

# **Отчёт по лабораторной работе №9**

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

Поляков Глеб Сергеевич

# 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. . . . .	8
3.1.2	Извлечение элемента из стека. . . . .	9
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Задание для самостоятельной работы . . . . .	17
<b>5</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

# Список иллюстраций

3.1	Организация стека в процессоре . . . . .	8
4.1	Текст программы lab9-1.asm . . . . .	11
4.2	Результат программы lab9-1.asm . . . . .	12
4.3	Результат программы lab9-1.asm . . . . .	13
4.4	Результат программы lab9-1.asm . . . . .	13
4.5	Текст программы lab9-2.asm . . . . .	14
4.6	Результат программы lab9-2.asm . . . . .	14
4.7	Текст программы lab9-3.asm . . . . .	15
4.8	Результат программы lab9-3.asm . . . . .	16
4.9	Результат программы lab9-3.asm . . . . .	17
4.10	Текст программы lab9-4.asm . . . . .	18
4.11	Результат программы lab9-4.asm . . . . .	19

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Создать каталог и файл lab9-1.asm
2. Написать текст программы lab9-1
3. Создать каталог и файл lab9-2.asm
4. Написать текст программы lab9-2
5. Создать каталог и файл lab9-3.asm
6. Написать текст программы lab9-3
7. Создать каталог и файл lab9-4.asm
8. Написать текст программы lab9-4

## 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.

На рис. 9.1 показана схема организации стека в процессоре.

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).

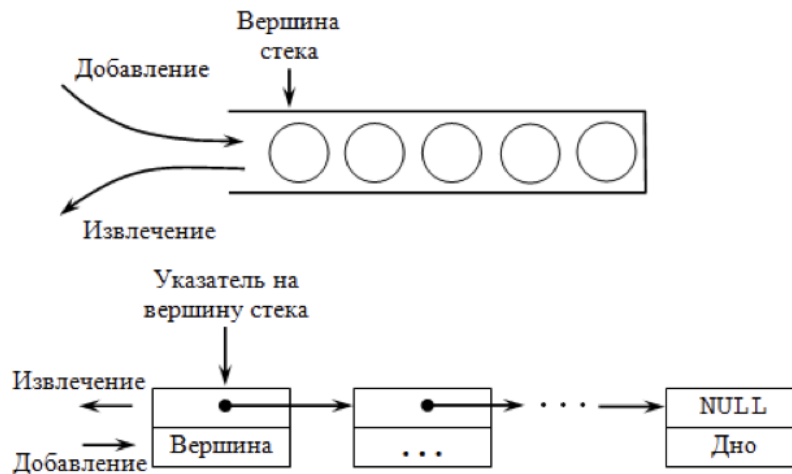


Рис. 3.1: Организация стека в процессоре

### 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. Примеры:

```
push -10      ; Поместить -10 в стек
push ebx      ; Поместить значение регистра ebx в стек
push [buf]    ; Поместить значение переменной buf в стек
push word [ax] ; Поместить в стек слово по адресу в ax
```

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.



### 3.1.2 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.

Примеры:

`pop eax` ; Поместить значение из стека в регистр `eax`

`pop [buf]` ; Поместить значение из стека в `buf`

`pop word[si]` ; Поместить значение из стека в слово по адресу в `si`

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

```
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.

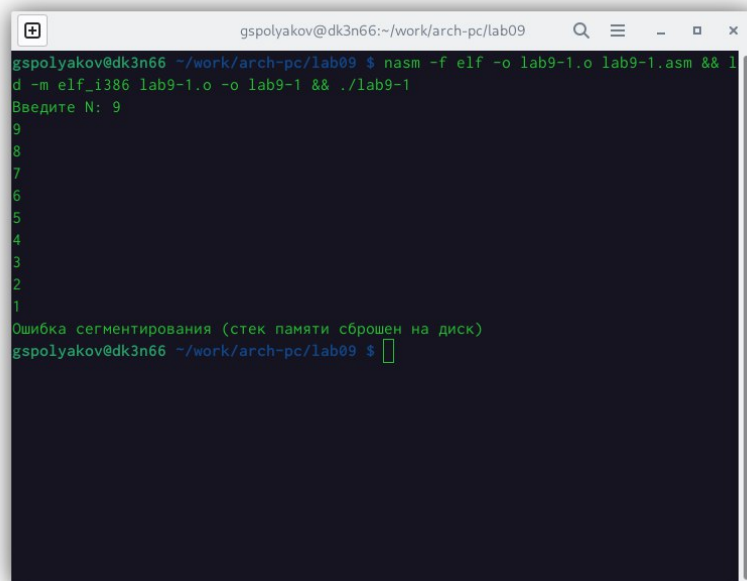
## 4 Выполнение лабораторной работы

1. Создал каталог и файл lab9-1.asm
2. Ввел текст программы в файл (рис. 4.1)

```
1  %include 'in_out.asm'
2
3  SECTION .data
4  msg1 db 'Введите N: ',0h
5
6  SECTION .bss
7  N: resb 10
8
9  SECTION .text
10 global _start
11
12 _start:
13     mov eax,msg1
14     call sprintf
15
16     mov ecx, N
17     mov edx, 10
18     call sread
19
20     mov eax,N
21     call atoi
22     mov [N],eax
23
24     mov ecx,[N]
25
26 label:
27     mov [N],ecx
28     mov eax,[N]
29     call iprintf
30
31     loop label
32
```

Рис. 4.1: Текст программы lab9-1.asm

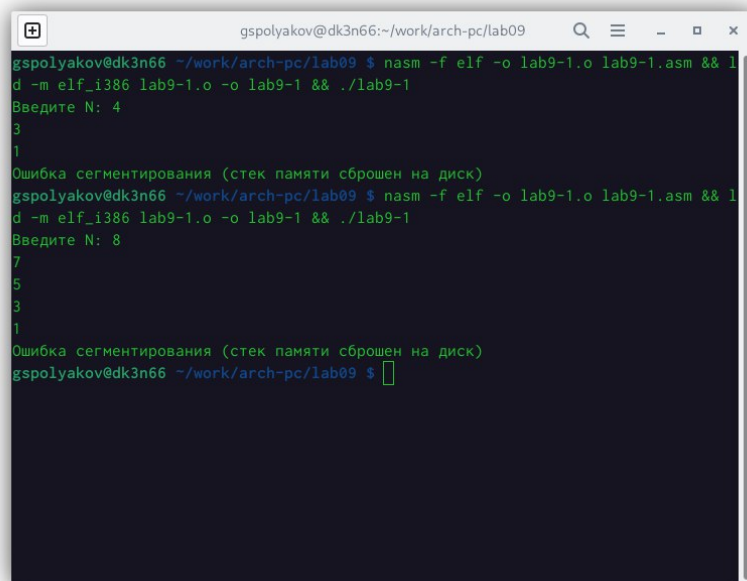
3. Проверил работоспособность программы (рис. 4.2)



```
gspolyakov@dk3n66:~/work/arch-pc/lab09
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-1.o lab9-1.asm && l
d -m elf_i386 lab9-1.o -o lab9-1 && ./lab9-1
Введите N: 9
9
8
7
6
5
4
3
2
1
Ошибка сегментирования (стек памяти сброшен на диск)
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $
```

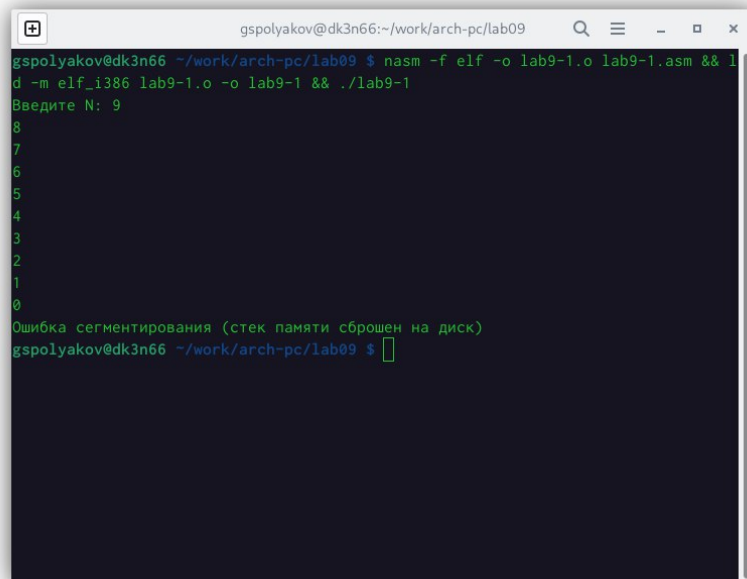
Рис. 4.2: Результат программы lab9-1.asm

4. Также проверил работу программы с изменениями (рис. 4.3) и (рис. 4.4), регистр `ecx` принимает значения 9-1, число проходов соответствует значению `N`.



```
gspolyakov@dk3n66:~/work/arch-pc/lab09
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-1.o lab9-1.asm && l
d -m elf_i386 lab9-1.o -o lab9-1 && ./lab9-1
Введите N: 4
3
1
Ошибка сегментирования (стек памяти сброшен на диск)
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-1.o lab9-1.asm && l
d -m elf_i386 lab9-1.o -o lab9-1 && ./lab9-1
Введите N: 8
7
5
3
1
Ошибка сегментирования (стек памяти сброшен на диск)
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $
```

Рис. 4.3: Результат программы lab9-1.asm



```
gspolyakov@dk3n66:~/work/arch-pc/lab09
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-1.o lab9-1.asm && l
d -m elf_i386 lab9-1.o -o lab9-1 && ./lab9-1
Введите N: 9
8
7
6
5
4
3
2
1
0
Ошибка сегментирования (стек памяти сброшен на диск)
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $
```

Рис. 4.4: Результат программы lab9-1.asm

5. Создал файл lab9-2.asm
6. Ввел текст программы в файл (рис. 4.5)

```

1  %include 'in_out.asm'
2
3  SECTION .data
4  msg1 db 'Введите N: ',0h
5
6  SECTION .bss
7  N: resb 10
8
9  SECTION .text
10 global _start
11
12 _start:
13     mov eax,msg1
14     call sprint
15
16     mov ecx, N
17     mov edx, 10
18     call sread
19
20     mov eax,N
21     call atoi
22     mov [N],eax
23
24     mov ecx,[N]
25
26 label:
27     mov [N],ecx
28     mov eax,[N]
29     call iprintf
30
31     loop label
32

```

Рис. 4.5: Текст программы lab9-2.asm

7. Также проверил работу программы с изменениями (рис. 4.6), программа обработала 3 аргумента.

```

gspolyakov@dk3n66:~/work/arch-pc/lab09
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-2.o lab9-2.asm && ld -m elf_i386 lab9-2.o -o lab9-2 && ./lab9-2 1 2 'X'
1
2
X
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $

```

Рис. 4.6: Результат программы lab9-2.asm

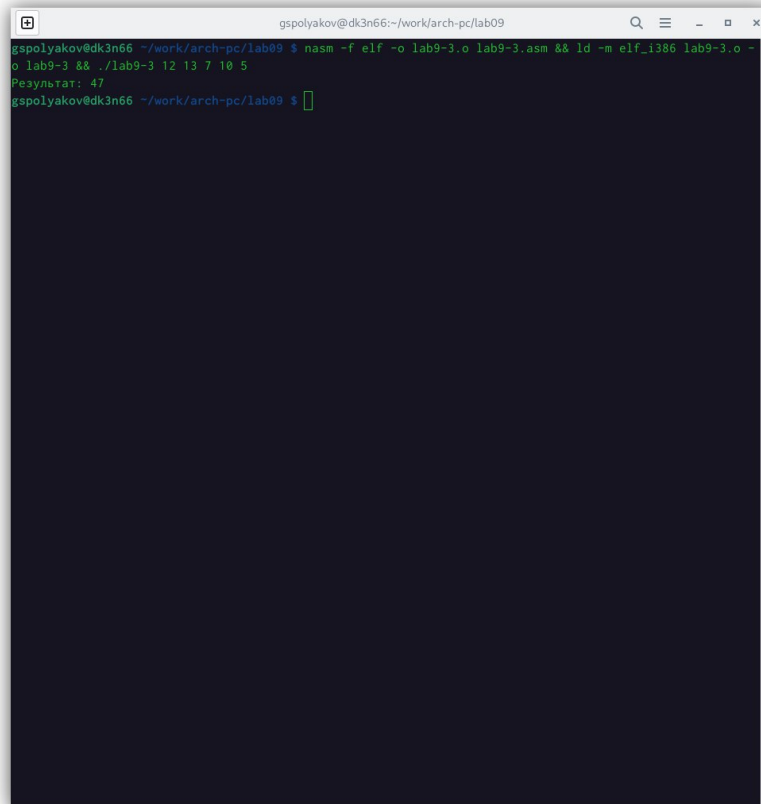
8. Создал файл lab9-3.asm

9. Ввел текст программы в файл (рис. 4.7)

```
1  %include 'in_out.asm'
2
3  SECTION .data
4  msg1 db "Введите N: ",0h
5
6  SECTION .bss
7  N: resb 10
8
9  SECTION .text
10 global _start
11
12 _start:
13     mov eax,msg1
14     call sprintf
15
16     mov ecx, N
17     mov edx, 10
18     call sread
19
20     mov eax,N
21     call atoi
22     mov [N],eax
23
24     mov ecx,[N]
25
26 label:
27     mov [N],ecx
28     mov eax,[N]
29     call iprintf
30
31     loop label
32
```

Рис. 4.7: Текст программы lab9-3.asm

10. Проверил работоспособность программы (рис. 4.8).

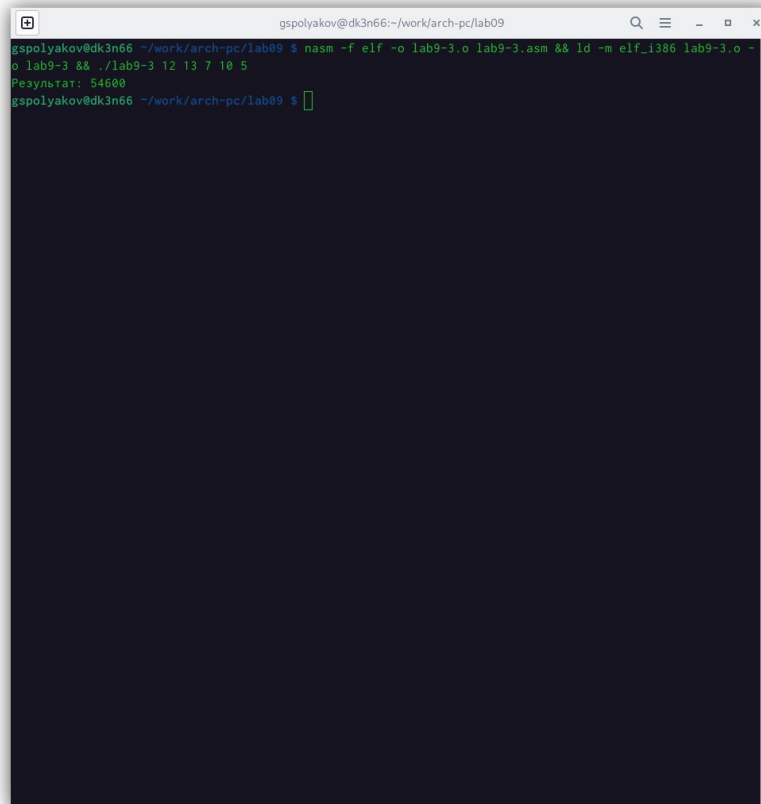
A terminal window with a dark background and light-colored text. The window title is "gspolyakov@dk3n66:~/work/arch-pc/lab09". The command prompt shows the user has run a command to compile and link a program. The output shows the program executed successfully, displaying the number 47.

```
gspolyakov@dk3n66:~/work/arch-pc/lab09 $ nasm -f elf -o lab9-3.o lab9-3.asm && ld -m elf_i386 lab9-3.o -o lab9-3 && ./lab9-3 12 13 7 10 5
Результат: 47
gspolyakov@dk3n66:~/work/arch-pc/lab09 $
```

Рис. 4.8: Результат программы lab9-3.asm

11. Также проверил работу программы с изменениями (рис. 4.9).



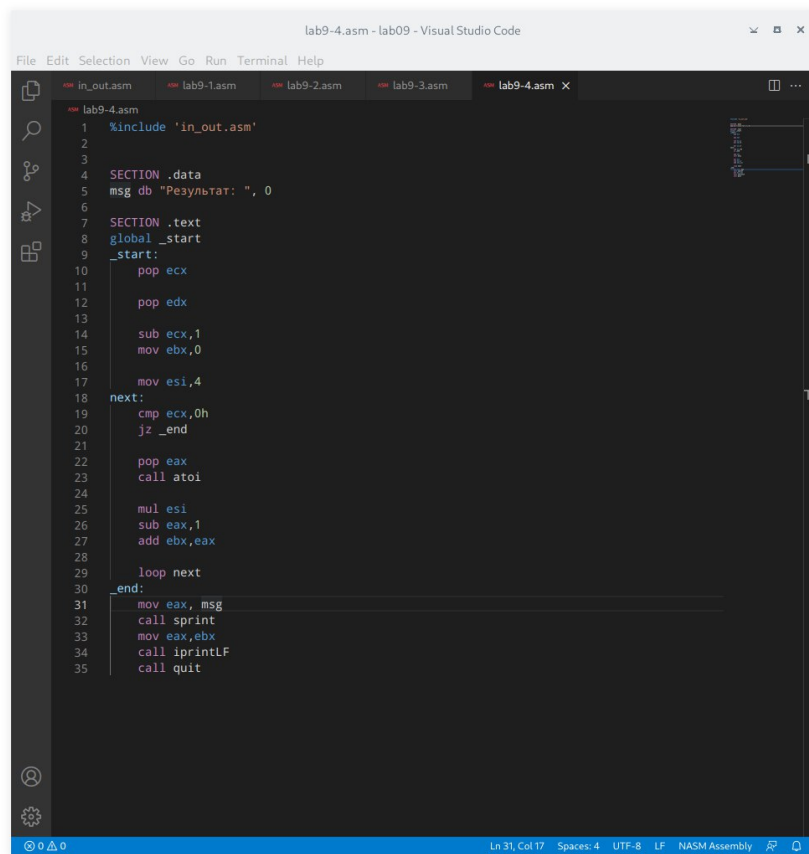
A terminal window with a dark background and light green text. The window title is 'gspolyakov@dk3n66:~/work/arch-pc/lab09'. The command entered is 'nasm -f elf -o lab9-3.o lab9-3.asm && ld -m elf\_i386 lab9-3.o -o lab9-3 && ./lab9-3 12 13 7 10 5'. The output is 'Результат: 54600'. The prompt is 'gspolyakov@dk3n66:~/work/arch-pc/lab09 \$' followed by a cursor.

```
gspolyakov@dk3n66:~/work/arch-pc/lab09 $ nasm -f elf -o lab9-3.o lab9-3.asm && ld -m elf_i386 lab9-3.o -o lab9-3 && ./lab9-3 12 13 7 10 5
Результат: 54600
gspolyakov@dk3n66:~/work/arch-pc/lab09 $
```

Рис. 4.9: Результат программы lab9-3.asm

## 4.1 Задание для самостоятельной работы

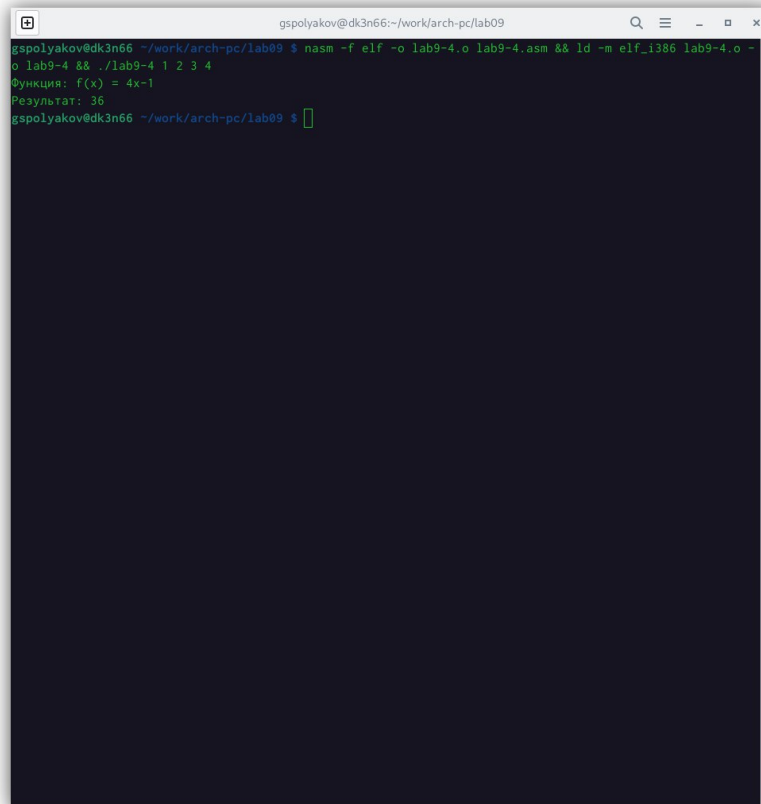
1. Создал файл lab9-4.asm
2. Ввел текст программы в файл (рис. 4.10)



```
lab9-4.asm
1 %include 'in_out.asm'
2
3
4 SECTION .data
5 msg db "Результат: ", 0
6
7 SECTION .text
8 global _start
9 _start:
10     pop ecx
11
12     pop edx
13
14     sub ecx,1
15     mov ebx,0
16
17     mov esi,4
18 next:
19     cmp ecx,0h
20     jz _end
21
22     pop eax
23     call atoi
24
25     mul esi
26     sub eax,1
27     add ebx,eax
28
29     loop next
30 _end:
31     mov eax, msg
32     call sprint
33     mov eax,ebx
34     call iprintf
35     call quit
```

Рис. 4.10: Текст программы lab9-4.asm

3. Проверил работоспособность программы (рис. 4.11)



```
gspolyakov@dk3n66:~/work/arch-pc/lab09
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf -o lab9-4.o lab9-4.asm && ld -m elf_i386 lab9-4.o -o lab9-4 && ./lab9-4 1 2 3 4
Функция: f(x) = 4x-1
Результат: 36
gspolyakov@dk3n66 ~/work/arch-pc/lab09 $
```

Рис. 4.11: Результат программы lab9-4.asm

## 5 Выводы

Делая лабораторную работу, я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки на языке Assembler.

## **Список литературы**