

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура компьютера**

Филатов Илья Гурамович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</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 . . . . .	16
4.3	Задание для самостоятельной работы . . . . .	21
<b>5</b>	<b>Выводы</b>	<b>24</b>
<b>6</b>	<b>Список литературы</b>	<b>25</b>

# Список иллюстраций

4.1	Создание каталога и файла . . . . .	8
4.2	Ввод программы . . . . .	9
4.3	Копирование файла . . . . .	9
4.4	Создание и запуск исполняемого файла . . . . .	10
4.5	Некорректная работа программы . . . . .	11
4.6	Изменение текста программы . . . . .	12
4.7	Создание и запуск исполняемого файла . . . . .	12
4.8	Некорректная работа программы . . . . .	14
4.9	Изменение текста программы . . . . .	15
4.10	Создание и запуск исполняемого файла . . . . .	16
4.11	Создание файла . . . . .	16
4.12	Ввод программы . . . . .	17
4.13	Создание и запуск исполняемого файла . . . . .	17
4.14	Создание файла . . . . .	17
4.15	Ввод программы . . . . .	18
4.16	Создание и запуск исполняемого файла . . . . .	18
4.17	Изменение текста программы . . . . .	19
4.18	Создание и запуск исполняемого файла . . . . .	20
4.19	Создание файла . . . . .	21
4.20	Создание и запуск исполняемого файла . . . . .	21
4.21	Создание и запуск исполняемого файла . . . . .	23

# 1 Цель работы

Приобрести навыки 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.

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).

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды

не имеют операндов.

Команда `rор` извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр `esp`, после этого уменьшает значение регистра `esp` на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти.

Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

Аналогично команде записи в стек существует команда `рора`, которая восстанавливает из стека все регистры общего назначения, и команда `рорf` для перемещения значений из вершины стека в регистр флагов.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`.

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

Открываю терминал. Создаю каталог для работы lab08 и перехожу в него. Создаю в нём файл lab8-1.asm (рис. 4.1).

A screenshot of a terminal window with a dark background. The title bar at the top shows a window icon and the text 'igfilatov@igfilatov:~/work/arch-pc/lab08'. The terminal contains four lines of text: the first line shows the command 'mkdir ~/work/arch-pc/lab08' being executed; the second line shows 'cd ~/work/arch-pc/lab08'; the third line shows 'touch lab8-1.asm'; and the fourth line shows the prompt '[igfilatov@igfilatov lab08]\$' followed by a cursor.

```
igfilatov@igfilatov:~/work/arch-pc/lab08
[igfilatov@igfilatov ~]$ mkdir ~/work/arch-pc/lab08
[igfilatov@igfilatov ~]$ cd ~/work/arch-pc/lab08
[igfilatov@igfilatov lab08]$ touch lab8-1.asm
[igfilatov@igfilatov lab08]$
```

Рис. 4.1: Создание каталога и файла

Открываю файл с помощью редактора gedit и ввожу текст программы из листинга 8.1 (рис. 4.2).



```

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 программы

Чтобы программа, которая содержит подпрограммы из in\_out.asm, работала корректно, копирую этот файл из каталога lab07 в каталог lab08 (рис. 4.3).

```

[igfilatov@igfilatov lab08]$ cd ~/work/arch-pc
[igfilatov@igfilatov arch-pc]$ cp lab07/in_out.asm lab08
[igfilatov@igfilatov arch-pc]$ cd lab08
[igfilatov@igfilatov lab08]$

```

Рис. 4.3: Копирование файла

Создаю исполняемый файл и проверяю его работу (рис. 4.4).

```
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-1.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[igfilatov@igfilatov lab08]$ ./lab8-1
Введите N: 4
4
3
2
1
[igfilatov@igfilatov lab08]$
```

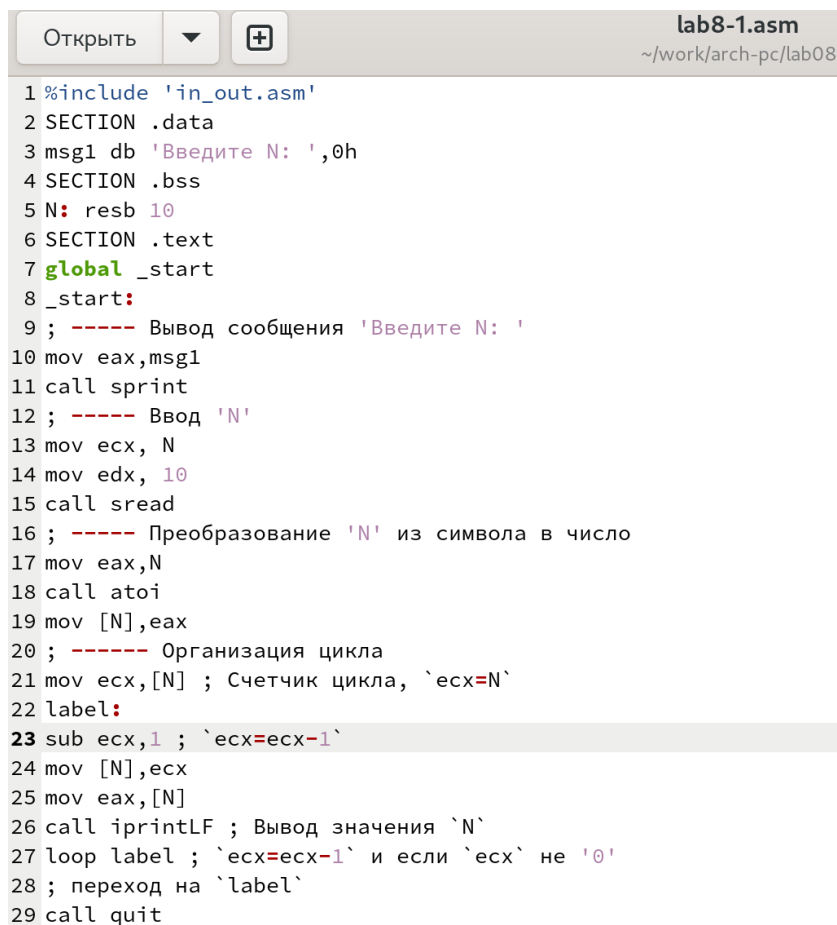
Рис. 4.4: Создание и запуск исполняемого файла

При этом, для не отрицательных значений только при  $N=0$  программа работает некорректно (рис. 4.5).

4294961763  
4294961762  
4294961761  
4294961760  
4294961759  
4294961758  
4294961757  
4294961756  
4294961755  
4294961754  
4294961753  
4294961752  
4294961751  
4294961750  
4294961749  
4294961748  
4294961747  
4294961746  
4294961745  
4294961744  
4294961743  
4294961742  
4294961741  
42949617

Рис. 4.5: Некорректная работа программы

Открываю текст программы и меняю его, добавляя значение регистра `ecx` в цикле (рис. 4.6).

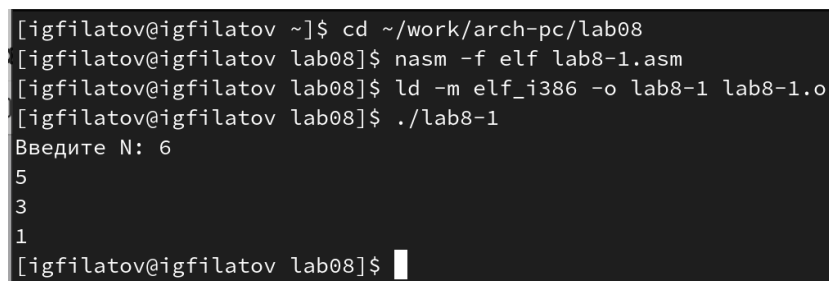


```
lab8-1.asm
~/work/arch-pc/lab08

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

Рис. 4.6: Изменение текста программы

Создаю и запускаю исполняемый файл. Он работает при чётных неотрицательных значениях `N`, а число проходов цикла в 2 раза меньше этого значения (рис. 4.7).



```
[igfilatov@igfilatov ~]$ cd ~/work/arch-pc/lab08
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-1.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[igfilatov@igfilatov lab08]$ ./lab8-1
Введите N: 6
5
3
1
[igfilatov@igfilatov lab08]$
```

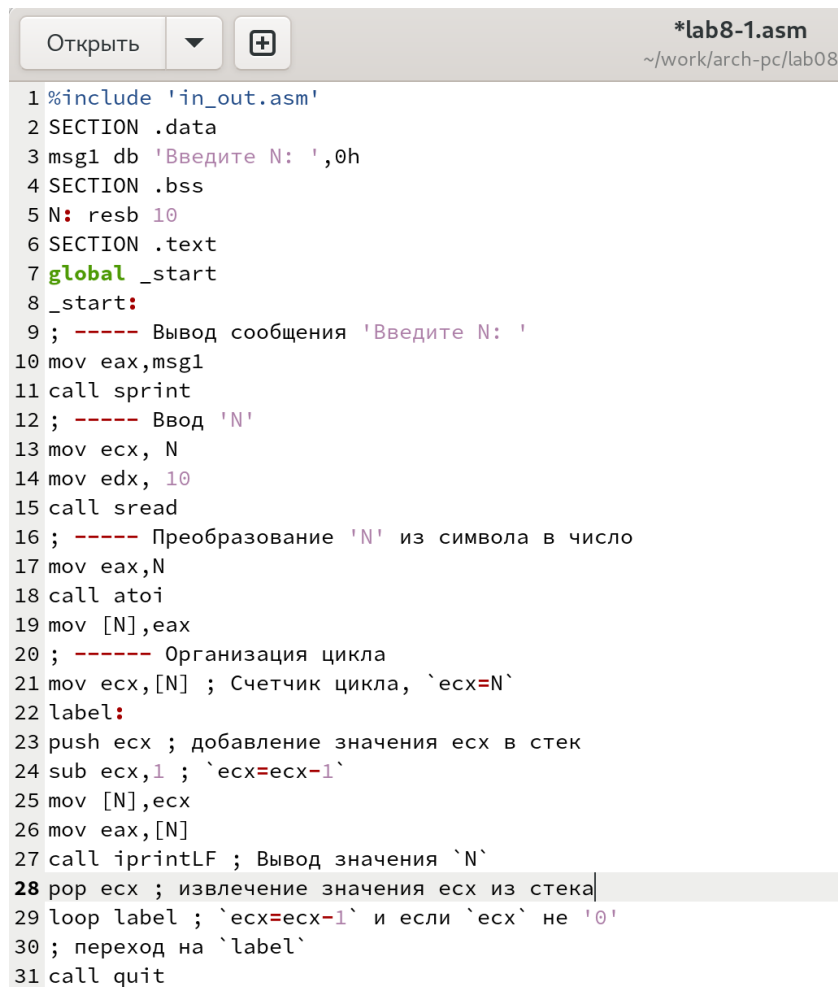
Рис. 4.7: Создание и запуск исполняемого файла

Эта программа работает некорректно при нечётных неотрицательных значениях (рис. 4.8).

4294855482  
4294855480  
4294855478  
4294855476  
4294855474  
4294855472  
4294855470  
4294855468  
4294855466  
4294855464  
4294855462  
4294855460  
4294855458  
4294855456  
4294855454  
4294855452  
4294855450  
4294855448  
4294855446  
4294855444  
4294855442  
4294855440  
4294855438  
4294

Рис. 4.8: Некорректная работа программы

Вношу изменения в текст программы, добавив команды push и pop (рис. 4.9).



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

Рис. 4.9: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. В данном случае число проходов соответствует значению N (рис. 4.10).

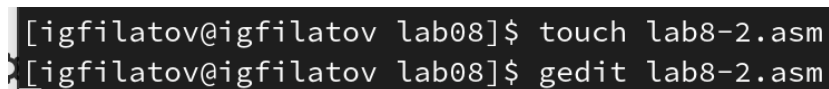


```
igfilatov@igfilatov:~/work/arch-pc/lab08
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-1.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[igfilatov@igfilatov lab08]$ ./lab8-1
Введите N: 4
3
2
1
0
[igfilatov@igfilatov lab08]$ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0
[igfilatov@igfilatov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

Рис. 4.10: Создание и запуск исполняемого файла

## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и открываю его с помощью редактора gedit (рис. 4.11).

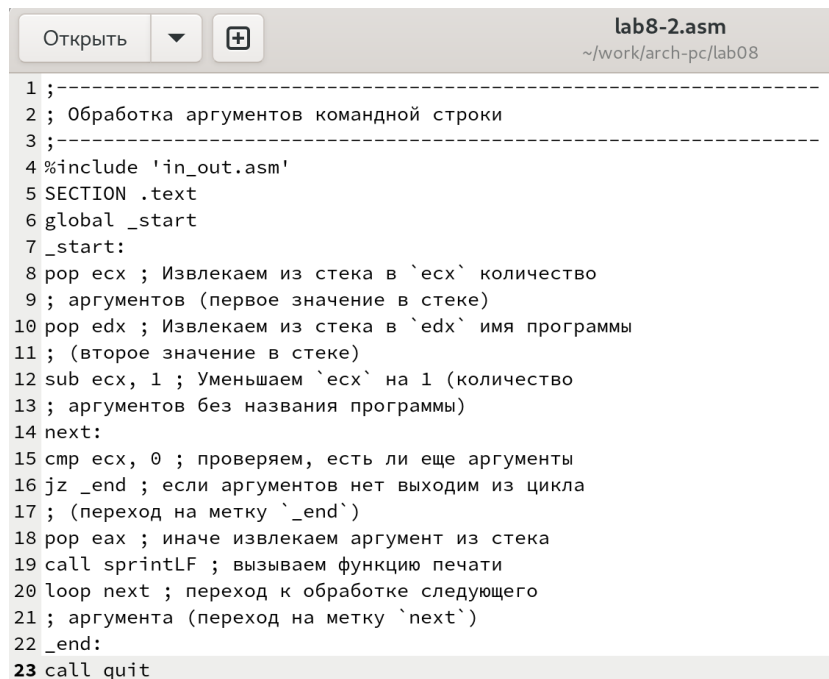


```
[igfilatov@igfilatov lab08]$ touch lab8-2.asm
[igfilatov@igfilatov lab08]$ gedit lab8-2.asm
```

Рис. 4.11: Создание файла

Ввожу текст программы из листинга 8.2 (рис. 4.12).

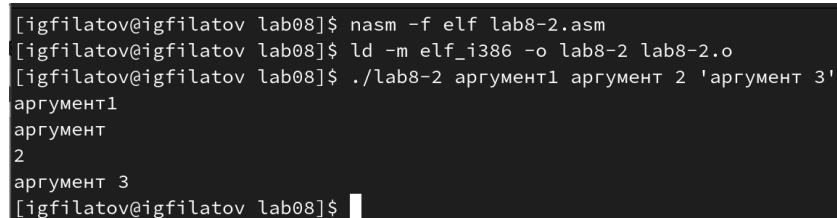




```
1 ;-----
2 ; Обработка аргументов командной строки
3 ;-----
4 %include 'in_out.asm'
5 SECTION .text
6 global _start
7 _start:
8 pop ecx ; Извлекаем из стека в `ecx` количество
9 ; аргументов (первое значение в стеке)
10 pop edx ; Извлекаем из стека в `edx` имя программы
11 ; (второе значение в стеке)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

Рис. 4.12: Ввод программы

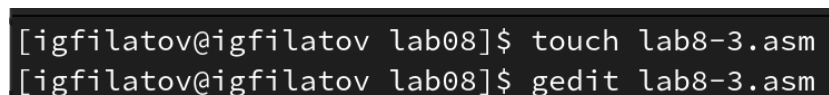
Создаю и запускаю исполняемый файл. Он обрабатывает все 3 аргумента (рис. 4.13).



```
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-2.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[igfilatov@igfilatov lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[igfilatov@igfilatov lab08]$
```

Рис. 4.13: Создание и запуск исполняемого файла

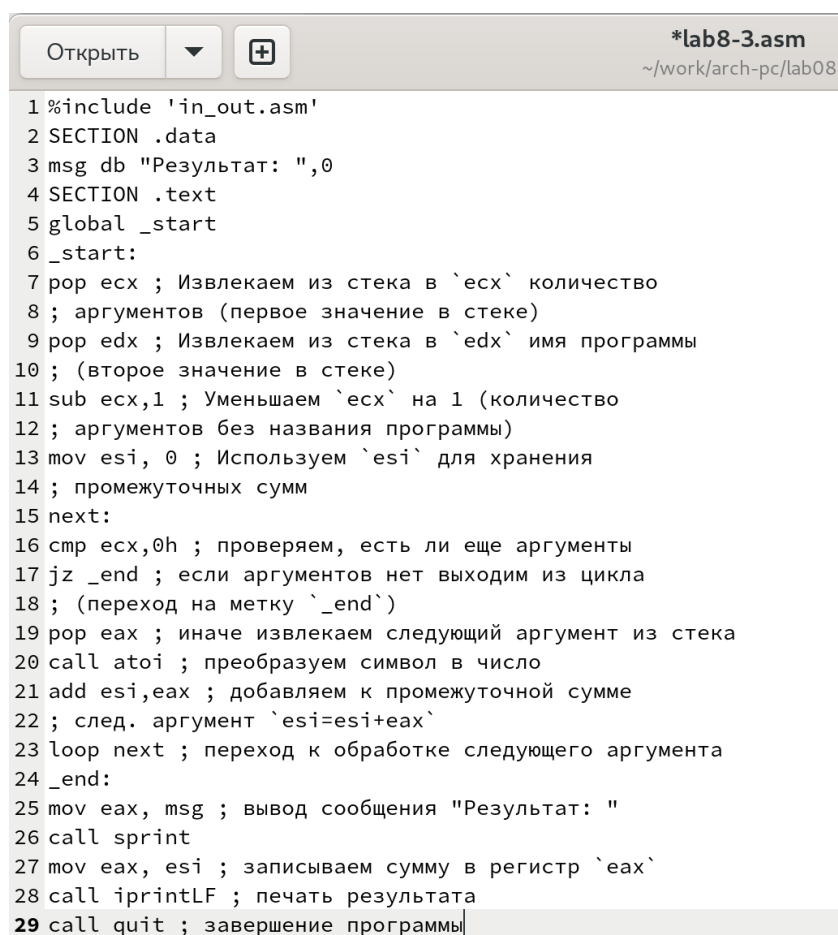
Создаю файл lab8-3.asm в каталоге ~/.work/arch-pc/lab08 и открываю его с помощью редактора gedit (рис. 4.14).



```
[igfilatov@igfilatov lab08]$ touch lab8-3.asm
[igfilatov@igfilatov lab08]$ gedit lab8-3.asm
```

Рис. 4.14: Создание файла

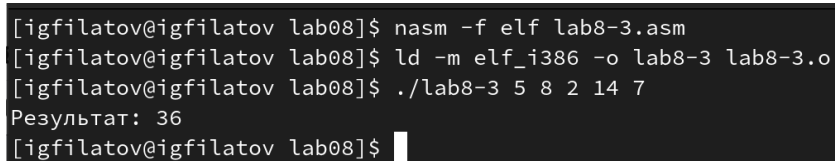
Ввожу текст программы из листинга 8.3 (рис. 4.15).



```
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. 4.15: Ввод программы

Создаю и запускаю исполняемый файл для нахождения суммы чисел (рис. 4.16).



```
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-3.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[igfilatov@igfilatov lab08]$ ./lab8-3 5 8 2 14 7
Результат: 36
[igfilatov@igfilatov lab08]$
```

Рис. 4.16: Создание и запуск исполняемого файла

Открываю текст программы и меняю его для вычисления произведения чисел (рис. 4.17).

```
lab8-3.asm
~/work/study/2024-2025/Архитектура компью
Открыть ▼ +
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 произведений
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 imul esi,eax ; умножаем на промежуточное произведение
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. 4.17: Изменение текста программы

Текст программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg db "Результат:",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx ; Извлекаем из стека в ecx количество
```

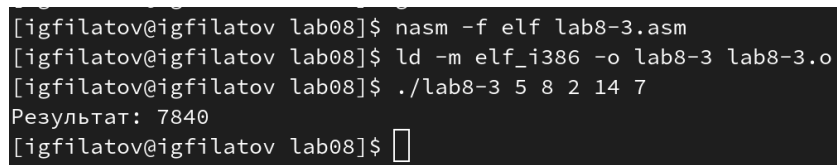
```
; аргументов (первое значение в стеке)
```

```
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 произведений
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
imul esi,eax ; умножаем на промежуточное произведение
; 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 произведение в регистр eax
call iprintLF ; печать результата
call quit ; завершение программы
Создаю исполняемый файл и проверяю его (рис. 4.18).

```



```

[igfilatov@igfilatov lab08]$ nasm -f elf lab8-3.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[igfilatov@igfilatov lab08]$ ./lab8-3 5 8 2 14 7
Результат: 7840
[igfilatov@igfilatov lab08]$ 

```

Рис. 4.18: Создание и запуск исполняемого файла

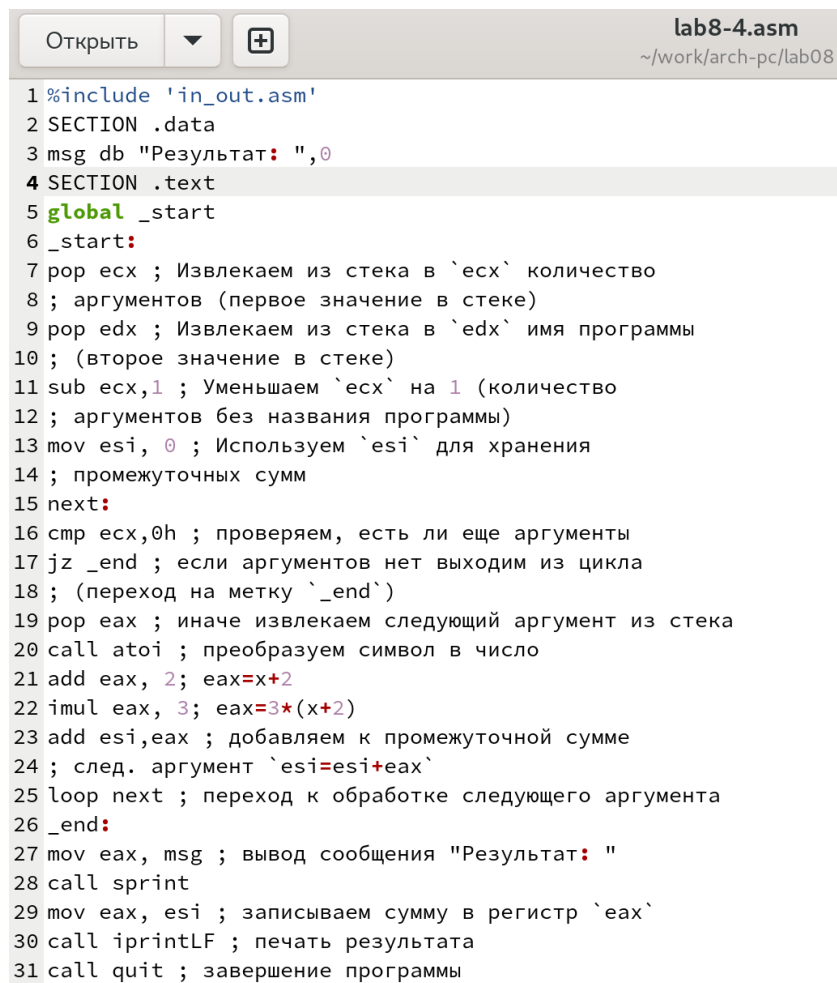
Создаю файл lab8-4.asm в каталоге ~/work/arch-рс/lab08 и открываю его с помощью редактора gedit (рис. 4.19).

```
[igfilatov@igfilatov lab08]$ touch lab8-4.asm  
[igfilatov@igfilatov lab08]$ gedit lab8-4.asm
```

Рис. 4.19: Создание файла

### 4.3 Задание для самостоятельной работы

Пишу программу для нахождения суммы значений функции  $3(x+2)$  из варианта 7 (рис. 4.20).



```
lab8-4.asm  
~/work/arch-pc/lab08  
Открыть ▼ +  
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 eax, 2; eax=x+2  
22 imul eax, 3; eax=3*(x+2)  
23 add esi,eax ; добавляем к промежуточной сумме  
24 ; след. аргумент `esi=esi+eax`  
25 loop next ; переход к обработке следующего аргумента  
26 _end:  
27 mov eax, msg ; вывод сообщения "Результат: "  
28 call sprint  
29 mov eax, esi ; записываем сумму в регистр `eax`  
30 call iprintLF ; печать результата  
31 call quit ; завершение программы
```

Рис. 4.20: Создание и запуск исполняемого файла

Текст программы:

```
%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 eax, 2; eax=x+2
imul eax, 3; eax=3*(x+2)
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

Создаю исполняемый файл и проверяю его (рис. 4.21).

```
[igfilatov@igfilatov lab08]$ nasm -f elf lab8-4.asm
[igfilatov@igfilatov lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[igfilatov@igfilatov lab08]$ ./lab8-4 5 8 2 14 7
Результат: 138
[igfilatov@igfilatov lab08]$ ./lab8-4 2 3 1
Результат: 36
```

Рис. 4.21: Создание и запуск исполняемого файла

## 5 Выводы

Я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки.



## **6 Список литературы**

1. Архитектура ЭВМ