

# **Отчёта по лабораторной работе 9**

**Программирование цикла. Обработка аргументов командной строки.**

Герра Гарсия Паола Валентина – НКАбд-05-22

# Содержание

|          |                                       |           |
|----------|---------------------------------------|-----------|
| <b>1</b> | <b>Цель работы</b>                    | <b>5</b>  |
| <b>2</b> | <b>Задание</b>                        | <b>6</b>  |
| <b>3</b> | <b>Теоретическое введение</b>         | <b>7</b>  |
| <b>4</b> | <b>Выполнение лабораторной работы</b> | <b>8</b>  |
| <b>5</b> | <b>Выводы</b>                         | <b>19</b> |
|          | <b>Список литературы</b>              | <b>20</b> |

## Список иллюстраций

|      |                                       |    |
|------|---------------------------------------|----|
| 4.1  | Файл lab9-1.asm . . . . .             | 8  |
| 4.2  | Работа программы lab9-1.asm . . . . . | 9  |
| 4.3  | Файл lab9-1.asm.....                  | 10 |
| 4.4  | Работа программы lab9-1.asm.....      | 11 |
| 4.5  | Файл lab9-1.asm.....                  | 12 |
| 4.6  | Работа программы lab9-1.asm.....      | 12 |
| 4.7  | Файл lab9-2.asm.....                  | 13 |
| 4.8  | Работа программы lab9-2.asm.....      | 14 |
| 4.9  | Файл lab9-3.asm.....                  | 15 |
| 4.10 | Работа программы lab9-3.asm.....      | 15 |
| 4.11 | Файл lab9-3.asm.....                  | 16 |
| 4.12 | Работа программы lab9-3.asm.....      | 17 |
| 4.13 | Файл lab9-4.asm.....                  | 18 |
| 4.14 | Работа программы lab9-4.asm.....      | 18 |

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Изучите примеры программ
2. Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $x$ .
3. Загрузите файлы на GitHub.

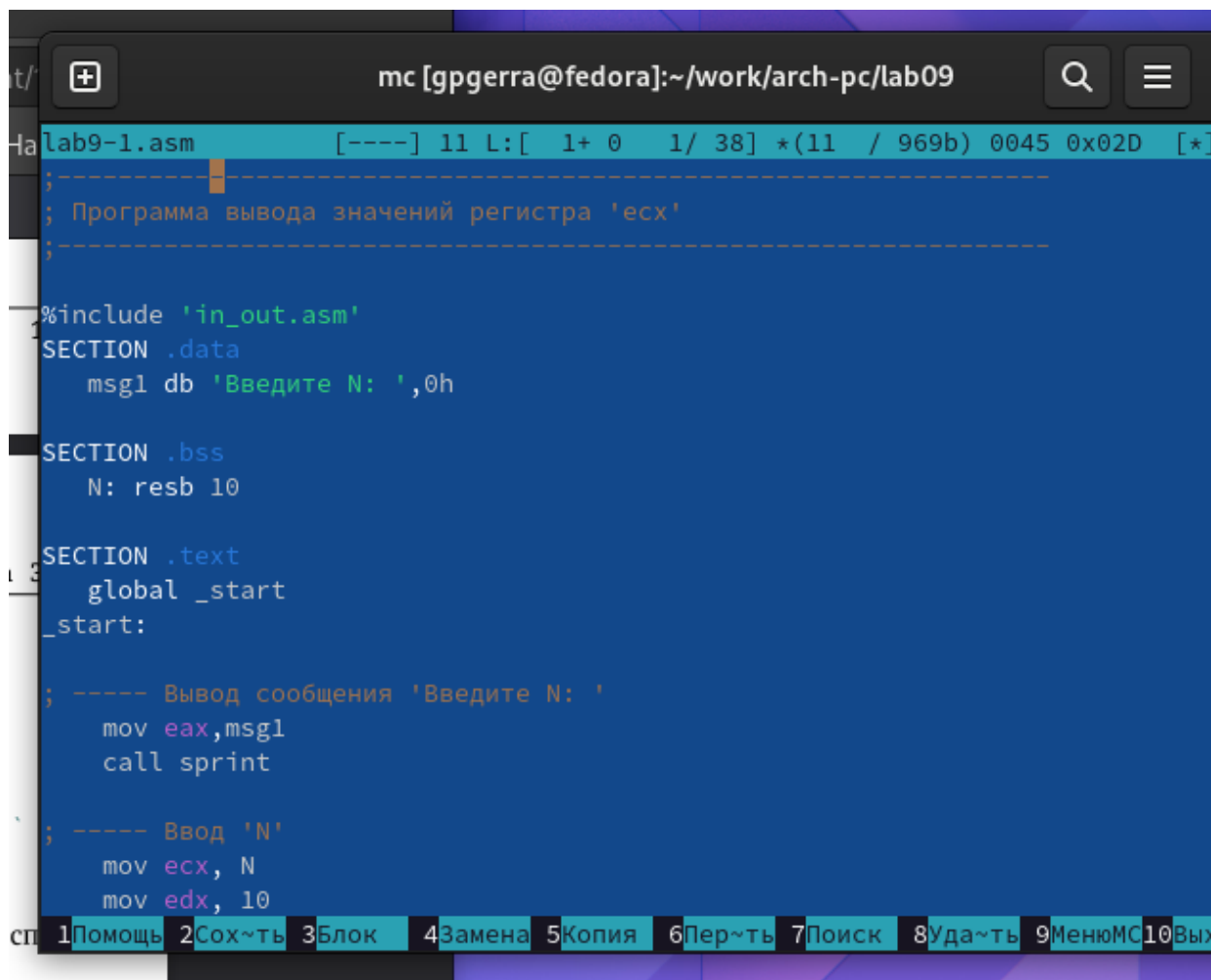
### 3 Теоретическое введение

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.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре esx. Наиболее простой является инструкция loop. Инструкция loop выполняется в два этапа. Сначала из регистра esx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

## 4 Выполнение лабораторной работы

1. Создайте каталог для программ лабораторной работы № 9, перейдите в него и создайте файл lab9-1.asm
2. Введите в файл lab9-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. (рис. 4.1, 4.2)



```
lab9-1.asm [----] 11 L: [ 1+ 0 1/ 38] *(11 / 969b) 0045 0x02D [*]
;
; 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
```

Рис. 4.1: Файл lab9-1.asm



```
gpgerra@fedora:~/work/arch-pc/lab09
[гpgerra@fedora lab09]$ nasm -f elf lab9-1.asm
[гpgerra@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[гpgerra@fedora lab09]$ ./lab9-1
Введите N: 9
9
8
7
6
5
4
3
2
1
[гpgerra@fedora lab09]$
```

```
mc [gpgerra@fedora]:~/work/arch-pc/lab09
lab9-1.asm  [-M--]  0 L: [ 15+23  38/ 38] *(812 / 826b) 0032 0x020 [*][X]

; ----- 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 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF

loop label

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюМС10Выход
```

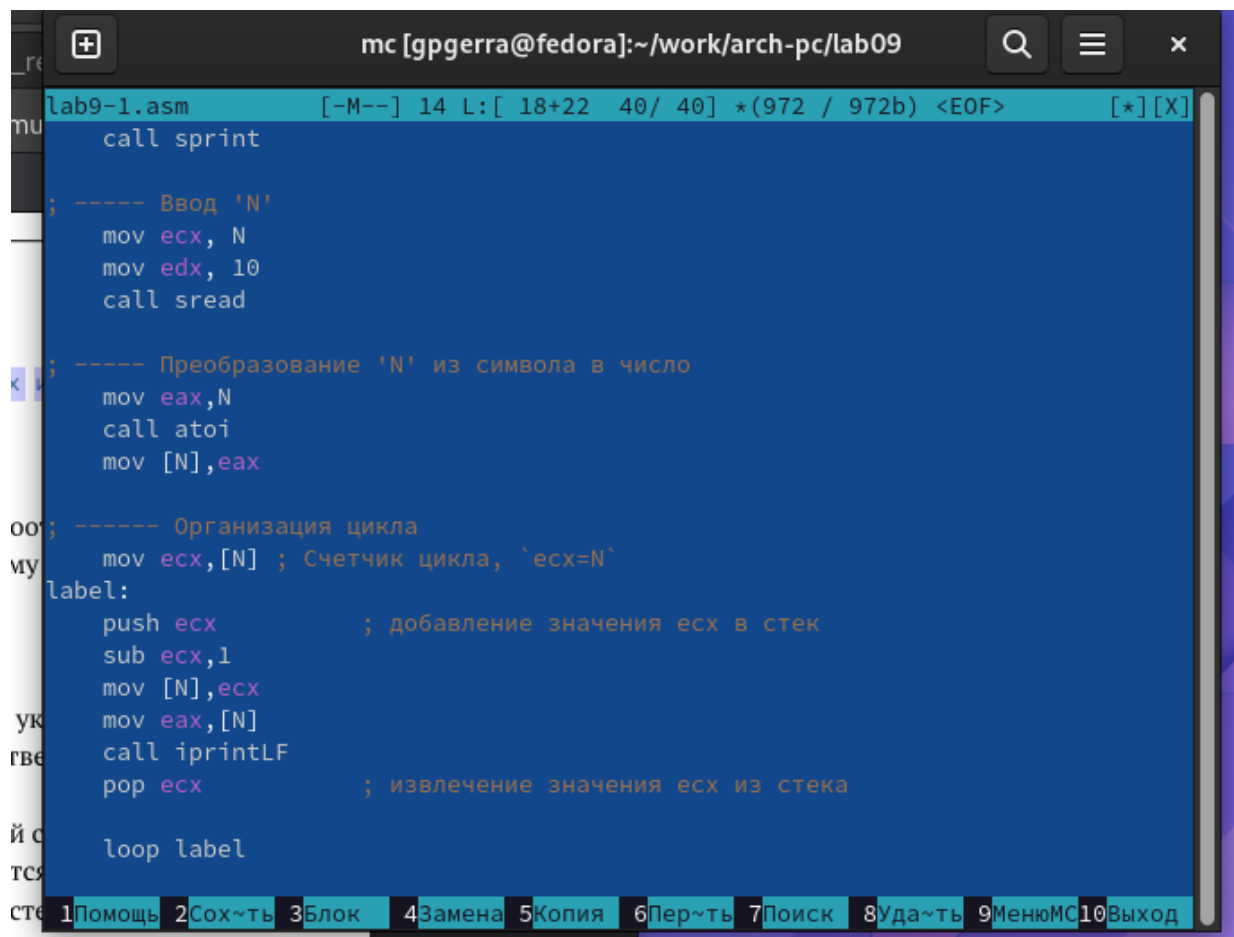
Рис. 4.3: Файл lab9-1.asm

```
[gpgerra@fedora lab09]$ ./lab9-1
Введите N: 7
7
6
5
4
3
2
1
[gpgerra@fedora lab09]$
```

Рис. 4.4: Работа программы lab9-1.asm

4. Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`. Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению `N` введенному с клавиатуры? (рис. 4.5, 4.6)

Программа выводит числа от `N-1` до `0`, число проходов цикла соответствует `N`.



```
lab9-1.asm [-M--] 14 L: [ 18+22 40/ 40] *(972 / 972b) <EOF> [*] [X]
call sprint

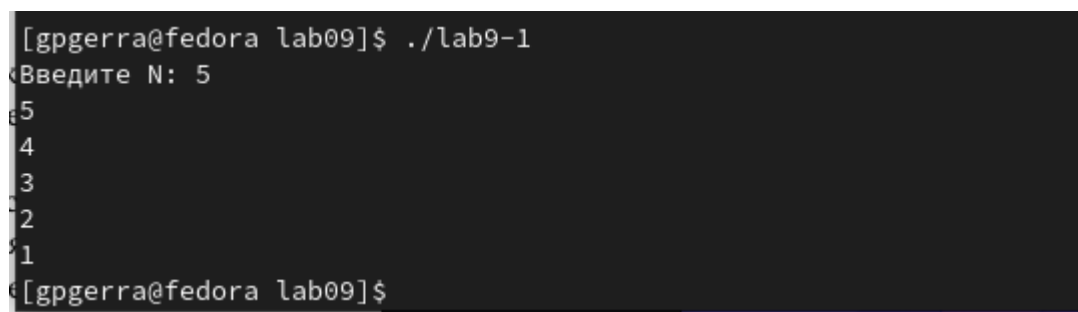
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread

; ----- 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 ; 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
```

Рис. 4.5: Файл lab9-1.asm

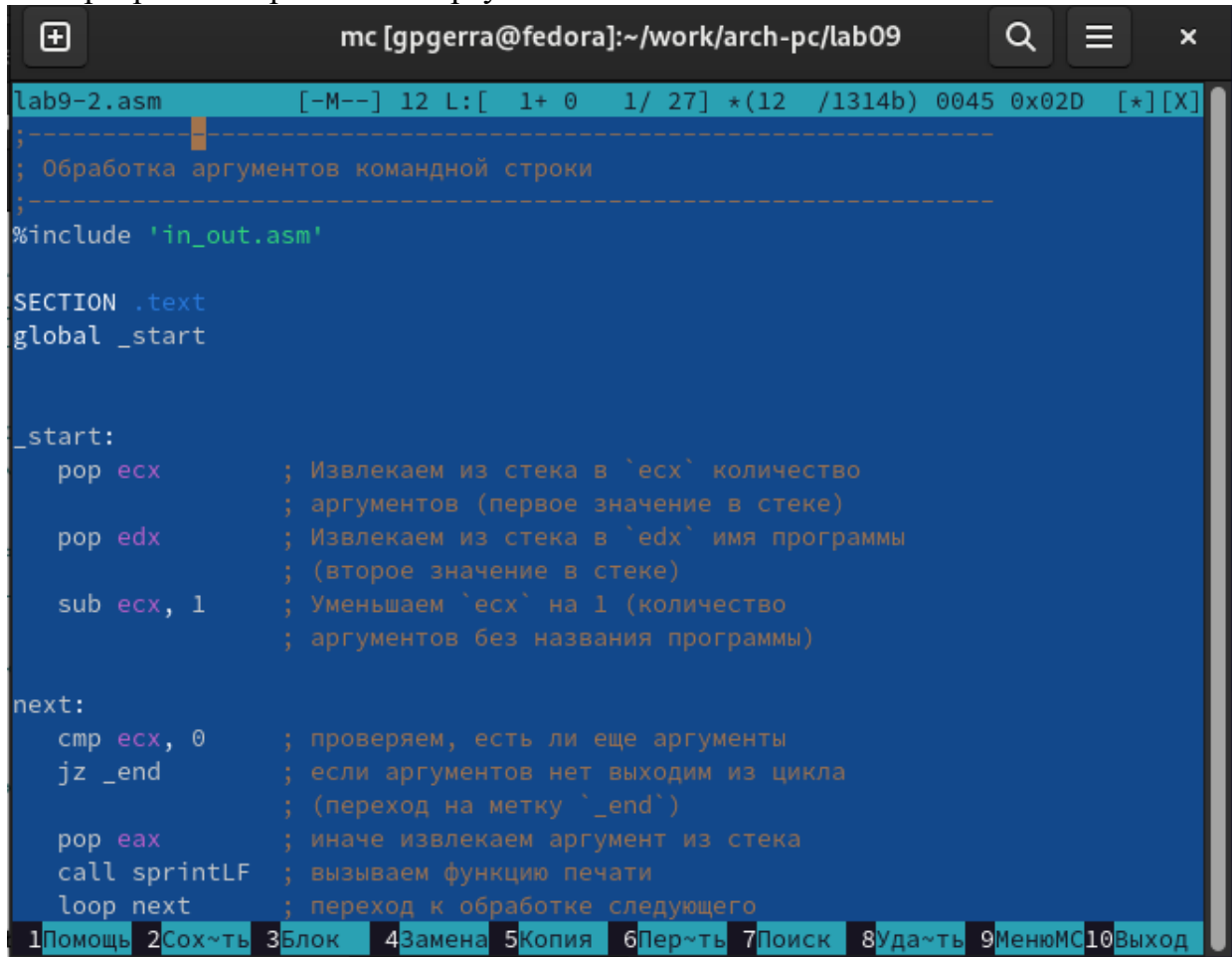


```
[gpgerra@fedora lab09]$ ./lab9-1
Введите N: 5
5
4
3
2
1
[gpgerra@fedora lab09]$
```

Рис. 4.6: Работа программы lab9-1.asm

5. Создайте файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и введите в него текст программы из листинга 9.2. Создайте исполняемый файл и запустите его, указав аргументы. (рис. 4.7, 4.8) Сколько аргументов было обработано программой?

Программа обработала 5 аргументов.



```
lab9-2.asm [-M--] 12 L:[ 1+ 0 1/ 27] *(12 /1314b) 0045 0x02D [*] [X]
; -----
; Обработка аргументов 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

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюMC10Выход
```

Рис. 4.7: Файл lab9-2.asm

```
[gpgerra@fedora lab09]$ nasm -f elf lab9-2.asm
[gpgerra@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[gpgerra@fedora lab09]$ ./lab9-2
[gpgerra@fedora lab09]$ ./lab9-2 аргумент 1 аргумент 2 аргумент 3
аргумент
1
аргумент
2
аргумент
3
[gpgerra@fedora lab09]$
```

Рис. 4.8: Работа программы lab9-2.asm

6. Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. 4.9, 4.10)

```

lab9-3.asm      [-M--] 26 L:[ 1+ 0  1/ 32] *(26 /1693b) 0010 0x00A  [*] [X]
#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
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюМС10Выход

```

Рис. 4.9: Файл lab9-3.asm

```

[gpgerra@fedora lab09]$ touch lab9-3.asm
[gpgerra@fedora lab09]$ mc

[gpgerra@fedora lab09]$ nasm -f elf lab9-3.asm
[gpgerra@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[gpgerra@fedora lab09]$ ./lab9-3
Результат: 0
[gpgerra@fedora lab09]$ ./lab9-3 12 12 14 7 5 6
Результат: 56
[gpgerra@fedora lab09]$

```

Рис. 4.10: Работа программы lab9-3.asm

7. Измените текст программы из листинга 9.3 для вычисления произведения аргументов командной строки. (рис. 4.11, 4.12)

Рис. 4.11: Файл lab9-3.asm



Рис. 4.12: Работа программы lab9-3.asm

8. Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x$  передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $x$ . (рис. 4.13, 4.14)

для варианта 10  $f(x) = 5(2+x)$

Рис. 4.13: Файл lab9-4.asm

Рис. 4.14: Работа программы lab9-4.asm

## 5 Выводы

В заключение, это позволило нам освоить работу со стеком, циклом и аргументами в ассемблере Nasm.

# Список литературы

1. Расширенный ассемблер: NASM