

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Бондаренко Кристина Антоновна

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Самостоятельная работа	12
4	Вывод	16

Список иллюстраций

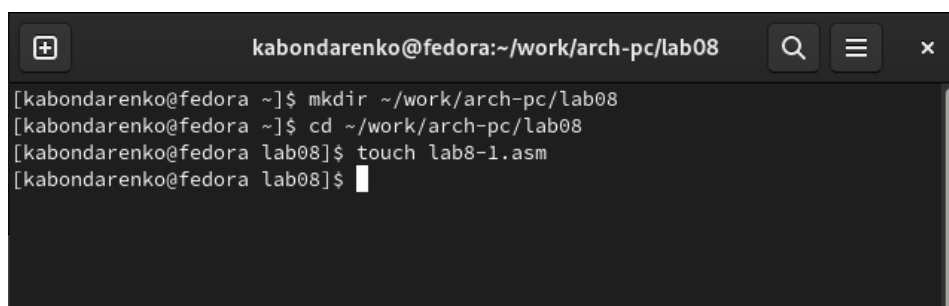
2.1	Создание директории	5
2.2	Редактирование файла	6
2.3	Запуск исполняемого файла	6
2.4	Уменьшение индекса	7
2.5	Запуск исполняемого файла	7
2.6	Редактирование программы	7
2.7	Создание исполняемого файла	8
2.8	Запуск исполняемого файла	9
2.9	Запуск программы	11
3.1	Создание файла	12
3.2	Редактирование файла	13
3.3	Запуск исполняемого файла	14

1 Цель работы

Получение навыков по организации циклов и работе со стеком на языке NASM.

2 Выполнение лабораторной работы

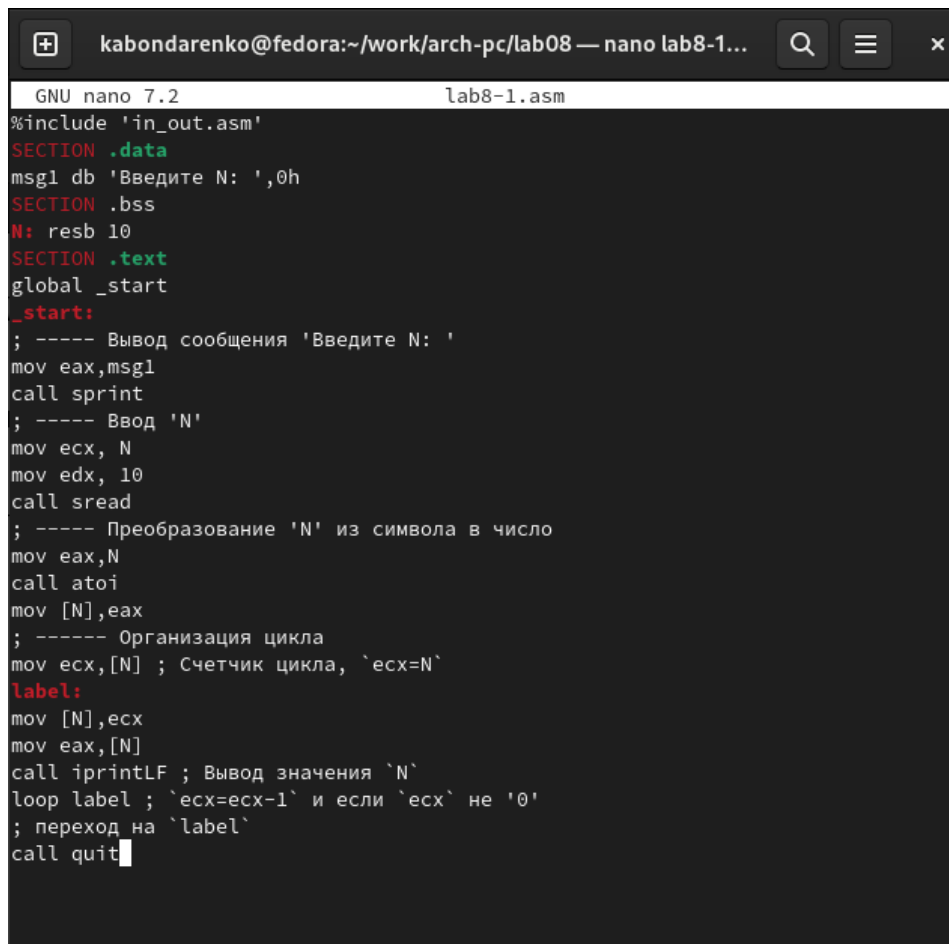
С помощью утилиты `mkdir` создаю директорию `lab08`, перехожу в нее и создаю файл для работы. (рис. [2.1])

A screenshot of a terminal window with a dark background. The window title is 'kabondarenko@fedora:~/work/arch-pc/lab08'. The terminal shows the following commands and their outputs: [kabondarenko@fedora ~]\$ mkdir ~/work/arch-pc/lab08, [kabondarenko@fedora ~]\$ cd ~/work/arch-pc/lab08, [kabondarenko@fedora lab08]\$ touch lab8-1.asm, and [kabondarenko@fedora lab08]\$ followed by a cursor.

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

Рис. 2.1: Создание директории

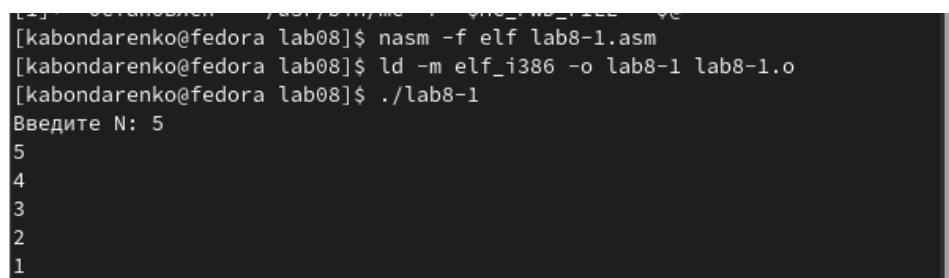
Открываю созданный файл `lab8-1.asm`, вставляю в него программу с использованием цикла для вывода чисел(рис. [2.2]).



```
GNU nano 7.2 lab8-1.asm
%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. 2.2: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. [2.3]).



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

Рис. 2.3: Запуск исполняемого файла

с помощью инструкции `sub` уменьшаю изначальный индекс на 1 единицу. (рис. [2.4]).

```

label:
sub ecx,1
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. 2.4: Уменьшение индекса

Создаю новый исполняемый файл программы и запускаю его (рис. [2.5]). Получаем результат отличный от ожидаемого

```

[kabondarenko@fedora lab08]$ nano lab8-1.asm
[kabondarenko@fedora lab08]$ nasm -f elf lab8-1.asm
[kabondarenko@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[kabondarenko@fedora lab08]$ ./lab8-1
Введите N: 10
9
7
5
3
1

```

Рис. 2.5: Запуск исполняемого файла

Изменяю текст программы так, чтобы получить нужный результат, используя стеки для запоминания данных. (рис. [2.6]).

```

label:
push ecx
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit

```

Рис. 2.6: Редактирование программы

Создаю исполняемый файл и проверяю работу программы (рис. [2.7]).

```
[kabondarenko@fedora lab08]$ nasm -f elf lab8-1.asm
[kabondarenko@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[kabondarenko@fedora lab08]$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[kabondarenko@fedora lab08]$
```

Рис. 2.7: Создание исполняемого файла

- Программа отработало верно.

Создаю новый файл lab8-2.asm для новой программы и вставляю программу, которая выводит все введенные пользователем аргументы (рис. [??]).

```
GNU nano 7.2 lab8-2.asm
#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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Создаю и запускаю новый исполняемый файл, проверяю работу программы (рис. [2.8]).


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

Рис. 2.8: Запуск исполняемого файла

- Программой было обработано 4 аргумента.
- Программа считает аргументами все символы до пробела, или значения, которые взяты в кавычки.*

Создаю новый файл lab8-3.asm и ввожу программу, которая складывает все числа введенные пользователем (рис. [??]).

```

GNU nano 7.2                                lab8-3.asm                                Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
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

```

^{^G} Справка ^{^O} Записать ^{^W} Поиск ^{^K} Вырезать ^{^T} Выполнить
^{^X} Выход ^{^R} ЧитФайл ^{^\} Замена ^{^U} Вставить ^{^J} Выводить

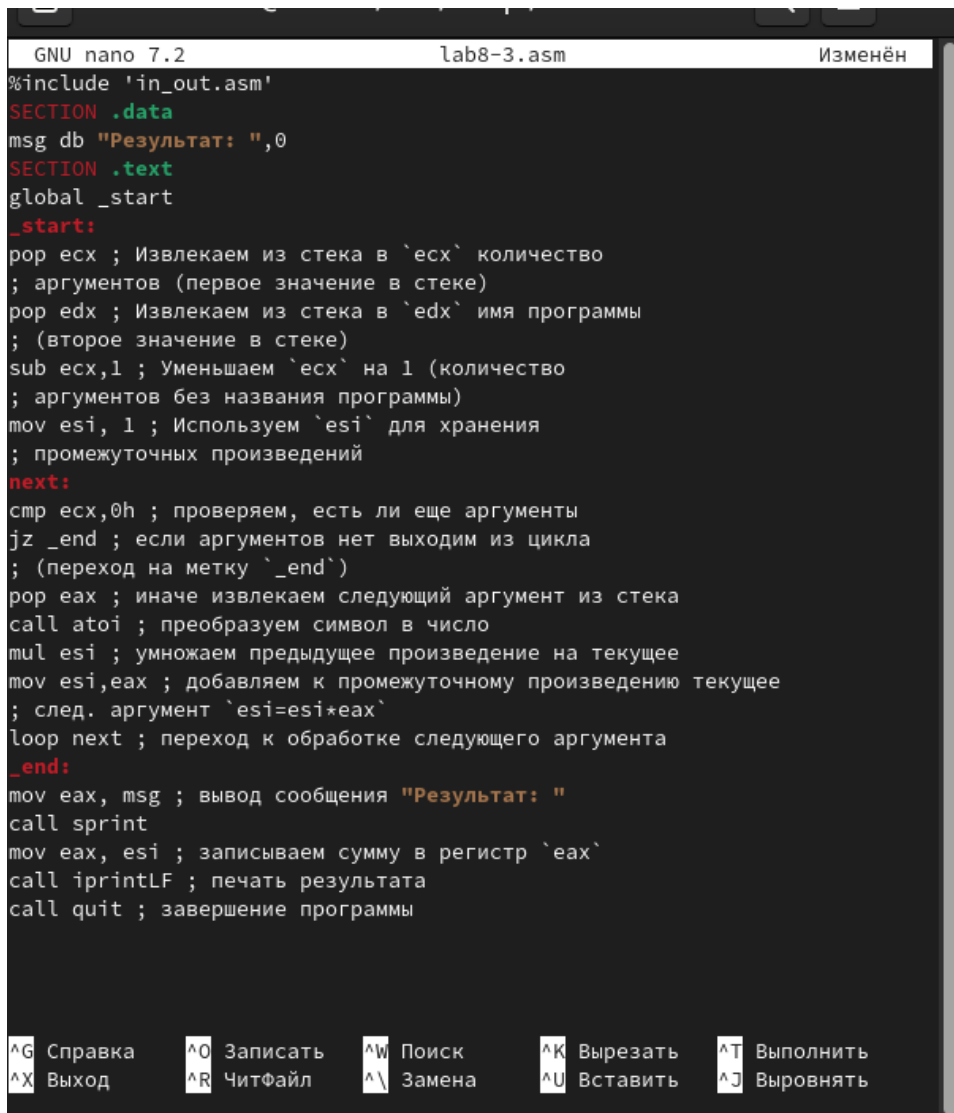
Запускаю исполняемый файл и проверяю работу программы (рис. [??]).

```

[kabondarenko@fedora lab08]$ nasm -f elf lab8-3.asm
[kabondarenko@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[kabondarenko@fedora lab08]$ ./lab8-3
Результат: 0
[kabondarenko@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[kabondarenko@fedora lab08]$

```

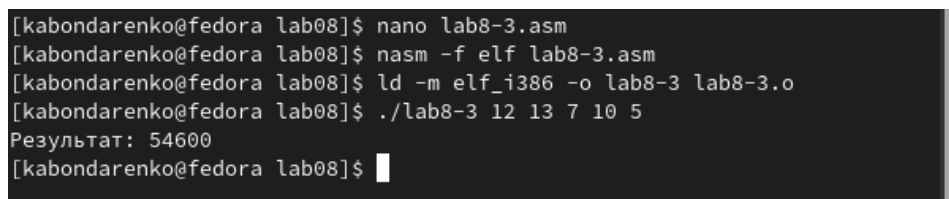
Изменяю текст программы так, чтобы она выводила произведение всех чисел, введенные пользователем. (рис. [??]).



```
GNU nano 7.2 lab8-3.asm Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mul esi ; умножаем предыдущее произведение на текущее
mov esi,eax ; добавляем к промежуточному произведению текущее
; 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

^G Справка      ^O Записать    ^W Поиск       ^K Вырезать    ^T Выполнить
^X Выход        ^R ЧитФайл    ^\ Замена      ^U Вставить    ^J Выровнять
```

Запускаю исполняемый файл и проверяю работу программы (рис. [2.9]).



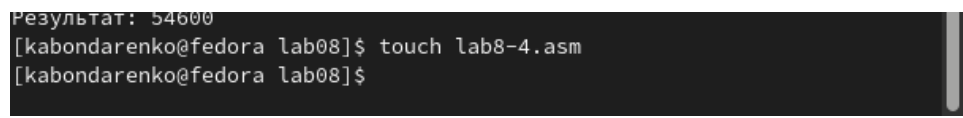
```
[kabondarenko@fedora lab08]$ nano lab8-3.asm
[kabondarenko@fedora lab08]$ nasm -f elf lab8-3.asm
[kabondarenko@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[kabondarenko@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 54600
[kabondarenko@fedora lab08]$
```

Рис. 2.9: Запуск программы

Программа отработала верно!

3 Самостоятельная работа

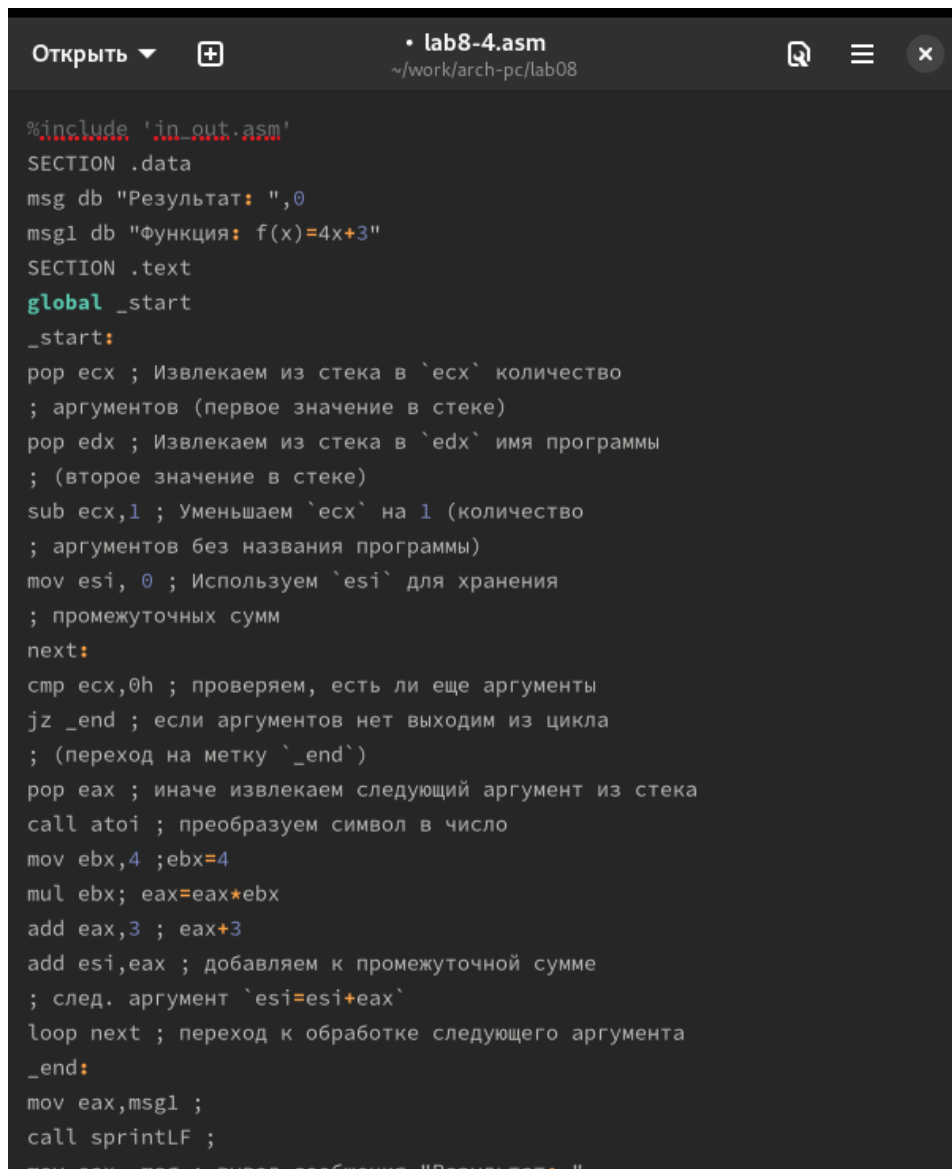
Создаю файл lab8-4.asm с помощью утилиты touch (рис. [3.1]).

A terminal window with a dark background. The first line shows the output of a command: "Результат: 54600". The second line shows a shell prompt "[kabondarenko@fedora lab08]\$" followed by the command "touch lab8-4.asm". The third line shows the prompt again after the command has been executed.

```
Результат: 54600
[kabondarenko@fedora lab08]$ touch lab8-4.asm
[kabondarenko@fedora lab08]$
```

Рис. 3.1: Создание файла

Ввожу в созданный файл текст программы, у, которая находит сумму значений функции (2 Вариант) $f(x)=4x+3$ для всех аргументов x , введенные пользователем.(рис. [3.2]).




```
Открыть ▾  • lab8-4.asm  
~/work/arch-pc/lab08  
  
%include 'in_out.asm'  
SECTION .data  
msg db "Результат: ",0  
msg1 db "Функция: f(x)=4x+3"  
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  
mov ebx,4 ; ebx=4  
mul ebx; eax=eax*ebx  
add eax,3 ; eax+3  
add esi,eax ; добавляем к промежуточной сумме  
; след. аргумент `esi=esi+eax`  
loop next ; переход к обработке следующего аргумента  
_end:  
mov eax,msg1 ;  
call sprintf ;  
mov eax,msg ; вывод сообщения "Результат: "
```

Рис. 3.2: Редактирование файла

Создаю исполняемый файл и запускаю его, при $x = 1, 2, 3, 4, 5, 6$ (рис. [3.3]).

```

[kabondarenko@fedora lab08]$ nasm -f elf lab8-4.asm
[kabondarenko@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[kabondarenko@fedora lab08]$ ./lab8-4 1 2 3 4 5 6
Функция: f(x)=4x+3
Результат: 102
[kabondarenko@fedora lab08]$

```

Рис. 3.3: Запуск исполняемого файла

Текст программы

```

#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=4x+3"

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
mov ebx,4 ;ebx=4

```

```
mul ebx; eax=eax*ebx
add eax,3 ; eax+3
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax,msg1 ;
call sprintfLF ;
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call fprintfLF ; печать результата
call quit ; завершение программы
```

4 Вывод

В ходе выполнения работы я получила навыки по организации циклов и по работе со стеком на языке NASM.