

Отчёт по лабораторной работе 8

Дисциплина: архитектура компьютера

Маныев Ресулбег Алексеевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация циклов в NASM	6
2.2	Обработка аргументов командной строки	12
2.3	Задание для самостоятельной работы	18
3	Выводы	21

Список иллюстраций

2.1	Код программы lab8-1.asm	7
2.2	Компиляция и запуск программы lab8-1.asm	8
2.3	Код программы lab8-1.asm	9
2.4	Компиляция и запуск программы lab8-1.asm	10
2.5	Код программы lab8-1.asm	11
2.6	Компиляция и запуск программы lab8-1.asm	12
2.7	Код программы lab8-2.asm	13
2.8	Компиляция и запуск программы lab8-2.asm	14
2.9	Код программы lab8-3.asm	15
2.10	Компиляция и запуск программы lab8-3.asm	16
2.11	Код программы lab8-3.asm	17
2.12	Компиляция и запуск программы lab8-3.asm	18
2.13	Код программы prog.asm	19
2.14	Компиляция и запуск программы prog.asm	20

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

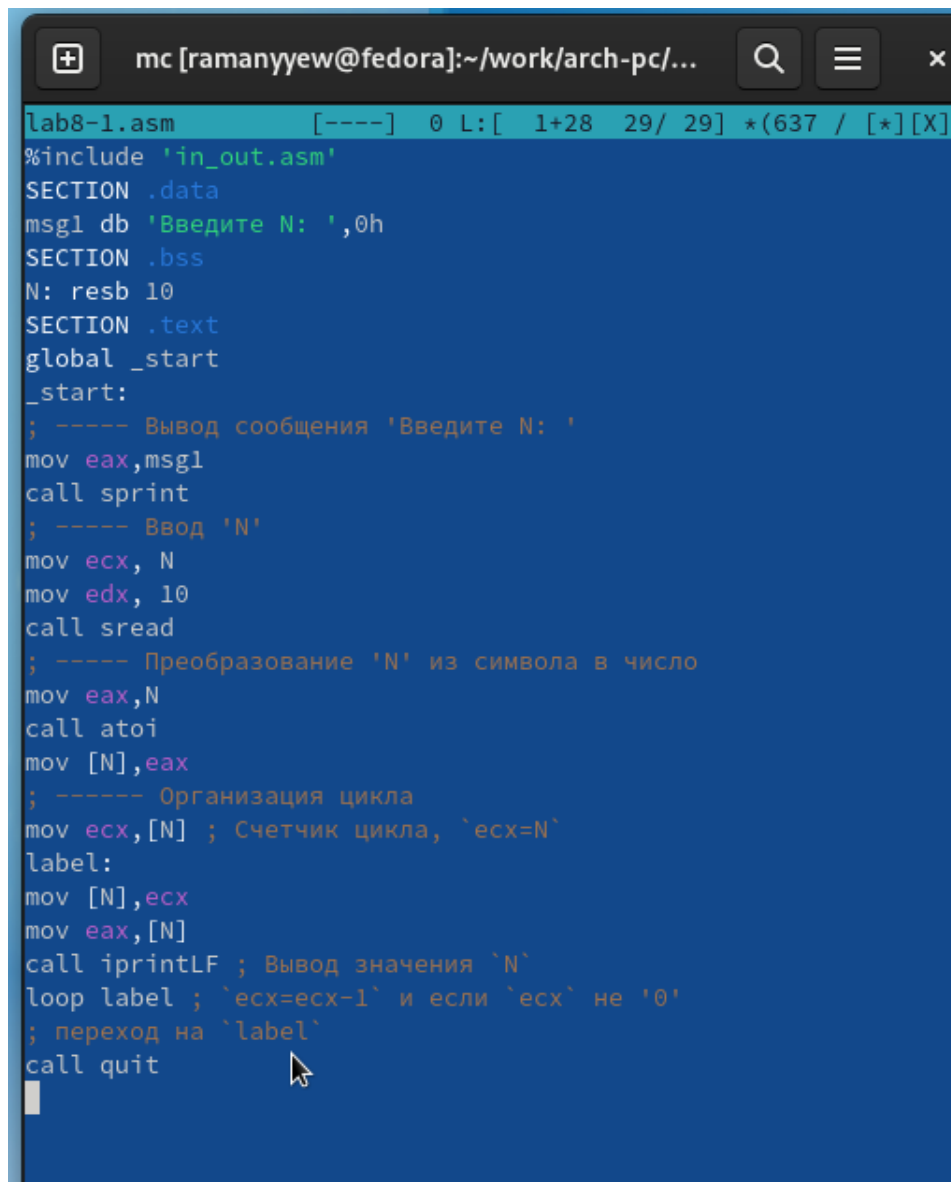
2.1 Реализация циклов в NASM

Был создан каталог для выполнения лабораторной работы № 8, а также создан файл с именем lab8-1.asm.

При использовании инструкции `loop` в NASM для реализации циклов, необходимо помнить о следующем: данная инструкция использует регистр `ecx` в качестве счетчика и на каждой итерации уменьшает его значение на единицу.

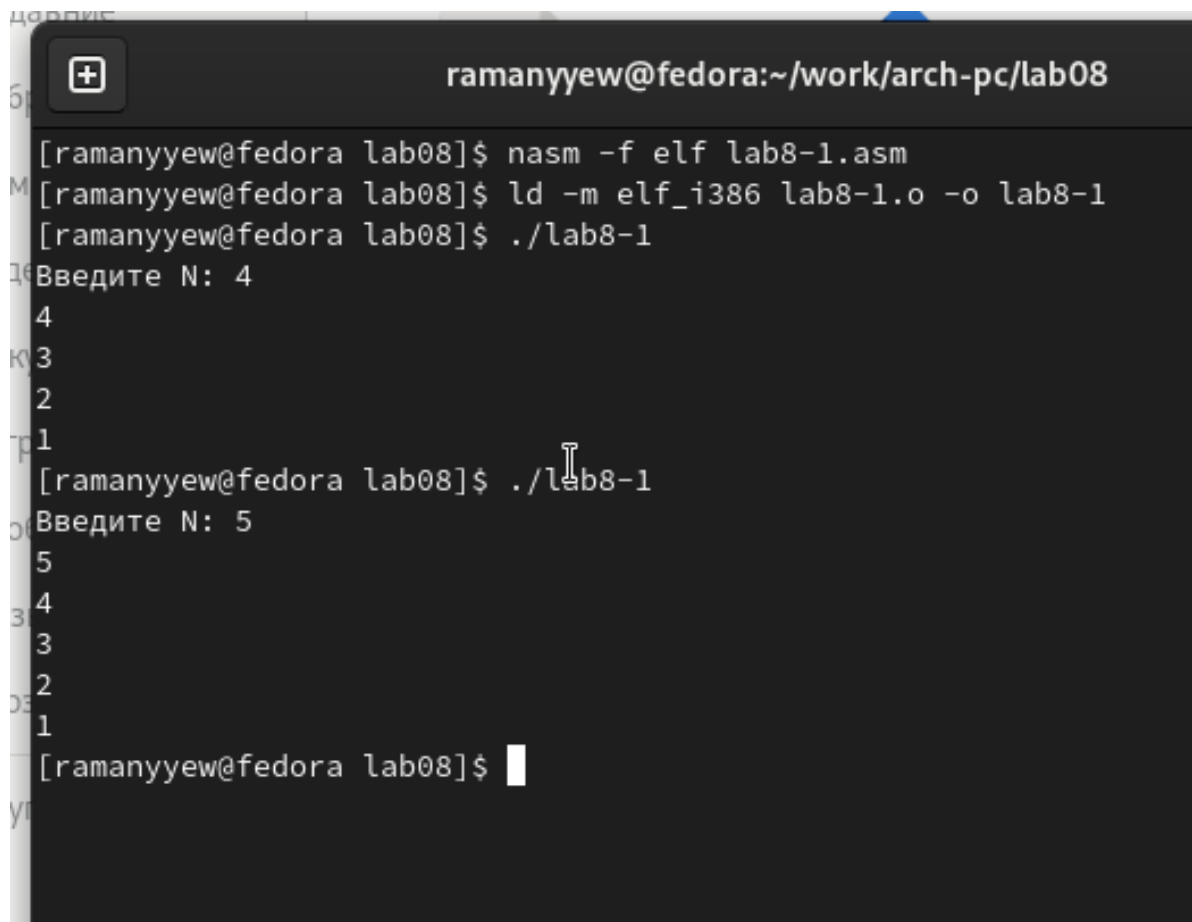
Для лучшего понимания этого процесса, рассмотрим пример программы, которая выводит значение регистра `ecx`.

Написал в файл lab8-1.asm текст программы из листинга 8.1. (рис. [2.1]) Создал исполняемый файл и проверил его работу. (рис. [2.2])



```
lab8-1.asm [----] 0 L: [ 1+28 29/ 29] *(637 / [*])[X]
#include 'in_out.asm'
SECTION .data
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 quit
```

Рис. 2.1: Код программы lab8-1.asm

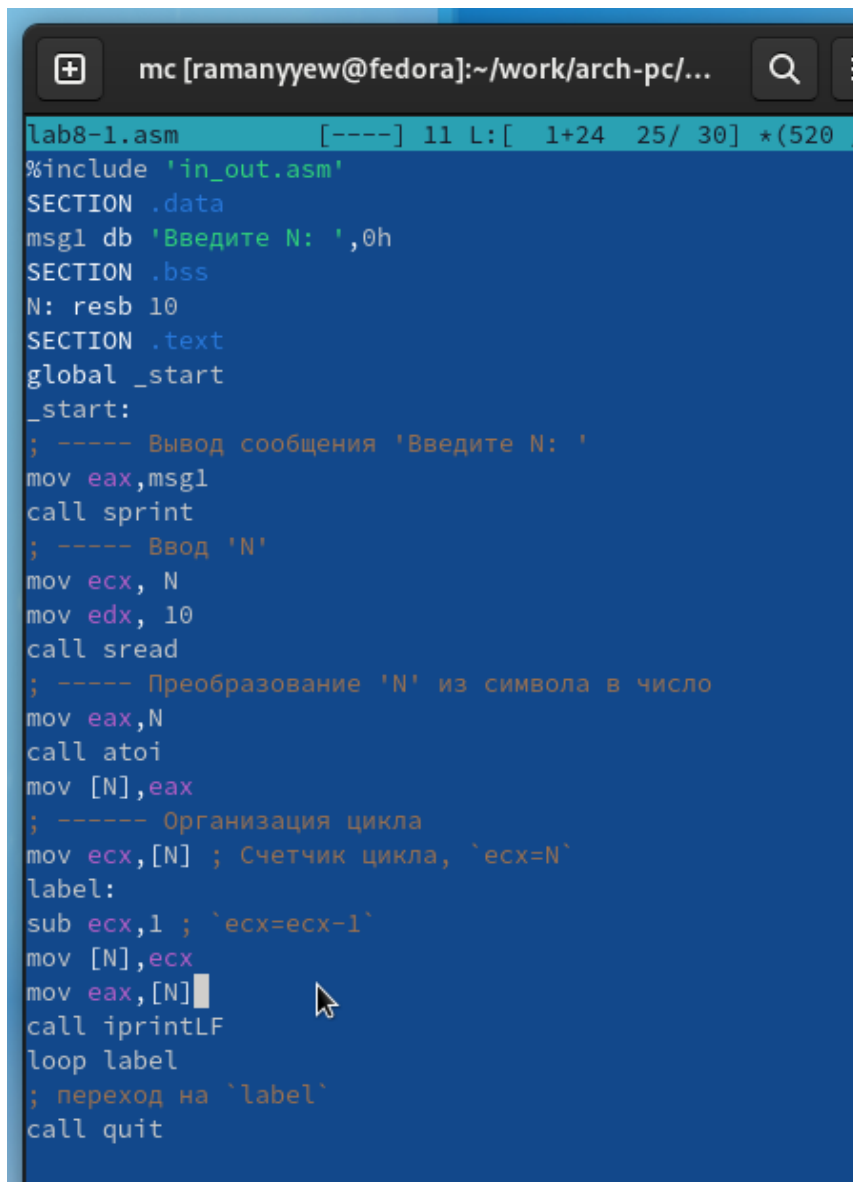


```
ramanyyew@fedora:~/work/arch-pc/lab08
[ramanyyew@fedora lab08]$ nasm -f elf lab8-1.asm
[ramanyyew@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[ramanyyew@fedora lab08]$ ./lab8-1
Введите N: 4
4
3
2
1
[ramanyyew@fedora lab08]$ ./lab8-1
Введите N: 5
5
4
3
2
1
[ramanyyew@fedora lab08]$
```

Рис. 2.2: Компиляция и запуск программы lab8-1.asm

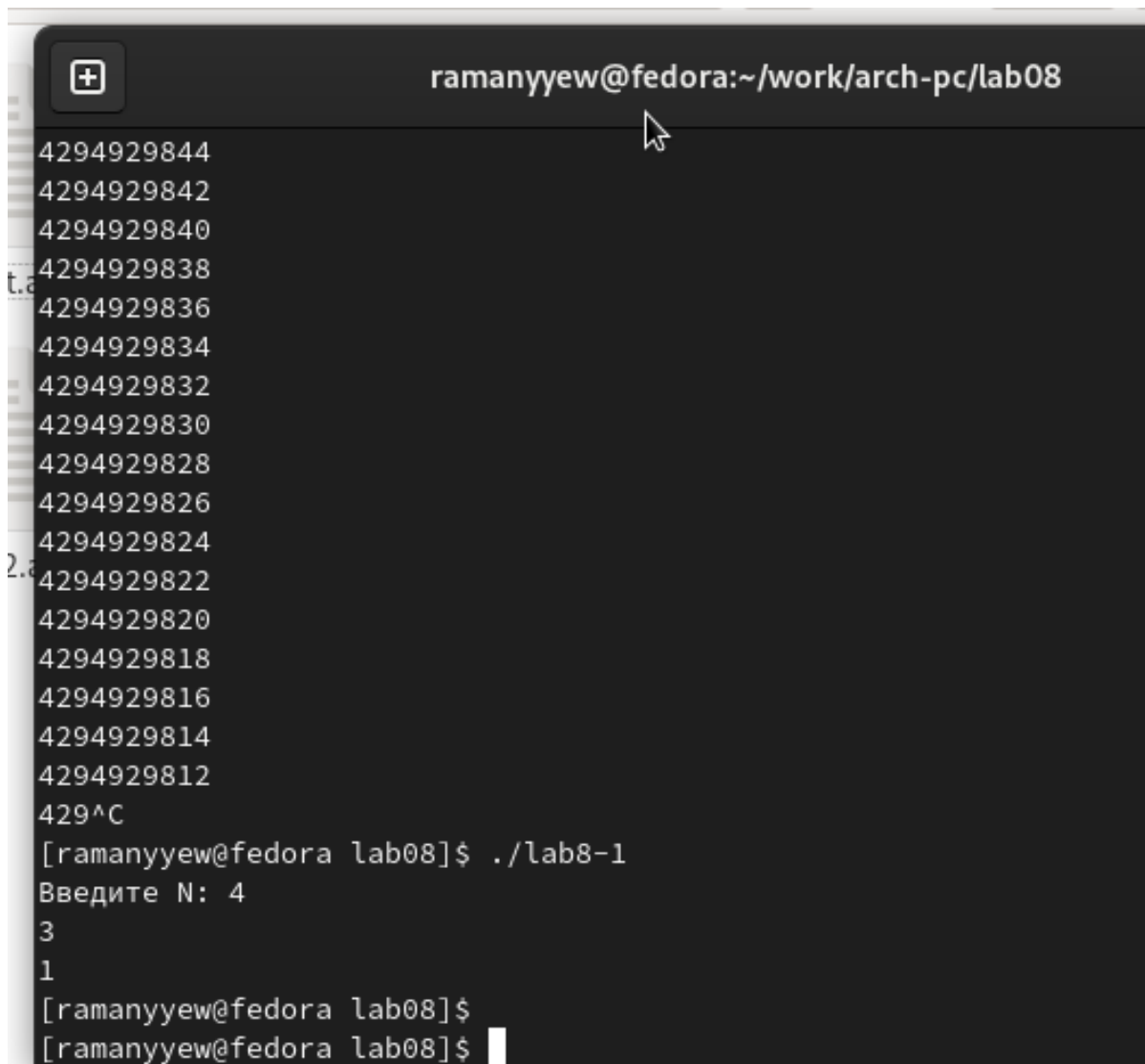
В данном примере демонстрируется, что использование регистра `ecx` в инструкции `loop` может привести к неправильной работе программы. В тексте программы были внесены изменения, которые включают изменение значения регистра `ecx` внутри цикла. (рис. [2.3])

Программа запускает бесконечный цикл при нечетном значении `N` и выводит только нечетные числа при четном значении `N`. (рис. [2.4])



```
lab8-1.asm [----] 11 L: [ 1+24 25/ 30] *(520 /
#include 'in_out.asm'
SECTION .data
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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label
; переход на `label`
call quit
```

Рис. 2.3: Код программы lab8-1.asm

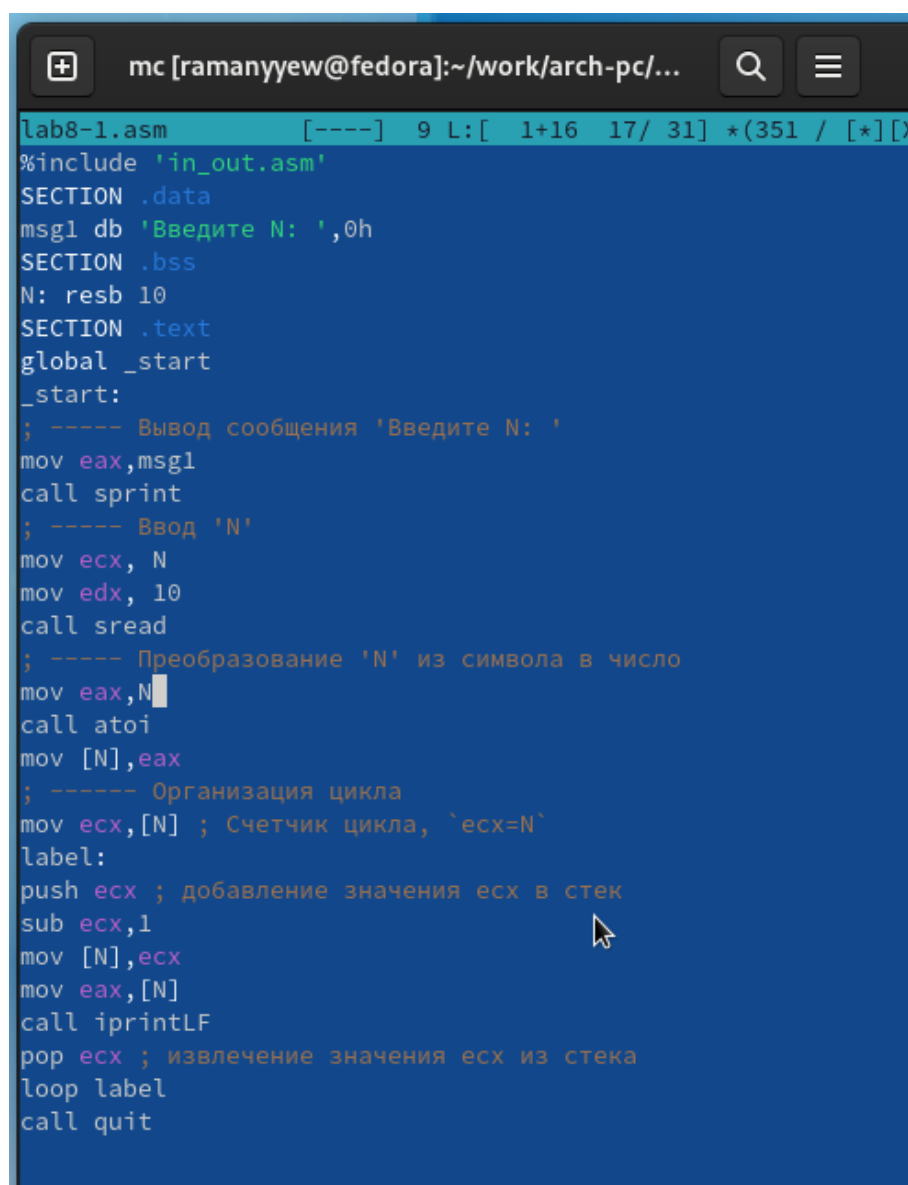
A terminal window with a dark background. The title bar shows a window icon, a plus sign, and the text 'ramanyyew@fedora:~/work/arch-pc/lab08'. The terminal content shows a list of memory addresses from 4294929844 down to 4294929812, followed by '429^C'. Then, the command '[ramanyyew@fedora lab08]\$./lab8-1' is entered. The output 'Введите N: 4' is shown, followed by the user input '3' and '1'. Finally, the prompt '[ramanyyew@fedora lab08]\$' is shown twice, with a cursor at the end of the second line.

```
ramanyyew@fedora:~/work/arch-pc/lab08
4294929844
4294929842
4294929840
4294929838
4294929836
4294929834
4294929832
4294929830
4294929828
4294929826
4294929824
4294929822
4294929820
4294929818
4294929816
4294929814
4294929812
429^C
[ramanyyew@fedora lab08]$ ./lab8-1
Введите N: 4
3
1
[ramanyyew@fedora lab08]$
[ramanyyew@fedora lab08]$
```

Рис. 2.4: Компиляция и запуск программы lab8-1.asm

Для того чтобы использовать регистр `ecx` в цикле и обеспечить корректность работы программы, можно применить стек. Внесены изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop` в стеке. (рис. [2.5])

Был создан исполняемый файл и проверена его работа. Программа выводит числа от $N-1$ до 0 , где количество проходов цикла соответствует значению N . (рис. [2.6])



```
lab8-1.asm [----] 9 L: [ 1+16 17/ 31] *(351 / [*]) [Y
#include 'in_out.asm'
SECTION .data
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:
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
call quit
```

Рис. 2.5: Код программы lab8-1.asm

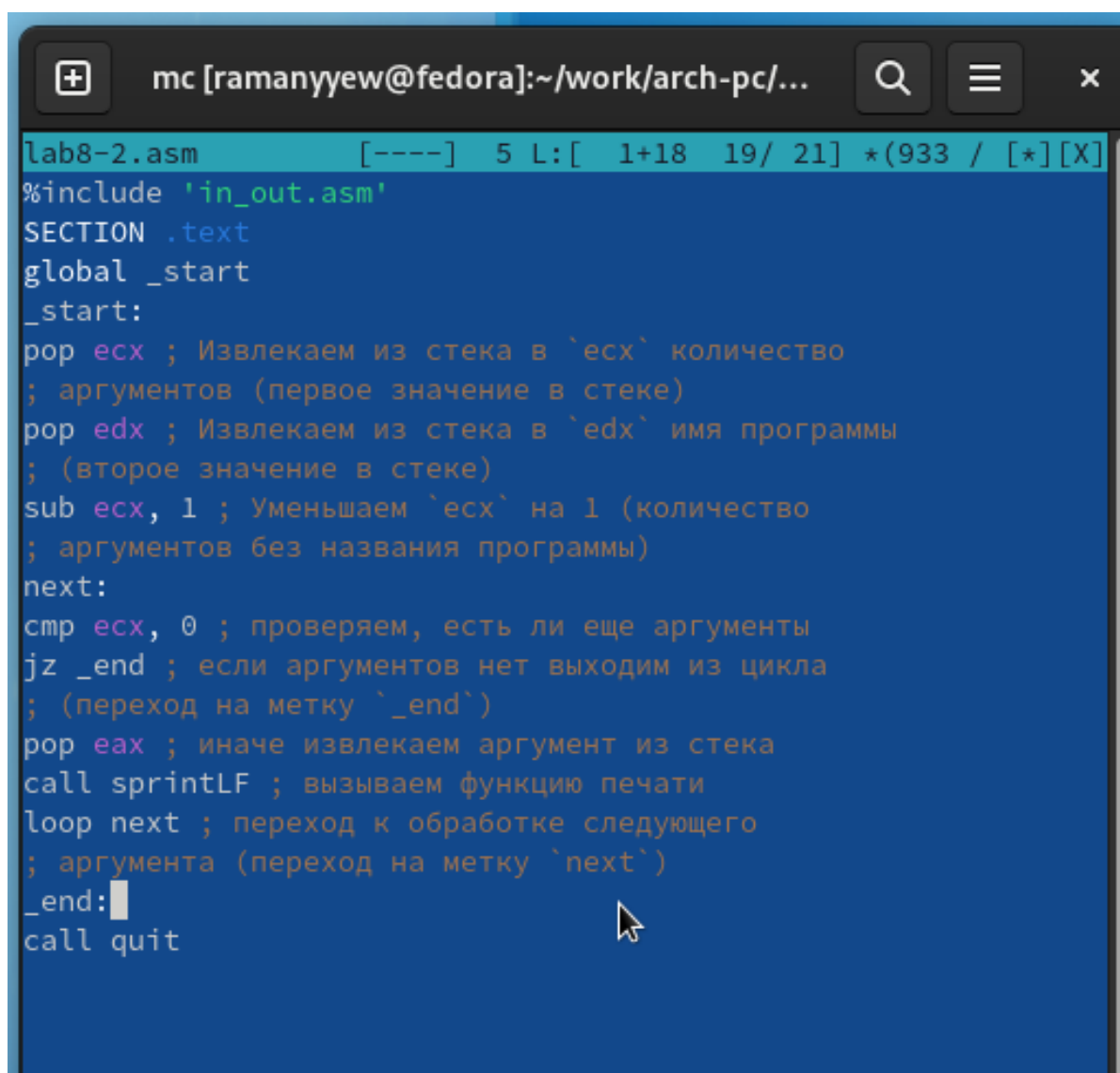
```
[ramanyyew@fedora lab08]$  
[ramanyyew@fedora lab08]$ nasm -f elf lab8-1.asm  
[ramanyyew@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1  
[ramanyyew@fedora lab08]$ ./lab8-1  
Введите N: 5  
4  
3  
2  
1  
0  
[ramanyyew@fedora lab08]$ ./lab8-1  
Введите N: 4  
3  
2  
1  
0  
[ramanyyew@fedora lab08]$
```

Рис. 2.6: Компиляция и запуск программы lab8-1.asm

2.2 Обработка аргументов командной строки

Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввел в него текст программы из листинга 8.2. (рис. [2.7])

Создал исполняемый файл и запустил его, указав аргументы. Программа обработала 5 аргументов. Аргументами считаются слова/числа, разделенные пробелом. (рис. [2.8])



```
lab8-2.asm [----] 5 L:[ 1+18 19/ 21] *(933 / [*])[X]
#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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

Рис. 2.7: Код программы lab8-2.asm

```
[ramanyyew@fedora lab08]$  
[ramanyyew@fedora lab08]$ nasm -f elf lab8-2.asm  
[ramanyyew@fedora lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2  
[ramanyyew@fedora lab08]$ ./lab8-2  
[ramanyyew@fedora lab08]$ ./lab8-2 argument 1 argument 2 'argument 3'  
argument  
1  
argument  
2  
argument 3  
[ramanyyew@fedora lab08]$
```

Рис. 2.8: Компиляция и запуск программы lab8-2.asm

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. [2.9]) (рис. [2.10])

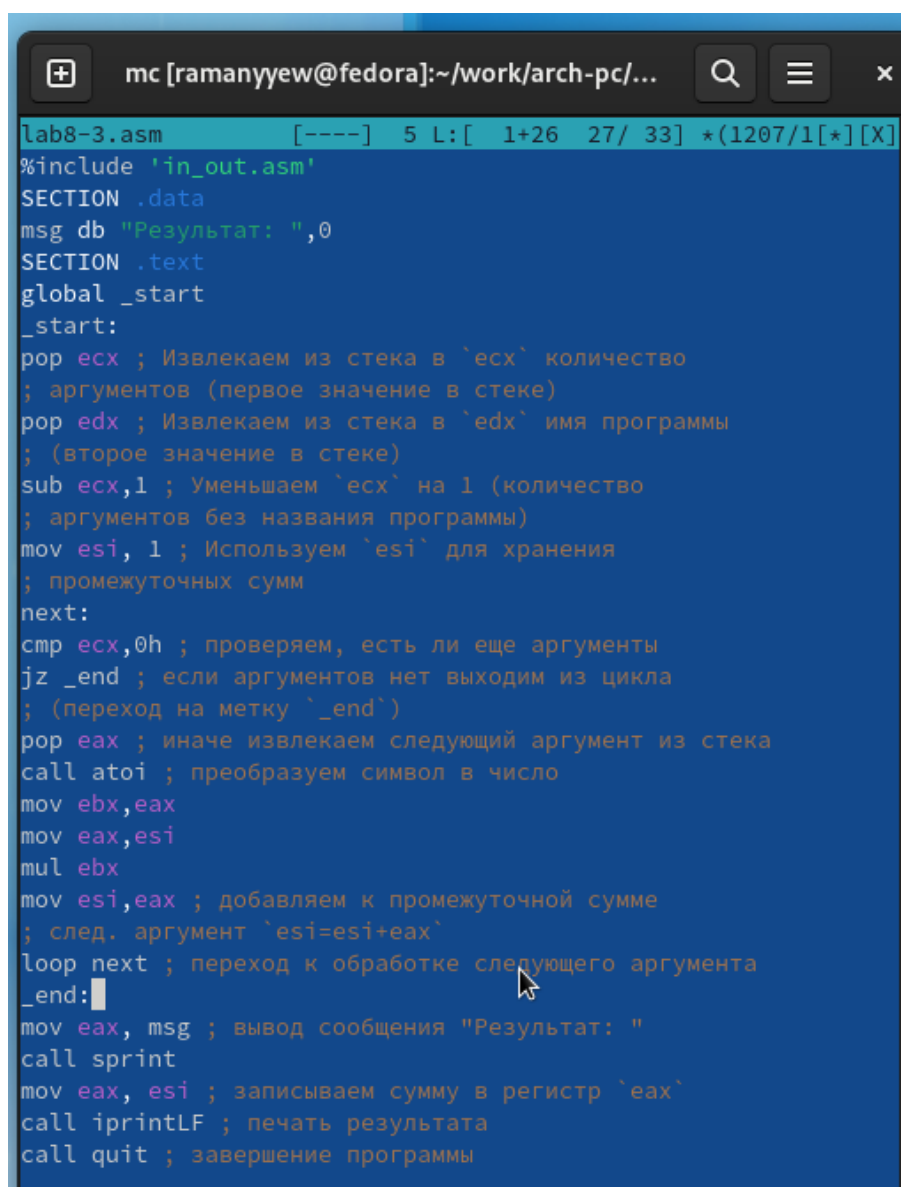
```
lab8-3.asm [----] 40 L: [ 1+26 27/ 30] *(1318/1[*] [X]
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе 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:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.9: Код программы lab8-3.asm

```
[ramanyyew@fedora lab08]$  
[ramanyyew@fedora lab08]$ nasm -f elf lab8-3.asm  
[ramanyyew@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3  
[ramanyyew@fedora lab08]$ ./lab8-3  
Результат: 0  
[ramanyyew@fedora lab08]$ ./lab8-3 6 9 8 7  
Результат: 30  
[ramanyyew@fedora lab08]$
```

Рис. 2.10: Компиляция и запуск программы lab8-3.asm

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. [2.11]) (рис. [2.12])



```
lab8-3.asm [----] 5 L: [ 1+26 27/ 33] *(1207/1[*] [X])
#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, 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
mov ebx,eax
mov eax,esi
mul ebx
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 sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 2.11: Код программы lab8-3.asm

```

[ramanyyew@fedora lab08]$
[ramanyyew@fedora lab08]$ nasm -f elf lab8-3.asm
[ramanyyew@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[ramanyyew@fedora lab08]$ ./lab8-3
Результат: 1
[ramanyyew@fedora lab08]$ ./lab8-3 6 9 8 7
Результат: 3024
[ramanyyew@fedora lab08]$
[ramanyyew@fedora lab08]$

```

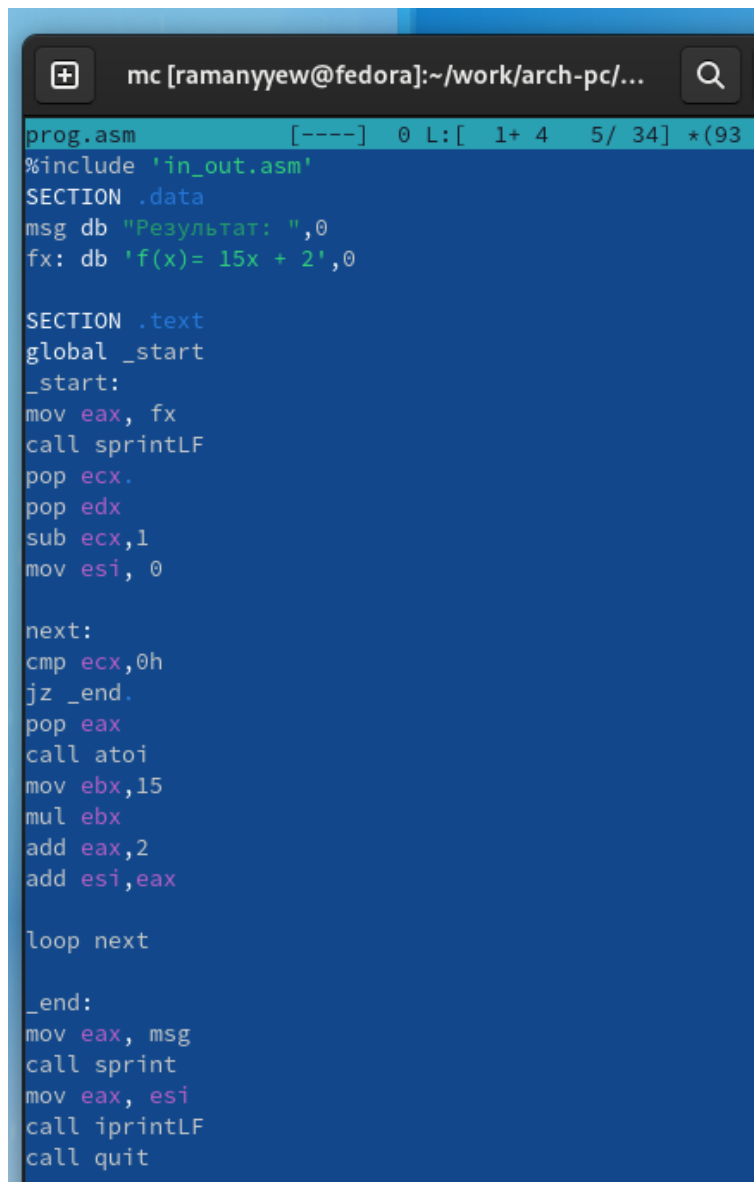
Рис. 2.12: Компиляция и запуск программы lab8-3.asm

2.3 Задание для самостоятельной работы

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 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x . (рис. [2.13]) (рис. [2.14])

Мой вариант 11:

$$f(x) = 15x + 2$$



```
mc [ramanyyew@fedora]:~/work/arch-pc/...
prog.asm [----] 0 L: [ 1+ 4 5/ 34] *(93
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)= 15x + 2',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintf
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov ebx,15
mul ebx
add eax,2
add esi,eax

loop next

_end:
mov eax, msg
call sprintf
mov eax, esi
call iprintLF
call quit
```

Рис. 2.13: Код программы prog.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(1) = 17, f(5) = 77$ Затем подал несколько аргументов и получил сумму значений функции.

```
[ramanyyew@fedora lab08]$  
[ramanyyew@fedora lab08]$ nasm -f elf prog.asm  
[ramanyyew@fedora lab08]$ ld -m elf_i386 prog.o -o prog  
[ramanyyew@fedora lab08]$ ./prog  
f(x)= 15x + 2  
Результат: 0  
[ramanyyew@fedora lab08]$ ./prog 1  
f(x)= 15x + 2  
Результат: 17  
[ramanyyew@fedora lab08]$ ./prog 5  
f(x)= 15x + 2  
Результат: 77  
[ramanyyew@fedora lab08]$ ./prog 5 7 8 9  
f(x)= 15x + 2  
Результат: 443  
[ramanyyew@fedora lab08]$
```

Рис. 2.14: Компиляция и запуск программы prog.asm

3 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `naasm`.