Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Соловьев Богдан Михайлович

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

# 2 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.

# 3 Выполнение лабораторной работы

Создаю файл lab8-1.asm (рис. [1](#fig:001)).

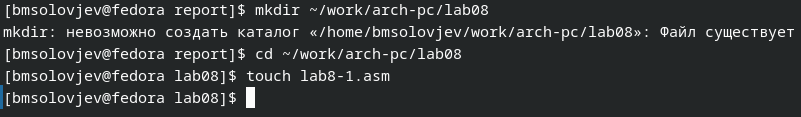


Figure 1: Создание файла

Записываю в созданный файл следующий код (рис. [2](#fig:002)).

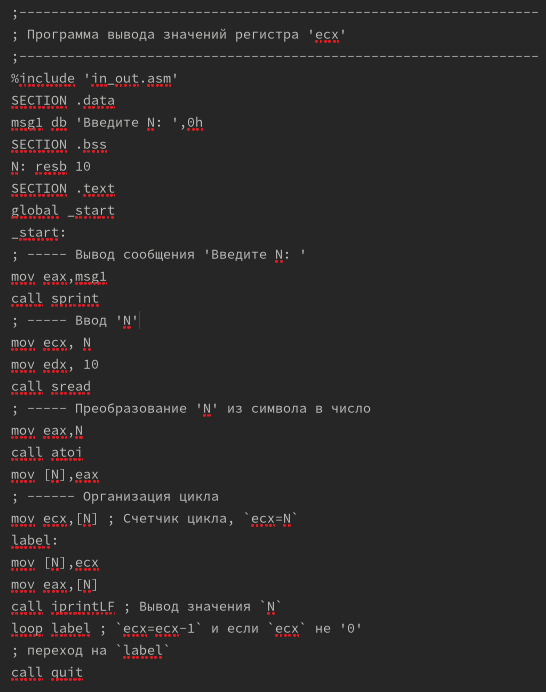


Figure 2: Код программы

Проверка работы созданного файла (рис. [3](#fig:003)).

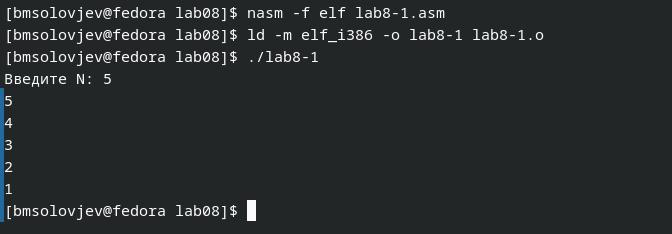


Figure 3: Выполнение программы

Изменяю код программы, теперь значение в регистре ecx не запоминается, и количество циклов становится бесконечным (рис. [4](#fig:004)).

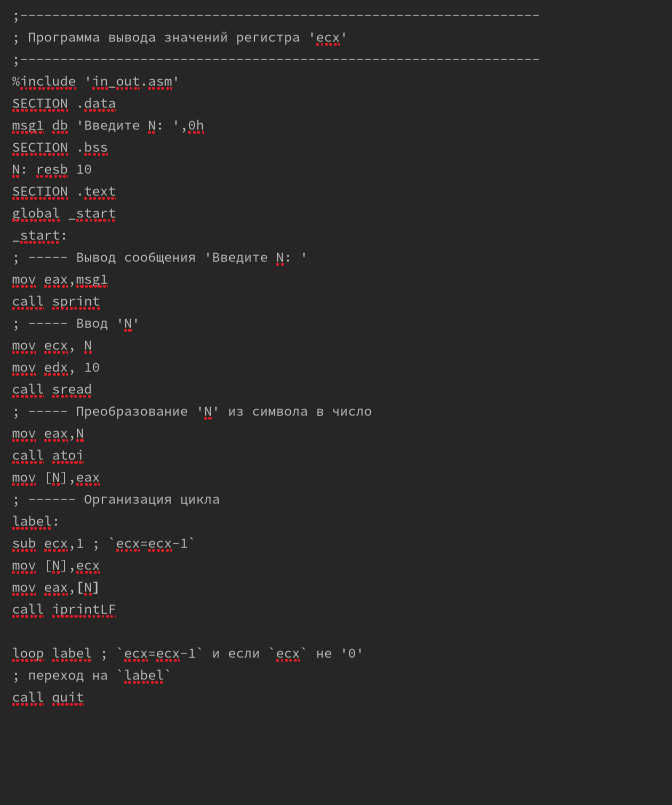


Figure 4: Неправильный код программы

Файл создаёт бесконечный цикл (рис. [5](#fig:005)).

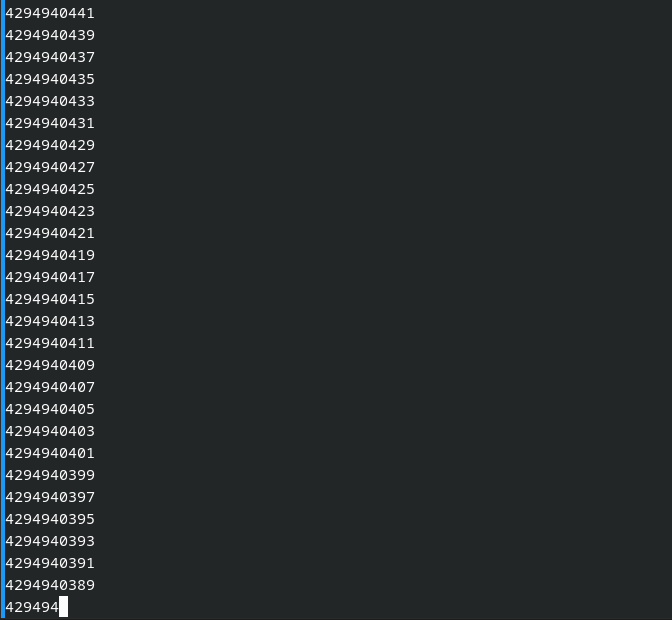


Figure 5: Бесконечный цикл

Изменяю код программы таким образом, чтобы в заносить и вытаскивать из стека значение счётчика цикла (рис. [6](#fig:006)).

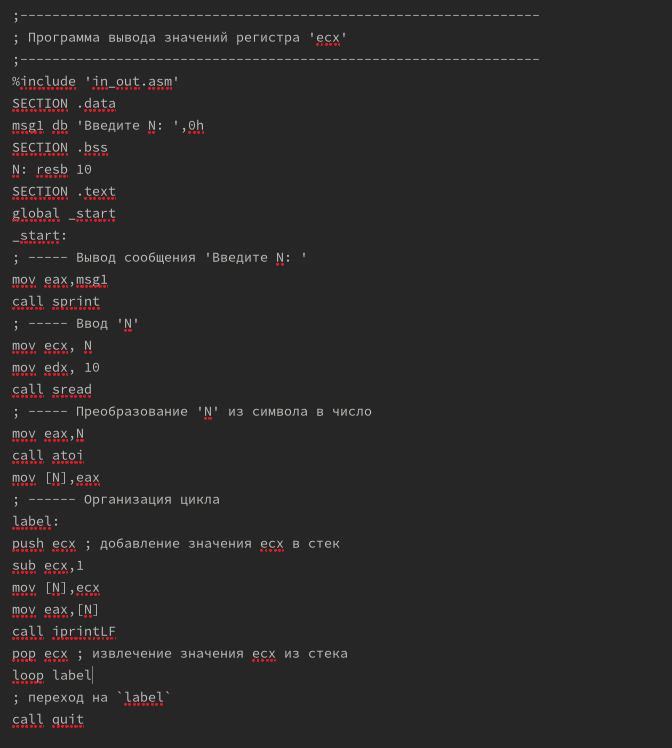


Figure 6: Поправленный код

Теперь программа снова работает корректно, и количество циклов соответствует N (рис. [7](#fig:007)).

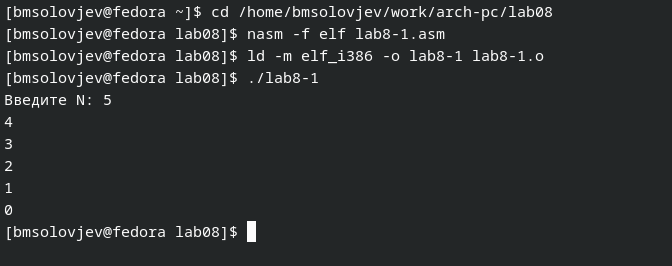


Figure 7: Программа работает

Создаю новый файл и ввожу в него новый текст программы (рис. [8](#fig:008)).

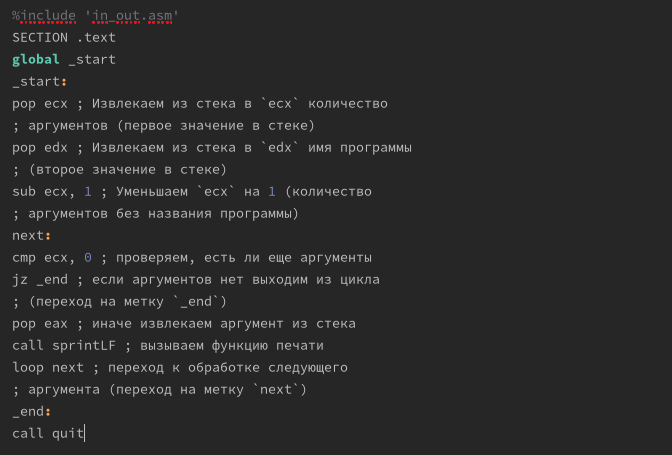


Figure 8: Текст файла lab8-2.asm

Проверка работы нового файла. При таком запросе программа обработала 4 аргумента (рис. [9](#fig:009)).

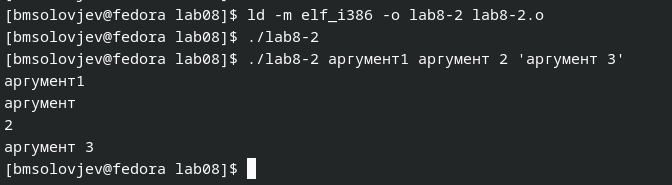


Figure 9: Программа принимает аргументы

Создаю новую программу, которая вычисляет сумму аргументов командной строки (рис. [10](#fig:010)).

%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  
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

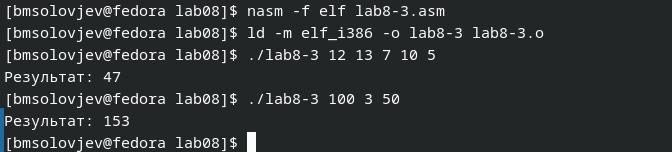


Figure 10: Программа считает сумму

Теперь меняю текст программы таким образом, чтобы она вычисляла произведение (рис. [11](#fig:011)).

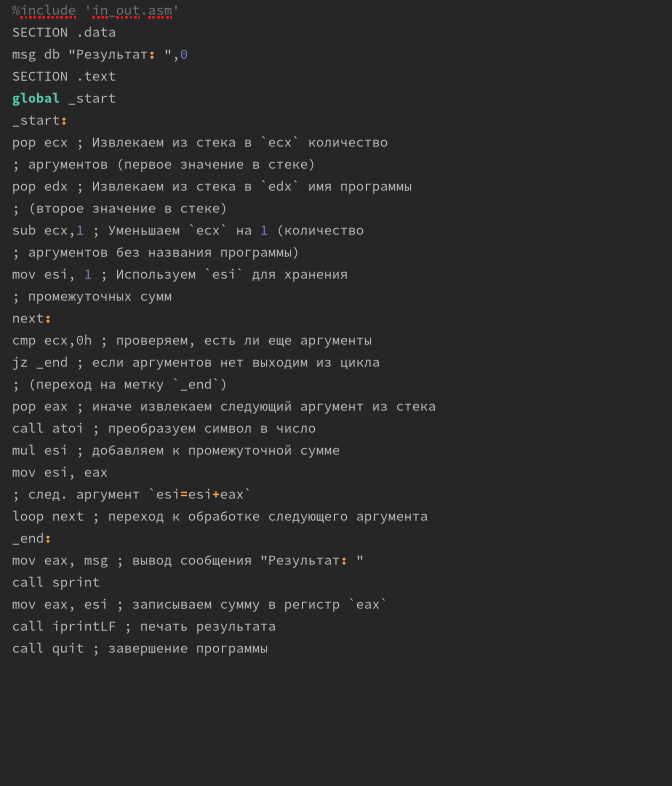


Figure 11: Программа считает произведие

Проверка работы нового файла. (рис. [12](#fig:012)).

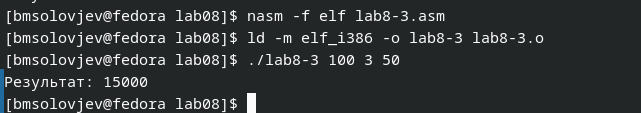


Figure 12: Программа считает произведение

# 4 Самостоятельная работа

Текст программы, считающей сумму значений функции f(x) = 10x -5 (рис. [13](#fig:013)).

%include 'in\_out.asm'  
SECTION .data  
msg0 db "f(x) = 10x - 5",0  
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  
mov ebx, 10  
mul ebx  
add eax, -5  
add esi, eax  
; след. аргумент `esi=esi+eax`  
loop next ; переход к обработке следующего аргумента  
\_end:  
mov eax, msg0  
call sprintLF  
mov eax, msg ; вывод сообщения "Результат: "  
call sprint  
mov eax, esi ; записываем сумму в регистр `eax`  
call iprintLF ; печать результата  
call quit ; завершение программы

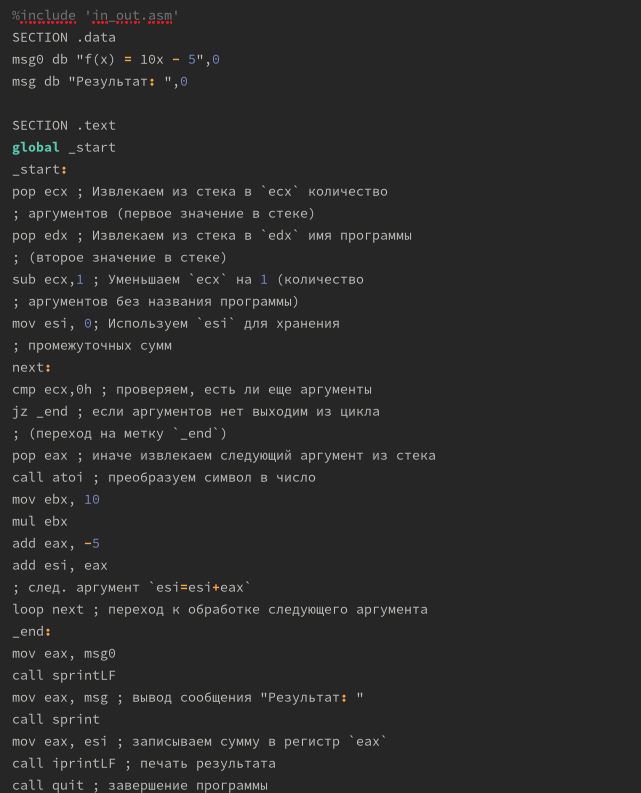


Figure 13: Код моей программы

Проверка работы программы (рис. [14](#fig:014)).

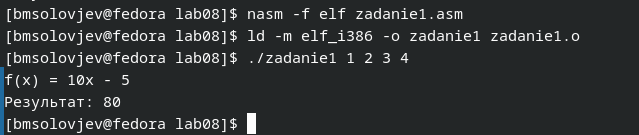


Figure 14: Проверка работы моей программы

# 5 Выводы

Выполнив эту лабораторную работу, я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки.

# Список литературы