

# **Отчёт по лабораторной работе 8**

**Архитектура компьютера**

Саид Курбанов

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Неализация циклов в NASM . . . . .	6
2.2	Обработка аргументов командной строки . . . . .	12
2.3	Задание для самостоятельной работы . . . . .	16
<b>3</b>	<b>Выводы</b>	<b>19</b>

## Список иллюстраций

2.1	Программа в файле lab8-1.asm . . . . .	7
2.2	Запуск программы lab8-1.asm . . . . .	8
2.3	Программа в файле lab8-1.asm . . . . .	9
2.4	Запуск программы lab8-1.asm . . . . .	10
2.5	Программа в файле lab8-1.asm . . . . .	11
2.6	Запуск программы lab8-1.asm . . . . .	12
2.7	Программа в файле lab8-2.asm . . . . .	13
2.8	Запуск программы lab8-2.asm . . . . .	13
2.9	Программа в файле lab8-3.asm . . . . .	14
2.10	Запуск программы lab8-3.asm . . . . .	14
2.11	Программа в файле lab8-3.asm . . . . .	15
2.12	Запуск программы lab8-3.asm . . . . .	16
2.13	Программа в файле task.asm . . . . .	17
2.14	Запуск программы task.asm . . . . .	18

## **Список таблиц**

# 1 Цель работы

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

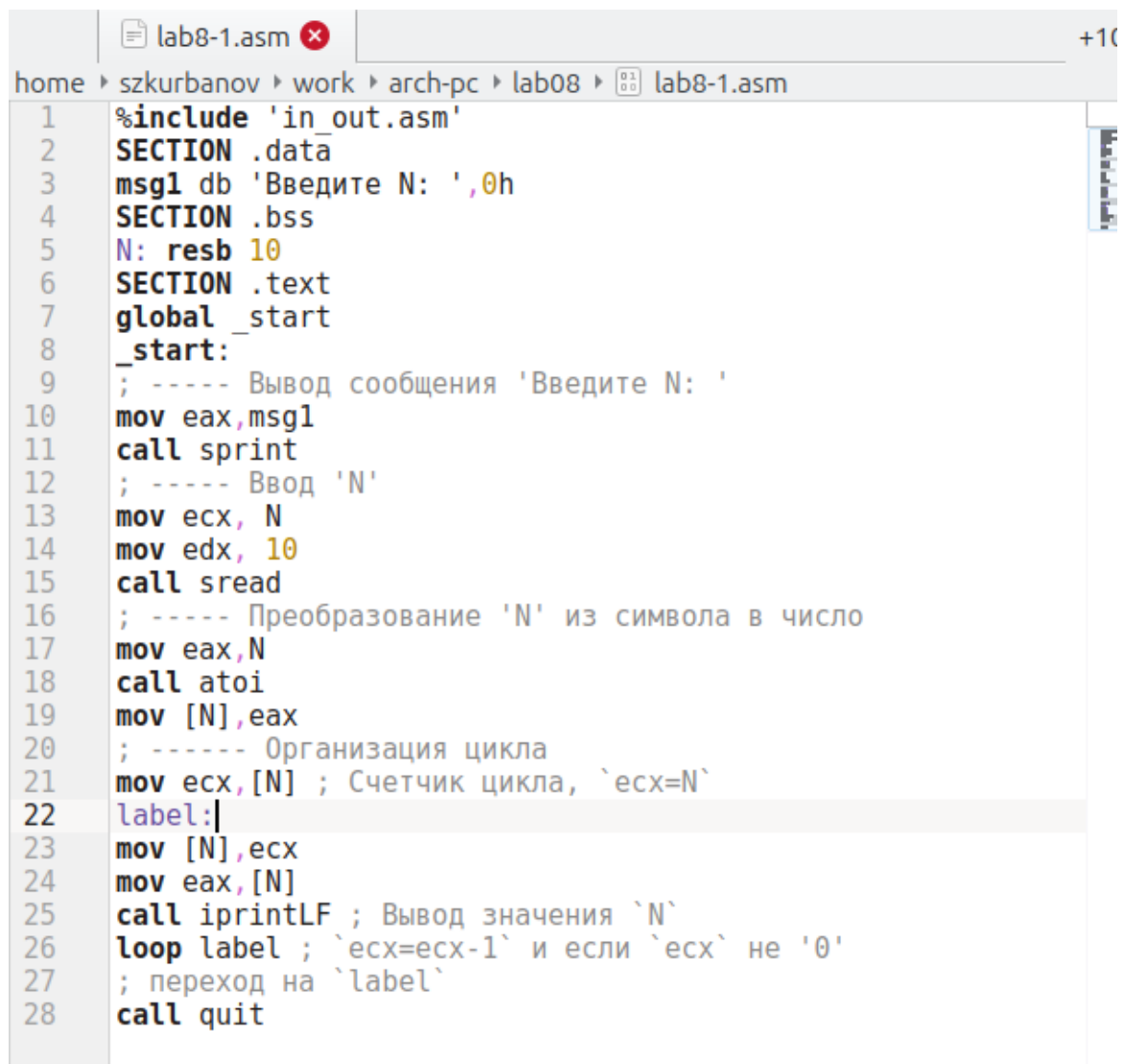
## 2 Выполнение лабораторной работы

### 2.1 Неализация циклов в NASM

Создал каталог для программ лабораторной работы № 8 и файл lab8-1.asm

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. Создал исполняемый файл и проверил его работу.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:|
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения `N`
26 loop label ; `ecx=ecx-1` и если `ecx` не '0'
27 ; переход на `label`
28 call quit
```

Рисунок 2.1: Программа в файле lab8-1.asm

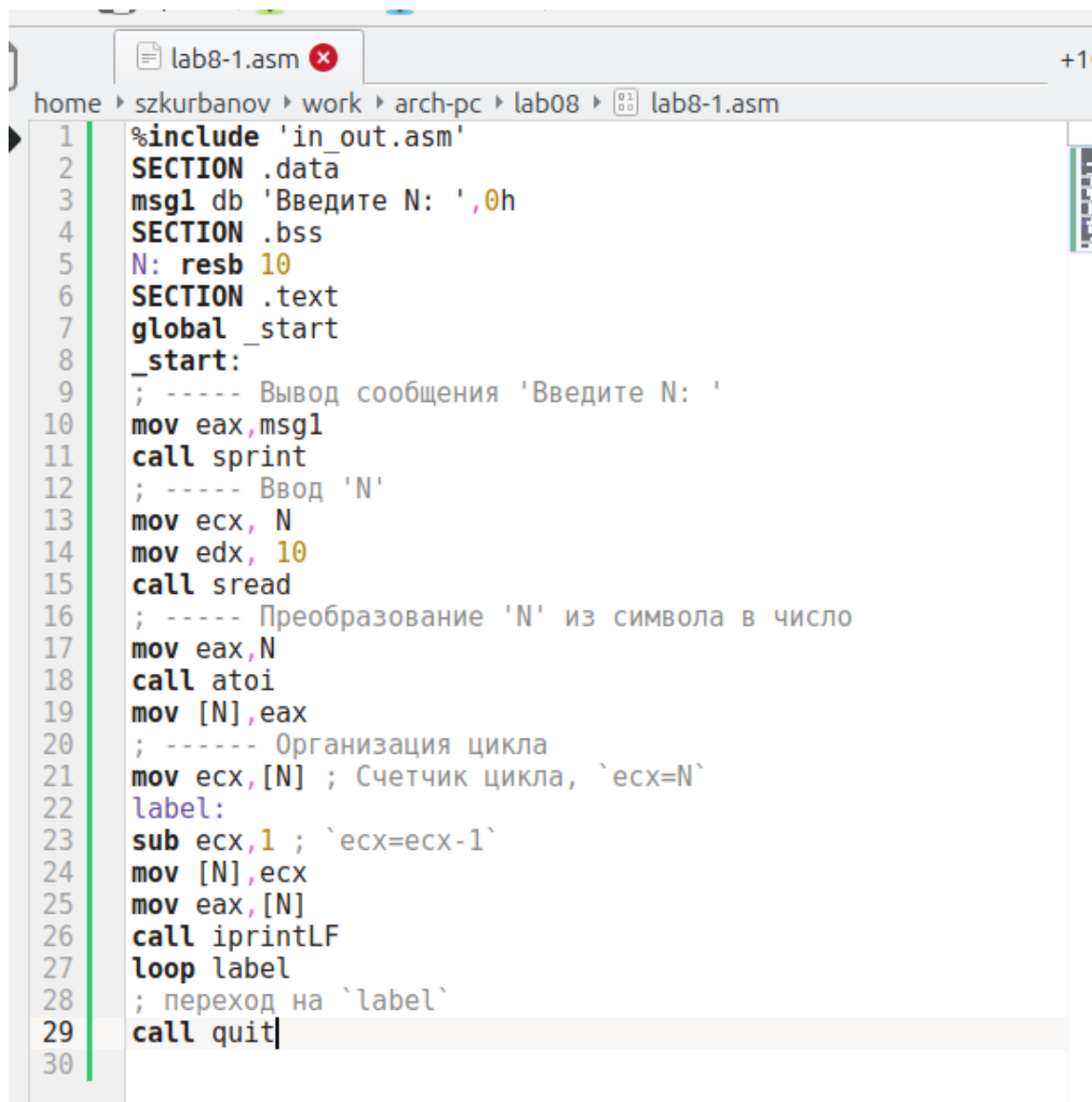
```
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
2
1
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.2: 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 значение регистра `ecx` в цикле.

Программа запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N`.





```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
28 ; переход на `label`
29 call quit
30
```

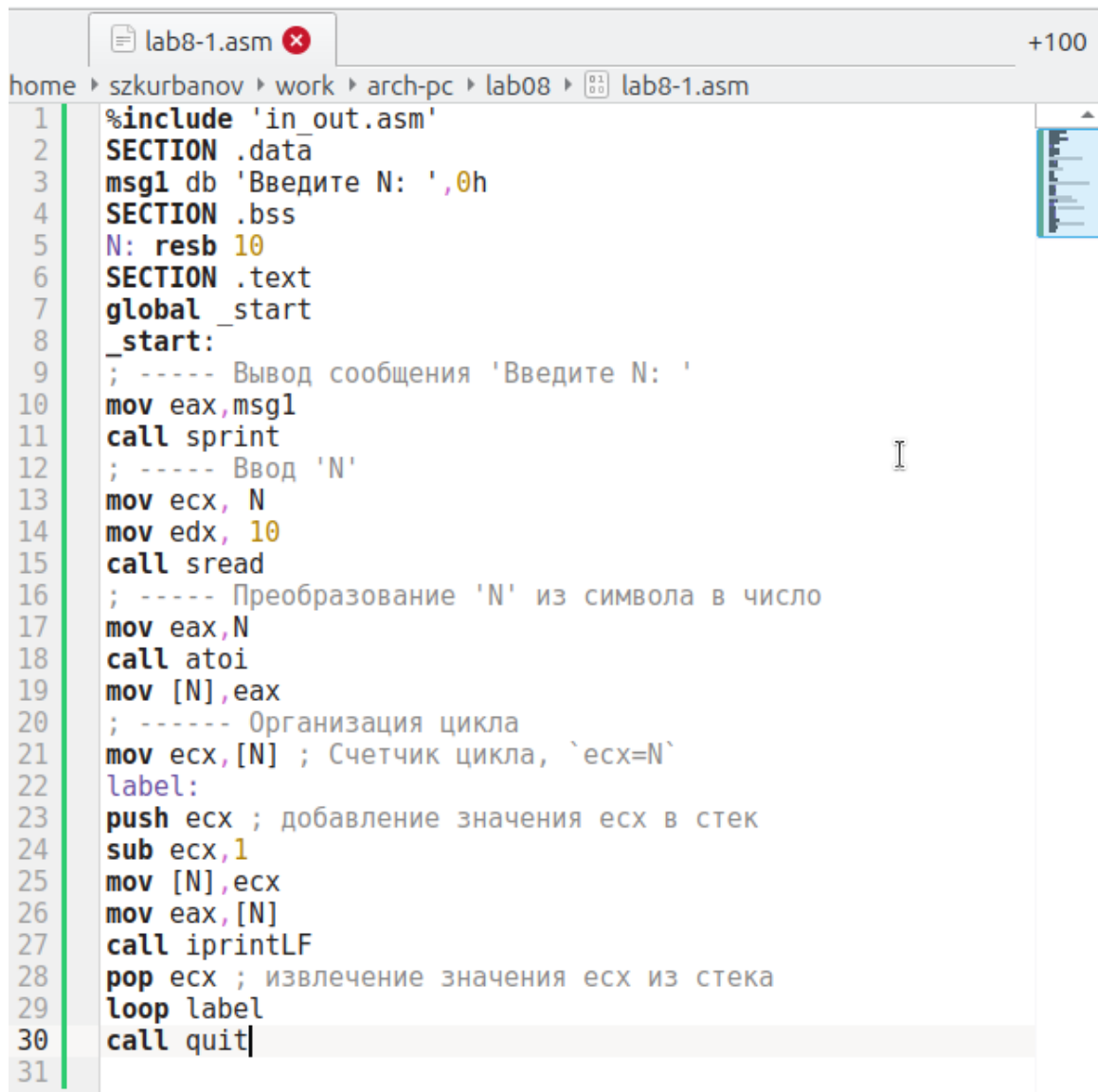
Рисунок 2.3: Программа в файле lab8-1.asm

```
4294796808
4294796806
4294796804
4294796802
4294796800
4294796798
4294796796
4294796794
4294796792
4294796790
42947^C
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.4: Запуск программы lab8-1.asm

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внес изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`. Создал исполняемый файл и проверьте его работу.

Программа выводит числа от  $N-1$  до 0, число проходов цикла соответствует  $N$ .



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 push ecx ; добавление значения ecx в стек
24 sub ecx,1
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx ; извлечение значения ecx из стека
29 loop label
30 call quit
31
```

Рисунок 2.5: Программа в файле lab8-1.asm

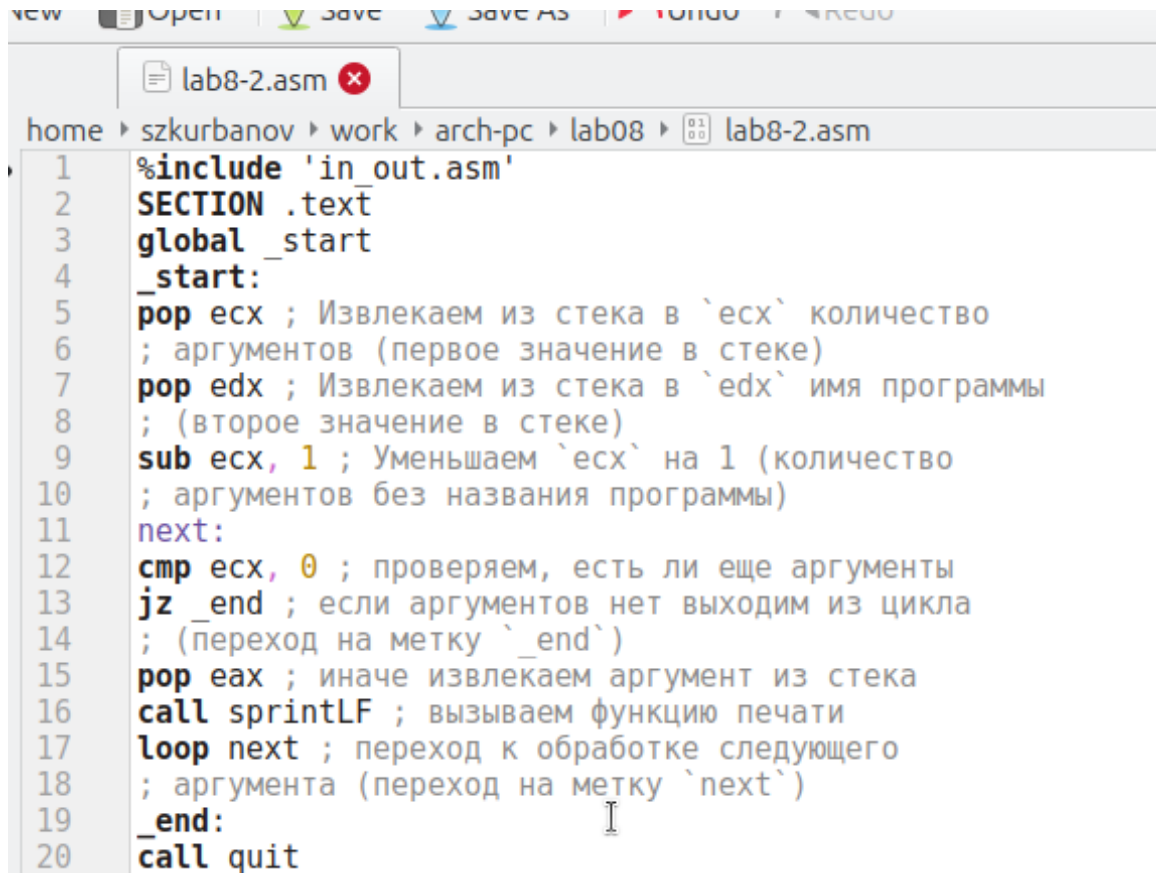
```
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
0
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.6: Запуск программы lab8-1.asm

## 2.2 Обработка аргументов командной строки

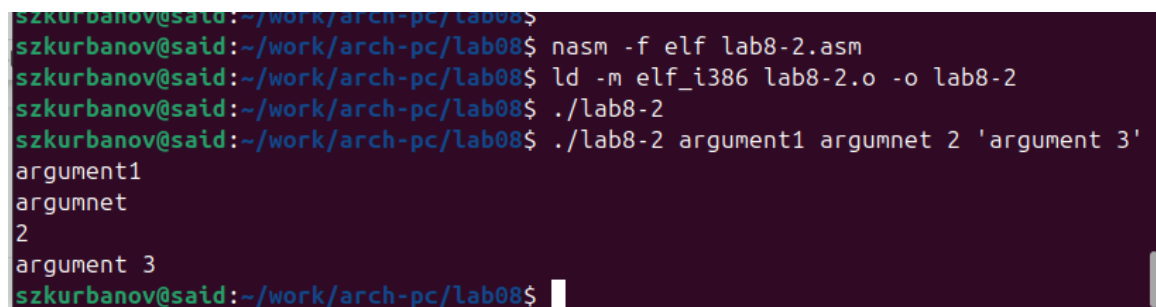
Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввел в него текст программы из листинга 8.2.

Создал исполняемый файл и запустил его, указав аргументы. Программа обработала 4 аргумента. Аргументами считаются слова/числа, разделенные пробелом.



```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

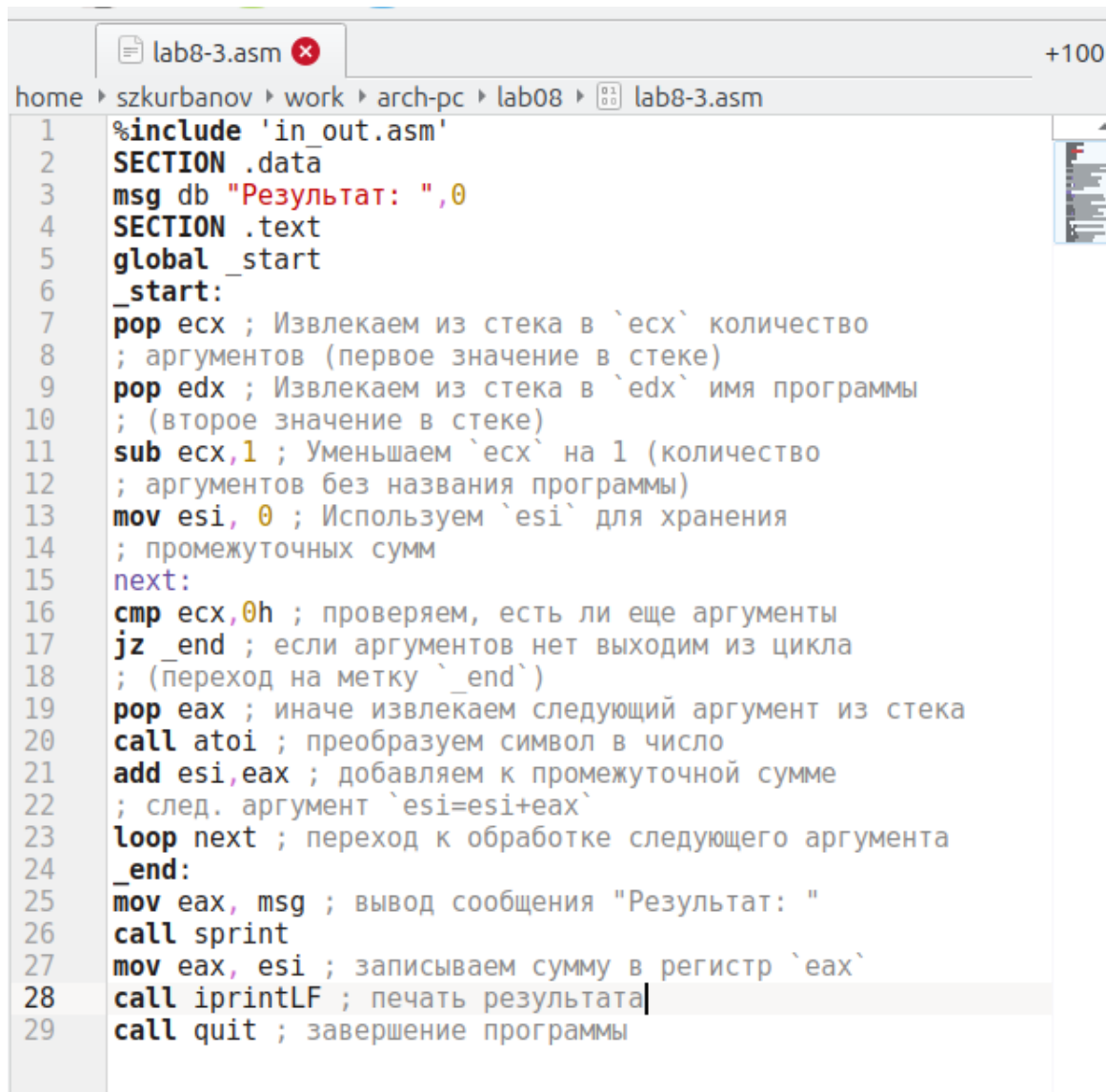
Рисунок 2.7: Программа в файле lab8-2.asm



```
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-2
argument1
argument2
2
argument 3
szkurbanov@said:~/work/arch-pc/lab08$
```

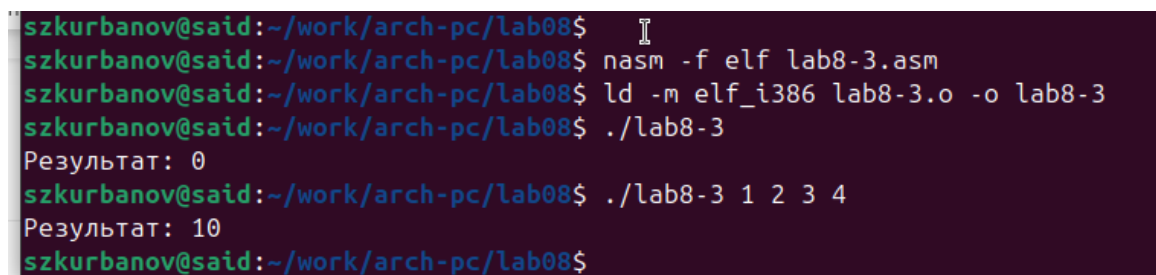
Рисунок 2.8: Запуск программы lab8-2.asm

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы.



```
lab8-3.asm
home > szkurbanov > work > arch-pc > lab08 > lab8-3.asm
1  %include 'in_out.asm'
2  SECTION .data
3  msg db "Результат: ",0
4  SECTION .text
5  global _start
6  _start:
7  pop ecx ; Извлекаем из стека в `ecx` количество
8           ; аргументов (первое значение в стеке)
9  pop edx ; Извлекаем из стека в `edx` имя программы
10          ; (второе значение в стеке)
11  sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12            ; аргументов без названия программы)
13  mov esi,0 ; Используем `esi` для хранения
14            ; промежуточных сумм
15  next:
16  cmp ecx,0h ; проверяем, есть ли еще аргументы
17  jz _end ; если аргументов нет выходим из цикла
18          ; (переход на метку `_end`)
19  pop eax ; иначе извлекаем следующий аргумент из стека
20  call atoi ; преобразуем символ в число
21  add esi,eax ; добавляем к промежуточной сумме
22            ; след. аргумент `esi=esi+eax`
23  loop next ; переход к обработке следующего аргумента
24  _end:
25  mov eax,msg ; вывод сообщения "Результат: "
26  call sprint
27  mov eax,esi ; записываем сумму в регистр `eax`
28  call iprintLF ; печать результата
29  call quit ; завершение программы
```

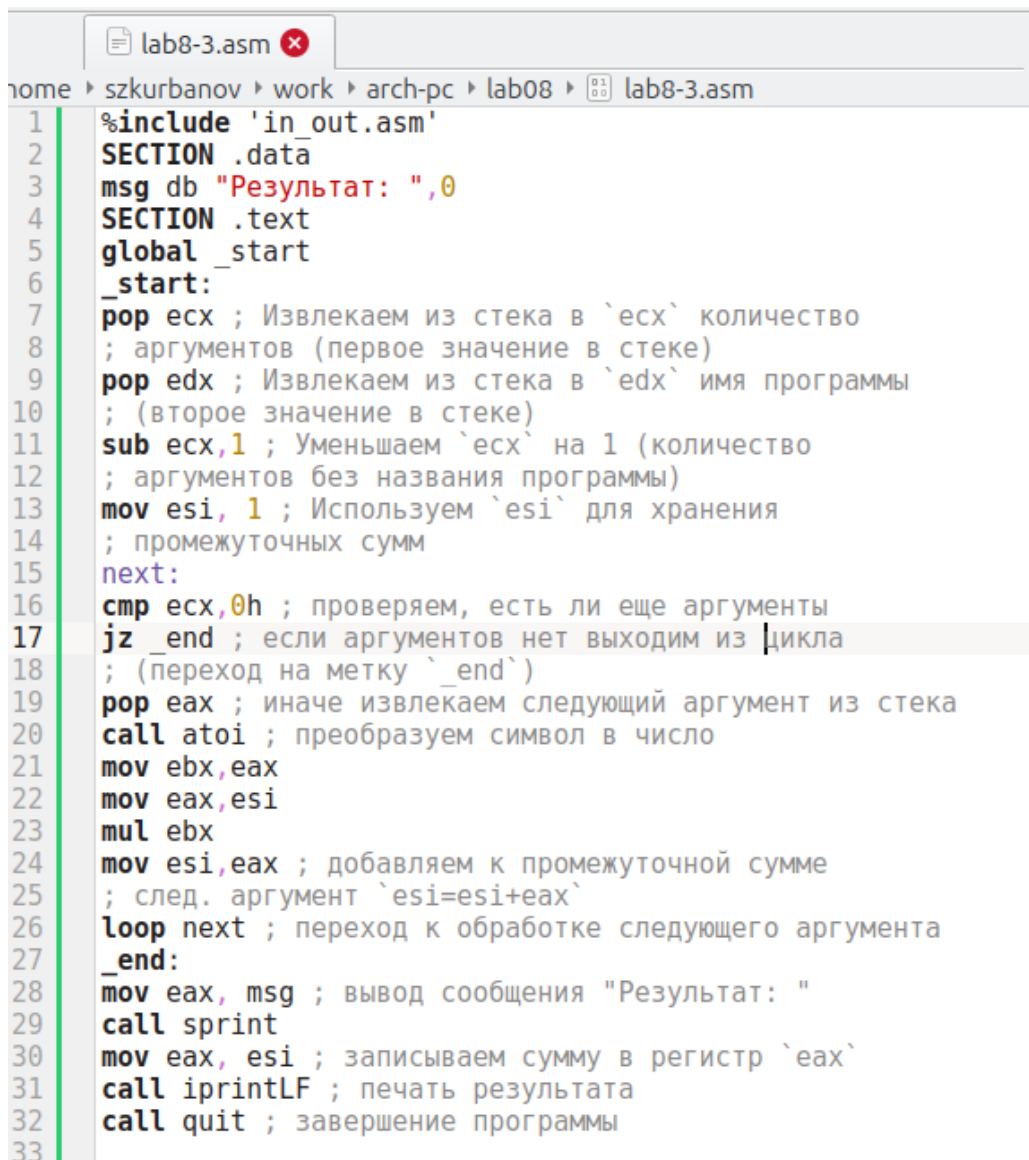
Рисунок 2.9: Программа в файле lab8-3.asm



```
szkurbanov@said:~/work/arch-pc/lab08$
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 10
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.10: Запуск программы lab8-3.asm

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi,1 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mov ebx,eax
22 mov eax,esi
23 mul ebx
24 mov esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент `esi=esi+eax`
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax,msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax,esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программы
33
```

Рисунок 2.11: Программа в файле lab8-3.asm

```
szkurbanov@said:~/work/arch-pc/lab08$  
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm  
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3  
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-3  
Результат: 1  
szkurbanov@said:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4  
Результат: 24  
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.12: Запуск программы lab8-3.asm

## 2.3 Задание для самостоятельной работы

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  $x$  передаются как аргументы. Вид функции  $f(x)$  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  $x$ .

для варианта 1  $f(x) = 2x + 15$



```
task.asm
home ▸ szkurbanov ▸ work ▸ arch-pc ▸ lab08 ▸ task.asm
1  %include 'in_out.asm'
2  SECTION .data
3  msg db "Результат: ",0
4  fx: db 'f(x)= 2x + 15',0
5
6  SECTION .text
7  global _start
8  _start:
9  mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 mov ebx,2
22 mul ebx
23 add eax,15
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprintf
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рисунок 2.13: Программа в файле task.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке  $f(0) = 15$ ,  $f(1) = 17$

Затем подал несколько аргументов и получил сумму значений функции.

```
szkurbanov@said:~/work/arch-pc/lab08$  
szkurbanov@said:~/work/arch-pc/lab08$ nasm -f elf task.asm  
szkurbanov@said:~/work/arch-pc/lab08$ ld -m elf_i386 task.o -o task  
szkurbanov@said:~/work/arch-pc/lab08$ ./task  
f(x)= 2x + 15  
Результат: 0  
szkurbanov@said:~/work/arch-pc/lab08$ ./task 1  
f(x)= 2x + 15  
Результат: 17  
szkurbanov@said:~/work/arch-pc/lab08$ ./task 1 3 4 5 6  
f(x)= 2x + 15  
Результат: 113  
szkurbanov@said:~/work/arch-pc/lab08$
```

Рисунок 2.14: Запуск программы task.asm

## 3 Выводы

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