# Topic 04: Software

IGCSE Computer Science 0478 | SenpaiCorner | Mr. Wei

**Learning Objectives**

By the end of this topic you should be able to:

- Distinguish between system software and application software; give examples of each.
- Describe the key functions of an operating system: managing files, memory, peripherals, multitasking, security, user accounts, interrupts, providing a user interface and an application platform.
- Explain the software hierarchy: Hardware → Firmware → OS → Applications.
- Define an interrupt; distinguish hardware interrupts from software interrupts; trace the interrupt handling cycle (priority check → pause → ISR → resume).
- Compare high-level and low-level languages (assembly language and machine code) across syntax, portability, speed, and hardware control.
- Describe how an assembler, compiler, and interpreter each translate code; compare compiler vs. interpreter for development and deployment.
- Identify and describe the common functions of an IDE: code editor, run-time environment, error diagnostics, auto-completion, auto-correction, prettyprint.

## 4.1 Types of Software and Interrupts

### System Software vs Application Software

**System software** refers to the programs that control or maintain the operation of the computer. It provides the environment and services required for other software and hardware to function, and it interacts directly with hardware to manage system resources.

**Application software** refers to programs designed to perform specific tasks for the user. It relies on system software to function and is written to fulfil a particular user purpose (e.g. document processing, playing media).

| System Software | Application Software |
|---|---|
| Provides services the *computer* requires | Provides services the *user* requires |
| Runs in the background; essential for operation | Run by the user to perform specific tasks |
| Manages hardware resources directly | Uses the system to do useful work for the user |
| **Examples:** Operating System, Utility Software (antivirus, disk defragmenter, backup tools), Device Drivers, Firmware | **Examples:** Word processor (Microsoft Word), Spreadsheet (Excel), Web browser (Chrome), Games, Graphic design software |

**System vs Application — The Exam Phrasing**

Cambridge mark schemes use precise language: *system software provides services that the **computer** requires*, whereas *application software provides services that the **user** requires*. Vague answers such as "system software helps the computer run" are not awarded marks. For a "describe the difference" question (4 marks), write one mark for system definition, one mark for application definition, one mark for a system software example, one mark for an application software example.

## The Operating System

An **operating system** is a core part of system software that manages both the hardware and software resources of a computer system, and provides a platform on which applications can run.

| OS Function | Description |
|---|---|
| Managing files | Handles file storage, naming, organisation, access, and retrieval. Allows users and programs to create, copy, open, close, move, rename, delete, and sort files. |
| Handling interrupts | Responds to interrupt signals from hardware or software; assigns a priority; runs the Interrupt Service Routine (ISR); then resumes the paused task. |
| Providing a user interface | Offers a GUI (Graphical User Interface) or CLI (Command-Line Interface) for users to interact with the computer. |
| Managing peripherals and drivers | Communicates with hardware devices (printers, keyboards, hard disks) using device drivers. |
| Managing memory | Allocates and deallocates memory to processes; keeps track of each memory location; ensures no two processes access the same location; manages virtual memory and transfers pages between RAM and virtual memory. |
| Managing multitasking | Allows multiple applications or processes to run concurrently by managing CPU scheduling. |

| OS Function | Description |
|---|---|
| Platform for application software | Acts as a bridge between hardware and software, enabling applications to run on the system. |
| Providing system security | Provides user authentication, access controls, encryption, and system protection mechanisms. |
| Managing user accounts | Supports multi-user environments with separate profiles, access rights, and privileges. |

**OS Functions — Exam-Ready Recall List**

Nine functions: **Files — Interrupts — User interface — Peripherals — Memory — Multitasking — Platform — Security — User accounts**. In a "give two *other* functions" question (2 marks total), name the function (1 mark) **and** describe its purpose (1 mark) for each. "Loading the bootstrap" is **not** an OS function — that is handled by firmware (confirmed by 2024 paper mark scheme).

**Managing Memory — 3-Mark Mark Scheme Response**

Describe the role of the operating system in managing memory. [3 marks]

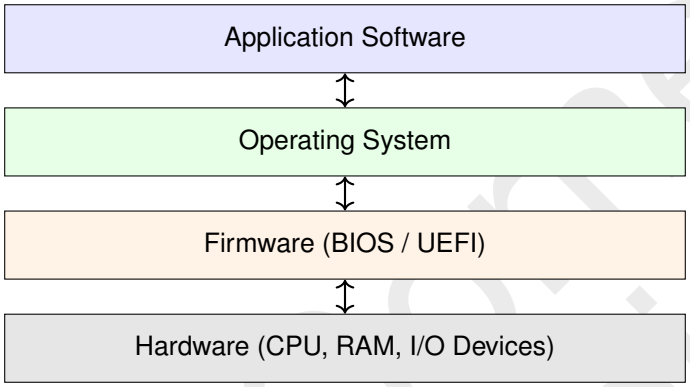
**Model response — any three from:**

- To make sure memory is used efficiently.
- Allocates/deallocates memory to processes.
- Checks that the processes being actioned have enough memory available.
- Moves data to and from RAM (between memory and storage).
- Makes sure that no two processes try to access the same memory location.
- Creates virtual memory; transmits pages between RAM and virtual memory.

## Software Hierarchy — Hardware, Firmware, OS and Applications

Computers function through several software layers built on top of the physical hardware.

- **Hardware** — the physical layer (CPU, RAM, hard drive, I/O devices).
- **Firmware** — software permanently programmed into ROM; the first code that runs when the system is powered on (e.g. BIOS or UEFI).
- **Operating System** — loaded by firmware into RAM during the boot process.
- **Application Software** — runs on top of the OS; what users directly interact with.




**Firmware is NOT the Operating System**

A common exam error is conflating firmware with the OS. **Firmware** is stored in ROM and runs *before* the OS loads — its job is to initialise hardware and locate the OS (bootloader). The **OS** is loaded into RAM from secondary storage by the firmware. Loading the bootstrap is a firmware task, *not* an OS function.

## Interrupts and Interrupt Handling

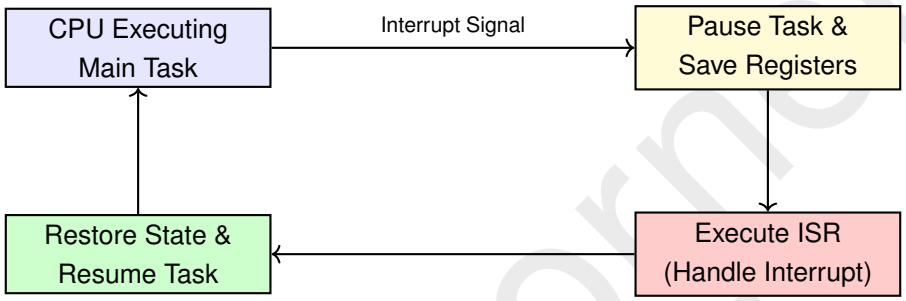
An **interrupt** is a signal sent to the CPU that temporarily halts the current process to alert the processor to an urgent task.

### Types of Interrupt

| Type | Cause | Examples |
|---|---|---|
| Hardware interrupt | Triggered by a physical hardware event | Key press on keyboard; mouse button click; moving the mouse; plugging in a device; printer out of paper; paper jam in printer |
| Software interrupt | Triggered by programs or errors during execution | Division by zero; illegal memory access; two processes trying to use the same memory location; null value |

### Interrupt Handling Process

1. The CPU receives the interrupt signal.
2. The OS **assigns a priority** to the interrupt and places it in a queue.
3. If the interrupt's priority is higher than the current task, the CPU **pauses** the current instruction and saves the current state (registers) on a stack.
4. The CPU fetches and checks the source of the interrupt, then calls the **Interrupt Service Routine (ISR)** — also called the interrupt handler — to process it.
5. Once the ISR completes, the CPU **restores** the saved state from the stack and **resumes** the previous task.




**Interrupt Priority and the Stack**

Students often write that the CPU "stops everything" when an interrupt arrives. This loses marks. The correct sequence is: the OS assigns a **priority**; if the interrupt's priority is **higher** than the current task, the CPU halts and pushes the current state onto a **stack** before fetching the ISR. Lower-priority interrupts wait in the queue — the current task is not abandoned, only temporarily paused.

**Interrupt Handling — 5-Mark Mark Scheme Response**

Describe how interrupts are used when a key is pressed on a keyboard. [5 marks]

**Mark-scheme points — any five from:**

1. Key press generates the interrupt.
2. Interrupt is given a priority.
3. Interrupt is sent to the CPU / placed in a queue.
4. CPU stops the current task to check the queue / service the interrupt.
5. CPU uses an Interrupt Service Routine (ISR) / interrupt handler to process it.
6. If the key press has the highest priority, the interrupt is processed first.

## 4.2 Types of Programming Language, Translators and IDEs

### High-Level vs Low-Level Languages

| Aspect | High-Level Language | Low-Level Language |
|---|---|---|
| Syntax | Similar to English; easier to read and write | Uses binary (machine code) or mnemonics |
| Hardware control | Limited — no direct register or memory access | Full control; can directly manipulate hardware |
| Portability | Portable / machine independent | Hardware-dependent; not portable |
| Execution speed | Slower (requires translation at run time) | Very fast; close to machine code |
| Ease of debugging | Easier; better error messages and readable syntax | Difficult to debug manually |
| Memory efficiency | Less memory efficient | More memory efficient |
| Translator needed | Compiler or interpreter | Assembler |
| Examples | Python, Java, C++, Visual Basic | Assembly language, machine code |

**Assembly language** is a form of low-level language that uses **mnemonics** (short, human-readable codes such as `ADD`, `MOV`, `LDA`) and requires an **assembler** to convert them into machine code. Assembly language is close to the language processed directly by computers and is used to communicate directly with hardware.

**High-Level Language Benefits — Exam Recall**

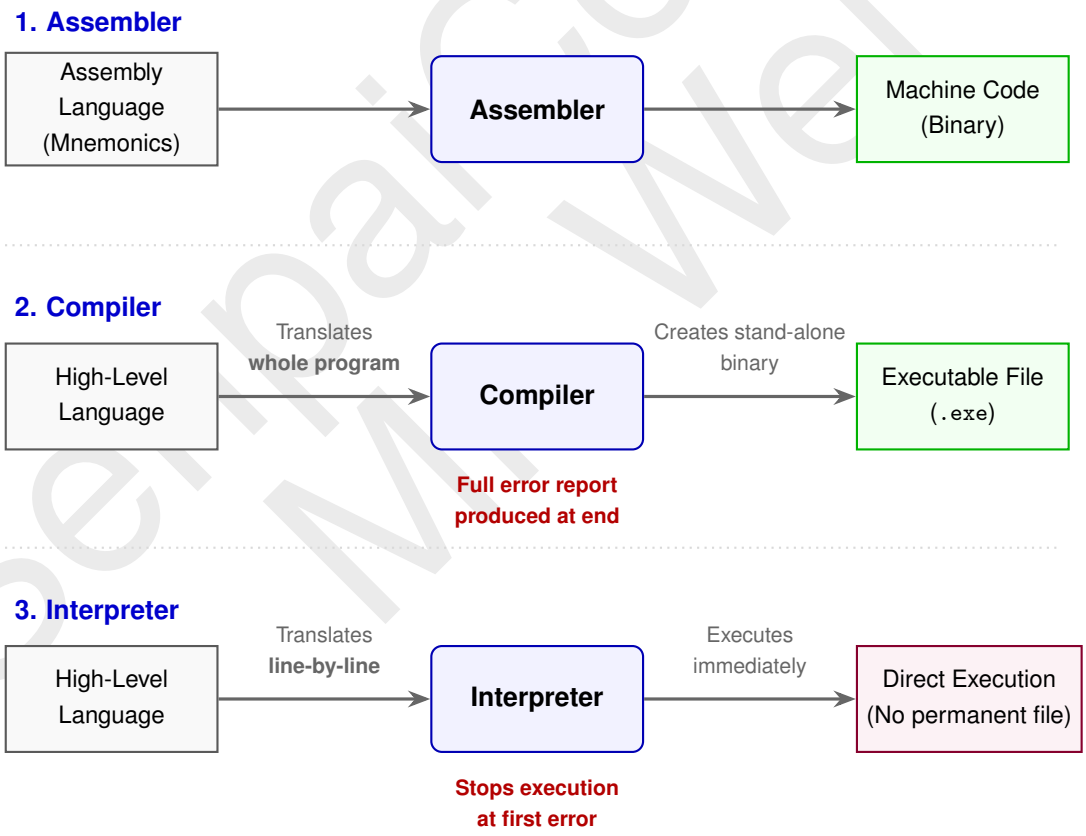
The standard benefits Cambridge awards for high-level languages: (1) easier to read/write/understand, (2) easier to debug, (3) machine independent/portable, (4) less likely to make errors, (5) programmer does not need knowledge of manipulating memory locations or registers. Pick any three for a 3-mark question (s25_12 Q3a pattern).

**Assembly Language — Memory Efficiency is the Advantage**

Cambridge mark schemes confirm that **assembly language is memory efficient** (correct answer to 2024 paper MCQ on assembly advantages). A common student error is writing that assembly is "easier to understand" — this is an advantage of *high-level* languages. Low-level language benefits are: memory efficient, direct hardware manipulation, faster execution, no translator required, can use specialised hardware, no portability requirement.

## Translators — Assembler, Compiler and Interpreter

| Translator | Input | Output | How it Works |
|---|---|---|---|
| Assembler | Assembly language | Machine code | Translates mnemonics into binary code; used for embedded systems and low-level hardware control. |
| Compiler | High-level language | Machine code — executable file | Translates the *whole* program at once before execution; creates a standalone `.exe` or binary; produces a full error report of all errors if compilation fails. |
| Interpreter | High-level language | No permanent output file | Translates and executes code *line by line*; stops at the first error showing its location; programmer corrects it and continues; used during development and debugging. |



**Compiler vs Interpreter — Detailed Comparison**

| Feature | Compiler | Interpreter |
|---|---|---|
| Translation unit | Whole program at once | One line (statement) at a time |
| Execution speed | Fast after compilation | Slower due to continuous translation |
| Error handling | All errors reported together in a single report after translation | Stops at first error; reports location immediately; allows correction and continuation |
| Output file | Creates executable file (`.exe` or binary) | No permanent file created |
| Source code security | Source code hidden via executable | Source code remains exposed/readable |
| Code portability | Executable tied to one platform | Source code portable to any system with the interpreter |
| Preferred use | Final/complete program deployment | Program development and debugging |

**Why Use an Interpreter During Development?**

Describe the advantages of using an interpreter instead of a compiler during software development. [4 marks]

**Mark-scheme points — any four from:**

- Each statement is translated and executed before the next.
- Translator stops when an error is found.
- Programmer can correct and continue from where the error was found (real-time debugging).
- Does not need to retranslate all the code again after a fix.
- Can test sections of the code without the whole program being complete.
- Can test without all the program code being functional or accurate.

**Using Both Translators Together — Development vs Deployment**

Programmers typically use an **interpreter** during development (fast error location, real-time debugging, partial code testing) and switch to a **compiler** once the program is complete — to generate a standalone executable file that no longer needs the interpreter to run. If the program compiles without errors, there are no syntax errors remaining (s20_12 Q2a mark scheme).

## Integrated Development Environments

An **IDE** (Integrated Development Environment) is a software suite that provides a comprehensive environment for programmers to write, test, debug, and run their code efficiently.

| IDE Function | Description |
|---|---|
| Code editor | Text editor that supports syntax highlighting and formatting; allows the programmer to write and modify the program. |
| Translator | Built-in compiler or interpreter to translate source code into machine code. |
| Run-time environment | Allows the user to run/test and execute code within the IDE and observe the output. |
| Error diagnostics | Identifies syntax and logical errors; displays line numbers and error messages so programmers can locate and fix faults. |
| Auto-completion | Predicts and completes code as the programmer types (e.g. variable names, function names, keywords). |
| Auto-correction | Fixes basic typos and formatting issues automatically; if a command word is misspelled it is corrected. |
| Prettyprint | Automatically formats code layout and colours keywords/identifiers for readability (syntax highlighting). |

**IDE Functions — Three-Mark Exam Pattern**

The most common IDE question format: "Integrated development environments provide translators. Identify three other common functions of an IDE." [3 marks]. The accepted answers are: **code editor, run-time environment, error diagnostics, auto-complete, auto-correct, prettyprint**. When the question asks for the function *and* its role (4-mark pattern), state the function name (1 mark) and describe what it does (1 mark) for each of two functions (s25_12 Q3b-ii pattern).