

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура Компьютера**

Ислам Вячеславович Карданов

# 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>
<b>5</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Работа с директориями и создание файла . . . . .	8
4.2	Редактирование файла . . . . .	9
4.3	Копирование, подготовка и исполнение файла . . . . .	10
4.4	Редактирование файла . . . . .	11
4.5	Создание и запуск исполняемого файла . . . . .	12
4.6	Создание, редактирование файла . . . . .	13
4.7	Создание и запуск исполняемого файла . . . . .	14
4.8	Редактирование файла . . . . .	15
4.9	Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла . . . . .	15
4.10	Запуск исполняемого файла . . . . .	16
4.11	Редактирование файла . . . . .	16
4.12	Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла . . . . .	17
4.13	Открытие листинга . . . . .	17
4.14	Редактирование файла . . . . .	18
4.15	Создание и запуск исполняемого файла . . . . .	18
4.16	Создание файла, редактирование, компиляция, обработка и запуск исполняемого файла . . . . .	19

# 1 Цель работы

Целью данной работы является приобретение практического опыта в написании программ с использованием циклов и обработкой аргументов командной строки.

## 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. 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. 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. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: 1) добавление элемента в вершину стека (push). 2) 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 pop 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. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре есх. Инструкция loor выполняется в два этапа. Сначала из регистра есх вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке.

## 4 Выполнение лабораторной работы

### 4.1) Реализация циклов в NASM.

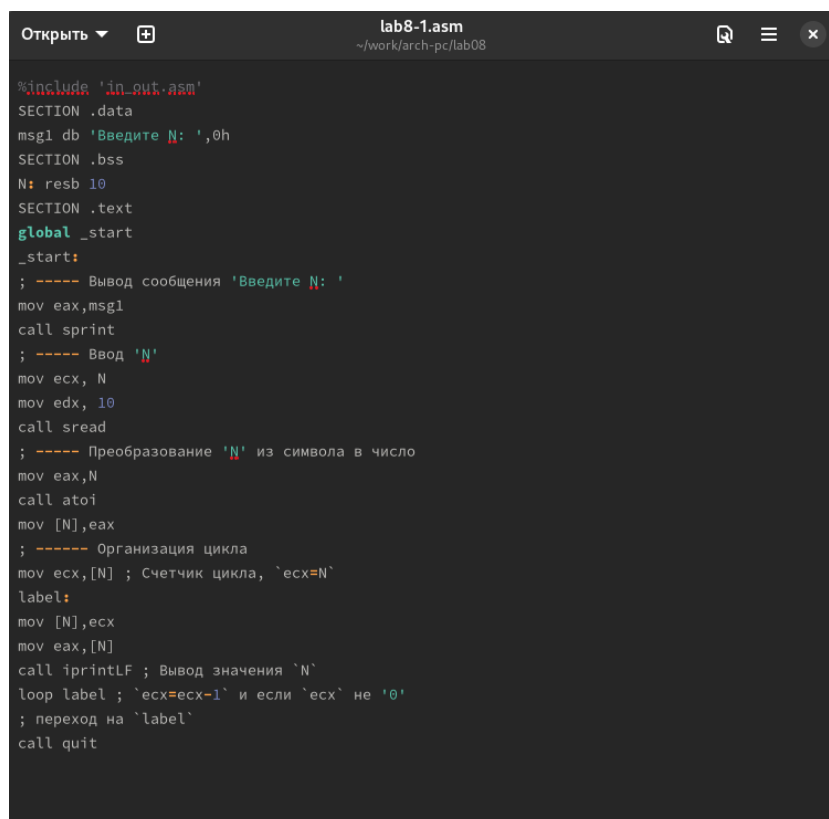
С помощью утилиты `mkdir` создаю директорию `lab08` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab8-1.asm`. (рис. 4.1).

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

Рис. 4.1: Работа с директориями и создание файла

Открываю созданный файл `lab8-1.asm`, вставляю в него следующую программу: (рис. 4.2).





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

%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

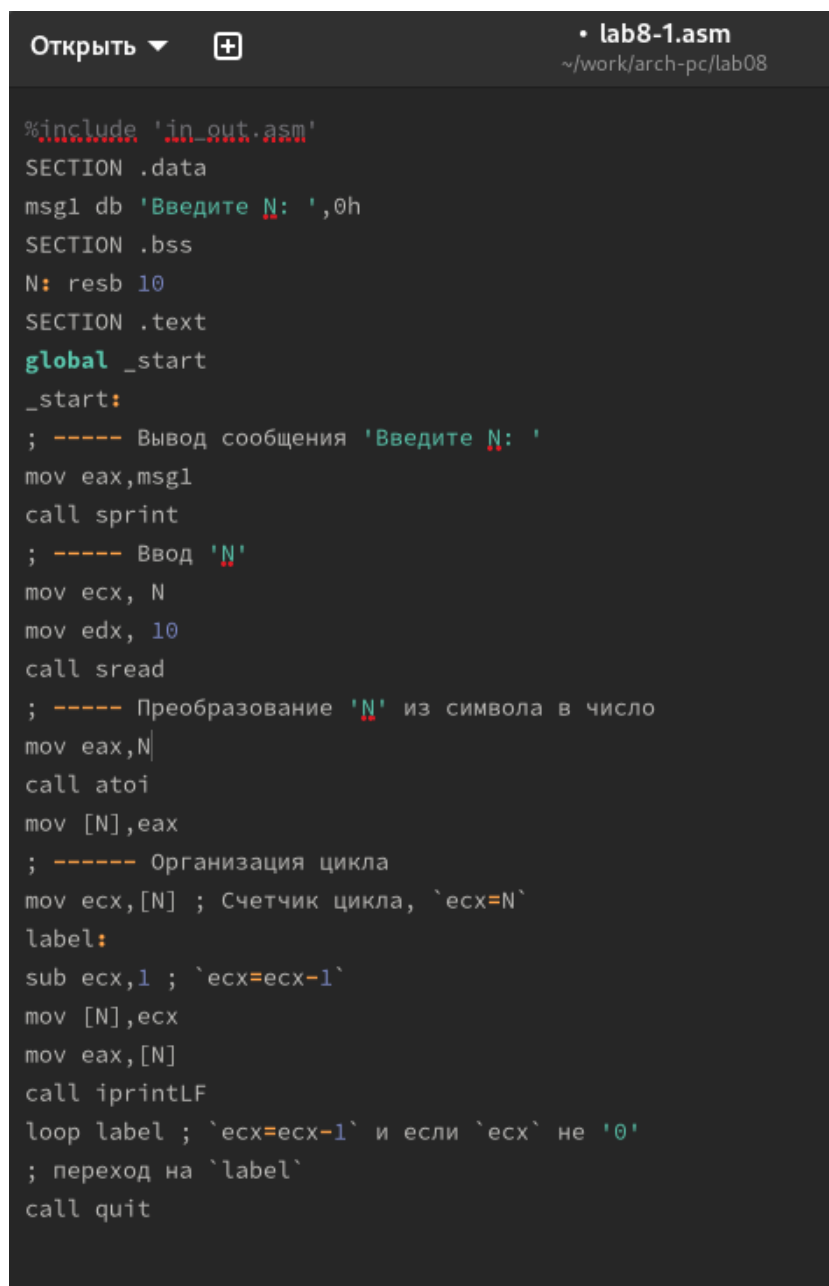
Рис. 4.2: Редактирование файла


Дублирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. будет использоваться в дальнейшем. Создаю исполняемый файл и запускаю его. Мы видим, что использование инструкции `loop` позволяет выводить значения регистра `ecx` циклично. (рис. 4.3).

```
[ivkardanov@fedora lab08]$ cp ~/3арпузки/in_out.asm in_out.asm
[ivkardanov@fedora lab08]$ nasm -f elf lab8-1.asm
[ivkardanov@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[ivkardanov@fedora lab08]$ ./lab8-1
Введите N: 8
8
7
6
5
4
3
2
1
[ivkardanov@fedora lab08]$
```

Рис. 4.3: Копирование, подготовка и исполнение файла

Изменяю значение esx в цикле. (рис. 4.4).



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

%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit
```

Рис. 4.4: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его. Мы видим, что регистр `ecx` в цикле принимает совершенно разные значения. И число проходов цикла далеко не соответствует ли значению ☒, введенному с клавиатуры. (рис. 4.5).

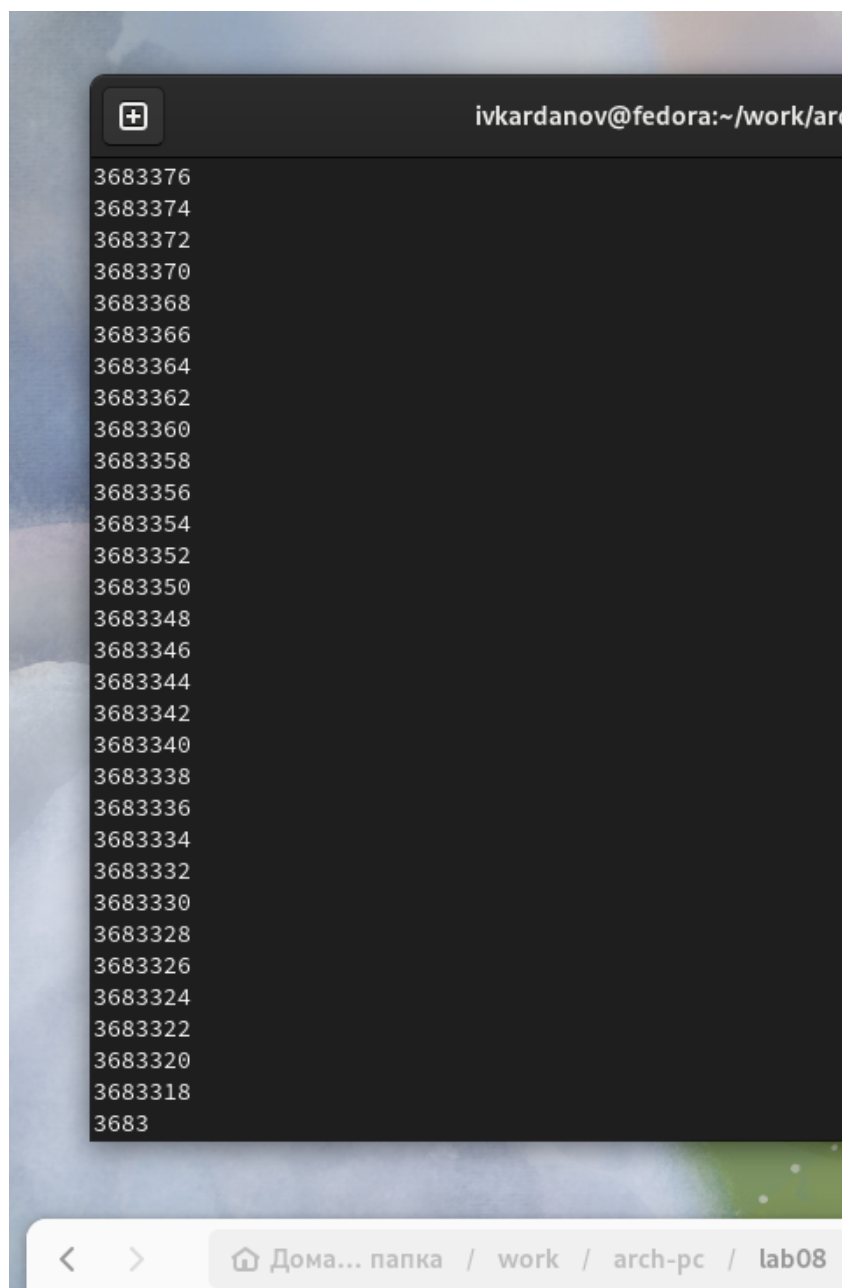
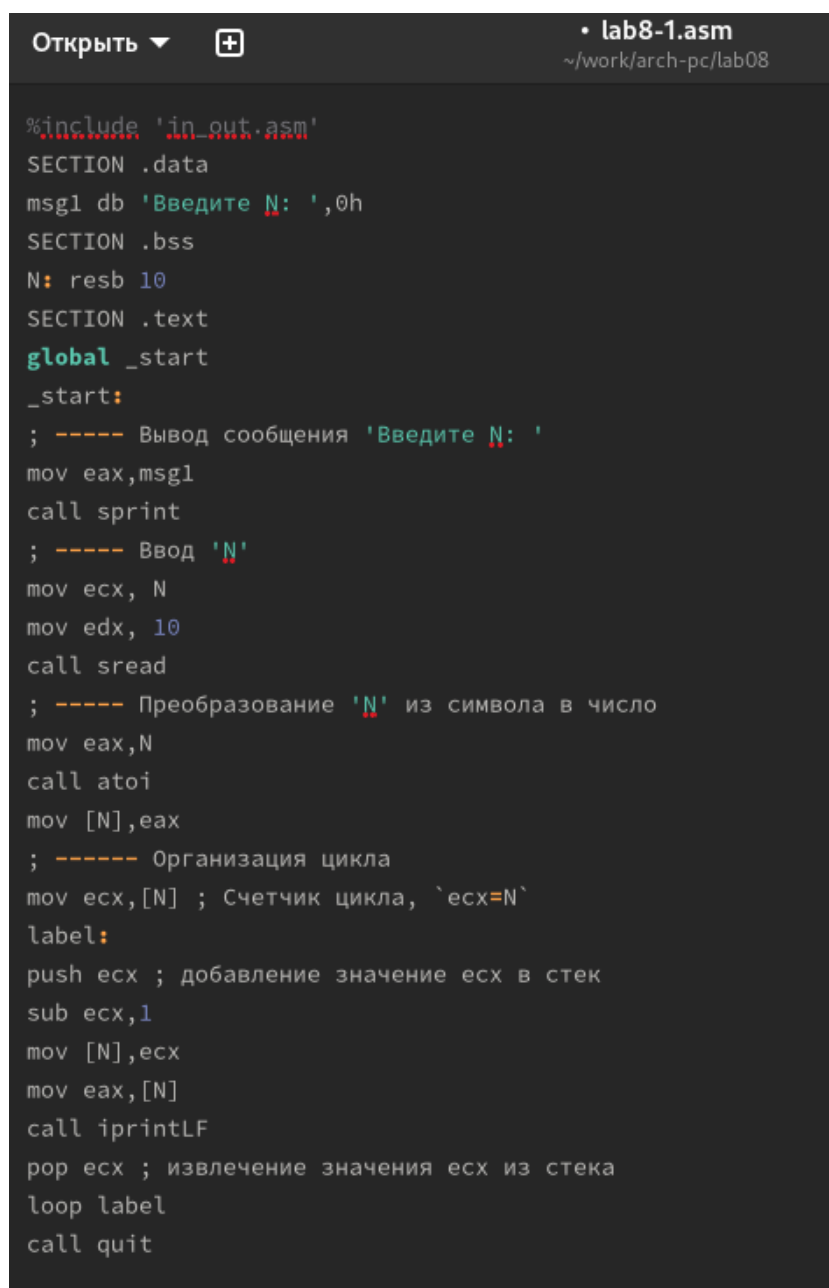


Рис. 4.5: Создание и запуск исполняемого файла

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



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

%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
push ecx ; добавление значение ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit
```

Рис. 4.6: Создание, редактирование файла

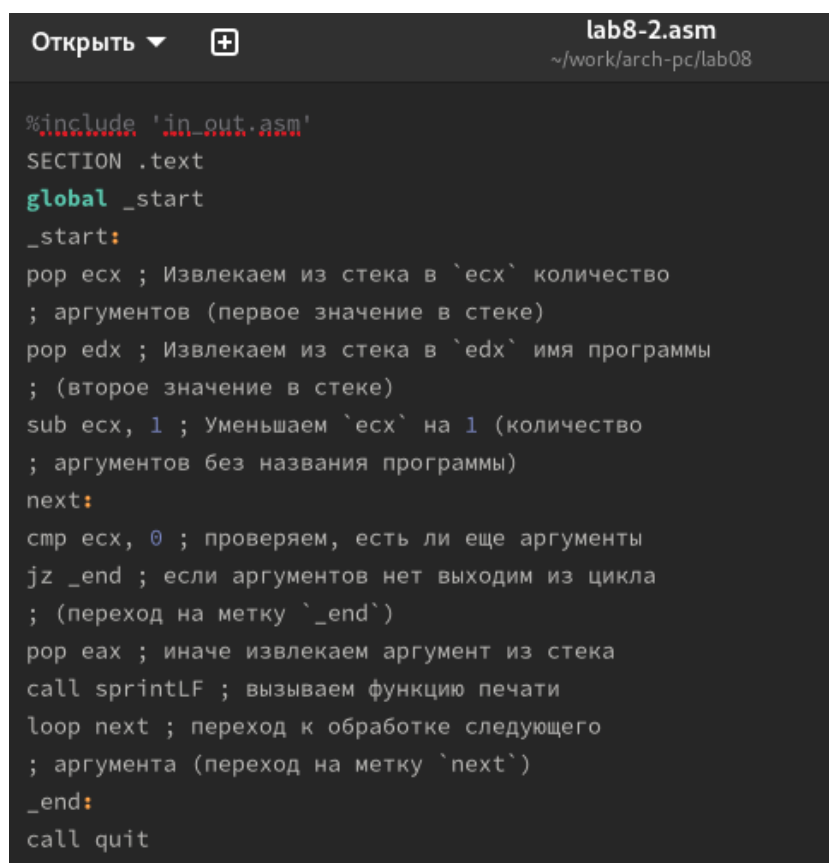
Создаю и запускаю исполняемый файл. В данном случае число проходов цикла соответствует значению ☒ введенному с клавиатуры. Счёт идёт, не от 8-ми, а от 7-ми, но включается 0 (рис. 4.7).

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

Рис. 4.7: Создание и запуск исполняемого файла

#### 4.2) Обработка аргументов командной строки.

Создаю файл lab8-2.asm. Редактирую его, вводя предлагаемую программу. (рис. 4.8).

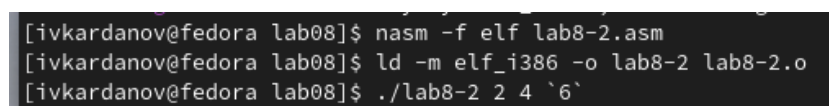


```
Открыть ▾ + lab8-2.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
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)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call printf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 4.8: Редактирование файла

Создаю исполняемый файл после редактирования. (рис. 4.9).



```
[ivkardanov@fedora lab08]$ nasm -f elf lab8-2.asm
[ivkardanov@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[ivkardanov@fedora lab08]$ ./lab8-2 2 4 6`
```

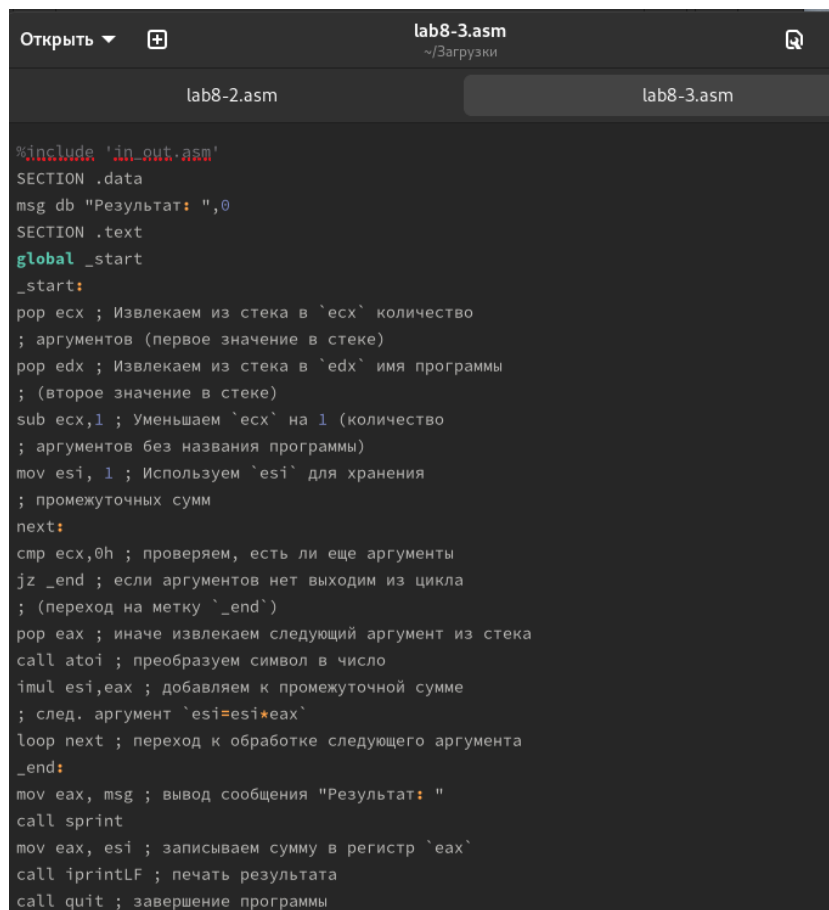
Рис. 4.9: Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла

Запускаю исполняемый файл. Программой было обработано 3 аргумента - ровно те, которые я указал при запуске. (рис. 4.10).

```
[ivkardanov@fedora lab08]$ ./lab8-2 1 6 '5'
1
6
5
```

Рис. 4.10: Запуск исполняемого файла

Создаю файл lab8-3.asm. Ввожу в него следующую программу: (рис. 4.11).



```
lab8-3.asm
~/Загрузки

lab8-2.asm lab8-3.asm

%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 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
    imul 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
```

Рис. 4.11: Редактирование файла

Создаю исполняемый файл. (рис. 4.12).



```
[ivkardanov@fedora lab08]$ touch lab8-3.asm  
[ivkardanov@fedora lab08]$ gedit lab8-3.asm  
[ivkardanov@fedora lab08]$ nasm -f elf lab8-3.asm  
[ivkardanov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
```

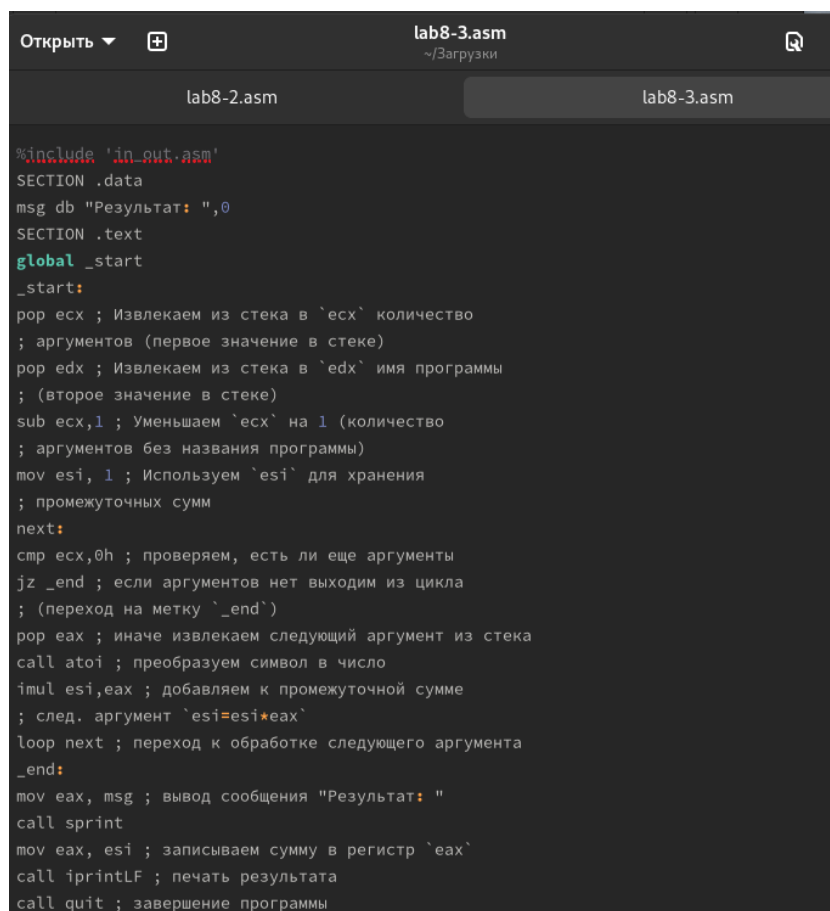
Рис. 4.12: Создание файла, открытие его в режиме правки, компиляция и обработка исполняемого файла

Указываю нужные аргументы. Убеждаюсь в правильности работы программы. (рис. 4.13).

```
[ivkardanov@fedora lab08]$ ./lab8-3 145 35 14 5 1  
Результат: 200  
[ivkardanov@fedora lab08]$
```

Рис. 4.13: Открытие листинга

Меняю текст программы для вычисления произведения аргументов в командной строке. (рис. 4.14).

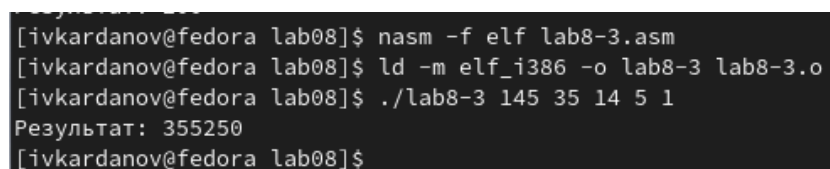


```
Открыть ▾ + lab8-3.asm ~/Загрузки
lab8-2.asm lab8-3.asm

%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 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
imul 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
```

Рис. 4.14: Редактирование файла

Создаю и запускаю исполняемый файл. При проверке вижу, что выводятся верные значения. (рис. 4.15).



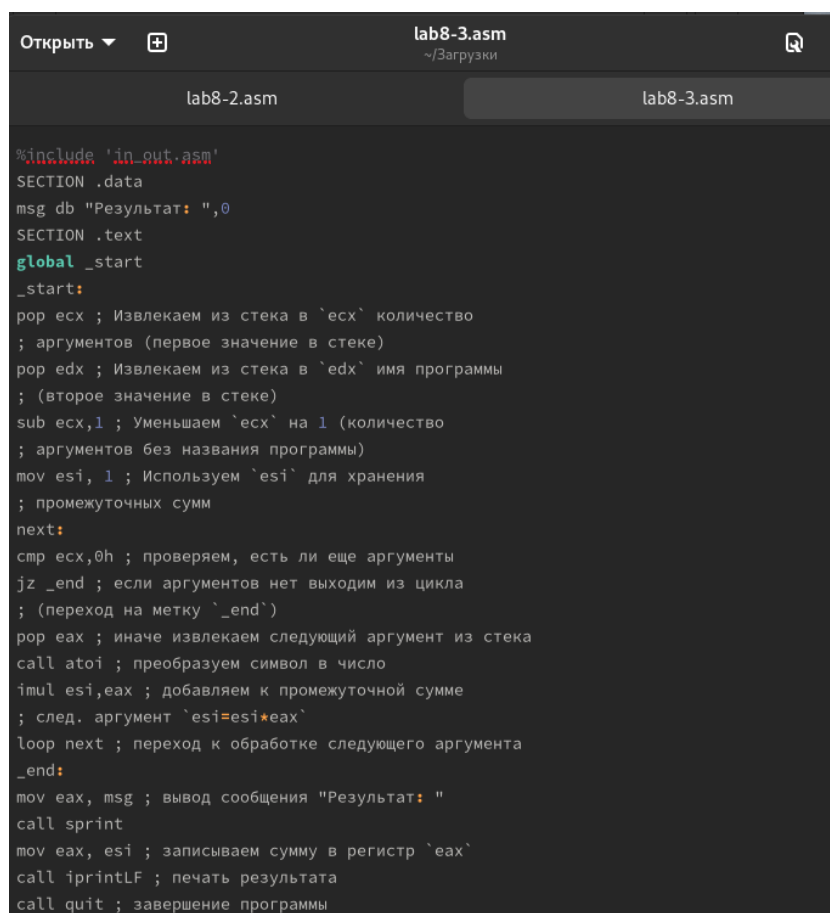
```
[ivkardanov@fedora lab08]$ nasm -f elf lab8-3.asm
[ivkardanov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[ivkardanov@fedora lab08]$ ./lab8-3 145 35 14 5 1
Результат: 355250
[ivkardanov@fedora lab08]$
```

Рис. 4.15: Создание и запуск исполняемого файла

#### 4.3) Выполнение заданий для самостоятельной работы

Создаю sr.asm с помощью утилиты touch. Открываю файл, ввожу в него текст программы для суммирования значений функции, предложенной в варианте 19,

полученным мною при выполнении прошлой лабораторной работы. Проводим привычные операции и запускаем файл. (рис. 4.16)



```
Открыть ▾ + lab8-3.asm
~/Загрузки

lab8-2.asm lab8-3.asm

%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 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
imul 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
```

Рис. 4.16: Создание файла, редактирование, компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для суммирования нескольких значений функции, предложенной в варианте 7.

```
“%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
imul eax,8
add eax,-3
add esi,eax ; добавляем к промежуточной сумме
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 ""

```

## 5 Выводы

При выполнении лабораторной работы я приобрел практический опыт в написании программ с использованием циклов и обработкой аргументов командной строки.

# Список литературы

Архитектура компьютера и ЭВМ