

# **Отчет по лабораторной работа №9**

**Группа НПИбд-02-22**

Стариков Данила Андреевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Основная часть</b>	<b>4</b>
2.1	Выполнение лабораторной работы . . . . .	4
2.1.1	Реализация циклов в NASM . . . . .	4
2.1.2	Обработка аргументов командной строки . . . . .	7
2.2	Выполнение заданий для самостоятельной работы. . . . .	9
<b>3</b>	<b>Выводы</b>	<b>14</b>

# 1 Цель работы

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

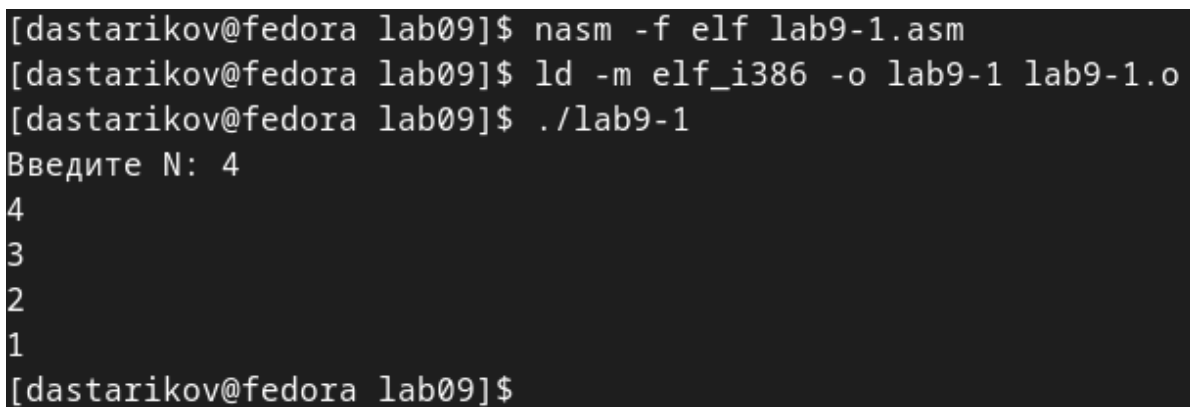
## 2 Основная часть

### 2.1 Выполнение лабораторной работы

#### 2.1.1 Реализация циклов в NASM

Для выполнения лабораторной работы создали каталог `~/work/study/arch-rc/lab09`. В нем создали файл `lab9-1.asm` и ввели текст из Листинга 2.1. Также положили в каталог файл `in_out.asm`, использованный в лабораторной работе №6.

Создали исполняемый файл и запустили его (Рис. 2.1).



```
[dastarikov@fedora lab09]$ nasm -f elf lab9-1.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[dastarikov@fedora lab09]$ ./lab9-1
Введите N: 4
4
3
2
1
[dastarikov@fedora lab09]$
```

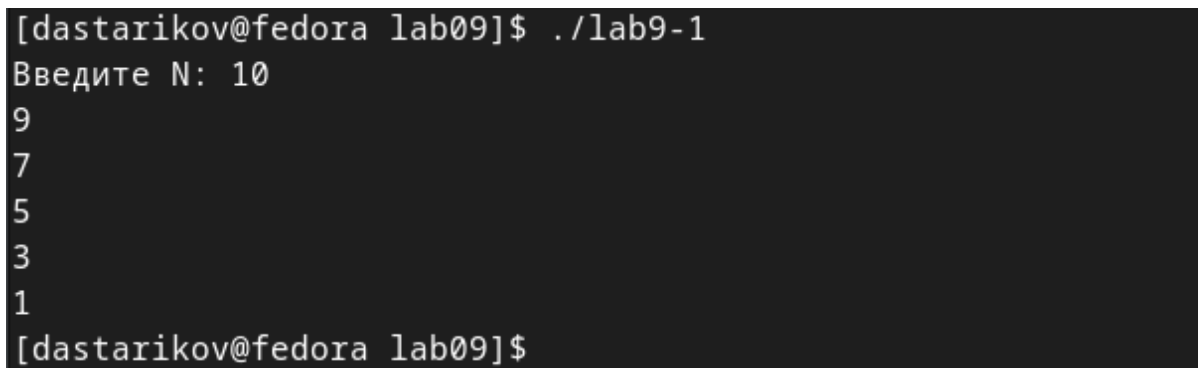
Рис. 2.1: Результат запуска файла `lab9-1`.

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. 2.2 и 2.3).



```
[dastarikov@fedora lab09]$ ./lab9-1  
Введите N: 10  
9  
7  
5  
3  
1  
[dastarikov@fedora lab09]$
```

Рис. 2.2: Программа завершила работу, но число циклов меньше N.

```
[dastarikov@fedora lab09]$ ./lab9-1
Введите N: 9
8
6
4
2
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
4294967270
```

Рис. 2.3: Бесконечный ход программы.

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

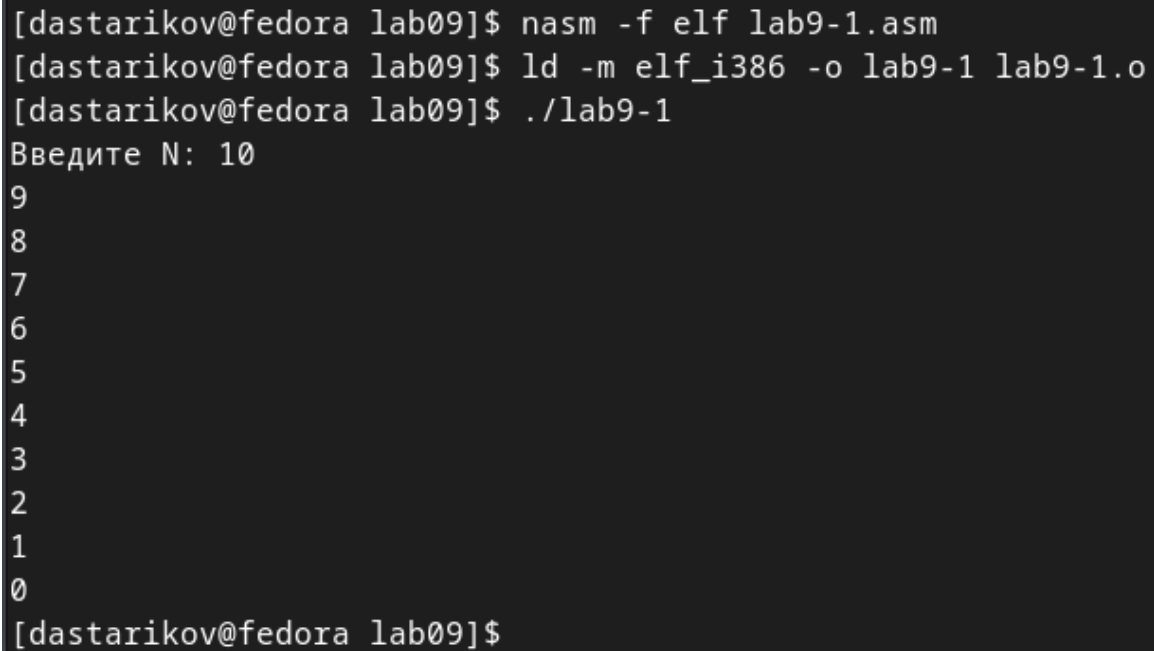
`label:`

```
    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. 2.4). Количество итерации цикла совпадает со значением N.



```
[dastarikov@fedora lab09]$ nasm -f elf lab9-1.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[dastarikov@fedora lab09]$ ./lab9-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
[dastarikov@fedora lab09]$
```

Рис. 2.4: Использование стека.

### 2.1.2 Обработка аргументов командной строки

В соответствии с Листингом 2.2 написали программу lab9-2.asm, выводящую аргументы командной строки. Создали исполняемый файл и запустили его с аргументами (Рис. 2.5). Программа обработала 4 аргумента, разделенных пробелами.

```
[dastarikov@fedora lab09]$ touch lab9-2.asm

[dastarikov@fedora lab09]$ nasm -f elf lab9-2.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[dastarikov@fedora lab09]$ ./lab9-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[dastarikov@fedora lab09]$
```

Рис. 2.5: 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 2.3). Создали исполняемый файл и запустили его (Рис. 2.6).

```
[dastarikov@fedora lab09]$ nasm -f elf lab9-2.asm
[dastarikov@fedora lab09]$ nasm -f elf lab9-3.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[dastarikov@fedora lab09]$ ./lab9-3 12 13 7 10 5
Результат: 47
[dastarikov@fedora lab09]$ |
```

Рис. 2.6: Результат работы программы lab9-3.

Заменяли 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 произведение на аргумент
mov esi, eax ; сохраняем значение получившейся суммы обратно в esi
```

Создали исполняемый файл и запустили его (Рис. 2.7).



```
[dastarikov@fedora lab09]$ nasm -f elf lab9-3.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[dastarikov@fedora lab09]$ ./lab9-3 3 4 5 10
Результат: 600
```

Рис. 2.7: 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 2.4), находящую сумму значений функции  $f(x) = 15 * x - 9$  для  $x = x_1, x_2, \dots, x_n$ , причем числа вводят как аргументы командной строки (Рис. 2.8).

```
[dastarikov@fedora lab09]$ touch lab9-var12.asm
[dastarikov@fedora lab09]$ nasm -f elf lab9-var12.asm
[dastarikov@fedora lab09]$ ld -m elf_i386 -o lab9-var12 lab9-var12.o
[dastarikov@fedora lab09]$ ./lab9-var12 1 2 3 4
Функция: f(x)=15*x-9
Результат: 114
[dastarikov@fedora lab09]$ ./lab9-var12 1 3 6 9 10
Функция: f(x)=15*x-9
Результат: 390
[dastarikov@fedora lab09]$ ./lab9-var12 4 8 10 3
Функция: f(x)=15*x-9
Результат: 339
[dastarikov@fedora lab09]$
```

Рис. 2.8: Результат работы программы lab9-var12.

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

---

## Листинг 2.1 Программа вывода значений регистра есх

---

```
;-----  
; Программа вывода значений регистра 'есх'  
;-----  
%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, `есх=N`  
label:  
    mov [N],ecx  
    mov eax,[N]  
    call iprintLF ; Вывод значения `N`  
  
    loop label ; `есх=есх-1` и если `есх` не '0'  
                ; переход на `label`  
    call quit
```

---

---

**Листинг 2.2** Программа выводющая 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 sprintf ; вызываем функцию печати
    loop next ; переход к обработке следующего
              ; аргумента (переход на метку `next`)

_end:
    call quit
```

---

---

**Листинг 2.3** 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
```

---

---

## Листинг 2.4 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
```

---

## **3 Выводы**

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