

# **Отчёт по лабораторной работе 8**

**Архитектура компьютеров и операционные системы**

Абдулфазова Лейла Али гызы

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация циклов в NASM . . . . .	9
4.2	Обработка аргументов командной строки . . . . .	15
4.3	Выполнение заданий для самостоятельной работы . . . . .	19
<b>5</b>	<b>Выводы</b>	<b>22</b>

## Список иллюстраций

4.1	Редактирование файла lab8-1.asm . . . . .	10
4.2	Тестирование программы lab8-1.asm . . . . .	11
4.3	Редактирование файла lab8-1.asm . . . . .	12
4.4	Тестирование программы lab8-1.asm . . . . .	13
4.5	Редактирование файла lab8-1.asm . . . . .	14
4.6	Тестирование программы lab8-1.asm . . . . .	15
4.7	Редактирование файла lab8-2.asm . . . . .	16
4.8	Тестирование программы lab8-2.asm . . . . .	16
4.9	Редактирование файла lab8-3.asm . . . . .	17
4.10	Тестирование программы lab8-3.asm . . . . .	17
4.11	Редактирование файла lab8-3.asm . . . . .	18
4.12	Тестирование программы lab8-3.asm . . . . .	19
4.13	Редактирование файла prog.asm . . . . .	20
4.14	Тестирование программы prog.asm . . . . .	21

## Список таблиц

# 1 Цель работы

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

## 2 Задание

1. Изучение циклов и стека в ассемблере
2. Изучение процесса передачи аргументов командной строки
3. Рассмотрение примеров с циклами и стеком
4. Выполнение заданий для самостоятельной работы

### 3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды.

Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров.

Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.

Для стека существует две основные операции:

- добавление элемента в вершину стека (push)
- извлечение элемента из вершины стека (pop)

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл. Инструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не

выполняется и управление передаётся команде, которая следует сразу после команды loop.



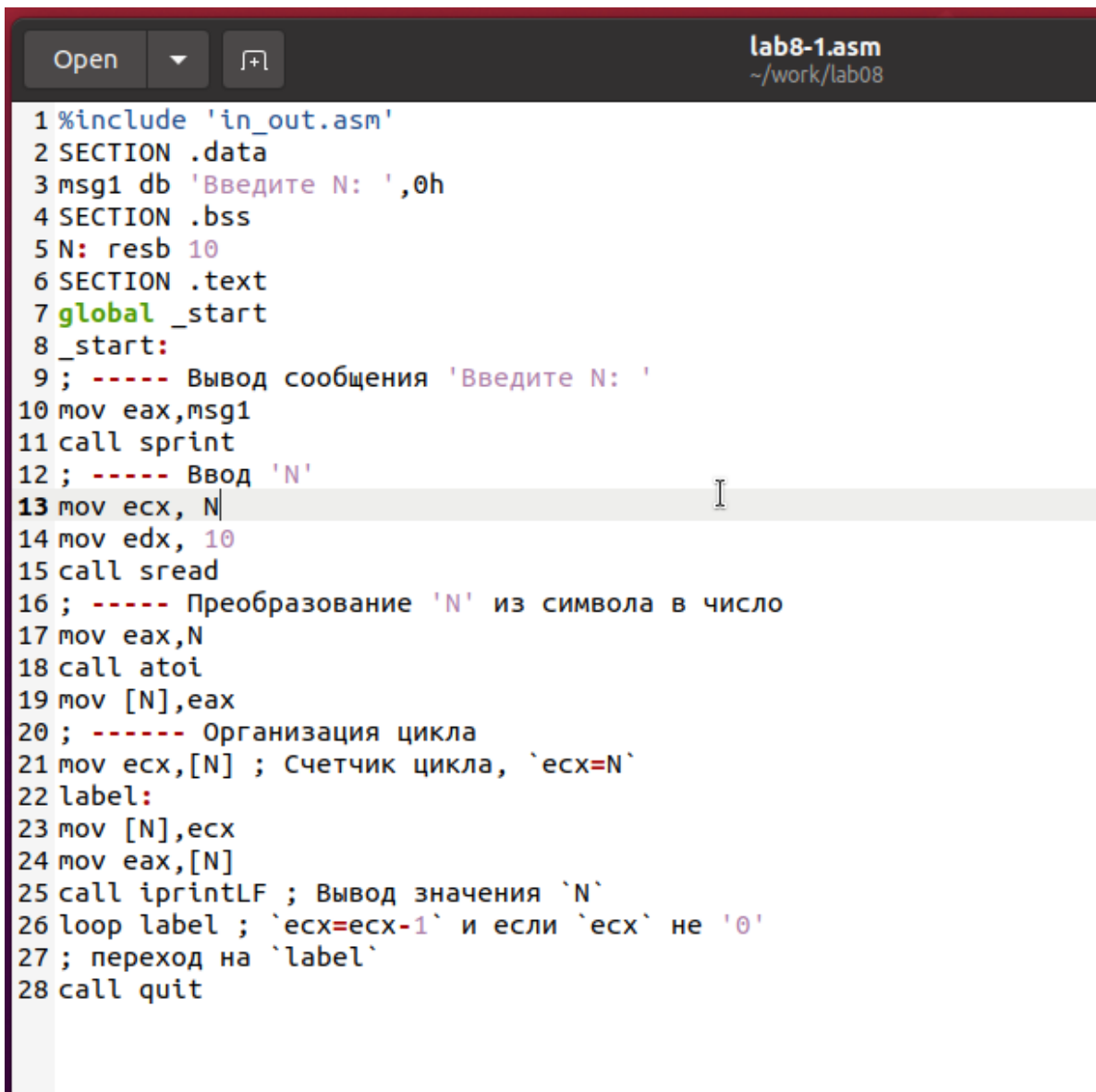
## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

Я создаю папку для выполнения лабораторной работы № 8 и файл с именем lab8-1.asm.

Стоит отметить, что при использовании команды `loop` в NASM для реализации циклов, необходимо помнить, что эта команда использует регистр `ecx` в роли счетчика и на каждом шаге уменьшает его значение на единицу.

Посмотрим на пример программы, которая выводит значение регистра `ecx`. В файл lab8-1.asm я внесла текст программы из листинга 8.1.



```
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
```

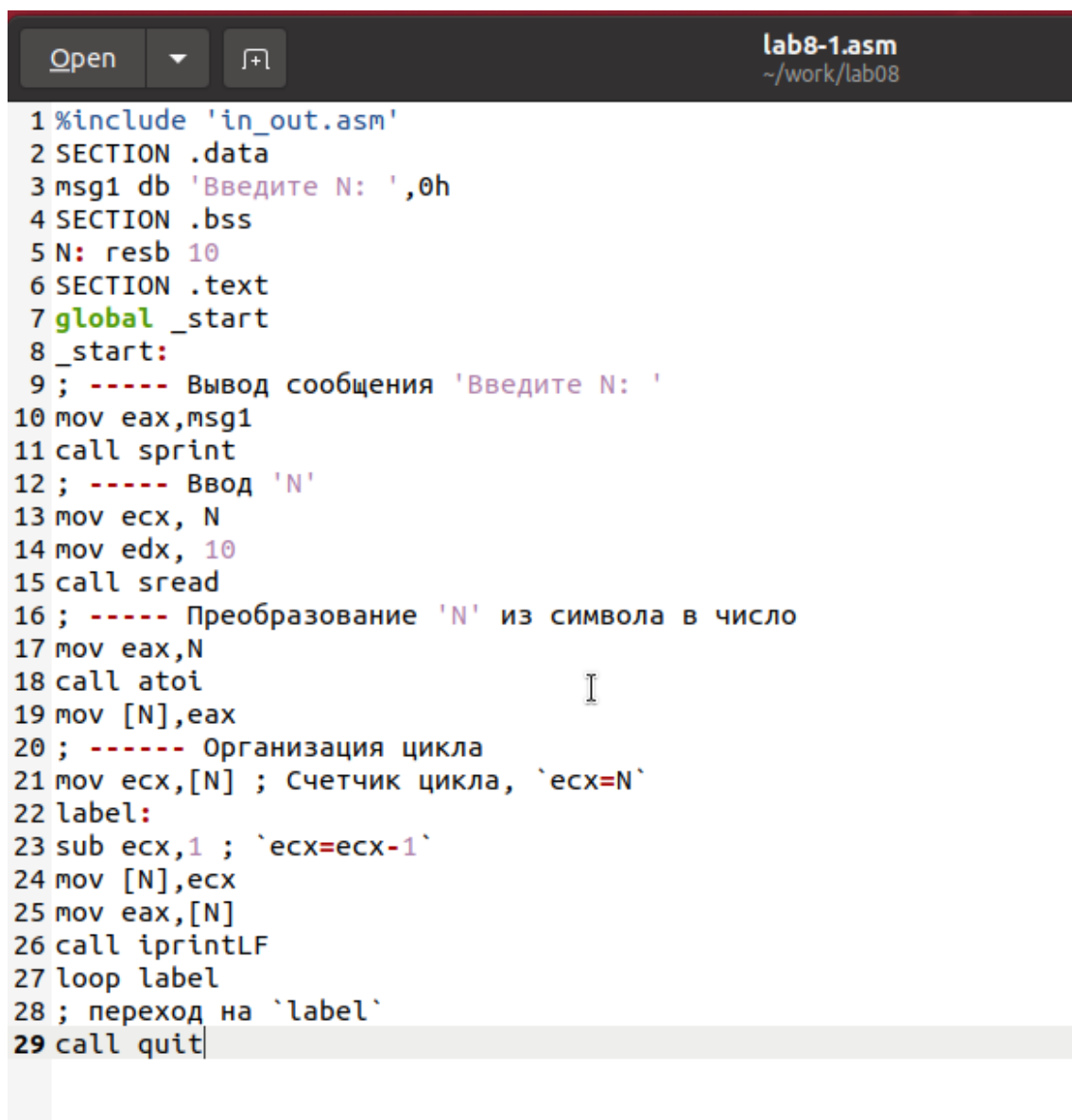
Рис. 4.1: Редактирование файла lab8-1.asm

Далее, я создаю исполняемый файл и проверяю его функционирование.

```
leila@ubuntu:~/work/lab08$ nasm -f elf lab8-1.asm
leila@ubuntu:~/work/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
leila@ubuntu:~/work/lab08$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
leila@ubuntu:~/work/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
leila@ubuntu:~/work/lab08$
```

Рис. 4.2: Тестирование программы lab8-1.asm

В этом примере показано, что использование регистра `ecx` в команде `loop` может привести к неправильному выполнению программы. В текст программы я вношу изменения, которые включают в себя изменение значения регистра `ecx` внутри цикла.



```
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
```

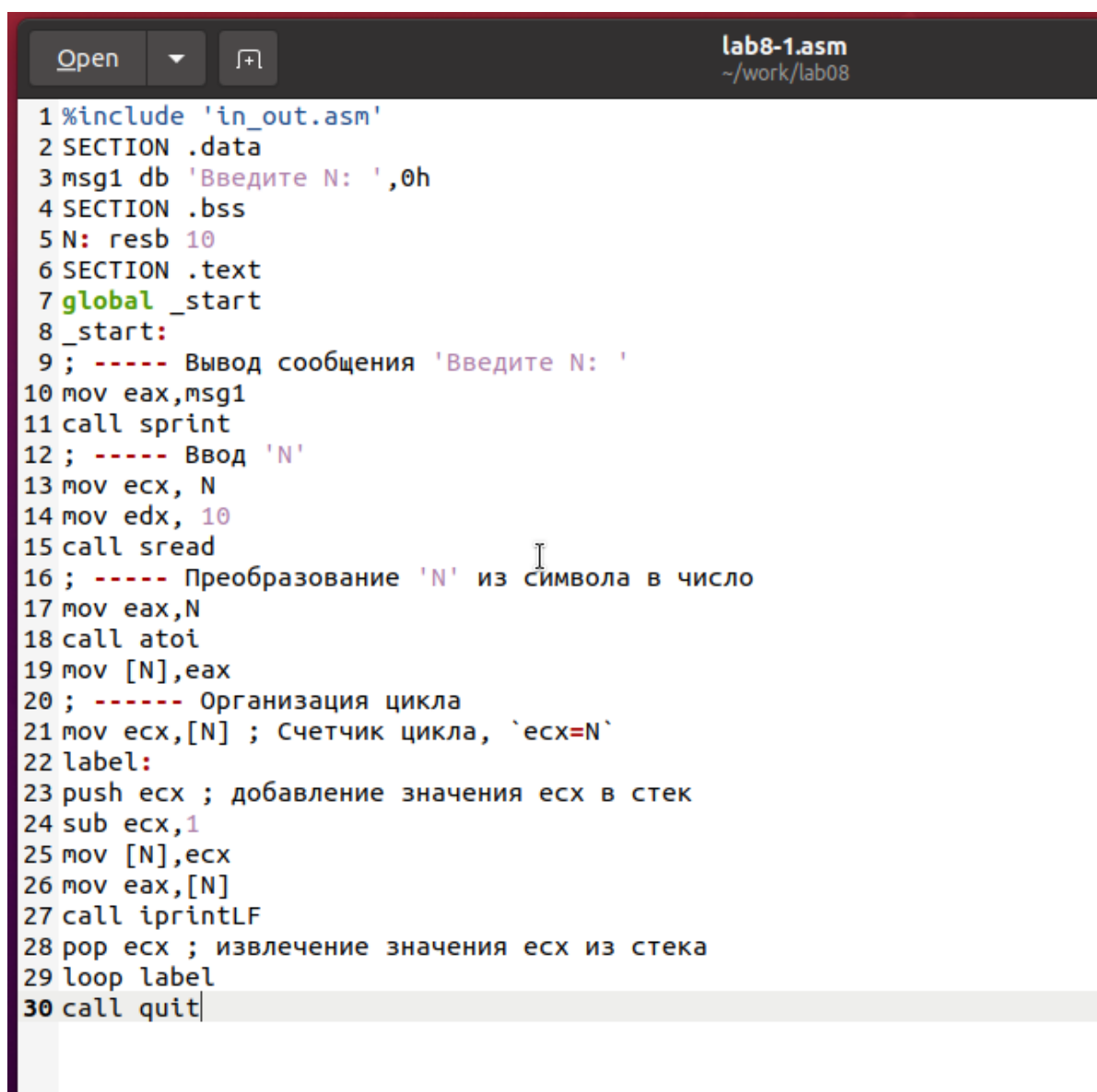
Рис. 4.3: Редактирование файла lab8-1.asm

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

```
4294915728
4294915726
4294915724
4294915722
4294915720
4294915718
4294915716
4294915714
429491571^C
leila@ubuntu:~/work/lab08$ ./lab8-1
Введите N: 6
5
3
1
leila@ubuntu:~/work/lab08$
```

Рис. 4.4: Тестирование программы lab8-1.asm

Чтобы использовать регистр `ecx` в цикле и обеспечить правильную работу программы, используется стек. Я внесла изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счётчика цикла `loop` в стеке.



```
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
```

Рис. 4.5: Редактирование файла lab8-1.asm

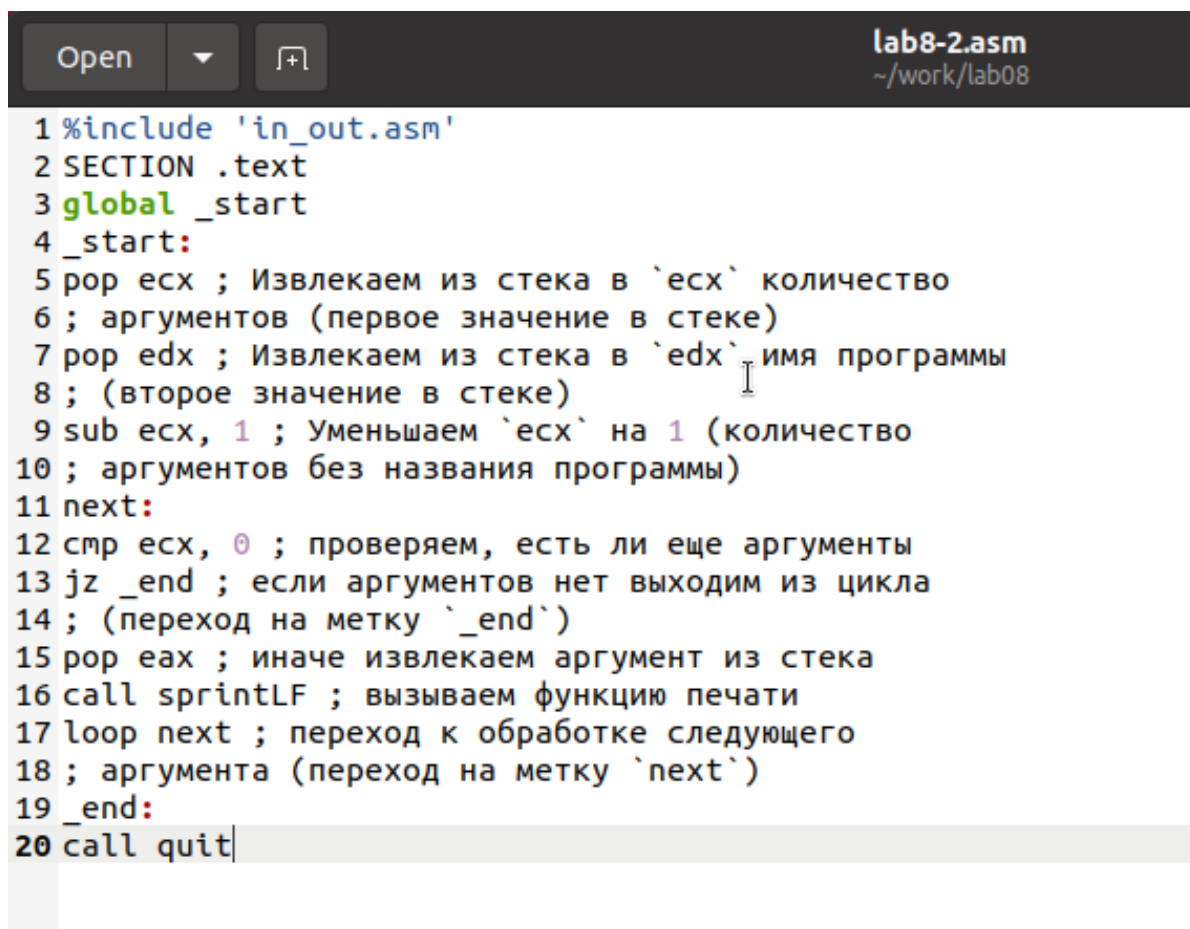
Затем был создан исполняемый файл и проверена его работа. Программа выводит числа от  $N-1$  до 0, где количество проходов цикла соответствует значению  $N$ .

```
leila@ubuntu:~/work/lab08$  
leila@ubuntu:~/work/lab08$ nasm -f elf lab8-1.asm  
leila@ubuntu:~/work/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1  
leila@ubuntu:~/work/lab08$ ./lab8-1  
Введите N: 6  
5  
4  
3  
2  
1  
0  
leila@ubuntu:~/work/lab08$ ./lab8-1  
Введите N: 5  
4  
3  
2  
1  
0  
leila@ubuntu:~/work/lab08$
```

Рис. 4.6: Тестирование программы lab8-1.asm

## 4.2 Обработка аргументов командной строки

Я изучила файл lab8-2.asm, в который внесла код программы из листинга 8.2.

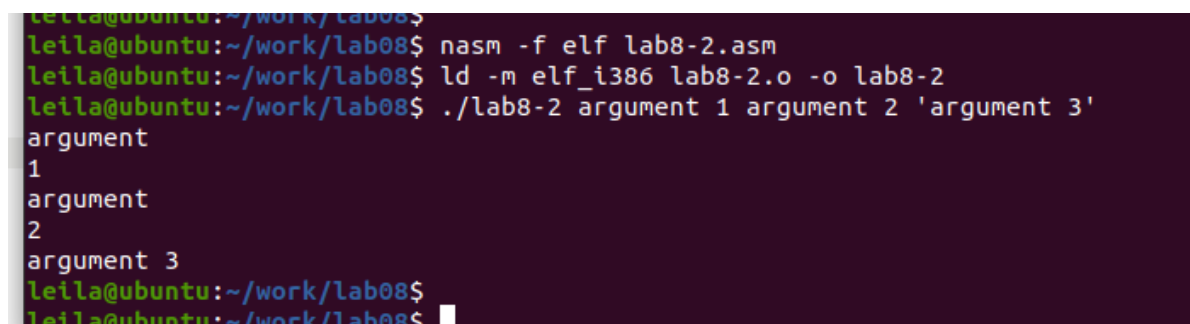


```
lab8-2.asm
~/work/lab08

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 sprintfLF ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 4.7: Редактирование файла lab8-2.asm

После этого был создан исполняемый файл, который я запустила с указанными аргументами. Программа эффективно обработала пять аргументов, которые представляют собой слова или числа, разделенные пробелами.



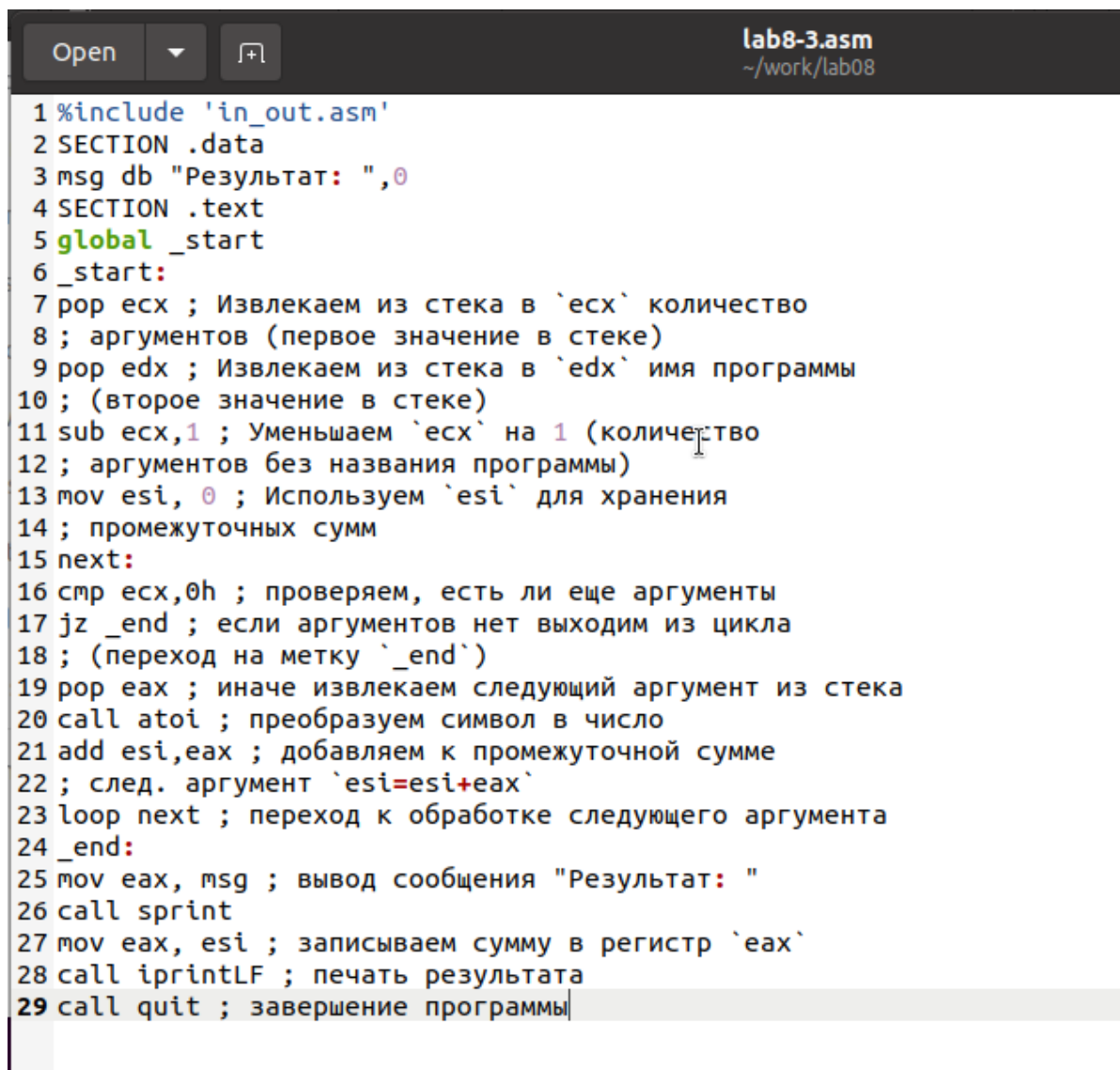
```
lella@ubuntu:~/work/lab08$
lella@ubuntu:~/work/lab08$ nasm -f elf lab8-2.asm
lella@ubuntu:~/work/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
lella@ubuntu:~/work/lab08$ ./lab8-2 argument 1 argument 2 'argument 3'
argument
1
argument
2
argument 3
lella@ubuntu:~/work/lab08$
lella@ubuntu:~/work/lab08$
```

Рис. 4.8: Тестирование программы lab8-2.asm

Давайте рассмотрим еще один пример программы. Эта программа выводит



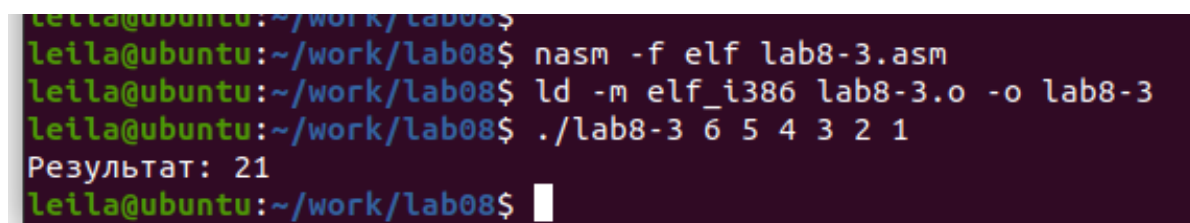
общую сумму чисел, которые были переданы в программу в качестве аргументов командной строки.



```
lab8-3.asm
~/work/lab08

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 iprintf ; печать результата
29 call quit ; завершение программы
```

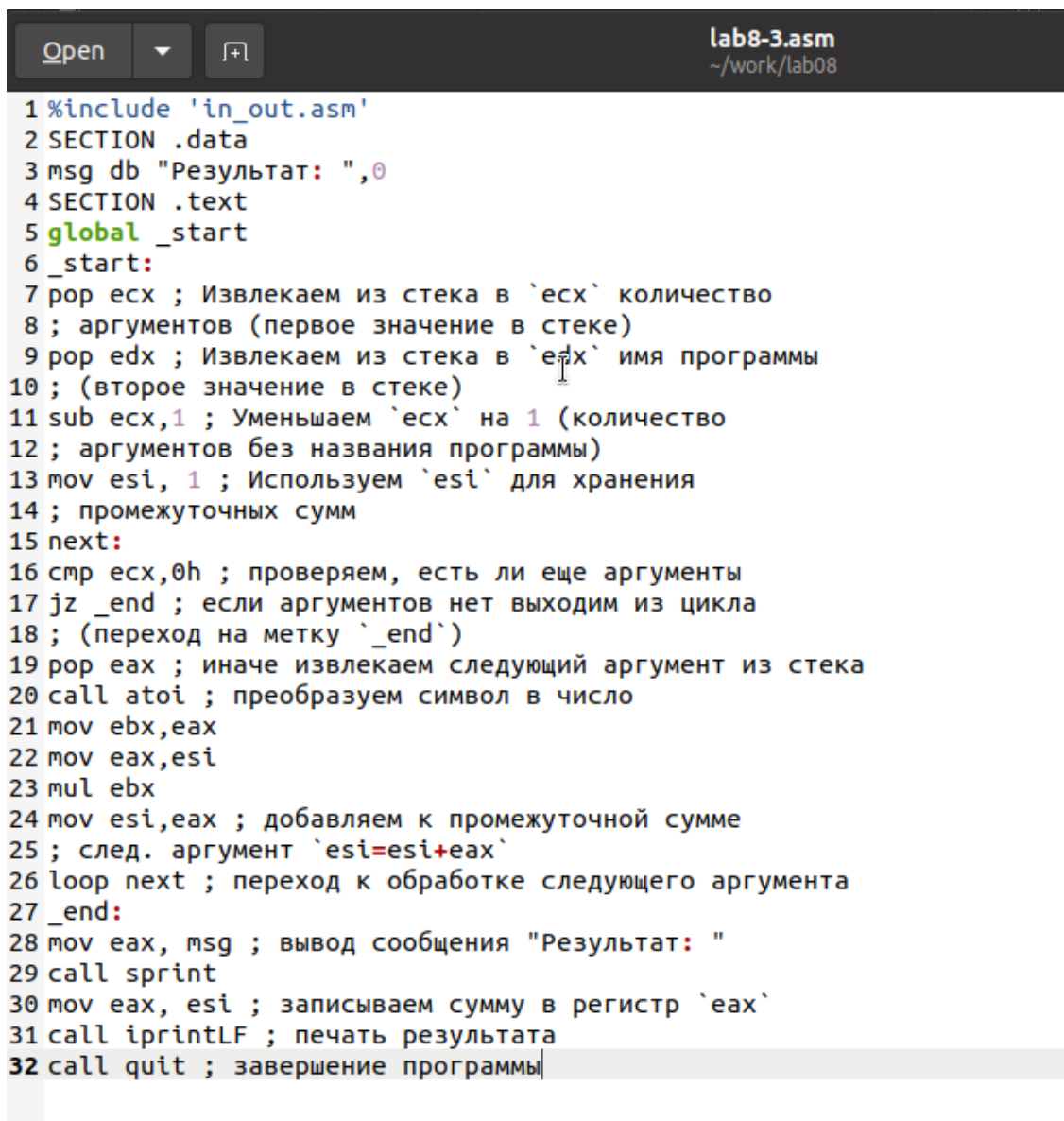
Рис. 4.9: Редактирование файла lab8-3.asm



```
leila@ubuntu:~/work/lab08$
leila@ubuntu:~/work/lab08$ nasm -f elf lab8-3.asm
leila@ubuntu:~/work/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
leila@ubuntu:~/work/lab08$ ./lab8-3 6 5 4 3 2 1
Результат: 21
leila@ubuntu:~/work/lab08$
```

Рис. 4.10: Тестирование программы lab8-3.asm

Я внесла изменения в код программы из листинга 8.3 с целью расчета 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
```

Рис. 4.11: Редактирование файла lab8-3.asm

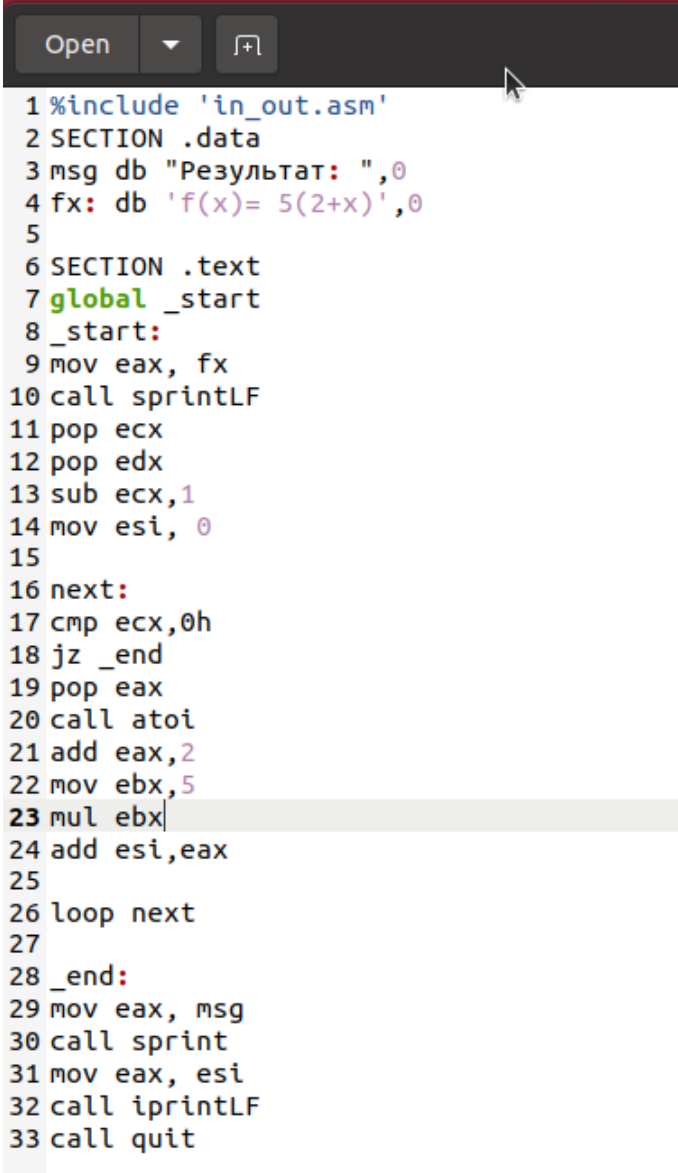
```
leila@ubuntu:~/work/lab08$  
leila@ubuntu:~/work/lab08$ nasm -f elf lab8-3.asm  
leila@ubuntu:~/work/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3  
leila@ubuntu:~/work/lab08$ ./lab8-3 6 5 4 3 2 1  
Результат: 720  
leila@ubuntu:~/work/lab08$  
leila@ubuntu:~/work/lab08$
```

Рис. 4.12: Тестирование программы lab8-3.asm

### 4.3 Выполнение заданий 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  $x$ .

Мой вариант 10:  $f(x) = 5(2 + x)$



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 5(2+x)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
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 add eax,2
22 mov ebx,5
23 mul ebx
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprint
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рис. 4.13: Редактирование файла prog.asm

```
leila@ubuntu:~/work/lab08$  
leila@ubuntu:~/work/lab08$ nasm -f elf prog.asm  
leila@ubuntu:~/work/lab08$ ld -m elf_i386 prog.o -o prog  
leila@ubuntu:~/work/lab08$ ./prog  
f(x)= 5(2+x)  
Результат: 0  
leila@ubuntu:~/work/lab08$ ./prog 0  
f(x)= 5(2+x)  
Результат: 10  
leila@ubuntu:~/work/lab08$ ./prog 1  
f(x)= 5(2+x)  
Результат: 15  
leila@ubuntu:~/work/lab08$ ./prog 6 5 4 3 2 1  
f(x)= 5(2+x)  
Результат: 165  
leila@ubuntu:~/work/lab08$  
leila@ubuntu:~/work/lab08$
```

Рис. 4.14: Тестирование программы prog.asm

## 5 Выводы

Освоили работу со стеком, циклом и аргументами на ассемблере `nasn`.