Acknowledgements

We are grateful to the AQA Examination Board for permission to use questions from past papers.

The answers in the Teacher’s Supplement are the sole responsibility of the authors and have neither been provided nor approved by the examination board.

We would also like to thank the following for permission to reproduce copyright photographs:

Screenshots of Arriva Bus App © Arriva PLC
Colossus photograph © The National Archives
Google Maps ‘StreetView’ © Google 2015
Other photographic images © Shutterstock

Graphics: Roger Stayte and Rob Heathcote
Cover picture © ‘Golden Moor’ 2014
Oil on canvas
Reproduced with the kind permission of Heather Duncan
www.heatherduncan.com

Design and artwork by OnThree
www.on-three.com

Typeset by Chapter One (London) Ltd

First edition 2015, reprinted 2016

A catalogue entry for this book is available from the British Library

Copyright © P.M.Heathcote and R.S.U.Heathcote 2016
All rights reserved

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the prior written permission of the copyright owner.
Preface

The aim of this textbook is to provide detailed coverage of the topics in the new AQA AS Level Computer Science specification.

The book is divided into six sections and within each section, each chapter covers material that can comfortably be taught in one or two lessons.

In the first year of this course there will be a strong emphasis on learning to program. You will start by learning the syntax of your chosen programming language – that is, the rules of how to write correct statements that the computer can understand. Then you will code simple programs, building up your skills to the point where you can understand and make additions and amendments to a program consisting of several hundred lines of code.

Sections 1 and 2 of this book can be studied in parallel with your practical programming sessions. It will give you practice in the skills you need to master.

At the end of the year, if you are sitting the AS level exam, Paper 1 will test your general knowledge of programming and your ability to code, and ask you to add and make amendments to a substantial program which you will have had a chance to look at and run beforehand.

Paper 2 will test your knowledge and understanding of the theory parts of the specification.

If you are sitting the A level exam after two years, rather than the AS level, these papers will be a useful end-of-year test of your grasp of the subject, before you go on to cover the second half of the specification.

Each chapter contains exercises and questions, some from past examination papers, and answers to all these are available to teachers only in a Teachers Supplement which can be ordered from our website www.pgonline.co.uk.

Approval message from AQA

This textbook has been approved by AQA for use with our qualification. This means that we have checked that it broadly covers the specification and we are satisfied with the overall quality. Full details of our approval process can be found on our website.

We approve textbooks because we know how important it is for teachers and students to have the right resources to support their teaching and learning. However, the publisher is ultimately responsible for the editorial control and quality of this book.

Please note that when teaching the AS Level Computer Science course, you must refer to AQA’s specification as your definitive source of information. While this book has been written to match the specification, it cannot provide complete coverage of every aspect of the course.

A wide range of other useful resources can be found on the relevant subject pages of our website: www.aqa.org.uk.
Contents

Section 1
Fundamentals of programming
   Chapter 1  Programming basics  2
   Chapter 2  Selection  8
   Chapter 3  Iteration  13
   Chapter 4  Arrays  17
   Chapter 5  Subroutines  21
   Chapter 6  Files and exception handling  29

Section 2
Problem solving and theory of computation  33
   Chapter 7  Solving logic problems  34
   Chapter 8  Structured programming  39
   Chapter 9  Writing and interpreting algorithms  42
   Chapter 10  Testing and evaluation  48
   Chapter 11  Abstraction and automation  52
   Chapter 12  Finite state machines  60

Section 3
Data representation  67
   Chapter 13  Number systems  68
   Chapter 14  Bits, bytes and binary  72
   Chapter 15  Binary arithmetic and the representation of fractions  77
   Chapter 16  Bitmapped graphics  83
   Chapter 17  Digital representation of sound  88
   Chapter 18  Data compression and encryption algorithms  93
# Section 4

## Hardware and software

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Hardware and software</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>Role of an operating system</td>
<td>103</td>
</tr>
<tr>
<td>21</td>
<td>Programming language classification</td>
<td>106</td>
</tr>
<tr>
<td>22</td>
<td>Programming language translators</td>
<td>110</td>
</tr>
<tr>
<td>23</td>
<td>Logic gates</td>
<td>114</td>
</tr>
<tr>
<td>24</td>
<td>Boolean algebra</td>
<td>118</td>
</tr>
</tbody>
</table>

# Section 5

## Computer organisation and architecture

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Internal computer hardware</td>
<td>126</td>
</tr>
<tr>
<td>26</td>
<td>The processor</td>
<td>132</td>
</tr>
<tr>
<td>27</td>
<td>The processor instruction set</td>
<td>138</td>
</tr>
<tr>
<td>28</td>
<td>Assembly language</td>
<td>142</td>
</tr>
<tr>
<td>29</td>
<td>Input-output devices</td>
<td>148</td>
</tr>
<tr>
<td>30</td>
<td>Secondary storage devices</td>
<td>154</td>
</tr>
</tbody>
</table>

# Section 6

## Communication: technology and consequences

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Communication methods</td>
<td>159</td>
</tr>
<tr>
<td>32</td>
<td>Network topology</td>
<td>164</td>
</tr>
<tr>
<td>33</td>
<td>Client-server and peer-to-peer</td>
<td>168</td>
</tr>
<tr>
<td>34</td>
<td>Wireless networking, CSMA and SSID</td>
<td>171</td>
</tr>
<tr>
<td>35</td>
<td>Communication and privacy</td>
<td>176</td>
</tr>
<tr>
<td>36</td>
<td>The challenges of the digital age</td>
<td>179</td>
</tr>
</tbody>
</table>

## References

<table>
<thead>
<tr>
<th>References</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>185</td>
</tr>
</tbody>
</table>

## Index

<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>186</td>
</tr>
</tbody>
</table>
String-handling functions

Programming languages have a number of built-in string-handling methods or functions. Some of the common ones in a typical language are:

- **len(string)**
  Returns the length of a string

- **string.substring(index1,index2)**
  Returns a portion of string inclusive of the characters at each index position

- **string.find(str)**
  Determines if str occurs in a string. Returns index (the position of the first character in the string) if found, and -1 otherwise. In our pseudocode we will assume that string(1) is the first element of the string, though in Python, for example, the first element is string(0)

- **ord("a")**
  Returns the integer value of a character (97 in this example)

- **chr(97)**
  Returns the character represented by an integer ("a" in this example)

**Q3:** What will be output by the following lines of code?

```python
x = "Come into the garden, Maud"
y = len(x)
z = x.find("Maud")
OUTPUT "x= ", x
OUTPUT "y= ", y
OUTPUT "z= ", z
```

To **concatenate** or join two strings, use the + operator.

- e.g. “Johnny” + “Bates” = “JohnnyBates”

### String conversion operations

- **int("1")**
  converts the character “1” to the integer 1

- **str(123)**
  converts the integer 123 into a string “123”

- **float("123.456")**
  converts the string “123.456” to the real number 123.456

- **str(123.456)**
  converts the real number 123.456 to the string “123.456”

- **date(year,month,day)**
  returns a number that you can calculate with

Converting between strings and dates is usually handled by functions built in to string library modules, e.g. `strtodate("01/01/2016")`.

Example:

```python
date1 ← strtodate("18/01/2015")
date2 ← strtodate("30/12/2014")
days ← date1 - date2
OUTPUT date1, date2, days
```

This will output

```
2015-01-18 2014-12-30 19
```
Chapter 12—Finite state machines

Objectives

- Understand what is meant by a finite state machine
- List some of the uses of a finite state machine
- Draw and interpret simple state transition diagrams for finite state machines with no output
- Draw a state transition table for a finite state machine with no output and vice versa

What is a finite state machine?

A finite state machine is a model of computation used to design computer programs and sequential logic circuits. It is not a “machine” in the physical sense of a washing machine, an engine or a power tool, for example, but rather an abstract model of how a machine reacts to an external event. The machine can be in one of a finite number of states and changes from one state to the next state when triggered by some condition or input (say, a signal from a timer).

In a finite state machine:

- The machine can only be in one state at a time
- It can change from one state to another in response to an event or condition; this is called a **transition**. Often this is a switch or a binary sensor.
- The Finite State Machine (FSM) is defined by a list of its states and the condition for each transition

There can be outputs linked to the FSM’s state, but in this chapter we will be considering only FSMs with no output.

Example 1

Draw an FSM to model the states and transitions of a door. The door can be open, closed or locked. It can change from the state of being open to closed, from closed to locked, but not, say, from locked to open. (It has to be unlocked first.)
Example 2

Draw an FSM to represent a light switch. When the button is pressed, the light goes on. When the button is pressed again, the light goes off.

There is just one input B to this system: Button pressed (B=1) or Button not pressed (B=0).

Notice that in each state, both the transitions B=0 and B=1 are drawn. If the light is off, the transition B=0 has no effect so the transition results in the same state. Likewise, if the light is on, as long as the button is not pressed, the light will stay on.

Usage of finite state machines

FSMs are widely used in modelling the design of hardware digital systems, compilers and network protocols. They are also used in the definition of languages, and to decide whether a particular word is allowed in the language.

A finite state machine which has no output is also known as a finite state automaton. It has a start state and a set of accept states which define whether it accepts or rejects finite strings or symbols. The finite state automaton accepts a string c₁, c₂…cₙ if there is a path for the given input from the start state to an accept state. The language recognised by the finite state automaton consists of all the strings accepted by it.

If, when you are in a particular state, the next state is uniquely determined by the input, it is a deterministic final state automaton. All the examples which follow satisfy this condition.

Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="symbol" alt="State" /></td>
<td>State</td>
</tr>
<tr>
<td><img src="symbol" alt="Start state" /></td>
<td>Start state</td>
</tr>
<tr>
<td><img src="symbol" alt="Accept state" /></td>
<td>Accept state</td>
</tr>
<tr>
<td><img src="symbol" alt="Transition" /></td>
<td>Transition</td>
</tr>
</tbody>
</table>
Example 3

Use an FSM to represent a valid identifier in a programming language. The rules for a valid identifier for this particular language are:

- The identifier must start with a lowercase letter
- Any combination of letters and lowercase numbers may follow
- There is no limit on the length of the identifier

In this diagram, the start state $S_1$ is represented by a circle with an arrow leading into it. The accept state $S_2$ is denoted by a double circle. $S_3$ is a “dead state” because having arrived here, the string can never reach the accept state.

Each character of the input string is input sequentially to the FSM and if the last character reaches the final state $S_2$ (the accept state), the string is valid and is accepted. If it ends up anywhere else the string is invalid.

Note that there can only be one starting state but there may be more than one accept state (or no accept states).

Q1: Which of the following strings is valid and accepted by this finite state machine?
   (i) a   (ii) bba   (iii) abbaa   (iv) bbbb
**What is encryption?**

Encryption is the transformation of data from one form to another to prevent an unauthorised third party from being able to understand it. The original data or message is known as plaintext. The encrypted data is known as ciphertext. The encryption method or algorithm is known as the cipher, and the secret information to lock or unlock the message is known as a key.

The Caesar cipher and the Vernam cipher offer polar opposite examples of security. Where the Vernam offers perfect security, the Caesar cipher is very easy to break with little or no computational power. There are many others methods of encryption – some of which may take many computers, many years to break, but these are still breakable and the principles behind them are similar.

**The Caesar cipher**

Julius Caesar is said to have used this method to keep messages secure. The Caesar cipher (also known as a shift cipher) is a type of substitution cipher and works by shifting the letters of the alphabet along by a given number of characters; this parameter being the key. Below is an example of a shift cipher using a key of 5. (An algorithm for this cipher is given as an example on page 46.)

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E |

**Q2:** Using the table above, what is the ciphertext for ‘JULIUS CAESAR’ using a shift of 5?

**Q3:** What word can be translated from the following ciphertext, which uses a key of -2: ZYBECP

You will no doubt be able to see the ease with which you might be able to decrypt a message using this system.

**DGYDQFH WR ERUGHU DQG DWWDFN DW GDZQ**

Even if you had to attempt a brute force attack on the message above, there are only 25 different possibilities (since a shift of zero means the plaintext and the ciphertext are identical). Otherwise you might begin by guessing the likelihood of certain characters first and go from there. Using cryptanalysis on longer messages, you would quickly find the most common ciphertext letter and could start by assuming this was an E, for example, or perhaps an A. *(Hint.)*

**Cryptanalysis and perfect security**

Other ciphers that use non-random keys are open to a cryptanalytic attack and can be solved given enough time and resources. Even ciphers that use a computer-generated random key can be broken since mathematically generated random numbers are not actually random; they just appear to be so. A truly random sequence must be collected from a physical and unpredictable phenomenon such as white noise, the timing of a hard disk read/write head or radioactive decay. A truly random key must be used with a Vernam cipher to ensure it is mathematically impossible to break.

**The Vernam cipher**

The Vernam cipher, invented in 1917 by the scientist Gilbert Vernam, is one implementation of a class of ciphers known as one-time pad ciphers, all of which offer perfect security if used properly. All others are based on computational security and are theoretically discoverable given enough time, ciphertext and computational power. Frequency analysis is a common technique used to break a cipher.
One-time pad

To provide perfect security, the encryption key or one-time pad must be equal to or longer in characters than the plaintext, be truly random and be used only once. The sender and recipient must meet in person to securely share the key and destroy it after encryption or decryption. Since the key is random, so will be the distribution of the characters meaning that no amount of cryptanalysis will produce meaningful results.

The bitwise exclusive or XOR

A Boolean XOR operation is carried out between the binary representation of each character of the plaintext and the corresponding character of the one-time pad. The XOR operation is covered in Chapter 23 and you may want to refer to this to verify the output for any combination of 0 and 1. Use the ASCII chart on page 73 for reference.

<table>
<thead>
<tr>
<th>Plaintext: M</th>
<th>Key: +</th>
<th>XOR: f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Q4: Using the ASCII chart and the XOR operator, what ciphertext character will be produced from the letter E with the key w?

Using this method, the message “Meet on the bridge at 0300 hours” encrypted using a one-time pad of +tkiGeMxGvnhoQ0xQDIILVdT4sIJm9qf will produce the ciphertext:

f□g#X3$cH#Y6!i(=vTg"Ci"1 L L

The encryption process will often produce strange symbols or unprintable ASCII characters as in the above example, but in practice it is not necessary to translate the encrypted code back into character form, as it is transmitted in binary. To decrypt the message, the XOR operation is carried out on the ciphertext using the same one-time pad, which restores it to plaintext.

Exercises

1. Explain the difference between lossy and lossless data compression.

2. Run-length encoding (RLE) is a pattern substitution compression algorithm. Data is stored in the format (colour, run) where 0 = White, 1 = Black.
   
   (0,1), (1,5), (0,1),
   (1,7),
   (1,1), (0,2), (1,1), (0,2), (1,1),
   (1,7),
   (0,1), (1,1), (0,1), (1,1), (0,1), (1,1), (0,1),
   (0,1), (1,1), (0,1), (1,1), (0,1), (1,1), (0,1),
   (0,1), (1,1), (0,1), (0,3), (0,1), (1,1), (0,1)
Assembly language instructions

Machine code was the first “language” used to enter programs by early computer programmers. The next advance in programming was to use mnemonics instead of binary codes, and this was called assembly code or assembly language. Each assembly language instruction translates into one machine code instruction.

Different mnemonic codes are used by different manufacturers, so there are several versions of assembly language.

Typical statements in machine code and assembly language are:

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Assembly code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 1100</td>
<td>LDA #12</td>
<td>Load the number 12 into the accumulator</td>
</tr>
<tr>
<td>0010 0010</td>
<td>ADD #2</td>
<td>Add the number 2 to the contents of the accumulator</td>
</tr>
<tr>
<td>0111 1111</td>
<td>STO 15</td>
<td>Store the result from the accumulator in location 15</td>
</tr>
</tbody>
</table>

The # symbol in this assembly language program signifies that the immediate addressing mode is being used.

Q5: Write a statement in a high level language which performs an operation equivalent to the three statements in the above machine code program, with the result being stored in a location called TOTAL.

Q6: Write a machine code program, and an equivalent assembly language program, to add the contents of locations 10 and 11 and store the result in location 14.

Exercises

1. A computer with a 16-bit word length uses an instruction set with 6 bits for the opcode, including the addressing mode.
   (a) What is an instruction set? [1]
   (b) How many instructions could be included in the instruction set of this computer? [1]
   (c) What is the largest number that can be used as data in the instruction? [1]
   (d) What would be the effect of increasing the space allowed for the opcode by 2 bits? [2]
   (e) What would be the benefits of increasing the word size of the computer? [2]

2. The high-level language statement
   \[ X = Y + 6 \]
   is to be written in assembly language.

   Complete the following assembly language statements, which are to be the equivalent of the above high level language statement. The LOAD and STORE instructions imply the use of the accumulator register.

   LOAD ........................................
   ............................................#6
   STORE .....................................
Parity

Computers use either even or odd parity. In an even parity machine, the total number of ‘on’ bits in every byte (including the parity bit) must be an even number. When data is transmitted, the parity bit is set at the transmitting end and parity is checked at the receiving end, and if the wrong number of bits are ‘on’, an error has occurred. In the diagram below the parity bit is the most significant bit (MSB).

Parity bit in even parity system

Q2: The ASCII codes for P and Q are 1010000 and 1010001 respectively. In an even parity transmission system, what will be the value of the parity bit for the characters P and Q?

Synchronous transmission

Using synchronous transmission, data is transferred at regular intervals that are timed by a clocking signal, allowing for a constant and reliable transmission for time-sensitive data, such as real-time video or voice. Parallel communication typically uses synchronous transmission – for example, in the CPU, the clock emits a signal at regular intervals and transmissions along the address bus, data bus and control bus start on a clock signal, which is shared by both sender and receiver.

Asynchronous transmission

Using asynchronous transmission, one byte at a time is sent, with each character being preceded by a start bit and followed by a stop bit.

The start bit alerts the receiving device and synchronises the clock inside the receiver ready to receive the character. The baud rate at the receiving end has to be set up to be the same as the sender’s baud rate or the signal will not be received correctly. The stop bit is actually a “stop period”, which may be arbitrarily long. This allows the receiver time to identify the next start bit and gives the receiver time to process the data before the next value is transmitted.

A parity bit is also usually included as a check against incorrect transmission. Thus for each character being sent, a total of 10 bits is transmitted, including the parity bit, a start bit and a stop bit. The start bit may be a 0 or a 1, the stop bit is then a 1 or a 0 (always different). A series of electrical pulses is sent down the line as illustrated below:
Index

A
abstraction, 52, 108
   data, 57
   functional, 56
   problem, 57
   procedural, 55
accumulator, 132, 138
active tags, 152
ADC, 90
address bus, 127, 128, 135
addressing mode
   direct, 139
   immediate, 139
algorithm, 2
   interpreting, 45
   sorting, 44
ALU, 132
Amazon, 179
analogue
   data, 89
   to digital conversion, 90
analysis, 34
AND, 10, 144
AND gate, 115
Application Programming Interface, 103
application software, 102
arithmetic logic unit, 127, 132
arithmetic operations, 3, 127, 143
arrays, 17, 19
ASCII, 73
assembler, 110
assembly language, 108, 109, 140, 142
asynchronous transmission, 162
audio bit depth, 88
automation, 58
automaton, 61

B
backing store management, 104
bandwidth, 161
barcode reader, 149
barcodes
   2-D, 148
   linear, 148
baud rate, 161
binary
   addition, 77
   converting to and from decimal, 69
   file, 31
   fixed point, 80
   floating point, 81
   multiplication, 78
   negative numbers, 79
   number system, 69
   search, 43
   subtraction, 80
bit, 72
   depth, 88
   rate, 161
bitmap image, 83
block-structured languages, 39
Blu-Ray, 155
Boolean algebra, 120
   Absorption rule, 120
   Associative rule, 120
   Commutative rule, 120
   Distributive rule, 120
Boolean operators, 10
bridges of Königsberg, 54
bubble sort, 44
bus, 127
   address, 128
   control, 128
   data, 128
byte, 72
bytecode, 112

C
cache memory, 135
Caesar cipher, 96
camera-based readers, 150
carry, 78
CASE, 10
CCD, 151
CD ROM, 155
Central Processing Unit, 126
check digit, 75
checksum, 75
ciphertext, 96
CIR, 133
client-server network, 168
clock speed, 135
CMOS, 151
Colossus computer, 106
colour depth, 83
comments, 3
compare and branch
   instructions, 143
compiler, 110, 112
composition, 57
compression
   dictionary-based, 95
   lossless, 93
   lossy, 93
computational thinking, 35, 52
Computer Misuse Act, 183
constants, 6
control bus, 127, 128
control unit, 127, 132
Copyright, Designs and Patents Act (1988), 183
CPU, 126
cryptanalysis, 96, 97
CSMA/CA, 173
CSMA/CD, 166
current instruction register, 133
cyber-attack, 177
cyber-bullying, 181

D
dac, 90
data
   analogue, 89
   boundary, 48
   bus, 127, 128, 135
   communication, 159
   digital, 89
   erroneous, 48
   structures, 17
   transfer operations, 143
types, 3
Data Protection Act (1998), 183
De Morgan’s laws, 118
decomposition, 57
denary, 80
destruction of jobs, 180
dictionary based compression, 95
digital
   camera, 151
data, 89
to analogue conversion, 90
disk defragmenter, 101
DPI, 83
driverless cars, 182
dry run, 49
dual-core processor, 134
E
EAN, 75
eBay, 179
economic impact of the Internet, 179
elementary data types, 17
embedded systems, 130
encryption, 96
Enigma code, 106
error checking, 74
ethics, 182
evaluating a program, 46
evaluation, 50
event messages, 91
exbi, 72
F
fetch-execute cycle, 134
field, 29
file, 29
binary, 31
text, 29
file server, 168
finite state
automaton, 61
machine, 60
first generation language, 53
FOR … ENDFOR, 15
frequency of a sound, 90
functions, 5, 21
string-handling, 5
G
gate
NOT, AND, OR, 114
XOR, NAND, NOR, 116
general purpose registers, 132
gibi, 72
Google, 179
Street View, 178
graph theory, 55
H
hard disk, 154
hardware, 100
Harvard architecture, 130
hexadecimal, 70
converting to and from decimal, 70
hierarchy chart, 40
high-level languages, 109
I
I/O controller, 127, 129
IF … THEN, 8
image resolution, 83
imperative language, 109
information hiding, 54, 57
Instagram, 181
instruction set, 107, 110
interfaces, 23, 129
internet security 172
internet of things, 182
interpreter, 111, 112
interrupt, 136
handling, 105
Interrupt Service Routine, 136
irrational number, 68
ISBN, 75
iteration, 13
J-K
Java Virtual Machine, 112
kibi, 72
kilobyte, 72
L
LAN, 164
laser
printer, 152
scanner, 150
latency, 161
library programs, 101
loader, 103
local area network, 164
logic gates, 114
logical bitwise operators, 144
logical operations, 127
low-level language, 108
M
MAC address, 167
machine code, 106
instruction format, 138
majority voting, 75
MAR, 133
MBR, 133
mebi, 72
memory
address register, 133
buffer register, 133
data register, 133
management, 104
metadata, 84
MIDI, 91
metadata, 91
mnemonics, 142
modular programming, 25
module, 39
modulo 10 system, 76
N
NAND gate, 116
natural number, 68
nested loops, 15
network
client-server, 168
peer-to-peer, 169
security, 172
station, 171
nibble, 72
NOR gate, 116
NOT, 10, 11, 144
gate, 114
number
irrational, 68
natural, 68
ordinal, 68
rational, 68
real, 68
number system
binary, 69
hexadecimal, 70
Nyquist’s theorem, 90
O
object code, 110
one-time pad, 97
opcode, 106, 138
operand, 106, 138
operating system, 100, 103
operation code, 106, 138
optical disk, 155
OR, 10, 144
OR gate, 115
overflow, 78
Oyster card, 152

P
parallel data transmission, 160
parity, 162
bit, 74
passive tags, 152
PC, 133
pebi, 72
peer-to-peer network, 169
pen-type reader, 149
peripheral management, 105
piracy, 170
pixel, 83
plaintext, 96
platform independence, 112
problem solving strategies, 36
procedure interface, 56
procedures, 21
processor, 127
instruction set, 138
performance, 134
scheduling, 104
program counter, 133
protocol, 163
pseudocode, 2

Q-R
quad-core processor, 134
Quick Response (QR) code, 148
Radio Frequency Identification, 151
raster image, 83
rational number, 68
record, 29
register, 127
relational operators, 8
REPEAT … UNTIL, 14
resolution, 83
resource management, 100
RFID, 151

RLE, 94
router, 171
RTS/CTS, 173
Run Length Encoding, 94

S
sample resolution, 88
secondary storage, 154
selection statement, 8
serial data transmission, 159
server
database, 168
file, 168
mail, 168
print, 168
web, 168
Service Set Identification, 172
Snowden, Edward, 176
software, 34, 100, 102
application, 102
bespoke, 102
off-the-shelf, 102
system, 100
utility, 101
solid-state disk, 156
sorting algorithms, 44
sound sample size, 89
source code, 110
SSD, 156
SSID, 172
stored program concept, 129
string conversion, 5
structured programming, 39
subroutines, 21
advantages of using, 25
user-written, 22
with interfaces, 23
substitution cipher, 96
synchronous transmission, 162
syntax error, 111
system
bus, 127
clock, 132

T
tebi, 72
test plan, 48
text file, 29
topology

U
Unicode, 74
user-defined data type, 29
user generated content, 181
user interface, 100
utility software, 101

V
variables, 6, 24
vector graphics, 85
Vernam cipher, 96
virtual memory, 104
virus checker, 101
von Neumann, 100
machine, 129

W
WAP, 171
WHILE … ENDWHILE, 13
whitelist, 172
Wi-Fi, 171
Protected Access, 172
Wilkes, Maurice, 100
WinZip, 101
wireless network
access point, 171
interface controller, 171
word, 128
word length, 135
WPA, 172

X-Z
XOR, 11, 144
gate, 116
yobi, 72
zebi, 72
The aim of this textbook is to provide a detailed understanding of each topic in the new AQA AS Level Computer Science specification. It is presented in an accessible and interesting way, with many in-text questions to test students’ understanding of the material and ability to apply it.

The book is divided into six sections, each containing six chapters. Each chapter covers material that can comfortably be taught in one or two lessons. It will also be a useful reference and revision guide for students throughout the AS and A Level courses.

About the authors

Pat Heathcote is a well-known and successful author of Computing textbooks. She has spent many years as a teacher of A Level Computing courses with significant examining experience. She has also worked as a programmer and systems analyst, and was Managing Director of Payne-Gallway Publishers until 2005.

Rob Heathcote has many years of experience teaching Computer Science and is the author of several popular textbooks on Computing. He is now Managing Director of PG Online, and writes and edits a substantial number of the online teaching materials published by the company.

Cover picture:

‘Golden Moor’
Oil on canvas, 40x40cm
© Heather Duncan
www.heatherduncan.com

This book has been approved by AQA.