

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра фундаментальной информатики и информационных технологий

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

Тема: Программирование цикла. Обработка аргументов командной строки.

Дисциплина: Архитектура компьютеров

Студент: Герчет Вячеслав

Группа: НКАбд-03-25

Студ. билет № 1132255650

Преподаватель: Штепа Кристина Александровна

МОСКВА

2025 г.

Содержание

1. Цель работы
2. Задание
3. Теоретическое введение
4. Выполнение лабораторной работы
 - 4.1 Реализация циклов в NASM
 - 4.2 Обработка аргументов командной строки
5. Выполнение самостоятельной работы
6. Выводы
7. Список литературы

Список иллюстраций

- 3.1 Создаем файл с помощью команды touch
- 3.2 Заполняем файл
- 3.3 Запускаем файл и проверяем его работу
- 3.4 Изменяем файл
- 3.5 Запускаем файл и смотрим на его работу
- 3.6 Редактируем файл
- 3.7 Проверяем, сошелся ли наш вывод с данным в условии выводом .
- 3.8 Создаем файл командой touch
- 3.9 Заполняем файл
- 3.10 Смотрим на работу программ
- 3.11 Создаем файл командой touch
- 3.12 Заполняем файл
- 3.13 Смотрим на работу программы
- 3.14 Изменяем файл
- 3.15 Проверяем работу файла(работает правильно)
- 3.16 Создаем файл командой touch
- 3.17 Пишем программу
- 3.18 Смотрим на работу программы при $x1=1$ $x2=2$ $x1=3$ (всё верно)
- 3.19 Смотрим на работу программы при $x1=4$ $x2=5$ (всё верно)

1. Цель работы

Целью лабораторной работы является получение практических навыков программирования циклов в NASM и освоение обработки аргументов командной строки.

2. Задание

Написать программы с использованием циклов и обработкой аргументов ко-мандной строки.

3. Теоретическое введение

В данной работе изучаются два ключевых механизма языка ассемблера NASM:

1. Стек

Стек - структура данных LIFO, используемая для хранения адресов возврата, локальных переменных и временных значений. Основные операции:

push - помещает значение в стек;

pop - извлекает значение из стека.

2. Организация циклов

Для циклов используется специальная инструкция loop, которая автоматически уменьшает значение регистра ecx и делает переход к метке, если ecx \neq 0. На основе этого создаются простые и вложенные циклы.

3. Аргументы командной строки

При запуске программы NASM аргументы автоматически помещаются в стек. Сначала извлекается количество аргументов, затем сами аргументы обрабатываются в цикле.

4. Выполнение лабораторной работы

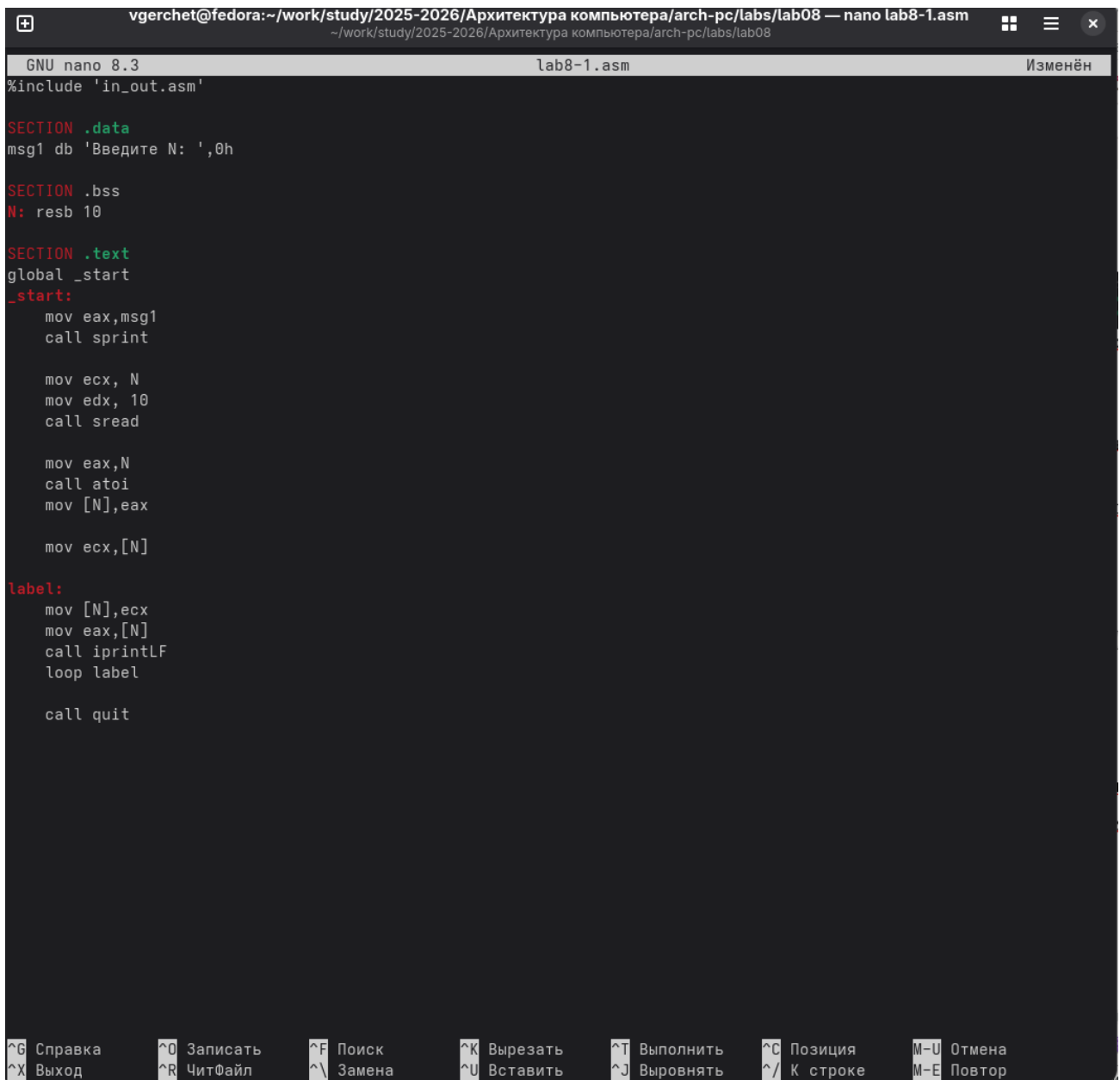
4.1 Реализация циклов в NASM

Создаем файл (рис. 3.1).

```
(base) vgerchet@fedora:~$ cd "/home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08"
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-1.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.1: Создаем файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 8.1 (рис. 3.2).



```
GNU nano 8.3 lab8-1.asm Изменён
~/.work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08

#include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

SECTION .text
global _start
_start:
    mov eax,msg1
    call sprint

    mov ecx, N
    mov edx, 10
    call sread

    mov eax,N
    call atoi
    mov [N],eax

    mov ecx,[N]

label:
    mov [N],ecx
    mov eax,[N]
    call iprintLF
    loop label

    call quit

^G Справка      ^O Записать     ^F Поиск       ^K Вырезать    ^T Выполнить   ^С Позиция     M-U Отмена
^X Выход        ^R ЧитФайл     ^\ Замена      ^U Вставить    ^J Выровнять   ^_ К строке    M-E Повтор
```

Рис. 3.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. 3.3).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
ld -m elf_i386 -o lab8-1 lab8-1.o
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
```

Рис. 3.3: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его, добавив изменение значения регистра в цикле (рис. 3.4)

```
label:
    sub ecx, 1
    mov [N],ecx
    mov eax,[N]
    call iprintLF
    loop label

    call quit
```

Рис. 3.4: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 3.5).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
ld -m elf_i386 -o lab8-1 lab8-1.o
./lab8-1
Введите N: 10
9
7
5
3
1
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.5: Запускаем файл и смотрим на его работу

Регистр ecx принимает значения 9,7,5,3,1(на вход подается число 10, в цикле

label данный регистр уменьшается на 2 командой sub и loop).

Число проходов цикла не соответствует числу N, так как уменьшается на 2.

Снова открываем файл для редактирования и изменяем его, чтобы все корректно работало (рис. 3.6).

```
label:
    push ecx
    sub ecx, 1
    mov [N],ecx
    mov eax,[N]
    call iprintLF
    pop ecx
    loop label

    call quit
```

Рис. 3.6: Редактируем файл

Создаем исполняемый файл и запускаем его (рис. 3.7).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
ld -m elf_i386 -o lab8-1 lab8-1.o
./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.7: Проверяем, сошелся ли наш вывод с данным в условии выводом

В данном случае число проходов цикла равна числу N.

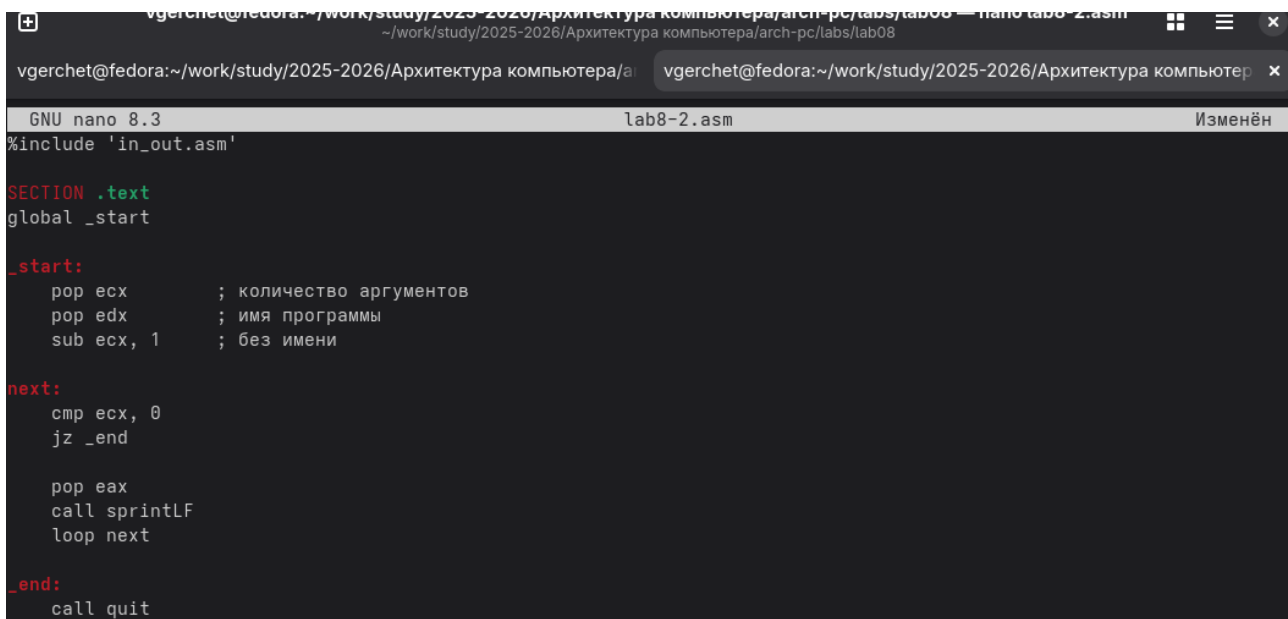
4.2 Обработка аргументов командной строки.

Создаем новый файл (рис. 3.8).

```
vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08
~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08
vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-2.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.8: Создаем файл командой touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 8.2 (рис. 3.9).



```
GNU nano 8.3 lab8-2.asm Изменён
%include 'in_out.asm'

SECTION .text
global _start

_start:
    pop ecx          ; количество аргументов
    pop edx          ; имя программы
    sub ecx, 1       ; без имени

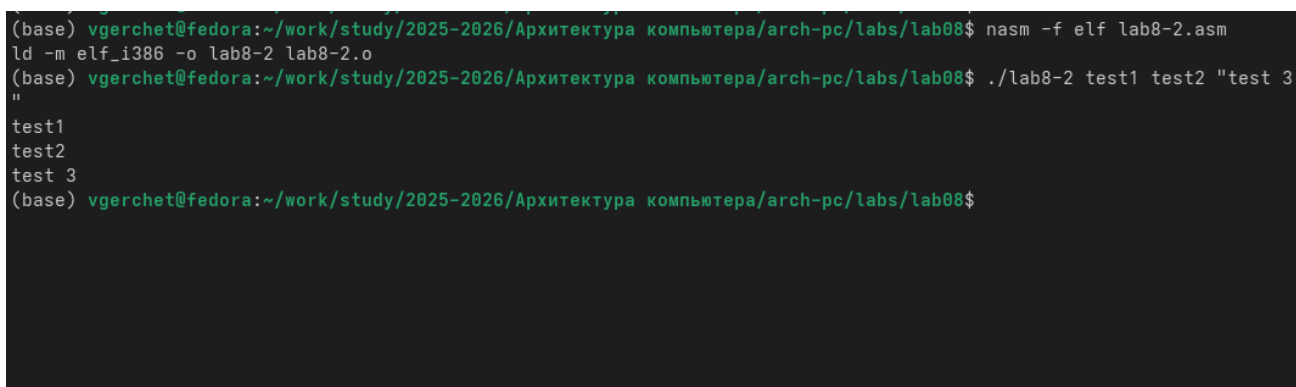
next:
    cmp ecx, 0
    jz _end

    pop eax
    call sprintLF
    loop next

_end:
    call quit
```

Рис. 3.9: Заполняем файл

Создаем исполняемый файл и проверяем его работу, указав аргументы (рис. 3.10).



```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-2.asm
ld -m elf_i386 -o lab8-2 lab8-2.o
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-2 test1 test2 "test 3"
test1
test2
test 3
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.10: Смотрим на работу программ

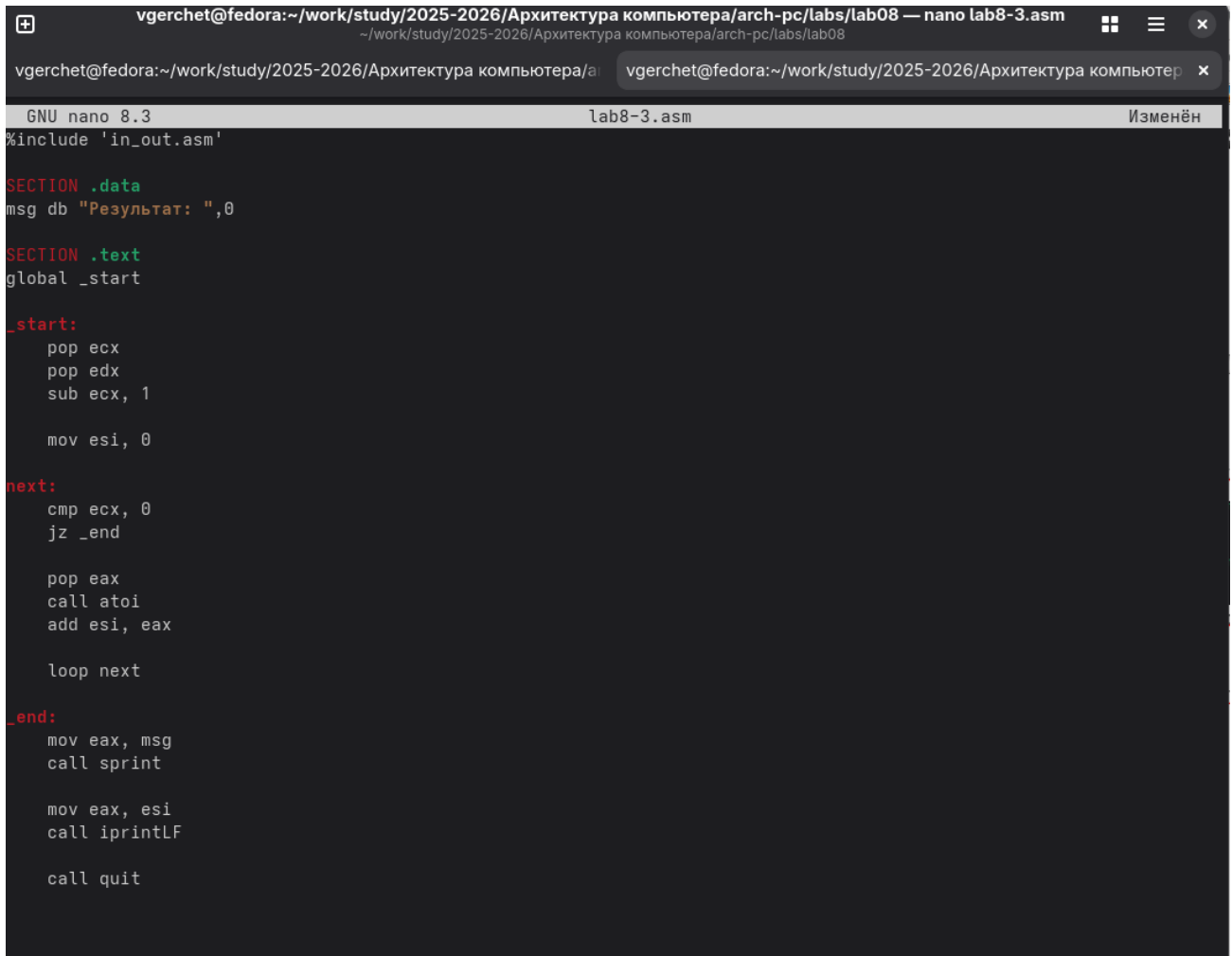
Программой было обработано 3 аргумента.

Создаем новый файл lab8-3.asm (рис. 3.11).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-3.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.11: Создаем файл командой touch

Открываем файл и заполняем его в соответствии с листингом 8.3 (рис. 3.12).



```
GNU nano 8.3 lab8-3.asm Изменён
#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx, 1

    mov esi, 0

next:
    cmp ecx, 0
    jz _end

    pop eax
    call atoi
    add esi, eax

    loop next

_end:
    mov eax, msg
    call sprint

    mov eax, esi
    call iprintLF

    call quit
```

Рис. 3.12: Заполняем файл

Создаём исполняемый файл и запускаем его, указав аргументы (рис. 3.13).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-3.asm
ld -m elf_i386 -o lab8-3 lab8-3.o
./lab8-3 12 13 7 10 5
Результат: 47
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.13: Смотрим на работу программы

Снова открываем файл для редактирования и изменяем его, чтобы вычислялось произведение вводимых значений (рис. 3.14).

```
mov esi, 1
next:
cmp ecx, 0
jz _end

pop eax
call atoi
imul esi, eax

loop next
```

Рис. 3.14: Изменяем файл

Создаём исполняемый файл и запускаем его, указав аргументы (рис. 3.15).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-3.asm
ld -m elf_i386 -o lab8-3 lab8-3.o
./lab8-3 2 3 4
Результат: 24
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.15: Проверяем работу файла(работает правильно)

5. Выполнение самостоятельной работы

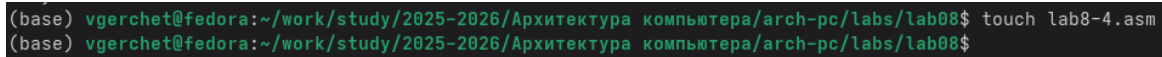
ВАРИАНТ-11

1. Напишите программу, которая находит сумму значений функции $f(x)$ для

$x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$.

Значения θ_i передаются как аргументы. Вид функции $\theta(\theta_i)$ выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах $\theta = \theta_1, \theta_2, \dots, \theta_n$.

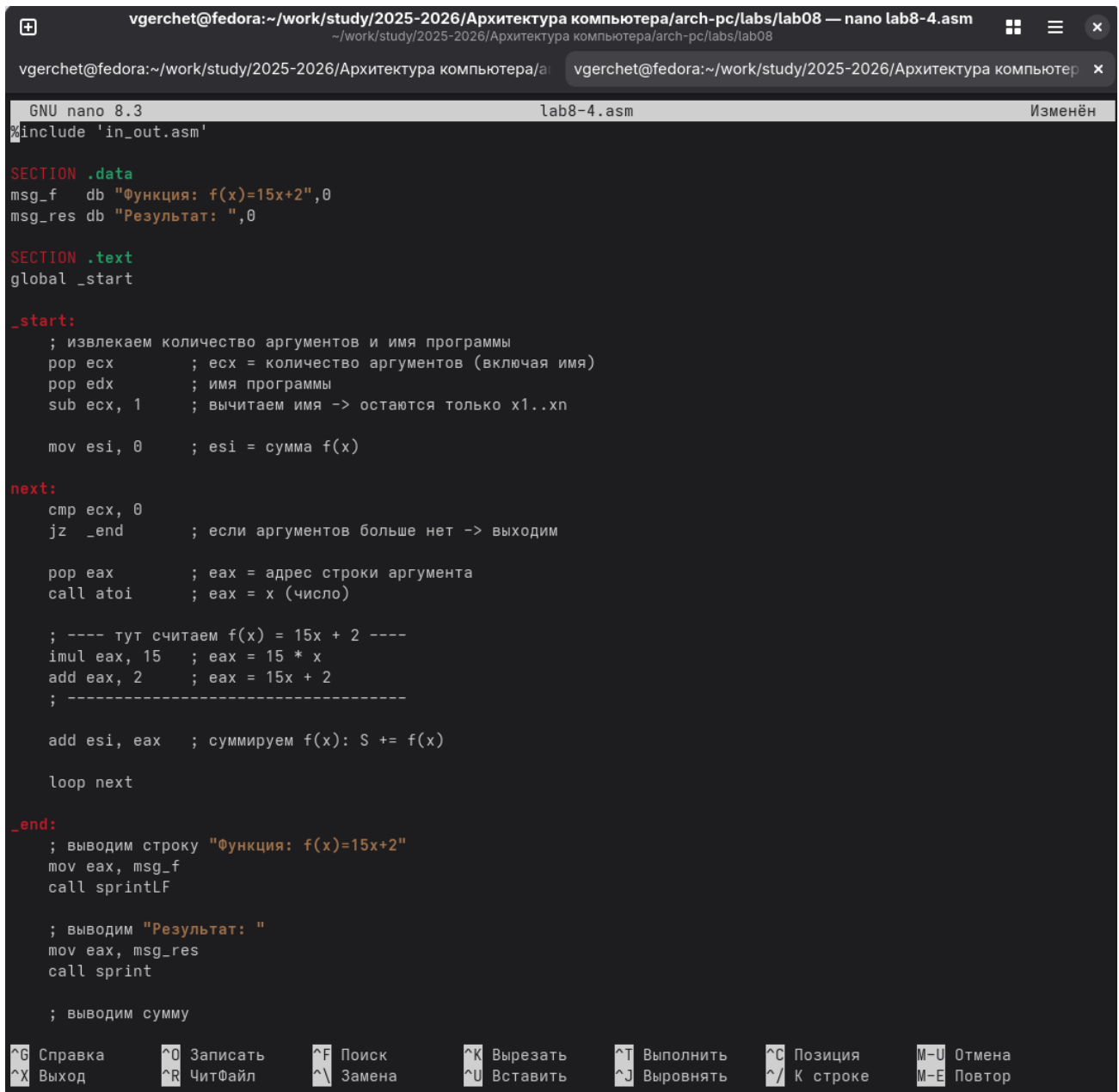
Создаем новый файл (рис. 3.16).



```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-4.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.16: Создаем файл командой touch

Открываем его и пишем программу, которая выведет сумму значений, получившихся после решения выражения $(f(x)=15x+2)$. (рис. 3.17).



```
GNU nano 8.3 lab8-4.asm Изменён
#include 'in_out.asm'

SECTION .data
msg_f db "Функция: f(x)=15x+2",0
msg_res db "Результат: ",0

SECTION .text
global _start

_start:
; извлекаем количество аргументов и имя программы
pop ecx ; ecx = количество аргументов (включая имя)
pop edx ; имя программы
sub ecx, 1 ; вычитаем имя -> остаются только x1..xn

mov esi, 0 ; esi = сумма f(x)

next:
cmp ecx, 0
jz _end ; если аргументов больше нет -> выходим

pop eax ; eax = адрес строки аргумента
call atoi ; eax = x (число)

; ---- тут считаем f(x) = 15x + 2 ----
imul eax, 15 ; eax = 15 * x
add eax, 2 ; eax = 15x + 2
; -----

add esi, eax ; суммируем f(x): S += f(x)

loop next

_end:
; выводим строку "Функция: f(x)=15x+2"
mov eax, msg_f
call sprintf

; выводим "Результат: "
mov eax, msg_res
call sprintf

; выводим сумму
```

^G Справка ^O Записать ^F Поиск ^K Вырезать ^T Выполнить ^C Позиция M-U Отмена
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/ К строке M-E Повтор

Рис. 3.17: Пишем программу

Транслируем файл и смотрим на работу программы (рис. 3.18).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-4.asm
ld -m elf_i386 -o lab8-4 lab8-4.o
./lab8-4 1 2 3
Функция:  $f(x)=15x+2$ 
Результат: 96
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.18: Смотрим на работу программы при $x_1=1$ $x_2=2$ $x_1=3$ (всё верно)

Транслируем файл и смотрим на работу программы (рис. 3.19).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-4 4 5
Функция:  $f(x)=15x+2$ 
Результат: 139
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab08$
```

Рис. 3.19: Смотрим на работу программы при $x_1=4$ $x_2=5$ (всё верно)

4. Выводы

В ходе лабораторной работы я разобрался, как в NASM организуются циклы с помощью инструкции `loop` и как правильно использовать стек для сохранения значений регистров. Научился обрабатывать аргументы командной строки, извлекать их из стека и выполнять с ними вычисления. Также выполнил программу по своему варианту и закрепил навыки работы с циклами, арифметикой и преобразованием данных.

5. Список литературы

NASM Documentation. <https://www.nasm.us/docs.html>

Демидова А. В. Архитектура ЭВМ. Лабораторная работа №8.

GDB: The GNU Project Debugger. <https://sourceware.org/gdb>