Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютеров

Карпова Анастасия Александровна

Содержание

[1 Цель работы 1](#_Toc152446781)

[2 Задание 1](#_Toc152446782)

[3 Теоретическое введение 1](#_Toc152446783)

[4 Выполнение лабораторной работы 2](#_Toc152446784)

[5 Выводы 8](#_Toc152446785)

[Список литературы 9](#_Toc152446786)

# 1 Цель работы

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 для самостоятельной 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. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop). 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. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. 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

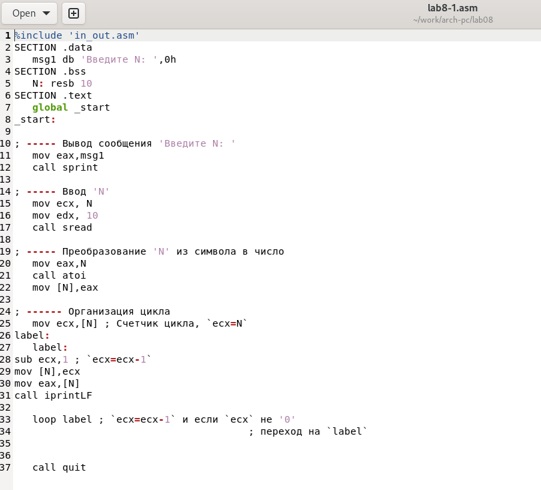
8.3.1. Реализация циклов в NASM

Создаю каталог для программам лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. 1).

Создание файла и каталога

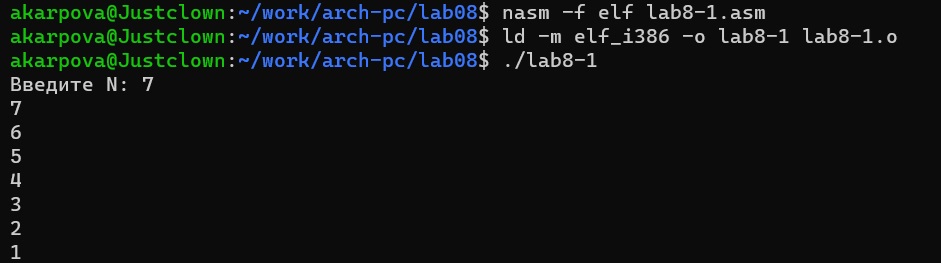
Создание файла и каталога

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 2)



Ввод программы

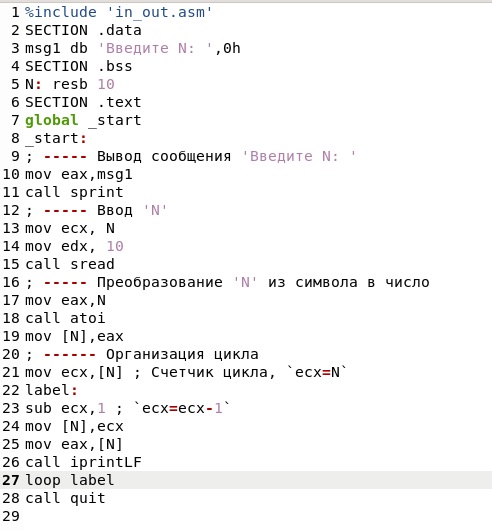
Создаю исполняемый файл и проверяю его работу. (рис. 3)



Создание исполняемого файла и проверка работы

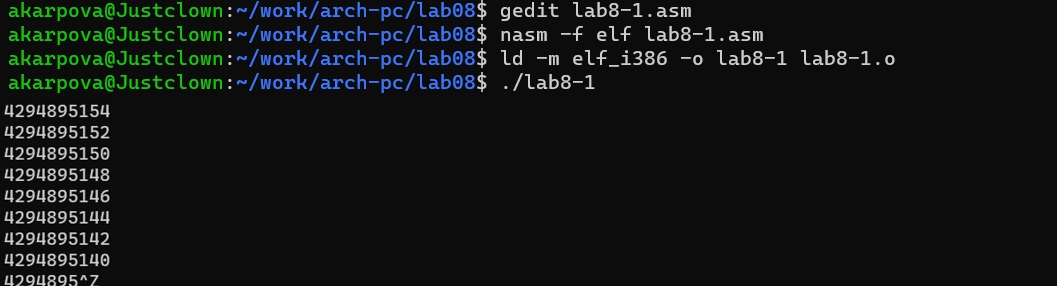
Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08

Программа выводит числа от N до 1 включительно Изменяю текст программы добавив изменение значение регистра ecx в цикле. (рис. 4)



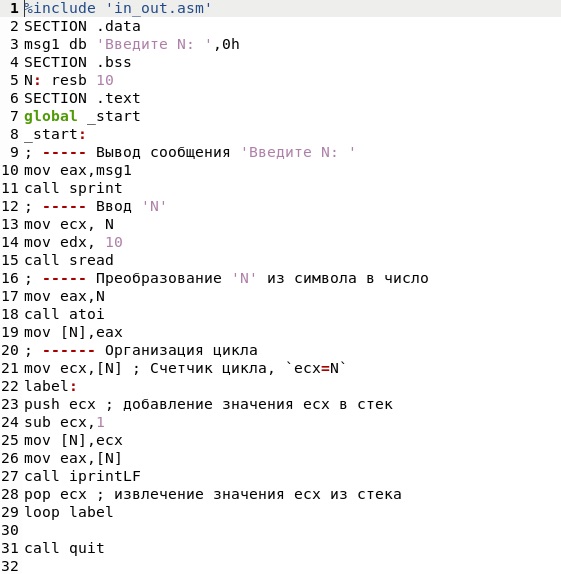
Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 5)



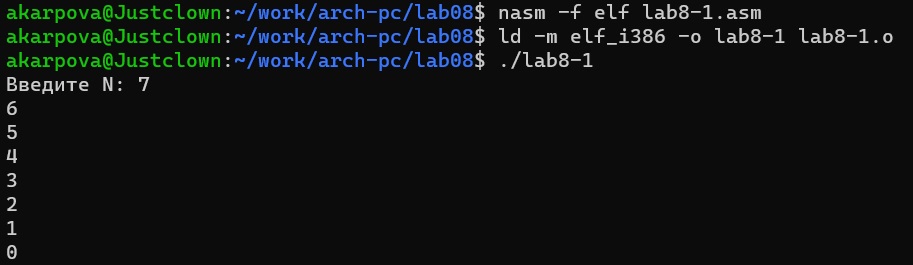
Создание исполняемого файла и проверка работы

В данном случае число проходов не соответствует введенному с клавиатуры значению. Снова изменяю текст программы, добавив команды push и pop для сохранения значения счётчика цикла loop (рис. 6)



Изменение программы

Создаю исполняемый файл и проверяю его работу (рис. 7)



Создание исполняемого файла и проверка работы

Теперь число проходов цикла соответствует введеному с клавиатуры значению и выводит числа от N-1 до 0 включительно

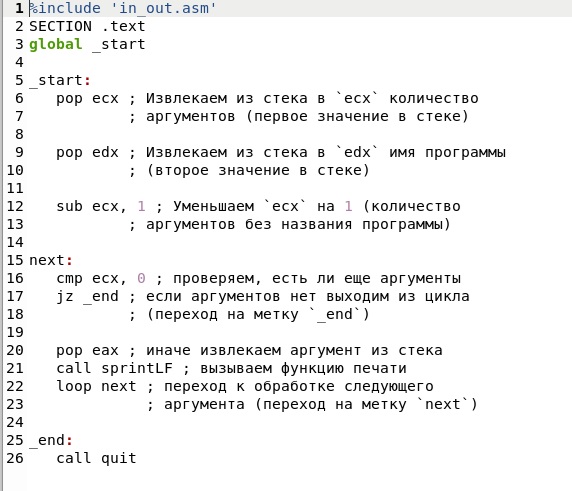
8.3.2. Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге lab08 (рис. 8)

Создание файла

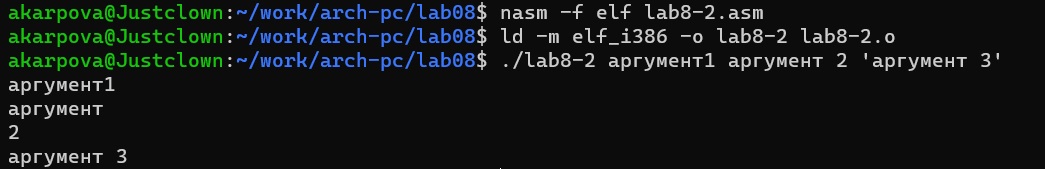
Создание файла

Ввожу в него текст программы из листинга 8.2. (рис. )



Ввод программы

Создаю исполняемый файл и запускаю его. (рис. 10)



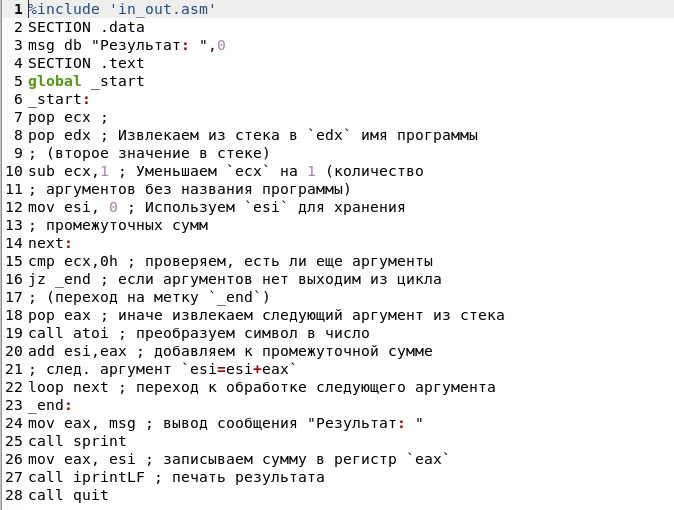
Создание исполняемого файла и проверка работы

Программа выводит 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа не считывает 2 как отдельные аргумент. Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге lab08. (рис. 11)

Создание файла

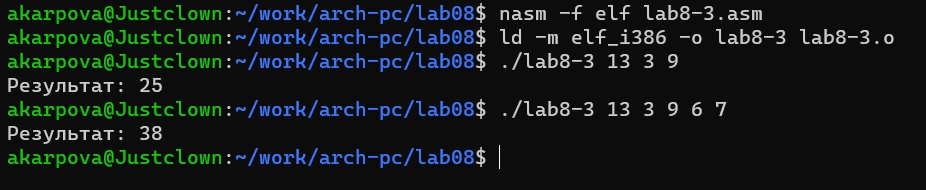
Создание файла

Ввожу в него текст программы из листинга 8.3. (рис. 12)



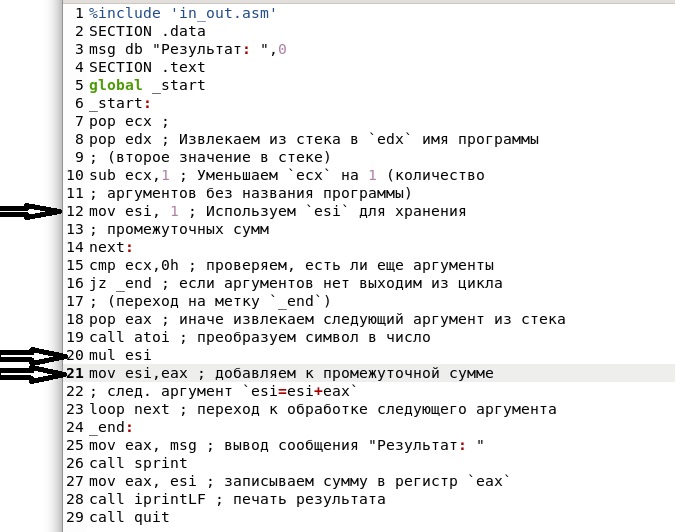
Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 13)



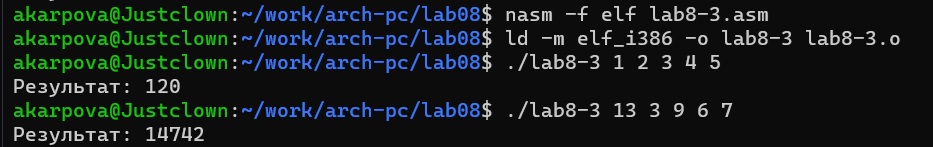
Создание исполняемого файла и проверка работы

Изменяю текст программы для вычисления произведения аргументов командной строки (рис. 14)



Изменение программы

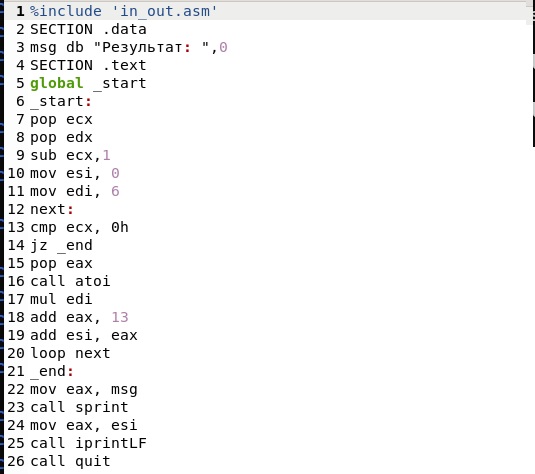
Создаю исполняемый файл и запускаю его, указав аргументы (рис. 15)



Создание исполняемого файла и проверка работы

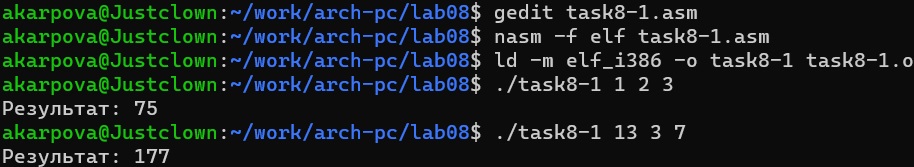
8.4. Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции f(x) = 6x+13 в соответствие с моим вариантом - 15, для x1, x2, …, xn. Значения xi передаются как аргументы. (рис. 16)



Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 17)



Создание исполняемого файла и проверка работы

# 5 Выводы

В ходе лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# Список литературы

Архитектура ЭВМ