From exam board		Link to the specification website				
Inferred and suggested by teachers		Download the specification as a PDF				
						All Isaac Computer Science Resources
Paper 1						**All Craig and Dave videos on YouTube**
Specification Ref	**Name of topic**	**Content**	🟥	🟧	🟩	**Resources to support (links on the classroom)**
4.1.1.16	Recursive techniques	Be familiar with the use of recursive techniques in programming languages (general and base cases and the mechanism for implementation). Be able to solve simple problems using recursion. **NOTE:** In order to understand the mechanism for implementation, you need to need to understand the following: **4.1.1.15 Role of stack frames in subroutine calls** Be able to explain how a stack frame is used with subroutine calls to store: • return addresses • parameters • local variables.				
4.2.1.2	Single- and multidimensional arrays (or equivalent)	Use arrays (or equivalent) in the design of solutions to simple problems. A one-dimensional array is a useful way of representing a vector. A two-dimensional array is a useful way of representing a matrix. More generally, an n-dimensional array is a set of elements with the same data type that are indexed by a tuple of n integers, where a tuple is an ordered list of elements.				

4.2.1.4	Abstract data types/data structures	Be able to distinguish between static and dynamic structures and compare their uses, as well as explaining the advantages and disadvantages of each. **Note:** it would also be helpful to understand the following concepts and uses of: • queues (linear, circular, priority) • stack • graph • tree • hash table • dictionary • vector.				
4.2.2	Queues	Be able to describe and apply the following to linear queues, circular queues and priority queues: • add an item • remove an item • test for an empty queue • test for a full queue.				
4.2.3	Stacks	Be able to describe and apply the following operations: • push • pop • peek or top • test for empty stack • test for stack full. Peek or top returns the value of the top element without removing it.				

4.2.4	Graphs	Be aware of a graph as a data structure used to represent more complex relationships. **AND** Be able to explain the terms: • graph • weighted graph • vertex/node • edge/arc • undirected graph • directed graph. **AND** Know how an adjacency matrix and an adjacency list may be used to represent a graph.				
4.2.5	Trees	Know that a tree is a connected, undirected graph with no cycles. ***NOTE:*** *a tree does not have to have a root.*				
4.3.1	Graph-traversal	Simple graph-traversal algorithms Be able to trace breadth-first and depth-first search algorithms and describe typical applications of both. Breadth-first: shortest path for an unweighted graph. Depth-first: Navigating a maze.				

4.3.4	Searching algorithms	**4.3.4.1 Linear search** Know and be able to trace and analyse the complexity of the linear search algorithm. Time complexity is O(n). **4.3.4.2 Binary search** Know and be able to trace and analyse the time complexity of the binary search algorithm. Time complexity is O(log n). **4.3.4.3 Binary tree search** Be able to trace and analyse the time complexity of the binary tree search algorithm. Time complexity is O(log n). **NOTE:** In order to understand binary tree search, you should also understand 4.3.2 Tree-traversal (4.3.2.1 Simple tree-traversal algorithms): Be able to trace the tree-traversal algorithms: • pre-order • post-order • in-order.				
4.3.5	Sorting algorithms	**4.3.5.1 Bubble sort** Know and be able to trace and analyse the time complexity of the bubble sort algorithm. This is included as an example of a particularly inefficient sorting algorithm, time-wise. Time complexity is O(n^2). **4.3.5.2 Merge sort** Be able to trace and analyse the time complexity of the merge sort algorithm. The 'merge' sort is an example of 'Divide and Conquer' approach to problem solving. Time complexity is O(nlog n).				

4.3.6	Optimisation algorithms	**4.3.6.1 Dijkstra’s shortest path algorithm** 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. **NOTE:** this links with 4.3.1.1 Simple graph-traversal algorithms				
4.4.1.1	Problem-solving	Be able to develop solutions to simple logic problems.				
4.4.1.2	Following and writing algorithms	Be able to hand-trace algorithms.				

4.4.4.3	Order of complexity	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 • logarithmic time • linear time • polynomial time • exponential time. **NOTE:** In order to understand Big-O fully, it is advisable to revise the following areas as well in section 4.4.4 Classification of algorithms: **4.4.4.1 Comparing algorithms** 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. Understand that some algorithms are more efficient: • time-wise than other algorithms • space-wise than other algorithms. Efficiently implementing automated abstractions means designing data models and algorithms to run quickly while taking up the minimal amount of resources such as memory. **4.4.4.2 Maths for understanding Big-0 notation** Be familiar with the mathematical concept of a function as a mapping from one set of values, the domain, to another set of values, drawn from the co-domain, for example $\mathbb{N} \rightarrow \mathbb{N}$.				

4.4.4.7	Halting problem	Describe the Halting problem (but not prove it), that is the unsolvable problem of determining whether any program will eventually stop if given particular input. 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.				
Paper 2			🟧			
Specification Ref	**Name of topic**	**Content**	🟥	🟧	🟩	
4.5.2	Number bases	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. Be familiar with, and able to use, hexadecimal as a shorthand for binary and to understand why it is used in this way				

4.5.3	Units of information	Know that: • the bit is the fundamental unit of information • a byte is a group of 8 bits. Know that the 2n 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, eg one kibibyte is written as 1KiB = 2^10 B and one kilobyte is written as 1 kB = 10^3 B. 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				
4.5.4.2	Unsigned binary arithmetic	Be able to: • add two unsigned binary integers • multiply two unsigned binary integers.				
4.5.4.3	Signed binary using two's complement	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.				

4.5.4.4	Numbers with a fractional part	Know how numbers with a fractional part can be represented in: • fixed point form in binary in a given number of bits • floating point form in binary in a given number of bits. Be able to convert for each representation from: • decimal to binary of a given number of bits • binary to decimal of a given number of bits.				
4.5.4.6	Absolute and relative errors	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.				
4.5.4.8	Normalisation of floating point form	Know why floating point numbers are normalised and be able to normalise unnormalised floating point numbers with positive or negative mantissas				
4.5.6.7	Digital representation of sound	Calculate sound sample sizes in bytes.				
4.5.6.8	Musical Instrument Digital Interface (MIDI)	Describe the purpose of MIDI and the use of event messages in MIDI. Describe the advantages of using MIDI files for representing music.				
4.6.1.2	Classification of software	Explain what is meant by: • system software • application software. Understand the need for, and attributes of, different types of software.				
4.6.1.3	System Software	Understand the need for, and functions of the following system software: • operating systems (OSs) • utility programs • libraries • translators (compiler, assembler, interpreter).				

4.6.1.4	Role of an operating system (OS)	Know that the OS handles resource management, managing hardware to allocate processors, memories and I/O devices among competing processes.				
4.6.2	Classification of programming languages	Know that low-level languages are considered to be: • machine-code • assembly language. Describe machine-code language and assembly language. Understand the advantages and disadvantages of machine-code and assembly language programming compared with high-level language programming.				
4.6.4	Logic Gates	Construct truth tables for the following logic gates: • NOT • AND • OR • XOR • NAND • NOR. Be familiar with drawing and interpreting logic gate circuit diagrams involving one or more of the above gates. Complete a truth table for a given logic gate 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.				

4.6.5	Boolean Algebra	Be familiar with the use of Boolean identities and De Morgan's laws to manipulate and simplify Boolean expressions.				
4.7.1	Internal hardware components of a computer	Be able to explain the difference between von Neumann and Harvard architectures and describe where each is typically used.				
4.7.2	The stored program concept	Be able to describe the stored program concept: machine code instructions stored in main memory are fetched and executed serially by a processor that performs arithmetic and logical operations.				
4.7.3.3	The processor instruction set	Understand the term 'processor instruction set' and know that an instruction set is processor specific. Know that instructions consist of an opcode and one or more operands (value, memory address or register).				
4.7.3.4	Addressing Modes	Understand and apply immediate and direct addressing modes.				
4.7.3.5	Machine-code/assembly language operations	Understand and apply the basic machine-code operations of: • 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 machine-code operations above when machine-code instructions are expressed in mnemonic form- assembly language, using immediate and direct addressing.				
4.7.4.1	Input and Output devices	Know the main characteristics, purposes and suitability of the devices and understand their principles of operation.				

4.7.4.2	Secondary storage devices	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: • hard disk • optical disk • solid-state disk (SSD).				
4.8.1	Individual (moral), social (ethical), legal and cultural issues and oppurtunities	Show awareness of current individual (moral), social (ethical), legal and cultural opportunities and risks of computing. Understand that: • developments in computer science and the digital technologies have dramatically altered the shape of communications and information flows in societies, enabling massive transformations in the capacity to: • monitor behaviour • amass and analyse personal information • distribute, publish, communicate and disseminate personal information. • computer scientists and software engineers therefore have power, as well as the responsibilities that go with it, in the algorithms that they devise and the code that they deploy. • software and their algorithms embed moral and cultural values. • the issue of scale, for software the whole world over, creates potential for individual computer scientists and software engineers to produce great good, but with it comes the ability to cause great harm. Be able to discuss the challenges facing legislators in the digital age.				

4.9.1	Communication	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. Define: • baud rate • bit rate • bandwidth • latency • protocol. Differentiate between baud rate and bit rate. Understand the relationship between bit rate and bandwidth.				
4.9.2.2	Types of networking between hosts	Explain the following and describe situations where they might be used: • peer-to-peer networking • client-server networking.				
4.9.3.1	The internet and how it works	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 Server (DNS) system.				
4.9.4.11	Thin- versus thick-client computing	Compare and contrast thin-client computing with thick-client computing.				

4.10.1	Conceptual data models and entity relationship modelling	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, ...).				
4.10.2	Relational databases	Explain the concept of a relational database. Be able to define the terms: • attribute • primary key • composite primary key • foreign key. **NOTE:** The content in this section will not be directly assessed but students will need to have an understanding of it to answer other questions				
4.10.3	Database design and normalisation techniques	Normalise relations to third normal form. Understand why databases are normalised.				
4.10.4	Structured Query Language (SQL)	Be able to use SQL to retrieve, update, insert and delete data from multiple tables of a relational database. Be able to use SQL to define a database table				
4.12.1.3	Function application	Know that function application means a function applied to its arguments.				
4.12.1.5	Compostition of functions	Know what is meant by composition of functions.				
4.12.2	Writing functional programs	Show experience of constructing simple programs in a functional programming language. Higher-order functions. Have experience of using the following in a functional programming language: • map • filter • reduce or fold.				