Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

# 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 задания 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.

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. Иструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

# 4 Выполнение лабораторной работы

## 4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 с помощью команды mkdir, перехожу в него и создаю файл lab8-1.asm с помощью команды touch (рис. 1).

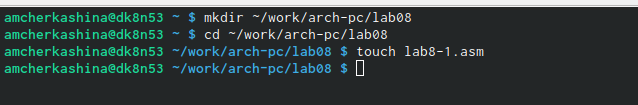


Рис. 1: Создание каталога и файла

С помощью команды cp копирую в текущий каталог файл in\_out.asm, т.к. он будет использоваться в программах данной лабораторной работы (рис. 2).

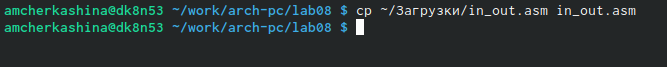


Рис. 2: Создание копии файла

Ввожу в файл lab8-1.asm текст программы вывода значений регистра ecx из листинга 8.1 (рис. 3).

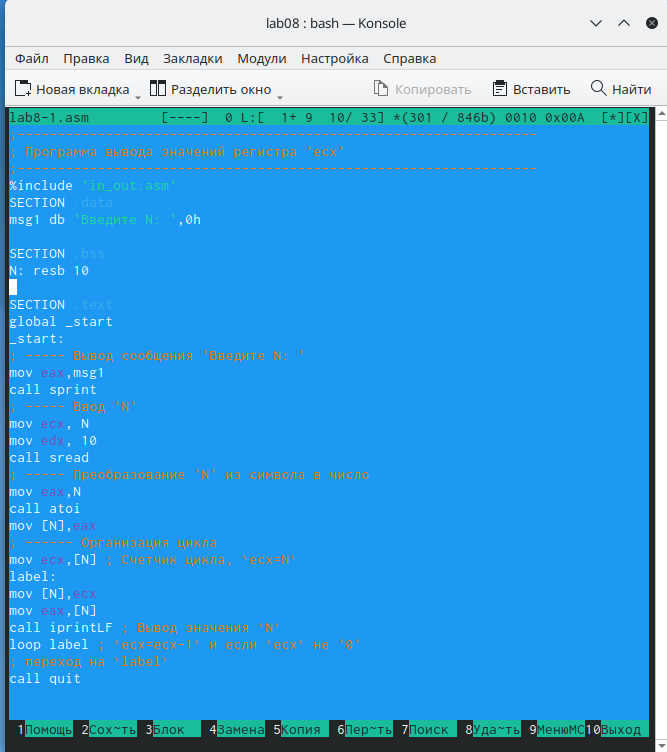


Рис. 3: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. 4).

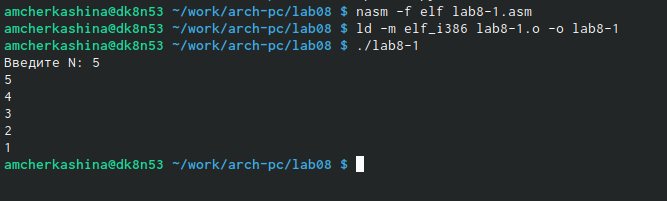


Рис. 4: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра ecx в цикле (рис. 5).

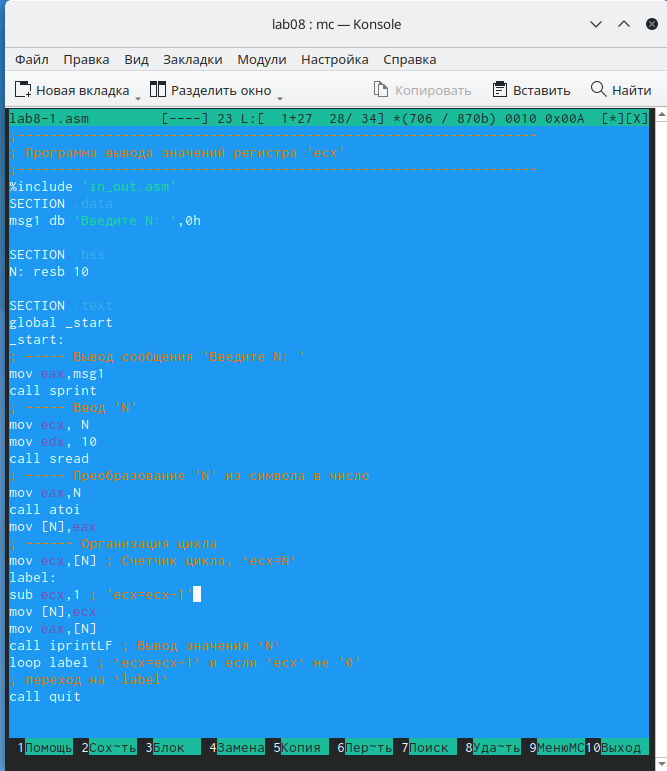


Рис. 5: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 6).

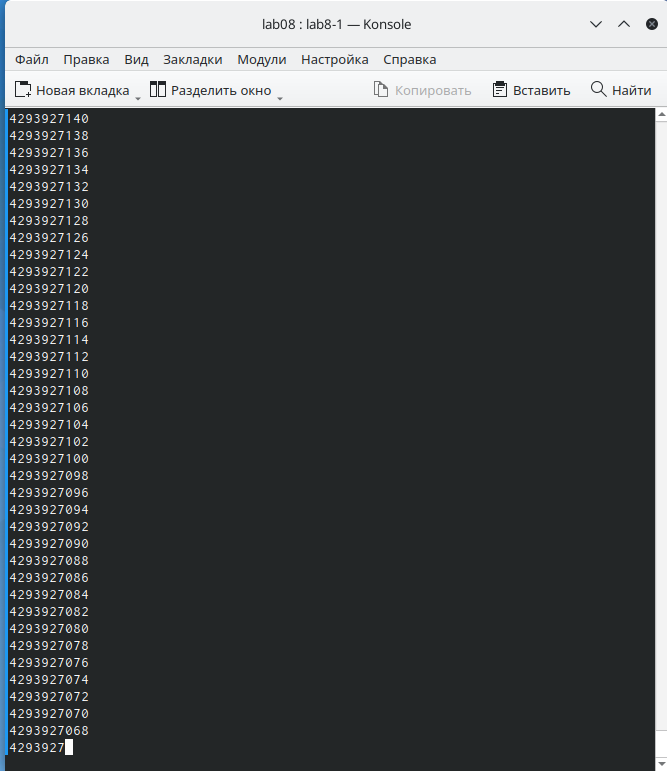


Рис. 6: Запуск нового исполняемого файла

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

Вношу изменения в текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. 7).

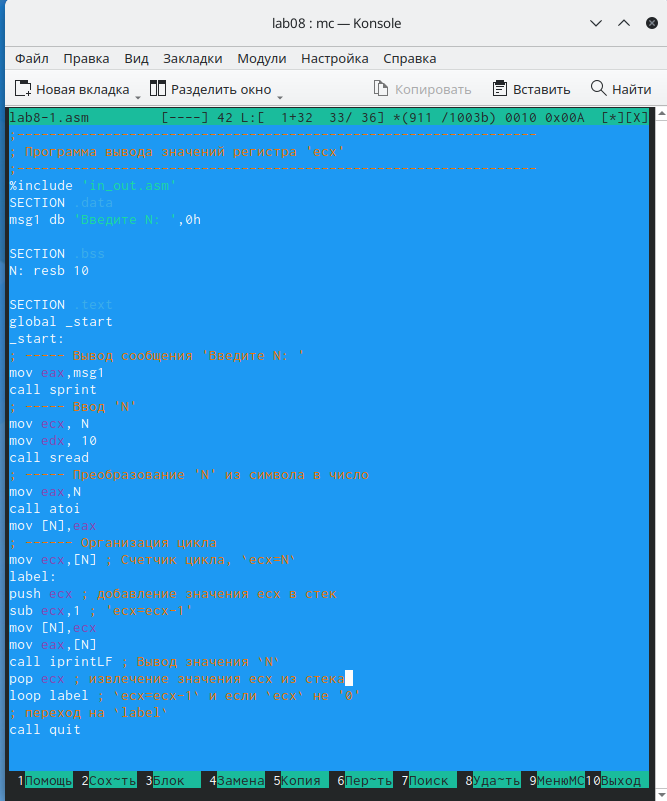


Рис. 7: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 8).

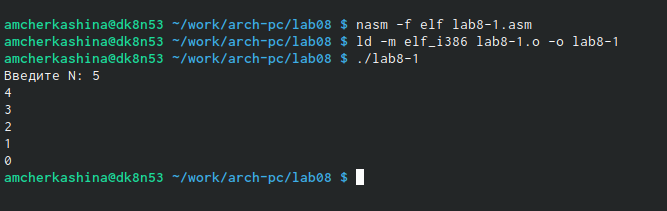


Рис. 8: Запуск обновленной программы

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 с помощью команды touch (рис. 9).

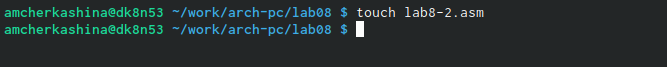


Рис. 9: Создание файла

Ввожу в него текст программы, выводящей на экран аргументы командной строки, из листинга 8.2 (рис. 10).

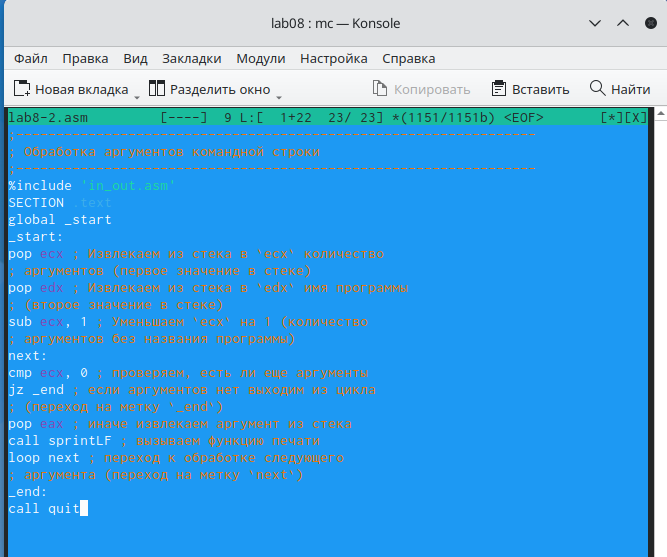


Рис. 10: Редактирование файла

Создаю исполняемый файл и запускаю его, указав следующие аргументы: аргумент1 аргумент 2 ‘аргумент 3’ (рис. 11).

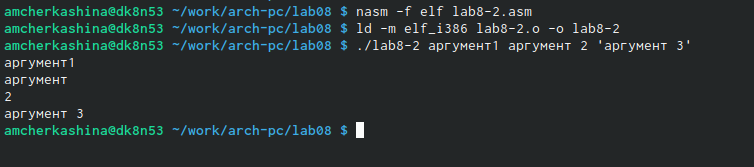


Рис. 11: Запуск исполняемого файла с нужными аргументами

Программа обработала и вывела 4 аргумента. Так как аргумент 2 не был взят в кавычки, в отличии от ‘аргумент 3’, из-за наличия пробела программа считывает “2” как отдельный аргумент.

Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 с помощью команды touch (рис. 12).

Создание файла

Рис. 12: Создание файла

Ввожу в него текст программы вычисления суммы аргументов командной строки из листинга 8.3 (рис. 13).

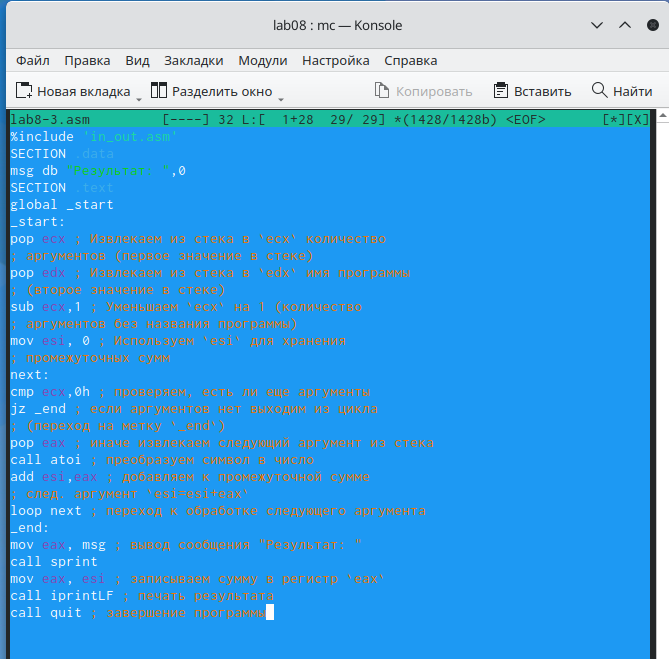


Рис. 13: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 14).

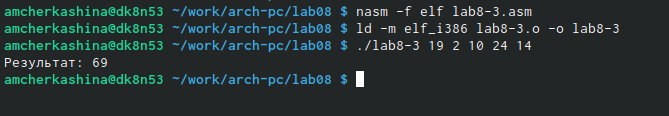


Рис. 14: Запуск исполняемого файла с аргументами

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. Для этого изменяю изначальное значение счетчика на 1, а add на mul (рис. 15).

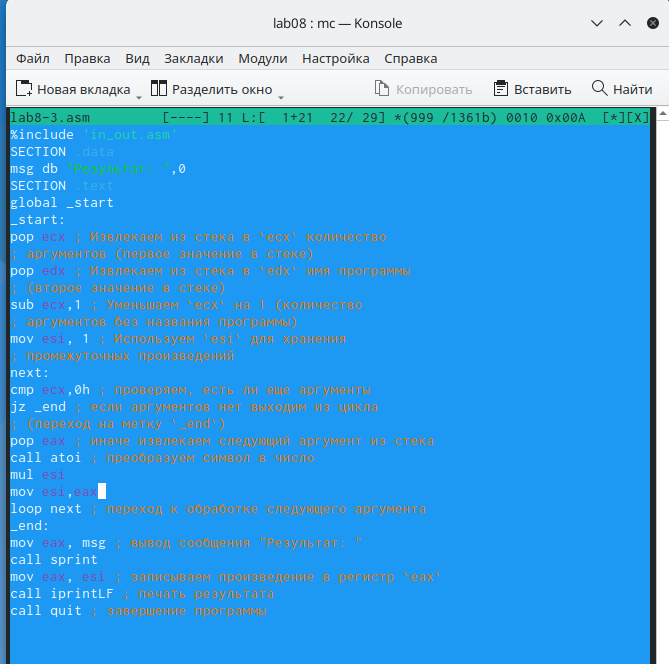


Рис. 15: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 16).

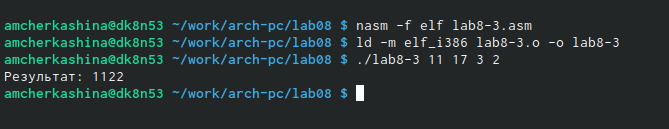


Рис. 16: Запуск исполняемого файла

**Листинг 4.1. Программа вычисления произведения аргументов командной строки**

%include 'in\_out.asm'  
SECTION .data  
msg db "Результат: ",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, 1 ; Используем `esi` для хранения  
; промежуточных произведений  
next:  
cmp ecx,0h ; проверяем, есть ли еще аргументы  
jz \_end ; если аргументов нет выходим из цикла  
; (переход на метку `\_end`)  
pop eax ; иначе извлекаем следующий аргумент из стека  
call atoi ; преобразуем символ в число  
mul esi  
mov esi,eax  
loop next ; переход к обработке следующего аргумента  
\_end:  
mov eax, msg ; вывод сообщения "Результат: "  
call sprint  
mov eax, esi ; записываем произведение в регистр `eax`  
call iprintLF ; печать результата  
call quit ; завершение программы

## 4.3 Выполнение задания для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции f(x) для x = x1, x2, …, xn, т.е. программу, выводящую значения f(x1) + f(x2) + … + f(xn). Значения xi передаются как аргументы. Вид функции f(x) выбираю из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным мной при выполнении лабораторной работы №6.

Мой вариант 11, соответственно пишу программу нахождения суммы значений функции f(x) = 15\*x + 2 (рис. 17).

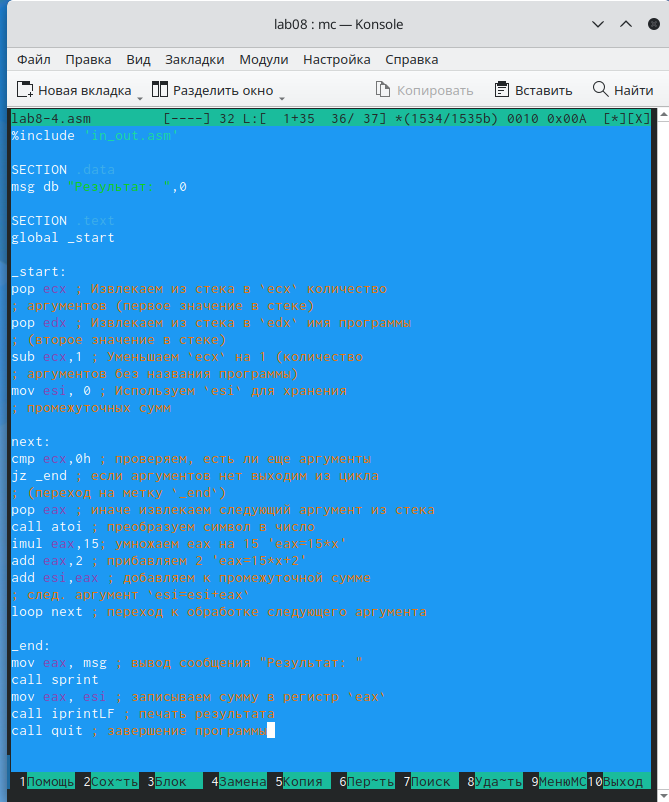


Рис. 17: Написание программы

Создаю исполняемый файл и проверяю его работу на нескольких наборах x = x1, x2, …, xn (рис. 18).

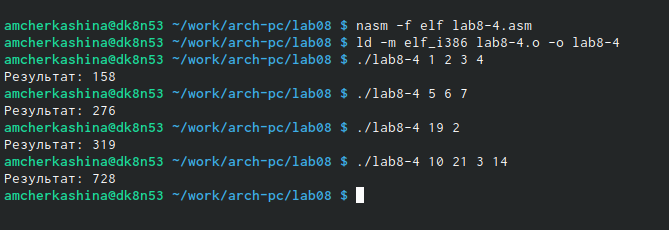


Рис. 18: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

\*\*Листинг 4.2. Программа нахождения суммы значений функции f(x) = 15\*x + 2\*\*

%include 'in\_out.asm'  
  
SECTION .data  
msg db "Результат: ",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 ; иначе извлекаем следующий аргумент из стека  
call atoi ; преобразуем символ в число  
imul eax,15; умножаем eax на 15 'eax=15\*x'  
add eax,2 ; прибавляем 2 'eax=15\*x+2'  
add esi,eax ; добавляем к промежуточной сумме  
; след. аргумент `esi=esi+eax`  
loop next ; переход к обработке следующего аргумента  
  
\_end:  
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

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# 6 Список литературы

1. Архитектура ЭВМ