

# **Лабораторная работа №8**

**Программирование цикла. Обработка аргументов командной строки**

Борисенкова София Павловна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выполнение задания для самостоятельной работы</b>	<b>15</b>
<b>4</b>	<b>Выводы</b>	<b>17</b>

# Список иллюстраций

2.1	Создание рабочей директории и файла lab8-1.asm . . . . .	6
2.2	Вставка кода из файла листинга 8.1 . . . . .	7
2.3	Копирование файла in_out.asm в рабочую директорию . . . . .	7
2.4	Сборка программы из файла lab8-1.asm и её запуск . . . . .	8
2.5	Изменение файла lab8-1.asm . . . . .	8
2.6	Повторная сборка программы из файла lab8-1.asm и её запуск . . .	9
2.7	Результат для нечетного N . . . . .	9
2.8	Результат для чётного N . . . . .	10
2.9	Редактирование файла lab8-1.asm . . . . .	10
2.10	Повторная сборка программы из файла lab8-1.asm и её запуск . . .	11
2.11	Запись кода из листинга 8.2 в файл lab8-2.asm . . . . .	11
2.12	Сборка программы из файла lab8-2.asm и её запуск . . . . .	12
2.13	Запись кода из листинга 8.3 в файл lab8-3.asm . . . . .	12
2.14	Сборка программы из файла lab8-3.asm и её запуск . . . . .	13
2.15	Изменение файла lab8-3.asm . . . . .	13
2.16	Повторная сборка программы из файла lab8-3.asm и её запуск . . .	14
3.1	Код файла самостоятельной работы . . . . .	15
3.2	Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения . . . . .	16

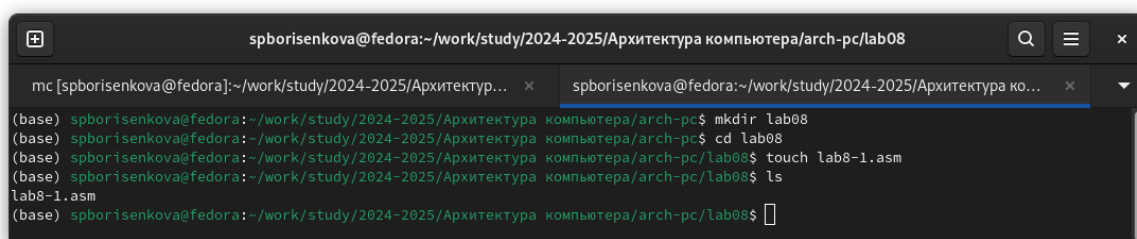
## **Список таблиц**

# 1 Цель работы

Научиться работать с циклами на языке Ассемблера, а также научиться обрабатывать аргументы командной строки

## 2 Выполнение лабораторной работы

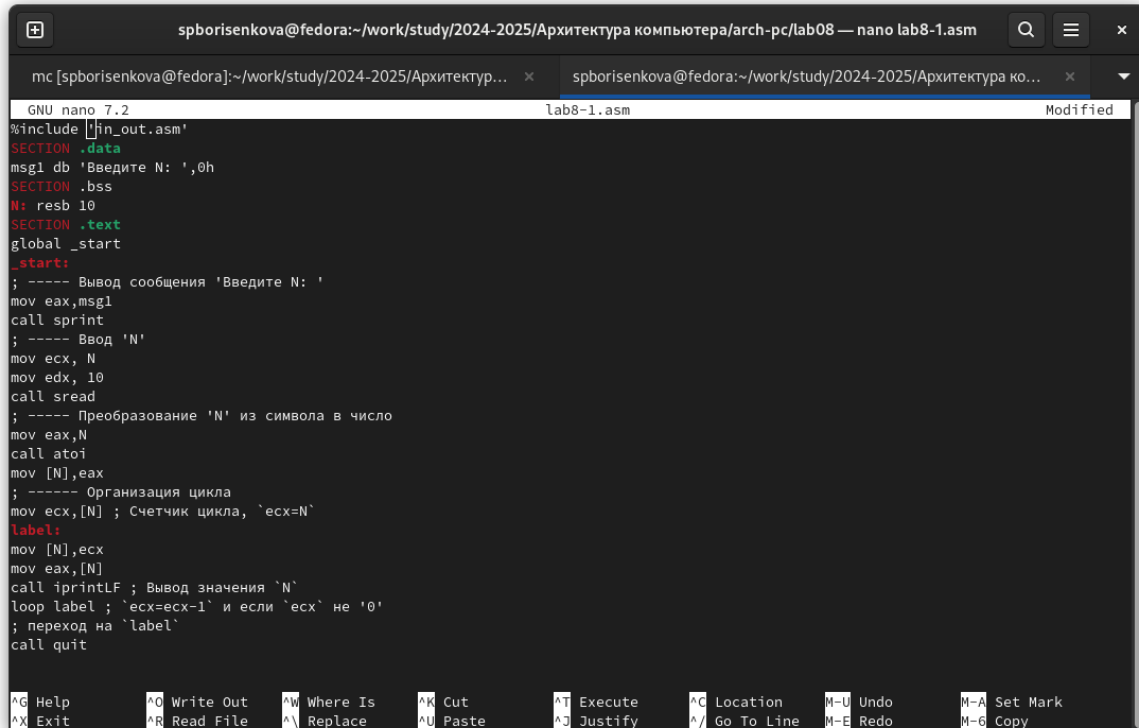
Для начала выполнения лабораторной работы создадим рабочую директорию и файл lab8-1.asm (рис. 2.1):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mkdir lab08
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd lab08
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-1.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ls
lab8-1.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.1: Создание рабочей директории и файла lab8-1.asm

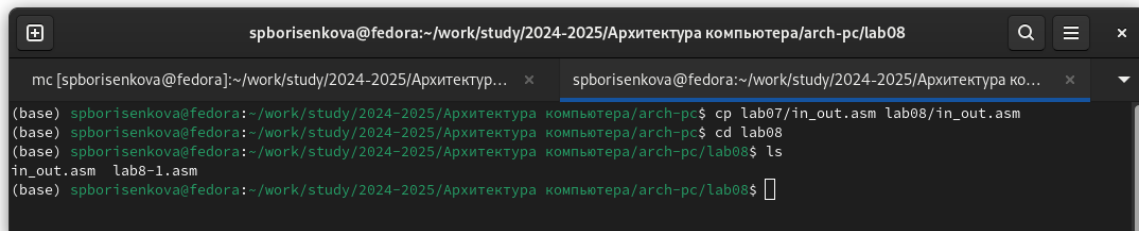
Теперь, вставим в ранее созданный файл из листинга 8.1. Он должен запускать цикл и выводить каждую итерацию число, на единицу меньше предыдущего (начинается выводить с числа N) (рис. 2.2):



```
GNU nano 7.2 lab8-1.asm Modified
mc [spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08] x spborisenkova@fedora:~/work/study/2024-2025/Архитектура ко... 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 iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 2.2: Вставка кода из файла листинга 8.1

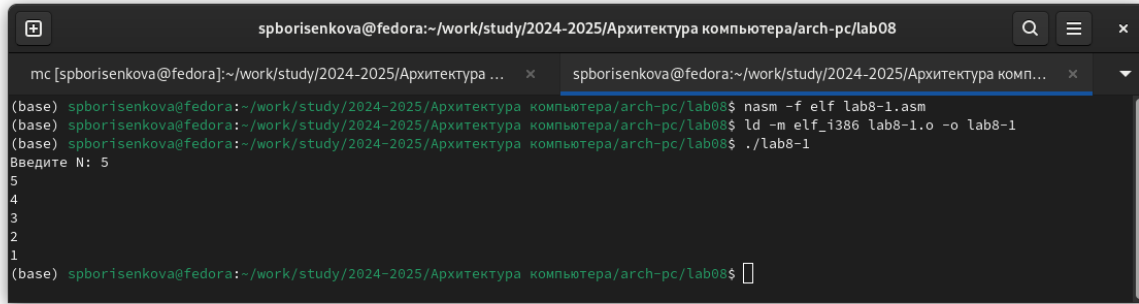
Чтобы собрать код, нужен файл `in_out.asm`. скопируем его из директории прошлой лабораторной работы (рис. 2.3):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cp lab07/in_out.asm lab08/in_out.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd lab08
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.3: Копирование файла `in_out.asm` в рабочую директорию

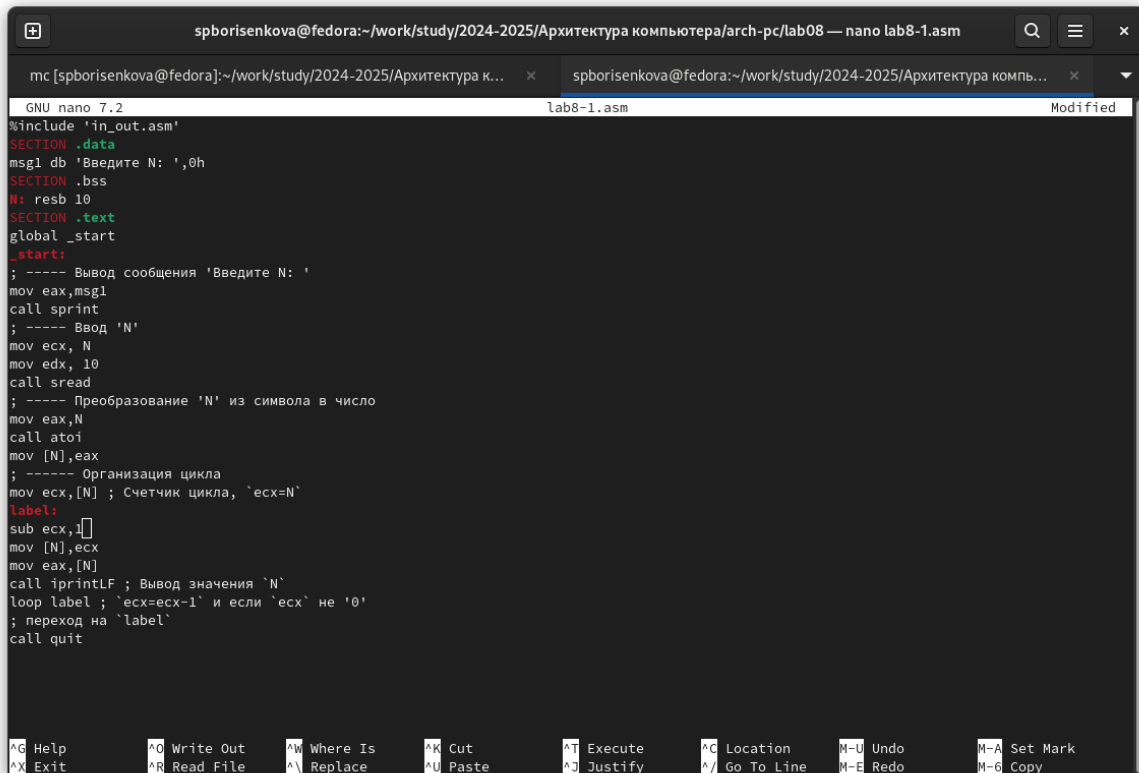
Теперь соберём программу и посмотрим на результат выполнения (рис. 2.4):



```
spborisenkova@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora]:~/work/study/2024-2025/Архитектура ... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура комп... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 5
5
4
3
2
1
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.4: Сборка программы из файла lab8-1.asm и её запуск

Как видим, она выводит числа от N до единицы включительно. Теперь попробуем изменить код, чтобы в цикле также отнималась единица у регистра ecx (рис. 2.5):



```
GNU nano 7.2 lab8-1.asm Modified
%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
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

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   ^U Undo       ^A Set Mark
^X Exit      ^R Read File  ^L Replace    ^U Paste      ^J Justify    ^_ Go To Line  ^E Redo       ^M Copy
```

Рис. 2.5: Изменение файла lab8-1.asm

Попробуем собрать программу и запустить её (рис. 2.6):



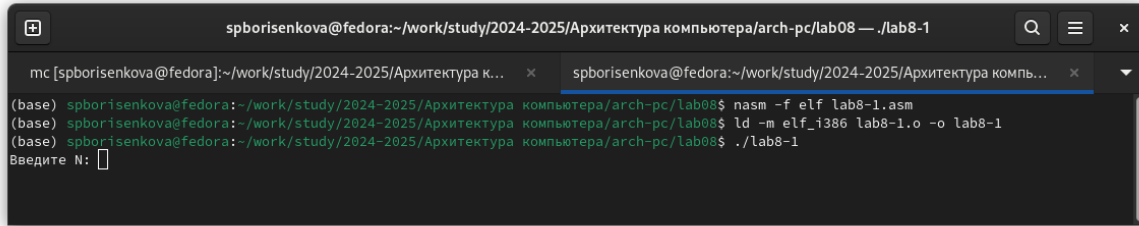


Рис. 2.6: Повторная сборка программы из файла lab8-1.asm и её запуск

Введём в качестве N число 5 и посмотрим на результат выполнения (рис. 2.7):

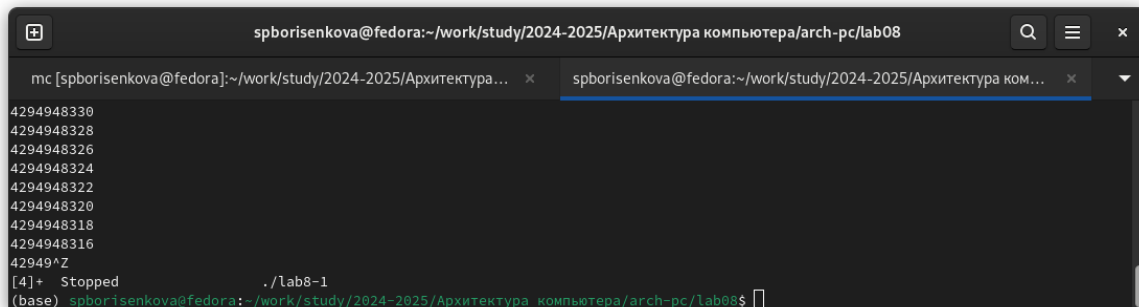
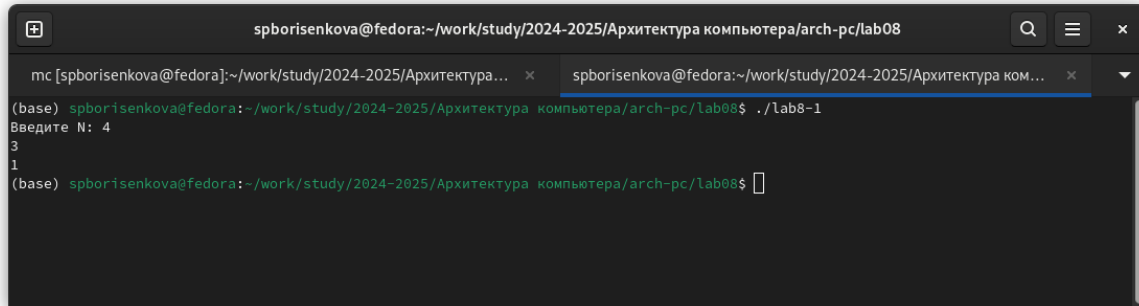


Рис. 2.7: Результат для нечетного N

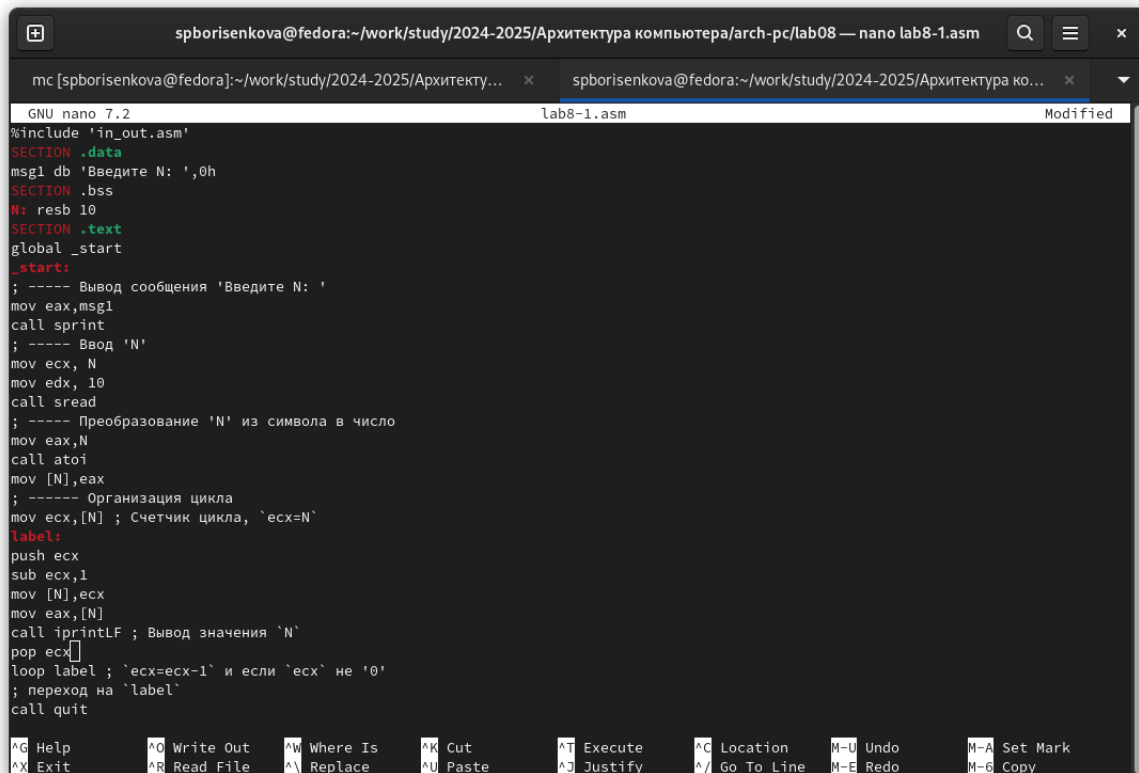
Как видим, цикл выполняется бесконечное количество раз. Это связано с тем, что цикл останавливается в тот момент, когда при проверке `esx` равен 0, но он каждое выполнение цикла уменьшается на 2, из-за чего, в случае нечётного числа, никогда не достигнет нуля. Регистр `esx` меняет своё значение дважды: стандартно -1 после каждой итерации и -1 в теле цикла из-за команды `sub`. Если на вход подать чётное число, цикл прогонится  $N/2$  раз, выводя числа от  $N-1$  до 1 (выводит через одно число) (рис. 2.8):



```
spborisenkova@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora]:~/work/study/2024-2025/Архитектура... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура ком... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.8: Результат для чётного N

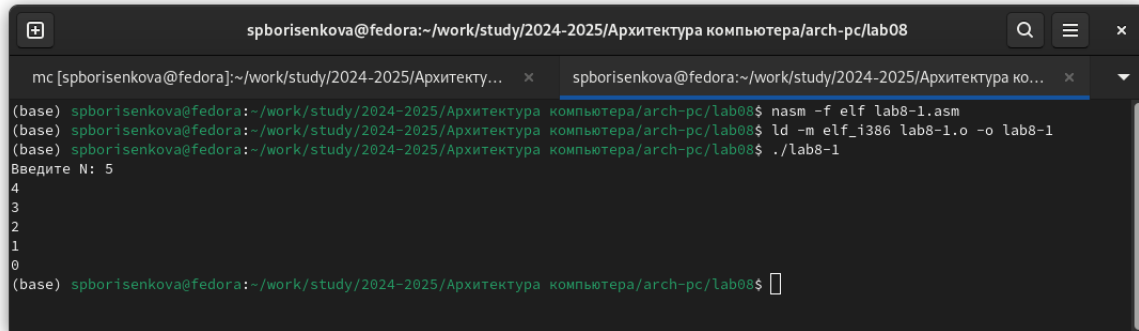
Таким образом, количество итераций цикла не равно N ни при подаче на вход чётного числа, ни при подаче нечётного. Теперь попробуем изменить программу так, чтобы она сохраняла значение регистра ecx в стек (рис. 2.9):



```
GNU nano 7.2 lab8-1.asm Modified
%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
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo M-G Copy
```

Рис. 2.9: Редактирование файла lab8-1.asm

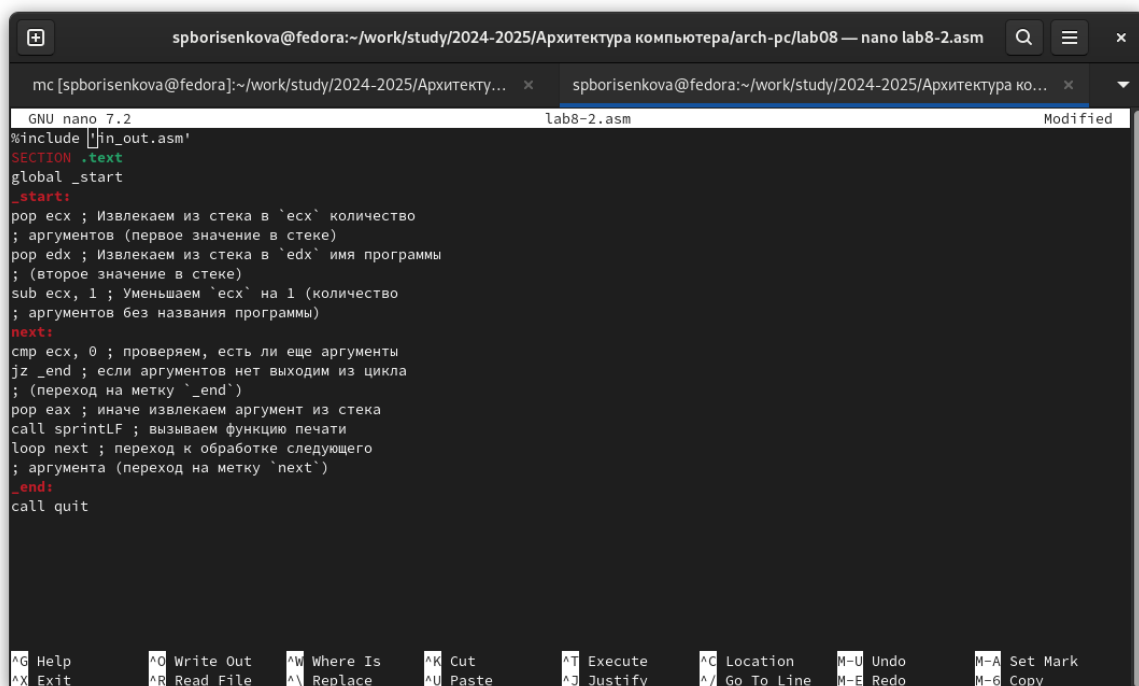
Попробуем собрать и запустить программу (рис. 2.10):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora]:~/work/study/2024-2025/Архитекту... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура ко... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
3
2
1
0
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.10: Повторная сборка программы из файла lab8-1.asm и её запуск

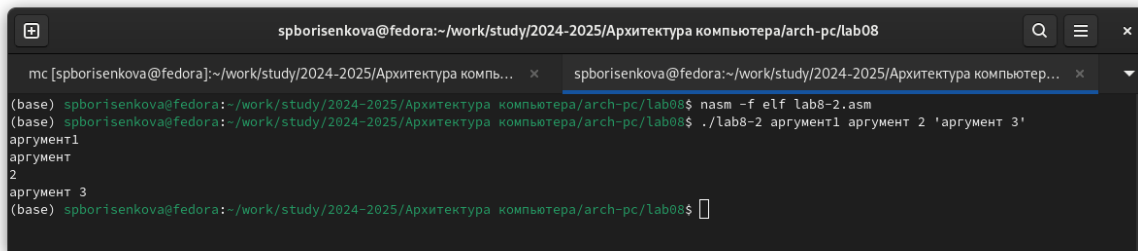
Теперь, программа выводит все числа от N-1 до нуля. Таким образом, число прогонов цикла равно числу N. Создадим второй файл и вставим в него код из файла листинга 8.2 (рис. 2.11):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08 — nano lab8-2.asm
mc [spborisenkova@fedora]:~/work/study/2024-2025/Архитекту... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура ко... x
GNU nano 7.2 lab8-2.asm Modified
#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. 2.11: Запись кода из листинга 8.2 в файл lab8-2.asm

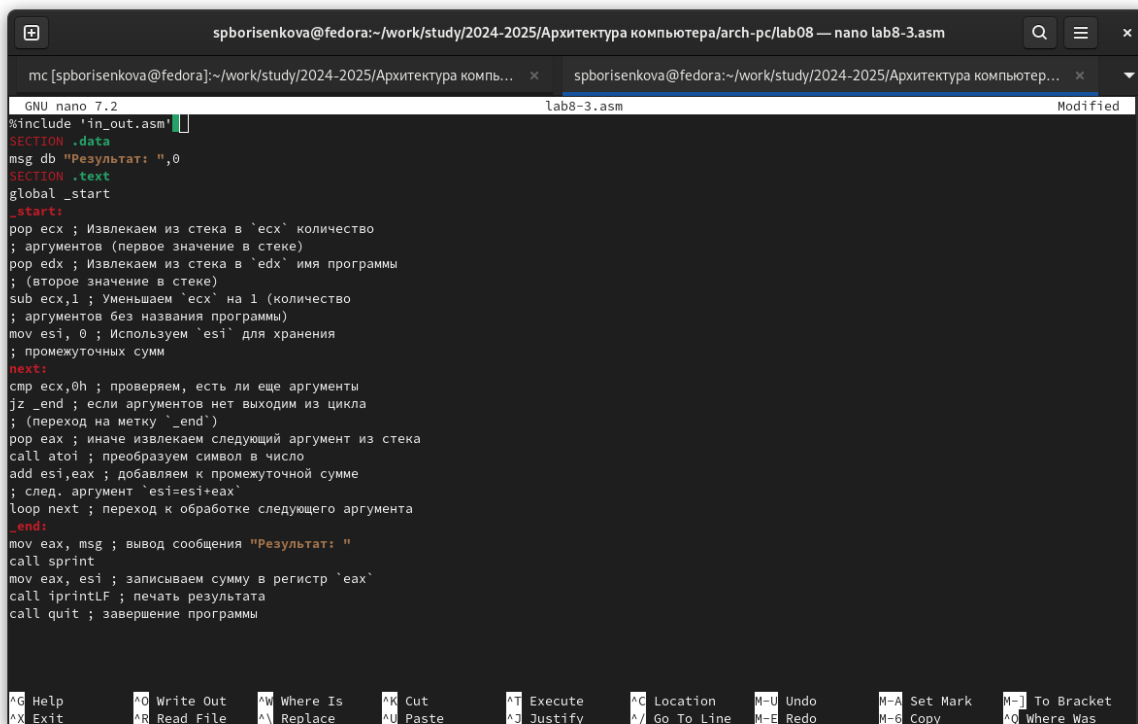
Соберём и запустим его. Посмотрим на результат (рис. 2.12):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora:~/work/study/2024-2025/Архитектура комп... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура компютер... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-2.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.12: Сборка программы из файла lab8-2.asm и её запуск

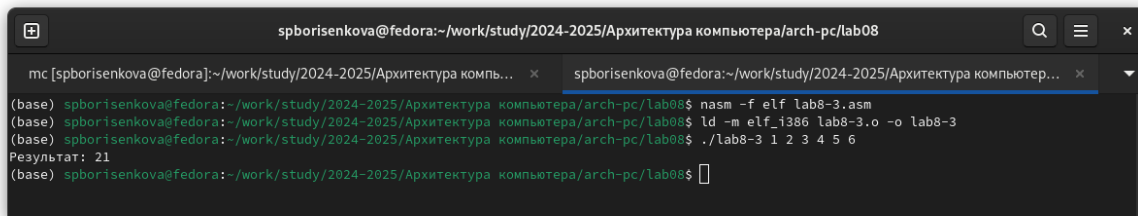
Как видим, он обработал 4 аргумента. Аргументы разделяются пробелом, либо, когда аргумент содержит в себе пробел, обрамляется в кавычки. Создадим третий файл и вставим в него код из листинга 8.3. Он будет находить сумму всех аргументов (рис. 2.13):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08 — nano lab8-3.asm
mc [spborisenkova@fedora:~/work/study/2024-2025/Архитектура комп... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура компютер... x
GNU nano 7.2 lab8-3.asm Modified
%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 iprintf ; печать результата
    call quit ; завершение программы
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   ^U Undo       ^A Set Mark   ^I To Bracket
^X Exit      ^R Read File  ^N Replace    ^V Paste      ^J Justify    ^_ Go To Line  ^E Redo       ^G Copy       ^Q Where Was
```

Рис. 2.13: Запись кода из листинга 8.3 в файл lab8-3.asm

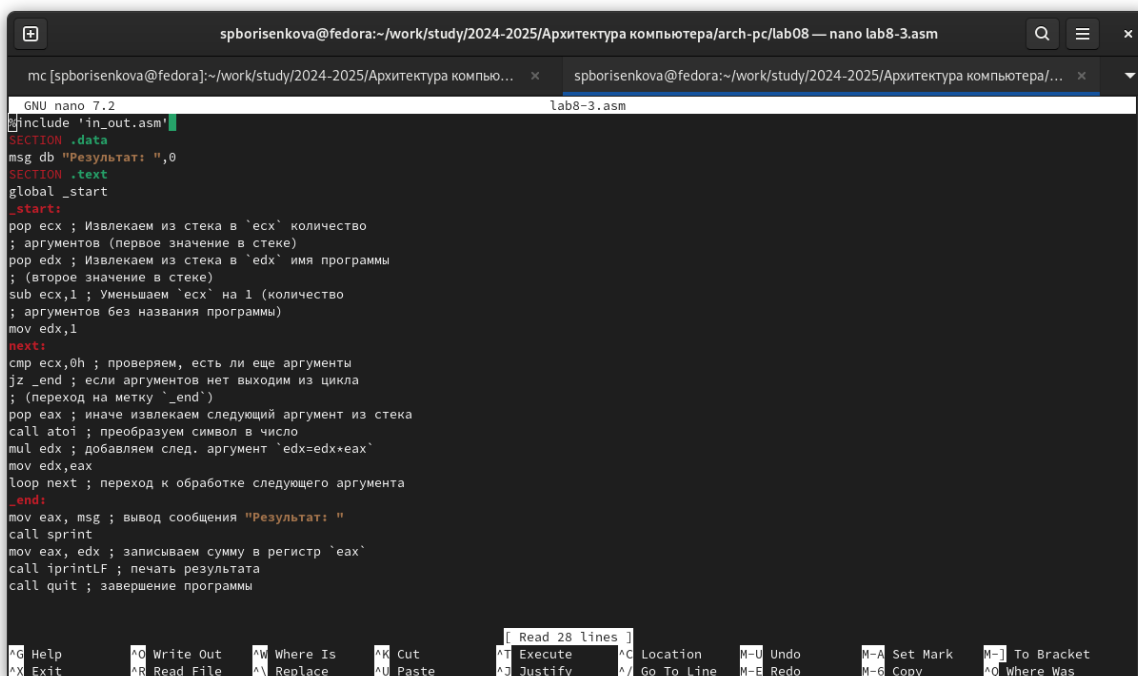
Теперь соберём программу и запустим её (рис. 2.14):



```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora:~/work/study/2024-2025/Архитектура комп... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютер... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-3.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3 4 5 6
Результат: 21
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.14: Сборка программы из файла lab8-3.asm и её запуск

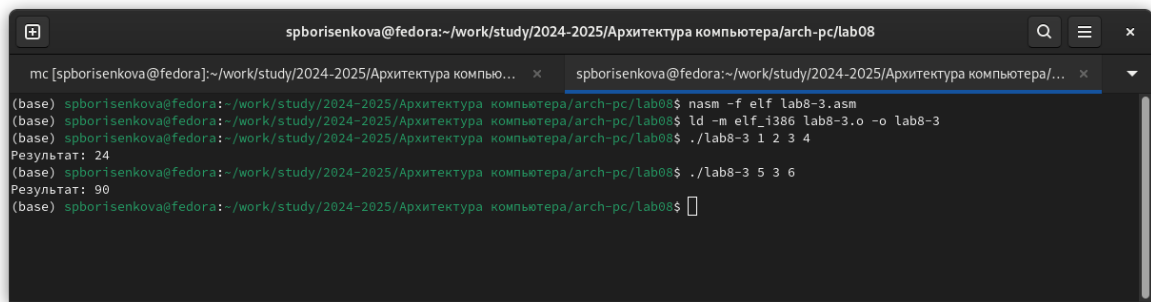
Как видим, программа действительно выводит сумму всех аргументов. Изменим её так, чтобы она находила не сумму, а произведение всех аргументов (рис. 2.15):



```
GNU nano 7.2 lab8-3.asm
#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 edx,1
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 edx ; добавляем след. аргумент 'edx=edx*eax'
    mov edx,eax
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax,msg ; вывод сообщения "Результат: "
    call sprint
    mov eax,edx ; записываем сумму в регистр 'eax'
    call iprintLF ; печать результата
    call quit ; завершение программы
```

Рис. 2.15: Изменение файла lab8-3.asm

Соберём программу и запустим её (рис. 2.16):



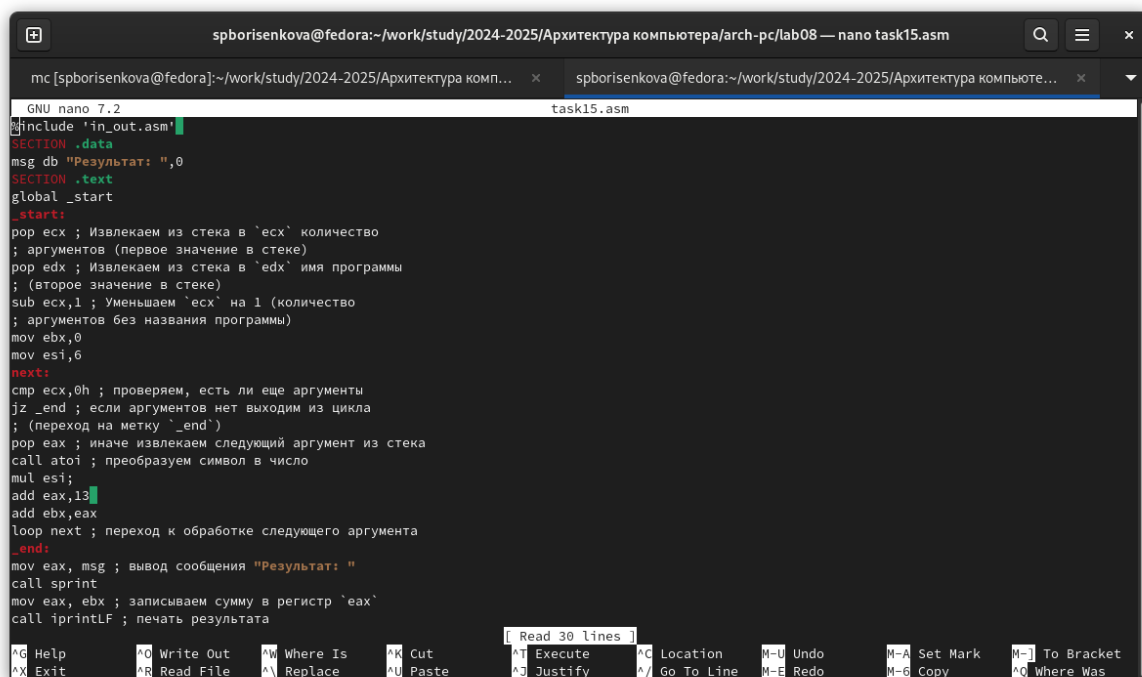
```
spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08
mc [spborisenkova@fedora]:~/work/study/2024-2025/Архитектура компью... x spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/... x
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-3.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 5 3 6
Результат: 90
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 2.16: Повторная сборка программы из файла lab8-3.asm и её запуск

Как видим, программа выводит правильный ответ

### 3 Выполнение задания для самостоятельной работы

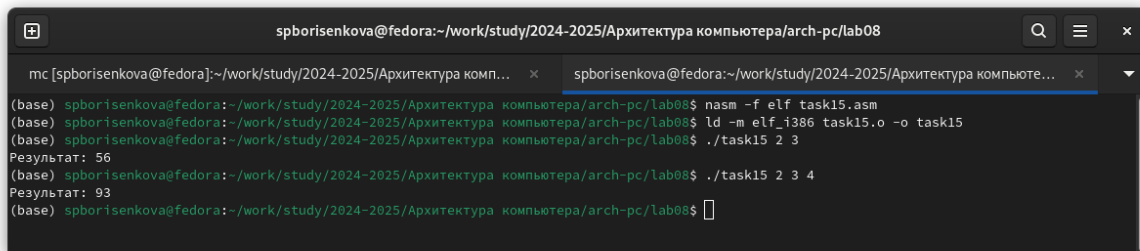
Для выполнения самостоятельной работы создадим файл task15.asm В рамках самостоятельной работы необходимо сделать задание под вариантом 15. Необходимо сложить результаты выполнения функции  $f(x)=6x+13$  для всех введённых аргументов (рис. 3.1):



```
GNU nano 7.2 task15.asm
#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 ebx,0
    mov esi,6
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;
    add eax,13
    add ebx,eax
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax,msg ; вывод сообщения "Результат: "
    call sprint
    mov eax,ebx ; записываем сумму в регистр `eax`
    call iprintf ; печать результата
```

Рис. 3.1: Код файла самостоятельной работы

Соберём и запустим программу, вводя различные аргументы (рис. 3.2):

A terminal window with a dark background and light text. The window title is "spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08". The terminal shows the following commands and output:

```
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf task15.asm
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 task15.o -o task15
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./task15 2 3
Результат: 56
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./task15 2 3 4
Результат: 93
(base) spborisenkova@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 3.2: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Пересчитав результат вручную, убеждаемся, что программа работает верно



## 4 Выводы

В результате выполнения лабораторной работы были получены навыки работы с циклами и обработкой аргументов из командной строки. Были написаны программы, использующие все вышеописанные аспекты.