AQA Computer Science A-Level

4.6.2 Classification of programming languages

Advanced Notes

Specification:

4.6.2.1 Classification of programming languages:

Show awareness of the development of types of programming languages and their classification into low-and high-level languages.

Know that low-level languages are considered to be:

- machine-code
- assembly language

Know that high-level languages include imperative high-level language.

Describe machine-code language and assembly language.

Understand the advantages and disadvantages of machine-code and assembly language programming compared with high-level language programming.

Explain the term ‘imperative high-level language’ and its relationship to low-level languages.

The development of types of programming languages

The limited speed and memory of early computers forced programmers to write programs using low-level languages. These languages directly manipulated the processor, required a great deal of effort on the part of the programmer and were prone to errors.

High-level languages were developed to allow for instructions to be communicated to a computer's processor, making the job of programming far easier.

Low-level languages

The earliest electronic computers could only be programmed with low-level languages. Programs written in low-level languages are specific to the type of processor they are written for and directly affect the computer's processor.

There are two categories of low-level language: machine code and assembly language.

Machine code

Machine code uses only the binary digits 1 and 0 to represent instructions. This makes programs written in machine code very long and extremely difficult for humans to understand. Because of this, machine code programs are prone to errors and difficult to debug.

```
01010101
11010110
01001011
10110110
```

Because machine code directly manipulates a computer's processor, it is a very powerful paradigm. Programmers are not constrained when using machine code. Furthermore, there is no need to translate machine code before executing it, making the paradigm useful for embedded systems and real-time applications where speed of execution is paramount.

Assembly language

Assembly language was developed with the intention of simplifying the process of writing computer programs. Mnemonics, such as `ADD` and `MOV`, are used in place of the binary instructions that machine code uses. This makes assembly language more compact and less error prone than machine code.

```
STR R4, #45
ADD R1, R2, 3
MOV R2, R1
HALT
```

Note

This example uses AQA's own assembly language instruction set.

Each assembly language instruction has a 1-to-1 correlation to a machine code instruction. For example, the assembly language instruction `MOV R2, R1` might be the exact equivalent of the machine code instruction `11011101`.

1	1	0	1	1	0	0	1
Four-bit identifier for the MOVE command				Two bits representing the number of a register. In this case, 2.		Two bits representing the number of a register. In this case, 1.	
MOV				R2		R1	

AQA have made their own assembly language for use in exams. You need to make sure you're familiar with using it to understand programs and even to write your own.

Synoptic Link

AQA's assembly language instruction set is covered in **structure and role of the processor and its components** under **computer organisation and architecture**.

High-level languages

High-level languages are the type of programming language that you're most likely used to using. Examples of high-level languages include C#, Java, Pascal, Python and VB.Net.

Unlike low-level languages, high-level languages are not platform specific. However, high-level languages must be translated into machine code by a compiler or an interpreter before they can be executed.

Synoptic Link

Different types of **translators** are covered in the notes for **types of program translator**.

High-level languages don't use binary digits or mnemonics but use English instructions like "While" and mathematical symbols like "+". This makes high-level languages much easier for humans to learn and understand as well as making them easier to debug.

Most high-level languages allow programmers to make use of built-in functions. These can save vast amounts of time when programming.

Features such as named variables, indentation and commenting make programs written in high-level languages far easier to debug than those written in low-level languages.

```
While x < y
    x = x + y
End While
```

Synoptic Link

Variables, constants and subroutines can be assigned **names** in high-level languages.

The importance of **meaningful identifier names** is explained in **programming concepts** under **fundamentals of programming**.

High-level languages include imperative high-level languages. In a similar way to low-level languages, imperative high-level languages are formed from instructions that specify how the computer should complete a task, in contrast to declarative programming which describes what a computer should do.

High-level languages vs. low-level languages

	Low-level		High-level
	Machine code	Assembly language	
Portability	Not portable. Programs are processor specific.		Portable. Programs are not specific to certain processors.
Ease of use	Code is difficult for humans to understand.	Mnemonics help to make code slightly easier for humans to understand.	Code uses English, making it easy for humans to understand.
Ease of debugging	Errors are very difficult to spot and correct.	Debugging is easier than with machine code but still far more difficult than with high-level languages.	Named variables, indentation and commenting make debugging fairly easy.
Ease of execution	Machine code is directly executed by processors.	An assembler must be used before a program is executed, but each instruction has a 1-to-1 correlation to a machine code instruction so translation is quick.	A compiler or interpreter must be used to translate source code into object code before it can be executed. This can be time consuming.