Programming language translation

Assembler

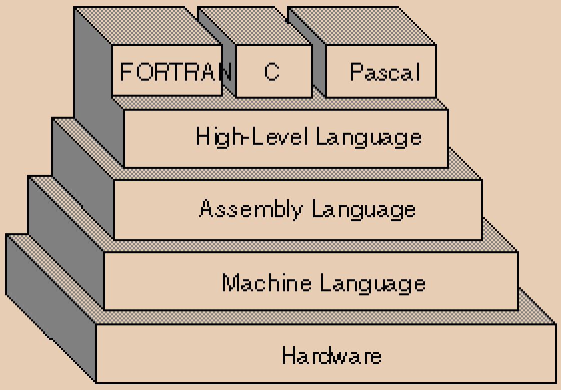
Assembly code is a low-level language, with each instruction in assembly code always being equivalent to one machine code instruction. There is a very strong relationship between the language and the machine code of a particular architecture.

The machine code instructions that a particular computer can execute are dependent on the hardware. They are called the instruction set.

Each CPU architecture has its own instruction set and therefore a unique assembly code.

Before assembly code can be executed, it must be translated into the equivalent machine code. This is done by a program called an assembler.

The assembler takes each assembly code instruction and converts it into the 0s and 1s of the corresponding machine code instruction. The input to the assembler is called the source code and the output (machine code) is called object code.



Compiler

A compiler is a program which translates high-level languages such as Visual Basic, Python, etc. into machine code.

The code written by the programmer, the source code, is input as data to the compiler. The compiler then scans through all the source code several times, each time performing different checks and building up tables of information needed to produce the final object code.

Different hardware platforms will require different compilers since the resulting object code will be hardware-specific. For example, Windows and Intel processors comprise one platform, Windows and AMD processors another, so separate compilers are required for each.

Object code can be saved once it is compiled so it can be run whenever needed, without the presence of the compiler, and of course more efficiently.

Input		Process		Output
Source code (e.g. Python program)	→	Compile	→	Object code (executable machine code)

Interpreter

Once a program has been written, and the source code saved, the programmer may call the program to run. An interpreter will look at each individual line of source code in sequence as it runs, analyse it and if it contains no syntax errors, translate it into machine code to be run.

It does not previously compile source code into machine code. Instead, each instruction is analysed and translated in real-time and the CPU executes each instruction before the interpreter moves on to translate the next instruction.

Java is a high-level programming language which is compiled to produce bytecode (an intermediate code) which is then interpreted by a virtual process machine (VM). Bytecode is code which is compiled and can then be interpreted.

Compilers vs Interpreters

Interpreters will show errors as soon as they reach an erroneous line of code. This makes debugging easier than when using a compiler.

The compilation is a slower process but once compiled, the machine/object code can be executed much quicker.

A problem with interpreters is that they will not attempt to run any further liens of code until a preceding erroneous lien of code has been amended. It may take many run through to identify all errors in a program.

A compile makes it difficult to test each individual line of code as bugs are reported after all code is compiled.

Bytecode

Interpreting each line of code before executing it has become much less common. Most interpreted languages such as Python and Java use an intermediate representation which combines compiling and interpreting.

Source code is compiled into bytecode which is then executed by a bytecode interpreter (interpreting it into machine code). The latter is done by a process virtual machine.

The bytecode may only be compiled once and for all (as in Java) or each time a change in the source code is detected before execution (as in python).

The main advantage of this intermediate bytecode is that it allows platform independence; any computer with a Java VM (virtual machine) can interpret bytecode into machine code for the particular machine code instruction set of the host.

The Java Virtual Machine masks inherent differences between computer architectures and operating systems.

A second feature of using Java bytecode is that it acts as an extra security layer between your computer and the program. You can download an untrusted program and then run the Java bytecode interpreter rather than the program itself, which guards against any malicious programs.

It is also possible to compile from python into Java bytecode (using the Jython compiler) and then use the Java interpreter to interpret and execute it.

Advantages of compilers over interpreters:

- ✓ A compiler has many advantages over an interpreter:
- ✓ Once compiled, the object code can be saved to the disk and run whenever required without the need for the compiler on the host machine.
- ✓ Object code executes faster than interpreting source code to object code.
- ✓ The object code produced by a compiler can be distributed or executed without having the compiler present.
- ✓ The object code is more secure, as it cannot be read without a great deal of `reverse engineering'. This means the source code is kept secure.
- ✓ Interpreters are very inefficient for loops in the source code. Every line in the loop will have to be interpreted x amount of times.

An interpreter has some advantages of a compiler

- ✓ Platform independence: The source code can be run on any machine which has the appropriate interpreter available (e.g. Java virtual machine).
- ✓ It is useful for program development as there is no need for lengthy recompilation each time an error is discovered.

A compiler is most appropriate for programs that are run regularly without frequent change. It is also appropriate when the object code produced by the compiler is going to be distributed or sold on to users outside the company. The source code is not present to can't be changed or amended.

Interpreters are best when programs need to be used across many platforms and contain simpler sequence and branching statements without iterations. They are also useful in program development for debugging and efficiency.