Programming Languages

Programming languages are divided into generations, as follows:

3rd Generation

These are called 'high level languages'. They are the programming languages that most closely resemble English. Python is one example of a 3rd generation language. Others include JavaScript, C++, Visual Basic and Pascal.

3rd Generation languages can be identified by the fact they use human words for commands, such as 'Print', 'While' and 'If'. There are many different 3rd Generation languages. Programming them is easier than programming lower level languages. Most modern software is developed using high-level languages, as they are easier to write and quicker to test. Programs written in this way are relatively easy to port from one machine to another, but need compiling differently for each specific hardware.

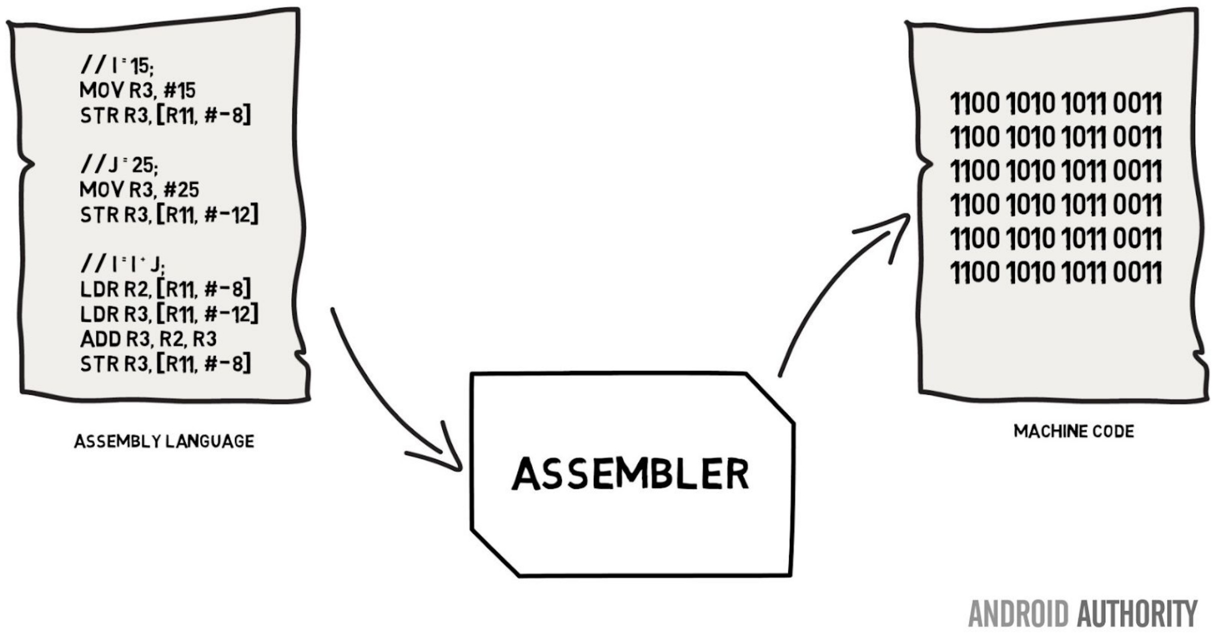
Question 1 - Why are 3rd generation languages easier to learn that low level languages?

Question 2 - Why are there so many different high level languages?

Question 3 - Why are 3rd generation languages easier to test and debug than low level languages?

2nd Generation

The 2nd Generation of program languages are known as Assembly, which is considered a low level language. Each command in assembly corresponds directly to a machine code instruction understandable by the CPU. However, rather than having to type in binary commands, machine code is made up of three letter command words. These are called Mnemonics, as they are easier to remember than the machine code commands.



```
//I=15;
MOV R3, #15
STR R3, [R11, #-8]

//J=25;
MOV R3, #25
STR R3, [R11, #-12]

//I=I+J;
LDR R2, [R11, #-8]
LDR R3, [R11, #-12]
ADD R3, R2, R3
STR R3, [R11, #-8]
```

```
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
```

Question 4 - What does Mnemonic mean?

Mnemonics found in assembly might include instructions like ADD, SUB, MUL, DIV, as well as LDA (load to accumulator), STR (store into memory) and CMP (compare two numbers). Assembly has a very limited set of instructions. These instructions are specific to a particular CPU - you cannot transfer a program written for one processor architecture to another, as it won't work because the second processor will not recognise the commands.

Assembly language is often used to program embedded systems, as it gives very precise control of hardware and makes very small, efficient, software.

Question 5 - What is an embedded system?

Question 6 - Assembly language tends to be very efficient (it doesn't take up much storage space and runs with few instructions) - why does this make it suitable for embedded systems?

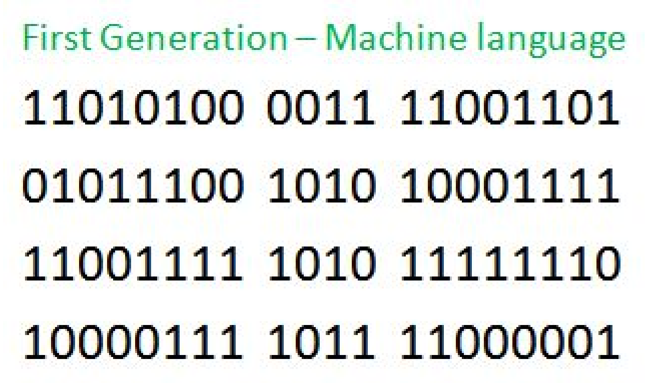
Assembly is also used for writing device drivers, and real time systems, where speed is essential.

Question 7 - What is a device driver?

Question 8 - What is a real time system? Give an example.

1st Generation

The 1st Generation programming language is called machine code. It is a way of giving instructions in pure binary - i.e. the language the processor actually uses. Typing in machine code accurately is very difficult to do, and therefore very rarely happens. Machine code is the lowest level language (i.e. it is the closest to the way the hardware actually works).



First Generation – Machine language

```
11010100 0011 11001101
01011100 1010 10001111
11001111 1010 11111110
10000111 1011 11000001
```

Question 9 - What do the ones and zeros in machine code physically represent when it comes to the hardware of the processor?

Question 10 - Why is it so hard to write in machine code?

Translators

We like to write code in 3rd generation languages, but processors only run machine code. Therefore, when a program has been written it has to be converted into machine code before it can run on a processor. This process is called Translation.

Diagram of Translators

There are three types of translator are:

Compilers

A compiler takes the whole of a program written in a high level language, and converts it into machine code. We say it converts the **Source Code** into **Object Code**.

Question 11 - What is Source Code?

Question 12 - What is Object Code?

Once the object code has been created it is saved, and can be run over and over again. You don't need to recompile the code unless changes have been made to the program. This saves time, as you don't have to recompile it each time the program runs. It also means that once you have completed a program you can compile it, and give the compiled, object code, to customers - they therefore can't see how the program was written, protecting your ideas and preventing them from easily making changes to your program.

One drawback of compiling code is that even if it encounters errors it will try to keep compiling and only report errors at the end. This means that the developer will only get error messages which they have to try to make sense of in order to try to find their errors. Also, if a change needs to be made to the program the developer must go back to the source code, they can't make change to the object code.

Interpreter

An interpreter translates code one line at a time - it translates a line and then runs it before translating the next line. It doesn't save the translated code, therefore the interpreter has to run each time that the code is run. This makes running code with an interpreter slow compared to running object code produced by a compiler.

The advantages of using an interpreter are that if an error is found the interpreter will stop right away, and report it to the developer - this makes finding where an error is much easier, and helps with debugging.

Also, source code is not platform specific - any computer with the right interpreter can run the code, as the interpreter will translate it for its own hardware. This is particularly useful for running code on websites - you don't know what platform someone is using to view your website (PC, Mac, iPhone, Android phone, etc.), so compiled code probably won't run. However, if a browser has an interpreter built in you can embed source code in your webpage and the interpreter will make it run on whichever machine is viewing the site.

Question 13 - Which commonly used programming language is designed to use an Interpreter that is built into most web browsers?

Question 14 - A developer might use both an Interpreter and a Compiler at different stages of developing a new piece of software. At what stage might each be used, and why?

Assemblers

Assemblers translate assembly language into machine code. Assembly language is very similar to machine code, so this is a very direct translation, as each instruction in Assembly corresponds directly to an instruction in machine code.