

SAMPLE MATERIAL

We are working with Cambridge Assessment International Education to gain endorsement for this forthcoming title



Endorsed for full syllabus coverage

Cambridge International AS & A Level

Computer Science

David Watson
Helen Williams



We are working with Cambridge Assessment International Education to gain endorsement for these forthcoming titles

Develop computational thinking and ensure full coverage of the revised Cambridge International AS & A Level Computer Science syllabus (9618) with this comprehensive Student's Book written by experienced authors and examiners.

It is supported by a Programming Skills Workbook and a Study and Revision Guide, as well as by Student and Whiteboard eTextbook editions and an Online Teacher's Guide.

Cambridge International AS & A Level Computer Science Student Book	ISBN 9781510457591	March 2019
---	--------------------	------------

Cambridge International AS & A Level Computer Science Programming Skills Workbook	ISBN 9781510457683	June 2019
--	--------------------	-----------

Cambridge International AS & A Level Computer Science Student eTextbook	ISBN 9781510457614	April 2019
--	--------------------	------------

Cambridge International AS & A Level Computer Science Whiteboard eTextbook	ISBN 9781510457621	March 2019
---	--------------------	------------

Cambridge International AS & A Level Computer Science Online Teacher's Guide	ISBN 9781510457652	July 2019
---	--------------------	-----------

The Study and Revision Guide is not going through the Cambridge International endorsement process

Online Teacher's Guide

Deliver more inventive and flexible Cambridge International AS & A Level lessons with a cost-effective range of online resources.

- » Save time planning and ensure syllabus coverage with a scheme of work, lesson plans, teaching activities and worksheets, and expert teaching guidance.
- » Improve students' confidence with quizzes and exam-style questions including sample answers.
- » Consolidate knowledge with answers to all questions in the Student Book.

The Online Teacher's Guide is available via the Dynamic Learning platform. To find out more and sign up for a free, no obligation Dynamic Learning Trial, visit www.hoddereducation.com/dynamiclearning.

We're here to help!

If we can help with questions, and to find out more, please contact us at international.sales@hoddereducation.com.

**Cambridge
International
AS & A Level**

Computer Science

David Watson
Helen Williams

Questions from the Cambridge International AS & A Computer Science papers are reproduced by permission of Cambridge Assessment International Education.

Cambridge Assessment International Education bears no responsibility for the example answers to questions taken from its past question papers which are contained in this publication.

Unless otherwise acknowledged, the questions, example answers and comments that appear in this book were written by the authors.

The publishers would like to thank the following who have given permission to reproduce the following material in this book:

Figure 9.1 Map data © 2018 Google, Imagery © 2018 Landsat/Copernicus

Every effort has been made to trace and acknowledge ownership of copyright. The publishers will be glad to make suitable arrangements with any copyright holders whom it has not been possible to contact.

Although every effort has been made to ensure that website addresses are correct at time of going to press, Hodder Education cannot be held responsible for the content of any website mentioned in this book. It is sometimes possible to find a relocated web page by typing in the address of the home page for a website in the URL window of your browser.

Hachette UK's policy is to use papers that are natural, renewable and recyclable products and made from wood grown in sustainable forests. The logging and manufacturing processes are expected to conform to the environmental regulations of the country of origin.

Orders: please contact Bookpoint Ltd, 130 Park Drive, Milton Park, Abingdon, Oxon OX14 4SE. Telephone: (44) 01235 827827. Fax: (44) 01235 400401. Email education@bookpoint.co.uk Lines are open from 9 a.m. to 5 p.m., Monday to Saturday, with a 24-hour message answering service. You can also order through our website: www.hoddereducation.com

© David Watson and Helen Williams 2019

First published 2019 by
Hodder Education,
An Hachette UK Company
Carmelite House
50 Victoria Embankment
London EC4Y 0DZ

www.hoddereducation.com

Impression number 10 9 8 7 6 5 4 3 2 1

Year 2023 2022 2021 2020 2019

All rights reserved. Apart from any use permitted under UK copyright law, no part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or held within any information storage and retrieval system, without permission in writing from the publisher or under licence from the Copyright Licensing Agency Limited. Further details of such licences (for reprographic reproduction) may be obtained from the Copyright Licensing Agency Limited, www.cla.co.uk

Cover photo © Terrance Emerson - stock.adobe.com

Illustrations by Aptara Inc. and Hodder Education

Typeset by Hodder Education

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

ISBN: 9781510457591

Contents

Introduction

AS LEVEL

1 Information representation

- 1.1 Data representation
- 1.2 Multimedia
- 1.3 File compression

2 Communication

- 2.1 Networks including the internet

3 Hardware

- 3.1 Memory and storage
- 3.2 Logic gates and logic circuits

4 Processor fundamentals

- 4.1 Central processing unit (CPU) architecture
- 4.2 Assembly language
- 4.3 Bit manipulation

5 System software

- 5.1 Operating system
- 5.2 Language translators

6 Security, privacy and data integrity

- 6.1 Security, privacy and data integrity
- 6.2 Data validation and data verification

7 Ethics and ownership

- 7.1 Legal, moral, ethical and cultural implications
- 7.2 Copyright issues
- 7.3 Artificial intelligence (AI)

8 Databases

- 8.1 Database concepts
- 8.2 Database management systems (DBMs)
- 8.3 Data definition language (DDL) and data manipulation language (DML)

9 Algorithm design and problem-solving

- 9.1 Computational thinking skills
- 9.2 Algorithms

10 Data types and structures

- 10.1 Data types and records
- 10.2 Arrays
- 10.3 Files
- 10.4 Introduction to abstract data types (ADT)

11 Programming

- 11.1 Programming basics
- 11.2 Constructs
- 11.3 Built-in-functions
- 11.4 Structured programming

12 Software development

- 12.1 Program development lifecycle
- 12.2 Program design
- 12.3 Program testing and maintenance

A LEVEL

13 Data representation

- 13.1 User-defined data types
- 13.2 File organisation and access
- 13.3 Floating-point numbers, representation and manipulation

14 Communication and internet technologies

- 14.1 Protocols
- 14.2 Circuit switching, packet switching

15 Hardware

- 15.1 Processors and parallel processing
- 15.2 Boolean algebra and logic circuits

16 System software

- 16.1 Purposes of an operating system (OS)
- 16.2 Translation software

17 Security

- 17.1 Encryption, encryption protocols and digital certificates

18 Artificial intelligence

18.1 Artificial intelligence

19 Computational thinking and problem solving

19.1 Algorithms

19.2 Recursion

20 Further programming

20.1 Programming paradigms

20.2 File processing and exception handling

Glossary

Index

9

Algorithm design and problem-solving

In this chapter, you will learn about:

- ★ computational thinking skills (abstraction and decomposition)
- ★ how to write algorithms that provide solutions to problems using structured English flowcharts, and pseudocode.

In order to design a computer system that performs a specific task or solves a given problem, the task or problem has to be rigorously defined and set out, showing *what* is going to be computed and *how* it is going to be computed.

This chapter introduces tools and techniques that can be used to design a software solution to work with associated computer hardware to form a computer system.

Practice is essential to develop skills in computational thinking. Designs shown with pseudocode or flowcharts can be traced to check if the proposed solution works, but the best way to actually test that a computer system works is to code it and use it or, even better, get somebody else to use it. Therefore, practical programming activities, alongside other activities, will be suggested at every stage to help reinforce the skills being learnt and develop the skill of programming.

The programming languages to use are:

- » Java
- » Python
- » VB.NET.

WHAT YOU SHOULD ALREADY KNOW

Can you answer these six questions and complete the following activity?

- 1 What is a procedure?
- 2 What is a function?
- 3 What is an algorithm?
- 4 What is structured English?
- 5 What is a flowchart?
- 6 What is pseudocode?

Write an algorithm using a flowchart to find the average of a number of integers. Both the number of values and each integer are to be input, and the average is to be output.

Use the flowchart of your algorithm to write the algorithm in pseudocode.

Use your pseudocode to write and test a program that includes a function to solve the problem.

9.1 Computational thinking skills

Computational thinking is used to study a problem and formulate an effective solution that can be provided using a computer. There are several techniques used in computational thinking, including abstraction, decomposition, algorithms and pattern recognition.

Key terms

Abstraction – the process of extracting information that is essential, while ignoring what is not relevant, for the provision of a solution.

Decomposition – the process of breaking a complex problem into smaller parts.

Pattern recognition – the identification of parts of a problem that are similar and could use the same solution.

Algorithm – an ordered set of steps to be followed in the completion of a task.

Structured English – a method of showing the logical steps in an algorithm, using an agreed subset of straightforward English words for commands and mathematical operations.

Flowchart – a diagrammatic representation of an algorithm.

Pseudocode – a method of showing the detailed logical steps in an algorithm, using keywords, identifiers with meaningful names, and mathematical operators.

Stepwise refinement – the practice of subdividing each part of a larger problem into a series of smaller parts, and so on, as required.

9.1.1 Using abstraction

Abstraction is an essential part of computational thinking. It enables computer scientists to develop clear models for the solution to complex problems. Abstraction involves extracting information that is essential, while ignoring what is not relevant for the provision of a solution, only including what is necessary to solve that problem.

Many items in common, everyday use are the result of using abstraction; for example, maps, calendars and timetables. Maps use abstraction to show what is required for a specific purpose; for example, a road map should only show the necessary detail required to drive from one place to another.



▲ Figure 9.1 Road map and satellite view

9.1.2 Using decomposition

Decomposition is also an essential part of computational thinking. It enables computer scientists to divide a complex problem into smaller parts that can be further subdivided into even smaller parts until each part is easy to examine and understand, and a solution can be developed.

Pattern recognition is used to identify those parts that are similar and could use the same solution. This leads to the development of reusable program code in the form of subroutines, procedures and functions.

9.2 Algorithms

9.2.1 Writing algorithms that provide solutions to problems

There are several methods of writing algorithms before attempting to program a solution. Here are three frequently used methods.

- » **Structured English** is a method of showing the logical steps in an algorithm, using an agreed subset of straightforward English words for commands and mathematical operations to represent the solution. These steps can be numbered.
- » A **flowchart** shows diagrammatically, using a set of symbols linked together with flowlines, the steps required for a task and the order in which they are to be performed. These steps, together with the order, are called an **algorithm**. Flowcharts are an effective way to show the structure of an algorithm.
- » **Pseudocode** is a method of showing the detailed logical steps in an algorithm, using keywords, identifiers with meaningful names, and mathematical operators to represent a solution. Pseudocode does not need to follow the syntax of a specific programming language, but it should provide sufficient detail to allow a program to be written in a high-level language.

ACTIVITY

You have been asked to write an algorithm for drawing regular polygons of any size.

In pairs, divide the problem into smaller parts, identifying those parts that are similar.

Write down your solution as an algorithm in structured English.

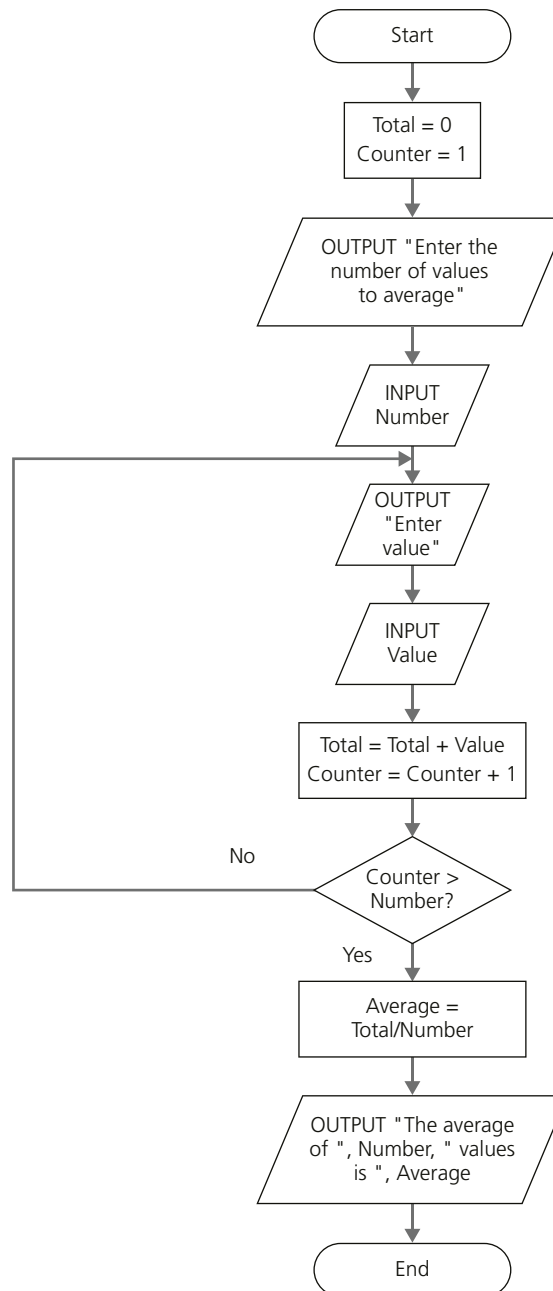
Swap your algorithm with another pair.

Test their algorithm by following their instructions to draw a regular polygon. Discuss any similarities and differences between your solutions.

Below, you will see the algorithm from the activity written using each of these three methods.

Structured English

- 1 Ask for the number of values
- 2 Loop that number of times
- 3 Enter a value in loop
- 4 Add the value to the Total in loop
- 5 Calculate and output average

Flowchart

▲ Figure 9.2

Pseudocode

```

Total ← 0
PRINT "Enter the number of values to average"
INPUT Number
FOR Counter ← 1 TO Number
    PRINT "Enter value"
    INPUT Value
    Total ← Total + Value
NEXT Counter
Average ← Total / Number
PRINT "The average of ", Number, " values is ", Average

```

9.2.2 Writing simple algorithms using pseudocode

Each line of pseudocode is usually a single step in an algorithm. The pseudocode used in this book follows the rules in the *Cambridge International AS & A Level Computer Science Pseudocode Guide for Teachers* and is set out using a fixed width font and indentation, where required, of four spaces, except for THEN, ELSE and CASE clauses that are only indented by two spaces.

All identifier names used in pseudocode should be meaningful; for example, the name of a person could be stored in the variable identified by Name. They should also follow some basic rules: they should only contain the characters A–Z, a–z and 0–9, and should start with a letter. Pseudocode identifiers are usually considered to be case insensitive, unlike identifiers used in a programming language.

It is good practice to keep track of any identifiers used in an identifier table, such as Table 9.1.

Identifier name	Description
StudentName	Store a student name
Counter	Store a loop counter
StudentMark	Store a student mark

▲ **Table 9.1**

Pseudocode statements to use for writing algorithms.

To input a value:

```
INPUT StudentName
```

To output a message or a value or a combination:

```

OUTPUT "You have made an error"
OUTPUT StudentName
OUTPUT "Student name is ", StudentName

```

To assign a value to a variable (the value can be the result of a process or a calculation):

```
Counter ← 1
Counter ← Counter + 1
MyChar ← "A"
LetterValue ← ASC(MyChar)
StudentMark ← 40
Percentage ← (StudentMark / 80) * 100
Oldstring ← "Your mark is"
NewString ← OldString & " ninety-seven"
```

Operators used in pseudocode assignment statements:

ACTIVITY

Identify the values stored in the variables when the assignment statements in the example above have all been completed. The function `ASC` returns the ASCII value of a character.

```
+ Addition
- Subtraction
* Multiplication
/ Division
& String concatenation
← Assignment
```

To perform a selection using `IF` statements for a single choice or a choice and an alternative, and `CASE` statements when there are multiple choices or multiple choices and an alternative:

IF – single choice

```
IF MyValue > YourValue
  THEN
    OUTPUT "I win"
ENDIF
```

IF – single choice with alternative

```
IF MyValue > YourValue
  THEN
    OUTPUT "I win"
  ELSE
    OUTPUT "You win"
ENDIF
```

CASE – multiple choices

```
CASE OF Direction
  "N": Y ← Y + 1
  "S": Y ← Y - 1
  "E": X ← X + 1
  "W": X ← X - 1
ENDCASE
```

CASE – multiple choices with alternative

```
CASE OF Direction
  "N": Y ← Y + 1
  "S": Y ← Y - 1
  "E": X ← X + 1
  "W": X ← X - 1
  OTHERWISE : OUTPUT "Error"
ENDCASE
```

Relational operators used in pseudocode selection statements:

=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Programming languages may not always have the same selection constructs as pseudocode, so it is important to be able to write a program that performs the same task as a solution given in pseudocode.

ACTIVITIES

- In the programming language you have chosen to use, write a short program to input `MyValue` and `YourValue` and complete the single choice with an alternative `IF` statement shown on page 12. Note any differences in the command words you need to use and the construction of your programming statements compared with the pseudocode.
- In the programming language you have chosen to use, write a short program to set `X` and `Y` to zero, input `Direction` and complete the multiple choice with an alternative `CASE` statement shown on page 12 and output `X` and `Y`. Note any differences in the command words you need to use and the construction of your programming statements compared to the pseudocode.

To perform iteration using `FOR`, `REPEAT-UNTIL` and `WHILE` loops:

```
Total ← 0
FOR Counter ← 1 TO 10
    OUTPUT "Enter a number "
    INPUT Number
    Total ← Total + Number
NEXT Counter
OUTPUT "The total is ", Total
```

```
FOR Counter ← 1 TO 10 STEP 2
    OUTPUT Counter
NEXT Counter
```

A `FOR` loop has a fixed number of repeats, the `STEP` increment is an optional expression that must be a whole number.

```
REPEAT
    OUTPUT "Please enter a positive number "
    INPUT Number
UNTIL Number > 0
```

Statements in a `REPEAT` loop are always executed at least once.

ACTIVITY

In the programming language you have chosen to use, write a short program to perform the same tasks as the four loops shown. Note any differences in the command words you need to use and the construction of your programming statements compared to the pseudocode.

```
Number ← 0
WHILE Number >= 0 DO
    OUTPUT "Please enter a negative number "
    INPUT Number
ENDWHILE
```

Statements in a WHILE loop may sometimes not be executed.

Programming languages may not always use the same iteration constructs as pseudocode, so it is important to be able to write a program that performs the same task as a solution given in pseudocode.

WHILE and REPEAT loops and IF statements make use of comparisons to decide whether statements within a loop are repeated or a statement or group of statements are executed. The comparisons make use of relational operators and the logic operators AND, OR and NOT. The outcome of these comparisons is always either true or false.

```
REPEAT
    OUTPUT "Please enter a positive number less than fifty"
    INPUT Number
UNTIL (Number > 0) AND (Number < 50)
```

ACTIVITY

In pseudocode, write statements to check that a number input is between 10 and 20 or over 100. Make use of brackets to ensure that the order of the comparisons is clear.

A simple algorithm can be clearly documented using these statements. A more realistic algorithm to find the average of a number of integers input would include checks that all the values input are whole numbers and that the number input to determine how many integers are input is also positive.

This can be written in pseudocode by making use of the function INT(x) that returns the integer part of x:

```
Total ← 0
REPEAT
    PRINT "Enter the number of values to average"
    INPUT Number
UNTIL (Number > 0) AND (Number = INT(Number))
FOR Counter ← 1 TO Number
    REPEAT
        PRINT "Enter an integer value "
        INPUT Value
    UNTIL Value = INT(Value)
    Total ← Total + Value
NEXT Counter
Average ← Total / Number
PRINT "The average of ", Number, " values is ", Average
```

The identifier table for this algorithm is presented in Table 9.2.

Identifier name	Description
Total	Running total of integer values entered
Number	Number of integer values to enter
Value	Integer value input
Average	Average of all the integer values entered

▲ Table 9.2

ACTIVITY

In pseudocode, write an algorithm to set a password for a user when they have input the same word twice. Then allow the user three attempts to enter the correct password. Complete an identifier table for your algorithm.

Finally, check your pseudocode algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

9.2.3 Writing pseudocode from a structured English description

There are no set rules for writing structured English – the wording just needs to be unambiguous and easily understandable. Pseudocode is more precise and usually follows an agreed set of rules.

From a structured English description, the following things need to be possible:

- » Any variables that need to be used can be identified and put in an identifier table – these can be items input or output as the results of calculations.
- » Input and output can be identified from the wording used, for example, Enter, Read, Print, Write.
- » Selection can be identified from the wording used, for example, If, Then, Choose.
- » Any iteration required can be identified from the wording used, for example, Loop, Repeat.
- » Any processes needed can be identified from the wording used, for example, Set, Calculate.

When the identifier table is complete, each structured English statement can be used to write one or more pseudocode statements, keeping the same order as the structured English.

Here is an example of an algorithm to calculate a runner's marathon time in seconds, using structured English:

- 1 Enter time taken to run marathon in **hours**, **minutes** and **seconds**
- 2 Calculate and store **marathon time in seconds**
- 3 Output **marathon time in seconds**

This can be used to identify the variables required and complete the identifier table (Table 9.3).

Identifier name	Description
MarathonHours	The hours part of the marathon time
MarathonMinutes	The minutes part of the marathon time
MarathonSeconds	The seconds part of the marathon time
TotalMarathonTimeSeconds	Total marathon time in seconds

▲ Table 9.3

Using these identifiers, each step of the structured English algorithm can be converted to pseudocode, as demonstrated below.

1 Enter time taken to run marathon in hours, minutes and seconds

There are three variables used: MarathonHours, MarathonMinutes and MarathonSeconds. This is explicitly input and implicitly output as the user needs to understand what input is required. The pseudocode required is as follows.

```

OUTPUT "Enter the time you took to run the marathon"
OUTPUT "Enter hours"
INPUT MarathonHours
OUTPUT "Enter minutes"
INPUT MarathonMinutes
OUTPUT "Enter seconds"
INPUT MarathonSeconds

```

2 Calculate and store marathon time in seconds

This is a process using the variables MarathonHours, MarathonMinutes and MarathonSeconds and using an assignment statement to store the result in TotalMarathonTimeSeconds. The pseudocode required is as follows.

```

TotalMarathonTimeSeconds ← (MarathonHours * 60
+ MarathonMinutes) * 60 + MarathonSeconds

```

3 Output marathon time in seconds

This is output using the variable TotalMarathonTimeSeconds. The pseudocode required is as follows.

```

OUTPUT "Time for marathon in seconds ",
TotalMarathonTimeSeconds

```


ACTIVITY


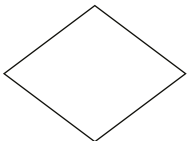


The structured English description has been extended below to check the runner's time against their personal best.

- 1 Enter time taken to run marathon in hours, minutes and seconds
- 2 Calculate and store marathon time in seconds
- 3 Output marathon time in seconds
- 4 Enter personal best time in seconds
- 5 If marathon time in seconds is shorter than the personal best time then
- 6 Reset personal best time in seconds
- 7 Output the personal best time

Extend the identifier table and write the extra pseudocode to complete the algorithm. Then check your algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

9.2.4 Writing pseudocode from a flowchart

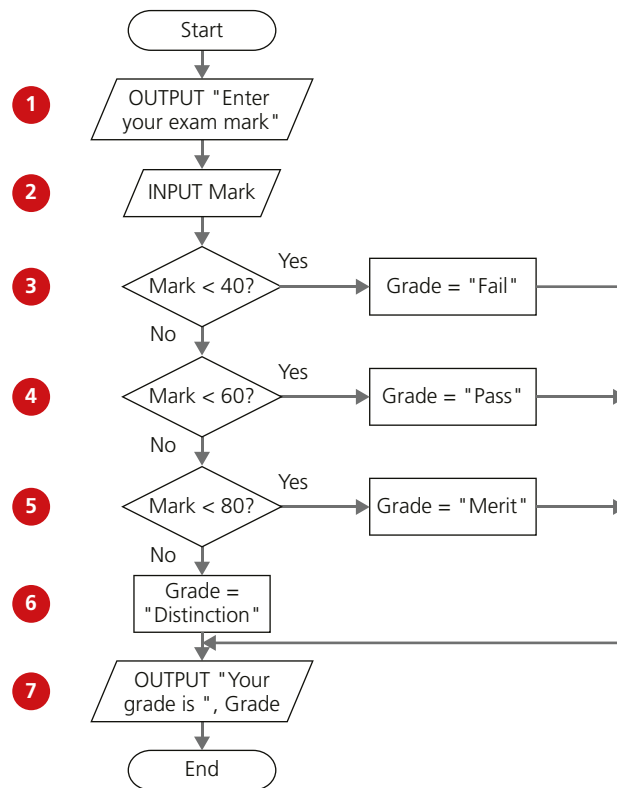
Flowcharts are diagrams showing the structure of an algorithm using an agreed set of symbols, as shown in Table 9.4.

Pseudocode	Flowchart symbol
INPUT or OUTPUT	
IF or CASE Part of FOR, REPEAT and WHILE	
FOR, REPEAT and WHILE	 Returning flowline
Assignment ← using a calculation or a pre-defined process e.g. INT	

▲ **Table 9.4**

Flowcharts can be used to identify any variables required and then complete an identifier table. Each flowchart symbol can be used to identify and write one or more pseudocode statements.

Here is an example of a flowchart of an algorithm that can be used to check an exam grade:



```

1 OUTPUT "Enter your exam mark"
2 INPUT Mark
3 IF Mark < 40
  THEN
    Grade ← "Fail"
  ELSE
4   IF Mark < 60
    THEN
      Grade ← "Pass"
    ELSE
5     IF Mark < 80
      THEN
        Grade ← "Merit"
      ELSE
6        Grade ← "Distinction"
      ENDIF
    ENDIF
  ENDIF
7 OUTPUT "Your grade is ", Grade
  
```

▲ **Figure 9.3**

The same algorithm is presented in pseudocode on the left. Below is the identifier table:

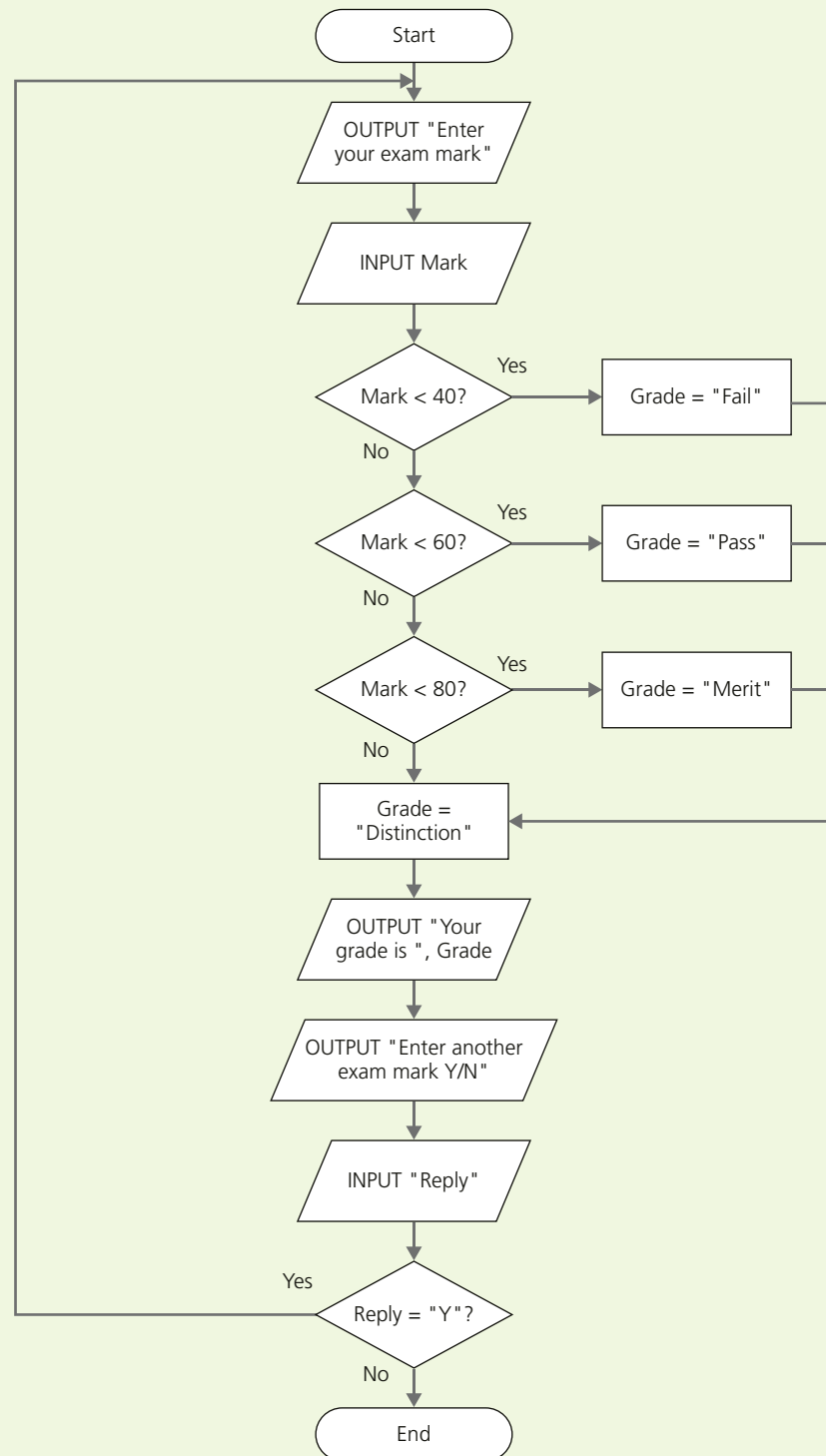
Identifier name	Description
Mark	Exam mark
Grade	Exam grade

▲ **Table 9.5**

3, 4, 5 and 6 form a nested selection (IF) structure, as each following statement is part of the ELSE clause. It is only at 7 that the selection is complete. The flowchart shows this clearly; the pseudocode uses indentation to show the nesting.

ACTIVITY

The flowchart has been extended to allow more than one mark to be input.



▲ **Figure 9.4**

Extend the identifier table and write the extra pseudocode to complete the algorithm. Then check your algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

9.2.5 Stepwise refinement

The algorithms looked at so far have been short and simple. When an algorithm is written to solve a more complex problem, decomposition is used to break the problem down into smaller and more manageable parts. These parts then need to be written as a series of steps where each step can be written as a statement in a high-level programming language, this process is called **stepwise refinement**.

Many problems are more complex than they seem if a robust solution is to be developed. Look at the first step of the structured English to calculate a time in seconds.

- 1 **Enter time taken to run marathon in hours, minutes and seconds**
- 2 Calculate and store marathon time in seconds
- 3 Output marathon time in seconds

The first step can be further broken down, as follows:

- 1.1 Enter the hours
- 1.2 Enter the minutes
- 1.3 Enter the seconds

Each of these steps can be broken down further:

- 1.1.1 Input value for hours
- 1.1.2 Check input in the range 2 to 8
- 1.1.3 Reject if out of range or not a whole number and re-input value step 1.1.1
- 1.1.4 Accept and store value in hours
- 1.2.1 Input value for minutes
- 1.2.2 Check input in the range 0 to 59
- 1.2.3 Reject if out of range or not a whole number and re-input value step 1.2.1
- 1.2.4 Accept and store value in minutes
- 1.3.1 Input value for seconds
- 1.3.2 Check input in the range 0 to 59
- 1.3.3 Reject if out of range or not a whole number and re-input value step 1.3.1
- 1.3.4 Accept and store value in seconds

These steps can now be written in pseudocode. For example, the input routine for the seconds:

```
REPEAT
    OUTPUT "Enter seconds"
    INPUT Value
UNTIL (Value >= 0) AND (Value <= 59) AND (Value = INT(Value))
MarathonSeconds ← Value
```

ACTIVITY

Look at the algorithm to calculate the area of a chosen shape written in structured English below. Use stepwise refinement to break each step into more manageable parts then rewrite the algorithm using pseudocode.

- 1 Choose the shape (square, triangle, circle)
- 2 Enter the length(s)
- 3 Calculate the area
- 4 Output the area

Then check your pseudocode algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

End of chapter questions

- 1 Algorithms can be shown as structured English, flowcharts and pseudocode. Explain what is meant by:
 - a) structured English [2]
 - b) a flowchart [2]
 - c) pseudocode. [2]
- 2 Several techniques are used in computational thinking. Explain what is meant by:
 - a) abstraction [2]
 - b) decomposition [2]
 - c) pattern recognition. [2]
- 3 Describe, using an example, the process of stepwise refinement. [2]
- 4 Computer programs have to evaluate expressions.
 - Study the sequence of pseudocode statements.
 - Write down the value assigned to each variable.

```

DECLARE h, w, r, Perimeter, Area : REAL
DECLARE A, B, C, D, E           : BOOLEAN

h ← 13.6 w ← 6.4
Perimeter ← (h + w) * 2

r ← 10
Area 3.142 * r^2

Z ← 11 + r / 5 + 3
A ← NOT(r > 10)

```

- a) A [1]
- b) Perimeter [1]
- c) Area [1]
- d) Z [1]

- 5 Study the pseudocode and answer the following questions. Line numbers have been added to help you.

```

01 REPEAT
02 OUTPUT "Menu Temperature Conversion"
03 OUTPUT "Celsius to Fahrenheit          1"
04 OUTPUT "Fahrenheit to Celsius         2"
05 OUTPUT "Exit                          3"
06 OUTPUT "Enter choice"
07 IF Choice = 1 OR Choice = 2
08   THEN
09     OUTPUT "Enter temperature"
10     INPUT Temperature
11     IF Choice = 1
12       THEN
13         ConvertedTemperature ← 1.8*Temperature + 32
14       ELSE
15         ConvertedTemperature ← (Temperature - 32) * 5 / 9
16     ENDIF
17     OUTPUT "Converted temperature is ", ConvertedTemperature
18   ELSE
19     IF Choice <> 3
20       THEN
21         OUTPUT "Error in choice"
22       ENDIF
23   ENDIF
24 UNTIL Choice = 3

```

- a) Give the line number of:
- i) an assignment statement [1]
 - ii) a selection [1]
 - iii) an iteration. [1]
- b) Complete an identifier table for the algorithm. [3]
- c) Extend the algorithm to only allow four tries for a correct choice. [3]

- 6 A driver buys a new car. The value of the car reduces each year by a percentage of its current value. The percentage reduction is:
- in the first year, 40%
 - in each following year, 20%.

The driver writes a program to predict the value of the car in future years. The program requirements are:

- Enter the cost of the new car (to nearest \$).
 - Calculate and output the value of the car at the end of each year.
 - The program will end when either the car is nine years old, or when the value is less than \$1000.
- a) Study the incomplete pseudocode which follows in part b) and fill in the identifier table. [3]

Identifier	Data type	Description

- b) Complete the pseudocode for this design. [6]

```

OUTPUT "Enter purchase price"
INPUT PurchasePrice
CurrentValue ← .....
YearCount ← 1
WHILE ..... AND .....
    IF .....
    THEN
        CurrentValue ← CurrentValue * (1 - 40 / 100)
    ELSE
        CurrentValue ← .....
    ENDIF
    OUTPUT YearCount, CurrentValue
    .....
ENDWHILE

```

In this chapter you will learn about:

- ★ the differences between RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) processors
- ★ the importance and use of pipelining and registers in RISC processors
- ★ SISD, SIMD, MISD and MIMD basic computer architectures
- ★ the characteristics of massively parallel computers
- ★ interrupt handling on CISC and RISC computers
- ★ Boolean algebra including De Morgan's Laws
- ★ the simplification of logic circuits and expressions using Boolean algebra
- ★ producing truth tables from common logic circuits
- ★ half adder and full adder logic circuits
- ★ the construction and role of SR and JK flip-flop circuits
- ★ using Karnaugh maps in the solution of logic problems.

15.1 Processors and parallel processing

WHAT YOU SHOULD ALREADY KNOW

In Chapter 4, you learnt about processor fundamentals. Try the following three questions to refresh your memory before you start to read the first part of this chapter:

- 1 A computer uses the following status registers when carrying out the addition of two binary numbers:
 - a carry flag (C)
 - an overflow flag (V)
 - a negative flag (N).

Describe what happens to the above status registers when the following pairs of 8-bit binary numbers are added together and explain the significance of the flag values in both sets of calculation:

 - a) 0 0 1 1 1 1 0 0 and 0 1 0 0 0 1 1 0
 - b) 1 1 0 0 0 1 0 0 and 1 0 1 1 1 0 1 0
- 2 Describe the stages in the fetch-execute cycle.
- 3 a) A processor contains three buses: data bus, address bus and control bus.
 - i) What factors determine the width of a bus?
 - ii) Which of the three buses will have the smallest width?
 - iii) An address bus is increased from 16-bit to 64-bit. What would be the result of this upgrade to the processor?
- b) Explain the role of
 - i) the clock
 - ii) interrupts in a typical processor.

Key terms

CISC – complex instruction set computer.

RISC – reduced instruction set computer.

Pipelining – allows several instructions to be processed simultaneously without having to wait for previous instructions to finish.

Parallel processing – operation which allows a process to be split up and for each part to be executed by a different processor at the same time.

SISD – Single Instruction Single Data, computer architecture which uses a single processor and one data source.

SIMD – Single Instruction Multiple Data, computer architecture which uses many processors and different data inputs.

MISD – Multiple Instruction Single Data, computer architecture which uses many processors but the same shared data source.

MIMD – Multiple Instruction Multiple Data, computer architecture which uses many processors, each of which can use a separate data source.

Cluster – a number of computers (containing SIMD processors) networked together.

Super computer – a powerful mainframe computer.

Massively parallel computers – the linking together of several computers effectively forming one machine with thousands of processors.

15.1.1 RISC and CISC processors

Early computers made use of the Von Neumann architecture (see Chapter 4). Modern advances in computer technology have led to much more complex processor design. Two basic philosophies have emerged over the last few years:

- » developers who want the emphasis to be on the hardware used; the hardware should be chosen to suit the high-level language development
- » developers who want the emphasis to be on the software/instruction sets to be used; this philosophy is driven by ever faster execution times.

The first philosophy is part of a group of processor architectures known as **CISC (Complex Instruction Set Computer)**. The second philosophy is part of a group of processor architectures known as **RISC (Reduced Instruction Set Computer)**.

CISC processors

CISC processor architecture makes use of more internal instruction formats than RISC. The design philosophy is to carry out a given task with as few lines of assembly code as possible. Processor hardware must therefore be capable of handling more complex assembly code instructions. Essentially, CISC architecture is based on single complex instructions which need to be converted by the processor into a number of sub-instructions to carry out the required operation.

For example, suppose we wish to add the two numbers A and B together, we could write the following assembly instruction:

ADD A, B – this is a single instruction that requires several sub-instructions (multi-cycle) to carry out the *ADDition* operation

This methodology leads to shorter coding (than RISC) but may actually lead to more work being carried out by the processor.

RISC processors

RISC processors have fewer built-in instruction formats than CISC. This can lead to higher processor performance. The RISC design philosophy is built on the

use of less complex instructions, which is done by breaking up the assembly code instructions into a number of simpler single-cycle instructions. Ultimately, this means there is a smaller, but more optimised set of instructions than CISC. Using the same example as above to carry out the addition of two numbers A and B (this is the equivalent operation to ADD A, B):

LOAD X, A – this loads the value of A into a register X
 LOAD Y, B – this loads the value of B into a register Y
 ADD A, B – this takes the values for A and B from X and Y and adds them
 STORE Z – the result of the addition is stored in register Z

Each instruction requires one clock cycle (see Chapter 4). Separating commands such as LOAD and STORE reduces the amount of work done by the processor. This leads to faster processor performance since there are ultimately a smaller number of instructions than CISC. It is worth noting here that the optimisation of each of these simpler instructions is done through the use of pipelining (see the next section).

Table 15.1 shows the main differences between CISC and RISC processors.

CISC features	RISC features
Many instruction formats are possible	Uses fewer instruction formats/sets
There are more addressing modes	Uses fewer addressing modes
Makes use of multi-cycle instructions	Makes use of single-cycle instructions
Instructions can be of a variable length	Instructions are of a fixed length
Longer execution time for instructions	Faster execution time for instructions
Decoding of instructions is more complex	Makes use of general multi-purpose registers
It is more difficult to make pipelining work	Easier to make pipelining function correctly
The design emphasis is on the hardware	The design emphasis is on the software
Uses the memory unit to allow complex instructions to be carried out	Processor chips require fewer transistors

▲ Table 15.1

EXTENSION ACTIVITY

Find out how some of the newer technologies, such as EPIC (Explicitly Parallel Instruction Computing) and VLIW (Very Long Instruction Word) processor architectures, are used in computer systems.

Pipelining

One of the major developments resulting from RISC architecture is **pipelining**. This is one of the less complex ways of improving computer performance. Pipelining allows several instructions to be processed simultaneously without having to wait for previous instructions to be completed. To understand how this works, we need to split up the execution of a given instruction into its five stages:

- 1 instruction fetch cycle (IF)
- 2 instruction decode cycle (ID)
- 3 operand fetch cycle (OF)

- 4 instruction execution cycle (IE)
- 5 writeback result process (WB).

To demonstrate how pipelining works, we will consider a program which has six instructions (A, B, C, D, E and F). Figure 15.1 shows the relationship between processor stages and the number of required clock cycles when using pipelining. It shows how pipelining would be implemented with each stage requiring one clock cycle to complete.

		Clock cycles									
		1	2	3	4	5	6	7	8	9	10
Processor stages	IF	A	B	C	D	E	F				
	ID		A	B	C	D	E	F			
	OF			A	B	C	D	E	F		
	IE				A	B	C	D	E	F	
	WB					A	B	C	D	E	F

▲ Figure 15.1

This functionality clearly requires processors with several registers to store each of the stages.

Execution of an instruction is split into a number of stages; as each stage completes, the first stage of the first instruction can now be executed. Then the second instruction can start execution before the first one has completed, and so on, until all six instructions are processed.

In this example, by the time instruction 'A' has completed, instruction 'F' is at the first stage and instructions 'B' to 'E' are at various in-between stages in the process. As Figure 15.1 shows, a number of instructions can be processed at the same time, and there is no need to wait for an instruction to go through all five cycles before the next one can be implemented. In the example shown, the six instructions require 10 clock cycles to go to completion. Without pipelining, it would require 30 (6×5) cycles to complete (since each of the six instructions requires five stages for completion).

Interrupts

In Chapter 4, we discussed interrupt handling in processors where each instruction is handled sequentially before the next one can start (five stages for instruction 'A', then five stages for instruction 'B', and so on).

Once the processor detects the existence of an interrupt (at the end of the fetch-execute cycle), the current program would be temporarily stopped (depending on interrupt priorities), and the status of each register stored. The processor can then be restored to its original status before the interrupt was received and serviced.

However, with pipelining, there is an added complexity; as the interrupt is received, there could be a number of instructions still in the pipeline. The usual way to deal with this is to discard all instructions in the pipeline except for the last instruction in the write-back (WB) stage.

The interrupt handler routine can then be applied to this remaining instruction and, once serviced, the processor can restart with the next instruction in the

sequence. Alternatively, although much less common, the contents of the five stages can be stored in registers. This allows all current data to be stored, allowing the processor to be restored to its previous status once the interrupt has been serviced.

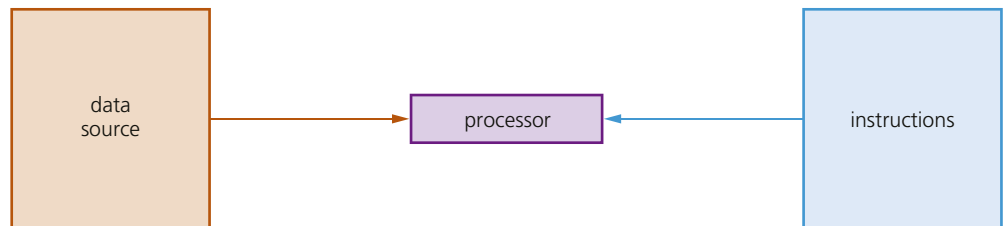
15.1.2 Parallel processing

Parallel processor systems

There are many ways that parallel processing can be carried out. The four categories of basic computer architecture presently used are described below.

SISD (Single Instruction Single Data)

SISD (single instruction single data) uses a single processor that can handle a single instruction and which also uses one data source at a time. Each task is processed in a sequential order. Since there is a single processor, this architecture does not allow for parallel processing. It is most commonly found in applications such as early personal computers.



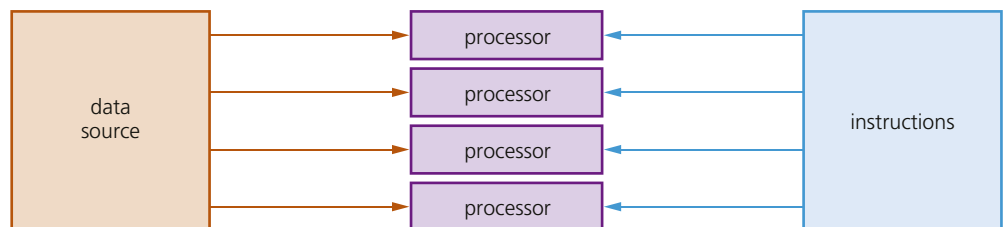
▲ Figure 15.2 SISD diagram

SIMD (Single Instruction Multiple Data)

SIMD (single instruction multiple data) uses many processors. Each processor executes the same instruction but uses different data inputs – they are all doing the same calculations but on different data at the same time.

SIMD are often referred to as array processors; they have a particular application in graphics cards. For example, suppose the brightness of an image made up of 4000 pixels needs to be increased. Since SIMD can work on many data items at the same time, 4000 small processors (one per pixel) can each alter the brightness of each pixel by the same amount at the same time. This means the whole of the image will have its brightness increased consistently.

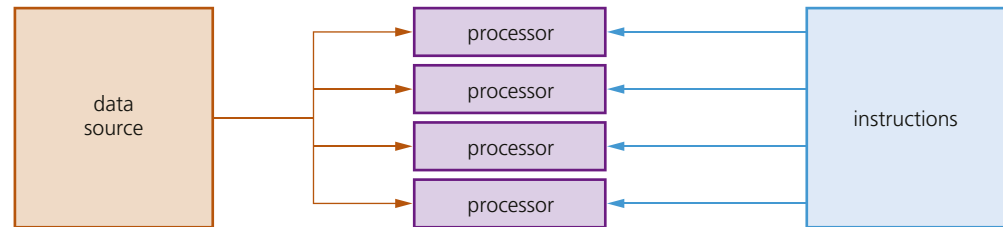
Other applications include sound sampling – or any application where a large number of items need to be altered by the same amount (since each processor is doing the same calculation on each data item).



▲ Figure 15.3 SIMD diagram

MISD (Multiple Instruction Single Data)

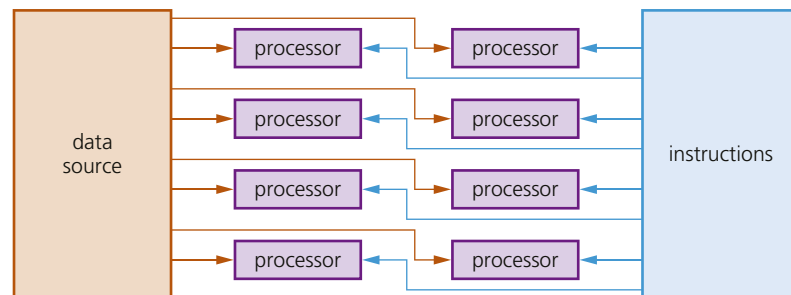
MISD (multiple instruction single data) uses several processors. Each processor uses different instructions but uses the same shared data source. MISD is not a commonly used architecture (MIMD tends to be used instead). However, the American Space Shuttle flight control system did make use of MISD processors.



▲ Figure 15.4 MISD diagram

MIMD (Multiple Instruction Multiple Data)

MIMD (multiple instruction multiple data) uses multiple processors. Each one can take its instructions independently, and each processor can use data from a separate data source (the data source may be a single memory unit which has been suitably partitioned). The MIMD architecture is used in multicore systems (for example, by super computers or in the architecture of multi-core chips).



▲ Figure 15.5 MIMD diagram

There are a number of factors to consider when using parallel processing.

When carrying out parallel processing, processors need to be able to communicate. The data which has been processed needs to be transferred from one processor to another.

When software is being designed, or programming languages are being chosen, they must be capable of processing data from multiple processors at the same time.

It is a much faster method for handling large volumes of *independent data*; any data which relies on the result of a previous operation (*dependent data*) would not be suitable in parallel processing. Data used will go through the same processing, which requires this independence from other data.

Parallel processing overcomes the Von Neumann 'bottleneck' (in this type of architecture, data is constantly moving between memory and processor, leading to *latency*; as processor speeds have increased, the amount of time they remain idle has also increased since the processor's performance is limited to the internal data transfer rate along the buses). Finding a way around this issue

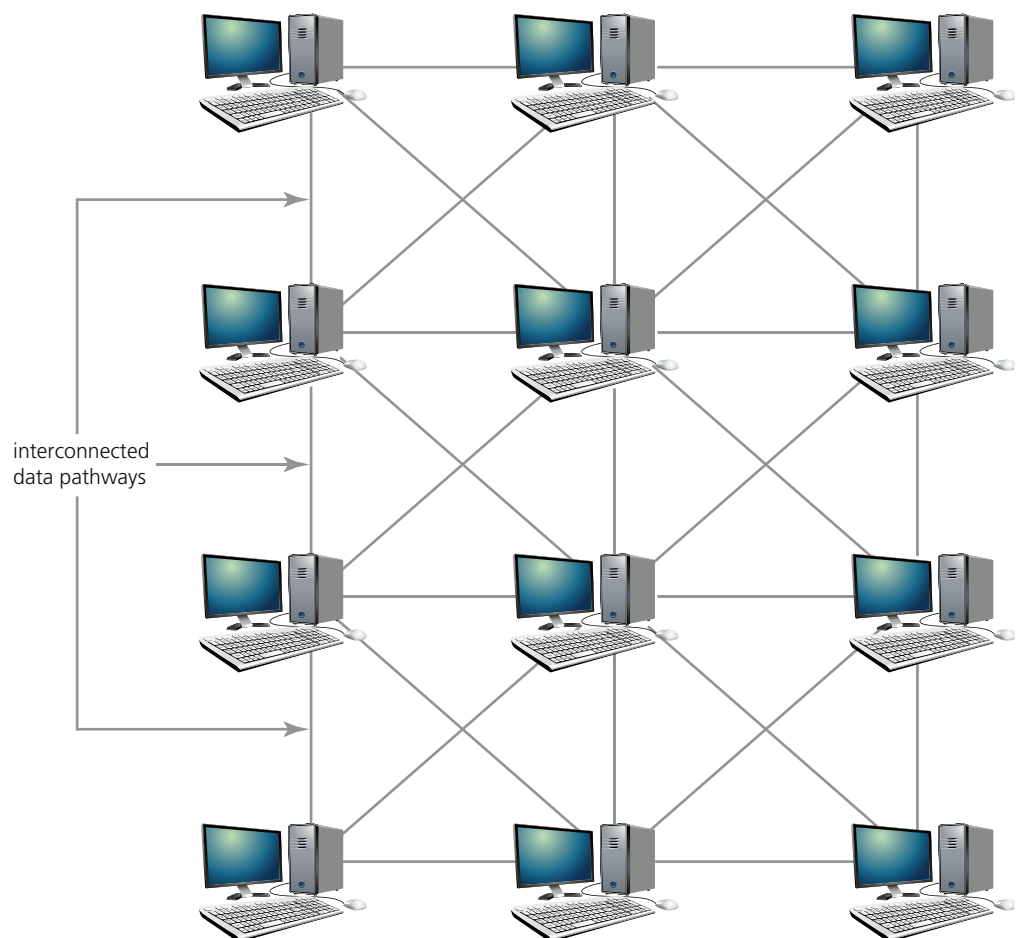
is one of the driving forces behind parallel computers in an effort to greatly improve processor performance.

However, parallel processing requires more expensive hardware. When deciding whether or not to use this type of processor, it is important to take this factor into account.

Parallel computer systems

SIMD and MIMD are the most commonly used processors in parallel processing. A number of computers (containing SIMD processors) can be networked together to form a **cluster**. The processor from each computer forms part of a larger pseudo-parallel system which can act like a **super computer**. Some textbooks and websites also refer to this as grid computing.

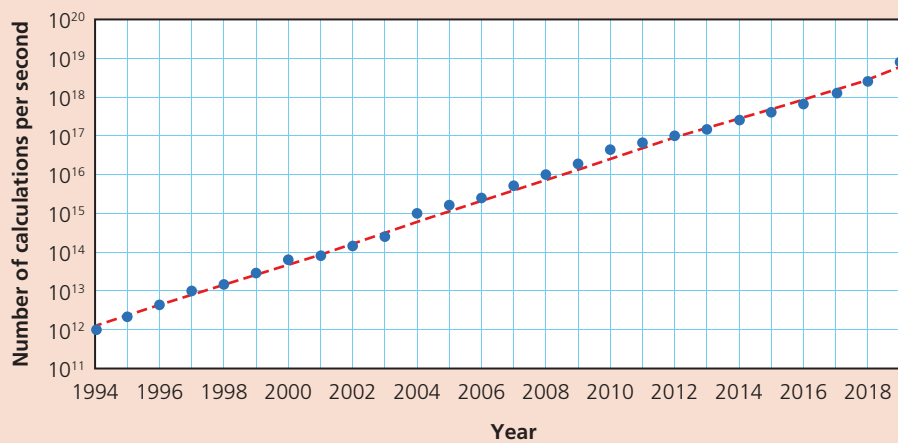
Massively parallel computers have evolved from the linking together of a number of computers, effectively forming one machine with several thousand processors. This was driven by the need to solve increasingly complex problems in the world of science and mathematics. By linking computers (processors) together in this way, it massively increases the processing power of the 'single machine'. This is subtly different to cluster computers where each computer (processor) remains largely independent. In massively parallel computers, each processor will carry out part of the processing and communication between computers is achieved via interconnected data pathways. Figure 15.6 shows this simply.



▲ Figure 15.6 Typical massively parallel computer (processor) system showing interconnected pathways

EXTENSION ACTIVITIES

- 1 Find out more about the applications of multi-computer systems (cluster and massively parallel computers). In particular, research their uses in seismology, astronomy, climate modelling, nuclear physics and weather forecasting models.
- 2 Look at Figure 15.7. Determine, from research, the main reasons for the almost linear expansion in the processing speed of computers over the last 25 years. The data in the graph compares *Number of calculations per second* against *Year*.



▲ Figure 15.7

ACTIVITIES

- 1
 - a) Describe why RISC is an important development in processor technology.
 - b) Describe the main differences between RISC and CISC technologies.
- 2
 - a) What is meant by the von Neumann bottleneck?
 - b) How does the von Neumann bottleneck impact on processor performance?
- 3
 - a) What are the main differences between cluster computers and massively parallel computers?
 - b) Describe one application which uses massively parallel computers. Justify your choice of answer.
- 4 A processor uses pipelining. The following instructions are to be input:
 - 1 LOAD A
 - 2 LOAD B
 - 3 LOAD C
 - 4 ADD A,B,C
 - 5 STORE D
 - 6 OUT D

Draw a diagram to show how many clock cycles are needed for these six instructions to be carried out. Compare your answer to the number of clock cycles needed for a processor using sequential processing.

Key terms

Boolean algebra – a form of algebra linked to logic circuits and based on TRUE and FALSE.

Half adder circuit – carries out binary addition on two bits giving sum and carry.

Full adder circuit – two half adders combined to allow the sum of several binary bits.

Combination circuit – circuit in which the output depends entirely on the input values.

Sequential circuit – circuit in which the output depends on input values produced from previous output values.

Flip-flop circuits – electronic circuits with two stable conditions using sequential circuits.

Cross-coupling – interconnection between two logic gates which make up a flip-flop.

Positive feedback – the output from a process which influences the next input value to the process.

Sum of products (SoP) – a Boolean expression containing AND and OR terms.

Karnaugh maps (K-maps) – a method used to simplify logic statements and logic circuits; uses Gray codes.

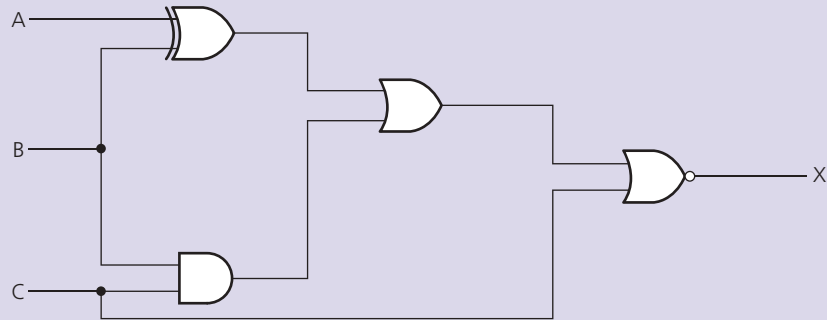
Gray codes – ordering of binary numbers such that successive numbers differ by one bit value only e.g. 00 01 11 10.

15.2 Boolean algebra and logic circuits

WHAT YOU SHOULD ALREADY KNOW

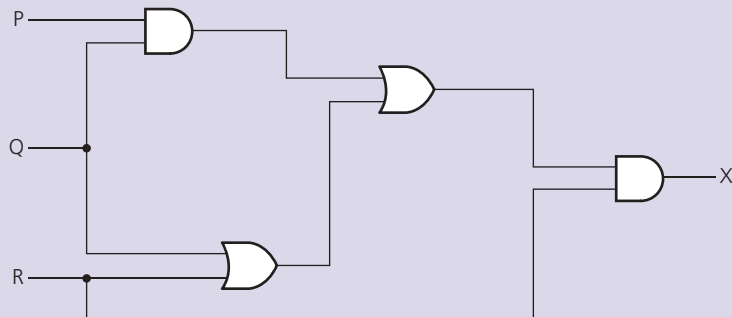
In Chapter 3, you learnt about logic gates and logic circuits. Try the following three questions to refresh your memory before you start to read the second part of this chapter.

- 1 Produce a truth table for the logic circuit shown in Figure 15.8.



▲ Figure 15.8

- 2 Draw a simplified version of the logic circuit shown in Figure 15.9 and write the Boolean expressions to represent Figure 15.9 and your simplified version.



▲ Figure 15.9

- 3 The warning light on a car comes on (= 1) if either one of three conditions occur:
 - sensor1 **and** sensor2 detect a fault (give an input of 1) **OR**
 - sensor2 **and** sensor3 detect a fault (give an input of 1) **OR**
 - sensor1 **and** sensor3 detect a fault (give an input of 1).
 - a) Write a Boolean expression to represent the above problem.
 - b) Give the logic circuit to represent the above system.
 - c) Produce a truth table and check your answers to parts a) and b) agree.

15.2.1 Boolean algebra

Boolean algebra is named after the mathematician George Boole. It is a form of algebra linked to logic circuits and is based on the two statements:

TRUE (1)

FALSE (0)

The notation used in this book to represent these two Boolean operators is:

\bar{A}	which is also written as	NOT A
$A.B$	which is also written as	A AND B
$A + B$	which is also written as	A OR B

Table 15.2 summarises the rules that govern Boolean algebra. It also includes the additional De Morgan's Laws. Also note that, in Boolean algebra, $1 + 1 = 1$, $1 + 0 = 1$, and $\bar{\bar{A}} = A$ (remember your logic gate truth tables in Chapter 3).

Commutative Laws	$A + B = B + A$	$A.B = B.A$
Associative Laws	$A + (B + C) = (A + B) + C$	$A.(B.C) = (A.B).C$
Distributive Laws	$A.(B + C) = (A.B) + (A.C)$ $(A + B).(A + C) = A + B.C$	$A + (B.C) = (A + B).(A + C)$
Tautology/Idempotent Laws	$A.A = A$	$A + A = A$
Tautology/Identity Laws	$1.A = A$	$0 + A = A$
Tautology/Null Laws	$0.A = 0$	$1 + A = 1$
Tautology/Inverse Laws	$A.\bar{\bar{A}} = 0$	$A + \bar{\bar{A}} = 1$
Absorption Laws	$A.(A + B) = A$ $A + A.B = A$	$A + (A.B) = A$ $A + \bar{\bar{A}}.B = A + B$
De Morgan's Laws	$\overline{(A.B)} = \bar{A} + \bar{B}$	$\overline{(A + B)} = \bar{A}.\bar{B}$

▲ Table 15.2 The rules that govern Boolean algebra

Table 15.3 shows proof of De Morgan's Laws. Since the last two columns in each section are identical, then the two De Morgan's Laws hold true.

A	B	\bar{A}	\bar{B}	$\bar{A} + \bar{B}$	$\overline{A.B}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Both columns have the same values

A	B	\bar{A}	\bar{B}	$\bar{A}.\bar{B}$	$\overline{A + B}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Both columns have the same values

▲ Table 15.3 Proof of De Morgan's Laws

Simplification using Boolean algebra

Example 15.1

Simplify $A + B + \bar{A} + \bar{B}$

Solution

Using the *associate laws* we have: $A + B + \bar{A} + \bar{B} \Rightarrow (A + \bar{A}) + (B + \bar{B})$

Using the *inverse laws* we have: $(A + \bar{A}) = 1$ and $(B + \bar{B}) = 1$

Therefore, we have $1 + 1$, which is simply $1 \Rightarrow A + B + \bar{A} + \bar{B} = 1$

Example 15.2

Simplify $A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$

Solution

Rewrite the expression as: $A.B.C + (\bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C})$

This becomes: $(A.B.C + \bar{A}.B.C) + (A.B.C + A.\bar{B}.C) + (A.B.C + A.B.\bar{C})$

which transforms to: $B.C.(A + \bar{A}) + A.C.(B + \bar{B}) + A.B.(C + \bar{C})$

Since $A + \bar{A}$, $B + \bar{B}$ and $C + \bar{C}$ are all equal to 1

then we have: $B.C.1 + A.C.1 + A.B.1 \Rightarrow B.C + A.C + A.B$

ACTIVITY

Simplify the following logic expressions showing all the stages in your simplification.

- $A.C + B.\bar{C}.D + A.\bar{B}.C + A.C.D$
- $B + \bar{A}.\bar{B} + A.C.D + A.\bar{C}$
- $\bar{A}.B.C + A.B.\bar{C} + A.B.C + \bar{A}.B.\bar{C}$
- $\bar{A}.(A + B) + (B + A.A).(A + \bar{B})$
- $(A + C).(A.D + A.\bar{D}) + A.C + C$

15.2.2 Further logic circuits*Half adder circuit and full adder circuit*

In Chapter 3, the use of logic gates to create logic circuits to carry out specific tasks was discussed in much detail. Two important logic circuits used in computers are:

- » the **half adder circuit**
- » the **full adder circuit**.

INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

▲ Table 15.4

Half adder

One of the basic operations in any computer is binary addition. The half adder circuit is the simplest circuit. This carries binary addition on 2 bits generating two outputs:

- » the sum bit (S)
- » the carry bit (C).

Consider $1 + 1$. It will give the result 1 0 (denary value 2). The '1' is the carry and '0' is the sum. Table 15.4 shows this as a truth table.

Figure 15.10 shows how this is often shown in graphic form (left) or as a logic circuit (right):



▲ Figure 15.10

Other logic gates can be used to produce the half adder (see next section).

As you have probably guessed already, the half adder is unable to deal with the addition of several binary bits (for example, an 8-bit byte). To enable this, we have to consider the full adder circuit.

Full adder

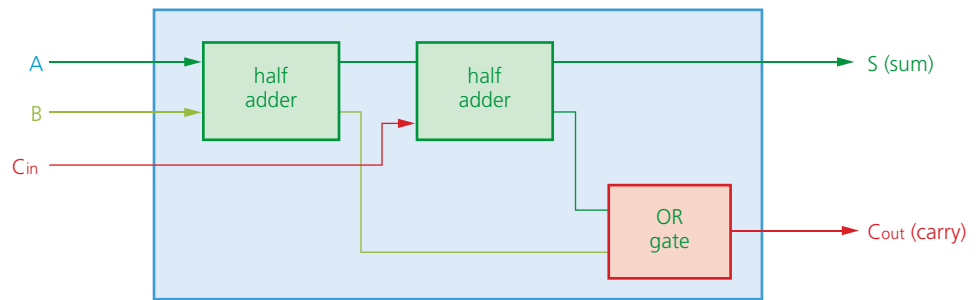
Consider the following sum using 5-bit numbers.

A	0	1	[1]	1	0	
B	0	0	[0]	1	1	
S	1	0	[0]	0	1	← this is the sum produced from the addition
C	1	1	[1]			← this is the carry from the previous bit position

▲ Figure 15.11

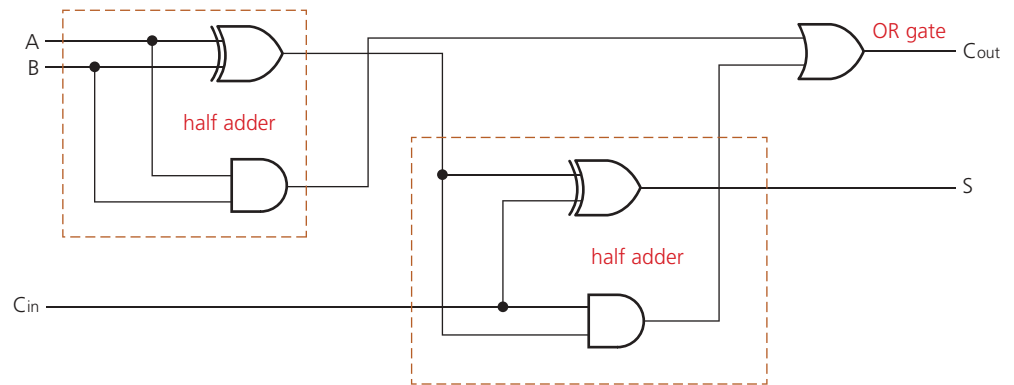
The sum shows how we have to deal with CARRY from the previous column. There are three inputs to consider in this third column, for example, A = 1, B = 0 and C = 1 (S = 0).

This is why we need to join two half adders together to form a full adder:



▲ Figure 15.12

This has an equivalent logic circuit; there are a number of ways of doing this. For example, the following logic circuit uses OR, AND and XOR logic gates.



▲ Figure 15.13

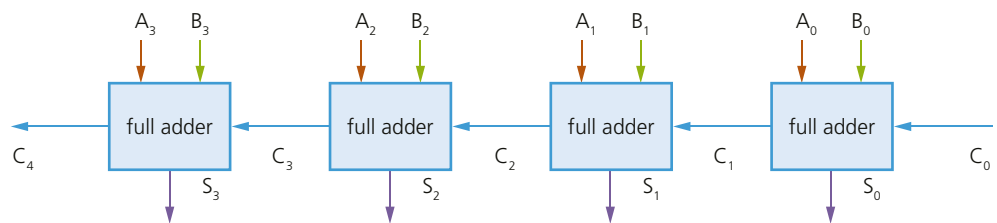
Table 15.5 is the truth table for the full adder circuit.

INPUTS			OUTPUTS	
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As with the half adder circuits, different logic gates can be used to produce the full adder circuit.

The full adder is the basic building block for multiple binary additions. For example, Figure 15.14 shows how two 4-bit numbers can be summed using four full adder circuits.

▲ Table 15.5



▲ Figure 15.14

EXTENSION ACTIVITIES

- 1 Find out why NAND gates are used to produce logic circuits even though they often increase the complexity and size of the overall circuit.
- 2 Produce half adder and full adder circuits using NOR gates only.

ACTIVITIES

- 1
 - a) Produce a half adder circuit using NAND gates only.
 - b) Generate a truth table for your half adder circuit in part a) and confirm it matches the one shown in Section 15.2.4.
- 2
 - a) Produce a full adder circuit using NAND gates only.
 - b) Generate a truth table for your full adder circuit in part a) and confirm it matches the one shown in Section 15.2.4.

15.2.3 Flip-flop circuits

All of the logic circuits you have encountered up to now are **combination circuits** (the output depends entirely on the input values).

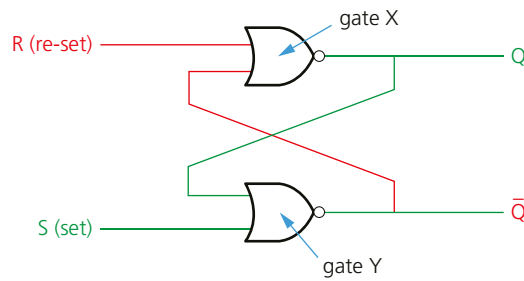
We will now consider a second type of logic circuit, known as a **sequential circuit** (the output depends on the input value produced from a previous output value).

Examples of sequential circuits include **flip-flop circuits**. This chapter will consider two types of flip-flops: SR flip-flops and JK flip-flops.

SR flip-flops

SR flip-flops consist of two **cross-coupled** NAND gates (note: they can equally well be produced from NOR gates). The two inputs are labelled 'S' and 'R', and the two outputs are labelled 'Q' and 'Q̄' (remember Q̄ is equivalent to NOT Q).

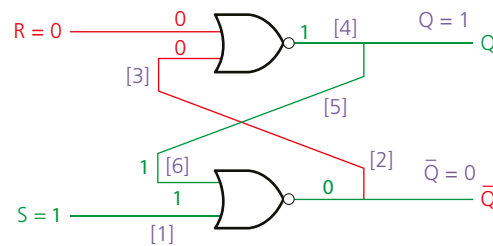
In this chapter, we will use SR flip-flop circuits constructed from NOR gates, as shown in Figure 15.15.



The output from gate 'X' is Q and the output from gate 'Y' is \bar{Q} . The inputs to gate 'X' are R and \bar{Q} (shown in red on Figure 15.15); the inputs to gate 'Y' are S and Q (shown in green on Figure 15.15). The output from each NOR gate gives a form of **positive feedback** (known as cross-coupling, as mentioned earlier).

▲ Figure 15.15 SR flip-flop circuit

We will now consider the truth table to match our SR flip-flop using the initial states of $R = 0$, $S = 1$ and $Q = 1$. The sequence of the stages in the process is shown in Figure 15.16.



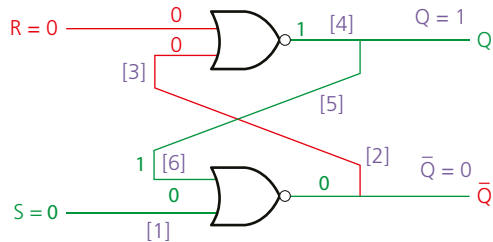
Sequence: [1] → [2] → [3] → [4] → [5] → [6]

Which gives:

S	R	Q	\bar{Q}
1	0	1	0

▲ Figure 15.16

Now consider what happens if we change the value of S from 1 to 0.



Sequence: [1] → [2] → [3] → [4] → [5] → [6]

Which gives:

S	R	Q	\bar{Q}
1	0	1	0

▲ Figure 15.17

The reader is left to consider the other options which lead to the truth table, Table 15.6, for the flip-flop circuit.

	INPUTS		OUTPUTS		Comment
	S	R	Q	\bar{Q}	
(a)	1	0	1	0	
(b)	0	0	1	0	following S = 1 change
(c)	0	1	0	1	
(d)	0	0	0	1	following R = 1 change
(e)	1	1	0	0	

▲ Table 15.6

Explanation:

$S = 1, R = 0, Q = 1, \bar{Q} = 0$

is the set state in this example

$S = 0, R = 0, Q = 1, \bar{Q} = 0$

is the re-set state in this example

$S = 0, R = 1, Q = 0, \bar{Q} = 1$

here the value of Q in line **(b)** remembers the value of Q from line **(a)**; the value of Q in line **(d)** remembers the value of Q in line **(c)**

$S = 0, R = 0, Q = 0, \bar{Q} = 1$

R changes from 1 to 0 and has no effect on outputs (these values are remembered from line **(c)**)

$S = 1, R = 1, Q = 0, \bar{Q} = 0$

this is an invalid case since \bar{Q} should be the compliment (opposite) of Q.

The truth table shows how an input value of $S = 0$ and $R = 0$ causes no change to the two output values; $S = 0$ and $R = 1$ reverses the two output values; $S = 1$ and $R = 0$ always gives $Q = 1$ and $\bar{Q} = 0$ which is the set value.

The truth table shows that SR flip-flops can be used as a storage/memory device for one bit; because a value can be remembered but can also be changed it could be used as a component in a memory device such as a RAM chip.

It is important that the fault condition in line **(e)** is considered when designing and developing storage/memory devices.

JK flip-flops

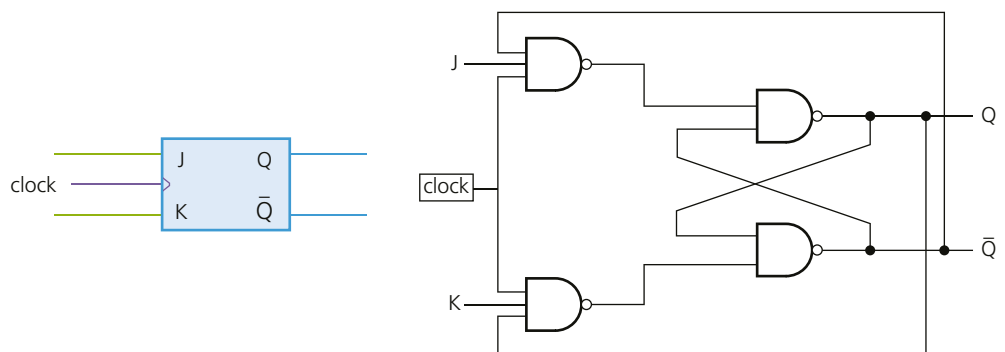
The SR flip-flop has the following problems:

- » Invalid S, R conditions (leading to conflicting output values) need to be avoided.
- » If inputs do not arrive at the same time, the flip-flop can become unstable.

To overcome such problems, the JK flip-flop has been developed. A clock and additional gates are added, which help to synchronise the two inputs and also prevent the illegal states shown in line **(e)** of Table 15.6. The addition of the synchronised input gives four possible input conditions to the JK flip-flop:

- » 1
- » 0
- » no change
- » toggle (which takes care of the invalid S, R states).

The JK flip-flop is represented as shown in Figure 15.18.



▲ Figure 15.18 JK flip-flop symbol (left) and JK flip-flop using NAND gates only (right)

Table 15.7 is the simplified truth table for the JK flip-flop.

J	K	Value of Q before clock pulse	Value of Q after clock pulse	OUTPUT
0	0	0	0	Q is unchanged after clock pulse
0	0	1	1	
1	0	0	1	Q = 1
1	0	1	1	
0	1	0	0	Q = 0
0	1	1	0	
1	1	0	1	Q value toggles between 0 and 1
1	1	1	0	

▲ Table 15.7

- » When $J = 0$ and $K = 0$, there is no change to the output value of Q .
- » If the values of J or K change, then the value of Q will be the same as the value of J (\bar{Q} will be the value of K).
- » When $J = 1$ and $K = 1$, the Q -value toggles after *each* clock pulse, thus preventing illegal states from occurring (in this case, toggle means the flip-flop will change from the 'Set' state to the 'Re-set' state or the other way round).

Use of JK flip-flops

- » Several JK flip-flops can be used to produce shift registers in a computer.
- » A simple binary counter can be made by linking up several JK flip-flop circuits (this requires the toggle function).

15.2.4 Boolean algebra and logic circuits

In Section 15.2.1, the concept of Boolean algebra was introduced. One of the advantages of this method is to represent logic circuits in the form of Boolean algebra.

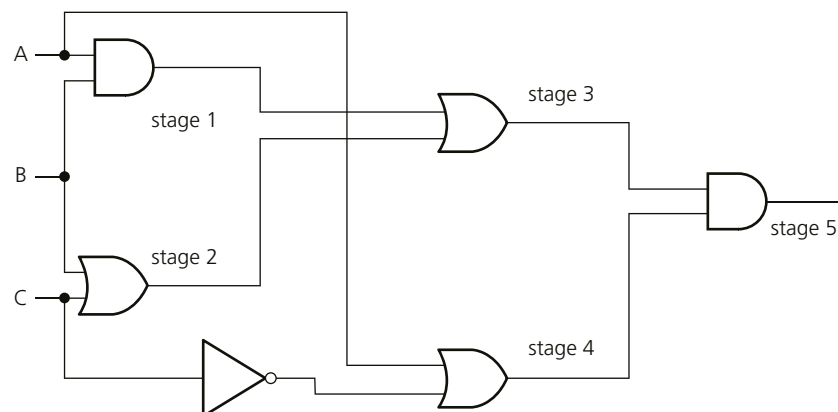
It is possible to use the truth table and apply the **sum of products (SoP)**, or the Boolean expression can be formed directly from the logic circuit.

EXTENSION ACTIVITIES

- 1 Find out how JK flip-flops can be used as shift registers and binary counters in a computer.
- 2 Where else in computer architecture are flip-flop circuits used? Find out why they are used in each case you describe.

Example 15.3

Write down the Boolean expression to represent this logic circuit (Figure 15.19).



▲ Figure 15.19

Solution

Stage 1: A AND B

Stage 2: B OR C

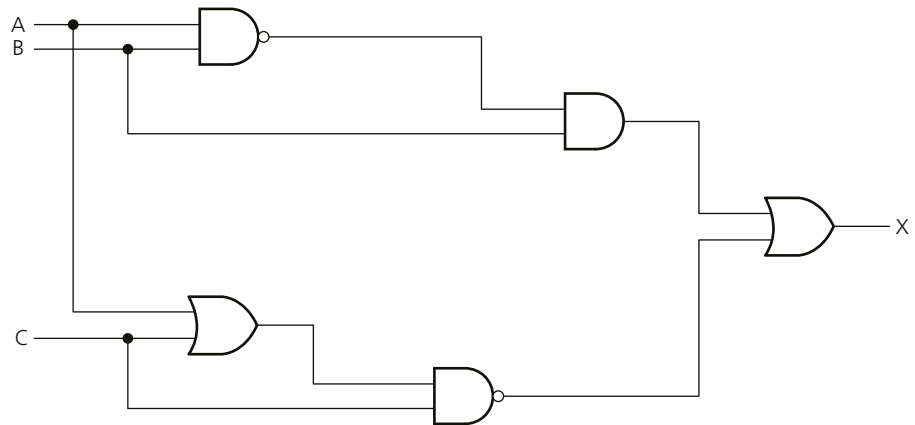
Stage 3: stage 1 OR stage 2 $\Rightarrow (A \text{ AND } B) \text{ OR } (B \text{ OR } C)$

Stage 4: A OR (NOT C)

Stage 5: stage 3 AND stage 4

 $\Rightarrow ((A \text{ AND } B) \text{ OR } (B \text{ OR } C)) \text{ AND } (A \text{ OR } (\text{NOT } C))$ Written in Boolean algebra form: $((A.B) + (B + C)).(A + \bar{C})$ **Example 15.4**

Write the Boolean expression which represents this logic circuit (Figure 15.20).



▲ Figure 15.20

Solution

In this example, we will first produce the truth table and then generate the Boolean expression from the truth table, Table 15.8.

INPUTS			OUTPUT
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

▲ Table 15.8

To produce the Boolean expression from the truth table, we only consider those rows where the output (X) is 1:

$$(\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C + \bar{A}.B.\bar{C} + A.B.\bar{C})$$

If we apply the Boolean algebra laws, we get:

$$(\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C) + (\bar{A}.B.\bar{C} + A.B.\bar{C})$$

$$\Rightarrow ((\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C}) + (\bar{A}.B.\bar{C} + A.B.\bar{C})) + (\bar{A}.B.C + A.B.\bar{C})$$

$$\Rightarrow \bar{A}.\bar{C}.(B + B) + \bar{B}.\bar{C}.(A + A) + (\bar{A}.B.C + A.B.\bar{C})$$

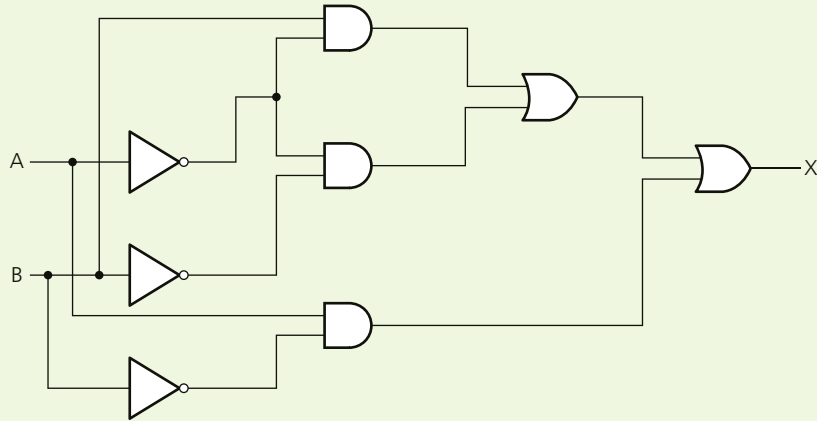
$$\Rightarrow \bar{A}.\bar{C} + \bar{B}.\bar{C} + \bar{A}.B.C + A.B.\bar{C}$$

Therefore, written as a Boolean expression: $\bar{A}.\bar{C} + \bar{B}.\bar{C} + \bar{A}.B.C + A.B.\bar{C}$

We therefore end up with a simplified Boolean expression which has the same effect as the original logic circuit. The reader is left the task of producing the truth table from the above expression to confirm they are both the same.

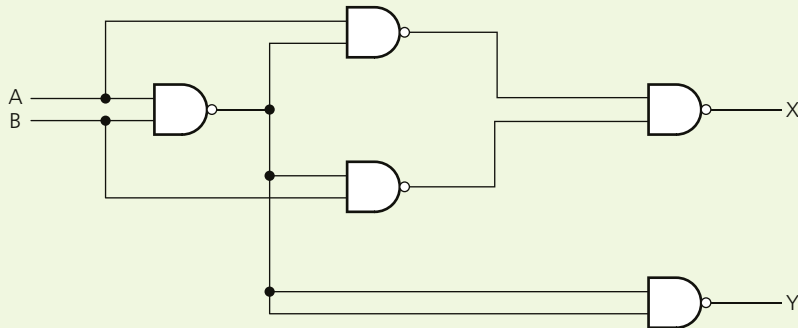
ACTIVITIES

- 1 Produce simplified Boolean expressions for the logic circuits in Figure 15.21 (you can do this directly from the logic circuit or produce the truth table first).



▲ Figure 15.21

- 2 Produce simplified Boolean expressions for the logic circuits in Figure 15.22 (you can do this directly from the logic circuit or produce the truth table first).



▲ Figure 15.22

EXTENSION ACTIVITY

Karnaugh maps make use of **Gray codes**. Find out the origin of Gray codes and other applications of the code.

15.2.5 Karnaugh maps (K-maps)

In the previous activities, it was frequently necessary to simplify Boolean expressions. Sometimes, this can be a long and complex process. **Karnaugh maps** were developed to help simplify logic expressions/circuits.

Example 15.5

Produce a Boolean expression for the truth table (Table 15.9) for the NAND gate.

INPUTS		OUTPUT
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

▲ Table 15.9

Solution

Using sum of products gives the following expression:

$$\bar{A}.B + \bar{A}.\bar{B} + A.\bar{B}$$

Boolean algebra rules produce the simplified expression:

$$\bar{A} + \bar{B}$$

Using Karnaugh maps is a much simpler way to do this.

Each group in the Karnaugh map in Figure 15.23 combines output values where $X = 1$.

	\bar{B}	B
\bar{A}	1	1
A	1	0

▲ Figure 15.23

Thus, $\bar{A}.\bar{B} = 1$, $\bar{A}.B = 1$ and $A.\bar{B} = 1$

The red ring shows \bar{A} as

1	1
---	---

and the green ring shows \bar{B} as

1
1

giving $\bar{A} + \bar{B}$.

As you might expect, there are a number of rules governing Karnaugh maps.

Karnaugh map rules

- The values along the top and the bottom follow Gray code rules.
- Only cells containing a 1 are taken account of.
- Groups can be a row, a column or a rectangle.
- Groups must contain an even number of 1s (2, 4, 6, etc.).
- Groups should be as large as possible.
- Groups may overlap within the above rules.
- Single values can be regarded as a group even if they cannot be combined with other values to form a larger group.
- The final Boolean expression can only consider those values which remain *constant* within the group (i.e. remain a 1 or a 0 throughout the group).

Produce a Boolean expression for the truth table, Table 15.10.

INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

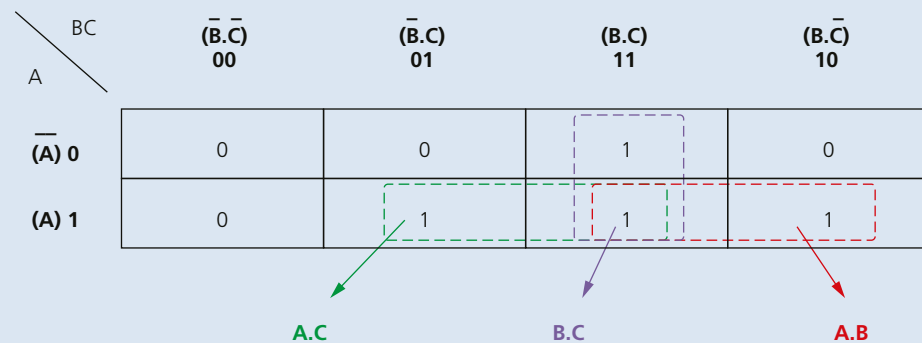
▲ Table 15.10

Solution

Sum of products gives:

$$A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$$

We can now produce the following Karnaugh map to represent this truth table (each 1 value in the K-map represents the above sum of products; so there will be four 1-values in the K-map, where A and BC intersect, where \bar{A} and BC intersect, where A and $\bar{B}C$ intersect, and where A and $B\bar{C}$ intersect):



▲ Figure 15.24

- Green ring: A remains 1, B changes from 0 to 1 and C remains 1 $\Rightarrow A.C$
- Purple ring: A changes from 0 to 1, B remains 1 and C remains 1 $\Rightarrow B.C$
- Red ring: A remains 1, B remains 1 and C changes from 1 to 0 $\Rightarrow A.B$

This gives the simplified Boolean expression: $A.C + B.C + A.B$

Example 15.7

Produce a Boolean expression for the truth table, Table 15.11.

INPUTS				OUTPUT	Sum of products
A	B	C	D	X	
0	0	0	0	1	$\bar{A}\bar{B}\bar{C}\bar{D}$
0	0	0	1	1	$\bar{A}\bar{B}\bar{C}D$
0	0	1	0	1	$\bar{A}\bar{B}C\bar{D}$
0	0	1	1	1	$\bar{A}\bar{B}CD$
0	1	0	0	0	
0	1	0	1	1	$\bar{A}B\bar{C}D$
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	1	$A\bar{B}\bar{C}D$
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	1	$A\bar{B}\bar{C}D$
1	1	1	0	0	
1	1	1	1	0	

▲ Table 15.11

Solution

The sum of products is shown in the right-hand column. This produces the Karnaugh map shown in Figure 15.25.

AB \ CD	$\bar{A}\bar{B}$ 00	$\bar{A}B$ 01	$A\bar{B}$ 11	AB 10
$\bar{C}\bar{D}$ 00	1	0	0	0
$\bar{C}D$ 01	1	1	1	1
CD 11	1	0	0	0
$C\bar{D}$ 10	1	0	0	0

▲ Figure 15.25

This gives $\bar{A}\bar{B} + \bar{C}D$

Notice the following possible K-map options:

This gives the value D since the values of A and B change and the value of C changes (0 to 1); only D is constant at 1.

AB \ CD	($\bar{A}\bar{B}$) 00	($\bar{A}B$) 01	($A\bar{B}$) 11	(AB) 10
($\bar{C}\bar{D}$) 00	1	0	0	0
($\bar{C}D$) 01	1	1	1	1
($C\bar{D}$) 11	1	1	1	1
(CD) 10	0	0	0	0

Columns 1 and 4 can be joined to form a *vertical cylinder*. The values of both C and D change, the value of A changes, the value of B is constant at 0 giving: \bar{B}

AB \ CD	($\bar{A}\bar{B}$) 00	($\bar{A}B$) 01	($A\bar{B}$) 11	(AB) 10
($\bar{C}\bar{D}$) 00	1	0	0	1
($\bar{C}D$) 01	1	0	0	1
($C\bar{D}$) 11	1	0	0	1
(CD) 10	1	0	0	1

The two 1-values can be combined to form a *horizontal cylinder*; values of A and B are constant at 0 and 1 respectively; the value of D is constant at 0; values of C changes from 0 to 1; giving: $\bar{A}.B.\bar{D}$

AB \ CD	($\bar{A}\bar{B}$) 00	($\bar{A}B$) 01	($A\bar{B}$) 11	(AB) 10
($\bar{C}\bar{D}$) 00	0	1	0	0
($\bar{C}D$) 01	0	0	0	0
($C\bar{D}$) 11	0	0	0	0
(CD) 10	0	1	0	0

The four 1-values can be combined at the four corners; value B is constant at 0 and value D is also constant at 0, giving: $\bar{B}.\bar{D}$

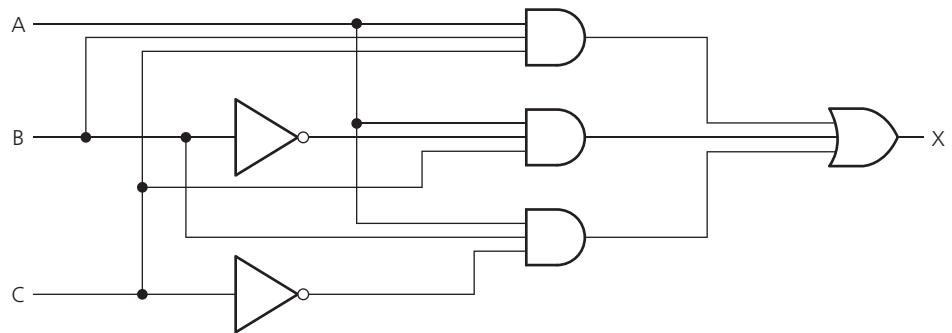
AB \ CD	($\bar{A}\bar{B}$) 00	($\bar{A}B$) 01	($A\bar{B}$) 11	(AB) 10
($\bar{C}\bar{D}$) 00	1	0	0	1
($\bar{C}D$) 01	0	0	0	0
($C\bar{D}$) 11	0	0	0	0
(CD) 10	1	0	0	1

ACTIVITIES

- 1
 - a) Draw the truth table for the Boolean expression:
 $\bar{A}.\bar{B}.C.D + \bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.C.D + A.B.\bar{C}.D + A.B.C.D$
 - b) Draw the Karnaugh map for the Boolean expression in part a).
 - c) Draw a logic circuit for the simplified Boolean expression using AND or OR gates only.
- 2
 - a) Draw the truth table for the Boolean expression:
 $\bar{A}.B.C + A.B.\bar{C} + A.B.C + \bar{A}.B.\bar{C}$
 - b) Draw the Karnaugh map for the expression in part a) and hence write a simplified Boolean expression.
- 3 Four binary signals (A, B, C and D) are used to define an integer in the hexadecimal range (0 to F). The decimal digit satisfies one of the following criteria (i.e. gives an output value of $X = 1$):
 - $X = 1$ if
 - $A = 0$
 - $B = C$, but $A \neq B$ and $A \neq C$
 - $B = 0, C = 0$
 - a) Complete the truth table (with headings A, B, C, D, X) for the above criteria.
 - b) Construct the Karnaugh map to represent the above criteria and produce a simplified Boolean expression.
 - c) Hence, draw an *efficient* logic circuit using AND, OR and NOT gates only. Indicate which input value is not actually required by the logic circuit.

End of chapter questions

- 1
 - a) Write down the Boolean expression to represent the logic circuit below. [3]

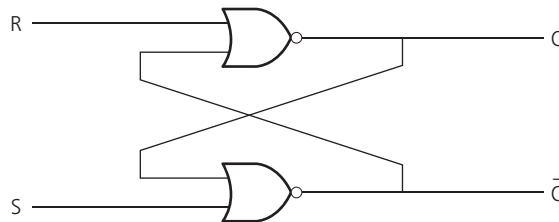


- b) Produce the Karnaugh map to represent the above logic circuit and hence write down a simplified Boolean expression. [3]
 - c) Draw a simplified logic circuit from your Boolean expression in part b) using AND and OR gates only. [2]

- 2 a) Consider the following truth table.

INPUTS				OUTPUT
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

- i) Draw a Karnaugh map from this truth table. [3]
 ii) Use your Karnaugh map from part a) i) to produce a Boolean expression. [4]
- b) Use the laws of Boolean algebra to simplify:
 i) $(A + C).(A.D + A.\bar{D}) + A.C + C$ [2]
 ii) $\bar{A}.(A + B) + (B + A.A).(A + \bar{B})$ [2]
- 3 a) An SR flip-flop is constructed from NOR gates:



- i) Complete the truth table for the SR flip-flop. [4]
 ii) One of the S, R combinations in the truth table should not be allowed to occur. State the values of S and R that should not be allowed to occur. Explain your choice of values. [3]

INPUTS		OUTPUTS	
S	R	Q	\bar{Q}
1	0	1	0
0	0		
0	1		
0	0		
1	1		

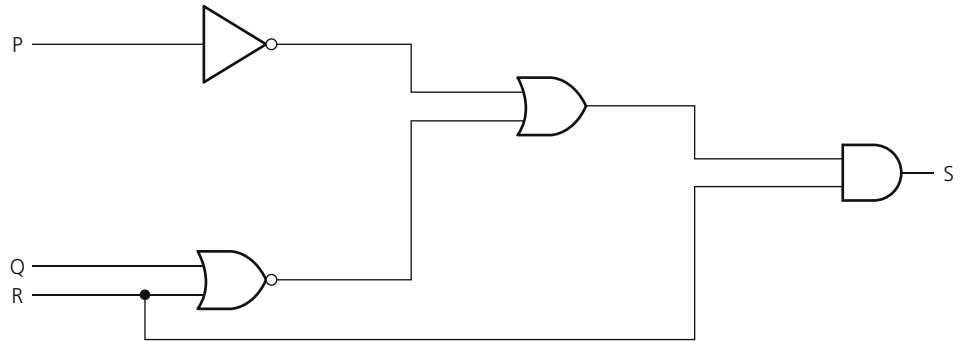
- b) JK flip-flops are another type of flip-flop.
- i) What are the three inputs to a JK flip-flop? [1]
 - ii) Give an advantage of using JK flip-flops. [1]
 - iii) Describe two uses of JK flip-flops in computers. [2]
- 4 a) Describe four types of processors used in parallel processing. [4]
- b) A hardware designer decided to look into the use of parallel processing. Describe three features of parallel processing he needs to consider when designing his new system. [3]
- c) A computer system uses pipelining. An assembly code program being run has 8 instructions. Compare the number of clock cycles required when using pipelining compared to a sequential computer. [3]
- 5 a) Four descriptions and four types of computer architecture are shown below. Draw a line to connect each description to the appropriate type of computer architecture. [4]

Description	Computer architecture
A computer that does not have the ability for parallel processing	SIMD
The processor has several ALUs; each ALU executes the same instructions but on different data	MISD
There are several processors; each processor executes different instructions drawn from a common pool; each processor operates on different data drawn from a common pool	SISD
There is only one processor executing one set of instructions on a single set of data	MIMD

- b) In a massively parallel computer, explain what is meant by:
- i) massive [1]
 - ii) parallel. [1]
- c) There are both hardware and software issues that have to be considered for parallel processing to succeed. Describe one hardware and one software issue. [4]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q4 November 2015

6 A logic circuit is shown.



- a) Write the Boolean expression corresponding to this logic circuit. [4]
- b) Complete the truth table for this logic circuit. [2]

P	Q	R	Working space	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- c) i) Complete the Karnaugh map (K-map) for the truth table in part b). [1]

		PQ			
		00	01	11	10
R	0				
	1				

- ii) The K-map can be used to simplify the function in part a). Draw loop(s) around appropriate groups to produce an optional sum of products. [1]
- iii) Write a simplified sum of products, using your answer to part ii). [1]
- d) One Boolean identity is:
 $(A + B).C = A.C + B.C$
 Simplify the expression for S in part a) to the expression for S in part c) iii).
 Use your given identity and De Morgan's Laws. [3]

Cambridge International AS & A Level Computer Science 9608 Paper 32 Q3 June 2017

Develop computational thinking and ensure full coverage of the revised Cambridge International AS & A Level Computer Science syllabus (9618) with this comprehensive Student's Book written by experienced authors and examiners.

- » Improve understanding with clear explanations, examples, illustrations and diagrams, plus a glossary of key terms.
- » Reinforce learning with a range of activities, exercises, and exam-style questions.
- » Prepare for further study with extension activities that go beyond the requirements of the syllabus and prompt further investigation about new developments in technology.
- » Follow a structured route through the course with in-depth coverage of the full AS & A Level syllabus.

Dynamic Learning

This book is fully supported by Dynamic Learning – the online subscription service that helps make teaching and learning easier. Dynamic Learning provides unique tools and content for:

- front-of-class teaching
- streamlining planning and sharing lessons
- focused and flexible assessment preparation
- independent, flexible student study



Sign up for a free trial – visit: www.hoddereducation.com/dynamiclearning



This resource is endorsed by
Cambridge Assessment International Education

- ✓ Supports the full syllabus (9618) for examination from 2021
- ✓ Has passed Cambridge International's rigorous quality-assurance process
- ✓ Developed by subject experts
- ✓ For Cambridge schools worldwide

This textbook has been written for the revised Cambridge International AS & A Level Computer Science syllabus (9618).

We are working with Cambridge Assessment International Education to gain endorsement for this forthcoming title.

For over 25 years we have been trusted by Cambridge schools around the world to provide quality support for teaching and learning. For this reason we have been selected by Cambridge Assessment International Education as an official publisher of endorsed material for their syllabuses.

