

# Шаблон отчёта по лабораторной работе

8

Баптишта Матеуж

# 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>9</b>
4.1	Реализация циклов в NASM. . . . .	9
4.2	Обработка аргументов командной строки. ....	14
4.3	Задание для самостоятельной работы.....	19
<b>5</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	создание файлов . . . . .	9
4.2	ввод текста.....	10
4.3	запуск исполняемого файла .....	11
4.4	изменение текста программы.....	12
4.5	запуск обновленной файла.....	12
4.6	изменение текста программы.....	13
4.7	запуск исполняемого файла .....	14
4.8	ввод текста.....	15
4.9	запуск исполняемого файла .....	15
4.10	ввод текста.....	16
4.11	запуск исполняемого файла .....	17
4.12	изменение текста программы.....	18
4.13	запуск исполняемого файла .....	18
4.14	текст программы.....	19
4.15	запуск исполняемого файла .....	20

## Список таблиц

# 1 Цель работы

- Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

## 2 Задание

- Реализация циклов в NASM.
- Обработка аргументов командной строки.
- 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.
- 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.
- 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.

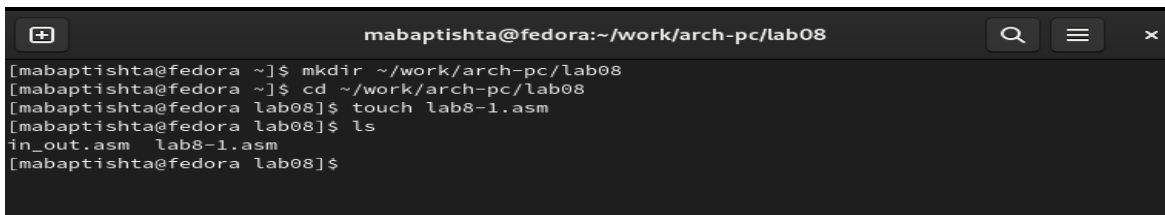
- Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл.



## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM.

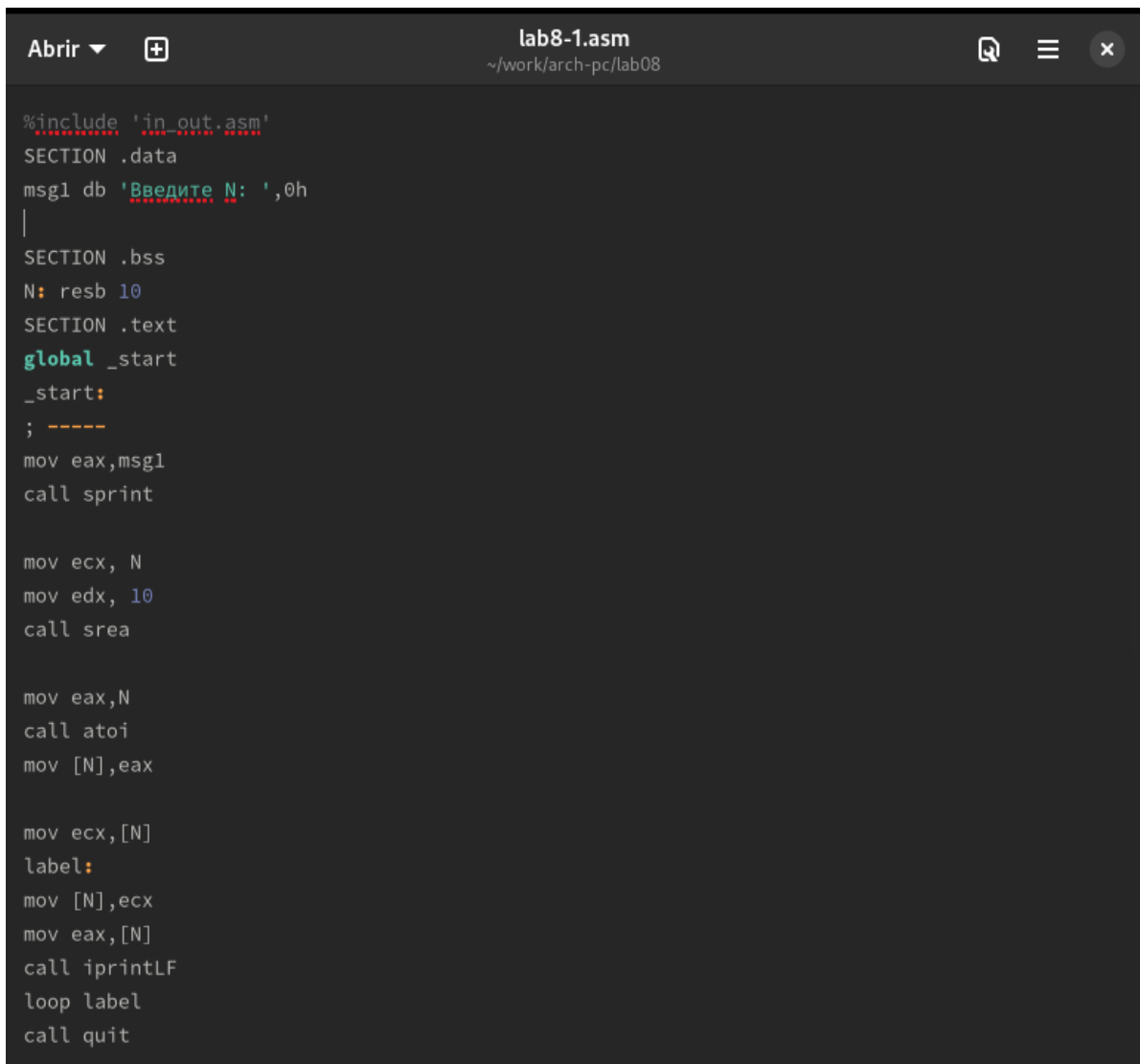
- Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm.(рис.[4.1]).



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

Рис. 4.1: создание файлов

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

A screenshot of a code editor window titled 'lab8-1.asm' with a path '~/.work/arch-pc/lab08'. The editor contains assembly code for a program that reads a string 'N' and prints it. The code includes sections for data, bss, and text, and uses various assembly instructions like 'mov', 'call', and 'loop'.

```
Abrir ▾ + 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:
; -----
mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call srea

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
call quit
```

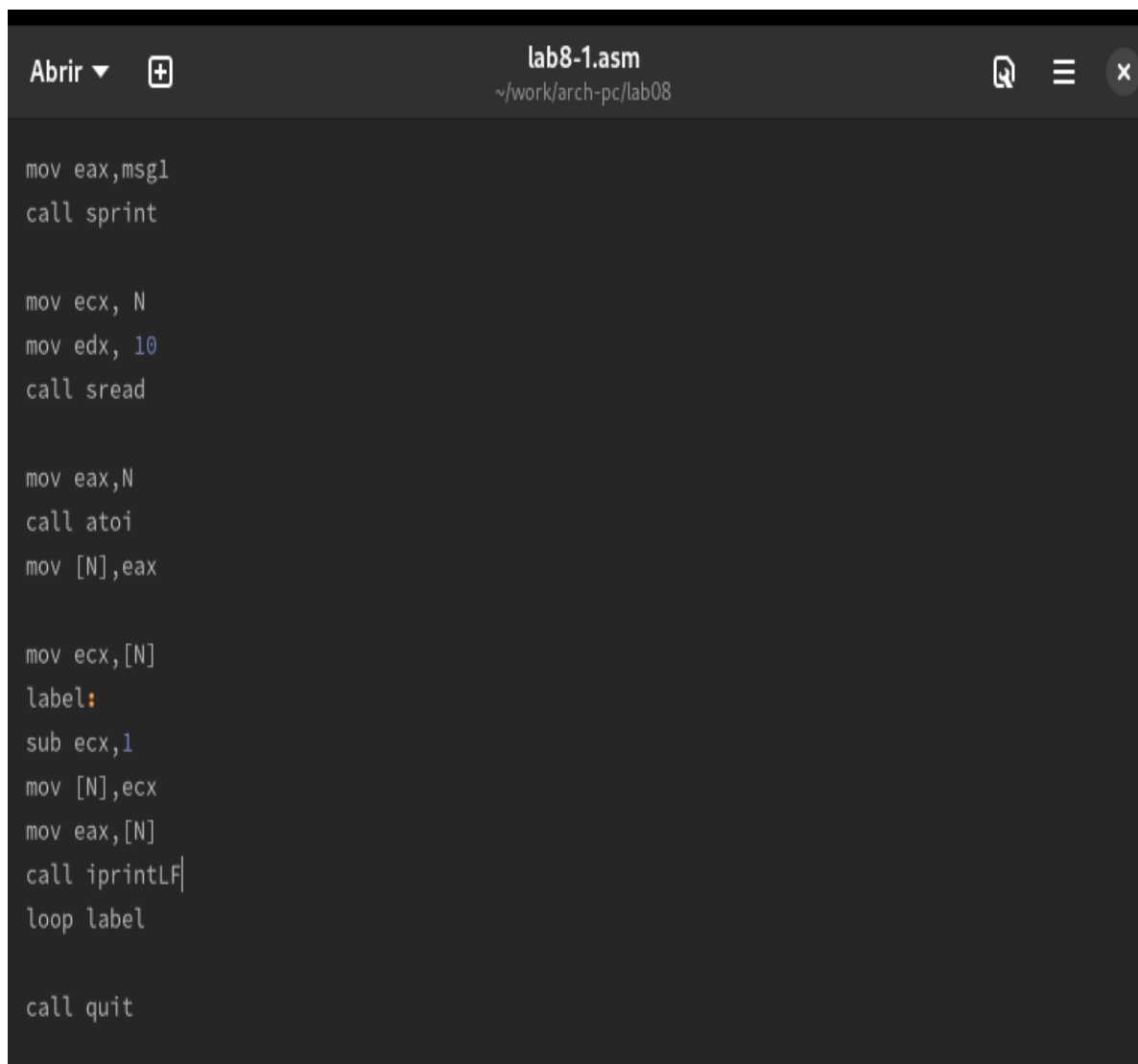
Рис. 4.2: ввод текста

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

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

Рис. 4.3: запуск исполняемого файла

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

A screenshot of a text editor window titled 'lab8-1.asm' with a path '~/.work/arch-pc/lab08'. The editor contains the following assembly code:

```
mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

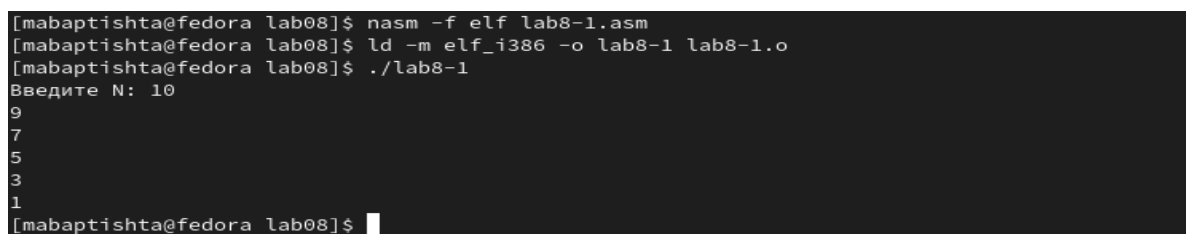
mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

call quit
```

Рис. 4.4: изменение текста программы

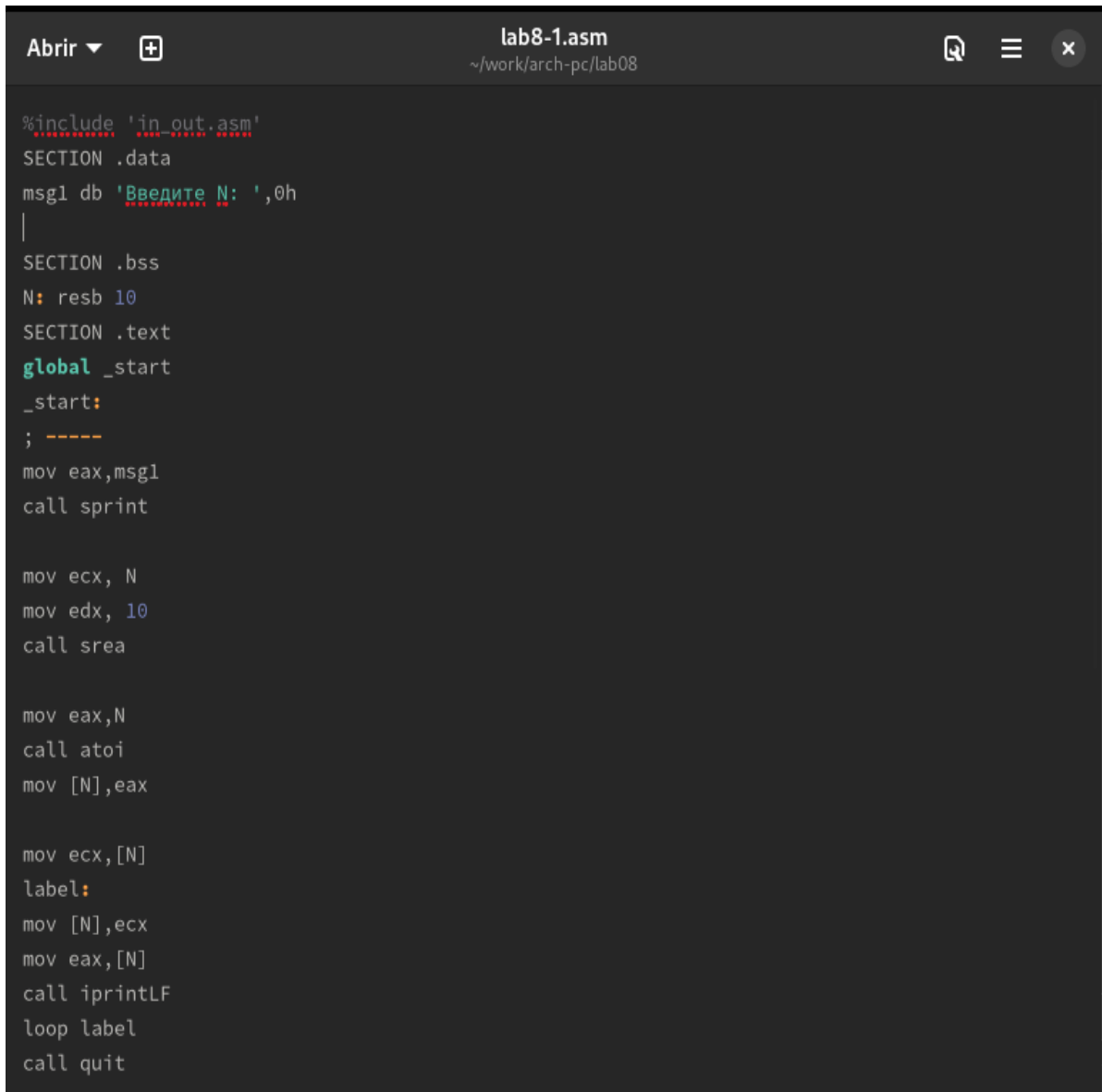
- Создаю исполняемый файл и проверяю его работу. (рис.[4.5]).

A screenshot of a terminal window showing the compilation and execution of the assembly program. The commands and output are as follows:

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

Рис. 4.5: запуск обновленной файла

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



```
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:
; -----
mov eax,msg1
call sprint

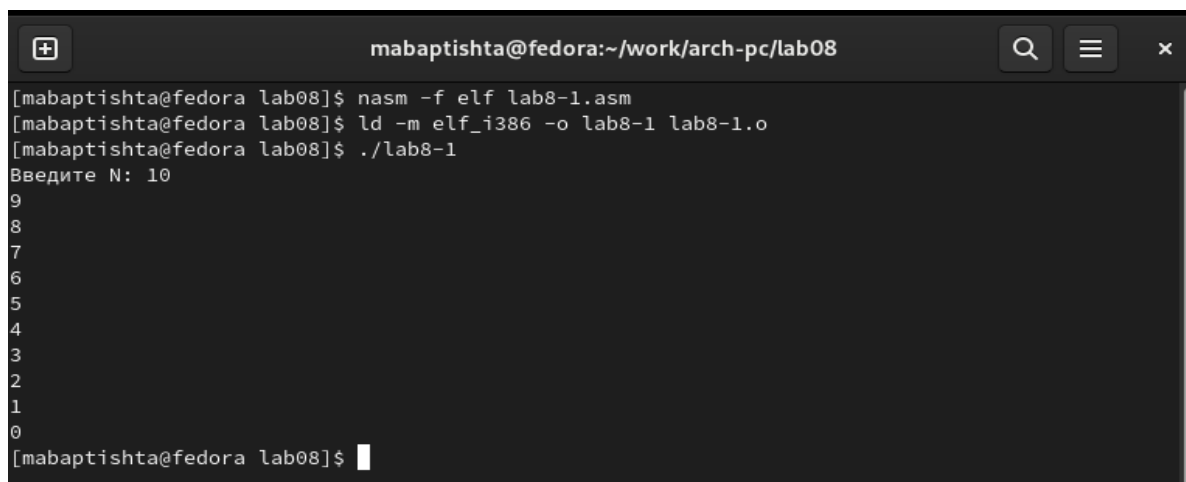
mov ecx, N
mov edx, 10
call srea

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
call quit
```

Рис. 4.6: изменение текста программы

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

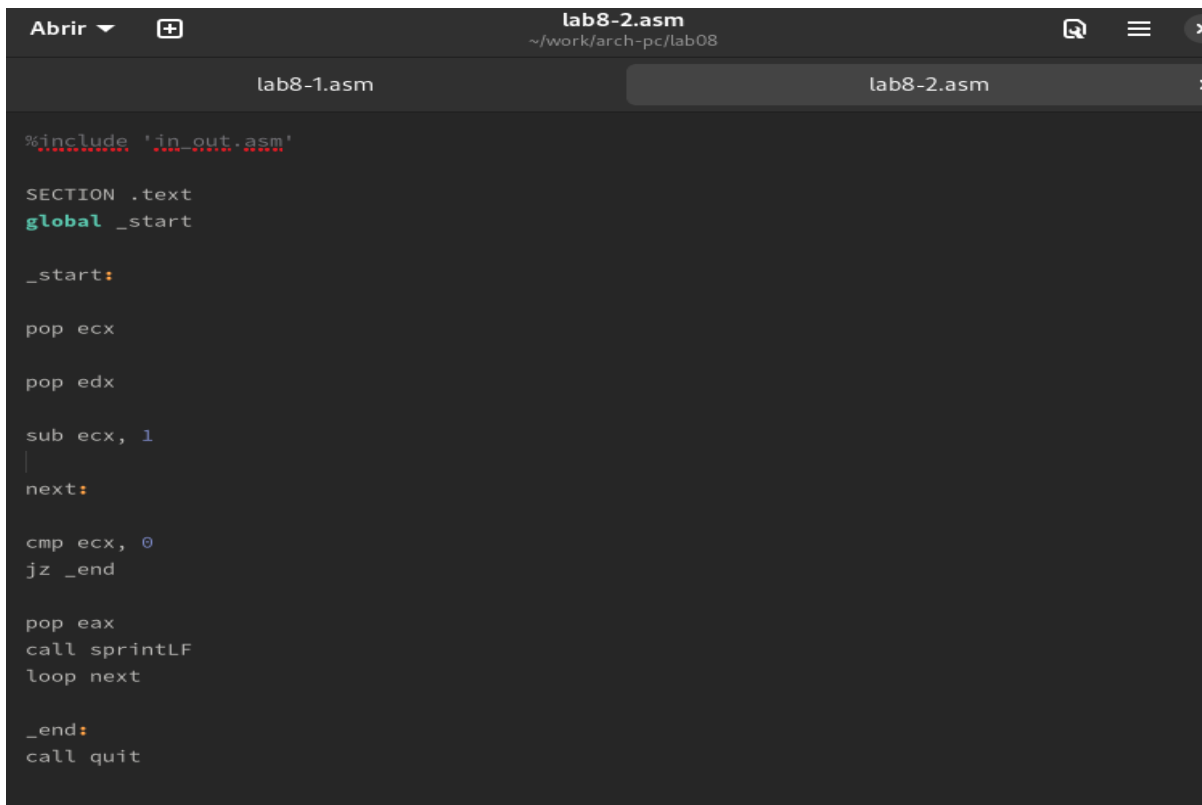


```
mabaptishta@fedora:~/work/arch-pc/lab08
[mabaptishta@fedora lab08]$ nasm -f elf lab8-1.asm
[mabaptishta@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[mabaptishta@fedora lab08]$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[mabaptishta@fedora lab08]$
```

Рис. 4.7: запуск исполняемого файла

## 4.2 Обработка аргументов командной строки.

- На этом шаге мы создали файл `lab8-2.asm`, затем заполнили в нем наш код. (рис.[4.8]).



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

lab8-1.asm lab8-2.asm

%include 'in_out.asm'

SECTION .text
global _start

_start:

pop ecx

pop edx

sub ecx, 1
|
next:

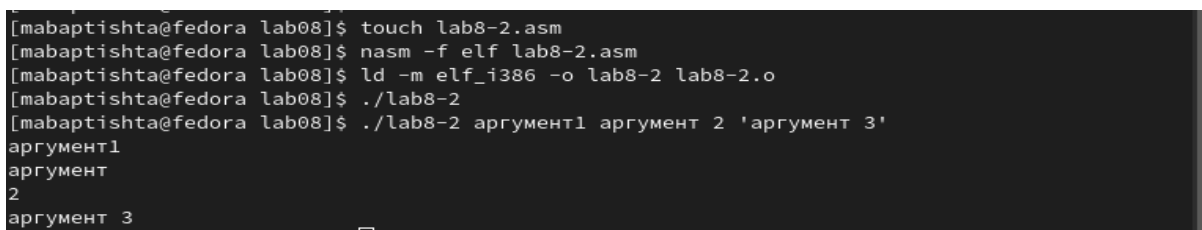
cmp ecx, 0
jz _end

pop eax
call sprintLF
loop next

_end:
call quit
```

Рис. 4.8: ввод текста

- Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис.[4.9]).

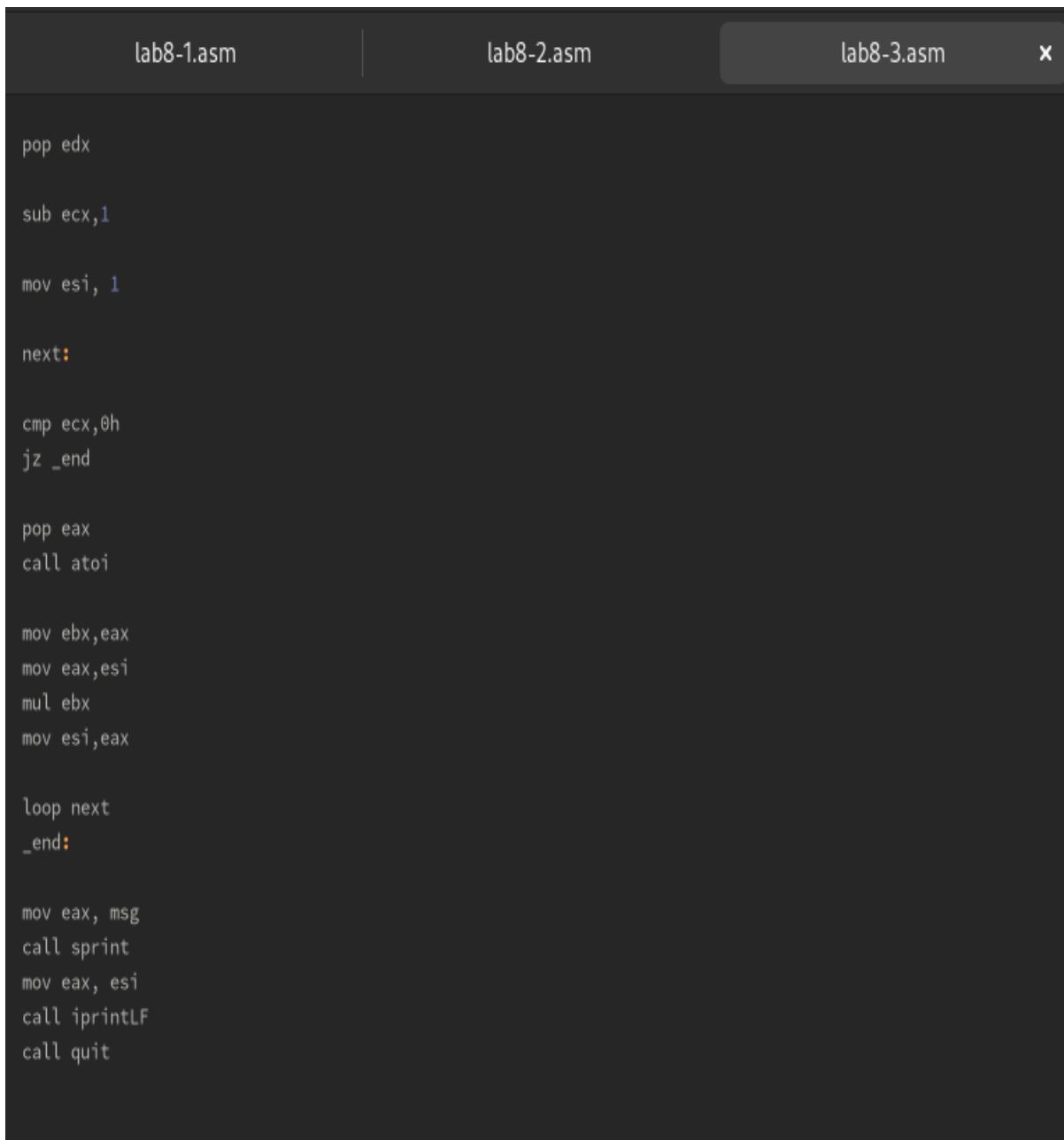


```
[mabaptishta@fedora lab08]$ touch lab8-2.asm
[mabaptishta@fedora lab08]$ nasm -f elf lab8-2.asm
[mabaptishta@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[mabaptishta@fedora lab08]$ ./lab8-2
[mabaptishta@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3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

Рис. 4.9: запуск исполняемого файла

- И, как вы можете видеть, на этот раз при запуске программы мы добавили в команду три аргумента, и в этом случае были обработаны три аргумента

- Первым делом мы создали файл lab8-3.asm, затем заполнили кодом программы. (рис.[4.10]).



```
lab8-1.asm | lab8-2.asm | lab8-3.asm x
pop edx
sub ecx,1
mov esi, 1
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 4.10: ввод текста

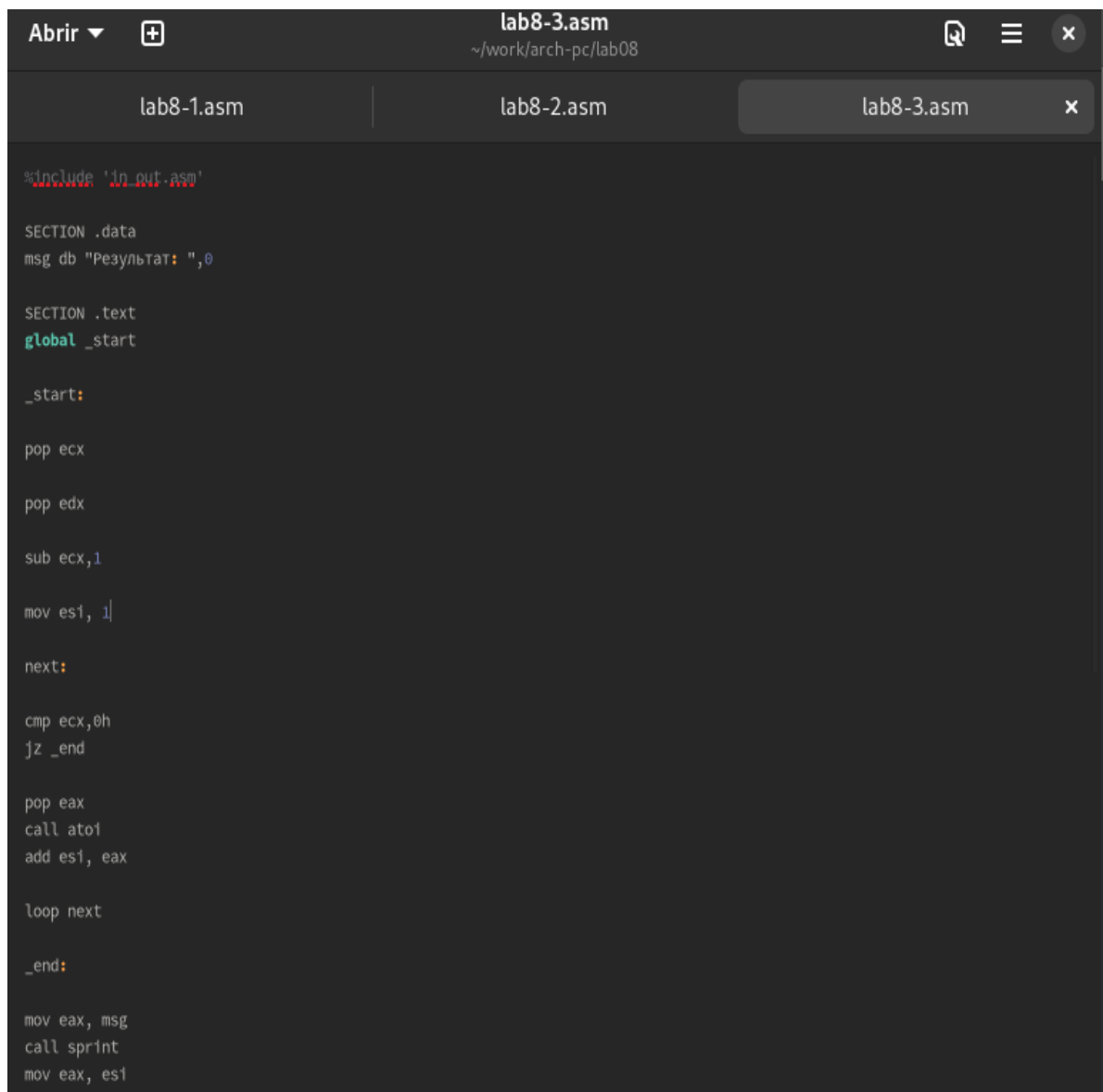
- Создаю исполняемый файл и запускаю его, указав аргументы.(рис.[4.11]).



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

Рис. 4.11: запуск исполняемого файла

- Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис.[4.12]).



```
lab8-3.asm
~/work/arch-pc/lab08

lab8-1.asm | lab8-2.asm | lab8-3.asm x

%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start

_start:

pop ecx

pop edx

sub ecx,1

mov esi, 1

next:

cmp ecx,0h
jz _end

pop eax
call atoi
add esi, eax

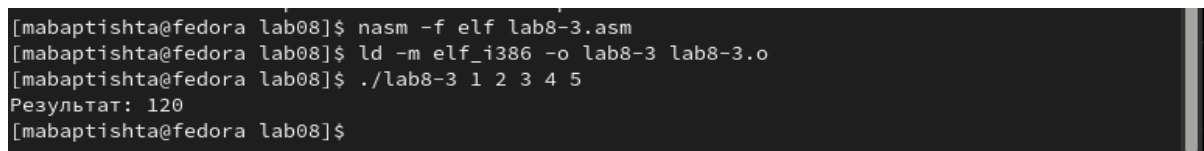
loop next

_end:

mov eax, msg
call sprint
mov eax, esi
```

Рис. 4.12: изменение текста программы

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

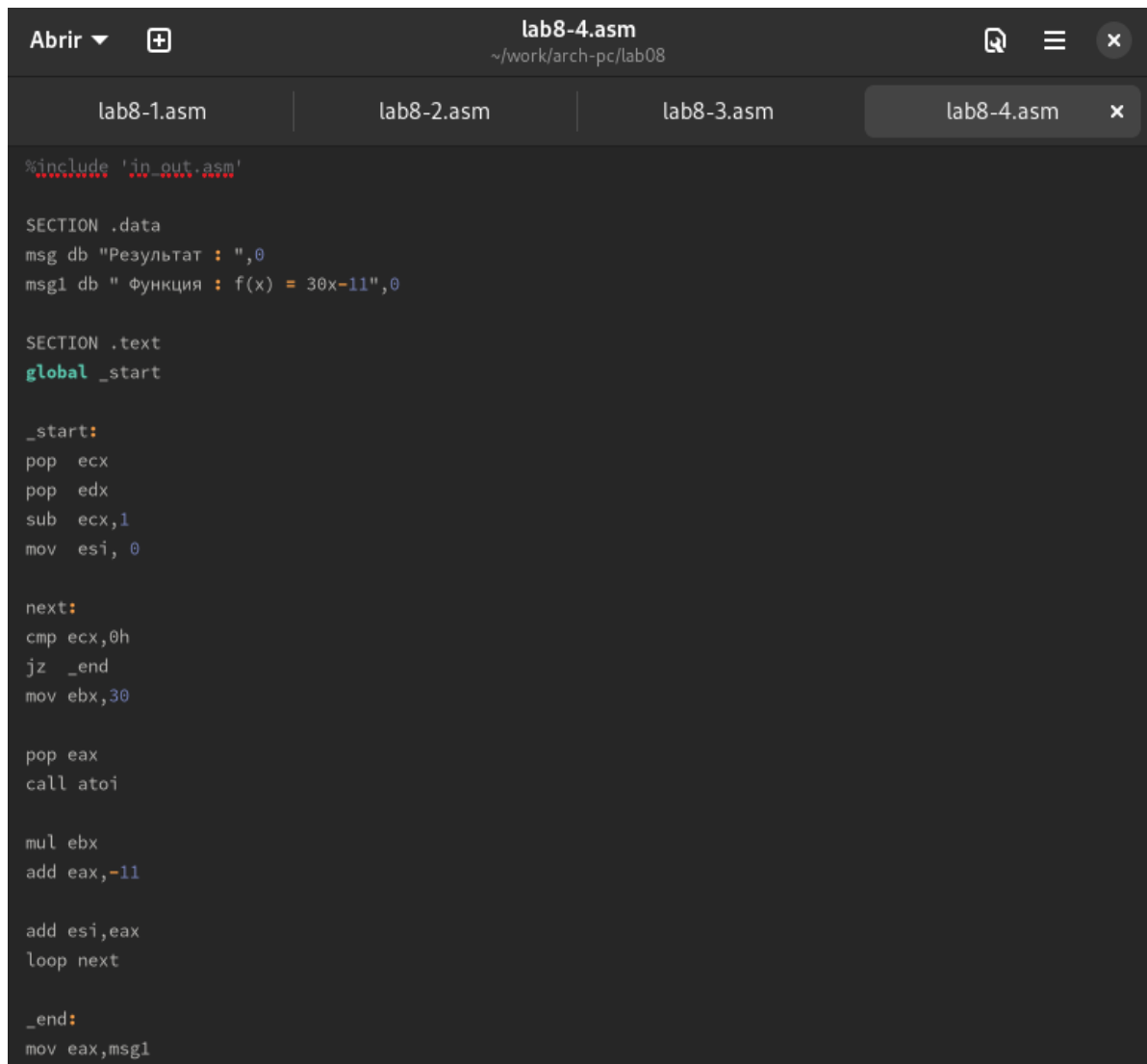


```
[mabaptishta@fedora lab08]$ nasm -f elf lab8-3.asm
[mabaptishta@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[mabaptishta@fedora lab08]$ ./lab8-3 1 2 3 4 5
Результат: 120
[mabaptishta@fedora lab08]$
```

Рис. 4.13: запуск исполняемого файла

## 4.3 Задание для самостоятельной работы.

- В этой части мы должны были написать программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$
- сначала мы создали наш файл lab8-4.asm, где будет находиться наш код, затем мы написали программу. (рис.[4.14]).



```
lab8-4.asm
~/work/arch-pc/lab08

lab8-1.asm | lab8-2.asm | lab8-3.asm | lab8-4.asm x

%include 'in_out.asm'

SECTION .data
msg db "Результат : ",0
msg1 db " Функция : f(x) = 30x-11",0

SECTION .text
global _start

_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
mov ebx,30

pop eax
call atoi

mul ebx
add eax,-11

add esi,eax
loop next

_end:
mov eax,msg1
```

Рис. 4.14: текст программы

- Создаю исполняемый файл и проверьте его работу на нескольких наборах  $x$

=  $x_1, x_2, \dots, x_n$ . (рис.[4.15]).

```
[mabaptishta@fedora lab08]$ touch lab8-4.asm
[mabaptishta@fedora lab08]$ nasm -f elf lab8-4.asm
[mabaptishta@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[mabaptishta@fedora lab08]$ ./lab8-4 1 2 3 4
Функция : f(x) = 30x-11
Результат : 256
[mabaptishta@fedora lab08]$
```

Рис. 4.15: запуск исполняемого файла

## 5 Выводы

- Благодаря этой лабораторной работе мы научились писать программы с использованием циклов и обработки аргументов командной строки, что поможет нам в дальнейшей лабораторной работе.

# Список литературы

::: {#refs} :::