

Software

IGCSE Computer Science 0478 · Topic 4 revision notes

Everything from the 6 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

- | | |
|---|--|
| 1. <u>System software and application software</u> | 4. <u>High-level and low-level languages</u> |
| 2. <u>The operating system, firmware and the hardware</u> | 5. <u>Assemblers, compilers and interpreters</u> |
| 3. <u>Interrupts</u> | 6. <u>Integrated development environments (IDEs)</u> |

1 System software and application software

Syllabus 4.1

System software provides the services that the **computer** requires to run and be managed. It includes the **operating system** and **utility software** (such as anti-virus, back-up and defragmentation). **Application software** provides the services that the **user** requires to carry out a task, such as writing a letter, browsing the web or editing a photo. Applications cannot run without the system software underneath them.

Two kinds of software

System software

services the COMPUTER needs

- Operating system
- Anti-virus (utility)
- Disk defragmenter
- Back-up utility
- Device drivers

Application software

services the USER needs

- Word processor
- Spreadsheet
- Web browser
- Photo editor
- Game

Fig. 1: system software serves the computer; application software serves the user

Worked example: Classify

Classify each as system or application software: a spreadsheet, an anti-virus program, a printer driver, a presentation program.

1. Spreadsheet: helps the user calculate and present data, so application.
2. Anti-virus: protects the computer from malware, a utility, so system.
3. Printer driver: lets the OS communicate with the printer, so system.
4. Presentation program: helps the user make slides, so application.

Answer: System: anti-virus, printer driver. Application: spreadsheet, presentation program.

Ask who it serves.

- ✓ **System software:** services the computer requires; the OS and utility software.
- ✓ **Application software:** services the user requires to do a task.
- ✓ **Utilities:** anti-virus, back-up, defragmenter, compression, encryption.
- ✓ **Describe, not just name:** say what the software does as well as giving an example.

Watch out!

Naming a brand ("Windows") is fine as an example, but when asked to describe system software you must say what it does: it provides the services the computer needs, such as managing hardware and running applications.

A web browser is **application** software, even though it feels like part of the computer. It does a job for the user.

2 The operating system, firmware and the hardware

Syllabus 4.1

The **operating system (OS)** manages the computer's hardware and software and provides a **platform for running applications**. Its functions are: managing **files**, handling **interrupts**, providing an **interface**, managing **peripherals and drivers**, managing **memory**, managing **multitasking**, providing a platform for applications, providing **system security** and managing **user accounts**. Applications run on the OS, the OS is run on the **firmware**, and the firmware (the **bootloader**) runs on the **hardware**.

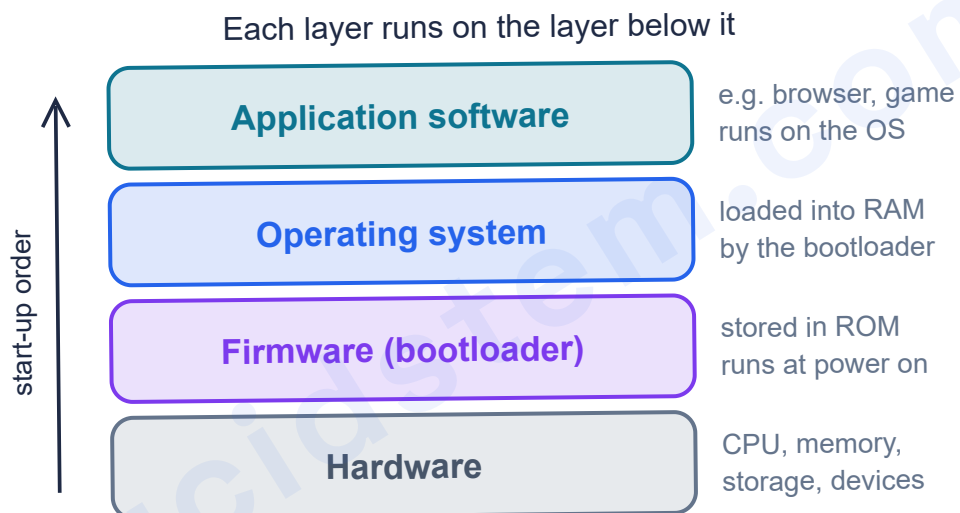


Fig. 1: applications run on the operating system, which is run on the firmware, which runs on the hardware

Worked example: Start-up order

Describe how hardware, firmware and the operating system allow a user to open a word processor.

1. Power on: the hardware runs the bootloader, firmware stored in ROM.
2. The bootloader loads the operating system from secondary storage into RAM and starts it.
3. The OS is run on the firmware and takes control of the hardware.
4. The user opens the word processor; the OS loads it into memory and it runs on the OS, using the OS's services.

Answer: Hardware runs the bootloader (firmware); the bootloader loads the OS; the application runs on the OS.

Name it, then explain it.

- ✓ **Nine functions:** files, interrupts, interface, peripherals and drivers, memory, multitasking, platform for applications, security, user accounts.
- ✓ **Add detail:** say what the OS does for each function, not just its name.
- ✓ **Layers:** application on OS, OS on firmware, firmware (bootloader) on hardware.
- ✓ **Bootloader:** firmware in ROM that loads the OS into RAM at start-up.

Watch out!

"It manages memory" on its own often scores nothing. Add the detail: it *allocates* memory to each program and stops programs overwriting each other's memory.

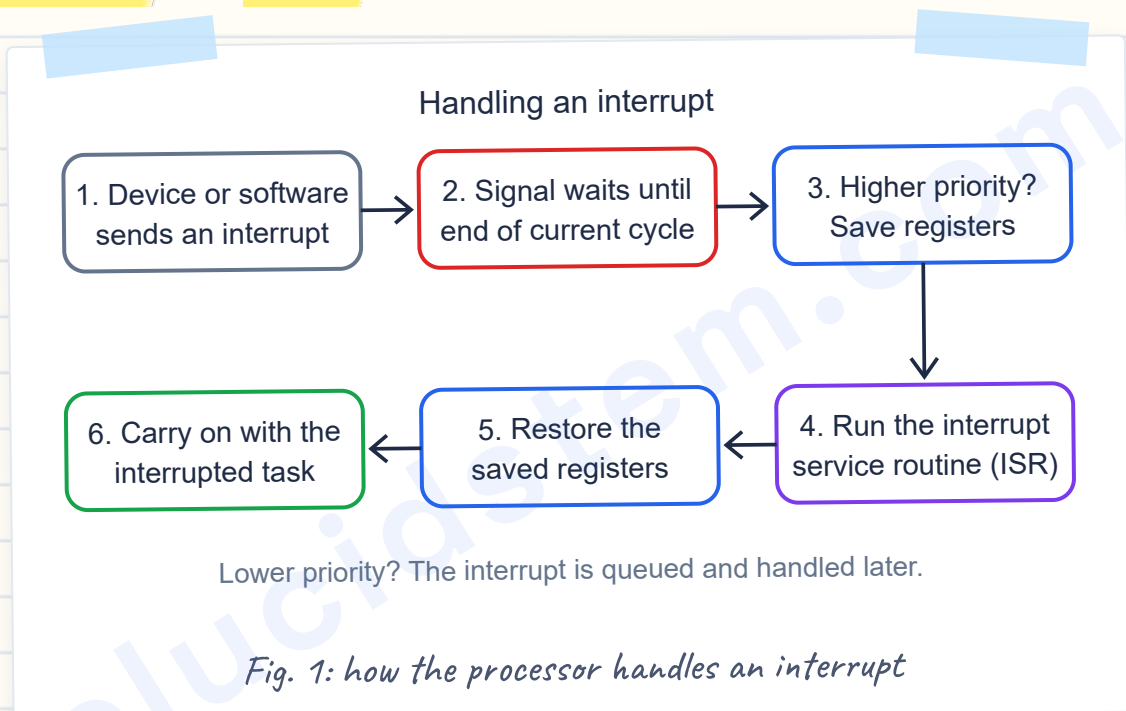
The bootloader is **firmware**, not the operating system. The OS is usually stored on the SSD or hard disk, not in ROM.

thelucidstem.com

3 Interrupts

Syllabus 4.1

An **interrupt** is a signal sent to the processor, by a device or by software, saying that something needs **attention**. **Hardware interrupts** come from devices, e.g. pressing a key or moving the mouse. **Software interrupts** come from programs, e.g. **division by zero** or two processes trying to access the **same memory location**. At the end of each fetch-decode-execute cycle the processor checks for interrupts. If one has a higher **priority** than the current task, it **saves** the contents of its registers, runs the matching **interrupt service routine (ISR)**, then **restores** the registers and carries on.



Worked example: A key press

A user is typing in a word processor while a large file is being saved. Describe what happens when the user presses the letter K.

1. The keyboard (hardware) generates an interrupt and sends the signal to the processor.
2. At the end of the current cycle the processor checks for interrupts and finds the keyboard interrupt has a higher priority than the save.
3. The processor saves the contents of its registers for the saving task.
4. It runs the ISR for the keyboard: the code for K is read and the letter is displayed.
5. The registers are restored and saving the file continues where it left off.

Answer: Interrupt sent, checked at end of cycle, priority compared, registers saved, keyboard ISR run, registers restored, save resumes.

Say every step.

- ✓ **Generated:** hardware (key press, mouse) or software (division by zero, memory clash) sends a signal.
- ✓ **Checked:** at the end of the current fetch-decode-execute cycle; compare priorities.
- ✓ **Handled:** save registers, run the interrupt service routine, restore registers.
- ✓ **Result:** the event is dealt with and the original task resumes.

Watch out!

Interrupts are not only for errors. A key press is an interrupt too: it simply tells the processor there is new input to deal with.

Do not say the processor stops immediately. It finishes the current fetch-decode-execute cycle first, then checks for interrupts.

thelucidstem.com

4 High-level and low-level languages

Syllabus 4.2

A **high-level language** (such as Python, Java or Visual Basic) uses English-like statements, so it is **easy to read, write and debug** and is **machine independent**: the same program can run on different types of processor once translated. A **low-level language** is close to the processor's own instructions: **machine code** (binary) and **assembly language**, which uses short **mnemonics** such as LDD and ADD. Low-level code is hard to read and debug and is **machine dependent**, but allows **direct manipulation of the hardware**. Assembly language needs an **assembler** to translate it into machine code.

One task, three levels: add two numbers and store the total

High-level language

Total ← Price + Tax

Assembly language (low-level, mnemonics)

LDD 200 ADD 201 STO 202

load, add, store

Machine code (low-level, binary the CPU runs)

0001 1100 1000 0010 1100 1001 0011 1100 1010

Fig. 1: the same calculation as high-level code, assembly language and machine code (illustrative opcodes)

Worked example: Choose a language

A programmer is writing a driver that must control a new graphics card's registers as fast as possible. Another is writing a school timetable app that must run on Windows and macOS. Which type of language should each use?

1. The driver needs direct manipulation of the hardware and efficient code: a low-level language (assembly).
2. The app must run on different machines and be easy to maintain: a high-level language, which is machine independent and easier to write and debug.

Answer: Driver: low-level (assembly). Timetable app: high-level.

Four comparison points.

- ✓ **Reading and writing:** high-level is English-like; low-level is mnemonics or binary.
- ✓ **Debugging:** easier in high-level.
- ✓ **Machine independence:** high-level is portable; low-level is for one processor type.
- ✓ **Hardware:** only low-level allows direct manipulation of hardware. Assembly needs an assembler.

Watch out!

"High-level is easier" is too vague for a mark. Say *why*: it uses English-like statements, so it is easier to read, write and debug.

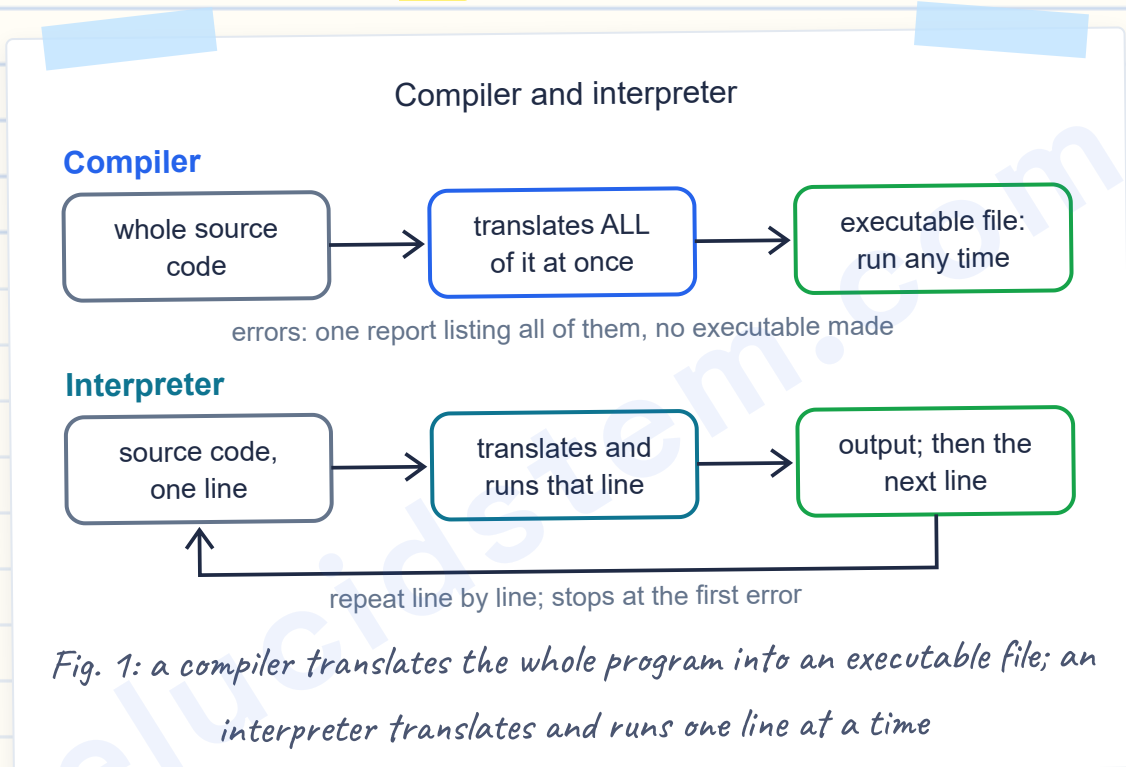
Assembly language is **low-level**, even though it uses letters. It still matches the processor's instructions one for one, and it still needs an assembler.

thelucidstem.com

5 Assemblers, compilers and interpreters

Syllabus 4.2

A **translator** turns source code into machine code. An **assembler** translates **assembly language**. A **compiler** translates the **whole** high-level program at once before it is executed, producing an **executable file**; if there are errors it gives an **error report for the whole code**. An **interpreter** translates and executes the code **line by line** and **stops when it finds an error**. An interpreter is mostly used while **developing** a program; a compiler is used to translate the **final** program.



Worked example: Compiler

What happens when the program above is compiled?

1. The compiler tries to translate all five lines before anything is executed.
2. It finds the invalid operator `>==` in line 2 and the missing closing quote in line 4, and carries on checking to the end.
3. It produces one error report listing every error found, with line numbers.
4. No executable file is produced, so none of the program runs until the errors are fixed.

Answer: An error report for the whole code; no executable; nothing runs.

Whole or line by line.

- ✓ **Assembler:** assembly language to machine code.
- ✓ **Compiler:** whole high-level program at once; executable file; error report for the whole code.
- ✓ **Interpreter:** translates and executes line by line; stops at the first error.
- ✓ **Use:** interpreter while developing; compiler for the final program.

Watch out!

An assembler is never used for a high-level language, and a compiler or interpreter is never used for assembly language. Match the translator to the language.

A compiler does not run the program while it is translating. It reports all the errors it can find, and only an error-free program becomes an executable file.

thelucidstem.com

6 Integrated development environments (IDEs)

Syllabus 4.2

An **integrated development environment (IDE)** is software that provides the tools a programmer needs to **write, translate, test and debug** program code in one place. Common functions are: a **code editor**, a **run-time environment**, **translators** (a compiler and/or interpreter), **error diagnostics**, **auto-completion**, **auto-correction** and **prettyprint**. Together they make programs quicker to write and errors quicker to find and fix.

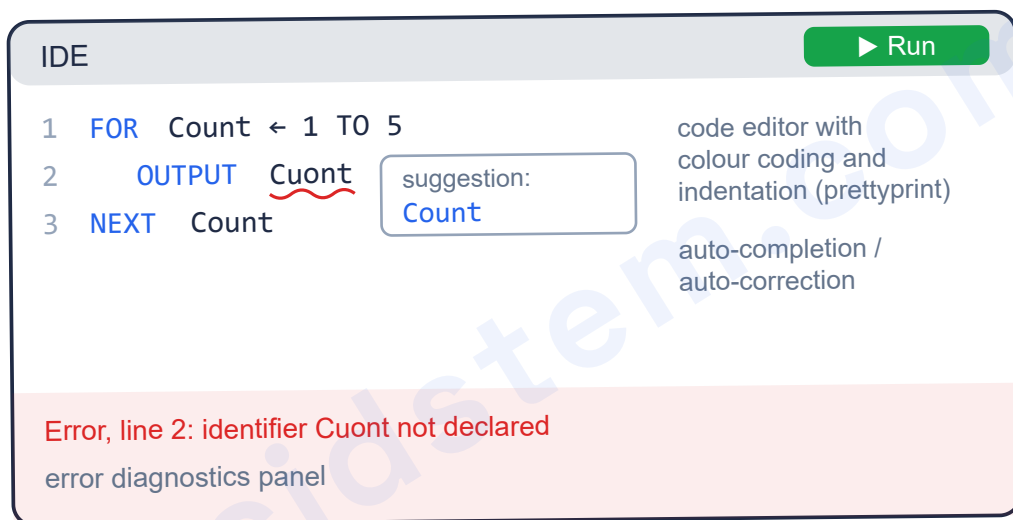


Fig. 1: an IDE window: the editor highlights an error on line 2, suggests a fix, and the diagnostics panel reports it

Worked example: Match the function

Name the IDE function: (a) keywords such as WHILE appear in blue and the loop body is indented; (b) typing Tot shows a list containing Total; (c) the line with a missing bracket is underlined in red with a message; (d) the program is executed and its output appears in a window.

1. (a) Colour coding and indentation make the structure clear: prettyprint.
2. (b) Suggesting the rest of an identifier: auto-completion.
3. (c) Locating and describing an error: error diagnostics.
4. (d) Running and testing the program inside the IDE: run-time environment.

Answer: (a) prettyprint (b) auto-completion (c) error diagnostics (d) run-time environment

Seven functions.

- ✓ **Write:** code editor, auto-completion, auto-correction, prettyprint.
- ✓ **Translate:** built-in compiler and/or interpreter.
- ✓ **Test and debug:** run-time environment, error diagnostics.
- ✓ **For each:** say what it does and how it helps the programmer.

Watch out!

Saying an IDE "helps you write code" earns little. Name the function and explain how it helps, e.g. auto-completion suggests the rest of an identifier, so it is typed faster and without spelling mistakes.

Auto-completion and auto-correction are different. Auto-**completion** finishes what you are typing; auto-**correction** fixes a mistake you have made.