

# **Отчёт по лабораторной работе 8**

**дисциплина: Архитектура компьютеров**

Грачев Я. М. НПИбд-01-24

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Реализация циклов в NASM . . . . .	6
2.2	Самостоятельное задание . . . . .	15
<b>3</b>	<b>Выводы</b>	<b>18</b>

## Список иллюстраций

2.1	Создан каталог . . . . .	6
2.2	Программа lab8-1.asm . . . . .	7
2.3	Запуск программы lab8-1.asm . . . . .	8
2.4	Программа lab8-1.asm . . . . .	9
2.5	Запуск программы lab8-1.asm . . . . .	9
2.6	Программа lab8-1.asm . . . . .	10
2.7	Запуск программы lab8-1.asm . . . . .	11
2.8	Программа lab8-2.asm . . . . .	12
2.9	Запуск программы lab8-2.asm . . . . .	12
2.10	Программа lab8-3.asm . . . . .	13
2.11	Запуск программы lab8-3.asm . . . . .	13
2.12	Программа lab8-3.asm . . . . .	14
2.13	Запуск программы lab8-3.asm . . . . .	15
2.14	Программа lab8-task1.asm . . . . .	16
2.15	Запуск программы lab8-task1.asm . . . . .	17

## **Список таблиц**

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

## 2 Выполнение лабораторной работы

### 2.1 Реализация циклов в NASM

Создал каталог для программ лабораторной работы № 8 и файл `lab8-1.asm` (рис. 2.1).

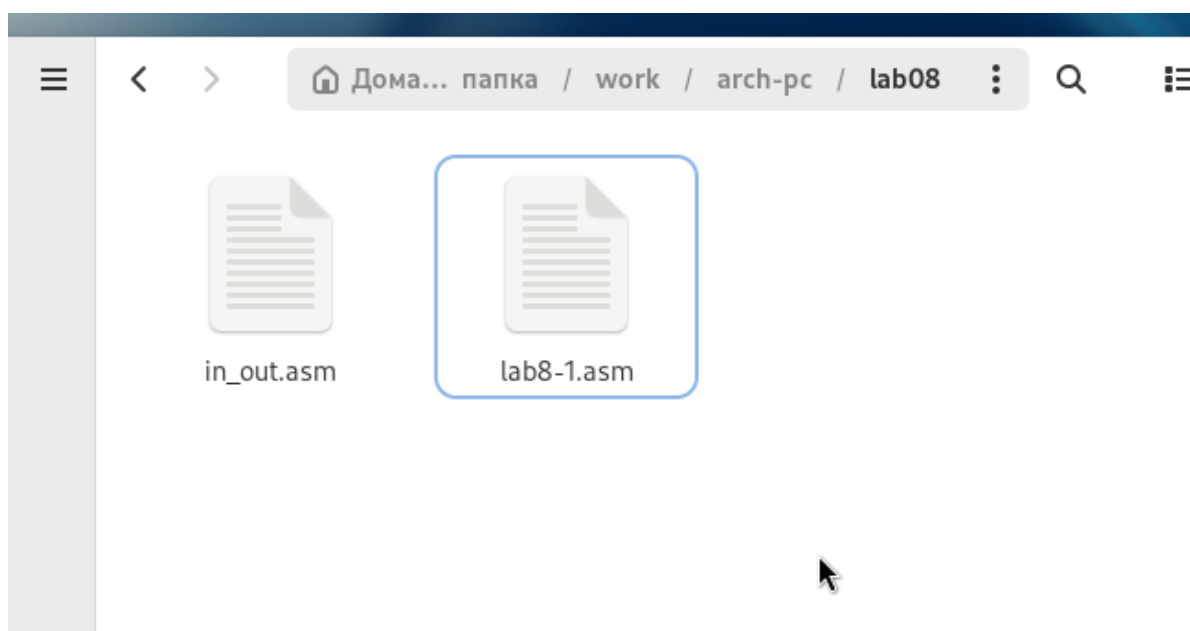
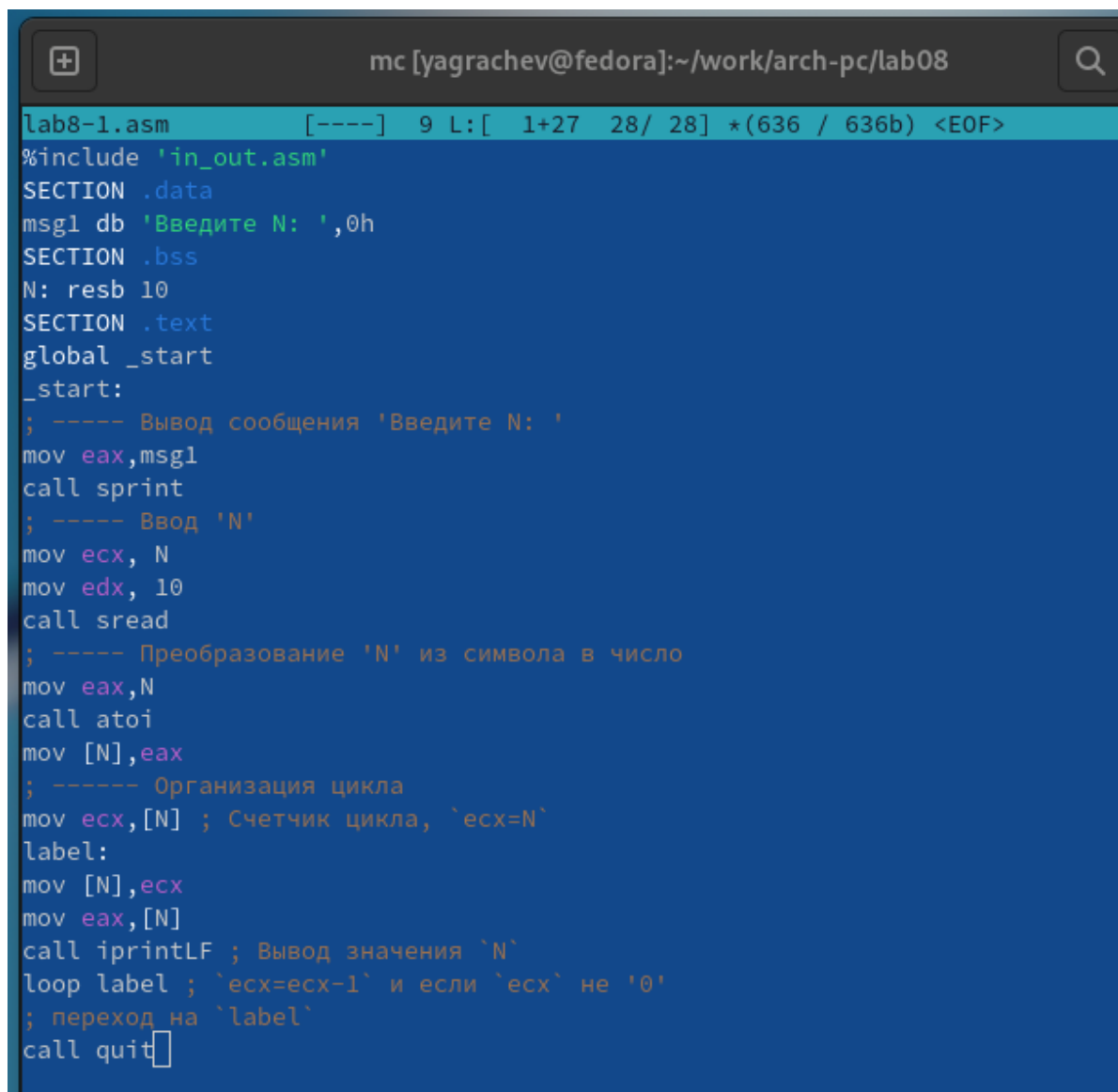


Рис. 2.1: Создан каталог

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`.

Написал в файл lab8-1.asm текст программы из листинга 8.1 (рис. 2.2). Создал исполняемый файл и проверил его работу (рис. 2.3).



```
lab8-1.asm [----] 9 L: [ 1+27 28/ 28] *(636 / 636b) <EOF>
#include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

Рис. 2.2: Программа lab8-1.asm

```
yagrachev@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
yagrachev@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 8
8
7
6
5
4
3
2
1
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
yagrachev@fedora:~/work/arch-pc/lab08$
```

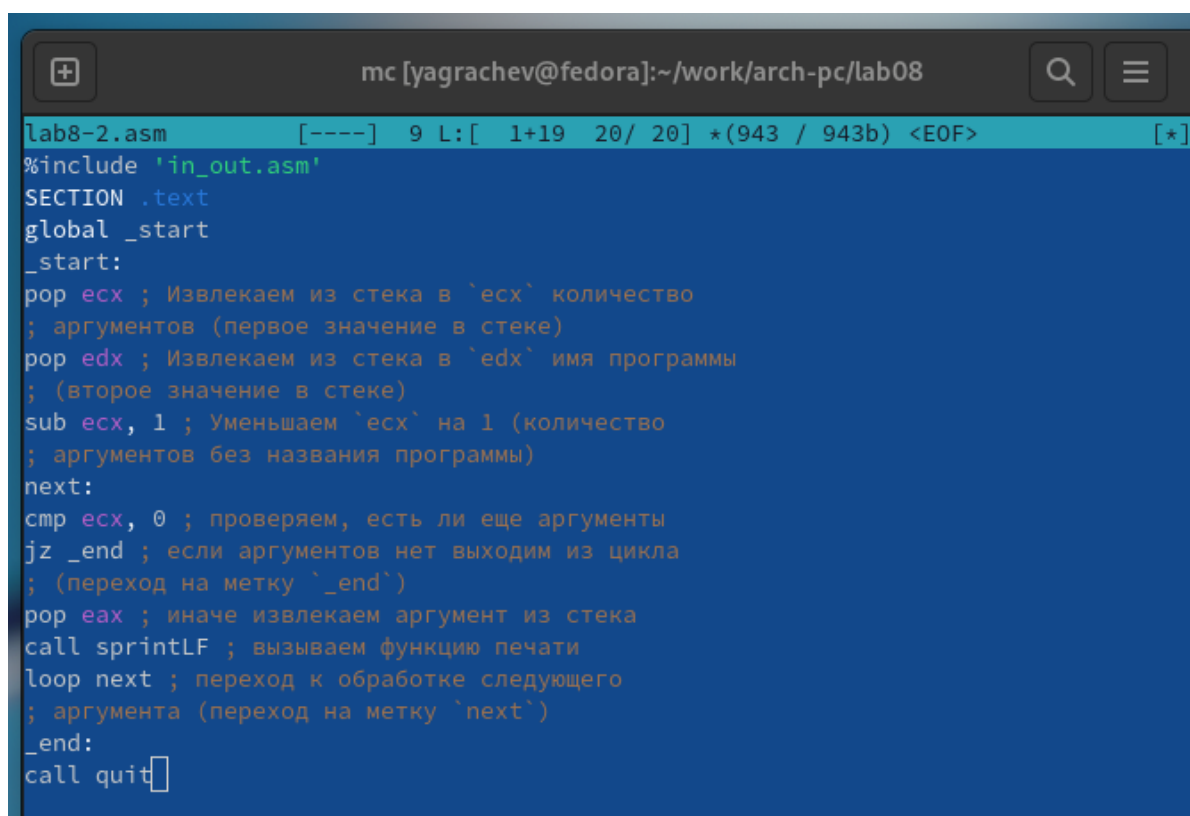
Рис. 2.3: Запуск программы lab8-1.asm

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Изменил текст программы, добавив изменение значения регистра `ecx` в цикле (рис. 2.4). Программа запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N` (рис. 2.5).





применить стек. Внес изменения в программу, добавив команды push и pop для сохранения значения счетчика цикла loop (рис. 2.6). Создал исполняемый файл и проверил его работу (рис. 2.7). Программа выводит числа от N-1 до 0, количество проходов цикла соответствует N.



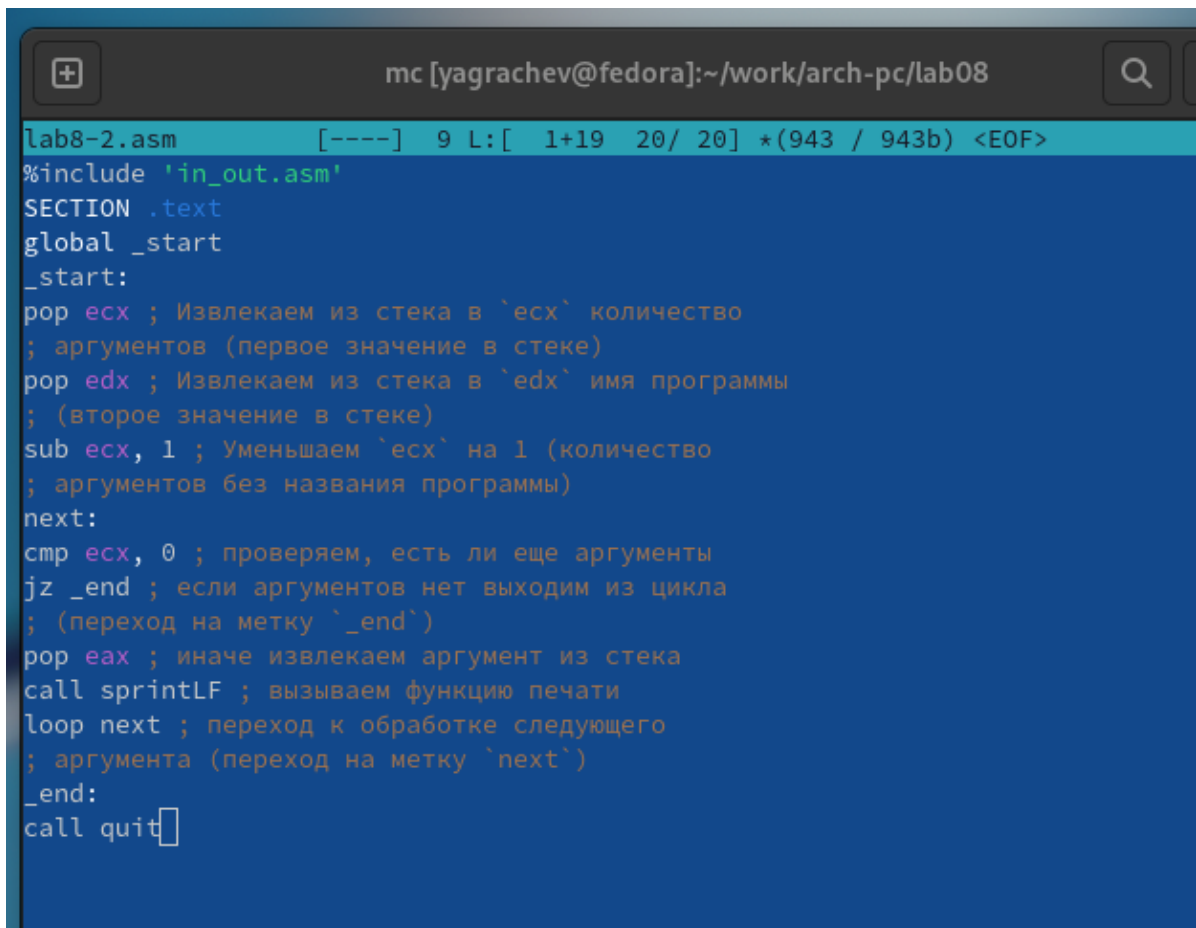
```
lab8-2.asm [----] 9 L:[ 1+19 20/ 20] *(943 / 943b) <EOF> [*]  
%include 'in_out.asm'  
SECTION .text  
global _start  
_start:  
pop ecx ; Извлекаем из стека в `ecx` количество  
; аргументов (первое значение в стеке)  
pop edx ; Извлекаем из стека в `edx` имя программы  
; (второе значение в стеке)  
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество  
; аргументов без названия программы)  
next:  
cmp ecx, 0 ; проверяем, есть ли еще аргументы  
jz _end ; если аргументов нет выходим из цикла  
; (переход на метку `_end`)  
pop eax ; иначе извлекаем аргумент из стека  
call sprintf ; вызываем функцию печати  
loop next ; переход к обработке следующего  
; аргумента (переход на метку `next`)  
_end:  
call quit
```

Рис. 2.6: Программа lab8-1.asm

```
yagrachev@fedora:~/work/arch-pc/lab08$  
yagrachev@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
yagrachev@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1  
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 8  
7  
6  
5  
4  
3  
2  
1  
0  
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 3  
2  
1  
0  
yagrachev@fedora:~/work/arch-pc/lab08$
```

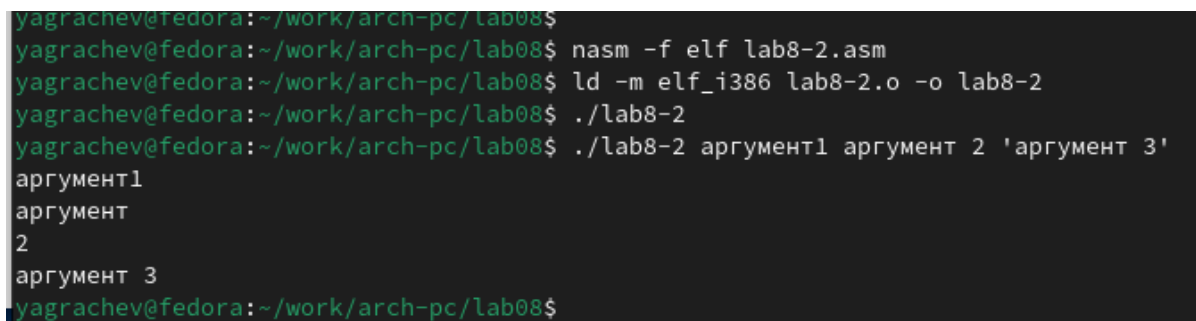
Рис. 2.7: Запуск программы lab8-1.asm

Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и написал в него текст программы из листинга 8.2 (рис. 2.8). Скомпилировал исполняемый файл и запустил его, указав аргументы. Программа обработала 4 аргумента. Аргументами считаются слова или числа, разделенные пробелом (рис. 2.9).



```
lab8-2.asm [----] 9 L: [ 1+19 20/ 20] *(943 / 943b) <EOF>
#include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

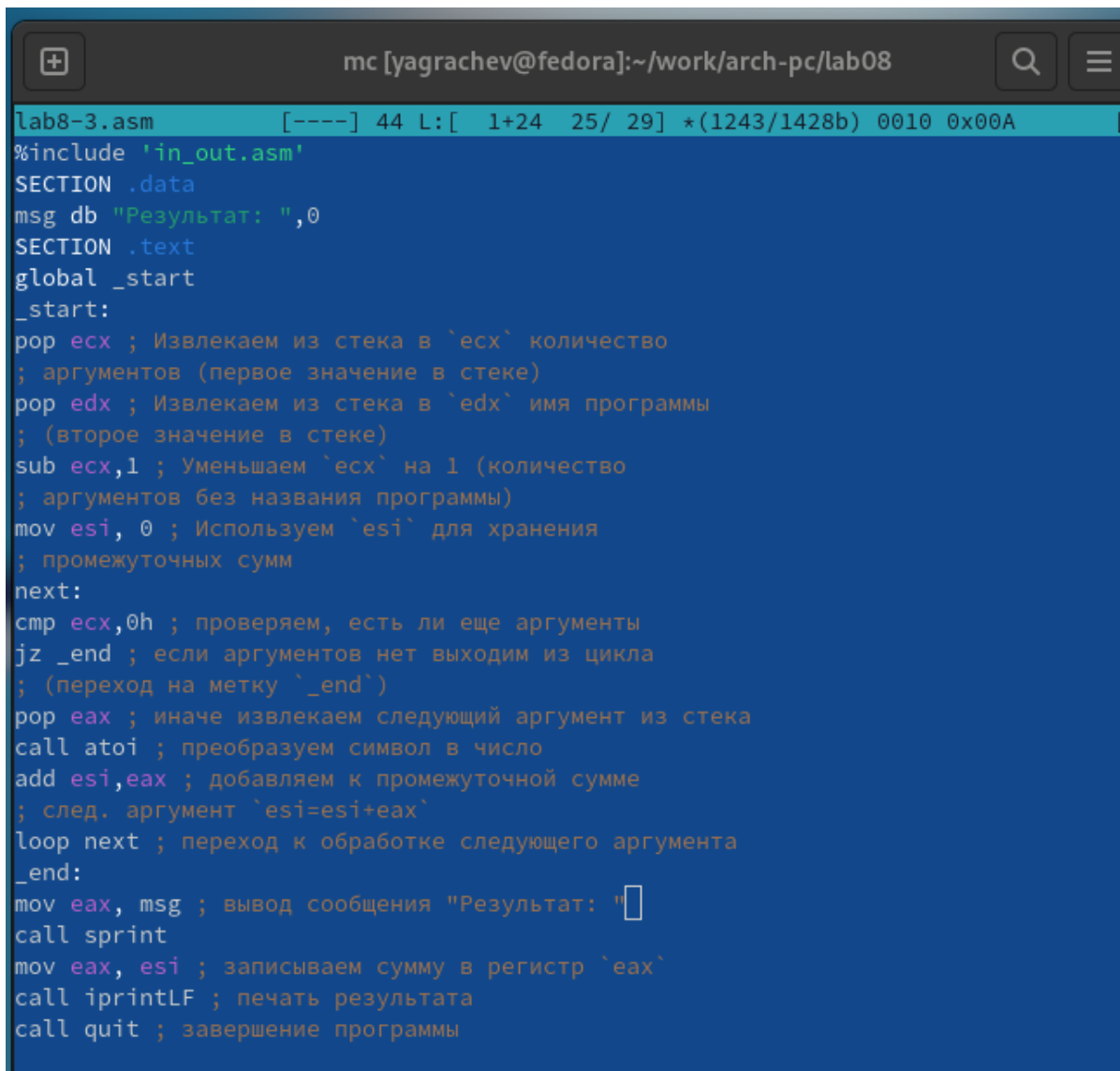
Рис. 2.8: Программа lab8-2.asm



```
yagrachev@fedora:~/work/arch-pc/lab08$
yagrachev@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
yagrachev@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-2
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
yagrachev@fedora:~/work/arch-pc/lab08$
```

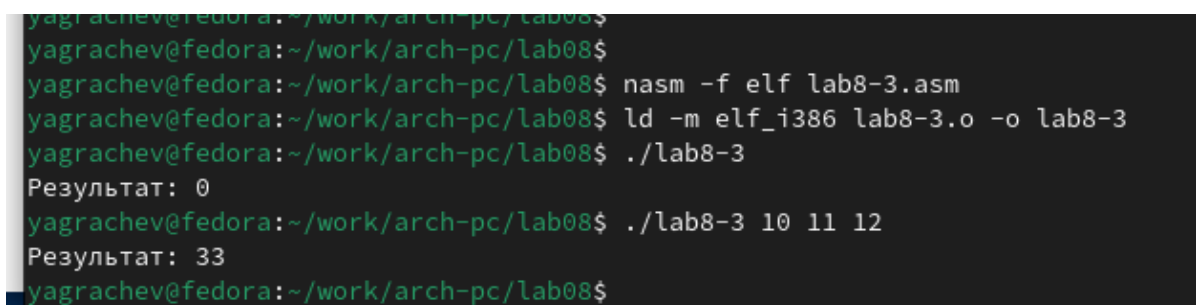
Рис. 2.9: Запуск программы lab8-2.asm

Рассмотрим еще один пример программы, которая выводит сумму чисел, передаваемых в программу как аргументы (рис. 2.10, рис. 2.11).



```
lab8-3.asm [----] 44 L: [ 1+24 25/ 29] *(1243/1428b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

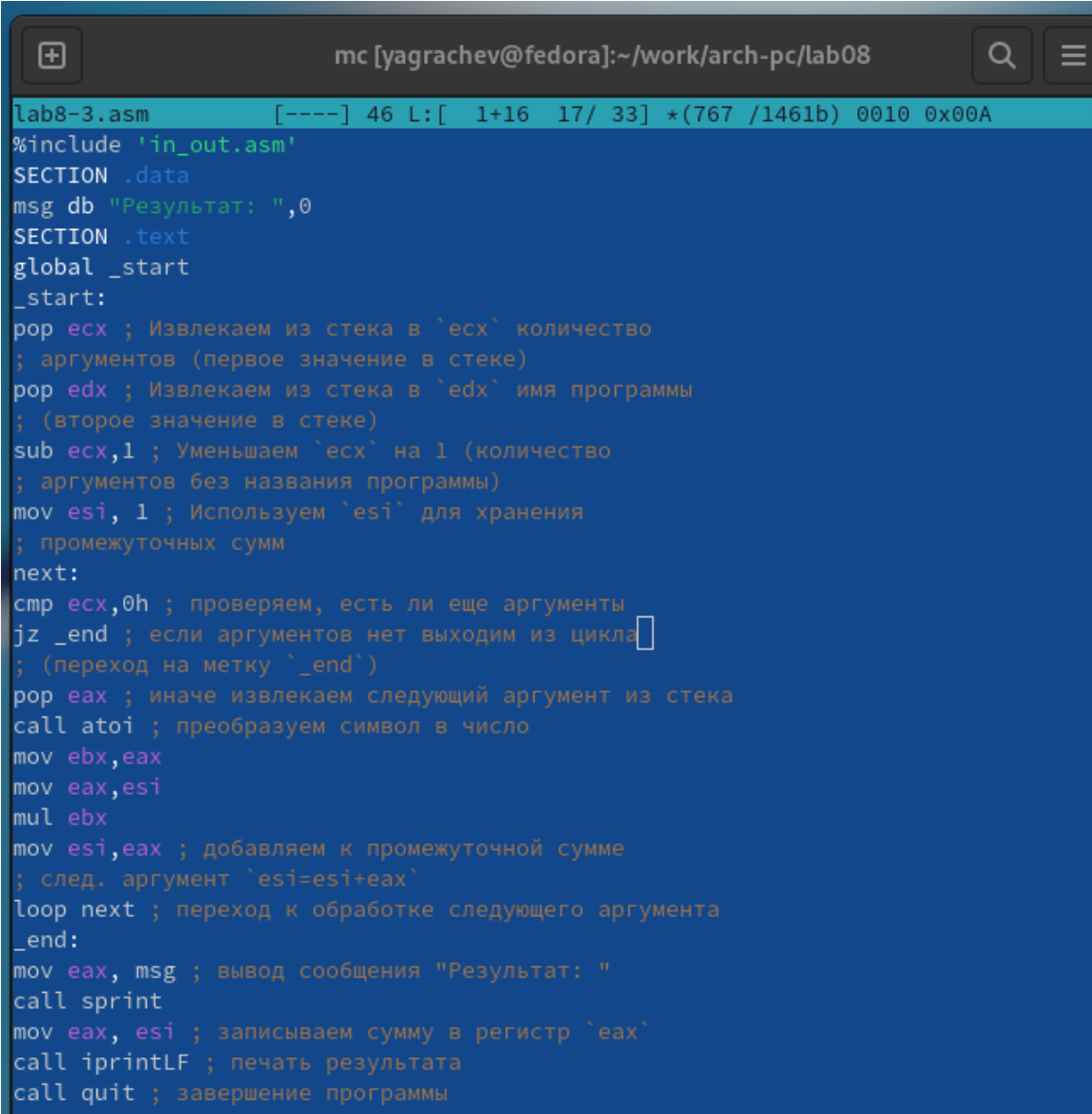
Рис. 2.10: Программа lab8-3.asm



```
yagrachev@fedora:~/work/arch-pc/lab08$
yagrachev@fedora:~/work/arch-pc/lab08$
yagrachev@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
yagrachev@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-3 10 11 12
Результат: 33
yagrachev@fedora:~/work/arch-pc/lab08$
```

Рис. 2.11: Запуск программы lab8-3.asm

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки (рис. 2.12, рис. 2.13).



```
lab8-3.asm [----] 46 L: [ 1+16 17/ 33] *(767 /1461b) 0010 0x00A [
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.12: Программа lab8-3.asm

```
yagrachev@fedora: ~/work/arch-pc/lab08$ cd ~/elf_i386/lab8-3.0 -o lab8-3
yagrachev@fedora: ~/work/arch-pc/lab08$
yagrachev@fedora: ~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
yagrachev@fedora: ~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
yagrachev@fedora: ~/work/arch-pc/lab08$ ./lab8-3 10 11 12
Результат: 1320
yagrachev@fedora: ~/work/arch-pc/lab08$ ./lab8-3
Результат: 1
yagrachev@fedora: ~/work/arch-pc/lab08$
```

Рис. 2.13: Запуск программы lab8-3.asm

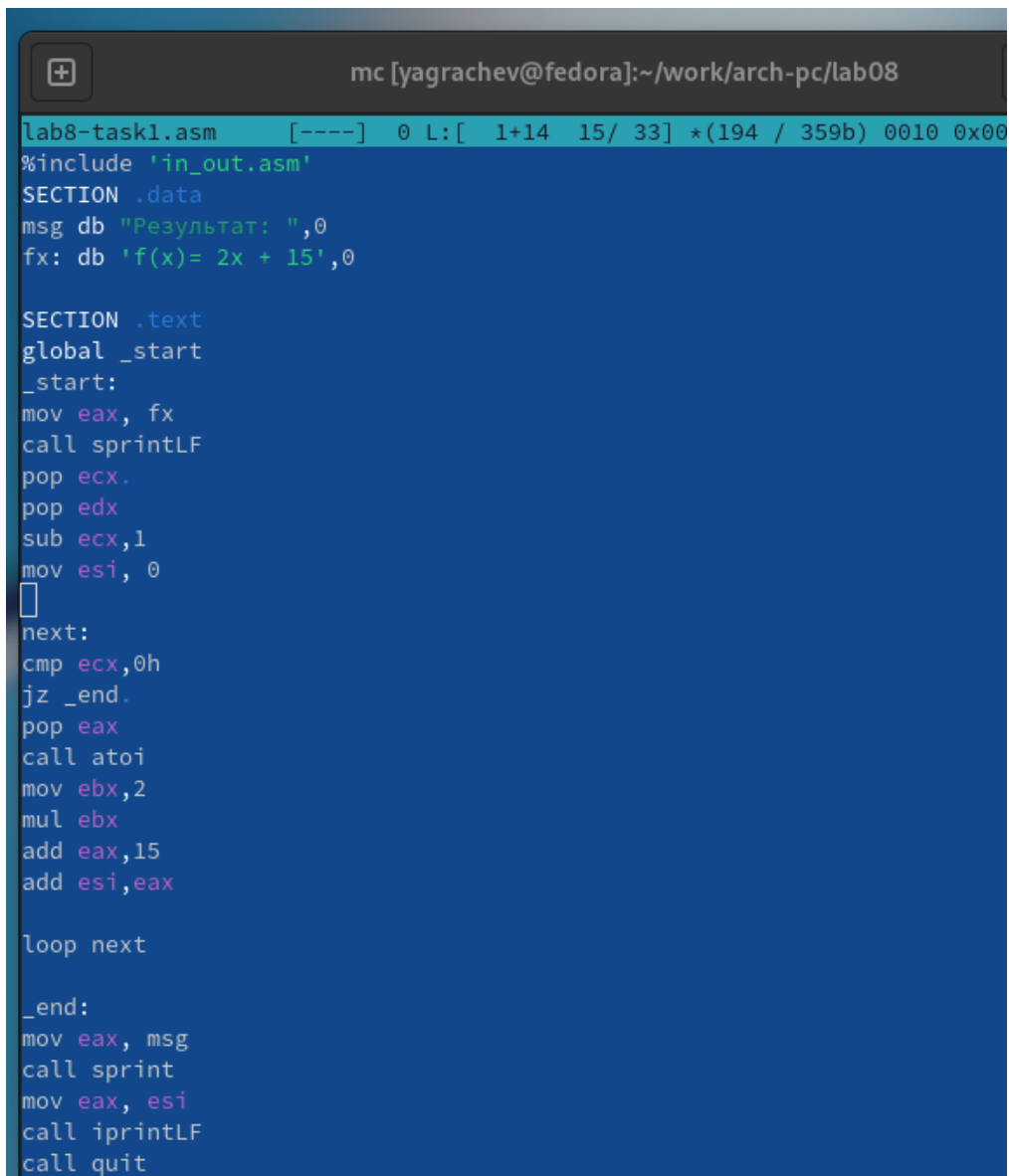
## 2.2 Самостоятельное задание

Написал 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  $x$  передаются как аргументы. Вид функции  $f(x)$  выбирается в соответствии с вариантом, полученным при выполнении лабораторной работы № 7.

Создал исполняемый файл и проверил его работу на нескольких наборах  $x$  (рис. 2.14, рис. 2.15).

Для варианта 1:

$$f(x) = 2x + 15$$



```
lab8-task1.asm  [----]  0 L: [ 1+14 15/ 33] *(194 / 359b) 0010 0x00
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)= 2x + 15',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,2
mul ebx
add eax,15
add esi,eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 2.14: Программа lab8-task1.asm



```
yagrachev@fedora:~/work/arch-pc/lab08$  
yagrachev@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-task1.asm  
yagrachev@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-task1.o -o lab8-task1  
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-task1 0  
f(x)= 2x + 15  
Результат: 15  
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-task1 1  
f(x)= 2x + 15  
Результат: 17  
yagrachev@fedora:~/work/arch-pc/lab08$ ./lab8-task1 10 11 12  
f(x)= 2x + 15  
Результат: 111  
yagrachev@fedora:~/work/arch-pc/lab08$
```

Рис. 2.15: Запуск программы lab8-task1.asm

Убедился, что программа считает правильно  $f(0) = 15$ ,  $f(1) = 17$ .

## 3 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `naasm`.