Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Козин Иван Евгеньевич

Содержание

1 Цель работы

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

2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Самостоятельное написание программы по материалам 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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 (рис. 1).

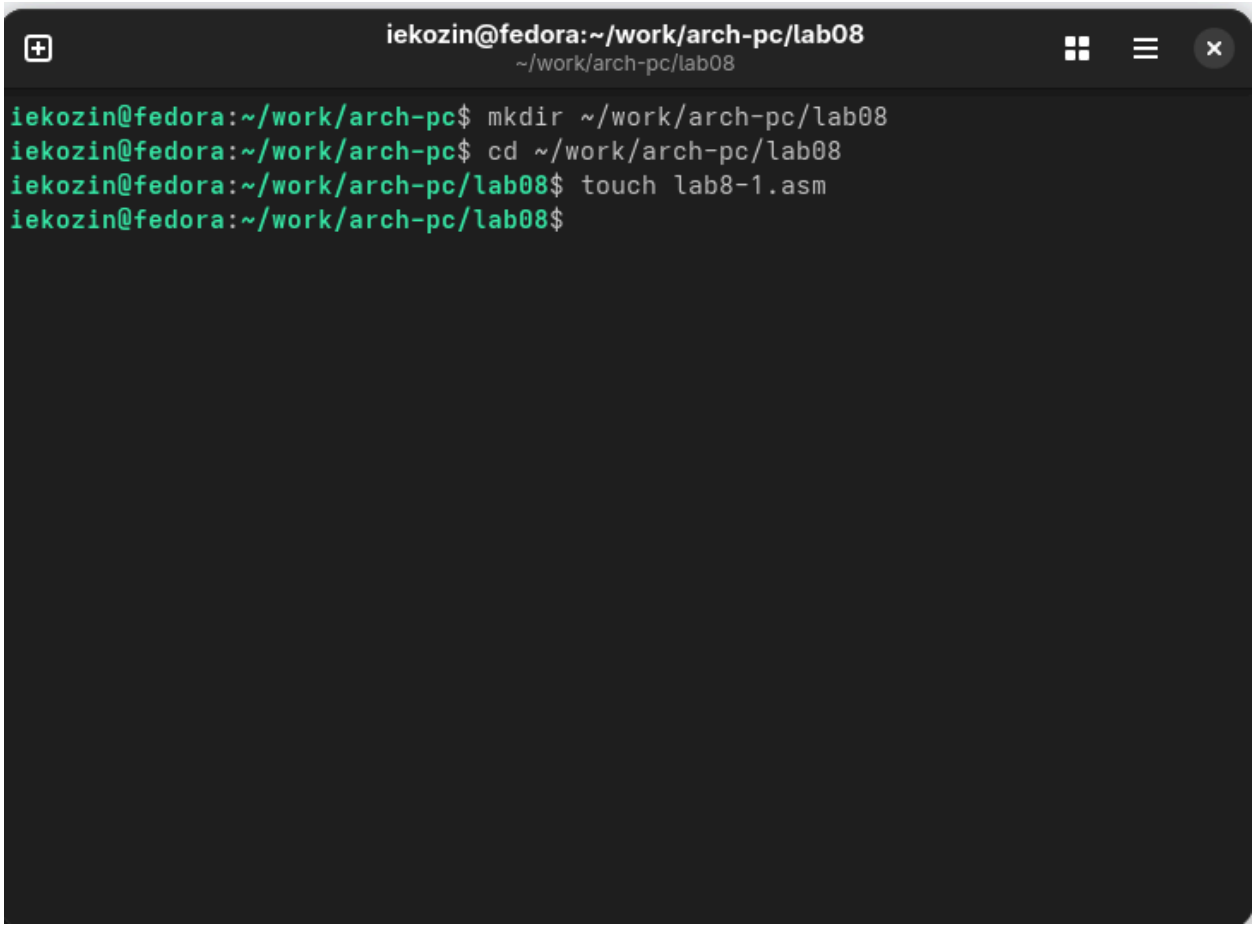

Рис. 1: Создание каталога

Копирую в созданный файл программу из листинга. (рис. 2).

Рис. 2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 3).

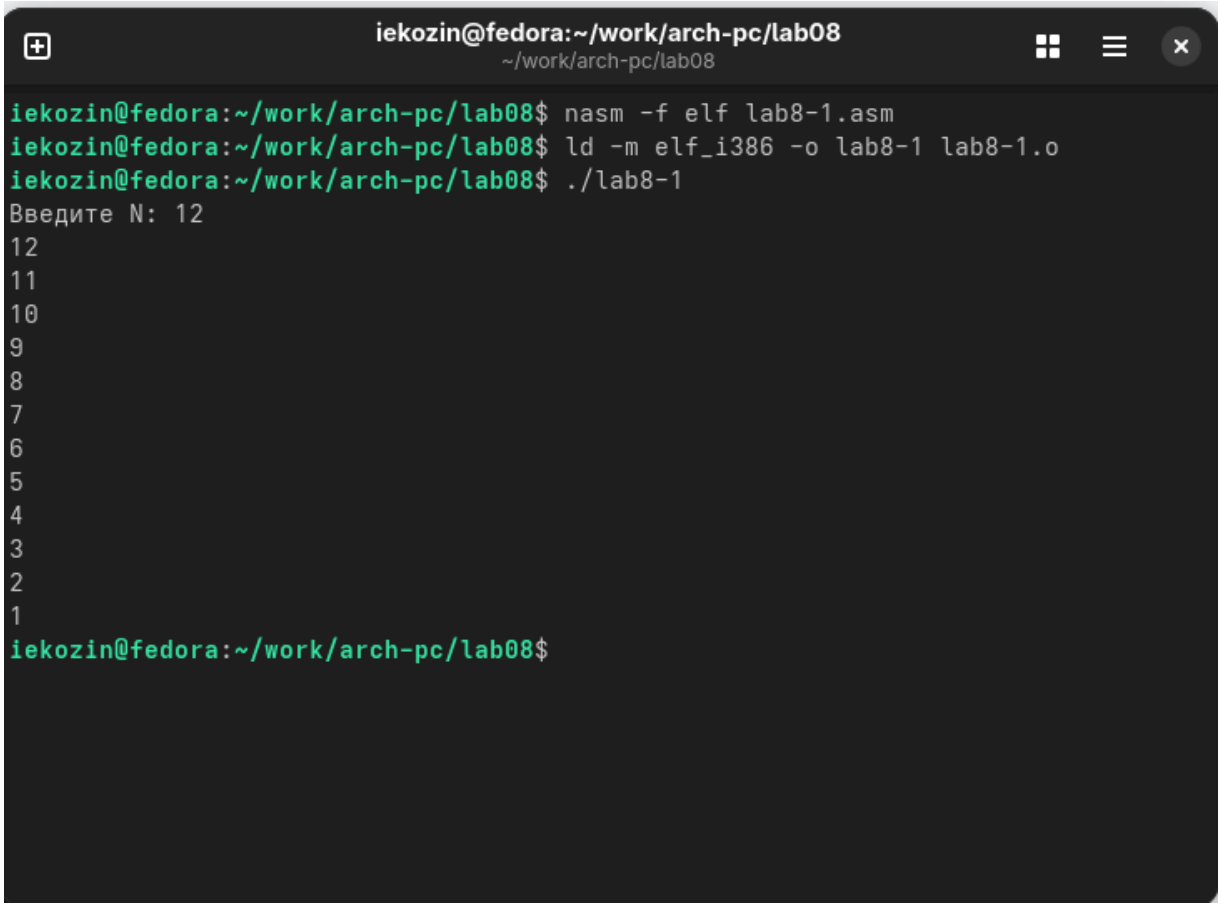

Рис. 3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра ecx (рис. 4).

```
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 ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`

call quit
```

Рис. 4: Изменение программы

Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 5).

```
iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 6).

Рис. 6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 7).

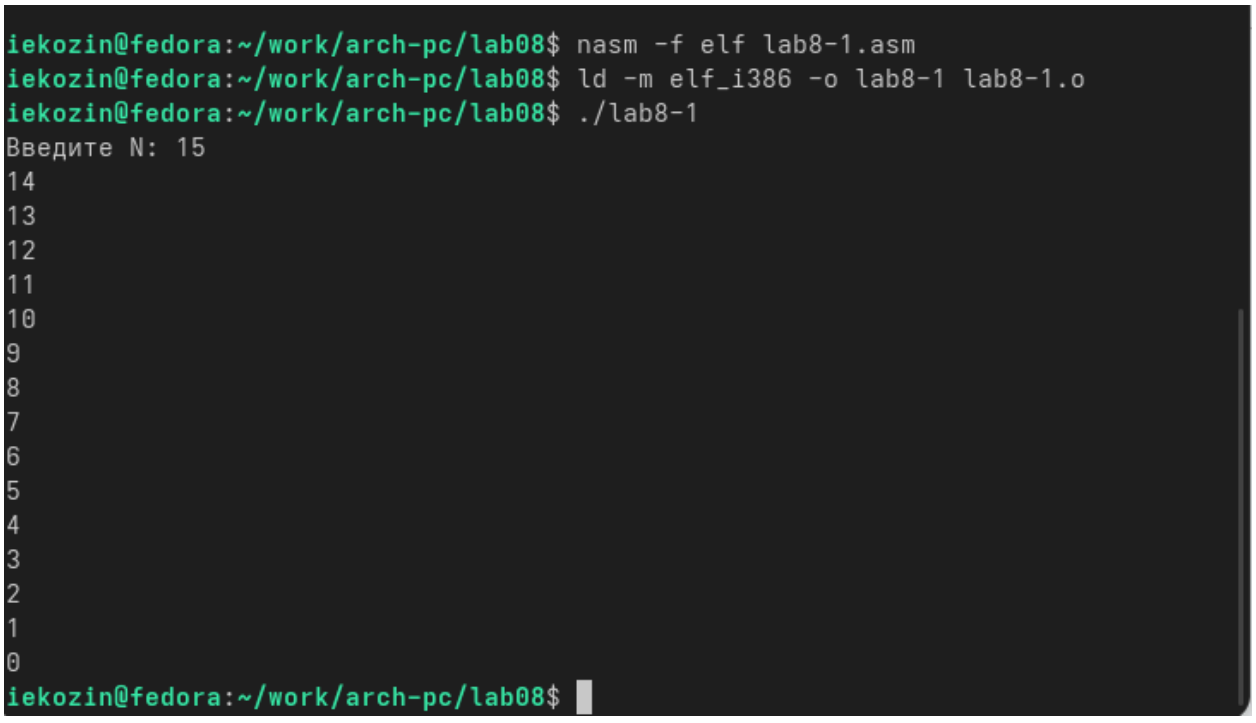

Рис. 7: Запуск измененной программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 8).

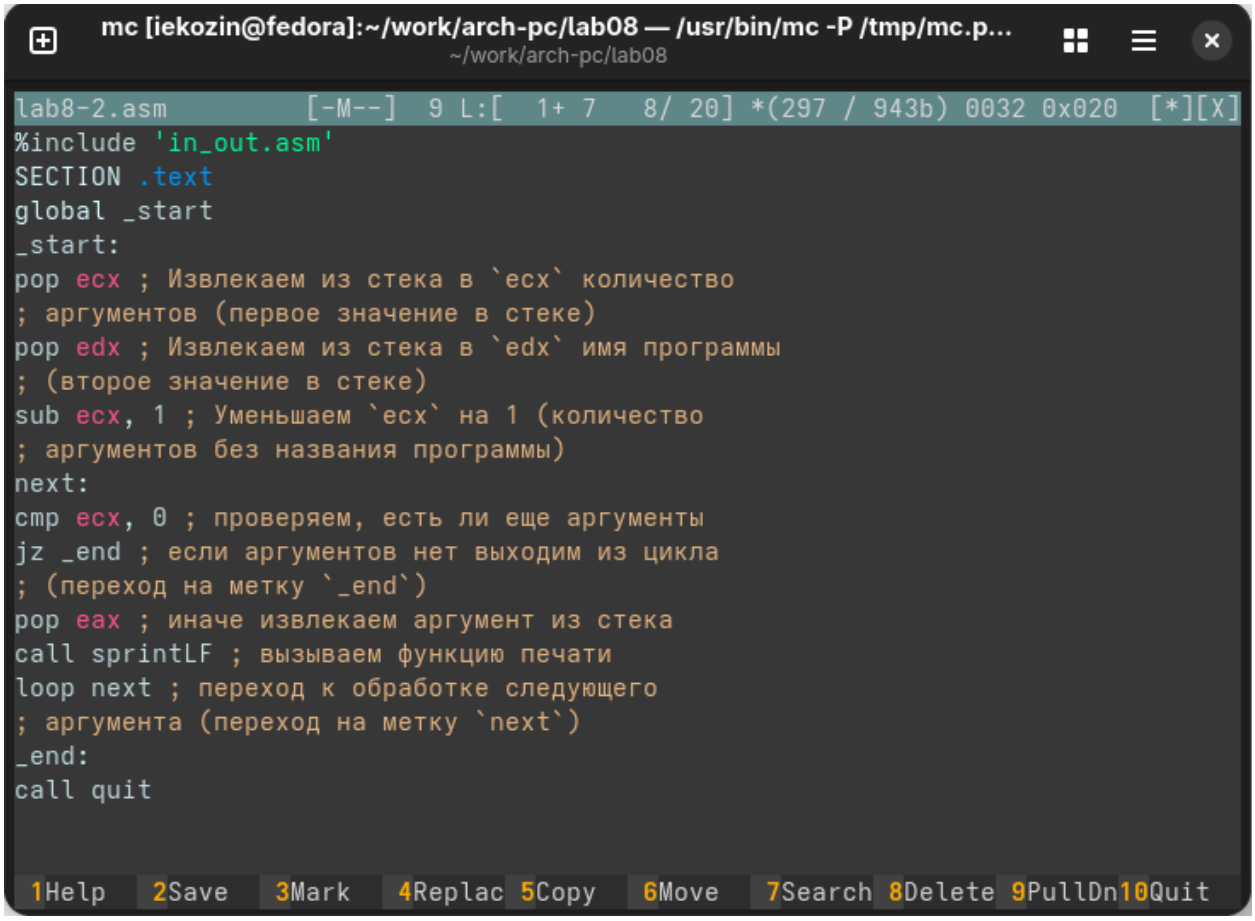

Рис. 8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. 9).

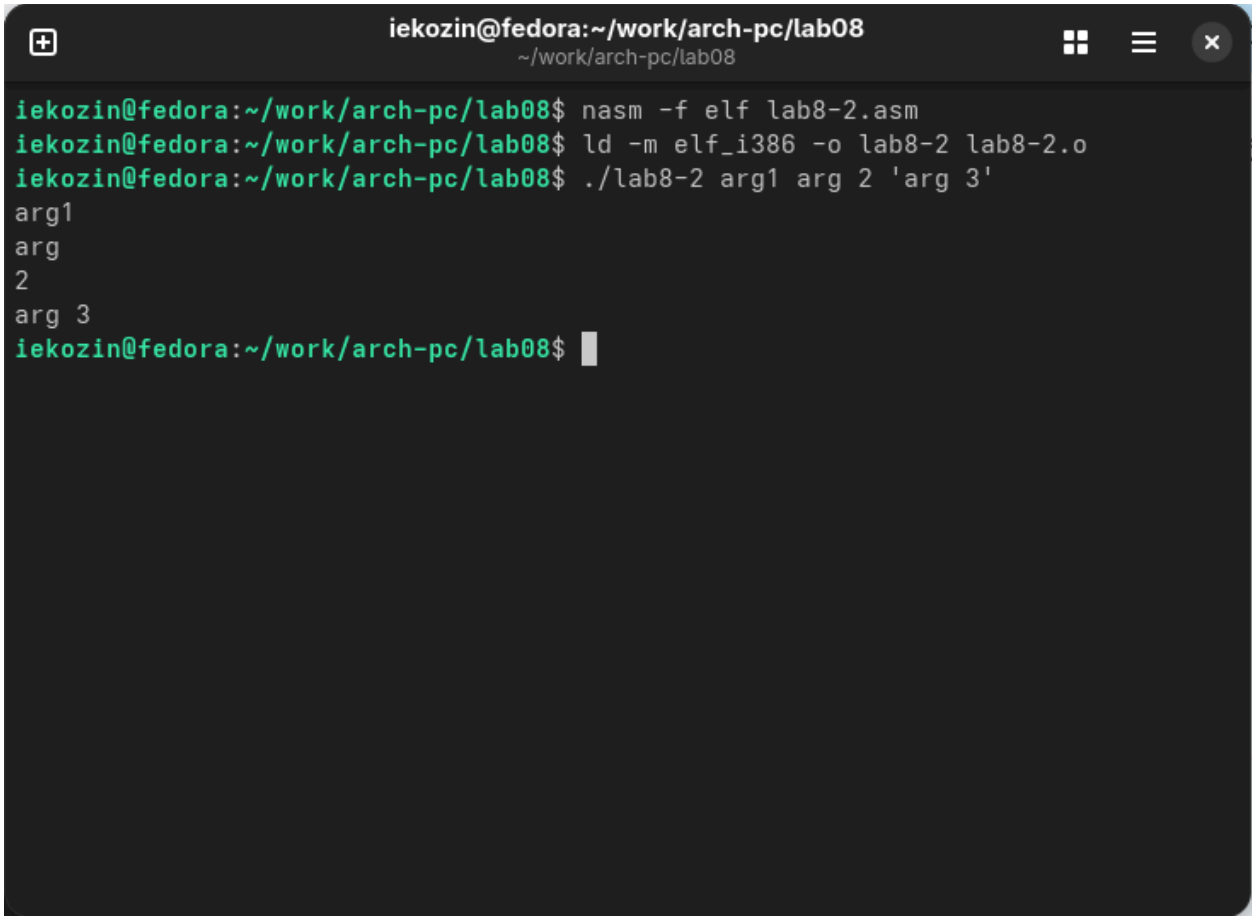

Рис. 9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 10).

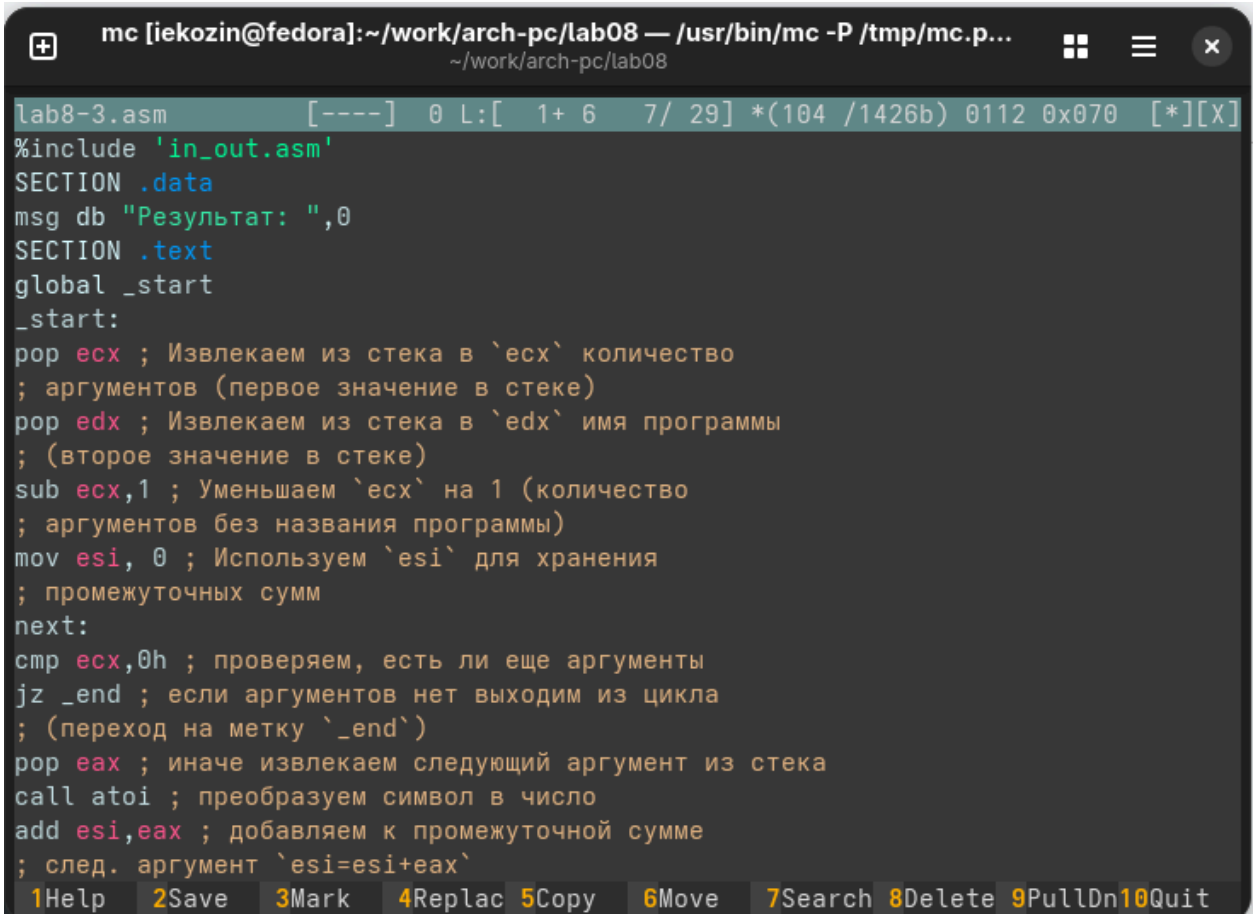

Рис. 10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 11).

```
iekozin@fedora:~/work/arch-pc/lab08$ touch lab8-3.asm
iekozin@fedora:~/work/arch-pc/lab08$ mc

iekozin@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
iekozin@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
iekozin@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
iekozin@fedora:~/work/arch-pc/lab08$
```

Рис. 11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 12).

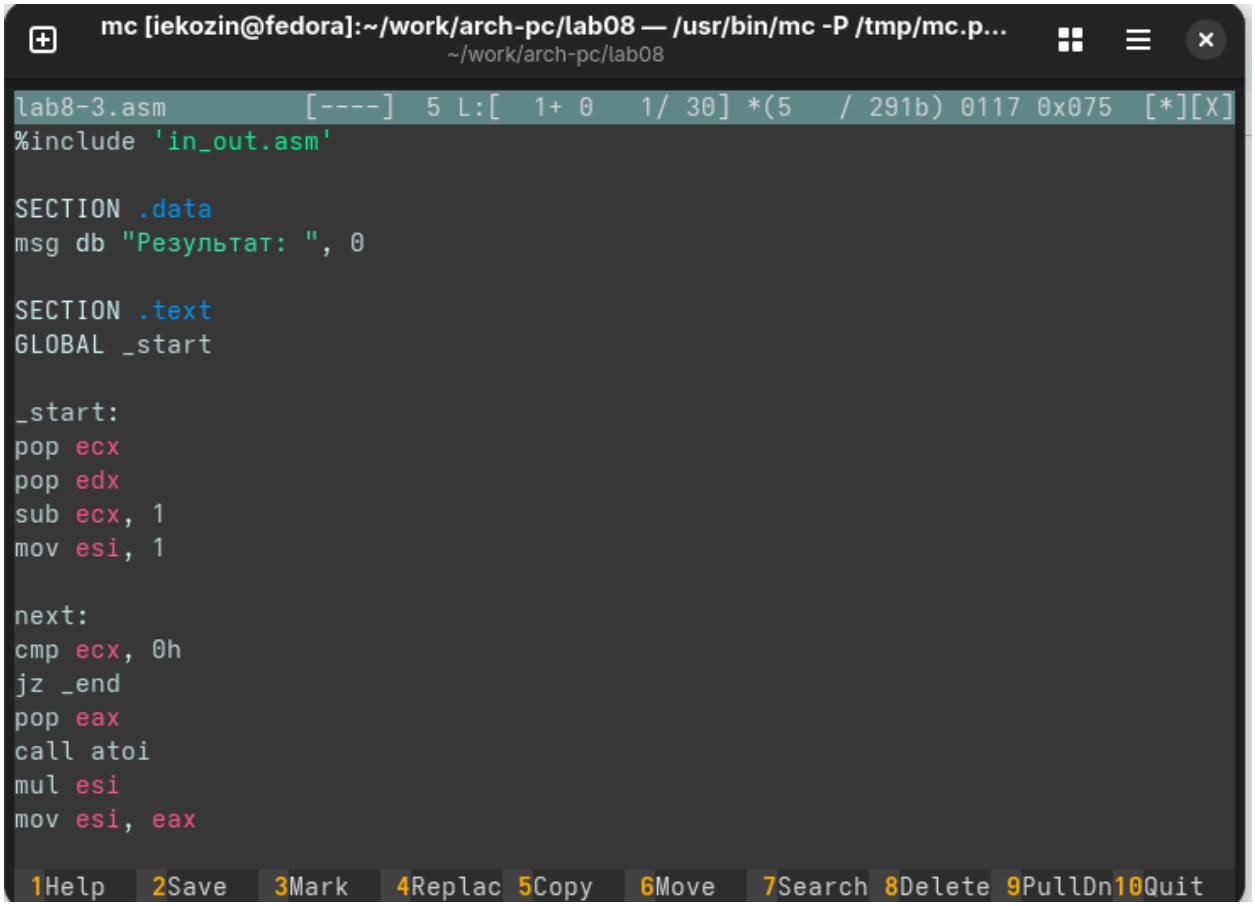

Рис. 12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 13).

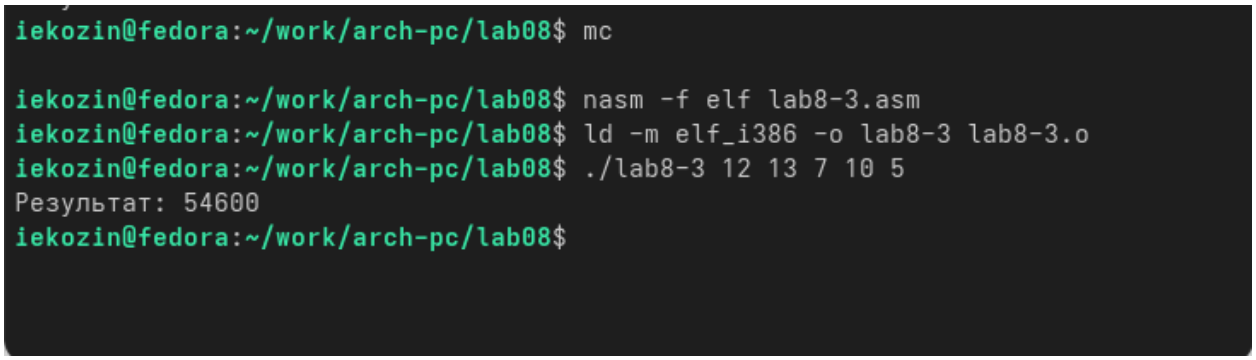

Рис. 13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программму, которая будет находить сумма значений для функции f(x) = 2*(x-1), которая совпадает с моим четвертым варинтом (рис. 14).

Рис. 14: Написание программы для самостоятельной работы

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 2*(x-1)", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
    ; Выводим описание функции
    mov eax, msg_func
    call sprintLF

    ; Аргументы передаются через стек
    ; pop ecx = argc, pop edx = адрес возврата (не используется)
```

```
    pop ecx
    pop edx
    sub ecx, 1           ; уменьшаем на 1, т.к. первый аргумент – имя
программы
    mov esi, 0           ; esi = сумма

next_arg:
    cmp ecx, 0
    jz print_result

    pop eax              ; берем следующий аргумент
    call atoi            ; преобразуем в число

    ; f(x) = 2 * (x - 1)
    sub eax, 1           ; x - 1
    shl eax, 1           ; умножаем на 2
    add esi, eax         ; добавляем к сумме

    dec ecx
    jmp next_arg

print_result:
    mov eax, msg_result
    call sprint
    mov eax, esi
    call iprintLF

    call quit
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 15).

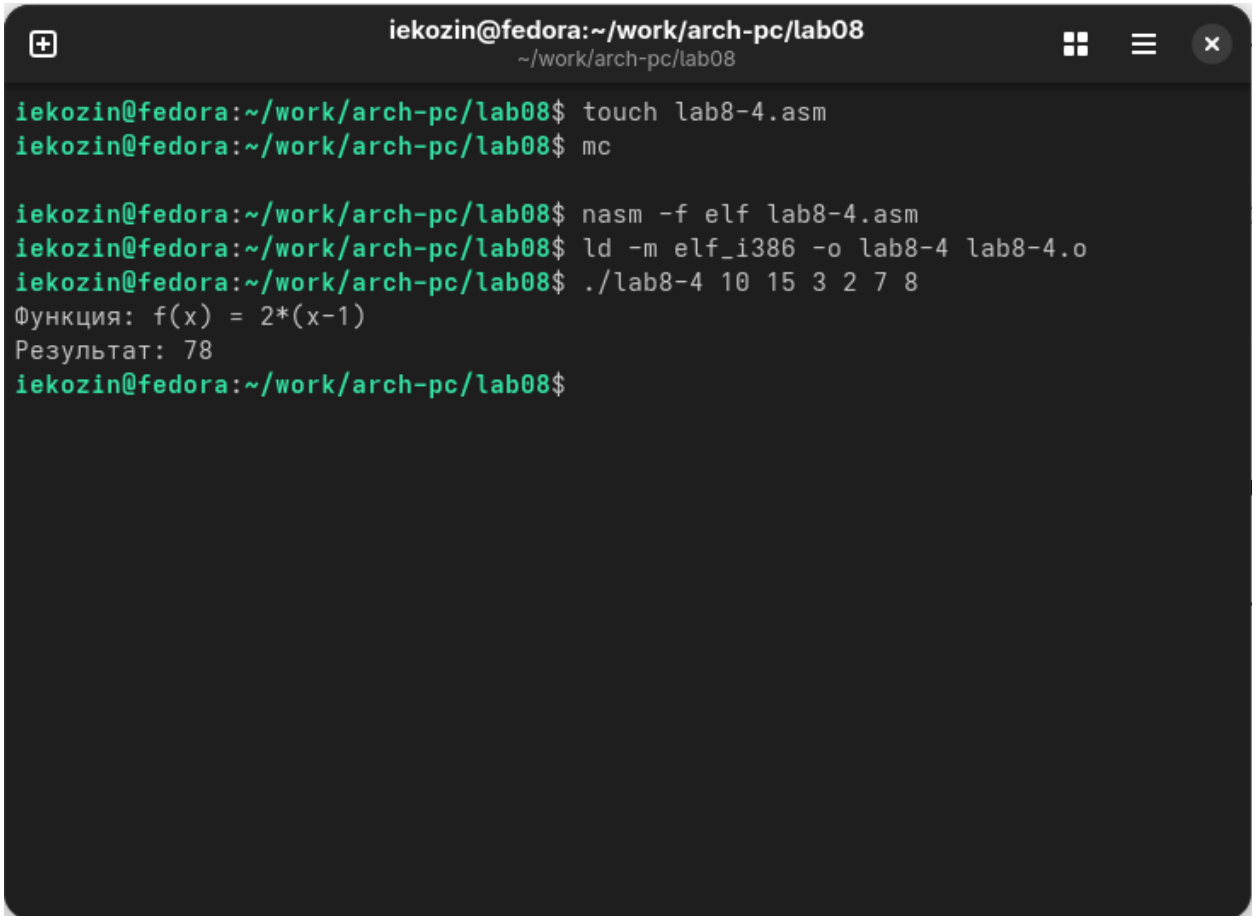

Рис. 15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием циклов а также научился обрабатывать аргументы командной строки.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.