OXFORD AQA
INTERNATIONAL QUALIFICATIONS

International AS and A-level

Computer Science

(9645) Specification

For teaching from September 2024 onwards

For International AS exams
May/June 2025 onwards

For International A-level exams
May/June 2026 onwards

For teaching and examination outside the United Kingdom

Contents

Are you using the latest version of this specification?

- You will always find the most up-to-date version of this specification on our website at **oxfordaqa.com/9645**
- We will write to you if there are significant changes to the specification.

1 Introduction

1.1 Why choose OxfordAQA International AS and A-levels?

Our international qualifications enable schools that follow a British curriculum to benefit from the best education expertise in the United Kingdom (UK).

Our International AS and A-levels offer the same rigour and high quality as AS and A-levels in the UK and are relevant and appealing to students worldwide. They reflect a deep understanding of the needs of teachers and schools around the globe and are brought to you by Oxford University Press and AQA, the UK's leading awarding body.

Providing valid and reliable assessments, these qualifications are based on over 100 years of experience, academic research and international best practice. They reflect the latest changes to the British system, enabling students to progress to higher education with up-to-date qualifications

You can find out about OxfordAQA at **oxfordaqa.com**

1.2 Why choose our International AS and A-level Computer Science?

We have worked closely with teachers to develop a relevant, engaging and up-to-date computer science specification to inspire, motivate and challenge all students regardless of their academic ability.

Particular care has been taken to make the language used in question papers as accessible as possible and suitable for those students for whom English is not their first language. UK English spellings will be used in examination papers. British idiosyncratic terms however, will be avoided to aid students' understanding.

Advances in computing are transforming the way we work and our Computer Science specification reflects these changes. This flexible, accessible and rigorous qualification, is backed by top quality support, resources and professional development. This up-to-date specification focuses on the knowledge, understanding and skills students need to progress to higher education or thrive in the workplace.

You can find out about all our International AS and A-level Computer Science qualification at **oxfordaqa.com/9645**

1.3 Recognition

OxfordAQA meet the needs of international students. They are an international alternative and comparable in standard to the Ofqual regulated qualifications offered in the UK. Our qualifications have been independently benchmarked by UK ENIC, the UK national agency for providing expert opinion on qualifications worldwide. They have confirmed they can be considered 'comparable to the overall GCE A-level and GCSE standard offered in the UK'.

To read their report and see the latest list of universities who have stated they accept these international qualifications, visit **oxfordaqa.com/recognition**

1.4 The Oxford International Programme learner attributes

In order to equip students with the skills they need for success both now and in the future, we have worked with Oxford University Press to create the Oxford International Programme. This combines the Oxford International Curriculum with OxfordAQA qualifications, creating an integrated offer for international schools, from Early Years to A-level.

At its core we have introduced the Oxford International Programme learner attributes – the skills and competencies that enable our students to thrive academically, socially and personally.

The learner attributes, alongside our focus on demonstrating higher order critical thinking skills, ensure that students are equipped to get the grades that will take them places, and build the skills they need to be successful when they get there.

Empowered & independent

Our students are independent, critical thinkers who are adaptable and look to develop strategies to be lifelong learners. They are confident leading on projects but also work well in a collaborative environment.

Inventive & curious

Our students are inventive, resourceful, and creative. They question the world around them with a sense of wonder, and aspire to shape a better future for themselves and their community.

Future-ready

Our students are more prepared to succeed in the world that lies ahead and have the knowledge, skills, and drive to achieve any objective they may set themselves. They are comfortable being challenged, acquiring new skills quickly, and seeking new adventures.

Ambitious & self-motivated

Our students are ambitious and want to strive for success in every aspect of their lives. They take the initiative, approaching every task with an eagerness to learn and take ownership of their own learning with the utmost integrity.

1.5 Support and resources to help you teach

We know that support and resources are vital for your teaching and that you have limited time to find or develop good quality materials.

That's why we've worked with experienced teachers to provide resources that will help you confidently plan, teach and prepare for exams.

Teaching resources

You will have access to:

- sample schemes of work to help you plan your course with confidence
- training and support to help you deliver our qualifications
- student textbooks that have been checked and approved by us
- command words with exemplars
- computer science vocabulary with definitions.

Preparing for exams

You will have access to the support you need to prepare for our exams, including:

- specimen papers and mark schemes
- exemplar student answers with examiner commentaries.

Analyse your students' results with Enhanced Results Analysis (ERA)

After the first examination series, you can use this tool to see which questions were the most challenging, how the results compare to previous years and where your students need to improve. ERA, our free online results analysis tool, will help you see where to focus your teaching.

Information about results, including maintaining standards over time, grade boundaries and our post-results services, will be available on our website in preparation for the first examination series.

Help and support

Visit our website for information, guidance, support and resources at **oxfordaqa.com/9645**

You can contact the subject team directly at **info@oxfordaqa.com** or
call us on +44 (0)161 696 5995 (option 1 and then 1 again).

Please note: We aim to respond to all email enquiries within two working days.

Our UK office hours are Monday to Friday, 8am – 5pm.

2 Specification at a glance

2.1 Subject content

AS

1. Procedural programming
2. Fundamental data structures
3. Program design
4. Searching and sorting algorithms
5. Representing data
6. Computer systems
7. Computer organisation and architecture
8. Machine code and assembly language

A-level

9. Object-oriented and additional programming
10. Advanced data structures
11. Advanced algorithms
12. Functional programming
13. Theory of computation
14. Networking and cyber security
15. Databases
16. Artificial intelligence

2.2 Assessments

AS

Unit 1: Programming

What's assessed

This paper tests a student's ability to program, as well as their knowledge, skills and understanding of sections 3.1 to 3.4 of the specification:

- Procedural programming
- Fundamental data structures
- Program design
- Searching and sorting algorithms

How it's assessed

- On-screen exam available in C#, Python or VB.Net
- 2 hours
- 75 marks
- 50% of AS
- 20% of A-level

Questions

Students answer a series of short questions and write and test programs, providing their responses in an Electronic Answer Document (EAD) provided by OxfordAQA.

+

Unit 2: Concepts and principles of computer science

What's assessed

This paper tests a student's knowledge, skills and understanding of sections 3.5 to 3.8 of the specification:

- Representing data
- Computer systems
- Computer organisation and architecture
- Machine code and assembly language

How it's assessed

- Written exam
- 1 hour 30 minutes
- 75 marks
- 50% of AS
- 20% of A-level

Questions

A series of short-answer and extended-answer questions.

A-level

Unit 3: Advanced Programming

What's assessed

This paper tests a student's ability to program using both procedural and object-oriented programming, as well as their knowledge, skills and understanding of sections 3.9 to 3.11 of the specification:

- Object-oriented and additional programming
- Advanced data structures
- Advanced algorithms

Programming is a synoptic activity, so questions will also require students to be familiar with the content of sections 3.1 to 3.4 of the AS specification.

How it's assessed

- On-screen exam available in C#, Python or VB.Net
- 2 hours 30 minutes
- 90 marks
- 30% of A-level

Questions

Students answer a series of short questions and write and test programs, providing their responses in an Electronic Answer Document (EAD) provided by OxfordAQA.

+

Unit 4: Advanced concepts and principles of computer science

What's assessed

This paper tests a student's knowledge, skills and understanding of sections 3.12 to 3.16 of the specification:

- Functional programming
- Theory of computation
- Networking and cyber security
- Databases
- Artificial intelligence

How it's assessed

- Written exam
- 1 hour 30 minutes
- 75 marks
- 30% of A-level

Questions

A series of short-answer and extended-answer questions.

3 Subject content

3.1 Procedural programming

3.1.1 Data types

Content	Additional information
Understand the concept of a data type.	
Understand and use the following data types appropriately: • integer • real/float • Boolean • character • string • date/time.	Students using Python could use Python type hints to help them become more familiar with the concept of data types, but this is not a requirement.

3.1.2 Programming concepts

Content	Additional information
Use, understand and know how the following statement types can be combined in programs: • variable declaration • assignment • selection • iteration • subroutine.	Students will not need to use the terms function and procedure to distinguish between subroutines that do and do not return values.
Use definite and indefinite iteration.	
Use nested selection and nested iteration structures.	
Use meaningful identifier names and know why it is important to use them.	
Be able to display output and obtain input from the user.	
Understand the importance of adding comments to explain how code works and be able to read comments in code and add comments to code.	

3.1.2.1 Arithmetic operations in a programming language

Content	Additional information
Be familiar with and be able to use: ● addition ● subtraction ● multiplication ● real/float division ● integer division, including remainder (DIV and MOD) ● exponentiation (power) ● rounding ● truncation.	

3.1.2.2 Relational operations in a programming language

Content	Additional information
Be familiar with and be able to use: ● equal to ● not equal to ● less than ● greater than ● less than or equal to ● greater than or equal to.	

3.1.2.3 Boolean operations in a programming language

Content	Additional information
Be familiar with and be able to use: ● NOT ● AND ● OR ● XOR. Know that the order of precedence between NOT, AND and OR operators is: ● NOT – highest ● AND ● OR – lowest.	Students will not be required to know the order of precedence of XOR, which varies between languages and systems.

3.1.2.4 String-handling operations in a programming language

Content	Additional information
Be familiar with and be able to use: ● length ● position ● substring ● concatenation ● character → character code ● character code → character ● string conversion operations. Required string conversion operations: ● string → integer ● string → float ● integer → string ● float → string ● date/time → string ● string → date/time.	

3.1.2.5 Random number generation in a programming language

Content	Additional information
Be familiar with, and be able to use, random number generation.	

3.1.2.6 Exception handling

Content	Additional information
Be familiar with, and be able to use, exception handling.	

3.1.2.7 Subroutines

Content	Additional information
Be familiar with subroutines and their uses.	
Know that a subroutine is a named 'out of line' block of code that may be executed (called) by simply writing its name in a program statement.	
Be able to explain the advantages of using subroutines in programs.	
Be able to describe the use of parameters to pass data within programs.	Students will not be required to distinguish between passing parameters by value or reference, but may use these techniques if they wish to and they are supported by the programming language that the student is using.
Be able to use subroutines that return values to the calling routine.	Students should know how to return more than one value from a subroutine. In C# and VB.Net this can be achieved using tuples.
Know that variables that are accessible throughout an entire program are known as global variables.	
Know that subroutines may declare their own variables, called local variables, and that local variables: • exist only while the subroutine is executing • are accessible only within the subroutine.	Students using C# or VB.Net should also be aware of block scope variables. Block scope variables only exist inside the block of code that they are declared within. Typical blocks of code would include selection and iteration structures. Questions will not be asked about block scope, but students may wish to use this scope when programming.
Know that it is good practice to limit the scope of a variable where possible.	In Python, local variables should be used in preference to global variables. In C# and VB.Net, block scope or local variables should be used in preference to global variables.
Be able to contrast local variables with global variables.	
Be able to explain how a stack frame is used with subroutine calls to store: • the return address • parameters • local variables.	

3.2 Fundamental data structures

Content	Additional information
Be familiar with the concept of data structures.	
Be able to distinguish between static and dynamic data structures and compare their uses, as well as explaining the advantages and disadvantages of each.	

3.2.1 Arrays and lists

Content	Additional information
Be able to use arrays and lists when solving problems, including the use of two-dimensional arrays and lists of lists.	Python does not contain built-in support for arrays, but students can use arrays, for example, by importing the `array` (currently does not support string elements or multi-dimensional arrays) or `numpy` modules. In programming exams, it will be acceptable to use either lists or arrays to solve a problem unless the question states that a list must be used or an array must be used. Students will not be required to use arrays of more than two dimensions in exam questions, but may do so if they wish to.

3.2.2 Records

Content	Additional information

Be familiar with the composition of a group of values (known as fields) into a record which can be manipulated as a single entity.	In Python, classes can be used in a non-object-oriented way to create records simply. For example:
	
	`class Coordinate():`
	`  def __init__(self):`
	`    self.x = 0`
	`    self.y = 0`
	
	`myposition = Coordinate()`
	`myposition.x = 10`
	`myposition.y = 5`
	
	Alternatively, data classes can be used in Python 3.7 onwards. For example:
	
	`from dataclasses import dataclass`
	
	`@dataclass`
	`class Coordinate:`
	`    x: int = 0`
	`    y: int = 0`
	
	In C# and Visual Basic, `struct` and `Structure` can be used to create records.

3.2.3 Queues

Content	Additional information
Understand that a queue is a first-in first-out (FIFO) data structure.	
Be able apply the following operations to data stored in a queue: ● add an item (enqueue) ● remove an item (dequeue).	
Be able to describe situations in which a queue is an appropriate data structure to use.	
Understand how to implement a linear queue and a circular queue using a one-dimensional array. For implementations in a one-dimensional array, understand how to test if the queue is empty or full.	
Have experience of using a programming language library that implements the queue data structure.	

3.2.4 Stacks

Content	Additional information
Understand that a stack is a last-in first-out (LIFO) data structure.	
Be able apply the following operations to data stored in a stack: ● push ● pop ● peek (also known as top).	Peek or top returns the value of the top (most recently pushed) element without removing it.
Be able to describe situations in which a stack is an appropriate data structure to use.	
Understand how to implement a stack using a one-dimensional array. For implementations in a one-dimensional array, understand how to test if the stack is empty or full.	
Have experience of using a programming language library that implements the stack data structure.	

3.3 Program design

3.3.1 Structured approach to programming

Content	Additional information
Understand the structured approach to program design and construction.	Students should be able to describe the structured approach including: • modularised programming • the use of parameters and return values • the use of local or (if supported by language) block scope variables.
Be able to explain the advantages of the structured approach.	
Be able to construct and use hierarchy charts and structure charts when designing programs.	

3.3.2 Abstraction and decomposition

Content	Additional information
Be familiar with the concept of abstraction.	Abstraction is removing unnecessary details from a problem to facilitate its solution.
Be familiar with the concept of decomposition.	Decomposition is breaking a problem into a number of sub-problems, so that each sub-problem accomplishes an identifiable task, which might itself be further subdivided. Decomposition can be achieved through the use of subroutines.

3.3.3 Following and writing algorithms

Content	Additional information
Understand the term algorithm.	An algorithm is a sequence of steps that can be followed to complete a task and that always terminates.
Be familiar with the use of pseudocode to express algorithms.	Students will be expected to be able to understand algorithms written in pseudocode but will not be expected to write pseudocode in the exam.
Be able to convert an algorithm from pseudocode into high-level language program code.	
Be able to hand-trace algorithms.	

3.3.4 Aspects of software development

Content	Additional information
Analysis	Be aware that before a problem can be solved, it must be defined, the requirements of the system that solves the problem must be established and a data model created. Requirements of the system must be established by interaction with the intended users of the system. The process of clarifying requirements may involve prototyping/agile approach.
Design	Be aware that before constructing a solution, the solution should be designed and specified, for example planning data structures for the data model, designing algorithms, designing an appropriate modular structure for the solution and designing the human user interface. Be aware that design can be an iterative process involving a prototyping/agile approach.
Implementation	Be aware that the models and algorithms need to be implemented in the form of data structures and code (instructions) that a computer can process. Be aware that the final solution may be arrived at using an iterative process employing prototyping/an agile approach with a focus on solving the critical path first. The critical path consists of the part of a solution that everything else depends upon.
Testing	Be aware that the implementation must be tested for the presence of errors, using selected test data covering normal (typical), boundary and erroneous data. Students should have practical experience of designing and applying test data, normal, boundary and erroneous to the testing of programs so that they are familiar with these test data types and the purpose of testing.
Evaluation	Know the criteria for evaluating a computer system, including: ● correctness ● efficiency ● maintainability.

3.4 Searching and sorting algorithms

3.4.1 Searching algorithms

Content	Additional information
Know the linear search algorithm and be able to trace it.	Students may be asked to write code to implement a linear search algorithm in the exam. Students should know that an inefficient version of the linear search algorithm will always look at every item in a list, whilst a more advanced version will stop as soon as the item is found.
Know the binary search algorithm and be able to trace it.	Students will need to understand how the binary search algorithm operates and may be asked to write code to implement a binary search algorithm in the exam.
Compare the linear search and binary search algorithms.	Students should be able to compare the algorithms in terms of: • their time efficiency. • the conditions under which they can be used (if the array/list has to be ordered or not). Formal comparisons using big-O notation will not be required at AS, but may be required at A-level.

3.4.2 Sorting algorithms

Content	Additional information
Know the bubble sort algorithm and be able to trace it.	This is included as an example of a particularly inefficient sorting algorithm. Students will be expected to know that a number of versions of the algorithm exists and that the most basic version that uses definite iteration can be made more efficient by: • reducing the number of items checked by one after each pass through a list • using indefinite iteration so that the sort process stops when one pass has been made through the list without any swaps being made. Students may be asked to write code to implement a bubble sort algorithm in the exam.
Know the merge sort algorithm and be able to demonstrate how it operates.	Students will not be asked to write code to implement the merge sort algorithm in the exam but may be asked to explain or demonstrate how the algorithm works using a set of data.
Compare the efficiency of the bubble sort and merge sort algorithms. Understand that the time-efficiency of the bubble sort algorithm depends upon the extent to which the list is sorted at the start.	Students should be able to compare the algorithms in terms of their efficiency of use of time and memory. Formal comparisons using Big O notation will not be required at AS but may be required at A-level. Students should be able to explain that an efficient version of the bubble sort algorithm will terminate more quickly if the list is already partially sorted.

3.5 Representing data

3.5.1 Number bases

Content	Additional information
Be familiar with the concept of a number base, in particular: ● decimal (base 10) ● binary (base 2) ● hexadecimal (base 16).	
Convert between decimal, binary and hexadecimal number bases.	Number base conversions will not be examined independently, but students may be required to be able to convert between these number bases whilst answering questions on other topics, such as floating point numbers and assembly language.

3.5.2 Units of information

Content	Additional information
Know that: ● the bit is the fundamental unit of information ● a byte is a group of 8 bits.	
Know that 2^n different values can be represented with n bits.	For example, 3 bits can be configured in $2^3 = 8$ different ways: 000, 001, 010, 011, 100, 101, 110, 111
Know that quantities of bytes can be described using binary prefixes representing powers of 2 or using decimal prefixes representing powers of 10, eg one kibibyte is written as 1KiB $= 2^{10}$ B and one kilobyte is written as 1kB $= 10^3$ B.	The terms kibibyte, mebibyte, etc have replaced the historic meanings of the terms kilobyte, megabyte etc.
Know the names, symbols and corresponding powers of 2 for the binary prefixes: ● kibi, Ki - 2^{10} ● mebi, Mi - 2^{20} ● gibi, Gi - 2^{30} ● tebi, Ti - 2^{40}	
Know the names, symbols and corresponding powers of 10 for the decimal prefixes: ● kilo, k - 10^3 ● mega, M - 10^6 ● giga, G - 10^9 ● tera, T - 10^{12}	

3.5.3 The binary number system

3.5.3.1 Unsigned binary

Content	Additional information
Know the difference between unsigned binary and signed binary.	
Know that in unsigned binary the minimum and maximum values for a given number of bits, n, are 0 and $2^n - 1$ respectively.	

3.5.3.2 Unsigned binary arithmetic

Content	Additional information
Be able to: ● add two unsigned binary integers ● multiply two unsigned binary integers.	

3.5.3.3 Signed binary using two's complement

Content	Additional information
Know that signed binary can be used to represent negative integers and that one possible coding scheme is two's complement.	Two's complement is the only representation of negative integers that will be examined. Students are expected to be able to convert between signed binary and decimal and vice versa.
Know how to: ● represent negative and positive integers in two's complement ● perform subtraction using two's complement ● calculate the range of a given number of bits, n.	The range of n bits is -2^{n-1} to $+2^{n-1} - 1$

3.5.3.4 Numbers with a fractional part

Content	Additional information
Know how numbers with a fractional part can be represented in: • fixed point form in binary in a given number of bits and format • floating point form in binary in a given number of bits and format.	Students are not required to know the Institute of Electrical and Electronic Engineers (IEEE) standard, only to know, understand and be able to use a simplified floating representation consisting of a mantissa and an exponent.
For both fixed point and floating point be able to convert: • from decimal to binary using a given number of bits and format • from binary to decimal using a given number of bits and format.	Exam questions on floating point numbers will use a format in which both the mantissa and exponent are represented using two's complement.
Understand that in floating point the number of bits in the mantissa determines the precision with which numbers can be represented and the number of bits in the exponent determines the range of numbers that can be represented.	

3.5.3.5 Rounding errors

Content	Additional information
Know and be able to explain why both fixed point and floating point representation of decimal numbers may be inaccurate.	For a real number to be represented exactly by the binary number system, it must be capable of being represented by a binary fraction in the given number of bits and format. Some values cannot ever be represented exactly, for example the decimal number 0.1.

3.5.3.6 Absolute and relative errors

Content	Additional information
Be able to calculate the absolute error of numerical data stored and processed in computer systems.	
Be able to calculate the relative error of numerical data stored and processed in computer systems.	
Be able to explain why relative error is a more useful measure than absolute error.	

3.5.3.7 Underflow and overflow

Content	Additional information
Explain overflow and underflow and describe the circumstances in which they occur.	Overflow occurs when the result of a calculation is so large that it cannot be represented in the available number of bits. Underflow occurs when the result of a calculation is so close to zero that, in a given number of bits, its closest representation is zero.

3.5.3.8 Compare fixed and floating point

Content	Additional information
Compare the advantages and disadvantages of fixed point and floating point representations in terms of: ● range ● precision ● speed of calculation.	

3.5.3.9 Normalisation of floating point form

Content	Additional information
Know why floating point numbers are normalised and be able to normalise un-normalised floating point numbers with positive or negative mantissas.	

3.5.4 Representing characters

Content	Additional information
Understand that characters are represented using a character set, in which each character is assigned a unique code. Describe ASCII and Unicode coding systems for coding character data. Explain why Unicode was introduced.	Codes are represented as numbers and stored as bit patterns.
Know that ASCII is a 7-bit encoding system.	ASCII can represent $2^7 = 128$ characters.
Know that Unicode has more than one encoding system. Know that UTF-8: ● is the most widely used Unicode encoding ● is a variable-length encoding system where 8, 16, 24 or 32 bits are used to represent a character ● was designed for backwards compatibility with ASCII and that in UTF-8 and ASCII the numbers 0 to 127 represent the same characters.	
Know that in both Unicode and ASCII characters are grouped into blocks: ● numeric digits start at code 48 ● English uppercase letters start at code 65 ● English lowercase letters start at code 97.	Students will not be expected to remember the Unicode or ASCII values of particular characters in the exam, but may be asked to work out the code of another character in a block if given the code of the first character in the block. For example, if told the ASCII code of the character A, a student could be expected to work out the ASCII code of the character G.
Understand the difference between the character code representation of a decimal digit and its pure binary representation.	For example, the pure binary representation of the digit 6 is 110 but its representation as a character code in ASCII in binary is 0110110 and in UTF-8 it is 00110110.

3.5.5 Representing graphics

3.5.5.1 Bitmapped graphics

Content	Additional information
Describe how bitmaps represent images.	
Understand the following for bitmaps: • size in pixels • colour depth.	The size of an image is calculated as the width of an image in pixels multiplied by the height of the image in pixels using the notation: width in pixels x height in pixels. Colour depth is the number of bits used to represent the colour of a pixel.
Calculate storage requirements for bitmapped images, excluding metadata. Be aware that bitmap image files may also contain metadata.	Excluding metadata: storage requirements = size in pixels x colour depth where size in pixels is width in pixels x height in pixels.
Be familiar with typical metadata.	Examples: width, height, colour depth.

3.5.5.2 Vector graphics

Content	Additional information
Explain how vector graphics represents images using lists of objects.	The properties of each geometric object/shape in the vector graphic image are stored as a list.
Give examples of typical properties of objects.	Typical properties include: • x and y coordinates of top left corner of object • colour of object outline • line width of object outline • colour of object fill.

3.5.5.3 Comparing bitmapped and vector graphics

Content	Additional information
Compare the vector graphics approach with the bitmapped graphics approach and understand the advantages and disadvantages of each.	
Be aware of appropriate uses of the vector graphics and bitmapped graphics approaches.	

3.5.6 Representing sound

3.5.6.1 Analogue and digital

Content	Additional information
Understand the difference between analogue and digital quantities.	
Describe the principles of operation of: ● an analogue to digital converter (ADC) ● a digital to analogue converter (DAC).	
Be able to describe the use of an ADC in the recording of analogue audio and the use of a DAC in the playback of digital audio.	

3.5.6.2 Digital representation of sound and Nyquist's theorem

Content	Additional information
Describe the digital representation of sound in terms of: ● sample resolution ● sampling rate.	Sample resolution is the number of bits used to represent one sample. Sampling rate is the number of samples taken in a second and is usually measured in hertz. 1 hertz = 1 event/cycle per second. In the context of sampling, 1 hertz = 1 sample taken per second.

Calculate sound sample sizes in bytes.	File size (bits) = rate (Hz) x res x secs
	rate = sampling rate (Hz)
	res = sample resolution
	secs = number of seconds.

Be able to describe Nyquist's theorem and use it to calculate the minimum sampling rate that should be used to sample a sound in order to be able to represent it accurately.	

3.5.7 Basic encryption methods

Content	Additional information
Understand what is meant by encryption and be able to define it.	Encryption is the use of an algorithm to convert plaintext into ciphertext so that it cannot easily be understood by a third party who does not know both the method used to encrypt the data and the encryption key. Exam questions will be primarily focussed on the Caesar and Vernam ciphers. If any other type of cipher is used in an exam question, it will be explained in the question and will be of similar conceptual difficulty.
Be familiar with the terms cipher, plaintext and ciphertext. Know what a key is.	A key is a value, such as a sequence of letters or bits, that is applied to plaintext by a cipher to convert the plaintext into ciphertext.
Be familiar with the Caesar cipher and be able to apply it to encrypt a plaintext message and decrypt a ciphertext message. Be able to explain why the Caesar cipher is easily cracked.	The Caesar cipher is easily cracked as: ● there are only 25 possible keys ● the frequency of letters in the ciphertext corresponds to the frequency of letters in the plaintext, making it susceptible to frequency analysis ● once the shift applied to one letter is identified, the shift applied to all letters is known.
Be familiar with the Vernam cipher (sometimes known as a one-time pad cipher) and be able to apply it to encrypt a plaintext message and decrypt a ciphertext message.	
Know what perfect security is. Know that the Vernam cipher can be perfectly secure. Describe the conditions that must be met for the Vernam cipher to be perfectly secure.	A cipher is perfectly secure if nothing can be learned about the plaintext from examining the ciphertext. For the Vernam cipher to be perfectly secure: ● the key must be chosen completely randomly ● the key must be at least as long as the plaintext ● the key must be used only once ● the key must be known only to the sender and receiver.
Compare the Caesar and Vernam ciphers.	Caesar and Vernam ciphers provide completely different levels of security: the Caesar cipher provides very little security, the Vernam cipher offers perfect security subject to conditions being met. Between these two types are ciphers that are computationally secure.
Know that most ciphers used by computers rely on computational security. Understand the concept of a computationally secure cipher. Compare Vernam cipher with ciphers that depend on computational security.	A computationally secure cipher is a cipher for which information about the plaintext can be learned from the ciphertext, but algorithms to crack the cipher take an unfeasible time to complete or have a very low probability of success.

Content	Additional information
Understand the key exchange problem.	The key exchange problem is the problem of sharing the key between the sender and receiver without it being at risk of being intercepted.
Know the difference between a symmetric and an asymmetric cipher.	A symmetric cipher uses the same key to encrypt and decrypt the data. Both the Caesar and Vernam ciphers are symmetric ciphers. An asymmetric cipher uses different mathematically related keys to encrypt and decrypt the data: since the encryption and decryption keys are different the key exchange problem does not apply.

3.5.8 Error detection and correction

Content	Additional information
Describe and explain the use of: ● parity bits ● majority voting ● checksums. Compare the use and effectiveness of these three methods.	

3.6 Computer systems

3.6.1 Hardware and software

Content	Additional information
Understand the relationship between hardware and software and be able to define the terms: ● hardware ● software.	Hardware is the electronic components of a computer system. Software is the sequences of instructions that are executed using the hardware.

3.6.2 Software

Content	Additional information
Explain what is meant by: ● system software ● application software.	
Understand the need for, and functions of the following system software: ● operating systems (OSs) ● utility programs ● libraries ● translators (compiler, assembler, interpreter).	For utilities, students will be expected to understand that utility programs add additional functionality to assist with the management of a computer system and to be able to give examples, such as a virus checker or compression program.
Understand that a role of the operating system is to hide the complexities of the hardware from the user and other software.	
Be able to describe the following functions of an operating system: ● scheduling ● memory allocation ● I/O device management ● interrupt handling.	Students only need to understand how main memory is allocated. Virtual memory does not need to be covered.

3.6.3 Programming languages and translation software

3.6.3.1 Classification of programming languages

Content	Additional information
Show awareness of the development of types of programming languages and their classification into low-level and high-level languages.	Students do not need to be familiar with the terms first generation, second generation and third generation.
Know that low-level languages are considered to be: ● machine code ● assembly language.	
Describe machine code and assembly language.	
Understand the advantages and disadvantages of machine code and assembly language programming compared with high-level language programming.	
Explain the term 'imperative high-level language'.	An imperative high-level language is a high-level language in which the commands describe the process that should be followed to carry out a task.

3.6.3.2 Types of program translator

Content	Additional information
Understand the role of each of the following: ● assembler ● compiler ● interpreter.	
Explain the differences between compilation and interpretation. Describe situations in which each would be appropriate.	
Explain why an intermediate language such as bytecode is produced as the final output by some compilers and how it is subsequently used.	Intermediate languages are used because: ● intermediate language code is more portable than machine code ● security checks can be performed on intermediate language code before it is executed ● intermediate language code can use less memory than equivalent machine code. The intermediate language code is used by either: ● a virtual machine, which will interpret the intermediate language code to execute it, or ● a just-in-time (JIT) compiler that converts the intermediate language code into machine code suitable for the computer it is being executed on.
Understand the difference between source code and object (executable) code.	

3.7 Computer organisation and architecture

3.7.1 Internal hardware components of a computer

Content	Additional information
Have an understanding and knowledge of the basic internal components of a computer system.	
Understand the role of the following components and how they relate to each other: ● processor ● main memory ● address bus ● data bus ● control bus ● I/O controllers.	
Understand the need for, and means of, communication between components. In particular, understand the concept of a bus and how address, data and control buses are used.	
Understand the concept of addressable memory.	Exam questions on this topic will state the size of each addressable memory location.
Be able to describe the stored program concept.	The stored program concept is that machine code instructions stored in main memory are fetched and executed serially by a processor that performs arithmetic and logical operations.
Be able to explain the difference between von Neumann and Harvard architectures and understand the advantages of each.	

3.7.2 The processor and its components

Content	Additional information
Explain the role and operation of a processor and its major components: ● arithmetic logic unit (ALU) ● control unit ● clock ● general-purpose registers ● dedicated registers, including: ● program counter (PC) ● current instruction register (CIR) ● memory address register (MAR) ● memory buffer register (MBR) ● status register (SR).	

3.7.3 The Fetch-Execute cycle and interrupts

Content	Additional information
Explain how the Fetch-Execute cycle is used to execute machine code programs including the stages in the cycle (fetch, decode, execute) and details of registers and buses used.	
Describe the role of interrupts and interrupt service routines (ISRs); their effect on the Fetch-Execute cycle; and the need to save the volatile environment while the interrupt is being serviced.	The volatile environment consists of the program counter, status register, stack pointer and any other registers that might be altered by the ISR. Students will not be required to explain how interrupts with different priorities are dealt with.

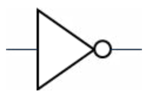
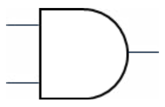
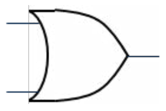
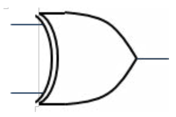
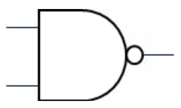
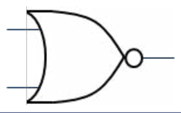
3.7.4 Factors affecting processor performance

Content	Additional information
Explain the effect on processor performance of: • multiple cores • cache memory • clock speed • word length • address bus width • data bus width.	

3.7.5 Secondary storage

Content	Additional information
Explain the need for secondary storage within a computer system.	Students should be able to distinguish between secondary storage and main memory.
Know the main characteristics, purposes, suitability and understand the principles of operation of the following devices: • magnetic hard disk • solid-state drive (SSD).	
Make comparisons between the use of magnetic hard disk and solid-state storage and make a judgement about their suitability for different applications.	
Explain the term cloud storage. Explain the advantages and disadvantages of cloud storage when compared to local storage.	Cloud storage is storage of data on servers at a remote location that is accessed via the Internet.

3.7.6 Logic gates

Content	Additional information
Construct truth tables for the following logic gates: ● NOT ● AND ● OR ● XOR ● NAND ● NOR	Students should know and be able to use the following symbols: NOT  AND  OR  XOR  NAND  NOR 
Be familiar with drawing and interpreting logic circuit diagrams involving one or more of the above gates.	
Complete a truth table for a given logic circuit.	
Write a Boolean expression for a given logic gate circuit.	The following notation will be used: ● NOT: ¯ ● AND: · ● OR: + ● XOR: ⊕
Draw an equivalent logic gate circuit for a given Boolean expression.	
Recognise and trace the logic of the circuits of a half-adder and a full-adder.	
Construct the circuit for a half-adder.	Students will not be required to construct the circuit for a full-adder.
Be familiar with the use of the edge-triggered D-type flip-flop as a memory unit.	Students should know that the D-type flip-flop has a clock and data input and that when the clock signal goes high the current state of the input is stored and will be output until the clock signal goes high again. Knowledge of the internal operation of the D-type flip-flop is not required. Students will not be required to draw circuits that use D-type flip-flops but may be asked to state the output of a simple circuit that includes D-type flip-flops.

3.7.7 Boolean algebra

Content	Additional information
Be familiar with the use of Boolean identities and De Morgan's laws to manipulate and simplify Boolean expressions.	The order of precedence of operators from highest to lowest is: • NOT: $\overline{}$ • AND: $\cdot$ • OR: $+$ Brackets can be used to change the order of evaluation. Students should know how the laws of distributivity, associativity and commutativity apply: • Commutativity: $A+B=B+A$ $A\cdot B=B\cdot A$ • Associativity: $A+(B+C)=(A+B)+C$ $A\cdot(B\cdot C)=(A\cdot B)\cdot C$ • Distributivity (AND distributes over OR): $A\cdot(B+C)=A\cdot B+A\cdot C$ Students should be able to apply the following Boolean identities: $\overline{\overline{A}}=A$ $A\cdot A=A$ $A+A=A$ $A\cdot\overline{A}=0$ $A+\overline{A}=1$ $A\cdot 0=0$ $A+0=A$ $A\cdot 1=A$ $A+1=1$ $A\cdot(A+B)=A$ $A+A\cdot B=A$ $A+\overline{A}\cdot B=A+B$ $A\oplus B= A\cdot\overline{B}+\overline{A}\cdot B$

	Students should be able to apply De Morgan's laws: $A+B=\overline{\overline{A}\cdot\overline{B}}$ $A\cdot B=\overline{\overline{A}+\overline{B}}$

3.8 Machine code and assembly language

3.8.1 Instruction format

Content	Additional information
Understand the term 'processor instruction set' and know that an instruction set is processor specific.	
Know that instructions consist of an opcode, an addressing mode and one or more operands (value, memory address or register). Know that the format of an instruction, in machine code or assembly language, may be dependent on the type of instruction.	Knowledge of the format of a particular instruction set is not required. For example, in a 32-bit ARM data processing instruction: ● The instruction is stored in 12 bits: ● 4 bits are used to represent the opcode ● 1 bit is used to represent the addressing mode for `<operand2>` ● the remaining bits are used to represent other information such as the condition under which the instruction should be executed. ● Up to three operands are stored in 20 bits: ● 2 x 4 bits are allocated to storing two register numbers ● 12 bits are allocated to storing `<operand2>`, which can be a register number or a value to process, depending on the addressing mode.
Understand and apply immediate, direct and indirect addressing modes.	Immediate addressing: the operand is the value to operate on. Direct addressing: the operand is the address from which the value to operate on should be fetched. Address can be interpreted as meaning either main memory address or register number. Indirect addressing: the operand is a register that contains the address in main memory from which the value to operate on should be fetched.
Know that in machine code instructions are expressed in binary and that in assembly language they are expressed as mnemonics.	

3.8.2 Assembly language programming

Content	Additional information
Understand and apply the basic operations of: • load • add • subtract • store • branching (conditional and unconditional) • compare • logical bitwise operators (AND, OR, NOT, XOR) • logical • shift right • shift left • halt.	Exam questions will use the OxfordAQA assembly language instruction set, which is based on the ARM processor instruction set. A copy of the instruction set will be included in any exam paper that includes an assembly language question and can be found in **6 Appendix: Standard OxfordAQA assembly language instruction set.**
Use the basic operations above to write, trace and reason about assembly language programs using immediate, direct and indirect addressing modes.	An exam question could require a student to write a program to meet a written requirement or to implement an algorithm expressed in pseudocode. A student should be able to convert pseudocode into assembly language and convert assembly language into pseudocode.

3.9 Object-oriented and additional programming

3.9.1 Classes, objects and instantiation

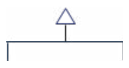
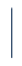
Content	Additional information
Know why the object-oriented paradigm is used.	
Be familiar with the concepts of: • class • property/attribute • method • object • instantiation • encapsulation • inheritance • overriding • association and be able to apply and identify these.	

Content	Additional information
Know what a class is and be able to create classes.	A class defines methods and property/attribute fields that capture the common behaviours and characteristics of objects.
Know what an object is and how to create objects using instantiation.	Objects based on a class are created using an explicit or implicit constructor.
Know that a constructor is called when an object is instantiated from a class and be able to create constructor functions. Know that a constructor initialises an object to a given state.	

3.9.2 Encapsulation

Content	Additional information
Know what encapsulation is and how it is used in object-oriented programming. Be able to design classes to use encapsulation appropriately.	Know that encapsulation is when the ways in which a class operates and how the class represents data are hidden from other classes.
Be able to use access modifiers to determine which classes can access properties and methods in a class.	Be able to use the public, private and protected access modifiers. Students using Python should be familiar with the naming conventions used to represent public, private and protected. Note that access modifiers are also known as access specifiers.
Use getter and setter methods to provide controlled access to data stored in a class.	

3.9.3 Relationships between classes

Content	Additional information
Know that inheritance is a relationship between classes in which one class is a more specialised version of an existing class. Be able to create classes using inheritance.	For example, a Car class could be a more specialised version of a Vehicle class.
Know that when inheritance is used there is a base class and a subclass. Be able to use the protected access modifier to determine which methods and properties of a base class can be accessed in subclasses without being accessible to objects in other classes.	Base class is also known as parent class. Subclass is also known as derived class. Properties and methods that are common to the base class and the subclass are defined in the base class but accessible to both the parent and base class. The protected access specifier is used in the base class to give a subclass access to specific methods and properties.
Know what overriding is. Be able to use overriding.	Know that overriding is when a method from the base class is redefined in a subclass so that an instance of the subclass can behave in a different way to an instance of the base class. Students using C# and VB.Net will need to declare a method as virtual so that it can be overridden in a subclass.
Know that association is a relationship between two objects in which one object can make use of another. Be able to use association.	Association is a much weaker relationship than inheritance. For example, in a football simulation, a Team object might be associated with Player objects. Students will not be required to distinguish between aggregation and composition.
Be familiar with the use of class diagrams to show inheritance and association relationships.	The following line types will be used to represent relationships. Inheritance:  Association:  The public access specifier will be indicated by the + symbol, the private access specifier by the - symbol and the protected access specifier by the # symbol.

3.9.4 Additional programming

3.9.4.1 Files

Content	Additional information
Be able to read/write from/to a text file.	

3.9.4.2 Recursion

Content	Additional information
Know that a recursive subroutine is a subroutine that calls itself.	
Know that recursive subroutines have a base and a recursive case.	The base case is the case in which the subroutine does not call itself. The recursive case is the case in which the subroutine calls itself. Recursive subroutines must have a base case to prevent the function recursing indefinitely. Note that there may be more than one base case and/or more than one recursive case in a subroutine.
Be able to solve simple problems using recursion.	Students should be able to read and write code that uses recursion and be able to trace recursive algorithms.

3.10 Advanced data structures

3.10.1 Graphs

Content	Additional information
Be aware of a graph as a data structure used to represent complex relationships.	
Be familiar with typical uses for graphs.	
Be able to explain the terms: • graph • weighted graph • vertex/node • edge/arc • undirected graph • directed graph.	
Know how an adjacency matrix and an adjacency list may be used to represent a graph.	Students should be familiar with the use of an adjacency matrix to represent both weighted and unweighted graphs.
Be able to compare the use of adjacency matrices and adjacency lists.	Adjacency matrices are more appropriate when: • edges must be added or removed frequently • the presence/absence of edges is tested frequently • a graph is dense, ie there are many edges relative to the number of vertices. Adjacency lists are more appropriate when a graph is sparse, ie there are few edges relative to the number of vertices.

3.10.2 Trees

Content	Additional information
Know that a tree is a connected, undirected graph with no cycles.	Note that a tree does not have to have a root.
Know that a rooted tree is a tree in which one vertex has been designated as the root. A rooted tree has parent-child relationships between nodes. The root is the only node with no parent and all other nodes are descendants of the root.	A class hierarchy in object-oriented programming is an example of a rooted tree, with `Object` as the root and all other classes descending for it.
Know that a binary tree is a rooted tree in which each node has at most two children.	
Understand how a binary tree can be used as a binary search tree.	

3.10.3 Hash tables

Content	Additional information
Be familiar with the concept of a hash table and its uses.	A hash table is a data structure that creates a mapping between keys and values.
Know what a hash function is. Be able to apply simple hash functions. Understand the properties of a good hash function.	A hash function uses a record's key to calculate a value which represents the position that the record is stored at in a hash table. Hash functions often use the MOD operator to limit the maximum size of the hash table by always returning a value in the range 0 .. hash table size - 1. Good hash functions generate an even distribution of records throughout the table and are quick to compute.
Know what is meant by a collision and how collisions are handled using rehashing.	A collision occurs when two key values compute the same hash. When a collision occurs, the second record, that should have been stored at the position occupied by the first record, is instead stored in the next available position.
Understand the advantages and disadvantages of using a hash table as an alternative to an array.	

3.10.4 Priority Queues

Content	Additional information
Know what a priority queue is.	
Be able apply the following operations to data stored in a priority queue: ● add an item (enqueue) ● remove an item (dequeue).	
Be able to describe situations in which a priority queue is an appropriate data structure to use.	Priority queues can be used to implement some graphing algorithms, such as a breadth-first search or Dijkstra's algorithm.
Understand how to implement a priority queue using a one-dimensional array. For implementations in a one-dimensional array, understand how to test if the priority queue is empty or full.	

3.10.5 Dictionaries

Content	Additional information
Be familiar with the concept of a dictionary.	A collection of key-value pairs in which the value is accessed via the associated key.
Be familiar with simple applications of dictionaries.	
Have experience of using a programming language library that implements the dictionary data structure.	

3.11 Advanced algorithms

3.11.1 Graph algorithms

Content	Additional information
Be able to trace breadth-first and depth-first search algorithms and describe typical applications of both.	Example applications are: ● breadth-first: shortest path for an unweighted graph ● depth-first: navigating a maze.
Be able to implement breadth-first and depth-first search algorithms in program code.	
Understand and be able to trace Dijkstra's shortest path algorithm. Be aware of applications of shortest path algorithm.	Students will not be expected to recall the steps in Dijkstra's shortest path algorithm or to write program code to implement it, unless the algorithm is given to them on a question paper.

3.11.2 Tree algorithms

Content	Additional information
Be able to trace the tree-traversal algorithms: ● pre-order ● post-order ● in-order.	
Understand how to implement pre-order, post-order and in-order tree-traversal algorithms.	
Be able to describe uses of tree-traversal algorithms.	Pre-order: copying a tree, producing a prefix expression from an expression tree. In-order: binary search tree, outputting the contents of a binary search tree in ascending order. Post-order: producing a postfix expression from an expression tree, emptying a tree.

3.12 Functional programming

3.12.1 The functional programming paradigm

Content	Additional information
Know that a function, f, has a function type A → B where A is the argument type and B is the result type. The name of a function and its type are expressed as f: A → B	Informally, a function is a rule that, for each element in some set A of inputs, assigns an output from set B, but without necessarily using every member of B. For example, f: {a,b,c,…,z} → {0,1,2,…,25} could use the rule that maps a to 0, b to 1, and so on, using all values which are members of set B.
Know that A is called the domain and B is called the co-domain.	The domain is a set from which the function's input values are chosen. The co-domain is a set from which the function's output values are chosen. Not all of the co-domain's members need to be outputs. For example, the function sqr: ℝ → ℝ will never output a negative value.

Be familiar with the natural, integer, rational and real number systems, which can be used as the domains and co-domains of functions.
 	
Be familiar with the concept of a natural number and the set ℕ of natural numbers (including zero).
 	
Be familiar with the concept of an integer and the set ℤ of integers.
 	
Be familiar with the concept of a rational number and the set ℚ of rational numbers, and that this set includes the integers.
 	
Be familiar with the concept of a real number and the set ℝ of real numbers.
	ℕ = {0, 1, 2, 3, … }
	
	ℤ = { …, -3, -2, -1, 0, 1, 2, 3, … }
	
	ℚ is the set of numbers that can be written as fractions (ratios of integers). Since a number such as 7 can be written as 7/1, all integers are rational numbers.
	
	ℝ is the set of all 'possible real world quantities'.

Know that a function is a first-class object in functional programming languages and in imperative programming languages that support such objects. Know that first-class objects (or values) are objects which may: ● appear in expressions ● be assigned to a variable ● be assigned as arguments ● be returned in function calls.	For example, integers, floating point values, characters and strings are first class objects in many programming languages.

Content	Additional information
Know that function application means a function applied to its arguments.	The process of giving particular inputs to a function is called function application, for example `add(3, 4)` represents the application of the function `add` to integer arguments 3 and 4. The type of this function is `add: integer x integer → integer` where `integer x integer` is the Cartesian product of the set integer with itself.

Know what is meant by composition of functions.	The operation functional composition combines two functions to get a new function.
	
	Given two functions:
	
	`f: A → B`
	
	`g: B → C`
	
	The function `g ○ f`, called the composition of g and `f`, is a function whose domain is A and co-domain is C and so has the type `A → C`.
	
	If the domain and co-domains of `f` and g are ℝ, and `f(x) = (x + 2)` and `g(y) = y`^{3}. Then:
	
	`g ○ f = (x + 2)`^{3}
	
	`f` is applied first and then g is applied to the result returned by `f`.

Understand what a higher-order function is.	A function is higher-order if it takes a function as an argument or returns a function as a result, or does both.

3.12.2 Writing functional programs

Content	Additional information
Show experience of constructing simple programs in a functional programming language.	Exam questions on functional programming will use the Haskell language. Functional programming techniques can also be used in C#, Python and VB.Net.
Know how to define and use a function with argument(s).	For example: `add a b = a + b`
Know how to apply the arithmetic operations of addition, subtraction, multiplication, division and power. Know how to perform Boolean comparisons and combine them.	^ is used for exponentiation (power). Students should be able to use ==, /= (not equal to), >, <, `&&` (and), `||` (or), `not`.

Have experience of defining functions using recursion, with both base and recursive case(s).
 	
Be familiar with the use of pattern matching to recognise values and determine which case of a recursive function to apply.
	For example:
	
	`fact 0 = 1`
	
	`fact n = n * fact n - 1`

Have experience of using the `map` higher-order function.	`map` is the name of a higher-order function that applies a given function to each element of a list, returning a list of results.
	
	For example:
	
	`square x = x * x`
	
	`numbers = [1, 2, 3, 4]`
	
	`map square numbers`
	
	produces the list:
	
	`[1,4,9,16]`

Have experience of using the `filter` higher-order function.	`filter` is the name of a higher-order function that processes a data structure, typically a list, in some order to produce a new data structure containing exactly those elements of the original data structure that match a given condition.
	
	For example:
	
	`filter (<10) [15, 8, 20, 3]`
	
	produces the list:
	
	`[8,3]`

Content	Additional information

Have experience of using a fold (also known as reduce) higher-order function.
 	
Know that there are two fold functions in Haskell: `foldl` starts with the leftmost item of a list and works forward, while `foldr` starts with the rightmost item and works backward.
	`foldl` and `foldr` are the names of higher-order functions which reduce a list of values to a single value by repeatedly applying a combining function, together with an initial value, to a list of values.
	
	For example:
	
	`numbers = [1, 2, 3, 4]`
	
	`foldl (+) 0 numbers`
	
	evaluates to `10` since ((((0 + 1) + 2) + 3) + 4) = 10.
	
	This is because it combines the list by repeatedly applying the addition function to it starting with 0 at the left.
	
	`foldr (+) 0 numbers`
	
	also evaluates to `10` since ((((4 + 0) + 3) + 2) + 1) = 10.
	
	However:
	
	`foldl (-) 0 [1, 2, 3, 4]`
	
	evaluates to `-10` since ((((0 – 1) – 2) – 3) – 4) = -10
	
	whilst:
	
	`foldr (-) 0 [1, 2, 3, 4]`
	
	evaluates to `-2` since ((((4 – 0) – 3) – 2) – 1) = -2

3.12.3 Lists in functional programming

Content	Additional information

Know that a list can store a sequence of values and that lists are represented using brackets.
 	
Know that a list can be empty.
 	
Know how to define a list.
	For example, `[7, 2, 1]` is a list containing the three numbers `7`, `2` and `1`.
	
	`[]` is the empty list.
	
	An example of defining a list:
	
	`listOne = [10, 7, 8]`
	
	`listTwo = ["Adam", "Ananya"]`

Be familiar with representing a list as a concatenation of a head and a tail. Know that the head is an element of a list and the tail is a list.	For example, in Haskell the list `[4, 3, 5]` can be written in the form `head:tail` where `head` is the first item in the list and `tail` is the remainder of the list. In the example, we have `4:[3, 5]`. We call 4 the head of the list and `[3, 5]` the tail. Note that the tail of a list is always a list, even if it only contains a single element, so should be written in brackets.

Describe and apply the following list operations:
 	
● return head of list
● return tail of list
● test for empty list
● return length of list
● construct an empty list
● append an item to a list.
 	
Have experience writing programs for the list operations mentioned above in Haskell.
	For example:
	
	`listOne = [1, 2, 3]`
	
	`listTwo = [4, 5]`
	
	`listThree = []`
	
	`head listOne` evaluates to 1
	
	`tail listOne` evaluates to `[2, 3]`
	
	`null listOne` evaluates to `False`
	
	`null listThree` evaluates to `True`
	
	`length listOne` evaluates to 3
	
	`listOne ++ listTwo` evaluates to `[1, 2, 3, 4, 5]`

Understand how the head and the tail of a list argument to a function can be referred to inside the function.	For example, if the function `sum` below was passed the argument `[8, 3, 2]` then `x` would reference the value 8 and `xs` would reference the list `[3, 2]`. `sum (x:xs) = x + sum (xs)`

3.13 Theory of computation

3.13.1 Finite state machines (FSMs)

Content	Additional information
Be able to draw and interpret simple state transition diagrams and state transition tables for FSMs with output and without output.	For FSMs with output, only Mealy machines are required, not Moore machines.

3.13.2 Regular expressions and regular languages

Content	Additional information
Be able to form and use simple regular expressions for string matching.	Students should be familiar with the metacharacters: ● * (0 or more repetitions) ● + (1 or more repetitions) ● ? (0 or 1 repetitions, ie optional) ● | (alternation, ie or) ● ( ) to group regular expressions. Students should also be able to use character classes, for example [CH] matches the character C or the character H and [A – Z] matches any character between A and Z, including A and Z. Any other metacharacters used in an exam question will be explained as part of the question.
Be able to describe the relationship between regular expressions and FSMs.	Regular expressions and FSMs are equivalent ways of defining a regular language.
Be able to: ● write a regular expression to recognise the same language as a given FSM ● design an FSM to recognise the same language as a given regular expression.	
Know that a language is called regular if it can be represented by a regular expression. Know that any regular language can be recognised by an FSM. Know that a regular expression can be written to describe the language recognised by any FSM.	

3.13.3 Turing machines

Content	Additional information
Know that a Turing machine can be viewed as a computer with a single fixed program, expressed using: ● a finite set of states in a state transition diagram ● a finite alphabet of symbols ● an infinite tape with marked-off squares ● a sensing read-write head that can travel along the tape, one square at a time. One of the states is called a start state and states that have no outgoing transitions are called halting states.	Exam questions will only be asked about Turing machines that have one tape that is infinite in one direction. No distinction will be made between the input alphabet and the tape alphabet.
Be able to: ● represent transition rules using a transition function ● represent transition rules using a state transition diagram ● hand-trace simple Turing machines.	
Understand the equivalence between a transition function and a state transition diagram.	
Be able to explain the importance of Turing machines and the Universal Turing machine (UTM) to the subject of computation.	Turing machines provide a (general/formal) model of computation and provide a definition of what is computable. The UTM is a Turing machine that can simulate the behaviour of any other Turing machine. A description of the Turing machine to simulate and the machine's input are stored on the UTM's tape.

3.13.4 Backus-Naur Form (BNF) and syntax diagrams

Content	Additional information
Be able to check language syntax by referring to BNF or syntax diagrams and formulate simple production rules.	
Know that BNF can represent some languages that regular expressions cannot represent.	For example, BNF can represent the following languages that cannot be represented by regular expressions: ● palindromes ● a language where strings must contain equal numbers of certain characters.

3.13.5 Classification of algorithms

3.13.5.1 Comparing algorithms

Content	Additional information
Understand that algorithms can be compared by expressing their complexity as a function relative to the size of the problem. Understand that the size of the problem is the key issue.	The size of the problem usually refers to the amount of data to be processed, for example the number of elements in a list that an algorithm is being applied to.
Understand that some algorithms are more efficient than others: ● time-wise ● space-wise.	Efficiently implementing algorithms means designing data models and algorithms to run quickly while taking up the minimal amount of resources such as memory.

3.13.5.2 Order of complexity

Content	Additional information
Be familiar with Big O notation to express time complexity and be able to apply it to cases where the running time requirements of the algorithm grow in: ● constant time – O(1) ● logarithmic time – O(log n) ● linear time – O(n) ● polynomial time – $O(n^a)$ ● exponential time – $O(a^n)$.	
Be able to derive the time complexity of a simple algorithm.	
Know that algorithms may have a best and a worst-case time complexity and that these may be different.	For example, bubble sort has a worst-case time complexity of $O(n^2)$ and a best-case time complexity of O(n).
Know the best and worst-case time complexity of the following algorithms: ● linear search ● binary search ● bubble sort ● merge sort ● Dijkstra's shortest path algorithm ● searching a binary search tree.	

3.13.5.3 Classification of algorithmic problems

Content	Additional information
Know that algorithms may be classified as being either: ● tractable – problems that have a polynomial (or less) time solution are called tractable problems ● intractable – problems that have no polynomial (or less) time solution are called intractable problems.	
Know that heuristic methods are often used when tackling intractable problems. Understand that a heuristic is a set of rules or knowledge about the problem domain and that a heuristic method might: ● find an approximate but non-optimal solution to a problem ● change some of the problem constraints.	Students will be expected to explain what a heuristic method is but do not need to be familiar with any specific heuristic techniques.

3.13.5.4 Computable and non-computable problems

Content	Additional information
Be aware that some problems cannot be solved algorithmically.	
Describe the Halting problem.	The Halting problem is the unsolvable problem of determining whether a program will eventually stop if given particular input, without executing the program. Students will not be required to prove that the Halting problem is unsolvable.
Understand the significance of the Halting problem for computation.	The Halting problem demonstrates that there are some problems that cannot be solved by a computer.

3.14 Networking and cyber security

3.14.1 Communications

3.14.1.1 Communication methods

Content	Additional information
Define serial and parallel transmission methods and discuss the advantages of serial over parallel transmission.	
Define and compare synchronous and asynchronous data transmission.	
Describe the purpose of start and stop bits in asynchronous data transmission.	A start bit brings the clock of the receiver into phase with the clock of the sender. A stop bit allows the next start bit to be recognised and in older devices gave the receiver time to deal with the received data.

3.14.1.2 Communication basics

Content	Additional information
Define: ● baud rate ● bit rate ● bandwidth ● latency ● protocol.	
Understand the difference between baud rate and bit rate.	Bit rate can be higher than baud rate if more than one bit is encoded in each signal change.
Understand the relationship between bit rate and bandwidth.	Bit rate is directly proportionate to bandwidth.

3.14.2 Networking

3.14.2.1 Types of networking between hosts

Content	Additional information
Explain the following and describe situations where they might be used: ● peer-to-peer networking ● client-server networking.	In a peer-to-peer network, each computer has equal status. In a client-server network, most computers are nominated as clients and one or more as servers. The clients request services from the servers, which provide these services, for example file server, email server.

3.14.2.2 Thin-client versus thick-client computing

Content	Additional information
Compare and contrast thin-client computing with thick-client computing. Understand the different hardware and networking requirements of thin-client and thick-client systems.	

3.14.2.3 Wired networking

Content	Additional information
Compare copper and fibre-optic cables.	Comparisons should be made in terms of cost, speed and capacity. Students should be able to select the most appropriate type of cable for a given situation.
Be able to describe how the protocol Carrier Sense Multiple Access with Collision Detection (CSMA/CD) works.	

3.14.2.4 Wireless networking

Content	Additional information
Explain the purpose of Wi-Fi.	A wireless local area network that is based on international standards. Used to enable devices to connect to a network wirelessly.
Be familiar with the purpose of a Service Set Identifier (SSID).	
Be familiar with the components required for wireless networking.	Wireless network adapter. Wireless access point. Note that in many home networks a single device combines the functions of a switch, router and wireless access point.
Be familiar with how wireless networks are secured.	Strong encryption of transmitted data using WPA2, SSID (Service Set Identifier) broadcast disabled, MAC (Media Access Control) address allow list.
Be able to describe how the wireless protocol Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with and without Request to Send/Clear to Send (RTS/CTS) works.	

3.14.2.5 Compare wired and wireless networking

Content	Additional information
Compare the advantages and disadvantages of wireless networking compared to wired networking.	

3.14.3 The Internet

Content	Additional information
Understand the structure of the Internet.	
Understand the role of packet switching and routers.	
Know the main components of a packet.	Including: • source address • destination address • packet sequence number • time to live • payload • error detection/correction information.
Explain how routing is achieved across the Internet.	
Describe the term 'uniform resource locator' (URL) in the context of internetworking.	
Explain the terms 'fully qualified domain name' (FQDN), 'domain name' and 'IP address'.	
Describe how domain names are organised.	
Understand the purpose and function of the domain service and its reliance on the Domain Name System (DNS). Be able to outline how the Domain Name System works using Domain Name Servers.	
Explain the service provided by Internet registries and why they are needed.	

3.14.4 The Transmission Control Protocol/Internet Protocol (TCP/IP) protocol

3.14.4.1 The TCP/IP model

Content	Additional information
Describe the role of the four layers of the TCP/IP stack (application, transport, internet, link).	
Understand why networking protocols like TCP/IP use layers.	
Describe the role of sockets in the TCP/IP stack.	
Be familiar with the role of MAC (Media Access Control) addresses.	

3.14.4.2 Standard application layer protocols

Content	Additional information
Be familiar with the following protocols: ● FTP (File Transfer Protocol) ● HTTP (Hypertext Transfer Protocol) ● HTTPS (Hypertext Transfer Protocol Secure) ● IMAP (Internet Message Access Protocol) ● POP3 (Post Office Protocol (v3)) ● SMTP (Simple Mail Transfer Protocol) ● SSH (Secure Shell).	
Be familiar with FTP client software and an FTP server, when transferring files using anonymous or authenticated access. Be aware that FTP is being replaced by alternative protocols such as SFTP that encrypt files for secure transmission.	
Be familiar with how SSH is used for remote management. Be familiar with using SSH to log in securely to a remote computer and execute commands.	
Describe the role of an email server when sending or receiving emails. Describe how SMTP and POP3 or IMAP protocols are used when an email is sent.	
Describe the role of a web server in serving up web pages in text form. Understand the role of a web browser in retrieving web pages and web page resources and rendering these accordingly.	

3.14.4.3 IP addresses

Content	Additional information
Know that an IP address is split into a network identifier part and a host identifier part.	
Know that networks can be divided into subnets and know how a subnet mask is used to identify the network identifier part of the IP address.	
Know that there are currently two standards of IP address, v4 and v6.	
Know why v6 was introduced.	
Distinguish between routable and non-routable IP addresses.	

3.14.4.4 Dynamic Host Configuration Protocol (DHCP)

Content	Additional information
Understand the purpose of the DHCP system.	
Be able to describe how the DHCP system works.	
Understand the advantages of the DHCP system over manual configuration.	

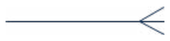
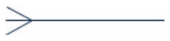
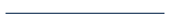
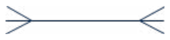
3.14.5 Cyber security

Content	Additional information
Understand how a firewall works (packet filtering, proxy server, stateful inspection).	
Understand symmetric and asymmetric (private/ public key) encryption and key exchange.	
Understand how digital certificates and digital signatures are obtained and used.	
Discuss worms, trojans and viruses, and the vulnerabilities that they exploit	

3.15 Databases

3.15.1 Relational databases

3.15.1.1 Conceptual data models and entity relationship modelling

Content	Additional information
Produce a data model from given data requirements for a simple scenario involving multiple entities.	
Produce entity relationship diagrams representing a data model and entity descriptions in the form: Entity1(Attribute1, Attribute2, ...).	Underlining can be used to identify the attribute(s) which form the entity identifier. The degree of relationships will be indicated as follows: One-to-many:  Many-to-one:  One-to-one:  Many-to-many: 

3.15.1.2 Key concepts

Content	Additional information
Explain the concept of a relational database.	
Be able to define the terms: ● attribute/field ● entity identifier/primary key ● composite entity identifier/composite primary key ● foreign key ● relation/table.	The terms attribute, entity identifier and relation are usually used in the context of abstract models of a database. The terms field, primary key and table are usually used in the context of implemented databases.

3.15.1.3 Database design and normalisation techniques

Content	Additional information
Be able to normalise relations to third normal form.	Students should know what properties are possessed by a relation in third normal form. Students will not be expected to differentiate between first, second and third normal forms.
Understand why databases are normalised.	

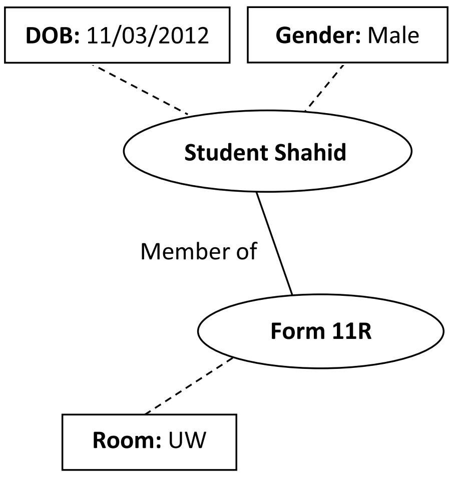
3.15.1.4 Structured Query Language (SQL)

Content	Additional information
Be able to use SQL to retrieve, update, insert and delete data from multiple tables of a relational database.	Know how to use the commands: ● `SELECT, FROM, WHERE, ORDER BY, GROUP BY` ● `UPDATE` ● `INSERT` ● `DELETE`
Be able to use aggregate SQL functions.	Know how to use the functions: ● `COUNT` ● `SUM` ● `AVG` ● `MIN` ● `MAX`
Be able to use SQL to define a database table.	Know how to create a table using data types for integer and real numbers, strings, Boolean values, dates and times. Know how to specify the primary key of a table. Know how to add a simple foreign key constraint using a single field.

3.15.1.5 Client server databases

Content	Additional information
Know that a client server database system provides simultaneous access to the database for multiple clients (known as concurrent access).	
Understand that concurrent updates of a database can result in the lost update problem.	If two clients edit a record simultaneously, the first saved update can be lost when it is overwritten by the second one.
Know how concurrent access can be controlled using record locks to preserve the integrity of a database.	

3.15.2 Big data

Content	Additional information
Know that 'Big Data' is a catch-all term for data that cannot be stored or processed using traditional methods. Big Data can be described in terms of: • volume – too big to fit into a single server • velocity – streaming data, milliseconds to seconds to respond • variety – data in many forms such as structured, unstructured, text, multimedia.	Whilst its size receives all the attention, the most difficult aspect of Big Data really involves its lack of structure. This lack of structure poses challenges because: • analysing the data is made significantly more difficult • relational databases are not appropriate because they require the data to fit into a row-and-column format.
Know that when data sizes are so big as not to fit on to a single server: • the processing must be distributed across more than one machine • functional programming is a solution, because it makes it easier to write correct and efficient distributed code.	Functional programming languages provide support for distributed processing through: • immutable data structures • statelessness • higher-order functions that can combine the results of processing on different servers (map-reduce).
Be familiar with the: • fact-based model for representing data • graph schema for capturing the structure of the dataset • nodes, edges and properties in graph schema.	Each fact within a fact-based model captures a single piece of information. In a graph-based schema: • An individual entity about which data are stored is represented as an oval node. • The properties of an entity are drawn in rectangles, which are attached to the oval for the entity by a dashed line. • Relationships between the entities are represented by solid line edges drawn between the them, labelled with text to describe the nature of the relationship. For example:  **DOB:** 11/03/2012 **Gender:** Male **Student Shahid** Member of **Form 11R** **Room:** UW

3.16 Artificial intelligence

3.16.1 Applications of artificial intelligence

Content	Additional information
Know the characteristics of artificial intelligence.	Students should be aware that there is no one accepted definition of artificial intelligence. Systems are often described as being artificially intelligent if they solve a complex problem and arrive at a solution: ● using a similar method to that which a human might follow, or ● that is at least as good as a human might arrive at. The problems solved by artificially intelligent systems are typically problems that historically have not been solvable by computer systems because they were believed to require human intelligence and could not be solved algorithmically using available computer hardware.
Be aware that current artificial intelligence systems are not generally intelligent, but work in narrow fields.	Generally intelligent systems are an area of research. Some people argue that modern deep learning systems are early versions of generally intelligent systems.
Be aware of the following common application areas for artificial intelligence: ● generative AI ● search and recommendation systems ● playing strategic games ● medical diagnosis.	

3.16.2 Creating artificially intelligent systems

Content	Additional information
Know what a neural network is and understand how neural networks can be used. Be able to describe the structure of a neural network. Know that backpropagation is commonly used to train a neural network.	A neural network is a network of nodes that are connected together in a similar way to neurons in the human brain. The nodes in a neural network are built up in layers. The outputs of nodes in one layer are weighted to form the inputs to nodes in the next layer. A simple neural network has three layers: an input layer, a hidden processing layer and an output layer.
Know that deep learning systems use neural networks with several hidden layers of nodes to produce output and that using more layers of nodes allows for more complex problems to be solved.	

Content	Additional information
Know that machine learning is a type of artificial intelligence in which the performance of the system is improved based on experience.	
Know that artificially intelligent systems are often trained using data and that care must be taken when selecting the training data to ensure that a system does not develop bias.	

3.16.3 Benefits and risks of artificial intelligence

Content	Additional information
Understand the benefits associated with the use of artificial intelligence.	Example benefits include: ● improved and more consistent decision making. For example, more accurate analysis of medical data ● ability to analyse very large data sets quickly ● continuous availability ● lower cost of operation ● can be available to more people and in areas that human expertise is not readily available.
Understand the risks associated with the use of artificial intelligence.	Students should consider the following risks arising from the use of artificial intelligence: ● elimination of jobs and the social impact of this ● bias in decision making, which could be based on characteristics such as gender or race ● false information can be output if the data used to train a system is incorrect ● plagiarism ● use in surveillance systems. ● risk to human existence from superintelligent systems.

4. Scheme of assessment

Find mark schemes, and specimen papers for new courses, on our website at: **oxfordaqa.com/computer-science**

These qualifications are modular. The full International A-level is intended to be taken over two years. The specification content for the International AS is half that of an International A-level.

The International AS can be taken as a stand-alone qualification or it can count towards the International A-level.

To complete the International A-level, students can take the International AS in their first year and the International A2 in their second year or they can take all the units together in the same examination series at the end of the two year course.

The International AS content will be 50% of the International A-level content. International AS assessments contribute 40% of the total marks for the full International A-level qualification. The remaining 60% comes from the International A2 assessments.

All materials are available in English only.

4.1 Availability of assessment units and certification

Exams and certification for this specification are available as follows:

	Availability of units		Availability of certification	
	International AS	**International A2**	**International AS**	**International A-level**
June 2025	✓		✓	
January 2026	✓		✓	
June 2026	✓	✓	✓	✓
January 2027 onwards	✓	✓	✓	✓
June 2027 onwards	✓	✓	✓	✓

4.2 Aims and learning outcomes

This specification encourages students to develop a broad range of the knowledge, understanding and skills of Computer Science, as a basis for progression into further learning and/or employment.

The specification encourages students to develop:

- an understanding of, and the ability to apply, the fundamental principles and concepts of computer science, including abstraction, decomposition, algorithms and data representation
- the ability to analyse problems in computational terms through practical experience of solving such problems by writing programs to do so
- the capacity for thinking creatively, innovatively, analytically, logically and critically
- the capacity to see relationships between different aspects of computer science
- mathematical skills related to computer science.

4.3 Assessment Objectives

The exams will measure how students have achieved the following assessment objectives.

- AO1: Demonstrate knowledge and understanding of the key concepts and principles of computer science.
- AO2: Apply knowledge and understanding of key concepts and principles of computer science.
- AO3: Analyse problems in computational terms in order to develop and test programmed solutions and demonstrate an understanding of programming concepts.

4.3.1 Assessment Objective weightings for International AS Computer Science

Assessment Objectives (AOs)	Unit weightings (approx %)		Overall weighting of AOs (approx %)
	Unit 1	Unit 2	
AO1	0	27.5	27.5
AO2	0	17.5	17.5
AO3	50	5	55
Overall weighting of components (%)	50	50	100

4.3.2 Assessment Objective weightings for International A-level Computer Science

Assessment Objectives (AOs)	Unit weightings (approx %)				Overall weighting of AOs (approx %)
	Unit 1	Unit 2	Unit 3	Unit 4	
AO1	0	11	0	16	27
AO2	0	7	0	9	16
AO3	20	2	30	5	57
Overall weighting of components (%)	20	20	30	30	100

4.4 Assessment weightings

The raw marks awarded on each unit will be transferred to a uniform mark scale (UMS) to meet the weighting of the units and to ensure comparability between units sat in different exam series. Students' final grades will be calculated by adding together the uniform marks for all units. The maximum raw and uniform marks are shown in the table below.

Unit	Maximum raw mark	Percentage weighting A-level (AS)	Maximum uniform mark
Unit 1	75	50	80
Unit 2	75	50	80
International AS qualification	-	100	160
Unit 1	75	20	80
Unit 2	75	20	80
Unit 3	90	30	120
Unit 4	75	30	120
International A-level qualification	-	100	400

5 General administration

We are committed to delivering assessments of the highest quality and have developed practices and procedures that support this aim. To ensure that all students have a fair experience, we have worked with other awarding bodies in England to develop best practice for maintaining the integrity of exams. This is published through the Joint Council for Qualifications (JCQ). We will maintain the same high standard through their use for OxfordAQA.

More information on all aspects of administration is available at **oxfordaqa.com/exams-administration**

For any immediate enquiries please contact **info@oxfordaqa.com**

Please note: We aim to respond to all email enquiries within two working days.

Our UK office hours are Monday to Friday, 8am – 5pm local time.

5.1 Entries and codes

You should use the following subject award entry codes:

Qualification title	OxfordAQA entry code
OxfordAQA International Advanced Subsidiary Computer Science	9646
OxfordAQA International Advanced Level Computer Science	9647

Please check the current version of the Entry Codes book and the latest information about making entries on **oxfordaqa.com/exams-administration**

You should use the following unit entry codes:

Unit 1 – CS01

Unit 2 – CS02

Unit 3 – CS03

Unit 4 – CS04

A unit entry will not trigger certification. You will also need to make an entry for the overall subject award in the series that certification is required.

Exams will be available May/June and in January

5.2 Overlaps with other qualifications

There is overlapping content in the International AS and A-level specifications. This helps you teach the International AS and A-level together.

This specification overlaps with the AQA UK AS and A-level Computer Science (7516/7517).

5.3 Awarding grades and reporting results

The International AS qualification will be graded on a five-point scale: A, B, C, D and E.

The International A-level qualification will be graded on a six-point scale: A*, A, B, C, D and E.

To be awarded an A*, students will need to achieve a grade A on the full A-level qualification and 90% of the maximum uniform mark on the aggregate of the A2 units.

Students who fail to reach the minimum standard for grade E will be recorded as U (unclassified) and will not receive a qualification certificate.

We will publish the minimum raw mark needed for each grade in each unit when we issue students' results. We will report a student's unit results to schools in terms of uniform marks and unit grades and we will report qualification results in terms of uniform marks and grades.

The relationship between uniform marks and grades is shown in the table below.

	Uniform mark range per unit and per qualification					
Grade	Unit 1	Unit 2	International AS Computer Science	Unit 3	Unit 4	International A-level Computer Science
Maximum uniform mark	80	80	160	120	120	400
A*						*See note below
A	64–80	64–80	128–160	96–120	96–120	320–400
B	56–63	56–63	112–127	84–95	84–95	280–319
C	48–55	48–55	96–111	72–83	72–83	240–279
D	40–47	40–47	80–95	60–71	60–71	200–239
E	32–39	32–39	64–79	48–59	48–59	160–199

* For the award of grade A*, a student must achieve grade A in the full International A-level qualification and a minimum of 216 uniform marks in the aggregate of Unit 3 and Unit 4.

5.4 Resits

Unit results remain available to count towards certification, whether or not they have already been used, provided the specification remains valid. Students can resit units as many times as they like, so long as they're within the shelf-life of the specification. The best result from each unit will count towards the final qualification grade. Students who wish to repeat a qualification may do so by resitting one or more units.

To be awarded a new subject grade, the appropriate subject award entry, as well as the unit entry/entries, must be submitted.

5.5 Previous learning and prerequisites

There are no previous learning requirements. Any requirements for entry to a course based on this specification are at the discretion of schools.

5.6 Access to assessment: equality and inclusion

Our general qualifications are designed to prepare students for a wide range of occupations and further study whilst assessing a wide range of competences.

The subject criteria have been assessed to ensure they test specific competences. The skills or knowledge required do not disadvantage particular groups of students.

Exam access arrangements are available for students with disabilities and special educational needs.

We comply with the *UK Equality Act 2010* to make reasonable adjustments to remove or lessen any disadvantage that affects a disabled student. Information about access arrangements is issued to schools when they become OxfordAQA centres.

5.7 Working with OxfordAQA for the first time

You will need to apply to become an OxfordAQA centre to offer our specifications to your students. Find out how at **oxfordaqa.com/centreapprovals**

5.8 Private candidates

Centres may accept private candidates for examined units/components only with the prior agreement of OxfordAQA. If you are an approved OxfordAQA centre and wish to accept private candidates, please contact OxfordAQA at: **info@oxfordaqa.com**

Private candidates may also enter for examined only units/components via the British Council; please contact your local British Council office for details.

6 Appendix: Standard OxfordAQA assembly language instruction set

`LDR Rd, <memory ref>`	Load the value stored in the memory location specified by `<memory ref>` into register `d`
`LDR Rd, [Rn]`	Load the value stored in the memory location specified in register `n` into register `d`
`STR Rd, <memory ref>`	Store the value that is in register `d` into the memory location specified by `<memory ref>`
`STR Rd, [Rn]`	Store the value that is in register `d` into the memory location specified by register `n`
`ADD Rd, Rn, <operand2>`	Add the value specified in `<operand2>` to the value in register `n` and store the result in register `d`
`SUB Rd, Rn, <operand2>`	Subtract the value specified by `<operand2>` from the value in register `n` and store the result in register `d`
`MOV Rd, <operand2>`	Copy the value specified by `<operand2>` into register `d`
`CMP Rn, <operand2>`	Compare the value stored in register `n` with the value specified by `<operand2>`
`B <label>`	Always branch to the instruction at position `<label>` in the program.
`B<condition> <label>`	Branch to the instruction at position `<label>` if the last comparison met the criterion specified by `<condition>`. Possible values for `<condition>` and their meanings are: `EQ:` equal to `NE:` not equal to `GT:` greater than `GE:` greater than or equal to `LT:` less than `LE:` less than or equal to
`AND Rd, Rn, <operand2>`	Perform a bitwise logical AND operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`
`ORR Rd, Rn, <operand2>`	Perform a bitwise logical OR operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`
`EOR Rd, Rn, <operand2>`	Perform a bitwise logical XOR (exclusive or) operation between the value in register `n` and the value specified by `<operand2>` and store the result in register `d`
`MVN Rd, <operand2>`	Perform a bitwise logical NOT operation on the value specified by `<operand2>` and store the result in register `d`
`LSL Rd, Rn, <operand2>`	Logically shift left the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`
`LSR Rd, Rn, <operand2>`	Logically shift right the value stored in register `n` by the number of bits specified by `<operand2>` and store the result in register `d`
`HALT`	Stops the execution of the program.

Labels: A label is placed in the code by writing an identifier followed by a colon (`:`). To refer to a label, the identifier of the label is placed after the branch instruction.

Interpretation of `<operand2>`

`<operand2>` can be interpreted in two different ways, depending on whether the first character is a `#` or an `R`:

- `#` – use the decimal value specified after the `#`, eg `#25` means use the decimal value 25
- `Rm` – use the value stored in register `m`, eg `R6` means use the value stored in register 6

The available general purpose registers that the programmer can use are numbered 0–12

OXFORD AQA
INTERNATIONAL QUALIFICATIONS

Fairness *first*

Thank you for choosing OxfordAQA, the international exam board that puts fairness first.

Benchmarked to UK standards, our exams only ever test subject ability, not language skills or cultural knowledge.

This gives every student the best possible chance to show what they can do and get the results they deserve.

Get in touch

You can contact us at *oxfordaqa.com/contact-us*

or email *info@oxfordaqa.com*