answer to labrotary work 8

Discipline: Computer Architecture

ali hosseinabadi

Content

List of illustrations

List of Tables

1 Goal of the Work

Acquiring skills in writing programs using loops and processing command-line arguments.

2 Assignment

1. Loop implementation in NASM
2. Processing command-line arguments
3. Independent program writing based on the materials of the laboratory work

3 Theoretical Introduction

A stack is a data structure organized according to the LIFO principle ("Last In — First Out"). The stack is part of the processor architecture and is implemented at the hardware level. The processor has special registers (ss, bp, sp) and commands for working with the stack.

The main function of the stack is to save return addresses and pass arguments when calling procedures. In addition, memory is allocated in it for local variables, and register values can be temporarily stored.

4 Performing the Laboratory Work

4.1 Implementing Loops in NASM

I create a file for laboratory work No. 8 (Fig. -fig. 4.1).

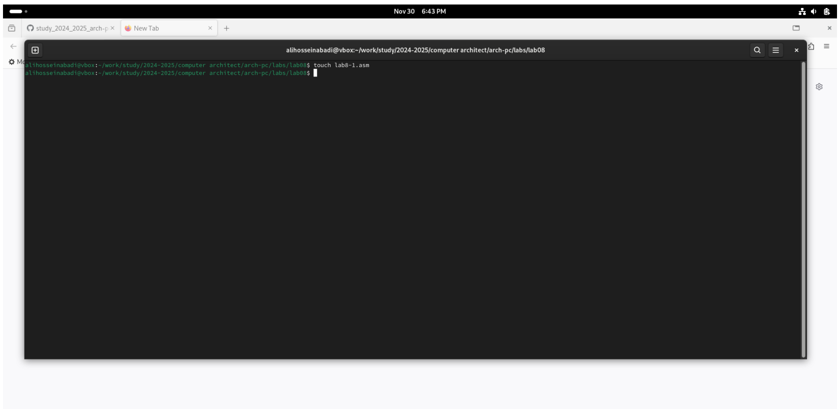


Fig. 4.1: create file

I copy the program from the listing into the created file (Fig. -fig. 4.2).

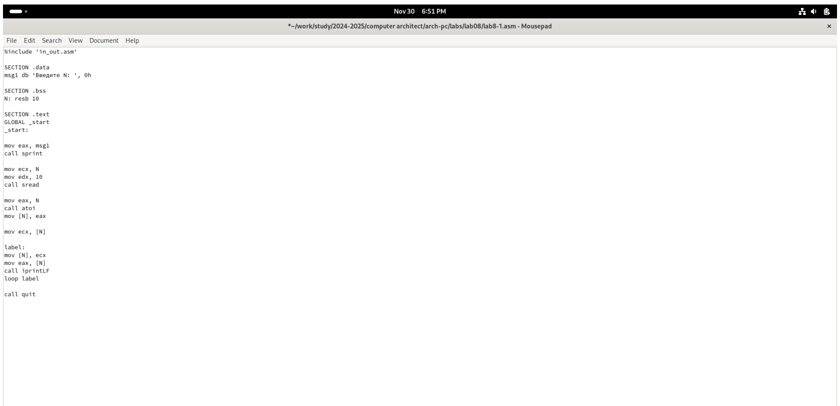


Fig. 4.2: copy program from list

I run the program; it shows the operation of loops in NASM (Fig. -fig. 4.3).

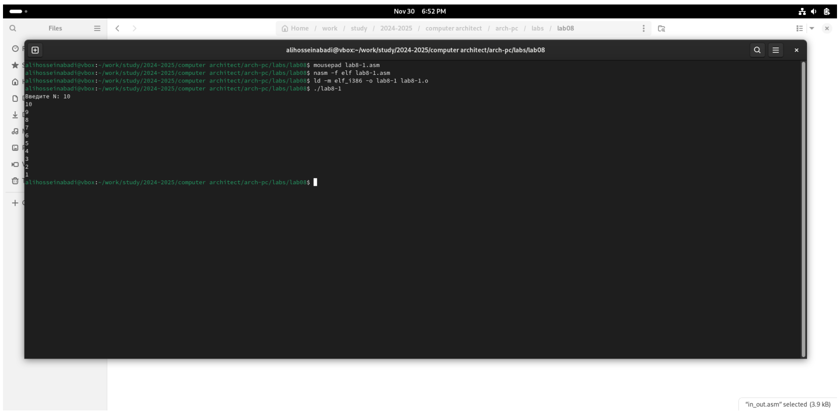


Fig. 4.3: run it

I replace the original program so that in the loop body I change the value of the ecx register (Fig. -fig. 4.4).

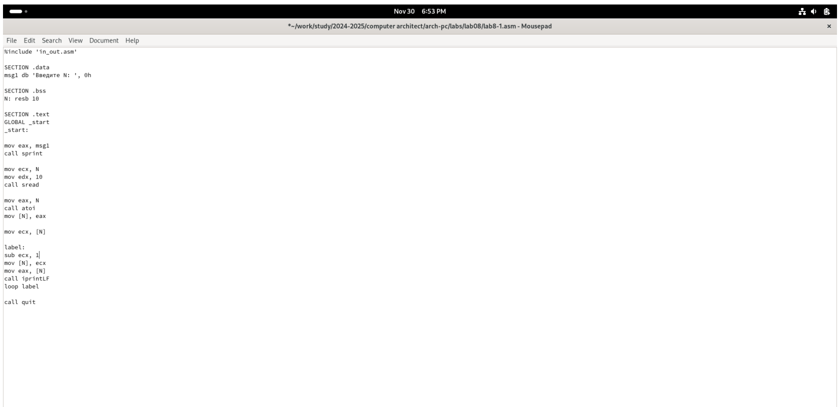


Fig. 4.4: change the program

Due to the fact that now the ecx register decreases by 2 values on each iteration, the number of iterations is halved (Fig. -fig. 4.5).

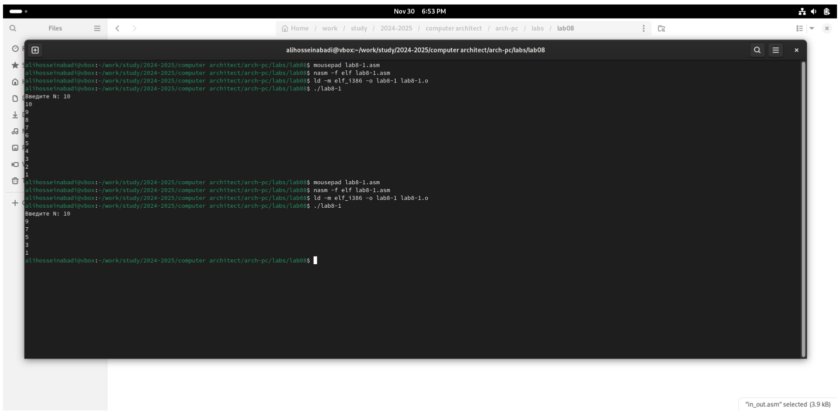


Fig. 4.5: run the new one

I add the push and pop commands to the program (Fig. -fig. 4.6).

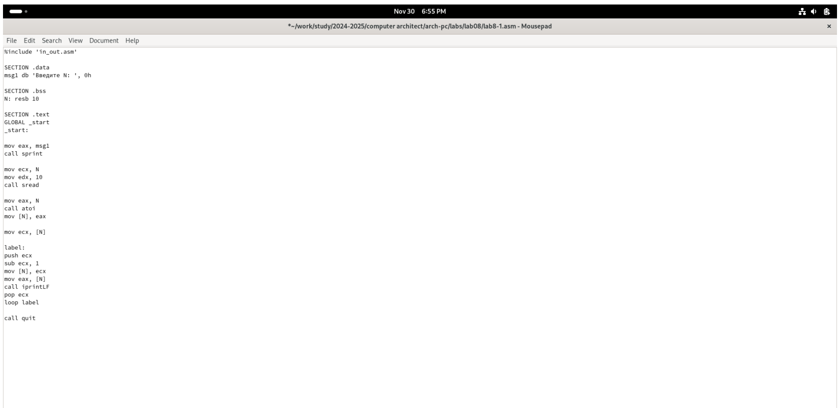


Fig. 4.6: Adding push and pop to the program loop

Now the number of iterations matches the entered N, but there was a shift in the output numbers by -1 (Fig. -fig. 4.7).

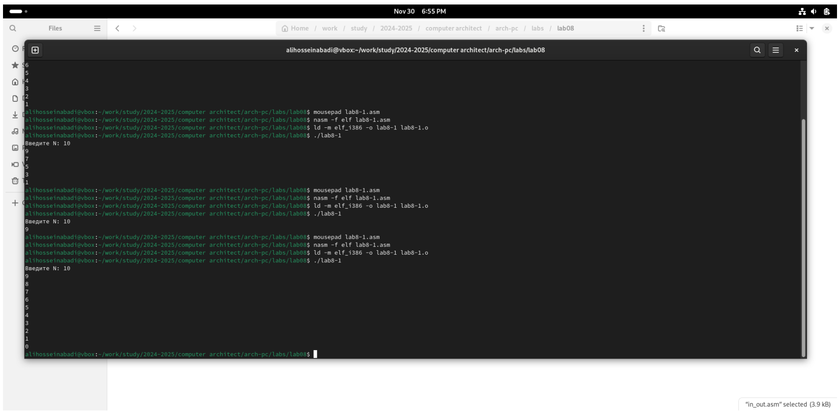


Fig. 4.7: run the new program

4.2 Processing Command-Line Arguments

I create a new file for the program and copy the code from the next listing into it (Fig. -fig. 4.8).

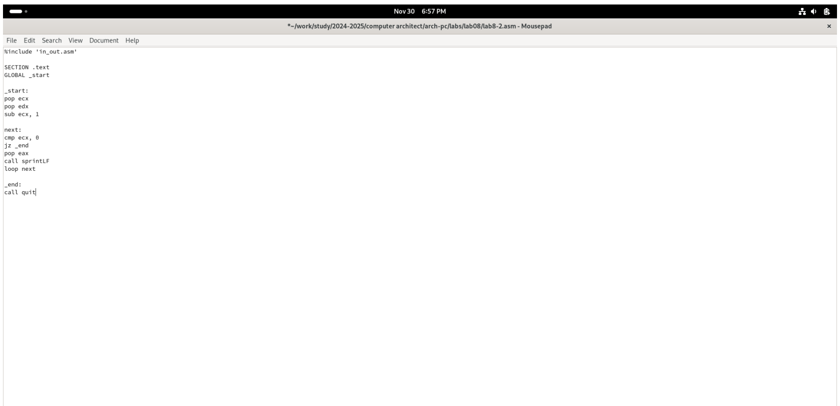


Fig. 4.8: copy program from the list

I compile the program and run it, specifying the arguments. The program processed the same number of arguments as were entered (Fig. -fig. 4.9).

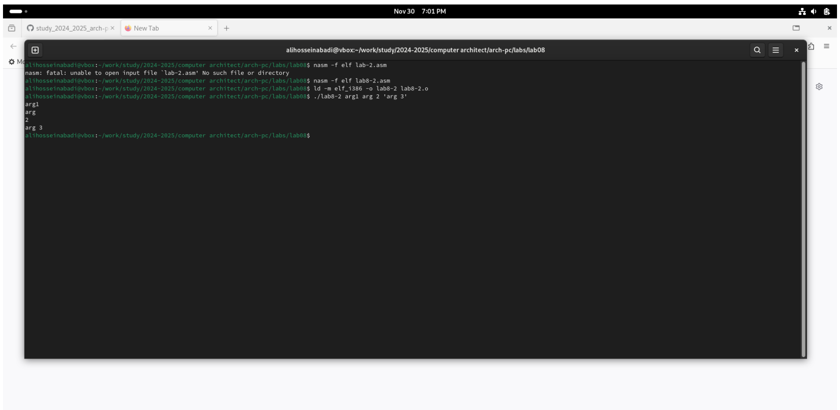
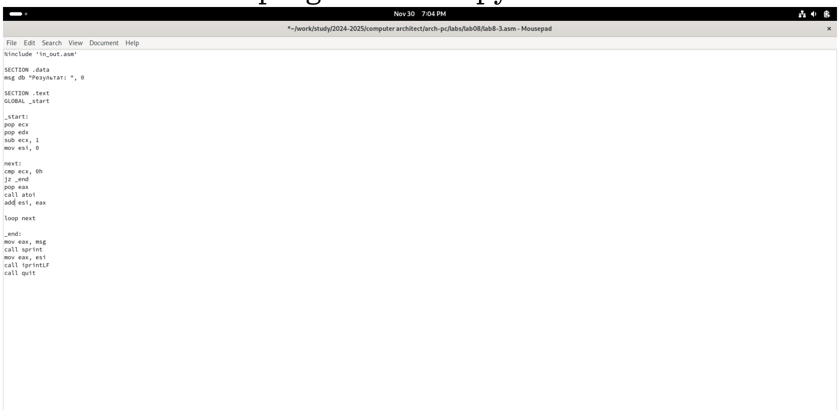


Fig. 4.9: run the code

I create a new file for the program and copy the code from the third listing into it (Fig.


-fig. **??**).

I compile the program and run it, specifying some numbers as arguments; the program adds them (Fig. -fig. 4.10).

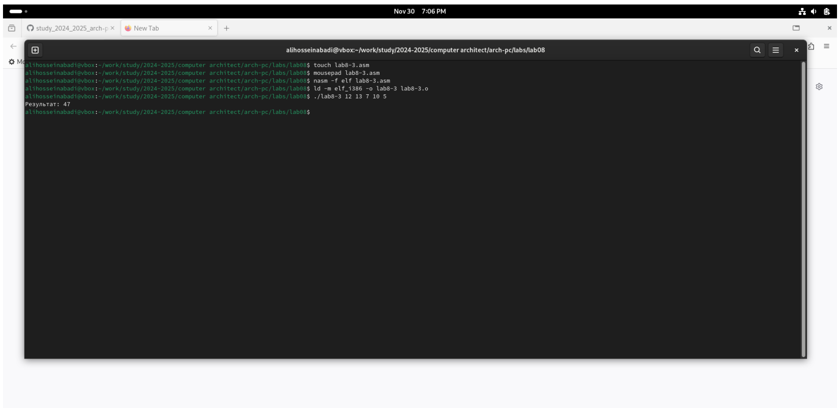


Fig. 4.10: run the third program

I change the program's behavior so that it multiplies the specified arguments instead of adding them (Fig. -fig. 4.11).

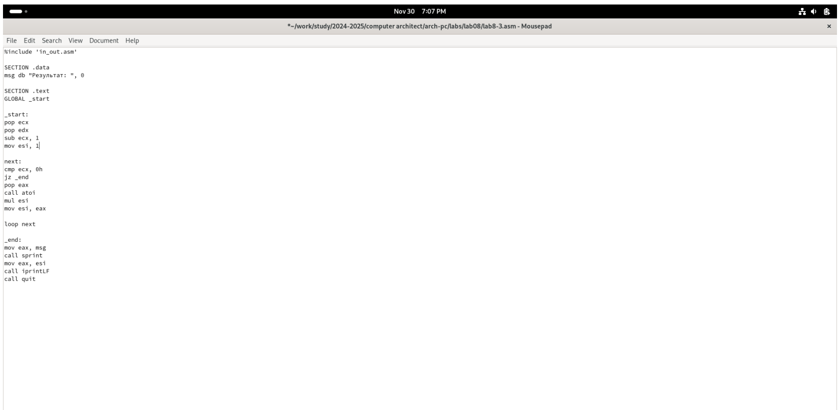


Fig. 4.11: change the program

The program now actually multiplies the input numbers (Fig. -fig. 4.12).

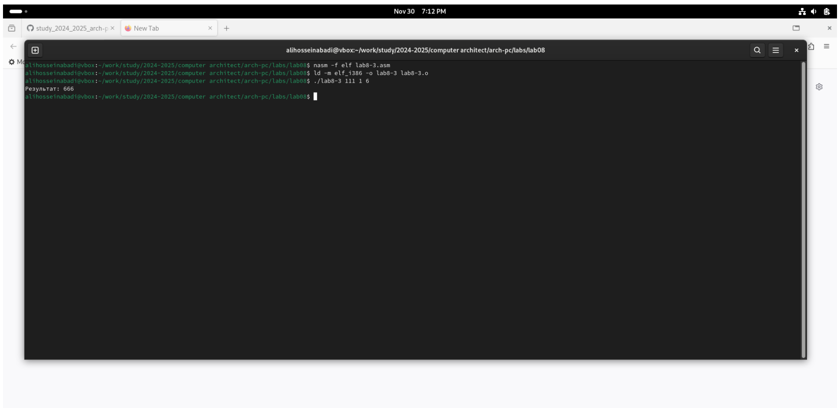


Fig. 4.12: run the new prorgram

4.3 Independent Work Assignment

I write a program that will find the sum of the values for the function f(x) = 5(2+x), which matches my ninth variant (Fig. -fig. 4.13).

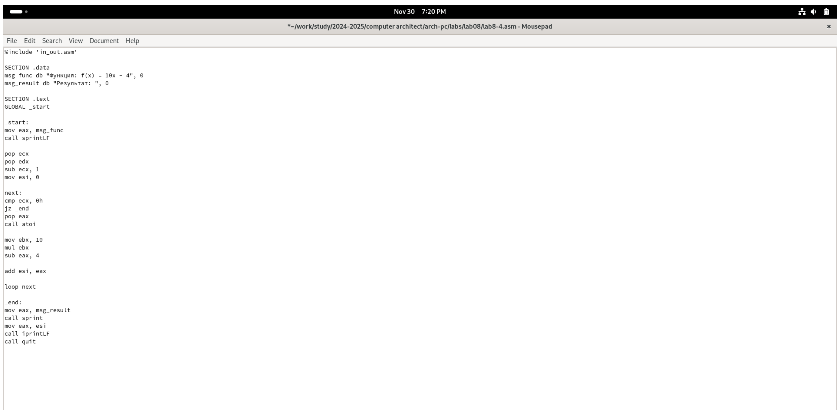


Fig. 4.13: write the program for individual program

Program code: “‘

%include ‘in_out.asm’

SECTION .data msg_func db “Функция: f(x) = 10x - 4”, 0 msg_result db “Результат:”, 0

SECTION .text GLOBAL _start

_start: mov eax, msg_func call sprintLF

pop ecx pop edx sub ecx, 1 mov esi, 0

next: cmp ecx, 0h jz _end pop eax call atoi

mov ebx, 10 mul ebx sub eax, 4

add esi, eax

loop next

_end: mov eax, msg_result call sprint mov eax, esi call iprintLF call quit

”’

I check the program’s operation, specifying several numbers as arguments (Fig. -fig. 4.14).

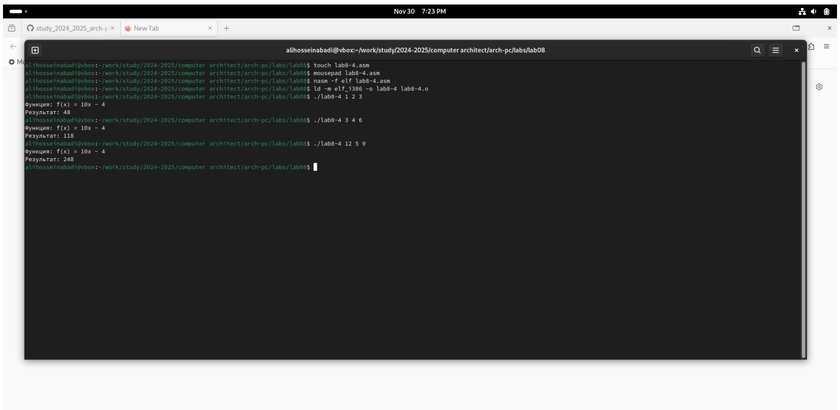


Fig. 4.14: run the program

5 Conclusions

As a result of this laboratory work, I acquired skills in writing programs using loops and also learned how to process command-line arguments.

6 References

1. Course on TUIS
2. Laboratory Work No. 8
3. Programming in NASM Assembler Language, Stolyarov A. V.