

# **Отчёт по лабораторной работе №8**

**Дисциплина: архитектура компьютера**

Черкашина Ангелина Максимовна

# 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>
4.1	Реализация циклов в NASM . . . . .	9
4.2	Обработка аргументов командной строки . . . . .	14
4.3	Выполнение задания для самостоятельной работы . . . . .	19
<b>5</b>	<b>Выводы</b>	<b>22</b>
<b>6</b>	<b>Список литературы</b>	<b>23</b>

## Список иллюстраций

4.1	Создание каталога и файла . . . . .	9
4.2	Создание копии файла . . . . .	9
4.3	Редактирование файла . . . . .	10
4.4	Запуск исполняемого файла . . . . .	10
4.5	Изменение текста программы . . . . .	11
4.6	Запуск нового исполняемого файла . . . . .	12
4.7	Изменение текста программы . . . . .	13
4.8	Запуск обновленной программы . . . . .	13
4.9	Создание файла . . . . .	14
4.10	Редактирование файла . . . . .	14
4.11	Запуск исполняемого файла с нужными аргументами . . . . .	15
4.12	Создание файла . . . . .	15
4.13	Редактирование файла . . . . .	16
4.14	Запуск исполняемого файла с аргументами . . . . .	16
4.15	Изменение текста программы . . . . .	17
4.16	Запуск исполняемого файла . . . . .	17
4.17	Написание программы . . . . .	19
4.18	Запуск исполняемого файла и проверка его работы . . . . .	20

## **Список таблиц**

# 1 Цель работы

Целью данной лабораторной работы является 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. Выполнение задания 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. 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 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

### 4.1 Реализация циклов в NASM

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

```
amcherkashina@dk8n53 ~ $ mkdir ~/work/arch-pc/lab08
amcherkashina@dk8n53 ~ $ cd ~/work/arch-pc/lab08
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ touch lab8-1.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.1: Создание каталога и файла

С помощью команды `cp` копирую в текущий каталог файл `in_out.asm`, т.к. он будет использоваться в программах данной лабораторной работы (рис. 4.2).

```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ cp ~/Загрузки/in_out.asm in_out.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.2: Создание копии файла

Ввожу в файл `lab8-1.asm` текст программы вывода значений регистра `ecx` из листинга 8.1 (рис. 4.3).

```
lab8-1.asm [----] 0 L: [ 1+ 9 10/ 33] *(301 / 846b) 0010 0x00A [*][X]
;
; 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
```

Рис. 4.3: Редактирование файла

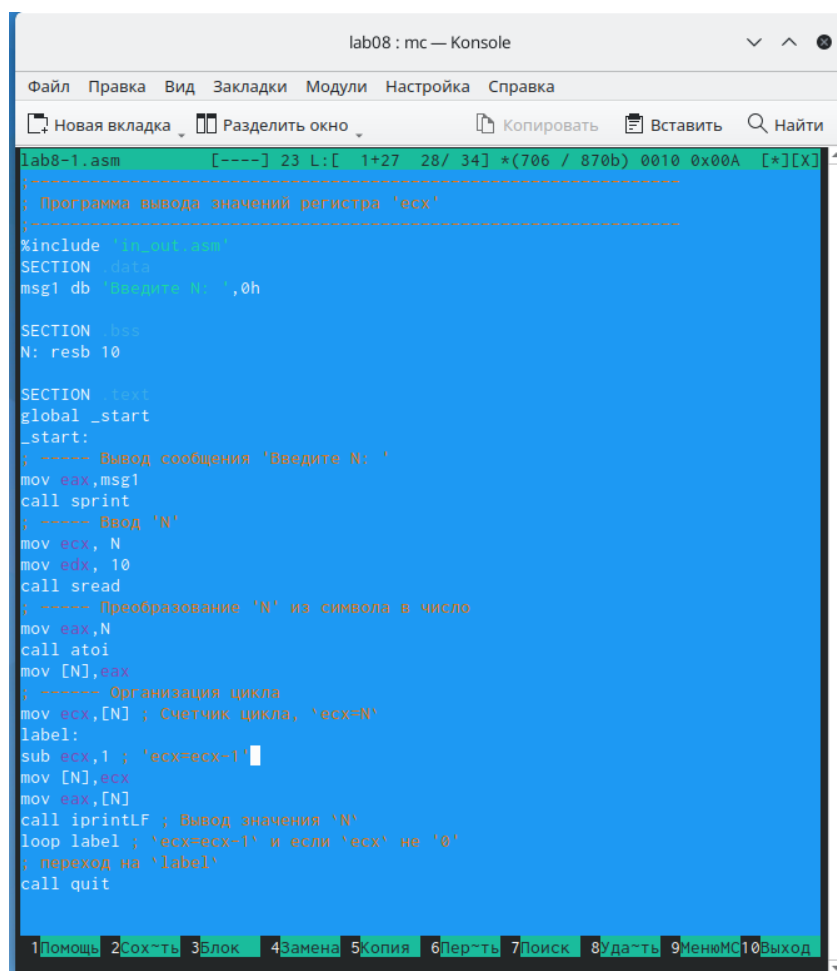
Создаю исполняемый файл и запускаю его (рис. 4.4).

```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-1.o -o lab8-1
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.4: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.  
Изменяю текст программы, добавив изменение значения регистра ecx в цикле

(рис. 4.5).



```
lab8-1.asm [----] 23 L: [ 1+27 28/ 34] *(706 / 870b) 0010 0x00A [*][X]
;
; Программа вывода значений регистра 'ecx'
;
-----
%include "in_out.asm"
SECTION .data
msg1 db "Введите N: ",0h

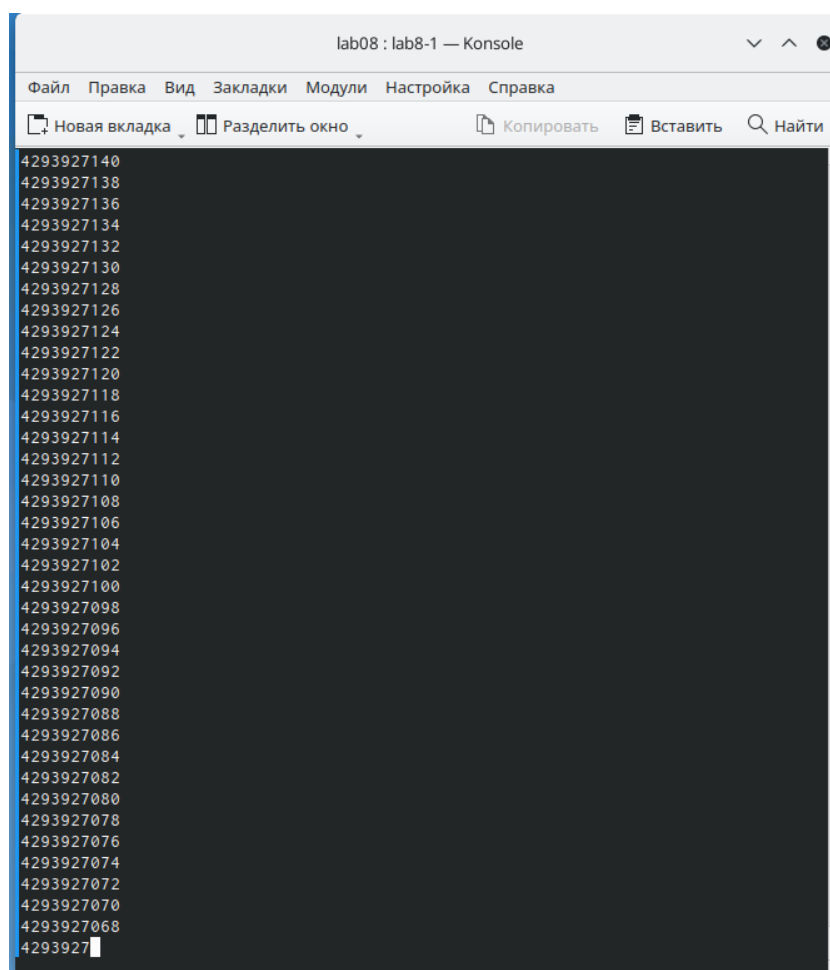
SECTION .bss
N: resb 10

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:
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

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюMC10Выход
```

Рис. 4.5: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.6).



```
lab08 : lab8-1 — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
4293927140
4293927138
4293927136
4293927134
4293927132
4293927130
4293927128
4293927126
4293927124
4293927122
4293927120
4293927118
4293927116
4293927114
4293927112
4293927110
4293927108
4293927106
4293927104
4293927102
4293927100
4293927098
4293927096
4293927094
4293927092
4293927090
4293927088
4293927086
4293927084
4293927082
4293927080
4293927078
4293927076
4293927074
4293927072
4293927070
4293927068
4293927068
```

Рис. 4.6: Запуск нового исполняемого файла

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению N.

Вношу изменения в текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. 4.7).

```
lab8-1.asm [----] 42 L: [ 1+32 33/ 36] *(911 /1003b) 0010 0x00A [*][X]
;
; 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:
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

1 Помощь 2 Сох-ть 3 Блок 4 Замена 5 Копия 6 Пер-ть 7 Поиск 8 Уда-ть 9 МенюМС 10 Выход
```

Рис. 4.7: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.8).

```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-1.o -o lab8-1
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.8: Запуск обновленной программы

В данном случае число проходов цикла соответствует введенному с клавиатуры

значению и выводит числа от N-1 до 0 включительно.

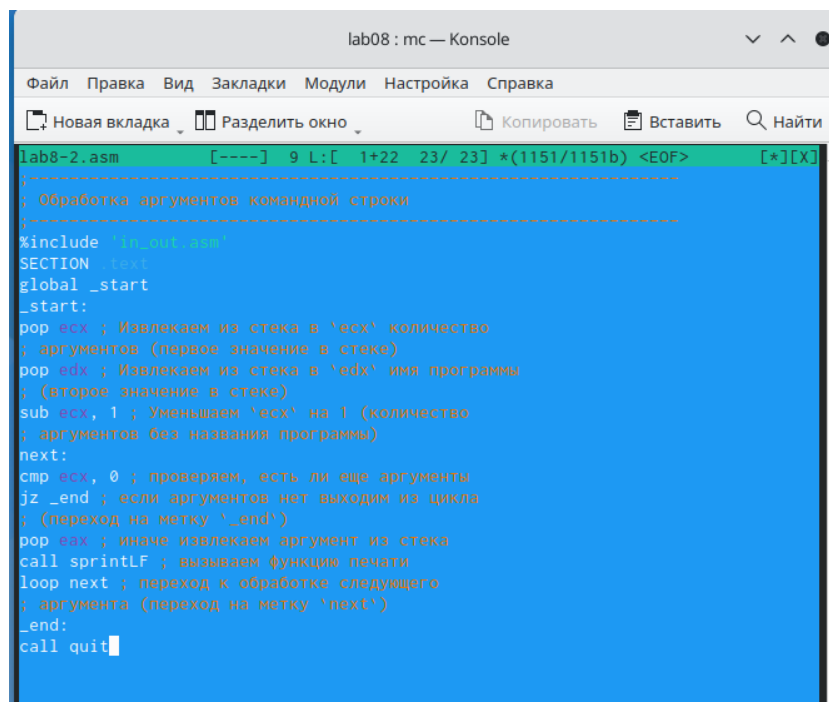
## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 с помощью команды touch (рис. 4.9).

```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ touch lab8-2.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.9: Создание файла

Ввожу в него текст программы, выводящей на экран аргументы командной строки, из листинга 8.2 (рис. 4.10).



```
lab08 : mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
[+] Новая вкладка  [ ] Разделить окно  [ ] Копировать  [ ] Вставить  [ ] Найти
lab8-2.asm  [----]  9  L: [ 1+22  23/ 23]  *(1151/1151b)  <E0F>  [*][X]
; 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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку 'next')
_end:
call quit
```

Рис. 4.10: Редактирование файла

Создаю исполняемый файл и запускаю его, указав следующие аргументы: аргумент1 аргумент 2 'аргумент 3' (рис. 4.11).

```

amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-2.o -o lab8-2
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $

```

Рис. 4.11: Запуск исполняемого файла с нужными аргументами

Программа обработала и вывела 4 аргумента. Так как аргумент 2 не был взят в кавычки, в отличие от 'аргумент 3', из-за наличия пробела программа считывает "2" как отдельный аргумент.

Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 с помощью команды touch (рис. 4.12).

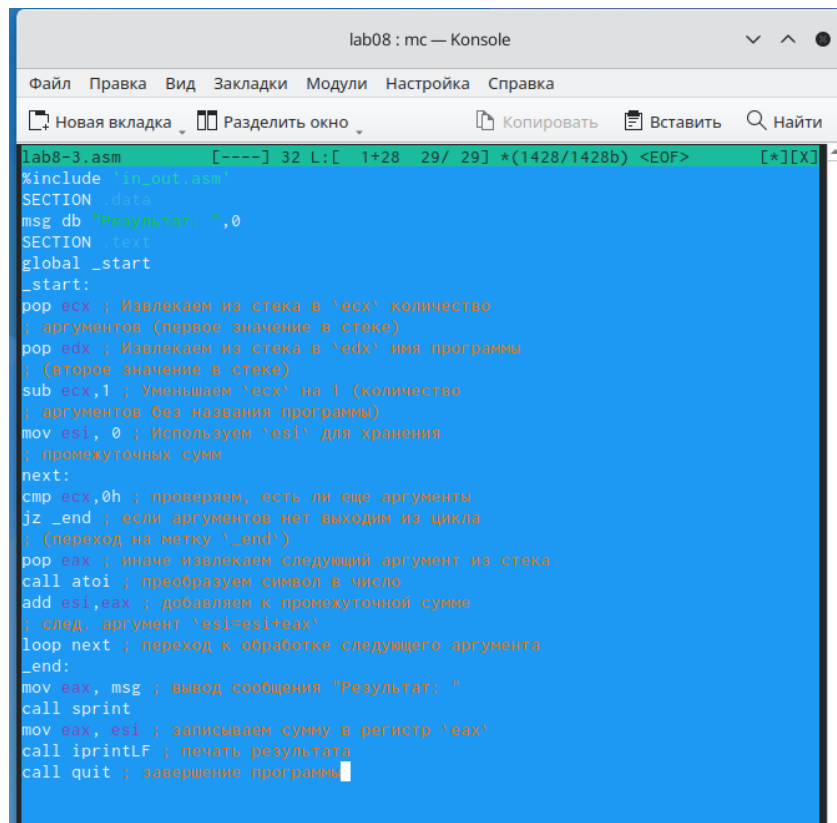
```

amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ touch lab8-3.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $

```

Рис. 4.12: Создание файла

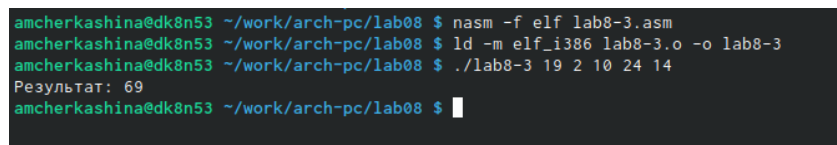
Ввожу в него текст программы вычисления суммы аргументов командной строки из листинга 8.3 (рис. 4.13).



```
lab8-3.asm [----] 32 L: [ 1+28 29/ 29] *(1428/1428b) <EOF> [*][X]
#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
```

Рис. 4.13: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.14).

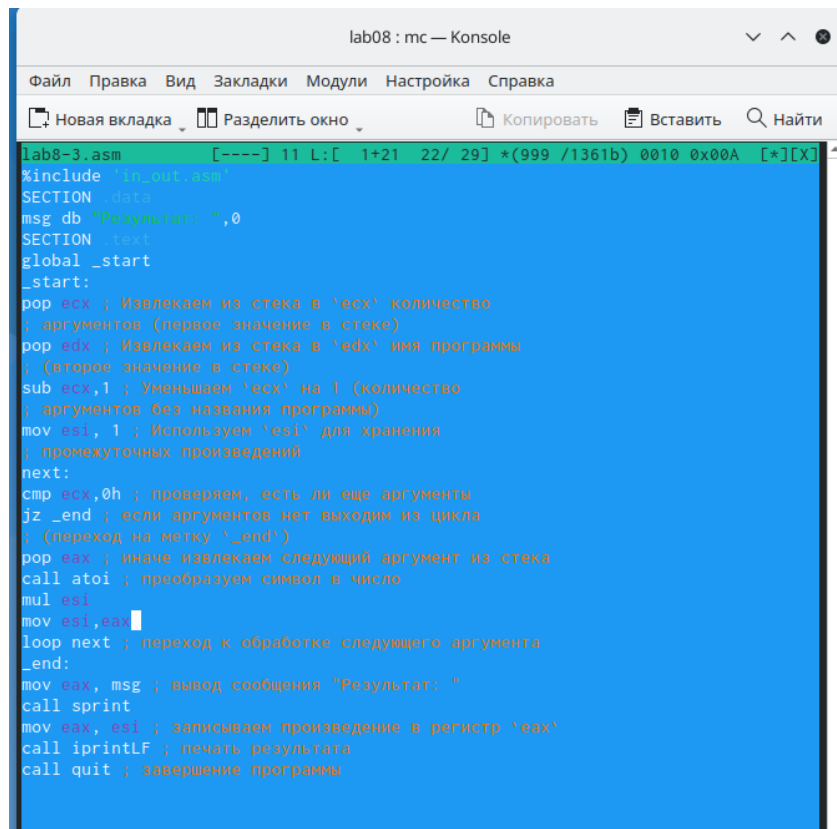


```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-3.o -o lab8-3
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-3 19 2 10 24 14
Результат: 69
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.14: Запуск исполняемого файла с аргументами

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. Для этого изменяю изначальное значение счетчика на 1, а add на mul (рис. 4.15).

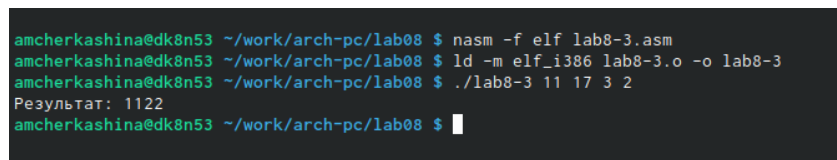




```
lab8-3.asm [----] 11 L: [ 1+21 22/ 29] *(999 /1361b) 0010 0x00A [*][X]
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в 'ecx' количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в 'edx' имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем 'esi' для хранения
; промежуточных произведений
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 sprintf
mov eax, esi ; записываем произведение в регистр 'eax'
call iprintf ; печать результата
call quit ; завершение программы
```

Рис. 4.15: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.16).



```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-3.o -o lab8-3
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-3 11 17 3 2
Результат: 1122
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.16: Запуск исполняемого файла

#### Листинг 4.1. Программа вычисления произведения аргументов командной строки

```
%include 'in_out.asm'

SECTION .data

msg db "Результат: ",0
```

```

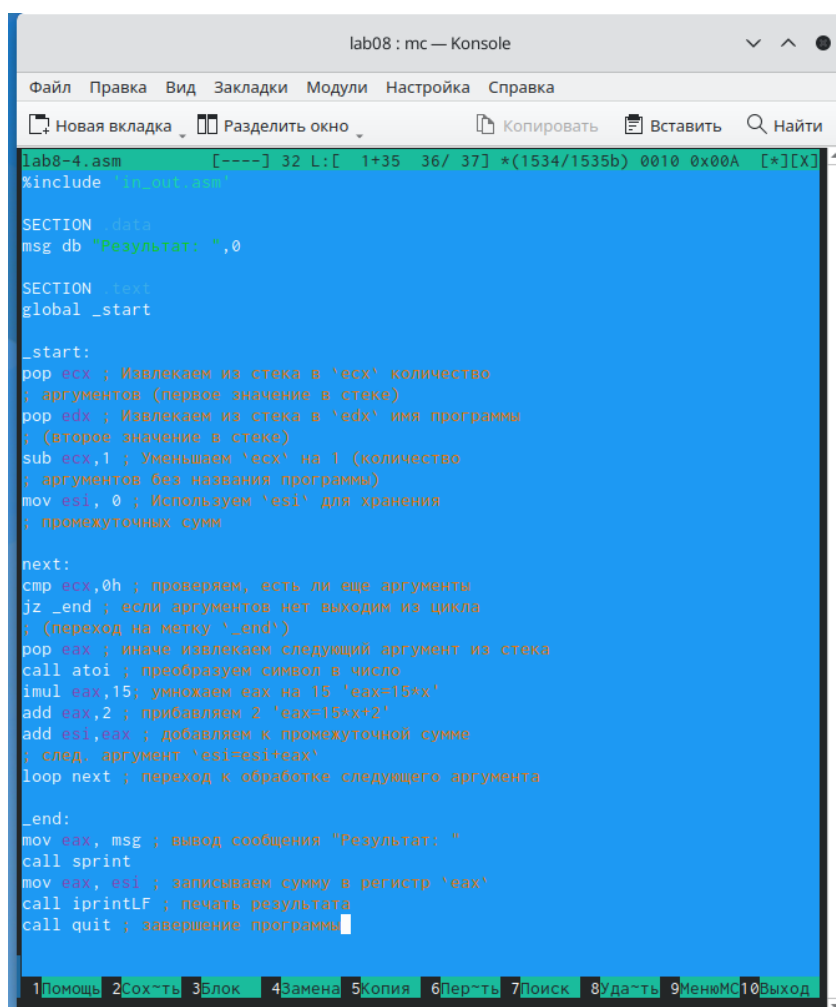
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем `esi` для хранения
; промежуточных произведений
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 произведение в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

```

## 4.3 Выполнение задания для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программу, выводящую значения  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы. Вид функции  $f(x)$  выбираю из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным мной при выполнении лабораторной работы №6.

Мой вариант 11, соответственно пишу программу нахождения суммы значений функции  $f(x) = 15 \cdot x + 2$  (рис. 4.17).



```
lab08-4.asm  [----] 32 L: [ 1+35 36/ 37] *(1534/1535b) 0010 0x00A [*][X]
#include "io_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
imul eax,15 ; умножаем eax на 15 'eax=15*x'
add eax,2 ; прибавляем 2 'eax=15*x+2'
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
```

Рис. 4.17: Написание программы

Создаю исполняемый файл и проверяю его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$  (рис. 4.18).

```
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ld -m elf_i386 lab8-4.o -o lab8-4
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Результат: 158
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-4 5 6 7
Результат: 276
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-4 19 2
Результат: 319
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $ ./lab8-4 10 21 3 14
Результат: 728
amcherkashina@dk8n53 ~/work/arch-pc/lab08 $
```

Рис. 4.18: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

**\*\*Листинг 4.2.** Программа нахождения суммы значений функции  $f(x) = 15 \cdot x + 2$

```
%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

**imul** **eax**,15; умножаем **eax** на 15 ' $eax=15*x$ '

**add** **eax**,2 ; прибавляем 2 ' $eax=15*x+2$ '

**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

## 5 Выводы

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

## **6 Список литературы**

### **1. Архитектура ЭВМ**