

Отчёт по лабораторной работе 8

Архитектура компьютера

Соловьев Серафим

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация циклов в NASM	6
2.2	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	17
2.13	Код программы lab8-4.asm	19
2.14	Компиляция и запуск программы lab8-4.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 организован каталог для выполнения лабораторного задания №8, в котором также был сформирован файл с наименованием lab8-1.asm.

Когда вы используете команду loop в NASM для создания циклических структур, важно учитывать, что она использует регистр ecx как счетчик, автоматически декрементируя его на один с каждым проходом цикла. Для наглядности рассмотрим пример кода, который демонстрирует значение регистра ecx.

В файл lab8-1.asm был введен код из примера 8.1. После этого была собрана исполняемая версия и проведена ее проверка.

```
lab08-1.asm      [----]  0 L:[ 1+28 29/ 29] *(637 / 637b) <EOF> [*]
#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

```
[sasoloviev@fedora-VirtualBox lab08]$ nasm -f elf lab08-1.asm
[sasoloviev@fedora-VirtualBox lab08]$ ld -m elf_i386 lab08-1.o -o lab08-1
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-1
Введите N: 3
3
2
1
[sasoloviev@fedora-VirtualBox lab08]$
```

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

В данном случае видно, что использование регистра `ecx` в команде `loop` может стать причиной ошибочного поведения программы. Я изменил код, изменив обработку значения регистра `ecx` во время цикла.

Теперь программа входит в бесконечный цикл, если `N` нечетное, и выводит только нечетные числа, если `N` четное.


```
mc [sasoloviev@fedora-VirtualBox]:~/work/arch-pc/lab08
lab08-1.asm [----] 6 L: [ 1+21 22/ 30] *(472 / 58)
#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
```

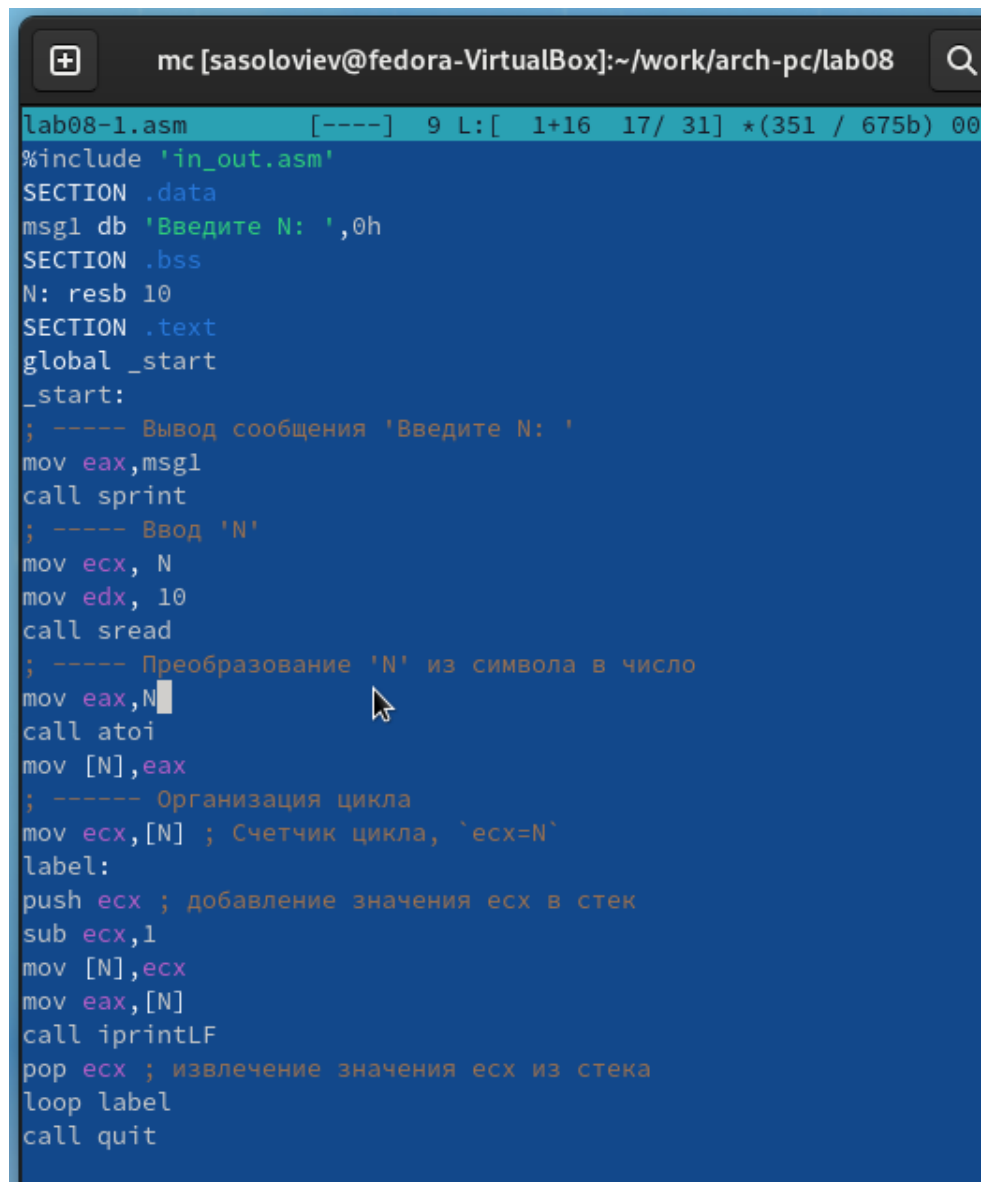
Рис. 2.3: Код программы lab8-1.asm

```
4294919646
4294919644
4294919642
4^C
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-1
Введите N: 4
3
1
[sasoloviev@fedora-VirtualBox lab08]$
```

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

Для корректного использования регистра `ecx` в цикле и обеспечения правильной работы программы можно применить стек. Я модифицировал код, добавив инструкции `push` и `pop`, чтобы сохранить значение счетчика цикла `loop` в стеке.

Была сформирована исполняемая версия и осуществлена ее проверка. Программа отображает числа от $N-1$ до 0, где число итераций соответствует величине N .



```
lab08-1.asm      [----]  9 L:[ 1+16 17/ 31] *(351 / 675b) 00
#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

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

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

Я создал файл с именем lab8-2.asm в папке ~/work/arch-pc/lab08 и занес в него код, взятый из примера 8.2.

После этого я собрал исполняемый файл из исходного кода и запустил его с параметрами. В итоге программа успешно обработала пять переданных ей параметров. Под параметрами понимаются элементы, разделяемые пробелами, которые могут быть текстом или числами.

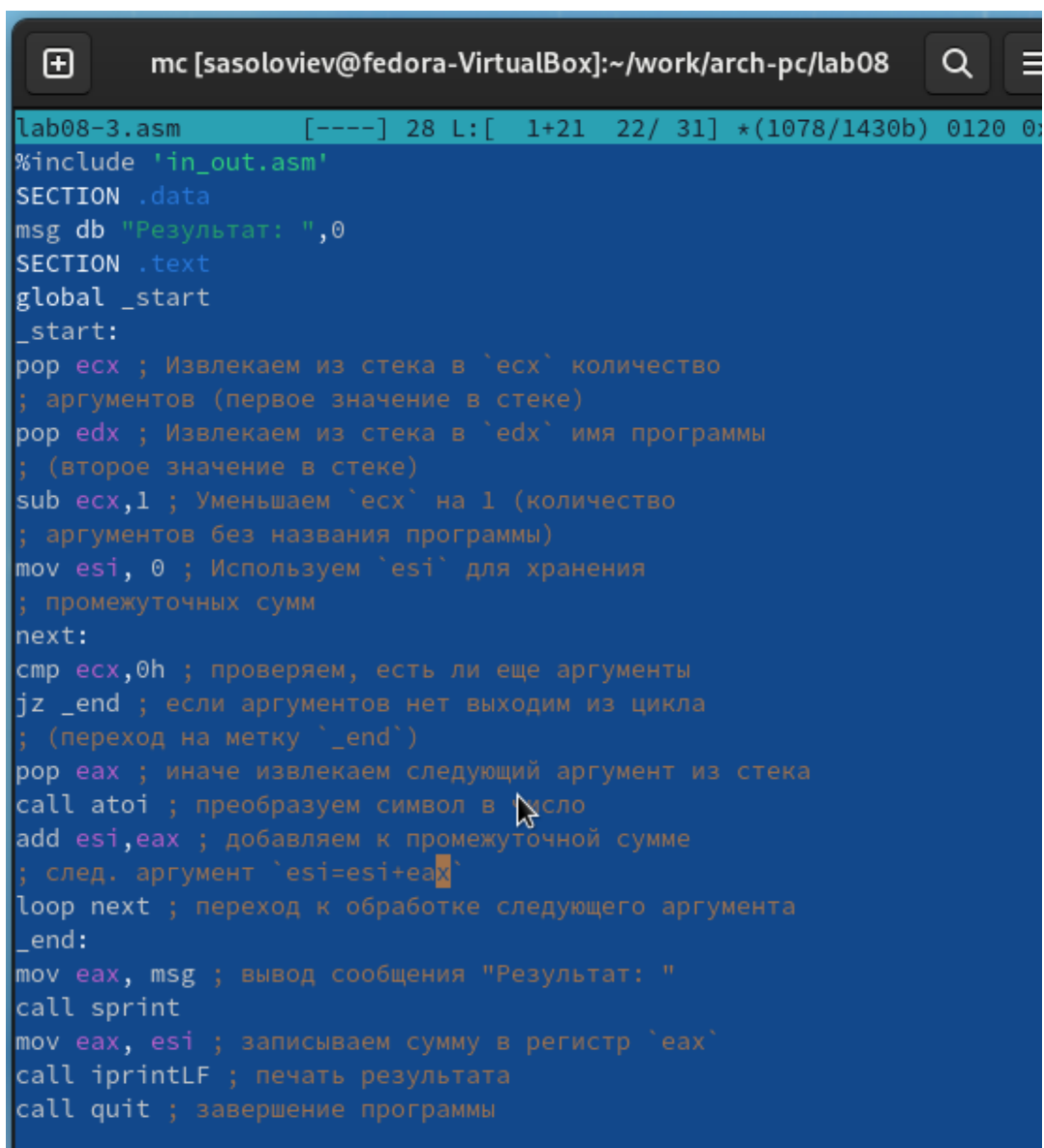
```
lab08-2.asm [----] 0 L:[ 1+ 6 7/ 21] *(202 / 944b) (
#include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `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. 2.7: Код программы lab8-2.asm

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

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

Теперь давайте рассмотрим другой пример программы, задачей которой является вывод на экран суммы чисел, передаваемых в неё в качестве параметров.



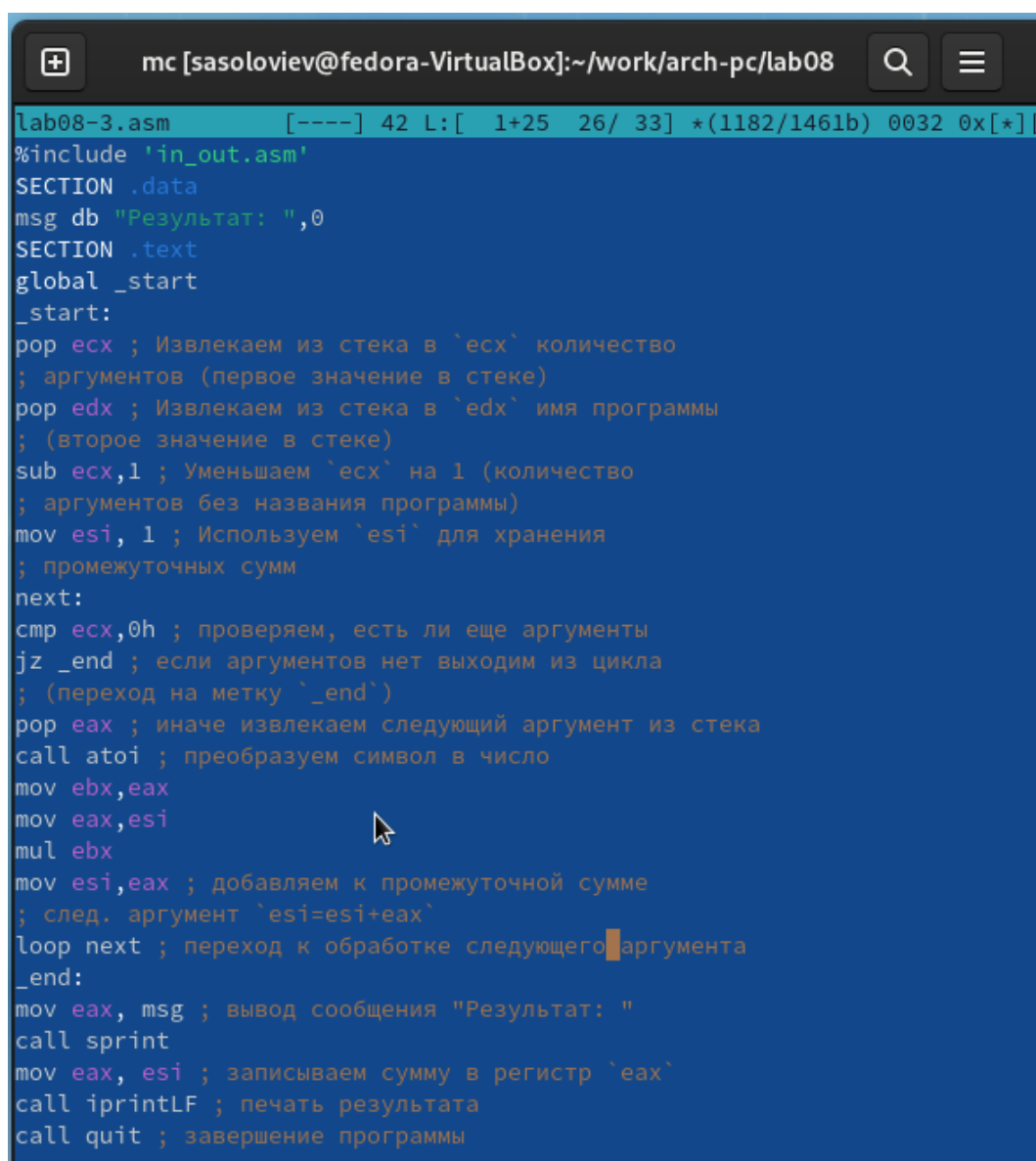
```
lab08-3.asm [----] 28 L: [ 1+21 22/ 31] *(1078/1430b) 0120 0:
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (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

```
[sasoloviev@fedora-VirtualBox lab08]$ nasm -f elf lab08-3.asm
[sasoloviev@fedora-VirtualBox lab08]$ ld -m elf_i386 lab08-3.o -o lab08-3
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-3
Результат: 0
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-3 2 3 4
Результат: 9
[sasoloviev@fedora-VirtualBox lab08]$
```

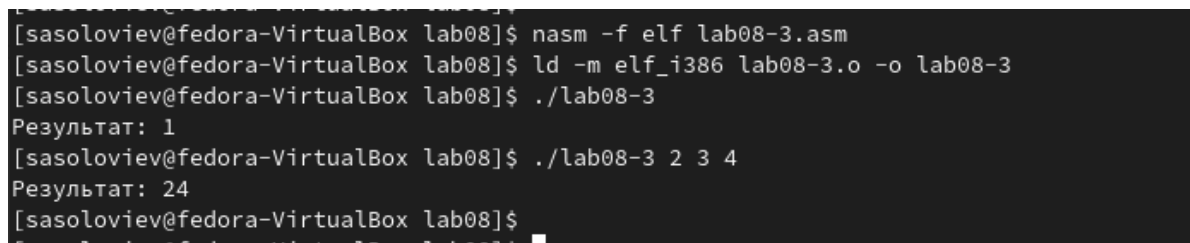
Рис. 2.10: Компиляция и запуск программы lab8-3.asm

Я внес изменения в код из примера 8.3 таким образом, чтобы программа теперь вычисляла произведение значений, переданных через командную строку.



```
lab08-3.asm [----] 42 L: [ 1+25 26/ 33] *(1182/1461b) 0032 0x[*]  
%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



```
[sasoloviev@fedora-VirtualBox lab08]$ nasm -f elf lab08-3.asm  
[sasoloviev@fedora-VirtualBox lab08]$ ld -m elf_i386 lab08-3.o -o lab08-3  
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-3  
Результат: 1  
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-3 2 3 4  
Результат: 24  
[sasoloviev@fedora-VirtualBox lab08]$
```

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

2.2 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 .

Мой вариант 18: $f(x) = 17 + 5x$

```
mc [sasoloviev@fedora-VirtualBox]:~/w
lab08-4.asm [----] 0 L: [ 2+11 13,
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)= 17 + 5x',0

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

next:
cmp ecx,0h
jz _end.
pop eax
call atoi
mov ebx,5
mul ebx
add eax,17
add esi,eax

loop next

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

Рис. 2.13: Код программы lab8-4.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(1) = 22, f(2) = 27$ Затем подал несколько аргументов и получил сумму значений функции.

```
[sasoloviev@fedora-VirtualBox lab08]$ nasm -f elf lab08-4.asm
[sasoloviev@fedora-VirtualBox lab08]$ ld -m elf_i386 lab08-4.o -o lab08-4
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-4
f(x)= 17 + 5x
Результат: 0
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-4 1
f(x)= 17 + 5x
Результат: 22
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-4 2
f(x)= 17 + 5x
Результат: 27
[sasoloviev@fedora-VirtualBox lab08]$ ./lab08-4 2 3 4 5 6 7
f(x)= 17 + 5x
Результат: 237
[sasoloviev@fedora-VirtualBox lab08]$
[sasoloviev@fedora-VirtualBox lab08]$
```

Рис. 2.14: Компиляция и запуск программы lab8-4.asm

3 Выводы

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