Отчет по лабораторной работа №9

Группа НПИбд-02-22

Стариков Данила Андреевич

Содержание

# 1 Цель работы

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

# 2 Основная часть

## 2.1 Выполнение лабораторной работы

### 2.1.1 Реализация циклов в NASM

Для выполнения лабораторной работы создали каталог ~/work/study/arch-pc/lab09. В нем создали файл lab9-1.asm и ввели текст из Листинга 1. Также положили в каталог файл in\_out.asm, использованный в лабораторной работе №6.

Листинг 1: Программа вывода значений регистра ecx

;-----------------------------------------------------------------  
; Программа вывода значений регистра 'ecx'  
;-----------------------------------------------------------------  
%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

Создали исполняемый файл и запустили его (Рис. 1).

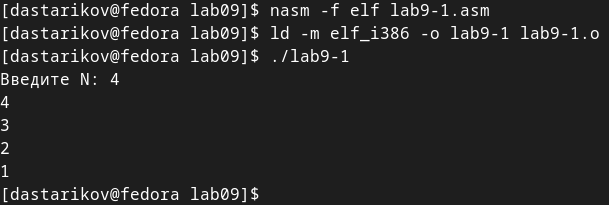


Рис. 1: Результат запуска файла lab9-1.

Так как регистр ecx используется для задания количества итераций цикла, его использование в теле цикла может привести к некорректной работе программы. Продемонстрировали это, добавив изменение регистра ecx в теле цикла:

label:  
 sub ecx,1 ; `ecx=ecx-1`  
 mov [N],ecx  
 mov eax,[N]  
 call iprintLF  
   
 loop label

В результате в ходе выполнения одной итерации цикла регистр уменьшается на 2, и общее количество итерации становится меньше, при этом в зависимости от ввода N, проверка ecx = 0 может не наступить, что приведет к бесконечному выполнению программы (Рис. 2 и 3).

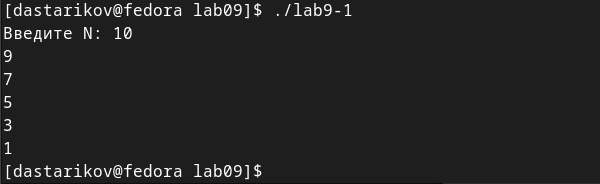


Рис. 2: Программа завершила работу, но число циклов меньше N.

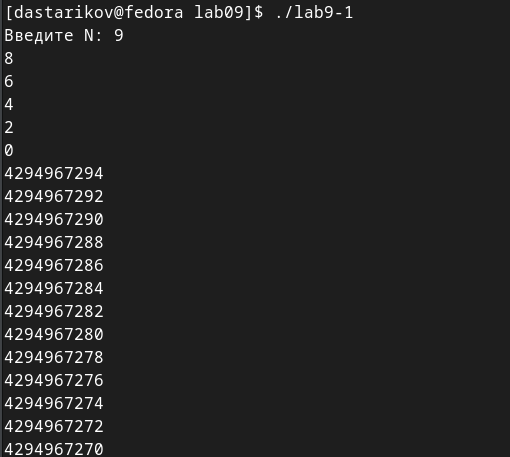


Рис. 3: Бесконечный ход программы.

Для использования значения регистра в цикле и сохранения работы в программе воспользовались стеком. Изменили текст программы, добавив команды push и pop:

label:  
 push ecx ; добавление значения ecx в стек  
 sub ecx,1  
 mov [N],ecx  
 mov eax,[N]  
 call iprintLF  
 pop ecx ; извлечение значения ecx из стека  
  
 loop label

Создали исполняемый файл и запустили его (Рис. 4). Количество итерации цикла совпадает со значением N.

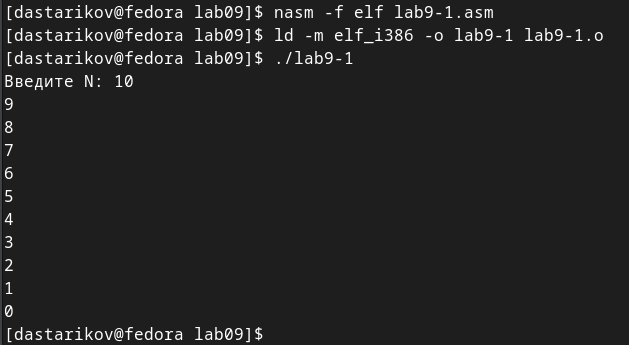


Рис. 4: Использование стека.

### 2.1.2 Обработка аргументов командной строки

В соотвествии с Листингом 2 написали программу lab9-2.asm, выводящую аргументы командной строки. Создали исполняемый файл и запустили его с аргументами (Рис. 5). Программа обработала 4 аргумента, разделенных проблемами.

Листинг 2: Программа выводящая на экран аргументы командной строки

%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

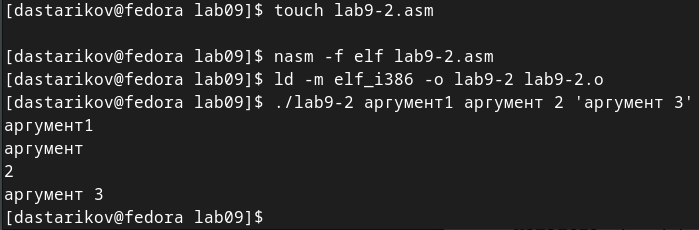


Рис. 5: Сообщения выводятся в обратном порядке.

В качестве другого примера работы с аргументами командной строки написали программу lab9-3.asm, выводящую сумму чисел, переданных как аргументы (Листинг 3). Создали исполняемый файл и запустили его (Рис. 6).

Листинг 3: Программа вычисления суммы аргументов командной строки

%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

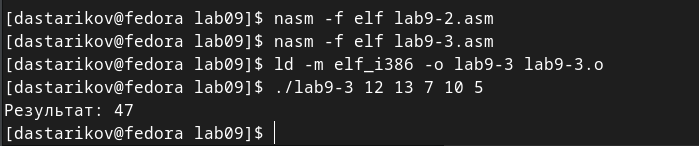


Рис. 6: Результат работы программы lab9-3.

Заменили текст программы, чтобы выводилось произведение чисел:

pop eax ; иначе извлекаем следующий аргумент из стека  
 call atoi ; преобразуем символ в число  
 mov ebx, eax ; сохраняем значений аргумент в ebx для умножения  
 mov eax, esi ; сохраняем esi в eax, чтобы домножить на аргумент  
 mul ebx ; умножаем eax\*ebx == промежуточное произвдение на аргумент  
 mov esi, eax ; сохраняем значение получившейся суммы обратно в esi

Создали исполняемый файл и запустили его (Рис. 7).

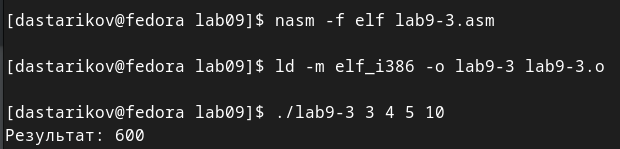


Рис. 7: Вывод произведения аргументов

## 2.2 Выполнение заданий для самостоятельной работы.

Для выполнения заданий выбран вариант 12, полученный при выполнении лабораторной работы №7.

Написали программу lab9-var12.asm (Листинг 4), находящую сумму значений функции для , причем числа вводят как аргументы командной строки (Рис. 8).

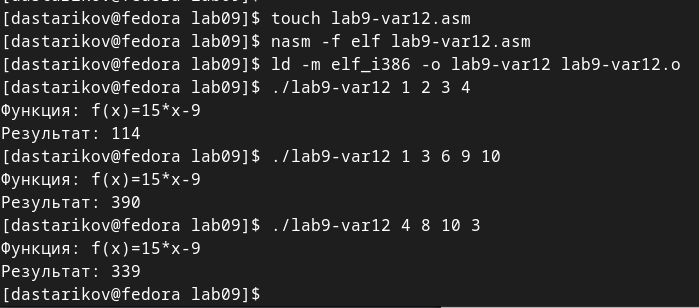


Рис. 8: Результат работы программы lab9-var12.

Листинг 4: Программа находит сумму значений функции

%include 'in\_out.asm'  
  
SECTION .data  
 msg1 db "Функция: f(x)=15\*x-9 ",0  
 msg2 db "Результат: ",0  
SECTION .text  
 global \_start  
  
\_start:  
  
 mov eax, msg1  
 call sprintLF  
   
 mov ebx, 15 ; регистр ebx используем для умножения  
   
 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  
 mul ebx ; eax=eax\*ebx  
 sub eax,9 ; eax=eax-9  
 add esi,eax ; добавляем к промежуточной сумме  
 ; след. аргумент `esi=esi+eax`  
 loop next ; переход к обработке следующего аргумента  
\_end:  
 mov eax, msg2 ; вывод сообщения "Результат: "  
 call sprint  
 mov eax, esi ; записываем сумму в регистр `eax`  
 call iprintLF ; печать результата  
 call quit ; завершение программы

Созданные файлы \*.asm скопировали в каталог ~/work/study/2022-2023/“Архитектура компьютера”/archpc/labs/lab09/ и загрузили на Github.

# 3 Выводы

В рамках лабораторной работы получили практические навыки построения циклов и обработки аргументов командной строки при написании программ.