Лабораторная работа №8

Дисциплина: Архитектура компьютера

Малюга Валерия Васильевна

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.

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

Создала каталог для программ лабораторной работы № 8, перешла в него и создала файл lab8-1.asm (рис. [1](#fig:001)).

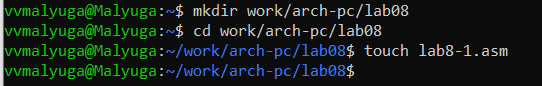


Figure 1: Создание каталога и файла lab8-1.asm

Ввела в файл lab8-1.asm текст программы из листинга 8.1. Создала исполняемый файл и проверила его работу (рис. [2](#fig:002)). Программа работает корректно, выводит числа от N до 1 включительно.

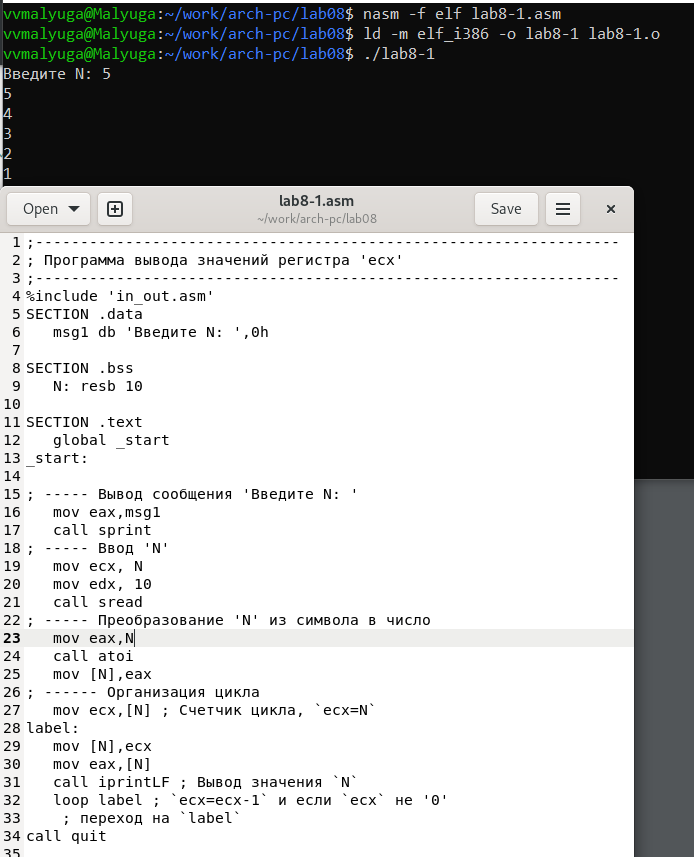


Figure 2: Создание исполняемого файла и его запуск

Изменила текст программы, добавив изменение значение регистра ecx в цикле. Создала исполняемый файл и проверила его работу (рис. [3](#fig:003)). В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

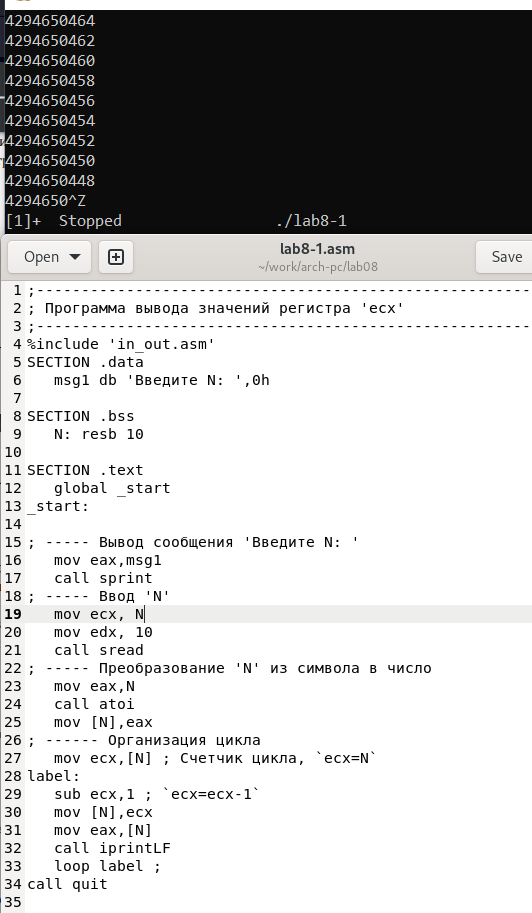


Figure 3: Изменение кода и запуск программы

Внесла изменения в текст программы, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. [4](#fig:004)). Создала исполняемый файл и проверила его работу. В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

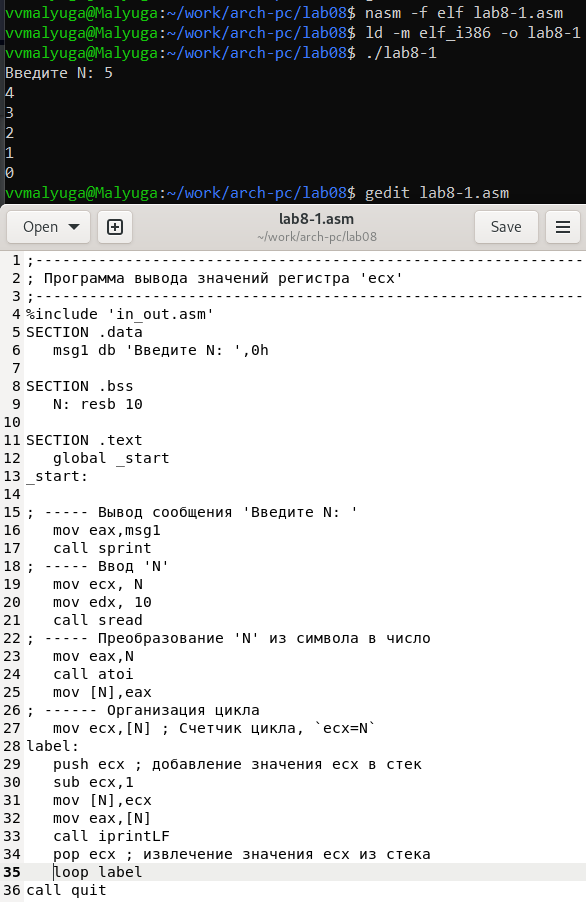


Figure 4: Изменение кода и запуск программы

## 4.2 Обработка аргументов командной строки

Создала файл lab8-2.asm в каталоге work/arch-pc/lab08 и ввела в него текст программы из листинга 8.2. Создала исполняемый файл и запустила его, указав аргументы: аргумент1 аргумент 2 ‘аргумент 3’ (рис. [5](#fig:005)). В данном случае программой было обработано 4 аргумента.

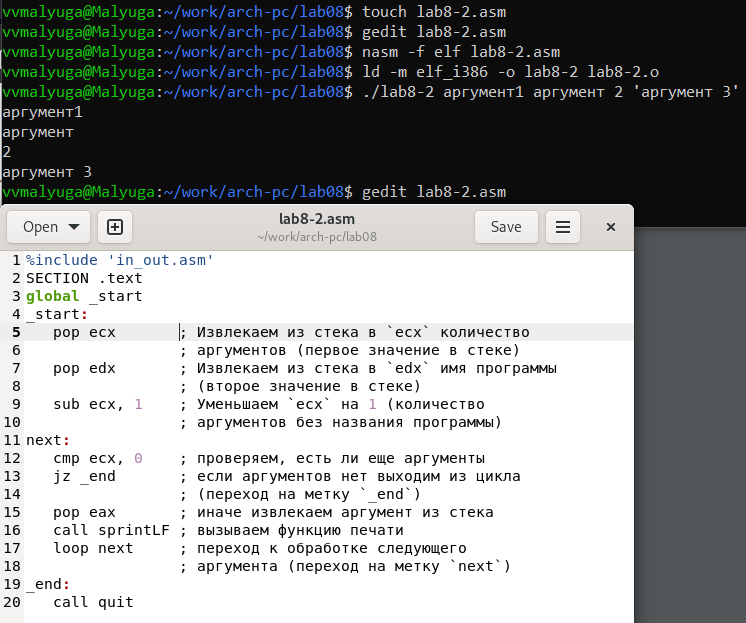


Figure 5: Создание исполняемого файла и его запуск

Создала файл lab8-3.asm в каталоге work/arch-pc/lab08 и ввела в него текст программы из листинга 8.3. Создала исполняемый файл и запустила его, указав аргументы: 12 13 7 10 5 (рис. [6](#fig:006)).

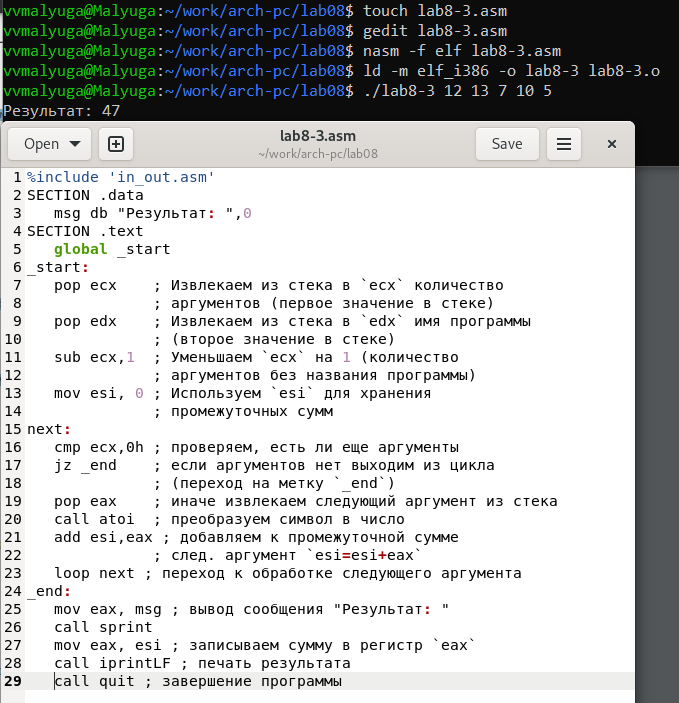


Figure 6: Создание исполняемого файла и его запуск

Изменила текст программы из листинга 8.3 для вычисления произведения аргументов командной строки (рис. [7](#fig:007)). Для этого в 13 строке изменила изначальное значение счетчика на 1, в 21 строке изменила add на imul.  
Прилагаю код:

%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 промежуточного результата  
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 esi, eax ; умножаем промежуточный результат на 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

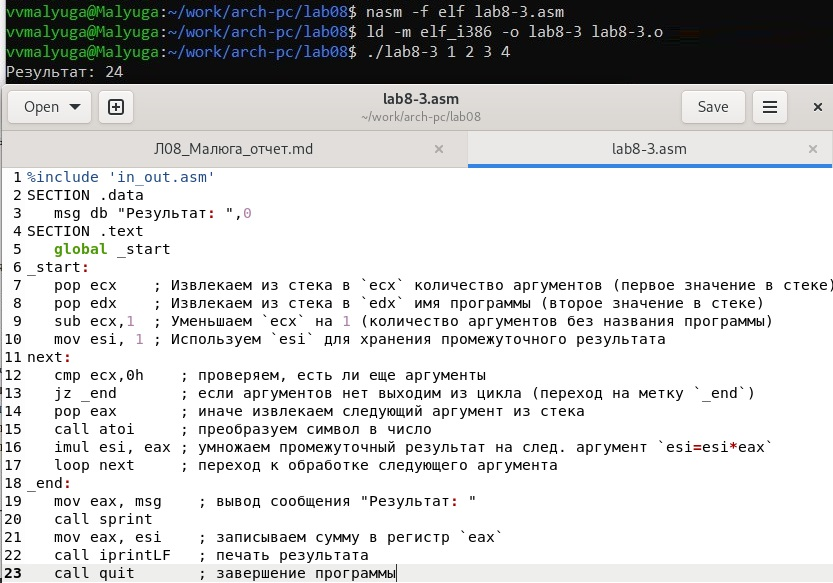


Figure 7: Изменение кода и запуск программы

## 4.3 Задание для самостоятельной работы

Написала программу, которая находит сумму значений функции f(x) = 15x + 2 (Вариант 11) для x = x1, x2, …, xn. Создала исполняемый файл и проверила его работу на нескольких наборах x = x1, x2, …, xn (рис. [8](#fig:008)).

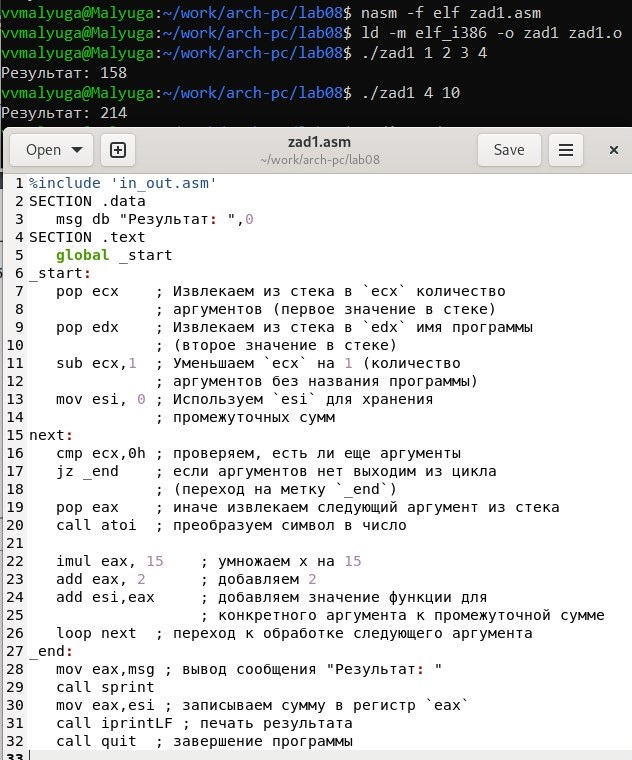


Figure 8: Запуск исполняемого файла и проверка его работы

Прилагаю код:

%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 x на 15  
 add eax, 2 ; добавляем 2   
 add esi,eax ; добавляем значение функции для   
 ; конкретного аргумента к промежуточной сумме  
 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

Благодаря данной лабораторной работе я приобрела навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ.