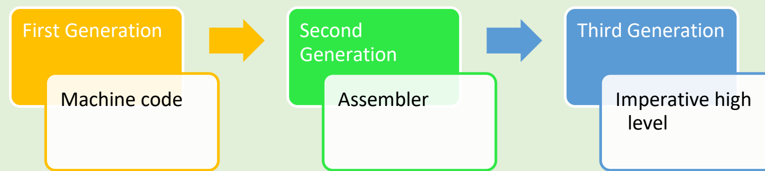
Programming Languages

Machine code, assembler and imperative high level programming languages are referred to as first, second and third generation programming languages respectively.



Imperative High Level Programming Language

- Imperative in this context means Command.
- High level programming language is closer to English language and is therefore easier to understand
- A translator is used to convert the instructions into code that the computer understand.
- High level languages allow programs to be written that is independent of the processor architecture so offer greater portability. It is up to the compiler to translate the code into the right machine code for an architecture
- There is a huge variety of high-level programming languages, and the choice depends on the specific application
- One instruction maps onto several machine code instructions
- Supports programming constructs such as sequencing, selection, iteration, assignment and functions for instance

Declarative high-level programming languages

- Declarative programming languages determine what is to be computed rather than how it is computed.
- Includes SQL, HTML, CSS, Functional programming

Low level programming language

- Low level programming languages refer to machine code and assembly language.
- The low level refers to a low level of abstraction.
- The low level language is close to the language understood by the computer where one operations maps to one instruction in the processor instruction set.
- Low level programming languages can control the hardware of the computer.
- Low level languages are not portable because they depend on the architecture of the processor as each processor has different instruction sets.
- Low level programming languages is difficult for humans to understand.
- Low level languages are appropriate for developing new operating systems, embedded systems and hardware device drivers.

Machine code

- Machine code is expressed in binary values 0 and 1. This is the language that computers understand. All codes whether assembler or high level programming languages need to be translated into machine code. Machine code is specific to a processor.
- Machine code instructions are made up of two parts the operator and the operand. The processor decodes the operator to identify the task that is to be carried out (eg. Add, load). The operand is the value or memory address that that instruction is to be operated on.

Machine code instruction	
Operator	Operand
0011	10010100

Assembly language

- Assembly language provides basic computer instructions for programs to run.
- There is a one-to-one relationship between machine code and assembly code instructions. One assembly language instruction maps to one machine code instruction, thus the structure of assembly language and machine code is the same
- Where machine codes uses 0 and 1 which are very difficult for programmer to understand, assembly language uses names (mnemonics) which is easier for the programmer.
- Each type of processor has its own instruction set and therefore its own assembly language and machine code. So Assembly code developed for one processor architecture will not run on another.

Assembly language sample Instruction set

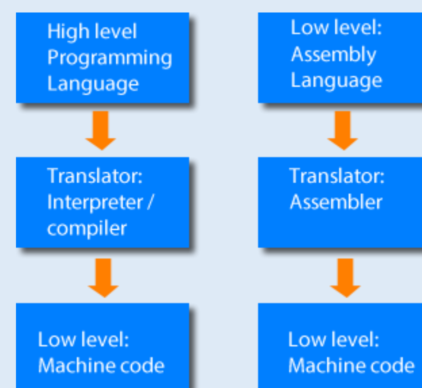
```
LOAD #23  # Load from RAM to processor
MOV a 23  # Transfer in number 23 into the variable a
ADD 2 3   # Add 2 values
STORE     # store data in RAM
```

Low level versus high level languages

	Advantages	Disadvantages
Low level	Produces code are better optimised and therefore are more efficient than high level languages because they are directly controlling the hardware Appropriate for developing new operating systems, embedded systems and hardware device drivers Some embedded systems do not have an interpreter or compiler so coding in a low level language in the only possibility.	Very difficult to understand and modify Assembly code is written for a specific processor architecture, and so is not portable to other computer architectures There is a one-to-one correspondence with the machine code and instruction so writing code is time consuming compared with writing code in a high-level language
High Level	Allow code to be written that is more portable. This means that the code can be run on different of the types of computer system with different processor architecture. Easier to understand and modify Take less time to write because a single instruction maps to several machine code instructions.	Needs a compiler or interpreter which is not always available particularly for embedded systems. Run slower because of the layers of abstraction and there is inefficiency in translator.

Program Translators

Program translators allow programs to be translated into machine code so the than programs can be run on a computer.



Interpreter

- Converts high level languages into machine code one instruction at a time on-the-fly while the program is running.
- Each instruction is converted to machine code once the previous instruction has been executed.
- Interpreters are good for debugging code because the program stops as soon as the error has been found.
- Running code this way is much slower running compiled code.
- The machine code is not saved.
- Both the source code and interpreter are needed for the code to be executed.

Compiler

- A compiler converts high level languages into machine code before the program is run.
- A compiler saves the machine code, so the source code is no longer needed.
- A compiler allows a program to be run faster that interpreted code.
- Software is normally distributed as compiled machine code. For proprietary software this is good because other people cannot copy the code and use it for their own applications.
- For compiled code to be executed only the machine code is needed a translator is not needed.
- Compiled code can only be run on the specific processor and OS on which it was compiled in the first place. Code using an interpreter can be run on any computer with any processor architecture, so long as the interpreter is available.
- There is a one-to-many correspondence between a high level language instruction and machine code instructions. That is one high level language instruction may represent multiple machine code instructions.

Assembler

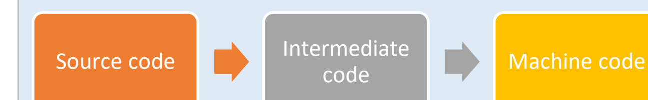
- Converts assembly language instructions into machine code.
- There is a one-to-one correspondence between a machine code instruction and an assembly code instruction.

Intermediate code / byte code

- Intermediate code is source code that has been translated into a standard form.
- This means that the source code does not need to be distributed.
- Intermediate code is not executable but will be converted into executable just before running.
- Intermediate code needs to run on a range of platforms so allows portability, but it does mean that each type of computer that it runs on needs to have a compiler developed for that architecture.

Source code and object code

- The source code it what is used to write the code normally in the high level language. On its own this code cannot be executed.
- The object code is the machine code that has been translated from the source code. This is code that can be executed.
- Software is normally distributed as compiled machine code.



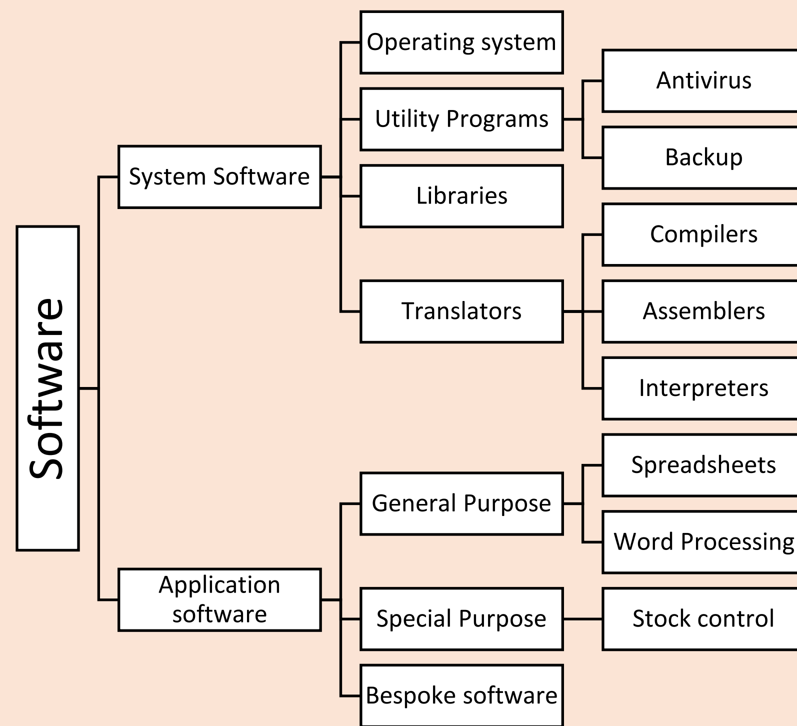
Software

A **computer system** has both hardware and software.

Hardware is the physical components that make up a device or computer system. These include both the internal components (eg motherboard, CPU, RAM) and also peripheral and networking devices such as printers and routers.

Software is the computer code, programs and algorithms that give instructions to the hardware to make it perform the desired task. Without the software the hardware will not get any instructions and it will not do anything.

Hierarchy of software



Application software

General purpose software - Software that is designed to be widely used in many ways for both business and personal use (eg applications such as word processing, presentation software, spreadsheet, and web browser).

Specialist Software – Software that is developed for a specific use or for a specific business, scientific, or educational area. For instance, air traffic control systems and stock control systems would fall under this category.

Bespoke software – The is tailor made software that is developed for a specific organisation or client. Bespoke software is expensive but meets the specific needs of an organisation.

System software

System software is concerned with the running of the computer. Its purpose is the control the computer hardware and manage the application software.

Program translators allow programs to be translated into machine code so that code can be run on a computer. Translators include interpreter, compiler and assembler.

Libraries are collections of prewritten code that can be used in software projects. Thee libraries significantly speed up the development process. Libraries can be reused across multiple applications.

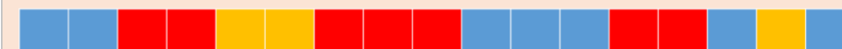
Utility programs are applications that help with the running of the machine. Common utility programs include:

Auto backup and restore - Incremental backup is useful because only files that have changed or been added since the last full backup needed to be backed up.

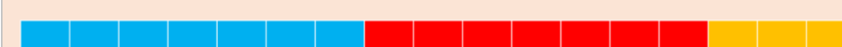
Anti-virus - Scans the computer to identify malicious code
Firewall Scans input and output packets and prevents malicious packets accessing the computer.

Disk defragmentation Organises files on a disk to be located contiguously. Often after defragmentation performance is improved because a file can be accessed from one location on a disk. Files can become fragmented when the original file increases in size and no longer fits into a contiguous location and has to be split over multiple locations.

Before defragmentation



After defragmentation



The role of the Operating System

- The most important piece of system software is the operating system.
- The operating system is system software with the role of managing the hardware and software resources.
- The OS handles management of the processor, memory, input/output devices, applications and security.
- The OS hides the complexity of the hardware from the user and provides a user interface.

Application management - Application software does not need to concern itself with interaction and complexities of managing the hardware because this is dealt with by the operating system. Application software needs to run on top of operating system which takes care of interaction with the hardware resources.

Processor resources – Allows multiple applications to be run simultaneously by manages the processing time between applications and cores and switching processing between applications very quickly. Multiple applications will access the processor resources via a schedule that alternates processing between applications. High priority applications will have more CPU time, but it means that lower priority applications will take longer to run.

Memory management – The OS distributes memory resources between programs and manages transfer of data and instruction code in and out of memory. Ensures that each application does not use excessive memory.

Input / Output devices – The OS controls interaction with input (eg keyboard) outputs (eg. Monitor) and storage (eg hard disk) using hardware drivers. Allows users to save files to the hard disk for instance.

Relation between application software, system software and computer hardware

Application Software

System Software

Computer Hardware