

# **Лабораторная работа №8**

**Архитектура компьютера**

Казначеева Кристина Никитична

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Задание</b>	<b>4</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
<b>4</b>	<b>Вывод</b>	<b>14</b>

# **1 Цель работы**

Лабораторная работа направлена на практическое освоение программирования с использованием циклов и обработки аргументов командной строки.

## 2 Задание

В данной лабораторной работе мы изучим реализацию циклов в NASM, программы вывода значений регистра `eax` и вычисления суммы аргументов командной строки, а также программу, выводящую на экран аргументы командной строки. Научимся обработке аргументов командной строки.

### 3 Выполнение лабораторной работы

Создадим каталог lab08 (рис. 3.1).

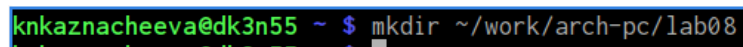
A terminal window with a dark background. The prompt is 'knkaznacheeva@dk3n55 ~ \$'. The command 'mkdir ~/work/arch-pc/lab08' is entered and executed. The prompt changes to 'knkaznacheeva@dk3n55 ~ \$'.

Рис. 3.1: Создание каталога

Перейдём в этот каталог и создадим файл lab8-1.asm (рис. 3.2).

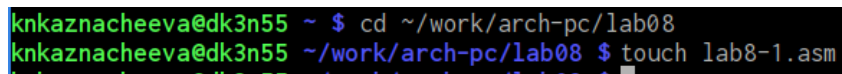
A terminal window with a dark background. The prompt is 'knkaznacheeva@dk3n55 ~ \$'. The command 'cd ~/work/arch-pc/lab08' is entered and executed. The prompt changes to 'knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 \$'. The command 'touch lab8-1.asm' is entered and executed. The prompt changes to 'knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 \$'.

Рис. 3.2: Создание файла

Введём в файл lab8-1.asm текст программы вывода значений регистра еsx (рис. 3.3).

```

lab8-1.asm      [----] 24 L:[ 1+ 5 6/ 31]
;
; 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
1По~щъ 2Со~ан 3Блок 4За~на 5Копия 6Пе~ть 7Пои

```

Рис. 3.3: Ввод текста программы

Создадим исполняемый файл и запустим его (рис. 3.4).

```

knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1

```

Рис. 3.4: Проверка работы исходного файла

Затем изменим текст программы, добавив изменение значение регистра ecx в цикле (рис. 3.5).

```

lab8-1.asm [----] 11 L:[ 14+ 8 22/ 31]
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:
label:
sub ecx, 1 ; 'ecx=ecx-1'
mov [N], ecx
mov eax, [N]
call iprintLF
loop label
1Посиль 2Сосан 3Блок 4Засна 5Копия 6Песть 7Пои

```

Рис. 3.5: Изменение текста программы

Запустим исполняемый файл и проверим его. Регистр `ecx` принимает в цикле значения, уменьшающиеся на 1 на каждой итерации цикла, начиная с начального значения, которое задано до начала цикла. Число проходов цикла точно соответствует значению `N`, которое было загружено в `ecx` до начала цикла. Цикл выполняется `N` раз (рис. 3.6).

```

knkaznacheeva@dk3n55 ~ $ cd work/arch-pc/lab08/
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab
8-1.o
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
7
5
3
1
Ошибка сегментирования (образ памяти сброшен на диск)

```

Рис. 3.6: Проверка работы исходного файла

Внесём изменения в текст программы, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 3.7).

```

lab8-1.asm      [-M--]  5  L:[ 15+ 4  19.
; ----- 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
1Помощь 2Создать 3Блок 4Зачна 5Копия 6Перейти

```

Рис. 3.7: Изменение текста программы

Создадим исполняемый файл и проверим его работу. При добавлении команды `push` и `pop` число проходов цикла в измененном коде не будет точно соответствовать значению `N`, введенному с клавиатуры. (рис. 3.8):

```

knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
Ошибка сегментирования (образ памяти сброшен на диск)

```

Рис. 3.8: Проверка работы исходного файла

Создадим файл `lab8-2.asm` в каталоге `~/work/arch-pc/lab08` (рис. 3.9).



```
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-2.asm
```

Рис. 3.9: Создание файла

Введём в него текст программы, выводящей на экран аргументы командной строки (рис. 3.10).

```
lab8-2.asm      [-M--] 43 L:[ 1+17 18/ 23] *(
;
; -----
;  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
1Пошь 2Соан 3Блок 4За~на 5Копия 6Пе~ть 7Поиск
```

Рис. 3.10: Ввод текста программы

Создадим исполняемый файл и запустим его, указав аргументы: аргумент1 аргумент 2 'аргумент 3'. В результате было обработано 3 аргумента (рис. 3.11).

```
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент
2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 3.11: Проверка работы исходного файла

Создадим файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 (рис. 3.12).

```
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-3.asm
```

Рис. 3.12: оздание файла

Ведём в него текст программы вычисления суммы аргументов командной строки (рис. 3.13).

```
lab8-3.asm      [-M--] 27 L:[ 1+17 18/ 29] *
#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')
```

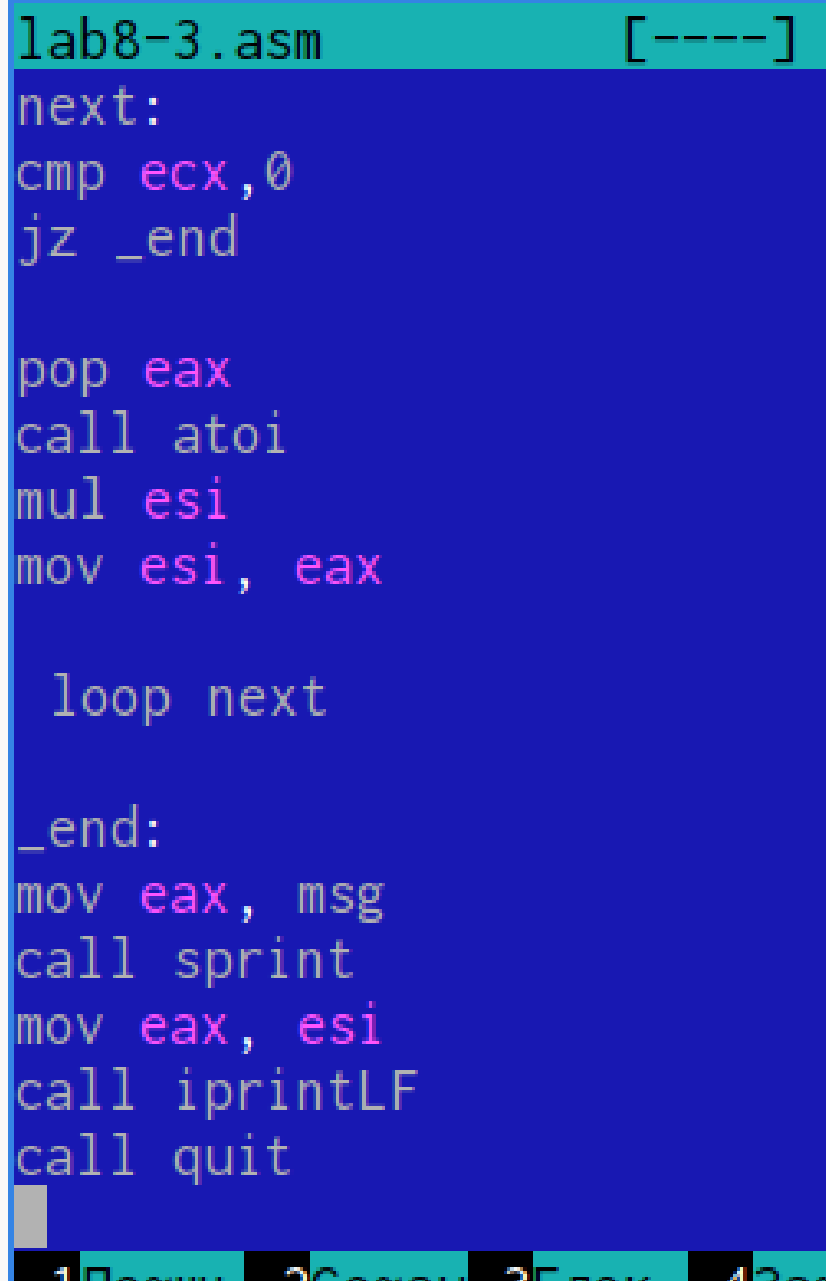
Рис. 3.13: Ввод текста программы

Создадим исполняемый файл и запустим его, указав аргументы (рис. 3.14).

```
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab
8-3.o
knkaznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 ./main 12 13 7 10 5
Результат: 47
```

Рис. 3.14: Проверка работы исходного файла

Изменим текст программы для вычисления произведения аргументов командной строки (рис. 3.15).



```
lab8-3.asm [----]
next:
cmp ecx,0
jz _end

pop eax
call atoi
mul esi
mov esi, eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 3.15: Изменение текста программы

Создадим исполняемый файл и проверим его работу (рис. 3.16).

```

knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 5 6 7
результат: 210
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $

```

Рис. 3.16: Проверка работы исходного файла

Создадим файл lab8-4.asm (рис. 3.17).

```

knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-4.o
knkznacheeva@dk3n55 ~/work/arch-pc/lab08 $

```

Рис. 3.17: Создание файла

Напишем программу, которая находит сумму значений функции  $f(x)=7(x+1)$  (вариант 14) для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_3)$ , где значения  $x$  передаются как аргументы (рис. 3.18).

```

lab8-4.asm      [-----] 60 L:[ 1+28 29/ 45] *(1017/1197b) 0010 0x00
%include 'in_out.asm'

SECTION .data
    msg_func db "Функция: f(x)=7(x+1)", 0xA, 0 ; Сообщение с функцией
    msg_result db "Результат: ", 0
    newline db 0xA, 0

SECTION .text
global _start
_start:
    pop ecx ; Количество аргументов
    pop edx ; Имя программы
    sub ecx, 1 ; Количество аргументов без имени программы
    mov esi, 0 ; Инициализируем сумму нулём
next:
    cmp ecx, 0 ; Есть ли ещё аргументы?
    jz end_loop

    pop eax ; Следующий аргумент
    call atoi ; Преобразуем в число

    ; Вычисляем f(x) = 7(x+1)
    mov ebx, 7
    mul ebx ; eax = eax * 7
    sub eax, 7 ; eax = eax + 7

    add esi, eax ; Добавляем значение функции к сумме

    loop next ; Переход к обработке следующего аргумента
end_loop

```

Рис. 3.18: Ввод текста программы

Создадим исполняемый файл и проверим его работу для значений  $x$ : 1, 2, 3, 4 (рис. 3.19).

```
knkznacheeva@dk3n35 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
knkznacheeva@dk3n35 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
knkznacheeva@dk3n35 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Функция: f(x)=7(x+1)
Результат: 98
```

Рис. 3.19: Проверка работы исходного файла

1. Команда `loop` реализует цикл, который повторяется `ecx` раз. Она выполняет три действия: • Уменьшает значение регистра `ecx` на 1. Регистр `ecx` обычно используется как счетчик цикла. • Проверяет значение регистра `ecx`. Если `ecx` не равно нулю, то происходит переход к указанной метке. • Переход к метке. Если `ecx` не равно нулю, выполнение переходит к метке, указанной после команды `loop`. Если `ecx` равно нулю, выполнение продолжается с инструкции, следующей за командой `loop`.
2. Для организации цикла без использования специальных команд управления циклами, таких как `loop`, используются условные переходы, например, `jnz` (jump if not zero) или `jz` (jump if zero).
3. Стек — это структура данных типа LIFO (Last-In, First-Out — последний вошел, первый вышел). В программировании стек используется для временного хранения данных, например, адресов возврата при вызовах функций, локальных переменных и промежуточных результатов вычислений.
4. Данные извлекаются из стека в порядке, обратном порядку их занесения (LIFO). Последний элемент, добавленный в стек (то есть, находящийся на вершине стека), является первым элементом, который будет извлечен. Операции добавления и извлечения данных в стеке часто называются `push` (добавление) и `pop` (извлечение).

## **4 Вывод**

В ходе лабораторной работы мы изучили программирование на ассемблере NASM, освоив работу с циклами и обработку аргументов командной строки. Мы реализовали примеры программ: вывод значения регистра ECX, вычисление суммы числовых аргументов командной строки и вывод самих аргументов на экран.