Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютера

Репкина Елизавета Андреевна

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. 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. На рис. 8.1 показана схема организации стека в процессоре. 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). Добавление элемента в стек. 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. Извлечение элемента из стека. 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.

# 4 Выполнение лабораторной работы

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

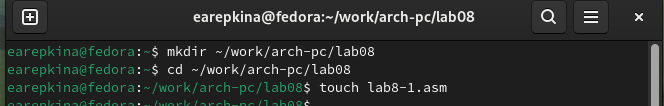


Рис. 1: Выполнение команд

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

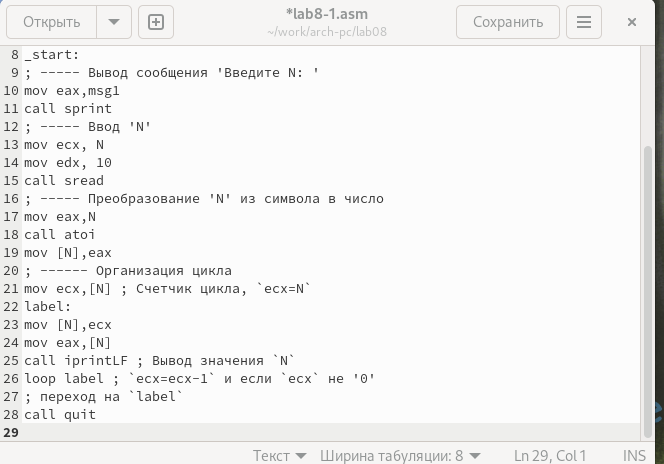


Рис. 2: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 3)

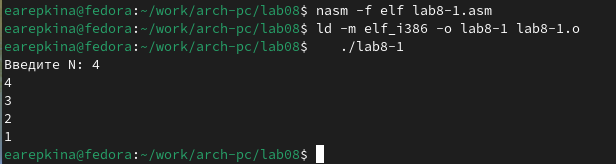


Рис. 3: Запуск файла

Данный пример показывает, что использование регистра ecx в теле цилка loop может привести к некорректной работе программы.

Меняю текст программы добавив изменение значение регистра ecx в цикле (рис. 4)

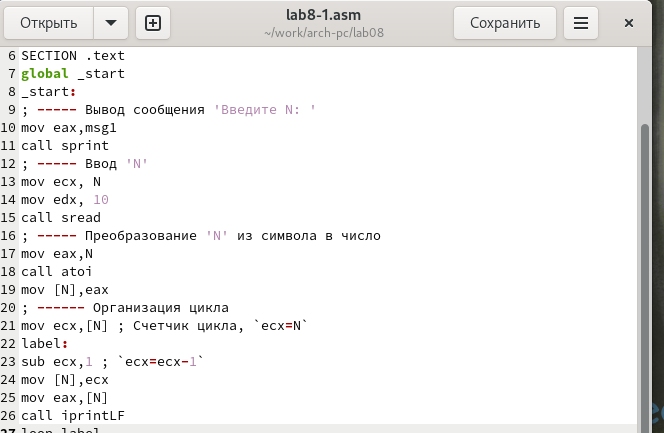


Рис. 4: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 5)

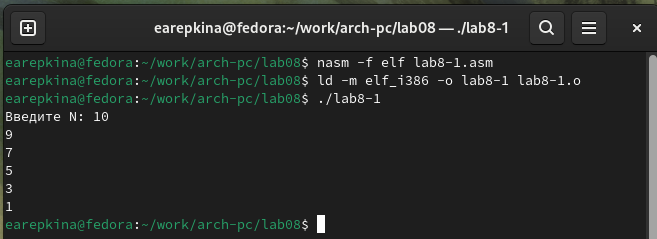


Рис. 5: Запуск файла

В данном случае число проходов цикла не соответствует значению N.

Вношу изменения в текст программы, добавив команды push и pop(рис. 6)

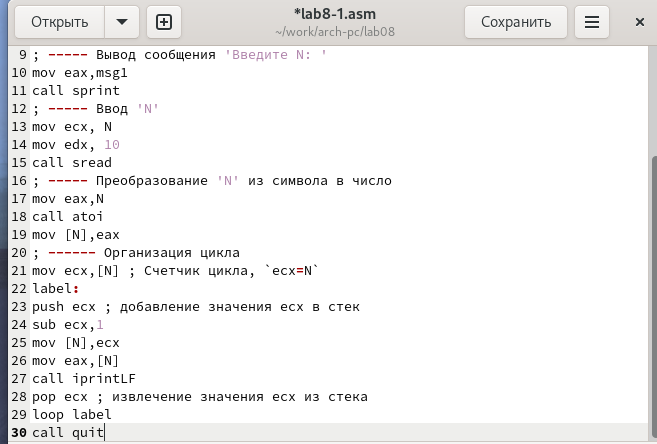


Рис. 6: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 7)

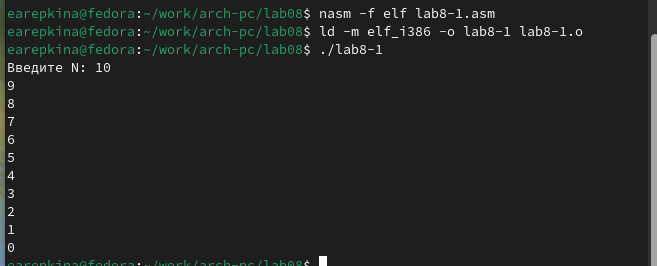


Рис. 7: Запуск файла

В данном случае число проходов цикла соответствует значению N.

1. Обработка аргументов командной строки

Создаю файл с названием lab8-2.asm (рис. 8)

Рис. 8: Создание файла

Рис. 8: Создание файла

ввожу в него текст программы из листинга 8.2 (рис. 9)

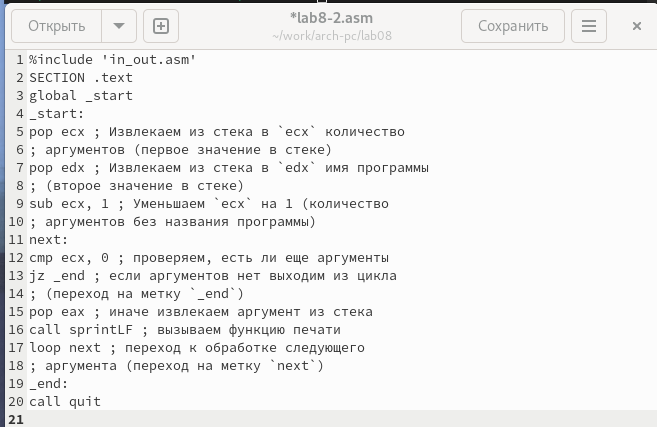


Рис. 9: Редактирование файла

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 10)

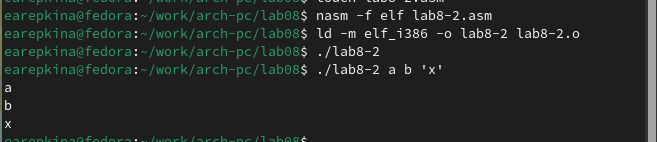


Рис. 10: Запуск исполняемого файла

Программа обработала 3 аргумента.

Создаю файл lab8-3.asm (рис. 11)

Рис. 11: Cоздание файла

Рис. 11: Cоздание файла

ввожу в него программу из листинга 8.3 (рис. 12)

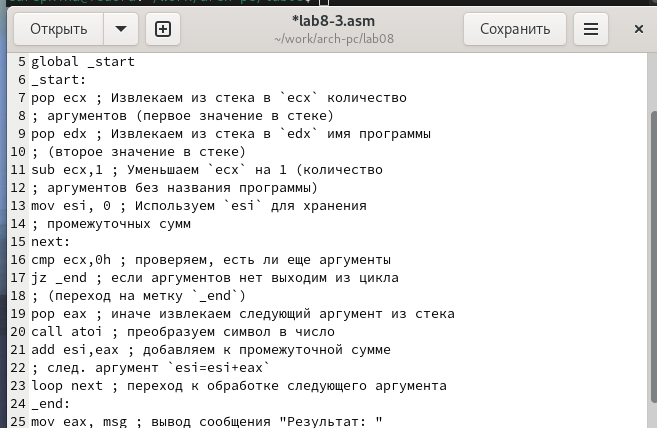


Рис. 12: Редактирование файла

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

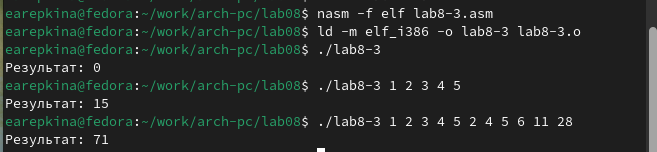


Рис. 13: Запуск исполняемого файла

Программа работает.

Теперь изменяю текст программы из листинга 8.3 так, чтобы она вычисляла произведение агрументов каждой строки (рис. 14)

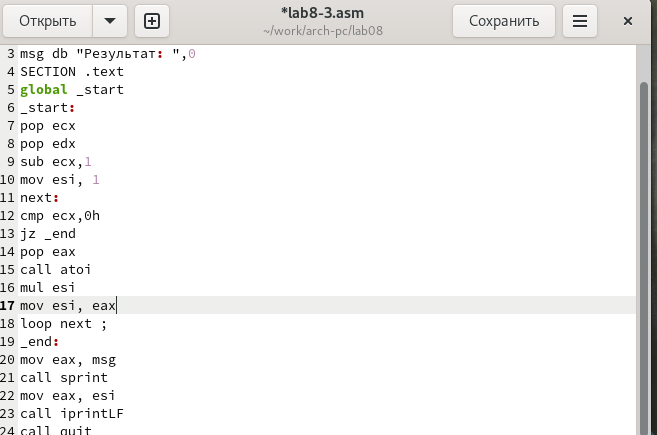


Рис. 14: Редактирование файла

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

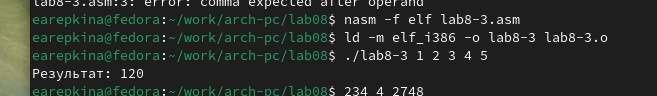


Рис. 15: Запуск исполняемого файла

1. Задания для самостоятельной работы

Создаю файл lab8-task.asm (рис. 16)

Рис. 16: Создание файла

Рис. 16: Создание файла

Начинаю написание программы, которая будет вычислять сумму значений f(x)=7(x+1).(вариант 14) (рис. 17)

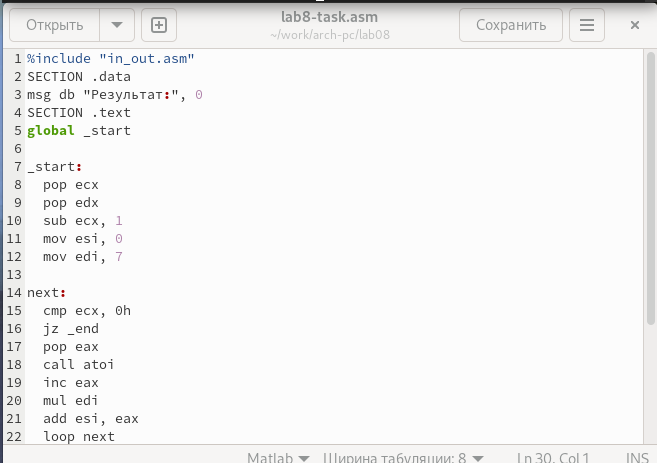


Рис. 17: Редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 18)

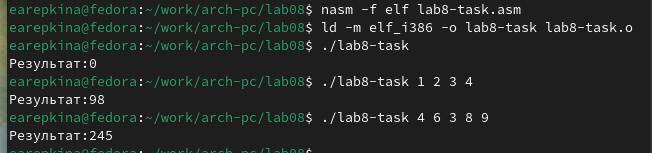


Рис. 18: Запуск исполняемого файла

Произведя несложные математические вычисления, делаю вывод, что программа работает верно

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, 7

next:

cmp ecx, 0h

jz \_end

pop eax

call atoi

inc eax

mul edi

add esi, eax

loop next

\_end:

mov eax, msg

call sprint

mov eax, esi

call iprintLF

call quit

# 5 Выводы

После выполнения данной лабораторной работы я приобрелa навыки написание программ с использованием циклов и обработкой агрументов командной строки

# Список литературы