

# **Отчёт по лабораторной работе 8**

**Архитектура компьютера**

Эргешов Байрам НКАбд-02-23

# 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	Задание для самостоятельной работы . . . . .	17
<b>3</b>	<b>Выводы</b>	<b>20</b>

## Список иллюстраций

2.1	Код программы lab8-1.asm . . . . .	7
2.2	Компиляция и запуск программы lab8-1.asm . . . . .	8
2.3	Код программы lab8-1.asm . . . . .	9
2.4	Компиляция и запуск программы lab8-1.asm . . . . .	10
2.5	Код программы lab8-1.asm . . . . .	11
2.6	Компиляция и запуск программы lab8-1.asm . . . . .	12
2.7	Код программы lab8-2.asm . . . . .	13
2.8	Компиляция и запуск программы lab8-2.asm . . . . .	13
2.9	Код программы lab8-3.asm . . . . .	14
2.10	Компиляция и запуск программы lab8-3.asm . . . . .	15
2.11	Код программы lab8-3.asm . . . . .	16
2.12	Компиляция и запуск программы lab8-3.asm . . . . .	17
2.13	Код программы program-1.asm . . . . .	18
2.14	Компиляция и запуск программы program-1.asm . . . . .	19

## Список таблиц

# 1 Цель работы

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

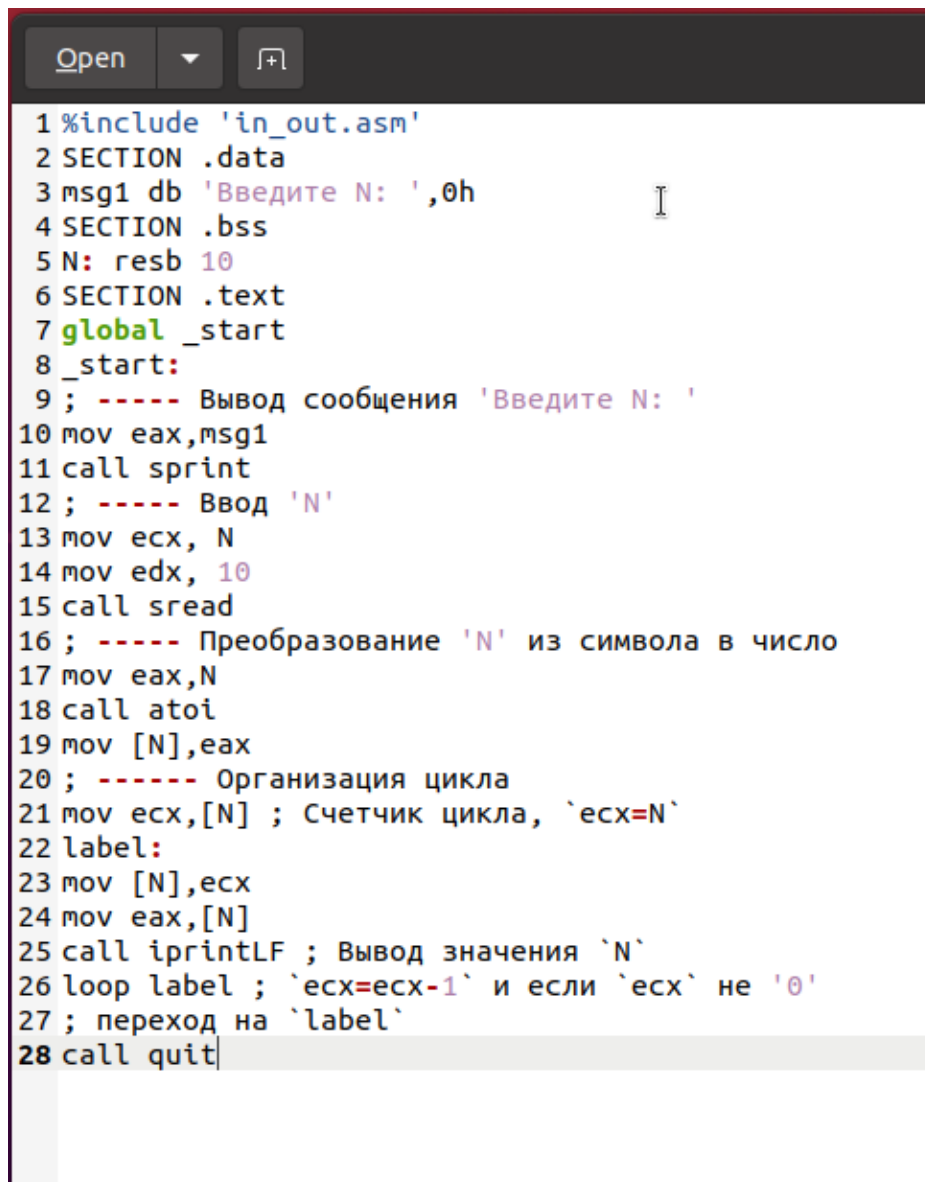
## 2 Выполнение лабораторной работы

### 2.1 Реализация циклов в NASM

ыл создан каталог для проведения лабораторной работы N8, а также был создан файл с именем lab8-1.asm.

При использовании инструкции `loop` в NASM для организации циклов, нужно помнить следующее: эта инструкция применяет регистр `ecx` в качестве счётчика и на каждой итерации уменьшает его значение на единицу. Для лучшего понимания этого процесса давайте рассмотрим пример программы, которая выводит значение регистра `ecx`.

Я написал в файл lab8-1.asm текст программы из листинга 8.1. Затем 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
```

Рис. 2.1: Код программы lab8-1.asm

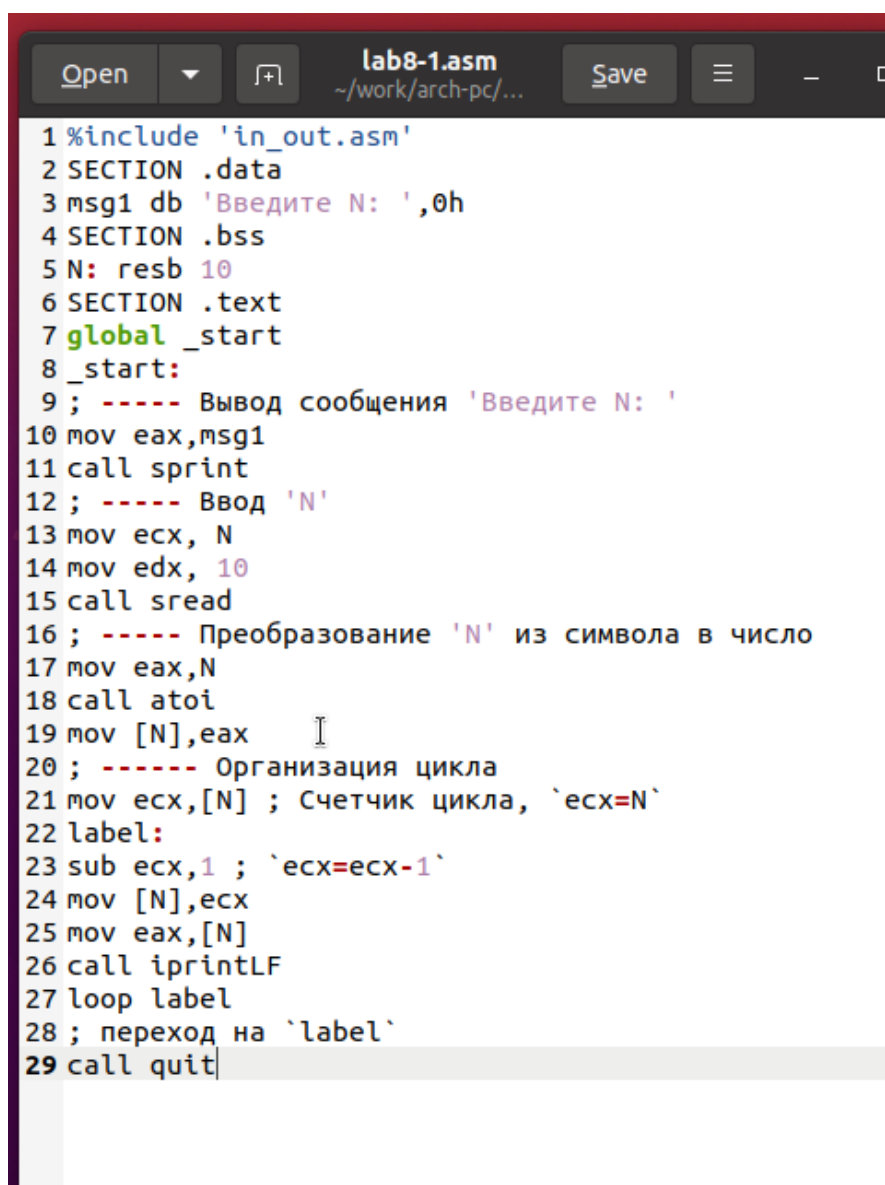
```
bobo@bergeshov:~/work/arch-pc/lab08$  
bobo@bergeshov:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
bobo@bergeshov:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1  
bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 3  
3  
2  
1  
bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-1  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

Рис. 2.2: Компиляция и запуск программы lab8-1.asm

В этом примере показано, что применение регистра `ecx` в инструкции `loop` может вызвать неправильную работу программы. В текст программы я внёс изменения, заключающиеся в модификации значения регистра `ecx` внутри цикла.

Теперь эта программа запускает бесконечный цикл при нечётном значении `N` и выводит только нечётные числа при чётном значении `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
```

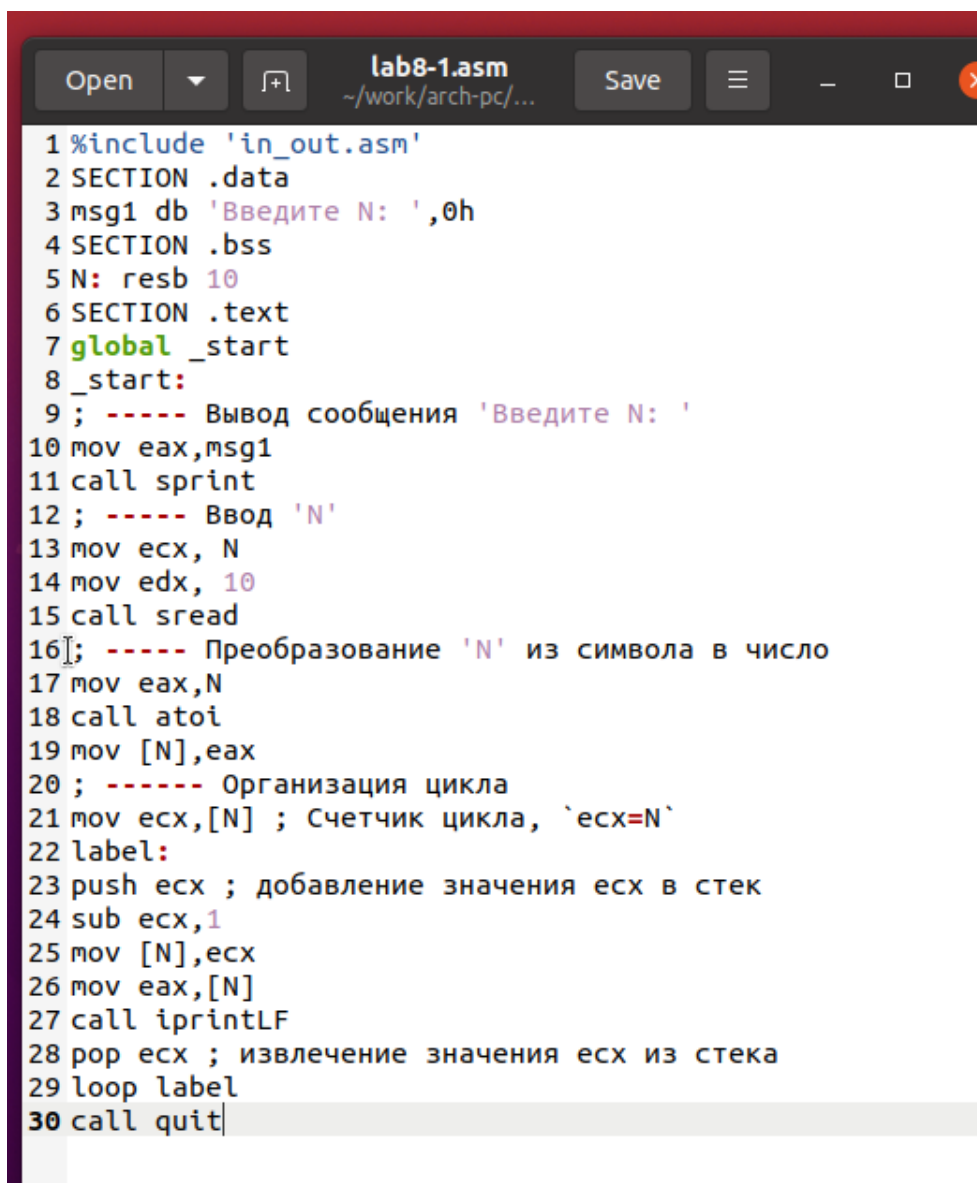
Рис. 2.3: Код программы lab8-1.asm

```
4294918140
4294918138
4294918136
4294918134
ация 4294918132
етчи 4294918130
4294918128
=ес> 4294918126
429491812^C
bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 6
5
bel 3
1
bobo@bergeshov:~/work/arch-pc/lab08$
```

Рис. 2.4: Компиляция и запуск программы lab8-1.asm

Для того, чтобы применить регистр `ecx` в цикле и обеспечить правильность работы программы, можно использовать стек. Я внёс изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счётчика цикла `loop` в стеке.

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



```
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
```

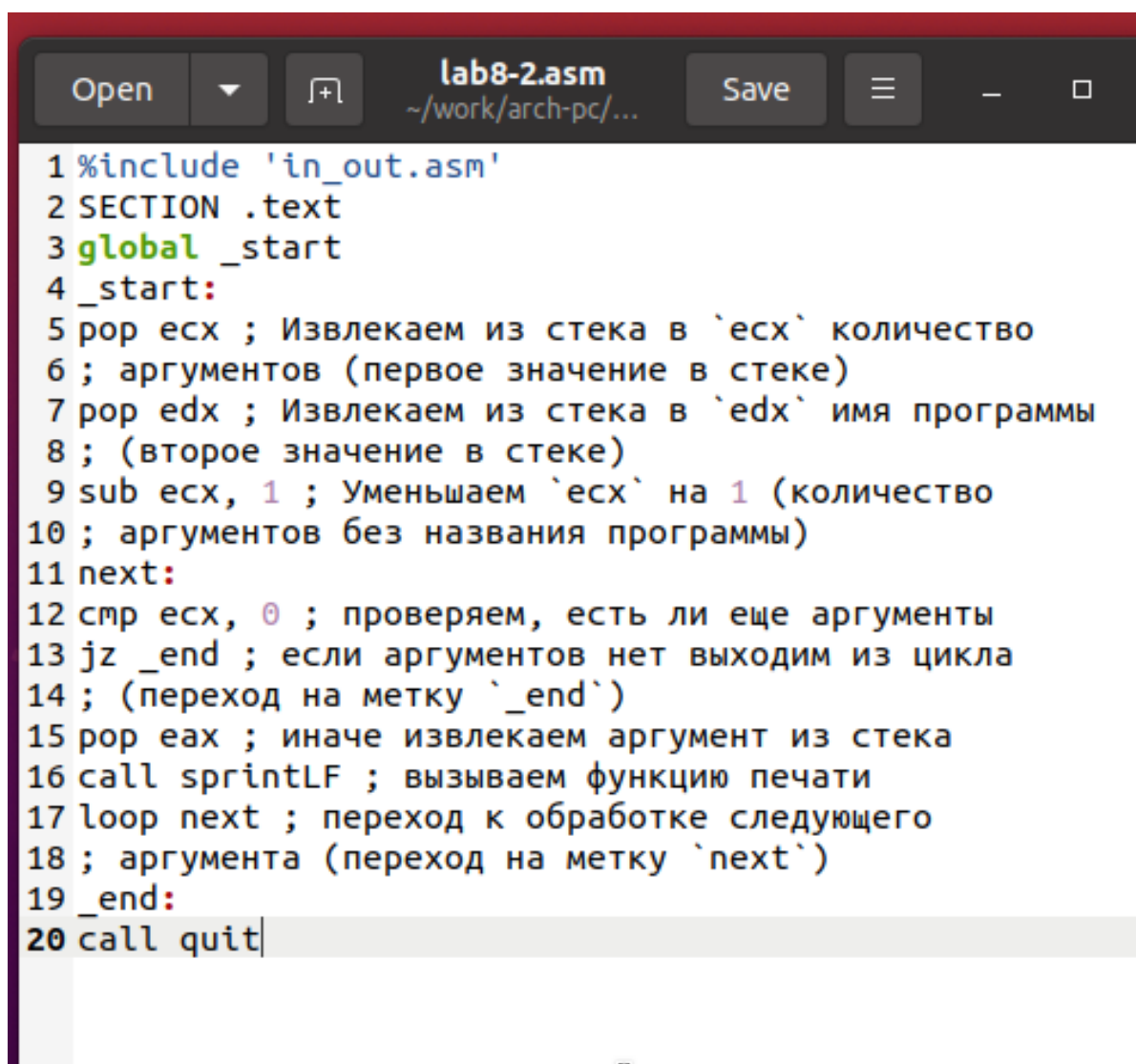
Рис. 2.5: Код программы lab8-1.asm

```
bobo@bergeshov:~/work/arch-pc/lab08$  
bobo@bergeshov:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
bobo@bergeshov:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1  
bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 3  
2  
ция1  
тчи0  
bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-1  
ениВведите N: 6  
5  
4  
3  
2  
ни1  
0  
bobo@bergeshov:~/work/arch-pc/lab08$
```

Рис. 2.6: Компиляция и запуск программы lab8-1.asm

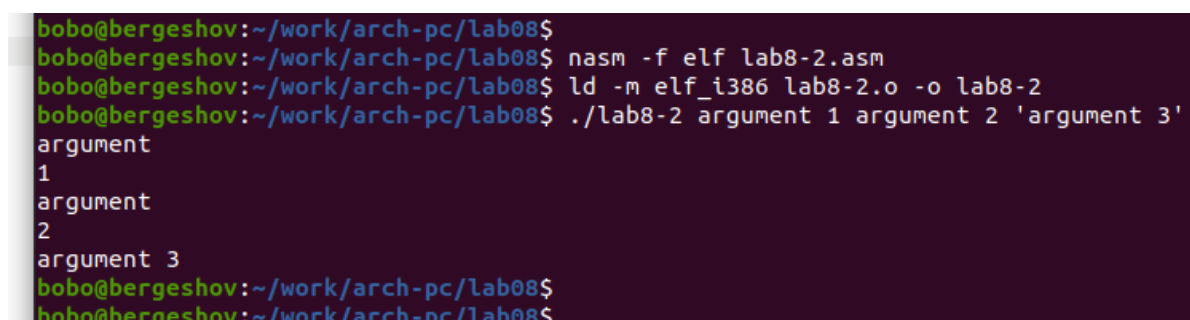
Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввел в него текст программы из листинга 8.2.

Затем создал исполняемый файл и запустил его, указав аргументы. Программа обработала 5 аргументов. Аргументами считаются слова/числа, разделённые 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
```

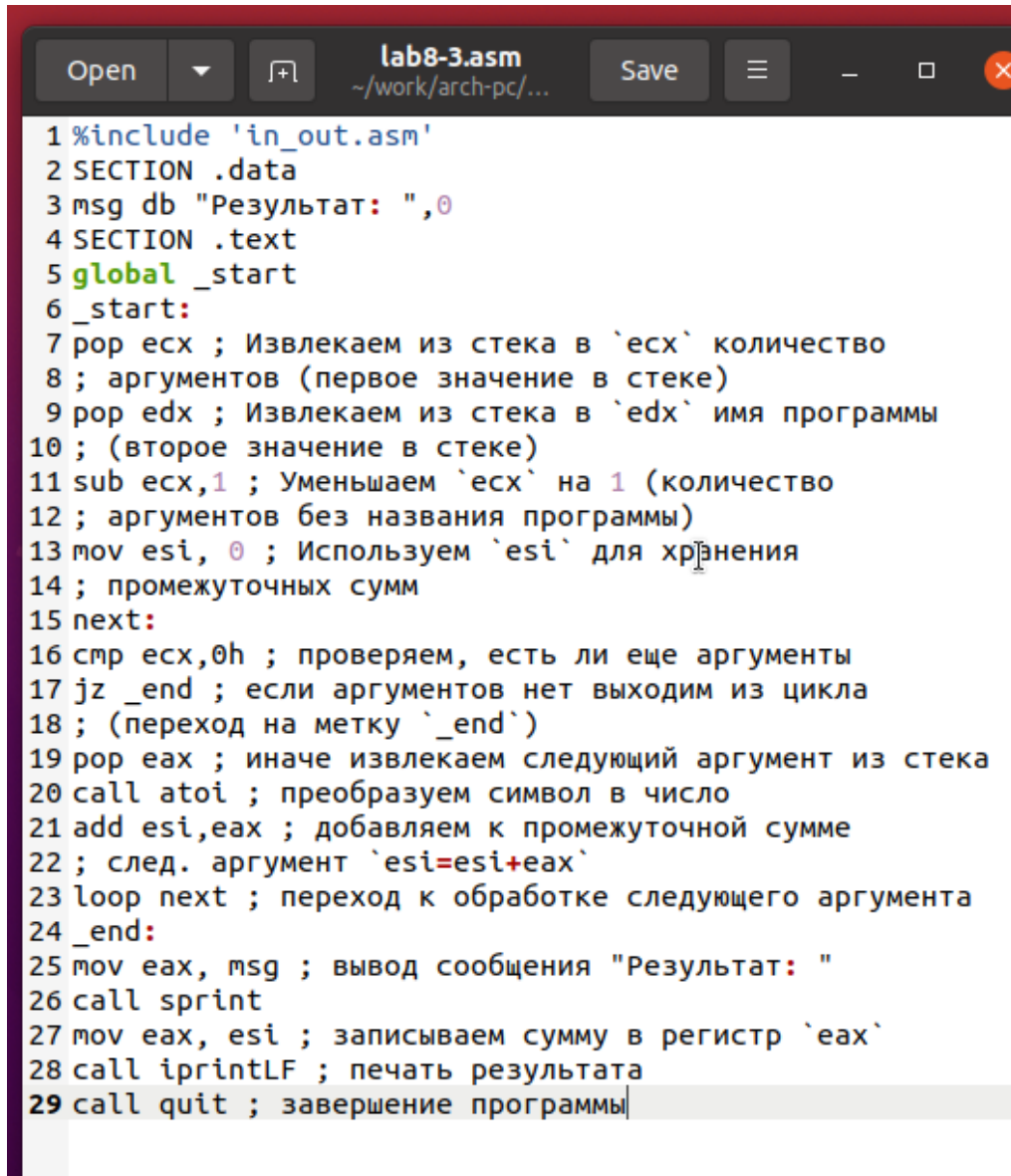
Рис. 2.7: Код программы lab8-2.asm



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

Рис. 2.8: Компиляция и запуск программы lab8-2.asm

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



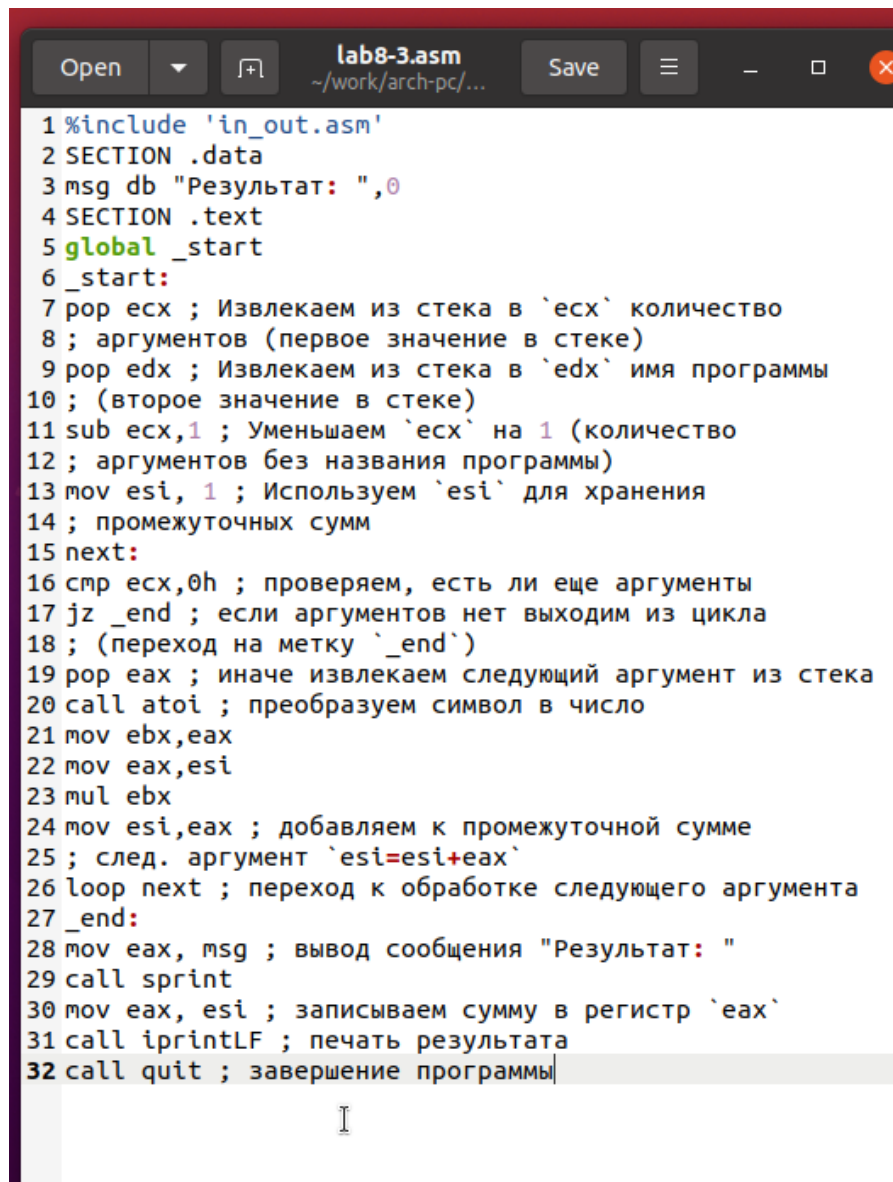
```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; 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
```

Рис. 2.9: Код программы lab8-3.asm

```
28 bobo@bergeshov:~/work/arch-pc/lab08$  
1 bobo@bergeshov:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm  
bobo@bergeshov:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3  
24 bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-3  
Результат: 0  
10 bobo@bergeshov:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4  
16 Результат: 10  
21 bobo@bergeshov:~/work/arch-pc/lab08$  
bobo@bergeshov:~/work/arch-pc/lab08$
```

Рис. 2.10: Компиляция и 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
```

Рис. 2.11: Код программы lab8-3.asm



```

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

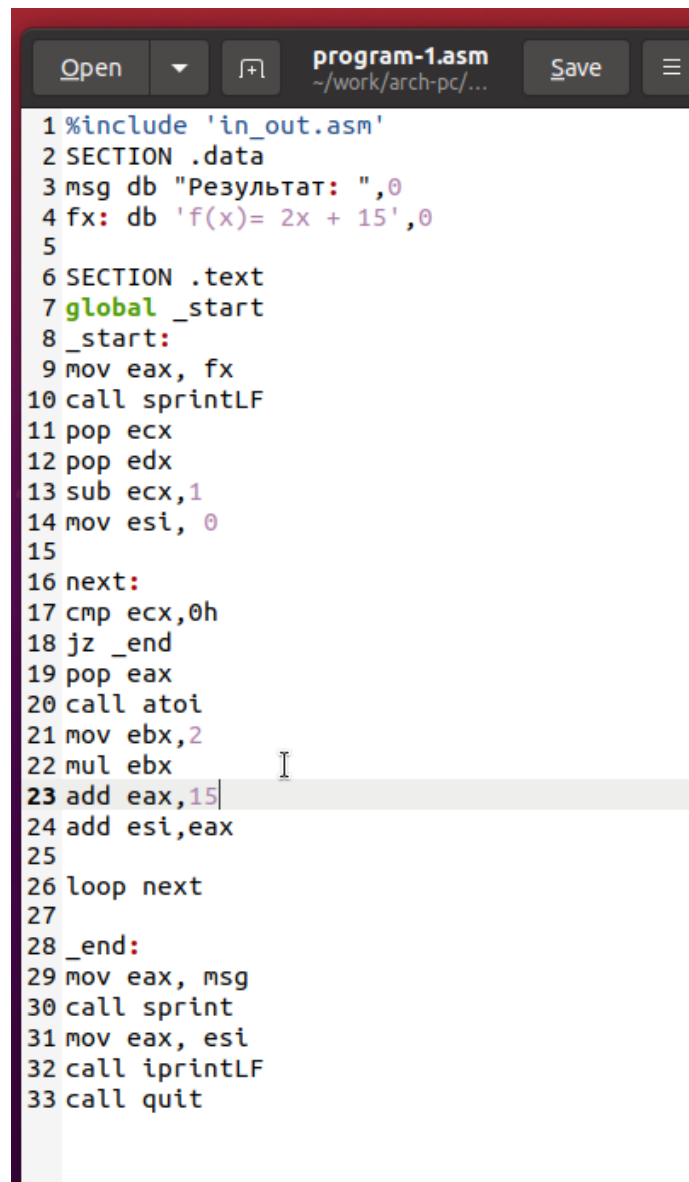
```

Рис. 2.12: Компиляция и запуск программы lab8-3.asm

## 2.2 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  $x$ .

Мой вариант 1:  $f(x) = 2x + 15$



```
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 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 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 sprint
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рис. 2.13: Код программы program-1.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке  $f(0) = 15, f(10) = 35$  Затем подал несколько аргументов и получил сумму значений функции.

```
bobo@bergeshov:~/work/arch-pc/lab08$ nasm -f elf program-1.asm
bobo@bergeshov:~/work/arch-pc/lab08$ ld -m elf_i386 program-1.o -o program-1
bobo@bergeshov:~/work/arch-pc/lab08$ ./program-1
f(x)= 2x + 15
Результат: 0
bobo@bergeshov:~/work/arch-pc/lab08$
bobo@bergeshov:~/work/arch-pc/lab08$ ./program-1 0
f(x)= 2x + 15
Результат: 15
bobo@bergeshov:~/work/arch-pc/lab08$ ./program-1 5
f(x)= 2x + 15
Результат: 25
bobo@bergeshov:~/work/arch-pc/lab08$ ./program-1 10
f(x)= 2x + 15
Результат: 35
bobo@bergeshov:~/work/arch-pc/lab08$ ./program-1 1 2 3 4
f(x)= 2x + 15
Результат: 80
bobo@bergeshov:~/work/arch-pc/lab08$
```

Рис. 2.14: Компиляция и запуск программы program-1.asm

## 3 Выводы

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