

# **Лабораторная работа №8**

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

Сафиуллина Айлина Саяровна

# 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>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Выводы</b>	<b>20</b>

# Список иллюстраций

4.1	Каталог lab08 . . . . .	9
4.2	Текст из листинга 8.1 . . . . .	10
4.3	Проверка исполняемого файла . . . . .	11
4.4	Замена текста программы . . . . .	11
4.5	Запуск исполняемого файла . . . . .	12
4.6	Замена текста программы . . . . .	13
4.7	Запуск исполняемого файла . . . . .	13
4.8	Создание файла lab8-2.asm . . . . .	14
4.9	Ввод текста из листинга 8.3 . . . . .	15
4.10	Запуск исполняемого файла . . . . .	15
4.11	Создание файла lab8-3.asm . . . . .	16
4.12	Ввод текста из листинга 8.3 . . . . .	16
4.13	Запуск исполняемого файла . . . . .	17
4.14	Создание файла lab8-4.asm . . . . .	17
4.15	Текст программы для функции 12 . . . . .	18
4.16	Запуск программы . . . . .	18
4.17	Запуск программы . . . . .	19
4.18	Запуск программы . . . . .	19

## **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. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

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. Аналогично команде записи в стек существует команда рора, которая восстанавливает из стека все регистры общего назначения, и команда porf для перемещения значений из

вершины стека в регистр флагов. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре есх. Наиболее простой является инструкция loor. Она позволяет организовать безусловный цикл. Инструкция loor выполняется в два этапа. Сначала из регистра есх 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 loor.



## 4 Выполнение лабораторной работы

### 1. Реализация циклов в NASM

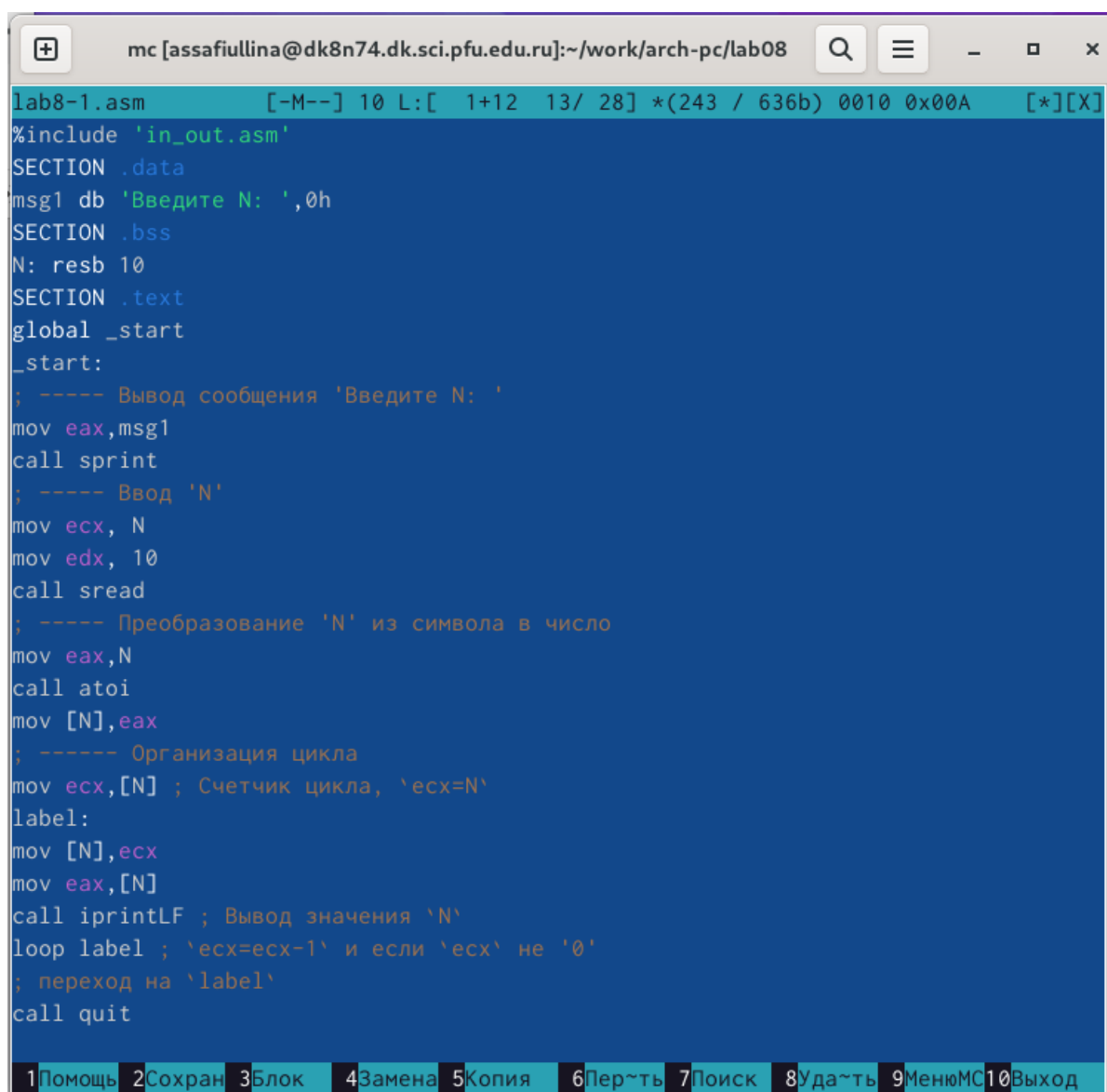
Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рис. 4.1).



```
assafiullina@dk8n74 ~ $ mkdir ~/work/arch-pc/lab08
assafiullina@dk8n74 ~ $ cd ~/work/arch-pc/lab08
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ touch lab8-1.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

Рис. 4.1: Каталог lab08

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



```
lab8-1.asm      [-M--] 10 L:[ 1+12 13/ 28] *(243 / 636b) 0010 0x00A  [*][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:
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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Переть 7Поиск 8Удалить 9МенюМС10Выход
```

Рис. 4.2: Текст из листинга 8.1

Создадим исполняемый файл и проверим его работу (рис. 4.3).

```

assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -t elf lab8-1.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 9
9
8
7
6
5
4
3
2
1
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ █

```

Рис. 4.3: Проверка исполняемого файла

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Изменим текст программы (рис. 4.4), добавив изменение значение регистра `ecx` в цикле:

```

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'

```

Рис. 4.4: Замена текста программы

Создадим исполняемый файл и запустим его. (рис. 4.5).

```
4294658330
4294658328
4294658326
4294658324
4294658322
4294658320
4294658318
4294658316
4294658314
4294658312
4294658310
4294658308
4294658306
4294658304
4294658302
4294658300
4294658298
4294658296
4294658294
4294658292
4294658290
4294658288
4294658286
4294658284
```

Рис. 4.5: Запуск исполняемого файла

В результате данных изменений цикл стал бесконечным и закольцованным. Далее внесем изменения (рис. 4.6) в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop.

```

label:
push ecx
sub ecx,1 ; ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'

```

Рис. 4.6: Замена текста программы

Создадим исполняемый файл и проверим его работу (рис. 4.7)

```

assafiullina@dk8n74 ~/work/arch-pc/lab08 $ mc
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 11
10
9
8
7
6
5
4
3
2
1
0
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ █

```

Рис. 4.7: Запуск исполняемого файла

После изменений программы число циклов стало соответствовать числу, введенному с клавиатуры.

## 2. Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, 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.

Создадим файл lab8-2.asm (рис. 4.8) в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.2. (рис. 4.8)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ touch lab8-2.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

Рис. 4.8: Создание файла lab8-2.asm

```
lab8-2.asm      [-M--]  9 L:[ 1+19 20/ 20] *(943 / 943b) <EOF>
#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
```

Рис. 4.9: Ввод текста из листинга 8.3

Создадим исполняемый файл и запустим его, указав аргументы в строке ./lab8-2 (рис. 4.10)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-2
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент
3'
аргумент1
аргумент
2
аргумент 3
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

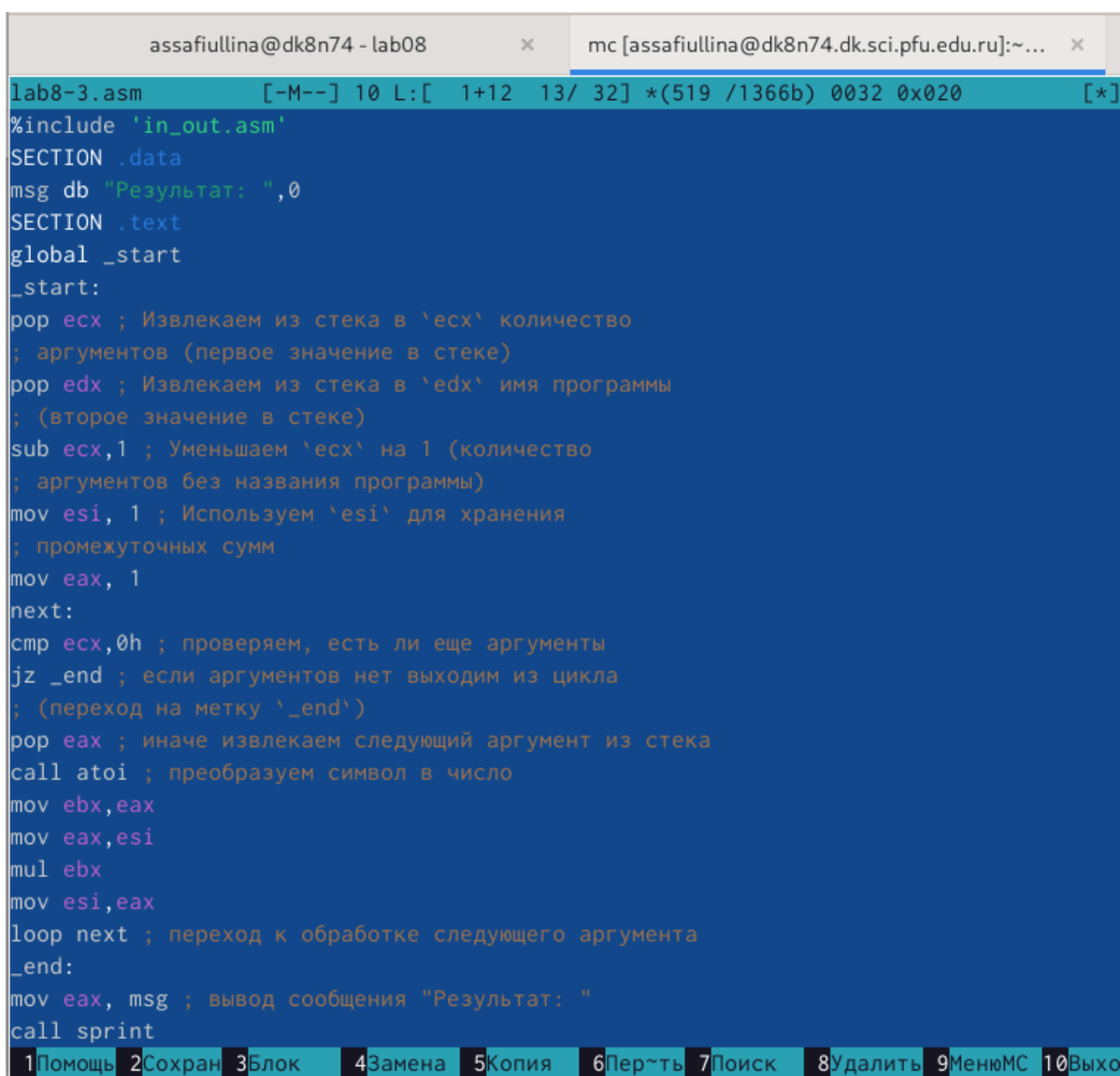
Рис. 4.10: Запуск исполняемого файла

Наша программа обработала 4 аргумента - аргумент1, аргумент, 2, 'аргумент 3'

Создадим файл lab8-3.asm (рис. 4.11) в каталоге ~/work/archpc/lab08 и введем в него текст программы из листинга 8.3 (рис. 4.12)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-3 11 2 4 14 9
Результат: 40
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

Рис. 4.11: Создание файла lab8-3.asm



```
lab8-3.asm      [-M--] 10 L: [ 1+12 13/ 32] *(519 /1366b) 0032 0x020  [*]
%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
mov eax, 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
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Удалить 9МенюМС 10Выхо
```

Рис. 4.12: Ввод текста из листинга 8.3



Поменяем текст программы из листинга 8.3, чтобы выводилось произведение аргументов командной строки

Создадим исполняемый файл и запустим его, указав аргументы (рис. 4.13)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-3 11 2 4 14 9
Результат: 11088
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ █
```

Рис. 4.13: Запуск исполняемого файла

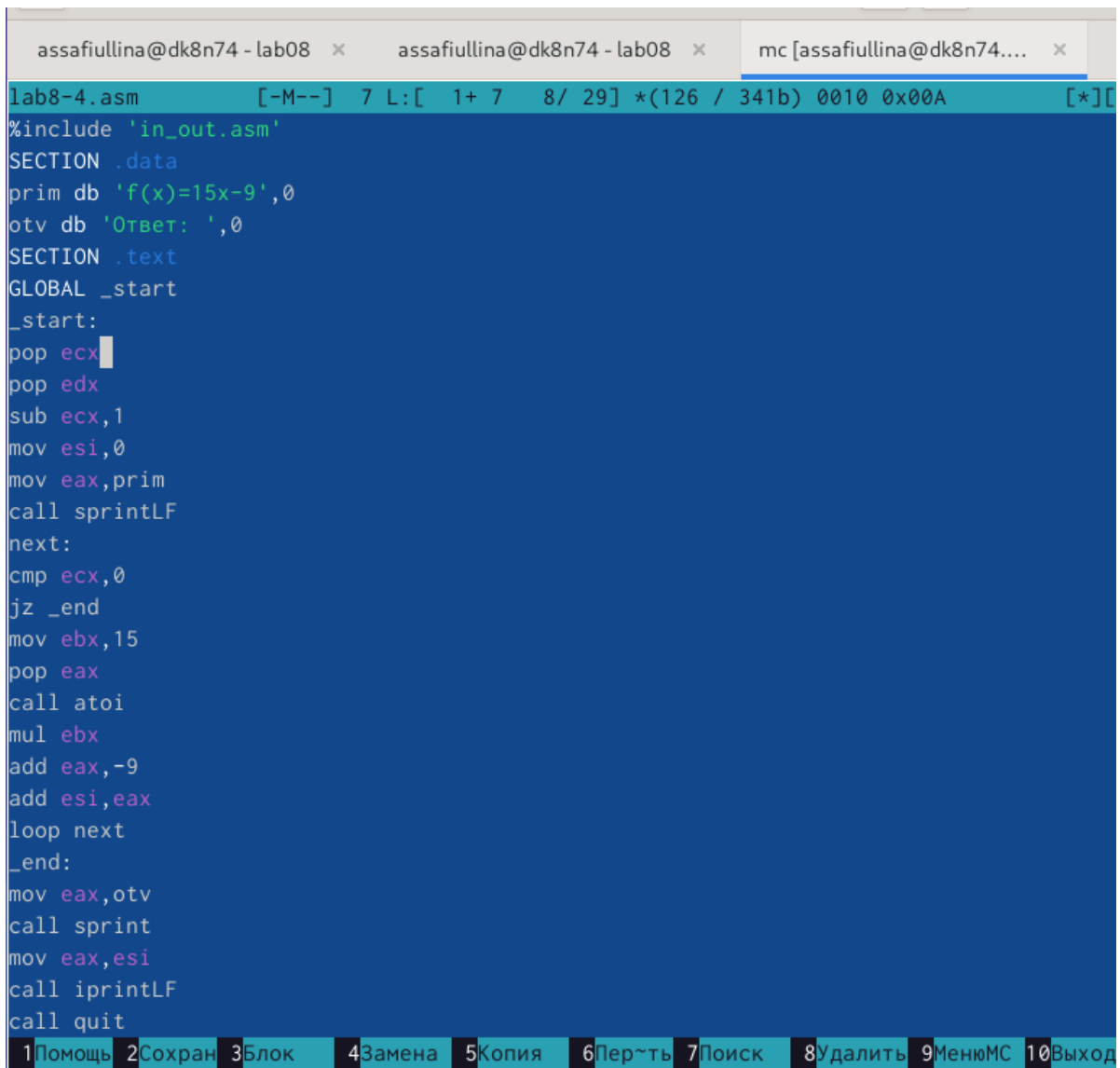
### 3. Задание для самостоятельной работы

Создадим файл lab8-4.asm для выполнения самостоятельной работы (рис. 4.14)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ touch lab8-4.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ █
```

Рис. 4.14: Создание файла lab8-4.asm

Напишем программу (рис. 4.15), 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. Так как у меня вариант 12, моя функция  $15x-9$ .



```
lab8-4.asm [-M--] 7 L:[ 1+ 7 8/ 29] *(126 / 341b) 0010 0x00A [*]  
%include 'in_out.asm'  
SECTION .data  
prim db 'f(x)=15x-9',0  
otv db 'Ответ: ',0  
SECTION .text  
GLOBAL _start  
_start:  
pop ecx  
pop edx  
sub ecx,1  
mov esi,0  
mov eax,prim  
call sprintfLF  
next:  
cmp ecx,0  
jz _end  
mov ebx,15  
pop eax  
call atoi  
mul ebx  
add eax,-9  
add esi,eax  
loop next  
_end:  
mov eax,otv  
call sprintf  
mov eax,esi  
call iprintLF  
call quit  
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.15: Текст программы для функции 12

Создадим исполняемый файл и проверим его работу на нескольких наборах.(рис. 4.16), (рис. 4.17), (рис. 4.18)

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm  
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o  
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-4  
f(x)=15x-9  
Ответ: 0  
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

Рис. 4.16: Запуск программы

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-4 9 4 17 3 1
f(x)=15x-9
Ответ: 465
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

Рис. 4.17: Запуск программы

```
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
assafiullina@dk8n74 ~/work/arch-pc/lab08 $ ./lab8-4 6 12 3 15 7
f(x)=15x-9
Ответ: 600
assafiullina@dk8n74 ~/work/arch-pc/lab08 $
```

---

Рис. 4.18: Запуск программы

## **5 Выводы**

В ходе выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов NASM и обработкой аргументов командной строки, а так же выполнила задание для самостоятельной работы.