Alex O’Connell

Dr. Arifuzzaman

CSCI 2450 Fall 2019

December 10, 2019

Application of Assembly Languages

Assembly language is any low-level programming language in which there is a strong relation between the instructions in the language and the architecture of the machine code instructions. Assembly makes it easy for a programmer to have direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues. Assembly language may also be referred to as symbolic machine code.

Assembly code is converted into executable machine code by a utility program called an assembler. This process refers to assembly, as in assembling the source code. Assembly language usually has one statement per machine instruction, but comments and statements that are assembler directives, macros, and symbolic labels of program and memory locations are often also supported.

Assembly language was first developed in the 1950’s when they were referred to as second generation programming languages. This eliminated much of the error prone and time-consuming first-generation programming needed with the earliest computers. This freed the programmer from having to remember numeric codes and calculating addresses. By the 1980s, the use of assembly had fallen and had been taken over by high level languages in search for improved programming productivity.

Each assembly language is specific to a particular computer architecture and sometimes to an operating system. However, some assembly languages do not provide specific syntax for operating system calls, and most assembly languages can be used universally with any operating system, as the language provides access to all the real capabilities of the processor, upon which all system call mechanisms ultimately rest. In contrast to assembly languages, most high-level programming languages are generally portable across multiple architectures but require interrupting or compiling, a much more complicated task than assembling. Unlike high level languages, assembly is not portable.

Assembly language uses mnemonic to represent low level machine instruction. Many operations require one or more operands in order to form a complete instruction. Most assemblers permit named constants, registers, and labels for program and memory locations, and can calculate expressions for operands. Thus, the programmers are freed from tedious repetitive calculations and assembler programs are much more readable than machine code. Depending on the architecture, these elements may also be combined for specific instructions or addressing modes using offsets or other data as well as fixed addresses. Many assemblers offer additional mechanisms to facilitate program development, to control the assembly process, as well as aid with debugging.

Data instructions are instructions used to define data elements to hold data and variables. They define the type of data, the length, and the alignment of data. These instructions can also define whether the data is available to outside programs or only to the program in which the data section is defined. Assembly directives are commands given to an assembler.

There are many different typical applications of assembly. Some different ones being:

* assembly language is typically used in a system boot code, the low-level code that initializes and tests the system hardware prior to booting the operating system and is often stored in ROM.
* Some compilers translate high level languages into assembly first before fully compiling, allowing the assembly code to be viewed for debugging and optimization purposes.
* Assembly language is useful in reverse engineering. Many programs are distributed only in machine code which is straightforward to translate into assembly, but more difficult to translate into a higher-level language. Different tools to disassemble the code can be used to put it back together which gives the programmer a chance to write the code in a high-level language.
* Assembly language is used to enhance speed of execution, especially in early Personal Computers with limited processing power and RAM.

Assembly is not useful for business application software, written for single platform, medium to large size. This is because there is limited formal structure so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code. Business application written for multiple platforms is also another example where assembly is probably not the right language to use. The code would have to be recoded separately for each platform, often using an assembler with a different syntax, which creates difficulties maintaining. Since a computer’s behavior is fundamentally defined but its instruction set, the logical way to learn such concepts is to study an assembly language.

Assembly language is as close to the processor as you can get as a programmer, so a well-designed algorithm is ideal. Assembly is great for speed optimization. Assembly language gives you complete control over the system’s resources. Much like an assembly line, you write code to push single values into registers, deal with memory addresses directly to retrieve values or pointers. These are the different applications of assembly languages.