Cambridge

OL- IGCSE

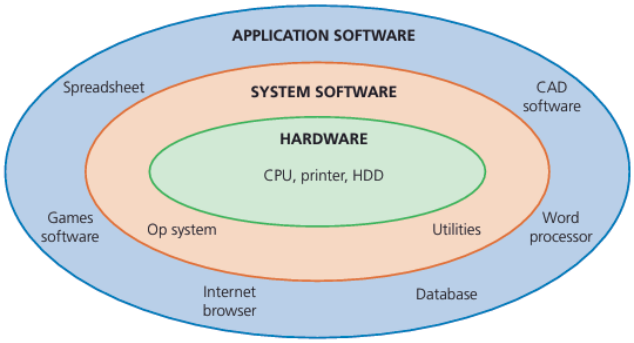
Computer science

CODE: (0478)

Chapter 04

Software

4.1 Types of software and interrupts

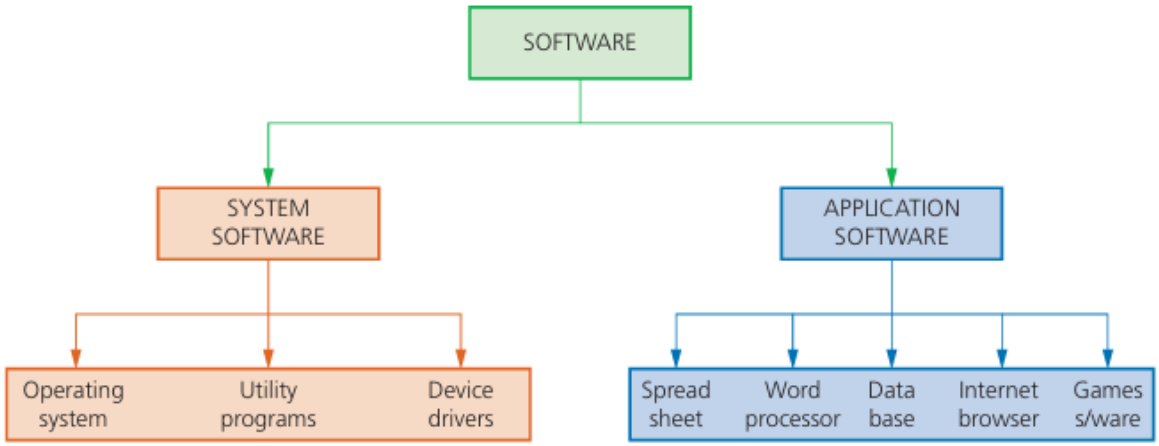


▲ **Figure 4.1** Software and hardware hierarchy

4.1.1 System software and application software

All computers begin life as a group of connected hardware items. Without software, the hardware items would be useless.

You will notice from Figure 4.1 that there are two types of software: system software and application software:



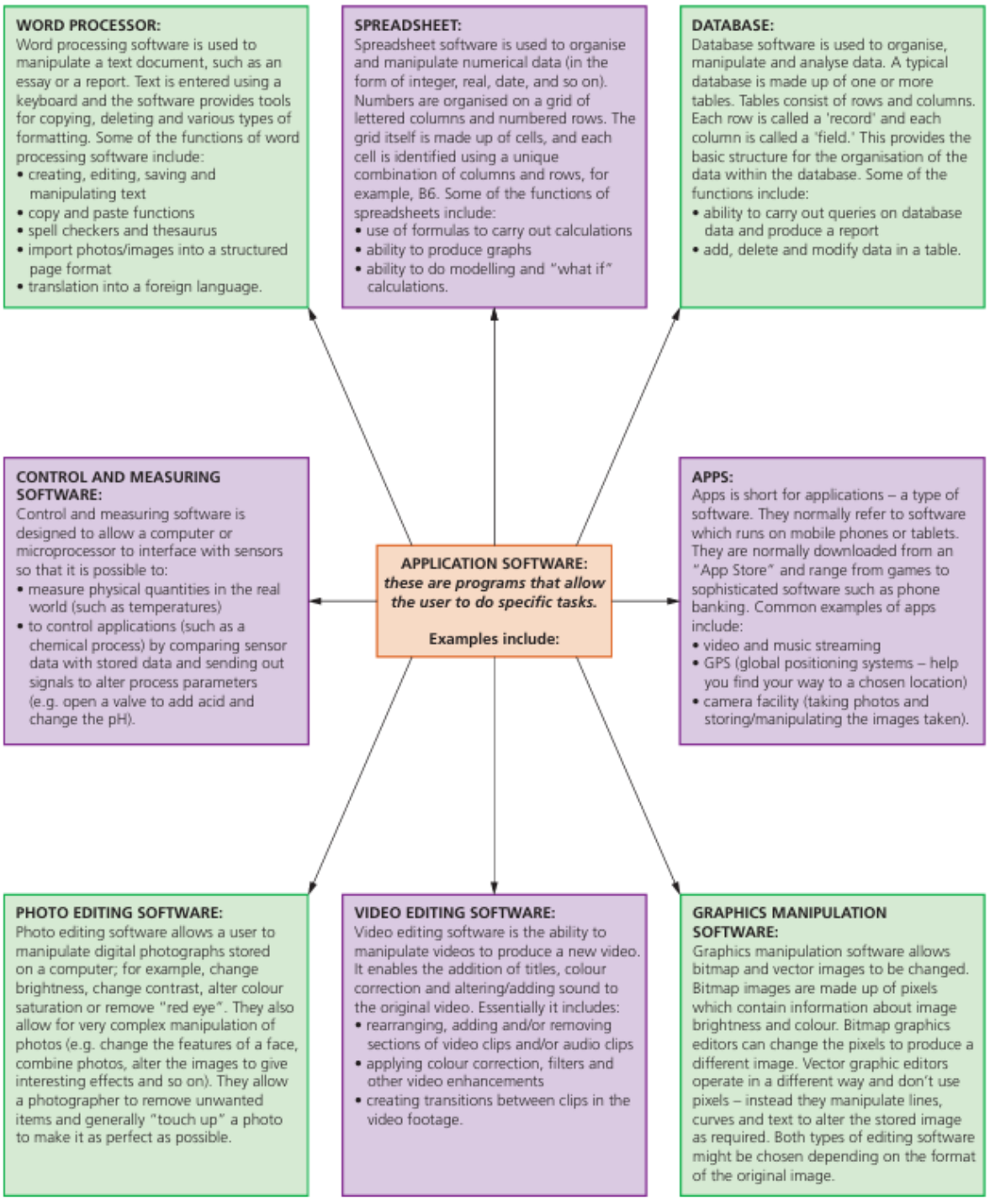
▲ **Figure 4.2** Software types

General features of system software

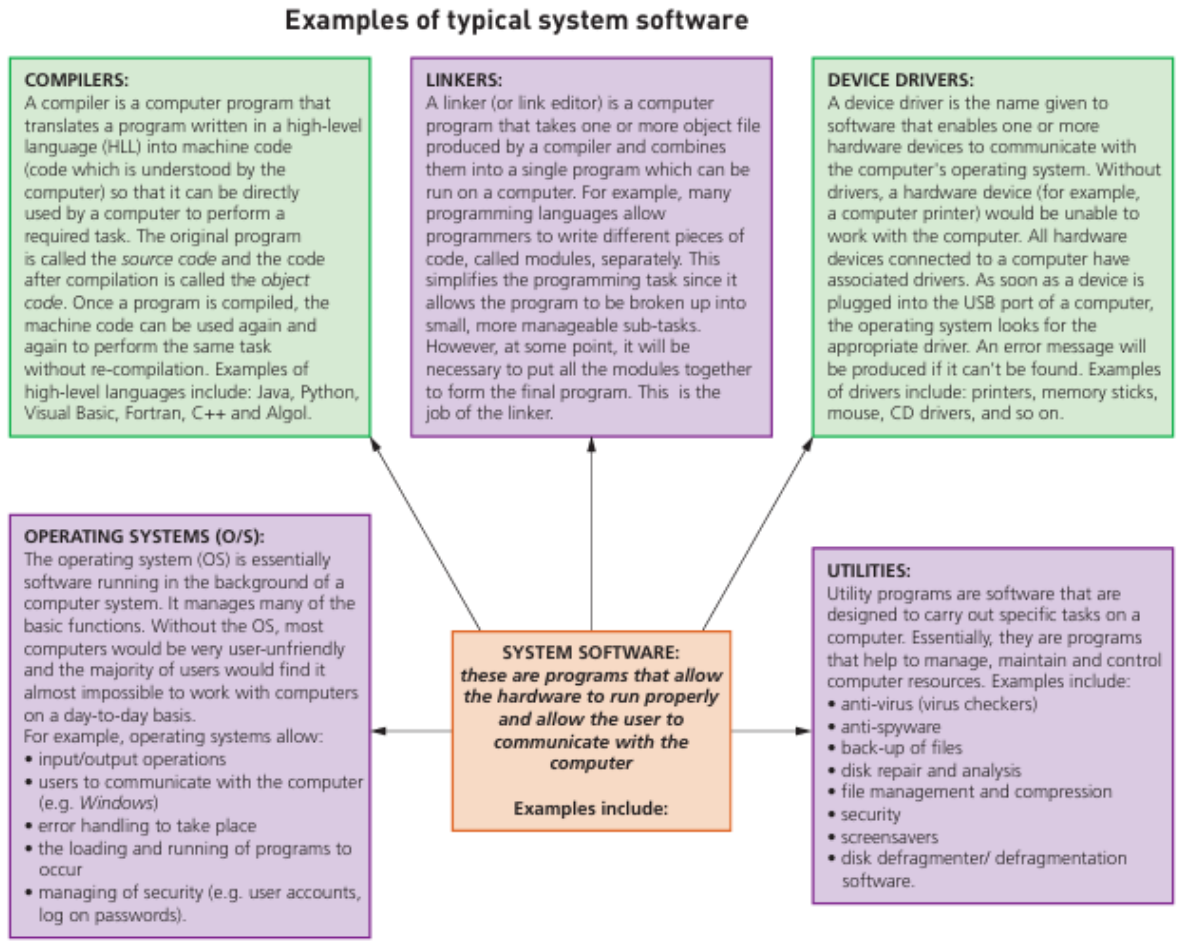
» Set of programs to control and manage the operation of computer hardware
» Provides a platform on which other software can run
» Required to allow hardware and software to run without problems
» Provides a human computer interface (HCI)
» Controls the allocation and usage of hardware resources.

General features of application software
» Used to perform various applications (apps) on a computer
» Allows a user to perform specific tasks using the computer's resources
» May be a single program (for example, NotePad) or a suite of programs (for example, Microsoft Office)
» User can execute the software as and when they require.

Examples of typical application software



▲ **Figure 4.3** Application software



▲ **Figure 4.4** System software

Utility software (utilities)

Computer users are provided with a number of utility programs (often simply referred to as utilities) that are part of the system software.

Utility programs offered by most computer system software include:

» Virus checkers
» Defragmentation software
» Disk contents analysis and repair
» File compression and file management
» Back-up software
» Security
» Screensavers.

Virus checkers (anti-virus software)

Any computer (including mobile phones and tablets) can be subject to a virus attack. Operating systems offer virus checkers, but these must be kept thoroughly up to date and should run in the background to maintain their ability to guard against being infected by such **malware**.

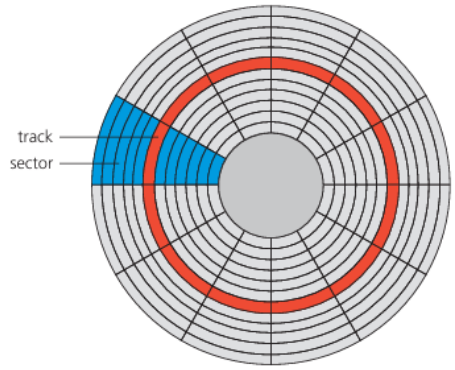
Running anti-virus software in the background on a computer will constantly check for virus attacks. Although various types of anti-virus software work in different ways they all have the following common features:

» They check software or files before they are run or loaded on a computer
» Anti-virus software compares a possible virus against a database of known viruses
» They carry out heuristic checking – this is the checking of software for types of behaviour that could indicate a possible virus; this is useful if software is infected by a virus not yet on the database
» Any possible files or programs which are infected are put into quarantine which: – allows the virus to be automatically deleted, or– allows the user to make the decision about deletion (it is possible that the user knows that the file or program is not infected by a virus – this is known as a false positive and is one of the drawbacks of anti-virus software)
» Anti-virus software needs to be kept up to date since new viruses are constantly being discovered
» Full system checks need to be carried out once a week,

Defragmentation software

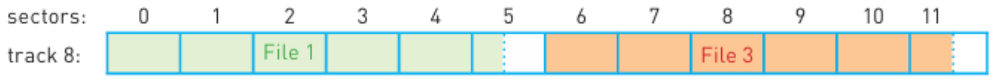
It would obviously be advantageous if files could be stored in **contiguous** sectors considerably reducing HDD head movements. (Note that due to the different operation of SSDs when accessing data, this is not a problem when using solid state devices.)

Consider the following scenario using a disk with 12 (numbered 0 to 11) sectors per surface:



▲ **Figure 4.5** Hard disk drive tracks and sectors

If this continues, the files just become more and more scattered throughout the disk surfaces. It is possible for sectors 4, 5 and 6 (on track 8) to eventually become used if the disk starts to fill up and it has to use up whatever space is available. A **disk defragmenter** will rearrange the blocks of data to store files in **contiguous** sectors wherever possible. After defragmentation Track 8 would now become:



Back-up software

While it is sensible to take manual back-ups using, for example, a memory stick or portable HDD, it is also good practice to use the operating system **back-up utility**. This utility will:

» Allow a schedule for backing up files to be made
» Only carry out a back-up procedure if there have been any changes made to a file.

For total security there should be three versions of a file:
1. The current (working) version stored on the internal HDD or SSD
2. A locally backed up copy of the file (stored on a portable SSD, for example) 3 a remote back-up version stored well away from the computer (for example, using cloud storage).

The Microsoft Windows environment offers the following facilities using the back-up utility:

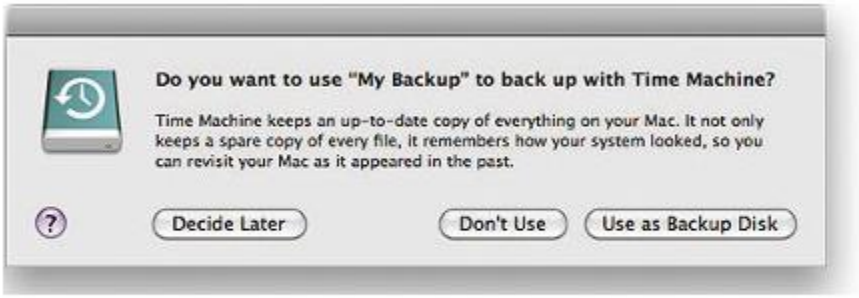
» Restore data, files or the computer from the back-up (useful if there has been a problem and files have been lost and need to be recovered)
» Create a restore point (this is basically a kind of 'time machine' where your computer can be restored to its state at this earlier point in time; this can be very useful if a very important file has been deleted and can't be recovered by any of the other utilities)
» Options of where to save back-up files; this can be set up from the utility to ensure files are automatically backed up to a chosen device.

Windows uses **File History**, which takes snapshots of files and stores them on an external HDD at regular intervals. Over a period of time, **File History** builds up a vast library of past versions of files – this allows a user to choose which version of the file they want to use. **File History** defaults to backing up every hour and retains past versions of files for ever unless the user changes the settings.

Mac OS offers the **Time Machine** back-up utility. Time machine will automatically:

» Back-up every hour
» Do daily back-ups for the past month, and
» Weekly back-ups for all the previous months.

The following screen shows the Time Machine message:



▲ **Figure 4.6** Time machine message on Mac OS

Security software

Security software is an over-arching utility that:
» Manages access control and user accounts (using user IDs and passwords)
» Links into other utility software, such as virus checkers and spyware checkers
» Protects network interfaces (for example, through the use of firewalls)
» Uses encryption and decryption to ensure any intercepted data is meaningless without a decryption key
» Oversees the updating of software (does the update request come from a legitimate source, for example).

Screensavers

Screensavers are programs that supply moving and still images on the monitor screen after a period of inactivity by the computer. They were originally developed to protect older CRT (cathode ray tube) monitors which would suffer from 'phosphor burn' if the same screen image remained for any length of time. With modern LCD and OLED screens, this problem no longer exists; consequently, screensavers are now mostly just a way of customising a device.

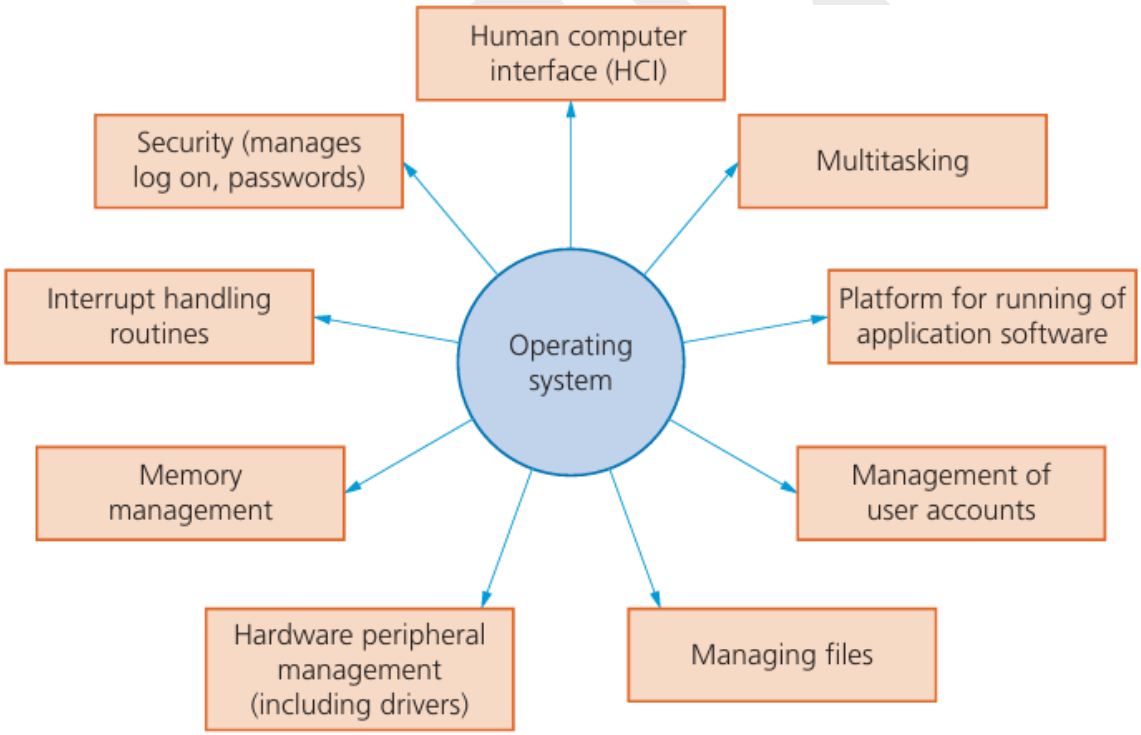
Device drivers

Device drivers are software that communicate with the operating system and translate data into a format understood by a hardware peripheral device. Without device drivers, a hardware device would be unable to work with a computer – a message such as 'device not recognised' would appear on the screen.

All USB device drivers contain a collection of information about devices called **descriptors**; this allows the USB bus to ask a newly connected device what it is.

4.1.2 Operating systems

To enable computer systems to function correctly and allow users to communicate with computer systems, software known as an **operating system** needs to be installed. An operating system provides both the environment in which applications can be run and a useable interface between humans and computer.



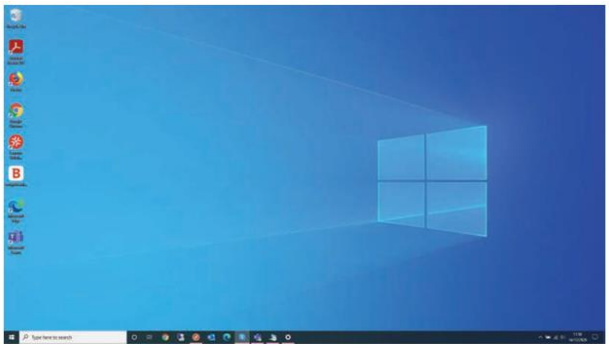
▲ **Figure 4.7** Operating system functions

Human computer interface (HCI)

The human computer interface (HCI) is in the form of a Command Line Interface (CLI) or a Graphical User Interface (GUI).

GUIs use various technologies and devices to provide the user interface. One of the most common is **WIMP (windows icons menu and pointing device),** which was developed for use on personal computers (PC).

More recently, devices such as mobile phones and tablets increasingly use touch screens and use **post-WIMP** interactions.



▲ **Figure 4.9** Windows screen showing icons

▼ **Table 4.1** Differences between GUI and CLI interfaces

Interface	Advantages	Disadvantages
command line interface (CLI)	the user is in direct communication with the computer the user is not restricted to a number of pre-determined options it is possible to alter computer configuration settings uses a small amount of computer memory	the user needs to learn a number of commands to carry out basic operations all commands need to be typed in which takes time and can be error-prone each command must be typed in using the correct format, spelling, and so on
graphical user interface (GUI)	the user doesn't need to learn any commands it is more user-friendly; icons are used to represent applications a pointing device (such as a mouse) is used to click on an icon to launch the application – this is simpler than typing in commands or a touch screen can be used where applications are chosen by simply touching the icon on the screen	this type of interface uses up considerably more computer memory than a CLI interface the user is limited to the icons provided on the screen needs an operating system, such as Windows, to operate, which uses up considerable memory

Who would use each type of interface?
» CLI: a programmer, analyst or technician;
basically, somebody who needs to have a direct communication with a computer to develop new software, locate errors and remove them, initiate memory dumps (contents of the computer memory at some moment in time), and so on

» GUI: the end-user who doesn't have or doesn't need to have any great knowledge of how the computer works; a person who uses the computer to run software or play games or stores/manipulates photographs, for example.

Memory management

Memory management carries out the following functions:
» Manages the primary storage (RAM) and allows data to be moved between RAM and HDD/SSD during the execution of programs » keeps track of all the memory locations

» Carries out memory protection to ensure that two competing applications cannot use the same memory locations at the same time. If this wasn't done the following might happen:
– data would probably be lost
– applications could produce incorrect results (based on the wrong data being in memory locations)
– potential security issues (if data is placed in the wrong location, it might make it accessible to other software, which would be a major security issue)
– in extreme cases, the computer could crash

Security management

Security management is another part of a typical operating system; the function of security management is to ensure the integrity, confidentiality and availability of data

This can be achieved as follows (many of these features are covered in more depth elsewhere in this book):

» By carrying out operating system updates as and when they become available
» Ensuring that anti-virus software (and other security software) is always up to date, preserving the integrity, security and privacy of data
» By communicating with, for example, a firewall to check all traffic to and from the computer
» By making use of privileges to prevent users entering 'private areas' on a computer that permits multi-user activity (this is done by setting up user accounts and making use of passwords and user IDs); this helps to ensure the privacy of data
» By maintaining access rights for all users
» By offering the ability for the recovery of data (and system restore) when it has been lost or corrupted » by helping to prevent illegal intrusion into the computer system (also ensuring the privacy of data).

Hardware peripheral management

Hardware management involves all input and output peripheral devices. Hardware management:

» Communicates with all input and output devices using device drivers
» Uses a device driver to take data from a file (defined by the operating system) and translates it into a format that the input/output device can understand
» Ensures each hardware resource has a priority so that they can be used and released as required
» Manages input/output devices by controlling queues and buffers; consider the role of the printer management when printing out a document:

– first of all, the printer driver is located and loaded into memory
– then the data is sent to a printer buffer ready for printing
– if the printer is busy (or the printing job has a low priority) then the data is sent to a printer queue before it can be sent to the printer buffer
– it will send various control commands to the printer throughout the printing process
– it receives and handles error messages and interrupts from the printer.

File management

The main tasks of file management include:

» File naming conventions which can be used i.e. filename.docx (where the extension can be .bat, .htm, .dbf, .txt, .xls, etc.)

» Performing specific tasks (for example, create, open, close, delete, rename, copy, and move)

» Maintaining the directory structures » ensuring access control mechanisms are maintained (for example, access rights to files, password protection, or making files available for editing or locking them)

» Ensuring memory allocation for a file by reading it from the HDD/SSD and loading it into memory.

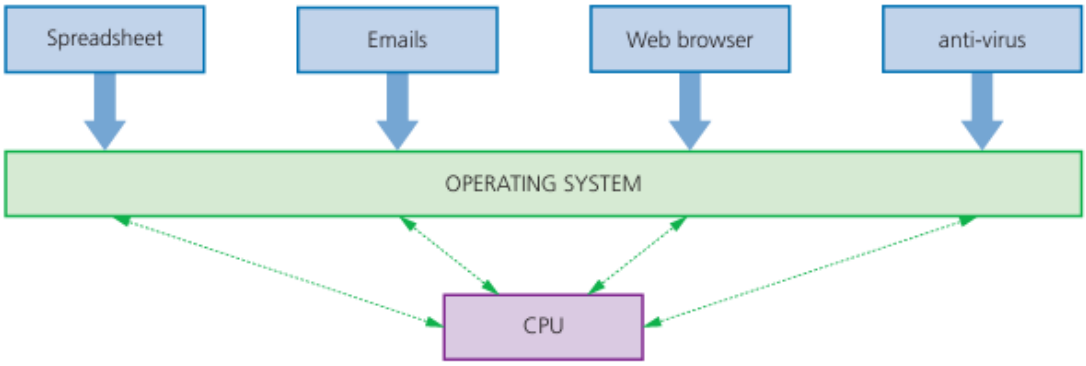
Interrupts,
Please refer to Section 4.1.4 for a discussion on interrupts.
Platform for running of application software
Please refer to Section 4.1.3 for a discussion on the running of application software.

Multitasking

Multitasking allows computers to carry out more than one task (i.e. a process) at a time. Each of the processes will share the hardware resources under the control of the operating system software.

To make sure that multitasking operates correctly (in other words, the processes don't clash with each other), the operating system needs to constantly monitor the status of each of the processes under its control:

» Resources are allocated to a process for a specific time limit
» The process can be interrupted while it is running
» The process is given a priority so it can have resources according to its priority (the risk here is that a low priority process could be starved of resources).



▲ **Figure 4.10** Multitasking diagram

Management of user accounts

Computers allow more than one user to log onto the system. It is therefore important that users' data is stored in separate parts of the memory for security reasons (also refer to security management earlier in this section).

The operating system is given the task of managing these different user accounts. This allows each user to:

» Customise their screen layout and other settings
» Use separate folders and files and to manage these themselves.

Very often an **administrator** oversees the management of these user accounts. The administrator can create accounts, delete user accounts and restrict user account activity.

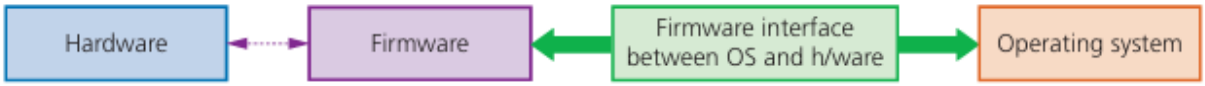
4.1.3 Running of applications

When a computer starts up, part of the operating system needs to be loaded into RAM – this is known as **booting up** the computer (or a **bootstrap loader).**

The BIOS is often referred to as firmware. **Firmware** is defined as a program that provides low level control for devices.

The BIOS program is stored in a special type of ROM, called an **EEPROM** (Electrically Erasable Programmable ROM).

However, while the BIOS is stored on an EEPROM, the BIOS **settings** are stored on a CMOS chip (Complementary Metal Oxide Semi-conductor).



▲ **Figure 4.11** Firmware interface between OS and hardware

4.1.4 Interrupts

An interrupt is a signal sent from a device or from software to the microprocessor. This will cause the microprocessor to temporarily stop what it is doing so that it can service the interrupt. Interrupts can be caused by:

» A timing signal » an input/output process (for example, a disk drive or printer requiring more data)

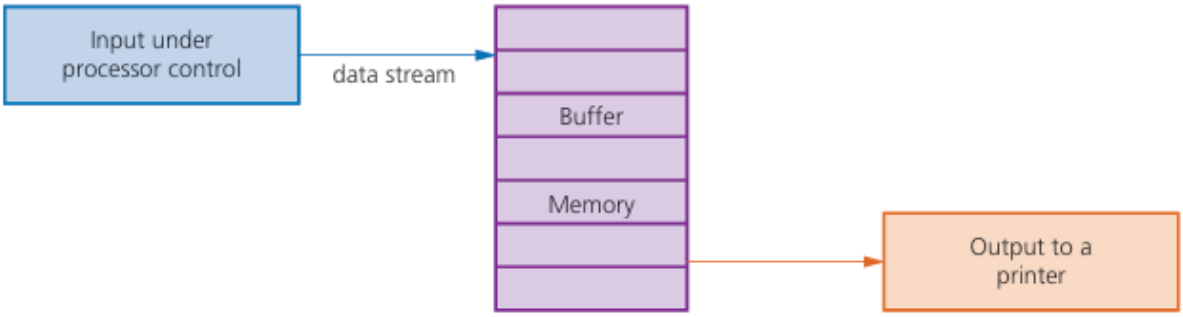
» A hardware fault (for example, a paper jam in the printer)

» User interaction

» Software errors that cause a problem

The computer needs to identify the interrupt type and also establish the level of **interrupt priority.**

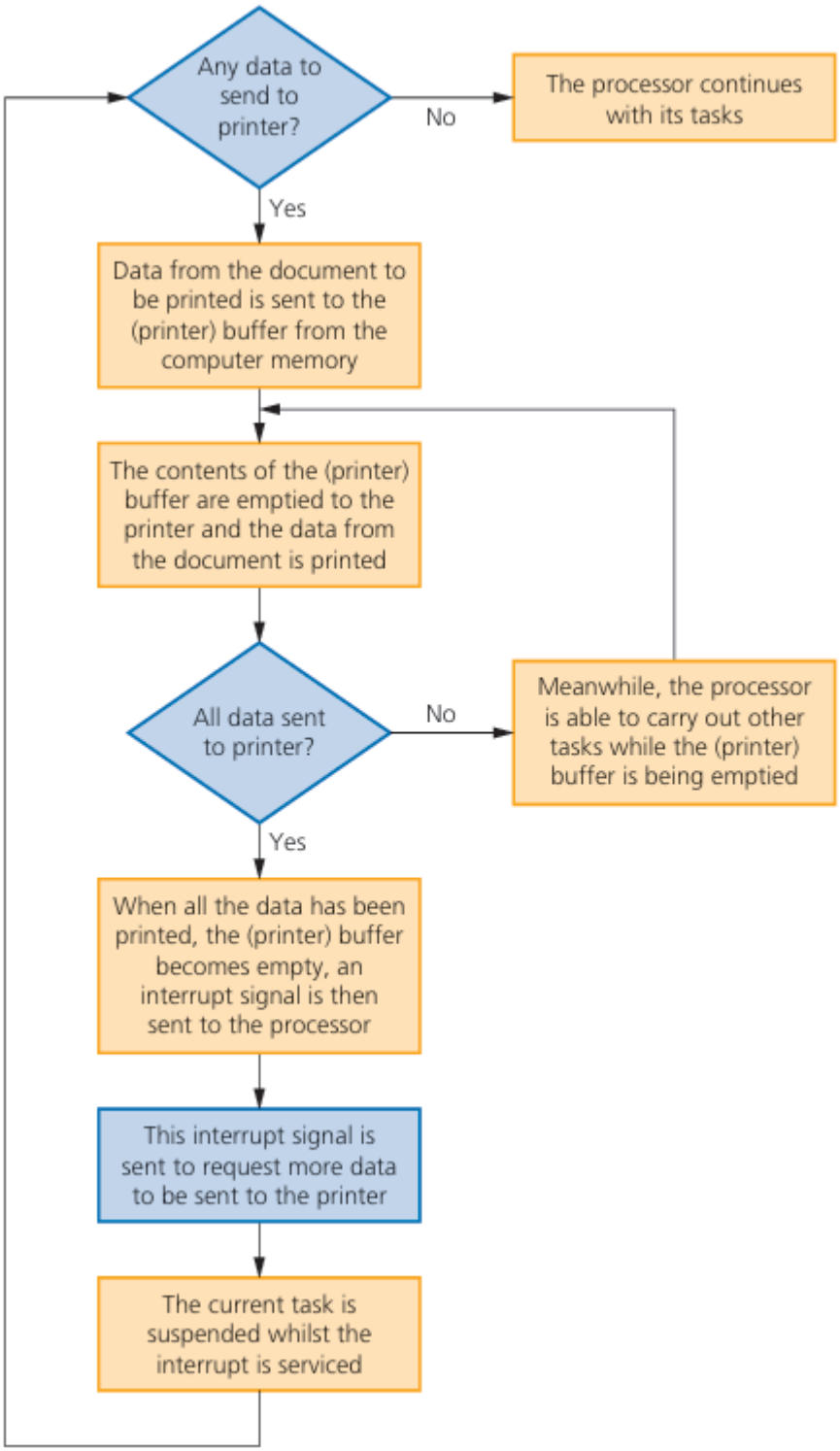
In reality, data is being passed in and out of memory very rapidly allowing both functions to be serviced. This can all be achieved by using an area in memory known as a **buffer**.



▲ **Figure 4.12** Use of a buffer when sending data to a printer (buffer used to store data temporarily since printer speed is much slower than microprocessor speed)

Whenever an interrupt is received it needs to be **serviced**. The status of the current task being run first needs to be saved. The contents of the Program Counter (PC) and other registers are saved. Then the **interrupt service routine (ISR)** is executed by loading the start address into the Program Counter (PC).

The important thing to remember here is the time taken to print out a document is **much** longer than the time it takes for the microprocessor to send data to the printer. Without buffers and interrupts, the microprocessor would remain idle waiting for a document to be printed.



▲ **Figure 4.13** Use of interrupts and buffers when printing a document

4.2 Types of programming language, translators and integrated development environments (IDEs)

Programmers use many different programming languages to communicate with computers. Computers only 'understand' their own language, called **machine code**. A program needs to be translated into machine code before it can be 'understood' by a computer.

A **computer program** is a list of instructions that enable a computer to perform a specific task. Computer programs can be written in **high-level languages** and **low-level languages** depending on the task to be performed and the computer to be used. Most programmers write programs in high-level languages.

4.2.1 High-level languages and low-level languages

High-level languages

High-level languages enable a programmer to focus on the problem to be solved and require no knowledge of the hardware and instruction set of the computer that will use the program.

High-level languages are designed with programmers in mind; programming statements are easier to understand than those written in a low-level language. This means that programs written in a high-level language are easier to:

» Read and understand as the language used is closer to English
» Write in a shorter time
» Debug at the development stage
» Maintain once in use

Low-level languages

Low-level languages relate to the specific architecture and hardware of a particular type of computer. Low-level languages can refer to **machine code**, the binary instructions that a computer understands, or **assembly language** that needs to be translated into machine code.

Machine code

Programmers do not usually write in machine code as it is difficult to understand, and it can be complicated to manage data manipulation and storage.

▼ **Table 4.2** Differences between high-level and low-level languages

Language	Advantages	Disadvantages
High-level	independent of the type of computer being used easier to read, write and understand programs quicker to write programs programs are easier and quicker to debug easier to maintain programs in use	programs can be larger programs can take longer to execute programs may not be able make use of special hardware
Low-level	can make use of special hardware includes special machine-dependent instructions can write code that doesn't take up much space in primary memory can write code that performs a task very quickly	it takes a longer time to write and debug programs programs are more difficult to understand

4.2.2 Assembly languages

Fewer programmers write programs in an assembly language. Those programmers who do, do so for the following reasons:

» To make use of special hardware
» To make use of special machine-dependent instructions
» To write code that doesn't take up much space in primary memory
» To write code that performs a task very quickly.

The following snippet of program to add two numbers together is written in a typical assembly language and consists of three statements:

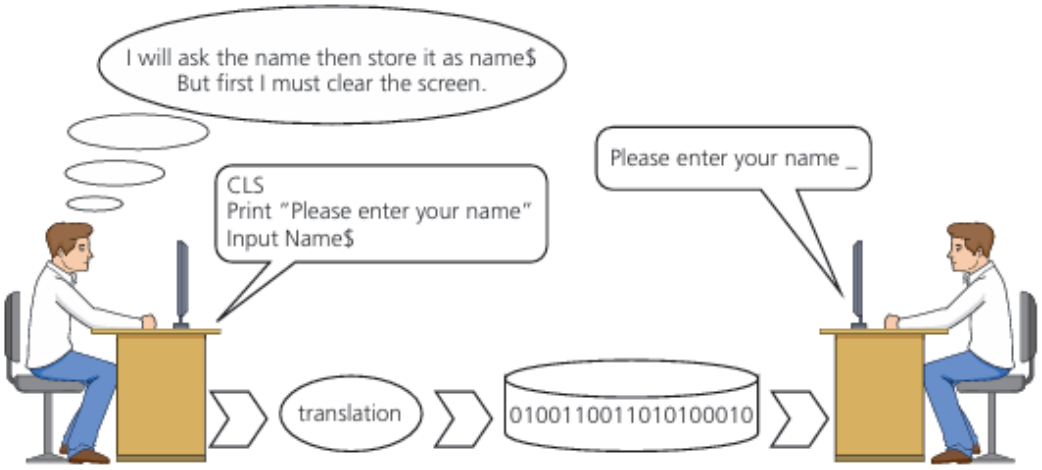
```
LDA             First
ADD             Second
STO             Sum
```

In order to understand this program, the programmer needs to know that:

- **LDA** means load value of the variable (in this case, **First**) into the accumulator
- **ADD** means add value of variable (in this case, **Second**) to the value stored in the accumulator
- **STO** replace the value of the variable (in this case, **Sum**) by the value stored in the accumulator

4.2.3 Translators

Computer programs can exist in several forms. Programs are written by humans in a form that people who are trained as computer programmers can understand.



▲ **Figure 4.15** Translation

A program must be translated into binary before a computer can use it; this is done by a utility program called a **translator**.

Compilers

A compiler is a computer program that translates an entire program written in a high level language (HLL) into machine code all in one go so that it can be directly used by a computer to perform a required task.

The high-level program statement:

```
Sum := FirstNumber + SecondNumber
```

becomes the following machine code instructions when translated:

```
0001          00010010
0100          00010011
0000          00011010
```

Interpreters

An **interpreter** is a computer program that reads a statement from a program written in a high-level language, translates it, performs the action specified and then does the same with the next statement and so on.

Assemblers

An assembler is a computer program that translates a program written in an assembly language into machine code so that it can be directly used by a computer to perform a required task.

▼ **Table 4.3** Translation programs summary

Compiler	Interpreter	Assembler
Translates a high-level language program into machine code.	Executes a high-level language program one statement at a time.	Translates a low level assembly language program into machine code.
An executable file of machine code is produced.	No executable file of machine code is produced.	An executable file of machine code is produced.
One high-level language statement can be translated into several machine code instructions.	One high-level language program statement may require several machine code instructions to be executed.	One low-level language statement is usually translated into one machine code instruction.
Compiled programs are run without the compiler.	Interpreted programs cannot be run without the interpreter.	Assembled programs are used without the assembler.
A compiled program is usually distributed for general use.	An interpreter is often used when a program is being developed.	An assembled program is usually distributed for general use.

4.2.4 Advantages and disadvantages of compilers and interpreters

▼ **Table 4.4** Comparing translators

Translators	Advantages	Disadvantages
Interpreter	easier and quicker to debug and test programs during development easier to edit programs during development	programs cannot be run without the interpreter programs can take longer to execute
Compiler	a compiled program can be stored ready for use a compiled program can be executed without the compiler a compiled program takes up less space in memory when it is executed a compiled program is executed in a shorter time	it takes a longer time to write, test and debug programs during development

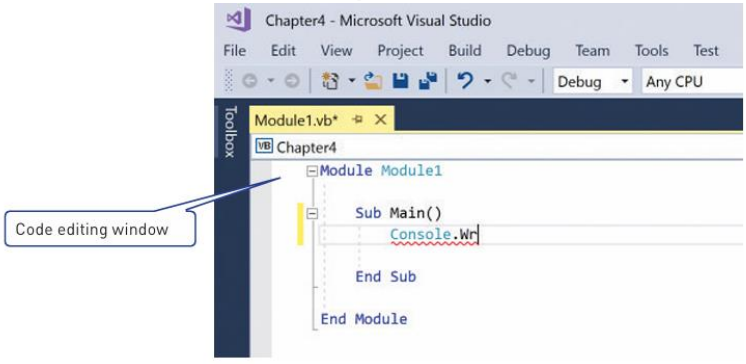
4.2.5 Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) is used by programmers to aid the writing and development of programs. There are many different IDEs available; some just support one programming language, others can be used for several different programming languages.

IDEs usually have these features:
» Code editors
» A translator
» A runtime environment with a debugger
» Error diagnostics
» Auto-completion
» Auto-correction
» An auto-documenter and prettyprinting.

Code editor

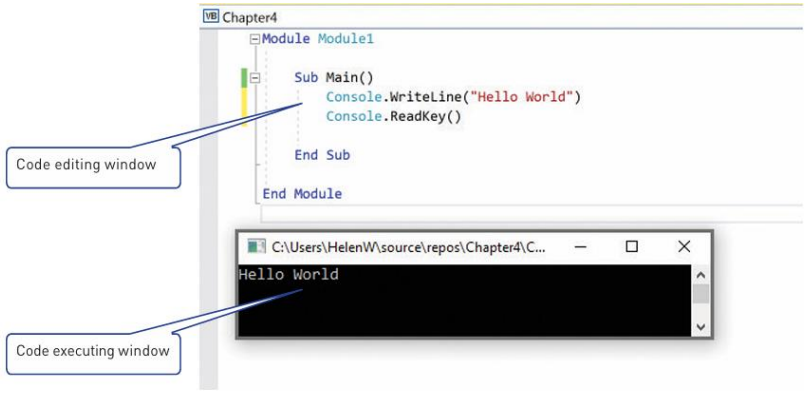
A code editor allows a program to be written and edited without the need to use a separate text editor. This speeds up the program development process, as editing can be done without changing to a different piece of software each time the program needs correcting or adding to.



▲ **Figure 4.16** Visual Studio code editor

Translator

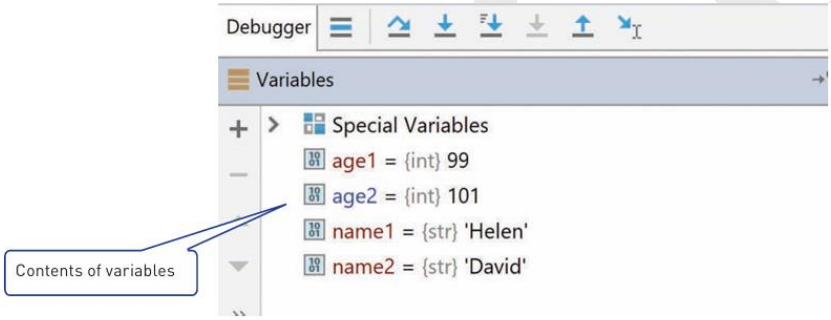
Most IDEs usually provide a translator, this can be a compiler and/or an interpreter, to enable the program to be executed. The interpreter is often used for developing the program and the compiler to produce the final version of the program to be used.



▲ **Figure 4.17** Visual Studio code editor and program running

A runtime environment with a debugger

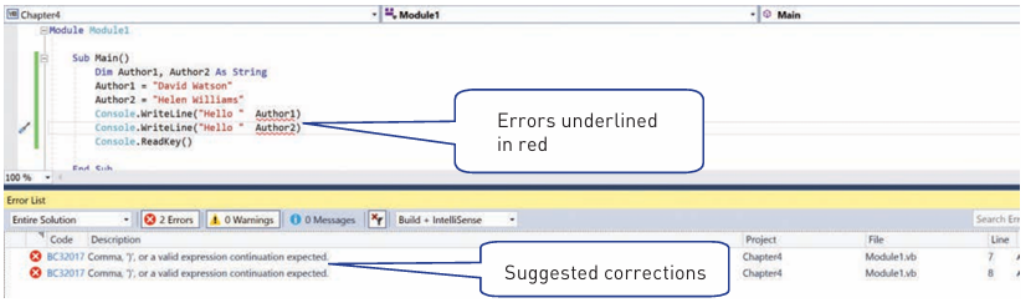
A debugger is a program that runs the program under development and allows the programmer to step through the program a line at a time (single stepping) or to set a breakpoint to stop the execution of the program at a certain point in the source code.



▲ **Figure 4.18** PyCharm debugger

Error diagnostics and auto-correction

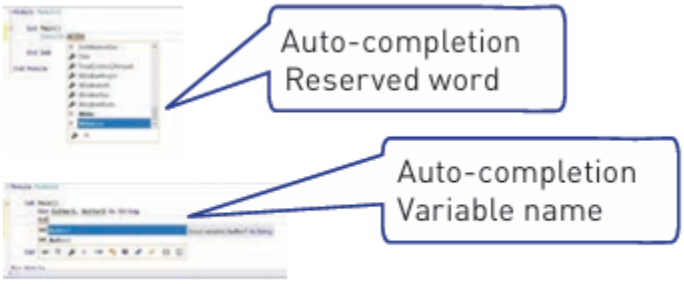
Dynamic error checking finds possible errors as the program code is being typed, alerts the programmer at the time and provides a suggested correction. Many errors can therefore be found and corrected during program writing and editing before the program is run.



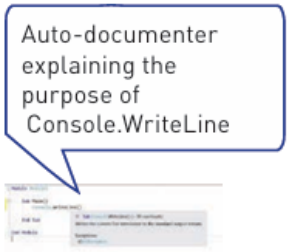
▲ **Figure 4.19** Visual Studio error list with suggested corrections

Auto-completion

Code editors can offer context-sensitive prompts with text completion for variable names and reserved words.



▲ **Figure 4.20** Visual Studio showing auto-completion



▲ **Figure 4.21** Visual Studio auto-documenter

Auto-documenter and pretty printing

Most code editors colour code the words in the program and lay out the program in a meaningful way – this is called **pretty printing.**

▲ **Figure 4.22** Visual Studio code editor showing prettyprinting

Key terms used throughout this chapter

utility programs (utilities) – part of an operating system which carries out certain functions such as virus checking, defragmentation and screensaver

malware – programs (such as viruses, worms and Trojan horses) that are installed on a user's computer with the aim of deleting, corrupting or manipulating data illegally

anti-virus software – software that quarantines and deletes files or programs infected by a computer virus; the software can run in the background or be initiated by the user

heuristic checking – checking software for behaviour that could indicate a possible virus

quarantine – to isolate (in order to later delete) a file or program identified by anti-virus software as being infected by a virus

defragmentation – a process that reorganises sectors on an HDD by rearranging blocks of data so that they are contiguous

contiguous – next to each other

back-up – make copies of files onto another storage media in case the original file becomes corrupted or is deleted

screensaver – software that supplies a still or moving image on a monitor if a computer has been inactive for a period of time

device driver – software that communicates with the operating system and translates data into a format understood by the device

descriptor – a collection of information about a device plugged into a USB port; this can be vendor ID (VID), product ID (PID) or serial number

operating system – software that provides an environment in which applications can run and also provides an interface between computer and human operator

boot up/bootstrap loader – a small program that is used to load other programs to correctly 'start-up' a computer system

EEPROM – stands for electronically erasable programmable ROM

human computer interface (HCI) – an interface supplied by the operating system to 'hide' the complexities of the software and hardware from the human user

command line interface (CLI) – an interface which allows communication with the computer by typing in commands using a keyboard

graphical user interface (GUI) – an interface that uses icons to represent apps and tasks which the user can select/launch by clicking on a mouse or using a touch screen

windows icons menu and pointing device (WIMP) – an interface that uses a pointing device such as a mouse to select options from screen icons or a menu

post-WIMP – a modern touch screen interface system that allows actions such as pinching and rotating

memory management – the part of an operating system that controls main memory

security management – the part of an operating system that ensures the integrity, confidentiality and availability of data

hardware management – the part of an operating system that controls all input and output devices; it is made up of sub-systems such as printer management

buffer – a memory area used to store data temporarily

file management – part of an operating system that manages files in a computer (for example, the ability to create, delete, copy, open, close and rename files)

interrupt – a signal sent from a device or software to a microprocessor requesting its attention; the microprocessor suspends all operations until the interrupt has been serviced

multitasking – a function that allows a computer to process more than one task/process at a time

administrator – a person responsible for the upkeep and maintenance of a computer system that involves multi-user function

user account – an agreement that allows an individual to use a computer; the user needs a user name and password to enter the user's area

error handling routine – a routine in a program or operating system that recognises and recovers a system from abnormal inputs or hardware faults (for example, recovery from an attempt to divide by zero)

firmware – a program that provides low level control for devices

interrupt priority – the priority assigned to an interrupt are given a priority so that the microprocessor knows which one needs to be serviced first and which interrupts are to be dealt with quickly

service (an interrupt) – when an interrupt is received, some action needs to be taken by the processor depending on what caused the interrupt; until this is resolved (that is, it is serviced), the interrupt cannot be removed to allow the processor to continue

interrupt service routine (ISR) – software that handles interrupt requests (for example, when the printer out of paper) and sends a request to the CPU for processing

machine code – a binary programming language, a program written in machine code can be loaded and executed without translation

high-level language (HLL) – a programming language that is independent of computer hardware, a program written in a HLL needs to be translated into machine code before it is executed.

low-level language (LLL) – a programming language that is dependent on computer hardware, both machine code and assembly language are LLLs

assembly language – a programming language that is dependent on computer hardware, a program written in an assembly language program needs to be translated into machine code before it is executed

assembler – a computer program that translates programming code written in assembly language into machine code

compiler – a computer program that translates a source program written in a high-level language to machine code

translator – converts a program written in a high-level language program into machine code

interpreter – a computer program that analyses and executes a program written in a high-level language line by line

Integrated Development Environment (IDE) – a suite of programs used to write and test a computer program written in a high-level language

debugging – finding errors in a computer program by running or tracing the program

prettyprinting – displaying source code using different colours and formatting, which make the code easier to read and understand

report window – a separate window in the runtime environment of an IDE that shows the contents of variables during the execution of a program

Revision questions

1. Complete the following by writing either compiler, interpreter or assembler in the spaces provided.
..... – translates source code into object code.
..... – translates low-level language into machine code.
..... – stops the execution of a program as soon as it encounters an error.

2. Motion sensors are used in a security system to detect intruders. Name three other sensors that could be used in the following applications. Give a different type of sensor for each application.

Application	Sensor
controlling street lights	
monitoring a river for pollution	
controlling traffic lights	

3. Draw a logic circuit corresponding to this logic statement:

X = 1 if (A is NOT 1) OR ((B is 1 OR C is 1) AND (B is NOT 1 OR A is NOT 1))

4. Three types of translators are assemblers, compilers and interpreters. X Tick (✓) the appropriate boxes to show which statements apply to each type of translator.

Statement	Assembler (✓)	Compiler (✓)	Interpreter (✓)
Translates high-level language into machine code			
Provides error diagnostics			
Translates whole program to object code in one operation			
Translates and executes one line of code at a time			

5. A Von Neumann model for a computer system has a central processing unit (CPU) that makes use of registers.
(a) Identify three registers that may be used.

b. The CPU is responsible for processing instructions. One stage of processing instructions is the decode stage.
(i) Identify the two other stages of processing instructions.

(ii) Identify the component of the CPU that is responsible for decoding instructions.

6. Both an interpreter and a compiler can be used when writing a program in a high-level language. (a) Explain why a programmer would make use of both an interpreter and a compiler.

(b) Give three reasons why a programmer would choose to write a program in a high-level language, instead of a low-level language.

7. (a) Give two reasons why a programmer would choose to write code in a low-level language

(b) High-level languages require either an interpreter or a compiler to translate the program. The table below lists a number of statements about language translators. Tick (3) to show which statements refer to interpreters and which refer to compilers.

Statements	Interpreter (✓)	Compiler (✓)
Translates the source code into machine code all at once		
Produces an executable file in machine code		
Executes a high-level language program one instruction at a time		
Once translated, the translator does not need to be present for the program to run		
An executable file is produced		

8. a). State four functions of an operating system

b. Six statements about assembly language are shown. Tick (3) whether the statement is true or false.

Statement	true (✓)	false (✓)
Assembly language uses mnemonic codes.		
Assembly language programs do not need a translator to be executed.		
Assembly language is a low-level programming language.		
Assembly language is specific to the computer hardware.		
Assembly language is machine code.		
Assembly language is often used to create drivers for hardware.		

9. (a) Many programmers write computer programs in high-level languages. The programs need to be translated into machine code to be read by the computer. State two types of translator that can be used.

(b) Explain two reasons why a computer programmer may choose to write a program in a high level language, rather than a low-level language.

(c) Three examples of computer code are given in the table. Tick (✓) to show whether each example of computer code is High-level language, Assembly language or Machine code.

Computer code	High-level language (✓)	Assembly language (✓)	Machine code (✓)
10110111 11001100 01011100			
FOR X = 1 TO 10 PRINT X NEXT X			
INP X STA X LDA Y			