

Отчёт по лабораторной работе № 9

Архитектура компьютера

Старцева Алина Сергеевна

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
3.1	Реализация циклов в NASM	6
3.2	Обработка аргументов командной строки	10
3.3	Задание для самостоятельной работы	13
4	Выводы	15

Список иллюстраций

3.1	lab9-1.asm	6
3.2	Текст программы	7
3.3	Исполняемый файл	7
3.4	Текст программы	8
3.5	Исполняемый файл	8
3.6	Значения в цикле	9
3.7	Текст программы	9
3.8	Исполняемый файл	10
3.9	lab9-2.asm	11
3.10	Текст программы	11
3.11	Исполняемый файл	11
3.12	lab9-3.asm	12
3.13	Текст программы	12
3.14	Исполняемый файл	12
3.15	Текст программы	13
3.16	Исполняемый файл	13
3.17	Текст программы	14
3.18	Исполняемый файл	14

1 Цель работы

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

2 Задание

1. Реализовать циклы в NASM
2. Выполнить обработку аргументов командной строки
3. Выполнить задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация циклов в NASM

Создали каталог для программ лабораторной работы № 9, перейшли в него и создали файл lab9-1.asm: (рис. 3.1)

```
[asstarceval@fedora ~]$ mkdir ~/work/arch-pc/lab09  
[asstarceval@fedora ~]$ cd ~/work/arch-pc/lab09  
[asstarceval@fedora lab09]$ touch lab9-1.asm
```

Рис. 3.1: lab9-1.asm

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрели программу, которая выводит значение регистра `ecx`. Внимательно изучили текст программы (Листинг 9.1).

Ввели в файл lab9-1.asm текст программы из листинга 9.1. (рис. 3.2) Создали исполняемый файл и проверили его работу. (рис. 3.3)

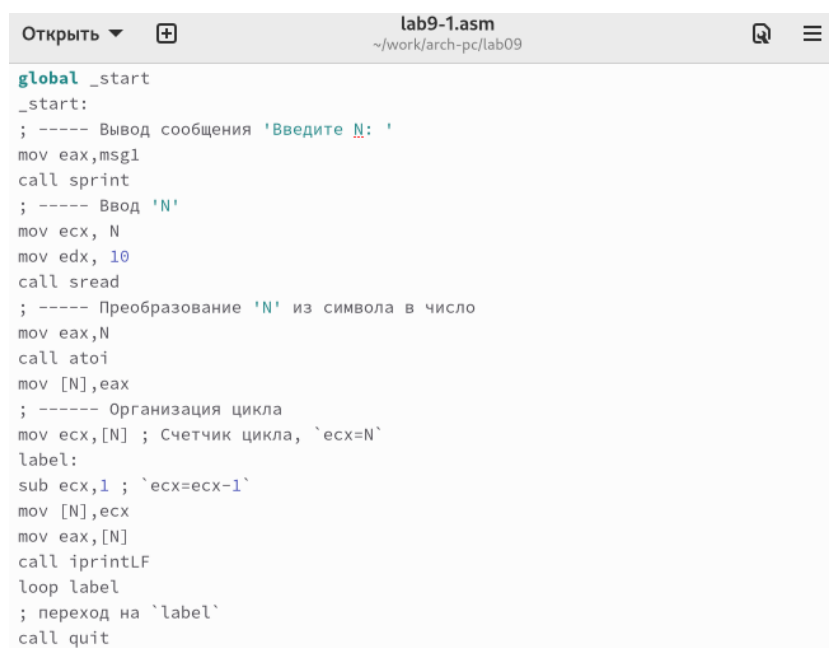
```
Открыть ▾ + lab9-1.asm ~/work/arch-pc/lab09
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
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. 3.2: Текст программы

```
[asstarceval@fedora lab09]$ nasm -f elf lab9-1.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[asstarceval@fedora lab09]$ ./lab9-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
```

Рис. 3.3: Исполняемый файл

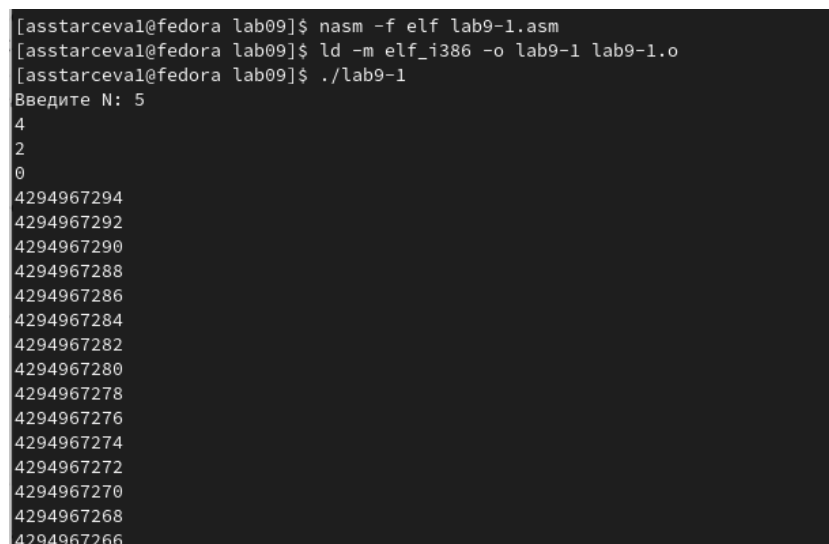
Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Изменили текст программы добавив изменение значения регистра ecx в цикле: (рис. 3.4)



```
Открыть + lab9-1.asm ~/work/arch-pc/lab09
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
loop label
; переход на `label`
call quit
```

Рис. 3.4: Текст программы

Создали исполняемый файл и проверили его работу. (рис. 3.5) Регистр `ecx` принимает следующие значения в цикле: (рис. 3.6). Число проходов цикла не соответствует значению `N` введенному с клавиатуры.



```
[asstarceval@fedora lab09]$ nasm -f elf lab9-1.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[asstarceval@fedora lab09]$ ./lab9-1
Введите N: 5
4
2
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
4294967270
4294967268
4294967266
```

Рис. 3.5: Исполняемый файл


```

4294967198
4294967196
4294967194
4294967192
4294967190
4294967188
4294967186
4294967184
4294967182
4294967180
4294967178
4294967176
4294967174
4294967172
4294967170
4294967168
4294967166
4294967164
4294967162
4294967160
4294967158
4294967156
4294967154

```

Рис. 3.6: Значения в цикле

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесли изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`: (рис. 3.7)

```

Открыть ▾  lab9-1.asm
~/work/arch-pc/lab09

_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
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
; переход на `label`
call quit

```

Рис. 3.7: Текст программы

Создали исполняемый файл и проверили его работу. (рис. 3.8) В данном случае число проходов цикла соответствует значению N введенному с клавиатуры.

```
[asstarceval@fedora lab09]$ nasm -f elf lab9-1.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[asstarceval@fedora lab09]$ ./lab9-1
Введите N: 16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
[asstarceval@fedora lab09]$
```

Рис. 3.8: Исполняемый файл

3.2 Обработка аргументов командной строки

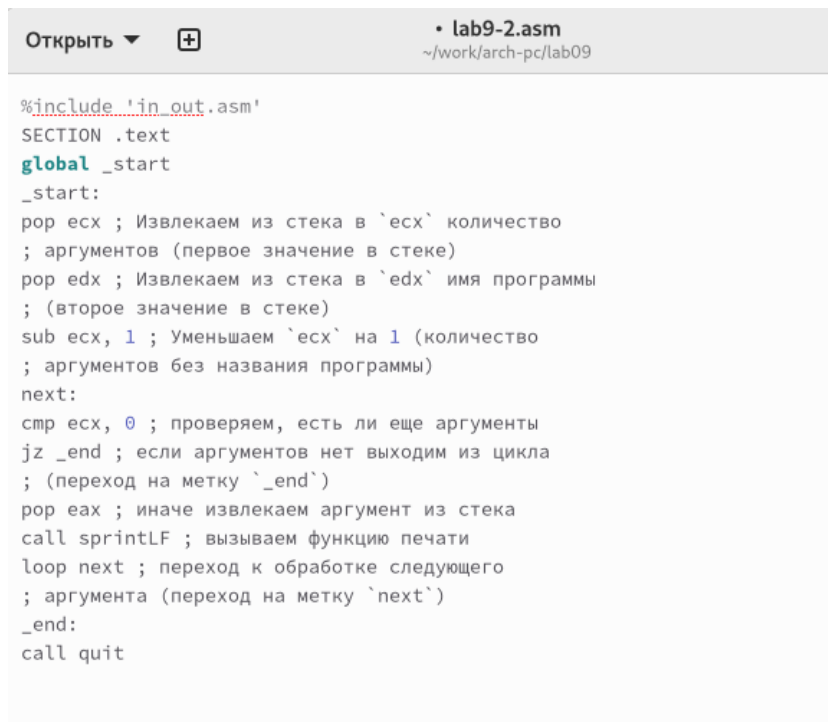
При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрели программу, которая выводит на экран аргументы командной строки. Внимательно изучили текст программы (Листинг 9.2).

Создали файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и ввели в него текст

программы из листинга 9.2. (рис. 3.9), (рис. 3.10) Создали исполняемый файл и запустили его, указав аргументы: (рис. 3.11)

```
[asstarceval@fedora lab09]$ touch lab9-2.asm
```

Рис. 3.9: lab9-2.asm



```
Открыть ▾ + lab9-2.asm
~/work/arch-pc/lab09

%include 'in_out.asm'
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por 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. 3.10: Текст программы

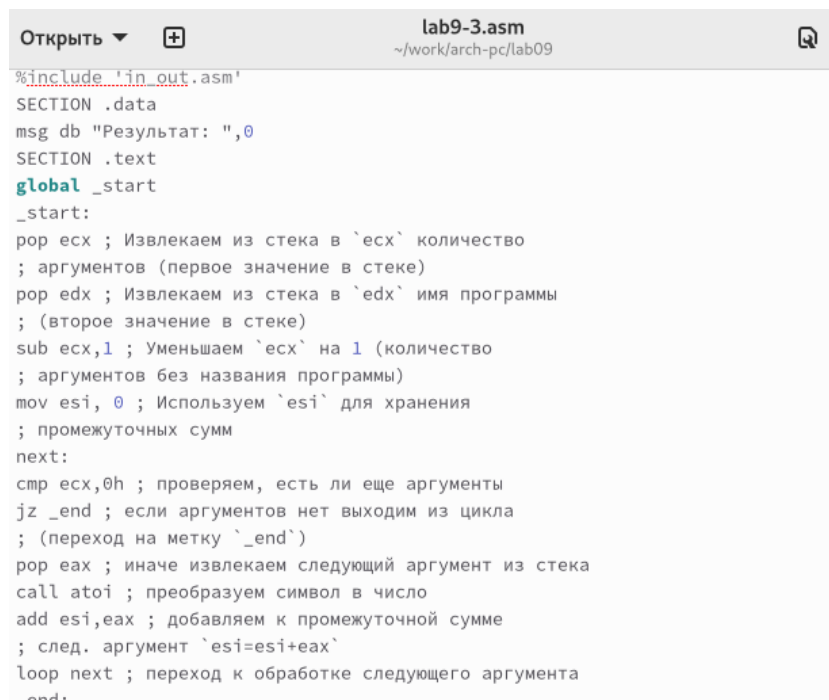
```
[asstarceval@fedora lab09]$ nasm -f elf lab9-2.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[asstarceval@fedora lab09]$ ./lab9-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[asstarceval@fedora lab09]$
```

Рис. 3.11: Исполняемый файл

Четыре аргумента было обработано программой. Рассмотрели еще один пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создали файл lab9-3.asm в каталоге ~/work/arch-pc/lab09 и ввели в него текст программы из листинга 9.3. (рис. 3.12), (рис. 3.13)

```
[asstarceval@fedora lab09]$ touch lab9-3.asm
```

Рис. 3.12: lab9-3.asm



```
Открыть ▾ + lab9-3.asm ~/work/arch-pc/lab09
%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:
```

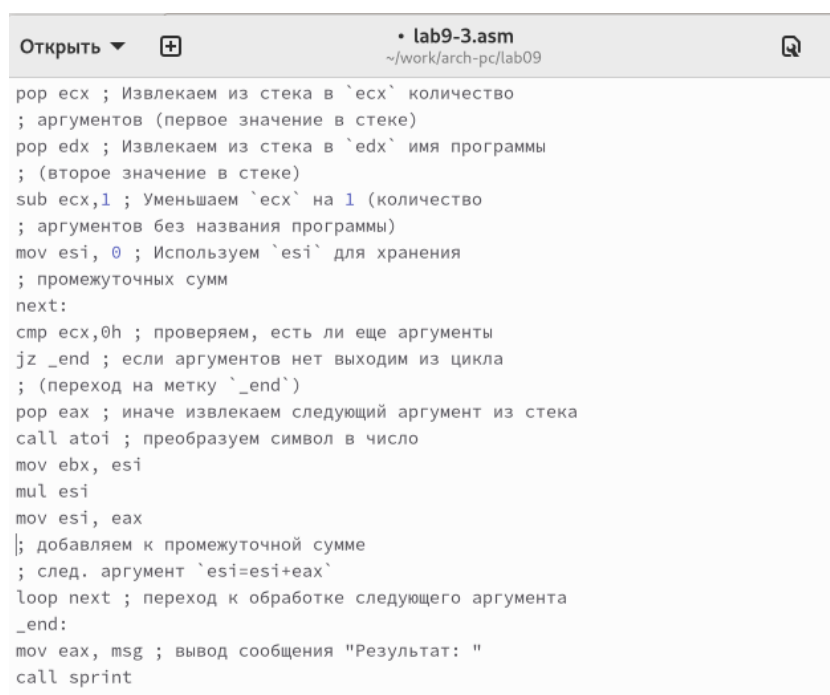
Рис. 3.13: Текст программы

Создали исполняемый файл и запустили его, указав аргументы. (рис. 3.14)

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

Рис. 3.14: Исполняемый файл

Изменили текст программы из листинга 9.3 для вычисления произведения аргументов командной строки. (рис. 3.15), (рис. 3.16)



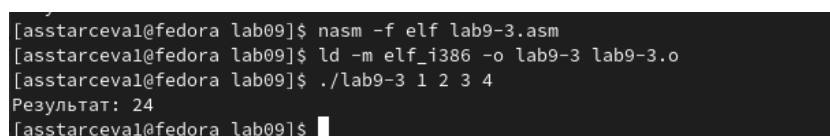
```

Открыть ▾ + lab9-3.asm
~/work/arch-pc/lab09

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
mov ebx, esi
mul esi
mov 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 printf

```

Рис. 3.15: Текст программы



```

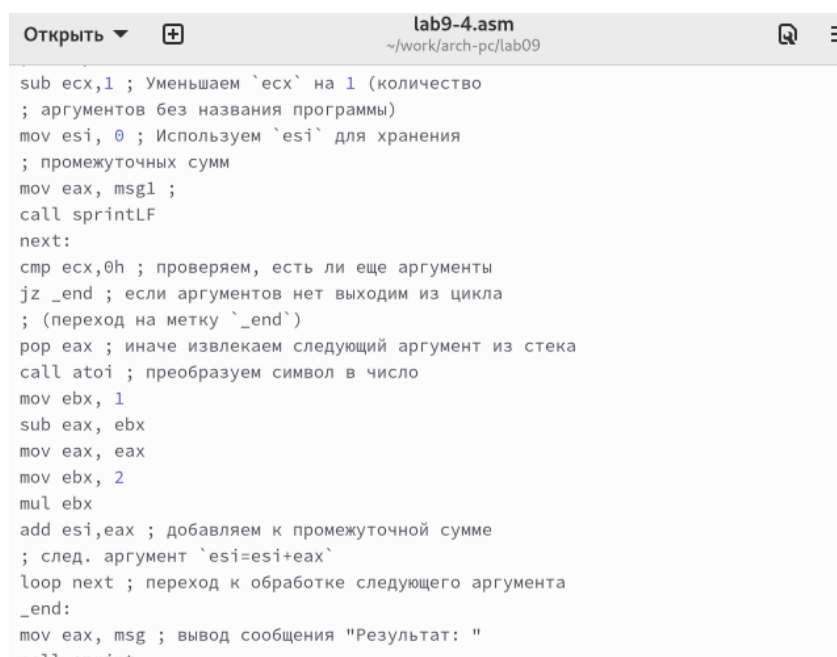
[asstarceval@fedora lab09]$ nasm -f elf lab9-3.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[asstarceval@fedora lab09]$ ./lab9-3 1 2 3 4
Результат: 24
[asstarceval@fedora lab09]$

```

Рис. 3.16: Исполняемый файл

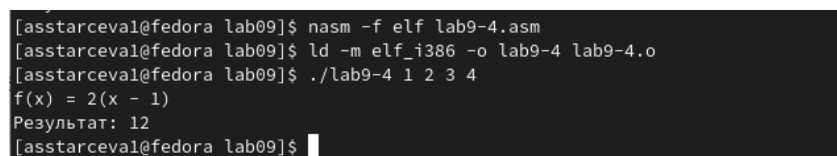
3.3 Задание для самостоятельной работы

Написали 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_i передаются как аргументы. Вид функции $f(x)$ выбрал из таблицы 9.1 вариантов заданий в соответствии с вариантом 4, полученным при выполнении лабораторной работы № 7. Создали исполняемый файл и проверили его работу на нескольких наборах. (рис. 3.17), (рис. 3.18)



```
Открыть + lab9-4.asm ~/work/arch-pc/lab09
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
mov eax, msg1 ;
call sprintf
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
mov ebx, 1
sub eax, ebx
mov eax, eax
mov ebx, 2
mul ebx
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 printf
```

Рис. 3.17: Текст программы



```
[asstarceval@fedora lab09]$ nasm -f elf lab9-4.asm
[asstarceval@fedora lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[asstarceval@fedora lab09]$ ./lab9-4 1 2 3 4
f(x) = 2(x - 1)
Результат: 12
[asstarceval@fedora lab09]$
```

Рис. 3.18: Исполняемый файл

4 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки.