the sixth bit changes from 1 to 0 when comparing lower and uppercase characters. This makes the conversion between the two an easy operation. It is also noticeable that the character sets (such as a to z, 0 to 9, and so on) are grouped together in sequence, which speeds up usability.

Extended ASCII uses 8-bit codes (128 to 255 in denary or 80 to FF in hex). This allows for non-English characters and for drawing characters to be included.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	154	9A	Ü	180	B4	┤	206	CE	╬	232	E8	Φ
129	81	ü	155	9B	¢	181	B5	╡	207	CF	╧	233	E9	Θ
130	82	é	156	9C	£	182	B6	╢	208	D0	╨	234	EA	Ω
131	83	â	157	9D	¥	183	B7	╖	209	D1	╤	235	EB	δ
132	84	ä	158	9E	Pts	184	B8	╕	210	D2	╥	236	EC	∞
133	85	à	159	9F	ƒ	185	B9	╣	211	D3	╙	237	ED	ø
134	86	å	160	A0	á	186	BA	║	212	D4	╘	238	EE	ε
135	87	ç	161	A1	í	187	BB	╗	213	D5	╒	239	EF	∩
136	88	ê	162	A2	ó	188	BC	╝	214	D6	╓	240	F0	≡
137	89	ë	163	A3	ú	189	BD	╜	215	D7	╫	241	F1	±
138	8A	è	164	A4	ñ	190	BE	╛	216	D8	╪	242	F2	≥
139	8B	ï	165	A5	Ñ	191	BF	┐	217	D9	┘	243	F3	≤
140	8C	î	166	A6	ª	192	C0	└	218	DA	┌	244	F4	⌠
141	8D	ì	167	A7	º	193	C1	┴	219	DB	┃	245	F5	⌡
142	8E	Ä	168	A8	¿	194	C2	┬	220	DC	▄	246	F6	÷
143	8F	Å	169	A9	⌐	195	C3	├	221	DD	▌	247	F7	≈
144	90	É	170	AA	¬	196	C4	─	222	DE	▐	248	F8	°
145	91	æ	171	AB	½	197	C5	┼	223	DF	▀	249	F9	∙
146	92	Æ	172	AC	¼	198	C6	╞	224	E0	α	250	FA	·
147	93	ô	173	AD	¡	199	C7	╟	225	E1	ß	251	FB	√
148	94	ö	174	AE	«	200	C8	╚	226	E2	Γ	252	FC	³
149	95	ò	175	AF	»	201	C9	╔	227	E3	π	253	FD	²
150	96	û	176	B0	░	202	CA	╩	228	E4	∑	254	FE	■
151	97	ù	177	B1	▒	203	CB	╦	229	E5	σ	255	FF	□
152	98	ÿ	178	B2	▓	204	CC	╠	230	E6	µ			
153	99	Ö	179	B3	│	205	CD	═	231	E7	τ			

▲ **Table 1.6** Extended ASCII code table

Since ASCII code has a number of disadvantages and is unsuitable for some purposes, different methods of coding have been developed over the years. One coding system is called **Unicode**. Unicode allows characters in a code form to represent all languages of the world, thus supporting many operating systems, search engines and internet browsers used globally. There is overlap with standard ASCII code, since the first 128 (English) characters are the same, but Unicode can support several thousand different characters in total. As can be seen in Tables 1.5 and 1.6, ASCII uses one byte to represent a character, whereas Unicode will support up to four bytes per character.

The Unicode consortium was set up in 1991. Version 1.0 was published with five goals, these were to

- create a universal standard that covered all languages and all writing systems
- produce a more efficient coding system than ASCII
- adopt uniform encoding where each character is encoded as 16-bit or 32-bit code
- create unambiguous encoding where each 16-bit or 32-bit value always represents the same character (it is worth pointing out here that the ASCII code tables are not standardised and versions other than the ones shown in tables 1.5 and 1.6 exist)
- reserve part of the code for private use to enable a user to assign codes for their own characters and symbols (useful for Chinese and Japanese character sets).

A sample of Unicode characters are shown in Table 1.7. As can be seen from the table, characters used in languages such as Russian, Greek, Romanian and Croatian can now be represented in a computer).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01A0	Ơ	ơ	Ƣ	ƣ	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Σ	ƪ	ƫ	T	ƭ	Ʈ	Ư
01B0	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ
01C0	ǀ	ǁ	ǂ	!	DŽ	Dž	dž	LJ	Lj	lj	NJ	Nj	nj	Ǎ	ǎ	Ǐ
01D0	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ
01E0	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ
01F0	ǰ	DZ	Dz	dz	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ
0200	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
0210	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
0220	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
0230	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ł	Ⱦ	ȿ
0240	ɀ	ɂ	ɂ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ
0250	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ
0260	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ
0270	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ
0280	ʀ	ʁ	ʂ	ʃ	ʄ	ʅ	ʆ	ʇ	ʈ	ʉ	ʊ	ʋ	ʌ	ʍ	ʎ	ʏ
0290	ʐ	ʑ	ʒ	ʓ	ʔ	ʕ	ʖ	ʗ	ʘ	ʙ	ʚ	ʛ	ʜ	ʝ	ʞ	ʟ
02A0	ʠ	ʡ	ʢ	ʣ	ʤ	ʥ	ʦ	ʧ	ʨ	ʩ	ʪ	ʫ	ʬ	ʭ	ʮ	ʯ
02B0	ʰ	ʱ	ʲ	ʳ	ʴ	ʵ	ʶ	ʷ	ʸ	ʹ	ʺ	ʻ	ʼ	ʽ	ʾ	ʿ

▲ **Table 1.7** Sample of Unicode characters

1.2 Multimedia

Key terms

Bit-map image – system that uses pixels to make up an image.

Pixel – smallest picture element that makes up an image.

Colour depth – number of bits used to represent the colours in a pixel, e.g. 8 bit colour depth can represent $2^8 = 256$ colours.

Bit depth – number of bits used to represent the smallest unit in, for example, a sound or image file – the larger the bit depth, the better the quality of the sound or colour image.

Image resolution – number of pixels that make up an image, for example, an image could contain 4096 × 3192 pixels (12 738 656 pixels in total).

Screen resolution – number of horizontal and vertical pixels that make up a screen display. If the screen resolution is smaller than the image resolution, the whole image cannot be shown on the screen, or the original image will become lower quality.

Resolution – number of pixels per column and per row on a monitor or television screen.

Pixel density – number of pixels per square centimetre.

Vector graphics – images that use 2D points to describe lines and curves and their properties that are grouped to form geometric shapes.

Sampling resolution – number of bits used to represent sound amplitude (also known as bit depth).

Sampling rate – number of sound samples taken per second.

Frame rate – number of video frames that make up a video per second.

Images can be stored in a computer in two common formats: bit-map image and vector graphic.

1.2.1 Bit-map images

Bit-map images are made up of **pixels** (picture elements); the image is stored in a two-dimensional matrix of pixels.

Pixels can take different shapes, such as  or 

EXTENSION ACTIVITY 1B

Find out how HTML is used to control the colour of each pixel on a screen. How is HTML used in the design stage of a web page screen layout?

When storing images as pixels, we have to consider

- at least 8 bits (1 byte) per pixel are needed to code a coloured image (this gives 256 possible colours by varying the intensity of the blue, green and red elements)
- true colour requires 3 bytes per pixel (24 bits), which gives more than one million colours
- the number of bits used to represent a pixel is called the **colour depth**.

In terms of images, we need to distinguish between **bit depth** and colour depth; for example, the number of bits that are used to represent a single pixel (bit depth) will determine the colour depth of that pixel. As the bit depth increases, the number of possible colours which can be represented also increases. For example, a bit depth of 8 bits per pixel allows 256 (2^8) different colours (the colour depth) to be represented, whereas using a bit depth of 32 bits per pixel results in 4 294 967 296 (2^{32}) different colours. The impact of bit depth and colour depth is considered later.

We will now consider the actual image itself and how it can be displayed on a screen. There are two important definitions here:

- **Image resolution** refers to the number of pixels that make up an image; for example, an image could contain 4096 × 3192 pixels (12 738 656 pixels in total).
- **Screen resolution** refers to the number of horizontal pixels and the number of vertical pixels that make up a screen display (for example, if the screen resolution is smaller than the image resolution then the whole image cannot be shown on the screen or the original image will now be a lower quality).

We will try to clarify the difference by using an example.

Figure 1.1 has been taken by a digital camera using an image resolution of 4096 × 3192 pixels:

▲ **Figure 1.1** Image taken by a digital camera

Suppose we wish to display Figure 1.1 on a screen with screen resolution of 1920 × 1080. To display this image the web browser (or other software) would need to re-size Figure 1.1 so that it now fits the screen. This could be done by removing pixels so that it could now be displayed, or part of the image could be cropped (and, in this case, rotated through 90°) as shown in Figure 1.2.

▲ **Figure 1.2** Image cropped and rotated through 90°