

answer to labrotary work 8

Discipline: Computer Architecture

ali hosseinabadi

Content

| | | |
|----------|---|-----------|
| 1 | Goal of the Work | 5 |
| 2 | Assignment | 6 |
| 3 | Theoretical Introduction | 7 |
| 4 | Performing the Laboratory Work | 8 |
| 4.1 | Implementing Loops in NASM | 8 |
| 4.2 | Processing Command-Line Arguments | 11 |
| 4.3 | Independent Work Assignment | 13 |
| 5 | Conclusions | 16 |
| 6 | References | 17 |

List of illustrations

| | | |
|------|--|----|
| 4.1 | create file | 8 |
| 4.2 | copy program from list | 8 |
| 4.3 | run it | 9 |
| 4.4 | change the program | 9 |
| 4.5 | run the new one | 10 |
| 4.6 | Adding push and pop to the program loop | 10 |
| 4.7 | run the new program | 11 |
| 4.8 | copy program from the list | 11 |
| 4.9 | run the code | 12 |
| 4.10 | run the third program | 12 |
| 4.11 | change the program | 13 |
| 4.12 | run the new program | 13 |
| 4.13 | write the program for individual program | 14 |
| 4.14 | run the program | 15 |

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).

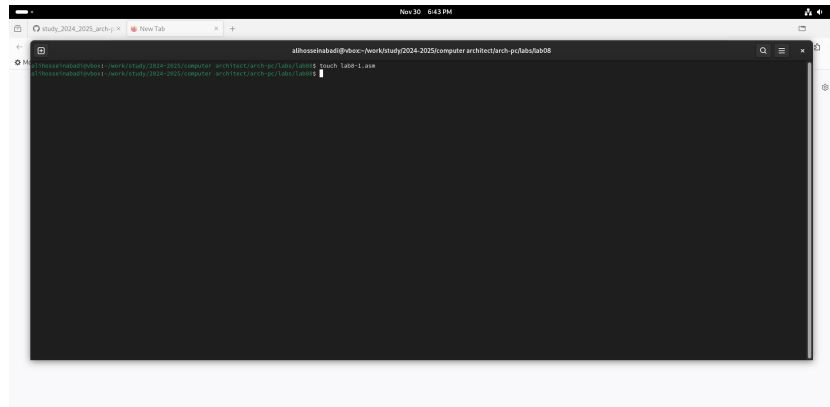


Fig. 4.1: create file

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

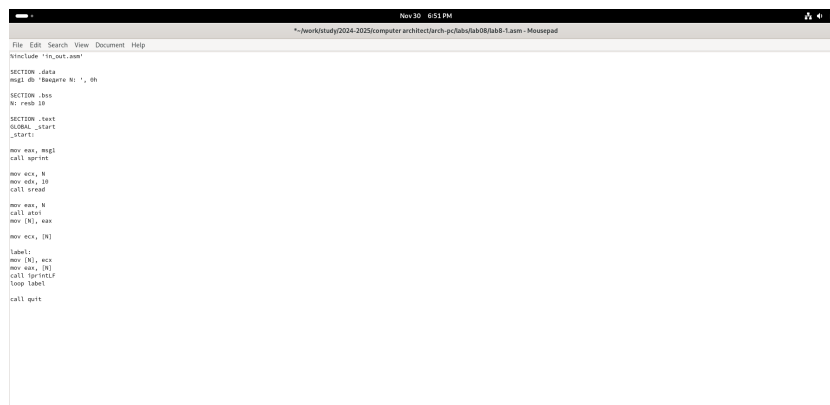
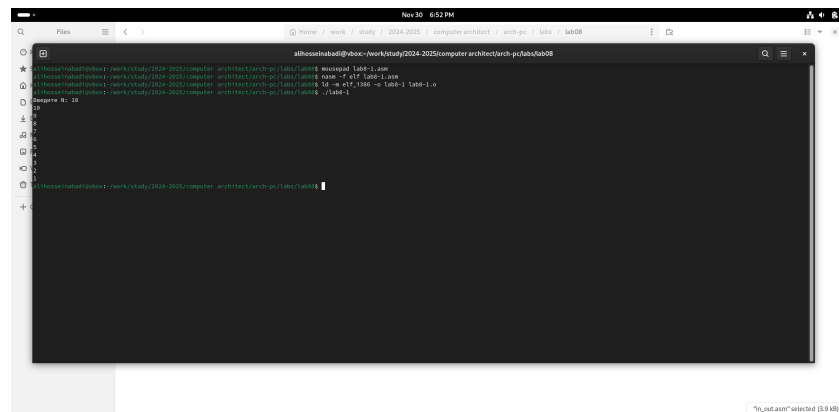


Fig. 4.2: copy program from list

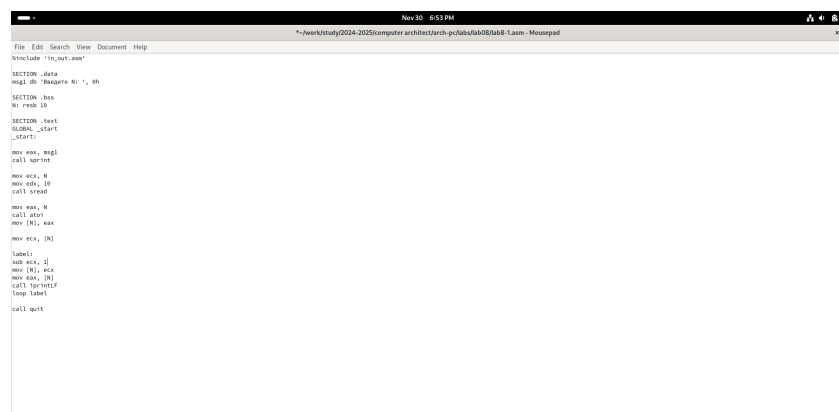
I run the program; it shows the operation of loops in NASM (Fig. -fig. 4.3).



```
Nov 30 6:52 PM
alhosseinabad@vbox:~/workstudy/2024-2025/computer architecture/lab0$ nasm -f elf lab0-1.asm
file format elf32-i386
alhosseinabad@vbox:~/workstudy/2024-2025/computer architecture/lab0$ ld -o lab0-1 lab0-1.o
ld: warning: no files specified
alhosseinabad@vbox:~/workstudy/2024-2025/computer architecture/lab0$ ld -o lab0-1 lab0-1.o
ld: warning: no files specified
alhosseinabad@vbox:~/workstudy/2024-2025/computer architecture/lab0$
```

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).



```
Nov 30 6:53 PM
~/workstudy/2024-2025/computer architecture/lab0/lab0-1.asm - Muesepad
File Edit Search View Document Help
#include 'io.h'
.section .data
msg db 'Tempare N: ', 0h
.section .bss
N: resb 10
.section .text
GLOBAL _start
_start:
mov eax, msg
call write
mov ecx, 0
mov ebx, 10
call read
mov eax, 0
call atoi
mov [N], eax
mov ecx, [N]
label:
sub ecx, 2
mov [N], ecx
mov ecx, [N]
call printf
jmp label
call quit
```

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).

```

athos@ubuntu:~/workstudy/2024-2025/computer-architecture-pc-labs/lab0$ miasm lab0-1.asm
N: 10
start: 0
0
1
2
3
4
5
6
7
8
9
athos@ubuntu:~/workstudy/2024-2025/computer-architecture-pc-labs/lab0$ miasm lab0-1.asm
N: 10
start: 0
0
1
2
3
4
5
6
7
8
9

```

Fig. 4.5: run the new one

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

```

#include "in_asm.asm"

SECTION .data
msg1 db "N: ", 0h

SECTION .bss
N: resb 10

SECTION .text
GLOBAL _start
_start:

    mov ecx, msg1
    call _printf

    mov ecx, N
    mov esi, 10
    call _printf

    mov ecx, 0
    call _atoi
    mov [N], ecx
    mov ecx, [N]

label:
    push ecx
    sub ecx, 1
    mov [N], ecx
    call _printf
    pop ecx
    loop label
    call _quit

```

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).

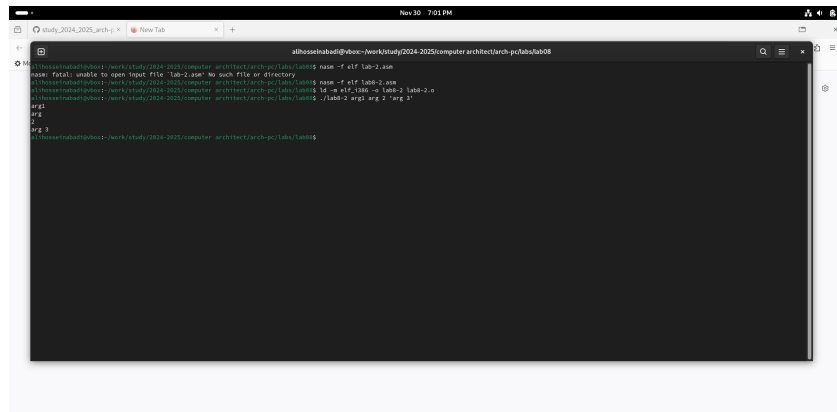
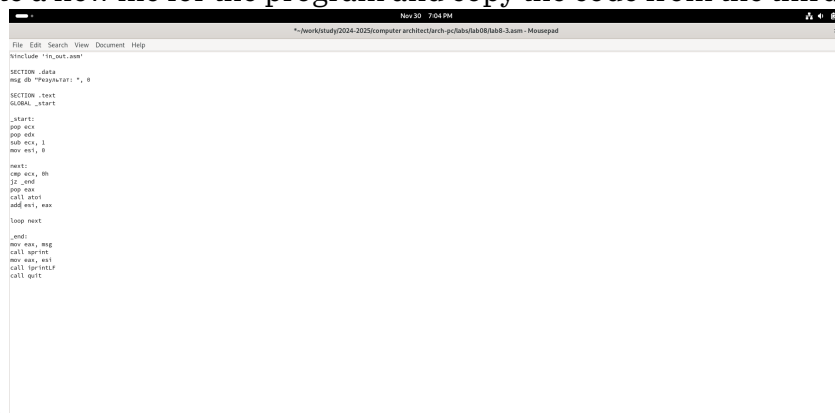


Fig. 4.9: run the code

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).

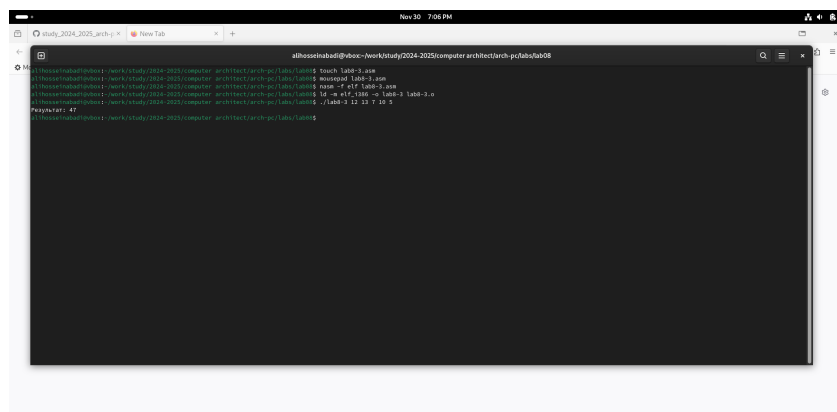
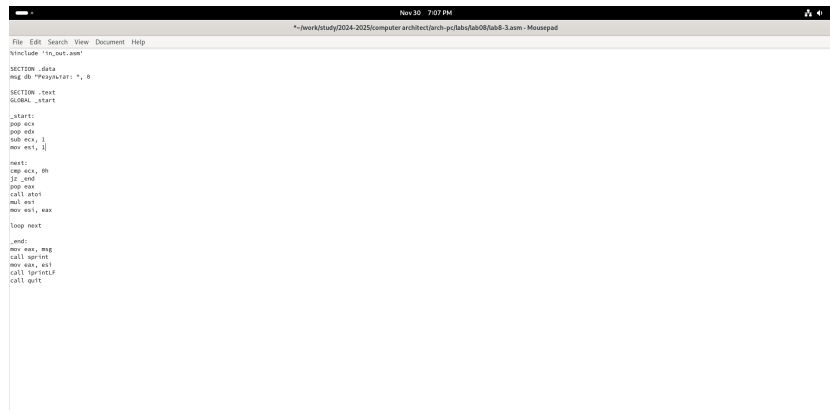


Fig. 4.10: run the third program

I change the program's behavior so that it multiplies the specified arguments instead

of adding them (Fig. -fig. 4.11).



```
File Edit Search View Document Help
~/workstudy/2024-2025/computer architecture-pclab/lab08/lab8_3.asm - Microsoft
#include "tn_asm.h"

SECTION .data
msg db "multiply: ", 0

SECTION .text
GLOBAL _start

_start:
    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 1

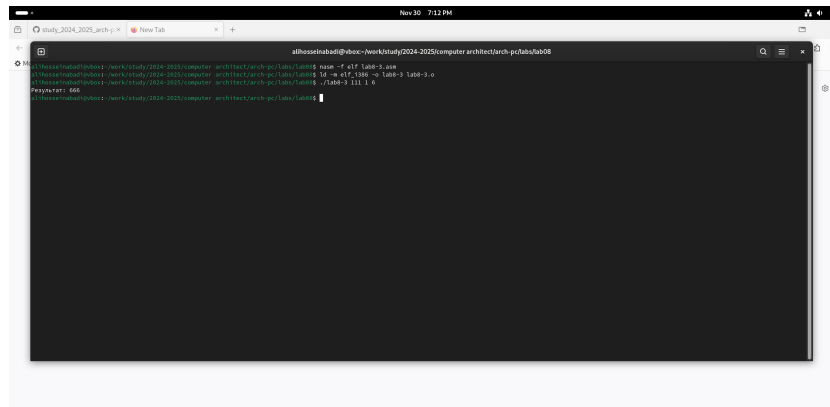
next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi
    mul esi
    mov esi, eax

loop next

_end:
    mov eax, msg
    call printf
    mov eax, nil
    call printf
    call _exit
```

Fig. 4.11: change the program

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



```
athosmabadi@doc:~/workstudy/2024-2025/computer architecture-pclab/lab08
$ ./lab8_3.asm
multiply: 5 2
10
```

Fig. 4.12: run the new program

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).

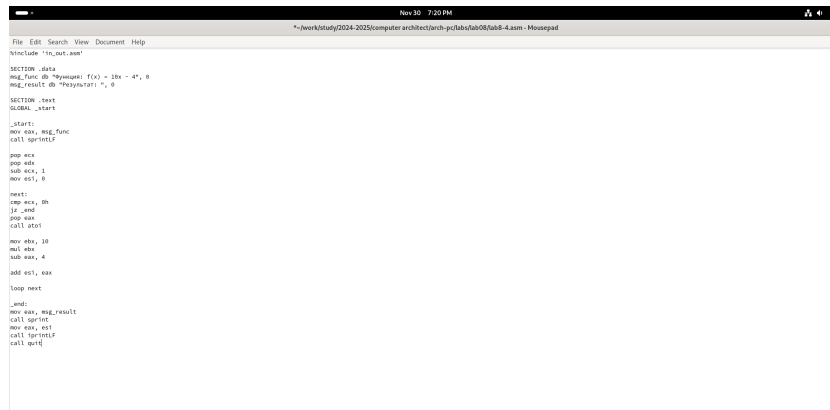


Fig. 4.13: write the program for individual program

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 sprintfLF

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

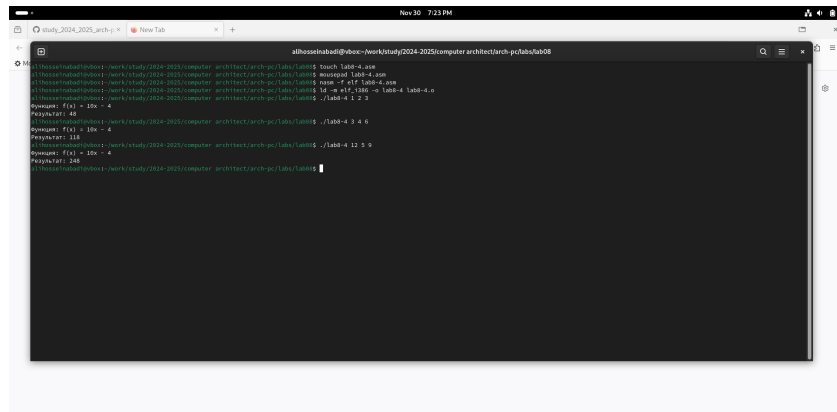
add esi, eax

loop next

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

”

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



```
Nov 30 7:23 PM
study_2024_2025_arch - New Tab
athosathabadi@doc:/workstudy/2024-2025/computer architecture-pc-lab/lab08

athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ touch lab0-4.asm
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ msvcrt64 lab0-4.asm
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ gcc -f elf lab0-4.asm
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ ls -ls a.out lab0-4.o
ls -ls a.out lab0-4.o
-rwxr-xr-x 1 athosathabadi lab0-4 1273
-rwxr-xr-x 1 athosathabadi lab0-4 1273
PyPy37a1: 48
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ ./lab0-4 3 4 5
PyPy37a1: 118
PyPy37a1: 118
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$ ./lab0-4 12 5 9
PyPy37a1: 248
PyPy37a1: 248
athosathabadi@doc:~/workstudy/2024-2025/computer architecture-pc-lab/lab08$
```

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.