Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров и операционные системы

Лисенков Егор Романович

Содержание

# 1 Цель работы

Благодаря этому, появляются навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание 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.

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.

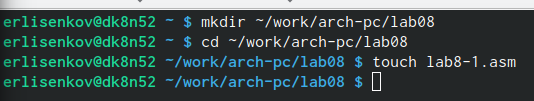
Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

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

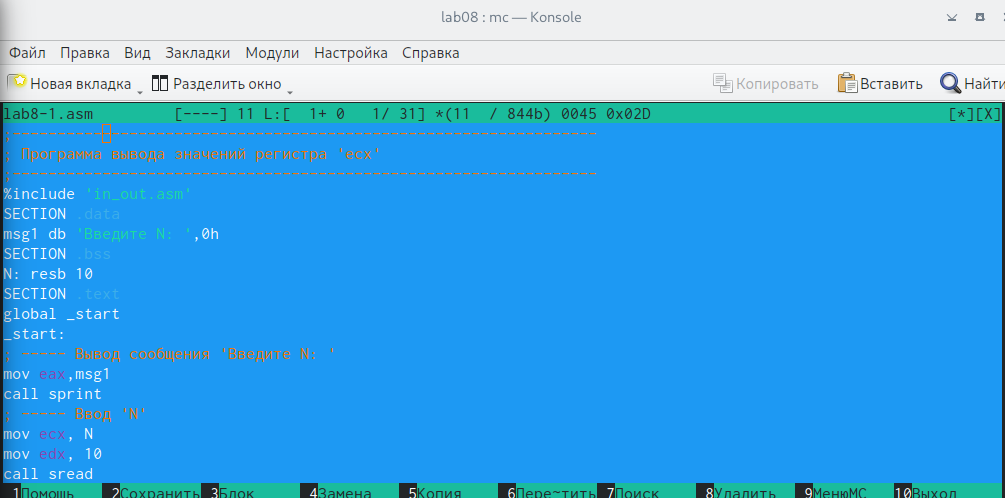
## 4.1 **Реализация циклов в NASM**

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



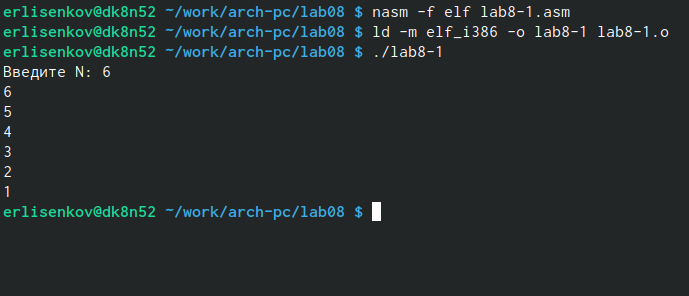
Создание файлов для лабораторной работы

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



Ввод текста из листинга 8.1

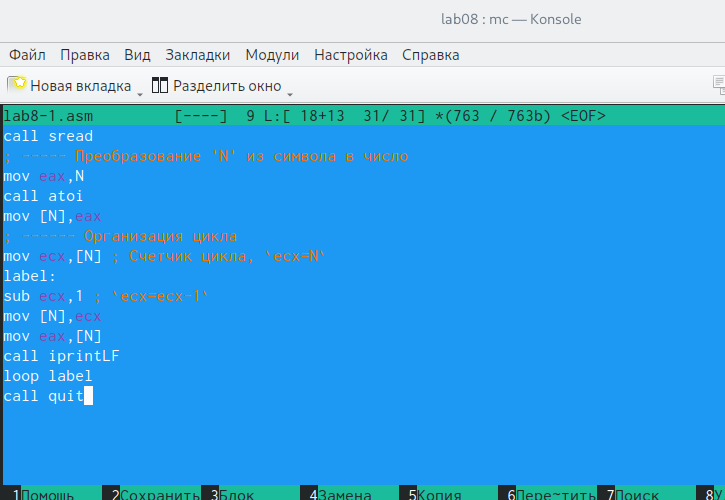
Создаю исполняемый файл и проверяю его работу. (рис. ??).



Запуск файла

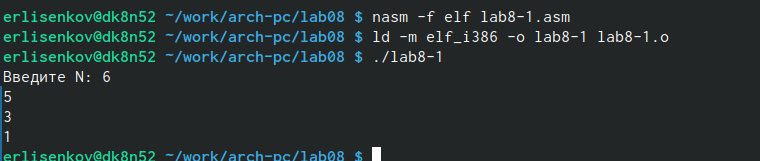
Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра ecx в цикле. (рис. ??).



Изменение текста

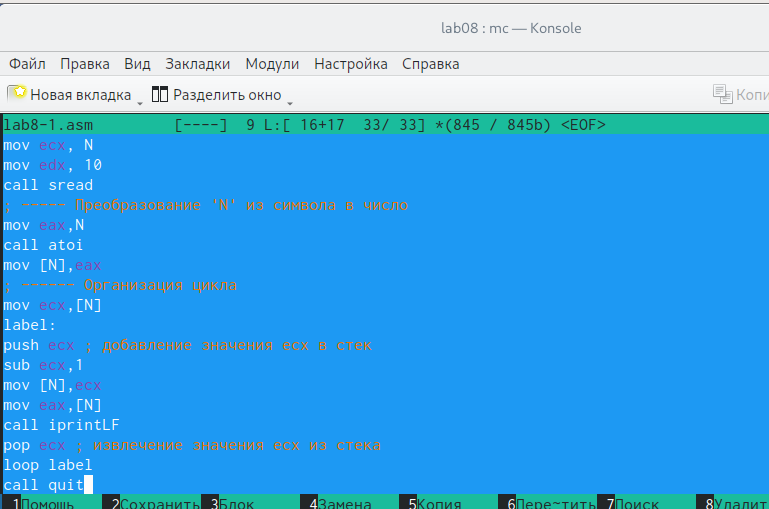
Создаю исполняемый файл и проверяю его работу. (рис. ??).



Запуск программы

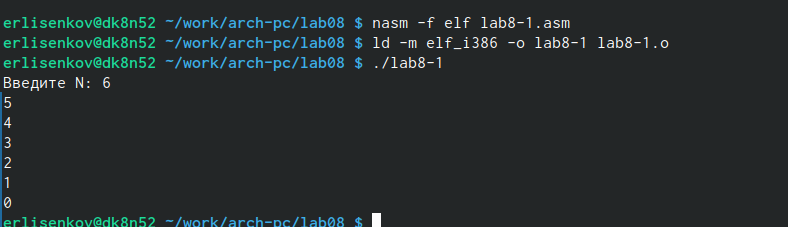
В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. ??).



Изменение текста программы

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

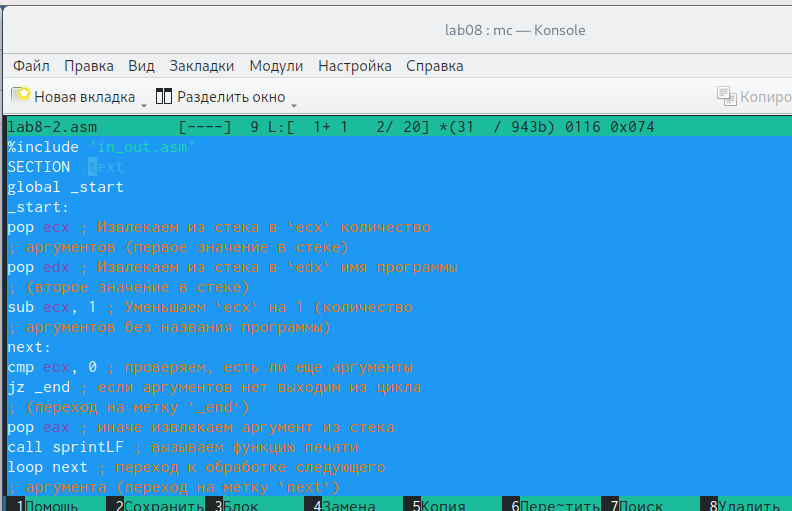


Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

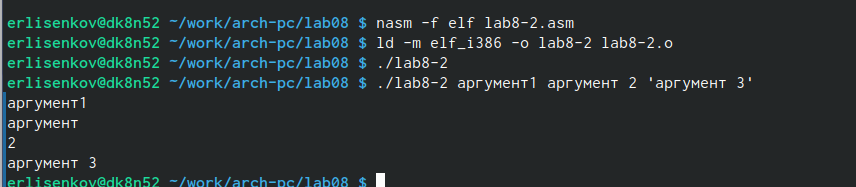
## 4.2 **Обработка аргументов командной строки**

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.2. (рис. ??).



Ввод текста программы из листинга 8.2

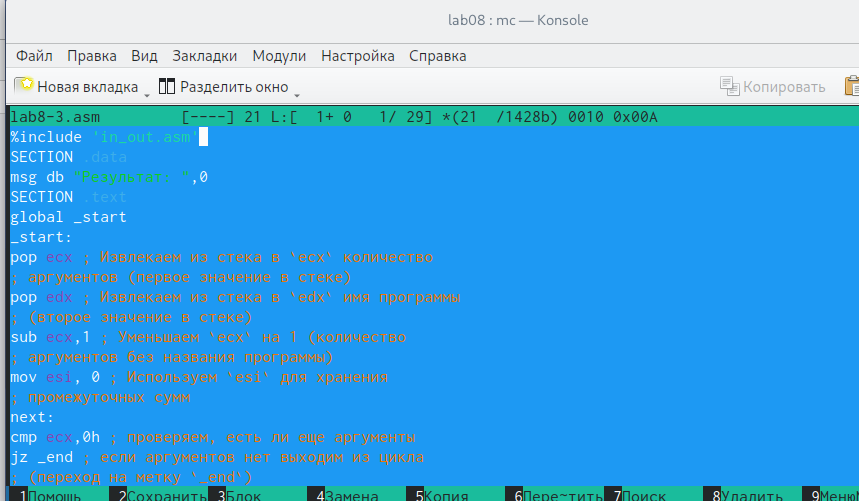
Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. ??).



Запуск исполняемого файла

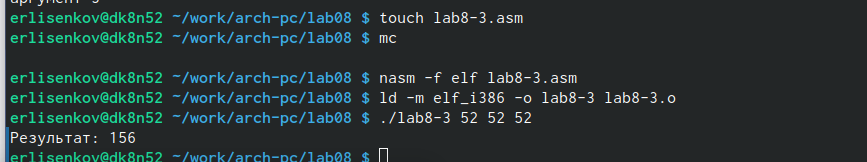
Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличии от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/archpc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. ??).



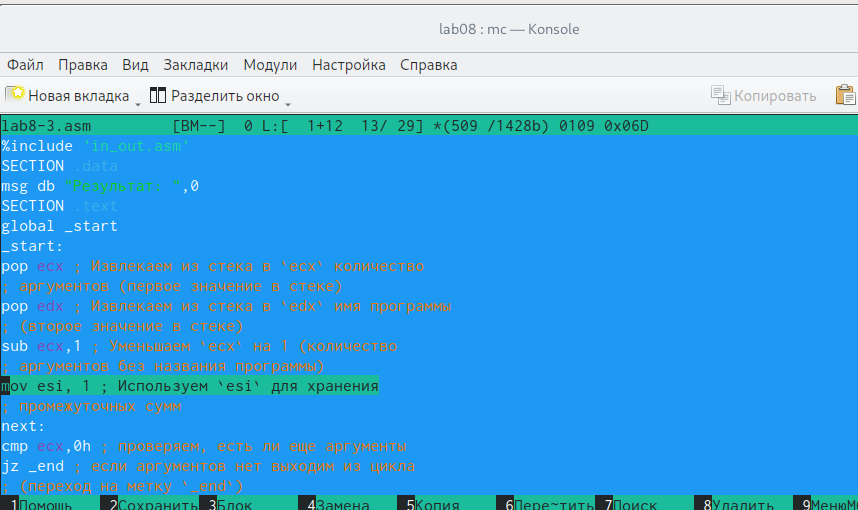
Ввод текста программы из листинга 8.3

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



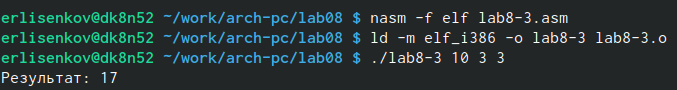
Запуск исполняемого файла

Изменю текст программы листинга 8.3 для выполнения программы. (рис. ??).



Изменение текста программы

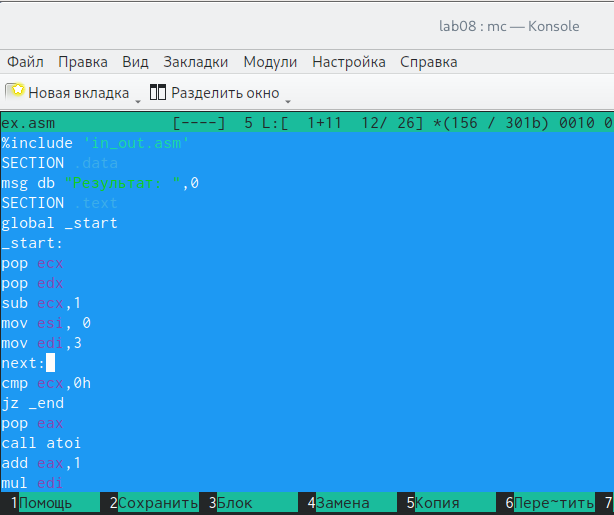
Создаю исполняемый файл и запускаю его, указав аргументы. (рис. ??).



Запуск исполняемого файла

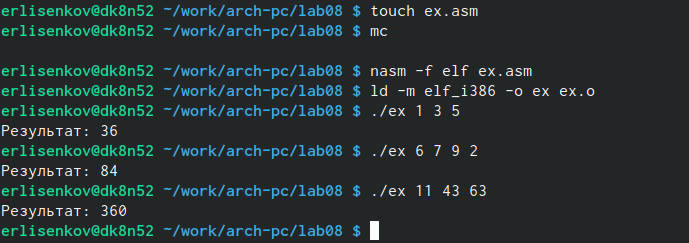
## 4.3 **Задание для самостоятельной работы**

Пишу текст программы к своему варианту №2 (рис. ??).



Изменение программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах x = x1, x2, …, xn. (рис. ??).



Запуск исполняемого файла

Замечу, что программа работает корректно!

Текст программы:

%include ‘in\_out.asm’

SECTION .data

msg db “Результат:”,0

SECTION .text

global \_start

\_start:

pop ecx

pop edx

sub ecx,1

mov esi, 0

mov edi,3

next:

cmp ecx,0h

jz \_end

pop eax

call atoi

add eax,1

mul edi

add esi,eax

loop next

\_end:

mov eax, msg

call sprint

mov eax, esi

call iprintLF

call quit

# 5 Выводы

Полученные знания являются актуальными для моего направления обучения. Они понадобятся мне в дальнейшей работе и помогут понимать работу компьютера.

# 6 Список литературы

Лабораторная работа №8. Программирование цикла. Обработка аргументов командной строки. Файл