Отчёт по лабораторной работе №8

дисциплина: Архитектура компьютера

Аносов Даниил Игоревич

Содержание

# 1 Цель работы

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

# 2 Задание

1. Напишите программу, которая находит сумму значений функции для , т.е. программа должна выводить значение . Значения передаются как аргументы. Вид функции выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах .

# 3 Выполнение лабораторной работы

## 3.1 Реализация циклов в NASM

Откроем терминал и создадим каталог для программ лабораторной работы №8. В новом каталоге создадим файл для первой программы *lab8-1.asm*. (рис. 1).

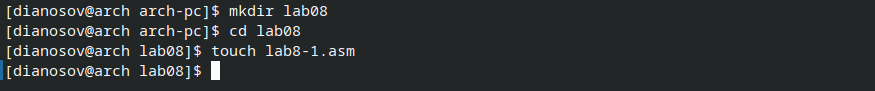


Рис. 1: Создание каталога для программ

Введём в этот файл текст программы из предложенного листинга. (рис. 2).

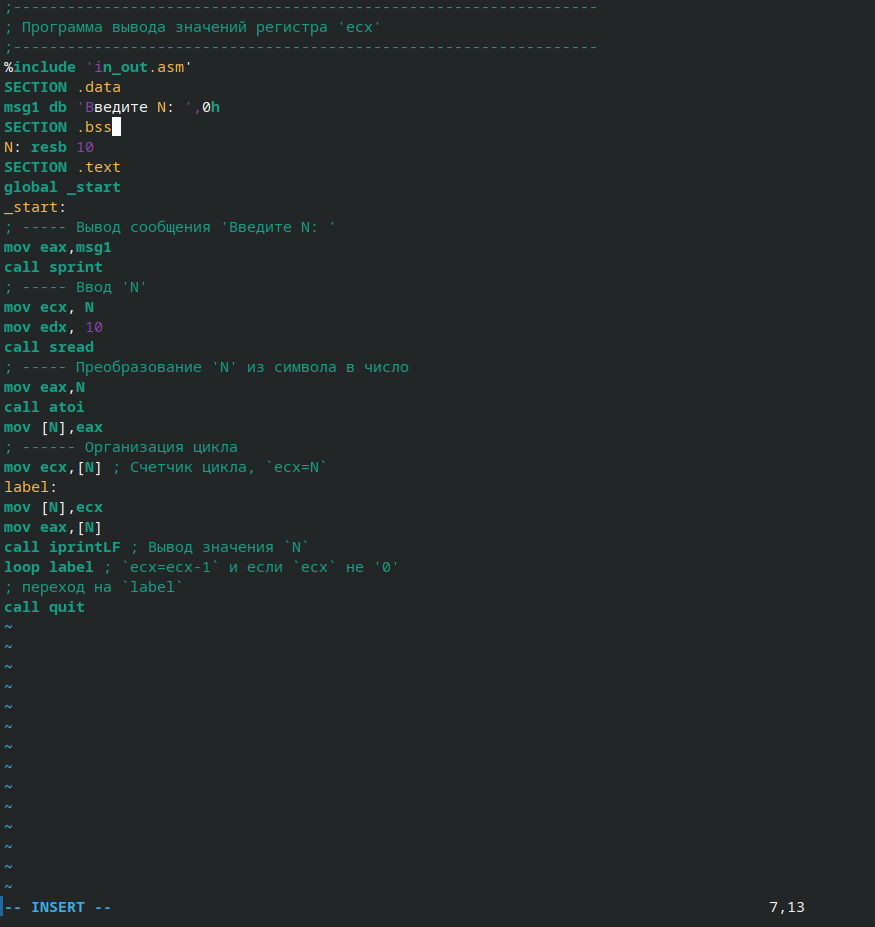


Рис. 2: Открытый Vim

Скомпилируем и запустим программу, предварительно скопировав из каталога предыдущей лабораторной работы вспомогательный файл с подпрограммами *in\_out.asm* (рис. 3).

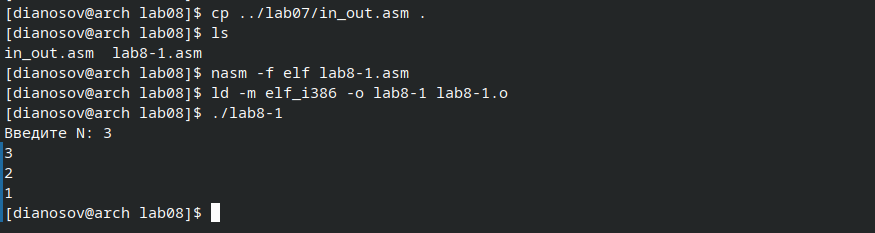


Рис. 3: Компиляция и первый запуск программы

Изменим текст программы, добавив изменение значения регистра ecx в цикле (рис. 4).

label:  
sub ecx,1 ; `ecx=ecx-1`  
mov [N],ecx  
mov eax,[N]  
call iprintLF  
loop label

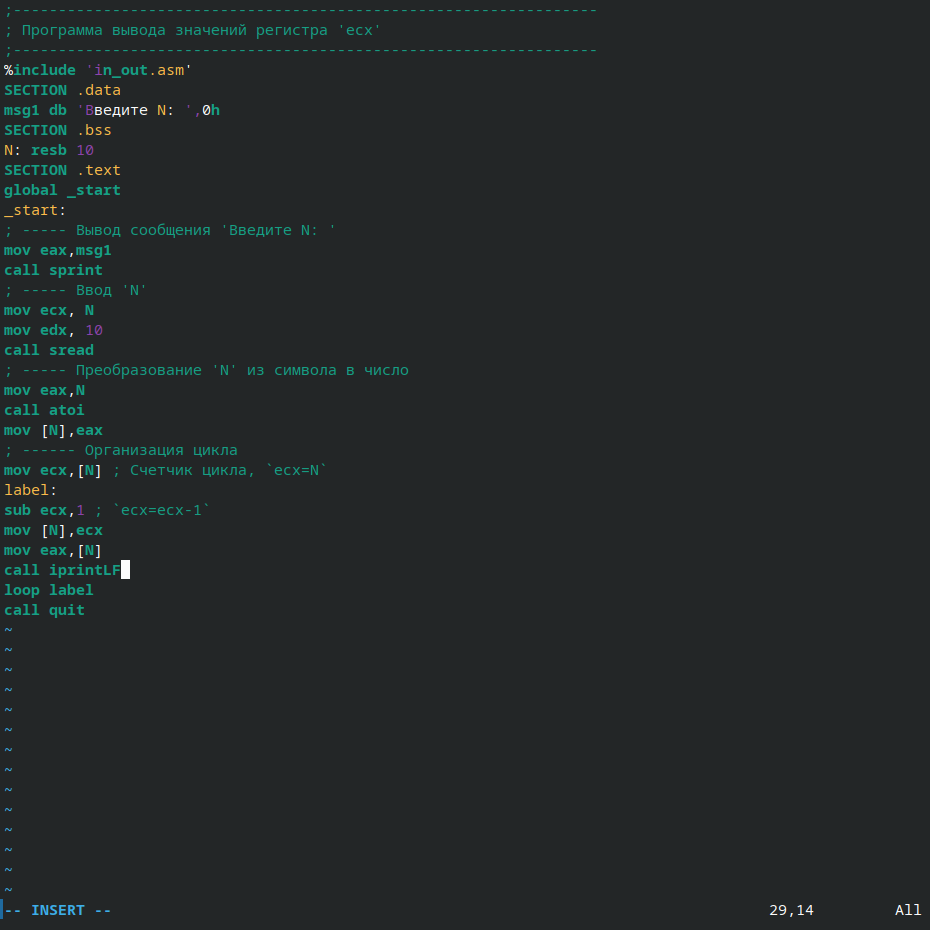


Рис. 4: Vim с обновленной программой

Скомпилируем и запустим измененную программу. Проверим её работу. (рис. 5).

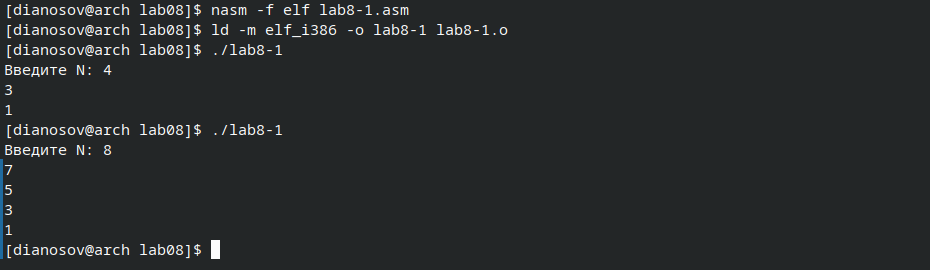


Рис. 5: Повторная компиляция и запуск программы

Какие значения принимает регистр ecx в цикле? Соответствует ли число проходов цикла значению , введенному с клавиатуры?

Регистр ecx принимает значения с шагом 2, начиная с . Поэтому, число проходов цикла равно , что меньше .

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

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

Откроем файл программы в **Vim** и отредактируем код (рис. 6).

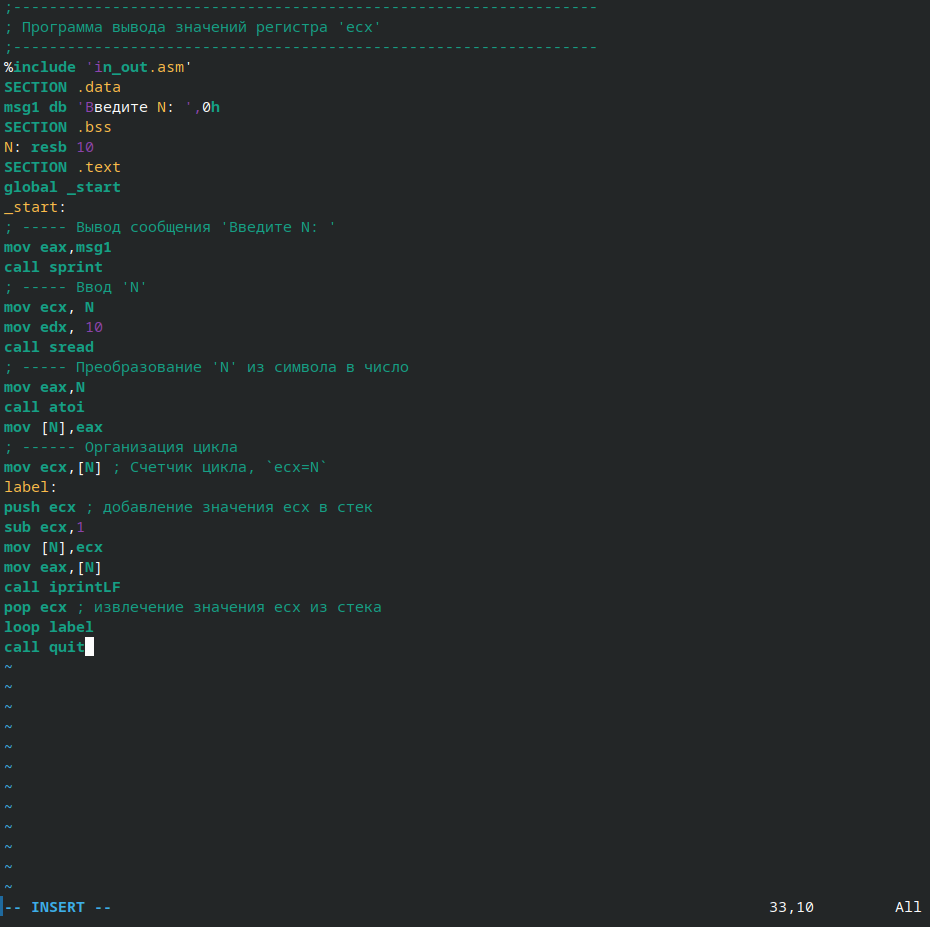


Рис. 6: Vim с обновленной программой

Проведём компиляцию программы с изменениями, проверим, как она работает (рис. 7).

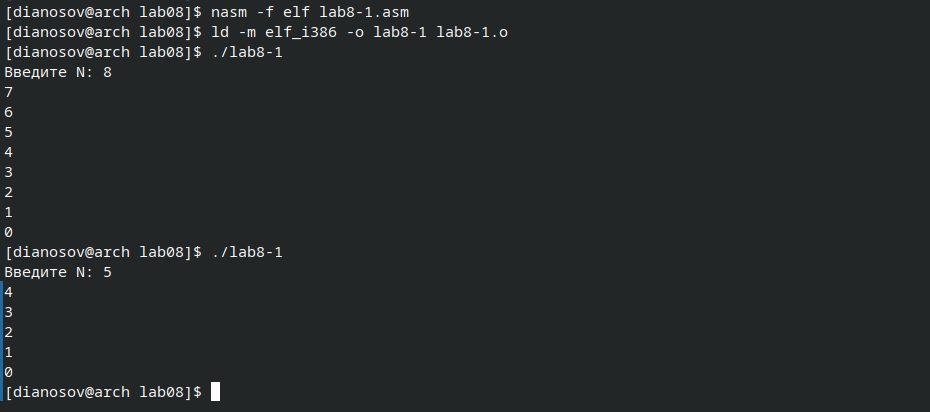


Рис. 7: Компиляция и запуск новой программы

Теперь, после того, как мы начали сохранять значения итератора ecx в стек, цикл стал работать корректно. Количество проходов цикла соответствует введённому .

## 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. В качестве примера такой программы рассмотрим программу, предложенную в листинге. Создадим для неё новый файл *lab8-2.asm* и откроем его в редакторе **Vim**.

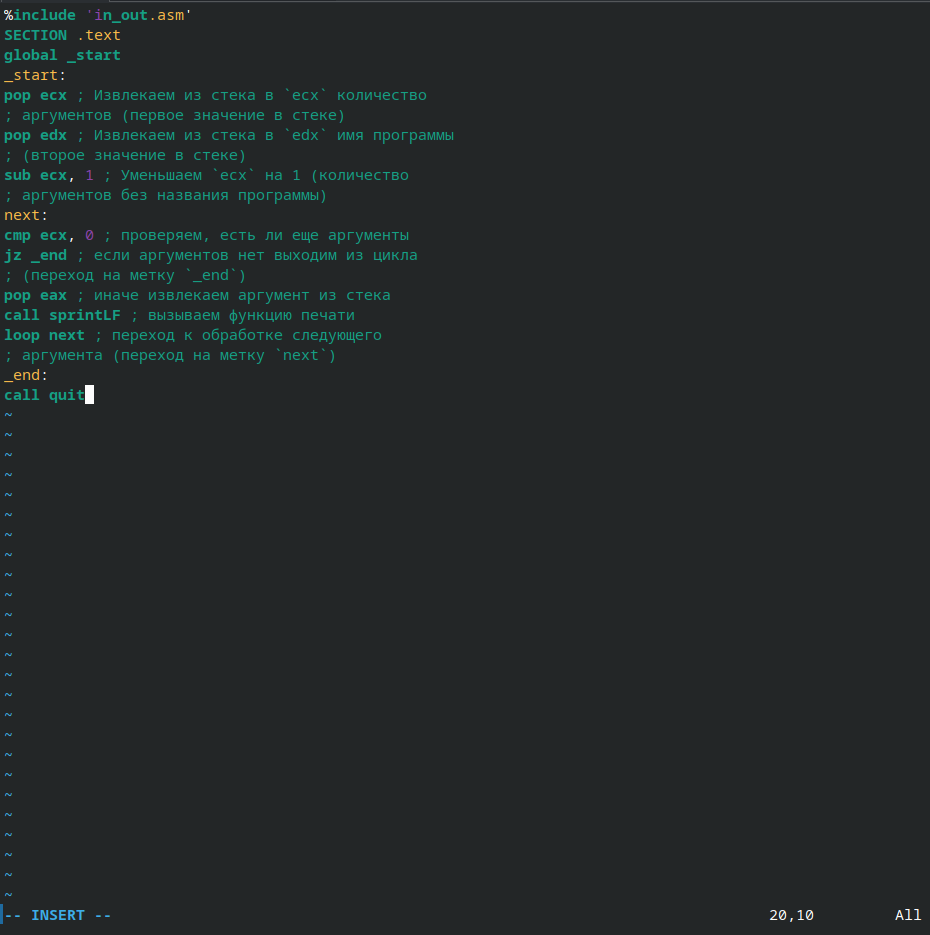


Рис. 8: Редактирование файла *lab8-2.asm*

Требуется проверить работу программы. Скомпилируем её и запустим исполняемый файл, указав аргументы в командной строке (рис. 9).

./lab8-2 аргумент1 аргумент 2 'аргумент 3'

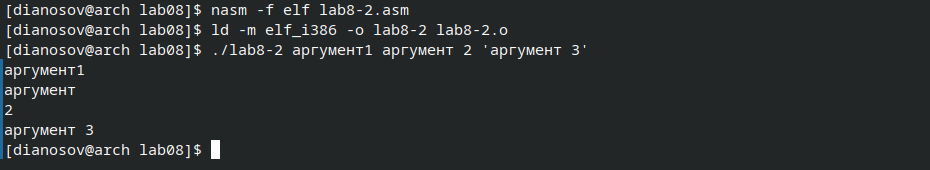


Рис. 9: Повторная компиляция и запуск новой программы

Программа обработала не 3 аргумента, а 4, так как слова, разделённые пробелом, написанные без кавычек (аргумент 2) обрабатываются как отдельные.

### 3.2.1 Вычисление суммы аргументов

Теперь создадим новый файл под названием *lab8-3.asm*. Введём в него код программы, вычисляющей сумму аргументов, переданных в командной строке. Файл откроем редактором **Vim** (рис. 10).

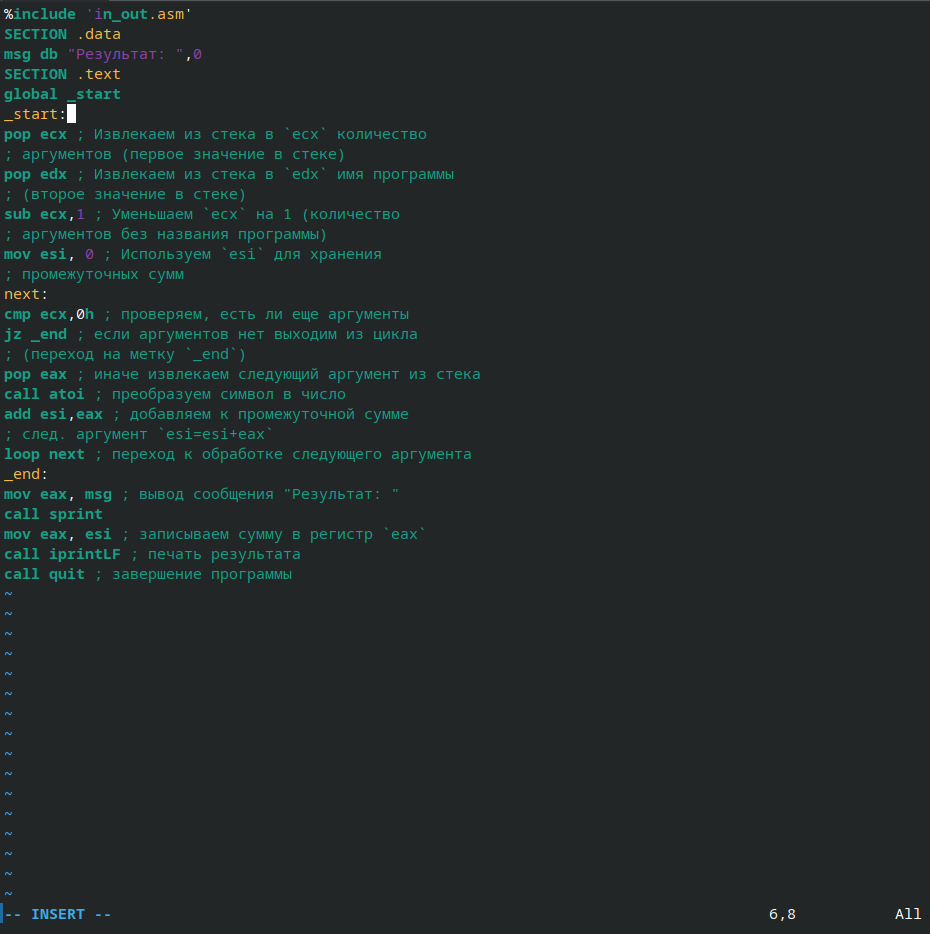


Рис. 10: **Vim** с файлом *lab8-3.asm*

Теперь скомпилируем программу и проверим её работу (рис. 11).

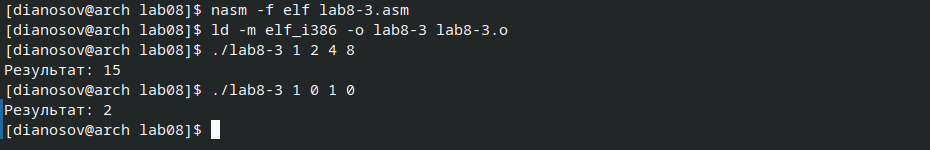


Рис. 11: Компиляция и запуск *lab8-3.asm*

### 3.2.2 Вычисление произведения аргументов

Теперь изменим код так, чтобы программа выводила не сумму, а произведение аргументов командной строки. Для этого вместо

pop eax  
call atoi  
add esi, eax

напишем

pop eax ; eax := следующий аргумент  
call atoi  
mov ebx, eax ; ebx := eax  
mov eax, esi ; eax := esi  
mul ebx ; eax := eax \* ebx  
mov esi, eax ; esi := eax = eax \* ebx = esi \* ebx

Также, изменим изначальное значенике esi с 0 на 1 (нейтральный элемент по умножению, а не по сложению).

Сделаем изменения в файле программы и скомплируем его. Проверим работу. (рис. 12, 13).

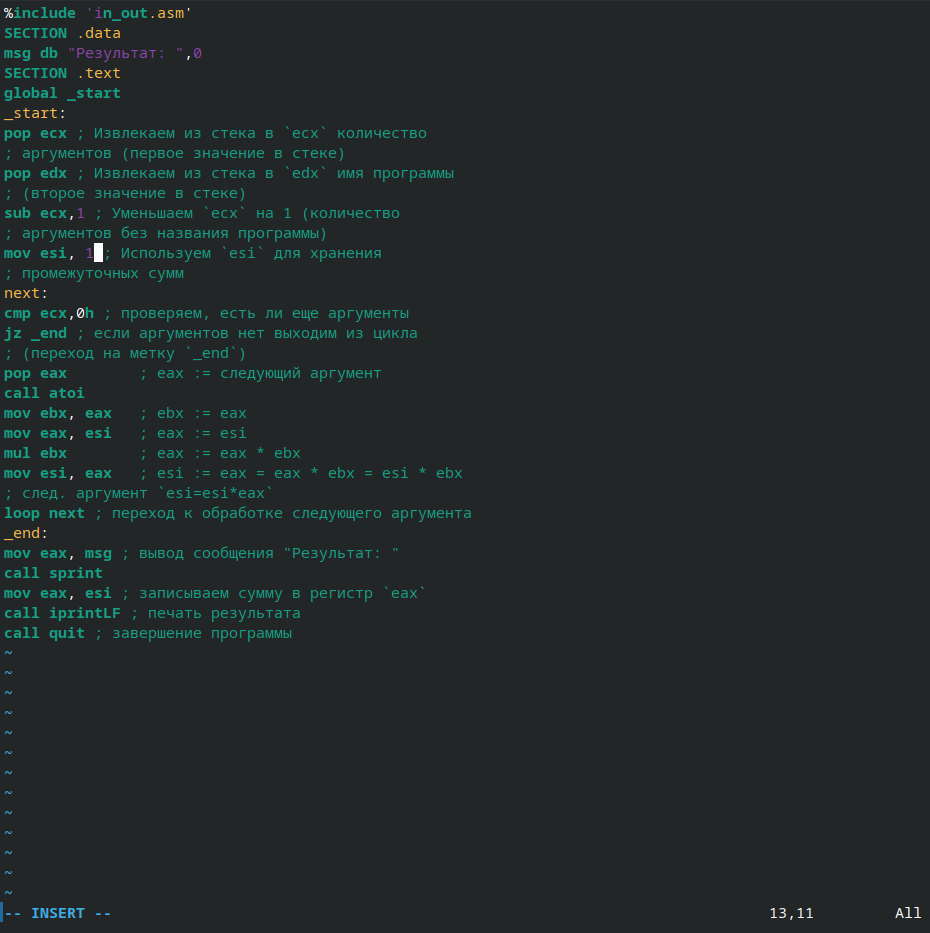


Рис. 12: **Vim** с измененным файлом *lab8-3.asm*

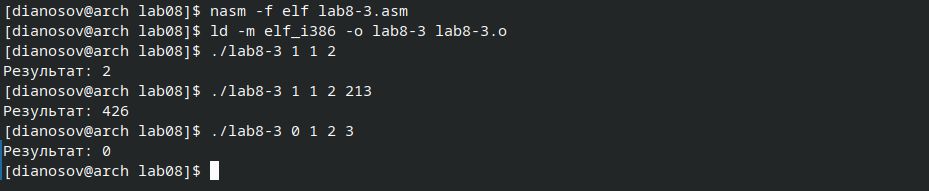


Рис. 13: Компиляция и запуск *lab8-3.asm*

Как видно, программа правильно выводит произведение переданных в командной строке аргументов.

# 4 Задание для самостоятельной работы

Для начала, нужно выбрать вид функции из таблицы. Варианту №6 соответствует функция $\\f(x) = 4x-3$

Создадим для выполнения задания файл *task.asm*, откроем его в редакторе **Vim** (рис. 14)

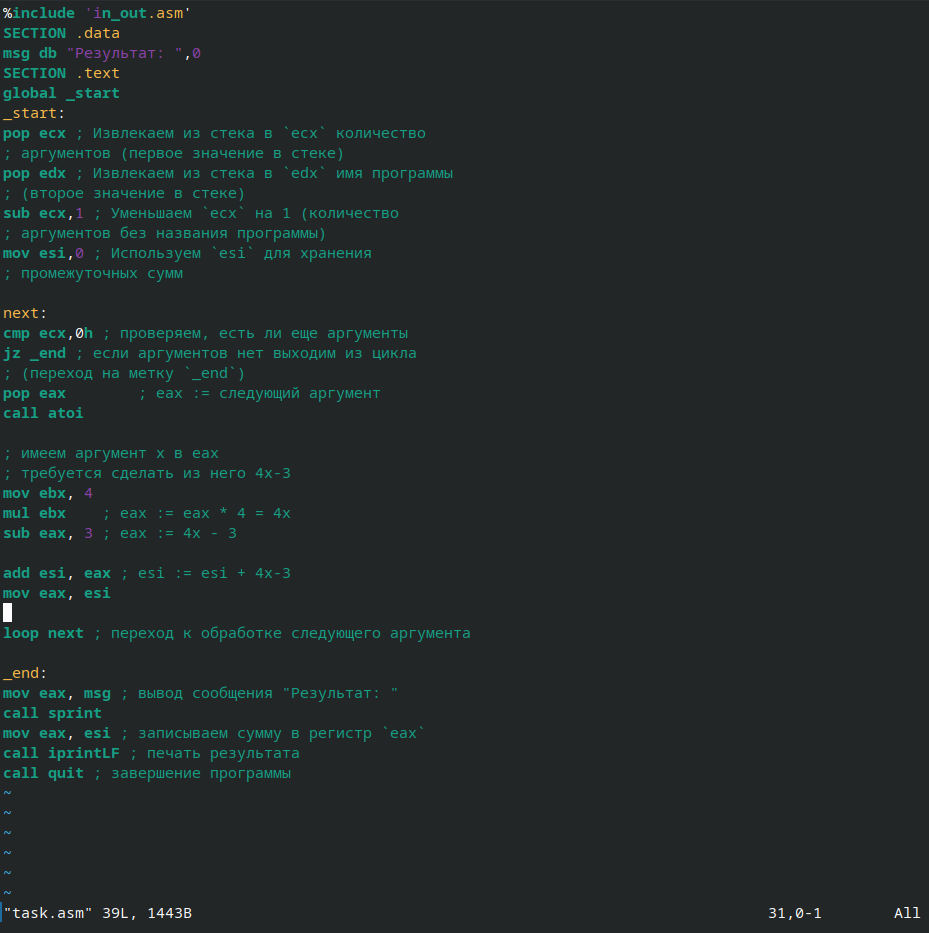


Рис. 14: **Vim** с файлом *task.asm*

Программа аналогична одной из уже рассмотренных в этой лабораторной работе. Теперь она суммирует (прибавляет к регистру esi) не сами аргументы (), а значения функции, соответствующие им: .

Протестируем работу программы (рис. 15).

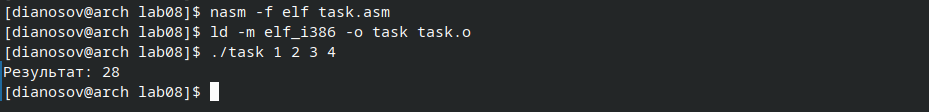


Рис. 15: Проверка работы *task.asm*

Выводится верный результат: $f(1)+f(2)+f(3)+f(4)=\\=4\times1-3+4\times2-3+4\times3-3+4\times4-3=\\=40-12=28$

Теперь сделаем так, чтобы программа выводила в начале строку Откроем программу в **Vim** и проведём необходимые изменения (рис. 16). В секцию констант .data добавим нужное сообщение, а в самом начале секции .text выведем его, используя функцию sprintLF.

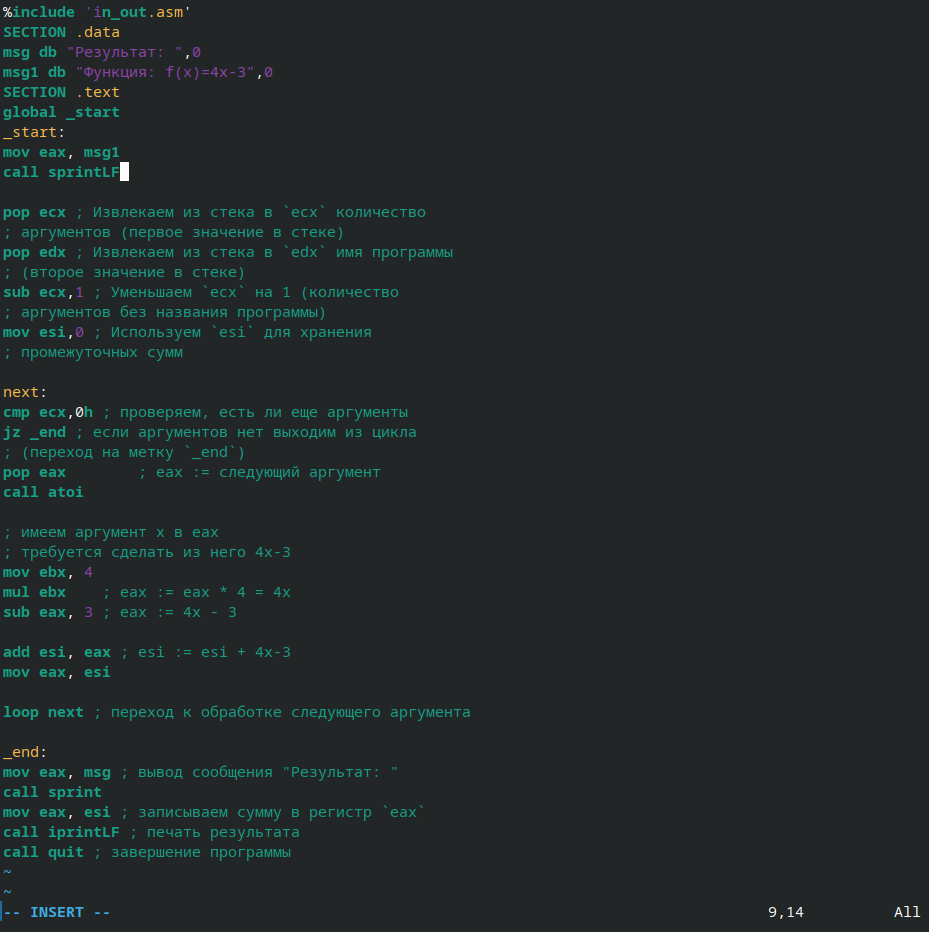


Рис. 16: **Vim** с обновленным файлом *task.asm*

Проверим же работу измененной программы (рис. 17).

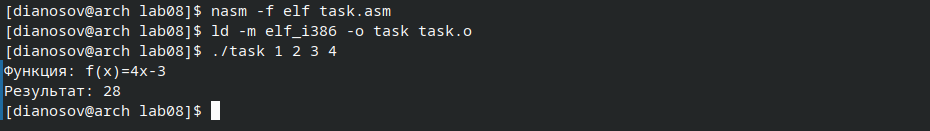


Рис. 17: Тестирование программы *task.asm*

Программа работает корректно. Задание выполнено.

Загрузим файлы на GitHub (рис. 18).

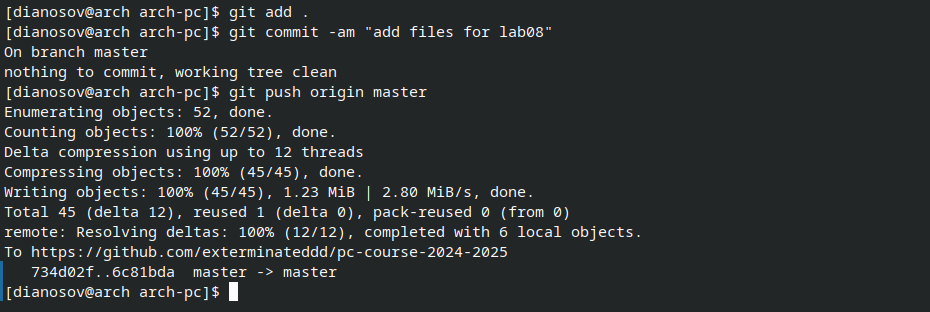


Рис. 18: Загрузка файлов на GitHub

# 5 Выводы

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