Computer Science Revision Guide

Draft

Hashan Punchihewa

1.1 Hardware

1.1.1 Structure and function of the processor

Registers

- **Register** - A register is a small area of memory inside the CPU to store data, with very high-speed access.

- **Program Counter (PC)** – A register which stores the address of the next instruction. It is incremented after being read. It controls the sequence in which instructions are executed. It is altered as a result of a “jump” instruction.

- **Accumulator (ACC)** - A register that stores the results of calculations by the CPU.

- **Memory Address Register (MAR)** - A register that stores the address of the memory location currently being either read from or written to.

- **Memory Data Register (MDR)** - A register that stores the data being either read from memory or written to memory. It acts as a buffer between memory and the CPU.

- **Current Instruction Register (CIR)** - A register that stores the instruction to be executed. It splits instructions into two parts: the opcode and operand. It also holds the opcode while it is being decoded. If the operand is an address it is sent to the MAR, if it is a value it is send to the MDR. It may also send an address to the PC for a “jump” instruction.

Components

- **Control Unit** - The control unit controls the fetch-decode-execute cycle and manages the execution of instructions, by transmitting control signals to other parts of the computer system. It also synchronises actions using its in-built clock.

- **ALU** - The ALU carries out arithmetic operations such as addition, and logical operations such as greater than comparisons. It also acts as a conduit through which all input/output to the computer goes.

Fetch-Decode-Execute Cycle

1. The contents of the program counter are copied into the memory address register.
2. The current instruction is copied from RAM into the memory data register.
3. The current instruction is copied from the memory data register into the current instruction register.
4. The program counter is incremented, such that it points to the next instruction.
5. The instruction is decoded.
6. The instruction is executed.

Factors affecting the performance of the CPU

- **Clock speed** - The higher the clock speed, the more fetch-decode-execute cycles the CPU can perform per second. The more instructions can be executed per second. The less time programs take to run.

- **Number of cores** - The more cores, the more instructions the CPU can execute concurrently. Therefore, more processes that can be run simultaneously.

- **Cache** - The more cache memory, more frequently accessed data and instructions can be stored in the CPU. This means RAM needs to be accessed less often. Significant as accessing cache is quicker than accessing RAM.

Exam Technique

If asked "*Describe how a fast clock speed improves a processor's performance*", mention not only that a faster clock speed means more clock cycles per second, but follow through your argument, and mention that this means there are more instructions executed per second, and therefore, a program takes less time to run.

CPUs v GPUs

CPU	**GPU**
MIMD	SIMD
Flexible	Optimised
Higher clock-rate	Lower clock-rate
Less cores	More cores
Complex instructions	Simple instructions

Buses

- **Bus** – A medium through which data is transferred between components in a computer system.

- **Address Bus** - Unidirectional bus used to transfer the address of memory to be read from or written to, from the processor to memory or input/output devices.

- **Data Bus** - Bidirectional bus used to transfer data between the processor and memory or input/output devices.

- **Control Bus** - Unidirectional bus used to transmit control signals e.g. 'read', 'write', etc. from the processor to memory or input/output devices.

Pipelining

- **Pipelining** - Where the stages of a CPU's fetch-decode-execute cycle are carried out at the same time on sequential instructions. This means that an instruction is fetched, at the same time as the previous one is being decoded, and the one before that is being executed.

Advantages of Pipelining	**Disadvantages of Pipelining**
Although instructions take the same time, the delay between completed instructions is reduced, increasing the throughput of the processor, increasing the clock rate.	Pipelining only works well, when a piece of code doesn't contain many jump instructions. This is as the processor might load the incorrect branch, in which case, the pipeline has to cleared and restarted.

Exam Technique

If asked "*Explain how pipelining would help a CPU execute the code more quickly*", then refer to what pipelining does, give an example and crucially mention whether the code has no jump instructions.

Architectures

- **Von Neumann Architecture** - This is where there is a single control unit, one instruction is processed at a time in a linear sequence in a fetch-decode-execute cycle and programs are stored with data. Allows self-modifying code.

- **Harvard Architecture** - Instructions and data are stored separately in their own memories, and each is accessed with a separate bus, meaning the CPU can fetch both instructions and data simultaneously. The memory for instructions may be read-only. The size of memory addresses and instruction addresses may be different.

- **Contemporary Architecture** - Instructions and data are stored separately in the CPU cache and there are separate pathways to the different types of data, thus when accessing data from cache, the CPU will behave as if it followed pure Harvard architecture. But instructions and data are stored together in main memory, and there is a single data bus between the CPU and main memory

1.1.2 Types of processors

Co-processors

- **Co-processor** – An additional processor used to supplement the functions and offload work from the main processor. It usually optimised for a specific purpose e.g. maths-intensive calculations.

- **GPU** – A co-processor optimised for maths-intensive calculations and highly data-parallel computation. An example of SIMD. Uses include processing polygon vectors for video games, bitcoin mining and machine learning.

RISC and CISC

- **Reduced Instruction Set Computing (RISC)** - A simple processor design, containing an instruction set with a limited number of instructions, where a single instruction performs a single task, and more complex tasks most be performed by combining simple instructions.

- **Complex Instruction Set Computing (CISC)** – A complicated processor design, containing an instruction set with many instructions, however a single instruction may take multiple clock cycles.

Features of RISC	Features of CISC
Simpler hardware	More complex hardware
Less heat	More heat
Further away from higher-level languages	Closer to higher-level languages
Longer code taking more space in main memory	Shorter code taking less space in main memory
Compiler has to do more work	Compiler has to do less work
Every instruction takes one clock cycle	Instructions may take more than one clock cycle
Limited number of addressing modes	Many addressing modes
Easier to carry out pipelining	More difficult to carry out pipelining

Parallel Systems

	Single instruction	**Multiple instructions**
Scalar processors	SISD (Single instruction, single data)	MISD (Single instruction, multiple data)
Array processors	SIMD (Single instruction, multiple data)	MIMD (Multiple instruction, multiple data)

Multi-core processors – Processors consisting of multiple processing units, that can each run different instructions simultaneously. (MIMD).

Array processor (or vector processor) – A processor that can perform the same instruction on multiple pieces of data simultaneously. (SIMD).

1.1.3 Input and Output

- **Input Device -** Hardware used by a user to put data into a computer system.

- **Output Device -** Hardware used to report information from a computer system to a user.

- **Random Access Memory (RAM)** - Volatile form of computer memory to store the running operating system, applications and open files. This means it is erased when the computer is switched off. Can be modified.

- **Read Only Memory (ROM)** - Non-volatile read-only form of computer memory to store the computer's boot sequence. Data is retained when the computer is switched off. Cannot be modified.

Barcode Readers

Barcode – A way of storing data using patterns of dark lines of different widths, pairs of which store digits. Barcodes typically have start or stop codes so they can be read both ways and a check digit for error checking. Typically, a barcode reader uses a laser or a light source to obtain a reflection of the lines.

Advantages of a Barcode Reader	Disadvantage of a Barcode Reader
Fast data entry	Barcodes can be damaged
Accurate	Can be difficult to get the barcode reader into the required position for larger items.

Scanners

Optical Character Recognition (OCR) – A device that transfers handwritten or printed text to editable text on a computer by optically scanning the text to determine the shape of characters. A use would be converting handwritten documents into a word processor file.

Optical Mark Recognition (OMR) – A device that recognises the presence of a mark on a sheet of paper by the reflection of light, the position of the marks conveys information to the machine. A use would be scanning in a multiple-choice test.

Magnetic Ink Character Recognition (MICR) – A device able to read information written in a special magnetic ink, which is also human-readable. A use would be reading in account numbers on cheques.

Printers

Advantages of a Dot Matrix Printer	Disadvantages of a Dot Matrix Printer
Possible to obtain multiple copies by using carbon paper	Slow and poor quality
	Not possible to print in colour

Advantages of an Inkjet Printer	Disadvantages of an Inkjet Printer
Faster and better quality than Dot Matrix	Needs to change colour cartridges often
Possible to print in colour	Less copies a minute than a laser printer

Advantages of a Laser Printer	Disadvantages of a Laser Printer
Faster than an Inkjet Printer	Expensive
Ink lasts longer so don’t need to change cartridges as often	Bulky and heavy

Magnetic Stripe Cards

Magnetic Stripe Card – A card that stores data by using the magnetism of a strip of magnetic material on the card. These are read using magnetic stripe readers.

Actuators

Actuator – A mechanical device or motor, used an output device for those who are visually impaired.

Sensors

These measure movement:

- **Gyroscope** – Sensor that measures changes in orientation.
- **Accelerometer** – Sensor that measures acceleration.
- **Altimeter** – Sensor that measures altitude.

Storage

Optical Storage – A storage medium where data is stored in a form that can be accessed through light.

Optical Discs (e.g. CDs/DVDs/Blu-Rays) – A secondary storage medium consisting of a disc, with data encoded onto it. A laser can be used to read data from it and write data to it. Bits are represented through how and whether light is reflected.

Magnetic Storage – A storage medium where bits patterns would be represented by different patterns of magnetism.

Hard Disk Drive (HDD) – A secondary storage medium, made of a disk consisting of one or more platters. Bits are represented by magnetised spots on the platters that rotate at high speeds. Platters are divided into tracks and tracks into sectors. A disk head which moves radially (in/out towards the centre of the platter) is used to write the data to the correct sector, as the sectors passes below it.

Magnetic Tape – A serial access magnetic secondary storage medium, consisting of plastic film, that stores data through magnetism.

Flash Storage – A non-volatile computer storage medium that relies on no moving parts, and instead stores data to NAND flash gates.

Virtual Storage

Virtual Storage – Virtual storage is where multiple physical storage drives are pooled together so they appear as a single logical storage drive. Examples include storage area networks, where many physical drives are accessed as a single logical drive across a network, and cloud storage, where cloud storage providers sell storage space on what appears to be a single logical pool, but which spans many physical drives.

Advantages of Cloud Storage	Disadvantages of Cloud Storage
Easily expanded	You need an internet connection.
Easily transferred	
Easily accessed	

1.2 Software

1.2.1 Systems Software

- **Operating System** – System software that manages the hardware of a computer system.

Functions of an operating system

- Memory management
- Processor scheduling
- Interrupt handling

Why is scheduling needed?

- Process as many jobs as possible in the least amount of time.
- Ensure jobs are processed fairly.
- Maximise the number of interactive users.
- Efficient use of processor time.

Scheduling

First Come First Served – A scheduling policy where jobs are executed in the order they enter the ready queue.

Shortest Job First – A scheduling policy where jobs are sorted in the ready queue in ascending order of the time required and new jobs are added in a way to preserve this order

Round Robin – A scheduling policy where each job is given a set amount of CPU time, and after that set amount, the job is interrupted and sent to the back of the ready queue; the next job in the queue then runs.

Round robin cannot act as a scheduling policy on its own, it must work in combination with another policy to choose which policy to go first. For example, first come first served in combination with round robin. Together they make a scheduling scheme.

Shortest Remaining Time - A scheduling policy where the job with the smallest expected time until completion is selected to run; if a new job is added to the ready queue with a smaller expected time than the currently running job then it will take over.

Multi-Level Feedback Queue – A scheduling policy where the processor categorises jobs into different queues, with different priorities. Jobs in a queue of higher priority are always chosen to be executed over jobs in a queue of lower priority, and the scheduler takes jobs from the front of the queue. Jobs are usually inserted at highest priority queue. Jobs that take up the entire time slice are demoted to the next lowest queue. Jobs always enter at the tail of a queue. The lowest queue uses round robin. In some implementations, when a job blocks for I/O, the job can be promoted to favour I/O bound jobs.

Why is memory management needed?

- Organise the use of main memory by converting logical addresses to physical addresses

- Allow programs to share memory
- Protect programs from using the same memory
- Allow programs larger than main memory to run.

Memory Management

- **Paging** – A memory management scheme used with virtual memory where memory is partitioned into physical pages and data are divided into logical pages. Logical pages are stored on disk, and logical pages are assigned physical pages and loaded into memory as they are needed. A page table records which physical page holds which logical page. Pages are of the same size. Pages are not allocated contiguously in memory.

- **Segmentation** – A way of partitioning where data is divided into segments which are of different sizes. Segments are stored on disk, and loaded into memory as they are needed. Segments can represent complete programs, that can be executed separately.

Programmers choose the segments, but not pages.

- **Virtual Memory** – Where an area of the secondary storage is used to supplement main memory, when main memory becomes full. Pages or segments are swapped between main memory and secondary storage. This happens so that parts of the program not currently in use may be run, allowing large programs to run.

- **Disk Thrashing** – A situation which occurs when using virtual memory, where there is a very high rate of disk accesses, such that more time is spent swapping pages than processing tasks.

BIOS

- **BIOS** – Software that runs immediately after a computer starts up to check that the system hardware works, by running a power-on self-test (POST), and loads a boot loader or an operating system into main memory.

Interrupts

- **Interrupt** – An interrupt is a signal sent to the processor from a device/process needs indicating it needs attention. Examples of interrupts include a power failure interrupt, a system clock interrupt and an I/O interrupt. Different interrupts have different priorities based on their importance.

- Why are they used?
  - To obtain processor time for a higher priority task.
  - To indicate a device/process needs attention.
- How a computer handles an interrupt:
  1. Complete the current fetch-decode-execute cycle
  2. Check the priority of the interrupt
  3. If it is of a lower priority than the current task, then stop.
  4. Push the contents of the CPU’s registers onto the stack

5. Load the relevant interrupt service routine by setting the program counter to the first instruction of the interrupt service routine
6. After the interrupt service routine is finished, restore the values of the registers, by popping them from the stack
7. The original process continues.

- **Interrupt Service Routine (ISR)** – A procedure in an operating system or device driver which is triggered to handle an interrupt.

- **Device Driver** – System software that operates or communicates with a hardware device attached to a computer. It also provides a software interface to the hardware device and interrupt service routines for handling any interrupts from the hardware.

Types of Operating Systems

- **Real Time Operating System** – An operating system that gives a response within a guaranteed time frame.

- **Distributed Operating System** – An operating system that runs allows multiple computers, connected through a network, to work together on the same problem.

- **Embedded Operating System** – An operating system which is designed for an embedded computer system with one specific function.

- **Multi User Operating System –** An operating system, which allows multiple users to use the system at the same time. Each user is allocated their own rights and user files are kept separate.

- **Multi-tasking Operating System –** An operating system is an operating system that only more than one process to run simultaneously. This is implemented using time-slicing, where each process is given a fixed amount of processing time, being the next process resumes. Processor is so fast that appears these tasks care carried out simultaneously.

Virtual Machines

Virtual Machine – A software program that is able to perform as if it were a separate computer and execute code as if it were a separate computer. Multiple virtual machines may co-exist on a single “host” computer simultaneously.

There exist 2 types of virtual machines:

System Virtual Machine – A virtual machine to emulate a whole operating system, within a host operating system, by executing its machine code in a virtualised environment.

Process Virtual Machine (or Application Virtual Machine) – A virtual machine to execute a single application inside a host OS, within a platform-independent virtualised environment. Process Virtual Machine will often run intermediate code (e.g. Java Virtual Machine), or execute code on behalf of an interpreter (e.g. Python).

Intermediate Code – Partially compiled code that is not machine-specific that can be run on any computer with the appropriate virtual machine.

Note that source code is compiled to intermediate code and the intermediate code is either interpreted.

1.2.2 Application Generation

- **Utility software** – A piece of software that performs a specific task to aid the maintenance and upkeep of a computer system. An example would be a disk defragmenter.
- **Application Software** – Software created for to perform specific and useful tasks for the user.
- **System Software** - Programs that control the operation and hardware of the computer system.

Open and Closed Software

- **Open source software** – A piece of software for which the source code is freely available for others to examine and amend.

Advantages of Open-Source Software	Disadvantages of Open-Source Software
Likely to be free	No guarantee of quality
Custom modifications can be made to the source code	
Copies of the source code can be made and distributed freely	

Advantages of Closed-Source Software	Disadvantages of Closed-Source Software
Likely to be of high-quality	Source code cannot be modified
There is usually a company who can provide support and fix bugs	Likely to cost money
	You cannot make copies or distribute the software

Compilation

- Lexical analysis:
  - Source code is used as input.
  - Tokens are created from individual symbols and from the reserved words in the programming language.
  - Variables are stored in a symbols table.
  - Redundant characters e.g. spaces are removed.
  - Whitespace is also removed.
  - Error diagnostics are given.
  - Code is prepared for syntax analysis

- Syntax analysis:

    - Label check, check whether a procedure or variable you reference actually exists
    - Checks whether control structures e.g. loops are closed correctly
    - Check if variables have been assigned illegal values
    - Does BIDMAS

- Semantic analysis:
    - Makes sure a variable is declared before it is used
    - Type checking

Syntax and semantic analysis both add more information to the symbols table.

- Code generation:
    - Produces machine code.
    - There are several instructions for each high-level language statement. All variables are given addresses.
    - Code optimisation is where code is made as efficient as possible e.g. by replacing redundant instructions.
    - This is to remove the number of instructions, so the size of the executable is smaller.
    - It is also to reduce processing time, so the program runs faster.

- **Assembler** - A program that translates assembly code into machine code.

- **Compiler** – A translator which converts source code to machine code to produce an executable program.

- **Interpreter** – A translator that translates one statement at a time, then allows it to be rune, before moving onto the next statement.

Compilers	**Interpreters**
Translates the whole program as a unit and creates an executable program	Translates one statement at a time before translating the next
Executable program is architecture specific	Reports one error and its position before stopping
May report a number of errors at once	Must be present each time the program is run
Can optimise code to improve speed/size	Code runs in a virtual machine
	Program runs more slowly due to translation
	Can run on a variety of devices improving portability

- **Linker** – Used to compile pieces of code that have already been compiled to produce a single executable file. It does this by dealing with references from the main program to other compiled modules.

- **Loader** – Copies the executable code into primary memory ready for execution.

- **Library** – A collection compiled sections of code that can perform useful tasks that can be used within other computer programs.

- Advantages of library programs:
  - Already been tested, so likely to be error-free
  - Already been written, so saves time
  - Having been written it is reusable
  - Written by experts, so code likely to be more efficient

1.2.3 Software Development

- **Waterfall Model** – A sequential development life cycle, where one stage is completed fully before moving onto the next stage. There is a heavy emphasis on planning. If you have to go back to a previous stage, you have to repeat all stages after that. Stages of the waterfall lifecycle:
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Testing
  - Maintenance

Advantages of the Waterfall Model	Disadvantages of the Waterfall Model
Simple and easy to manage	No space for flexibility
Well defined boundaries	Requirements cannot change
Easy to spot problems that might come up	End user not consulted after requirements

- **Spiral Model** – A risk-driven development life cycle, where development is conducted in a series of waterfall model-based iterations. These iterations end with a phase of risk analysis.

Advantages of the Spiral Model	Disadvantages of the Spiral Model
Focussing on risk	Very slow
	Getting someone who can do risk management properly is costly

- **Agile Methodology** – A set of principles for development life cycles which focusses on adaptive planning, early delivery and continuous improvement. Development occurs in incremental iterations, which create working prototypes. Development can adapt to changing requirements, and feedback can be received from clients early on.

- **Extreme Programming** – A development life cycle where development is conducted is a series of short cycles each producing a new release, where there is an emphasis of receiving continuous feedback from the client. There is also an emphasis of fixing bugs early on e.g. by writing unit tests before programming. Extreme programming also emphasises integrating and testing the system several times a day.

Advantages of the Extreme Model	Disadvantages of the Spiral Model
Very high-quality code	Costs a lot
Very efficient code	Requires regular contact with end user

- **Rapid Application Development** – A development life cycle where development is conducted is a series of short cycles each producing prototypes. The cycles iteratively improve the prototype till it becomes the final product. Little time is spent planning, as opposed to programming.

Advantages of the RAD	Disadvantages of the RAD
User is involved	Regular contact with client needed
Program very specific to requirements	Not very good for massive projects
	Not very good if you need efficient code

- Rapid Application Development and Extreme Programming are in essence agile methodologies.

1.2.4 Types of Programming Languages

- **Class** - A template defining methods and attributes, that can be instantiated to make objects.

- **Inheritance** - Where a class known as the child class or subclass derives the methods and attributes of a parent class or superclass. The subclass may override some of these methods and attributes, and define its own additional attributes and methods.

- **Assembly language** - A programming language that is closely related to the specific machine architecture, using mnemonics for instructions, and labels to allow jumping to other instructions. Each instruction is translated into one machine code instruction.

- **Procedural programming** – A programming paradigm where the program consists of a series of instructions that tell the computer what to do to solve the problem. It consists of sequence, selection, iteration and recursion. Code is written in blocks are procedures that can be called at any point in a program's execution.

- **Object-oriented programming** – A programming paradigm where details of the implementation are abstracted away using structures called classes, that can be reused, to reduce the amount of code and make programs easy to maintain.

- **Declarative programming** – A programming paradigm where you write a statement describing the problem you want to be solved, but the language implementation will decide how to solve it.

- **Logic programming** – A programming paradigm is where there is a set of facts and a set of rules, that are used to perform computation.

- **Functional programming** – A programming paradigm that treats computation as the evaluation of mathematical functions which accept input and return output.

Polymorphism – Where different subclasses of a superclass have different implementations of the same method.

Encapsulation – Where attributes of a class are declared as private, so they can only be accessed and modified through methods.

LMC Assembly

Mnemonic	**Instruction**
ADD	Add
SUB	Subtract
STA	Store
LDA	Load
BRA	Branch always
BRZ	Branch if zero
BRP	Branch if positive
INP	Input
OUT	Output
HLT	Halt program
DAT	Data location

Modes of Addressing Memory

- **Immediate Addressing** – The operand is the actual value to be operated on.
- **Direct Addressing** – The operand holds the memory address of the value to be operated on.
- **Indirect Addressing** – The operand is the location (typically a register) of the memory address of the data to be operated on.
- **Indexed Addressing** – The location of the data to be operated on is the contents of the index register added to some constant value.

An advantage of indirect addressing is that you can access more memory. Indexed addressing is used when manipulating arrays.

1.3 Data

1.3.1 Compression, Encryption and Hashing

- **Lossy Compression** - An algorithm that reduces the file size to represent data, but accuracy with which it represents the data is lost in the process.

- **Lossless Compression** - An algorithm that reduces the file size to represent data, such that the original data can be recovered without any loss in quality.

- **Run-length encoding** – A lossless compression algorithm where data is stored as a piece of data, e.g. a pixel in an image, followed by how many times it is repeated. This saves space when applied to files with long runs of the same item of data.

- **Dictionary coding** – A lossless compression algorithm where data is encoded using a dictionary that stores sets of strings that can be found in the data, the strings in the data are substituted with a reference to the position of the string in the dictionary.

- **Encryption** – Where data is transformed from one form, the plaintext, to another form, the ciphertext to prevent unauthorised third parties being able to understand it. The algorithm used to perform the encryption is called the cipher, and secret information to encode it is called the key.

- **Symmetric encryption** – Encryption where the same key can be used to encrypt and decrypt the data.

- **Asymmetric encryption** – Encryption where two different keys are used. A public key, which is made public, is used to encrypt data to send to the person who produced the keys. The private key, which is kept secret by this person, is used to decrypt data encrypted using the public key.

- **Hashing** – A hashing function provides a mapping between an arbitrary length input to data of a fixed size. It is one-way, as there is no way to obtain the original input from the hash. Uses of hashing include hash tables, and storing and authenticating people’s passwords.

1.3.2 Databases

- **Database** – A persistent store of data.

- **Flat-File Database** – A database where data is stored as a single table sequentially in a file with no structured relationships.

- **Relational Database** – A database where data is stored across multiple tables, with structured relationships between tables.

- **Table** – A collection of records.

- **Record** – All the data stored about a single entity.
- **Field** (also an attribute) – A single piece of information about a record.
- **Primary Key** – An attribute that uniquely identifies every single record in a table.
- **Foreign Key** – A non-primary key attribute in one table that stores values from a primary key attribute of another table, thus creating a relationship between the two tables.
- **Secondary Key** – A field whose data is stored in an index to allow fast queries involving the field.

Advantages of a Relational Database

- Reduces data redundancy
- Improves data consistency
- Easier to change data
- Easier to add data
- Different data access levels of data

Advantages of a Flat File Database

- Simple and easy to manage

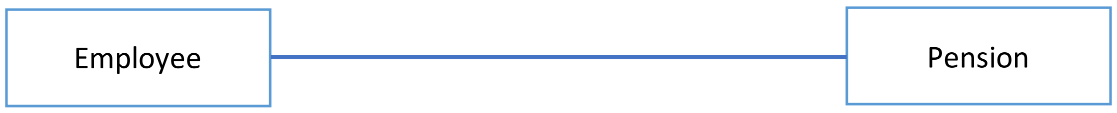
Data Definition Language (DDL) – Commands for defining the structure of a database e.g. creating/removing/modifying tables and columns

Data Manipulation Language (DML) – Commands for manipulating data within a database e.g. inserting/removing/modifying/querying rows in a table

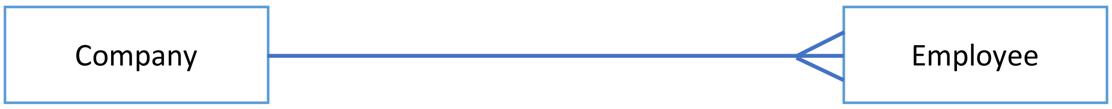
Data Dictionary – A file defining the structure of a database e.g. what tables there are, what columns are in each table, the data type of each column.

Database Management System (DBMS) – A computer program to interact with and manage a database e.g. running queries, generating reports, changing schemata, inserting data, creating forms.

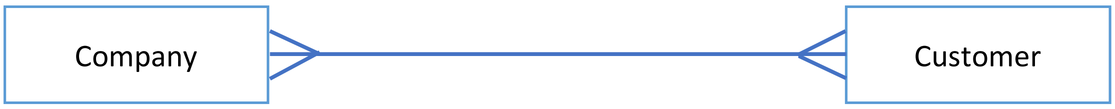
One to one relationship:



One to many relationship:



Many to many relationship:



Many to many relationships have to be stored using an associative table, that contains the two entities as foreign keys.

Normalisation

Normalisation - The process of analysing how to make databases more efficient by using separate tables to reduce redundant data.

Advantages of Normalisation
No redundant data, database is smaller in size
Smaller database, so queries run faster
No redundant data, means there is likely to be better data integrity

- **Unnormalised Form** – The table contains repeating groups of data.
- **First Normal Form** – The table contains no repeating groups of data. For instance, in a single field you cannot have stored 4 telephone numbers, separated by a comma. Each telephone number should be stored in its own record. If records are split into multiple records due to attributes containing repeating groups of data, then the new primary key is a composite key consisting of the original primary key field, and the attributes, that contained the repeating groups.
- **Second Normal Form** – There are no partial key dependencies, i.e. there is no attribute that only depends on part of the primary key. If this is the case, then the fields that only depend on part of the primary key, should be moved into their own table, where the primary key is the part of the original key that they depend on. Then, a foreign key needs to be added to the original table, in place of part of the primary key.
- **Third Normal Form** – There are no non-key dependencies, i.e. there are no fields that depend on an item of data that is not the primary key. If this does occur, then the fields should be moved into their own table, where the field they depend on becomes the primary key. Then a foreign key needs to be added to their original table, in place of the dependent key that was moved to its own table.

Query	**Meaning**
`SELECT * FROM people`	Get all the information about everybody stored in the database.

`SELECT name FROM people`	Get the name of everybody stored in the database.
`SELECT name, age FROM people`	Get the name and age of everybody stored in the database.
`SELECT * FROM people WHERE name = “Hashan”`	Get all the information about everybody called Hashan in the database.
`SELECT * FROM people WHERE name = “Hashan” AND age = 18`	Get all the information about every called Hashan and whose age is 18.
`SELECT * FROM people WHERE age > 10 AND age < 60`	Get all the information about people whose age is between (exclusive) 10 and 60.
`SELECT * FROM people WHERE age >= 10 AND age <= 60`	Get all the information about people whose age is between (inclusive) 10 and 60.
`SELECT * FROM people WHERE age != 18 OR name = “Hashan”`	Get all the information about everybody called Hashan or whose age is not 18.
`SELECT * FROM people WHERE age != 18 OR name LIKE “%Hashan”`	Get all the information about everybody whose age is not 18 or whose name ends with “Hashan”.
`SELECT * FROM people WHERE age != 18 OR name LIKE “Hash%an”`	Get all the information about everybody whose age is not 18 or whose name begins with “Hash” and ends with “an”.
`SELECT * FROM people WHERE pet_name IN (SELECT name FROM dog)`	Select every field of every record from a table called *people*, where the value of *pet_name* is an item that also happens to be a name of a record in *dog*.
`SELECT * FROM people WHERE pet_name IN (SELECT name FROM dog WHERE breed = “Rottweiler”)`	Select every field of every record from a table called *people*, where the value of *pet_name* is an item that also happens to be a name of a record in *dog* where the breed is “Rottweiler”.
`SELECT people.name, pet.breed FROM people JOIN pet ON people.pet_name = pet.name`	Selects the field’s *name* from *people* and *breed* from *pet*, joining the records together when the value of *pet_name* from a record from *people* has the same value as the value of *name* from a record from *pet*.
`SELECT people.name, pet.breed FROM people JOIN pet ON people.pet_name = pet.name WHERE name = “Hashan”`	Selects the field’s *name* from *people* and *breed* from *pet*, joining the records together when the value of *pet_name* from a record from *people* has the same value as the value of *name* from a record from *pet*. The *name* also has to be “Hashan”.
`DELETE FROM people`	Delete every record that exists in table *people*.
`DELETE FROM people WHERE name = “Hashan”`	Delete every record that exists in table, where name is “Hashan”.
`INSERT INTO people (name, age) VALUES (“Hashan”, 18)`	Insert a record into the table *people*, with name assigned to “Hashan” and age to 18.

`INSERT INTO people (name) VALUES (“Hashan”)`	Insert a record into the table *people*, with name assigned to “Hashan”.
`DROP TABLE` *`people`*	Completely delete a table including the data it contains.

- **Data Integrity** – This is where data is stored under the constraints of the database, e.g. the values of records are the correct type and validation rules are adhered to.
- **Referential Integrity** – This is where every piece of data stored under a foreign key refers to another row that exists in another table.

Transactions, ACID, Record Locking, Redundancy

Operations to modify a database are carried out in transactions. Transactions is one or more changes that are to be made to a database. A single transaction can contain changes to different tables. They must adhere to the 4 properties:

- **Atomicity** – A transaction is processed in its entirety or not at all. This means every change has to made, or none of the changes are made.
- **Consistency** – No transaction can violate any of the validation rules for maintaining the integrity of the database, this includes referential integrity.
- **Isolation** – The transaction is carried out independently, and each transaction should not affect other transactions.
- **Durability** – Once a transaction has been committed, it remains so, even in the event of a power cut.

- **Record Locking** – A technique of preventing simultaneous access to objects in a database to prevent inconsistent results. This is where a record is locked when it is to be updated, so that nobody else can access the same record, until the update is finished.

- **Redundancy** – This is where two or more identical database systems are maintained in different geographical locations, so that every transaction is applied to all database systems. In the event of one system failing, other systems can take over.

1.3.3 Networks

- **Local Area Network** – A group of computers linked together over a small geographical area.
- **Wide Area Network** A group of computers over geographically remote distances, usually connected through third-party communication links.

Characteristics of LAN
- Small geographical area
- More secure
- Hard-wired or short-range wireless

Characteristics of WAN

- Data vulnerable to interception
- Geographically remote distances
- Tends to use third-party communication links

Client-Server Network - One or more nodes act as a central server which centrally stores data and manages the network, while the rest are clients, which rely on the server.

Peer-to-Peer Network - No node is in overall control, each can share data and connected peripherals to others based on a set of access rights.

Advantages of Client-Server	**Disadvantages of Client-Server**
Files are stored in a central location, and different users can be given different access rights	A dedicated network manager is needed
Peripherals, backups and security can be controlled centrally	Servers are expensive
Software licenses and installation can be controlled centrally	If the servers fails, none of the clients will function

Advantages of Peer-to-Peer	**Disadvantages of Peer-to-Peer**
Easier to setup than Client-Server	Files are disparate, stored across multiple machines making them difficult to locate
If one node fails, other noes will still function	Backups and security cannot be controlled centrally, so it is the responsibility of each user to properly backup and secure his/her computer
No need for a dedicated network manager or additional nodes, to act as a server	

Protocols

Protocol – A set of rules that govern communication in a network.

The layers of the TCP/IP model can split into the following:

- Application Layer (e.g. HTTP) – Encodes the data being sent.
- Transport Layer (e.g. TCP) – Splits the data up into packets, assigns packet headers and adds port number.
- Network Layer (e.g. IP) – Adds the sender's and the recipient's IP address
- Network Access Layer (e.g. Ethernet or Wi-Fi) – Adds MAC address information to specify which hardware device the message comes from and is going to.

IP Address – An address assigned to devices to indicate where a packet of data is to be sent or has been sent from. Routers are assigned IP addresses from Internet Service Providers. These are public IP addresses and are unique across the internet. Routers provide private IP addresses to computers within a network. Private IP addresses are only unique within a network.

Features of the IP Protocol

- Adding the sender's and the recipient's IP address

Features of the TCP Protocol

- Allocation of port numbers
- Splitting data into packets
- Retransmitting packets if they were lost in transmit
- Checking data whether data has been changed during transmit, and attempting to correct it if it has (i.e. error correction)

Features of the Data Link Layer

- Adding the MAC address

Advantages of protocol layering

- Higher level protocols do not have to worry about the implementation details of lower level protocols and vice versa
- Higher level protocols can often work on top of multiple lower level protocols e.g. IP packets can be sent on Ethernet, which requires IP to run on top of the Ethernet protocol, or IP can be sent on Wi-Fi, wherein IP runs on top of the Wi-Fi protocol.
- Able to code for specific layers.

Disadvantages of protocol layering

- Overhead due to a message having to pass through multiple layers, increasing packet sizes.

MAC (Machine Address Code) address - A unique hardware identifier for devices that can connect to a network, consisting of 48 bits of data. A MAC address is usually built into the NIC of a computer. No two devices will ever have the same MAC address.

URL stands for Uniform Resource Locator. In a web address, *http* is the protocol. *www* is the host. *www.bbc.co.uk* is a fully qualified domain name. *uk* is the top-level domain name. The path is */news*.

Network Hardware

Hub - A hardware device that connects together multiple nodes in a network. A hub broadcasts all the data received from one port to all the other ports, regardless of to whom the packets are addressed. This uses up a lot of unnecessary bandwidth.

Bridge - A hardware device that connects together two networks, such that they act as a one larger network. A bridge checks to see if the recipient of a message is on the other network before relaying a packet.

Switch - A hardware device that connects together multiple nodes in a network, but rather than broadcasting all data, checks the MAC address of the recipient and sends it to the correct node, as opposed to all the nodes.

Modem - A hardware device that converts to and from digital data that a computer network uses to and from analogue signals that telephone network uses.

Wireless Access Point (WAP) - A hardware device that allows wireless devices to connect and access the network by relaying connections through to the hard-wired network the WAP is connected to.

Router - A network device that can form a LAN by connecting network-capable devices together, and chooses the node to forward packets such that packets follow the fastest route possible.

Repeater - A network device that receives degenerated signals and sends them "regenerated", in order to prevent signals degenerating too much such that they can no longer be understood.

DNS

Domain Name System – The system by which human-readable domain names (e.g. google.com) are converted into the corresponding IP addresses. A DNS server maps a domain name into an IP address. If it cannot resolve the domain name, it recursively passes the request to another DNS server, until the IP address is found.

Why is DNS used?

- Because IPs are hard to remember.
- Because IPs can change.

Network Security

- **Malware** – A program designed to disrupt, damage or gain unauthorised access to a computer system.
- **Firewall** – A piece of software or hardware that inspects packets against a set of preconfigured rules called packet filters, to decide whether the packet should be allowed through.
- **Proxy Server** – A proxy server intercepts all packets entering and leaving a network, hiding the true network address of the source from the recipient. This enables privacy. A proxy can also be used to filter requests to control the content that users can access. It can also be used to maintain a cache of commonly requested websites to save time.
- Encryption can also be used to keep messages travelling across the internet secure.

Packet and Circuit Switching

Packet Switching	**Circuit Switching**
No established route	Establishes a route along which to send packets for the duration of the message
Packets are sent on individual routes	Packets all follow the same route
Secure because impossible to intercept all packets	Because of all packets follow the same route, easy to intercept
Packets need to be reassembled	Packets remain in the correct order

Maximises the use of network	Ties up large areas of the network

1.3.4 Web Technologies

HTML, CSS and JavaScript

Basic Elements One Should Know:

Element	**Attributes**
`<html>`	
`<head>`	
`<title>`	
`<body>`	
`<h1>`	
`<h2>`	
`<h3>`	
`<a>`	`href`
`<div>`	
`<form>`	
`<p>`	
`<li>`	
`<ol>`	
`<ul>`	
`<script>`	`name, type=”text”, type=”submit”`

Elements That Do Not Require a Closing Tag:

Element	**Attributes**
`<link>`	`href, rel, type`
`<img>`	`src, alt, height, width`
`<input>`	

The only other attribute that one needs to know is the *style* attribute for all applicable elements. This is used for inline styles. Inline styles are bad because:

- Create redundant code.
- More code, means loads slower.
- Difficult to maintain.

External styles are where they are rewritten in a different file and linked with a *link* tag. Advantages are the opposite of the disadvantages of inline styles.

CSS Properties That One Should Know:

Property
`background-color`
`border-color`
`border-style`

`border-width`
`font-family`
`font-size`
`height`
`width`

One should know how to use CSS identifiers with ids, classes and tag names as well as CSS named and hexadecimal colours.

One should know how to use the JavaScript functions`,` and. One should also know of the `innerHTML` property of elements.

JavaScript	**Description**
`document.getElementById`	Method
`document.write`	Method
`alert`	Method
`innerHTML`	Property of an element

Search Engine Indexing

Search engines maintain an index of all pages on the internet, which your query is compared to. Querying an index is designed to be very fast. The index is continually updated by the search engine indexing algorithm, by removing pages, adding new ones, or updating the rank of each page. A spider is used by the search engine to crawl between pages, and index every page it finds. It checks meta tags, which contain data about the webpage, and follows all of the links on a page in order to collect data about a page. The more links a page has, and the higher the quality of those links, the higher ranked a page is in the index, so the easier it is to find with a query.

Client-side and Server-side processing

As the name suggests client-side processing is where data is processed on the client in the web browser, usually using JavaScript. Server-side is where the data is sent to the server e.g. through a HTML form, for processing.

Client-side processing	**Server-side processing**
Initial data validation	Further data validation
Can applies CSS styles	Performs database queries and updates
Reduces the load on the server	Performs complex calculations
	Encodes data to readable HTML

Client-side validation can be bypassed e.g. by forging server-side requests, so server-side validation always has to be implemented. So, if you do implement client-side validation, you still ned to implement server-side validation.

1.4 Data Types

1.4.1 Data Types

- **Primitive data type** – A data type built in to a programming language.

- One should be aware of integers, floating point numbers, characters, strings and Booleans.

Positive Binary Integers

All data in a computer system is stored as a collection of 0s and 1s, known as binary. The way meaningful data be it numbers or characters are encoded is called data representation. Binary can be easily used to represent numbers as all it is in essence is a numerical system, except with only with the digits 0 and 1. The numerical system, humans have come to naturally use is called denary or base 10, as it uses 10 digits (0-9). Binary or base 2 works similarly to denary. Like in denary, 0 in binary represents 0. Similarly, 1 in binary represents 1 in denary. However, since 2 is not a digit in binary, to represent 2 in binary, 10 is used.

The same principle operates in denary, when all the combinations of digits possible have run out, an additional column is used. So, 3 in binary is 11, and after that, since there cannot be a 12, to express 4, 100 is used. To express 5, 101 is used, for 6, 110 is used, 7, 111. And then for 8, another column must be added, so we get 1000. You may have noticed it was when we got to 2, 4, 8, that it became necessary to add another column. The same continues, 1000 in binary represents 8, 10000 represents 16, 100000 represents 32. This because they are the powers of 2 ($2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, etc.). The same happens in denary, except rather than the powers of 2, it is the powers of 10 (10, 100, 1000).

In primary school, you are likely to have been taught to think of numbers in terms of the units, tens, hundreds, thousands, column, etc.

Example

To write 563 in denary columns, you would write:

100s (10^2)	**10s (10^1)**	**1s (10^0)**
5	6	3

Note that $(5 * 10^2) + (6 * 10^1) + (3 * 10^0) = 563$.

The same principles apply to binary.

Example

For example, to convert the binary number 101011 to denary, you would write it out as follows:

32s (2^5)	**16s (2^4)**	**8s (2^3)**	**4s (2^2)**	**2s (2^1)**	**1s (2^0)**
1	0	1	0	1	1

Then you would multiply each digit by its respective value, and add up the results. So here you would do $(2^5 * 1) + (2^4 * 0) + (2^3 * 1) + (2^2 * 0) + (2^1 * 1) + (2^0 * 1)$, which equals 43.

In fact, since the only digits in binary are 0 and 1, the process all you actually have to do it look for the digits which are 1s, and add their respective values together, so here all you would do is 32 + 8 + 2 + 1, which equals 43. You may have realised that a binary number will only be negative if the final digit is 1, as the final digit is the only number with an odd column value.

In the exam, they will be likely to talk about 8-bit binary, i.e. where binary numbers are allotted 8 bits of space. If you are given an 8-bit binary number that only uses less than 8 digits for example it will have leading 0s to make up for it.

Example

So, 1011, would be written as 00001000 for 8-bit binary.

This is what happens in computer memory as well. Numbers are allotted a set number of bits, and if they use less then, the bits which are not used, are set to 0. This also means in your exam: you will have to make sure that if you are giving an answer, you put the required amount of leading 0s. The same principle applies to 16-bit binary etc.

So, to convert from a denary number to 8-bit-binary you would draw out a table, like before with 8 columns from 2^7 to 2^0, and you would look at each column value from left to right, to see if it is smaller than or equal to the number in question. If it is not, you write 0. If it is you subtract the column value from the number in question, and write 1 in that column.

Example

Let us convert 42 into 6-bit binary. We would draw out a table from 2^5 to 2^0.

32s (2^5)	**16s (2^4)**	**8s (2^3)**	**4s (2^2)**	**2s (2^1)**	**1s (2^0)**

32 ≤ 42, so we do 42 – 32 = 10. And we write 1 in the 32s column.

32s (2^5)	**16s (2^4)**	**8s (2^3)**	**4s (2^2)**	**2s (2^1)**	**1s (2^0)**
1					

However, 16 > 10, so we write 0 in the 16s column. 8 ≤ 10, so we do 10 – 8 = 2. And we write 1 in the 8s column. 2

32s (2^5)	**16s (2^4)**	**8s (2^3)**	**4s (2^2)**	**2s (2^1)**	**1s (2^0)**
1	0	1			

Again 4 > 2, so we would write 0 in the 4s column. But 2 ≤ 2, so we would write 1 in the 2s column. Then we do 2 – 2 = 0. Then we would write 0 in the 1s column.

32s (2^5)	16s (2^4)	8s (2^3)	4s (2^2)	2s (2^1)	1s (2^0)
1	0	1	0	1	0

Hexadecimal is another numerical system, but with 16 digits: 0-9 make up the first 10 digits, and the letters A-F make up the next 6. So, in hexadecimal $A_{16} = 10_{10}$ and $B_{16} = 11_{10}$, and so on. Here is a table comparing binary, denary and hexadecimal.

Binary	**Denary**	**Hexadecimal**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Because binary is base 2 and hexadecimal is base 16, and $2^4 = 16$, a set of 4 binary digits can be represented by 1 hexadecimal digit. You can use this to convert from binary to hexadecimal and vice versa really easily. To convert from binary to hexadecimal, draw the above table, convert each block of 4 binary digits to its corresponding hexadecimal digit. To convert form hexadecimal to binary, convert each hexadecimal digit to its corresponding block of 4 binary digits.

Converting from denary to hexadecimal and hexadecimal to denary is easy as well, as you can use binary as an intermediary base. For example, to convert from denary to hexadecimal, go from denary → binary and binary → hexadecimal.

Sign and Magnitude

- Sign and magnitude is a way of representing integers, where the first bit is a sign bit. If it is 0, the integer is positive, otherwise the integer is negative.

Example

To convert -8 to 8-bit sign and magnitude binary, first write it normally, i.e. 00001000. Then, because it is negative we make the most significant bit 1. So we have 10001000.

- Problems with sign and magnitude:
    - There are 2 different representations of 0.
    - Addition and subtraction are more complicated.

Two's Complement

- In two's complement the leftmost bit represents a negated version of what it would normally. For instance, in the case of 8-bit binary, this would mean the leftmost bit would represent -128.

Example

Say you're given *1010*, because we're doing 4-bit binary, the most significant bit is *1*, and that is worth -2^3, so it is worth -8. Then the only other bit to concern ourselves with is the second from the right, which is worth 2^1, so we do $-2^3 + 2^1$, to get -6.

- To negate a two's complement number, invert the bits and add one.

Example

Say you're given *0100* and you need to negate it, first you invert the bits to get *1011*. Then you add one to the number to get *1100*.

- To add two's complement numbers together, you simply do normally binary addition. If the addition of two positive numbers yields a negative number, then an overflow has occurred. If the addition, of two negative numbers has yields a positive number, then an overflow has occurred. Otherwise an overflow hasn't occurred.

Example

For instance, to add -3 and 5 using 4-bit 2's complement, first write them out.

To obtain, -3 first negate 3, which is 0011. This gives you 1101.
5 is just 0101.

Then simply, add 1101 + 0101 = 10010. Notice that there is an overflow. Or so it appears. But none of our rules have been violated. So, we simply ignore the final bit, and we arrive at 0010. Notice this is 2. Funnily enough -3 + 5 is 2.

- To subtract two's complement numbers, simply negate the second and add them together.

- An important thing to bear in mind is that if you have a negative two's complement number, e.g. 1111, then it is not the same thing as 01111. It is, however, the same thing as 11111. The same applies to two's complement numbers. 0111, is not the same thing as 111, but it is the same thing as 00111.

- The same applies the other way. If you have 00111, this is clearly the same as 0111. If you have 1111, this is the same as 1.

Example

Let us say we want to add 11 and 0101, and both of them are already two's complement numbers. Then we must make 11 into 1111. Then we perform the addition, to get 10100. What just happened? An *apparent* overflow! But because none of our two rules were violated, we simply ignore the new bit, to get 0100.

Two's Complement Floating Point Numbers

- Uses of floating point numbers
    - To store decimal numbers
    - To store very large or very small numbers

- Floating point numbers are represented using a mantissa followed by an exponent.

- In floating point binary, the mantissa is a decimal where the decimal point is after the first bit. Since the first bit before the decimal point has the value 2^0, the first bit after the decimal point has the value 2^{-1}, and so on.

Example

Suppose our mantissa is 0101. Adding the decimal point gives 0.101. Adding the value of bits gives $2^{-1} + 2^{-3}$ which is $^5/_8$. So, $^5/_8$ is the value of the mantissa.

- The value of the number is the mantissa multiplied by 2 to the power of the exponent. Note that multiplying by 2, is the equivalent of shifting to the left by a certain number of places. So, it is just the mantissa shifted to the left by the number of times represented by the exponent.

Example

Suppose our mantissa is 0101. Adding the decimal point gives 0.101. Adding the value of bits gives $2^{-1} + 2^{-3}$ which is $^5/_8$. So, $^5/_8$ is the value of the mantissa.

Let us now say the value of the exponent is 0101. This is 5 in denary. So, the we shift the mantissa by 5 to the left, or alternatively the decimal point, or radix point in computer science, to the right by 5. This gives us 010100. The value of the number is 20.

- Note that the mantissa and exponent are stored in two's complement binary, so if the exponent is negative then the mantissa is shifted to the right.

-

Example

Suppose our mantissa is 0101. Adding the decimal point gives 0.101. Adding the value of bits gives $2^{-1} + 2^{-3}$ which is $^5/_8$. So, $^5/_8$ is the value of the mantissa.

Let us now say the value of the exponent is 1101. This is -3 in denary. So, the we shift the mantissa by 3 to the right, or alternatively the decimal point, or radix point in computer science, to the left by 3. This gives us 0.000101. The value of the number is $2^{-4} + 2^{-6}$ which is $^5/_{64}$.

- The value of the mantissa is a binary number, but where the decimal point is after the first bit. For instance, if the mantissa is 0101, then it actually represents 0.101. The values of the bits after the decimal point, follow the same pattern as the values of the bits before the decimal point. So, the value of the first bit to the left of the decimal point is 2^0, so the value of the first bit to the right is 2^{-1}, i.e. ½.

- The exponent is just a regular two's complement

Example

Let us use a 4-bit mantissa and a 4-bit exponent. Let us use two's complement for the mantissa and the exponent. Let the mantissa be 0100. The only bit that is 1 is the second from the left. This is worth 2^{-1}, i.e. a ½. Let us say the exponent is 0010. This means it is 2. We do ½ * 2^2, to obtain 2.

Example

Let us use a 4-bit mantissa and a 4-bit exponent. Let us use two's complement for the mantissa and the exponent. Let the mantissa be 0100. The only bit that is 1 is the second from the left. This is worth 2^{-1}, i.e. a ½. Let us say the exponent is 1010. This means it is -6. We do ½ * 2^{-6}, to obtain $^1/_{64}$.

- A normalised floating-point number should begin with two different digits. A positive number should begin with 01, and a negative number should begin with 10.

Example

Let us use a 4-bit mantissa and a 4-bit exponent. Let us use two's complement for the mantissa and the exponent. Let the mantissa be 0010. The only bit that is 1 is the third from the left. This is worth 2^{-2}, i.e. a ¼. Let us say the exponent is 1010. This means it is -6. We do ¼ * 2^{-6}, to obtain $^1/_{128}$. Notice, that this is not a normalised number because the first two digits of the mantissa are the same. To remedy this, we shift the digits of the mantissa to the left by 1. This is equivalent to multiplying by 2. So, we must decrement the exponent by 1, which is the equivalent of dividing by 2. So, the new mantissa is 0100. The new exponent is 1001. Notice this is still equivalent to $^1/_{128}$.

Example

Let us use a 4-bit mantissa and a 4-bit exponent. Let us use two's complement for the mantissa and the exponent. Let the mantissa be 1101. This is equivalent to -$^3/_8$. Let us say the exponent is 0101. This means it is 5. We do -$^3/_8$ * 2^5, to obtain -12. Notice, that this is not a normalised number because the first two digits of the mantissa are the same. To remedy this, we shift the digits of the mantissa to the left by 1. By we want to preserve the number sign. So, we don't modify the first bit. This is still equivalent to multiplying by 2. And, we must decrement the exponent by 1, which is the equivalent of dividing by 2. So, the new mantissa is 1010. The new exponent is 0100. Notice this is still equivalent to -12.

- Adding and subtracting floating points numbers is performed by making the exponent the same. Then performing normal addition and subtraction.

Example

Let us add 0.5 and 6. In normalised two's complement floating-point binary with 6 bits for the mantissa and exponent these are:
0.5 – 010000 000000
6 – 011000 000011

First you add the decimal point and shift the numbers, by the exponent. So, the first number stays the same as 0.1, and the second number becomes 0.11 * 2^3 = 0110. Then add the numbers 0.1 + 0110 = 0110.1.

Then shift the decimal point, back so the number is normalised, so you have 0.1101. So, the mantissa is 011010. Since you shifted it by 3, the exponent is 3, i.e. 000011. That makes 6.5.

- If you are doing this with a two's complement mantissa, then you need to know that to convert a positive two's complement number of say 4 bits, to one of 8 bits you simply make the leftmost 4 bits 0. To do this for a negative two's complement number, you may the leftmost 4 bits 1.

Example

Let us add -0.5 and 6. In normalised two's complement floating-point binary with 6 bits for the mantissa and exponent these are:
-0.5 – 100000 111111
6 – 011000 000011

First you add the decimal point and shift the numbers, by the exponent. So, the first number is shifted to right to be 1.1, and the second number becomes 0.11 * 2^3 = 0110. Then add the numbers 1.1 + 0110. This gives you 1111.1 + 110 = 10101.1. An overflow appeared to have occurred, so you ignore the last bit, so you have 0101.1. Then shift the decimal point, back so the number is normalised, so you have 0.1011. So, the mantissa is 010110. Since you shifted it by 3, the exponent is 3, i.e. 000011. That makes 5.5.

Bitwise Manipulation

- A bitwise AND, OR or XOR is where the appropriate logic gates are applied to each corresponding bit on a pair of numbers.
- Logical shifts shift the number left or right and make the vacated bit 0.
- Arithmetic shifts shift the number left or right but keep the sign bit the same. When shifting right, this is done by moving a 1 into the vacated bit if the sign bit was 1. If shifting left, then the shift ignores the sign bit.
- Circular shifts shift the bits in a circle.

Character Set

- **Character Set** - The symbols a computer system can recognise. Normally corresponds to symbols on a keyboard. Each symbol has a numeric value. Examples include ASCII and Unicode.

1.4.2 Data Structures

Array – A static data structure consisting of a list of values, of the same data type, stored contiguously in a memory location accessible from a single identifier with an index.

Linked List – A dynamic data structure consisting of nodes which consist of data and a pointer to the next node.

Advantages of Arrays	Advantages of Linked Lists
Data is stored contiguously, so access times are constant i.e. arrays allow random access	Linked lists are dynamic data structures, so as many items as necessary can be added
Less overhead as no pointers stored	Easier to move items around and delete items, as only pointers, need to be changed
	Merging two linked lists is easier, as the pointer from the final node of the first list just needs to be set to the first node of the second.

Record – A collection of named fields, which may be of different data types, that are stored in a memory location together under a single identifier. To access an individual field, the record identifier and the field name are used in conjunction.

Tuple – A tuple is an immutable collection of values, which may of different data types, stored under a single identifier. To access an individual field, an index is used.

Queue - A first-in-first-out (FIFO) data structure, where the first item to be "enqueued" (added to the queue) is the first item to be "dequeued" (removed from the queue).

Stack - A last-in-first-out (LIFO) data structure, where the last item to be "pushed" (added to the stack) is the first item to be "popped" (removed from the stack).

A stack has a pointer to the last item, a queue has a pointer to the first and the last item.

Queues and stacks can be of a fixed capacity, i.e. static, or of a variable-length, i.e. dynamic.

Graph – A graph is a data structure consisting of a set of vertices (or nodes) connected by edges (or arcs).

Undirected Graph – A graph where all edges are bidirectional.
Directed Graph – A graph where all edges are one-way.

Tree - A data structure made from nodes which can have a number of children, which are also nodes. A node can only have on parent, except for the root node which has no parent.

Binary Search Tree – A tree where each node can have a maximum of two children, where nodes are stored in an order, by having the nodes in the left subtree be less than a given node, and the nodes in the right subtree greater than.

Pre-order traversal – The current node is visited before its left subtree and its right subtree are recursively traversed.

In-order traversal – The left subtree of a node is recursively traversed, before the current node is visited, and then the right subtree is recursively traversed.

Post-order traversal – The left subtree and right subtree of a node is recursively traversed, before the current node is visited.

Hash Table – A data structure where items of data are accessed by keys. The data is stored in buckets. To decide which bucket should hold a piece of data identified by a given key, a hashing algorithm is applied to the key to get the index of a particular bucket. A collision occurs if two keys are to be stored in the same bucket. One way of resolving this is storing linked lists in each of the buckets.

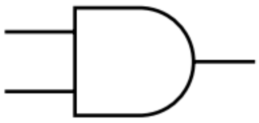
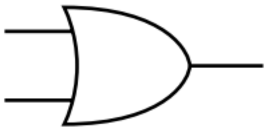
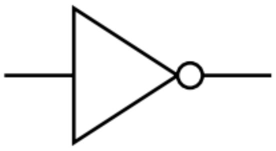
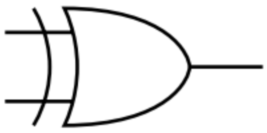
Applications of Queues

- Storing jobs to be processed by the CPU
- Storing requests to a website or a database to be processed

Applications of Stacks

- Calling functions (arguments stored in a stack)
- Many programming languages use stacks to parse code when compiling/interpreting

1.4.3 Boolean Algebra

Operation	**Symbol**	**Gate**
AND	$A \wedge B$	
OR	$A \vee B$	
NOT	$\neg A$	
XOR	$A \underline{\vee} B$	

You also need to be aware that: A ≡ B means that the Boolean expression A is equivalent to the Boolean expression to B.

Simplifying Boolean Expressions

Basic Laws	A ⋁ 1 ≡ 1
	A ⋁ 0 ≡ A
	A ⋀ 1 ≡ A
	A ⋀ 0 ≡ 0
Double Negation Law	¬¬ A ≡ A
Idempotent Laws	A ⋁ A ≡ A
	A ⋀ A ≡ A
Complement Laws	A ⋁ ¬A ≡ 1
	A ⋀ ¬A ≡ 0
Distributive Laws	A ⋀ (B ⋁ C) ≡ (A ⋀ B) ⋁ (A ⋀ C)
	A ⋁ (B ⋀ C) ≡ (A ⋁ B) ⋀ (A ⋁ C)
De Morgan's Laws	¬(A ⋁ B) ≡ (¬A ⋀ ¬B)
	¬(A ⋀ B) ≡ (¬A ⋁ ¬B)

AND and OR are also commutative and associative operations.

Karnaugh Maps

Karnaugh Maps are an alternative way of simplifying Boolean expressions. This is done by creating a table, where the rows are one variable and the columns are another variable. If doing Karnaugh maps on three-variable, four-variable problems, or even bigger problems, then one writes multiple variables as the columns/rows. For instance, if A, B and C are variables, then A might be the rows, and BC might be the columns. Each row and column represents a certain assignment of variables of a subset of the variables. Each cell represents a complete assignment of variables, and stores what the expression would evaluate to in that case. It is essential that the variable assignments that the rows or columns represent only differ by 1 bit. Then one selects the biggest groups of 1, 2, 4, 8, etc. ones that form a rectangle. The rectangle may wrap around the edges. Then one finds the expression that make these groups 1 and combines then using OR.

Adders and D-Type Flip Flops

- **Half Adder** – A half adder takes the input of two bits and gives a two-bit output, consisting of a sum and a carry.
- **Full Adder** – A full adder is made from two half adders to add three bits, where one of the bits is a carry. It produces a two-bit output.
- To construct an adder capable of adding two *n*-bit numbers, simply combine *n* full adders.
- **D-Type Flip Flop** – A flip flop is a circuit that can store one bit. It has two inputs, a control input and a clock signal. A D-type flip flop is a positive edge-triggered flip-flop; this means that it takes the values whenever the clock signal changes from 0 to 1, and this is the only time it does this.

1.5 Legal, moral and ethical issues

1.5.1 Computing Related Legislation

Data Protection Act 1999:

- Data must be processed fairly and lawfully
- Data must be adequate, relevant and not excessive
- Data must be accurate and kept up to date
- Data must not be kept for longer than necessary
- Data must only be used for the purpose for which it was collected
- Data must be kept secure
- Data must be handled in accordance with people's rights
- Data must not be transferred to countries outside the EU without adequate protection
- All data users must register with the Information Commissioner

Computer Misuse Act 1990 created four new offences:

- Unauthorised access to computer systems
- Unauthorised access with intent to commit or aid crime
- Unauthorised modification of computer systems
- Making, supplying or obtaining anything that can be used in the above offences.

Copyright, Designs and Patents Act 1988 makes it illegal to:

- Copy works of intellectual property (software, music) electronically to redistribute
- Renting works of intellectual property without the permission of the copyright holder
- Putting intellectual property on a shared network without the permission of the copyright holder

Regulation of Investigatory Powers Act 2000 allows:

- Certain public bodies to monitor people's Internet activities
- Certain public bodies to demand people hand over encryption keys
- Certain public bodies to demand ISPs hand over customer information and to install surveillance equipment

Exam Technique

You are likely to be asked questions on laws in 9 mark questions. In 9 mark questions, there are 4 marks for knowledge, 2 marks for application and 3 for evaluation. So, if asked about people stealing a film from a CGI company's corporate network, you should use the following structure:

- **Knowledge**: Unauthorised access to computer systems is illegal under the Computer Misuse Act.
- **Knowledge**: It is an additional offence to gain unauthorised access with the intention of committing crime.
- **Application**: If the person who got the films, hacked the company's corporate network to do so, then they will be in breach of the first offence.

- **Evaluation**: This is relevant only if the person who obtained them had to gain unauthorised access. It is not the same if somebody working for the firm leaked it.

1.5.2 Moral and ethical issues

Computers in the workforce has led to many low-skill manufacturing jobs and low-skill service jobs being automated. Firms such as Google and Amazon are able to make huge amounts of money, while employing very few people.

Automated decision making is where computer make important, potentially life or death decisions. For instance, a self-driving car may make a decision that kills somebody, or makes a decision that prioritises the life of a group of people over one person. In either case, there is a question of who is responsible for the deaths. The programmer? The manufacturer?

Artificial intelligence systems can be used in different scenarios. For instance, they can be used to detect credit card fraud, or predict customer shopping to order goods in advance or attempt to diagnose patients based on their symptoms.

The environmental considerations of computer systems include that technology has led to a massive increase in electronic waste being discarded in landfills. This is especially true since phones have a very short life cycle.

Examples of using computers to monitor behaviour include monitoring employees' internet activity, monitoring the internet activity of those suspected of crime and tagging criminals.

Different governments attempt to censor information. It could be that this information is against the political line of the government in countries such as China and North Korea. It could be the case that the information is pornographic or pirated content.

Piracy sites such as Pirate Bay allow people to access films, music, etc. for free. However, this also means producers of content such as films and music do not receive payment for their work. If done on a wide enough scale, this can make producing this content unprofitable. The internet also allows for offensive communications e.g. 'trolling' and cyber-bullying. This may be in the form of death threats of misogynistic slurs. It is difficult for platforms such as Facebook to decide what should be removed.

Ways to make websites accessible:

- Making the website friendly to screen readers
- Adding alternative text to images
- Taking into account colour blindness when designing colour schemes

2.1 Computational Thinking

2.1.1 Thinking Abstractly

Why is abstraction necessary?

- Remove things that are unnecessary

- They may require additional programming effort
- They may detract from the purpose of the program
- Save computational resources e.g. memory

2.1.2 Thinking Ahead

Caching

Caching is where frequently performed computations or data that has to be requested is stored, to reduce time in the future. For instance, your web browser is likely to cache frequently browsed web pages. The advantage is that it reduces time for a program to run. The disadvantage is that it increases the amount of storage used. You can think of it as a space-time tradeoff.

Reusable Components

This is where a program is made from a number of smaller independent components, that can be reused. This is done as reusable components might be useful in the future. This will save programmer time.

2.2 Problem Solving

2.2.1 Programming techniques

Sequence – Where one instruction is executed, followed by another.

Selection – Where a piece of code is only executed if a certain condition is met.

Iteration – Where a piece of code is executed repeatedly until a certain condition is met, or for a set number of times.

Variable – An identifier associated with a memory location used to store data and whose contents can be changed at runtime.

Global Variable – A variable that can be accessed from anywhere in a program.

Local Variable – A variable which can only be accessed from the construct it was defined in.

Constant – An identifier associated with a value that cannot be changed during a program's execution.

Array – A static data structure consisting of a list of values stored contiguously in a memory location accessible from a single identifier with an index.

Procedure – A named subsection of a program that performs a specific task but does not necessarily return a function.

Function – A named subsection of a program that performs a specific task and returns value.

Pass by value – This is where a copy of the data passed to a function is passed as a parameter.

Pass by reference – This is where a memory address to a piece of data to a function is passed as a parameter.

IDE – A single program used to develop programs made from a number of components e.g. an editor, a compiler, a debugger, etc.

2.3 Algorithms

2.3.1 Algorithms

Big O Notation

$O(1)$	Constant Time
$O(\lg n)$	Logarithmic Time
$O(n)$	Linear time
$O(n^k)$	Polynomial time, where k is a constant.
$O(k^n)$ usually $O(2^n)$	Exponential time, where k is a constant.

When we say an algorithm take $O(f(n))$, what we are strictly saying is that an algorithm's maximum running time (that it could possible take) as n, which is the input size e.g. the length of an array, varies is proportional to $f(n)$. Big O Notation provides an upper bound.

Intractable Problem

A problem is intractable is if it can be solved in theory, but in practice it cannot be solved. Normally, computer scientists saying that a problem is intractable if it takes an exponential time, as a function of its input. This may seem like a very rough definition, but it makes a lot of sense. Imagine a problem takes $O(2^n)$, and you wanted to solve it. Suppose with your current computer you could solve for when $n = 5$, for you to be able to solve $n = 6$ in the same time, your computer will have to twice as fast. To solve $n = 7$, it will have to be four times as fast. Even if computers doubled in speed every year, in a 100 years time you would only be able to solve $n = 105$.

Sorting Algorithms

Bubble Sort

```
procedure bubbleSort(input)
  for i = 0 to (input.length – 2)
     for j = 0 to (input.length – 2 – i)
       if input[j] > input[j+1] then
          // Swap input[j] and input[j + 1]
          a = input[j + 1]
          input[j + 1] = input[j]
```

```
        input[j] = a
      endif
    next j
  next i
endprocedure
```

Insertion Sort

```
procedure insertionSort(input)
  for i = 1 to (input.length – 1)
    key = input[i]
    while i > 0 && input[i – 1] > key
      input[i] = input[i – 1]
      i = i - 1
    endwhile
    input[i] = key
  next i
endprocedure
```

The merge sort algorithm below can actually be simplified with clever Boolean algebra but it is simpler to understand like this.

Merge Sort

```
// This methods merges the two sublists input[start..midpoint]
// and input[midpoint + 1..end]
procedure merge(input, start, midpoint, end)
  array workingMemory[end – start + 1]
  positionA = start // This is the position into the first list
  positionB = midpoint + 1 // This is the position into the second list
  for i = 0 to (end – start + 1):
    if positionA > midpoint then
      workingMemory[i] = input[positionB]
      positionB = positionB + 1
    else if positionB > end then
      workingMemory[i] = input[positionA]
      positionA = positionA + 1
    else
      if input[positionA] < input[positionB] then
        workingMemory[i] = input[positionA]
        positionA = positionA + 1
      else
        workingMemory[i] = input[positionB]
        positionB = positionB + 1
      endif
    endif
  next i
  for i = 0 to (end – start + 1):
    input[i] = workingMemory[i]
  next i
endprocedure

procedure mergesort(input, start, end)
  if start < end then
     // Integer Division
    midpoint = (start + end) / 2
    mergeSort(input, start, midpoint)
    mergeSort(input, midpoint + 1, end)
    merge(input, start, midpoint, end)
  endif
endprocedure
```

Quick Sort

```
// This algorithms splits the list into two sublists.
// One which is smaller than the pivot
// Another that is larger than the pivot
// The algorithm returns the position of the pivot
function partition(input, start, end)
  k = start – 1
  // The pivot is always taken to be the item at the end
  pivot = input[end]
  // The algorithm works by splitting the input, into three sublists
  // First the section of the list that is smaller than the pivot
  // This is input[start..k]
  // Since k initially equals start – 1, this section is empty
  // Then you have input[k+1..i]
  // This is the section which is greater than the pivot
  // Then you have input[i..end] which is the sublist yet to be processed
  // Eventually input[i..end] will be empty, so there will only be two sublists.
  for i = start to end – 1
   if input[i] <= pivot then
     k = k + 1
     // Swap input[k] and input[i]
     a = input[k]
     input[k] = input[i]
     input[i] = a
    endif
  next i
  // The last item to be smaller than or equal to the pivot is the pivot
  // So, k points to the location of the pivot
  return k
endfunction

procedure quicksort(input, start, end)
  if start < end then
    pivot = partition(input, start, end)
    quicksort(input, start, pivot - 1)
    quicksort(input, pivot + 1, end)
  endif
endprocedure
```

Binary Search

```
function binarySearch(input, search)
  lower = 0
  upper = input.length
  middle = (lower + upper) / 2
  while lower <= upper
    if input[middle] = search then
      return middle
    else if input[middle] < search then
      upper = middle + 1
    else
      lower = middle – 1
    endif
  endwhile
  return -1
endfunction
```

Linear Search

```
function linearSearch(input, search)
  for i = 0 to input.length
    if input[i] == search then
      return i
    endif
  next i
  return -1
endfunction
```

This is just one way of doing Dijkstra’s Algorithm. Note I find the shortest distance to every node. To make it more efficient, you can take an additional parameter which is the goal node and stop once you’ve found the shortest distance to the goal node.

Dijkstra’s Algorithm

```
// This function returns a hash map between nodes in the graph and the
// shortest distance from the startNode
// We are assuming there is a function dist(a, b) which gives you the distance between
// two connected nodes.
function dijkstra(startNode)
  map = createHashMap()
  notProcessed  = createLinkedList()
  processed = createLinkedList()
  // map is the name of the hash map we will be compiling during this algorithm
  // and we will return.
// notProcessed is the list of nodes that we have not found the shortest distance to
// but that we have encountered
// processed is the list of nodes that we have found the shortest distance to

// Initially, map will only store the shortest distance we have seen so far
// Once we are sure it is the shortest distance we move a node from notProcessed
// To processed

// Initially we have only encountered the start node.
notProcessed.add(startNode)

// Obviously you don't have to use linked lists
// You can use whatever data structure floats your boat
// But it has to be a dynamic data structure
// We do not stop until there is nothing left to processed
while !notProcessed.isEmpty()
  // First we find the node in notProcessed with the shortest distance so far
  minIndex = 0
  minDistance = -1
  for i = 0 to notProcessed.getLength()
    if map.get(notProcessed.get(i)) < minDistance
      minIndex = i
    endif
  endfor

  // That node is removed from notProcessed
  node = notProcessed.get(i)
  notProcessed.remove(node)
 // And added to processed
 processed.add(node)

 // Then we add its neighbours to notProcessed
 // Unless they're already in notProcessed or processed
 neighborus = node.getNeighbours()

for i = 0 to neighbours.getLength()
  x = neighbours.get(i)
   if !(processed.contains(x)) then
      if notProcessed.contains(x) then
        // If we are here it means that we have found another path to a node
        // That we have already seen
        // this might be the shorter path than already has been found
        if map.get(x) > map.get(node) + dist(node, x) then
           // Update the distance
           map.put(x, map.get(node) + dist(node, x))
        endif
      else
        // This means it is not in notProcessed
        // So we add it
        notProcessed.add(x)
        map.set(x, map.get(node) + dist(node, x))
      endif
    endif
  endfor

  return map
endfunction
```

Dijkstra's algorithm will always find the shortest distance from a node to a goal node. However, in doing so it checks every path. The A* pathfinding algorithm finds a path from one node to a goal node, but guesses i.e. uses a heuristic to choose where to search. For instance, the heuristic might be the shortest distance between the current node and the goal node (e.g. using Pythagoras' theorem). In this algorithm, I assume there is a heuristic algorithm *guess(current, goal)* which estimates the distances between *current* and *goal*. However, the A* algorithm will always find a path if one exists. My implementation of the algorithm does not reconstruct the path, but that can be added with a few modifications.

A* Pathfinding Algorithm

```
// This functions returns 0 if it has found a path
// Otherwise it returns 0
function A*(startNode, goalNode)
  map = createHashMap()
  notProcessed  = createLinkedList()
  processed = createLinkedList()
  // map is the name of the hash map we will be compiling during this algorithm
  // and we will return.

  // notProcessed is the list of nodes that we have not processed
  // but that we have encountered
  // processed is the list of nodes that we have processed

  // map will store the shortest distance to a node that we have seen so far

  // Initially we have only encountered the start node.
  notProcessed.add(startNode)

while !notProcessed.isEmpty()
   // Here is where the guessing begins
   // We choose a node based on the shortest distance we have seen so far to it
   // And an estimation of how long it will take to get to the goal node
   minIndex = 0
   minDistance = -1
   for i = 0 to notProcessed.getLength()
```

```
    if map.get(notProcessed.get(i)) + guess(notProcessed.get(i), goalNode) < minDistance
      minIndex = i
    endif
  endfor

  // That node is removed from notProcessed
  node = notProcessed.get(i)
  notProcessed.remove(node)
 // And added to processed
 processed.add(node)

 // Then we add its neighbours to notProcessed
 // Unless they're already in notProcessed or processed
 neighborus = node.getNeighbours()

for i = 0 to neighbours.getLength()
  x = neighbours.get(i)
  // First we check if the neighbour is the goal node
  if x == goalNode then
    // We did it!
    return 0
  endif
   if !(processed.contains(x)) then
     if notProcessed.contains(x) then
       // If we are here it means that we have found another path to a node
       // That we have already seen
       // this might be the shorter path than already has been found
       if map.get(x) > map.get(node) + dist(node, x) then
         // Update the distance
         map.put(x, map.get(node) + dist(node, x))
       endif
     else
       // This means it is not in notProcessed
       // So we add it
       notProcessed.add(x)
       map.set(x, map.get(node) + dist(node, x))
     endif
    endif
  endfor

 // If we get to here our algorithm has failed to find a path
 // Which means a path doesn't exist
 return -1
endfunction
```