Report on Laboratory Work No. 7

Discipline: Computer Architecture

ali hosseiabadi

Table of Contents

List of Figures

List of Tables

1 Purpose of the Work

Study of conditional and unconditional jump instructions. Acquiring skills in writing programs using jumps. Introduction to the purpose and structure of a listing file.

2 Assignment

1. Implement jumps in NASM
2. Study the structure of listing files
3. Independently write programs based on the materials of the laboratory work

3 Theoretical Introduction

To implement branching in assembly language, the so-called control transfer instructions or jump instructions are used. There are two types of jumps: * Conditional jump – the execution or non-execution of a jump to a specific point in the program depending on the condition check. * Unconditional jump – the execution of a control transfer to a specific point in the program without any conditions.

4 Laboratory Work Execution

4.1 Implementing Jumps in NASM

Create a directory for the laboratory work programs No. 7 (Fig. -fig. 4.1).

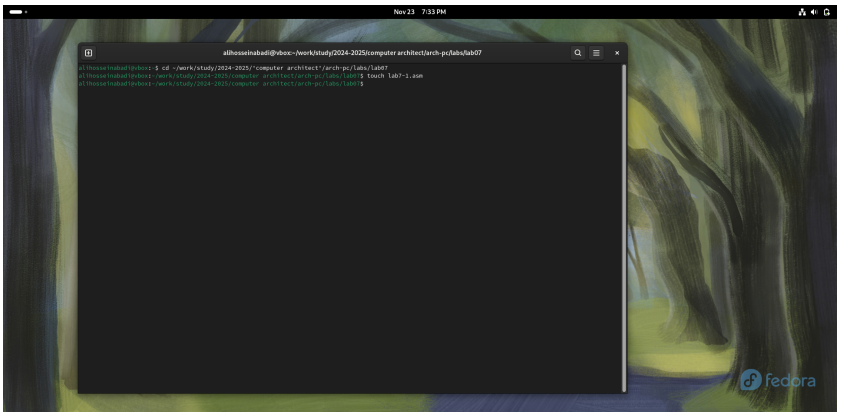


Fig. 4.1: Creating a directory and file for the program

Copy the code from the listing into the file of the future program. (Fig. -fig. 4.2).

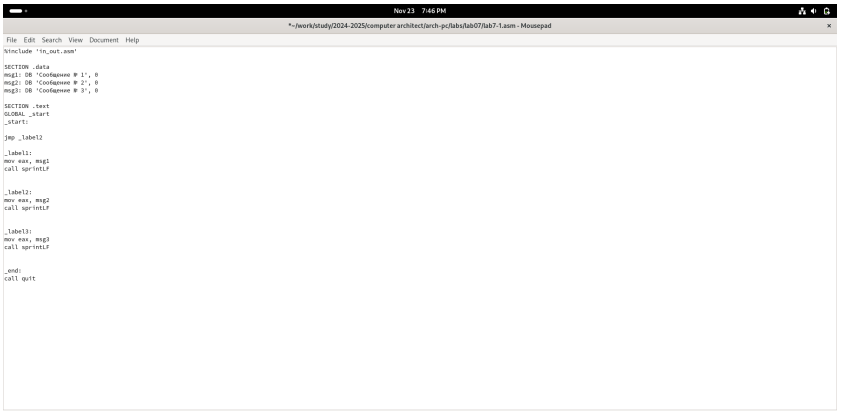


Fig. 4.2: Saving the program

Upon running the program, I confirmed that the unconditional jump indeed changes the order of instruction execution (Fig. -fig. 4.3).

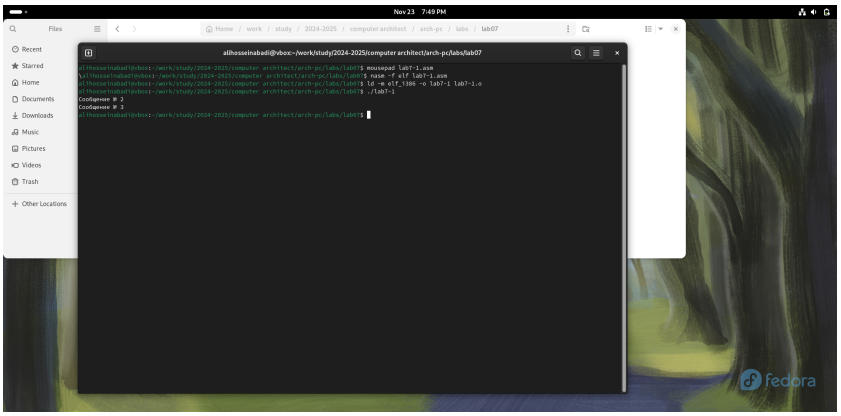


Fig. 4.3: Running the program

I modified the program so that the order of function execution changes (Fig. -fig. 4.4).

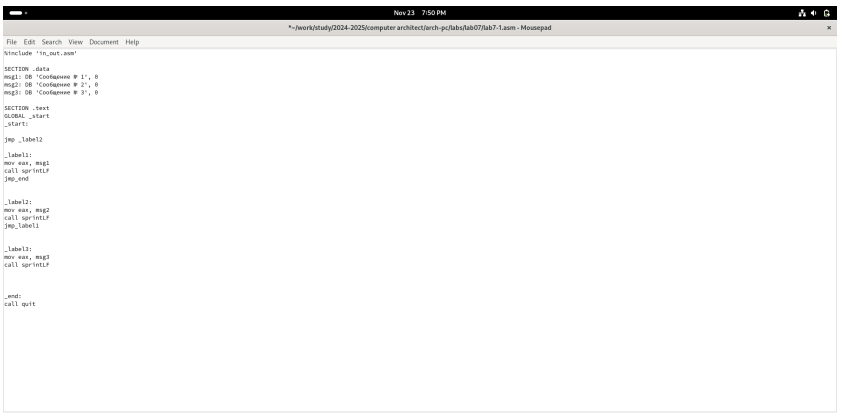


Fig. 4.4: Modifying the program

Run the program and check that the applied changes are correct (Fig. -fig. 4.5).

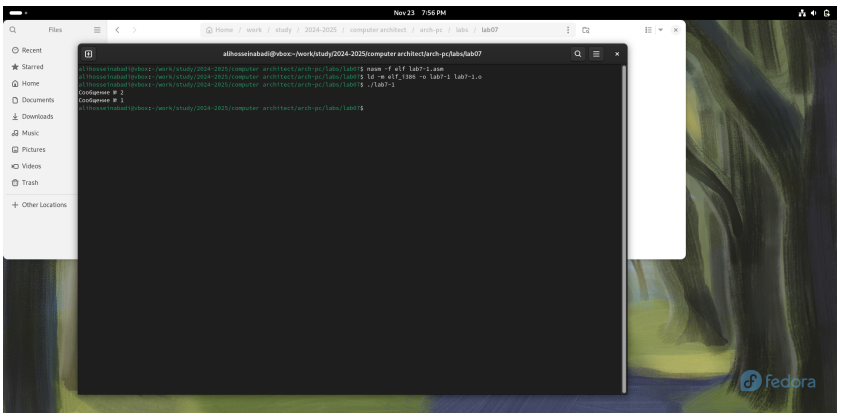


Fig. 4.5: Running the modified program

Now, I modify the program text so that all three messages are displayed in reverse order (Fig. -fig. 4.6).

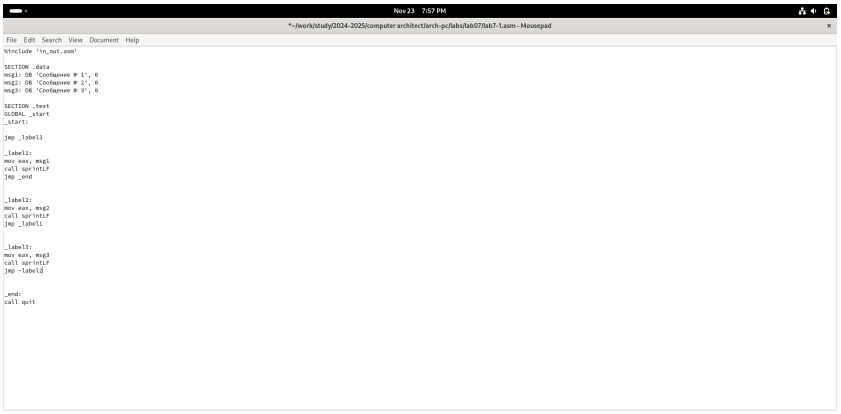


Fig. 4.6: Modifying the program

The work was completed correctly, the program displays the messages in the order I wanted (Fig. -fig. 4.7).

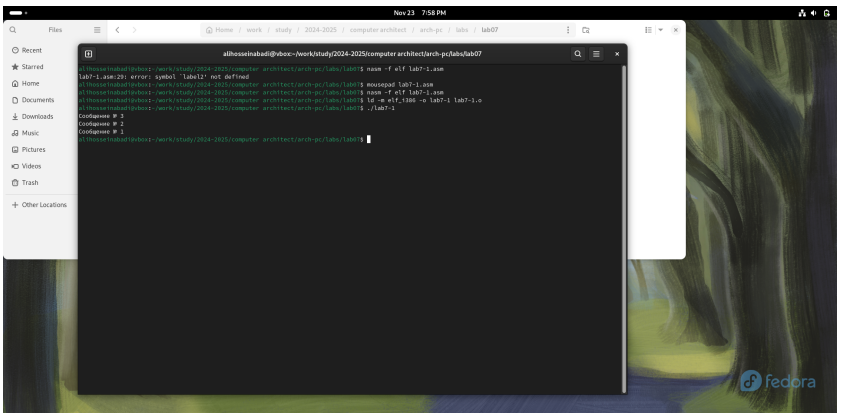


Fig. 4.7: Checking the changes

Create a new working file and insert the code from the next listing (Fig. -fig. 4.8).

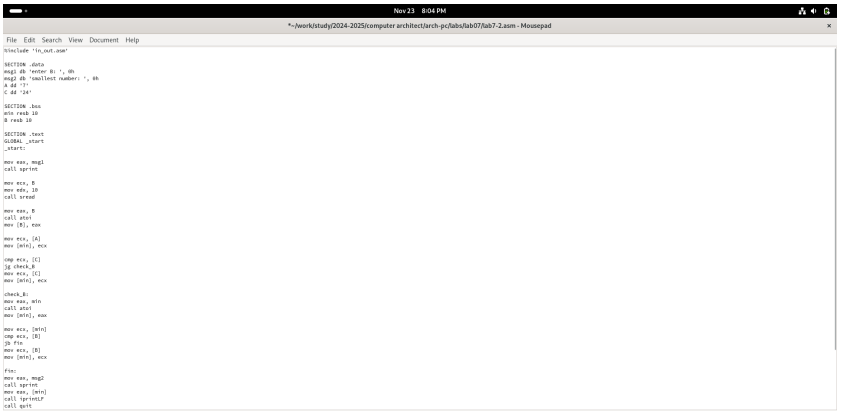


Fig. 4.8: Saving the new program

The program displays the value of the variable with the maximum value. I check the program with different input data (Fig. -fig. 4.9).

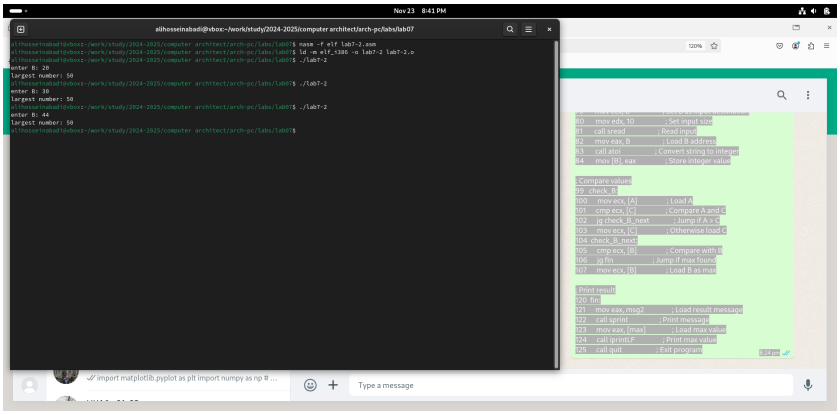


Fig. 4.9: Checking the program from the listing

4.2 Study of the Listing File Structure

Create a listing file using the -l flag of the nasm command and open it using the text editor mousepad (Fig. -fig. 4.10).

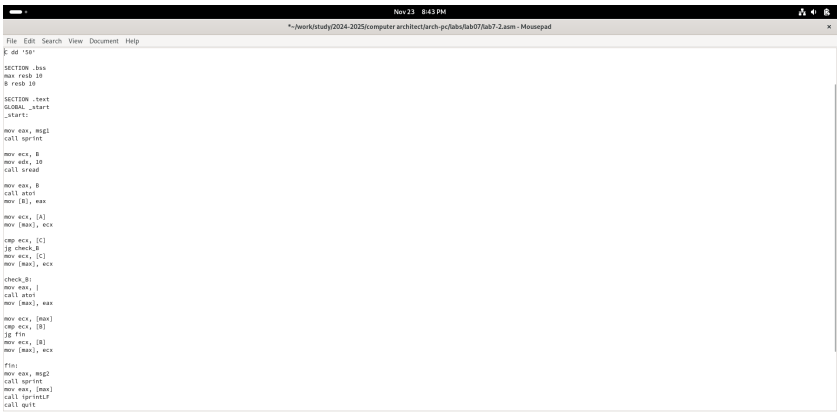


Fig. 4.10: Checking the listing file

The first value in the listing file is the line number, which may not match the line number of the original file. The second entry is the address, the offset of the machine code relative to the start of the current segment, followed by the machine code itself, and the line ends with the original program text with comments.

Delete one operand from a random instruction to check the behavior of the listing file in the future (Fig. -fig. 4.11).

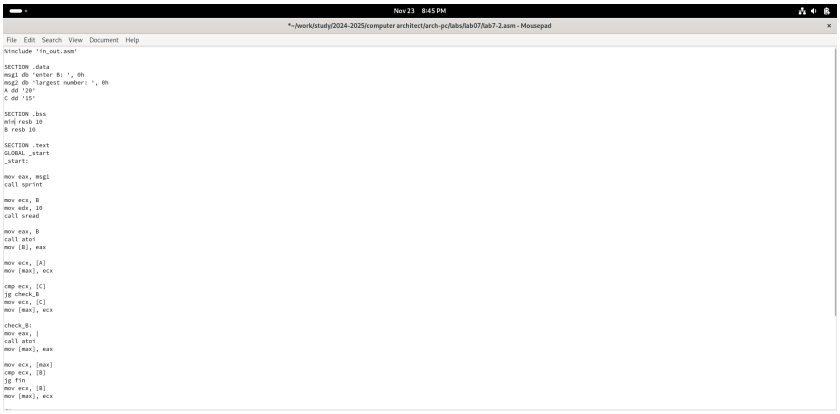


Fig. 4.11: Deleting an operand from the program

In the new listing file, an error that occurred during the translation of the file is displayed. No output files are created other than the listing file. (Fig. -fig. 4.12).

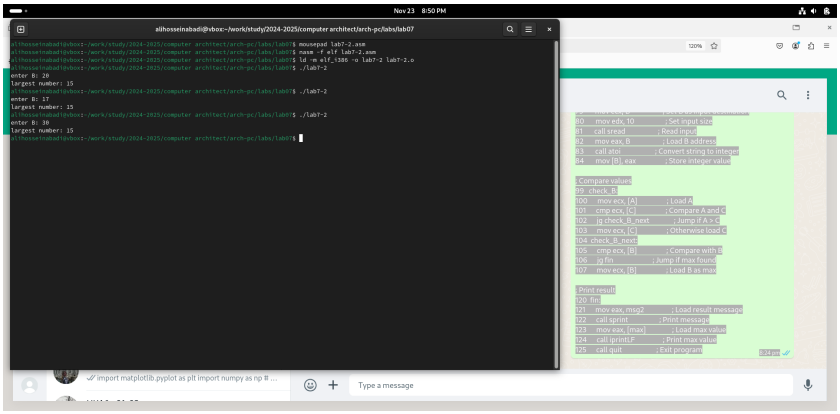


Fig. 4.12: Viewing the error in the listing file

4.3 Independent Work Assignments

I sincerely do not understand what variant I should have obtained during laboratory work No. 7, so I will use my variant – the ninth one – from the previous laboratory work. I return the operand to the function in the program and modify it so that it outputs the variable with the smallest value (Fig. -fig. 4.13).

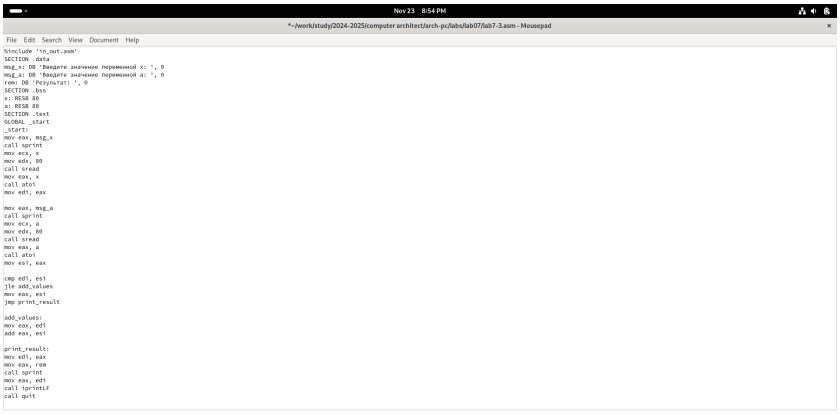
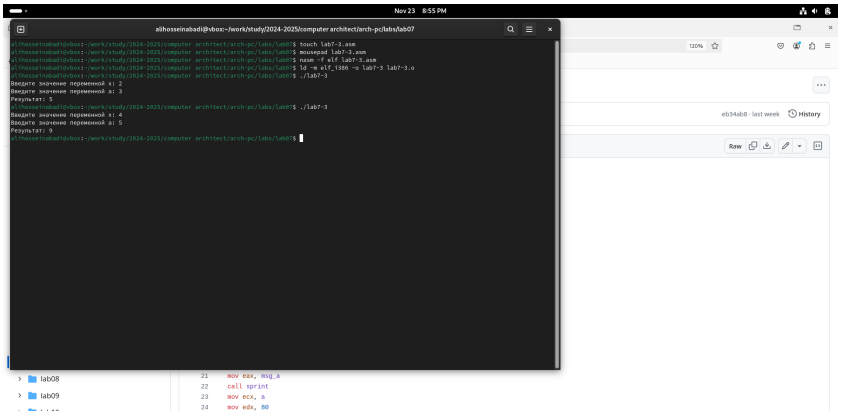


Fig. 4.13: First program of independent work



Code of the first program:

“‘NASM %include ‘in_out.asm’

SECTION .data msg1 db ‘Enter B:’, 0h msg2 db ‘The smallest number:’, 0h A dd ‘24’ C dd ‘15’

SECTION .bss min resb 10 B resb 10

SECTION .text GLOBAL _start _start:

mov eax, msg1 call sprint

mov ecx, B mov edx, 10 call sread

mov eax, B call atoi mov [B], eax

mov ecx, [A] mov [min], ecx

cmp ecx, [C] jg check_B mov ecx, [C] mov [min], ecx

check_B: mov eax, min call atoi mov [min], eax

mov ecx, [min] cmp ecx, [B] jb fin mov ecx, [B] mov [min], ecx

fin: mov eax, msg2 call sprint mov eax, [min] call iprintLF call qui