

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров и операционные системы

Закиров Нурислам Дамирович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	13
4.3	Задание для самостоятельной работы	15
5	Выводы	18
6	Список литературы	19

Список иллюстраций

4.1	Создание файлов для лабораторной работы	9
4.2	Ввод текста из листинга 8.1	10
4.3	Запуск исполняемого файла	10
4.4	Изменение текста программы	11
4.5	Запуск обновленной программы	11
4.6	Изменение текста программы	12
4.7	Запуск исполняемого файла	12
4.8	Ввод текста программы из листинга 8.2	13
4.9	Запуск исполняемого файла	13
4.10	Ввод текста программы из листинга 8.3	14
4.11	Запуск исполняемого файла	14
4.12	Изменение текста программы	15
4.13	Запуск исполняемого файла	15
4.14	Текст программы	16
4.15	Запуск исполняемого файла и проверка его работы	16

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация циклов 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. 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.

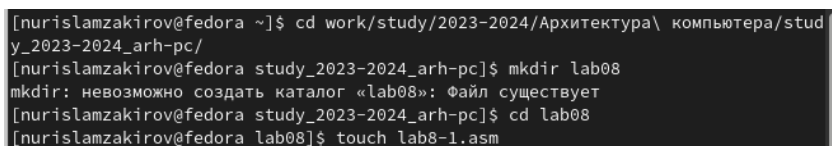
Для организации циклов существуют специальные инструкции. Для всех ин-

струкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loor. Она позволяет организовать безусловный цикл.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

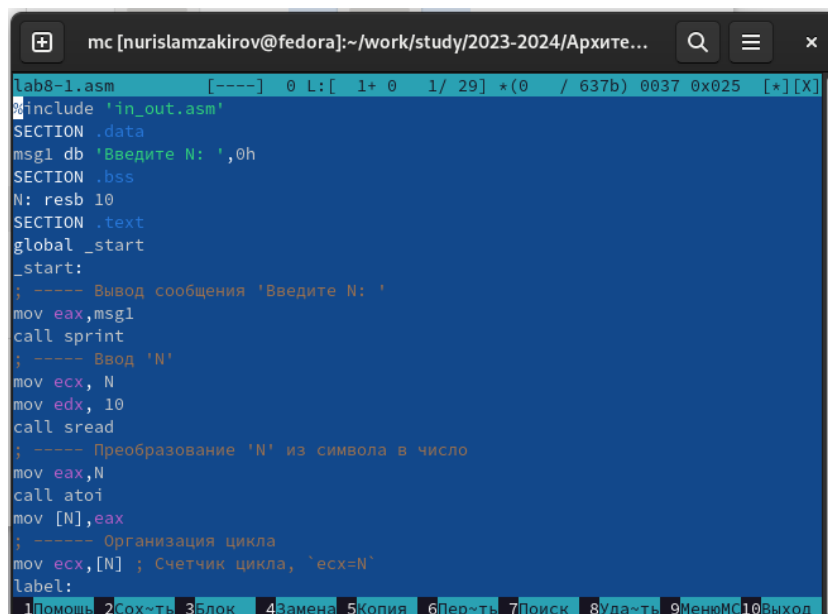
Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. 4.1).



```
[nurislamzakirov@fedora ~]$ cd work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh-pc/
[nurislamzakirov@fedora study_2023-2024_arh-pc]$ mkdir lab08
mkdir: невозможно создать каталог «lab08»: Файл существует
[nurislamzakirov@fedora study_2023-2024_arh-pc]$ cd lab08
[nurislamzakirov@fedora lab08]$ touch lab8-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

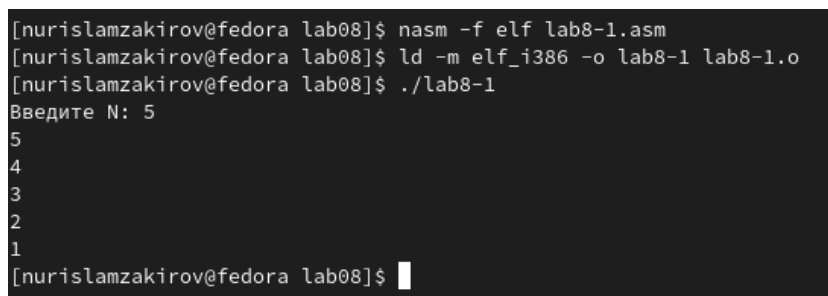
Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.2).



```
lab8-1.asm  [----]  0  L: [ 1+ 0  1/ 29]  *(0  / 637b) 0037 0x025  [*][X]
%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:
1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюМС10Выход
```

Рис. 4.2: Ввод текста из листинга 8.1

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

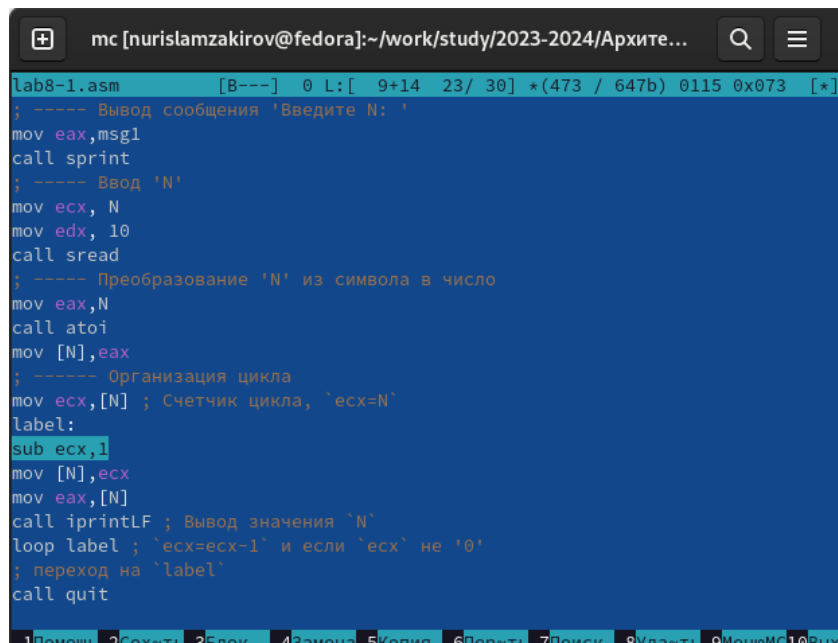


```
[nurislamzakirov@fedora lab08]$ nasm -f elf lab8-1.asm
[nurislamzakirov@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[nurislamzakirov@fedora lab08]$ ./lab8-1
Введите N: 5
5
4
3
2
1
[nurislamzakirov@fedora lab08]$
```

Рис. 4.3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

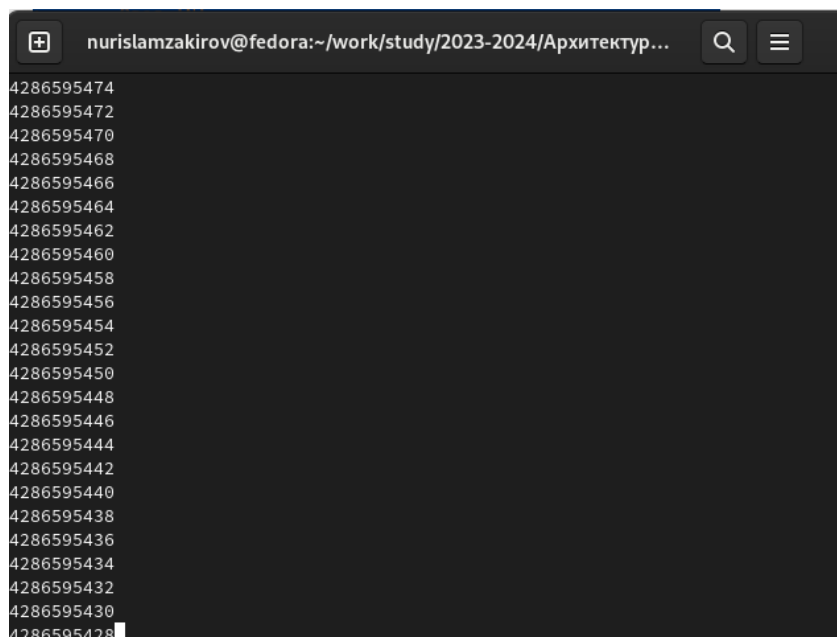
Изменяю текст программы, добавив изменение значения регистра ecx в цикле.
(рис. 4.4).



```
lab8-1.asm [B---] 0 L: [ 9+14 23/ 30] *(473 / 647b) 0115 0x073 [*]  
; ----- 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  
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. 4.4: Изменение текста программы

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



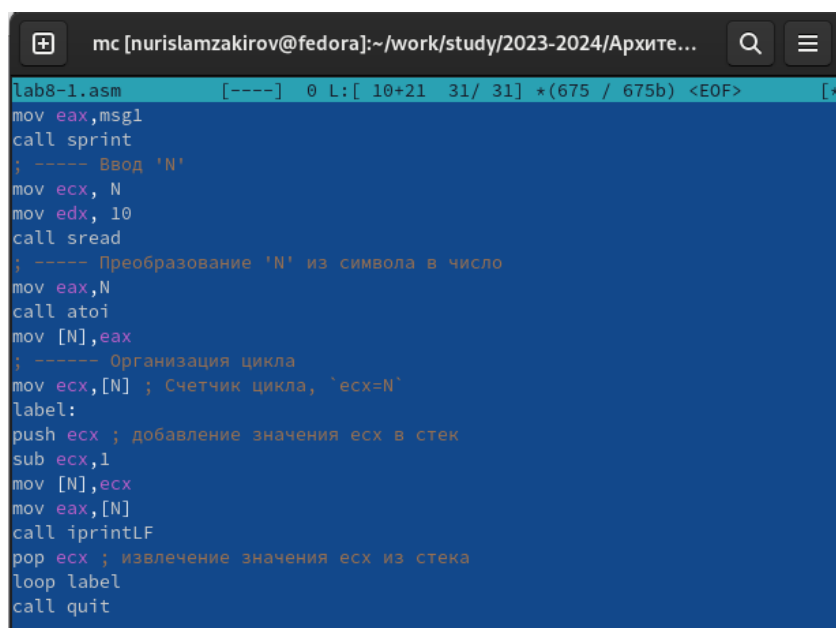
```
nurislamzakirov@fedora:~/work/study/2023-2024/Архитектур...  
4286595474  
4286595472  
4286595470  
4286595468  
4286595466  
4286595464  
4286595462  
4286595460  
4286595458  
4286595456  
4286595454  
4286595452  
4286595450  
4286595448  
4286595446  
4286595444  
4286595442  
4286595440  
4286595438  
4286595436  
4286595434  
4286595432  
4286595430  
4286595428
```

Рис. 4.5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры

туры значению.

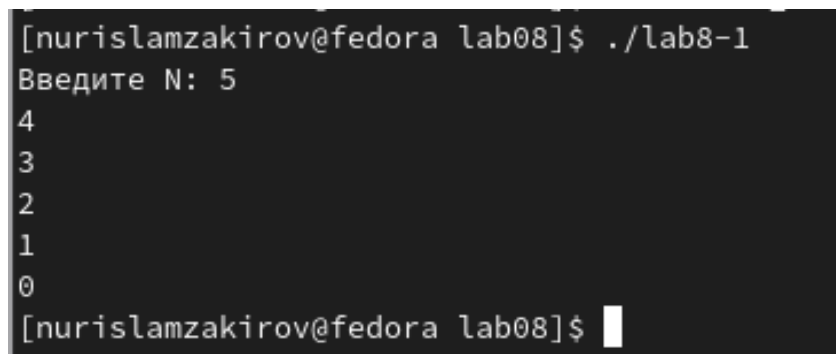
Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. 4.6).



```
lab8-1.asm [----] 0 L: [ 10+21 31/ 31] *(675 / 675b) <EOF> [*]
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
```

Рис. 4.6: Изменение текста программы

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



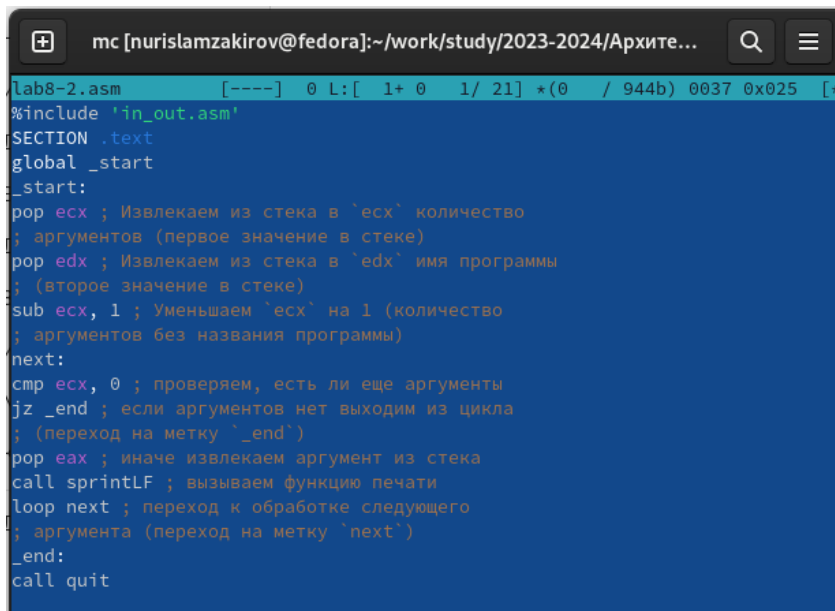
```
[nurislamzakirov@fedora lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[nurislamzakirov@fedora lab08]$
```

Рис. 4.7: Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

4.2 Обработка аргументов командной строки

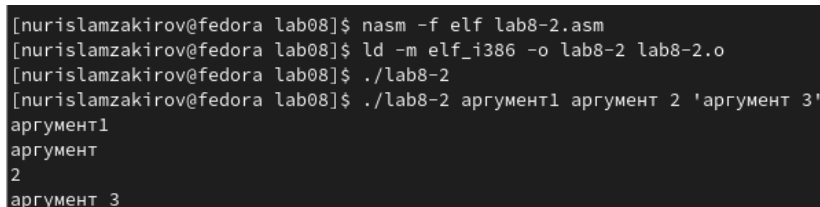
Создаю файл lab8-2.asm и ввожу в него текст программы из листинга 8.2. (рис. 4.8).



```
lab8-2.asm [-----] 0 L: [ 1+ 0 1/ 21] *(0 / 944b) 0037 0x025 [*]
#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 printf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 4.8: Ввод текста программы из листинга 8.2

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

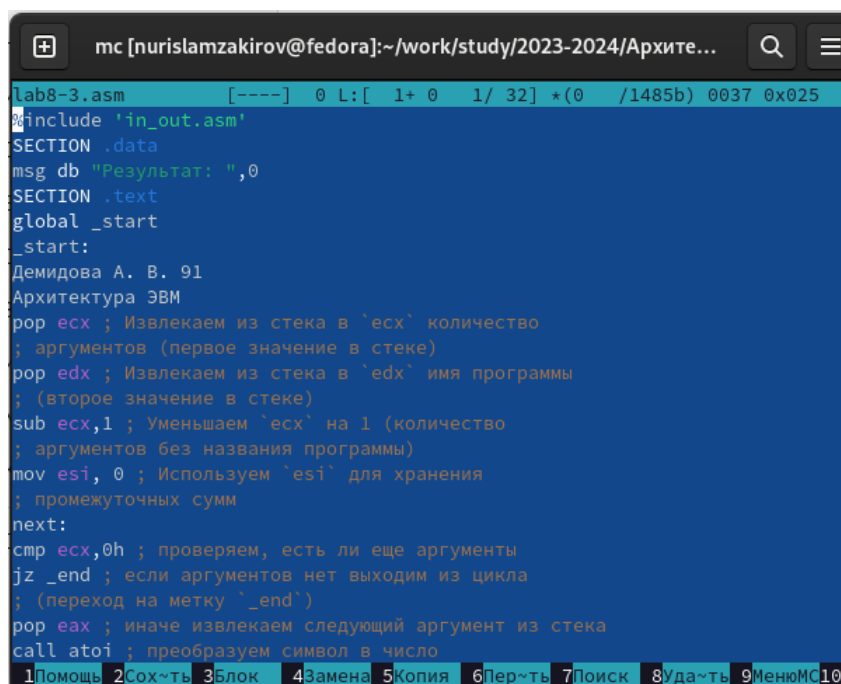


```
[nurislamzakirov@fedora lab08]$ nasm -f elf lab8-2.asm
[nurislamzakirov@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[nurislamzakirov@fedora lab08]$ ./lab8-2
[nurislamzakirov@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

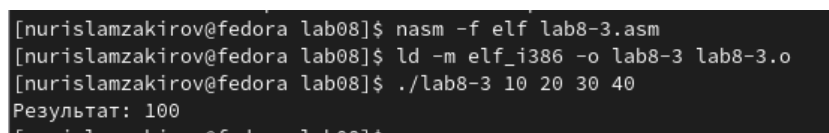
Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm и ввожу в него текст программы из листинга 8.3. (рис. 4.10).



```
lab8-3.asm [----] 0 L: [ 1+ 0 1/ 32] *(0 /1485b) 0037 0x025
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
Демидова А. В. 91
Архитектура ЭВМ
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
1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС10
```

Рис. 4.10: Ввод текста программы из листинга 8.3

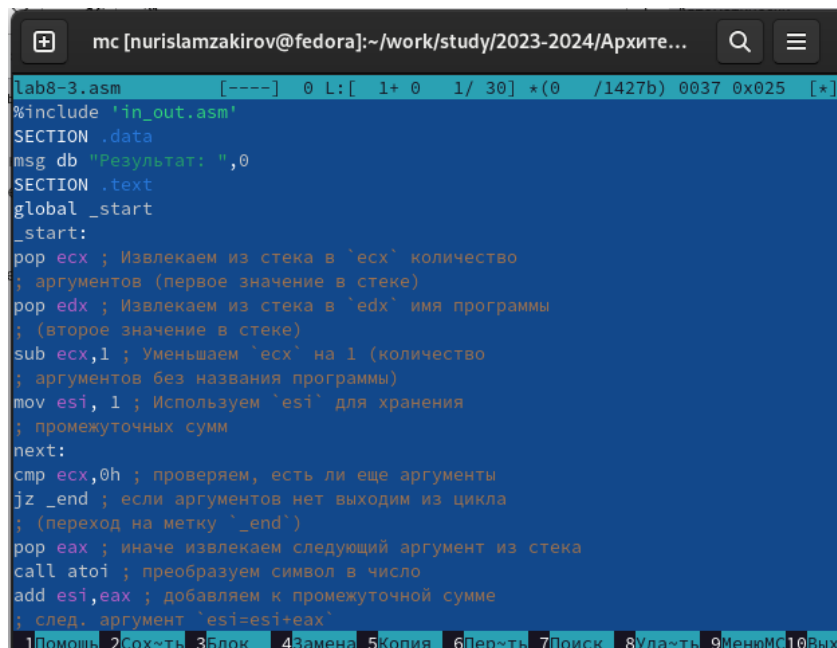
Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.11).



```
[nurislamzakirov@fedora lab08]$ nasm -f elf lab8-3.asm
[nurislamzakirov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[nurislamzakirov@fedora lab08]$ ./lab8-3 10 20 30 40
Результат: 100
[nurislamzakirov@fedora lab08]$
```

Рис. 4.11: Запуск исполняемого файла

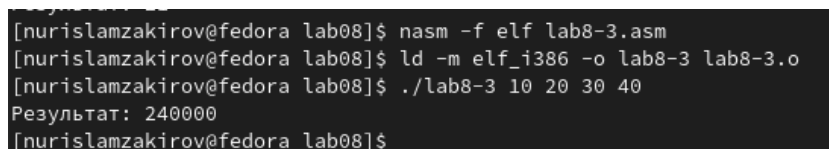
Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 4.12).



```
lab8-3.asm [----] 0 L: [ 1+ 0 1/ 30] *(0 /1427b) 0037 0x025 [*]
%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, 1 ; 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`
1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюMC 10Вых
```

Рис. 4.12: Изменение текста программы

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

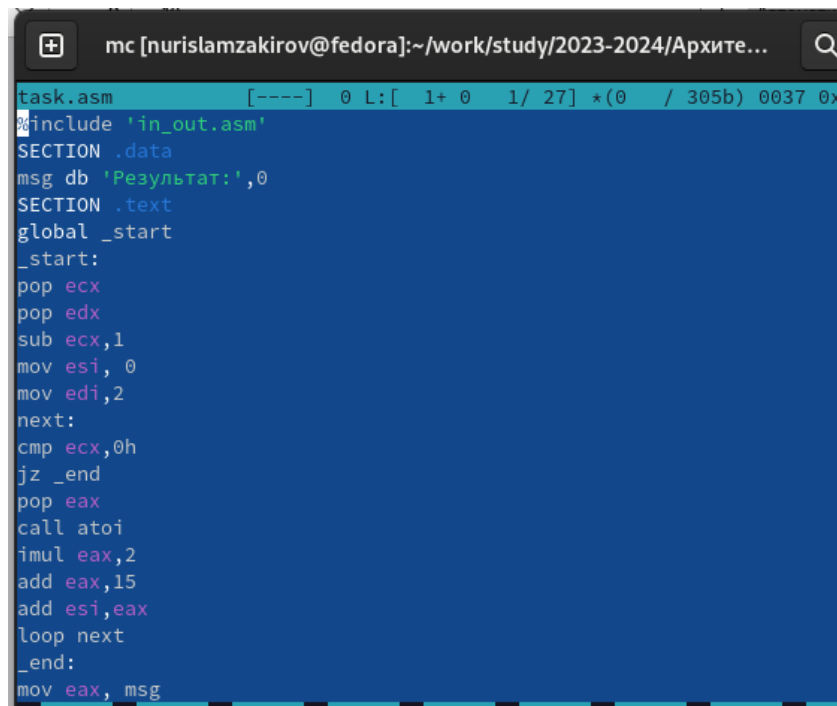


```
[nurislamzakirov@fedora lab08]$ nasm -f elf lab8-3.asm
[nurislamzakirov@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[nurislamzakirov@fedora lab08]$ ./lab8-3 10 20 30 40
Результат: 240000
[nurislamzakirov@fedora lab08]$
```

Рис. 4.13: Запуск исполняемого файла

4.3 Задание для самостоятельной работы

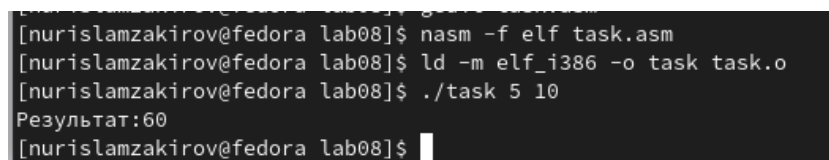
Пишу текст программы, которая находит сумму значений функции $f(x) = 2 \cdot x + 15$ в соответствии с моим номером варианта (1) для $x = x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы. (рис. 4.14).



```
task.asm [----] 0 L: [ 1+ 0 1/ 27] *(0 / 305b) 0037 0x
#include 'in_out.asm'
SECTION .data
msg db 'Результат:',0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,2
next:
cmp ecx,0h
jz _end
pop eax
call atoi
imul eax,2
add eax,15
add esi,eax
loop next
_end:
mov eax, msg
```

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).



```
[nurislamzakirov@fedora lab08]$ nasm -f elf task.asm
[nurislamzakirov@fedora lab08]$ ld -m elf_i386 -o task task.o
[nurislamzakirov@fedora lab08]$ ./task 5 10
Результат:60
[nurislamzakirov@fedora lab08]$
```

Рис. 4.15: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Текст программы:

```
%include 'in_out.asm'
SECTION .data
msg db 'Результат:',0
```



```

SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,2
next:
cmp ecx,0h
jz _end
pop eax
call atoi
imul eax,2
add eax,15
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

5 Выводы

Я приобрел навыки написания программ с использованием циклов и обработки аргументов командной строки благодаря этой лабораторной работе. Эти навыки будут полезны для моих последующих лабораторных работ.

6 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/Learningbash-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, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. 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.

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).