

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура компьютера**

Ларина Наталья Денисовна

# 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>8</b>
4.1	Реализация циклов в NASM . . . . .	8
4.2	Обработка аргументов командной строки . . . . .	12
4.3	Задание для самостоятельной работы . . . . .	14
<b>5</b>	<b>Выводы</b>	<b>16</b>
<b>6</b>	<b>Список литературы</b>	<b>17</b>

## Список таблиц

## Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	8
4.2	Ввод текста из листинга 8.1 . . . . .	8
4.3	Запуск исполняемого файла . . . . .	9
4.4	Изменение текста программы . . . . .	9
4.5	Запуск обновленной программы . . . . .	10
4.6	Изменение текста программы . . . . .	11
4.7	Запуск исполняемого файла . . . . .	11
4.8	Создание файла lab8-2.asm . . . . .	12
4.9	Ввод текста программы из листинга 8.2 . . . . .	12
4.10	Запуск исполняемого файла . . . . .	12
4.11	Создание файла lab8-3.asm . . . . .	13
4.12	Ввод текста программы из листинга 8.3 . . . . .	13
4.13	Запуск исполняемого файла . . . . .	13
4.14	Изменение текста программы . . . . .	14
4.15	Запуск исполняемого файла . . . . .	14
4.16	Создание файла lab8-task1.asm . . . . .	14
4.17	Текст программы . . . . .	15
4.18	Запуск исполняемого файла и проверка его работы . . . . .	15

# 1 Цель работы

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

## 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание 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.

На рис. 8.1 показана схема организации стека в процессоре.

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).

## 4 Выполнение лабораторной работы

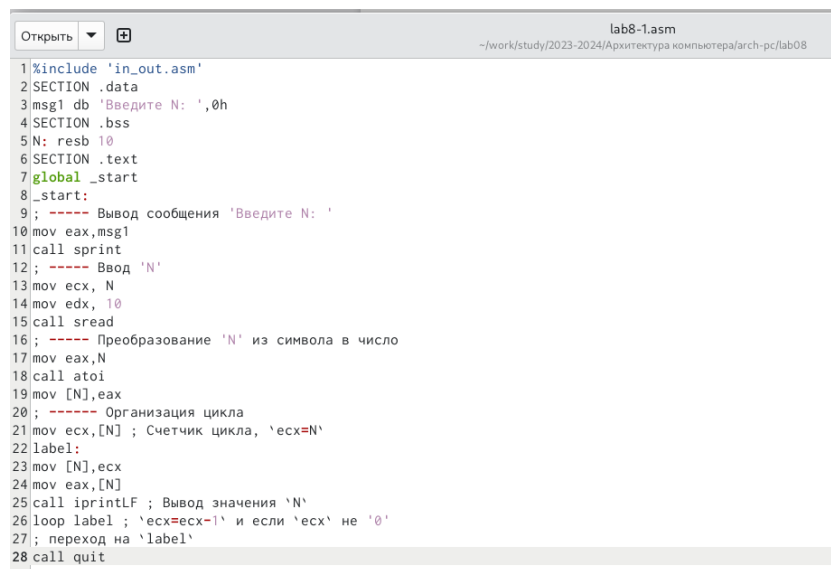
### 4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. 4.1).

```
ndlarina@dk4n69 ~ $ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08
ndlarina@dk4n69 ~ $ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab08
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-1.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ls
lab8-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.2).



```
lab8-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08

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. 4.2: Ввод текста из листинга 8.1



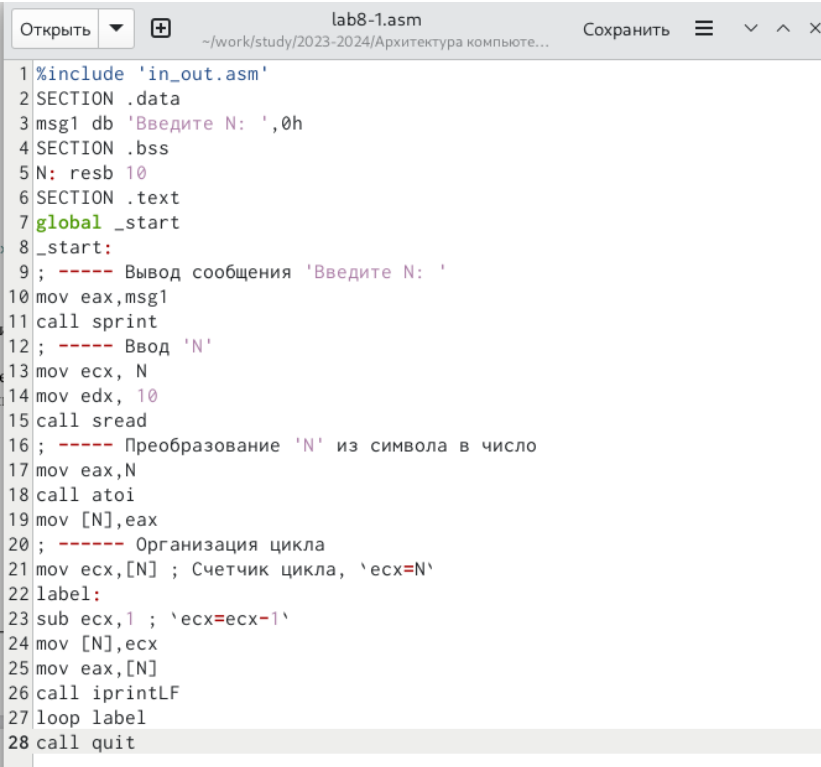
Создаю исполняемый файл и проверяю его работу. (рис. 4.3).

```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-1.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-1
Введите N: 7
7
6
5
4
3
2
1
```

Рис. 4.3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра ecx в цикле.  
(рис. 4.4).

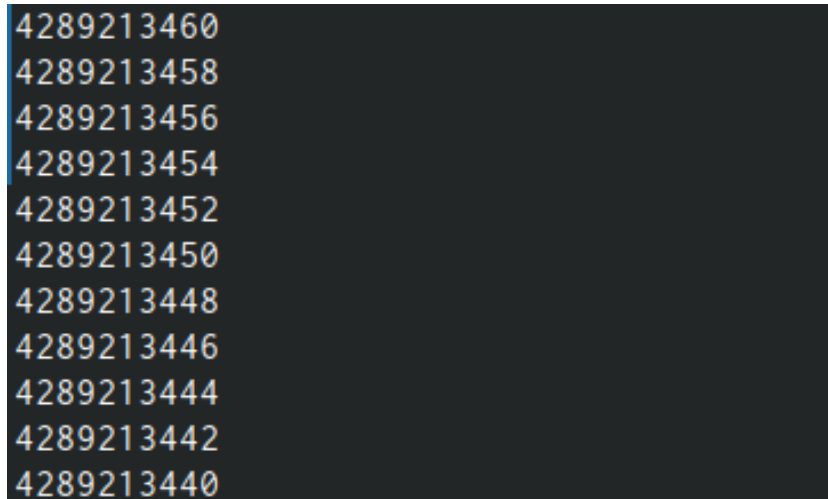


```
lab8-1.asm
~/work/study/2023-2024/Архитектура компьюте...
Сохранить

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 call quit
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.5).

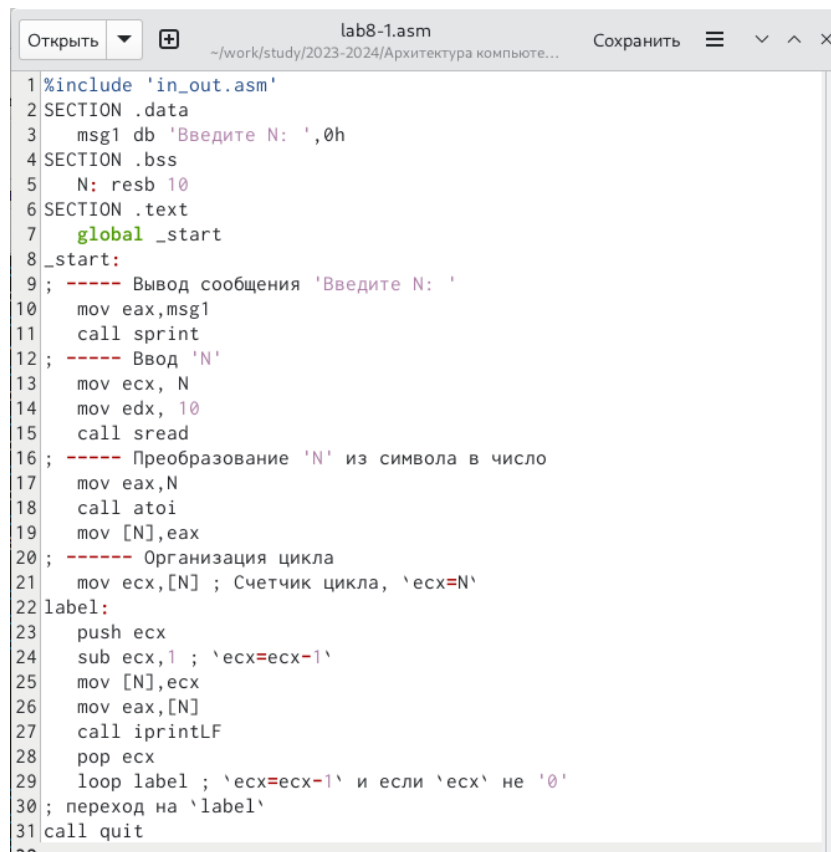


```
4289213460
4289213458
4289213456
4289213454
4289213452
4289213450
4289213448
4289213446
4289213444
4289213442
4289213440
```

Рис. 4.5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

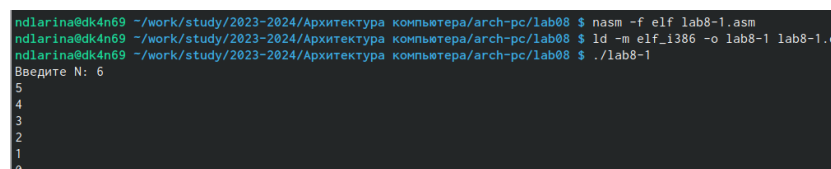
Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop`. (рис. 4.6).



```
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
24    sub ecx,1 ; 'ecx=ecx-1'
25    mov [N],ecx
26    mov eax,[N]
27    call iprintLF
28    pop ecx
29    loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
30 ; переход на 'label'
31 call quit
32
```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. 4.7).



```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-1.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
4
3
2
1
0
```

Рис. 4.7: Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

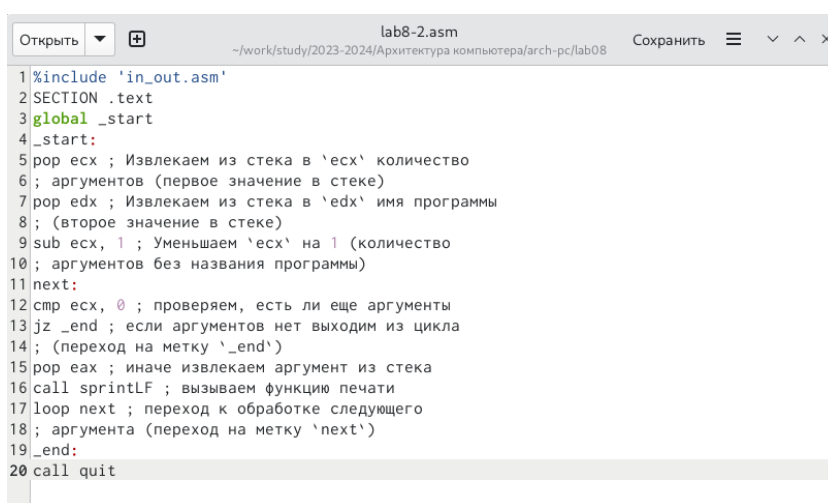
## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08. (рис. 4.8).

```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-2.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2.asm
```

Рис. 4.8: Создание файла lab8-2.asm

Ввожу в него текст программы из листинга 8.2. (рис. 4.9).



```
Открыть  lab8-2.asm  Сохранить
~/work/study/2023-2024/Архитектура компьютера/arch-pc/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 printf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку 'next')
19 _end:
20 call quit
```

Рис. 4.9: Ввод текста программы из листинга 8.2

Далее создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. 4.10).

```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-2.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
2
аргумент 3
```

Рис. 4.10: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный

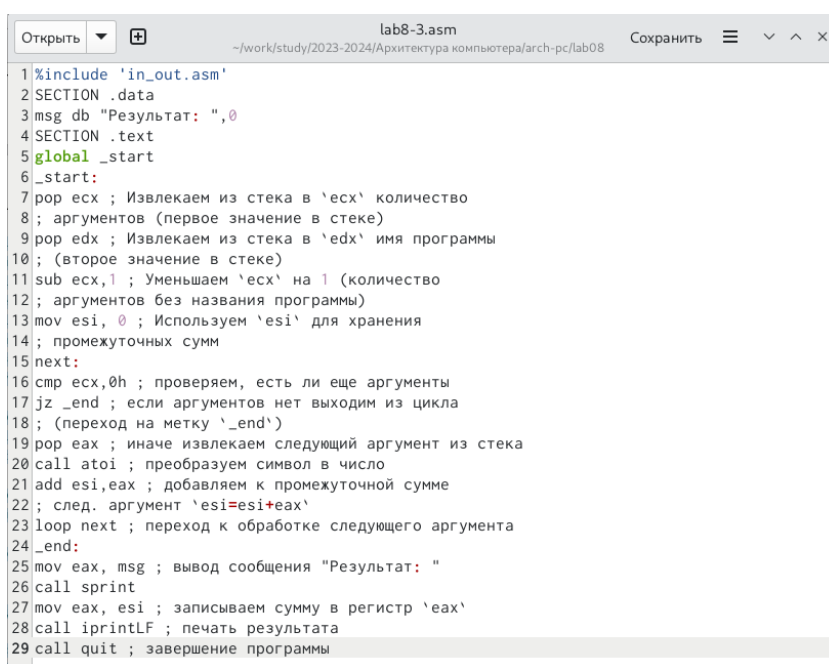
аргумент.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08. (рис. 4.11).

```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-3.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2 lab8-2.asm lab8-2.o lab8-3.asm
```

Рис. 4.11: Создание файла lab8-3.asm

Ввожу в него текст программы из листинга 8.3. (рис. 4.12).



```
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
```

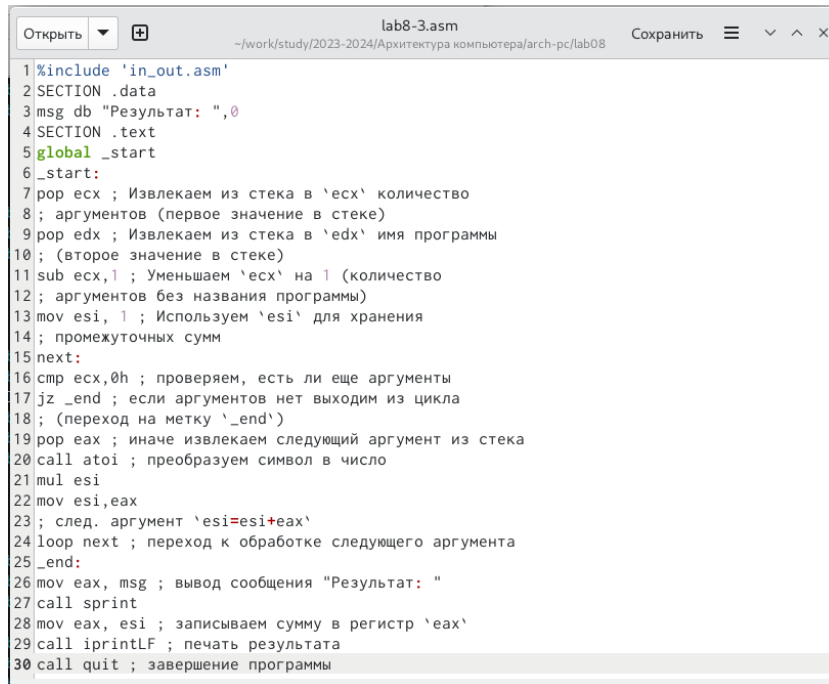
Рис. 4.12: Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.13).

```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-3.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.13: Запуск исполняемого файла

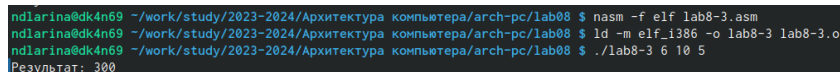
Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 4.14).



```
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 mul esi
22 mov esi,eax
23 ; след. аргумент 'esi=esi+eax'
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем сумму в регистр 'eax'
29 call iprintLF ; печать результата
30 call quit ; завершение программы
```

Рис. 4.14: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.15).

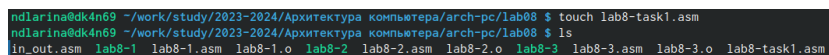


```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-3.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-3 6 10 5
Результат: 300
```

Рис. 4.15: Запуск исполняемого файла

## 4.3 Задание для самостоятельной работы

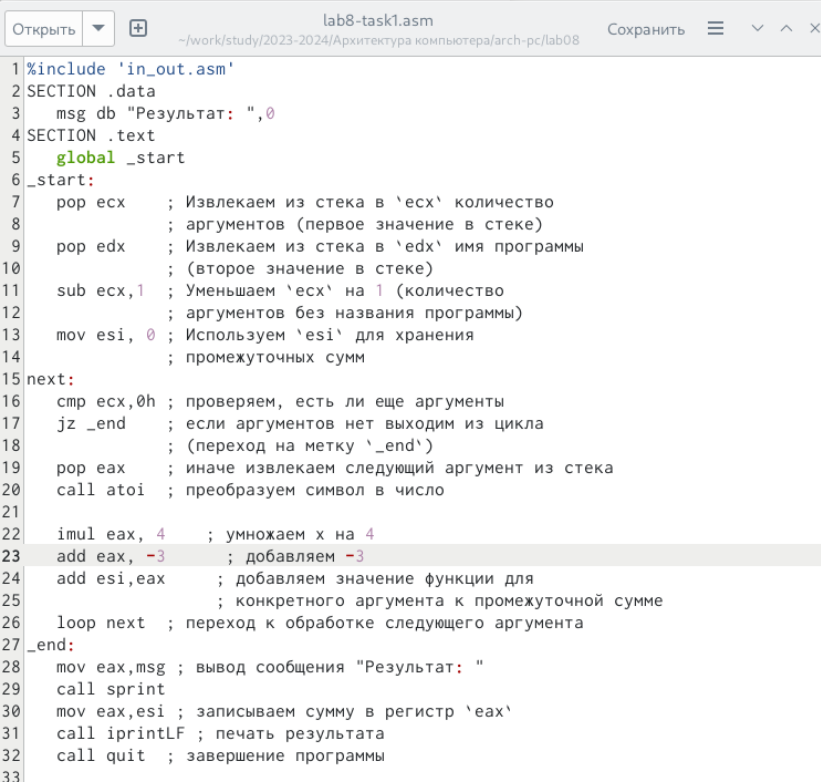
Создаю файл lab8-task1.asm в каталоге ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08. (рис. 4.16).



```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ touch lab8-task1.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ls
in_out.asm lab8-1 lab8-1.asm lab8-1.o lab8-2 lab8-2.asm lab8-2.o lab8-3 lab8-3.asm lab8-3.o lab8-task1.asm
```

Рис. 4.16: Создание файла lab8-task1.asm

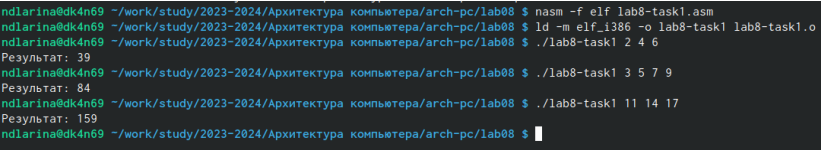
Пишу текст программы, которая находит сумму значений функции  $f(x) = 4 \cdot x - 3$  в соответствии с моим номером варианта (6) для  $x = x_1, x_2, \dots, x_n$ . Значения  $x_i$  передаются как аргументы. (рис. 4.17).



```
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
22     imul eax, 4 ; умножаем x на 4
23     add eax, -3 ; добавляем -3
24     add esi, eax ; добавляем значение функции для
25            ; конкретного аргумента к промежуточной сумме
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
```

Рис. 4.17: Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$ . (рис. 4.18).



```
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ nasm -f elf lab8-task1.asm
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ld -m elf_i386 -o lab8-task1 lab8-task1.o
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-task1 2 4 6
Результат: 39
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-task1 3 5 7 9
Результат: 84
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $ ./lab8-task1 11 14 17
Результат: 159
ndlarina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08 $
```

Рис. 4.18: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

## 5 Выводы

В ходе работы над данной лабораторной работой мне удалось приобрести навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ.



## **6 Список литературы**

1. Лабораторная работа №8