**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

Laboratory work 4:

Introduction in NASM

Elaborat:

st. gr. FAF-221 Berco Andrei

Verificat:

asist. Univ. Voitcovschi Vladislav

Chişinău - 2024

# **Purpose of the work:**

1. Familiarize yourself with the basics of Assembly Language: Before diving into NASM, it's important to understand the fundamentals of Assembly Language. Start by learning the basic concepts such as registers, memory, instructions, and the syntax used to write Assembly code.
2. Install NASM: The first step towards learning NASM is to install it on your machine. NASM is available for multiple platforms like Windows, Linux, and macOS. Download and install the version that is compatible with your system.
3. Write simple programs: Start by writing simple programs in NASM to get a feel for the language. Start with basic programs like printing messages on the screen, reading input from the user, and performing arithmetic operations. This will help you understand how NASM works and get comfortable with the syntax.
4. Debug your programs: Debugging is an essential part of programming, and NASM is no exception. Learn how to use debugging tools like GDB to identify and fix errors in your code. This will help you become more efficient in your programming and also give you a better understanding of how your code works.
5. Fundamentals of Assembly Language

Assembly Language serves as a fundamental layer of programming, allowing direct interaction with a computer's hardware. Unlike high-level programming languages, Assembly Language provides a close correspondence between the instructions written by the programmer and the operations executed by the CPU. This direct control over hardware makes it powerful for tasks requiring optimization, real-time processing, or low-level system programming.

* Registers:

Registers are small, high-speed storage locations within the CPU used for temporary data manipulation. They play a crucial role in Assembly Language programming by facilitating arithmetic, logical, and data transfer operations. For example, in x86 architecture:

MOV AX, 10 ; Move the value 10 into the AX register

ADD BX, AX ; Add the value of AX to the BX register

Here, the MOV instruction moves the value 10 into the AX register, and the ADD instruction adds the value of AX to the BX register.

* Memory:

Memory in Assembly Language refers to the primary storage of a computer system where data and instructions are stored during program execution. Accessing memory involves specifying memory addresses and using load and store operations to read from or write to those addresses. For instance:

MOV [memory\_address], AX ; Move the value of AX register to a memory location

MOV CX, [memory\_address] ; Move the value from a memory location to CX register

In these examples, memory\_address represents a specific memory location, and MOV instructions are used to transfer data between registers and memory.

* Instructions:

Assembly Language instructions are low-level commands that direct the CPU to perform specific operations. These instructions are represented by mnemonics, which are human-readable abbreviations for machine operations. Examples include:

MOV AX, BX ; Move data from BX register to AX register

ADD AX, 10 ; Add the value 10 to AX register

JMP label ; Jump to a specified label in the code

Each instruction performs a distinct operation, such as data movement, arithmetic, or control flow manipulation.

* Syntax:

Assembly Language syntax varies depending on the specific architecture and assembler being used. However, it generally consists of mnemonic instructions followed by operands and optional comments. For example:

; Calculate the sum of two numbers and store the result in AX register

MOV AX, 5 ; Move the value 5 into AX register

ADD AX, 10 ; Add the value 10 to AX register

In this example, MOV and ADD are instructions, AX and 10 are operands, and the semicolon ; denotes a comment.

* Data Types:

Assembly Language supports various data types, including integers, characters, strings, and floating-point numbers. Data types are represented using different instruction sets and formats depending on the architecture. For instance:

DB 'A' ; Define a single character

DW 1234 ; Define a 2-byte integer

DD 12345678 ; Define a 4-byte integer

These directives allocate memory and specify the data type for storage.

* Control Flow:

Control flow in Assembly Language governs the sequence of instructions executed by the CPU based on conditions and branching instructions. Examples include:

CMP AX, BX ; Compare the values in AX and BX registers

JE label ; Jump to a label if the previous comparison was equal

These instructions enable decision-making and looping constructs in Assembly programs, allowing for flexible program control.

1. Install NASM

* Firstly, I installed NASM from thje official site. Next, I put it in PATH variable, so I coul acces it from anywhere. Using *nasm -v,* I can check its version.



1. Simple programs + Debbuging

**First program:**

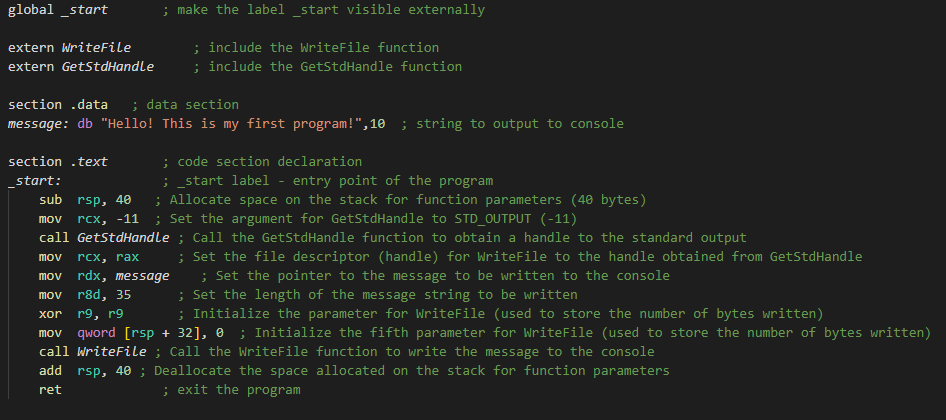


Figure 1. Program to print something to console

**Debugging with GDB:**

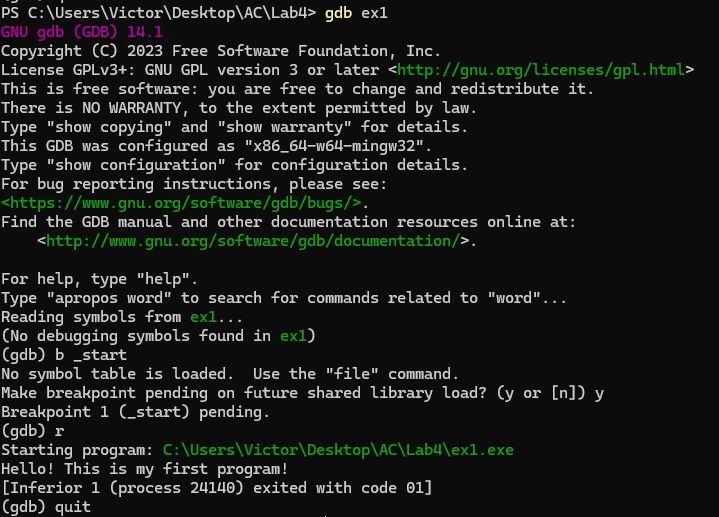


Figure 2. Debug info

**Console instructions:**

*nasm -f win64 ex1.asm -o ex1.o*

*link ex1.o kernel32.lib /entry:\_start /subsystem:console /out:ex1.exe*

*ex1.exe*

*gdb ex1*

*b \_start*

**Ouput:**



Figure 3. Ouput of first programm