Лабораторная работа №8

Архитектура компьютера.Программирование цикла.Обработка аргументов командной строки

Сафиуллина Айлина Саяровна

Содержание

# 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. 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. 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. 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. 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

1. Реализация циклов в NASM

Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рис. 1).

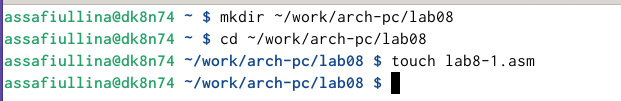


Рис. 1: Каталог lab08

При реализации циклов в NASM с использованием инструкции loop необходимо помнить о том, что эта инструкция использует регистр ecx в качестве счетчика и на каждом шаге уменьшает его значение на единицу. Введем в файл lab8-1.asm текст программы из листинга 8.1. (рис. 2).

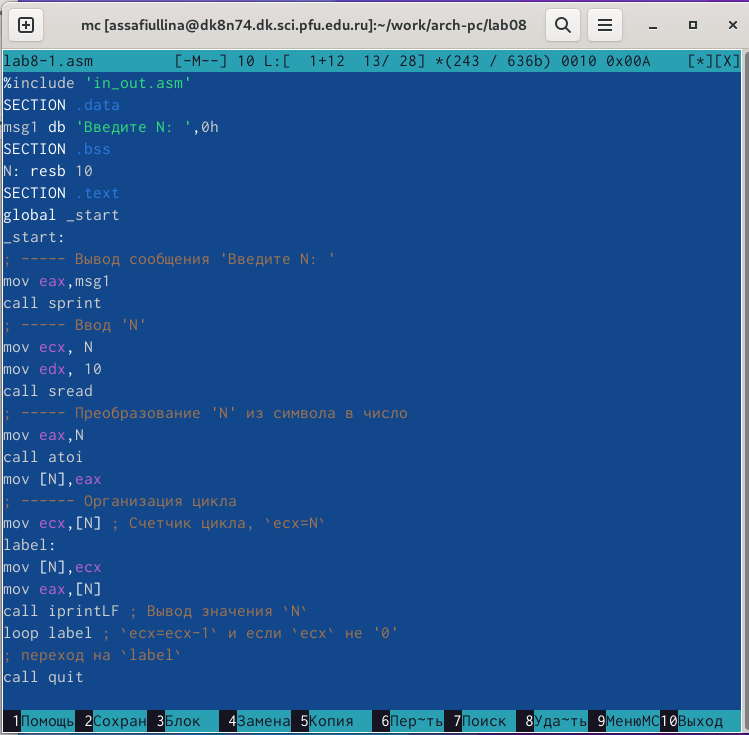


Рис. 2: Текст из листинга 8.1

Создадим исполняемый файл и проверим его работу (рис. 3).

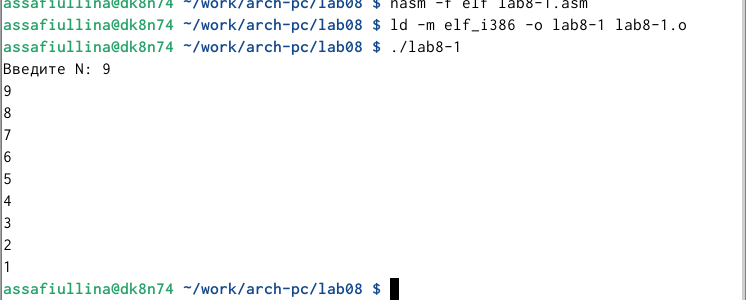


Рис. 3: Проверка исполняемого файла

Данный пример показывает, что использование регистра ecx в теле цилка loop может привести к некорректной работе программы. Изменим текст программы (рис. 4), добавив изменение значение регистра ecx в цикле:

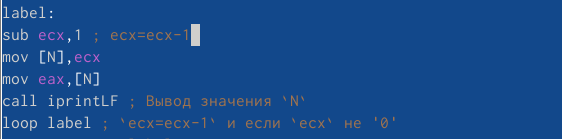


Рис. 4: Замена текста программы

Создадим исполняемый файл и запустим его. (рис. 5).

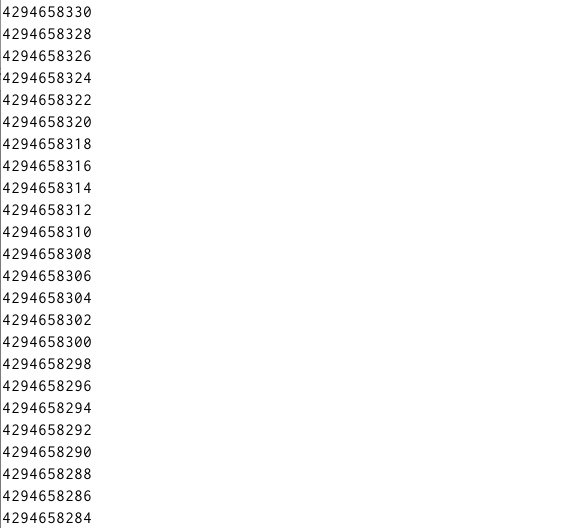


Рис. 5: Запуск исполняемого файла

В результате данных изменений цикл стал бесконечным и закольцованным. Далее внесем изменения (рис. 6) в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop.

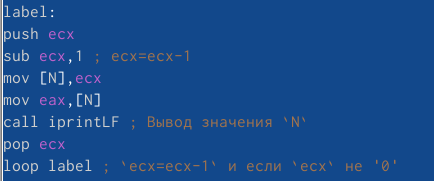


Рис. 6: Замена текста программы

Создадим исполняемый файл и проверим его работу (рис. 7)

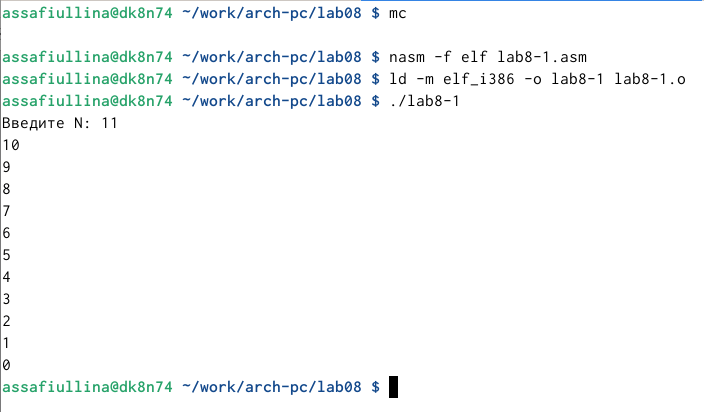


Рис. 7: Запуск исполняемого файла

После изменений программы число циклов стало соответствовать числу, введенному с клавиатуры.

1. Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы.

Создадим файл lab8-2.asm (рис. 8) в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.2. (рис. 8)

Рис. 8: Создание файла lab8-2.asm

Рис. 8: Создание файла lab8-2.asm

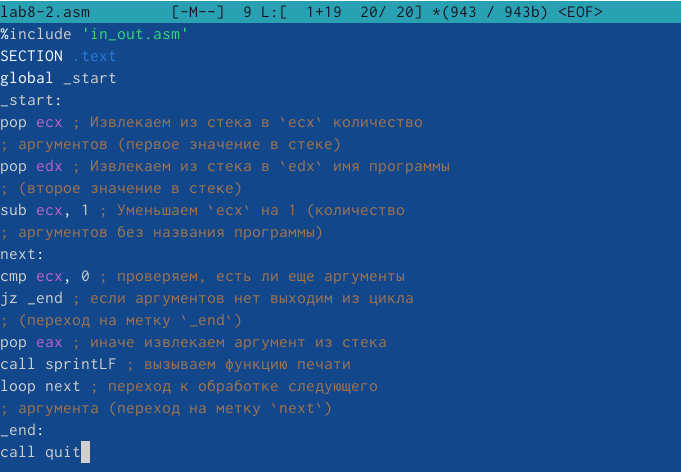


Рис. 9: Ввод текста из листинга 8.3

Создадим исполняемый файл и запустим его, указав аргументы в строке ./lab8-2 (рис. 10)

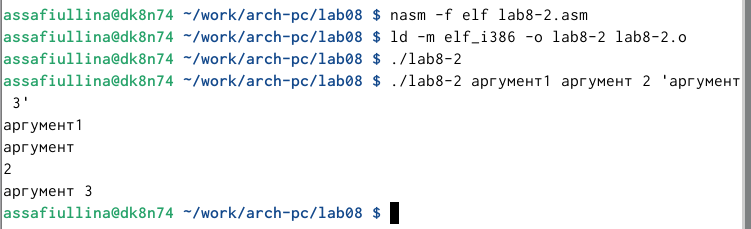


Рис. 10: Запуск исполняемого файла

Наша программа обработала 4 аргумента - аргумент1, аргумент, 2, ‘аргумент 3’

Создадим файл lab8-3.asm (рис. 11) в каталоге ~/work/archpc/lab08 и введем в него текст программы из листинга 8.3 (рис. 12)

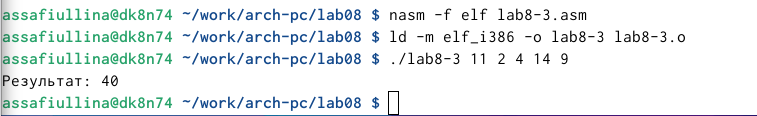


Рис. 11: Создание файла lab8-3.asm

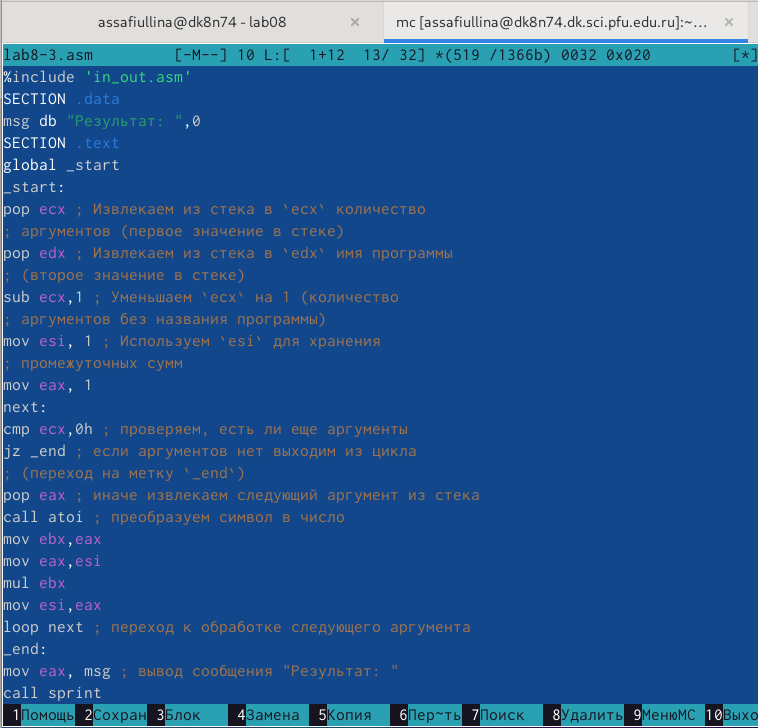


Рис. 12: Ввод текста из листинга 8.3

Поменяем текст программы из листинга 8.3, чтобы выводилось произведение аргументов командной строки

Создадим исполняемый файл и запустим его, указав аргументы (рис. 13)

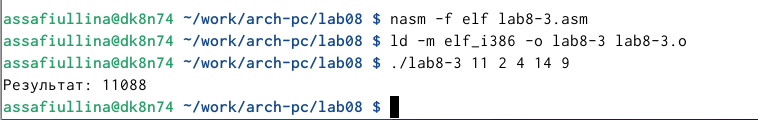


Рис. 13: Запуск исполняемого файла

1. Задание для самостоятельной работы

Создадим файл lab8-4.asm для выполнения самостоятельной работы (рис. 14)

Рис. 14: Создание файла lab8-4.asm

Рис. 14: Создание файла lab8-4.asm

Напишем программу (рис. 15) , которая находит сумму значений функции f(x) для x=x1,x2,…,xn, т.е. программа должна выводить значение f(x1)+f(x2)+…+f(xn). Значения xi передаются как аргументы. Так как у меня вариант 12, моя функция 15x-9.

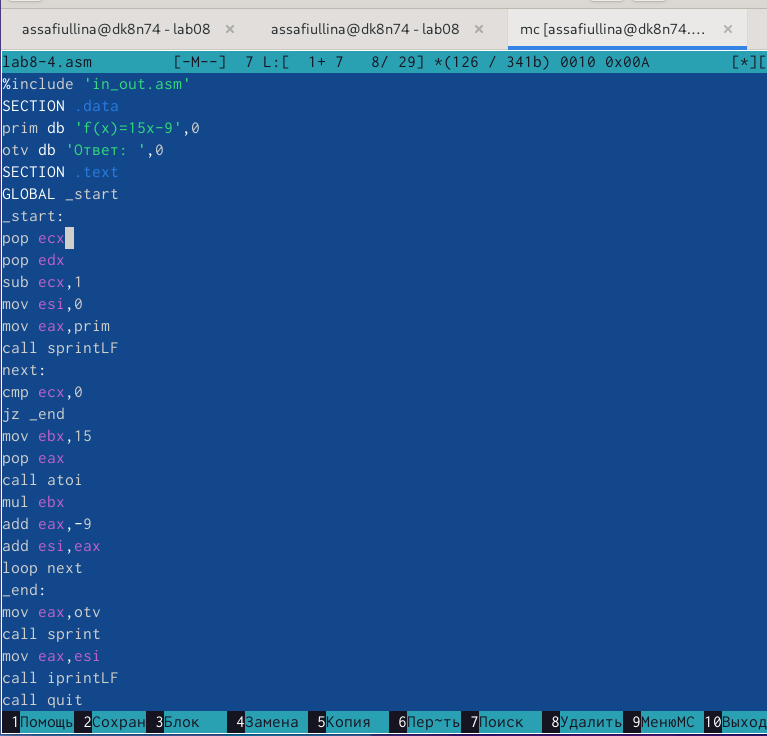


Рис. 15: Текст программы для функции 12

Создадим исполняемый файл и проверим его работу на нескольких наборах.(рис. 16), (рис. 17), (рис. 18)

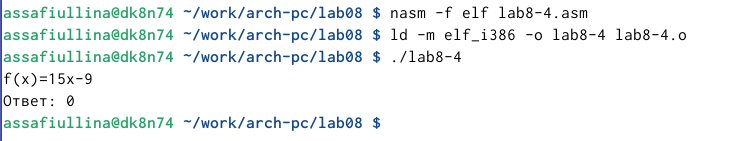


Рис. 16: Запуск программы

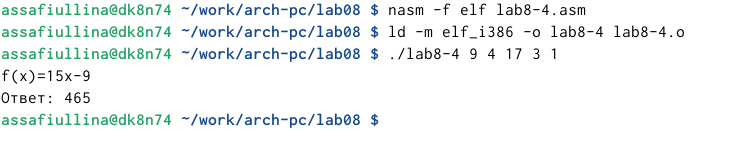


Рис. 17: Запуск программы

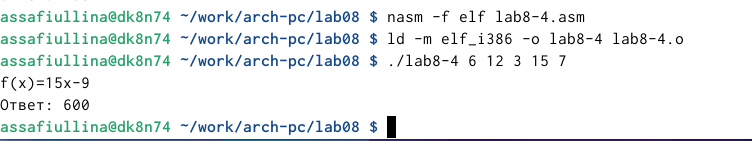


Рис. 18: Запуск программы

# 5 Выводы

В ходе выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов NASM и обработкой аргументов командной строки, а так же выполнила задание для самостоятельной работы.