Computer Science 1510

Lecture 4

Lecture Outline

- Introduction to programming

From hardware to software

- The different types of memory (hard disk, RAM, and cache/registers) all have different roles when executing a program.

- A program that is currently in use, as well as all of the data that it is using, is stored in RAM.

- When a program is executed it is loaded from disk to RAM. There are now two copies of the program, an inactive copy on hard disk, and an active copy in RAM.

- If the results from a program need to be saved, they must be copied (written) to hard disk.

- Cache and registers are used to store frequently used portions of a program or its data, since this is the fastest type of memory.

Ordered instruction execution

- A program is a sequence of instructions that must be followed to solve a particular problem.

- Recall that the CPU uses a Fetch-Decode-Execute cycle, where it obtains one instruction, performs the instructed task, and repeats the cycle for the next instruction. [1]

- Structured algorithms/programs use three methods of control:

  1. Sequential - Steps are performed in order, beginning to end (this is the default control mechanism).
  2. Selection - One of a number of alternative actions is selected and executed.
  3. Repetition - One or more steps are performed repeatedly.

[1] In modern day computers it is possible to have more than one instruction executed at the same time.

Machine language

- The CPU and ROM have built-in commands that allow execution of simple instructions. These commands are in *machine language*.

- Machine language is in the form of binary instructions containing an operation code (*opcode*) and an *operand* on which the given operation is performed.

- Example: To compute $A * B + C$, where A is stored in memory address 1024, B is stored in memory address 1025, and C is stored in memory address 1026, the following steps are required:

  1. Fetch the contents of memory location 1024 (A) and load it into a CPU register.
  2. Fetch the contents of memory location 1025 (B) and compute the product of this value and the value in the register.
  3. Fetch the contents of memory location 1026 (C) and add this value to the value in the register.
  4. Store the contents of the register in memory (in location 1027 for example).

- If the opcodes for load, store, add, and multiply are 16, 17, 35, and 36 respectively, then the machine language translation of the above could be:

  1. 00010000 00000000000010000000000
  2. 00100100 00000000000010000000001
  3. 00100011 00000000000010000000010
  4. 00010001 00000000000010000000011

- Early computers required that programs be written in machine language.

- Later, it became possible to write programs in *assembly language*.

- Assembly language uses names in place of numeric codes.

- The *assembler* is a program that translates assembly language into machine language.

- For example, in assembly language, `ADD X,Y` adds X and Y and stores the result in X.

Programming languages

- A programming language is the language used by a programmer to instruct a computer to perform certain tasks.

- Today, programs are written in high-level languages (e.g., Fortran, C, Java, Python) which allow programmers to write programs in a more user-readable form.

- Each programming language consists of a unique *syntax* – a vocabulary and a grammar, to which all programs must adhere.

- Computers are logical, not intelligent! The programmer must convey to the computer exactly what needs to be done and how to do it.

- Small changes in syntax can produce completely different results!

Compilers

- Programs written in high-level languages such as Fortran must be translated into machine/assembly language in order to be executed on a computer.

- For languages like Fortran and C, this is performed using a compiler.

- Compilers are specific to the given programming language, and also to a given architecture and operating system.

- A compiler is itself a program that performs the following tasks:
  - Reads the contents of a text file containing a program (the *source code*).
  - Checks to ensure that all syntax rules for the given language have been followed (if not the compiler produces an error).
  - Generates a binary version of the program (machine language).

Steps in the creation of a program

1. **Write the program**:
   Use a text editor to write the algorithm to solve the problem using some programming language.

2. **Compilation**:
   Use a compiler to convert the source code into machine language. The compiler performs the following line-by-line:

   - Check that the line has correct syntax.
   - If there are no syntax errors convert the line to assembly language and proceed to the next line.
   - If there are syntax errors, print an error message and stop compiling. The programmer must then go back and edit the source code to correct the error and begin the compilation process again.

3. **Object file**:
   When the source code is free of syntax errors the compiler converts the assembly code into an *object* file containing the machine code version of the source code.

Steps in the creation of a program

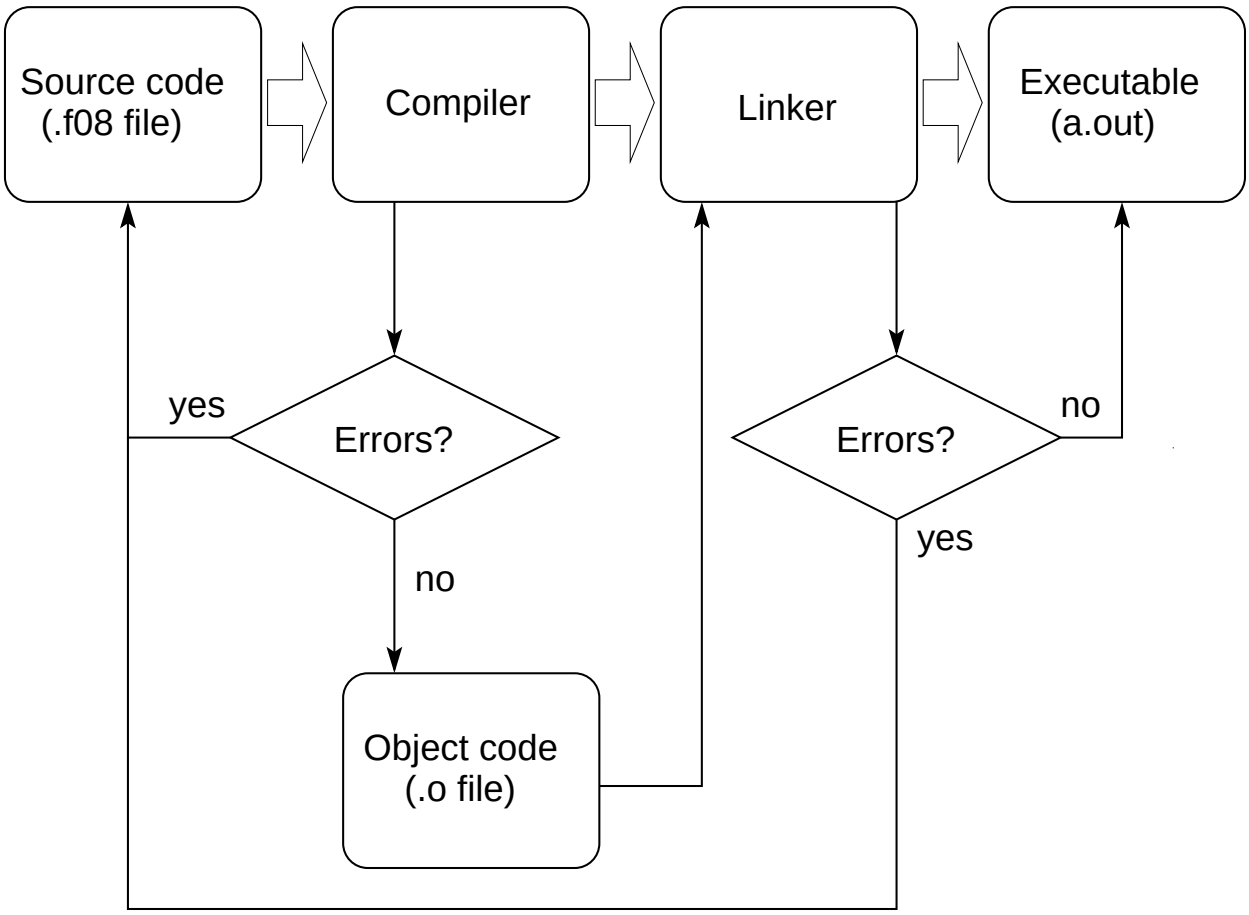
4. **Linking**:
   Following the creation of an object file, the compiler automatically sends this file to the *linker* which attaches special operating system run and load routines to the program, to make it capable of being executed.

5. **Executable file**:
   If there are no linking errors an *executable* file is created. In Linux, the default name of the executable is `a.out`. If there are linking errors then the programmer must go back and edit the source code to correct the error and begin the compilation process again.

6. **Running the program**:
   Once an executable file has been created, a user can *run* the program by typing the name of the executable. For example:

```
user@garfield[1] $ a.out
```

Compilation process (Fortran)



Architecture dependence

- In addition to different programming languages requiring different compilers, different computer architectures (i686, alpha, etc.) also require different compilers.

- Once a program has been compiled for a given architecture and operating system, then the resulting executable file can ONLY be run on that particular architecture and operating system.

- Each architecture has its own version of machine language.

- During the compilation process the source code is compiled for the particular architecture and linked for the particular operating system.

- Source code written in a standard language can be compiled on any computer that has a compiler for the language used.

Errors!

- When you first begin writing programs you will most likely receive a long list of errors when you try to compile your code (often dozens!).

- Don't be discouraged! Many of the errors that are reported are often due to earlier errors.

- The best approach is to first fix a few errors (even just 1 or 2) and try recompiling. You will likely find that the number of errors drops dramatically.

- Errors in source code are usually referred to as *bugs*, and the process of removing errors from your code is called *debugging*.

Types of programming errors

- **Logic errors**
  - No error messages are printed by the compiler.
  - An executable is created.
  - An incorrect solution is produced.

- **Syntax errors**
  - The rules of the programming language have not been followed.
  - An error is displayed during compile-time
  - The compiler prints error messages describing the error(s) and indicating the location(s) of the error(s).
  - No object file or executable is created.
  - Understanding the terminology used in error messages requires practice!!

Types of programming errors

- **Linker errors**
  - Error messages are displayed during linking.
  - No executable is created.
  - Can be caused by trying to use a library that doesn't exist, or incorrectly specifying the location of a file.
  - Since the linker uses the object file and not the source file, linker errors do not indicate where in the source code the error occurred.

- **Run-time errors**
  - No error messages are displayed during compilation.
  - An executable is created.
  - An error is displayed during execution (run-time).
  - Example: division by zero.