

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Незами Ахмад Белал

Содержание

1	Цель работы	5
2	Задание	6
2.1	1. Реализация циклов в NASM	6
2.2	2. Обработка аргументов командной строки	6
2.3	3. Самостоятельная работа	6
3	Выполнение лабораторной работы	7
3.0.1	Программа отработала верно!	15
4	Самостоятельная работа	16
5	Выводы	20

Список иллюстраций

3.1	Создание директории	7
3.2	Редактирование файла	8
3.3	Запуск исполняемого файла	9
3.4	Уменьшение индекса	9
3.5	Запуск исполняемого файла	10
3.6	Редактирование программы	10
3.7	Создание исполняемого файла	11
3.8	Создание файла	11
3.9	Вставляю текст в файл	12
3.10	Запуск исполняемого файла	12
3.11	Создание файла	13
3.12	Вставляю программу	14
3.13	Запуск программы	14
3.14	Редактирование файла	15
3.15	Запуск программы	15
4.1	Создание файла	16
4.2	Запуск исполняемого файла	17
4.3	Запуск исполняемого файла	18

Список таблиц

1 Цель работы

Получение навыков по организации циклов и работе со стеком на языке NASM.

2 Задание

2.1 1. Реализация циклов в NASM

2.2 2. Обработка аргументов командной строки

2.3 3. Самостоятельная работа

3 Выполнение лабораторной работы

1

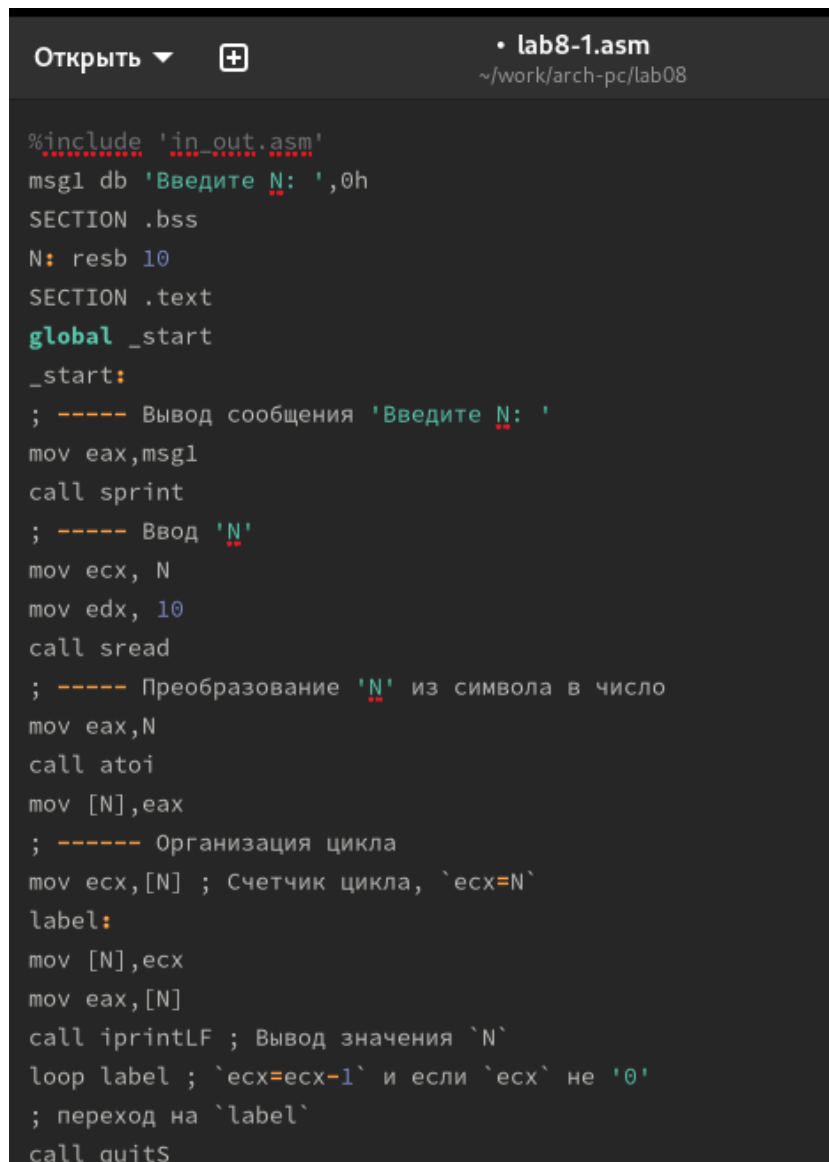
С помощью утилиты `mkdir` создаю директорию `lab08`, перехожу в нее и создаю файл для работы. (рис. [3.1]).

```
[nezami.a@fedora ~]$ mkdir ~/work/arch-pc/lab08
[nezami.a@fedora ~]$ cd ~/work/arch-pc/lab08
[nezami.a@fedora lab08]$ touch lab8-1.asm
[nezami.a@fedora lab08]$
```

Рис. 3.1: Создание директории

2

Открываю созданный файл `lab8-1.asm`, вставляю в него программу с использованием цикла для вывода чисел. (рис. [3.2]).



```
Открыть ▾ + • lab8-1.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
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 quits
```

Рис. 3.2: Редактирование файла

3

Создаю исполняемый файл программы и запускаю его. (рис. [3.3]).


```

[nezami.a@fedora lab08]$ touch lab8-1.asm
[nezami.a@fedora lab08]$ nasm -f elf lab8-1.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[nezami.a@fedora lab08]$ ./lab8-1
Введите N: 13
13
12
11
10
9
8
7
6
5
4
3
2
1
[nezami.a@fedora lab08]$

```

Рис. 3.3: Запуск исполняемого файла

4

С помощью инструкции `sub` уменьшаю изначальный индекс индекс на 1 единичку. (рис. [3.4]).

```

label:
sub ecx,1|
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 _exit

```

Рис. 3.4: Уменьшение индекса

5

Создаю новый исполняемый файл программы и запускаю его. (рис. [3.5]). Получаем результат отличный от ожидаемого

```
[nezami.a@fedora lab08]$ nasm -f elf lab8-1.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[nezami.a@fedora lab08]$ ./lab8-1
Введите N: 10
9
7
5
3
1
[nezami.a@fedora lab08]$
```

Рис. 3.5: Запуск исполняемого файла

6

Изменяю текст программы так, чтобы получить нужный результат, используя стеки для запоминания данных. (рис. [3.6]).

```
mov ecx,[N] ; счетчик цикла, ecx=N
label:
push ecx ; memory ecx=N
sub ecx,1; N-1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx ; N=N
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

Рис. 3.6: Редактирование программы

7

Создаю исполняемый файл и проверяю работу программы. (рис. [3.7]).

```
[nezami.a@fedora lab08]$ nasm -f elf lab8-1.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[nezami.a@fedora lab08]$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[nezami.a@fedora lab08]$
```

Рис. 3.7: Создание исполняемого файла

- Программа отработало верно.

8

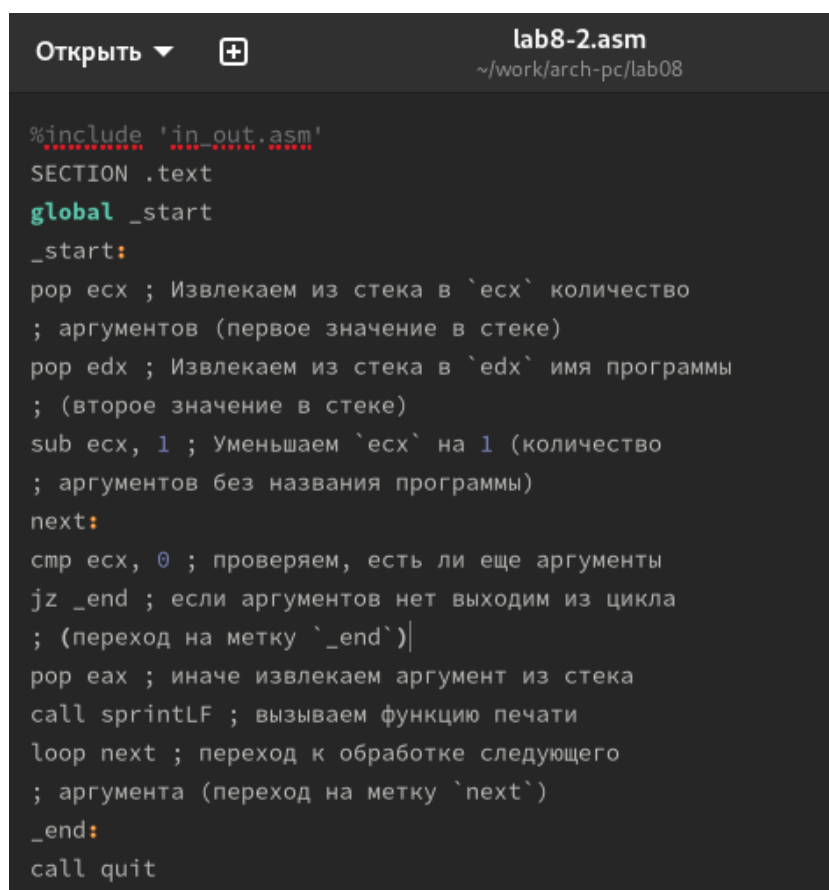
Создаю новый файл lab8-2.asm для новой программы. (рис. [3.8]).

```
[nezami.a@fedora lab08]$ touch lab8-2.asm
[nezami.a@fedora lab08]$
```

Рис. 3.8: Создание файла

9

Вставляю программу, которая выводит все введенные пользователем аргументы. (рис. [3.9]).



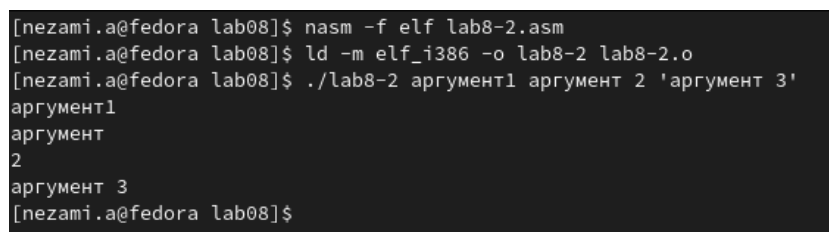
```
Открыть ▾ + lab8-2.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
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)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)|
pop 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. 3.9: Вставляю текст в файл

10

Создаю и запускаю новый исполняемый файл, проверяю работу программы.
(рис. [3.10]).



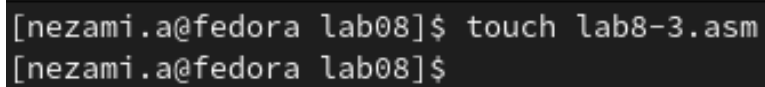
```
[nezami.a@fedora lab08]$ nasm -f elf lab8-2.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[nezami.a@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[nezami.a@fedora lab08]$
```

Рис. 3.10: Запуск исполняемого файла

- Программой было обработано 4 аргумента
 - Программа считает аргументами все символы до пробела, или значения, которые взяты в кавычки.
-

11

Создаю новый файл lab8-3.asm (рис. [3.11]).

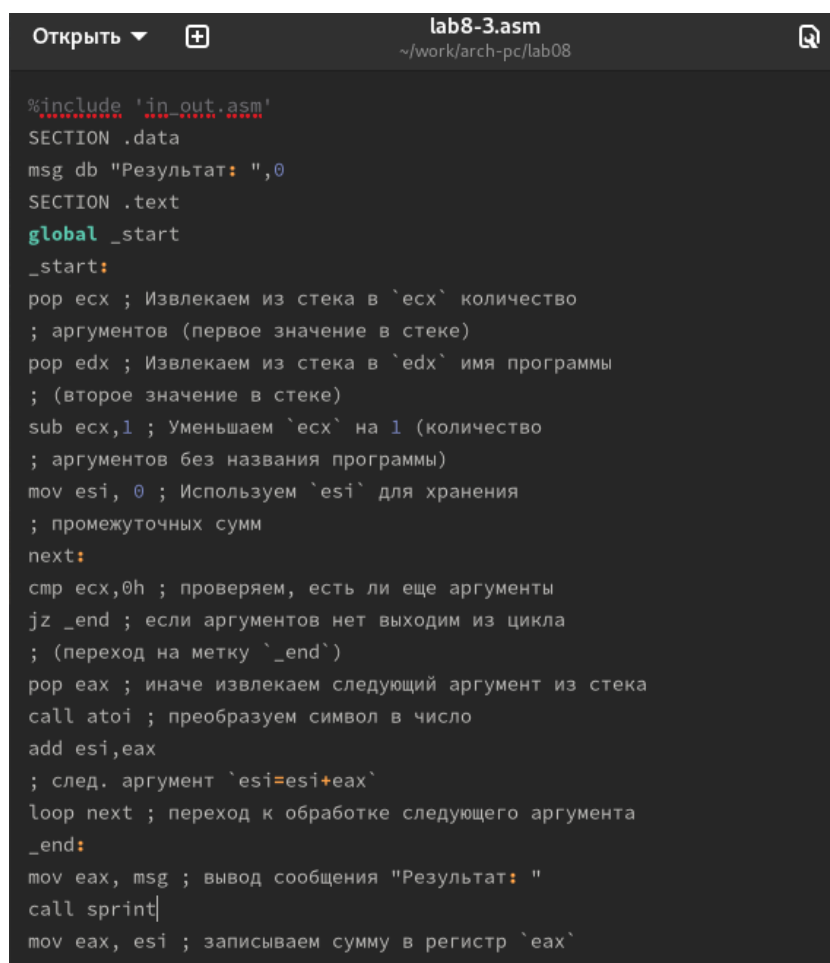


```
[nezami.a@fedora lab08]$ touch lab8-3.asm  
[nezami.a@fedora lab08]$
```

Рис. 3.11: Создание файла

12

Открываю файл и ввожу программу, которая складывает все числа введенные пользователем. (рис. [3.12]).



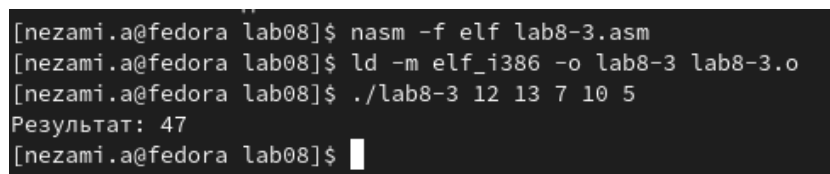
```
Открыть ▾ + lab8-3.asm
~/work/arch-pc/lab08

%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
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
int 3
```

Рис. 3.12: Вставляю программу

13

Запускаю исполняемый файл и проверяю работу программы. (рис. [3.13]).



```
[nezami.a@fedora lab08]$ nasm -f elf lab8-3.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[nezami.a@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[nezami.a@fedora lab08]$
```

Рис. 3.13: Запуск программы

14

Изменяю текст программы так, чтобы она выводила произведение всех чисел, введенные пользователем. (рис. [3.14]).

```
mov esi, 1 ; 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
mul esi ; умножаем предыдущее произведение на текущее
mov esi, eax ; добавляем к промежуточному произведению текущее
; след. аргумент `esi=esi*eax`
loop next ; переход к обработке следующего аргумента
_end:
```

Рис. 3.14: Редактирование файла

15

Запускаю исполняемый файл и проверяю работу программы. (рис. [3.15]).

```
[nezami.a@fedora lab08]$ nasm -f elf lab8-3.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[nezami.a@fedora lab08]$ ./lab8-3 4 5 3 2
Результат: 120
[nezami.a@fedora lab08]$
```

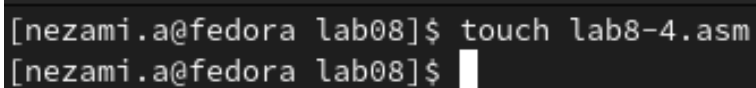
Рис. 3.15: Запуск программы

3.0.1 Программа отработала верно!

4 Самостоятельная работа

1

Создаю файл lab8-4.asm с помощью утилиты touch. (рис. [4.1]).

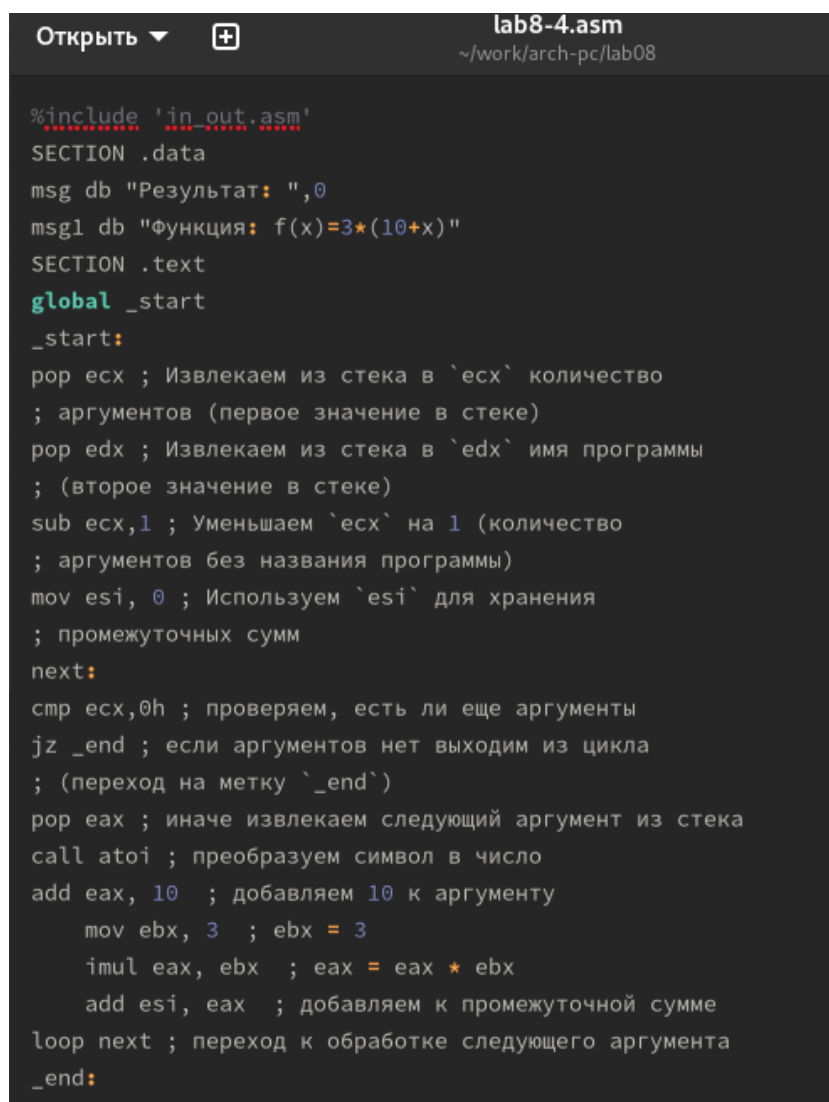


```
[nezami.a@fedora lab08]$ touch lab8-4.asm  
[nezami.a@fedora lab08]$
```

Рис. 4.1: Создание файла

2

Ввожу в созданный файл текст программы, у которой находит сумму значений функции (20 Вариант) $f(x)=3(10+x)$ для всех аргументов x , введенные пользователем. (рис. [4.2]).



```
lab8-4.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=3*(10+x)"
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 eax, 10 ; добавляем 10 к аргументу
mov ebx, 3 ; ebx = 3
imul eax, ebx ; eax = eax * ebx
add esi, eax ; добавляем к промежуточной сумме
loop next ; переход к обработке следующего аргумента
_end:
```

Рис. 4.2: Запуск исполняемого файла

3

Создаю исполняемый файл и запускаю его, при $x = 5, 3, 6$ (рис. [4.3]).

```

[nezami.a@fedora lab08]$ nasm -f elf lab8-4.asm
[nezami.a@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[nezami.a@fedora lab08]$ ./lab8-4 5 3 6
Функция: f(x)=3*(10+x)
Результат: 132
[nezami.a@fedora lab08]$

```

Рис. 4.3: Запуск исполняемого файла

Текст программы

```

#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=3*(10+x)"

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 eax, 10 ; добавляем 10 к аргументу

```

```
    mov ebx, 3    ; ebx = 3
    imul eax, ebx ; eax = eax * ebx
    add esi, eax  ; добавляем к промежуточной сумме
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg1 ;
call sprintf ;
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

5 Выводы

В ходе выполнения работы были получены навыки по организации циклов и по работе со стеком на языке NASM.