

Отчёта по лабораторной работе 9

Программирование цикла. Обработка аргументов командной строки.

Плето плето мбамби

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Файл lab9-1.asm	8
4.2	Работа программы lab9-1.asm	9
4.3	Файл lab9-1.asm	10
4.4	Работа программы lab9-1.asm	10
4.5	Файл lab9-1.asm	11
4.6	Работа программы lab9-1.asm	11
4.7	Файл lab9-2.asm	12
4.8	Работа программы lab9-2.asm	13
4.9	Файл lab9-3.asm	14
4.10	Работа программы lab9-3.asm	14
4.11	Файл lab9-3.asm	15
4.12	Работа программы lab9-3.asm	16
4.13	Файл lab9-4.asm	17
4.14	Работа программы lab9-4.asm	17

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

2 Задание

1. Изучите примеры программ
2. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x .
3. Загрузите файлы на GitHub.

3 Теоретическое введение

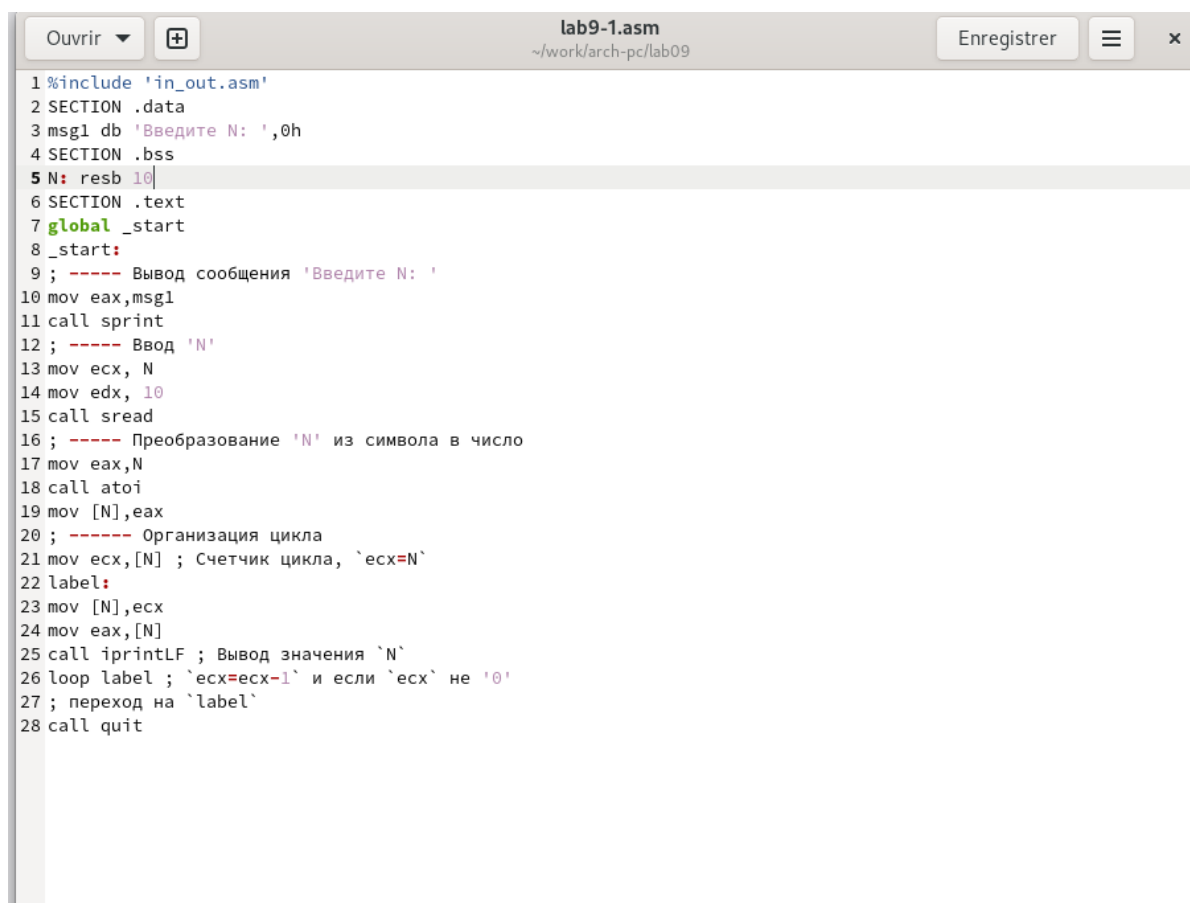
Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией 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.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loor. Инструкция loor выполняется в два этапа. Сначала из регистра esx 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.

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:

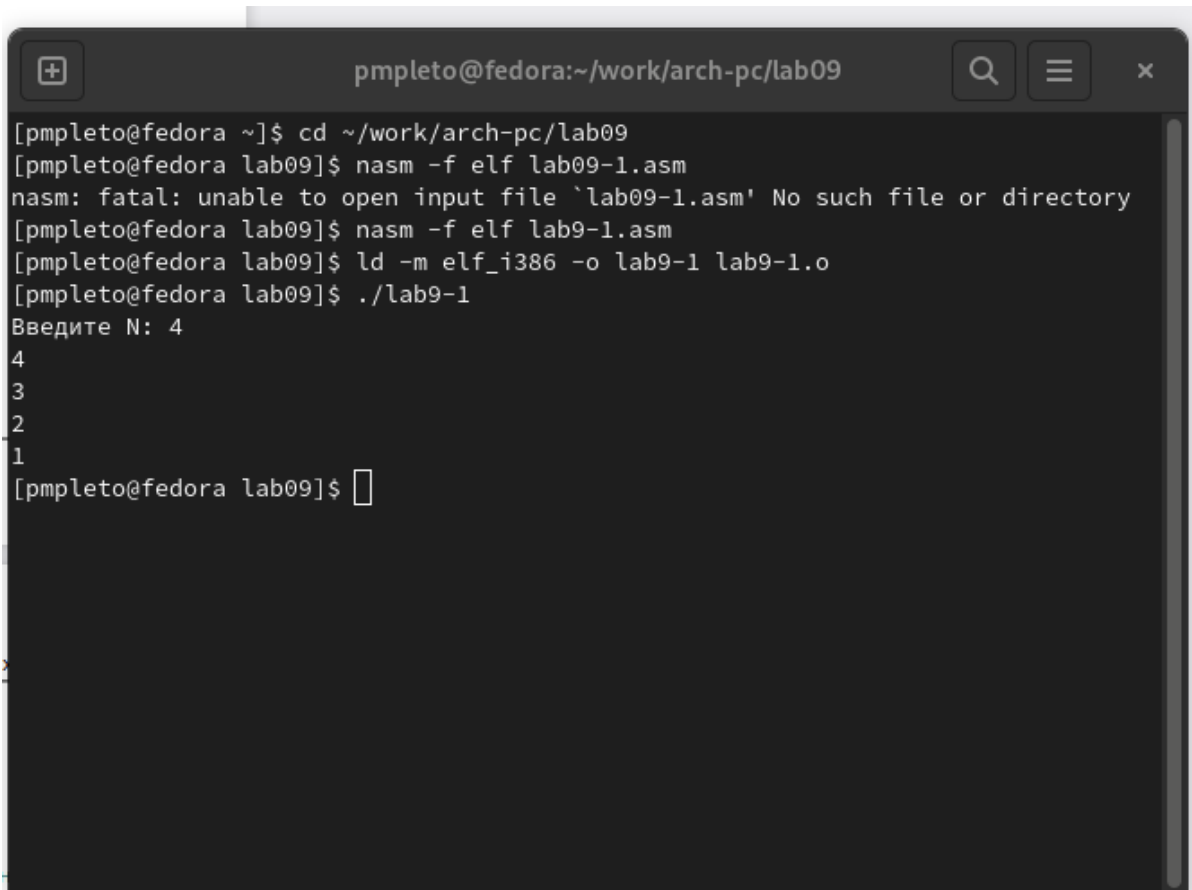
4 Выполнение лабораторной работы

1. Создайте каталог для программ лабораторной работы № 9, перейдите в него и создайте файл lab9-1.asm
2. Введите в файл lab9-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. (рис. 4.1, 4.2)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения `N`
26 loop label ; `ecx=ecx-1` и если `ecx` не `0`
27 ; переход на `label`
28 call quit
```

Рис. 4.1: Файл lab9-1.asm

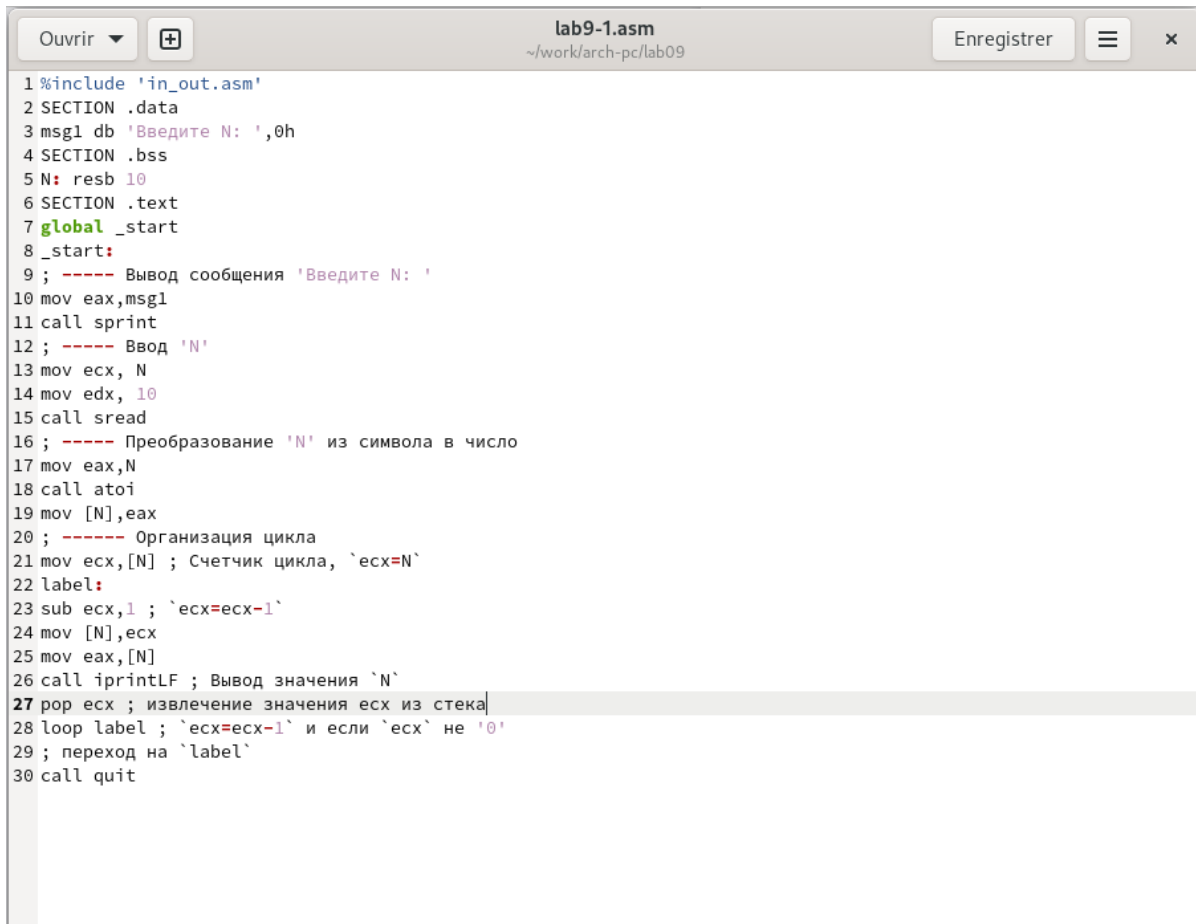


```
pmpeto@fedora:~/work/arch-pc/lab09
[pmpeto@fedora ~]$ cd ~/work/arch-pc/lab09
[pmpeto@fedora lab09]$ nasm -f elf lab09-1.asm
nasm: fatal: unable to open input file `lab09-1.asm' No such file or directory
[pmpeto@fedora lab09]$ nasm -f elf lab9-1.asm
[pmpeto@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[pmpeto@fedora lab09]$ ./lab9-1
Введите N: 4
4
3
2
1
[pmpeto@fedora lab09]$
```

Рис. 4.2: Работа программы lab9-1.asm

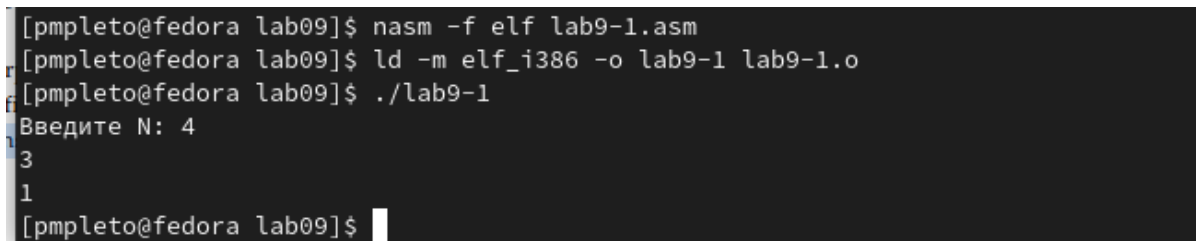
3. Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра `ecx` в цикле: Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению `N`, введенному с клавиатуры? (рис. 4.3, 4.4)

Программа запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N`.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, `ecx=N`
22 label:
23 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF ; Вывод значения `N`
27 pop ecx ; извлечение значения ecx из стека
28 loop label ; `ecx=ecx-1` и если `ecx` не '0'
29 ; переход на `label`
30 call quit
```

Рис. 4.3: Файл lab9-1.asm



```
[pmp1eto@fedora lab09]$ nasm -f elf lab9-1.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[pmp1eto@fedora lab09]$ ./lab9-1
Введите N: 4
3
1
[pmp1eto@fedora lab09]$
```

Рис. 4.4: Работа программы lab9-1.asm

4. Для использования регистра ecx в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop. Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов

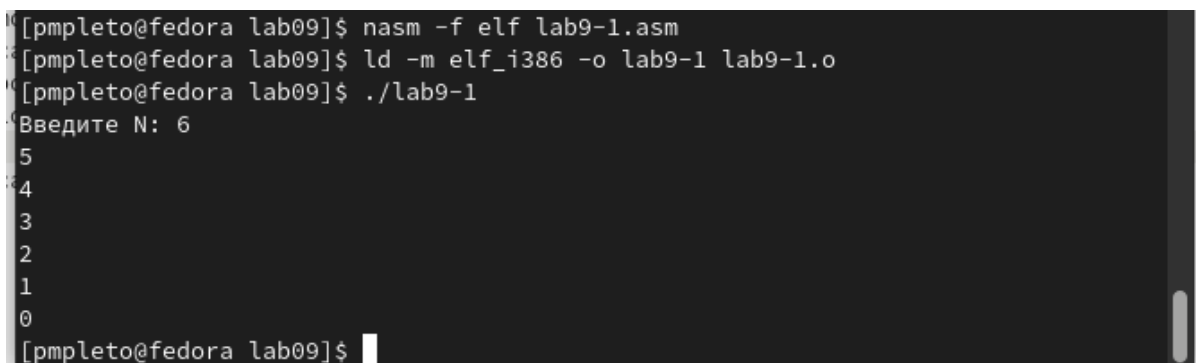
цикла значению N введенному с клавиатуры? (рис. 4.5, 4.6)

Программа выводит числа от N-1 до 0, число проходов цикла соответствует N.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 push ecx ; добавление значения ecx в стек
24 sub ecx,1
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx ; извлечение значения ecx из стека
29 loop label
30 ; переход на 'label'
31 call quit
```

Рис. 4.5: Файл lab9-1.asm

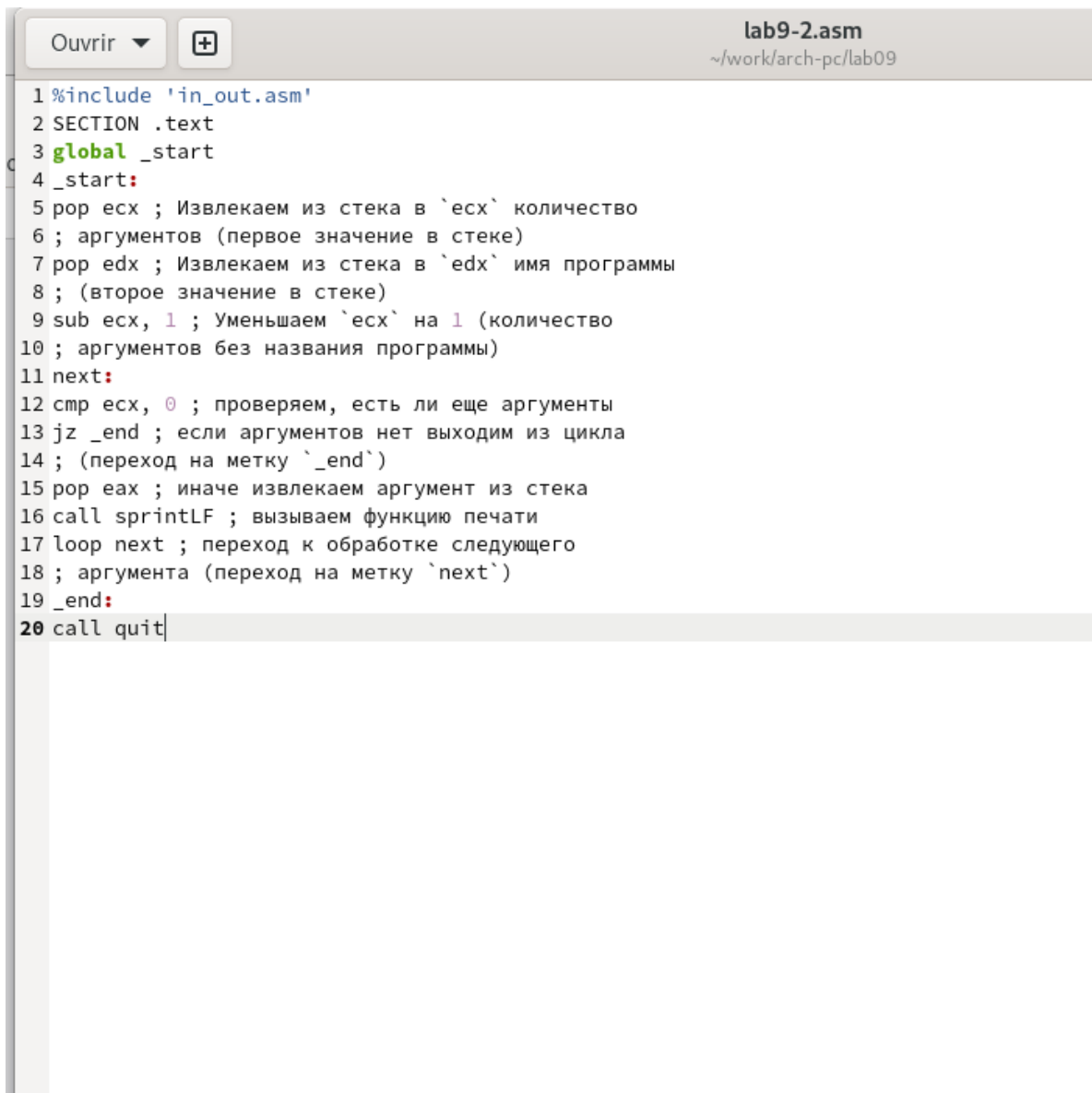


```
[pmp1eto@fedora lab09]$ nasm -f elf lab9-1.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[pmp1eto@fedora lab09]$ ./lab9-1
Введите N: 6
5
4
3
2
1
0
[pmp1eto@fedora lab09]$
```

Рис. 4.6: Работа программы lab9-1.asm

5. Создайте файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и введите в него текст программы из листинга 9.2. Создайте исполняемый файл и запустите его, указав аргументы. (рис. 4.7, 4.8) Сколько аргументов было обработано программой?

Программа обработала 5 аргументов.



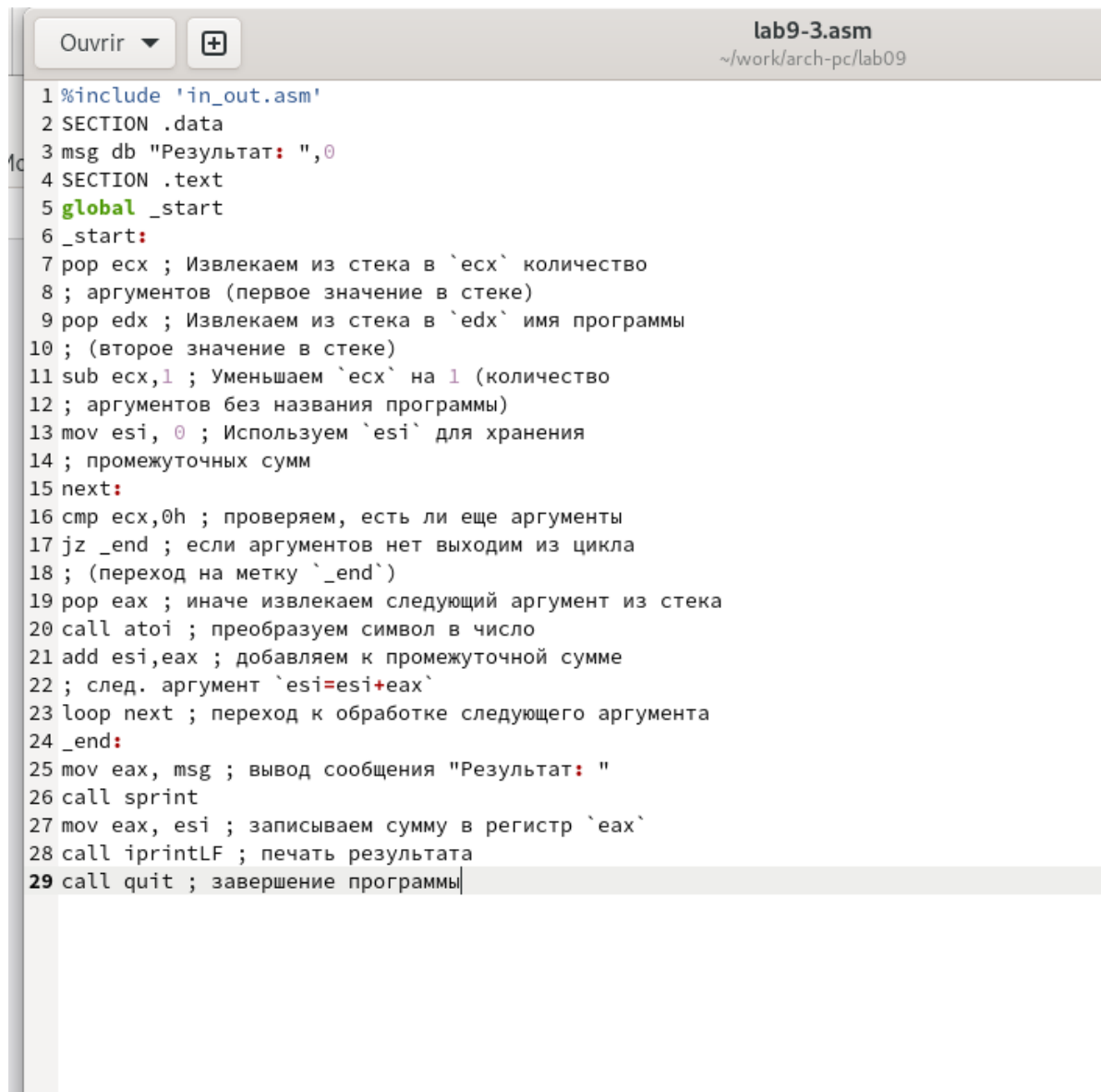
```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 4.7: Файл lab9-2.asm

```
аргумент 3
[pmp1eto@fedora lab09]$ nasm -f elf lab9-2.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[pmp1eto@fedora lab09]$ ./lab9-2 аргумент 1 аргумент 2 'аргумент 3'
аргумент
1
аргумент
2
аргумент 3
[pmp1eto@fedora lab09]$
```

Рис. 4.8: Работа программы lab9-2.asm

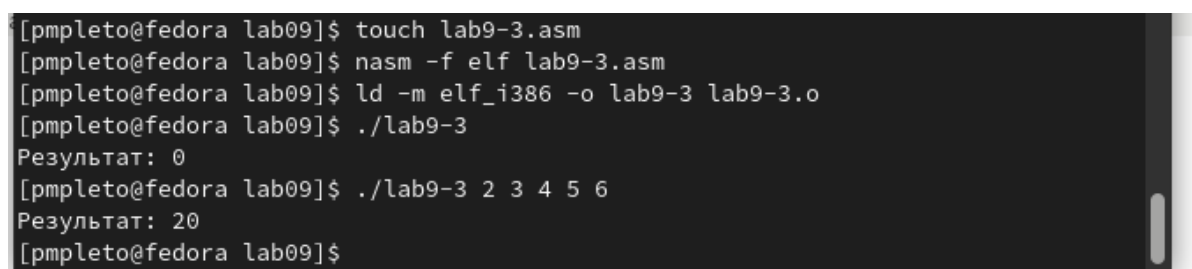
6. Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. 4.9, 4.10)



```
lab9-3.asm
~/work/arch-pc/lab09

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

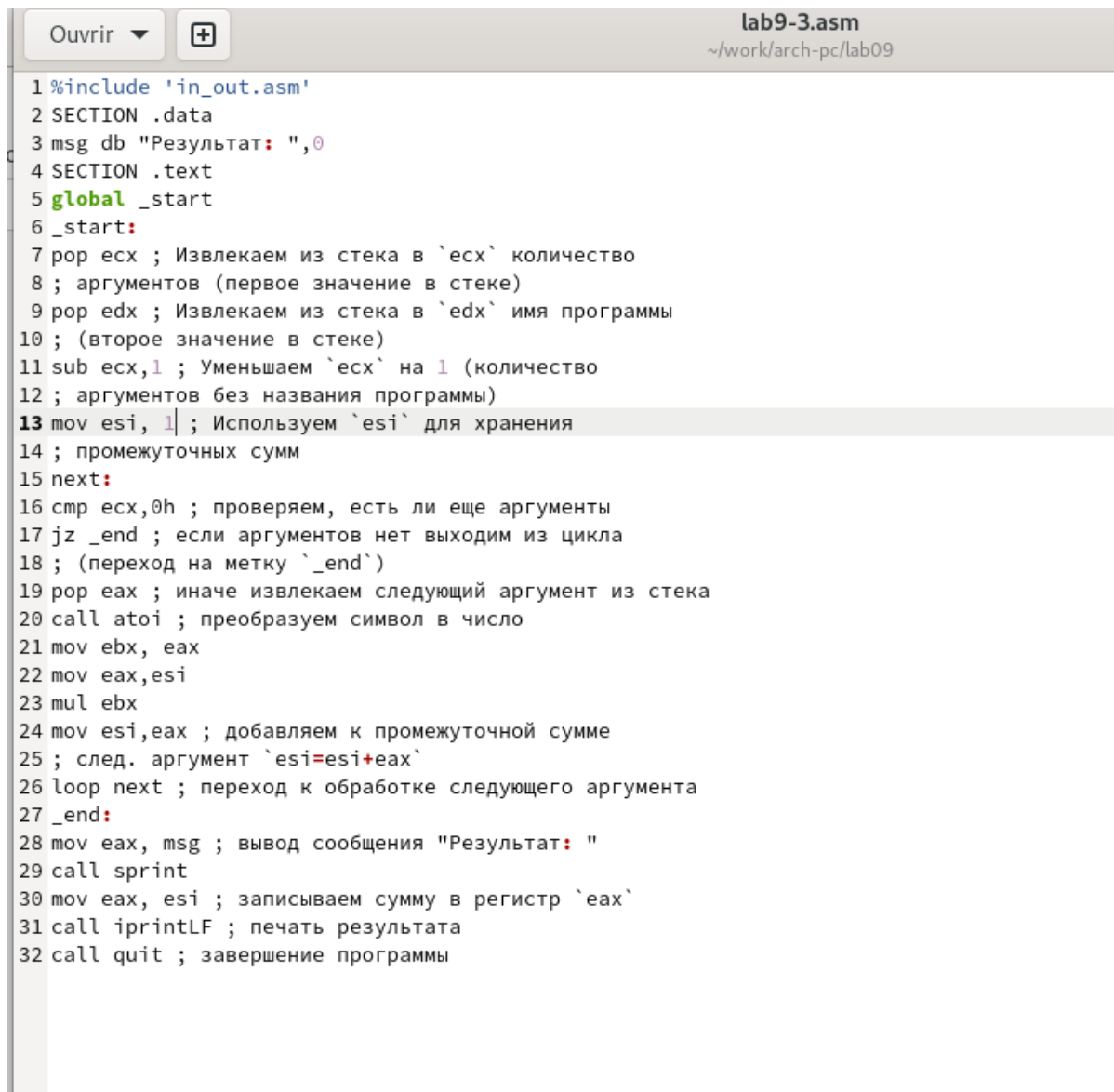
Рис. 4.9: Файл lab9-3.asm



```
[pmp1eto@fedora lab09]$ touch lab9-3.asm
[pmp1eto@fedora lab09]$ nasm -f elf lab9-3.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[pmp1eto@fedora lab09]$ ./lab9-3
Результат: 0
[pmp1eto@fedora lab09]$ ./lab9-3 2 3 4 5 6
Результат: 20
[pmp1eto@fedora lab09]$
```

Рис. 4.10: Работа программы lab9-3.asm

7. Измените текст программы из листинга 9.3 для вычисления произведения аргументов командной строки. (рис. 4.11, 4.12)



```
lab9-3.asm
~/work/arch-pc/lab09

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi,1 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mov ebx, eax
22 mov eax,esi
23 mul ebx
24 mov esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент `esi=esi+eax`
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax, msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax, esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программы
```

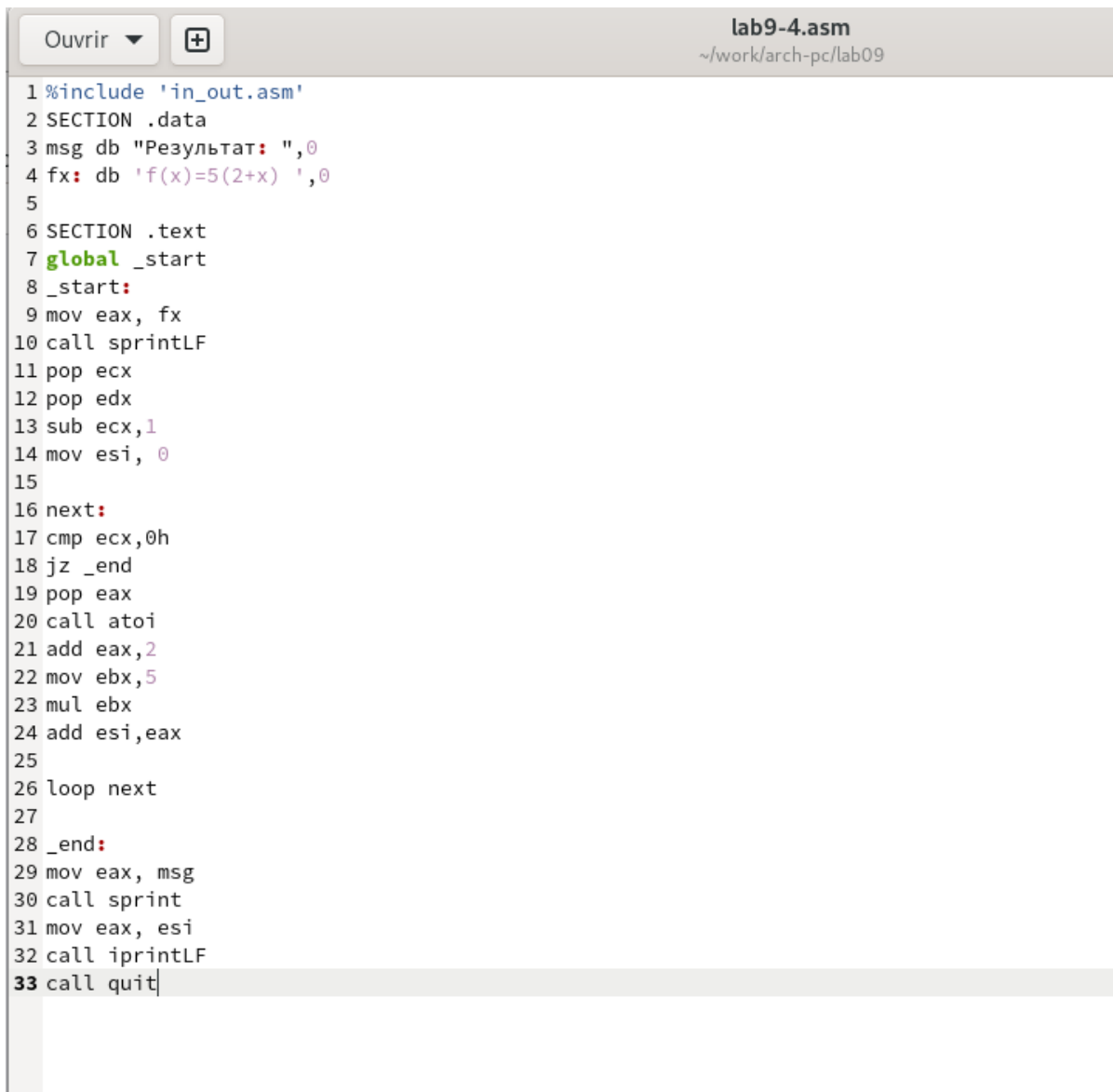
Рис. 4.11: Файл lab9-3.asm

```
[pmp1eto@fedora lab09]$ nasm -f elf lab9-3.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[pmp1eto@fedora lab09]$ ./lab9-3
Результат: 1
[pmp1eto@fedora lab09]$ ./lab9-3 2 3 4 5 6
Результат: 720
[pmp1eto@fedora lab09]$
```

Рис. 4.12: Работа программы lab9-3.asm

8. Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x . (рис. 4.13, 4.14)

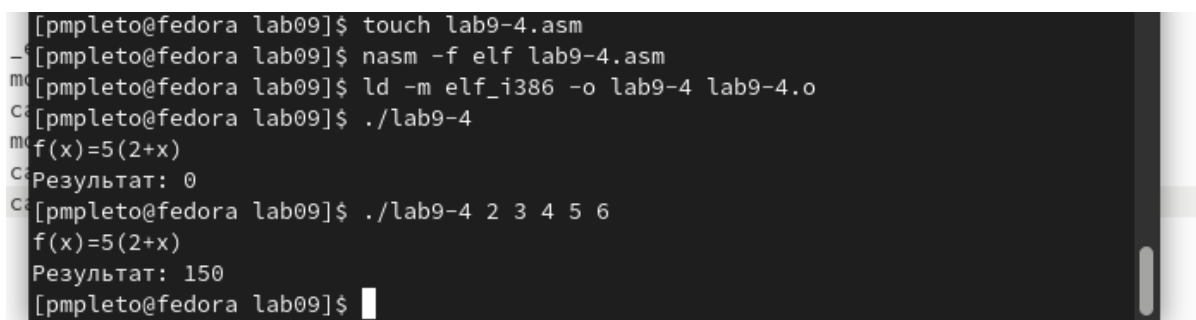
для варианта 10 $f(x) = 5(2+x)$



```
lab9-4.asm
~/work/arch-pc/lab09

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)=5(2+x) ',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
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 add eax,2
22 mov ebx,5
23 mul ebx
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprint
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рис. 4.13: Файл lab9-4.asm



```
[pmp1eto@fedora lab09]$ touch lab9-4.asm
[pmp1eto@fedora lab09]$ nasm -f elf lab9-4.asm
[pmp1eto@fedora lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[pmp1eto@fedora lab09]$ ./lab9-4
f(x)=5(2+x)
Результат: 0
[pmp1eto@fedora lab09]$ ./lab9-4 2 3 4 5 6
f(x)=5(2+x)
Результат: 150
[pmp1eto@fedora lab09]$
```

Рис. 4.14: Работа программы lab9-4.asm

5 Выводы

Наконец, мы научились писать программы, используя циклы и манипулируя аргументами командной строки. Кроме того, мы научились работать с аргументами командной строки

Список литературы

1. MASM, TASM, FASM, NASM под Windows и Linux