

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура компьютера**

Абрамова Ульяна Михайловна

# 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>
3.1	Организация стека . . . . .	7
3.1.1	Добавление элемента в стек . . . . .	7
3.1.2	Извлечение элемента из стека . . . . .	8
3.2	Инструкции организации циклов . . . . .	8
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация циклов в NASM . . . . .	9
4.2	Обработка аргументов командной строки . . . . .	13
4.3	Выполнение задания для самостоятельной работы . . . . .	16
<b>5</b>	<b>Выводы</b>	<b>18</b>
<b>6</b>	<b>Список литературы</b>	<b>19</b>

# Список иллюстраций

4.1	Написание программы . . . . .	9
4.2	Запуск программы . . . . .	10
4.3	Изменение программы . . . . .	10
4.4	Запуск программы . . . . .	11
4.5	Изменение программы . . . . .	12
4.6	Запуск программы . . . . .	12
4.7	Написание программы . . . . .	13
4.8	Запуск программы . . . . .	13
4.9	Написание программы . . . . .	14
4.10	Запуск программы . . . . .	14
4.11	Изменение программы . . . . .	15
4.12	Запуск программы . . . . .	15
4.13	Написание программы . . . . .	16
4.14	Запуск программы . . . . .	17

## **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. Выполнение задания для самостоятельной работы

## 3 Теоретическое введение

### 3.1 Организация стека

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).

#### 3.1.1 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.

### 3.1.2 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. Аналогично команде записи в стек существует команда `push`, которая восстанавливает из стека все регистры общего назначения, и команда `popf` для перемещения значений из вершины стека в регистр флагов.

## 3.2 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, типичная структура которого имеет следующий вид:

```
mov ecx, 100      ; Количество проходов
NextStep:
...
...              ; тело цикла
...
loop NextStep     ; Повторить `ecx` раз от метки NextStep
```

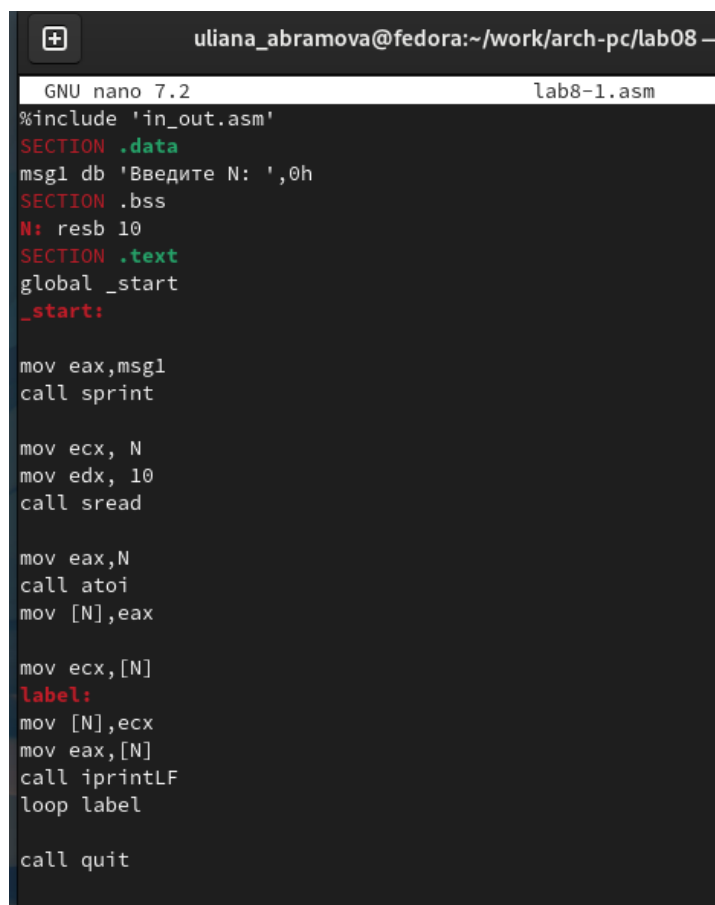
Инструкция `loop` выполняется в два этапа. Сначала из регистра `ecx` вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.



## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8, перехожу в него и создаю файл lab8-1.asm, в который записываю программу вывода значений регистра ecx (рис. 4.1).



```
GNU nano 7.2 lab8-1.asm
#include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

call quit
```

Рис. 4.1: Написание программы

Создаю исполняемый файл и проверяю его работу (рис. 4.2).

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

Рис. 4.2: Запуск программы

Далее изменяю текст программы, добавив изменение значения регистра `ecx` в цикле (рис. 4.3).

```
uliana_abramova@fedora:~/work/arch-pc/lab08 —
GNU nano 7.2 lab8-1.asm
#include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

call quit
```

Рис. 4.3: Изменение программы

Создаю исполняемый файл и проверяю его работу. Из-за изменения цикл за-

кольцевался и стал бесконечным (рис. 4.4).

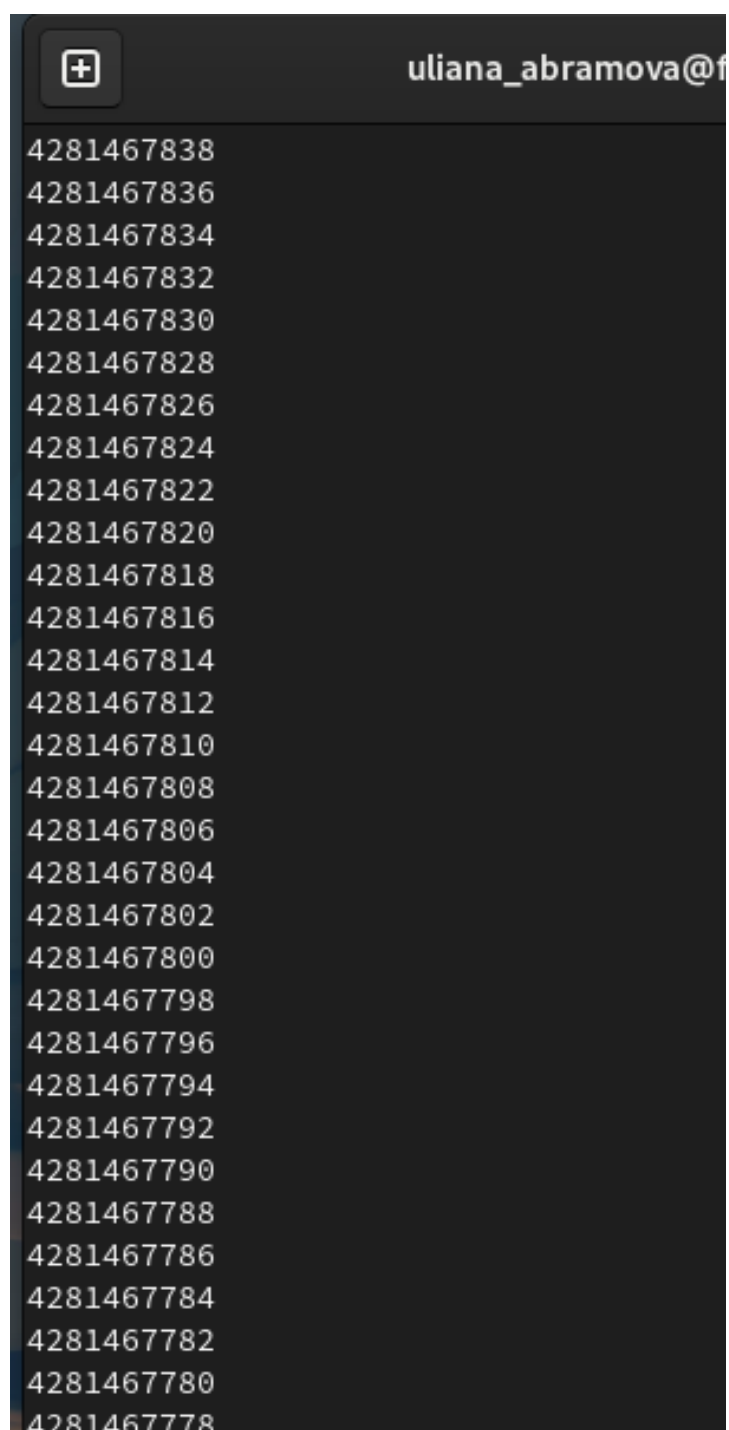
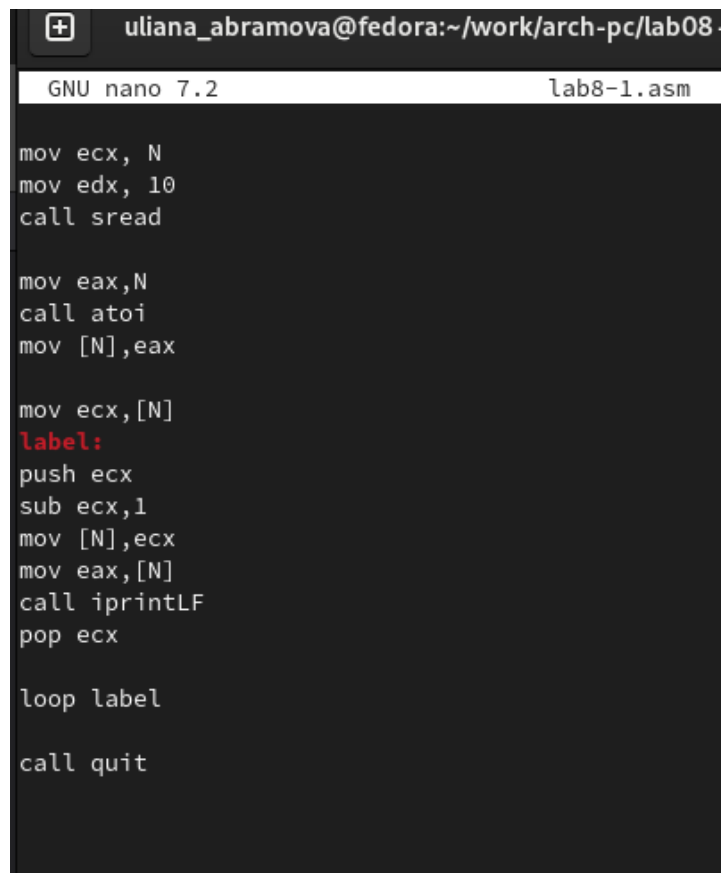


Рис. 4.4: Запуск программы

Снова ввожу изменения в текст программы для корректной работы цикла и

счетчика (рис. 4.5).



```
uliana_abramova@fedora:~/work/arch-pc/lab08 -
GNU nano 7.2 lab8-1.asm

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

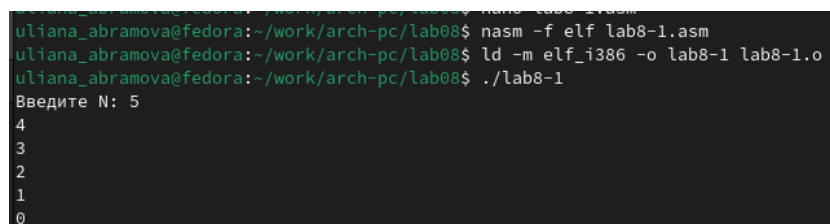
mov ecx,[N]
label:
push ecx
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx

loop label

call quit
```

Рис. 4.5: Изменение программы

Создаю исполняемый файл и проверяю его работу. По итогу изменения программы, число проходки циклов стало соответствовать числу введенному с клавиатуры (рис. 4.6).

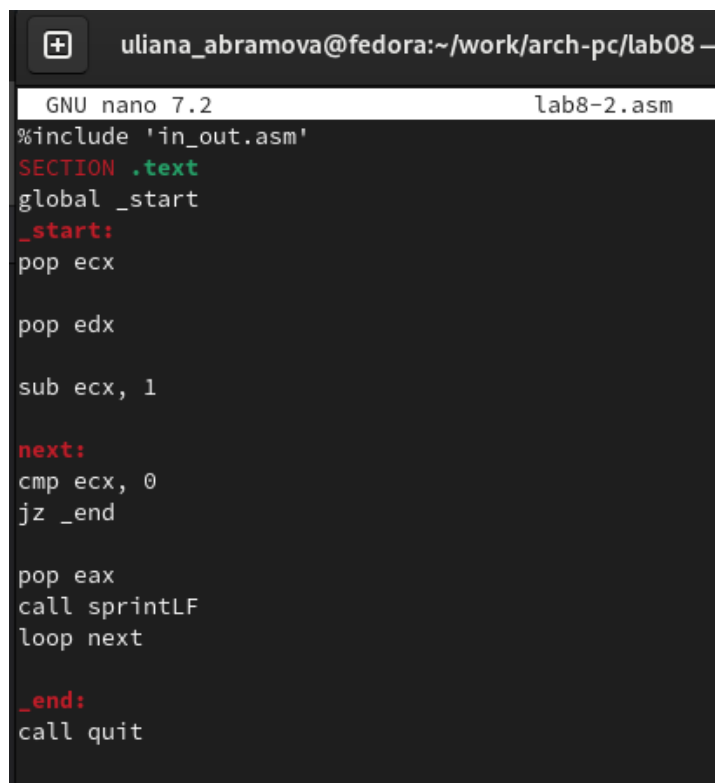


```
uliana_abramova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
uliana_abramova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
uliana_abramova@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
```

Рис. 4.6: Запуск программы

## 4.2 Обработка аргументов командной строки

Создаю в том же каталоге файл lab8-2.asm и пишу в нем программу, выводящую на экран аргументы командной строки (рис. 4.7).



```
uliana_abramova@fedora:~/work/arch-pc/lab08 —
GNU nano 7.2 lab8-2.asm
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx

pop edx

sub ecx, 1

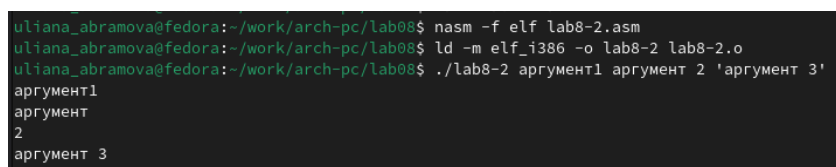
next:
cmp ecx, 0
jz _end

pop eax
call sprintLF
loop next

_end:
call quit
```

Рис. 4.7: Написание программы

Создаю исполняемый файл и проверяю его работу. Данная программа выводит все 3 введенных аргумента, но в разной вариации. (рис. 4.8)



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

Рис. 4.8: Запуск программы

Создаю новый файл lab8-3.asm в прежнем каталоге и пишу в него программу вычисления суммы аргументов командной строки (рис. 4.9).

```
uliana_abramova@fedora:~/work/arch-pc/lab8$ nano lab8-3.asm
GNU nano 7.2 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 iprintLF
call quit
```

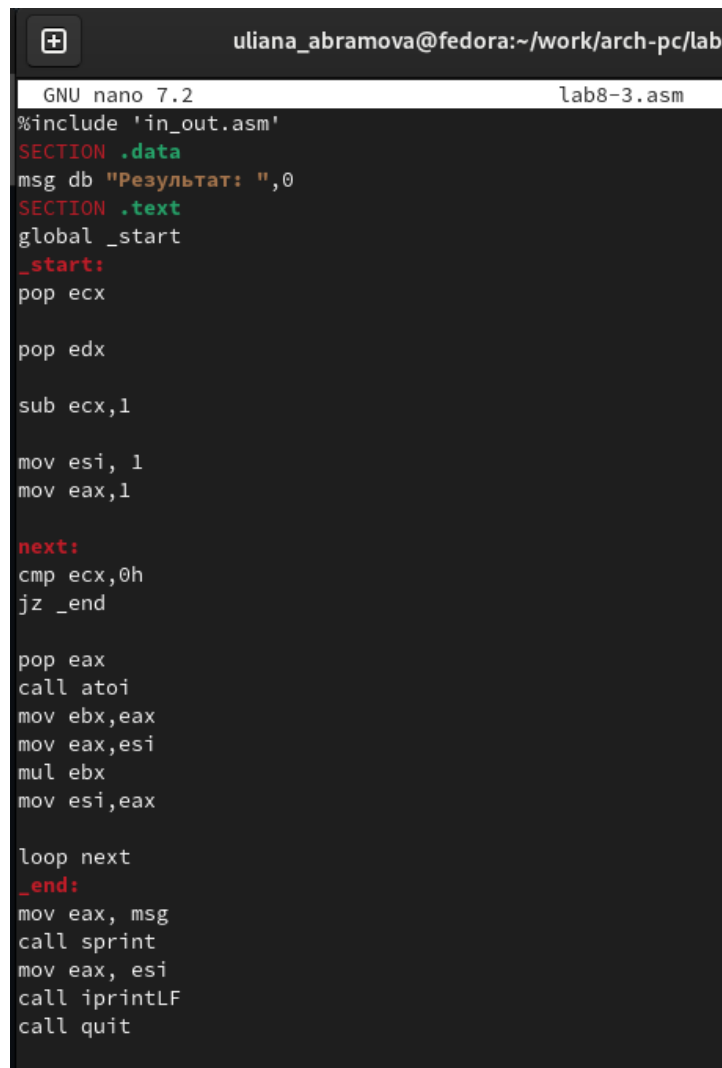
Рис. 4.9: Написание программы

Создаю исполняемый файл и проверяю его работу с указанными аргументами (рис. 4.10).

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

Рис. 4.10: Запуск программы

Изменяю текст программы для вычисления произведения аргументов командной строки (рис. 4.11).



```
uliana_abramova@fedora:~/work/arch-pc/lab8$ nano lab8-3.asm
GNU nano 7.2 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, 1
mov eax,1

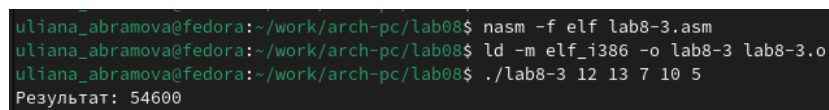
next:
cmp ecx,0h
jz _end

pop eax
call atoi
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax

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

Рис. 4.11: Изменение программы

Создаю исполняемый файл и проверяю его работу. Программа отработана верно (рис. 4.12).

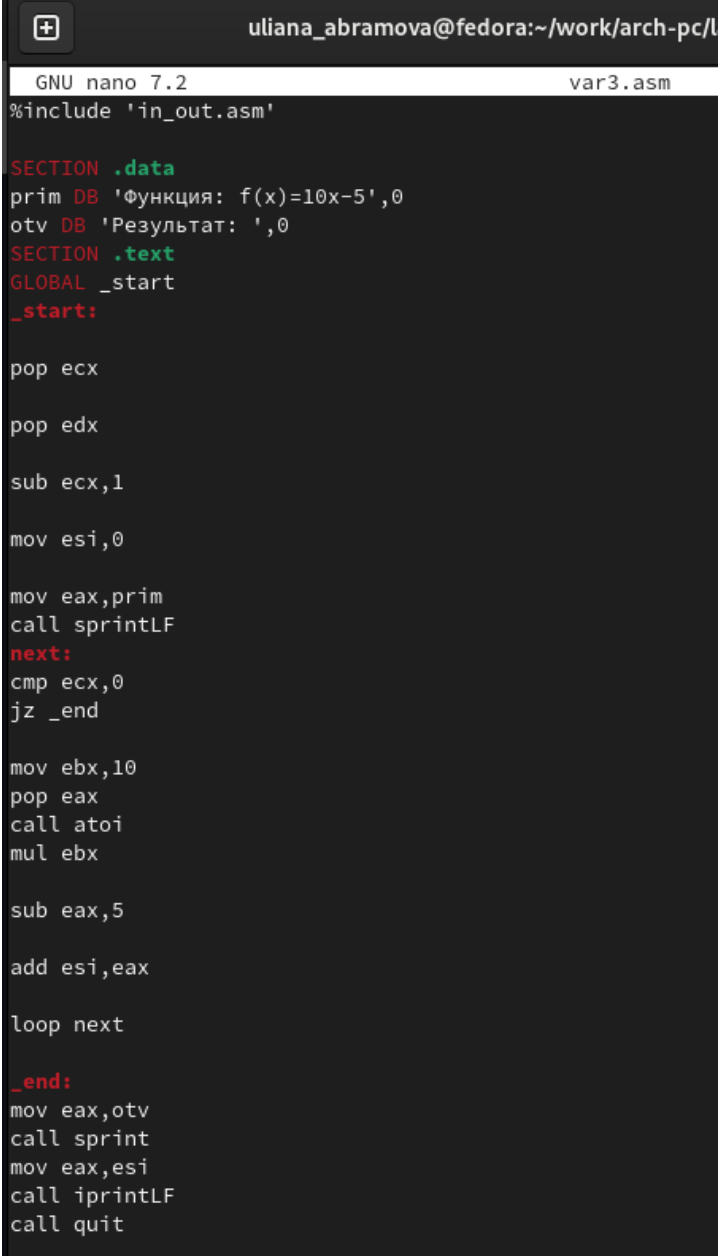


```
uliana_abramova@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
uliana_abramova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
uliana_abramova@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
```

Рис. 4.12: Запуск программы

## 4.3 Выполнение задания для самостоятельной работы

Нужно написать программу, которая находит сумму значений функции  $f(x)$  для  $x=x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1)+f(x_2)+\dots+f(x_n)$ . Вид функции моего, 3 варианта:  $f(x)=10x-5$ , соответственно программа написана для этого вида (рис. 4.13).



```
uliana_abramova@fedora:~/work/arch-pc/l
GNU nano 7.2                                var3.asm
#include 'in_out.asm'

SECTION .data
prim DB 'Функция: f(x)=10x-5',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,prim
call sprintf
next:
cmp ecx,0
jz _end

mov ebx,10
pop eax
call atoi
mul ebx

sub eax,5

add esi,eax

loop next

_end:
mov eax,otv
call sprintf
mov eax,esi
call iprintLF
call quit
```

Рис. 4.13: Написание программы



Создаю исполняемый файл и проверяю его работу при разных значениях аргументов. Программа отработана верно (рис. 4.14).

```
uliana_abramova@fedora:~/work/arch-pc/lab08$ nasm -f elf var3.asm
uliana_abramova@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o var3 var3.o
uliana_abramova@fedora:~/work/arch-pc/lab08$ ./var3 1 2 3 4
Функция:  $f(x)=10x-5$ 
Результат: 80
uliana_abramova@fedora:~/work/arch-pc/lab08$ ./var3 2 3 4
Функция:  $f(x)=10x-5$ 
Результат: 75
uliana_abramova@fedora:~/work/arch-pc/lab08$ ./var3 1 2 3 4 5
Функция:  $f(x)=10x-5$ 
Результат: 125
```

Рис. 4.14: Запуск программы

## **5 Выводы**

Я приобрела навыки написания программы с использованием цикла.

## **6 Список литературы**

1. Архитектура компьютера