Лабораторная работа №8

Дисциплина: Архитектура компьютера

Савостин Олег

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

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

# 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Выполнение заданий лабораторной работы.

# 3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек

Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ах, сх, dx, bх, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти

Аналогично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл.

# 4 Выполнение лабораторной работы

## 4.1 Реализация циклов в NASM.

Сперва, создаю каталог и файл для лабораторной работы.(рис. 1).

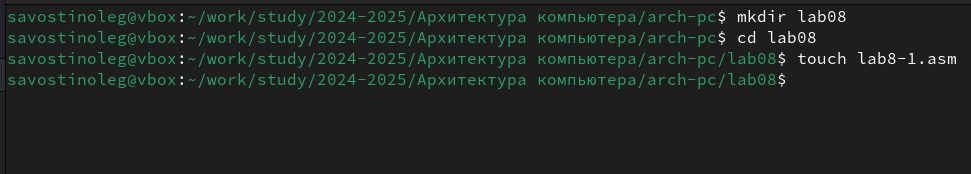


Рис. 1: Новый каталог и файл.

Вставляю в файл текст из Листинга 8.1 (рис. 2).

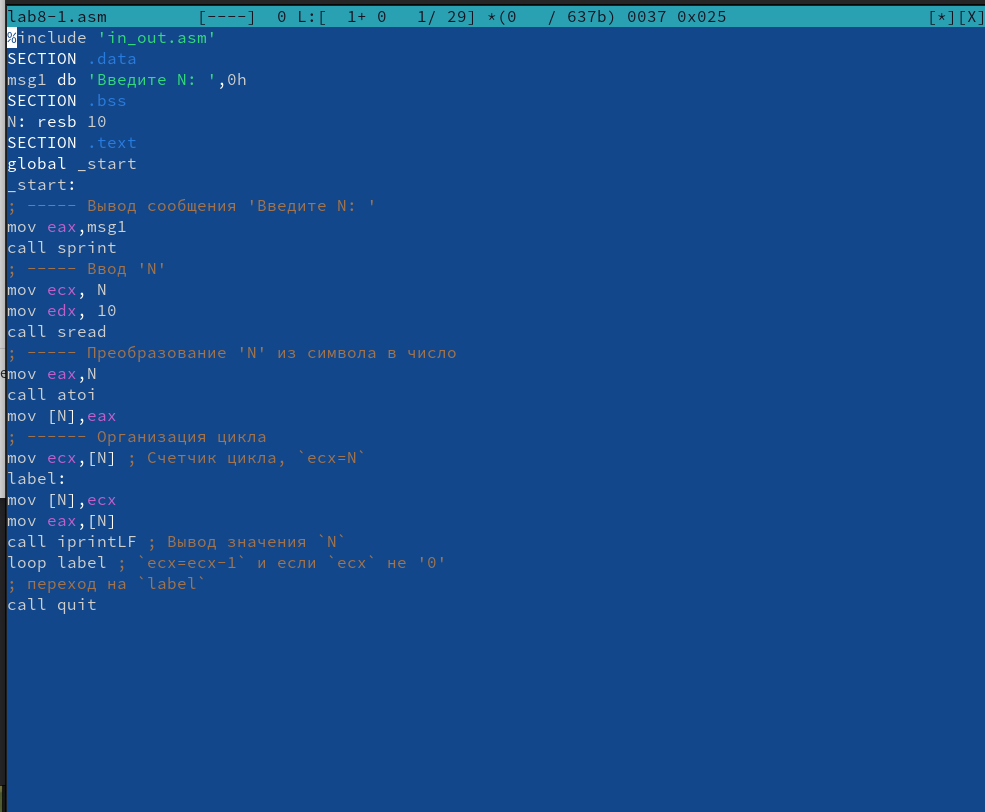


Рис. 2: Текст из Листинга 8.1

Создаю исполняемый файл и проверяю его работу. Он считывает числa с N до 1 (рис. 3).

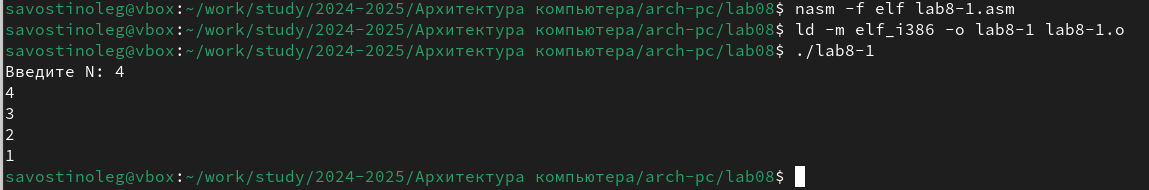


Рис. 3: Исполняемый файл Листинг 8.1

Изменяю текст файла, добавив изменение значение регистра еах в цикле (рис. 4).

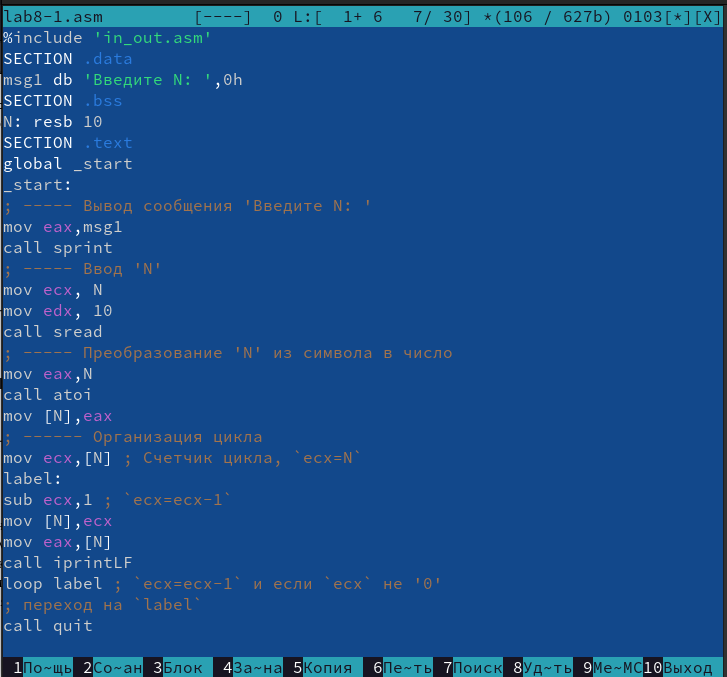


Рис. 4: Изменение текста кода

Создаю исполняемый файл. При запуске исполняемого файла, код считывает с N до 0, но затем начинается счет с 4294967294.(рис. 5).

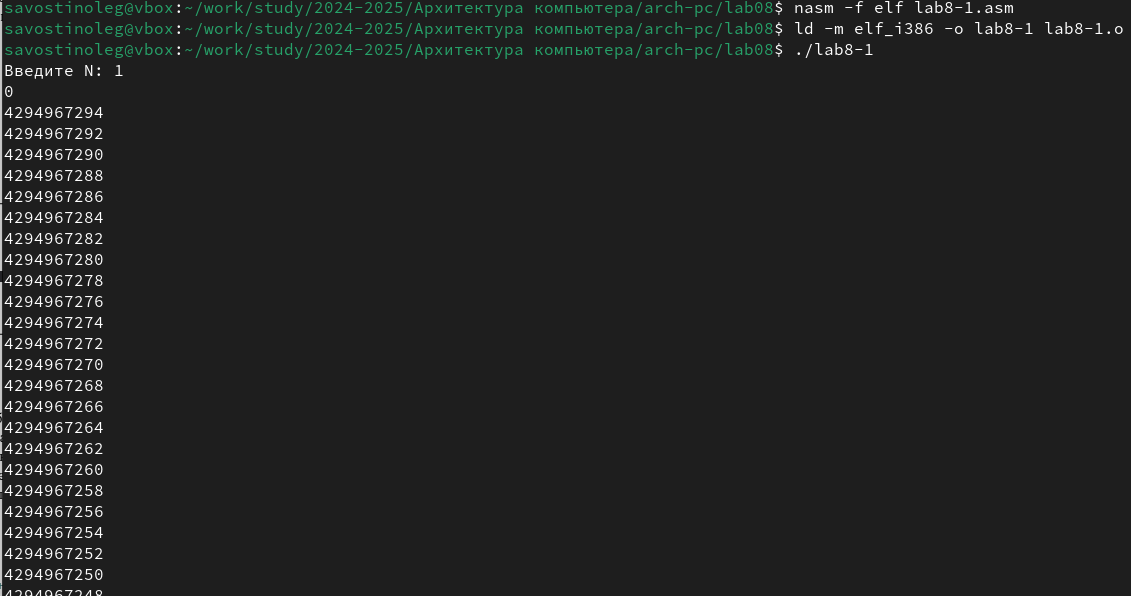


Рис. 5: Создание и запуск измененного исполняемого файла

Чтобы сохранить корректность программы, использую команду рор (рис. 6).

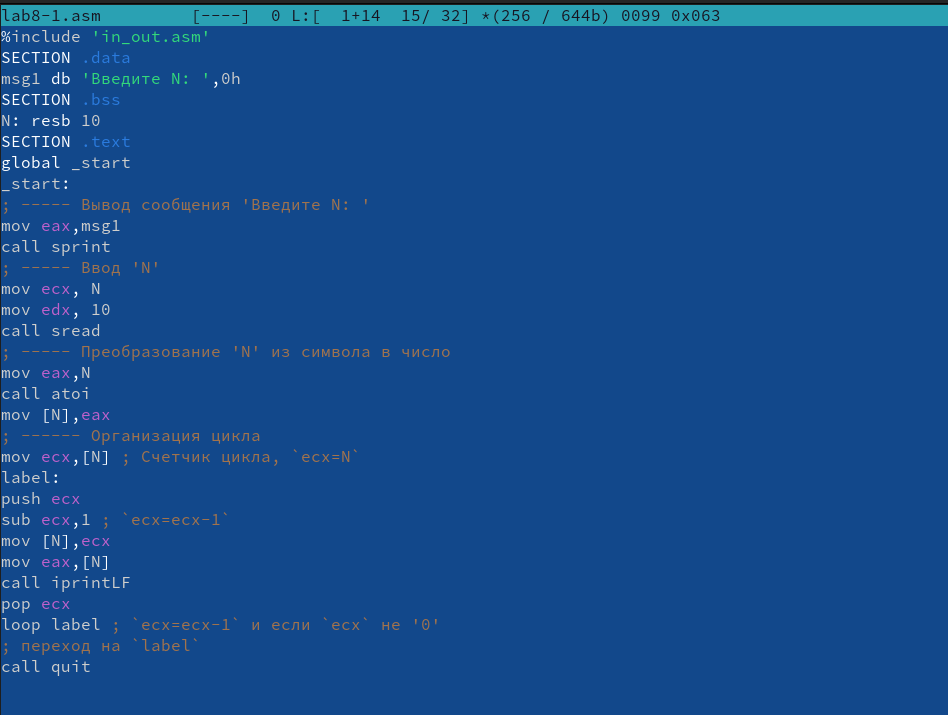


Рис. 6: Добавление pop ecx

Создаю исполняемый файл. Все работает корректно, однако, он теперь считает числа с N до 0 (рис. 7).

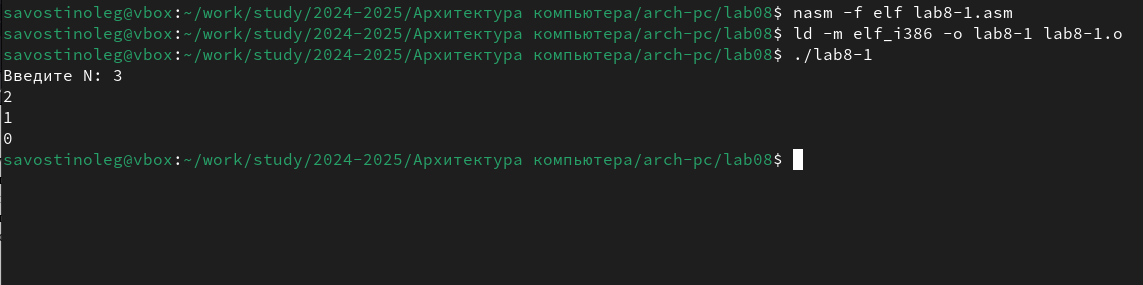


Рис. 7: Корректно работающий lab8-1.asm

## 4.2 Обработка аргументов командной строки

Создаю новый файл (рис. 8) и записываю в него текст из Листинга 8.2 (рис. 9).

Рис. 8: Создание нового файла

Рис. 8: Создание нового файла

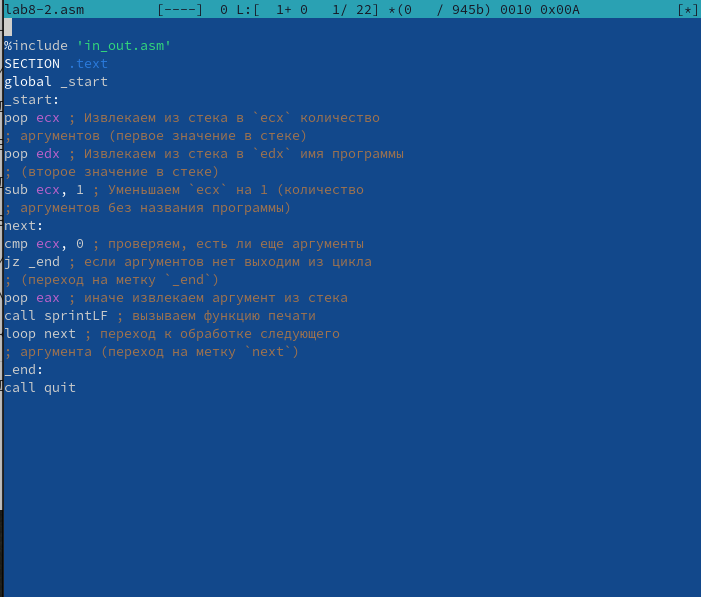


Рис. 9: Текст из Листинга 8.2 в новом файле

Теперь запускаю файл с тремя аргументами. Обработались все три, однако, аргумент 2 был выписан по другому. Это произошло, так как он засчитал этот аргумент как два разных элемента. Если записать его как третий аргумент, то он будет выведен идентично ему. (рис. 10).

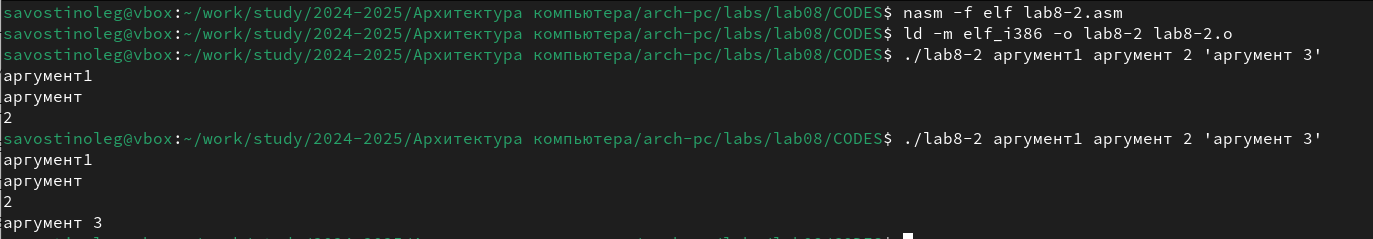


Рис. 10: Запуск исполняемого файла Листинга 8.2

Создаю третий файл(рис. 11) и ввожу в него код из Листинга 8.3(рис. 12).

Рис. 11: Создание третьего файла

Рис. 11: Создание третьего файла

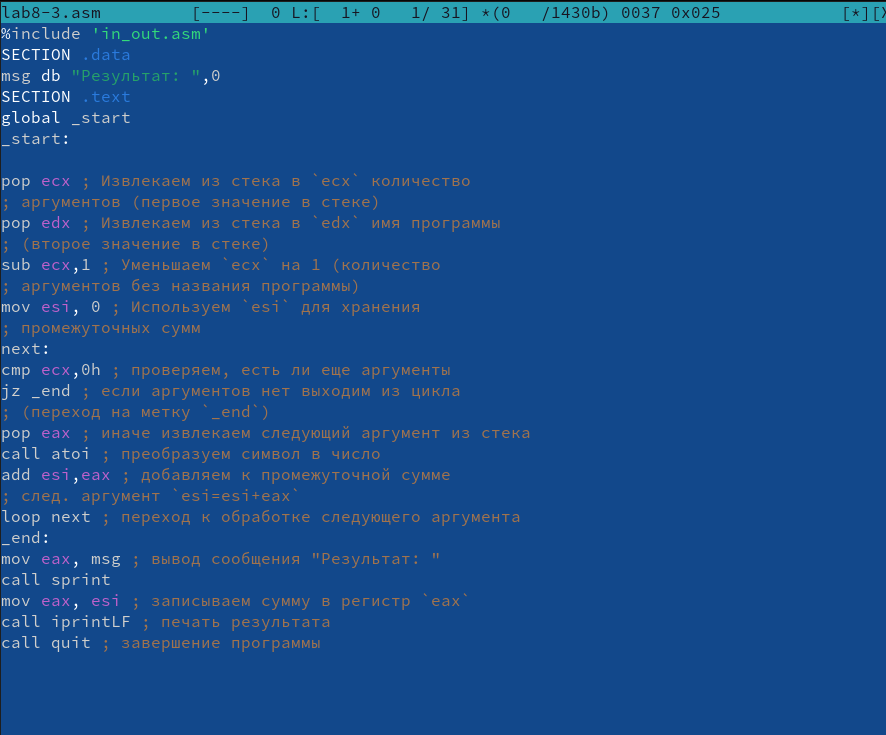


Рис. 12: Текст из Листинга 8.3

Создаю исполняемый файл и проверяю на его работу. (рис. 13).

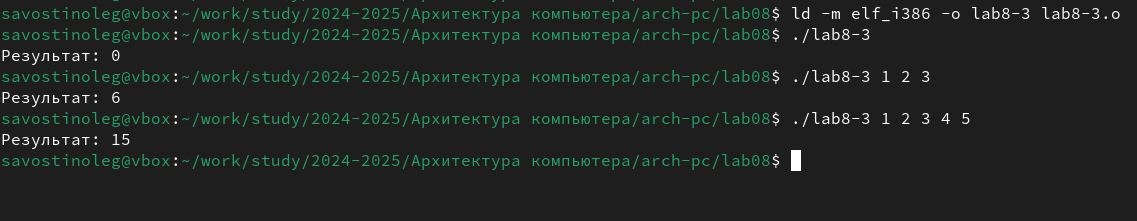


Рис. 13: Запуск третьего исполняемого файла

Изменяю код так, чтобы вместо вычисления суммы, он вычислял произведение аргументов (рис. 14).

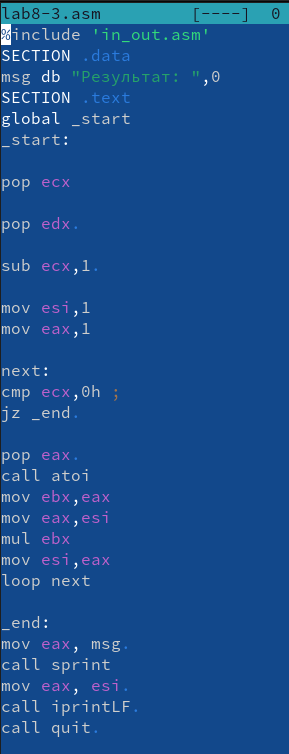


Рис. 14: Измененный файл Листинга 8.3

Создаю исполняемый файл и проверяю. Все готово(рис. 15).

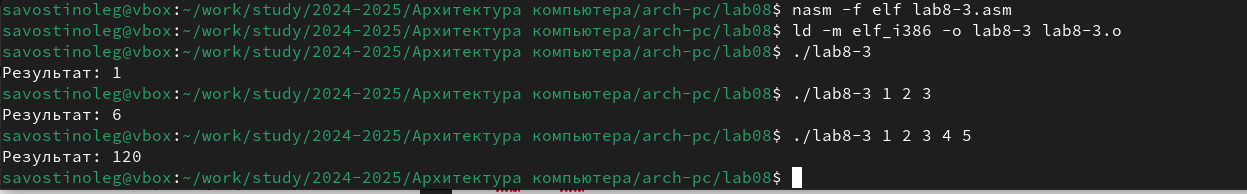


Рис. 15: Вычисление произведения аргументов

КОД

%include ‘in\_out.asm’

SECTION .data

msg db “Результат:”,0

SECTION .text

global \_start

\_start:

pop ecx

pop edx

sub ecx,1

mov esi,1

mov eax,1

next:

cmp ecx,0h ;

jz \_end

pop eax

call atoi

mov ebx,eax

mov eax,esi

mul ebx

mov esi,eax

loop next

\_end:

mov eax, msg

call sprint

mov eax, esi

call iprintLF

call quit

##Задание для самостоятельной работы

У меня 13 вариант, я сделал программу, вычисляющая сумму всех f(x)=12x-7(рис. 16).

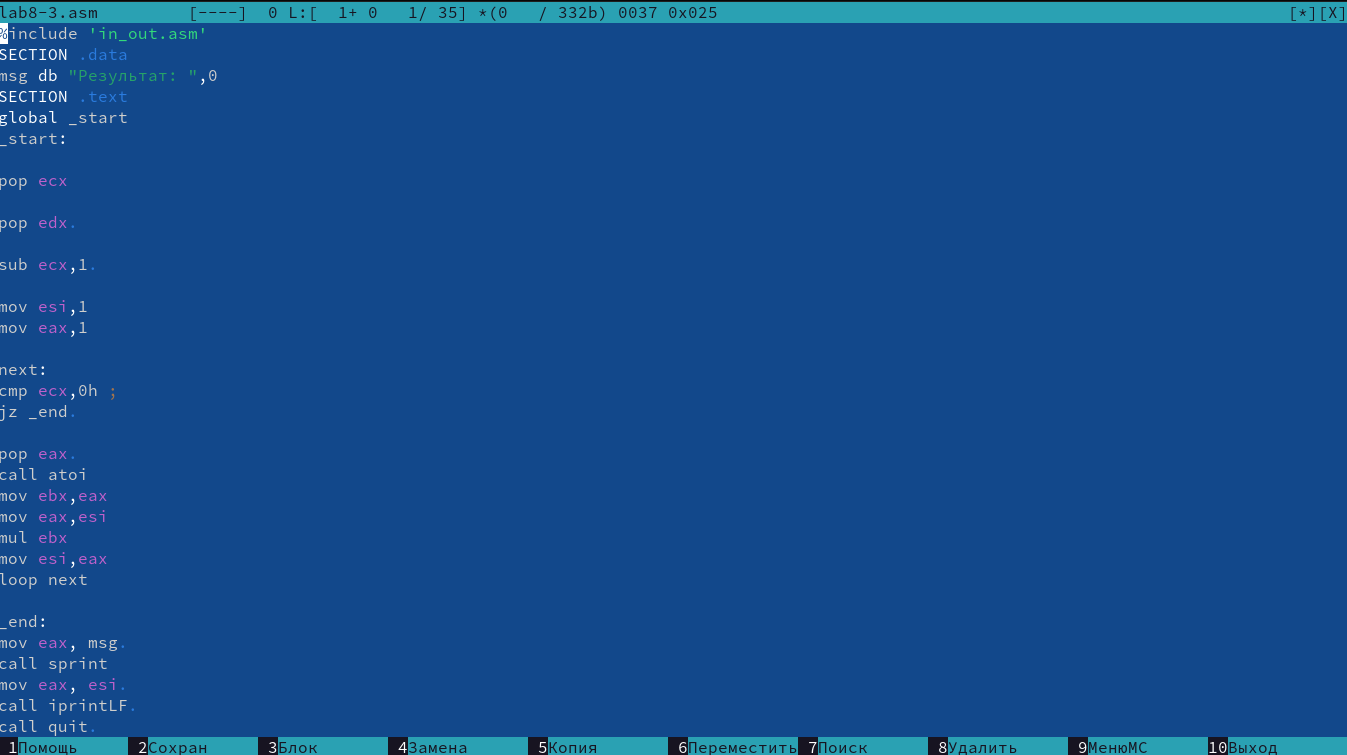


Рис. 16: Код программы

Проверяю на его правильность. Программа работает (рис. 17).

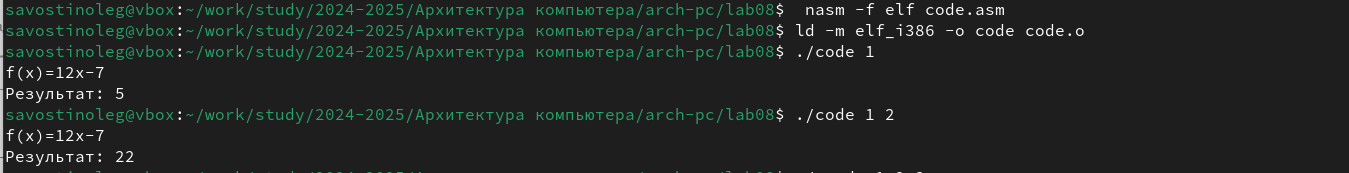


Рис. 17: Запуск исполняемого файла.

КОД

%include ‘in\_out.asm’

SECTION .data

funct db “f(x)=12x-7”,0

msg db “Результат:”,0

SECTION .text

global \_start

\_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,funct

call sprintLF

next:

cmp ecx,0h ;

jz \_end

mov ebx,12

pop eax

call atoi

mul ebx

sub eax,7

add esi,eax

loop next

\_end:

mov eax, msg

call sprint

mov eax, esi

call iprintLF

call quit

# 5 Выводы

В ходе данной работы я приобрел навыкы написания программ с использованием циклов и обработкой аргументов командной строки

# Список литературы

[Лабораторная работа №8](https://esystem.rudn.ru/pluginfile.php/2089548/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%968.%20%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D1%86%D0%B8%D0%BA%D0%BB%D0%B0.%20%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0%20%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2%20%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B9%20%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8..pdf) ::: {#refs} :::