# 44. IMPORTANCE OF ASSEMBLY LANGUAGE FOR PROGRAMMERS

*Urbanovich A.A.*

*Belarusian State University of Informatics and Radioelectronics*
*Minsk, Republic of Belarus*

*Subbotkina I.G. – Associate Professor*

A short introduction to CPU processes is given in this paper. The description of Assembly work principles, which are based on CPU processes, is presented. Based on Assembly work principles advantages and disadvantages of the language are defined. Taking them into consideration the importance of Assembler for programmers is determined.

Assembly language, commonly referred to as "assembler," is a low-level programming language that directly interacts with computer hardware. It emerged in the 1940s and was the primary tool for software

development until higher-level programming languages were introduced. But even now, when programmers have such advanced tools as C#, Java, Python, and etc., Assemblers continue to be utilized as "bridges" between high-level language operators and the sequence of binary digits[1]. Figure 1 demonstrates the conversion of high-level language operations into binary digits.

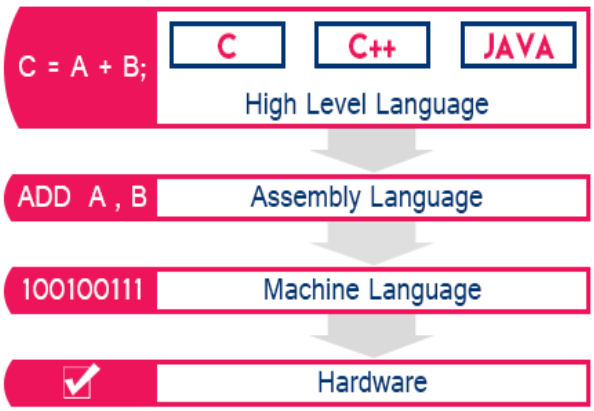


Figure 1 – Example of code translation

In order to gain a comprehensive understanding of assembler work, it is imperative to delve into the intricate internal processes of the central processing unit (CPU). CPU is only a little piece of silicon with billions of transistors, and that leads to the question “How does it work with numbers?”. In order to address this inquiry, an examination of numerical systems is warranted. The primary criterion for the selection of a numerical system in digital devices is its minimal number of values, as it facilitates the production and enhances the resilience of components that operate with these values against interference.

The binary number system is the most suitable option to work with because it operates with only two numbers: 1 and 0. These digits can be easily represented by various physical phenomena such as electricity (1 for power on and 0 for power off), magnetic field (1 for the presence of a magnetic field and 0 for the absence of a magnetic field), and so on. Hence each CPU instruction is technically implemented in the form of a sequence of electrical signals of varying voltage.

A computer processor is made up of several components, including the control unit, arithmetic logic unit (ALU), registers, cache memory, memory management unit (MMU), bus interface unit (BIU), clock generator, input/output (I/O) controller, interrupt controller, and floating-point unit (FPU).

The control unit manages the flow of data within the processor. As an input, CU fetches instruction or program command, which enters the command register. Once the instruction is fetched, the control unit decodes it to determine what operation needs to be performed. The decoding process involves breaking down the instruction into its component parts, such as the opcode (which specifies the operation) and the operands (which specify the data on which the operation will be performed). When the instruction has been decoded, the control unit sends signals to the appropriate components to perform the operation. For example, if the instruction is an arithmetic operation, the control unit sends signals to the arithmetic logic unit (ALU) to perform the calculation. Once the operation is completed, the result is stored in a register called the accumulator. The control unit then sends signals to store the result in memory or send it to another component for further processing [2]. Schematic representation of CPU processes is shown in figure 2.

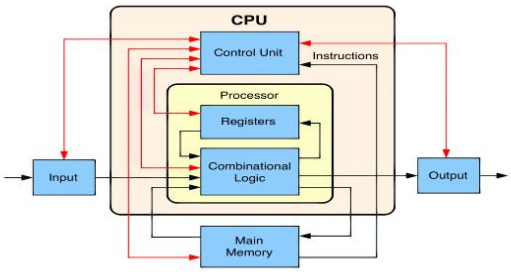


Figure 2 – Schematic representation of CPU internal processes

Assembly language works providing a set of instructions that are executed by the CPU based on the principles described above. These instructions are written in a form that is easily understood by the

CPU, allowing it to execute them quickly and efficiently. Each instruction performs a specific task, such as moving data from one location to another or performing arithmetic operations.

Assembly language programs are written using mnemonic codes that represent the individual instructions. Mnemonic can operate with CPU's registers, memory cells, numeric constants, labels and so on. These codes are then assembled into machine code, which is the actual code being executed by the CPU. The assembly process involves translating the mnemonic codes into their corresponding machine code instructions, which are then loaded into memory for execution. There is no specific syntax for Assembly language as there exist different processor's architectures, thus each Assembler variation has its own set of mnemonics, although they generate the same numeric machine code.

One of the most significant advantages of assembly language is its ability to access and manipulate hardware resources directly. This means that programmers can write code that is highly optimized for specific hardware platforms, resulting in faster and more efficient code. Assembly language also allows programmers to write code that is smaller and more compact than the code written in high-level languages like Java or Python.

Another important benefit of assembly language is its ability to provide greater control over the code execution. With assembly language, programmers can control the exact sequence of instructions that are executed by the CPU to adjust their programs for maximum performance. Assembly language also provides a greater degree of precision and accuracy than high-level languages, making it ideal for tasks that require exact calculations or precise control over hardware resources. It plays a vital role in the field of reverse engineering as well. By analyzing the assembly code of an application, developers can gain insights into how it works and identify potential vulnerabilities or security flaws [3].

One of the main disadvantages of Assembler is that programmers must be familiar with the specific instruction set architecture (ISA) of the hardware they are programming for. Different processors have different ISAs, and each ISA has its own set of instructions that can be used to perform specific tasks. As such, programmers must be able to read and understand technical documentation to determine which instructions are available and how they can be used to achieve specific goals. In addition to technical knowledge, programming in assembly language also requires a high level of attention to detail and a willingness to work with complex code. Assembly language code is often much more verbose than higher-level languages like Python or Java, and requires a deep understanding of how each instruction works and how it affects program execution. One more aspect cannot be overlooked: Assembly language has no data type control, therefore a programmer himself must determine the meaning of the value entered into memory: whether it is a number or a lowercase character, and the permissible operations on this value. Finally, programming in assembly language requires a lot of patience and persistence. Debugging code at this low level can be incredibly challenging, as errors are often difficult to detect and diagnose. Programmers must be willing to spend hours or even days working through complex code to identify and fix errors.

In conclusion, while assembly language may not be as widely used as it once was, it still holds immense importance in the world of computing. Its efficiency, low-level control, and ability to access hardware directly make it an ideal choice for certain types of applications. Additionally, learning assembly language can provide valuable insights into computers work and improve a programmer's overall understanding of software development.

References:

1. Assembly Language [Electronic resource]. – Mode of Access: https://www.investopedia.com/terms/a/assembly-language.asp – Date of Access: 03.03.23.

2. Почему Ассемблер – это круто, но сложно [Electronic resource]. – Mode of Access: https://thecode.media/assembler/ – Date of Access: 06.03.23.

3. Что такое ассемблер и нужно ли его изучать [Electronic resource]. – Mode of Access: https://skillbox.ru/media/code/chto-takoe-assembler/ – Date of Access: 09.03.23.