4 Hardware and Software

4.19 Hardware and Software

- **Hardware** is the electrical or electro-mechanical parts of a computer and its input/output and storage devices.
- **Software** comprises all the programs that are written to make computers function.

Software can be classified into **system software** and **application software**.

System Software

System software is the software needed to run the computer's hardware and application programs.

- An **operating system** is a set of programs that lies between application software and the computer hardware.

  Functions include hardware **resource management** and provision of a **user interface**.

- **Utility software optimises the performance** of the computer or perform tasks such as backing up files, compressing and encrypting data before transmission, firewall, etc.
  - **Disk defragmenter** reorganises a hard disk so that files which have been split up into blocks and stored all over the disk is **recombined in a single series of sequential blocks**. This makes reading a file quicker.
  - **Virus checker** checks your hard drive and internet downloads for viruses and remove them.
- **Libraries** are ready-compiled programs which can be run when needed.

  Libraries in Windows have a .dll extension.
- **Programming language translators** translate program code written by a programmer into machine code which can be run by the computer.

Application Software

Application software **performs specific user-oriented tasks**.

- **General-purpose software** can be used for many different purposes.

  E.g. word-processor, spreadsheet.
- **Special-purpose software** performs a single specific task or set of tasks.

  E.g. hotel booking systems.

- Software bought **off-the-shelf** are ready to use.
- **Bespoke software** package are specifically written by a team of programmers for a particular organisation, to satisfy their particular requirements.

Off-the-shelf software

- Cheaper to buy, as the cost is shared among all other people buying the package.
- It is **ready to be installed** immediately.
- Likely to be **well-documented, well-tested and bug-free**.

It may also contain a lot of features never used.

4.20 Role of an Operating System

An operating system is a set of program that **manages the operations of the computer** for the user. It **acts as a bridge** between the user and the computer's hardware, since a user cannot communicate with hardware directly.

- The operating system is held in **permanent storage** such as the hard disk.
- The **loader** is held in **ROM**.

When a computer is switched on, the loader in ROM **sends instructions to load the operating system** by copying it from storage into RAM.

The **Application Programming Interface** is provided to disguise the complexities of managing and communicating with its hardware from the user. So the user can complete their tasks without knowing the actual operations taking place behind the scenes to support their actions.

The operating system has the following functions

- **Memory management**
  - A PC allows a user to be working on several tasks at the same time.
  - Each program must be **allocated a specific area of memory** whilst the computer is running.
  - If the user wishes to switch from one application to another in a separate window, each application must be **stored in memory simultaneously**.
  - The allocation and management of space is controlled by the operating system.

  Virtual memory uses the hard disk as an extension of memory, it is used when the computer's RAM is not large enough to store all these programs simultaneously.

1. If an opened program is not in use at a particular time, the operating system may copy it and data to hard disk to **free up RAM** for another software.
2. When switched back to that program, the operating system will **reload it into memory**.

- **Processor scheduling** - with multiple programs running simultaneously, the operating system is responsible for **allocating processor time to each one** as they compete for the CPU.

  While one application is using the CPU for processing, the OS can **queue up the next process required** by another application to make the most efficient use of the processor.

  - A computer with a single-core processor can only process **one application** at a time.
  - By carrying out **small parts of multiple larger tasks** in turn, the processor can give the appearance of **carrying out several tasks simultaneously**.
  - This is called **multi-tasking**.

  The **scheduler** is the operating system module responsible for making sure that processor time is used as efficiently as possible. Its objective are to

  - Maximise throughput.
  - Be **fair to all users** on a multi-user system.
  - Provide **acceptable response time** to all users.
  - Ensure hardware resources are **kept as busy as possible**.

- **Backing store management** - the operating system keeps a directory of where the files are stored so that they can be quickly accessed.

  It also needs to know which areas of storage are free so that new files and applications can be saved.

  A file management system enables a user to

  - Move files and folders.
  - Delete files.
  - Protect others from **unauthorised access**.

- **Peripheral management** - the operating system communicates and ensures that peripherals are allocated to processes without causing conflicts.

- **Interrupt handling** - the OS is responsible for detecting the interrupt signal and displaying an appropriate error message for the user if appropriate.

  It is because a processor can be interrupted that **multi-tasking** can take place.

4.21 Programming Language Classification

Machine Code

Machine code consists of **binary digits** that the computer could understand.

A typical machine code instruction holds

- An **operation code** in the first few bits.
- An **operand** in the later bits.

  The operand is the **data** to be operated on, or the **address** where the data is held.

In comparison, high level languages **demonstrate abstraction**, because it doesn't show how the computer actually carry out the high level instruction, so the programmer can concentrate on the algorithm.

Machine code is called a **low-level programming language**, because the code reflects how the computer actually carries out the instruction - it is dependent on the actual architecture of the computer.

Assembly Language

- Each opcode is replaced by a **mnemonic**, which gives good clue of the what the operator is doing.
- The operand was replaced by a decimal number.

High-level Programming Languages

FORTRAN was the first high-level programming language.

These languages are called **imperative high-level languages**, because each instruction is a command to perform some step in the program, which consists of the **step-by-step instructions** needed to complete the task.

- High-level languages enable programmer to think and code in terms of algorithms, without worrying how each tiny step will be executed in machine code, and where each item of data is stored.
- Each instruction in a high level language is **translated into several low-level language instructions**.

The advantages of a high-level language are

- They are relatively **easy to learn**.
- It is must easier and **faster to write a program** in a high-level language.
- Programs written in high-level languages are easier to **understand, debug and maintain**.
- Programs written in a high-level language is **not dependent on the architecture** of a particular machine - they are **machine independent**.
- There are many **built-in library functions** available in most high-level languages.

Assembly language is still used when

- The program needs to **execute as fast as possible**.
- Occupy as **little space as possible**.
- Manipulate individual bits and bytes.

Such as in embedded system and device drivers.

4.22 Programming Language Translators

Each type of processor will have a **different instruction set** and **different assembly language**.

- Each instruction in assembly language is **equivalent to one machine code instruction**.
- And the machine code instructions that a particular computer can execute are **completely dependent on hardware**.

The assembler program takes each assembly language instruction and **converts it to the corresponding machine code instruction**.

- The input to the assembler is called the **source code**.
- The output is the **object code**.

Compiler

A compiler is a program that translates a high-level language into machine code.

1. Code written by the programmer - the **source code** is input as data to the compiler.
2. Which **scans through it** multiple times, each performing different **checks** and building up tables of information needed to produce the final object code.
3. The object code can then be saved and run whenever needed **without the presence of the compiler**.

Different hardware platforms will require different compilers, since the resulting **object code is hardware-specific**.

Advantages of compiler

- Object code can be saved on disk and run whenever required **without the need to recompile**.
- Object code **executes faster** than interpreted code.
- Object code produced by a compiler can be distributed or executed **without having the compiler** present.
- Object code is more secure, as it cannot be read without a great deal of reverse engineering.

Disadvantages of compiler

- If an error is discovered in the program, the whole program has to be recompiled.

A compiler is appropriate when

- A program is to be run **regularly and frequently**, with only **occasional change**.
- The object code produced by the compiler is to be **distributed or sold** to users, since the source code is not present it cannot be amended.

Interpreter

1. The interpreter software itself **contains subroutines** to carry out each high-level instruction.
2. When instructed to run a program, it **looks at each line** of the source code, **analyses it** and if contains no syntax errors, **calls the appropriate subroutine** within its own program code to execute the command.

Advantages of interpreter

- Useful for **program development** as no need for lengthy recompilation each time an error is discovered.
- Easier to **partially test** and debug programs.

Disadvantages of interpreter

- Runs slower - each statement has to be **translated to machine code** each time it is encountered.

A interpreter is used

- During program development, and compiled for distribution when the program is complete and correct.

- In a **Student environment** when students are learning to code, so they can **test parts** of a program before coding it all.

Bytecode

Bytecode is an **intermediate representation** which combines compiling and interpreting.

- The bytecode may be **compiled once** and for all (as in Java).
- Or **compiled each time** a change in the source code is detected (as in Python).
- The resulting bytecode is then executed by a **bytecode interpreter**.

Bytecode achieves **platform independence**

- Any computer that can run Java programs has a **Java Virtual Machine**, which masks the inherent differences between different computer architectures and operating systems.
- The JVM understands bytecode and **converts it into the machine code** for that particular computer.

Java bytecode acts as an **extra security layer** between the computer and the program - the Java bytecode interpreter is executed instead of the untrusted program, which guards against any malicious programs.

4.23 Logic Gates

Notations are as follows

- `NOT A`: $\bar{A}$
- `A AND B`: $A \cdot B$
- `A OR B`: $A + B$
- `A XOR B`: $A \oplus B$ or $(\bar{A} \cdot B) + (A + \bar{B})$
- `A NAND B`: $\overline{A \cdot B}$
- `A NOR B`: $\overline{A + B}$

4.24 Boolean Algebra

- **De Morgan's first law**: $\overline{A + B} = \overline{A} \cdot \overline{B}$
- **De Morgan's second law**: $\overline{A \cdot B} = \overline{A} + \overline{B}$

Associative rules

$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$$
$$X + (Y + Z) = (X + Y) + Z$$

Distributive rules

$$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$$
$$(X + Y) \cdot (W + Z) = X \cdot W + X \cdot Z + Y \cdot W + Y \cdot Z$$

Absorption rules

$$X + (X \cdot Y) = X$$
$$X \cdot (X + Y) = X$$