Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Иван Салиндер

Содержание

# 1 Цель работы

Получение навыков по организации циклов и работе со стеком на языке NASM.

# 2 Задание

## 2.1 1. Реализация циклов в NASM

## 2.2 2. Обработка аргументов командной строки

## 2.3 3. Cамостоятельная работа

# 3 Выполнение лабораторной работы

**1**

С помощью утилиты mkdir создаю директорию lab08, перехожу в нее и создаю файл для работы. (рис. [[1](#fig:001)]).

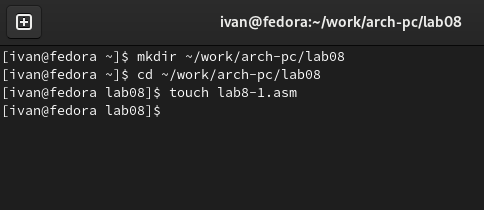


Figure 1: Создание директории

**2**

Открываю созданный файл lab8-1.asm, вставляю в него программу с использованием цикла для вывода чисел. (рис. [[2](#fig:002)]).

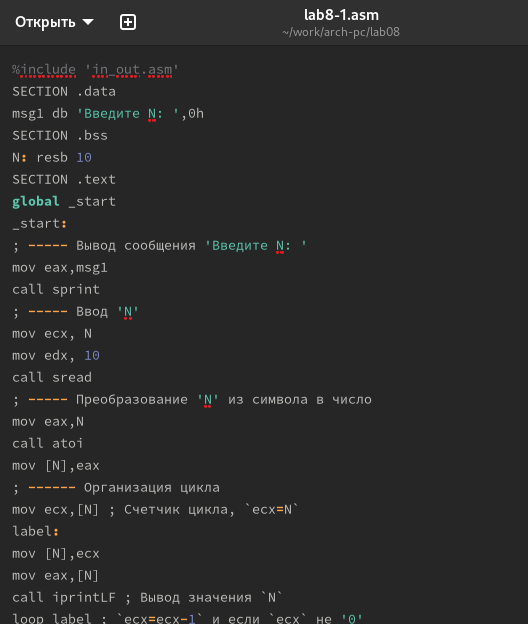


Figure 2: Редактирование файла

**3**

Создаю исполняемый файл программы и запускаю его. (рис. [[3](#fig:003)]).

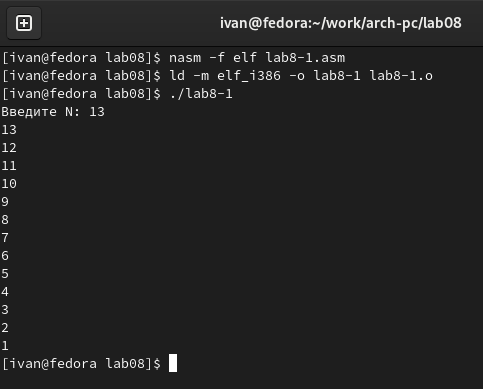


Figure 3: Запуск исполняемого файла

**4**

С помощью инструкции sub уменьшаю изначальный индекс индекс на 1 единичку. (рис. [[4](#fig:004)]).

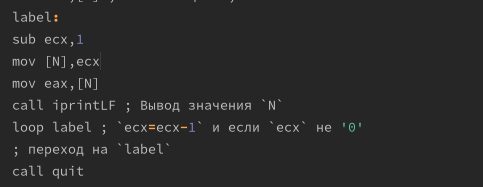


Figure 4: Уменьшение индекса

**5**

Создаю новый исполняемый файл программы и запускаю его. (рис. [[5](#fig:005)]). Получаем результат отличный от ожидаемого

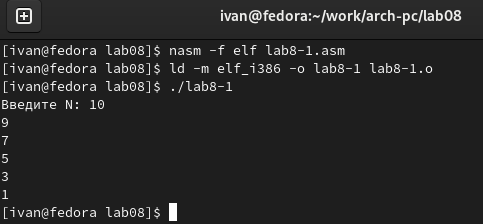


Figure 5: Запуск исполняемого файла

**6**

Изменяю текст программы так, чтобы получить нужный результат, используя стеки для запоминания данных. (рис. [[6](#fig:006)]).

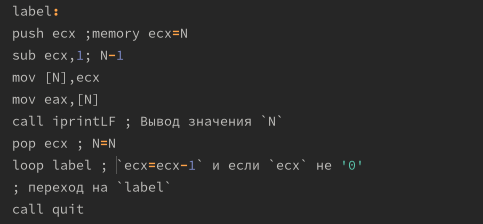


Figure 6: Редактирование программы

**7**

Создаю исполняемый файл и проверяю работу программы. (рис. [[7](#fig:007)]).

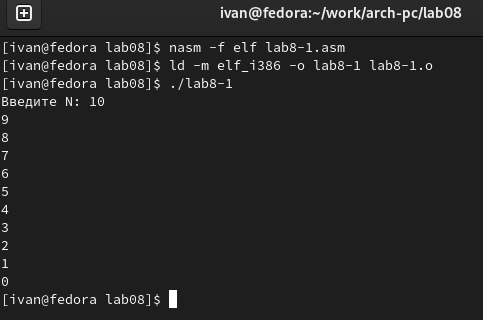


Figure 7: Создание исполняемого файла

* Программа отработало верно.

**8**

Создаю новый файл lab8-2.asm для новой программы. (рис. [[8](#fig:008)]).

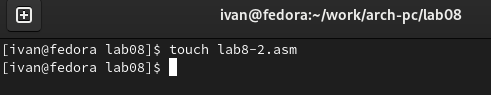


Figure 8: Создание файла

**9**

Вставляю программу, которая выводит все введенные пользователем аргументы. (рис. [[9](#fig:009)]).

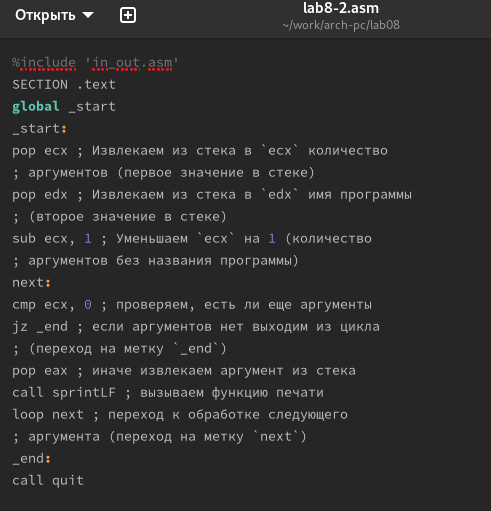


Figure 9: Вставляю текст в файл

**10**

Создаю и запускаю новый исполняемый файл, проверяю работу программы. (рис. [[10](#fig:010)]).

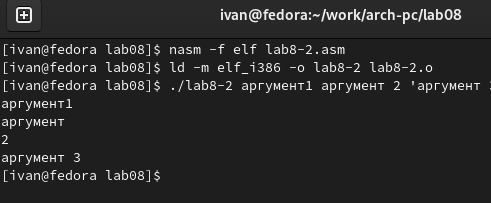


Figure 10: Запуск исполняемого файла

* Программой было обработано 4 аргумента
* Программа считает аргументами все символы до пробела, или значения, которые взяты в ковычки.

**11**

Создаю новый файл lab8-3.asm (рис. [[11](#fig:011)]).



Figure 11: Создание файла

**12**

Открываю файл и ввожу программу, которая складывает все числа введенные пользователем. (рис. [[12](#fig:012)]).

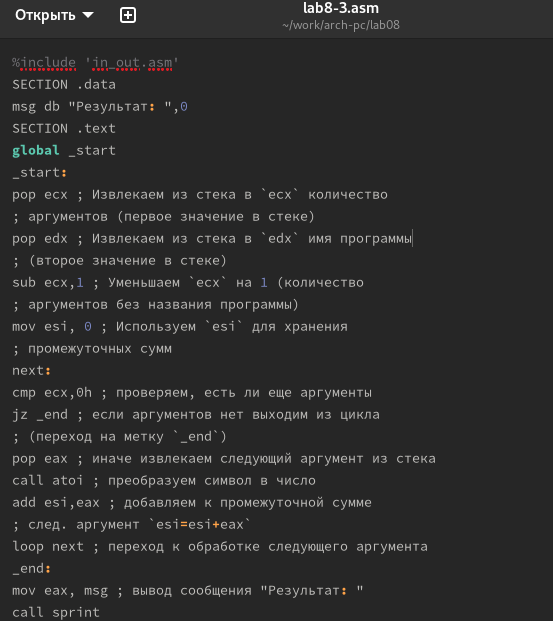


Figure 12: Вставляю программу

**13**

Запускаю испольняемый файл и проверяю работу программы. (рис. [[13](#fig:013)]).

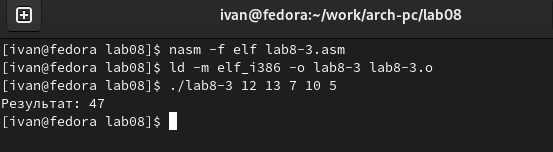


Figure 13: Запуск программы

**14**

Изменяю текст программы так, чтобы она выводила произведение всех чисел, введенные пользователем. (рис. [[14](#fig:014)]).

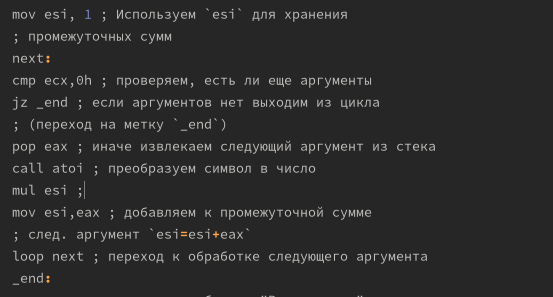


Figure 14: Редактирование файла

**15**

Запускаю испольняемый файл и проверяю работу программы. (рис. [[15](#fig:015)]).

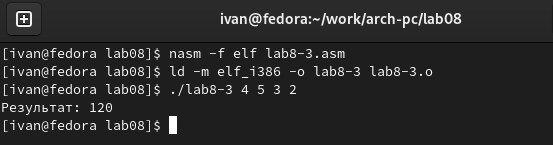


Figure 15: Запуск программы

### 3.0.1 Программа отработала верно!

# 4 Самостоятельная работа

**1**

Создаю файл lab8-4.asm с помощью утилиты touch. (рис. [[16](#fig:016)]).

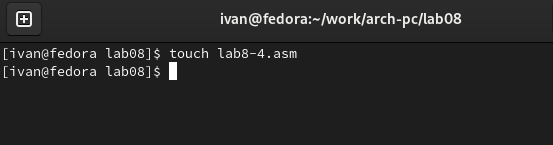


Figure 16: Создание файла

**2**

Ввожу в созданный файл текст программы, у которой находит сумму значений функции (4 Вариант) f(x)=2(𝑥 − 1) для всех аргументов x, введенные пользовтелем. (рис. [[17](#fig:017)]).

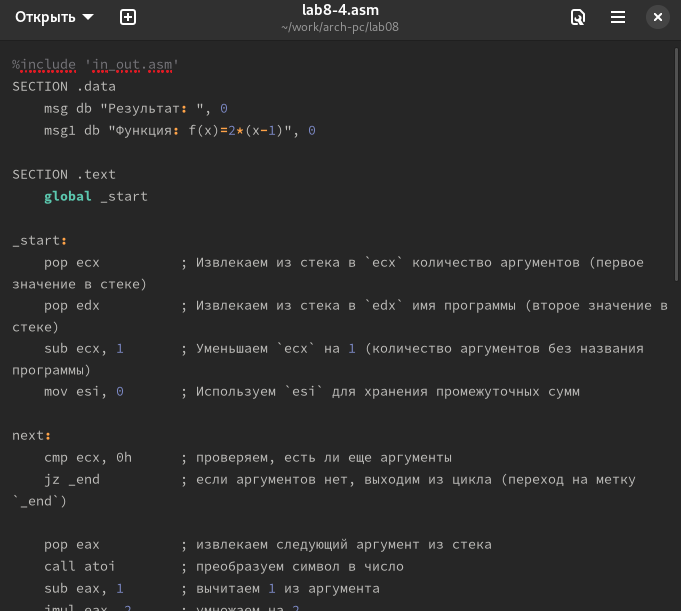


Figure 17: Запуск исполняемого файла

**3**

Создаю исполняемый файл и запускаю его, при x = 5, 3, 6 (рис. [[18](#fig:018)]).

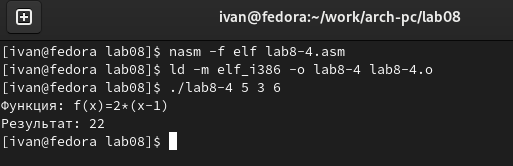


Figure 18: Запуск исполняемого файла

**Текст программы**

%include 'in\_out.asm'  
SECTION .data  
 msg db "Результат: ", 0  
 msg1 db "Функция: f(x)=2\*(x-1)", 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 ; извлекаем следующий аргумент из стека  
 call atoi ; преобразуем символ в число  
 sub eax, 1 ; вычитаем 1 из аргумента  
 imul eax, 2 ; умножаем на 2  
  
 add esi, eax ; добавляем к промежуточной сумме  
 loop next ; переход к обработке следующего аргумента  
  
\_end:  
 mov eax, msg1 ; вывод сообщения "Функция: f(x)=2\*(x-1)"  
 call sprintLF  
  
 mov eax, msg ; вывод сообщения "Результат: "  
 call sprint  
  
 mov eax, esi ; записываем сумму в регистр `eax`  
 call iprintLF ; печать результата  
  
 call quit ; завершение программы

# 5 Выводы

В ходе выполениния работы были получены навыки по организации циклов и по работе со стеком на языке NASM.