**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

*дисциплина: Архитектура компьютера*

Студент: Крыловецкий Денис Витальевич

Группа: НКАбд-03-25

**МОСКВА**

2025 г.

Оглавление

**Список иллюстраций**

## 1 Цель работы

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

## 2 Задания

1. Реализация циклом в NASM

2. Обработка аргументов командной строки.

3. Самостоятельное написание программы по материалам лабораторной 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 реализована 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 работы

### Реализация циклов в NASM

Создадим каталог для программам лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рисунок 0.1-0.3)

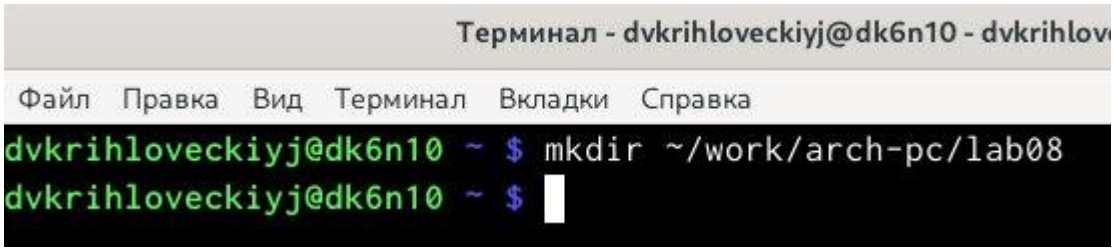


**Рисунок 0.1 создание каталога**

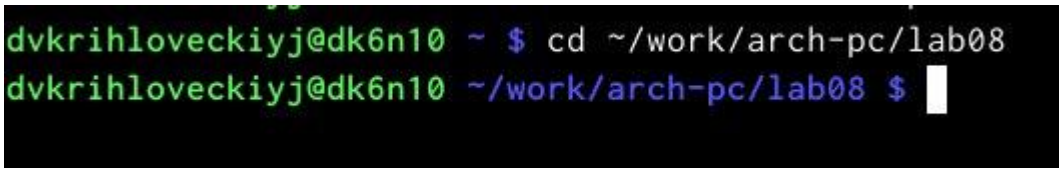


**Рисунок 0.2 переход в каталог**

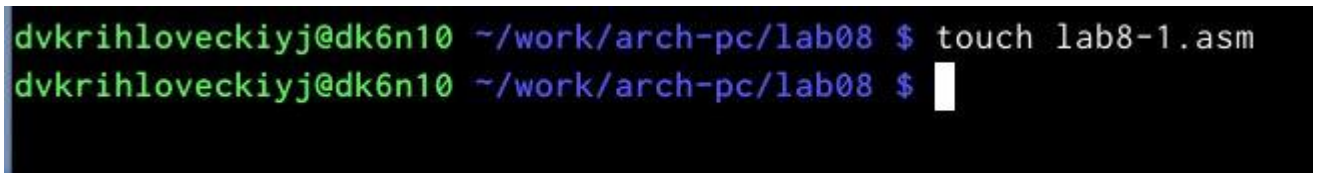


**Рисунок 0.3 создание файла lab8-1.asm**

Введем в файл lab8-1.asm текст программы из листинга (рисунок 0.4)

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
```

Открыть | lab8-1.asm ~/work/arch-pc/lab08 | Сохранить

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1 db 'Введите N: ',0h
5 SECTION .bss
6 N: resb 10
7 SECTION .text
8 global _start
9 _start:
10 ; ----- Вывод сообщения 'Введите N: '
11 mov eax,msg1
12 call sprint
13 ; ----- Ввод 'N'
14 mov ecx, N
15 mov edx, 10
16 call sread
17 ; ----- Преобразование 'N' из символа в число
18 mov eax,N
19 call atoi
20 mov [N],eax
21 ; ------ Организация цикла
22 mov ecx,[N] ; Счетчик цикла, `ecx=N`
23 label:
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
```

**Рисунок 0.4 запись программы**

Создадим исполняемый файл и проверим его работу (рисунок 0.5-0.8)

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.5 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.6 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N:
```

**Рисунок 0.7 запуск исполняемого файла**

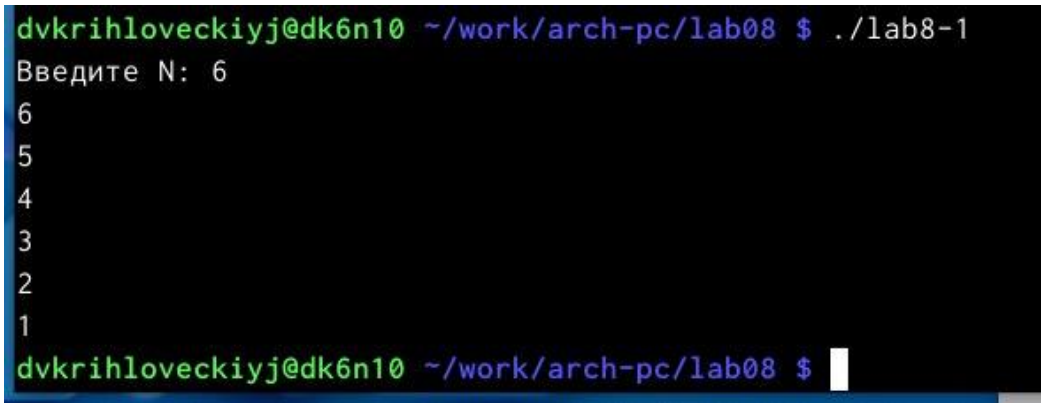


**Рисунок 0.8 проверка работы программы**

Изменим текст программы добавив изменение значения регистра ecx в цикле (рисунок 0.9-0.12)

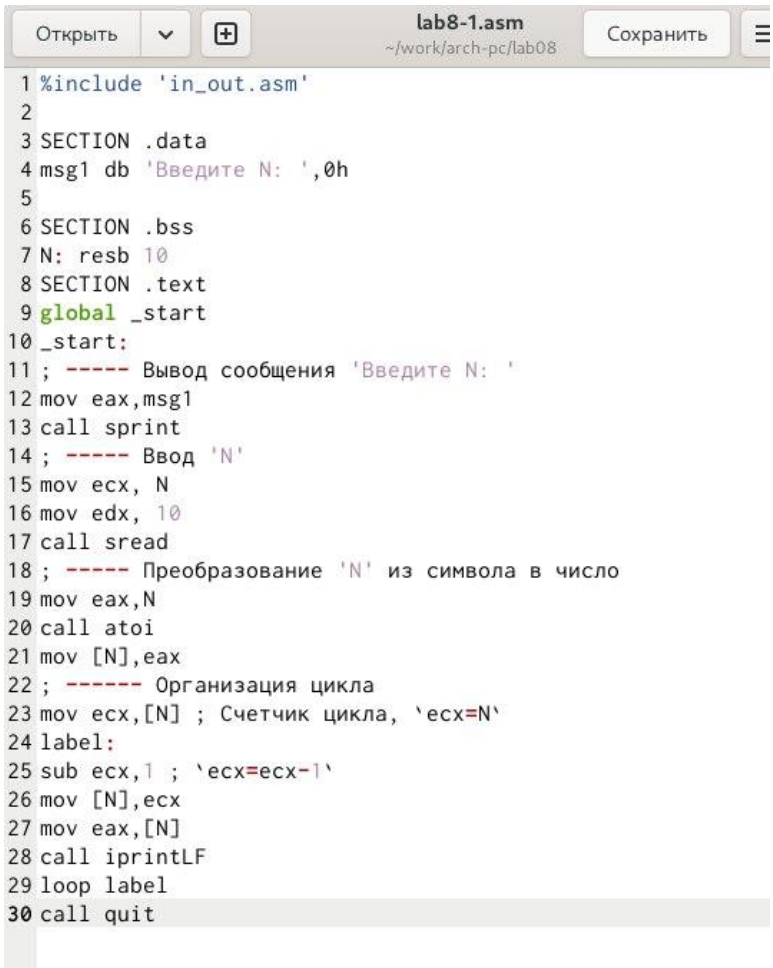


**Рисунок 0.9 изменение программы**

Создадим исполняемый файл и проверьте его работу

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
```

**Рисунок 0.10 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
```

**Рисунок 0.11 запуск исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
1
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.12 проверка работы программы**

Количество итераций уменьшается вдвое ввиду того, что регистр ecx на каждой итерации стал меньше на 2 значения.

Внесем изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop (рисунок 0.13)

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

```
%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 значения 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
```

**Рисунок 0.13 изменение программы**

Создадим исполняемый файл и проверим его работу (рисунок 0.14-0.16)

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
```

**Рисунок 0.14 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.15 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 8
7
6
5
4
3
2
1
0
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.16 проверка работы программы**

Произошло смещение выводимых чисел на -1, но теперь количество итераций совпадает введенному N

**Обработка аргументов командной строки**

Создадим файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга (рисунок 0.17)

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ touch lab8-2.asm
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ gedit lab8-2.asm
```

Открыть | lab8-2.asm ~/work/arch-pc/lab08 | Сохранить

```
%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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

**Рисунок 0.17 запись программы**

Создадим исполняемый файл и запустим его, указав аргументы (рисунок 0.18-0.19)

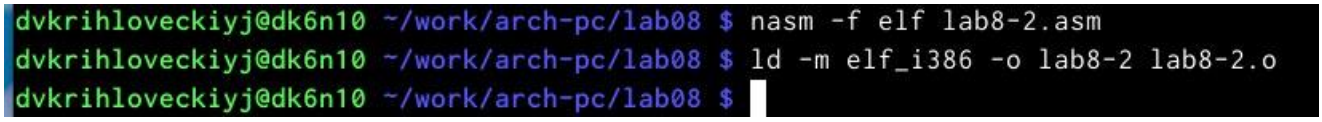


**Рисунок 0.18 создание исполняемого файла**

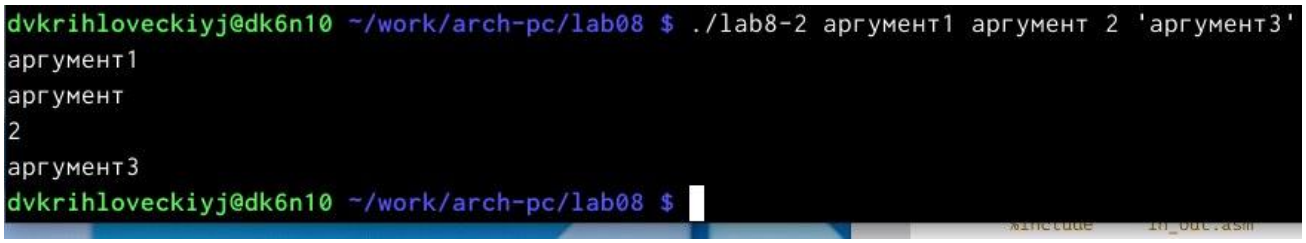


**Рисунок 0.19 проверка работы программы**

Программа обработала то же количество аргументов, что и было введено.

Создадим файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга (рисунок 0.20-0.21)


**Рисунок 0.20 создание файла**

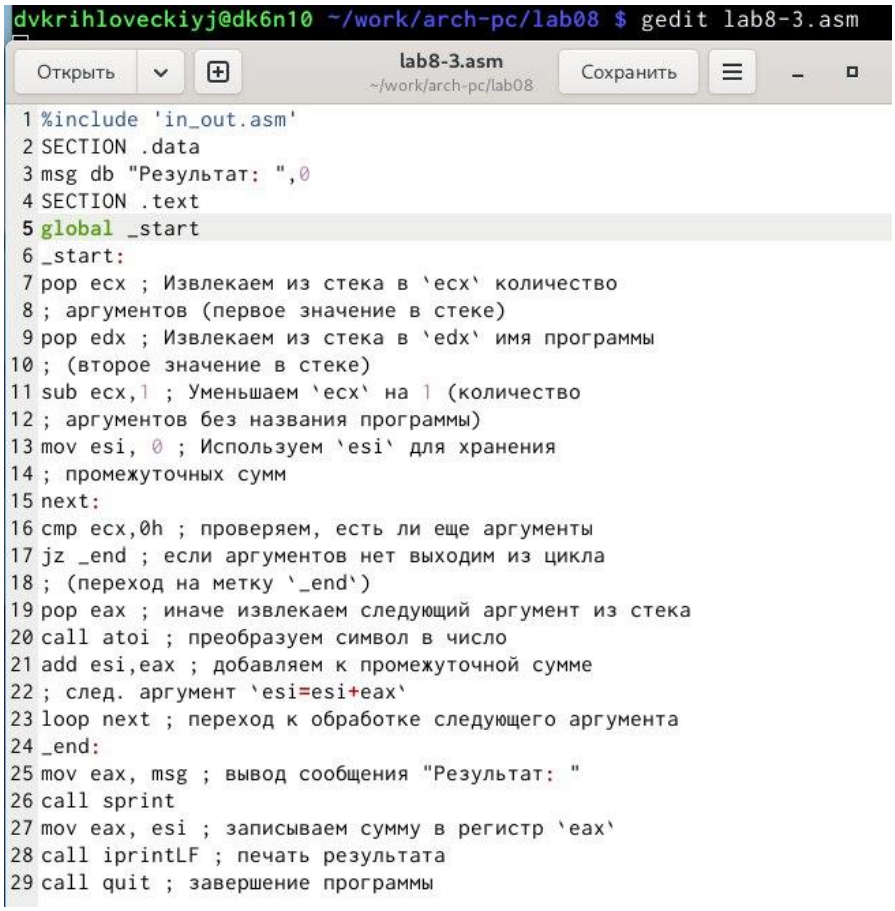


**Рисунок 0.21 запись программы**

Создадим исполняемый файл и запустим его, указав аргументы (рисунок 0.22-0.25)


**Рисунок 0.22 создание исполняемого файла**

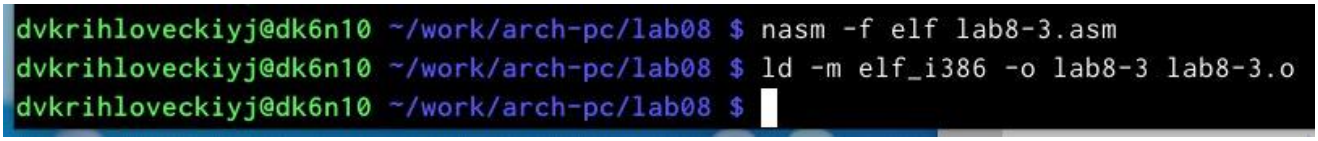


**Рисунок 0.23 создание исполняемого файла**


**Рисунок 0.24 запуск исполняемого кода**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ./lab8-3 11 15 5 10 1
Результат: 42
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $
```

**Рисунок 0.25 проверка работы программы**

Изменим текст программы из листинга для вычисления произведения аргументов командной строки (рисунок 0.26-0.29)

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ gedit lab8-3.asm
```

Открыть | lab8-3.asm ~/work/arch-pc/lab08 | Сохранить

```
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 pop edx ; Извлекаем из стека в `edx` имя программы
9 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
10 mov esi, 1 ; Используем `esi` для хранения
11
12 next:
13 cmp ecx,0h ; проверяем, есть ли еще аргументы
14 jz _end ; если аргументов нет выходим из цикла
15 pop eax ; иначе извлекаем следующий аргумент из стека
16 call atoi ; преобразуем символ в число
17 mul esi
18 mov esi,eax
19
20 loop next
21 _end:
22 mov eax, msg
23 call sprint
24 mov eax, esi
25 call iprintLF
26 call quit
```

**Рисунок 0.26 изменение программы**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
```

**Рисунок 0.27 создание исполняемого файла**

```
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
dvkrihloveckiyj@dk6n10 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
```

**Рисунок 0.28 создание исполняемого файла**

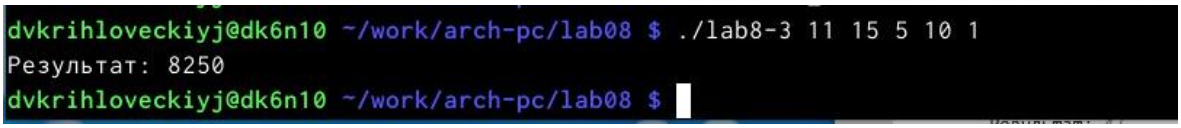


**Рисунок 0.29 проверка работы программы**

**Задание для самостоятельной работы**

Напишем программу, которая находит сумму значений функции □(□) для □ =□1,□2,...,□□, т.е. программа должна выводить значение □(□1)+□(□2)+...+□(□□). Выражение для f(x)= 7(x+1). (рисунок 0.30)

Открыть | lab8-4.asm ~/work/arch-pc/lab08 | Сохранить

```
%include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 7(x+1)", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_func
    call sprintLF

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

next:
    cmp ecx, 0h
    jz _end

    pop eax
    call atoi

    add eax, 1
    mov ebx, 7
    mul ebx

    add esi, eax
    loop next

_end:
    mov eax, msg_result
    call sprint
    mov eax, esi
    call iprintLF
    call quit
```

**Рисунок 0.30 запись кода**

Создадим исполняемый файл и проверьте его работу на нескольких наборах □ = □1,□2,...,□□ (рисунок 0.31-0.34)


**Рисунок 0.31 создание исполняемого файла**

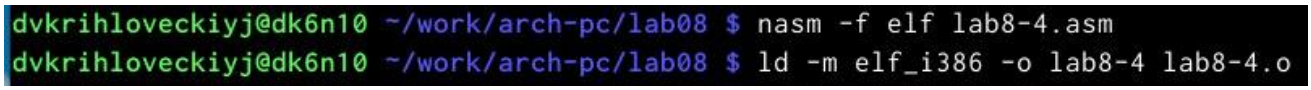


**Рисунок 0.32 создание исполняемого файла**


**Рисунок 0.33 запуск исполняемого файла**

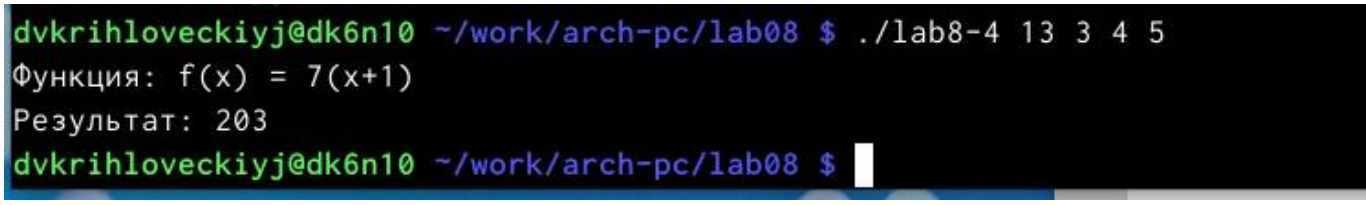


**Рисунок 0.34 проверка работы программы**

## 5 Выводы

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

**Список литературы**


1. GDB:TheGNUProjectDebugger.—URL:https://www.gnu.org/software/gdb/.

2. GNUBashManual.—2016.—URL:https://www.gnu.org/software/bash/manual/.

3. Midnight CommanderDevelopment Center.—2021.—URL: https://midnight-commander. org/.

4. NASMAssemblyLanguageTutorials.—2021.—URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. —(In a Nutshell). —ISBN 0596009658.—URL: http://www.amazon.com/Learning bash-Shell-Programming-Nutshell/dp/0596009658.

6. RobbinsA. Bash Pocket Reference.—O'Reilly Media,2016.—156 c.—ISBN 978-1491941591.

7. TheNASMdocumentation.—2021.—URL:https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash.—Packt Publishing,2017.—502 c.—ISBN 9781784396879.

9. Колдаев В. Д.,Лупин С. А.Архитектура ЭВМ.—М. : Форум,2018.

10. Куляс О. Л.,Никитин К. А. Курс программирования наASSEMBLER.—М. : Солон-Пресс, 2017.

11. НовожиловО.П.Архитектура ЭВМисистем.—М.:Юрайт,2016.

12. Расширенныйассемблер:NASM.—2021.—URL:https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А.,Немнюгин С.,Стесик О. Операционная система UNIX.—2-е изд.—БХВ Петербург, 2010.—656 с.—ISBN 978-5-94157-538-1.

**14. СтоляровА.Программированиенаязыкеассемблера NASMдляОСUnix.—2-еизд.— М. : МАКСПресс,2011.—URL:http://www.stolyarov.info/books/asm_unix.**

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э.,Бос Х. Современные операционные системы.—4-е изд.—СПб.: Питер, 2015. —1120 с.—(Классика Computer Science)