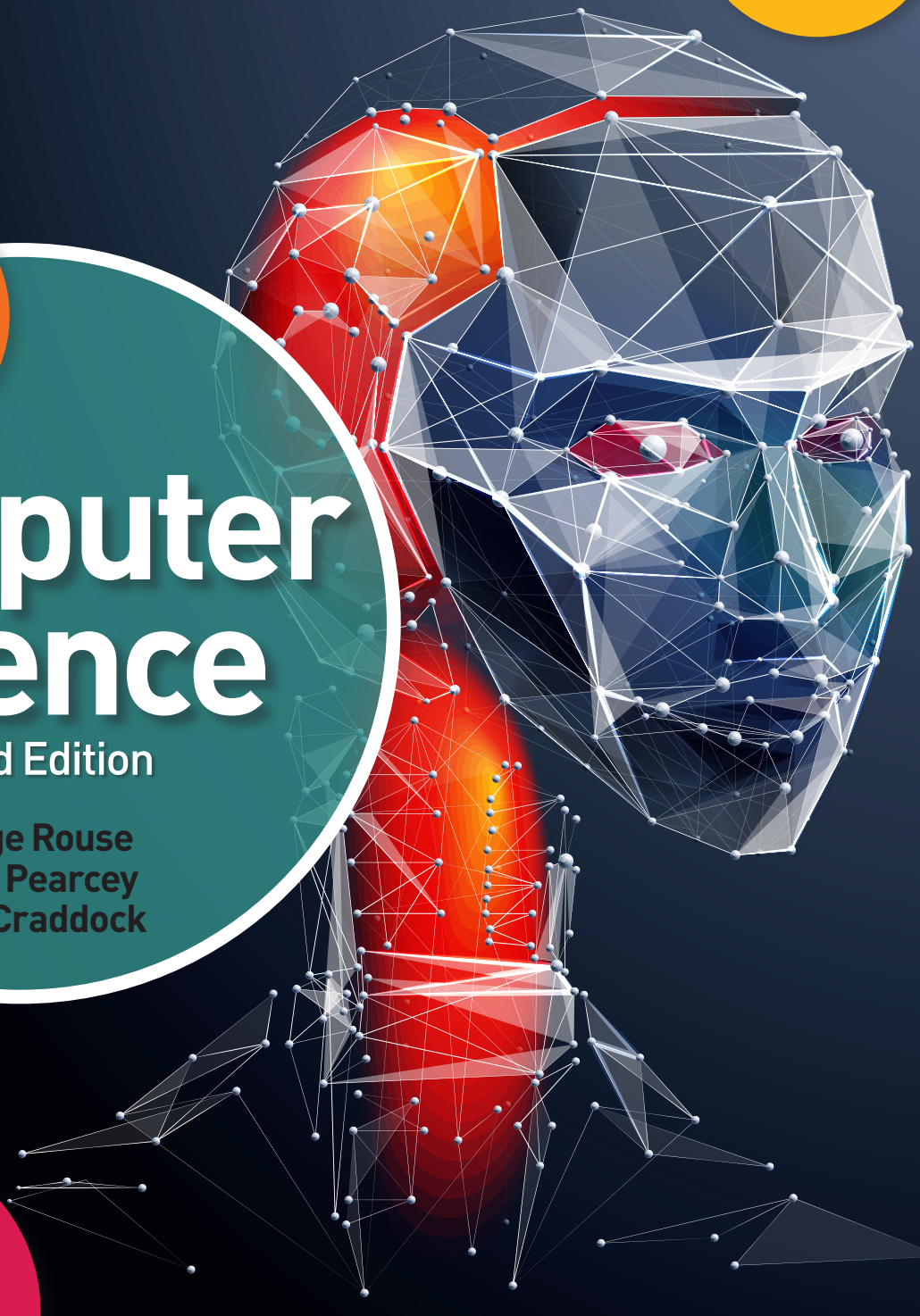
FOR THE
8525
SPECIFICATION

AQA
GCSE
(9–1)

Computer Science

Second Edition

George Rouse
Lorne Pearcey
Gavin Craddock

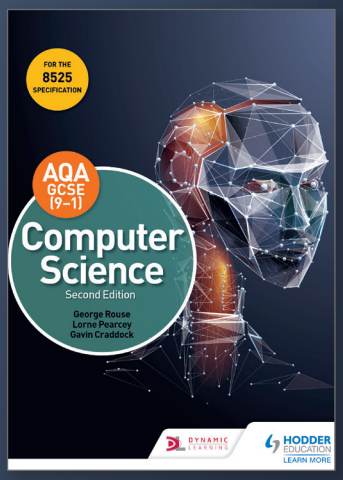


SAMPLE
CHAPTER

UPDATED
SPECIFICATION
FIRST
TEACHING
2020

DYNAMIC
LEARNING

HODDER
EDUCATION
LEARN MORE

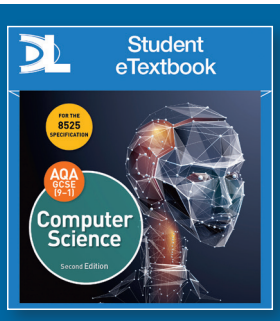


AQA GCSE (9–1) Computer Science, Second Edition

Authors: George Rouse, Lorne Pearcey, Gavin Craddock
ISBN: 9781510484306
Price: £23.99
Published: 28/08/2020

This sample chapter has been taken from this upcoming Student Book.

Also available for the updated AQA GCSE (9–1) Computer Science specification:



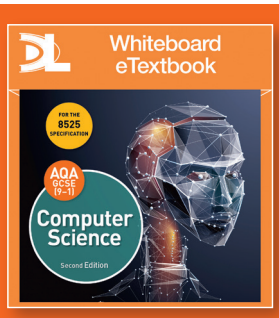
AQA GCSE (9–1) Computer Science Student eTextbook

Prices from: £6.00 + VAT per student

Available from: 25/09/2020

Allocate the textbook to your students so they can access it on the move via their phones or tablets. This pocket-friendly resource ensures students can:

- Add, edit and synchronise notes across two devices
- Access their personal copy on the move



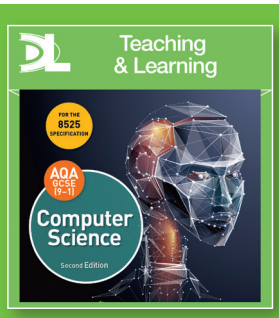
AQA GCSE (9–1) Computer Science Whiteboard eTextbook

Prices from: £50.00 + VAT

Available from: 28/08/2020

An online, interactive version of the printed textbook that enables you to:

- Display interactive pages to your class
- Add notes and highlight areas of text
- Insert double-page spreads into your lesson plans



AQA GCSE (9–1) Computer Science Teaching & Learning Resources

Prices from: £75.00 + VAT

Available from: 28/08/2020

Save time creating engaging lessons for the updated OCR GCSE (9–1) Computer Science specification by subscribing to our Teaching & Learning Resources that are packed full of ready-made:

- Topic-focused tasks
- Exam practice activities
- Lesson presentations

Please note that only the Student Book and Student eTextbook will be submitted to AQA's Approval process. The other resources will not be entered into the process.

CONTENTS

The Publishers would like to thank the following for permission to reproduce copyright material.

Photo credits

p. 7 *both* © George Rouse.

Acknowledgements

Every effort has been made to trace all copyright holders, but if any have been inadvertently overlooked, the Publishers will be pleased to make the necessary arrangements at the first opportunity.

ISBN: 978 1 3983 07339

Illustrations by Integra Software Services Pvt. Ltd., Pondicherry, India

Typeset by Integra Software Services Pvt. Ltd., Pondicherry, India

Printed in the UK by Ashford Colour Press Ltd

A catalogue record for this title is available from the British Library.

HOW THIS BOOK WILL HELP YOU

This textbook uses a range of learning features to help you get the most from it. We have selected a range of content for this sample chapter to show how useful these will be for your teaching and learning.

Important words

Highlighted in **orange**, these are terms that you will be expected to know and understand in your exams.

Important words

You will need to know and understand the following for the exam:

decimal
binary
hexadecimal
bit

Tech terms

Jargon or technical definitions in **blue** that you may find useful.

Tech term

Transistor An electronic component that acts like a switch – transistors can be manufactured at such a small size that hundreds of millions can fit on a small computer chip.

Key point

An important idea or concept.

Key point

You need to show your working in the examination so show the key multiplications to demonstrate clearly how you arrived at the answer.

Worked examples

Used to illustrate an idea or a concept, these will guide you through the reasoning behind each step of a calculation or process.

Worked example

For example, if we have a file that is 2.5 MB what is that in (a) kilobytes, and (b) bytes?

(a) 2.5 MB = 2.5 × 1000 = 2500 kilobytes

(b) 2.5 MB = 2.5 × 1000 × 1000 = 2 500 000 bytes

Beyond the spec

Information that you will not be expected to know or state in an exam but will aid understanding or add some useful context.

Beyond the spec

If there are two bitmap images of the same size and one is made up of more pixels, it is said to have a higher **resolution**. Higher resolution images have a larger file size because they are made of more pixels and so more data is required to store them.

Knowledge check

Quick check-ins that help you to recap and consolidate your understanding of the previous section.

Knowledge check

11 If the ASCII value of A is 65, what is the ASCII value of

(a) F

(b) G

(c) J

Recap and review

A targeted summary of everything you have learned in the chapter. Use this to help you recap as you work through your course.

Question practice

More formal questions to help you to prepare for both examination papers.

3

FUNDAMENTALS OF DATA REPRESENTATION

CHAPTER INTRODUCTION

In this chapter you will learn about:

3.1 Number bases

- Decimal, binary and hexadecimal
- Understand that computers use binary
- Explain why hexadecimal is used

3.2 Converting between number bases

- How binary can represent whole numbers
- How hexadecimal can represent whole numbers
- Converting between binary and decimal
- Converting between binary and hexadecimal
- Converting between decimal and hexadecimal

3.3 Units of information

- Bits
- Bytes
- Prefixes

3.4 Binary arithmetic

- Adding binary numbers
- Binary shifts
- Simple multiplication and division

3.5 Character encoding

- 7-bit ASCII and Unicode character sets
- Grouping of character codes

3.6 Representing images

- Pixels
- Bitmaps
- Image size
- Image depth
- Factors affecting bitmap file size
- Calculating bitmap file size
- Conversion of binary data into a bitmap image
- Conversion of a bitmap image into binary data

3.7 Representing sound

- Digital representation of sound
- Sample rate and resolution
- Calculating sound file size

3.8 Data compression

- The need for compression
- Different ways to compress data
- Huffman coding
- Huffman trees
- Calculating bits required for Huffman coding
- Run length encoding

3.3 Units of information

A computer uses electronic circuits etched onto computer chips to store data and instructions. These circuits contain electronic switches made from tiny **transistors**. Each switch can be in one of two states: **on** or **off**. The two states are represented by the numbers **1** or **0**. A computer uses combinations of these 1s and 0s to represent data and instructions.

As we have discussed, the **binary** number system only uses the two values 1 and 0 and therefore binary is used to describe the on-off status of all the switches in a computer.

One **bi**nary digi**t** is called a **bit**. The symbol for this is **b**. Computers often group 8 bits together as one unit of data. These 8 bits together are called a **byte** with the symbol **B**.

4 bits grouped together are called a **nibble**, which has no symbol.

In standard scientific notation the prefix 'kilo' means 1000 – for instance 1 **kilo**metre is 1000 metres. We use kilo, and a whole set of other units, based on this scientific notation:

8 bits (b)	1 byte (B)
1000 B	1 **kilobyte (kB)**
1000 kB	1 **megabyte (MB)**
1000 MB	1 **gigabyte (GB)**
1000 GB	1 **terabyte (TB)**
1000 TB	1 **petabyte (PB)**

Tech term

Transistor An electronic component that acts like a switch – transistors can be manufactured at such a small size that hundreds of millions can fit on a small computer chip.

Key point

You need to show your working in the examination so show the key multiplications to demonstrate clearly how you arrived at the answer.

Worked example

For example, if we have a file that is 2.5 MB what is that in (a) kilobytes, and (b) bytes?

(a) 2.5 MB = 2.5 × 1000 = 2500 kilobytes

(b) 2.5 MB = 2.5 × 1000 × 1000 = 2 500 000 bytes

Beyond the spec

In some sources, you will find people referring to 1 kilobyte as 1024 bytes, 1 megabyte as 1024 kilobytes, etc. There are historical reasons for doing so but now there are different prefixes for this – for example, 1024 bytes is called 1 kibibyte. (1024 might seem like a strange number in decimal but is used because it is a 'neater' number in binary.)

3.4 Binary arithmetic

Binary shifts

Moving the digits in a binary number left or right is called a **binary shift.**

Multiplication

Each time the value is shifted one place to the left, the number is multiplied by 2.

For example, 40 in binary is 101000:

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

If we shift the digits one place to the left, and add 0 to the right-hand 1 column, we get:

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

The original number that has been moved one place to the left has been highlighted.

In decimal this new number has the value 64 + 16 = 80. This is 40 multiplied by 2.

If we shift another place to the left – which is two places to the left from the original number – we get:

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

In decimal, this new number is 128 + 32 = 160. This is the original number 40 multiplied by 4.

Division

Each time the value is shifted one place to the right the number is divided by 2.

Starting again with the decimal number 40, which is 101000 in binary, if we shift the digits one place to the right we get:

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

In decimal, the new number 10100 is 16 + 4 = 20, which is 40 divided by 2.

If we shift another place to the right we get:

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

In decimal this is 8 + 2 = 10, which is 20 divided by 2 (or the original number 40 divided by 4).

Knowledge check

Apply the shifts described to the following binary numbers and state the decimal equivalents before and after the shift. Comment on what has happened to the value.

1 1100 shift 2 places to the right
2 11010 shift 1 place to the left
3 101 shift 3 places to the left
4 110000 shift 3 places to the right
5 111 shift 4 places to the left
6 10000000 shift 4 places to the right
7 10011 shift 3 places to the left
8 101100 shift 2 places to the right
9 What would you need to do to a binary number to a) multiply it by 8, b) divide by 16?
10 What is the effect of shifting left by 3 then right by 2?

3.5 Character encoding

Using binary codes to represent characters

When you press the keys on a keyboard, the computer registers this as a **binary code** to represent each character. This code can then be used to identify and display a character on screen or for printing. It is important for all computer systems to agree on these codes and their meanings if the data is to make any sense. There are, therefore, agreed international standards that are used to represent the character set for a computer system.

Character sets and bits per character

The **character set** of a computer is all the characters that are available to it. The number of characters in the character set depends upon how many characters can be represented by the associated codes. The first agreed standard was based on English with a limited number of extra symbols. Wider use of computers, and the need for many more languages and other symbols, has led to the development of more advanced coding standards for character sets.

ASCII

In 1960, the American Standards Association agreed a set of codes to represent the main characters used in English. This is called **ASCII** (American Standard Code for Information Interchange). This system was designed to provide codes for the following:

All the main characters, i.e. 26 uppercase and 26 lower case	52 characters
All the numeric symbols 0–9	10 characters
32 punctuation and other symbols plus 'space'	33 characters
32 non-printable control codes	32 characters

In total, this is 127 characters. The decimal number 127 is 1111111 in binary, which is a 7-bit number. This means that each character can be represented by a different 7-bit number, from 0000001 to 1111111. Initially the ASCII character set used 127 codes for the characters, with 0000000 meaning 'no character'. This gave a total of 128.

One additional bit was used for error checking purposes. This means that each ASCII character is represented by an 8-bit number and therefore that each character required 1 byte.

Some ASCII codes are:

7-bit binary code	Hex	Decimal	Character
0100000	20	32	'space'
1000001	41	65	A
1000010	42	66	B
1000011	43	67	C
1100001	61	97	a
1111001	79	121	y
1111010	7A	122	z
1111111	7F	127	'delete'

Unicode

Unicode was first developed to use 16 bits rather than the 7 bits of ASCII. This provided the ability to store 2^{16} or 65 536 unique characters. Later developments of Unicode used even more bits to represent billions of different characters, including graphical symbols and emojis.

To ensure compatibility of all of these systems, the original ASCII character codes are the same within Unicode. ASCII is now considered to be a subset of Unicode.

The ASCII codes for the main alphabetic characters are allocated to the uppercase characters in sequence, followed by the lowercase characters in sequence. For instance, A is 65, B is one more at 66, C is next at 67, and so on. This means that if you are given the character code for one letter, you can work out the character code for another letter.

There are also ASCII codes for the decimal numbers 0–9. These codes also run in order – for instance, the code for '1' in ASCII is 49, '2' is 50 and so on. If you are given the code for one number, you can work out the code for another number.

Lowercase characters start with 'a' as 97, 'b' as 98, and so on. This means that when we sort text, 'Z' (which is 90) comes before 'a'.

For example, if the animals goat, bear, ape, zebra and deer were written as **Goat**, **Bear**, **ape**, **Zebra**, **deer** and sorted using ASCII values, they will be in the order:

Bear, **Goat**, **Zebra**, **ape**, **deer**

Character set	Number of bits	Number of characters	Examples
ASCII	8	128	Upper and lowercase, numbers, punctuation, some control characters.
Unicode	16/32 bits	65 000/ 2 billion +	As above plus all known language characters and different characters including wingdings and emojis.

Knowledge check

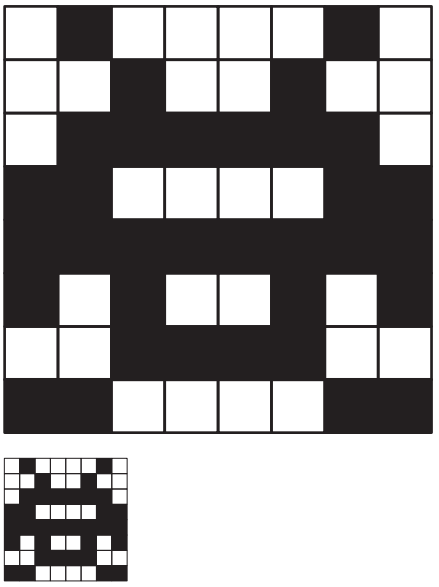
11 If the ASCII value of A is 65 what is the ASCII value of
- **(a)** F
- **(b)** G
- **(c)** J

3.6 Representing images

How an image is represented as a series of pixels and in binary

A simple image can be made up of black or white blocks. Binary numbers can represent these black and white blocks, using 1 for black and 0 for white. The image in Figure 3.6.1 uses 8 bits (1 byte) to represent each row. These blocks are the smallest element of an image and are called **pixels**. Pixel is short for **picture element**.

This image is just 8 pixels wide by 8 pixels high. As each row is represented by 1 byte, and there are eight rows, this image requires 8 bytes to store it.



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

Figure 3.6.1 A simple black and white image

Most images are not 'blocky' like this one. This is because they are made up of many more pixels. For instance, the simple black and white drawing in Figure 3.6.2 is 100 × 152 pixels. This requires 15 200 bits to store it, which is 15 200/8 = 1900 bytes, or just under 2 kilobytes.

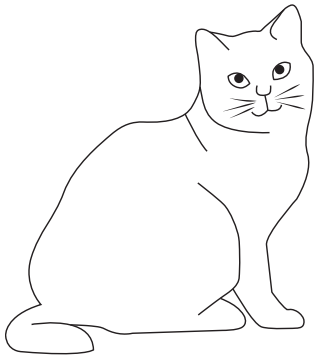


Figure 3.6.2 A higher resolution image

Beyond the spec

If there are two bitmap images of the same size and one is made up of more pixels, it is said to have a higher **resolution**. Higher resolution images have a larger file size because they are made of more pixels and so more data is required to store them.

Size of a bitmap image

The **size** of an image is the width of the image in pixels multiplied by the height of the image in pixels. So, a **bitmap** that is 5 pixels wide and 7 pixels high would be 5 × 7 pixels.

A bitmap image is always measured as width × height in pixels.

Tech term

Resolution How many bits there are in an image of a given width and height in mm – a higher resolution image has more pixels in the same width and height than a lower resolution image.

Colour depth

For colour images we need to store more than just a 1 or 0 for each pixel – we need to be able to store extra data to represent a range of colours.

For instance, if we want each pixel to be one of four different colours, we would need to represent each pixel as one of four different values. We can use a 2-bit binary number to do this, with the values 00, 01, 10 and 11. The colour of each pixel will be represented by one of these four binary codes. For instance, if 11 is black, 10 is green, 01 is red and 00 is white:

Figure 3.6.3 Binary representation of four colours

Then we can create a colour image as follows:

00	00	00	01	01	00	00	00
00	00	10	01	01	10	00	00
10	10	11	10	10	11	10	10
10	00	01	01	01	01	00	10
10	00	01	01	01	01	00	10
10	00	01	00	00	01	00	10
00	00	01	00	00	01	00	00
00	00	01	00	00	01	00	00

Figure 3.6.4 Image of a four-colour space invader with binary codes

The bitmap to represent this image can be seen in the figure and each row can be written down as:

0000000101000000 (top row)

0000100101100000 (second row)

1010111010111010 (third row)

and so on.

With 2 bits for each pixel, we can store $2^2 = 4$ colours.

If we use more bits to represent each pixel, we can represent more colours:

- With 3 bits per pixel, each pixel can be one of $2^3 = 8$ colours. (In binary, these eight codes are: 000, 001, 010, 011, 100, 101, 110, 111.)
- With 8 bits per pixel, each pixel can be one of $2^8 = 256$ colours.
- With 16 bits per pixel, each pixel can be one of $2^{16} = 65\,536$ colours!

Colour depth is the number of bits used per pixel. The more bits per pixel, the larger the range of colours we can have in the image. The more colours we have available, the better the representation of the image. However, the more colours we have, the more bits per pixel we need. This means that more data is required to store each pixel.

Consequently, the higher the colour depth, the larger the file needed to store the image.

Look at the image in Figure 3.6.5 of a sign in Portmeirion. The original image has a bit depth of 8, which is 2^8 or 256 colours, and is 1.2 MB in size. As we reduce the number of colours available the image become less well defined but the size of the file reduces, to 787 kB for eight colours and 542 kB for four colours.

Figure 3.6.5 The same image with eight colours and with just four colours

Calculating bitmap image size

To calculate the size of an image file, we need to know the colour depth and the width and height of the image in pixels.

File size = colour depth × image height (px) × image width (px)

Worked example

For example, what is the file size of an image with an 8-bit colour depth that is 1200 pixels high and 2000 pixels wide?

File size =8 × 1200 × 2000 = 19 200 000 bits

We divide by 8 to get this in bytes: 19 200 000/8 = 2 400 000 bytes or 2.4 MB

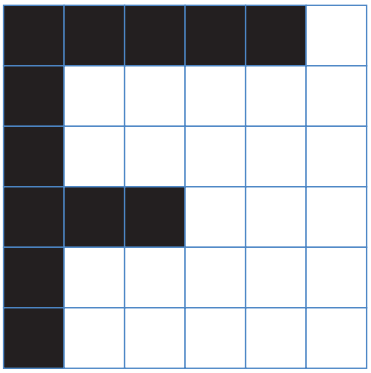
Knowledge check

12 What do we mean by colour depth?

13 How many colours can be represented using a 4-bit colour depth?

14 If 1 represents black and 0 represents white, draw the 5 × 5 pixel image with the bit pattern 00100 01010 10001 11111 10001

15 If 1 represents black and 0 represents white, what is the bit pattern for this image?



16 What is the file size for an image 200 pixels wide, 300 pixels high and with a colour depth of 4 bits per pixel?

17 A 10 pixel by 10 pixel image has 16 colours. Calculate the size of the image.

Important words

You will need to know and understand the following for the exam:

decimal
binary
hexadecimal
bit
byte
nibble
kilobyte (kB)
megabyte (MB)
gigabyte (GB)
terabyte (TB)
petabyte (PB)
binary shift
character set
ASCII
Unicode
pixel
bitmap
image size
colour depth

3.1 Number bases

Programmers use three different number systems:

- **decimal** (base 10)
- **binary** (base 2)
- **hexadecimal** (base 16).

Computers work in binary because:

- Computers use millions of tiny switches to store and process data.
- These switches have just two states, either 1 (on) or 0 (off), which can be represented in binary.
- Therefore, all data – numbers, characters, sounds and images – are represented in a computer as binary.

Hexadecimal (or hex) is used in computer science because it is easy to convert binary to and from hex, and hex is easier for people to work with than binary.

3.2 Converting between number bases

Converting binary to decimal

Our everyday counting system is called decimal (or denary).

- Decimal is a **base-10** number system
- This means it uses ten symbols or values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

We write decimal numbers such as 348 as follows:

100	10	1
3	4	8

In decimal each column heading in the table is ten times as big as the column to the right.

Binary is a **base-2** number system.

- This means it uses two symbols or values: 0 and 1.
- Each **b**inary dig**it** (0 or 1) is called a **bit**.

We write binary numbers as follows: 10100010

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

In binary each column heading is twice as big as the previous one.

To convert a binary number into decimal one, add together the column heading values for every column with a 1 in the binary number.

10100010 is 128 + 32 + 2 = 162 in decimal

Converting decimal to binary

Create a table with eight columns representing an 8-bit binary number.

128	64	32	16	8	4	2	1

- Starting from the left-hand 128 column, find the first column that is smaller than the decimal number you are converting and write a 1 in that column.
- Subtract that column heading value from the decimal number to get a remainder.
- Repeat the process using the remainder, i.e. finding the next column value that is smaller than the remainder.
- Eventually there will be a remainder of either 1 or 0 to be entered into the right-hand 0 column.

For example, the decimal number 84 is:

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

Converting hexadecimal to decimal

- Hexadecimal (hex) is a **base-16** number system.
- This means it uses 16 symbols or values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. It uses the letters A–F for the decimal values 10–15.

In hex, each column heading is 16 times as big as the previous one.

16	1

To convert hex to decimal:

- Convert each hex digit to its decimal equivalent.
- Multiply the column headings by the equivalent decimal value.
- Add the two values together.

For example, to convert CA to decimal:

16	1
C	A

C is 12 in decimal and A is 10 in decimal.

16 × 12 = 192

1 × 10 = 10

192 + 10 = **202**

CA is 202 in decimal.

Converting decimal to hexadecimal

- Divide the decimal number by 16 and write down the correct hexadecimal symbol for the result in the 16s column.
- Convert the remainder to the correct hexadecimal symbol and right that down in the 1s column.

For example, 175 in hexadecimal is:

175/16 = 10 remainder 15. The hexadecimal number is therefore:

16	1
A	F

Converting binary to hex

- Write the binary number as an 8-bit number.
- Split the 8-bit binary number into 4-bit numbers (called nibbles).
- Convert each nibble to the corresponding hex symbol.

For example, convert 11110 to hex:

- 11110 is a 5-bit number – this is 00011110 as an 8-bit number
- first nibble is 0001
- second nibble is 1110
- 0001 in decimal is 1, which is 1 in hex
- 1110 in decimal is 14, which is E in hex.

Hence **11110** is **1E** in hex.

Converting hexadecimal to binary

To convert from hex to binary, we simply replace each hex symbol with the equivalent binary nibble.

For example, convert 3F to binary:

- 3 is 3 in decimal, which is 0011 in binary – this is the first nibble.
- F is 15 in decimal, which is 1111 in binary – this is the second nibble.

So, **3F** is **00111111** in binary. This could also be written as just 111111.

In the exam you will only need to convert between number systems in the following ranges:

- 0 to 255 in decimal
- 00000000 to 11111111 in binary
- 00 to FF in hexadecimal.

3.3 Units of information

- Each stored binary digit is called a **bit** (**bi**nary digi**t**).
- A group of 8 bits is called a **byte**.
- Half a byte, 4 bits, is called a **nibble**.

4 bits (b)	1 nibble
8 bits (b)	1 byte (B)
1000B	1 **kilobyte (kB)**
1000kB	1 **megabyte (MB)**
1000MB	1 **gigabyte (GB)**
1000GB	1 **terabyte (TB)**

It is important to use the correct symbol, lowercase **b** for **bit** and uppercase **B** for **byte**.

3.4 Binary arithmetic

Adding binary numbers

When adding binary digits together there are several possibilities including:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 10 in binary, or 2 in decimal.
  In this case we write down 0 and carry 1 to the next column.
- 1 + 1 + 1 = 11 in binary or 3 in decimal.
- In this case we write down 1 and carry 1.

Binary shifts

Moving the binary digits to the left or right is called a **binary shift**.

Moving to the left **multiplies the value by 2** for each place the value is shifted.

Moving to the **right divides the number by 2** for each place the value is shifted.

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

The binary number 11010 is 26 in decimal.

If we shift the binary number one place to the left, we get:

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

Which is 52 (= 26 × 2) in decimal.

If we shift the original binary number one place to the right, we get:

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

This number is 13 (= 26/2) in decimal.

3.5 Character encoding

- Each character (e.g. A, b, !, etc.) is represented in a computer by a binary code.
- The **character set** of a computer is a list of all the characters available to the computer.

It is important that computer systems all agree on these codes and there are some agreed standards.

ASCII

- **ASCII** is an 8-bit binary code able to represent the Roman alphabet, numbers, some symbols and some control characters.
- 7 bits are used for characters with 1 bit used as an error check.
- There are 2^7 or 128 characters available.
- Uppercase and lowercase letters have different codes.
- For example, the letter D is represented by the binary code 1000100 in ASCII, which is the decimal number 68.

Unicode

- **Unicode** originally used a 16-bit binary code to represent many additional non-Roman characters and a wide range of symbols.
- The 16-bit code has 2^{16} or 65 536 characters available.
- Unicode has since been extended to use even more bits to represent billions of characters.
- The original ASCII codes are the same in Unicode so ASCII can be considered a subset of Unicode.

The binary codes for the main alphabetic characters are allocated to the uppercase characters in sequence, followed by the lowercase characters in sequence.

For instance, A is 65, B is one more at 66, C is next at 67, and so on. This means that if you are given the character code for one letter, you can work out the character code for another letter.

Lowercase characters start with a as 97, b as 98, and so on. This means that when we sort text, Z (which is 90) comes before a.

3.6 Representing images

- Images are represented on screen as a series of **pixels**.
- A pixel is the smallest element of an image – these are the dots that make up the image on screen or in a printout.
- Pixels are stored in a computer as binary codes.
- The number of bits used for each pixel determines how many colours each pixel can represent.

Image size

The image size is expressed as:

width in pixels x height in pixels

Colour depth

With 1 bit for each pixel we have just two possibilities: 0 or 1. This means a pixel can only be one of two colours.

The following row of pixels is described by the binary code 11100000, where 1 means black and 0 means white:

1	1	1	0	0	0	0	0

Two-colour encoding with one bit

To store more than two colours we need more bits per pixel:

- 2 bits to store 4 (2^2) colours per pixel
- 3 bits to store 8 (2^3) colours per pixel
- 8 bits to store 256 (2^8) colours per pixel

For instance, if we wanted each pixel to be either white, black, green or red then each pixel would be represented by 2 bits, where white is 00, red is 01, green is 10 and black is 11:

00	00	11	01	10	11	00	00

Four-colour encoding with two bits

Colour depth is the name for the number of bits used per pixel.

Calculating bitmap image size

The greater the number of pixels, the more data needs to be stored and the larger the file size.

- For instance, a 2-bit bitmap image measuring 10 × 20 pixels would be a larger file than a 2-bit bitmap image measuring 5 × 10 pixels.

The higher the colour depth the more data needs to be stored and the larger the file size.

- For instance, a 5 × 10 bitmap image with 4-bit colour depth would be a larger file than a 5 × 10 bitmap image with a 2-bit colour depth.

A higher colour depth means a larger number of colours can be represented, giving a better quality image.

An image file size can be calculated using the following formula:

- Size in bits = Width in pixels (W) × Height in pixels (H) × Colour depth in bits (D)
- Size in bytes = (Width in pixels (W) × Height in pixels (H) × Colour depth in bits (D))/8

3.7 Representing sound

Sounds are a series of vibrations that vary continuously and can take any value – this means they are **analogue**.

In order to store it on a computer, sound is sampled at regular intervals by a device that converts analogue to digital signals, and the digital values are stored as binary numbers.

- A **sample** is a measure of the **amplitude** (or amount) of sound vibration at a particular moment in time.
- The **sample rate** is the number of samples taken per second, measured in Hz (hertz). 1 Hz means one sample per second.
- The **sample resolution** is the number of bits used to represent the amplitude of vibration for each sample.
- The **duration** is the length of time that the sound is sampled for, measured in seconds.
- The higher the sample rate, the more frequently the sample is taken, and the better the approximation to the original sound – but the larger the file needed to store the data.
- The greater the sample resolution, the more accurate the measurement of vibration amplitude and the better the quality of the sound – but the larger the file needed to store the data.
- The longer the duration, the larger the file needed to store the data.

The file size (in bits) is determined by: sample rate × resolution × duration in seconds. This can also be written as:

File size (bits) = rate × res × secs

Where rate is sample rate, res is sample resolution and secs is duration in seconds.

For multiple channels, e.g. stereo, we need to multiply this by the number of channels.

File size (bits) = rate × res × secs × channels

3.8 Data compression

When transmitting files, storing very large files or storing a large number of files, we sometimes need to compress the data to make file sizes smaller.

Lossy compression:

- Some of the data is removed to make the file smaller.
- Algorithms remove data that is least likely to be noticed.
- The original file cannot be restored from the compressed version.

Lossless compression:

- None of the information is removed.
- Algorithms look for patterns in the data so that repeated data items only need to be stored only once, together with information about how to restore them.
- The original file can be restored.

Huffman coding

Huffman coding uses a binary **tree** to represent data, allocating a binary code to each data element (such as a character).

The most frequently occurring data elements are encoded with the shortest binary codes – this helps to reduce the overall encoded file size.

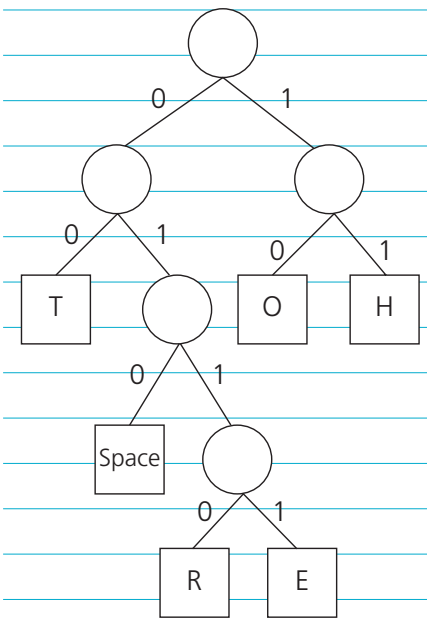
To create a Huffman tree to encode a short text phrase:

- Count how many times (frequency) each character occurs (including spaces).
- Assign each character to a node that is labelled with the character and the frequency of that character.
- Arrange these nodes in order from least frequent to most frequent.
- The two least frequent nodes are joined together to make a new node with their combined frequency.
- The new node is placed back in the list in the correct order according to this combined frequency.
- Add two branches from the new node to the two nodes that it replaced.

The process is repeated until there is just one node left in the list. At that point you:

- Label the right-hand paths as 1 and the left-hand paths as 0.
- The binary code for each character is created by reading the path to the character from the root of the tree.

For example, in this Huffman tree, O would be encoded as 10, R would be 0110.



Huffman trees are often drawn with the root at the top and branches below

Calculating the number of bits required in Huffman coding

Once data has been encoded using Huffman coding, you can calculate the number of bits required to store the coding.

- Use the Huffman tree to work out how many bits are needed for each character.

- For each character, multiply the number of bits by the frequency of the character to get the total number of bits that character needs in the whole phrase.
- Add all of these totals for each character together to work out the number of bits for the entire phrase.

For example:

Character	E	R	Sp	H	O	T
Frequency	1	1	2	3	3	4
Huffman code	0111	0110	010	11	10	00
Number of bits	4	4	3	2	2	2
Total (number of bits x frequency)	**4**	**4**	**6**	**6**	**6**	**8**

The total number of bits for the phrase HOT HOT HOTTER is: 4 + 4 + 6 + 6 + 6 + 8 = 34

If you calculate the number of bits needed for the same phrase in ASCII, you can compare this to the number of bits needed for Huffman coding. To calculate the number of bits needed for the phrase in ASCII:

- Count how many characters there are in the phrase, including spaces.
- Multiply this number by 7 (because each character in ASCII is represented by a 7-bit code).

Therefore the number of bits required for the phrase HOT HOT HOTTER in ASCII is: 14 x 7 = 98

Huffman coding this phrase instead of ASCII coding has therefore saved: 98 – 34 = 64 bits.

Run-length encoding (RLE)

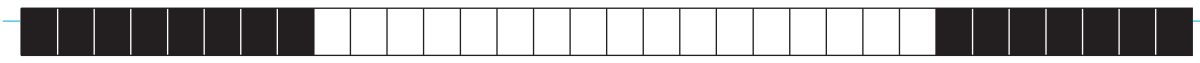
RLE compresses data by specifying how many times a character or pixel repeats, followed by the value of the character or pixel.

The text **AAAABBBBBCCCCC** is made of 14 characters. To store this in ASCII would take 14 x 7 = 98 bits.

We can, however, code the same text in RLE as: **4 65 5 66 5 67**.

To store the RLE would take 8 x 6= 48 bits.

For the black and white image below, we can use the same idea to count the number of each colour that occurs in a row:



Black and white image

In this 32 pixel image we have eight black pixels, 17 white pixels and seven black pixels. We can store this in 3 bytes with the most significant bit being 1 or 0 to represent the colour, the remaining seven bits the number of pixels of that colour.

10001000 00010001 10000111

This uses just three bytes instead of four.

QUESTION PRACTICE

3 Fundamentals of data representation

01 01.1 Convert the decimal value 77 to binary held in a single byte. [1 mark]

01.2 Convert the decimal value 59 to a hexadecimal value. [1 mark]

01.3 How many bits are needed to store a single hexadecimal digit? [1 mark]

01.4 Add together the following three binary numbers, showing your working. [3 marks]

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

02 For the binary number 11001:

02.1 Convert the binary to decimal. [1 mark]

02.2 Show the result of applying a binary shift of two places to the left. [1 mark]

02.3 Convert this result to decimal. [1 mark]

02.4 What is the effect of applying a binary shift of two places to the left? [1 mark]

02.5 What would be the result of applying a binary shift of 1 place to the right? [2 marks]

03 03.1 What is meant by the character set of a computer? [1 mark]

03.2 Explain how ASCII represents the character set of a computer. [2 marks]

03.3 Write down the sort order of the following list of words in a program using ASCII or Unicode to represent the character set: 'Apple, grape, cherry, Damson'. [1 mark]

03.4 Describe **two** differences between using an ASCII character set and a Unicode character set. [1 mark]

03.5 In Unicode, the character G is represented by the numeric code 71. Which character is represented by the numeric code 68? [1 mark]

A: d B: J C: D D: F E: 4

04 04.1 How does the number of bits in an image affect the size of the file? [1 mark]

04.2 Describe two differences between an image with a 1-bit colour depth and an image with a 2-bit colour depth. [2 marks]

04.3 How many colours can be represented using a 4-bit colour depth? Show your working. [2 marks]

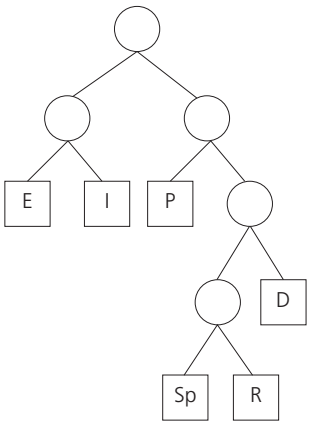
04.4 In the image, using 0 to represent white and 1 to represent black, write down the Run Length Encoded data to store this image in 3 bytes in binary. [3 marks]

05 05.1 Explain how a continuously changing sound can be captured and stored on a computer. [3 marks]

05.2 Explain what is meant by lossy compression. [2 marks]

05.3 Explain what is meant by lossless compression and when it should be used. [3 marks]

06 This is a Huffman tree for the text string "PIED PIPER".



The bit pattern for R is 1101 because it is right, right, left, right to get to R from the root of the tree.

06.1 Complete the bit patterns for the rest of the characters in the string. [5 marks]

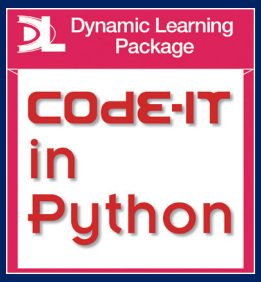
Character	Number of occurrences	Bit pattern
P	3	
I	2	
E	2	
D	1	
R	1	1101
Space	1	

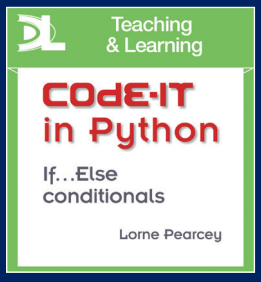
06.2 Calculate how many bits are needed to store the text string using Huffman codes. [3 marks]

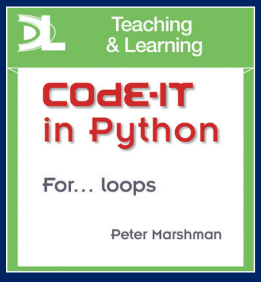
06.3 Calculate how many bits would be required to store the text string using ASCII codes. [2 marks]

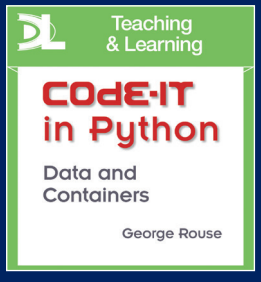
Code-IT in Python

Discover a new way to help students learn how to code with Code-IT in Python.









Please note, none of the Code-IT in Python modules are approved by AQA.

Help your GCSE students progress beyond simple programming skills and removing any fear they might have transitioning from a block-based language to a text-based language with our coding resource, Code-IT for Python.

Code-IT in Python consists of 14 stand-alone modules, each focused on a different programming content required at GCSE – allowing you to pick and choose which modules your students need. Using our responsive, online environment, students are encouraged to write and test their own code in order to solve Coding Challenges.

It's not your average 'how-to code' product

A digital resource that provides your students with a learning journey through essential programming skills required at GCSE.

It will fill students' coding skills gap

Designed to focus on a range of programming skills, Code-IT in Python will equip your students with the necessary tools needed to complete any GCSE programming task effectively and efficiently.

It will save you time!

Auto-marked Coding Challenges require students to write, test and de-bug their code. Feedback is given immediately, so students can understand the areas they need to amend and learn from.

It's packed full of resources

As well as detailed progress reports on students' activity, you will find guidance sheets, lesson ideas and starter presentations to help reinforce learning and cut down on the time you spend creating new resources.

Pick and choose the modules you want

£30 + VAT per module for one-year access. Save up to 20% by exploring our bundle offers.

Visit our website or contact your local Sales Representative to find out more about Code-IT in Python and to register for a free trial.

www.hoddereducation.co.uk/code-it **computing@hoddereducation.co.uk**

SAMPLE CHAPTER

UPDATED SPECIFICATION FIRST TEACHING 2020

AQA GCSE (9–1) Computer Science

Second Edition

FOR THE 8525 SPECIFICATION

Written by leading Computer Science teachers, this brand-new textbook will guide students through the updated AQA GCSE Computer Science specification topic by topic, and provide them with standalone recap and review sections, practice questions, worked examples and clear explanations of complex topics.

This textbook:

- prepares students for assessment with numerous practice questions for all topics
- develops computational thinking skills
- provides differentiated material with the 'beyond the spec' feature
- includes standalone recap and review sections at the end of each chapter
- provides definitions of technical terms, along with a glossary of words to ensure students feel confident with the assessment.

Authors
George Rouse, Lorne Pearcey and **Gavin Craddock** are highly respected and widely published authors of resources.

HODDER EDUCATION
t: 01235 827827
e: education@bookpoint.co.uk
w: hoddereducation.co.uk

ISBN 978-1-3983-0733-9

FSC