

# **Отчет по лабораторной работе №8**

**дисциплина: Архитектура компьютера**

Михайлова Регина Алексеевна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение заданий для самостоятельной работы	18
4	Выводы	20
	Список литературы	21

## Список иллюстраций

2.1	Создание файла . . . . .	6
2.2	Листинг 8.1. . . . .	8
2.3	Результат работы программы . . . . .	9
2.4	Измененная программа . . . . .	10
2.5	Запуск файла . . . . .	11
2.6	Добавление команд в текст программы . . . . .	12
2.7	Запуск файла . . . . .	13
2.8	Листинг 8.2. . . . .	14
2.9	Проверка работы программы . . . . .	15
2.10	Результат запуска . . . . .	16
2.11	Измененный текст программы . . . . .	17
2.12	Запуск файла . . . . .	17
3.1	Текст программы самостоятельного задания . . . . .	19
3.2	Проверка работы программы . . . . .	19

## Список таблиц

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

Создайте каталог для программ лабораторной работы № 8, перейдите в него и создайте файл lab8-1.asm (рис. 2.1):

```
mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.asm
```

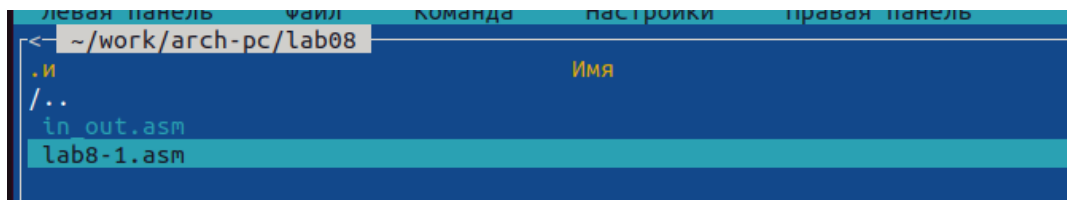


Рис. 2.1: Создание файла

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`. Внимательно изучите текст программы (Листинг 8.1).

Листинг 8.1. Программа вывода значений регистра `ecx`

```
;-----  
; Программа вывода значений регистра 'ecx'  
;-----  
  
%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

```

Введите в файл lab8-1.asm текст программы из листинга 8.1 (рис. 2.2). Создайте исполняемый файл и проверьте его работу (рис. 2.3).

```

#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: Листинг 8.1.



```
ramikhalova@ramikhalova:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
```

Рис. 2.3: Результат работы программы

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра `ecx` в цикле (рис. 2.4):

```
abel:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
```

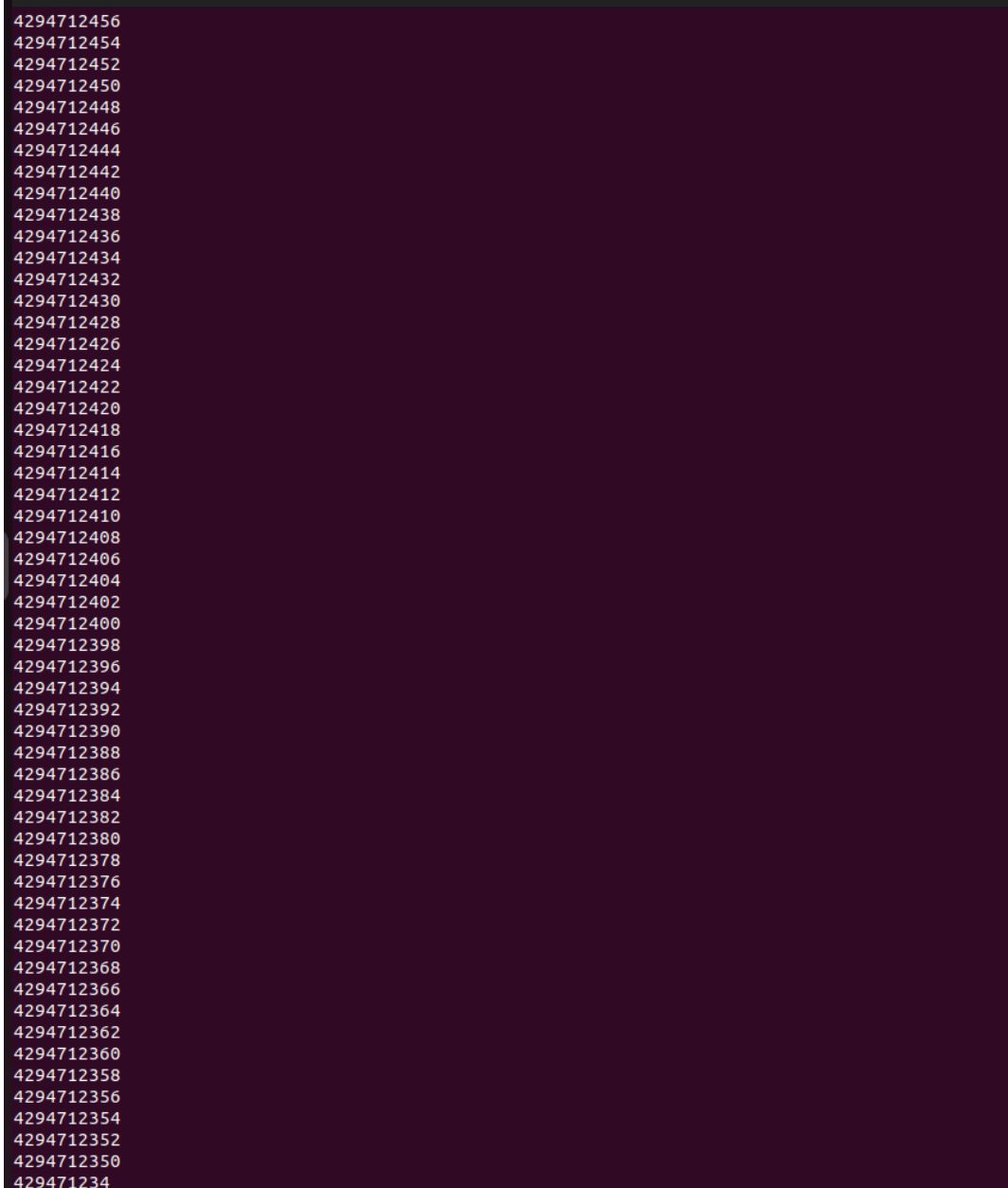
```

%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 ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit

```

Рис. 2.4: Измененная программа

Создайте исполняемый файл и проверьте его работу (рис. 2.5).



```
4294712456
4294712454
4294712452
4294712450
4294712448
4294712446
4294712444
4294712442
4294712440
4294712438
4294712436
4294712434
4294712432
4294712430
4294712428
4294712426
4294712424
4294712422
4294712420
4294712418
4294712416
4294712414
4294712412
4294712410
4294712408
4294712406
4294712404
4294712402
4294712400
4294712398
4294712396
4294712394
4294712392
4294712390
4294712388
4294712386
4294712384
4294712382
4294712380
4294712378
4294712376
4294712374
4294712372
4294712370
4294712368
4294712366
4294712364
4294712362
4294712360
4294712358
4294712356
4294712354
4294712352
4294712350
429471234
```

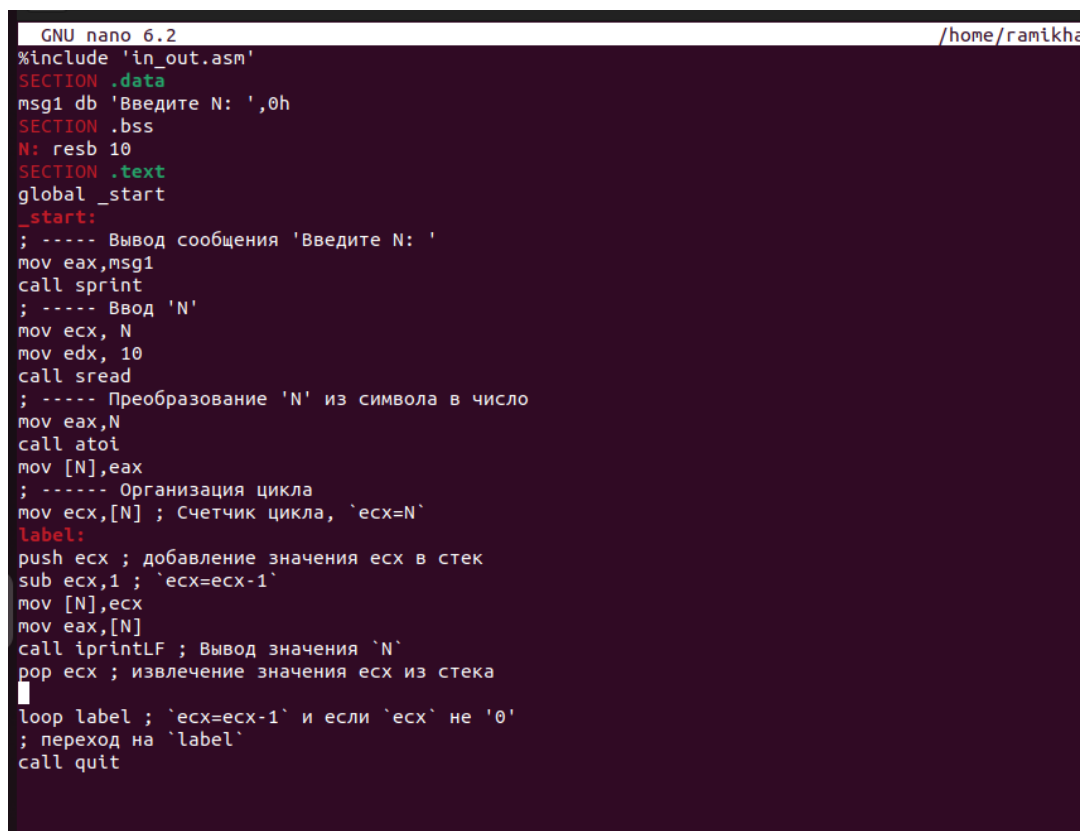
Рис. 2.5: Запуск файла

Мы можем заметить, что программа выводит большое количество чисел, число проходов гораздо больше заявленного N. Регистр принимает большие четные значения, близкие к 5000000000.

Для использования регистра ехх в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы

добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 2.6):

```
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
```



```
GNU nano 6.2 /home/ramikha
#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 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx ; извлечение значения ecx из стека
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 2.6: Добавление команд в текст программы

Создайте исполняемый файл и проверьте его работу (рис. 2.7).

```

ramikhalova@ramikhailova:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
ramikhalova@ramikhailova:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
ramikhalova@ramikhailova:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0

```

Рис. 2.7: Запуск файла

Теперь число проходов цикла соответствует значению ☒.

Для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучите текст программы (Листинг 8.2)

Листинг 8.2. Программа выводящая на экран аргументы командной строки

```

;-----
; Обработка аргументов командной строки
;-----

%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:

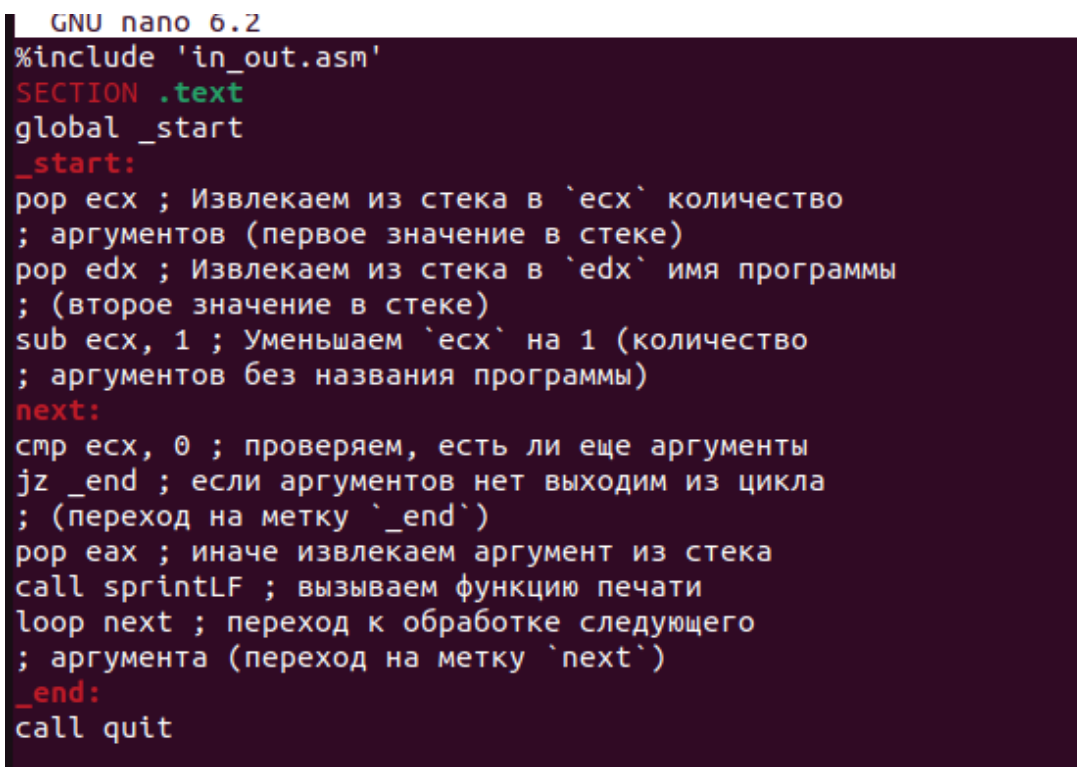
```

```

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

```

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст про- граммы из листинга 8.2 (рис. 2.8).



```

GNU nano 6.2
#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

```

Рис. 2.8: Листинг 8.2.

Создайте исполняемый файл и запустите его, указав аргументы (рис. 2.9):  
user@dk4n31:~\$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'

```

ramikhalova@ramikhailova:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 2.9: Проверка работы программы

Сколько аргументов было обработано программой? Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создайте файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.3.

Листинг 8.3. Программа вычисления суммы аргументов командной строки

```

%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

```

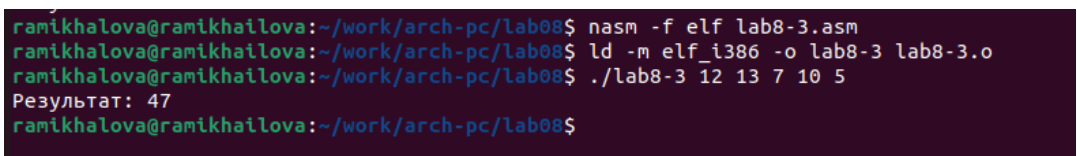
```

call atoi ; преобразуем символ в число
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

```

Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы (рис. 2.10):

```
user@dk4n31:~$ ./main 12 13 7 10 5 Результат: 47 user@dk4n31:~$
```



```

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

```

Рис. 2.10: Результат запуска

Измените текст программы из листинга 8.3 (рис. 2.11) для вычисления произведения аргументов командной строки (рис. 2.12).



```

#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 ; 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
mul esi;
mov 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 iprintfLF ; печать результата
call quit ; завершение программ

```

Рис. 2.11: Измененный текст программы

```

ramikhalova@ramikhalova:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
ramikhalova@ramikhalova:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24
ramikhalova@ramikhalova:~/work/arch-pc/lab08$ █

```

Рис. 2.12: Запуск файла

### 3 Выполнение заданий для самостоятельной работы

1. Напишите программу (рис. 3.1), которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу (рис. 3.2) на нескольких наборах  $x = x_1, x_2, \dots, x_n$ .



## 4 Выводы

В ходе лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

## Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science)