

# **Выпоолнение лабораторной работы №8**

**Дисциплина: Архитектура компьютера**

Ефремова Полина Александровна

# 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>
3.0.1	Что такое стек в ассемблере? . . . . .	7
3.1	Добавление элемента в стек. . . . .	8
3.2	Извлечение элемента из стека. . . . .	8
3.3	Инструкции организации циклов . . . . .	8
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
4.1	Реализация циклов в NASM. . . . .	10
4.2	Обработка аргументов командной строки . . . . .	16
<b>5</b>	<b>Задание для самостоятельной работы.</b>	<b>25</b>
<b>6</b>	<b>Выводы</b>	<b>29</b>
	<b>Список литературы</b>	<b>30</b>

## Список иллюстраций

4.1	Новый каталог для лабораторной работы №8 . . . . .	10
4.2	Ввод программы . . . . .	11
4.3	Компиляция и вывод программы . . . . .	12
4.4	Изменение значения esx в программе . . . . .	13
4.5	Компиляция и вывод программы . . . . .	14
4.6	Изменение программы . . . . .	15
4.7	Компиляция и вывод программы . . . . .	15
4.8	Создание файла . . . . .	16
4.9	Ввод программы . . . . .	17
4.10	Компиляция и вывод программы . . . . .	18
4.11	Создание файла . . . . .	19
4.12	Ввод программы . . . . .	21
4.13	Компиляция и вывод программы . . . . .	21
4.14	Изменение на умножение . . . . .	22
4.15	Компиляция и вывод программы . . . . .	24
5.1	Создание файла . . . . .	25
5.2	Ввод программы . . . . .	26
5.3	Запуск программы . . . . .	28
5.4	Запуск программы . . . . .	28

## **Список таблиц**

# 1 Цель работы

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

## **2 Задание**

1. Изучение теории на заданную тему.
2. Практика с использованием предложенных листингов.
3. Практика с самостоятельным написанием программы.

## 3 Теоретическое введение

### 3.0.1 Что такое стек в ассемблере?

Работа процедур тесно связана со стеком. Стек называется область программы для временного хранения данных. Стек в ассемблере работает по правилу «Первым зашёл — последним вышел, последним зашёл — первым вышел». В любой период времени в стеке доступен только первый элемент, то есть элемент, загруженный в стек последним. Выгрузка из стека верхнего элемента делает доступным следующий элемент. Это напоминает ящик, в который поочерёдно ложатся книги. Чтобы получить доступ к книге, которую положили первой, необходимо достать поочерёдно все книги, лежащие сверху. Элементы стека располагаются в специально выделенной под стек области памяти, начиная со дна стека по последовательно уменьшающимся адресам. Адрес верхнего доступного элемента хранится в регистре-указателе стека SP. Стек может входить в какой-либо сегмент или быть отдельным сегментом. Сегментный адрес стека помещается в сегментный регистр SS. Пара регистров SS:SP образует адрес доступной ячейки стека.

Работа со стеком осуществляется с помощью команд PUSH и POP.

Примеры для PUSH и POP:

**push cs**; значение cs можно затолкать в стек

**push bx**

**push ax**

`pop ax`

`pop bx`

`pop ax`; а вот изменить значение `cs` нельзя (`pop cs` - ошибка)!

### 3.1 Добавление элемента в стек.

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.

Существует ещё две команды для добавления значений в стек. Это команда `pusha`, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: `ax`, `cx`, `dx`, `bx`, `sp`, `bp`, `si`, `di`. А также команда `pushf`, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

### 3.2 Извлечение элемента из стека.

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.

### 3.3 Инструкции организации циклов

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее



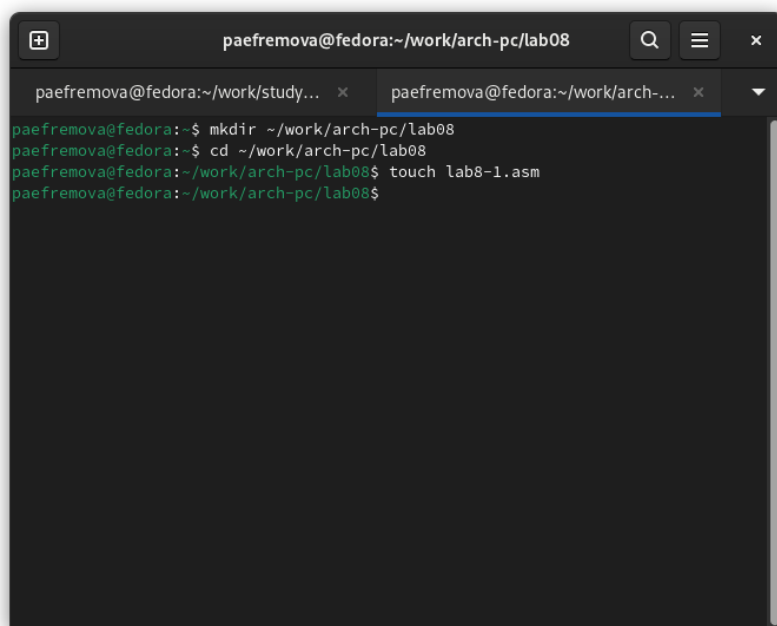
простой является инструкция loop. Она позволяет организовать безусловный цикл

Инструкция loop выполняется в два этапа. Сначала из регистра ехх вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM.

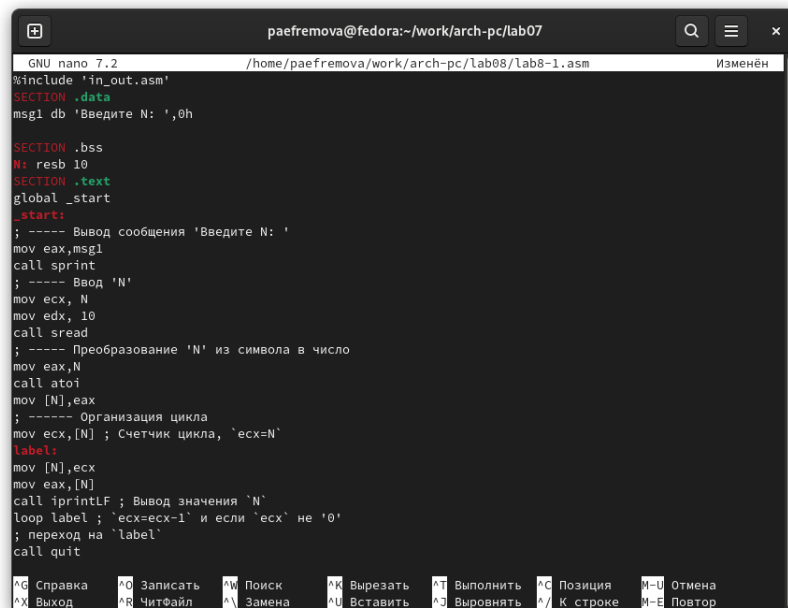
1. Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. 4.1).



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

Рис. 4.1: Новый каталог для лабораторной работы №8

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



```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-1.asm
%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 iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция ^M-U Отмена
^X Выход ^R ЧитФайл ^A Замена ^U Вставить ^D Выровнять ^V К строке ^M-E Повтор
```

Рис. 4.2: Ввод программы

```
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

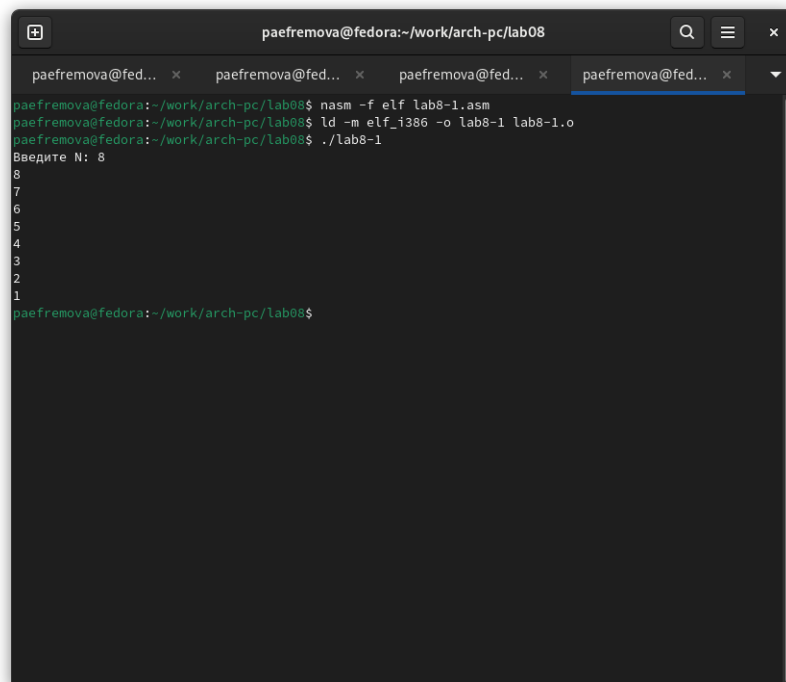
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
```

```

; ----- 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

```

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



```

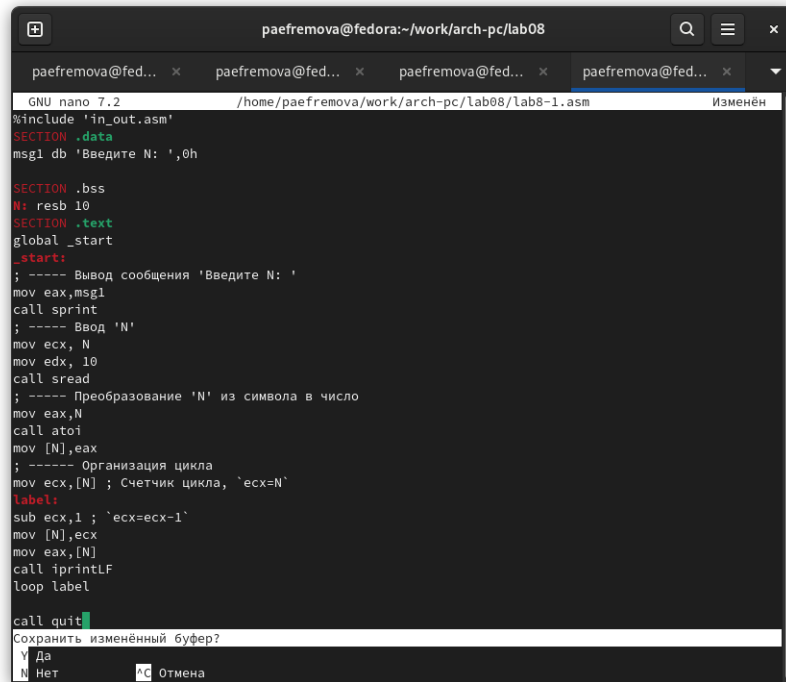
paefremova@fedora:~/work/arch-pc/lab08
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 8
8
7
6
5
4
3
2
1
paefremova@fedora:~/work/arch-pc/lab08$

```

Рис. 4.3: Компиляция и вывод программы

## Листинг 8.1. Программа вывода значений регистра esx

4. Изменяю текст программы, добавив изменение значение регистра esx в цикле. (рис. 4.4) и (рис. 4.5).

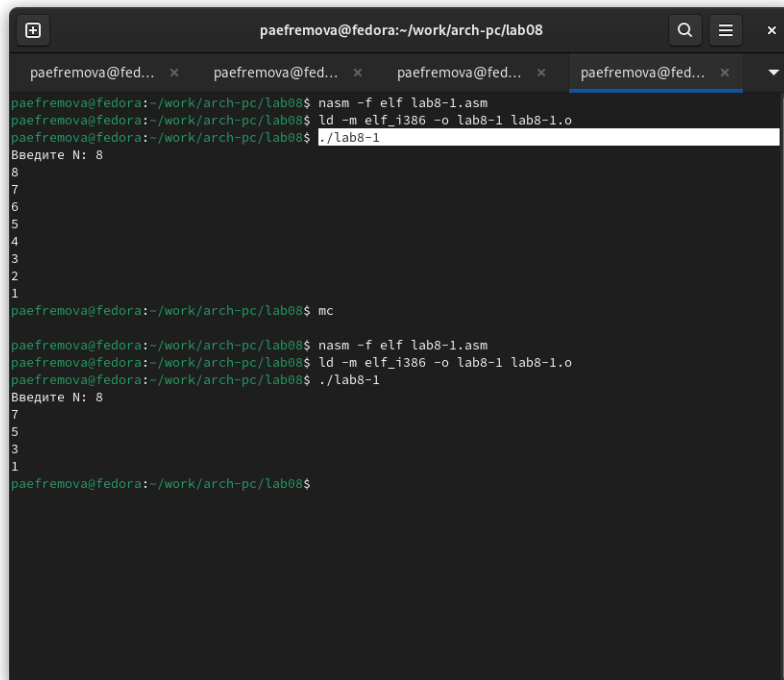


```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h

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
loop label

call quit
Сохранить измененный буфер?
Y Да
N Нет  Ctrl+C Отмена
```

Рис. 4.4: Изменение значения esx в программе



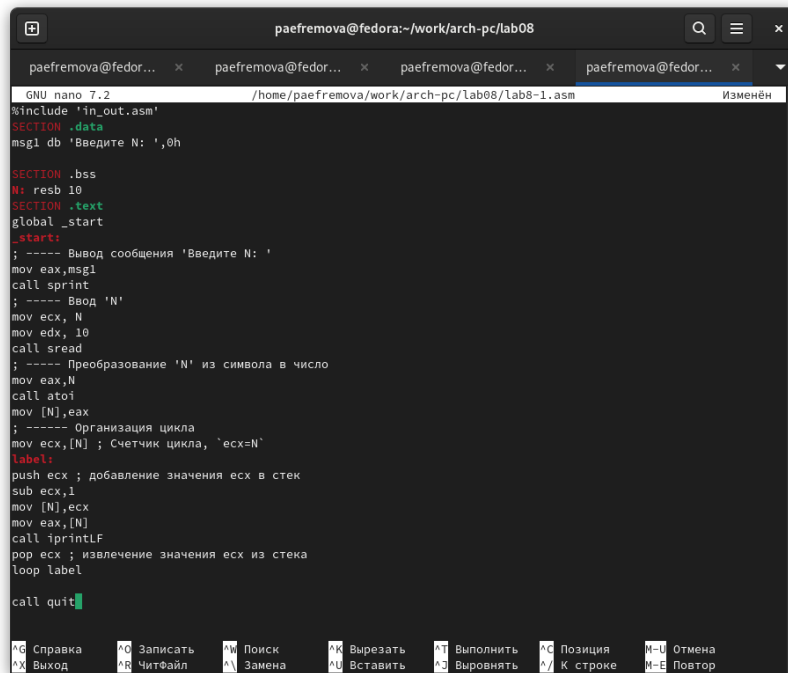
```
paefremova@fedora:~/work/arch-pc/lab08
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 8
8
7
6
5
4
3
2
1
paefremova@fedora:~/work/arch-pc/lab08$ mc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

Рис. 4.5: Компиляция и вывод программы

Внутри цикла, перед выводом, инструкция `sub ecx, 1` уменьшает значение `ecx` на 1. Затем уменьшенное значение из `ecx` записывается обратно в переменную `N` (`mov [N], ecx`). Значение, которое находится в `ecx` (уменьшенное на 1) выводится на экран. Инструкция `loop label` уменьшает `ecx` еще на 1 и переходит к метке `label`, если `ecx` не равно 0.

Поэтому, число проходов цикла не соответствует значению `N`, введенному с клавиатуры. Цикл выполняется `N` раз, но значения, которые выводятся, это от `N-1` до 1

5. Вношу изменения в текст программы добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop`. (рис. 4.6) и (рис. 4.7).

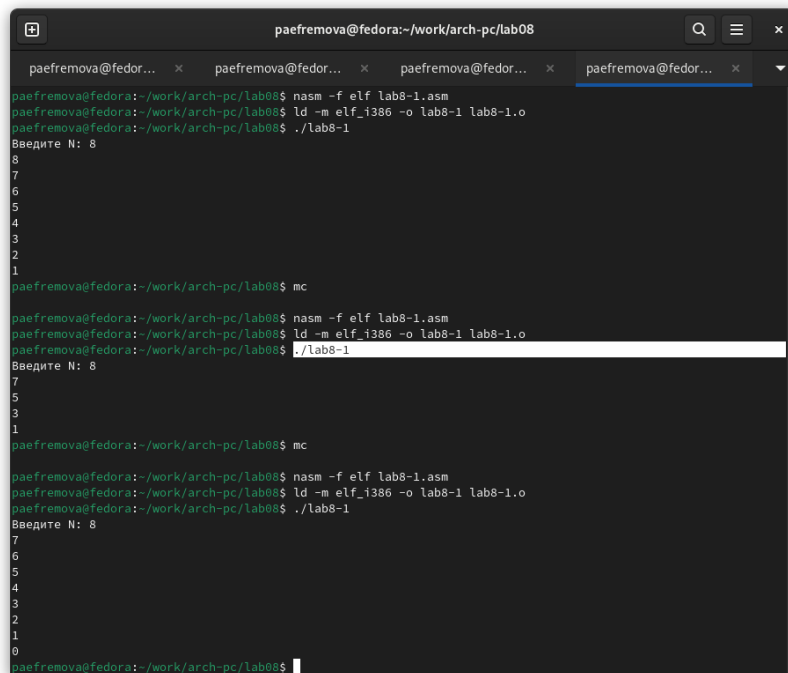


```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h

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 read
; ----- 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 iprintf
pop ecx ; извлечение значения ecx из стека
loop label

call quit
```

Рис. 4.6: Изменение программы



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

Рис. 4.7: Компиляция и вывод программы

Цикл выполняется N раз, но инструкция loop уменьшает esx на 1 перед проверкой на равенство нулю. Поэтому, если пользователь введет N, цикл будет выполняться N раз, но значения, которые выводятся, не будут от 1 до N. Вместо этого, будут выводиться числа от N-1 до 0, т.е N раз но со смещением на -1.

## 4.2 Обработка аргументов командной строки

1. Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. (рис. 4.8).

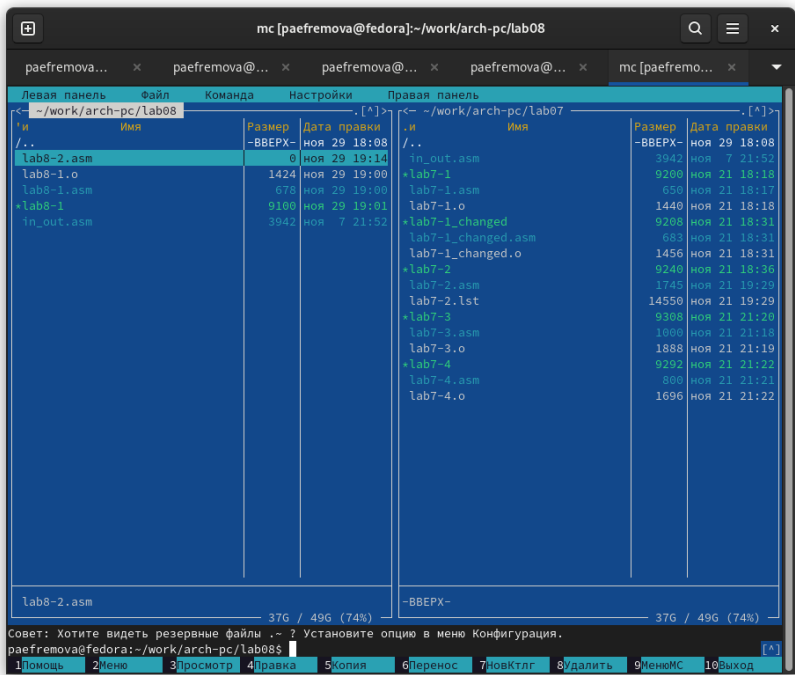


Рис. 4.8: Создание файла

2. Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.9).



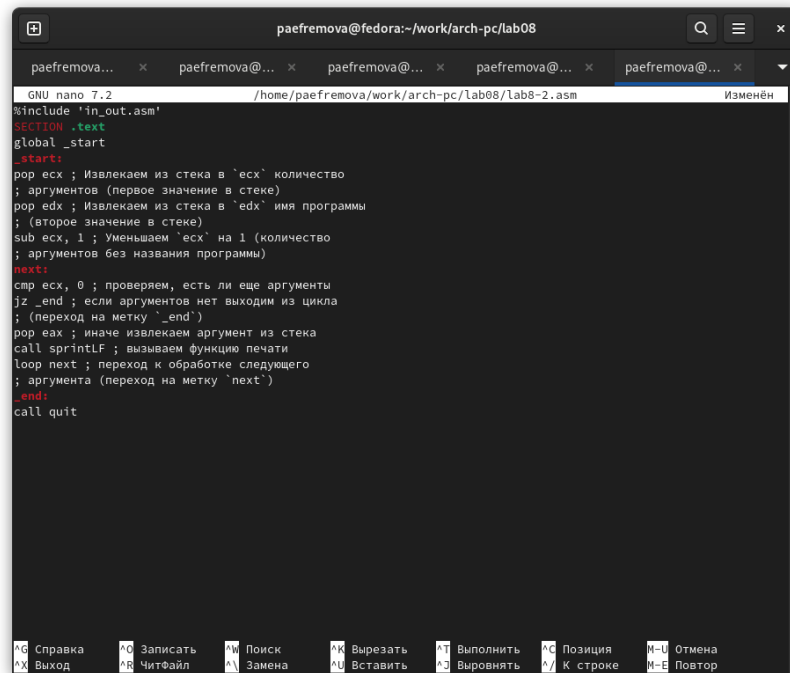


Рис. 4.9: Ввод программы

## Листинг 8.2. Программа выводящая на экран аргументы командной строки

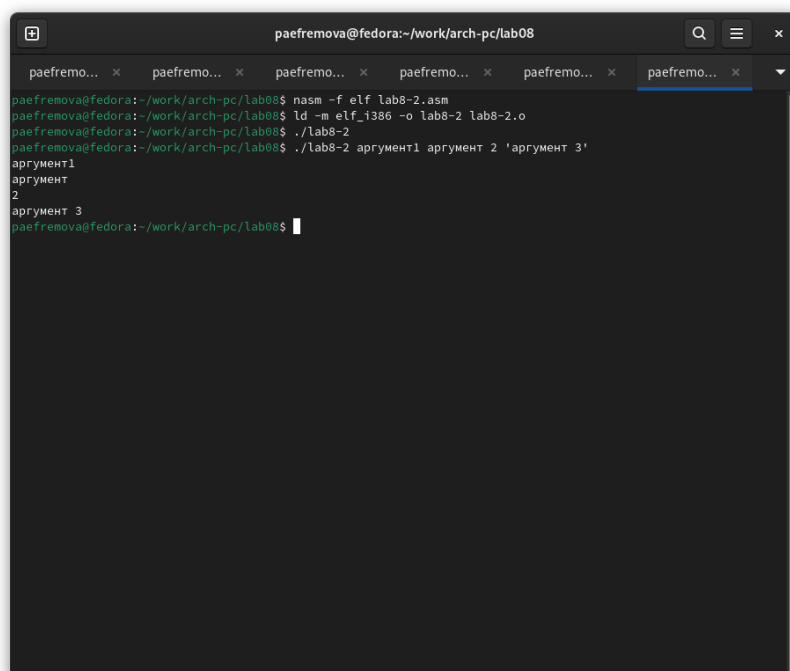
```

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

3. Создаю исполняемый файл и проверяю его работу. (рис. 4.10).



```
paefremova@fedora:~/work/arch-pc/lab08
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-2
аргумент1
аргумент
2
аргумент 3
paefremova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.10: Компиляция и вывод программы

Программой было обработано то же количество аргументов, что и было введено.

4. Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08. (рис. 4.11).

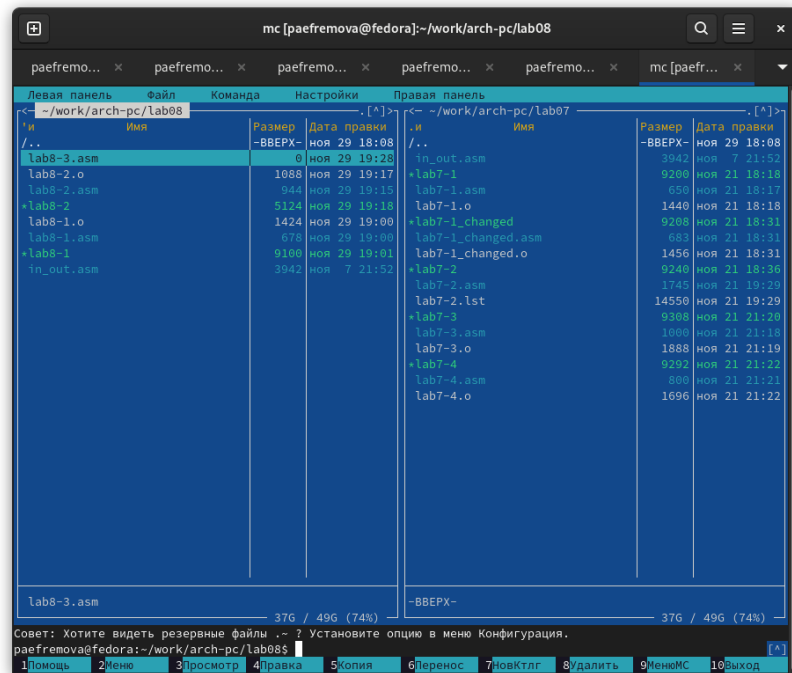


Рис. 4.11: Создание файла

- Ввожу в файл lab8-3.asm текст программы из листинга 8.1. (рис. 4.12).

### 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
```

```
pop edx
sub ecx,1

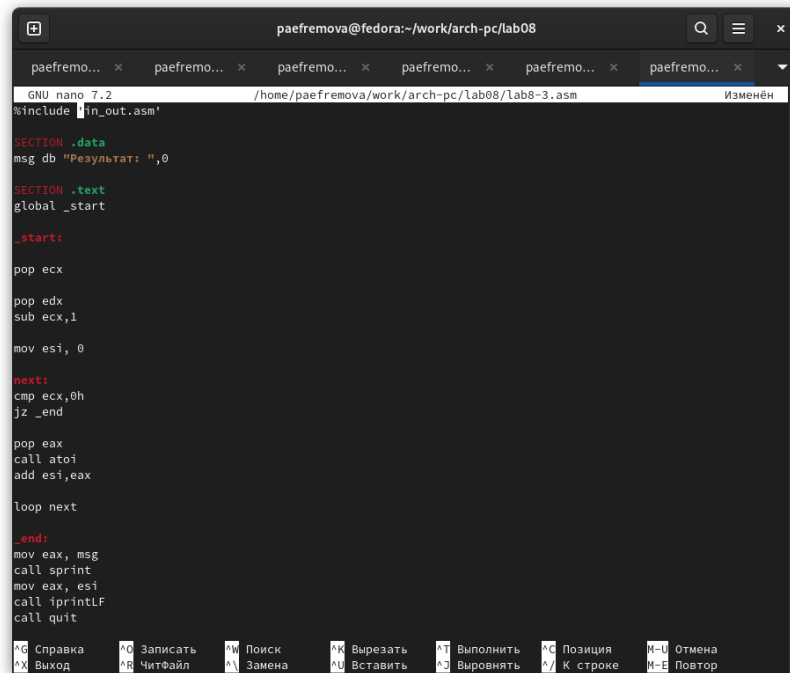
mov esi, 0

next:
cmp ecx,0h
jz _end

pop eax
call atoi
add esi,eax

loop next

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



```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-3.asm
%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

next:
    cmp ecx,0h
    jz _end

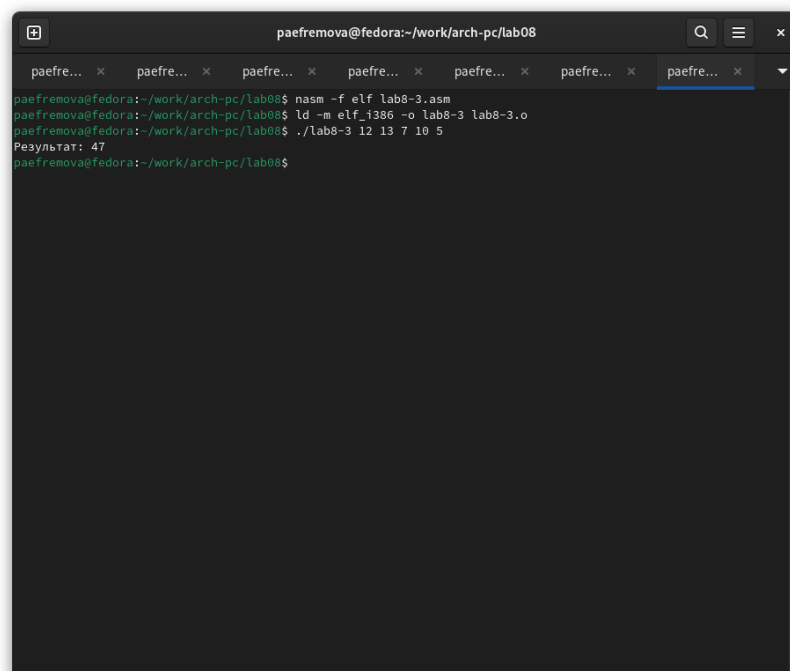
    pop eax
    call atoi
    add esi,eax

    loop next

_end:
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintf
    call quit
```

Рис. 4.12: Ввод программы

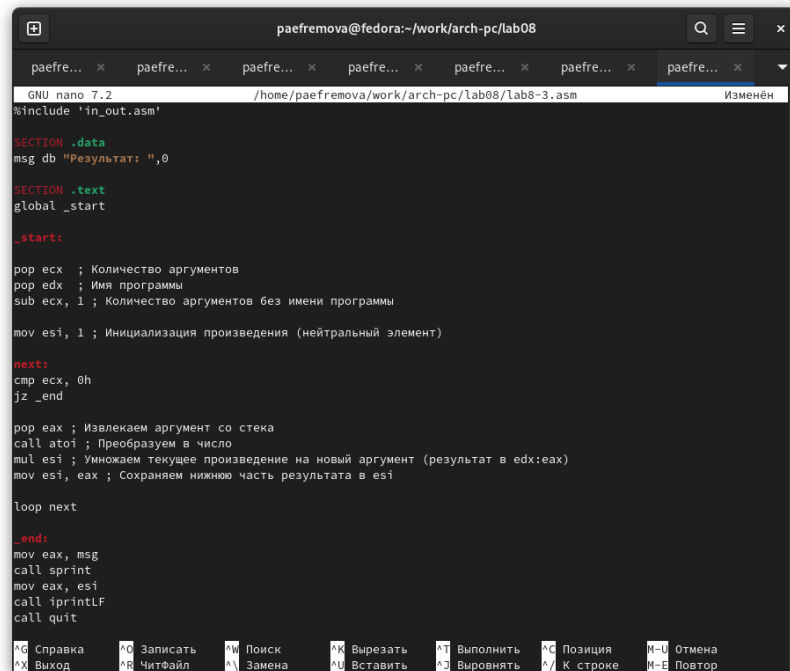
6. Создаю исполняемый файл и проверяю его работу. (рис. 4.13).



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

Рис. 4.13: Компиляция и вывод программы

7. Изменяю исполняемый файл, чтобы аргументы умножались, а не складывались (рис. 4.14).



```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-3.asm
#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start

_start:

pop ecx ; Количество аргументов
pop edx ; Имя программы
sub ecx, 1 ; Количество аргументов без имени программы

mov esi, 1 ; Инициализация произведения (нейтральный элемент)

next:
cmp ecx, 0h
jz _end

pop eax ; Извлекаем аргумент со стека
call atoi ; Преобразуем в число
mul esi ; Умножаем текущее произведение на новый аргумент (результат в edx:eax)
mov esi, eax ; Сохраняем нижнюю часть результата в esi

loop next

_end:
mov eax, msg
call sprintf
mov eax, esi
call iprintf
call quit
```

Рис. 4.14: Изменение на умножение

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

```
SECTION .data
```

```
msg db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx ; Количество аргументов
```

```
pop edx ; Имя программы
```

```

sub ecx, 1 ; Количество аргументов без имени программы

mov esi, 1 ; Инициализация произведения (нейтральный элемент)

next:
cmp ecx, 0h
jz _end

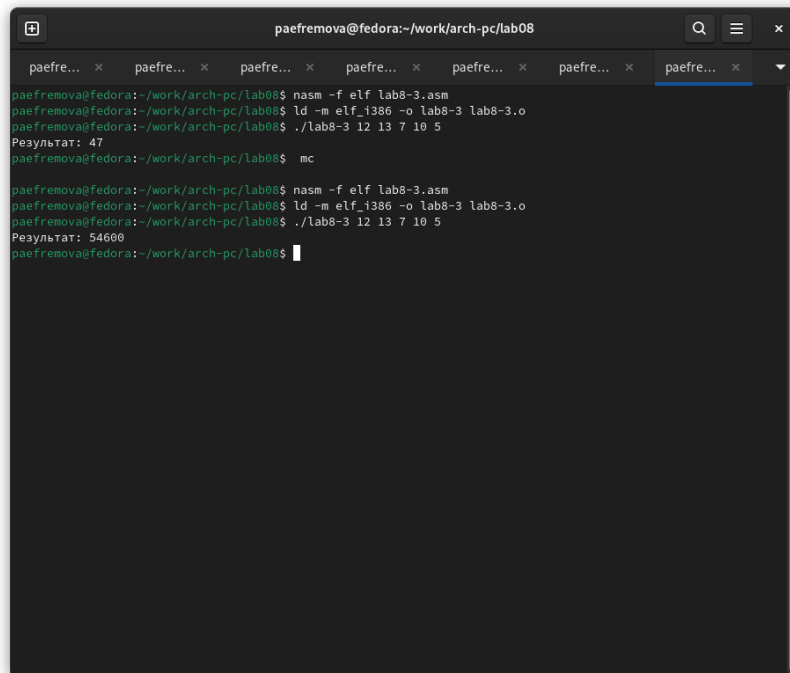
pop eax ; Извлекаем аргумент со стека
call atoi ; Преобразуем в число
mul esi ; Умножаем текущее произведение на новый аргумент (результат в edx:eax)
mov esi, eax ; Сохраняем нижнюю часть результата в esi

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

8. Создаю исполняемый файл и проверяю его работу. (рис. 4.15).

A terminal window titled 'paefremova@fedora:~/work/arch-pc/lab08' with a search icon, menu icon, and close button. The terminal shows a series of commands and their outputs. The first command is 'nasm -f elf lab8-3.asm', followed by 'ld -m elf\_1386 -o lab8-3 lab8-3.o'. Then, the user enters 'mc', which produces the output 'Результат: 47'. The next command is 'mc' again, which produces 'Результат: 54660'. The terminal ends with a prompt 'paefremova@fedora:~/work/arch-pc/lab08\$'.

```
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_1386 -o lab8-3 lab8-3.o
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
paefremova@fedora:~/work/arch-pc/lab08$ mc
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_1386 -o lab8-3 lab8-3.o
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54660
paefremova@fedora:~/work/arch-pc/lab08$
```

Рис. 4.15: Компиляция и вывод программы



## 5 Задание для самостоятельной работы.

1. Создаю файл для выполнения самостоятельного задания и открываю его. (рис. 5.1).

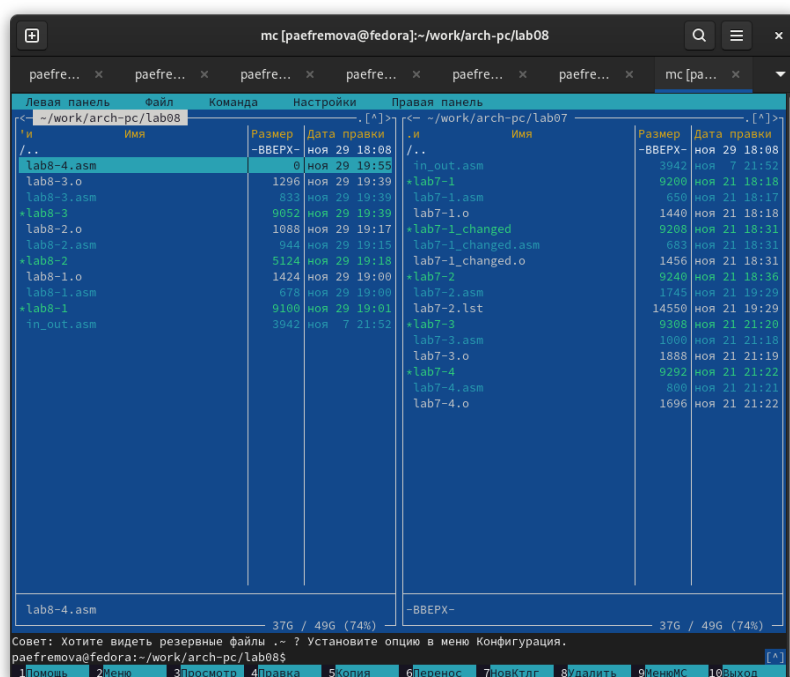
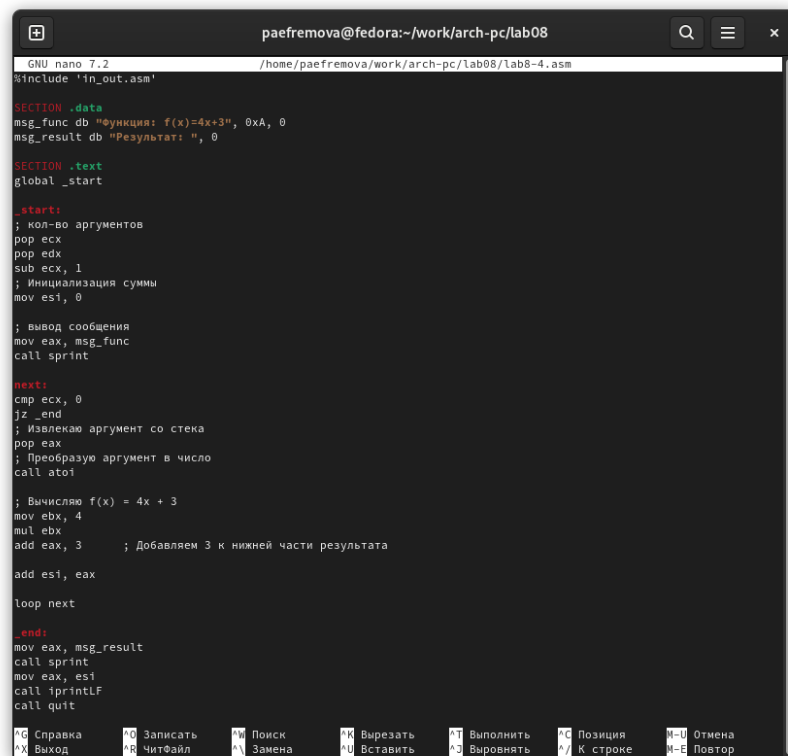


Рис. 5.1: Создание файла

2. Ввожу программу для вычисления функции из варианта 5 (такой же вариант был для прошлых лабораторных) (рис. 5.2).



```
GNU nano 7.2 /home/paefremova/work/arch-pc/lab08/lab8-4.asm
#include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x)=4x+3", 0xA, 0
msg_result db "Результат: ", 0

SECTION .text
global _start

_start:
; кол-во аргументов
pop ecx
pop edx
sub ecx, 1
; инициализация суммы
mov esi, 0

; вывод сообщения
mov eax, msg_func
call sprint

next:
cmp ecx, 0
jz _end
; Извлекаю аргумент со стека
pop eax
; Преобразую аргумент в число
call atoi

; Вычисляю f(x) = 4x + 3
mov ebx, 4
mul ebx
add eax, 3 ; Добавляем 3 к нижней части результата
add esi, eax

loop next

_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintlf
call quit

?Q Справка ?Q Записать ?W Поиск ?K Вырезать ?T Выполнить ?C Позиция ?U Отмена
?X Выход ?A ЧитФайл ?N Замена ?U Вставить ?D Выровнять ?Y К строке ?E Повтор
```

Рис. 5.2: Ввод программы

Ниже прикрепляю сам код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x)=4x+3", 0xA, 0
msg_result db "Результат: ", 0

SECTION .text
global _start

_start:
; кол-во аргументов
pop ecx
```

```

pop edx
sub ecx, 1
; Инициализация суммы
mov esi, 0

; вывод сообщения
mov eax, msg_func
call sprint

next:
cmp ecx, 0
jz _end
; Извлекаю аргумент со стека
pop eax
; Преобразую аргумент в число
call atoi

; Вычисляю  $f(x) = 4x + 3$ 
mov ebx, 4
mul ebx
add eax, 3 ; Добавляем 3 к нижней части результата

add esi, eax

loop next

_end:
mov eax, msg_result
call sprint

```

```
mov eax, esi
call iprintLF
call quit
```

3. Запускаю программу, и вот что получилось (рис. 5.3).

```
paefremova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
paefremova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
paefremova@fedora:~/work/arch-pc/lab08$ ./main 1 2 3 4
bash: ./main: Нет такого файла или каталога
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция:  $f(x)=4x+3$ 
Результат: 52
paefremova@fedora:~/work/arch-pc/lab08$
```

Рис. 5.3: Запуск программы

Ну и еще один пример работы программы (рис. 5.4).

```
paefremova@fedora:~/work/arch-pc/lab08$ ./lab8-4 5 4 6 7
Функция:  $f(x)=4x+3$ 
Результат: 100
paefremova@fedora:~/work/arch-pc/lab08$
```

Рис. 5.4: Запуск программы

## 6 Выводы

Благодаря данной лабораторной работе я научилась использовать большее количество команд на языке ассемблер. Теперь я знаю больше об организации стека, а также как добавлять и извлекать элементы оттуда. Мне понятна и инструкция организации циклов, а также их реализация.

## **Список литературы**

1. Лабораторная работа №8
2. Программирование на языке ассемблера NASM Столяров А. В.
3. Стек в ассемблере - использование команд push и pop