

Отчёт по лабораторной работе №8

Программирование цикла. Обработка аргументов командной строки

Лань Цянын

1. Цель работы

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

2. Порядок выполнения лабораторной работы

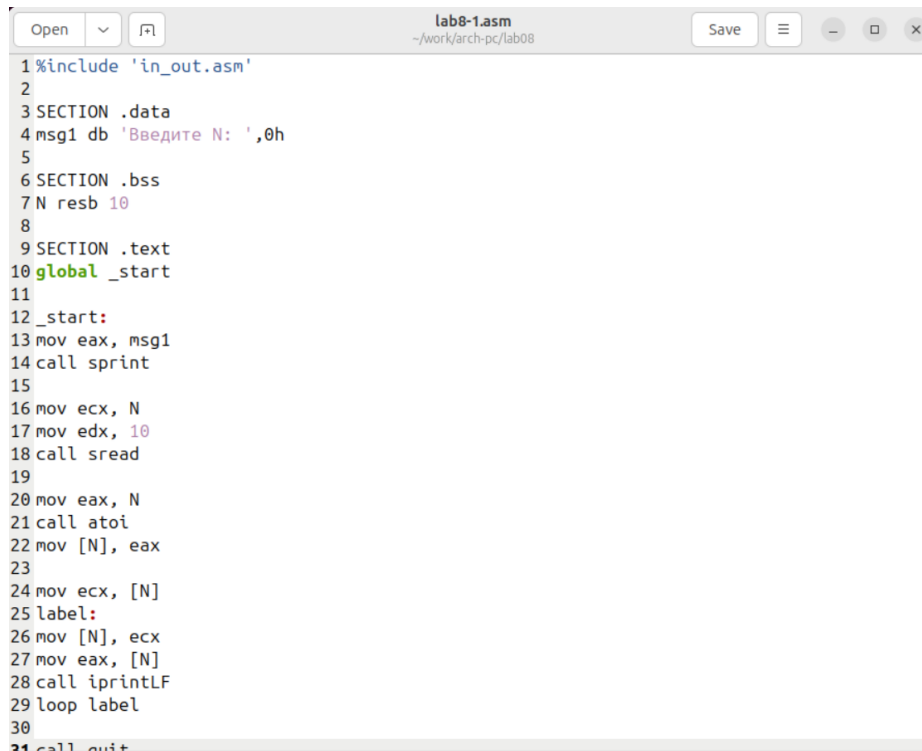
2.1. Реализация циклов в NASM

Для выполнения лабораторной работы №8 создан каталог Lab08, внутри него создан файл lab8-1.asm в соответствии с инструкциями методического пособия. (рис. Рисунок 1).

```
clanj1@clanj1:~/work/arch-pc/lab07$ mkdir ~/work/arch-pc/lab08
clanj1@clanj1:~/work/arch-pc/lab07$ cd ~/work/arch-pc/lab08
clanj1@clanj1:~/work/arch-pc/lab08$ touch lab8-1.asm
clanj1@clanj1:~/work/arch-pc/lab08$ gedit lab8-1.asm
```

Рисунок 1: Создание каталога и файла для лабораторной работы №8

Программа lab8-1.asm была написана в точном соответствии с Листингом 8.1 из методического пособия. (рис. Рисунок 2)



```
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 label:
26 mov [N], ecx
27 mov eax, [N]
28 call iprintLF
29 loop label
30
31 call quit
```

Рисунок 2: Исходный код программы lab8-1.asm (Листинг 8.1)

Программа запрашивает у пользователя число N, преобразует его из строки в целое число и затем использует регистр ecx как счётчик для цикла, выводя последовательно значения от N до 1 с помощью инструкции loop.

Программа lab8-1.asm была откомпилирована и запущена. Для её работы в каталог lab08 был предварительно скопирован необходимый файл in_out.asm. (рис. Рисунок 3)



```
clanji1@clanji1:~/work/arch-pc/lab08$ cp /media/sf_VB_Share/in_out.asm ~/work/arch-
pc/lab08
clanji1@clanji1:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
clanji1@clanji1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
clanji1@clanji1:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
```

Рисунок 3: Компиляция, сборка и запуск программы lab8-1

Программа успешно выполняется, используя инструкцию `loop` для организации цикла, который выводит значения от введённого числа 5 до 1.

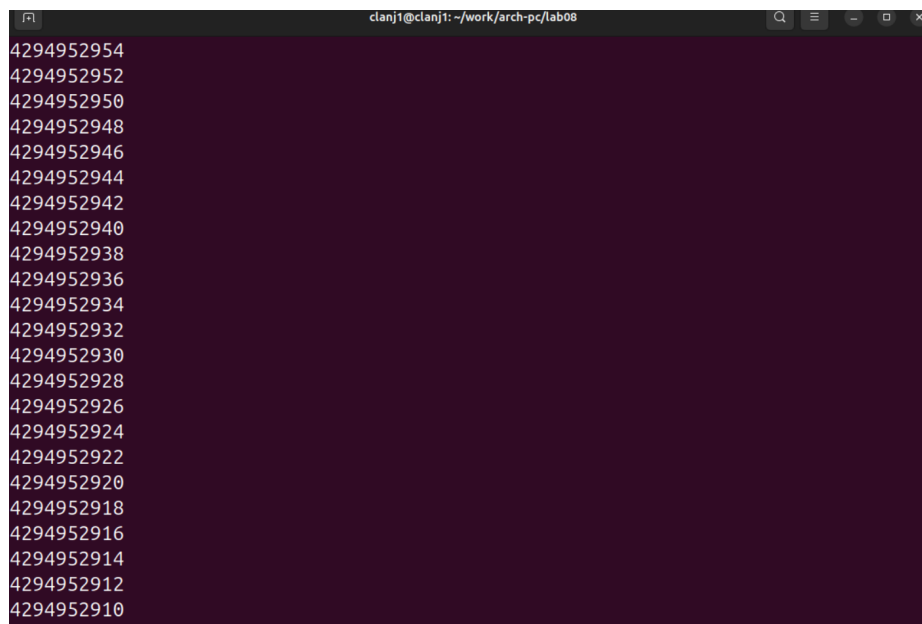
Исходный код программы `lab8-1.asm` был модифицирован в соответствии с заданием: в тело цикла `label` добавлена инструкция `sub ecx,1`, которая явно уменьшает значение регистра-счётчика на каждой итерации. (рис. Рисунок 4)



```
4
5
6 SECTION .data
7 msg1 db 'Введите N: ',0h
8
9 SECTION .bss
10 N resb 10
11
12 SECTION .text
13 global _start
14 _start:
15 mov eax, msg1
16 call sprint
17
18 mov ecx, N
19 mov edx, 10
20 call sread
21
22 mov eax, N
23 call atoi
24 mov [N], eax
25
26 mov ecx, [N]
27 label:
28 sub ecx,1
29 mov [N],ecx
30 mov eax,[N]
31 call iprintLF
32 loop label
33
34 call quit
```

Рисунок 4: Модифицированный код программы `lab8-1.asm`

Программа, модифицированная добавлением `sub ecx,1` в тело цикла, при выполнении вошла в бесконечный цикл. (рис. Рисунок 5)



```
clanji@clanji1: ~/work/arch-pc/lab08
4294952954
4294952952
4294952950
4294952948
4294952946
4294952944
4294952942
4294952940
4294952938
4294952936
4294952934
4294952932
4294952930
4294952928
4294952926
4294952924
4294952922
4294952920
4294952918
4294952916
4294952914
4294952912
4294952910
```

Рисунок 5: Бесконечный цикл в модифицированной программе

Это происходит потому, что инструкция `loop` также уменьшает `ecx` на 1, и их совместное действие может привести к тому, что `ecx` никогда не достигнет нуля (например, перескочив с 1 на -1), в результате чего условие завершения цикла не выполняется.

Ответы на вопросы □

1. Какие значения принимает регистр `ecx` в цикле?

Значение регистра `ecx` уменьшается дважды за одну итерацию цикла: один раз инструкцией `sub`, и второй раз — инструкцией `loop`. В результате выводимая последовательность чисел уменьшается на 2 (например: $N-1$, $N-3$, $N-5$ и т.д.). При нечётном значении N регистр `ecx` принимает отрицательные значения, что приводит к заикливанию программы.

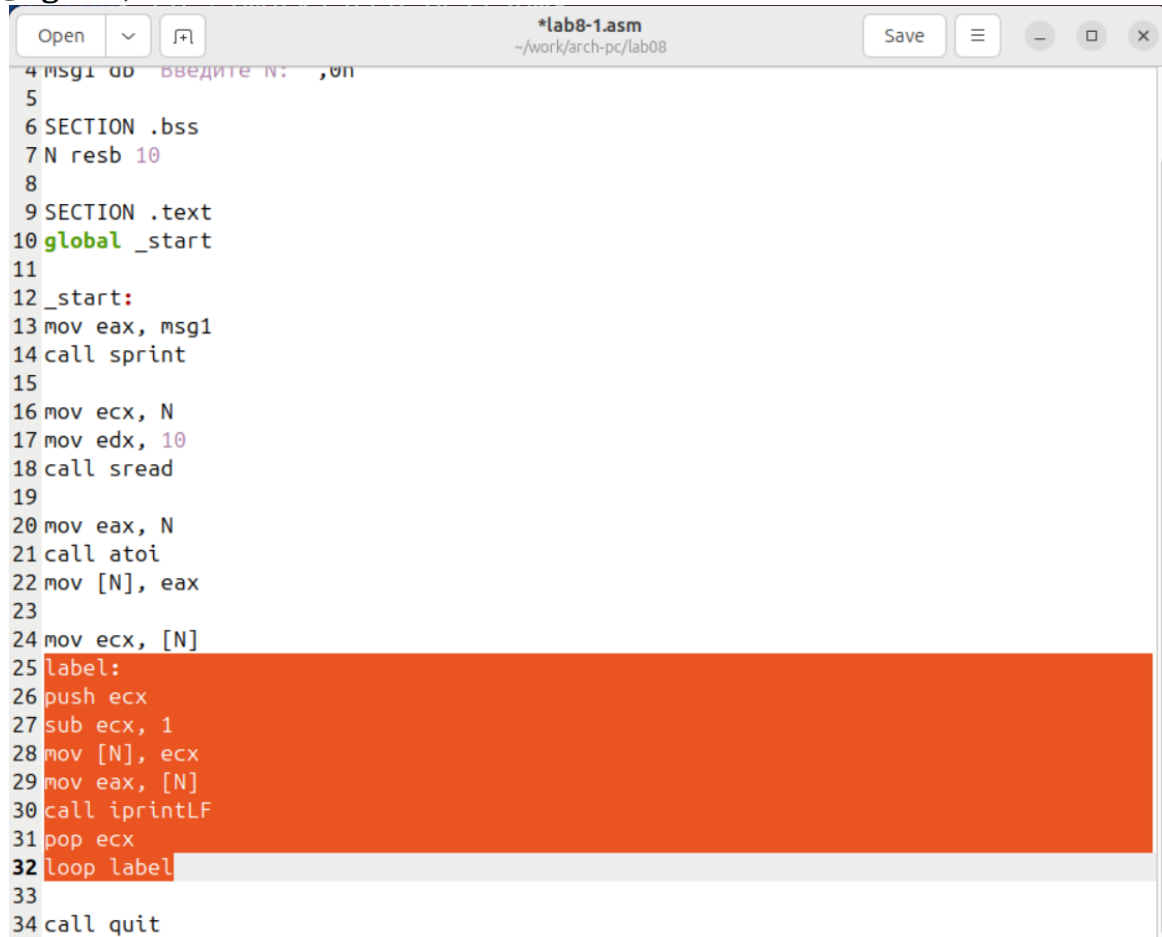
2. Соответствует ли число проходов цикла значению N ?

Нет. Поскольку регистр `ecx` уменьшается два раза за одну итерацию, фактическое число проходов цикла примерно равно $N/2$ (при чётном N). При нечётном значении N программа не завершается и переходит в

бесконечный цикл.

Программа lab8-1.asm была модифицирована с использованием стека для сохранения и восстановления значения счётчика цикла ecx, что позволяет корректно работать инструкции loop при изменении ecx в теле цикла(рис.

?@fig-006)



```
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 label:
26 push ecx
27 sub ecx, 1
28 mov [N], ecx
29 mov eax, [N]
30 call iprintLF
31 pop ecx
32 loop label
33
34 call quit
```

{#fig-

006 width=80%

Программа, модифицированная с использованием стека для сохранения значения счётчика ecx, была успешно откомпилирована и запущена(рис. Рисунок 6)

```
clanj1@clanj1:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
clanj1@clanj1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
clanj1@clanj1:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
```

Рисунок 6: Выполнение программы с использованием стека

Использование `push esx` и `pop esx` обеспечивает корректную работу инструкции `loop`, гарантируя выполнение цикла ровно N раз, что подтверждается выводом последовательных чисел (например, при $N=5$ выводится 4, 3, 2, 1, 0)

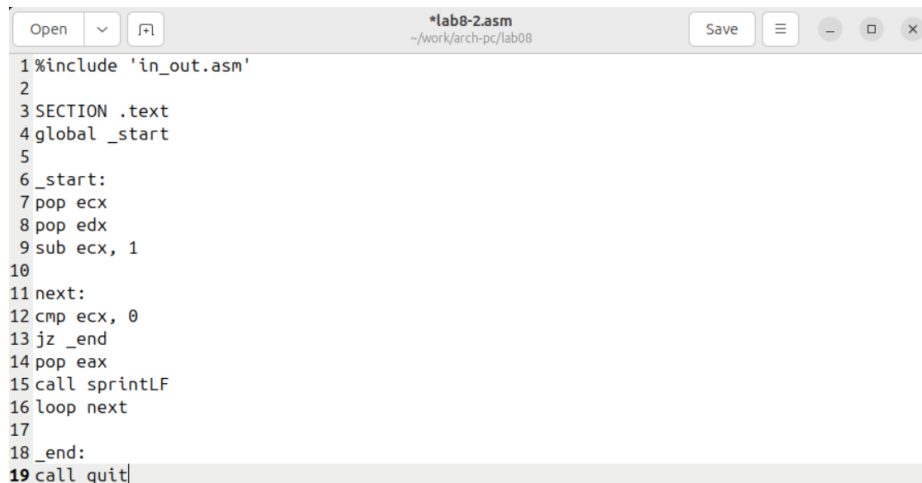
Ответ на вопрос задания:

Соответствует ли число проходов цикла значению N ?

Да, полностью соответствует. Использование стека для сохранения состояния регистра `esx` устраняет проблему двойного уменьшения счётчика, и цикл выполняется ровно N раз, как и задумано изначально.

2.2. Обработка аргументов командной строки

Программа `lab8-2.asm` для обработки аргументов командной строки была написана в точном соответствии с Листингом 8.2(рис. Рисунок 7)



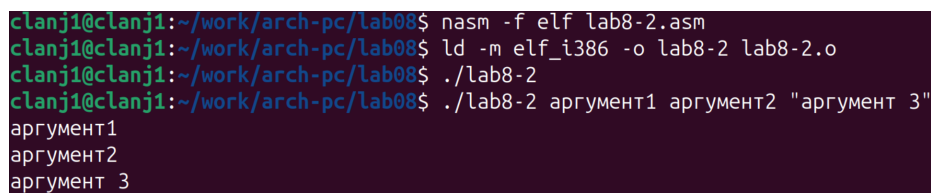
```

1 %include 'in_out.asm'
2
3 SECTION .text
4 global _start
5
6 _start:
7 pop ecx
8 pop edx
9 sub ecx, 1
10
11 next:
12 cmp ecx, 0
13 jz _end
14 pop eax
15 call sprintf
16 loop next
17
18 _end:
19 call quit

```

Рисунок 7: Исходный код программы lab8-2.asm

Программа lab8-2.asm была успешно откомпилирована, собрана и протестирована с передачей аргументов командной строки (рис. Рисунок 8)



```

clanji@clanji:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
clanji@clanji:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
clanji@clanji:~/work/arch-pc/lab08$ ./lab8-2
clanji@clanji:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент2 "аргумент 3"
аргумент1
аргумент2
аргумент 3

```

Рисунок 8: Выполнение программы lab8-2 с аргументами командной строки

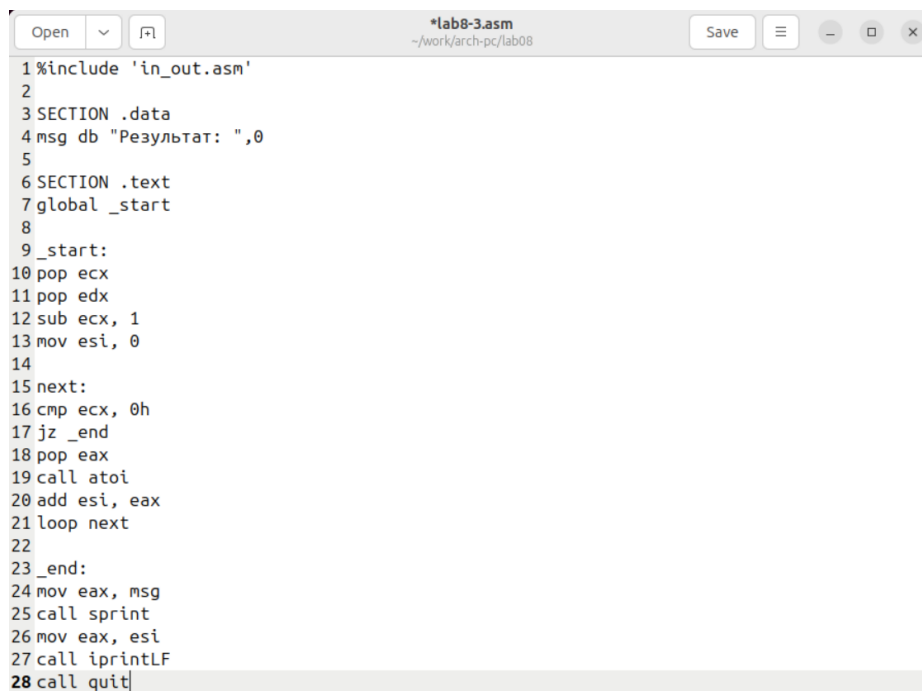
Программа корректно обработала и вывела все три переданных аргумента, что подтверждает правильность реализации извлечения аргументов из стека и работы циклической обработки

Ответ на вопрос задания:

Сколько аргументов было обработано программой?

Программа обработала **3 аргумента**, что соответствует количеству аргументов, переданных в командной строке (после имени программы).

Программа lab8-3.asm, реализующая вычисление суммы числовых аргументов командной строки, была написана на основе Листинга 8.3 (рис. Рисунок 9)



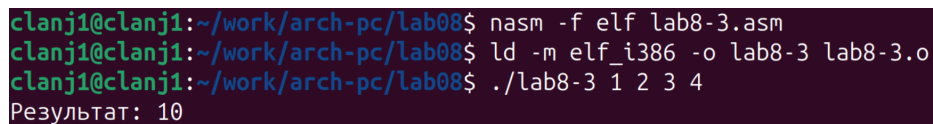
```

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

```

Рисунок 9: Исходный код программы lab8-3.asm

Программа lab8-3.asm была откомпилирована и запущена с передачей числовых аргументов для проверки работы алгоритма суммирования (рис. Рисунок 10)



```

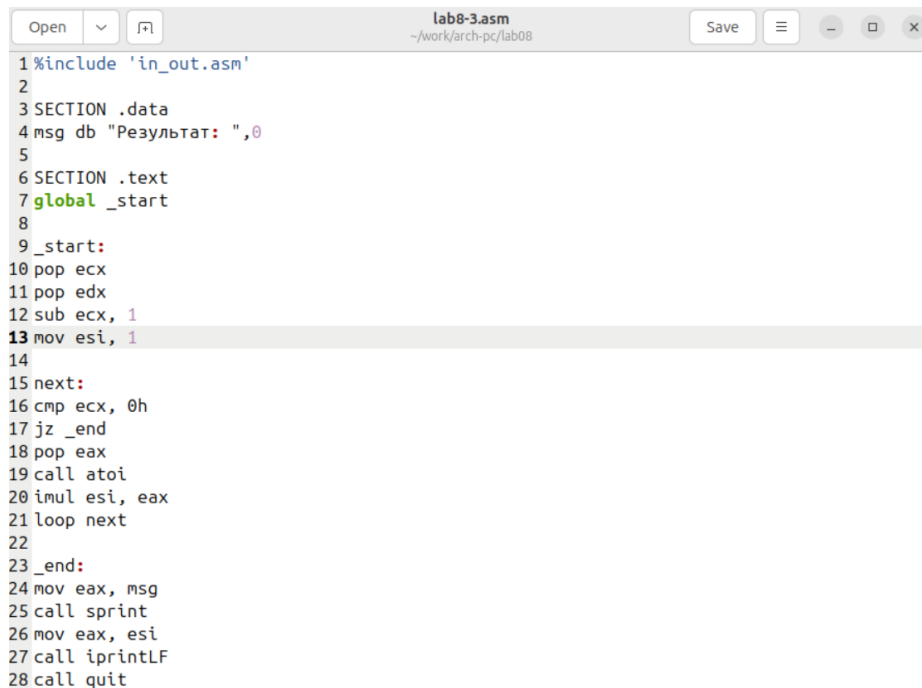
clanj1@clanj1:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
clanj1@clanj1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
clanj1@clanj1:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 10

```

Рисунок 10: Компиляция, сборка и запуск программы lab8-3

Программа успешно обработала аргументы 1 2 3 4. Ожидаемый результат суммирования — 10. Это подтверждает корректную работу всех этапов: извлечения количества аргументов, их преобразования из строк в числа (atoi), накопления суммы в регистре esi и вывода итога

Программа lab8-3.asm была модифицирована для вычисления **произведения** числовых аргументов командной строки вместо их суммы (рис. Рисунок 11)



```

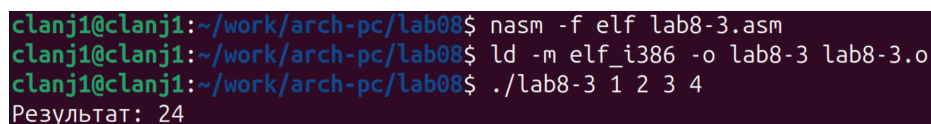
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10 pop ecx
11 pop edx
12 sub ecx, 1
13 mov esi, 1
14
15 next:
16 cmp ecx, 0h
17 jz _end
18 pop eax
19 call atoi
20 imul esi, eax
21 loop next
22
23 _end:
24 mov eax, msg
25 call sprint
26 mov eax, esi
27 call iprintLF
28 call quit

```

Рисунок 11: Модифицированный код программы lab8-3.asm для вычисления произведения

Регистр `esi` (в коде `est`) инициализируется значением **1** вместо `0`, так как произведение любого числа на 1 равно самому числу. Инструкция сложения `add est, eax` заменена на инструкцию умножения **`imul est, eax`**

Модифицированная программа `lab8-3.asm` была успешно откомпилирована и запущена для проверки вычисления произведения аргументов (рис. Рисунок 12)



```

clanj1@clanj1:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
clanj1@clanj1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
clanj1@clanj1:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24

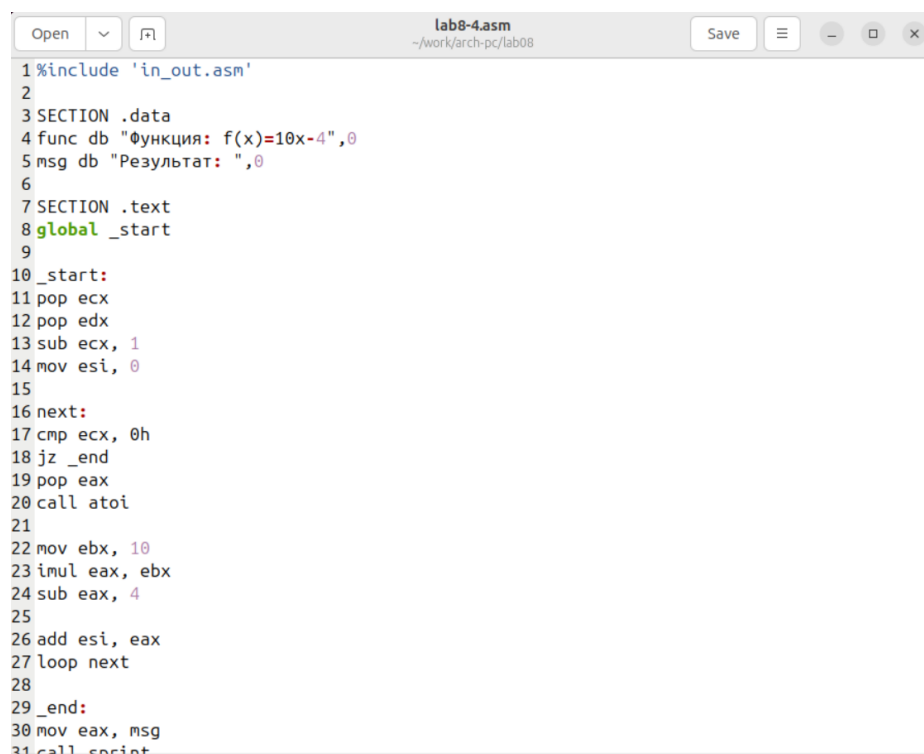
```

Рисунок 12: Выполнение программы для вычисления произведения

Программа корректно вычислила и вывела результат 24, что соответствует произведению чисел 1, 2, 3 и 4.

3. Задание для самостоятельной работы

Программа lab8-4.asm была написана для вычисления суммы значений функции $f(x) = 10x - 4$ (вариант 9) для набора чисел x , переданных в качестве аргументов командной строки (рис. Рисунок 1)



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 func db "Функция: f(x)=10x-4",0
5 msg db "Результат: ",0
6
7 SECTION .text
8 global _start
9
10 _start:
11 pop ecx
12 pop edx
13 sub ecx, 1
14 mov esi, 0
15
16 next:
17 cmp ecx, 0h
18 jz _end
19 pop eax
20 call atoi
21
22 mov ebx, 10
23 imul eax, ebx
24 sub eax, 4
25
26 add esi, eax
27 loop next
28
29 _end:
30 mov eax, msg
31 call _printf
```

Рисунок 1: Исходный код программы lab8-4.asm

Программа lab8-4.asm была успешно откомпилирована и протестирована

на наборе значений 1 2 3 4(рис. Рисунок 2)

```
clanj1@clanj1:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
clanj1@clanj1:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
clanj1@clanj1:~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Результат: Функция:  $f(x)=10x-4$ 
84
```

Рисунок 2: Тестирование программы lab8-4.asm

Программа вывела ожидаемый результат: 84□ Это подтверждает, что она корректно вычисляет значение функции $f(x) = 10x - 4$ для каждого переданного аргумента x , суммирует эти значения и выводит итог

4 Выводы

В ходе лабораторной работы № 8 были изучены инструкции организации циклов в NASM и способы обработки аргументов командной строки. Рассмотрено влияние изменения регистра `ecx` на корректность работы инструкции `loop` и показана необходимость использования стека для сохранения счётчика цикла. Получены практические навыки работы с циклами, стеком и аргументами командной строки, что соответствует цели лабораторной работы.