Algorithm Fundamentals

Student: Gabriel Darlei Lopez Franco

Teacher: Luis Gerardo Camara Salinas

Unit 1

Stages of Program Compilation

Compilation Stages

There are 4 stages of the compilation, preprocessor, compiler, assembler and linker. Each one of these perform different functions needed to get the executable code. First the preprocessor expands the code then, the compiler assembles it, afterwards the assembler make the object code, finally the linker finish the job dropping the executable file.

Preprocessor

The code is rewritten in a content tool. The source code is first passed to the preprocessor, which then executes this code. After separating the code, pass the extension code to the compiler.

Some of the functions this stage does is, for example deleting the comments on the code, extending the macros and some file input from the libraries.   
  
The main stage of aggregation is known as preprocessing. At this stage, lines starting with the # character are decoded by the preprocessor into preprocessor commands. These commands constitute the basic diatonic language with possessive syntax and semantics. The language is used to reduce source code cycles by including documentation, characterizing macros, and providing the ability to restrict code skipping.  
  
The preprocessor performs initial processing before interpreting the command. This includes concatenating continuation lines and removing comments from the source program.

Compiler

This stage has a tricky name because the compiler stage is not exactly the compilation of the source code, instead, the preprocessed code is translated into assembly instructions specific to the target processor architecture. These form an intermediate human-readable language.

The existence of this step allows the code to contain inline assembly instructions and allows the use of a different assembler.  
  
Some compilers also support the use of an integrated assembler, which directly generates machine code in the compilation phase, avoiding the overhead of generating intermediate assembly instructions and calling the assembler.

Assembler

The main function of this part is to translate the assembly instructions into an object code it also assigns the machine delivers to emblematic names. The output consists of actual instructions to be run by the target processor. Since each low-level computational build is processor-specific, the collection of programs is accomplished through direct balanced planning from collection code to machine code. Once again, the compiler must convert extensive high-level source code into processor-explicit machine code.

Linker

The linker is the final step in which the object code generated during the assembly phase that consists of machine instructions that the processor understands, is converted in an executable code. To generate this executable program, existing parts must be rearranged and missing parts filled. This process is called linking.  
  
The linker will arrange object code fragments so that functions in some fragments can successfully call functions in other fragments. It will also add snippets containing library function instructions used by the program. In the case of a "hello world!" program, the linker will add object code for the puts function.

Levels of Programming

Microcode

Microcode is the first specified level of programming, processor and machine instruction sets. It is a combination of small instruction that sets derived from machine language. The microcode performs short control-level register operations, that consists of multiple microinstructions, each microinstruction performing one or more micro-operations.  
  
Microcode is often confused with the machine language but they are different . Machine language runs on top of hardware abstraction. However, microcode handles lower-level or circuit-based operations. Since microcode is usually embedded in hardware, it cannot be changed.

Low Level

Also known as high level languages its principal characteristic is that they are easily interpreted by humans. These languages have a lot of improves over assembly like looping constructs that helps in repetitive task or for some special features, procedures, functions (for structural programming), some typing – the trappings of modern programming languages.

High Level Programs

These are very convenient, but also very far removed from the computer they are running on. Some of they features are:

Type checking

Easier to debug

You may never even see a memory address. As a result, they typically aren’t as efficient.

They still may not be portable: implementation dependence.

The major goal of the low-level language is to utilize high-level languages like PHP, C#, and Swift to build software programmes and scripts that operate, manage, and modify the computer's hardware and instructions set architecture. The source code can be created and edited by a software developer in a high-level language with the help of using a basic text editor or a programming IDE.

References:

PC Mag Staff (28 February 2019). “Encyclopedia: Definition of Compiler”. PCMag.com. Retrieved 28 February 2017.

1. Fraser, Christopher W.; Hanson, David R. (April 1982). “A Machine Independent Linker”. Software: Practice and Experience. John Wiley & Sons Ltd.

*The four stages of compilation*. (2020, September 27). Techopedia. Retrieved January 12, 2021, from https://teachcomputerscience.com/the-four-stages-of-compilation/

*What is a Low-Level Language?* (2020, August 31). Java T Point. Retrieved January 12, 2021, from https://www.javatpoint.com/what-is-a-low-level-language