РУДН. Архитектура компьютеров

Отчёт по лабораторной работе №8

Косинов Никита Андреевич, НПМбв-02-20

Содержание

# 1 Цель работы

Многие логики программ требуют повторять одну и ту же оперцию несколько раз: фиксированное, либо зависящее от параметров, заданных пользователем. В любом случае, в Ассемблере предусмотрена такая возможность.

Цель данной работы - научиться писать программы, использующие логику циклов, а также познакомиться со структорой стека.

# 2 Ход работы

Лабораторная работа выполнена с использованием консоли **OC Linux** и языка программирования ассемблера **NASM**.

1. Реализация цикла фиксированной длины;
2. Реализация цикла, зависящего от аргументов;
3. Работа с аргументами.

В конце выполнена самостоятельная работа.

# 3 Реализация цикла фиксированной длины

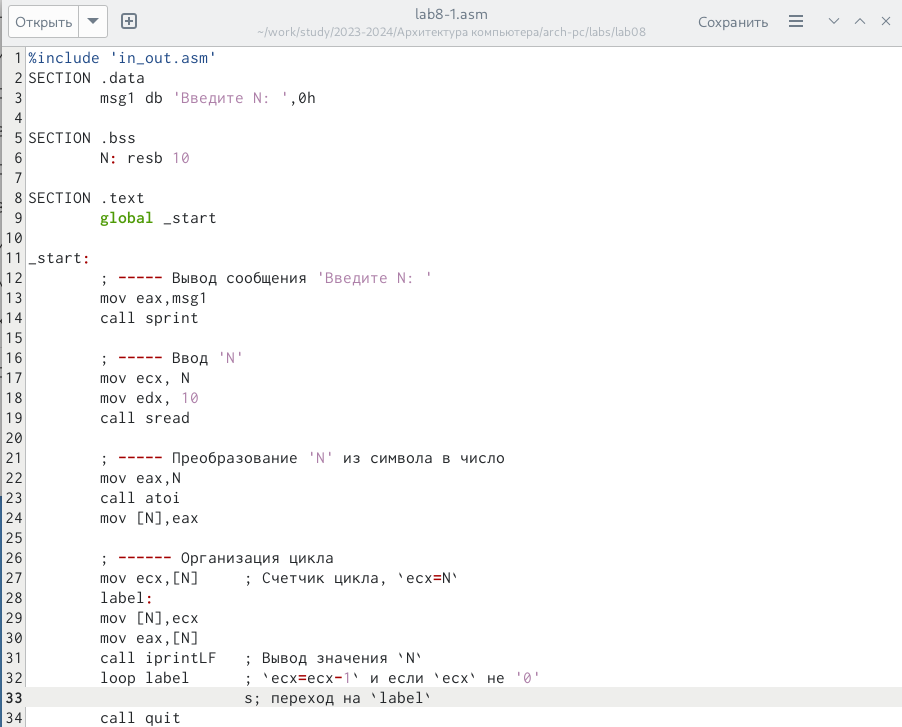
Чтобы организовать цикл в **NASM**, можно использовать инструкцию **loop**. Она уменьшает значение, хранящееся в регистре **ecx** на 1, и, если получает не ноль, переходит на указанную метку. Таким образом, метка указанная над командой **loop** возвращает нас на несколько шагов назад, но с новым значением **ecx**. Так мы получаем цикл на **ecx** операций.

1. Создаём рабочий файл.

Создание файла

Создание файла

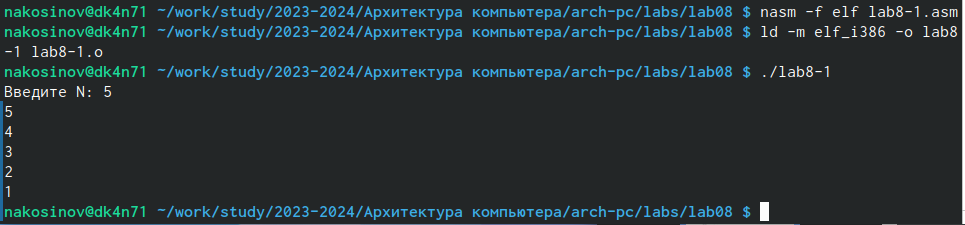
1. Пишем код с использованием цикла длины **N**, где последний вводится пользователем.



Код программы lab8-1

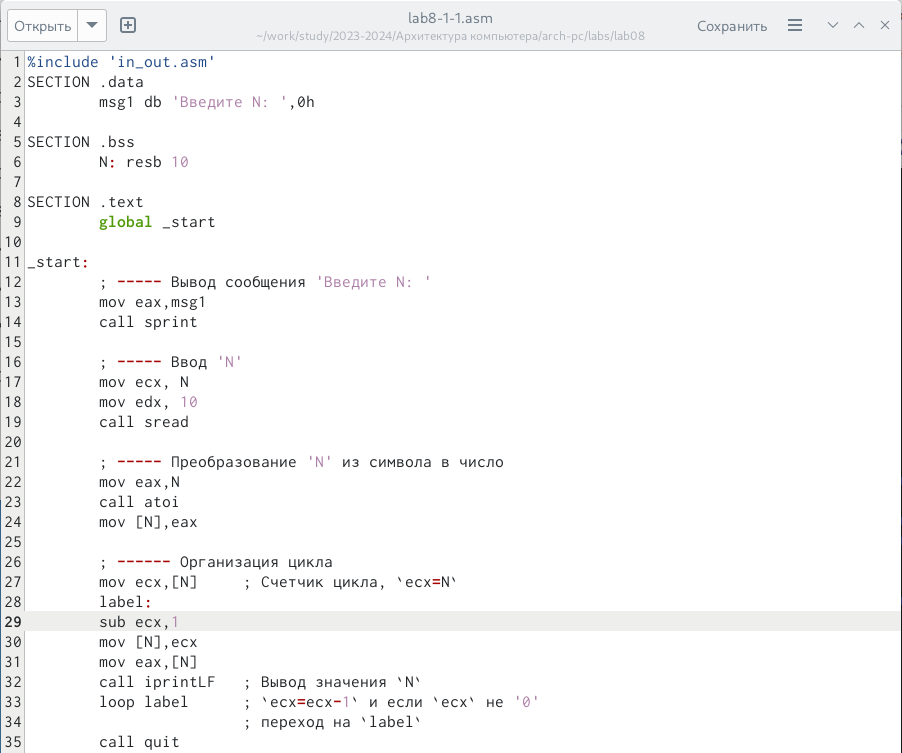
%include 'in\_out.asm'  
SECTION .data  
 msg1 db 'Введите N: ',0h  
  
SECTION .bss  
 N: resb 10  
  
SECTION .text  
 global \_start  
  
\_start:  
 ; ----- Вывод сообщения 'Введите N: '  
 mov eax,msg1  
 call sprint  
   
 ; ----- Ввод 'N'  
 mov ecx, N  
 mov edx, 10  
 call sread  
   
 ; ----- Преобразование 'N' из символа в число  
 mov eax,N  
 call atoi  
 mov [N],eax  
   
 ; ------ Организация цикла  
 mov ecx,[N] ; Счетчик цикла, `ecx=N`  
 label:  
 mov [N],ecx  
 mov eax,[N]  
 call iprintLF ; Вывод значения `N`  
 loop label ; `ecx=ecx-1` и если `ecx` не '0'  
 ; переход на `label`  
 call quit

1. Компилируем и запускаем. Изначально в **ecx** записывает число 5, далее каждую итерацию мы выводим его на экран и уменьшаем на 1. По достижении 0 программа завершает работу



Исполнение lab8-1

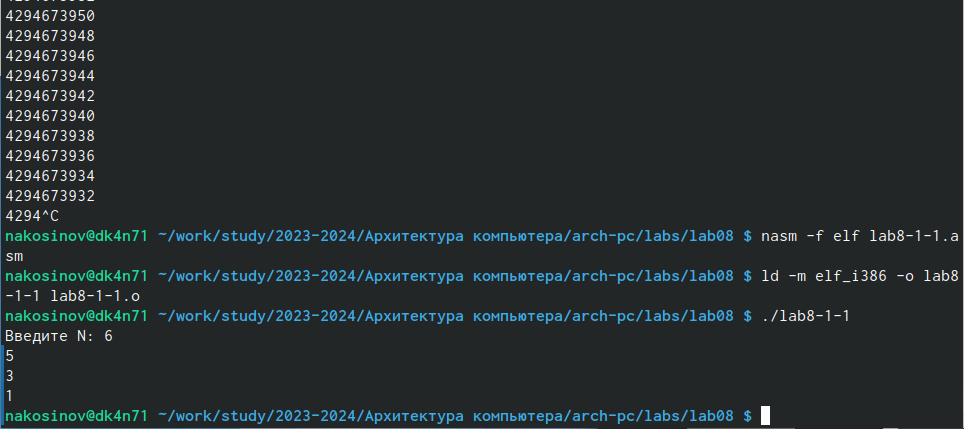
1. Изменим работу программы, добавив уменьшение **ecx** на 1 вручную.



Код программы lab8-1-1

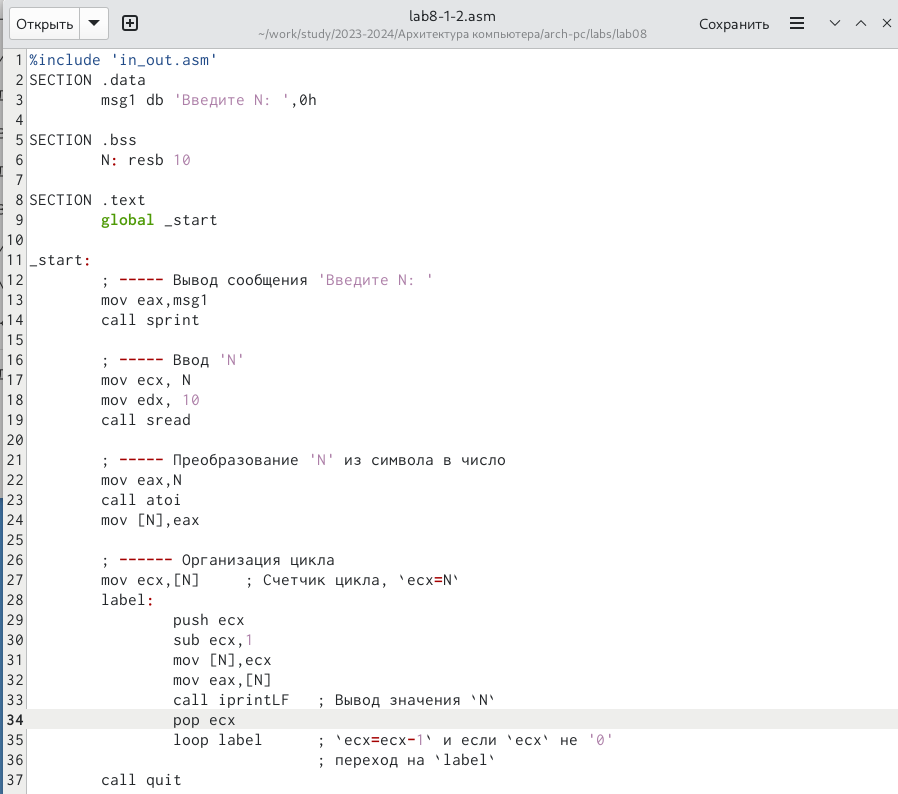
%include 'in\_out.asm'  
SECTION .data  
 msg1 db 'Введите N: ',0h  
  
SECTION .bss  
 N: resb 10  
  
SECTION .text  
 global \_start  
  
\_start:  
 ; ----- Вывод сообщения 'Введите N: '  
 mov eax,msg1  
 call sprint  
   
 ; ----- Ввод 'N'  
 mov ecx, N  
 mov edx, 10  
 call sread  
   
 ; ----- Преобразование 'N' из символа в число  
 mov eax,N  
 call atoi  
 mov [N],eax  
   
 ; ------ Организация цикла  
 mov ecx,[N] ; Счетчик цикла, `ecx=N`  
 label:  
 sub ecx,1  
 mov [N],ecx  
 mov eax,[N]  
 call iprintLF ; Вывод значения `N`  
 loop label ; `ecx=ecx-1` и если `ecx` не '0'  
 ; переход на `label`  
 call quit

1. Запустим, попробовав **N=7** и **N=6**. Во втором случае видно, что выводятся числа с шагом 2. Действительно, **ecx** уменьшается дважды на 1 за одну итерацию: нами и инструкцией **loop**. В первом случае же, раз **N** нечётно, после 1 мы получим -1, т.е. не 0, и программы продолжит работу. При этом, -1 в **ecx** - это самое большое число, поэтому мы видим такой результат.



Исполнение lab8-1-1

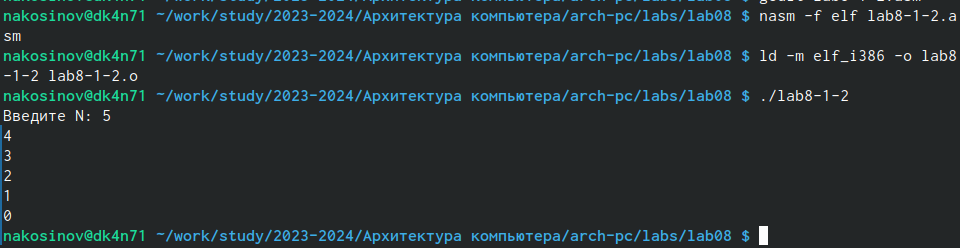
1. Для исправления такого рода недочёта, можно перед изменением значения **ecx** его сохранить, например, в стек. А перед концом шага цикла достать обратно.



Код программы lab8-1-2

%include 'in\_out.asm'  
SECTION .data  
 msg1 db 'Введите N: ',0h  
  
SECTION .bss  
 N: resb 10  
  
SECTION .text  
 global \_start  
  
\_start:  
 ; ----- Вывод сообщения 'Введите N: '  
 mov eax,msg1  
 call sprint  
   
 ; ----- Ввод 'N'  
 mov ecx, N  
 mov edx, 10  
 call sread  
   
 ; ----- Преобразование 'N' из символа в число  
 mov eax,N  
 call atoi  
 mov [N],eax  
   
 ; ------ Организация цикла  
 mov ecx,[N] ; Счетчик цикла, `ecx=N`  
 label:  
 push ecx  
 sub ecx,1  
 mov [N],ecx  
 mov eax,[N]  
 call iprintLF ; Вывод значения `N`  
 pop ecx  
 loop label ; `ecx=ecx-1` и если `ecx` не '0'  
 ; переход на `label`  
 call quit

1. Компилируем и запустим. Получим значения **ecx**, уменьшенные на 1.

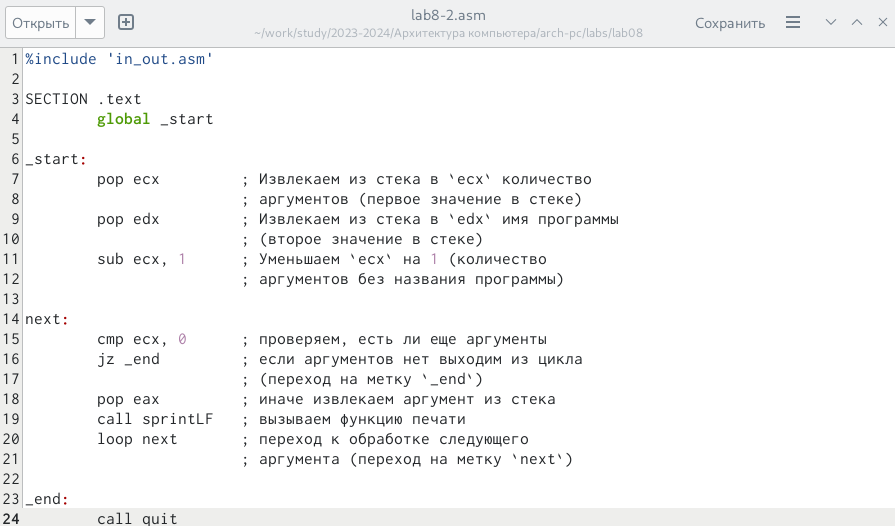


Исполнение lab8-1-2

# 4 Реализация цикла, зависящего от аргументов

При запуске программы есьт возможность указать какие-либо аргументы, которые программа может использовать. Попробуем написать код, который работате с введёнными с клавиатуры аргументами.

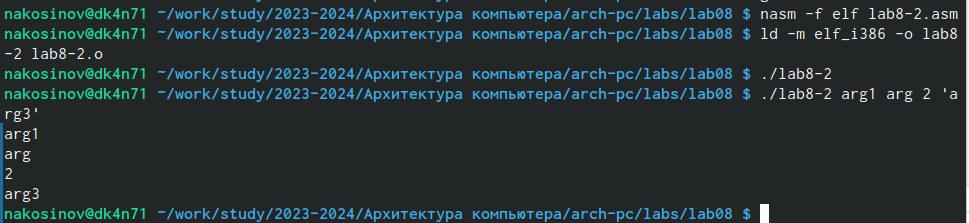
1. Пишем код программы: вывести указанные аргументы, каждый на новой строке. Количество аргументов при этом можно взять из стека аргументов, пропуская при этом название программы, также записанное в стек.



Код программы lab8-2

%include 'in\_out.asm'  
  
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)  
   
next:  
 cmp ecx, 0 ; проверяем, есть ли еще аргументы  
 jz \_end ; если аргументов нет выходим из цикла  
 ; (переход на метку `\_end`)  
 pop eax ; иначе извлекаем аргумент из стека  
 call sprintLF ; вызываем функцию печати  
 loop next ; переход к обработке следующего  
 ; аргумента (переход на метку `next`)  
   
\_end:  
 call quit

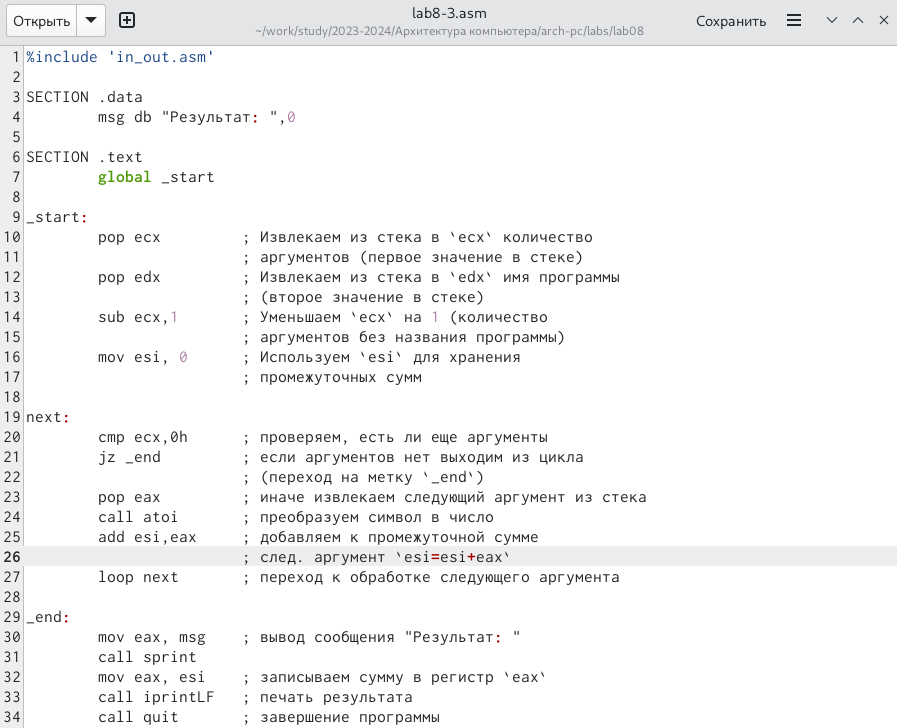
1. Компилируем и запускаем.



Исполнение lab8-2

# 5 Работа с аргументами

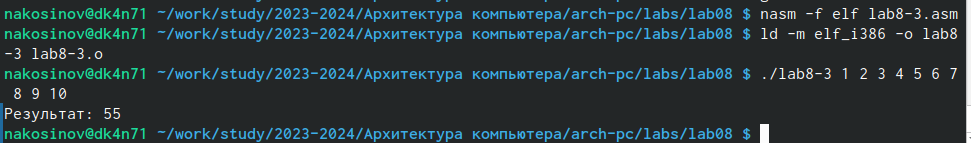
1. Попробуем сдлеать с аргументами что-то дополнительно, например, посчитать их сумму.



Код программы lab8-3

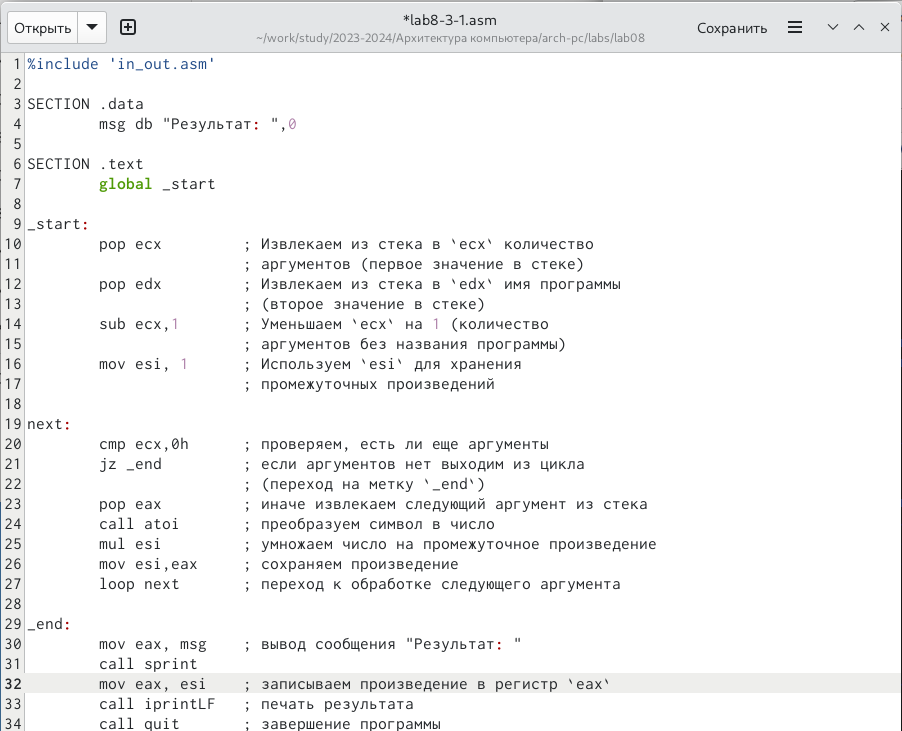
%include 'in\_out.asm'  
  
SECTION .data  
 msg db "Результат: ",0  
  
SECTION .text  
 global \_start  
  
\_start:  
 pop ecx ; Извлекаем из стека в `ecx` количество  
 ; аргументов (первое значение в стеке)  
 pop edx ; Извлекаем из стека в `edx` имя программы  
 ; (второе значение в стеке)  
 sub 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

1. Компилируем и запускаем. Программа работает корректно: .



Исполнение lab8-3

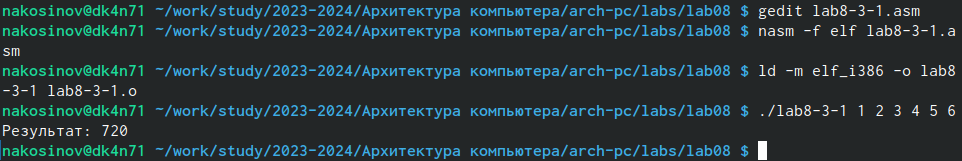
1. Изменим программу, чтобы она считала произведение чисел. Не забываем “обнулить” регистр **esi** единицей.



Код программы lab8-3-1

%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  
 ; промежуточных произведений  
   
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  
 mul esi ; умножаем число на промежуточное произведение  
 mov esi,eax ; сохраняем произведение  
 loop next ; переход к обработке следующего аргумента  
   
\_end:  
 mov eax, msg ; вывод сообщения "Результат: "  
 call sprint  
 mov eax, esi ; записываем произведение в регистр `eax`  
 call iprintLF ; печать результата  
 call quit ; завершение программы

1. Запустим и посчитаем, например .

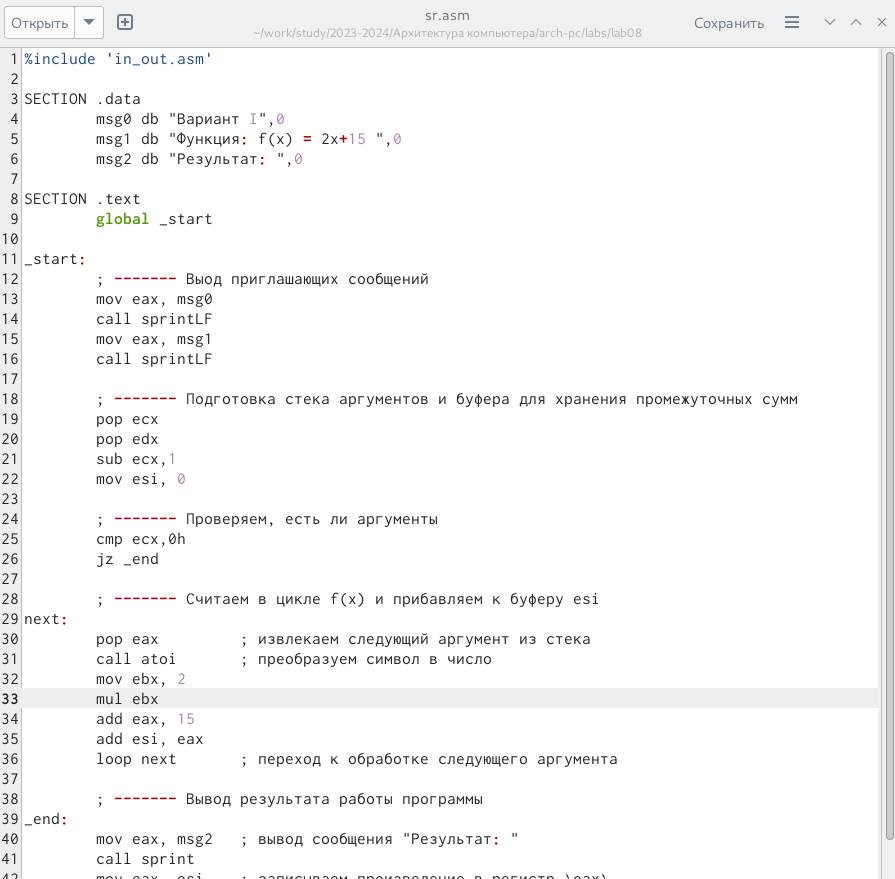


Исполнение lab8-3-1

# 6 Самостоятельная работа

В самостоятельной работе предлагается взять функцию и посчитать сумму значений этой функции от введённых аргументов.

1. Напишем код с комментариями.



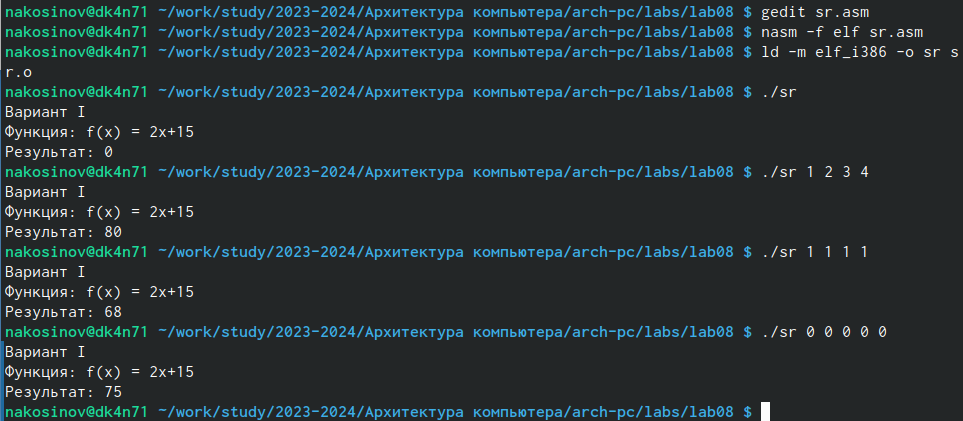
Код программы sr

Код программы sr

Код программы sr

%include 'in\_out.asm'  
  
SECTION .data  
 msg0 db "Вариант I",0  
 msg1 db "Функция: f(x) = 2x+15 ",0  
 msg2 db "Результат: ",0  
  
SECTION .text  
 global \_start  
  
\_start:  
 ; ------- Выод приглашающих сообщений  
 mov eax, msg0  
 call sprintLF  
 mov eax, msg1  
 call sprintLF  
   
 ; ------- Подготовка стека аргументов и буфера для хранения промежуточных сумм  
 pop ecx   
 pop edx   
 sub ecx,1   
 mov esi, 0   
   
 ; ------- Проверяем, есть ли аргументы  
 cmp ecx,0h  
 jz \_end  
  
 ; ------- Считаем в цикле f(x) и прибавляем к буферу esi  
next:  
 pop eax ; извлекаем следующий аргумент из стека  
 call atoi ; преобразуем символ в число  
 mov ebx, 2  
 mul ebx  
 add eax, 15  
 add esi, eax  
 loop next ; переход к обработке следующего аргумента  
   
 ; ------- Вывод результата работы программы  
\_end:  
 mov eax, msg2 ; вывод сообщения "Результат: "  
 call sprint  
 mov eax, esi ; записываем произведение в регистр `eax`  
 call iprintLF ; печать результата  
 call quit ; завершение программы

1. Скомпилируем и запустим несколько раз, указывая разные наборы аргументов. Убедимся, что во всех случаях программа работает корректно, в том числе, если аргументы не указаны.



Исполнение sr

# 7 Выводы

В ходе данной лабораторной работы мы познакомились со структурой стека, научились брать из автоматически созданного стека аргументы и их количество, реализовали работу с аргументами с использованием циклической структуры.