Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютера

Осина Виктория Александровна

Содержание

# 1 Цель работы

Целью данной лабораторной работы ялвяется 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. На рис. 8.1 показана схема организации стека в процессоре.

Стек имеет вершину, адрес последнего добавленного элемента, который хранится в ре- гистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указа- тель 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 ин- струкция loop.

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 lab08 для программ лабораторной работы №8, перехожу в него и создаю файл lab8-1.asm, проверяю, что файл создан (рис. [1](#fig:001))

Figure 1: Создание каталога lab08 и файла lab8-1.asm в нем

Figure 1: Создание каталога lab08 и файла lab8-1.asm в нем

Перед работой с программами копирую файл in\_out.asm в каталог и проверяю, что файл находится в нужном каталоге (рис. [2](#fig:002)) (рис. [3](#fig:003))

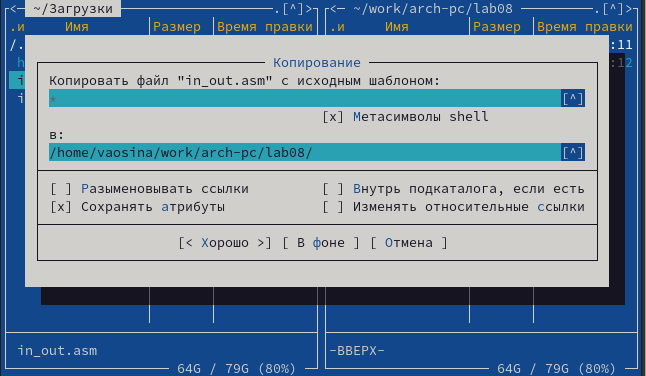


Figure 2: Копирование файла in\_out.asm

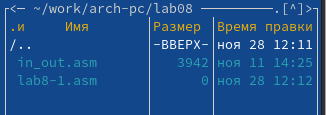


Figure 3: Проверка, что файл находится в нужном каталоге

Открываю lab8-1.asm в редакторе и ввожу в него текст программы, которая выводит значение регистра ecx (рис. [4](#fig:004)) и (рис. [5](#fig:005)).

Figure 4: Открытие файла в редакторе

Figure 4: Открытие файла в редакторе



Figure 5: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его (рис. [6](#fig:006)). В результате выводятся цифры от N до 1.

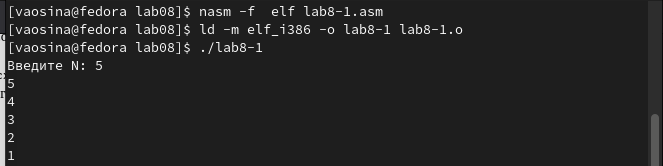


Figure 6: Создание исполняемого файла и его запуск

Изменяю текст программы добавив значение регистра ecx в цикле (рис. [7](#fig:007))



Figure 7: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. [8](#fig:008)). Теперь программа работает некорректно и цикл не прерывается. Число проходов цикла не соответствует значению N, введенному с клавиатуры.

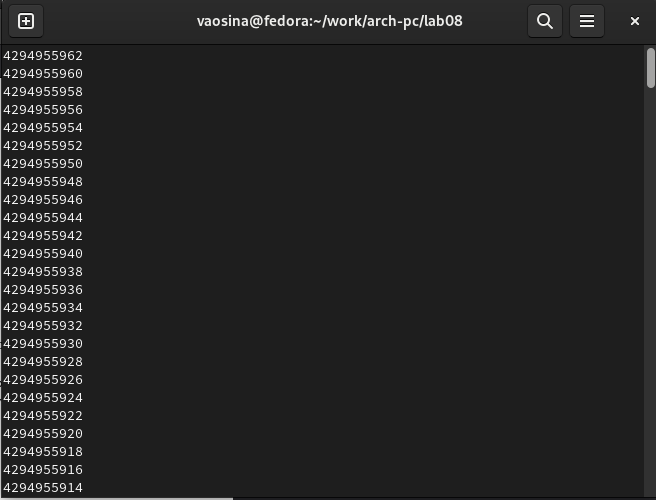


Figure 8: Создание исполняемого файла и его запуск

Для использования регистра ecx в цикле и сохранения корректности работы программы можно использовать стек. Поэтому вношу изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop. (рис. [9](#fig:009)).

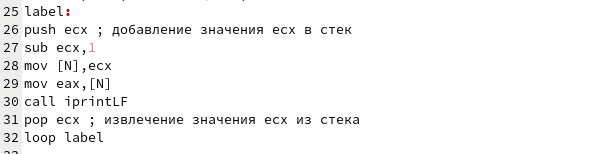


Figure 9: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. [10](#fig:010)) В результате выводят цифры от N-1 до 0 включительно. Число проходов цикла соответствует значению N, введенному с клавиатуры.

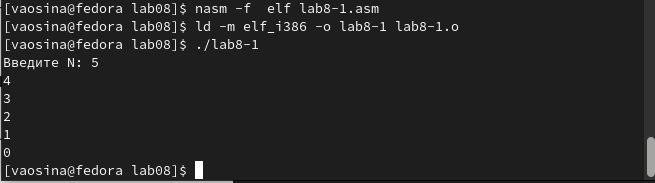


Figure 10: Создание исполняемого файла и его запуск

## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и открываю его в редакторе (рис. [11](#fig:011)).

Figure 11: Создание файла lab8-2.asm и его открытие в редакторе

Figure 11: Создание файла lab8-2.asm и его открытие в редакторе

Ввожу в файл текст программы,которая выводит на экран аргументы командной строки (рис. [12](#fig:012)).

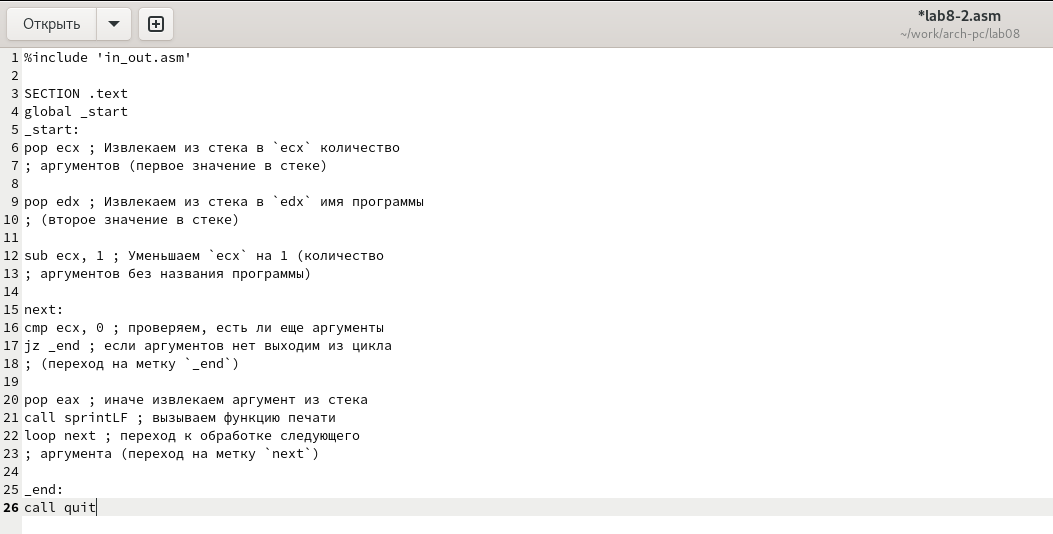


Figure 12: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его.(рис. [13](#fig:013)). Командой было обработано 4 аргумента, так как они разделены пробелом, поэтому, если аргумент содержит пробел, его надо заключать в кавычки

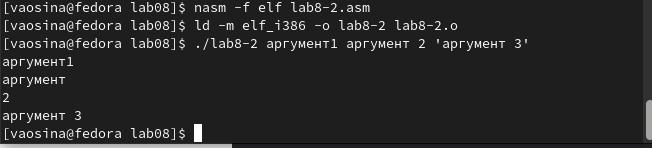


Figure 13: Создание исполняемого файла и его запуск

Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и открываю его в редакторе (рис. [14](#fig:014)).

Figure 14: Создание файла lab8-3.asm и его открытие в редакторе

Figure 14: Создание файла lab8-3.asm и его открытие в редакторе

Ввожу в файл текст программы, которая выводит сумму чисел, которые пере- даются в программу как аргументы. (рис. [15](#fig:015))

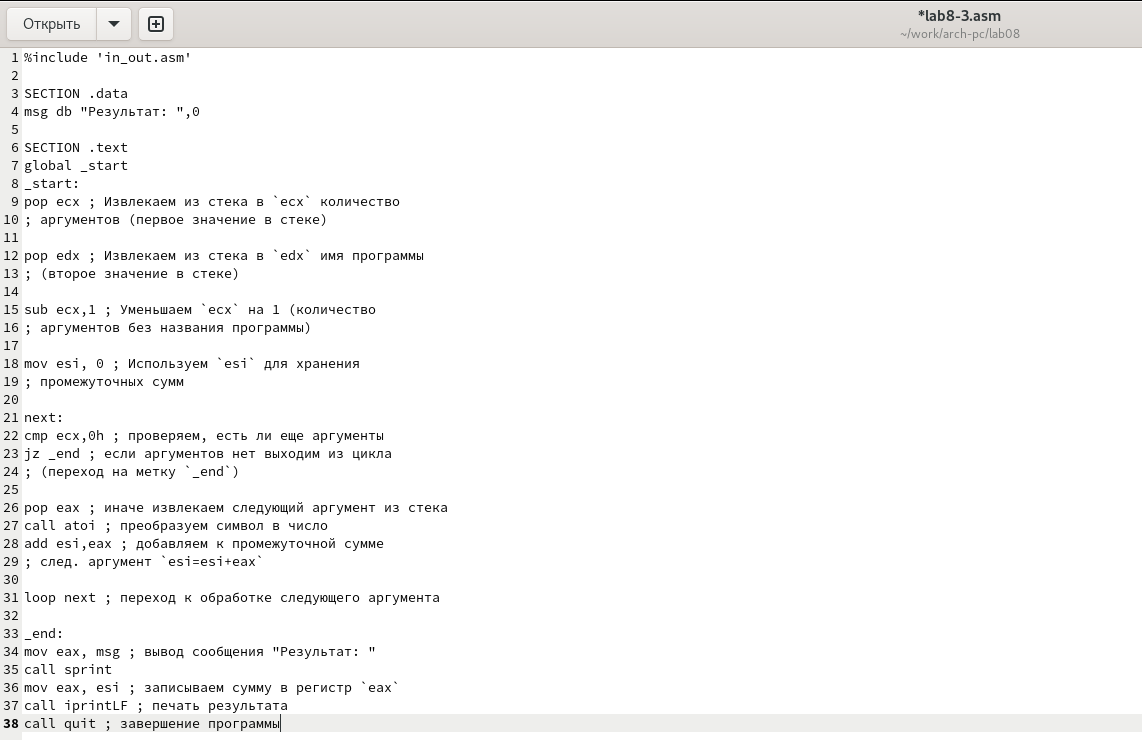


Figure 15: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его. (рис. [16](#fig:016)) и (рис. [17](#fig:017))

Программа работает корректно.

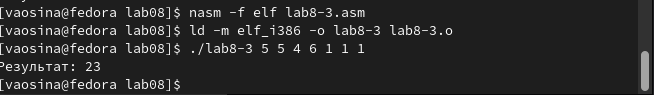


Figure 16: Создание исполняемого файла и его запуск

Изменяю текст программы так, чтобы вместо суммы чисел выводилось произведение аргументов командной строки

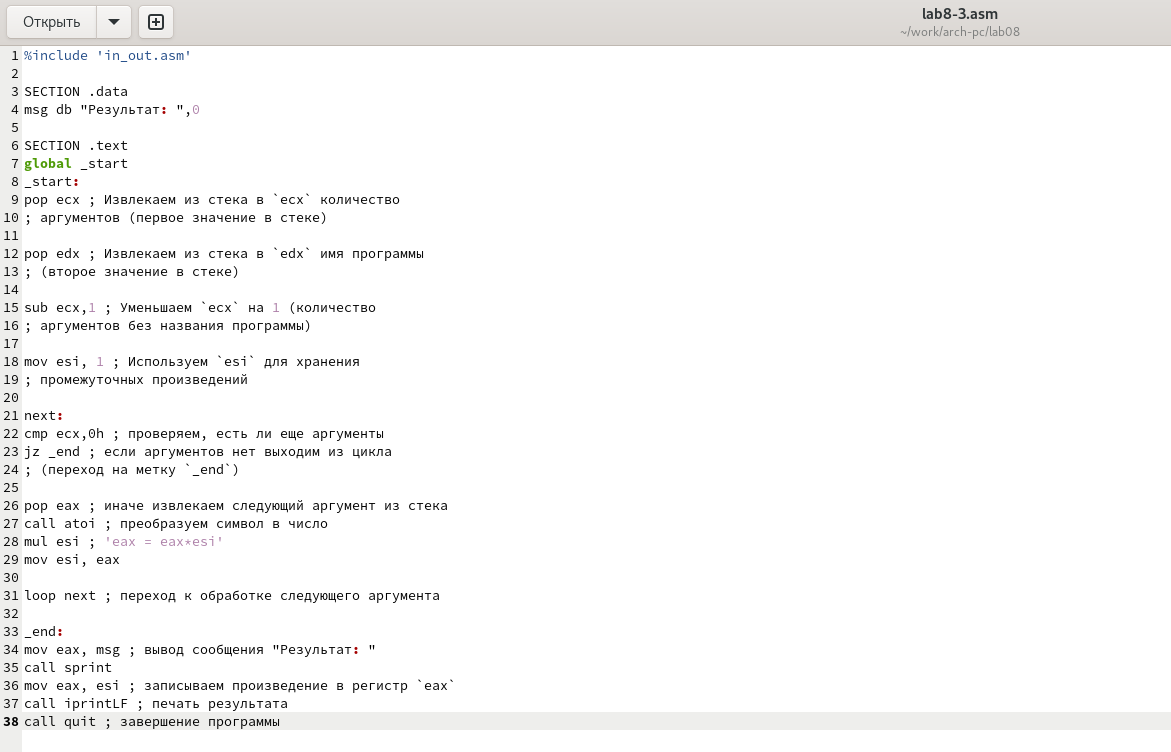


Figure 17: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. [18](#fig:018))

Программа работает корректно.

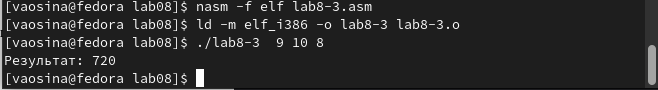


Figure 18: Создание исполняемого файла и его запуск

## 4.3 Выполнение задания для самостоятельной работы

Необходимо написать программу, 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.

В соответствии с моим варинтом (вариант7), вид функции 3(х + 2).

Создаю файл var7.asm в каталоге ~/work/arch-pc/lab08 и открываю его в редакторе (рис. [19](#fig:019))

Figure 19: Создание файла var7.asm

Figure 19: Создание файла var7.asm

Ввожу в файл текст программы (рис. [20](#fig:020)).

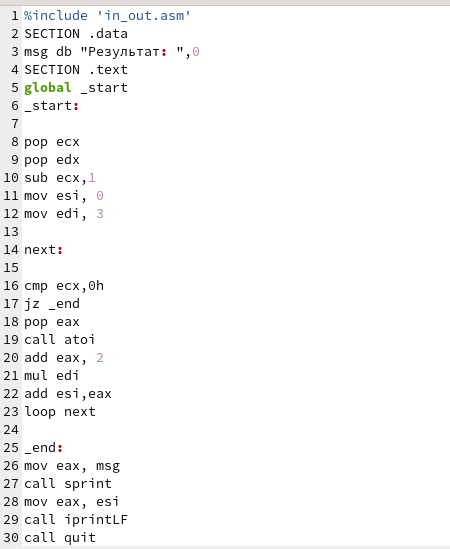


Figure 20: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его. (рис. [21](#fig:021)) и (рис. [22](#fig:022)) Проверяю работу программы на нескольких наборах x = x1, x2, …, xn. Программа работает корректно

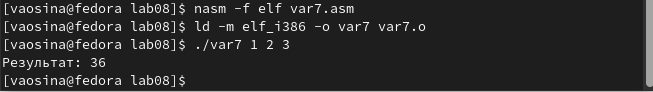


Figure 21: Создание исполняемого файла и его запуск

Figure 22: Второй набор значений

Figure 22: Второй набор значений

**Проверка**

```3(x+2), x1 = 1, x2 = 2, x3 = 3

3(1+2) + 3(2+2) + 3(3+2) = 3*3 + 3*4 + 3\*5 = 9 + 12 + 15 = 36

3(x+2), x1 = 8, x2 = 3, x3 = 6

3(8+2) + 3(3+2) + 3(6+2) = 3*10 + 3*5 + 3\*8 = 30 + 15 + 24 = 69

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, 2  
mul edi  
add esi,eax  
loop next  
  
\_end:  
mov eax, msg  
call sprint  
mov eax, esi  
call iprintLF  
call quit

# 5 Выводы

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

# 6 Список литературы

1. [Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089095/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%968.%20%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D1%86%D0%B8%D0%BA%D0%BB%D0%B0.%20%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0%20%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2%20%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B9%20%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8..pdf)