

We are working with Cambridge Assessment International Education towards endorsement of these titles.



CAMBRIDGE
UNIVERSITY PRESS

Executive
Preview

Computer Science

for Cambridge IGCSE™ & O Level

MULTI-COMPONENT SAMPLE

Sarah Lawrey, Victoria Ellis, Dave Duddell, Chris Roffey & Richard Morgan



Second edition

 Cambridge Assessment
International Education

Endorsed for full syllabus coverage



Brighter Thinking

Better Learning

Dear Cambridge Teacher,

The new Cambridge IGCSE™ and O Level Computer Science series will publish in Spring 2021, updated for the new Cambridge International syllabuses (0478/0984/2210) for first teach in September 2021.

This Executive Preview contains sample content from the series, including:

- A guide explaining how to use the series
- The table of contents from the Coursebook, Teacher's Resource and Programming Book for Java
- The first chapter from the Coursebook, Teacher's Resource and Programming Book for Java

This new series has been designed around extensive research interviews and lesson observations with teachers and students around the world following the course. To bring this new series to life, we have included exciting case studies and questions from Microsoft® Research to help students contextualise their learning. We have also introduced a 'computer science in context' feature, linking learning to life outside the classroom walls to help you to further engage your students.

We have increased our focus on developing students' programming skills, from tasks in the coursebook, updated programming books for each of the three syllabus recommended languages (Visual Basic, Python and now, Java) and new chapters in each of our resources supporting the new programming scenario challenge. We have also introduced a 'skills focus' feature in the coursebook that hones in on key skills such as computational thinking and mathematical skills.

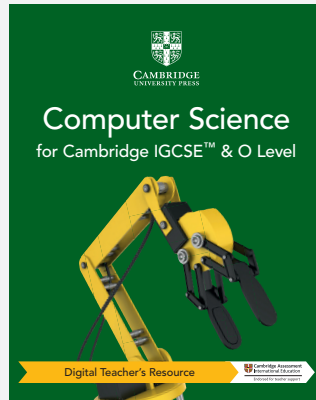
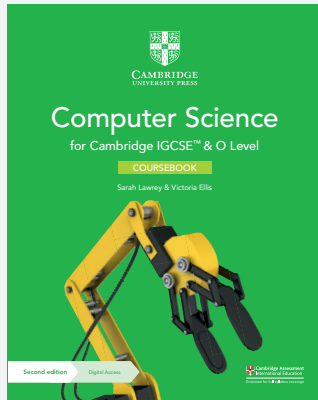
As we develop new resources, we keep up-to-date with best practice in pedagogies. For this series we have added new features to the coursebook, such as 'getting started' questions and activities to help you to evaluate students' learning starting points and 'reflection' questions after certain tasks to encourage students to take control of their own learning journey. We have developed our differentiated support, with a three-tier approach to programming tasks in the coursebook and programming books, progressing from 'getting started'/'demo', to 'practice', to 'challenge', supporting all your learners' different needs.

Finally, we have updated the teacher's resource. From teaching activity, assessment and homework ideas, to how to tackle common misconceptions in each topic, to a new feature developing your own teaching skills, we hope that this resource will inspire and support you and save you time.

We are working with Cambridge Assessment International Education towards endorsement of these titles.

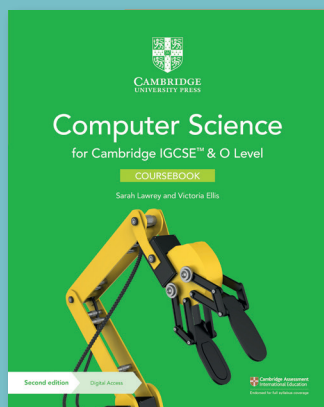
Visit our website to view the full series or speak to your local sales representative. You can find their contact details here:

cambridge.org/education/find-your-sales-consultant



Gemma Coleman
Commissioning Editor for Cambridge IGCSE™ and O Level Computer Science,
Cambridge University Press

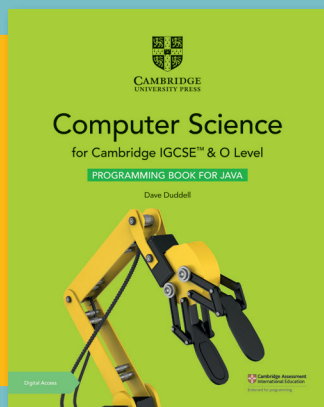
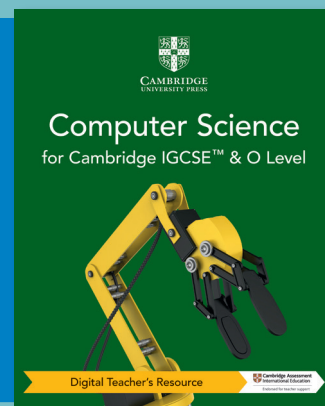
> How to use this series



The coursebook provides coverage of the full Cambridge IGCSE™ and O Level Computer Science syllabuses (0478/0984/2210) for first examination from 2023. Each chapter explains facts and concepts and uses relevant real-world contexts to bring topics to life, including two case studies from Microsoft® Research. There is a skills focus feature containing worked examples and questions to develop learners' mathematical, computational thinking and programming skills, as well as a programming tasks feature to build learners' problem-solving skills. The programming tasks include 'getting started' skills development questions and 'challenge' tasks to ensure support is provided for every learner. Questions and exam-style questions in every chapter help learners to consolidate their understanding.

The digital teacher's resource contains detailed guidance for all topics of the syllabus, including common misconceptions to elicit the areas where learners might need extra support, as well as an engaging bank of lesson ideas for each syllabus topic. Differentiation is emphasised with advice for identification of different learner needs and suggestions of appropriate interventions to support and stretch learners.

The digital teacher's resource also contains scaffolded worksheets for each chapter, as well as practice exam-style papers. Answers are freely accessible to teachers on the 'supporting resources' area of the Cambridge GO platform.



There are three programming books: one for each of the recommended languages in the syllabuses – Python, Microsoft Visual Basic and Java. Each of the books are made up of programming tasks that follow a scaffolded approach to skills development. This allows learners to gradually progress through 'demo', 'practice' and 'challenge' tasks to ensure that every learner is supported. There is also a chapter dedicated to programming scenario tasks to provide support for this new area of the syllabuses. The digital part of each book contains a comprehensive solutions chapter, giving step-by-step answers to the tasks in the book.



CAMBRIDGE
UNIVERSITY PRESS

Computer Science

for Cambridge IGCSE™ & O Level

COURSEBOOK

Sarah Lawrey & Victoria Ellis



Second edition

Digital Access

 Cambridge Assessment
International Education

Endorsed for full syllabus coverage

> Contents

The items in **orange** are accessible to teachers for free on the supporting resources area of Cambridge GO.

How to use this book	vi	4 Software	78
How to use this series	viii	4.1 Types of software	80
Introduction	1	4.2 Operating system	81
1 Data representation	8	4.3 Interrupts	84
1.1 Binary and hexadecimal	10	4.4 Types of programming language	85
1.2 Binary manipulation and negative numbers	16	4.5 Translators	87
1.3 How do computers represent text, images and sound?	21	4.6 Integrated Development Environment (IDE)	90
1.4 Measuring data storage	25	5 The internet and its uses	94
1.5 Data compression	27	5.1 The internet and the world wide web	96
2 Data transmission	33	5.2 Requesting and retrieving web pages	97
2.1 The structure of a data packet	35	5.3 The purpose of a web browser	100
2.2 Packet switching	36	5.4 Digital currency	105
2.3 Methods of data transmission	38	5.5 Cyber security	106
2.4 The universal serial bus (USB) interface	43	6 Automated and emerging technologies	125
2.5 Detecting errors in data transmission	44	6.1 Automated systems	126
2.6 Encryption	49	6.2 Robotics	132
3 Hardware	54	6.3 Robots in context	132
3.1 The role of the CPU	56	6.4 Artificial intelligence	136
3.2 The fetch stage	57	7 Algorithm design and problem solving	143
3.3 The decode stage	58	7.1 Program development life cycle	145
3.4 The execute stage	58	7.2 Analysis	146
3.5 Factors that can affect the performance of a CPU	60	7.3 Design	146
3.6 Input and output devices	62	7.4 Pseudocode	154
3.7 Data storage	66	7.5 Coding	157
3.8 Virtual memory	69	7.6 Testing	157
3.9 Cloud storage	70	7.7 Common algorithms	159
3.10 Network hardware	72	7.8 Trace tables	179
		7.9 Finding the purpose of an algorithm	184
		7.10 Finding errors in a program and correcting the errors	187

8	Programming	196	10	Boolean logic	324
8.1	Programming concepts	198	10.1	The role of logic gates	326
8.2	Data types	202	10.2	The NOT logic gate	326
8.3	Input and output	205	10.3	The AND logic gate	327
8.4	Arithmetic operators	209	10.4	The OR logic gate	329
8.5	Sequence	214	10.5	The NAND logic gate	330
8.6	Selection	216	10.6	The NOR logic gate	331
8.7	Iteration	231	10.7	The XOR logic gate	333
8.8	Totalling	241	10.8	Logic expressions	337
8.9	Counting	243	10.9	Representing truth tables	339
8.10	String manipulation	245	10.10	Representing logic expressions	344
8.11	Nested statements	251	10.11	Representing problem statements	347
8.12	Subroutines	255			
8.13	Library routines	265	11	Programming scenarios practice	353
8.14	Maintainable programs	267	11.1	Programming scenario	355
8.15	Arrays	270	11.2	Method 1: Identifying the inputs, processes and outputs	355
8.16	File handling	286	11.3	Writing the code	357
9	Databases	295	11.4	Method 2: Practically carrying out the program and identifying the stages	365
9.1	Database structure	297			
9.2	SQL	302		Microsoft case studies	379
9.3	SELECT ... FROM	302		Glossary	384
9.4	SELECT ... FROM ... WHERE	305		Index	392
9.5	Order by	311		Acknowledgements	398
9.6	SUM	312		Coursebook answers	
9.7	COUNT	313			

> How to use this book

Throughout this book, you will notice lots of different features that will help your learning. These are explained below. Answers to all questions and activities are accessible to teachers in the ‘supporting resources’ area of cambridge.org/go.

LEARNING INTENTIONS

These set the scene for each chapter, help with navigation through the coursebook and indicate the important concepts in each topic.

GETTING STARTED

This contains questions and activities on subject knowledge you will need before starting this chapter.

COMPUTER SCIENCE IN CONTEXT

This feature presents real-world examples and applications of the content in a chapter, encouraging you to look further into topics. The ‘Computer science in context’ feature that opens each chapter also ends with discussion questions.

KEY WORDS

Key vocabulary is highlighted in the text when it is first introduced. Definitions are then given in the margin and can also be found in the glossary at the back of this book.

SKILLS FOCUS

This feature supports your computational thinking, mathematical and programming skills. The features include useful explanations, step-by-step examples and questions for you to try out yourselves.

PROGRAMMING TASKS

Programming tasks give you the opportunity to develop your programming and problem-solving skills.

Each task is divided into three parts:

Getting started

- These questions will help build your basic problem-solving skills.

Practice

- These questions will ask you to design a possible solution to a problem or complete a coding task.

Challenge

- These questions will stretch and challenge your problem-solving, logic and programming skills even further.

TIPS

These are short suggestions to remind you about important learning points.

COMMAND WORDS

Command words that appear in the Cambridge Assessment International Education syllabus are highlighted in the exam-style questions when they are first introduced, and are accompanied with their syllabus definitions in the margins. Not all command words from the syllabus are included in the exam-style questions, but you will find all command word definitions in the glossary at the back of the book.

ACTIVITY

Activities give you an opportunity to check your understanding throughout the text in a more active way, for example, by creating presentations, posters or role plays.

At the end of some activities, you will find opportunities to help you assess your own work, or that of your classmates, and consider how you can improve the way you learn.

Questions

Appearing throughout the text, questions give you a chance to check that you have understood the topic you have just read about.

Further Information: This feature highlights the advanced aspects in this book that go beyond the immediate scope of the syllabus.

REFLECTION

These activities ask you to think about the approach that you take to your work, and how you might improve this in the future.

SUMMARY

There is a summary of key points at the end of each chapter.

EXAM-STYLE QUESTIONS

Questions at the end of each chapter provide more demanding exam-style questions, some of which may require use of knowledge from previous chapters.

SELF-EVALUATION CHECKLIST

The summary checklists are followed by ‘I can’ statements which match the Learning intentions at the beginning of the chapter. You might find it helpful to rate how confident you are for each of these statements when you are revising. You should revisit any topics that you rated ‘Needs more work’ or ‘Getting there’.

I can...	See topic	Needs more work	Getting there	Confident to move on
add two 8-bit binary numbers.	1.2			

› Chapter 1

Data representation

IN THIS CHAPTER YOU WILL:

- learn why computers use binary to process data
- develop an understanding about how binary is used to represent different forms of data, such as text, sound and images
- learn to convert between the number systems denary, binary and hexadecimal
- learn to add two binary numbers and understand what is meant by overflow
- learn to perform a binary shift and how this affects the number
- learn to use two's complement to represent negative binary numbers
- learn how the file size of data is measured
- learn how and why data is compressed.

GETTING STARTED

Working with a partner, write the words 'text', 'sound' and 'image' on a piece of paper. Around each word, write words that you can think of that are linked to each of the three words. For example, you could write the words 'sound wave' next to the word 'sound'.

DEBUGGING USING HEXADECIMAL

Computer programs are the foundation of computer games. If there is an error with the program, this can cause unexpected behaviour in the game. For example, a character may move the wrong way when a button is pressed. The programmer needs to find out why the character is moving incorrectly. The amount of code that needs to be assessed to find this error in a computer game is enormous and the binary values involved will be vast. Therefore, the programmer will often use **hexadecimal** to debug the error. One way this can be done is to look at the address for a variable. The address is the location in memory where the data is stored and it is shown in hexadecimal. This address is then looked up in a table to find the value held in the address. The value is also given as hexadecimal. If the value in the address is, for example, 0000 or FFFF (this can mean that the variable has not been initiated) the programmer will know that there is a problem with this value and has located the variable with which the error is occurring. The programmer can then find out why the variable is not holding a valid value.

If the addresses and values were given as **binary**, rather than as hexadecimal, then it would be more difficult to identify if an error has occurred. For example, it is easier to see if an address contains a valid value, e.g. F2A5, rather than 1111001010100101.



Figure 1.1: Data represented by hexadecimal letters and numbers

KEY WORDS

hexadecimal: a base-16 number system that uses the values 0–9 and characters A–F.

binary: a base-2 number system that uses the values 0 and 1.

CONTINUED

Discussion questions

- 1 Do you think that you have ever seen or used hexadecimal before? If you have, can you provide an example of where you have seen or used it?
- 2 Why is it important for a programmer to be able to effectively debug a computer program?

1.1 Binary and hexadecimal

Humans process a large amount of data on a daily basis. Did you know that when we have a conversation with someone, we process approximately 60 bits of information per second? Our brain is designed to process a maximum of about 120 bits per second. That's why it is so difficult to listen to two people talk to us at once!

As humans, we process data that is called **analogue data**. Everything we see, hear, feel and smell is a continuous stream of analogue data to our senses.

Questions

- 1 Can you think of **three** examples of analogue data?
- 2 Why is analogue data described as continuous data?

The components that process data in a computer are made up of very small **logic gates**. These gates cannot process analogue data, they can only process **digital data**. This data is stored in **registers**. You can find out more about logic gates in Chapter 10. Digital data is discrete data that only has two values, 1 and 0. Any data that we want a computer to process must be converted into digital data. This type of data is also called binary data.

Number systems

In our daily lives we mostly use a **denary** number system. This is called a base-10 number system and uses the digits 0–9. The units in a denary number system are ones, tens, hundreds and thousands. Each time the unit increases by the power of 10.

For example, the denary number 123 has:

Increases by power of 10		Increases by power of 10	
One hundred	Two tens	Three ones	
1	2	3	

Computers use a binary number system. This is called a base-2 number system and uses the digits 0 and 1. The units in a binary number system increase by the power of 2.

Any denary values need to be converted to binary values so that they can be processed by a computer.

KEY WORDS

analogue data: a continuous stream of data that is processed by humans.

logic gate: a very small component in a computer system that controls the flow of electricity.

digital data: discrete data that only uses the values 0 and 1.

register: a small component in a computer system that is used to temporarily store data.

denary: a base-10 number system that uses the values 0–9.

SKILLS FOCUS 1.1

CONVERTING BETWEEN DENARY AND BINARY NUMBERS

The size of a binary number is referred to by the number of bits it has. A 4-bit binary number will have 4 values that are either 1 or 0, for example, 1001. An 8-bit binary number will have 8 values that are also either 1 or 0. Binary values can get very large, some have as many as 64 bits; now that's a very big number! You will need to be able to understand binary numbers that have as many as 16 bits.

First, focus on converting denary numbers to 4-bit binary. To convert denary numbers to 4-bit binary the units that you need to use are 1, 2, 4 and 8. You can indicate with a 1 or a 0 whether that unit is required. For example, if you convert the denary number 13 to a 4-bit binary number you can use the following steps:

Write down the binary units from 8 to 1. An easy way to do this is using a table.

8	4	2	1

Do you notice how the units increase by the power of 2 each time (going from right to left)?

Compare the denary number (13) to the first binary unit (8) to see whether it is equal to or greater than the denary number. 13 is greater than 8, which means that the unit 8 is required and you can write a 1 beneath it. 8 can then be subtracted from the denary value, $13 - 8 = 5$.

8	4	2	1
1			

Now compare the denary number (5) to the next binary unit (4) to see whether it is equal to or greater than the denary number. 5 is greater than 4, which means that the unit 4 is required and you can write a 1 beneath it. 4 can then be subtracted from the denary value, $5 - 4 = 1$.

8	4	2	1
1	1		

Then compare the denary number (1) to the next binary unit (2) to see whether it is equal to or greater than the denary number. 1 is not greater than 2, which means that the unit 2 is not required and you can write a 0 beneath it. No subtraction is needed this time as you haven't used the unit.

8	4	2	1
1	1	0	

Finally, compare the denary number (1) to the last binary unit (1) to see whether it is equal to or greater than the denary number. 1 is equal to 1, which means that the unit is required and you can write a 1 beneath it. 1 can then be subtracted from the denary value. $1 - 1 = 0$.

8	4	2	1
1	1	0	1

CONTINUED

This means that if you convert the denary number 13 to 4-bit binary, you get the binary number 1101.

Questions

- 1 What is the largest denary number that can be created with a 4-bit binary number?
- 2 Convert the denary number 6 to a 4-bit binary number.
- 3 Convert the binary number 9 to a 4-bit binary number.

To convert denary numbers to 8-bit binary the units that you need to use are 1, 2, 4, 8, 16, 32, 64 and 128. You can indicate, again, with a 1 or a 0 whether that unit is required. For example, if you convert the denary number 150 to an 8-bit binary number you can use the same method as before:

Write down the binary units from 128 to 1:

128	64	32	16	8	4	2	1

Compare the denary number (150) to the first binary unit (128) to see whether it is equal to or greater than the denary number. 150 is greater than 128, which means that the unit 128 is required and you can write a 1 beneath it. 128 can then be subtracted from the denary value, $128 - 150 = 22$.

128	64	32	16	8	4	2	1
1							

Next, compare the denary number (22) to the next binary unit (64) to see whether it is equal to or greater than the denary number. 22 is not greater than 64, which means that the unit 64 is not required and you can write a 0 beneath it. No subtraction is needed this time as you haven't used the unit.

128	64	32	16	8	4	2	1
1	0						

Then compare the denary number (22) to the next binary unit (32) to see whether it is equal to or greater than the denary number. 22 is not greater than 32, which means that the unit 32 is not required and you can write a 0 beneath it. No subtraction is needed this time as you haven't used the unit.

128	64	32	16	8	4	2	1
1	0	0					

Now, continue this process until you have compared all the binary units. Remember to subtract the unit from the denary number if it is required. You should end with the following result:

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

CONTINUED

This means that if you convert the denary number 150 to 8-bit binary, you get the binary number 10010110.

A quick check that you can do on your calculation is to look at the last unit on the right of your binary number. If the denary number you are converting is an odd number, this number should be 1. If the denary number you are converting is an even number, this number should be 0.

Questions

4 Convert the denary number 58 to an 8-bit binary number.

5 Convert the denary number 123 to an 8-bit binary number.

If you need to convert a binary number into a denary number, you can use simple addition. You just need to add together all the units that are represented by the value 1.

Try converting the binary number 1010. A simple way to do this is to use a table. You can write the units above and the binary number below:

8	4	2	1
1	0	1	0

You just need to add together all the units that have a 1 below. This means the calculation is $8 + 2 = 10$. Therefore, if you are converting the binary number 1010 to denary, you get the denary number 10.

If you need to convert an 8-bit binary number, you need to extend the units so that you have 8 of them. Remember to increase by the power of 2 each time.

Try converting the binary number 10011001. You can write the values in a table again to make it clearer:

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

You just need to add together all the units again that have a 1 below. This means the calculation is $128 + 16 + 8 + 1 = 153$. Therefore, if you are converting the binary number 10011001 to denary, you get the denary number 153.

Questions

6 What is the largest denary number that can be created with an 8-bit binary number?

7 Convert the 8-bit binary number 01100011 to a denary number.

8 You need to be able to convert up to 16-bit binary values. Fill in the missing units in the table and calculate the denary number for the given binary number.

								256	128	64	32	16	8	4	2	1
0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1

REFLECTION

How did you know how to add in the correct missing units for Question 8? How did you work out each unit?

Hexadecimal

Hexadecimal is also a number system. Humans can use hexadecimal, but computers cannot process it. Like denary numbers, hexadecimal numbers also need to be converted to binary to be processed by a computer.

Hexadecimal is a base-16 system and uses 16 symbols, these are the numbers 0–9 and the letters A–F. This is to allow the denary values 0–15 to be represented in hexadecimal. This is shown in Table 1.1.

Denary	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 1.1: Hexadecimal representations of denary numbers

The reason for using the symbols A–F for the denary numbers 10–15 is so only one symbol is needed for the denary number in hexadecimal.

SKILLS FOCUS 1.2

CONVERTING BETWEEN DENARY NUMBERS AND HEXADECIMAL

To convert a denary number to hexadecimal, it is easier to convert it to binary first. For example:

First convert the denary number 7 to 4-bit binary.

8	4	2	1
0	1	1	1

To convert the binary number to hexadecimal you can use simple addition again. $4 + 2 + 1 = 7$. If you look at the symbol table (Table 1.1), you can see that the hexadecimal for 7 is also 7. Therefore, the denary number 7 is 0111 as binary and 7 as hexadecimal.

So, what happens with denary numbers that are greater than 9?

First convert the denary number 12 to 4-bit binary.

8	4	2	1
1	1	0	0

To convert the binary number to hexadecimal you can use simple addition again. $8 + 4 = 12$. If you look at the symbol table (Table 1.1), you can see that the hexadecimal for 12 is C. Therefore, the denary number 12 is 1100 as binary and 12 as hexadecimal.

Converting the denary number to binary first isn't necessary for denary numbers that are less than 16, but it makes the process easier when you need to convert larger denary numbers.

So, how do you convert a larger denary number to hexadecimal?

CONTINUED

First, convert the denary number 201 to 8-bit binary. You know you can use 8-bit binary because the denary number is less than 256.

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

Each hexadecimal symbol only uses 4 bits. Therefore, you need to split the 8-bit binary number into two 4-bit binary numbers.

8	4	2	1	8	4	2	1
1	1	0	0	1	0	0	1

You then convert each 4-bit binary number again using simple addition.

$$8 + 4 = 12$$

$$8 + 1 = 9$$

The hexadecimal for 12 is C and the hexadecimal for 9 is 9. Therefore, the denary number 201 converted to binary is 11001001 and to hexadecimal is C9.

To convert a hexadecimal number to a binary number of a denary number you can just reverse the process.

To convert the hexadecimal number 5E to a binary number you need to convert each symbol to 4-bit binary. Remember E is the number 14.

8	4	2	1	8	4	2	1
0	1	0	1	1	1	1	0
5				E			

Therefore, the hexadecimal 5E converted to a binary number is 01011110.

To convert the hexadecimal 5E to denary, you can use the binary number that you have just calculated.

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

You can add all the units together to get the denary value. $64 + 16 + 8 + 4 + 2 = 94$.

The hexadecimal 5E converted to a denary number is 94.

Questions

- 1 Convert the hexadecimal A2 to a binary number.
- 2 Convert the binary number 100111010001 to hexadecimal.
- 3 Convert the denary number 350 to hexadecimal.
- 4 Convert the hexadecimal 3AC to a denary number.

Why is hexadecimal used in computer science?

Programmers often use hexadecimal when writing and analysing computer programs. They do this because it is easy for a human to read hexadecimal, rather than binary.

For example, if a programmer looked at a set of data that is stored in registers, as binary it could look like this:

10011100101110111000111011100010111101011010

The programmer may be trying to find out if the data stored matches the correct data which should be:

10011100101110111001011011100010111101011010

At a glance, it is difficult to see whether the two values match or if an error has occurred.

If that stored data is represented in hexadecimal it would be:

9CBB8EE2F5A

Comparing it to the correct value that should be:

9CBB96E2F5A

It is much easier to see that the data does not match, and the programmer can identify that an error has occurred.

The benefits of using hexadecimal rather than binary for a programmer are:

- Hexadecimal is a shorter way of representing binary, so it takes up less space on the computer screen or piece of paper.
- Hexadecimal is easier for a programmer to read, write and understand.
- Hexadecimal is easier to debug and locate errors in data.

ACTIVITY 1.1

Each computer has a **MAC address**. This MAC address is written in hexadecimal. This is one of the ways that hexadecimal is used in computer science.

Use the internet to find three more uses of hexadecimal in computer science.

KEY WORD

MAC address: a unique address that is given to a computer at the manufacturing stage that can be used to identify the computer.

1.2 Binary manipulation and negative numbers

Binary addition

You need to be able to add together two 8-bit binary numbers. When binary numbers are added together there are four rules that you will need to use:

Rule 1 is $0 + 0 = 0$

Rule 2 is $1 + 0 = 1$

Rule 3 is $1 + 1 = 10$ (this is because 10 in binary is 2)

Rule 4 is $1 + 1 + 1 = 11$ (this is because 11 in binary is 3)

You can apply these rules to add the binary numbers 10010100 and 00011110.

$$\begin{array}{r}
 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 \end{array}$$

You need to start at the right and begin applying the rules. The first calculation is $0 + 0$. Rule

The next calculation is $0 + 1$. Rule 2 means the result is 1.

$$\begin{array}{r}
 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 1 \quad 0
 \end{array}$$

The next calculation is $1 + 1$. Rule 3 means the result is 10. You need to write the 0 in the answer section and carry the 1 to the next calculation.

$$\begin{array}{r}
 1 \\
 + \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 0 \quad 1 \quad 0
 \end{array}$$

The next calculation is $1 + 0 + 1$, which results in $1 + 1$. Rule 3 means the result is 10. Write the 0 in the answer section again and carry the 1.

$$\begin{array}{r}
 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 0 \quad 0 \quad 1 \quad 0
 \end{array}$$

The next calculation is $1 + 1 + 1$. Rule 4 means the result is 11. Write 1 in the answer section and carry 1.

$$\begin{array}{r}
 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 1 \quad 0 \quad 0 \quad 1 \quad 0
 \end{array}$$

The next calculation is $1 + 0 + 0$, which results in $1 + 0$. Rule 2 means the result is 1. Write 1 in the answer section. There is no carry this time.

$$\begin{array}{r}
 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \\
 0 \\
 + \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0
 \end{array}$$

The next calculation is $0 + 0$. Rule 1 means the result is 0. Write 0 in the answer section.

			1	1	1			
+	1	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	0
			0	1	0	0	1	0

The final calculation is $1 + 0$. Rule 2 means the result is 1. Write 1 in the answer section.

			1	1	1			
+	1	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	0
	1	0	1	1	0	0	1	0

Therefore, when the binary numbers 10010100 and 00011110 are added, the answer is 10110010.

Questions

3 Add the 8-bit binary values 00011001 and 01000011.

4 Add the 8-bit binary values 00100110 and 01010111.

You should now know that the largest number that can be stored in an 8-bit binary register is 255. What happens when two 8-bit binary values are added, and the result is larger than 255?

The following example, which is almost complete, is very much like the previous calculation with a slightly different binary number. Can you spot the difference?

			1	1	1			
+	1	0	0	1	0	1	0	0
	1	0	0	1	1	1	1	0
			0	1	0	0	1	0

If you now try and do the final calculation, that would be $1 + 1 = 10$. You write 0 in the answer section, but there is nowhere to carry the 1. This is because the two 8-bit binary values are added the result is 306, which is greater than 255. Therefore, the number cannot fit in the 8-bit register and an **overflow error** is created.

			1	1	1			
+	1	0	0	1	0	1	0	0
	1	0	0	1	1	1	1	0
	1	0	0	1	1	0	0	0

1
Overflow error

KEY WORD

overflow error: a type of error that occurs when a number larger than a register can store is generated.

Each computer has a predefined limit, for example, 16-bit registers. What is the largest number that could be stored in a 16-bit register? If a number larger than this is generated in a process, an overflow error will occur.

Logical binary shifts

Another process that can occur with binary numbers that are stored in a register is called a logical binary shift. This kind of shift can move the binary values in the number to the left or to the right.

Consider the binary number:

0	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---

If a logical binary shift to the left is performed on the binary number, each value in the binary number is moved one place to the left. The result would be:

0	1	0	1	1	0	0	
---	---	---	---	---	---	---	--

You should notice that the left most 0 has been removed from the register. This is because there was no place to the left to shift this value. You should also notice that there is now an empty section in the register at the right. When a binary number is shifted, the resulting empty sections are filled with a 0.

0	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---

If you convert the first binary number to denary, you get the denary number 44. If you convert the binary number after the logical left shift has been performed, you get the denary number 88. This means that each logical shift to the left multiplies the binary number by 2.

Consider the binary number again:

0	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---

If a logical shift to the right is performed on the binary number, each value in the binary number is moved one place to the right. The result would be:

	0	0	1	0	1	1	0
--	---	---	---	---	---	---	---

You should notice that the right most 0 has been removed from the register. This is because there was no place to the right to shift this value. You should also notice that there is now an empty section in the register at the left. When a binary number is shifted, the resulting empty sections are, again, filled with a 0.

0	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

You already know that the first binary number converted to a denary number is 44. If you convert the binary number after the logical right shift has been performed, you get the denary number 22. This means that each logical shift to the right divides the binary number by 2.

Consider another binary number:

1	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

When a logical left shift is performed on the binary value the result is:

1	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---

When the first binary number is converted to a denary number, the result is 236.
When the binary number is converted after the logical shift to the left has been performed, the denary value is 216. You should notice that 236 multiplied by 2 is not 216. Therefore, one of the problems with a logical shift is that data can lose its integrity as vital bits can be removed from the register.

More than one logical shift can be performed on a register. For example, two logical shifts to the left can be performed. This time, each value in the register is shifted two places to the left.

The binary value:

1	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

would become:

1	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

Questions

- 5 Describe what happens in a logical binary shift.
- 6 Perform **two** logical shifts to the right on the binary number 11101100.

Negative binary numbers

All the binary numbers that you have used so far have been positive binary numbers. Numbers can sometimes be negative number though, for example, -150 . How are negative numbers represented as binary? Binary only has two values that can be used, a 0 and a 1, so you can't just put a minus symbol ($-$) in front of the number, as that symbol doesn't exist in binary. Therefore, a method exists that can be used to represent negative binary numbers and it is called two's complement. Most modern computers use this method.

The method of two's complement is a simple one. First you invert all the values in the binary number (change the 1s to 0s and the 0s to 1s), then you add 1 to the result. You can use this process to find out how the denary value -35 is represented as binary.

Convert 35 to a binary number:

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

Invert each of the values in the binary number:

1	1	0	1	1	1	0	0
---	---	---	---	---	---	---	---

Then simply add 1:

1	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

11011101 is the binary representation of -35 using the two's complement method.

1.3 How do computers represent text, images and sound?

You have already learnt that computers only process binary. This means that all data must be converted to binary to be processed by a computer. You have already looked at converting numbers, but text, images and sound all have to be converted to binary too, so they can be processed by a computer. How is this done?

Converting text to binary

When a computer converts text to binary to be processed it uses a tool called a **character set**. There are two main types of character set that a computer can use, ASCII and Unicode. An ASCII character set represents characters using 8-bit binary numbers. This means that it can represent up to 256 characters (0 to 255). This is fine for certain languages, but some languages have an alphabet that requires more than 256 characters; therefore, Unicode was invented. A Unicode character set represents characters using 16-bit binary numbers. This means it can represent a much greater number of characters, approximately 65 000. This means that many more characters can be developed in different alphabets and also symbols such as emojis can also be used.

Each character has a set binary value, for example, the letter A is 01000001.

KEY WORD

character set: it contains all the characters that are in that character set and the binary value that is assigned to each character.

ACTIVITY 1.2

Use the internet to find an example of an ASCII character set and a Unicode character set. What kind of different characters are represented? Are upper and lowercase letters the same?

Using the ASCII table, try to decode this message:

01000011011011110100110101110000011101010111010001100101011100
100101001101

10001101101001010001010110111001100011011001010110100101110011
011001100111

01010110111000100001

Remember, each letter will be represented as an 8-bit binary number.

Try writing a message for your friend to decode.

Converting images to binary

An image is made up of tiny dots called **pixels**. If an image was simply created using the colours black and white, each pixel would be either black or white. The binary value 1 could be used to represent the colour black and the binary value 0 could be used to represent the colour white.

KEY WORD

pixel: a very small dot of colour that is displayed with many others to create an image.

If each pixel is converted to its binary value, a data set such as the following could be created:

```
0011111000100000101000000011001010011000000011010001011001110010100000100
01111100
```

If the computer is informed that the image that should be created using this data is 9 pixels wide and 10 pixels high, it can set each pixel to black or white and create the image (See Figure 1.2). The type of data that is used to provide information, such as the dimensions and **resolution** of an image, is called **metadata**.

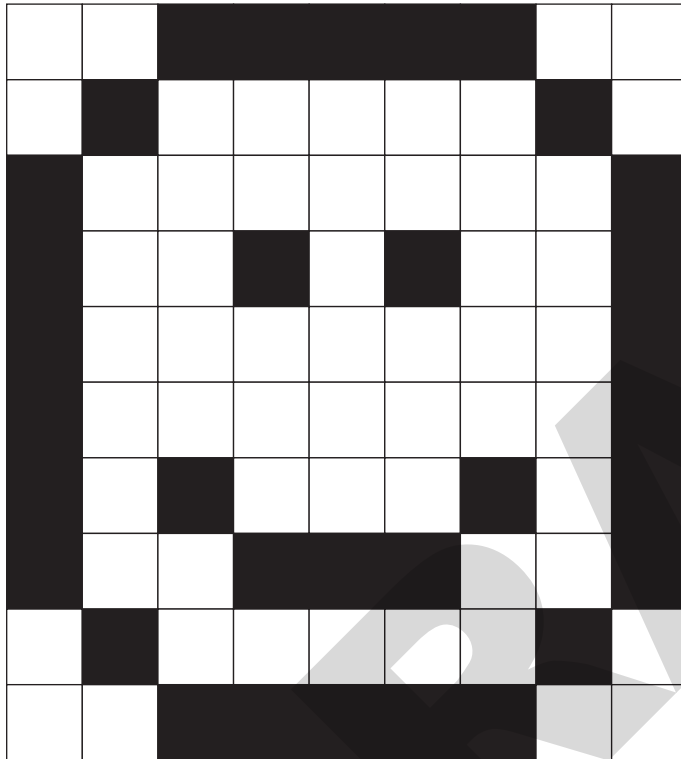


Figure 1.2: An image created just from black and white pixels

Most images use a lot more colours than black and white. Each colour has its own binary values. Colours are created by computer screens using the Red Green Blue (RGB) colour system. This system mixes the colours red, green and blue in different amounts to achieve each colour.

ACTIVITY 1.3

Use the internet to find an RGB colour scale. You should notice that each colour in the scale is first presented as hexadecimal. The first two hexadecimal values are the amount of red, the second two the amount of blue and the last two the amount of green.

Practise your hexadecimal to binary conversion by converting your favourite colour to its binary value.

KEY WORDS

resolution: the dimensions of an image.

metadata: additional data that is stored with an image that can provide information such as the dimensions of the image and the time and date the image was taken.

Most images are created using thousands of pixels. Each of those pixels needs quite a lot of data. Multiply that by the thousands of pixels that make up the image and that is a huge amount of data. This is why image files can often be so large in size.

Each image has a resolution and a **colour depth**. The resolution is the number of pixels wide and the number of pixels high that are used to create the image. The colour depth is the number of bits that are used to represent each colour. For example, each colour could be represented using 8-bit, 16-bit or 32-bit binary numbers (see Figure 1.3). The greater the number of bits, the greater the range of colours that can be represented.

If the colour depth of an image is reduced, the quality of the image is often reduced.

KEY WORD

colour depth: the number of bits that are used to create each colour in an image



Figure 1.3: 8-bit colour

16-bit colour

32-bit colour

Can you see how the quality of the image is reduced in comparison to the number of bits used?

If the resolution or the colour depth of an image is changed, this will have an effect on the size of the image file. If the resolution is increased, the image will be created using more pixels, so more data will need to be stored. If the colour depth of the image is increased, each pixel will need more data to display a greater range of colours so more data will need to be stored. Both will result in a larger file size for the image.

COMPUTER SCIENCE IN CONTEXT

People often have to sacrifice the quality of their images when displaying them online. This is because having files that are very large in size can take more time to download. This may frustrate users as it may mean they have to wait for a website to load, due to the number of large files it contains.

Converting sound to binary

Sound is made up of sound waves. When sound is recorded, this is done at set time intervals. This process is known as **sound sampling**.

KEY WORD

sound sampling: a little piece of sound that is recorded at regular time intervals.

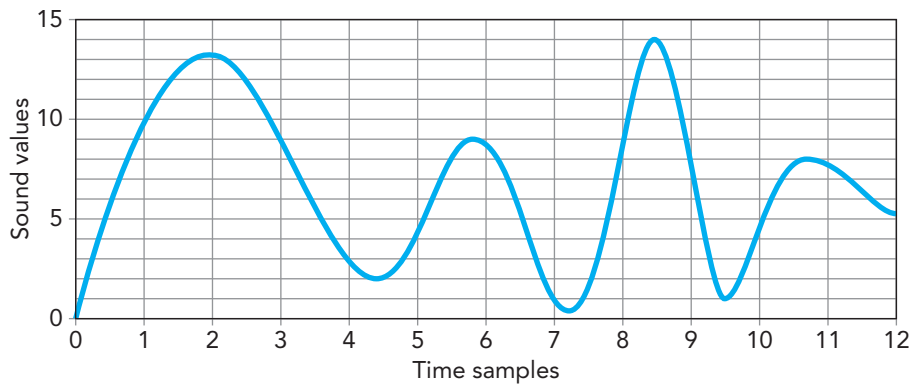


Figure 1.4: A sound wave

Figure 1.4 is an example of a sound wave.

If you recorded the sound at each time sample, you would have the following set of values:

Time sample	1	2	3	4	5	6	7	8	9	10	11	12
Sound value	9	13	9	3.5	4	9	1.5	9	8	5	8	5.5

Each sound value is converted to binary to be processed by the computer. Each sound value is then used to playback the sound, when the recording is played. If the recorded sound wave is played it would now look like Figure 1.5:

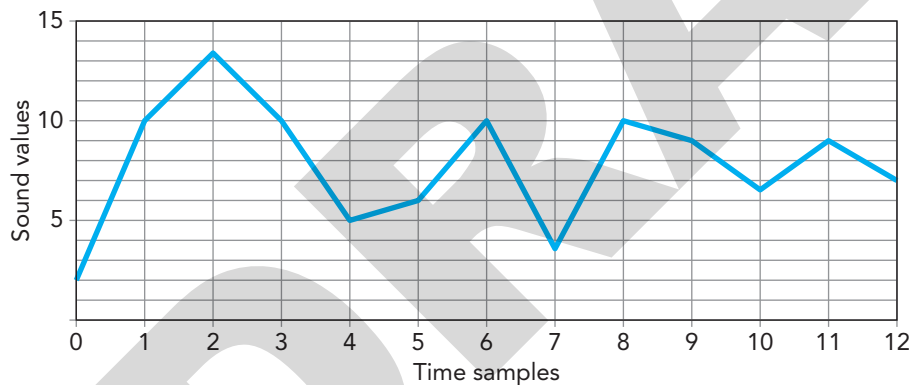


Figure 1.5: A sound wave created by playing back a recorded sound

You will notice that the two sound waves look different. This is because the number of samples taken is too far apart, so vital data in between time samples has been lost. This will result in a lack of accuracy in the sound that is recorded. Therefore, more samples need to be taken each second to improve the accuracy of the recorded sound. The number of samples taken in a second is called the **sample rate**. Sample rates are measured in hertz. 1 hertz is equal to 1 sample per second. A common sample rate is 44.1 khz (kilohertz), this would require 44 100 samples to be taken each second. That is a lot of data! If the sample rate is increased, the amount of data required for the recording is increased. This increases the size of the file that stores the sound.

Further data is required to represent sound, and this is the **sample resolution**. This is the number of bits that are used to represent each sample. A common sample resolution is

KEY WORDS

sample rate: the number of samples recorded each second.

sample resolution: the number of bits that are used to record each sound sample.

16-bit. The higher the sample resolution, the greater the variations in amplitude that can be stored for each sample. This means that aspects such as the loudness of the sound can be recorded more accurately. This will also increase the amount of data that needs to be stored for each sample. This increases the size of the file that stores the sound.

1.4 Measuring data storage

Earlier in the chapter, you learnt that all data need to be converted to digital data to be processed by a computer, and you now know how that is done for different types of data. But how is data measured when it is stored?

COMPUTER SCIENCE IN CONTEXT

There are times when you will need to know the size of a data file, such as when you are attaching a file to an email. Most email providers have a file size limit, so you need to know the size of that file, to know if it can be attached to your email.

Data is measured using different units, which are shown in Table 1.2.

Unit	Description
Bit	This is the smallest measurement for data
Nibble	There are 4 bits in a nibble
Byte	There are 8 bits in a byte
Kibibyte (KiB)	There are 1024 bytes in a Kibibyte
Mebibyte (MiB)	There are 1024 Kibibytes in a Mebibyte
Gibibyte (GiB)	There are 1024 Mebibytes in a Gibibyte
Tebibyte (TiB)	There are 1024 Gibibyte in a Tebibyte
Pebibyte (PiB)	There are 1024 Tebibytes in a Pebibyte
Exbibyte (EiB)	There are 1024 Pebibytes in an Exbibyte

Table 1.2: Data measuring units

Although most computers will calculate the size of a file for you, you need to be able to show that you can do this manually for the exam. This is just for an image file and a sound file.

SKILLS FOCUS 1.3

CALCULATING THE SIZE OF A FILE

There are several pieces of information that you need to know to calculate the size of an image file, these are:

- the number of images in the file
- the resolution of the image
- the colour depth of the image.

CONTINUED

When you know this information, you can perform a simple calculation to get the size of the image file, which is:

width of image $\times$ height of image $\times$ colour depth of image $\times$ number of images in file

Therefore, if you have the following information about an image file:

- 10 images in the file
- image resolution is 100×150
- colour depth is 8-bit colour,

you can calculate the size of the image file as follows:

- $100 \times 150 = 15,000$ (this means there are 15 000 pixels in the image)
- $15\,000 \times 8 = 120,000$ (this means that there are 120 000 bits in each image)
- $120\,000 \times 10 = 1\,200\,000$ (this means there are 1 200 000 bits in the image file).

You will be asked to calculate the size of an image file to a certain unit of measurement. The answer given here (1 200 000) is the number of bits that are in the image file. If you wanted to know how many bytes are in the image file, you need to divide the final answer by 8, as there are 8 bits in each byte, for example:

$$1\,200\,000 \div 8 = 150\,000 \text{ bytes}$$

If you wanted to know how many kibibytes the file was, you need to divide the answer in bytes by 1024, for example:

$$150\,000 \div 1024 = 146.5 \text{ KiB (to 1 decimal place)}$$

Questions

- 1 How many times would you need to divide the answer in bytes by 1024 to find the answer in Gibibytes?
- 2 How many pixels would there be in an image that has a resolution of 1000×1500 ?

There are several pieces of information that you need to know to calculate the size of a sound file, these are:

- the length of the soundtrack
- the sample rate of the soundtrack
- the sample resolution of the soundtrack.

When you know this information, you can perform a simple calculation to get the size of the sound file, which is:

$$\text{sample rate} \times \text{sample resolution} \times \text{length of soundtrack}$$

Therefore, if you have the following information about a sound file:

- length of 30 seconds
- sample rate of 44,100 hz
- sample resolution of 8 bits,

you can calculate the size of the sound file as follows:

- $44\,100 \times 8 = 352\,800$ (this means there are 352 800 bits per second)
- $352\,800 \times 30 = 10\,584\,000$ (this means there are 10 584 000 bits in the sound file).

CONTINUED

You will be asked to calculate the size of a sound file to a certain unit of measurement. The answer given (10 584 000) is the number of bits that are in the sound file. If you wanted to know how many bytes are in the sound file, you need to divide the final answer again by 8, as there are 8 bits in each byte, for example:

$$10\,584\,000 \div 8 = 1\,323\,000 \text{ bytes}$$

If you wanted to know how many mebibytes the file was, you need to divide the answer in bytes by 1024, then by 1024 again, for example:

$$1\,323\,000 \div 1024 \div 1024 = 1.3 \text{ MiB (to 1 decimal place)}$$

1.5 Data compression

You may already have experience of having large files and struggling to have enough storage space to store them. One thing that can be done to reduce the size of a file is **compression**.

Compressing a file will reduce the size of the file that can have several benefits, such as:

- Not as much storage space is needed to store the file.
- It will take less time to transmit the file from one device to another.
- It will be quicker to upload and download the file.
- Not as much bandwidth is needed to transmit the file over the internet.

There two type of compression that can be used, **lossy** and **lossless**. Each compresses the data in a different way.

Lossy compression uses a compression algorithm that finds the unnecessary and redundant data in the file. This data is permanently removed from the file. This type of compression is mainly used on an image file or a sound file.

Unnecessary data in an image file that could be removed are colours that the human eye cannot distinguish. This means that a lossy compression algorithm for an image file could reduce the size of the file by reducing the colour depth. It could also reduce the size of the file by reducing the image resolution, reducing the number of pixels used to create the image.

Unnecessary data in a sound file that could be removed are sounds that cannot be heard by the human ear. It could also be the removal of softer sounds that are played at the same time as louder sounds, as humans will only hear the louder sound. The method that is used to find these kinds of sounds in a sound file is called **perceptual music shaping**. This is the most common type of method that is used in the file format MP3. The size of a sound file can also be reduced by reducing the sample rate and the sample resolution.

KEY WORDS

compression: a method that uses an algorithm to reduce the size of a file.

lossy: a compression method that reduces the size of a file by permanently removing data.

lossless: a compression method that reduces the size of a file by temporarily altering the data.

KEY WORD

perceptual music shaping: a process that is used in lossy compression that removes sounds that are not normally heard by the human ear.

Questions

- 7 Why do you think lossy compression is not commonly used on text files?
- 8 Will lossy compression affect the quality of an image? Why?
- 9 Will lossy compression affect the quality of a sound recording? Why?

SELF ASSESSMENT

What were the points that you thought about regarding lossy compression to answer question 7? Did you relate these points back to why lossy compression is not suitable for a text file, or did you just make points about the drawbacks of lossy compression in general?

Lossless compression uses a compression algorithm to reduce the size of the file, without permanently removing any data. A common method that can be used to do this is by finding repeating data in the file and grouping this data together. For example, consider the following message:

WHEN IT IS SNOWING HEAVILY LOOK OUTSIDE.
LOOK OUTSIDE IT IS SNOWING HEAVILY.

Excluding the spaces between the words and the full stops, the message has a total of 62 characters. 1 character requires 1 byte of storage, so 62 bytes of storage would be needed to store this message.

When you look at the message, you should notice that it consists of words that are mostly repeated. Instead of storing all 62 characters individually, the words and the positions at which they occur in the message can be stored in a lookup table, for example:

Word	Position(s) in the message
WHEN	1
IT	2, 10
IS	3, 11
SNOWING	4, 12
HEAVILY	5, 13
LOOK	6, 8
OUTSIDE	7, 9

The lookup table needs 1 byte for each character in each word and 1 byte for each position the word occurs in the message. Therefore, 33 bytes are needed to store the words and 13 bytes to store the positions, giving a total of 46 bytes. This is much less than the 62 bytes we required with our original method. No data has been lost and we have reduced our storage requirements by 26%, quite a saving! To recreate the message, the computer simply retrieves the words and places them in the positions allocated.

A similar process can be applied to an image file with a lossless compression algorithm called **run length encoding (RLE)**. This algorithm will group together repeating pixels and store how many times they occur. For example, consider the simple image in Figure 1.6:

KEY WORD

run length encoding (RLE): An algorithm that groups together repeating patterns and indexes them.

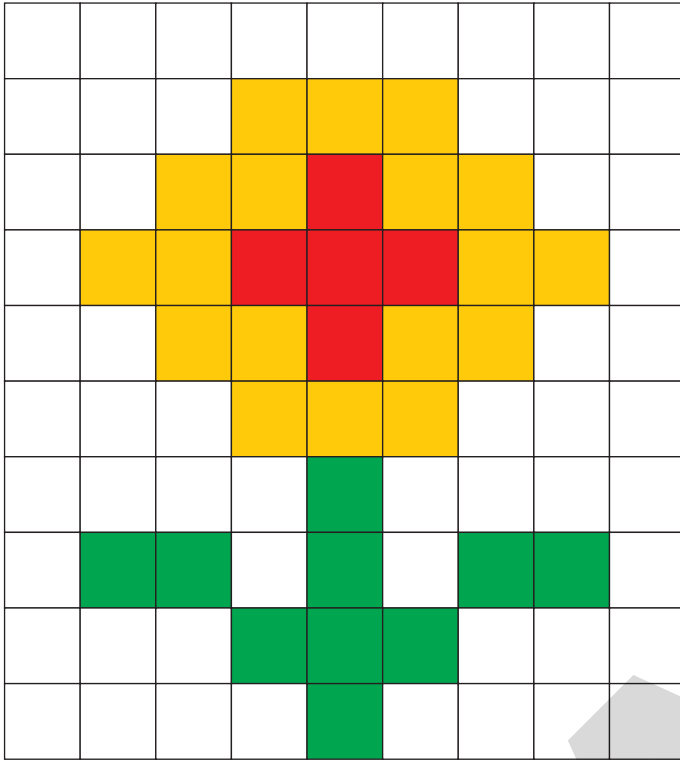


Figure 1.6: A simple pixel image

If W is white, Y is yellow, R is red and G is green, RLE could compress the image into the following data:

12W, 3Y, 5W, 2Y, 1R, 2Y, 3W, 2Y, 3R, 2Y, 3W, 2Y, 1R, 2Y, 5W, 3Y, 7W, 1G, 5W, 2G, 1W, 1G, 1W, 2G, 4W, 3G, 7W, 1G, 4W.

This means that the amount of data can be reduced as pixels are grouped together and the data for each pixel does not need to be stored. The resolution of the image is stored with the data, so that the computer knows the size to recreate it when the data is decompressed.

ACTIVITY 1.4

Why don't you try creating a simple image and writing out the RLE data for it? You could then give it to your friend, along with the image resolution, and they can have fun trying to recreate your image.

Peer Assessment

Ask your friend whether you got the RLE correct for your image in Activity 1.4. Ask them if you could have done anything to make your RLE any clearer. Ask them if you missed any important data.

SUMMARY

Computers require any data to be converted to binary before it can be processed by the computer.
Different number systems exist such as denary, binary and hexadecimal.
Hexadecimal is used in computer science for uses such as aiding programmers and representing MAC addresses.
Binary numbers can be added using four rules. If this creates a result greater than 255 for 8-bit binary numbers, an overflow error occurs.
Logical binary shifts can be performed on binary numbers. These can shift left or shift right and can be done multiple times. This changes the binary number and can affect the accuracy of the binary number.
Text is converted to binary using a character set such as ASCII or Unicode.
Images are converted to binary by taking each pixel and providing it with a colour code that is converted to a binary value. The larger the resolution and colour depth of an image, the greater the amount of data that will need to be stored in the image file. This also increases the quality of the image.
Sound is converted to binary by sampling the sound at a set time interval. The value of each sound sample is converted to binary. The greater the sample rate and resolution of the sound, the greater the amount of data that will need to be stored in the sound file. This also increases the accuracy of the recording.
Data is measured in different units such as bits, bytes and mebibytes.
Data can be compressed, using lossy or lossless compression, to reduce the size of a file. This means it will take up less storage space and it can be quicker to transmit the file.

EXAM-STYLE QUESTIONS

- There are two different types of data, known as analogue data and digital data.
 - State** what is meant by analogue data. [1]
 - State what is meant by digital data. [1]

[Total: 2]
- Describe** the difference between a denary number system and a binary number system. [4]
- A music venue has a small display screen built into the top of each seat. The display screen displays the number of the seat. Each seat number is displayed as hexadecimal.
 - Convert the two given seat numbers into 12-bit binary values. [4]
 - 21A
 - 90D
 - Write the seat number that would be displayed for the binary number 000010011100. [2]

[Total: 6]
- Two registers store the given 8-bit binary values. Register 1 stores 00110111 and register 2 stores 01110011.
 - Add the two binary values and write the result of the calculation. Show all your working. [3]

COMMAND WORDS

state: express in clear terms.

describe: state the points of a topic / give characteristics and main features.

CONTINUED

- b A logical left shift of three places is performed on Register 1. Write the contents of register 1 after the logical shift has been performed. [1]

[Total: 4]

- 5 Show how the value -28 is represented in binary using two's complement. Show all your working. [2]

- 6 A student wants to store some images that they have taken with their digital camera. They do not have enough storage space on their external hard disk drive to store the images at their current file size.

- a State how the student can reduce the amount of data stored for each image. [1]

- b The student has 20 images to store, that have a resolution of 150×250 and a colour depth of 16-bit.

Calculate the size of the file that will store the images in MiB. [3]

[Total: 4]

- 7 A blogger displays images of their daily life on their blog. They also have sound files of hints and tips that they record available on their blog.

- a The blogger compresses the images that they display on their blog. Give two benefits of compressing an image file. [2]

- b The blogger uses lossless compression to compress the sound file on their blog.

Explain how lossless compression will compress the sound file that the blogger makes available on their blog. [4]

[Total: 6]

- 8 Four file sizes are given.
Tick (✓) to show which file size is the largest.

File size	Tick (✓) the largest file size
1 GiB	
2033 bytes	
1050 MiB	
1500 KiB	

[1]

COMMAND WORDS

show (that): provide structured evidence that leads to a given result.

calculate: work out from given facts, figures or information.

give: produce an answer from a given source or recall / memory.

explain: set out purposes or reasons / make the relationships between things evident / provide why and/or how and support with relevant evidence.

SELF-EVALUATION CHECKLIST

After studying this chapter, think about how confident you are with the different topics. This will help you to see any gaps in your knowledge and help you to learn more effectively.

You might find it helpful to rate how confident you are for each of these statements when you are revising. You should revisit any topics that you rated 'Needs more work' or 'Getting there'.

CONTINUED

I can...	See topic	Needs more work	Getting there	Confident to move on
explain why data needs to be converted to binary to be processed by a computer.	1.1			
convert between denary numbers and binary numbers.	1.1			
convert between denary numbers and hexadecimal.	1.1			
convert between binary numbers and hexadecimal.	1.1			
provide examples of the use of hexadecimal in computer science.	1.1			
explain why hexadecimal is used in computer science.	1.1			
add two 8-bit binary numbers.	1.2			
explain what is meant by an overflow error.	1.2			
perform a logical binary shift on a binary number, including left and right shifts and multiple shifts.	1.2			
use two's complement to represent negative numbers.	1.2			
explain how character sets are used by a computer to represent text.	1.3			
explain how pixels are used by a computer to create images.	1.3			
explain how the file size of an image is affected by the image resolution and colour depth.	1.3			
explain how a sound wave is sampled by a computer.	1.3			
explain how the file size of a sound recording is affected by the sample rate and sample resolution.	1.3			
understand how the size of a data file is measured.	1.4			
calculate the size of an image file.	1.4			
calculate the size of a sound file.	1.4			
understand the need for data compression and the effect it has on a file.	1.5			
explain how different types of data are compressed using lossy compression.	1.5			
explain how different types of file are compressed using lossless compression.	1.5			



CAMBRIDGE
UNIVERSITY PRESS

Computer Science

for Cambridge IGCSE™ & O Level



Digital Teacher's Resource



Cambridge Assessment
International Education

Endorsed for teacher support

We are working with Cambridge Assessment International Education towards endorsement of these titles.

DRAFT

> Contents

The items in **orange** are freely accessible to teachers on the ‘supporting resources’ area of the Cambridge GO platform.

Introduction

About the authors

How to use this series

How to use this Teacher’s Resource

How to use this Teacher’s Resource as Professional Development

About the syllabuses

About the examination

Lesson plan template

Syllabus content correlations

Teaching strategies

Active learning

Assessment for learning

Improving learning through questioning

Metacognition

Differentiation

Language awareness

Developing learners’ language skills

Skills for life

Teaching plans

Teaching notes

Chapter 1: Data representation

Chapter 2: Data transmission

Chapter 3: Hardware

Chapter 4: Software

Chapter 5: The internet and its uses

Chapter 6: Automated and emerging technologies

Chapter 7: Algorithm design and problem solving

Chapter 8: Programming

Chapter 9: Databases

Chapter 10: Boolean logic

Guidance for programming scenario tasks

Programming ideas

Differentiated worksheets

Practice exam-style paper:

Computer Systems

Practice exam-style paper:

Algorithms, Programming and Logic

Glossary

Coursebook Answers

Worksheet Answers

Practice exam-style papers mark scheme

> How to use this teacher's resource

This digital teacher's resource contains both general guidance and teaching notes that help you to deliver the content in our Cambridge IGCSE and O Level Computer Science Coursebook. You will find **answers** to the Coursebook questions and exercises on the supporting resources area of Cambridge GO – they are freely available to teachers only.

There are **teaching notes** for each chapter of the Coursebook. Please note that sometimes subsections in the coursebook are combined together in one sub-chapter of the digital teacher's resource. You can see an overview of where all topics are covered in the downloadable teaching plan, in the 'resources' column. Each set of teaching notes contains the following features to help you deliver the chapter.

At the start of each chapter there is a **teaching plan** for that chapter. This summarises the topics covered in the chapter, including the number of learning hours recommended for each topic, an outline of the learning content, and the resources from this series that can be used to deliver the topic:

Sub-chapter	Suggested learning hours	Learning content	Resources
6.1 Automated systems	3	Describe how sensors, microprocessors and actuators can be used in collaboration to create automated systems. Describe the advantages and disadvantages of an automated system in a given scenario.	Coursebook: 6.1 Automated systems  Worksheet 6.1: Automated systems

Each chapter also includes information on any **background knowledge** that students should have before studying this chapter, advice on helpful **language support**, and a **teaching skills focus** that will help you develop your skills across a number of key pedagogical areas within the context of different syllabus contexts.

BACKGROUND KNOWLEDGE

- Learners should be familiar with the function of AND, OR and NOT from their programming experience. They should also be aware that computers use logic gates through the hardware theory section.
- There is no other expected knowledge from learners.

Before the teaching notes for the individual sections, there are a selection of useful links to **digital resources**.

Links to digital resources

- www.cambridge.org/links/kctd7001
Interactive binary counter to see how the bits affect the number.
- www.cambridge.org/links/kctd7002
Interactive text RLE encoder where text can be entered and it will identify repeated elements that can be compressed.

At the beginning of the teaching notes for the individual sections there is an outline of the **learning plan** for that section.

We are working with Cambridge Assessment International Education towards endorsement of these titles.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

LEARNING PLAN

Syllabus learning objectives	Learning intentions	Success criteria
Understand how and why computers use binary to represent all forms of data.	Identify why computers represent data in binary.	Learners will be able to identify why computers represent data in binary.

There is also a feature highlighting any **common misconceptions** that students may have about the topic and how you can overcome these.

Common misconceptions

Misconception	How to elicit	How to overcome
Learners may try and work out binary from left to right, i.e. 1 2 4 8, etc. instead of right to left, i.e. 8 4 2 1.	Ask learners to demonstrate how they convert binary to denary and vice-versa.	Correct learners when used incorrectly, and whenever giving examples place the values above the binary digits and reinforce right to left.

For each section, there is a selection of **starter ideas**, **main activities** and **plenary ideas**. You can pick out individual ideas and mix and match them depending on the needs of your class. The activities include suggestions for how they can be differentiated or used for assessment.

Starter ideas

1 Getting started (10 minutes)

Description and purpose: Ask learners to compare the words they have identified with another pair, and identify any words they have not included. Ask learners for their words and create a class-set on the board. Ask learners for definitions of their terms as they give them, and explain any concepts that come up.

At the end of the teaching notes for the individual sections are a selection **homework ideas**.

Homework ideas

1 Calculations

Description: Give learners a set of denary and binary numbers for them to convert to the other. This will allow learners to apply their knowledge.

This digital teacher's resource also includes **worksheets** for each chapter, a power point and two **exam-style papers**.

- ↓ To help with lesson planning, a blank **lesson plan template** is available to download from Cambridge GO (as part of this digital teacher's resource).

> How to use this teacher's resource to supplement PD

We regularly hear from teachers that the Professional Development (PD) they feel they get the most out of is face-to-face training. However, we also hear that not all teachers have the time or budget to get out of the classroom, so here's some handy suggestions and information about how to use this digital teacher's resource to supplement your own professional development. After all, we are all lifelong learners!

Teaching skills focus

We have created a new 'teaching skill focus' feature that appears once every chapter, covering a different teaching skill with suggestions of how you can implement it in the teaching of the topic. From differentiation, to assessment for learning, to metacognition, this feature aims to support you with trying out a new technique or approach in your classroom and reflecting upon your own practices.

Try it out once per teaching topic, or when you have time, and develop your skills in a supported and contextualised way.

Approaches to learning and teaching

Our teacher's resources now contain guidance on the key pedagogies underpinning our course content and how we understand and define them. You can find detailed information for you to read in your own time about active learning, assessment for learning, metacognition, differentiation, language awareness and skills for life taken from our *Approaches to Learning and Teaching* series.

Why not try reading each support document alongside the relevant Teaching Skills Focus for an extra bit of bedtime reading?

Teaching activity ideas

This digital teacher's resource contains over 200 starter, main and plenary activity ideas for you to choose from and find the ones that work best in your classroom. Use them to support your creativity, breathe new life into a topic and build upon them with your own ideas.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

The information in this section is based on the 0478/0984/2210 syllabuses for examination from 2023. You should always refer to the appropriate syllabus document for the year of your students' examination to confirm the details and for more information. The syllabus document is available on the Cambridge International website at www.cambridgeinternational.org.

> About the syllabuses

The Cambridge IGCSE and O Level Computer Science syllabuses (0478/0984/2210) have been updated for first examining from September 2023. They have been designed to provide a comprehensive and engaging introduction to Computer Science. The syllabuses are divided into ten topic areas. Component one of the syllabus on computer systems focuses on the six topic areas namely: Data presentation; Data transmission; Hardware; Software; The internet and its uses and Automated; and emerging technologies. The remaining four topics: Algorithm design and problem-solving; programming; Databases and Boolean logic focus on component two of the syllabus – Algorithms, programming and logic.

Learners are required to undertake practical work through the qualifications, gaining experience in decomposing problems, producing algorithms, writing computer programs in a high-level language, and exploring SQL in databases. These topics will be assessed on paper, so learners should also have experience of producing these algorithms without the use of a computer.

There are a number of changes in these qualifications from the previous versions such as:

- Introduction of new topics such as Artificial Intelligence and SQL.
- Reduction in some areas such as HTML.
- Expansion of programming requirements such as 2D arrays and file handling.
- Movement of topics between the papers such as Boolean logic.

📄 **Syllabus correlation grid** from Cambridge GO.

> About the examination

There are two papers: one is based on Computer systems and the other one is based on Algorithms, Programming and Logic. Learners need to take both papers.

Both papers cover the three Assessment Objectives.

The assessment objectives (AOs) are:

AO1

Demonstrate knowledge and understanding of the principles and concepts of computer science.

AO2

Apply knowledge and understanding of the principles and concepts of computer sciences to a given context, including the analysis and design of computational or programming problems.

AO3

Provide solutions to problems by:

- evaluating computer systems
- making reasoned judgements
- presenting conclusions.

We are working with Cambridge Assessment International Education towards endorsement of these titles.

> **CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE**

Computer systems Paper consists of structured and free response questions covering topics 1–6 in the specification. This paper has a focus on computer science theory and applications of this theory. This paper has the largest focus on AO1 (60%) and then an equal split between AO2 and AO3. This means that there is more of an emphasis on knowledge and understanding of the topic areas, but there will be requirements to apply this knowledge and to make reasoned judgements.

The Paper on Algorithms, Programming and Logic consists of structure and free response questions covering topics 7–10 in the specification. This paper has a focus on problem solving with the 60% of the marks coming from AO1 – application of knowledge. There is then an equal spread between AO1 and AO2.

The final question in the Paper on Algorithms, Programming and Logic will be a banded response question. A scenario will be described that requires a computer program to solve, and learners will need to write an algorithm to solve this problem. This question is not marked by point, the mark scheme is banded and will list what is expected for each stage.

There is no choice of questions on any of the papers: they must all be answered.

> 1 Data representation

Please note that sometimes sub-chapters in the coursebook are combined together in one sub-chapter of this teacher's resource. Please check the 'resources' column of this teaching plan to see which coursebook sub-chapters are covered in each teacher's resource sub-chapter.

Teaching plan

Sub-chapter	Suggested learning hours	Learning content	Resources
1.1 Binary and hexadecimal	6	Understand why computers use binary. Convert between binary and denary, and vice-versa. Understand the use of hexadecimal. Convert between hexadecimal and denary, and hexadecimal and binary.	Coursebook: 1.1 Binary and hexadecimal Programming Book for Python: 8.2 FOR Loops Programming Book for Microsoft Visual Basic: 5.3 Using the loop counter 📄 Worksheet 1.1: Hexadecimals
1.2 Binary manipulation and negative numbers	3	Add two positive 8-bit binary integers and understand the concept of overflow. Perform a logical binary shift left and right on a positive 8-bit binary integer. Use two's complement to represent negative 8-bit binary integers.	Coursebook: 1.2 Binary manipulation and negative numbers 📄 Addition.pptx 📄 Worksheet 1.2: Two's complement
1.3 Text, sound and images	3	Understand how and why a computer represents text, and the use of character sets, including ASCII and Unicode. Understand how a computer represents sound. Understand how a computer represents an image.	Coursebook: 1.3 How do computers represent text, images and sound? 📄 Worksheet 1.3: Crack the code
1.4 Data storage and compression	2	Understand how data storage is measured. Calculate the file size of an image and sound file. Understand the purpose and need for data compression. Understand how lossy and lossless compression works.	Coursebook: 1.4 Measuring data storage 1.5 Data compression

CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

BACKGROUND KNOWLEDGE

- Learners should be aware that computers store data in binary. Some learners may have converted binary numbers into denary in mathematics.
- Learners should know that a computer can store a range of data including sound and video.
- Learners should be familiar with data storage values, most likely bytes, kilobytes, megabytes and gigabytes. They are unlikely to have used kibibytes, etc. and will need the differences clearly explaining.
- This section does not require learners to understand how the sound and images are captured, i.e. how a microphone works, they only need to be able to explain how the recorded data is stored in a file.

TEACHING SKILLS FOCUS

Area of focus: active learning. The focus of learning is on the learners themselves to support how they learn, instead of what they are learning. By exploration of a topic, and physical interaction, they are learning through their own successes and failures.

Specific focus: independent work. This is where learners explore a topic individually, without reliance on another person. It does not stop learners from interacting with peers, but they are performing the tasks on their own.

Benefits of independent work: supports research and exploration skills, requires learners to make decisions about what they do next, and allows them to make mistakes and find their own solutions.

Practice: give learners a grid where they can draw their own images by colouring in the squares. By selecting binary codes for the colours they have chosen they can explore how images are stored. By encouraging this exploration learners will develop independent skills.

The use of active learning allows learners to work out answers for themselves, by performing the role of the computer instead of just looking at how it works.

Learners will come across problems, for example, that 3 bits is not enough to represent 10 different colours. By actually finding and implementing the changes needed, they should remember this process more than if they were shown it.

Reflect: consider the following questions:

- Did the learners engage with the activity?
- Was there enough support for the less able to explore the topic?
- Were there enough extension activities to challenge the more able?
- How did learners tackle the problems they encountered? Were they reliant on being told the answer, or did they attempt to solve it themselves?
- What would you change next time you use active learning?

LANGUAGE SUPPORT

For definitions of key words, please see the glossary.

Some terms can appear in more than one place with a different use, for example, *resolution* can be *image resolution* or *sound resolution*. Make sure in discussions that the reference to image or sound is clear, i.e. not using resolution in isolation.

The data storage methods have very similar names and acronyms and learners need to be

able to differentiate between these. The first three (*bit*, *nibble*, *byte*) can be related to eating a biscuit and the amount eaten. *Kibibyte* can be linked to kilo being 1000.

Lossy and **lossless** compression can be remembered by using their names, i.e. *lossy compression* means that data is lost permanently, whereas *lossless* means that all the data remains.

Links to digital resources

- www.cambridge.org/links/kctd7001
Interactive binary counter to see how the bits affect the number.
- www.cambridge.org/links/kctd7002
Interactive text RLE encoder where text can be entered and it will identify repeated elements that can be compressed.
- www.cambridge.org/links/kctd7003
Interactive image storage that lets you change the colour of a pixel and then shows how this changes the colour in the stored data.
- www.cambridge.org/links/kctd7004
Interactive bit shifting webpage where learners can enter a binary number and then see the result of a shift.
- www.cambridge.org/links/kctd7005
Interactive binary calculator that allows you to enter two binary numbers and outputs the addition.
- ASCII and UNICODE character tables

1.1 Binary and hexadecimal

This section covers Coursebook sub-chapter topic 1.1.

LEARNING PLAN		
Syllabus learning objectives	Learning intentions	Success criteria
• Understand how and why computers use binary to represent all forms of data.	• Identify why computers represent data in binary.	Learners will be able to identify why computers represent data in binary.
• Understand the denary, binary and hexadecimal number systems.	• Identify the number bases for denary and binary.	Learners will be able to identify that denary is base 10 and that binary is base 2.
• Convert between: • positive denary and positive binary • positive denary and positive hexadecimal • positive hexadecimal and positive binary.	• Convert a binary number to denary. • Convert a denary number to binary. • Convert a binary number to hexadecimal. • Convert a hexadecimal number to binary. • Convert a denary number to hexadecimal. • Convert a hexadecimal number to denary.	Learners will be able to convert between binary and denary. Learners will be able to convert between binary and hexadecimal, and denary and hexadecimal.
• Understand how and why hexadecimal is used as a beneficial method of data representation.	• Identify uses of hexadecimal. • Explain why hexadecimal is used.	Learners will be able to identify uses of hexadecimal and explain why hexadecimal is used.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

Common misconceptions

Misconception	How to elicit	How to overcome
Learners may try and work out binary from left to right, i.e. 1 2 4 8, etc. instead of right to left, i.e. 8 4 2 1.	Ask learners to demonstrate how they convert binary to denary and vice-versa.	Correct learners when used incorrectly, and whenever giving examples place the values above the binary digits and reinforce right to left.
That the smallest binary number that can be represented is 0.	Ask learners what the smallest number is for a set number of bits.	Show learners that there is also the number 0.
The numbers above the binary digits start with 0 or 2, e.g. 8 4 2 or 8 4 2 1.	Ask learners to demonstrate how they convert binary to denary and vice-versa.	Correct learners where this is done and reinforce starting with 1 and then multiplying by 2 repeatedly.
Learners treat hexadecimal numbers such as 21 as the denary number 21.	Give learners example questions with these types of hexadecimal number.	Ask learners for their answers and make it clear when a number is hexadecimal and when it is not.
Learners mix the letters from 10 and above, e.g. allowing 16 as G.	Ask learners to write the hexadecimal values from 0 to F.	Show learners that only 0 to F can be represented in 4 bits.

Starter ideas

1 Getting started (10 minutes)

Description and purpose: Carry out the Getting started activity in chapter 1 of the Coursebook.

Ask learners to compare the words they have identified with another pair, and identify any words they have not included. Ask learners for their words and create a class-set on the board. Ask learners for definitions of their terms as they give them, and explain any concepts that come up.

What to do next: Introduce the idea of data being stored in binary and that text, sound and images are all pieces of data that can be stored in binary.

2 Binary combinations (10 minutes)

Description and purpose: Show learners a 4-digit binary number, e.g. 0101. Ask learners to write as many different 4-digit binary numbers as they can think of. As learners finish tell them that there should be 16 different combinations, ask them to check if they have 16. Ask learners to swap their lists and identify which are missing (if any), and if there are any duplicates.

What to do next: Main teaching idea 1, 'Binary combinations'.

Main teaching ideas

1 Binary combinations (30 minutes)

Learning intention: Understand how to convert a binary number to denary.

Description and purpose: After Starter idea 2, ask learners to repeat the process with 5 binary numbers. Ask learners how many combinations they found and tell them that they should have 32. Ask learners to work out the relationship between the quantity of binary numbers and how many combinations there are. Link to number bases. Ask learners what standard decimal numbers are (denary), i.e. base 10. Explain that binary is base 2.

> Differentiation ideas:

Support – give learners a paper template with columns for them to separate the binary digits and view them in a more structured manner.

Challenge – ask learners to look at 7+ digits and what the combinations are.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

2 Conversion spreadsheet (30 minutes)

Learning intention: Convert denary numbers to binary and vice-versa.

Description and purpose: Recap basic spreadsheet formulae and how to implement these in a spreadsheet. Ask learners to work in pairs to create a spreadsheet that allows the user to enter an 8-bit binary number and calculate the value, and then the same for a denary number.

> **Differentiation ideas:**

Support – give learners a partially complete spreadsheet, e.g. where it tells them where the user will enter data, and tells them where to put their formulae.

Challenge – ask learners to increase up to 16-bits.

> **Assessment ideas:** Ask learners to test each other's systems and evaluate how well they work.

3 Hexadecimal and binary (20 minutes)

Learning intention: Understand how to convert a binary number to hexadecimal and vice-versa.

Resources: Worksheet 1.1, 'Hexadecimals'.

Description and purpose: Introduce learners to the process of converting a binary number into its hexadecimal equivalent; splitting into groups of 4, then giving each group one digit from 0 to F. Ask learners to complete Part 1 of the Hexadecimal worksheet. Ask learners for their answers and to demonstrate how they calculated them, give learners the correct answers for each question. Repeat but converting a hexadecimal number into binary by putting each digit into 4 bits. Ask learners to complete Part 2 of the Hexadecimal worksheet. Ask learners for their answers and to demonstrate how they completed them.

> **Differentiation ideas:**

Support – give learners a list of the binary numbers and the hexadecimal equivalents for them to identify the correct digits for each.

Challenge – ask learners to write a hexadecimal number for another learner to calculate.

> **Assessment ideas:** Ask learners to mark their own work and make corrections separately.

Plenary ideas

1 Special values (10 minutes)

Description and purpose: Ask learners to work out what is the smallest number that can be represented by 5 bits (0), the largest number that can be represented by 5 bits (1), and the quantity of different numbers that can be represented by 5 bits (32). Ask learners to repeat this with 6 bits. Ask learners to identify the pattern; the quantity of numbers is the next binary value, e.g. in 5 bits there are 32 combination which is the value of the 6th binary digit.

2 Number match (10 minutes)

Resources: List of binary, hexadecimal and denary numbers.

Description and purpose: Ask learners to match each binary number with its denary and/or hexadecimal number. These could be on a board, or on individual pieces of paper, or they could have to draw lines between them.

> **Assessment ideas:** Ask learners to check each other's answers and where there are differences work out what the correct answer is.

Homework ideas

1 Calculations

Description: Give learners a set of denary and binary numbers for them to convert to the other. This will allow learners to apply their knowledge.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

2 Find your MAC/IP address

Description: Ask learners to find either the MAC address of one of their computers, or the IP address, and then write this number in denary, binary and hexadecimal. This will allow learners to apply their knowledge and explore future topics in networking.

1.2 Binary manipulation and negative numbers

This section covers Coursebook sub-chapter topic 1.2.

LEARNING PLAN		
Syllabus learning objectives	Learning intentions	Success criteria
Add two positive 8-bit binary integers.	Perform binary addition on two positive 8-bit binary integers.	Learners will be able to add together two 8-bit binary integers.
Understand the concept of overflow and why it occurs in binary addition.	Explain when overflow occurs and its meaning.	Learners will be able to identify when overflow occurs and explain its meaning.
Perform a logical binary shift on a positive 8-bit binary integer and understand the effect this has on the positive binary integer.	- Perform a left and right binary shift. - Explain the effect of a shift on a binary number.	Learners will be able to shift a binary number left and right, and explain the effect of a shift on a binary number.
Use two's complement to represent negative 8-bit binary integers.	- Convert a positive binary integer to a two's complement 8-bit integer and vice-versa. - Convert a negative denary integer to a two's complement 8-bit integer and vice-versa.	Learners will be able to convert positive and negative denary integers into two's complement, and vice-versa.

Common misconceptions

Misconception	How to elicit	How to overcome
Learners may attempt to keep the bits lost in a left or right shift.	Ask learners to show on the board the result of a shift.	Give a narrative to the learners explaining if it's correct why – and if it's wrong why.
Learners may not carry in addition.	Give learners questions where they need to add binary numbers and show their working.	Work through an answer with the learner and reinforce how to use the carries.
Learners may attempt to add from left to right.	Give learners questions where they need to perform shifts.	Show learners how to work one answer and ask them to follow the same method for the next.
Learners may give a positive two's complement binary number starting with a 1.	Ask learners to write the two's complement for +200.	Show learners how leaving a 1 at the start will mean it's actually a negative number, and they have to add another 0 at the start. Link back to overflow in addition.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

Starter ideas

1 Adding (10 minutes)

Resources: Addition.pptx.

Description and purpose: Show learners the first slide and ask them to convert the binary number to denary, and then add these values together. Ask learners for their answers and show them the answers. Ask learners to then convert these denary answers back into binary.

What to do next: Main teaching idea 1, 'Shifting'.

2 Shifting (10 minutes)

Resources: Document with binary numbers, as well as their shifted results.

Description and purpose: Give learners a list of binary numbers that have the same combinations of 1s and 0s but shifted, for example, 00001101 and 00011010. Ask learners to work out the difference between the two values. Repeat this with a number of questions. Ask learners for their answers and link to the role, e.g. multiplication or division by 2, 4, etc.

What to do next: Main teaching idea 3, 'Negative numbers'.

Main teaching ideas

1 Shifting (30 minutes)

Learning intention: Perform a logical left and right binary shift on a positive binary integer. Understand the loss of data in a logical shift.

Description and purpose: Explain that each shift to the left multiplies the binary number by 2. Show learners with an example, work out the denary of the number before and after.

Explain that each shift to the right divides the binary number by 2. Show learners with an example, work out the denary of the number before and after.

Show learners that the bits at either end of the register are lost when a logical shift is performed, e.g. 10101010 multiplied by 2 is 01010100. Discuss how logical shifts lose precision.

Write several binary numbers on a board and a shift for learners to perform on each one. Ask learners for their answers and to demonstrate the steps.

> Differentiation ideas:

Support – put learners with a stronger learner when working through the answers.

Challenge – ask learners to consider how the drawback of loss of precision could be dealt with, such as using additional register to store the values.

2 Positive numbers (15 minutes)

Learning intention: Convert a positive binary integer to a two's complement 8-bit integer and vice-versa.

Resources: Worksheet 1.2, 'Two's complement'.

Description and purpose: Explain to learners that binary numbers can represent negative numbers. Show learners how the first bit in a two's complement binary integer identifies if it is positive (0) or negative (1).

Explain how positive numbers are worked out exactly the same as ordinary binary numbers. Show learners a set of worked examples. Give learners Worksheet 1.2 and ask them to complete Part 1. Ask learners for their answers and to mark and correct their own work.

> Differentiation ideas:

Support – give learners a binary table with the key values 8 4 2 1, etc. written in to help them.

3 Negative numbers (40 minutes)

Learning intention: Convert a negative binary integer to a two's complement 8-bit integer and vice-versa.

Resources: Worksheet 1.2, 'Two's complement'.

Description and purpose: Explain that when a two's complement binary number starts with a 1, it needs to be converted to work out its value. Select one method to demonstrate to learners how to convert a negative two's complement binary number to denary. Work through a number of negative

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

binary two's complement integers, ask learners to tell you what each stage is. Ask learners to complete Worksheet 1.2 Part 2. Ask learners for their answers and to mark and correct their own work.

Repeat, but turning a negative denary integer into two's complement. Ask learners to complete Worksheet 1.2 Part 3. Ask learners for their answers and to mark and correct their own work.

> Differentiation ideas:

Support – give learners a list of instructions to follow for each conversion that they can follow.

Challenge – ask learners to work out the smallest number that can be represented in 8-bits, the largest number and how many different numbers there are.

> **Assessment ideas:** Learners can mark and correct their own answers.

Plenary ideas

1 Shifting (10 minutes)

Resources: List of shifts and their results.

Description and purpose: Ask learners to match the shifts with their results such as 1 shift left will match with multiply by 2.

2 Two's complement match (10 minutes)

Resources: List of two's complement binary integers and their descriptions.

Description and purpose: Give learners a set of two's complement binary integers, and descriptions, e.g. the largest smallest 8-bit number, the two's complement binary integer –23, etc. and ask learners to match the descriptions with the answers. Ask learners for their answers and discuss any errors.

> **Assessment ideas:** Learners can check their own learning when reviewing the answers.

Homework ideas

1 Coursebook questions

Description: Ask learners to complete the questions in the coursebook and then their own answers in the next lesson. This will give learners experience of giving answers in writing.

2 Key binary and hexadecimal cards

Description: Give learners a set of cards with key words or questions from Coursebook, sub-chapter 1.2. Ask learners to complete the back with a description or definition. This will consolidate understanding of terms and provide a testing resource for future use.

1.3 Text, sound and images

This section covers Coursebook sub-chapter topic 1.3.

LEARNING PLAN

Syllabus learning objectives	Learning intentions	Success criteria
Understand how and why a computer represents text and the use of character sets, including American standard code for information interchange (ASCII) and UNICODE.	Explain how text is represented by a character set.	Learners will be able to describe how a computer represents characters, and the characteristics of ASCII and UNICODE.

CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

CONTINUED

Syllabus learning objectives	Learning intentions	Success criteria
Understand how and why a computer represents sound, including the effects of the sample rate and sample resolution.	Describe how a sound wave is sampled to produce a sequence of binary values.	Learners will be able to define key terms such as sampling, sample rate, resolution, etc. Learners will be able to describe how a sound file is sampled and how its sample rate and sampling resolution will affect the sound and the sound file.
Understand how and why a computer represents an image, including the effects of the resolution and colour depth.	Describe how a bitmap image is stored in a computer, including key terms such as pixel, resolution and colour depth.	Learners will be able to describe how an image is represented in binary, and how the resolution and colour depth will affect the image and the image file.

Common misconceptions

Misconception	How to elicit	How to overcome
Learners often mix up the quality of sound and the similarity to the original.	Ask learners for the benefits of increasing the sampling resolution and sample rate.	Explain that sound does not have a quality, it is outputting a set sound. The overall sound file, e.g. when recording voices, sounds better because it is more like the original – the amplitudes are more like the original and there are few gaps between.
Learners usually explain that each colour has a code and miss that that code must be unique.	Ask learners how a bitmap image is stored in a computer.	Demonstrate a set of colours with codes – some of which are the same, and ask learners why it is important that the word unique is used.

Starter ideas

1 Crack the code (10 minutes)

Resources: Worksheet 1.3, 'Crack the code'.

Description and purpose: Give learners Worksheet 1.3 and ask them to decipher the message. Ask learners for the answer.

What to do next: Main teaching idea 1, 'Character sets'.

2 Image (15 minutes)

Resources: Print a 10 × 10 grid of squares for each learner and list of colours to use for each square.

Description and purpose: Ask learners to complete the grid by following the colour instructions given.

What to do next: Main teaching idea 3, 'Images'.

Main teaching ideas

1 Character sets (30 minutes)

Learning intention: Understand how and why a computer represents text using a character set.

Resources: ASCII and UNICODE character tables, e.g. www.cambridge.org/links/kctd7006
www.cambridge.org/links/kctd7007

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

Description and purpose: Explain that so far they have only looked at how numbers are represented. Introduce the idea of a character set and give learners a copy of a character set table. Ask learners to write a message using the character codes (denary or binary) – highlight the differences between capital (uppercase) and lowercase letters, and symbols including the space. Ask learners to swap their messages and try and work out what they wrote.

> **Differentiation ideas:**

Support – give learners a limited character set with the denary/binary numbers and letters only (lowercase and uppercase), then slowly introduce more characters such as symbols.

Challenge – ask learners if they can find any patterns in the binary numbers, e.g. the value for B is 1 more than the value for b.

2 Sound (45 minutes)

Learning intention: Understand how a computer represents sound.

Resources: Example sound wave with x-axis labelled for milliseconds and y-axis labelled with amplitudes.

Description and purpose: Show learners the image of an analogue sound wave. Link to physics and how the height is the amplitude, the distance between the waves the frequency, etc. Explain that this is analogue data, i.e. it can be any value – but that computers only use binary. Show learners how the amplitude can be recorded and then stored as a binary number. Give learners the sound wave with amplitude values written on the axis, and ask them to write down the amplitude value in binary every millisecond. Ask learners to compare their answers with another learner and make relevant corrections. Extend this by asking learners to take samples more often, and/or less often and to identify the differences between the values recorded, i.e. more samples = more accurate to the original wave.

> **Differentiation ideas:**

Support – give learners a table to complete with the milliseconds completed, for them to write the denary and then binary values alongside.

Challenge – ask learners to consider what will happen if the interval between samples is changed.

3 Images (30 minutes)

Learning intention: Understand how a computer represents an image, including the effects of the resolution and colour depth.

Resources: Grids to represent bitmap images.

Description and purpose: Ask learners to colour in the grid to make an image – each square must have only one colour in – with a maximum of 4 colours. Ask learners to write the name of each colour they have used and to give it a unique 2-bit binary number. Ask them to write the sequence of bits to represent the image they have drawn, e.g. the 2 bits for the first pixel, then the second, etc. Ask them to give their colour list and binary numbers to another learner to attempt to recreate the image. Link to key terms such as pixel, colour depth, etc. Ask learners questions such as what happens if you increase the number of bits per pixel? What will happen to the image (and the file size)?

> **Differentiation ideas:**

Support – give learners a list of the colours they can use, and the binary values for each colour.

Challenge – ask learners to identify how many colours they could represent if they increased it to 3-bits per pixel, etc.

Plenary ideas

1 Write a message (10 minutes)

Resources: ASCII or UNICODE character set, e.g. www.cambridge.org/links/kctd7008

Description and purpose: Ask learners to write their name in the given character set, differentiating between capitals and lowercase letters.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

2 Spot the difference (10 minutes)

Resources: Two strings that look similar but would not equate to the same, e.g. “my house” and “MyHouse”.

Description and purpose: Tell learners that these two strings will not have the same values, e.g. if they were a password they would not match. Ask learners to identify all of the reasons why the two strings are not the same.

Homework ideas

1 Key images and sound cards

Description: Give learners a list of the key terms about text, images and sound. Ask them to write a definition on the back of each. This will allow learners to consolidate their understanding of terms and produce a future testing resource.

2 Exam-style questions

Description: Give learners a set of exam-style questions about text, images and sound. Review the answers in the next lesson. This will allow learners to experience the style of questions they will be required to answer in the examination.

1.4 Data storage and compression

This section covers Coursebook sub-chapter topics 1.4 and 1.5.

LEARNING PLAN		
Syllabus learning objectives	Learning intentions	Success criteria
Understand how data storage is measured.	Understand that there are different data storage values.	Learners will be able to describe the different data storage values, and put these into numeric order.
Calculate the file size of an image file and a sound file, using information given.	Calculate the file size of an image file and a sound file.	Learners will be able to calculate the file size of an image file and a sound file.
Understand the purpose of and need for data compression.	Describe what is meant by compression and why it is needed.	Learners will be able to describe what is meant by compression and identify its purpose in specific contexts.
Understand how files are compressed using lossy and lossless compression methods.	Describe lossy and lossless compression and recommend one for a scenario.	Learners will be able to describe lossy and lossless compression. Learners will be able to select lossy or lossless compression for a given scenario and then justify their choice.

Common misconceptions

Misconception	How to elicit	How to overcome
Learners think that reducing the sound sampling rate reduces the quality of the sound produced. The quality of the sound is constant, the difference is that the sound is no longer identical to the original, i.e. it has changed.	Ask learners questions such as what happens if the sampling rate is increased or decreased?	Explain that sound is constant and its quality does not change. The change is that the value output is no longer the same as the original sound.

Starter ideas

1 Order the sizes (10 minutes)

Resources: List of different binary values, e.g. 3 GiB, 1 PiB.

Description and purpose: Give learners the list of binary values and ask them to put them in order from smallest to largest. Ask learners for their answers and reorder the values to show the learners the answer.

What to do next: Main teaching idea 2, 'Image file size'.

2 Sound and image terms (10 minutes)

Resources: List of descriptions of sound and image terms.

Description and purpose: Ask learners to identify the terms being described. Ask learners for their answers and review if they are correct and if not then why.

What to do next: Sound file size calculations.

Main teaching ideas

1 File sizes (20 minutes)

Learning intention: Understand how data storage is measured.

Description and purpose: Give learners a list of the acronyms and names for different sizes, e.g. GiB, bit. Ask learners to work in pairs to find out what they stand for and how large each file size is. Ask the pairs which is the smallest and then continue up to the largest, describing each one in turn.

> **Differentiation ideas:**

Support – give learners a document to complete with a table to write the full name and the size.

Challenge – ask learners to calculate how many bits are in each data storage, and if there are any values above exbibytes.

2 Image file size (30 minutes)

Learning intention: Calculate the file size of an image and sound file.

Resources: Descriptions of image files with their resolution and colour depth.

Description and purpose: Recap work on images and the features of an image file; resolution and colour depth. Ask learners how these values can be used to calculate the file size. Show learners, using an example, how to calculate the file size of an image. Give learners a set of questions that give learners the description of an image file and asks them to calculate the file size and present it in different values, e.g. how many bits, how many bytes, how many kibibytes?

> **Differentiation ideas:**

Support – structure the first few questions for learners, for example, having a formula where they fill in the gaps and then calculate.

Challenge – ask learners to find out what other data an image file might store other than the data for the pixels.

> **Assessment ideas:** Learners can mark their own answers and add corrections.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: TEACHER'S RESOURCE

3 RLE (30 minutes)

Learning intention: Understand how lossless compression can compress a file.

Resources: Grid for an image.

Description and purpose: Recap image work, e.g. pixels and colour depth. Ask learners to draw a bitmap image using the grid. Show learners how the image can be compressed using RLE by storing the colour and then the number of times it occurs in sequence for the whole image. Ask learners to give their compressed file data to another learner who has to decompress the data by redrawing the image.

> Differentiation ideas:

Support – give the learner a structured document to help them encode their data, e.g. asking them to write the name of the first colour, and how many times it then occurs in a sequence.

Challenge – ask learners to consider how RLE can be used with other types of file, e.g. text or sound files.

Plenary ideas

1 File size match (10 minutes)

Resources: Descriptions of different files and their contents, and a list of file sizes.

Description and purpose: Ask learners to match the files with their file sizes. Go through each file and ask learners which file size it matches to, explain any differences.

2 Compression choice (10 minutes)

Resources: List of scenarios that need compression, e.g. an image file that is being sent by email to a friend.

Description and purpose: Ask learners to work in pairs to decide whether lossy or lossless compression should be used for each scenario, and to justify their answers. Ask the pairs for their answers and question why they have made that decision.

Homework ideas

1 Image vs text

Description: Give learners the description of an image and the thousand words statement, 'An image is worth a thousand words' – ask learners to use their understanding of files and file sizes to determine whether the statement is true or false. Learners should calculate the file size of the image, and the text file and then decide which is largest. This will allow learners to apply their understanding to another scenario, and then evaluate the result.

2 Exam questions

Description: Give learners a set of exam questions that require them to calculate file sizes, and/or that are about compression. Ask learners to answer the questions, then review these in the next lesson where learners can mark their own answers and make corrections where needed. This will allow learners to experience the style of questions they will need to answer.

Name _____ Date _____

1.1 Hexadecimals

Part 1 – binary to hexadecimal

Getting started

1 0110 _____

2 0000 _____

3 1100 _____

4 1010 _____

Practice

5 1011 0111 _____

6 1111 1101 _____

7 0010 0001 0000 _____

8 1011 1111 1010 _____

Challenge

9 1111000010101111 _____

10 1000011101100001 _____

Part 2 – hexadecimal to binary

Getting started

1 5 _____

2 A _____

3 E _____

Practice

4 21 _____

5 10 _____

6 AB _____

7 D5 _____

Challenge

8 1EF _____

9 20B _____

10 ABCD _____

Name _____ Date _____

1.2 Two's complement

Part 1

Convert the two's complement numbers to denary, and the denary numbers to two's complement.

	Positive two's complement	Denary
Getting started	0111	
		6
	01000	
		9
Practice	01110001	
		38
	01000001	
		89
Challenge	01010111	
		100
	00111111	
		127

We are working with Cambridge Assessment International Education towards endorsement of these titles.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: WORKSHEET 1.2

Part 2

Convert the negative two's complement binary to denary.

	Negative two's complement	Working	Denary
Getting started	1001		
	1010		
	10011		
	101011		
Practice	11100101		
	11001011		
	10001111		
	10101010		
Challenge	11010101		
	10011110		
	10000001		
	10000000		

We are working with Cambridge Assessment International Education towards endorsement of these titles.

> CAMBRIDGE IGCSE™ & O LEVEL COMPUTER SCIENCE: WORKSHEET 1.2

Part 3

Convert the denary to negative two's complement binary.

	Denary	Working	Negative two's complement
Getting started	-2		
	-10		
	-24		
	-35		
Practice	-58		
	-61		
	-75		
	-89		
Challenge	-100		
	-116		
	-124		
	-130		

Name _____ Date _____

1.3 Crack the code

Getting started

Use the code in the table below to work out the message below.

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

k	l	m	n	o	p	q	r	s	t
11	12	13	14	15	16	17	18	19	20

u	v	w	x	y	z	
21	22	23	24	25	26	27

8 5 12 12 15 27 3 15 13 16 21 20 9 14 7 27 9 19 27 6 21 14

Practice

Finished already? Convert each number in the message above into 8 bit binary.

Challenge

Answer the following questions:

- 1 If each character can have only 4 bits, how many different characters can you have?
- 2 If each character can have 8 bits, how many different characters can you have?
- 3 Create a set of codes that use 8 bits for capital letters and numbers. You cannot use the same code twice, and you cannot use any of the numbers used in the code at the start of this worksheet.

We are working with Cambridge Assessment International Education towards endorsement of these titles.

DRAFT



CAMBRIDGE
UNIVERSITY PRESS

Computer Science

for Cambridge IGCSE™ & O Level

PROGRAMMING BOOK FOR JAVA

Dave Duddell



Digital Access



Cambridge Assessment
International Education

Endorsed for programming

We are working with Cambridge Assessment International Education towards endorsement of these titles.

DRAFT

> Contents

The items in **orange** are available in the digital edition that accompanies this book.

Acknowledgements	iv	4.5	Input, output and comments in Java code	38
Introduction	vii	4.6	Integrated Development Environments	41
How to use this series	viii	4.7	Some tasks using arithmetic	42
How to use this book	ix	5	Selection	
1 Introduction to Java		5.1	The basic concepts	49
1.1 The origins and reasons for the development of the Java language	2	5.2	Relational expressions	50
1.2 The Java Platform	3	5.3	Logic expressions	51
2 Algorithms		5.4	IF statements	55
2.1 Algorithms	6	5.5	CASE statements	65
2.2 Structured English	7	6	Iteration	
2.3 Program logic constructs	7	6.1	Iteration and loop constructs	72
2.4 Design tools	10	6.2	The For loop	74
3 Variables and arithmetic operators		6.3	The While loop	81
3.1 Using variables in a high-level programming language	17	6.4	The Do while and Repeat until loops	85
3.2 Variable types	17	6.5	Exiting a loop	86
3.3 Declaring variables	18	7	Subroutines	
3.4 Variables, constants and literals	20	7.1	Terminology	90
3.5 Initialising variables	21	7.2	Subroutine use	91
3.6 Assignment statements and arithmetic operators	22	7.3	Java use of subroutines	94
3.7 Use of an identifier table as part of the design process	28	7.4	The scope of variables	99
4 Options when learning programming		7.5	Using pre-defined subroutines	100
4.1 The Java Platform	35	7.6	Random number generation	100
4.2 Input and output	35	7.7	Mathematics functions	103
4.3 Object-oriented programming (OOP)	36	7.8	String-handling methods (functions)	105
4.4 Running a simple program using the Java Platform	36	7.9	Using text files for input and output	110
		7.10	Exception handling	111
		7.11	The Java framework for file handling	111
		8	Arrays	
		8.1	Terminology and concepts	115
		8.2	Declaring, instantiating and initialising	115
		8.3	Iteration using arrays	116
		8.4	Using arrays in standard methods of solution	121
		8.5	Using combinations of arrays	126

9	Designing algorithms		11.1	Using a debugger	158
9.1	The context for the design process	131	11.2	Using a trace table in a dry-run	162
9.2	Top-down design	133			
9.3	Structure diagrams	133			
10	Checking inputs		12	Programming scenario task preparation	
10.1	Validation and verification	140	12.1	Suggestions for tackling the question	167
10.2	Categories of validation	140	12.2	Coding a solution	170
10.3	Programming validation	142	12.3	Final thoughts	171
10.4	Verification	149			
10.5	Check digit	150	13	Examination practice	176
11	Testing			Appendix – Language-specific reference and OOP	187
11.1	Categories of error	154		Glossary	192
11.2	The testing framework	155			
11.3	Debugging	157		Solutions	

> How to use this book

Throughout this book, you will notice lots of different features that will help your learning. These are explained below.

Learning intentions

These set the scene for each chapter, help with navigation through the Java subject material and indicate the important concepts in each topic.

IN THIS CHAPTER YOU WILL:

- understand why Java is so widely used
- learn how Java programs can easily be transferred to different devices
- learn about the Java Platform product.

Key words

Key vocabulary is highlighted in the text when it is first introduced. Definitions are then given in the margin, which explain the meanings of these words and phrases. You will also find definitions of these words in the Glossary at the back of this book.

KEY WORDS

microchip: a solid-state device with installed software.

port: a word used to describe the transfer and installation of software from one system or device to a different system or device.

Skills focus

This feature supports your computational thinking, mathematical and programming skills. They include useful explanations, step-by-step examples and questions for you to try out yourselves.

SKILLS FOCUS 3.1

ASSIGNMENT STATEMENTS CONTAINING ARITHMETIC EXPRESSIONS

This Skills Focus will illustrate what you will need to consider when choosing assignment statements to use in a program.

Let's consider that a program is to be used to calculate a reasonable estimate of the time it will take for a bus to be driven from one bus station to another. The requirements for this program are as follows:

Tips

These contain advice which could relate to best practice, common errors or misunderstandings to be avoided or sometimes to clarifying terminology. For example, a tip to help clear up misunderstandings between pseudocode and Java.

TIP

1 litre is equal to 1000 cubic centimetres.

Quick question

This feature requests a brief pause for you to consider some aspect of the subject material under discussion. The question might request the identification of a fault or limitation in what has been presented and how it could be improved. This might require recall of some related knowledge you should already have.

QUICK QUESTION

Do you know which version of Java you expect to use? If you are unsure, you could visit the web pages provided by Oracle.

Pseudocode

This is used to present a detailed design for an algorithm. The syllabus for Cambridge IGCSE™ and O Level Computer Science has defined a syntax for pseudocode to be used in examination paper questions. This syntax is used in this book.

```
IF candidateMark >= 40
  THEN
    grade ← "Pass"
  ELSE
    grade ← "Fail"
ENDIF
```


Programming tasks

Programming tasks give you the opportunity to develop your programming and problem-solving skills. There are three different types of programming task:

Demo: A problem scenario is introduced. This is followed by suggestions as to how you might have provided a solution if you had been tackling this yourself. You will be presented with a task and a step-by-step solution will be provided to help familiarise you with the techniques required.

DEMO TASK 3.1

As you can see, programmers have two decisions to make when choosing variables for a program. The first is the choice of sensible data types. The second is the choice of sensible names. This can be illustrated by considering the following problem.

Practice: again introduces a problem scenario but this time you are asked to provide a solution. This allows you to develop your skills.

PRACTICE TASK 3.1

Consider a program that is to be used to record information about members of a group. It is required that names and ages will be used when the program runs. What variables can you use for the name and the age?

Try to think of at least two variables each to store values for the name and for the age. When you have made a decision, write declaration statements in Java for the variables you have chosen. You should aim to make the variable names as meaningful as possible.

Challenge: challenge tasks will stretch and challenge you even further.

CHALLENGE TASK 2.3

In Challenge task 2.2, you created a Structured English design for a program to be used by a teacher. The design had some added features to make it more useful. You now need to document that design as a flowchart. You can use the Structured English to label the symbols.

Summary

There is a summary of key points at the end of each chapter.

SUMMARY

Java is the most widely used programming language in the world. Its success is due to the portability of Java programs.
Java provides software to run on devices that are connected to the Internet.
Portability allows you to easily transfer programs to other types of device. It is achieved through the use of Java Byte Code and the Java Virtual Machine.
The product that is provided is called a Java Platform.
Java Standard Edition (JSE) is the version of the Java Platform that is used for developing programs to be run on a PC.
The two major parts of the Java Platform are the Java Development Kit (JDK) and the Java Runtime Environment (JRE).

End-of-chapter questions

Questions at the end of each chapter which you can use to test your knowledge, understanding or skills. Some of these may require the use of knowledge from previous chapters. Answers to these questions can be found in the Solutions chapter, on the digital part of this resource.

END-OF-CHAPTER QUESTIONS

- 1 State what code is used by the Java Virtual Machine.
- 2 Describe the difference uses of JDK and JRE.

› Chapter 2

Algorithms

IN THIS CHAPTER YOU WILL:

- understand the concept of an algorithm
- learn how the three logic constructs (sequence, selection and iteration) can be used in an algorithm
- understand that a preliminary algorithm design can be presented in Structured English
- learn that a detailed algorithm design can be created using pseudocode or a flowchart
- learn about the components used to construct a flowchart
- create a preliminary design using a flowchart.

Introduction

The aim of this book is to support you while you are learning how to write computer programs using Java. This chapter introduces some concepts that you need to consider before you begin to write program code. These concepts apply whichever programming language you use.

First of all, you need to understand that a program can be described as an algorithm – a series of actions to achieve a specific outcome. You then need to be familiar with methods of presenting an algorithm design. The following statements sum up the reasons why the content of this chapter is important:

- Actually writing a computer program should be seen as the last step in a problem solving process.
- Unless the problem is very simple, a solution should be documented as an algorithm design before it is implemented as a program. A documented design has all of the details recorded so that there can be no misunderstandings when coding begins.
- You need to be familiar with methods that are used to document a design.
- In the chapters dealing with different elements of the Java language, you will be provided with designs for problem solutions. These may be presented together with the corresponding Java code or with a request for you to write the code.

2.1 Algorithms

There are different ways that a computer can be used to solve a problem. In this book, you are going to learn about a particular approach. In this approach, a program is designed, then implemented by writing program code using a high-level programming language. This is a type of language that tells the computer what it must do and how it should do it. Java is one of many languages of this type.

A program written in Java will consist of one or more **algorithms**. An algorithm can be defined as a series of actions. These actions must follow a logic that is suitable for solving a problem.

We do not use the term ‘algorithm’ much in daily life. However, there are many examples of activities in our daily routine that follow a series of actions – in other words, an algorithm. Examples include making a cup of tea, following a cooking recipe, repairing a tyre. The following is a suggestion for an algorithm to be used when two players are going to play chess.

- 1 Place the chessboard on a table with a white square bottom right.
- 2 Place the pieces on the board.
- 3 Choose who is to play white.
- 4 White makes the first move.
- 5 Players take turns to make a move.
- 6 Play continues until there is a final result.

There are several points to note here:

- 1 It is not essential to number the actions. However, using the numbers highlights the fact that, in this example, the actions must be carried out one after another.

KEY WORD

algorithm: a series of actions required to achieve a specific outcome.

- 2 There is a lot of detail missing, such as how the chess pieces are arranged on the board and how a final result is achieved. We will discuss this later in the chapter.
- 3 The order in which the actions in the algorithm are performed must follow a logic that is suitable for solving the problem.
- 4 The version shown here does have a slight problem with the logic.

2.2 Structured English

In Section 2.1, a full English sentence was used to define each action in the chess algorithm. This approach could be used as a first step in producing a design for an algorithm for a computer program. However, it would take a long time to write it, and then a long time to read it when the next stage in the design process began.

An alternative is to use **Structured English**. Here is an example for the chess algorithm discussed in Section 2.1:

Place board on table.
Place pieces.
Choose white.
White moves.
Players take turns.
Continue until final result.

Each action is now described using English words, but using as few as possible. There is no defined way of writing Structured English. In Section 2.3, Structured English is used to define the design for some algorithms that illustrate the logic constructs that can be used in an algorithm. When you come to design a program for yourself, you may find it useful to begin with a Structured English representation of the design.

KEY WORD

Structured English: a method of documenting a design using brief English statements.

PRACTICE TASK 2.1

Consider the order of the actions in the chess-playing algorithm design in Section 2.2. Identify the problem with the logic. Suggest an improved version by rearranging the order of the actions.

2.3 Program logic constructs

A **logic construct** controls the order in which algorithm actions take place. There are three options: sequence, selection and iteration.

Sequence

It is possible for an algorithm to consist of a simple **sequence** of actions where each action happens once and there are no alternative routes available. The chess problem in Section 2.2 is an example.

As another example, we can consider the design of an algorithm for a program that a teacher might use. This simple program takes a score achieved by a candidate in an exam and converts it to a percentage mark.

KEY WORDS

logic construct: a means of controlling the order in which algorithm actions are performed.

sequence: a logic construct where individual actions in an algorithm follow one after another in order.

Algorithm 1

The design could be written in Structured English as follows:

Input total mark possible.
Input candidate mark.
Calculate percentage.
Output percentage.

DEMO TASK 2.1

You have been asked to write a program that will calculate the number of tins of paint that would be needed to paint a wall. You are not ready to start programming yet, but you are going to consider a design.

Solution

The first thing you need to do is to think about the problem and decide what data you would need for the calculation. The following would be a sensible list:

- The length of the wall.
- The height of the wall.
- The area that one tin of paint would cover.
- How many coats of paint you would expect to use.

You can see that you will need to calculate the area of the wall before you can calculate the number of tins needed. The following is, therefore, a sensible Structured English design for your program:

Input length.
Input height.
Calculate area.
Input area covered by one tin.
Input number of coats.
Calculate number of tins needed.
Output number of tins needed.

QUICK QUESTION

Would there be any problem if there was a different order of the tasks in the above design?

Selection

As an example to illustrate this construct, we can return to considering a program for a teacher. Suppose the teacher now wants a program that converts the percentage mark to a grade. It is no longer possible for the algorithm to consist of one simple sequence.

Algorithm 2

The following could be a suitable design:

Input total mark possible.

Input candidate mark.

Calculate percentage.

Grade as Distinction if percentage above 80.	} Selection items
Grade as Merit if percentage in range 61–80.	
Grade as Pass if percentage in range 40–60.	
Grade as Fail if percentage less than 40.	

Output grade.

This algorithm illustrates the use of the **selection** construct. You should note that there is a **condition** being tested to control which action happens (the choice of grade depends on the candidate percentage). In this example, the use of the selection construct does not affect the logical flow of the whole program. In other examples, there might be two alternative sections of the algorithm that could be followed. The result from testing the condition chooses which one is followed.

KEY WORDS

selection: a logic construct in which alternative algorithm actions are possible with the choice dependent on testing a condition.

condition: the criteria that are tested which provide either TRUE or FALSE as an answer.

Iteration

Algorithm 3

So far, the program is only providing a grade for one candidate. If the teacher needs a program to output a grade for every candidate, then a simple change can be made to the design, as follows:

Do the following for each candidate: } Iteration

Input total mark possible.

Input candidate mark.

Calculate percentage.

Grade as Distinction if percentage above 80.

Grade as Merit if percentage in range 61–80.

Grade as Pass if percentage in range 40–60.

Grade as Fail if percentage less than 40.

Output grade.

This illustrates the use of **iteration**, which is the last of the three logic constructs. This is where a group of actions is repeated several times.

There are two points that are worth noting at this stage. The first is that there is a lack of detail in these designs. For example, there is no formula included for the calculation of the percentage. The second is that indentation has been used to help the reader to follow the logic. Indentation is where the start of the text on a line of the design is moved away from the left-hand margin. The example in Algorithm 3 shows the use of two levels of indentation.

KEY WORD

iteration: a logic construct in which a sequence of actions in an algorithm is repeated several times.

TIP

Iteration is a rather formal word. Many people will refer to this as 'looping'.

PRACTICE TASKS 2.2–2.3

- 2.2** For all of these designs, it is possible that the teacher would find the program more useful if something else were output as well as the percentage or grade. Identify a sensible additional output, then rewrite one of the designs to include this extra output. You might need an extra input as well!
- 2.3** The Structured English in Algorithm 3 does not have the most sensible logic. Rearrange the statements to improve the logic.

CHALLENGE TASKS 2.1–2.2

- 2.1** A program is needed to work as a calculator. The user is asked to input two separate numbers and an arithmetic operation (+, × or −). The program must then calculate the answer. This answer must be output.
- Create a Structured English design for this program including suitable input and output.
- 2.2** Consider again the design in Algorithm 2. Identify one or more reasons why the teacher might want to add to the algorithm to extend what the algorithm does. Explain your reasons. Provide a Structured English design that includes the extra features.

2.4 Design tools

As we have seen in the previous examples, a programmer will sometimes be writing a program requiring only a few lines of code. At the other extreme, a large organisation may, from time to time, require the development of a much more complex software-based system.

For the complex software-based system, the development team will have a wide variety of design tools available, from which they can choose the most suitable. By contrast, a program with few lines of code can be written without any use of design tools.

If there is going to be a design process, then in the final stage a **detailed design** must be documented. The amount of detail in this detailed design should be sufficient for the programming to begin.

In Section 2.3, three designs are presented in Structured English for an algorithm to be used by a teacher. These designs illustrate the program logic constructs that are used in an algorithm (sequence, selection and iteration). You are not expected to use Structured English for a detailed design. However, it is useful to discuss the level of detail that should be included in these designs and whether there is sufficient detail for programming to begin. There are three aspects to consider:

- 1** Each algorithm requires the calculation of a percentage. Such a simple calculation probably does not need to be further detailed in the designs.
- 2** Algorithms 2 and 3 require a grade to be given depending on the percentage mark. The detail provided for this is essential because each teacher will have their own specific set of rules for deciding on a grade.

TIP

In the context of software development, you will often come across a reference to a system or to a program. It is best to consider these as having the same meaning.

KEY WORD

detailed design: the final product of a design process that contains enough detail for programming to begin.

- 3 Algorithm 3 involves an iteration. A detailed design would need to specify how this should be implemented in the algorithm.

Figure 2.1 shows the stages of design leading to program coding. It shows that there are two techniques available for detailed design: pseudocode and flowcharts.

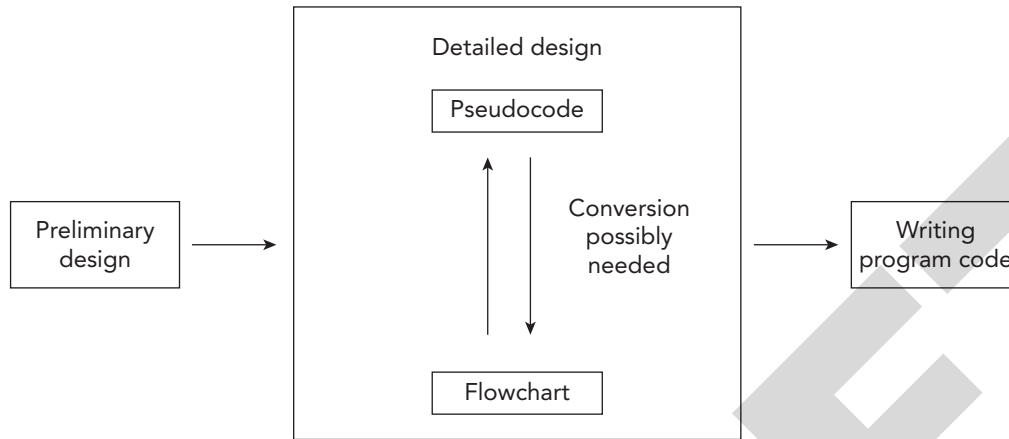


Figure 2.1: Stages of design leading to program coding

This section provides some comments about pseudocode and some details regarding flowcharts.

Pseudocode

Pseudocode is a way of representing the sequence and logic of a program using natural language words, a bit like Structured English but much more detailed. You can use pseudocode to provide a design that has all of the coding components that would be used in an actual programming language. Therefore, a design presented in pseudocode looks like a program code listing. The design can be understood by programmers of any language.

This book will introduce examples of pseudocode in each of the following chapters alongside the corresponding Java code.

KEY WORD

pseudocode: a form of coding used to provide a detailed design which is similar to that used in a programming language.

Flowcharts

A **flowchart** is a graphical representation of a detailed design for an algorithm. It uses a set of symbols, which are illustrated and defined in Table 2.1.

KEY WORD

flowchart: a diagram used to document a detailed design for an algorithm that shows the logical flow of the actions.

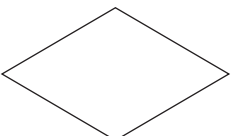
Symbol	Use
	The terminator symbol: there is one at the start and one at the end (referred to as 'stop') of the flowchart sequence.
	An action symbol: it is used to show that the action required is either an input of a value or the output of a value. There must be only one arrow going in and one arrow going out.
	An action symbol: sometimes referred to as the 'process symbol'. It is where one or more actions can be defined. There must be only one arrow going in and one arrow going out.
	An action symbol: referred to as the 'decision symbol'. It defines a condition to be tested. There must be only one arrow going in but there must be two arrows coming out. One of the outgoing arrows must be marked TRUE or T and the other marked FALSE or F.
	The arrow is used to define the sequence or flowline of the actions defined in the flowchart.

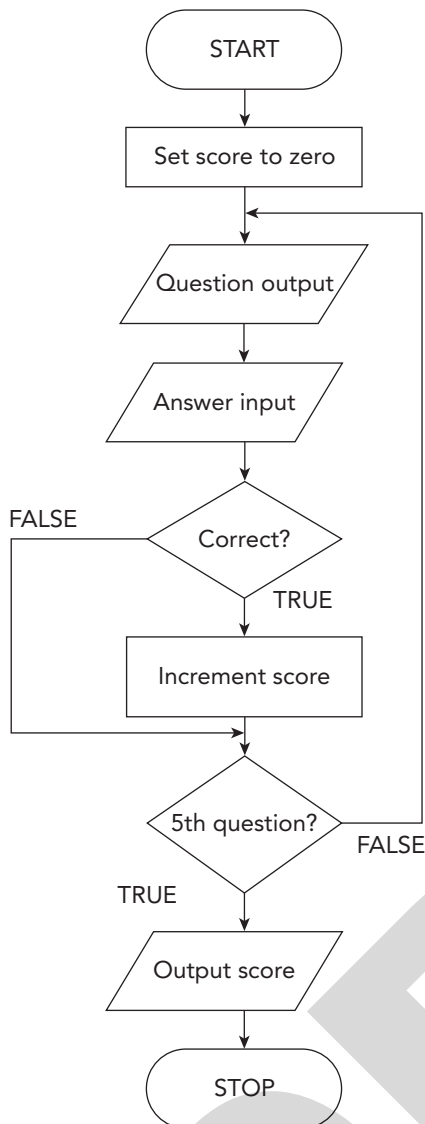
Table 2.1: Flowchart symbols

There are a couple of points to make about the use of flowcharts:

- 1 A flowchart is normally presented as a design going from top to bottom, rather than from left to right, with the start terminator at the top and the stop terminator at the bottom.
- 2 A flowchart is normally used for a detailed design. In this case, each of the three action symbols must contain specific details of the action. Pseudocode is suitable for this.

In the following chapters, when a detailed design is provided for you, this will usually be presented as a flowchart and as pseudocode.

Flowchart 2.1 shows an example of a flowchart constructed as a preliminary design for a program that provides a quiz for a user. Note how iteration is included by the use of an arrow returning to an earlier stage in the algorithm.



Flowchart 2.1: A flowchart for a program that provides a quiz for a user

The overall logic is defined here in this preliminary design. A running score is **initialised** to zero before any questions are asked. This score is increased only if the user provides a correct answer. Only five questions are asked.

The design does not consider how the questions and the answers are stored in the program.

KEY WORD

initialise: an action that provides a value for a variable before the variable is first used in a program.

DEMO TASK 2.2

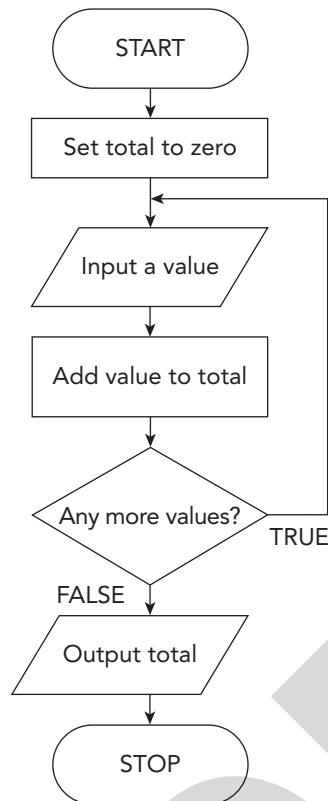
You wish to create a program that adds up a series of values that are input. You decide to create a flowchart as a **preliminary** design. At this stage, you are going to use Structured English to label the flowchart symbols. (In later chapters you will be using pseudocode.)

CONTINUED

Solution

After considering the requirement, you realise that a loop (iteration) will be needed. In each iteration, the newly input number must be added to a running total. For this to work properly, the total must be set to zero before the loop begins.

Flowchart 2.2 shows the flowchart for the preliminary design.



Flowchart 2.2: A preliminary design flowchart for an algorithm to add up values

PRACTICE TASK 2.4

Create a flowchart for a program that could be used by a teacher. You can use the Structured English design for Algorithm 1 provided in Section 2.3. You are not asked to provide pseudocode to define the actions.

CHALLENGE TASK 2.3

In Challenge Task 2.2, you created a Structured English design for a program to be used by a teacher. The design had some added features to make it more useful. You now need to document that design as a flowchart. You can use Structured English from your design to label the symbols.

SUMMARY

An algorithm consists of a series of actions.
An algorithm uses logic constructs to define the order in which the algorithm actions are performed.
The three logic constructs are: sequence, selection and iteration.
Structured English can be used to define an initial design for an algorithm.
Structured English contains brief statements, not full sentences.
The final stage of the design process should be the creation of a detailed design.
There should be sufficient detail in a detailed design to allow program code to be written.
A detailed design for an algorithm can be presented as pseudocode or as a flowchart.
Pseudocode uses code components similar to ones used in a programming language and can be read and understood by programmers of any programming language.
A flowchart is a diagram which shows the order in which actions are performed in the algorithm.

END-OF-CHAPTER QUESTIONS

- 1 Give definitions for the following terms:
 - i an algorithm
 - ii a logic construct
 - iii sequence
 - iv selection
 - v iteration.

- 2 State what the following flowchart symbols are used for:



- 3 Consider the following scenario:

A shopkeeper needs a program to calculate the total bill for items bought by a customer. The program is to work as follows:

- The following will be repeated for each different item bought:
 - 1 The price for one item will be input.
 - 2 The number of that type of item bought will then be input.
 - 3 The price will be calculated for the number bought of this item.
 - 4 This price will be added to a running total.
- The total price to pay will be output.

For example, if you were to buy five pencils and some other items:

- 1 The price of one pencil will be input.
- 2 The input for the number of pencils bought will be 5.
- 3 The price for these five pencils will be calculated.
- 4 That price is added to a running total.

Construct a flowchart for this program. You can use Structured English to label the symbols.