OXFORD AQA
INTERNATIONAL QUALIFICATIONS

The international exam board *that puts fairness first*

OxfordAQA International AS

Computer Science (9645)

Scheme of work

For teaching from September 2024 onwards
For International AS exams in May/June 2025 onwards

OxfordAQA is a partnership between AQA, the UK's largest awarding body, and Oxford University Press, a department of the University of Oxford. We offer globally relevant international GCSEs, AS and A-levels to schools across Europe, Africa and Asia.

This scheme of work suggests possible teaching and learning activities for each section of the specification that covers AS level content. There are far more activities suggested than it would be possible to teach. It is intended that teachers should select activities appropriate to their students and the curriculum time available. The first two sections summarise the specification references, whilst the learning outcomes indicate what most students should be able to achieve after the work is completed. The resources section indicates resources commonly available to schools, and other references that may be helpful. The timings are only suggested, as are the possible teaching and learning activities, which include references to experimental work. Resources are only given in brief and risk assessments should be carried out.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Contents

You can use the title links to jump directly to the different sections of this scheme of work (Use Ctrl and click to follow the link)

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Introduction

This outline scheme of work is intended to help teachers plan and implement the teaching of the Oxford AQA International AS Level Computer Science (9645) specification. The purpose of this scheme of work is to provide advice and guidance to teachers, not to prescribe and restrict their approach to the specification. This scheme has been produced by a senior Computer Science examiner and is intended to be helpful for teachers who are new to the specification. There are many other ways of organising the work, and there is absolutely no requirement to use this scheme. The scheme of work should be used alongside the specimen assessment material.

Assumed coverage

This scheme assumes that the AS Level Computer Science content is a one-year course. A separate scheme of work is available that covers the A2 Computer Science content which is an additional one year of study.

In this scheme of work, we have made the assumption that it will be taught over about 30 weeks with 4½ to 5 hours of contact time per week. Teachers will need to fine tune the timings to suite their own students and the time available. It could also be taught by one teacher or by more than one teacher with topics being taught concurrently.

International terminology

International terminology will be used, for more guidance please see subject specific vocabulary.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Unit 1: Programming

Programming languages

A large section of this scheme of work covers practical programming activities. Centres must choose **one** of these programming languages:

- C#
- Python
- VB.Net

These links contain general resources that can be used to support each programming language. Teachers and students may find these links useful in their delivery/study. These resources are by no means exhaustive and centres are encouraged to use other resources.

Python:

Python official website:
The Python Tutorial — Python 3.12.4 documentation

Python official website:
The Python Tutorial — Python 2.7.18 documentation

Python tutor videos with schematic animations:
youtube.com/watch?v=joGNzHR4GhI

Dive into Python:
diveintopython3.net/

Dive into Python:
diveintopython.net/

Python tutor:
pythontutor.com/

Wikibooks:
en.wikibooks.org/wiki/A_Beginner%27s_Python_Tutorial

C#:

Home and Learn:
homeandlearn.co.uk/csharp/csharp.html

MSDN:
learn.microsoft.com/en-gb/

Wikibooks:
en.wikibooks.org/wiki/C_Sharp_Programming

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

VB.Net:

Tutorialspoint:
tutorialspoint.com/vb.net/

Wikibooks:
wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Programming/A_program

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1 Procedural programming

3.1.1 Data types

Specification content

- Understand the concept of a data type.
- Understand and use data types appropriately: integer, real, float, Boolean, character, string, date/time.

Learning outcomes

- Know why data types are needed in programming code.
- Select and use different data types for different tasks.

Suggested timing

1 hour

Guidance

Possible teaching activities include:

- **Class discussion:** Start the unit off by asking key questions such as:
    - What is meant by the term programming code is?
    - What experiences do you have programming?
    - What programming languages have you used before and what have you used them for?
    - What is meant by the term data type and what data types do you know?
    - Why do we need to have different data types in programming code?

- **Teacher demonstration:** Give practical demonstrations on how to define different data types in the chosen programming language. Model potential problems that can arise when not setting these correctly.

- **Practical exercises:** Ask students to complete programming exercises where they have to choose and set different data types in the chosen language.

Resources

- Small programming exercises / examples of programming code
- Notes on data types:
    - studysmarter.co.uk/explanations/computer-science/computer-programming/data-types-in-programming/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1.2 Programming concepts

Note: Students will not need to use the terms function and procedure to distinguish between subroutines that do and do not return values.

Specification content

- Use, understand and know how the following statement types can be combined in programs: variable declaration, assignment, selection, iteration and subroutine.
- 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.

Learning outcomes

- Implement variables, assignments, selection, iteration and subroutines within a programming language.
- Know why it is important to use meaningful identifier names.
- Be able to display output and obtain input from the user.
- Know why it is important to use code comments.

Suggested timing

5 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start off by asking key questions such as:
    - What is meant by the terms variable and assignment?
    - What is meant by the term selection? What can we use to implement this?
    - What is meant by the term iteration? What is the difference between definite and indefinite?
    - What is meant by the term subroutine?

- **Teacher Demonstration:** Show students how to implement variables, assignment, selection, iteration and subroutines. While doing this, model good practice by using suitable identifier names and code comments. It may be better to cover each of these separately and then allow students to consolidate their understanding by completing practical exercises.

- **Practical Exercises:** Give students different challenges that allow them to implement variables, assignment, selection, iteration and subroutines.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Programming challenges that allow students to implement variables, assignment, selection, iteration and subroutines.
- Notes on programming constructs:
    - youtube.com/watch?v=VuQAyz-6yUs

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1.2.1 Arithmetic operations in a programming language

3.1.2.2 Relational operations in a programming language

3.1.2.3 Boolean operations in a programming language

Note: Students will not be required to know the order of precedence of XOR, which varies between languages and systems.

Specification content

- Be familiar with and be able to use arithmetic operations (addition, subtraction, multiplication, real/float division, integer division (DIV and MOD), exponentiation, rounding and truncation).
- Be familiar with and be able to use relational operations (equal to, not equal to, less than, greater than, less than or equal to and greater than or equal to).
- Be familiar with and be able to use Boolean operations (NOT, AND, OR, XOR)
- Know the order of precedence between NOT, AND and OR operators.

Learning outcomes

- Select and use different arithmetic, relational and Boolean operators within programming code.
- Know the order of precedence between NOT, AND and OR operators.

Suggested timing

1.5 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start off by asking key questions such as:
    - What is meant by the term operator?
    - What is the difference between arithmetic, relational and Boolean operators?

- **Teacher Demonstration:** Show students the different symbols that are used for each operator and demonstrate how to use these within the chosen programming language.

- **Practical Exercises:** Give students different challenges that allow them to use and implement the different operators.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Programming challenges that allow students to use and implement arithmetic, relational and Boolean operators.
- Notes on Mathematical operators:
    - youtube.com/watch?v=cYQlYaZOZho

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1.2.4 String-handling operations in a programming language

3.1.2.5 Random number generation in a programming language

Specification content

- Be familiar with and be able to use different string handling operations (length, position, substring, concatenation, character → character code, character code → character, string conversion operations).
- Be familiar with, and be able to use, random number generation.

Learning outcomes

- Implement different string handling operations within programming code.
- Implement string conversation operations within programming code.
- Generate and use random numbers within programming code.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start off by asking key questions such as:
    - What is meant by the term string handling?
    - Why is string handling useful?
    - What is meant by the terms: length, position, substring, concatenation, character → character code, character code → character?
    - What is meant by the term string conversion and why is this needed?

- **Teacher Demonstration:** Show students how to implement the different string handling operations and string conversions operations within the chosen language. Show students how to generate random numbers within the chosen language.

- **Practical Exercises:** Give students different challenges that allow them to implement string handling operations, implement string conversions and generate and use random numbers.

Resources

- Programming challenges that allow students to use and implement the different string handling operations, string conversions operations and random numbers.
- Notes on string handling operators:
    - bbc.co.uk/bitesize/guides/zb33rwx/revision/1

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1.2.6 Exception handling

Specification content

- Be familiar with, and be able to use, exception handling.

Learning outcomes

- Implement exception handling within a programming language.

Suggested timing

1.5 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students what is meant by the term exception handling and give examples of different types of errors it can be used on (e.g. code trying to access a file that does not exist, division by zero).

  Next, show students how to implement exception handling in the chosen programming language.

- **Practical Exercises:** Give students different challenges that allow them to implement exception handling techniques.

Resources

- Programming challenges that allow students to use and implement exception handling.
- Notes on exception handling:
  - youtube.com/watch?v=w-FbaQLDVrY

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.1.2.7 Subroutines

Note: 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.

Specification content

- Be familiar with subroutines and their uses.
- Know that a subroutine is a named ‘out of line’ block of code that may be executed.
- Be able to explain the advantages of using subroutines in programs.
- Be able to describe the use of parameters to pass data within programs.
- Be able to use subroutines that return values to the calling routine.
- 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.
- Know that it is good practice to limit the scope of a variable where possible.
- Be able to contrast local variables with global variables.
- Be able to explain how a stack frame is used with subroutine.

Learning outcomes

- Implement subroutines with parameters within programming code.
- Call subroutines and store the return value within programming code.
- Know the benefits of building programs using subroutines.
- Implement global and local variables within programming code and know why global variables should generally be avoided.

Suggested timing

5 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students what is meant by the term subroutine and the benefits of using them.

- Next, demonstrate how to implement these within the chosen programming language including the use of parameters. Show students how to call the subroutines by calling them with the necessary parameters.

- Tell students what is meant by the return value and demonstrate how this should be stored. This should then be followed up with individual consolidation exercises.

- When students have a clear understanding of subroutines, introduce students to the concept of local and global variables and model how to define these within the chosen programming language. Demonstrate the problems that can be caused by using global variables and why they are generally avoided.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Practical Exercises:** Give students different challenges that allow them to:
    - Implement subroutines with parameters.
    - Call subroutines with parameters and store the return value.
    - Implement local and global variables.

Resources

- Programming challenges that allow students to implement subroutines, local variables and global variables.
- Notes on subroutines:
    - youtube.com/watch?v=DC1Aw6BCrR8

- Notes on local and global variables:
    - youtube.com/watch?v=yMqC5VIqcNA

- Notes on stack frames with subroutines:
    - youtube.com/watch?v=LbHsQcRlsbU&list=PLCiOXwirraUDd7mhZFyNFCPPwWzCAyh4w

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.2 Fundamental data structures

3.2.1 Arrays and lists

3.2.2 Record

Note: 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.

Specification content

- 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.
- Be able to use arrays and lists when solving problems, including the use of two-dimensional arrays and lists of lists.
- Be familiar with the composition of a group of values (known as fields) into a record which can be manipulated as a single entity.

Learning outcomes

- Know what is meant by the term data structure and the difference between static and dynamic data structures.
- Implement arrays and lists within programming code.
- Implement record structures within programming code.

Suggested timing

3 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students what is meant by the term data structure and why they are needed.

- Next demonstrate how to implement arrays within the chosen programming language including two-dimensional arrays. Once these have been defined, demonstrate how to manipulate the different elements in the array using the array index. Next demonstrate how to implement lists, including lists within lists. This should then be followed up with individual consolidation exercises.

- When students have a clear understanding of data structures, arrays and lists, introduce students to the record data structure and demonstrate how to implement this within the chosen language. This should then be followed up with individual consolidation exercises.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Practical Exercises:** Give students different challenges that allow them to:
    - Implement arrays including two-dimensional arrays.
    - Implement lists including lists of lists.
    - Implement record structures.

Resources

- Programming challenges that allow students to implement arrays, lists and records.
- Notes on arrays and lists:
    - geeksforgeeks.org/array-data-structure-guide/?ref=home-articlecards
    - youtube.com/watch?v=h3Pm4n6icIM
    - learnpython.com/blog/python-array-vs-list/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.2.3 Queues

3.2.4 Stacks

Specification content

- Understand that a queue is a first-in first-out (FIFO) data structure.
- Be able to apply operations to data stored in a queue.
- 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.
- Understand that a stack is a last-in first-out (LIFO) data structure.
- Be able to apply operations to data stored in a stack.
- 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.

Learning outcomes

- Know the difference between a stack and a queue.
- Know the different operations that can be applied to a stack and a queue.
- Know situations when a stack and a queue would be used.
- Implement stacks and queues using one-dimensional arrays within a programming language.

Suggested timing

5 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students what is meant by the term queue and why they are used. Tell students how they are structured with the use of a one-dimensional array and then manipulated with a head and tail pointer. Next demonstrate how to implement a queue within the chosen programming language. Once this has been defined, demonstrate how to add and remove data from the queue.

- Next introduce students to a stack, why they are used and explain how this is different to a queue. Tell students how they are structured with the use of a one-dimensional array and then manipulated with a stack pointer. Next demonstrate how to implement a stack within the chosen programming language. Once this has been defined, demonstrate how to add and remove data from the stack.

- **Practical Exercises:** Give students different challenges that allows them to:
    - implement queue data structures and apply enqueue and dequeue operations
    - implement stack data structures and apply push, pop and peek operations

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Programming challenges that allow students to implement stacks and queues.
- Notes on stacks and queues:
    - geeksforgeeks.org/stack-data-structure/?ref=shm
    - geeksforgeeks.org/queue-data-structure/?ref=shm
    - youtube.com/watch?v=WnjVVJ0kIgQ
    - youtube.com/watch?v=1hEK9yWs_cM

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.3 Program design

3.3.1 Structured approach to programming

3.3.2 Abstraction and decomposition

Specification content

- Understand the structured approach to program design and construction.
- Be able to explain the advantages of the structured approach.
- Be able to construct and use hierarchy charts and structure charts when designing programs.
- Be familiar with the concept of abstraction.
- Be familiar with the concept of decomposition.

Learning outcomes

- Know what is meant by a structured approach, its features and the benefits of this.
- Construct hierarchy charts and structure charts.
- Know what is meant by abstraction and decomposition.

Suggested timing

3 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Tell students what is meant by the term structured approach and then ask students to draw upon their knowledge of programming and discuss the different features of the structured approach. Discussions will most likely include the use of modules, parameters, return values and local variables.

- **Teacher Demonstration:** Tell students the purpose of hierarchy charts and structure charts and how to construct these.

- **Individual Work:** Give students a variety of different problems and ask them to design a solution to these problems by constructing hierarchy charts and structure charts.

- **Teacher Demonstration:** Tell students the meaning of abstraction and decomposition and the reasons why these are used.

- **Individual/Group/Paired Work:** Give students a variety of different problems and ask students to determine how abstraction and decomposition can be used. This can be strengthened further with research on the benefits of these tools.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Problems/scenarios for students to construct hierarchy charts and structure charts.
- Problems that allow students to determine how abstraction and decomposition can be used.
- Notes on abstraction and decomposition:
    - learnlearn.uk/alevelcs/abstraction-pattern-recognition-decomposition/
    - youtube.com/watch?v=kK5Zrl7v1ds&list=PLCiOXwirraUCDWr2XWh7Wg69D7q1Jw6ac&index=5
    - youtube.com/watch?v=nnOMsBykeks&list=PLCiOXwirraUCDWr2XWh7Wg69D7q1Jw6ac&index=9

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.3.3 Following and writing algorithms

Note: Students will be expected to be able to understand algorithms written in pseudocode but will not be expected to write pseudocode in the exam.

Specification content

- Understand the term algorithm.
- Be familiar with the use of pseudocode to express algorithms.
- Be able to convert an algorithm from pseudocode into high-level language program code.
- Be able to hand-trace algorithms.

Learning outcomes

- Know what is meant by the term algorithm.
- Convert pseudocode into high-level language program code.
- Construct and trace an algorithm using trace tables.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students what is meant by the term algorithm and how these can be expressed in pseudocode. Model how to convert a simple algorithm written in pseudocode into high-level language program code.

- Next, tell students the purpose of a trace table, how to construct these and model how to trace an algorithm using the trace table.

- **Individual/Group/Paired Work:** Give students pseudocode algorithms and ask them to construct and complete trace tables for these. Once they have done this, ask them to draw on their knowledge of programming from across the unit and convert these into high-level language program code.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Algorithms written in pseudocode to allow students to construct trace tables and convert them into high-level language program code.
- Notes on trace tables:
    - 101computing.net/using-trace-tables/
    - youtube.com/watch?v=CwLnL--66tY
    - en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Trace_tables

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.3.4 Aspects of software development

Specification content

- 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.
- 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 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 implementation must be tested for the presence of errors, using selected test data covering normal (typical), boundary and erroneous data.
- Know the criteria for evaluating a computer system.

Learning outcomes

- Know the different stages of software development.
- Identify key activities that occur in each stage of software development.

Suggested timing

2.5 hours

Guidance

Possible teaching activities include:

- **Teacher Input:** Introduce students to software development lifecycles and go through the common stages: analysis, design, implementation, testing and evaluation. Cover some of the key activities that occur in each stage as outlined in the specification.

- **Research Work:** Ask students to research other key activities and tools that can be used in each stage of software development. For example, during the design stage, students can research the different areas that need to be planned (e.g. data structures) and the tools that can be used to achieve this (eg algorithms).

- **Present findings**: Students should present their findings from their research work.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Notes on software development stages:
  - youtube.com/watch?v=sb6H9VHv53c&list=PLCiOXwirraUAk9K3cwd1eeh9C6N5kaMSI
  - clouddefense.ai/system-development-life-cycle/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.4 Searching and sorting algorithms

3.4.1 Searching algorithms

Note: Formal comparisons using Big-O notation will not be required at AS but may be required at A-level.

Specification content

- Know the linear search algorithm and be able to trace it.
- Know the binary search algorithm and be able to trace it.
- Compare the linear search and binary search algorithms.

Learning outcomes

- Write and trace programming code for linear and binary searches.
- Compare and know the conditions when linear and binary searches can be used.

Suggested timing

4 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students the purpose of search algorithms. Next demonstrate the use of the linear and binary search on different sets of data and explain how they work.

- **Class Discussion:** Ask students key questions such as:
    - Which search algorithm do you think is the most efficient? Why?
    - What are the conditions in which these search algorithms can be used?
    - How do you think these could be implemented in programming code?

- **Individual Work:** Give students different sets of data and trace tables. Ask students to trace linear and binary searches.

- **Individual/Group/Paired Work:** Ask students to implement a linear search and a binary search within a high-level programming language. If needed, give students pseudocode for each of these search algorithms to help them.

Resources

- Worksheet containing linear and binary search trace tables.
- Notes on linear and binary search algorithms:
    - geeksforgeeks.org/linear-search-vs-binary-search/
    - freecodecamp.org/news/search-algorithms-linear-and-binary-search-explained/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.4.2 Sorting algorithms

Notes:

- Students will be expected to know that a number of versions of the bubble sort algorithm exists and that the most basic version uses definite iteration.
- 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.
- Formal comparisons using Big-O notation will not be required at AS, but may be required at A-level.

Specification content

- Know the bubble sort algorithm and be able to trace it.
- Know the merge sort algorithm and be able to demonstrate how it operates.
- 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.

Learning outcomes

- Write and trace programming code for a bubble sort.
- Trace programming code for a merge sort.
- Compare the bubble sort and merge sort algorithms.

Suggested timing

4 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Tell students the purpose of sorting algorithms. Next demonstrate the use of a bubble sort and merge sort on different sets of data and explain how they work.

- **Class Discussion:** Ask students key questions such as:
    - Which sorting algorithm do you think is the most efficient? Why?
    - How do you think the bubble sort algorithm can be made more efficient?
    - Which algorithm do you think uses more memory? Why?
    - How do you think the bubble sort algorithm could be implemented in programming code?

- **Individual Work:** Give students different sets of data and trace tables. Ask students to trace a bubble sort and merge sort in both ascending and descending order.

- **Individual/Group/Paired Work:** Ask students to implement a bubble sort within a high-level programming language. If needed, give students the pseudocode for this to help them.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Worksheet containing bubble sort and merge sort trace tables.
- Notes on the bubble sort algorithm:
  - geeksforgeeks.org/bubble-sort-algorithm/
  - w3schools.com/dsa/dsa_algo_bubblesort.php
  - wikipedia.org/wiki/Bubble_sort
- Notes on the merge sort algorithm:
  - geeksforgeeks.org/merge-sort/?ref=shm
  - w3schools.com/dsa/dsa_algo_mergesort.php

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Unit 2: Concepts and principles of computer science

3.5 Representing data

3.5.1 Number bases

Note: 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

Specification content

- Be familiar with the concept of a number base, in particular: decimal (base 10), binary (base 2) and hexadecimal (base 16).
- Convert between decimal, binary and hexadecimal number bases.

Learning outcomes

- Convert between decimal, binary and hexadecimal number bases.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start by asking students why computer systems store data in binary format.

- **Teacher Demonstration:** Tell students what decimal, binary and hexadecimal numbers are and show students how to convert between these number bases.

- **Individual/Group/Paired Work:** Give students decimal, binary and hexadecimal conversion questions. Students can check their answers using online conversion calculators.

Resources

- Worksheet containing number base conversion questions.
- Notes on number basis:
    - youtube.com/watch?v=DoQV6wp35Gw
    - educative.io/answers/decimal-binary-and-hex-conversion-table
    - bbc.co.uk/bitesize/guides/zd88jty/revision/1

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.2 Units of information

Specification content

- Know that the bit is the fundamental unit of information and a byte is a group of 8 bits.
- Know that 2^n different values can be represented with n bits
- Know that quantities of bytes can be described using binary prefixes representing powers of 2 or using decimal prefixes representing powers of 10.
- Know the names, symbols and corresponding powers of 2 for the binary prefixes.
- Know the names, symbols and corresponding powers of 10 for the decimal prefixes.

Learning outcomes

- Calculate the number of unique values that can be represented from a given number of bits.
- Know and calculate between binary and decimal prefixes.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start by asking students key questions:
  - What is a bit?
  - What is a byte?
  - Why do we have different units of information?

- **Teacher Demonstration:** Tell students the difference between binary and decimal prefixes and the different units for each. Then show students how to convert between the different units of information and how to calculate the number of unit values that can be represented from a given number of bits.

- **Individual/Group/Paired Work:** Give students binary and decimal units of information conversion questions. Students can check their answers using online conversion calculators.

Resources

- Worksheet containing units of information conversion questions.
- Notes on information units:
  - youtube.com/watch?v=9kzCoDFrMGE
  - blocksandfiles.com/2022/04/23/decimal-and-binary-prefixes/
  - knowitallninja.com/lessons/units-numbers-aqa/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.3.1 Unsigned binary

3.5.3.2 Unsigned binary arithmetic

Specification content

- 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 2n - 1 respectively.
- Be able to add two unsigned binary integers and multiply two unsigned binary integers.

Learning outcomes

- Know the difference between unsigned binary and signed binary.
- Carryout unsigned binary addition and multiplication.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Start by asking students key questions:
  - What is the difference between a signed and unsigned binary number?
  - Why do you think we use unsigned binary numbers?

- **Teacher Demonstration:** Show students how to carry out unsigned binary arithmetic by adding two unsigned binary numbers and multiplying two unsigned binary numbers.

- **Individual/Group/Paired Work:** Give students unsigned binary addition and multiplication questions. Students can check their answers using online calculators.

Resources

- Worksheet containing unsigned binary arithmetic questions.
- Notes on unsigned binary arithmetic:
  - studyrocket.co.uk/revision/a-level-computer-science-aqa/fundamentals-of-data-representation/unsigned-binary-arithmetic
  - youtube.com/watch?v=t15dhDG_WUA
  - geeksforgeeks.org/add-two-unsigned-numbers-using-bits/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.3.3 Signed binary using two's complement

Note: 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.

Specification content

- Know that signed binary can be used to represent negative integers and that one possible coding scheme is two's complement.
- 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.

Learning outcomes

- Represent negative and positive integers in two's complement format.
- Perform subtraction using two's complement.
- Calculate the range of a given number of bits in two's complement.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Show students how to:
  - Represent negative and positive numbers in two's complement format.
  - Perform subtraction using two's complement.
  - Calculate the range of a given number of bits.

- **Individual/Group/Paired Work:** Give students consolidation questions on two's complement numbers including two's complement subtraction. Students can check their answers using online calculators.

Resources

- Worksheet containing two's complement binary questions.
- Notes on two's complement binary:
  - tutorialspoint.com/two-s-complement
  - sites.google.com/rgc.aberdeen.sch.uk/rgc-highercomputing/computer-systems/data-representation/twos-complement
  - youtube.com/watch?v=CglODZZm_Z4

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.3.8 Compare fixed and floating point

3.5.3.4 Numbers with a fractional part

Notes:

- 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.
- Exam questions on floating point numbers will use a format in which both the mantissa and exponent are represented using two's complement.

Specification content

- Compare the advantages and disadvantages of fixed point and floating point representations.
- Know how numbers with a fractional part can be represented in fixed point form in binary in a given number of bits and format and floating point form in binary in a given number of bits and format.
- For both fixed point and floating point be able to convert from decimal to binary using a given number of bits and format and from binary to decimal using a given number of bits and format.
- 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.

Learning outcomes

- Know how to represent numbers with a fractional part in fixed point and floating point form.
- Convert from decimal to fixed point and floating point binary and vice-versa.
- Know the impact of changing the number of bits that is allocated to the mantissa and exponent.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Show students how to convert decimal numbers to fixed point binary and vice-versa. Tell students the problems with fixed-point binary (i.e. limited range).

- Next, introduce students to floating point binary. Tell students the difference between the mantissa and exponent and how to convert decimal numbers to floating point binary and vice-versa.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Class Discussion:** What do you think is the impact of allocating more bits to the mantissa? What do you think is the impact of allocating more bits to the exponent?

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions on decimal to fixed point and floating point binary and vice versa.

- **Research Activity:** Ask students to research the advantages and disadvantages of fixed point and floating point representations in terms of:
    - Range
    - Precision
    - Speed of calculation

Resources

- Worksheet containing fixed point and floating point binary conversion questions.
- Notes on fixed point and floating point binary:
    - tutorialspoint.com/fixed-point-and-floating-point-number-representations
    - geeksforgeeks.org/fixed-point-representation/
    - geeksforgeeks.org/fixed-point-representation/
    - youtube.com/watch?v=dcIDAnfp8Dc&t=12s

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.3.5 Rounding errors

3.5.3.6 Absolute and relative errors

3.5.3.7 Underflow and overflow

Specification content

- Know and be able to explain why both fixed point and floating point representation of decimal numbers may be inaccurate.
- 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.
- Explain overflow and underflow and describe the circumstances in which they occur.

Learning outcomes

- Know the meaning of rounding errors, absolute errors and relative errors.
- Know the meaning of underflow and overflow and why they occur.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Is it possible to represent 0.1 in binary? Why? What would you do to store this? What would it get rounded to?

- **Teacher Demonstration:** Show students how to work out the closest binary representation that can be used to represent 0.1. Then introduce students to the terms absolute error and relative error and how to calculate these. Next, introduce students to the terms underflow and overflow and why these occur.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions calculating absolute and relative errors.

Resources

- Notes on errors:
  - youtube.com/watch?v=cYujRJOZ1DA
  - youtube.com/watch?v=e9QsbmwckbE
  - youtube.com/watch?v=vH7pQxWTTio

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.3.9 Normalisation of floating point form

Specification content

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

Learning outcomes

- Normalise an un-normalised floating point number with positive or negative mantissas.

Suggested timing

1.5 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Give students three or four floating point numbers that have the same decimal value. Ask students to calculate the value of these and then discuss how this is possible.

- **Teacher Demonstration:** Tell students how to determine if a floating point number is normalised or not. Then model how to normalise an un-normalised floating point number and the benefits of doing this.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions on normalising un-normalised floating point numbers.

Resources

- Worksheet containing floating point normalisation questions.
- Notes on floating point normalisation:
    - youtube.com/watch?v=RcY0aiSsyqI
    - teachict.com/as_as_computing/ocr/H447/F453/3_3_4/floating_point/miniweb/pg10.htm
    - tutorchase.com/notes/cie-a-level/computer-science/13-3-4-normalisation-of-floating-point-numbers

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.4 Representing characters

Note: 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.

Specification content

- 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.
- Know that ASCII is a 7-bit encoding system.
- Know that Unicode has more than one encoding system.
- Know that in both Unicode and ASCII characters are grouped into blocks.
- Understand the difference between the character code representation of a decimal digit and its pure binary representation.

Learning outcomes

- Know how characters are represented using character sets and how characters are grouped into blocks.
- Know the difference between ASCII and Unicode.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Computer systems store data in binary format. How are characters stored in binary?

- **Teacher Input:** Tell students how characters in a character set are assigned a unique value and characters are grouped into blocks. Tell students that two key character sets are ASCII and Unicode.

- **Research Work:** Ask students to research ASCII and Unicode including:
    - The different blocks (ie where numeric, uppercase and lowercase letters start).
    - The number of bits used for each.
    - The benefits and drawbacks of each.
    - Why Unicode was introduced.
    - The features of UTF-8.

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Notes on ASCII and Unicode:
  - geeksforgeeks.org/ascii-vs-unicode/
  - tutorialspoint.com/ascii-vs-unicode

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.5.1 Bitmapped graphics

3.5.5.2 Vector graphics

3.5.5.3 Comparing bitmapped and vector graphics

Specification content

- Describe how bitmaps represent images.
- Understand the following for bitmaps: size in pixels and colour depth.
- Calculate storage requirements for bitmapped images, excluding metadata.
- Be aware that bitmap image files may also contain metadata.
- Be familiar with typical metadata.
- Explain how vector graphics represent images using lists of objects.
- Give examples of typical properties of objects.
- 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.

Learning outcomes

- Know how images can be constructed in bitmap and vector format.
- Know key terms/properties associated with bitmap images and vector graphics.
- Calculate the size of a bitmap image.
- Know and give examples of metadata.
- Know appropriate uses, benefits and drawbacks of bitmap images and vector graphics.

Suggested timing

3 hours

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Using a simple image (e.g. outline of a house), demonstrate how it would be constructed as a bitmap image (with pixels) and as a vector (with lines/shapes).

- Next focus on bitmap images. Tell students the meaning of a pixel and colour depth and how to calculate the storage requirements for a bitmap image.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing information about different bitmap images for students to calculate the storage requirements.

- **Teacher Demonstration:** Next focus on vector graphics and the typical properties of vector objects.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing simple vector graphics for students to identify the properties used.

- **Research Work:** Ask students to research:
    - Appropriate uses of each
    - Benefits of each
    - Drawbacks of each

- **Student Feedback:** Students should present their findings from their research to the class.

Resources

- Worksheet containing information about bitmap images and vector graphics.
- Notes on bitmap images and vector graphics:
    - tutorialspoint.com/What-is-Bitmap
    - tutorialspoint.com/difference-between-raster-and-vector
    - filecamp.com/blog/vector-vs-bitmap-images-explained/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.6.1 Analogue and digital

3.5.6.2 Digital representation of sound and Nyquist's theorem

Specification content

- Understand the difference between analogue and digital quantities.
- Describe the principles of operation of an analogue to digital converter (ADC) and 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.
- Describe the digital representation of sound in terms of sample resolution and sampling rate.
- Calculate sound sample sizes in bytes.
- 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.

Learning outcomes

- Know the difference between analogue and digital sound and why ADC and DAC are used.
- Know how sound is stored in a computer system.
- Know key terms associated with sound files.
- Calculate the size of a sound file.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Teacher Input:** Tell students the difference between analogue and digital sound and the purpose of an ADC and DAC. Tell students how sound is stored in a computer system and cover the key terms such as sample resolution and sample rate.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing information about different sound files for students to calculate the storage requirements.

- **Research Work:** Ask students to research:
    - The Nyquist's theorem.
    - How to 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.

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Worksheet containing information about sound files.
- Notes on sound files and Nyquist's theorem:
    - tutorialspoint.com/digital_communication/digital_communication_sampling.htm
    - techtarget.com/whatis/definition/Nyquist-Theorem
    - youtube.com/watch?v=VyqGzUbTphs&list=PLCiOXwirraUA69WUAMYyFicC5qbQ4PGc4
    - youtube.com/watch?v=v7RwEnirp7I&list=PLCiOXwirraUA69WUAMYyFicC5qbQ4PGc4&index=2
    - youtube.com/watch?v=aI5HsKIRhQw&list=PLCiOXwirraUA69WUAMYyFicC5qbQ4PGc4&index=3

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.7 Basic encryption methods

Note: 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.

Specification content

- Understand what is meant by encryption and be able to define it.
- Be familiar with the terms cipher, plaintext and ciphertext.
- Know what a key is.
- 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.
- 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.
- Compare the Caesar and Vernam ciphers.
- 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.
- Understand the key exchange problem.
- Know the difference between a symmetric and an asymmetric cipher.

Learning outcomes

- Know the meaning of encryption, cipher, plaintext, ciphertext and key.
- Be able to compare and apply the Caesar cipher and Vernam cipher.
- Know why the Caesar cipher is easily cracked and how the Vernam cipher can be perfectly secure.
- Know the key exchange problem and the difference between a symmetric and an asymmetric cipher.

Suggested timing

3 hours

Guidance

Possible teaching activities include:

- **Teacher Input:** Tell students the meaning of key terms such as encryption, cipher, plaintext, ciphertext and key.

- **Teacher Demonstration:** Show students how to apply the Caesar cipher to encrypt a plaintext message and decrypt a ciphertext message.

- **Class Discussion:** Why do you think the Caesar cipher can be easily cracked?

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Teacher Demonstration:** Show students how to apply the Vernam cipher to encrypt a plaintext message and decrypt a ciphertext message.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions that allow them to apply the Caesar cipher and Vernam cipher.

- **Research Work:** Ask students to research:
    - What is meant by the term perfect security.
    - How the Vernam cipher can be perfectly secure.
    - The conditions that must be met for the Vernam cipher to be perfectly secure.
    - How ciphers rely on computational security.
    - The key exchange problem.
    - The difference between a symmetric and an asymmetric cipher.

- **Student Feedback:** Students should present their findings from their research to the class.

Resources

- Worksheet containing exercises to allow students to apply the Caesar cipher and Vernam cipher.
- Notes on Caesar cipher and Vernam cipher:
    - geeksforgeeks.org/caesar-cipher-in-cryptography/
    - geeksforgeeks.org/vernam-cipher-in-cryptography/
    - youtube.com/watch?v=kS9CnbhytV0
    - youtube.com/watch?v=Tc0W5i0bXPk

- Notes on encryption terms:
    - geeksforgeeks.org/difference-between-symmetric-and-asymmetric-key-encryption/
    - youtube.com/watch?v=thESC86I2Ps
    - jscape.com/blog/key-exchange

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.5.8 Error detection and correction

Specification content

- Describe and explain the use of: parity bits, majority voting and checksums.
- Compare the use and effectiveness of these three methods.

Learning outcomes

- Use different error detection and correction methods.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion**: Why do you think errors occur when data is being transmitted on a network?

- **Teacher Demonstration:** Show students how to parity bits, majority voting and checksums to detect errors.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions that allow them to apply their understanding of the different error detection and correction methods.

Resources

- Worksheet containing exercises to allow students to apply their understanding of the different error detection and correction methods.
- Notes on different error detection and correction methods:
    - youtube.com/watch?v=bUtafZywvSQ
    - learnlearn.uk/igcsecs/error-detection/#google_vignette

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.6 Computer systems

3.6.1 Hardware and software

3.6.2 Software

Notes:

- 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.
- Students only need to understand how main memory is allocated. Virtual memory does not need to be covered.

Specification content

- Understand the relationship between hardware and software.
- Explain what is meant by: system software and application software.
- Understand the need for, and functions of the following system software.
- 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 functions of an operating system.

Learning outcomes

- Know the difference between hardware and software.
- Know the purpose and difference between system software and application software.
- Know the functions of an operating system.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** What is the difference between hardware and software. What is the difference between system software and application software?

- **Research Work:** Ask students to research the need for, and functions of:
    - Operating systems (OSs)
    - Utility programs
    - Libraries
    - Translators (compiler, assembler, interpreter)

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Research Work:** Ask students to research these functions of an operating system:
  - Scheduling
  - Memory allocation
  - I/O device management
  - Interrupt handling

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.6.3.1 Classification of programming languages

3.6.3.2 Types of program translator

Note: Students do not need to be familiar with the terms first generation, second generation and third generation.

Specification content

- Show awareness of the development of types of programming languages and their classification into low-level and high-level languages.
- Know that low-level languages are considered to be machine code and 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’.
- Understand the role of each of the following: assembler, compiler and 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.
- Understand the difference between source code and object (executable) code.

Learning outcomes

- Know why low-level and high-level languages exist.
- Know what is meant by the terms machine code and assembly language and the advantages of these.
- Know the difference between compilation and interpretation and the role of an assembler, compiler and interpreter.
- Know what is meant by the terms intermediate language and imperative high-level language.
- Know the meaning of source code and object code.

Suggested timing

2 hours

Guidance

- **Class Discussion:** What is the difference between low-level and high-level languages? What is the difference between machine code and assembly language?

- **Research Work:** Ask students to research machine code and assembly language including:
    - Their features
    - Why they are used
    - The advantages and disadvantages of each

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Student Feedback:** Students should present their findings from their research to the class.

- **Class Discussion:** What is the purpose of a translator? What are the differences between compilation and interpretation?

- **Research Work:** Ask students to research:
    - The meaning of imperative high-level languages.
    - The role of an assembler, compiler and interpreter.
    - Situations when code would be compiled and interpreted.
    - The purpose of intermediate language.
    - The difference between source code and object code.

- **Student Feedback:** Students should present their findings from their research to the class.

Resources

- Notes on machine code and assembly languages:
    - geeksforgeeks.org/difference-between-machine-language-and-assembly-language/

- Notes on translators:
    - geeksforgeeks.org/language-processors-assembler-compiler-and-interpreter/
    - shiksha.com/online-courses/articles/difference-between-compiler-interpreter-and-assembler-blogId-159693

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7 Computer organisation and architecture

3.7.1 Internal hardware components of a computer

Specification content

- 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 and 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.
- Be able to describe the stored program concept.
- Be able to explain the difference between von Neumann and Harvard architectures and understand the advantages of each.

Learning outcomes

- Know different internal components within a computer system and the communication between these components.
- Know what is meant by the term addressable memory and the stored program concept.
- Know the difference between the von Neumann and Harvard architectures.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Name as many internal components as you can and briefly state the purpose of each.

- **Teacher Input:** Tell students the purpose of these components: processor, main memory, address bus, data bus, control bus and I/O controllers. If available, students would benefit from being shown physical components.

- **Research Work:** Ask students to research the internal components and make notes. In particular, they should research the communication between these components and how buses are used.

- **Student Feedback:** Students should present their findings of their research to the class.

- **Teacher Input:** Tell students what is meant by the terms addressable memory and the stored program concept.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

- **Research Work:** Ask students to research the von Neumann and Harvard architectures including:
    - Their structure
    - The difference between the two architectures
    - Their benefits
- **Student Feedback:** Students should present their findings from their research to the class.

Resources

Notes on the internal hardware components of a computer:

- adacomputerscience.org/concepts/arch_components?examBoard=all&stage=all
- youtube.com/watch?v=IksflGq6Vbw&list=PLCiOXwirraUA5PIzH22V-KXNfFmzd16yR
- youtube.com/watch?v=CCOJma3qfKA&list=PLCiOXwirraUA5PIzH22V-KXNfFmzd16yR&index=2
- zszszzcomputersciencecafe.com/31-processor-architecture.html

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7.2 The processor and its components

3.7.4 Factors affecting processor performance

Specification content

- Explain the role and operation of a processor and its major components.
- Factors affecting processor performance.

Learning outcomes

- Know the role of the processor and its main components.
- Know the factors affecting the performance of the processor.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** Name as many major processor components as you can and briefly state the purpose of each.

- **Teacher Input:** Tell students the purpose of these major CPU components: Arithmetic Logic Unit (ALU), control unit, clock, general-purpose registers and dedicated registers (program counter (PC), current instruction register (CIR), memory address register (MAR), memory buffer register (MBR), status register (SR)).

- **Research Work:** Ask students to research the internal processor components and make notes.

- **Student Feedback:** Students should present their findings from their research to the class.

- **Teacher Input:** Tell students the different areas that impact the performance of the processor: multiple cores, cache memory, clock speed, word length, address bus width and data bus width.

- **Research Work:** Ask students to research why these factors affect the performance of the processor.

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Notes on the processor and its components:
    - adacomputerscience.org/concepts/arch_processor?examBoard=all&stage=all
    - studyrocket.co.uk/revision/a-level-computer-science-aqa/fundamentals-of-computer-organisation-and-architecture/the-processor-and-its-components
    - youtube.com/watch?v=IksflGq6Vbw&t=12s

- Notes on the factors affecting processor performance:
    - adacomputerscience.org/concepts/arch_performance?examBoard=all&stage=all
    - studyrocket.co.uk/revision/a-level-computer-science-aqa/fundamentals-of-computer-organisation-and-architecture/processor-factors-affecting-processor-performance
    - youtube.com/watch?v=SQy_wg9oZl0

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7.3 The Fetch-Execute cycle and interrupts

Specification content

- 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.

Learning outcomes

- Know how a processor will fetch, decode and execute machine code and the registers and buses used during this process.
- Know the processor handles an interrupt.

Suggested timing

1.5 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** What do you think the terms fetch, decode, execute mean?
- **Teacher Input:** Tell students a clear definition of the processor stages (fetch, decode, execute) with clear reference to machine code.
- **Research Work:** Ask students to research how registers and buses are used during the fetch, decode, execute cycle.
- **Student Feedback:** Students should present their findings from their research to the class.
- **Class Discussion:** List as many example processes that would require the immediate attention of the CPU.
- **Teacher Input:** Tell students a clear definition of an interrupt and how these are handled with reference to the ISR.

Resources

- Notes on the fetch, decode, execute cycle and CPU interrupts:
  - adacomputerscience.org/concepts/arch_fe_cycle?examBoard=all&stage=all
  - youtube.com/watch?v=yyHFI5juppA
  - youtube.com/watch?v=PCYI1o592dQ
  - learnlearn.uk/alevelcs/secondary-storage-devices/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7.5 Secondary storage

Specification content

- Explain the need for secondary storage within a computer system.
- Know the main characteristics, purposes, suitability and understand the principles of operation of the following devices: magnetic hard disk and 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.

Learning outcomes

- Know the need for secondary storage and the differences between magnetic hard disk drives and SSD.
- Know the term cloud storage and its benefits and drawbacks.

Suggested timing

1.5 hours

Guidance

Possible teaching activities include:

- **Class Discussion:** What is the difference between secondary storage and main memory? Why does a computer system need both of these?

- **Research Work:** Ask students to research the difference between a magnetic hard disk and solid-state drive (SSD), including:
    - main characteristics
    - purpose
    - suitability

- **Student Feedback:** Students should present their findings from their research to the class.

- **Teacher Input:** Present various scenarios to students and ask them to make judgements about which type of secondary storage is the most suitable.

- **Teacher Input:** Tell students what is meant by the term cloud storage.

- **Research Work:** Ask students to research the benefits and drawbacks of cloud storage compared to local storage.

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Notes on secondary storage and cloud storage:
    - studysmarter.co.uk/explanations/computer-science/computer-organisation-and-architecture/secondary-storage/
    - learnlearn.uk/alevelcs/cloud-computing/

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7.6 Logic gates

Notes:

- The specification states the logic gate symbols and notation that will be used in exams.
- 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.

Specification content

- Construct truth tables for the following logic gates: 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.
- 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.
- Be familiar with the use of the edge-triggered D-type flip-flop as a memory unit.

Learning outcomes

- Know the different logic gates and use them to construct truth tables and draw logic circuits and their associated truth tables.
- Write a Boolean expression for a given logic gate circuit.
- Know, draw and trace the logic for a half-adder and know and trace the logic for a full-adder circuit.
- Know what edge-triggered D-type flip-flops are used for.

Suggested timing

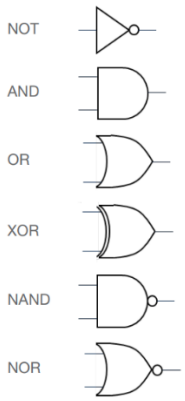
4 hours

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Guidance

Possible teaching activities include:

- **Teacher Demonstration:** Present the different logic gate symbols to students and clarify their meaning and associated truth table.



- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions requiring them to draw circuits with one or more of the above gates. Questions could also ask students to draw the associated truth tables for these circuits.

- **Teacher Demonstration:** Show students how to write a Boolean expression for different logic circuits using this notation:

    - NOT: $\overline{\ }$
    - AND: ·
    - OR: +
    - XOR: $\oplus$

- **Individual/Group/Paired Work:** Give students consolidation exercises containing logic circuits requiring them to write the Boolean expression.

- **Class Discussion:** Show students a half-adder circuit and full-adder circuit and ask students to determine what they do.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions that require them to draw and trace a half-adder circuit and trace a full-adder circuit.

- **Research Work:** Ask students to research D-type flip-flops including:
    - The purpose of the clock
    - The data input
    - How they work

- **Student Feedback:** Students should present their findings from their research to the class.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

Resources

- Worksheets with consolidation exercises containing:
    - questions requiring students to draw one or more logic gates.
    - logic circuits requiring students to write the Boolean expressions.
    - questions requiring students to draw and trace a half-adder and trace a full-adder circuit.
- Notes on logic gate symbols and Boolean expressions:
    - electronics-tutorials.ws/boolean/bool_7.html
    - adacomputerscience.org/concepts/boolean_logic_gates?examBoard=all&stage=all
    - youtube.com/watch?v=41Xa1hDl5K0
    - youtube.com/watch?v=YhZJKquQbS4
- Notes on adder and flip-flop circuits:
    - youtube.com/watch?v=kT1uRfQL2FY
    - youtube.com/watch?v=b0i1ShnNFKM
    - mrgoff.com/aCS_CS_adders

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.7.7 Boolean algebra

Note: The specification contains detailed additional information that should be used when simplifying Boolean expressions.

Specification content

- Be familiar with the use of Boolean identities and De Morgan's laws to manipulate and simplify Boolean expressions.

Learning outcomes

- Simplify Boolean expressions using Boolean identities and De Morgan's law.

Suggested timing

2 hours

Guidance

Possible teaching activities include:

- **Practical Exercises:** Why do you think it is important to simplify expressions?

- **Teacher Demonstration:** Show students how to simplify expressions using:
    - The order of precedence of operators
    - The laws of:
        - Distributivity
        - Associativity
        - Commutativity
    - Boolean identities as indicated in the specification
    - De Morgan's laws

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions requiring them to simplify Boolean expressions using the guidance given in the specification.

Resources

- A worksheet with questions for students to simplify Boolean expressions using the guidance given in the specification.
- Notes on Boolean algebra:
    - filestore2.aqa.org.uk/resources/computing/AQA-7516-7517-TG-BA.PDF
    - pmt.physicsandmathstutor.com/download/Computer-Science/A-level/Notes/AQA/06-Fundamentals-of-Computer-Systems/Intermediate/6.5.%20Boolean%20Algebra%20-%20Intermediate.pdf
    - youtube.com/watch?v=HoH0PrS3WNI&list=PLCiOXwirraUA9HyHoqOaGBU_k6nBRCb22&index=6&t=91s

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

3.8 Machine code and assembly language

3.8.1 Instruction format

3.8.2 Assembly language programming

Note: 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 the specification in section 6 Appendix: Standard OxfordAQA assembly language instruction set.

Specification content

- 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.
- Understand and apply immediate, direct and indirect addressing modes.
- Know that in machine code instructions are expressed in binary and that in assembly language they are expressed as mnemonics.
- Understand and apply the basic operations (load, add, subtract, store, branching (conditional and unconditional), compare, logical bitwise operators (AND, OR, NOT, XOR), logical (shift right, shift left), halt).
- Use the basic operations above to write, trace and reason about assembly language programs using immediate, direct and indirect addressing modes.

Learning outcomes

- Know the purpose of an instruction set and the difference between an opcode and operand.
- Know and apply different addressing modes.
- Know how instructions are expressed in machine code and assembly language.
- Apply the basic operations to write, trace and reason about assembly language programs.

Suggested timing

4 hours

Guidance

Possible teaching activities include:

- **Teacher Input:** Tell students the purpose of an instruction set and the difference between an opcode and operand.

- **Research Work:** Ask students to research:

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.

    - How instructions are expressed in machine code and assembly language
    - Immediate addressing
    - Direct addressing
    - Indirect addressing
- **Student Feedback:** Students should present their findings from their research to the class.

- **Teacher Demonstration:** Tell students the different assembly language operations (load, add, subtract, store, branching (conditional and unconditional), compare, logical bitwise operators (AND, OR, NOT, XOR), logical (shift right, shift left), halt).

- Demonstrate how to use these operations to write assembly language.

- **Individual/Group/Paired Work:** Give students consolidation exercises containing questions requiring them to write assembly language using the standard OxfordAQA assembly language instruction set found in the specification.

Resources

- The standard OxfordAQA assembly language instruction set.
- Worksheet containing questions requiring students to write assembly language using the standard OxfordAQA assembly language instruction set.

Copyright © 2024 Oxford International AQA Examinations and its licensors. All rights reserved.