Program Translators

Machine code is the binary code that the CPU can process. It's 1s and 0s.

All computer programs need to be converted into **machine code** before they can be run by a computer.

This is true for any computer program – whether it's been written in a **high-level programming language** such as Python or a **low-level programming language** such as Assembly Language.

To do this, we use a **program translator**.

Three Types of Translator

You need to know the differences between the three types of translator and why each is used

There are three different types of translator you need to know about:

- assemblers
- compilers
- interpreters

These are used in different ways to convert code into machine code so that it can be run by the CPU.

1. Assemblers:

The key point here is that an assembler converts assembly language into machine code.

Assemblers convert **Assembly Language** into **Machine Code**.

One of the key things about assembly language code is that there is a 1:1 equivalence (or correspondence) between the code and machine code – so each line of assembly code directly translates into a single line of machine code. This is done using the assembler.

Each CPU type needs to have its own version of assembly language, although some assemblers can operate on different sorts of machine.

2. Compilers

Java and C# use compilers to create something called bytecode – which is an intermediate stage. This is then processed using an interpreter – so Java uses both a compiler and an interpreter (just to be confusing)

Compilers are one way of converting high-level program code into machine code. Examples of high-level languages that use compilers include C and C++.

Compilers translate the **whole program** before any of it is run. This takes more time to do – especially with a very long program. But the compilation creates a separate machine code file which can be saved as a **executable file** (an exe file). These run very quickly – because they are already translated – which is a significant advantage, especially when a program will be used frequently.

Compiled software is much easier for a user to run

It also means that:

- the end user doesn't need to have the compiler installed on their computer – just to be able to run executable files
- the end user can't see the high-level source code – so they can't change it, copy it, debug it or steal it. This helps software writers keep their copyrighted software from being stolen

A problem with exe files

Executable files can be used as a way of distributing malware such as viruses or Trojans. Because the user can't see inside the exe file they have no idea what it contains.

You also need different exe files for different types of machine – what runs on a Mac won't run on a Windows system, and what runs on a modern system might not run on an old system – and vice versa.

This links to unit 6 – cyber security

3. Interpreters

Interpreters also convert high-level programs into machine code, but do it in a different way. Examples of high-level languages which use interpreters include Python, Perl and Ruby.

JavaScript, when it's used in webpages, is interpreted by the user's web browser when the page is loaded. This runs the code and does whatever the JavaScript is set to do

An interpreter works through the high-level program code **line by line**, converting it as the program runs. So each instruction is translated into machine code as it is actually processed by the CPU.

This means it takes longer to run the program. This is particularly true if the program includes long loops with many iterations – the contents of the loop are translated each time the loop is worked through.

The differences between an interpreter and a compiler are important to know.

It also means that sections of code which are never run aren't translated. So if a user has a choice and goes down one route, the other part of the program will never be translated – which is more efficient, but can hide errors.

It also means that:

- users must have the interpreter installed on their machine – which is time and hassle and may be beyond some users
- users can see the source code – so they can change it, copy it, debug it and steal it
- developing programs might be easier using interpreted languages – you can find errors and debug more easily
- interpreted programs can be run on multiple platforms much more easily – so long as the interpreter is available

Interpreted languages are particularly good for small programs which might need to be run quickly – for example, the JavaScript on the webpage you downloaded this file from.

For example, IDLE for Python is available for Windows, Mac and Linux. This means it's easy to run the same Python program on any platform –so long as you have IDLE

Activities:

a) What is the key job of a program translator? Why is this necessary?
b) Name the three types of program translator
c) Describe what an assembler does
d) What is the key difference between a compiler and an interpreter?
e) Explain the advantages and disadvantages of compilers and interpreters