

# **Отчет по лабораторной работе №8**

**Программирование цикла. Обработка аргументов командной строки.**

Симонова Полина Игоревна

# 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>7</b>
4.1	Реализация циклов в NASM . . . . .	7
4.2	Обработка аргументов командной строки . . . . .	10
4.3	Задание для самостоятельной работы . . . . .	13
<b>5</b>	<b>Выводы</b>	<b>16</b>
<b>6</b>	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание каталога . . . . .	7
4.2	Копирование программы из листинга . . . . .	8
4.3	Запуск программы . . . . .	8
4.4	Изменение программы . . . . .	9
4.5	Запуск измененной программы . . . . .	9
4.6	Добавление push и pop в цикл программы . . . . .	10
4.7	Запуск измененной программы . . . . .	10
4.8	Копирование программы из листинга . . . . .	11
4.9	Запуск второй программы . . . . .	11
4.10	Копирование программы из третьего листинга . . . . .	12
4.11	Запуск третьей программы . . . . .	12
4.12	Изменение третьей программы . . . . .	13
4.13	Запуск измененной третьей программы . . . . .	13
4.14	Запуск программы для самостоятельной работы . . . . .	15

# 1 Цель работы

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

## 2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Задание для самостоятельной работы

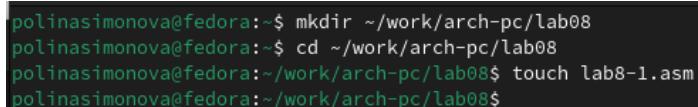
### 3 Теоретическое введение

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.

## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

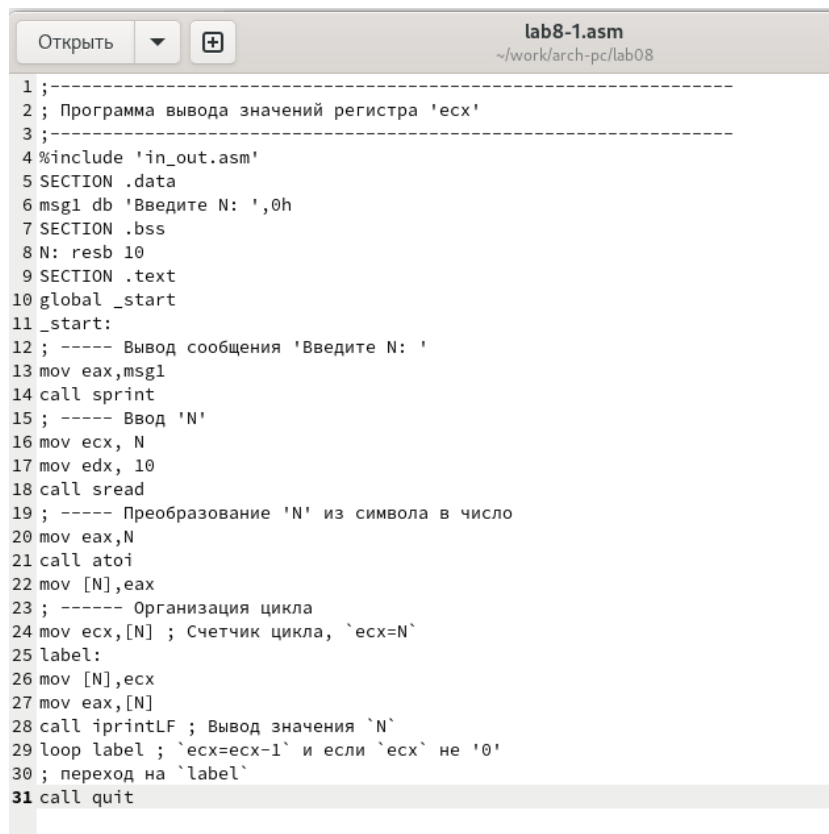
Создаю каталог для программ лабораторной работы №8 и файл lab8-1.asm, куда буду копировать программу из листинга. (рис. -fig. 4.1).



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

Рис. 4.1: Создание каталога

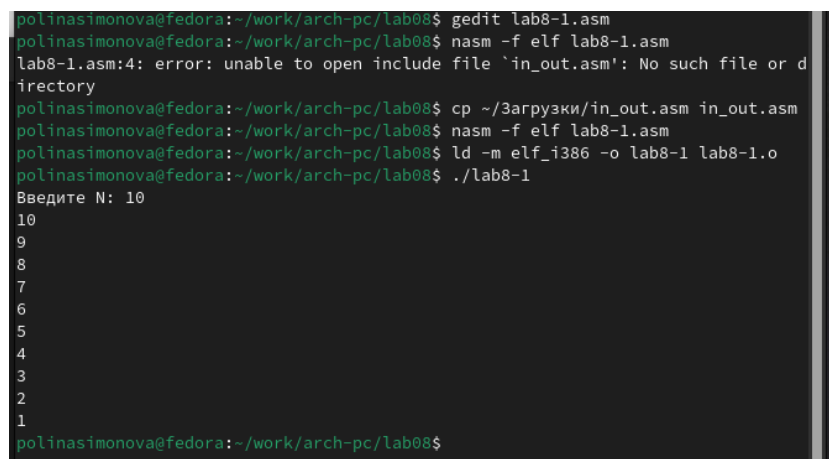
Копирую в созданный файл программу из листинга. (рис. -fig. 4.2).



```
1 ;-----  
2 ; Программа вывода значений регистра 'ecx'  
3 ;-----  
4 %include 'in_out.asm'  
5 SECTION .data  
6 msg1 db 'Введите N: ',0h  
7 SECTION .bss  
8 N: resb 10  
9 SECTION .text  
10 global _start  
11 _start:  
12 ; ----- Вывод сообщения 'Введите N: '  
13 mov eax,msg1  
14 call sprint  
15 ; ----- Ввод 'N'  
16 mov ecx, N  
17 mov edx, 10  
18 call sread  
19 ; ----- Преобразование 'N' из символа в число  
20 mov eax,N  
21 call atoi  
22 mov [N],eax  
23 ; ----- Организация цикла  
24 mov ecx,[N] ; Счетчик цикла, `ecx=N`  
25 label:  
26 mov [N],ecx  
27 mov eax,[N]  
28 call iprintLF ; Вывод значения `N`  
29 loop label ; `ecx=ecx-1` и если `ecx` не `0`  
30 ; переход на `label`  
31 call quit
```

Рис. 4.2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. -fig. 4.3).



```
polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-1.asm  
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
lab8-1.asm:4: error: unable to open include file `in_out.asm': No such file or d  
irectory  
polinasimonova@fedora:~/work/arch-pc/lab08$ cp ~/3арпузки/in_out.asm in_out.asm  
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o  
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 10  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
polinasimonova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск программы

Изменяю изначальную программу так, что в теле цикла я изменяю значение регистра ecx (рис. -fig. 4.4).



```

4 %include 'in_out.asm'
5
6 SECTION .data
7 msg1 db 'Введите N: ',0h
8
9 SECTION .bss
10 N: resb 10
11
12 SECTION .text
13 global _start
14 _start:
15
16 ; ----- Вывод сообщения 'Введите N: '
17 mov eax,msg1
18 call sprint
19
20 ; ----- Ввод 'N'
21 mov ecx, N
22 mov edx, 10
23 call sread
24
25 ; ----- Преобразование 'N' из символа в число
26 mov eax,N
27 call atoi
28 mov [N],eax
29
30 ; ----- Организация цикла
31 mov ecx,[N] ; Счетчик цикла, `ecx=N`
32
33 label:
34 sub ecx, 1
35 mov [N],ecx
36 mov eax,[N]
37 call iprintLF ; Вывод значения `N`
38 loop label ; `ecx=ecx-1` и если `ecx` не '0'
39 ; переход на `label`
40 call quit

```

Рис. 4.4: Изменение программы

Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. -fig. 4.5).

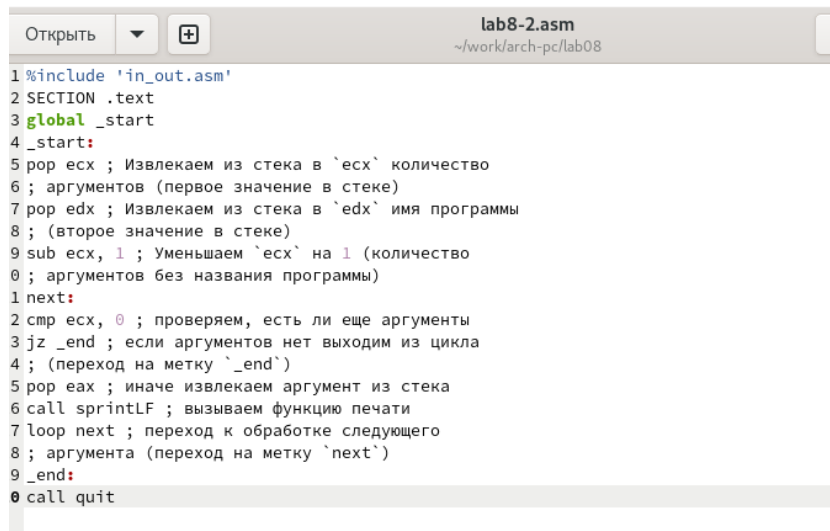
```

polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-1.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
polinasimonova@fedora:~/work/arch-pc/lab08$

```

Рис. 4.5: Запуск измененной программы

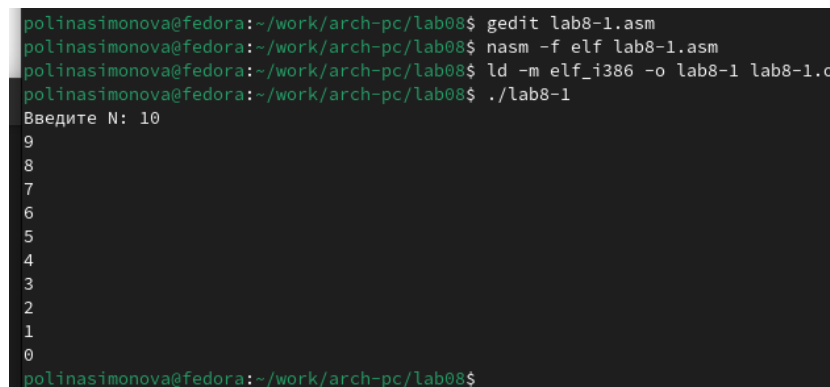
Добавляю команды push и pop в программу (рис. -fig. 4.6).



```
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
0 ; аргументов без названия программы)
1 next:
2 cmp ecx, 0 ; проверяем, есть ли еще аргументы
3 jz _end ; если аргументов нет выходим из цикла
4 ; (переход на метку `_end`)
5 pop eax ; иначе извлекаем аргумент из стека
6 call sprintf ; вызываем функцию печати
7 loop next ; переход к обработке следующего
8 ; аргумента (переход на метку `next`)
9 _end:
0 call quit
```

Рис. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. -fig. 4.7).



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

Рис. 4.7: Запуск измененной программы

## 4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. -fig. 4.8).

```

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. 4.8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. -fig. 4.9).

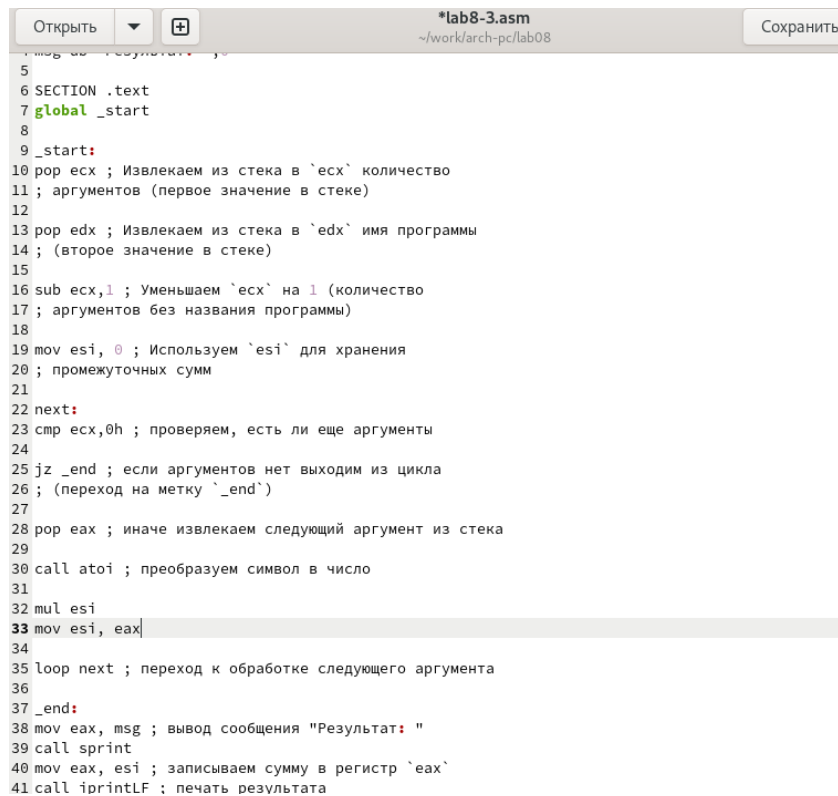
```

polinasimonova@fedora:~/work/arch-pc/lab08$ touch lab8-2.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-2.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg 2 'arg 3'
arg1
arg
2
arg 3
polinasimonova@fedora:~/work/arch-pc/lab08$

```

Рис. 4.9: Запуск второй программы

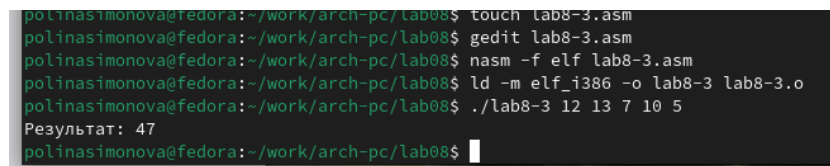
Создаю новый файл для программы и копирую в него код из третьего листинга (рис. -fig. 4.10).



```
5
6 SECTION .text
7 global _start
8
9 _start:
10 pop ecx ; Извлекаем из стека в `ecx` количество
11 ; аргументов (первое значение в стеке)
12
13 pop edx ; Извлекаем из стека в `edx` имя программы
14 ; (второе значение в стеке)
15
16 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
17 ; аргументов без названия программы)
18
19 mov esi, 0 ; Используем `esi` для хранения
20 ; промежуточных сумм
21
22 next:
23 cmp ecx,0h ; проверяем, есть ли еще аргументы
24
25 jz _end ; если аргументов нет выходим из цикла
26 ; (переход на метку `_end`)
27
28 pop eax ; иначе извлекаем следующий аргумент из стека
29
30 call atoi ; преобразуем символ в число
31
32 mul esi
33 mov esi, eax
34
35 loop next ; переход к обработке следующего аргумента
36
37 _end:
38 mov eax, msg ; вывод сообщения "Результат: "
39 call sprint
40 mov eax, esi ; записываем сумму в регистр `eax`
41 call iprintLF ; печать результата
```

Рис. 4.10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. -fig. 4.11).



```
polinasimonova@fedora:~/work/arch-pc/lab08$ touch lab8-3.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-3.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
polinasimonova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. -fig. 4.12).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 3 * (x + 2)", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11     mov eax, msg_func
12     call sprintf
13
14     pop ecx
15     pop edx
16     sub ecx, 1
17     mov esi, 0
18
19 next:
20     cmp ecx, 0h
21     jz _end
22     pop eax
23     call atoi
24
25     ; Новая формула для f(x) = 3 * (x + 2)
26     add eax, 2 ; Сначала добавляем 2 (x + 2)
27     mov ebx, 3 ; Устанавливаем множитель 3
28     mul ebx ; Умножаем на 3
29
30     add esi, eax
31
32     loop next
33
34 _end:
35     mov eax, msg_result
36     call sprintf
37     mov eax, esi

```

Рис. 4.12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 4.13).

```

polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-3.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
polinasimonova@fedora:~/work/arch-pc/lab08$

```

Рис. 4.13: Запуск измененной третьей программы

## 4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумму значений для функции  $f(x) = 3(x+2)$ , которая совпадает с моим вариантом - 7.

Код программы:

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

## SECTION .data

msg\_func db "Функция:  $f(x) = 3 * (x + 2)$ ", 0

msg\_result db "Результат: ", 0

## SECTION .text

GLOBAL \_start

\_start:

mov eax, msg\_func

call sprintf

pop ecx

pop edx

sub ecx, 1

mov esi, 0

next:

cmp ecx, 0h

jz \_end

pop eax

call atoi

*; Новая формула для  $f(x) = 3 * (x + 2)$*

add eax, 2 *; Сначала добавляем 2 (x + 2)*

mov ebx, 3 *; Устанавливаем множитель 3*

mul ebx *; Умножаем на 3*

add esi, eax

```
loop next
```

```
_end:
```

```
mov eax, msg_result
```

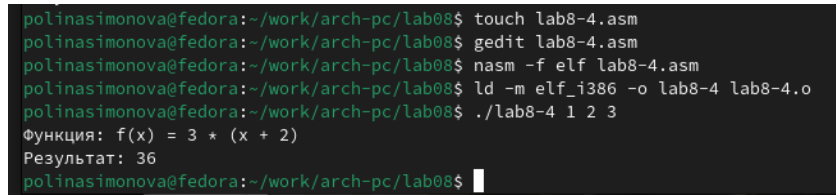
```
call sprint
```

```
mov eax, esi
```

```
call iprintLF
```

```
call quit
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. -fig. 4.14).



```
polinasimonova@fedora:~/work/arch-pc/lab08$ touch lab8-4.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ gedit lab8-4.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
polinasimonova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
polinasimonova@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3
Функция: f(x) = 3 * (x + 2)
Результат: 36
polinasimonova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.14: Запуск программы для самостоятельной работы

## **5 Выводы**

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов, а также научилась обрабатывать аргументы командной строки.



## **6 Список литературы**

1. Курс на ТУИС
2. Лабораторная работа №8