

# Отчёт по лабораторной работе №8

## Дисциплина: архитектура компьютеров и операционные системы

Лисенков Егор Романович

### Содержание

1	Цель работы .....	1
2	Задание.....	1
3	Теоретическое введение .....	1
4	Выполнение лабораторной работы .....	2
4.1	Реализация циклов в NASM .....	2
4.2	Обработка аргументов командной строки.....	5
4.3	Задание для самостоятельной работы .....	7
5	Выводы.....	9
6	Список литературы.....	9

## 1 Цель работы

Благодаря этому, появляются навыки написания программ с использованием циклов и обработкой аргументов командной строки.

## 2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание для самостоятельной работы.

## 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. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается.

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

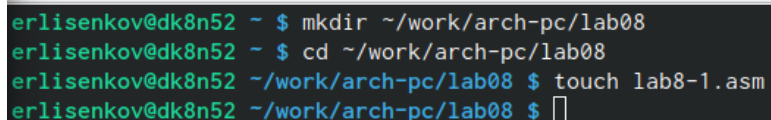
Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл.

## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM

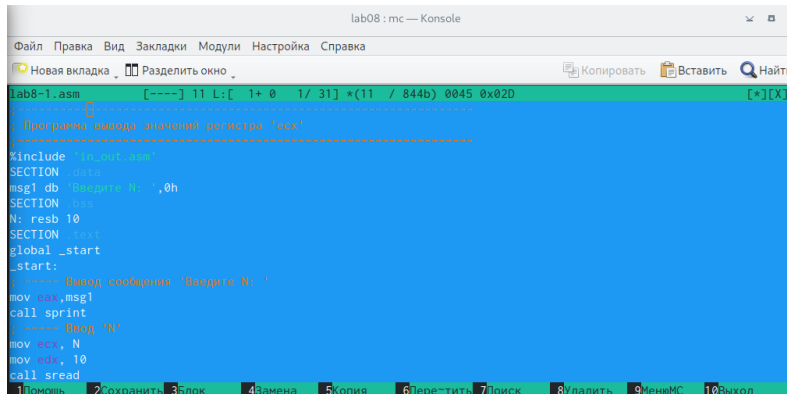
Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. ??).



```
erlisenkov@dk8n52 ~ $ mkdir ~/work/arch-pc/lab08
erlisenkov@dk8n52 ~ $ cd ~/work/arch-pc/lab08
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

*Создание файлов для лабораторной работы*

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. ??).



```
lab8-1.asm [----] 11 L: [ 1+ 0 1/ 31] *(11 / 844b) 0045 0x02D [*][X]
; Программа вывода значений регистра 'ecx'
; =====
%include "lab08.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 ebx, 10
call sread
```

### Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу. (рис. ??).

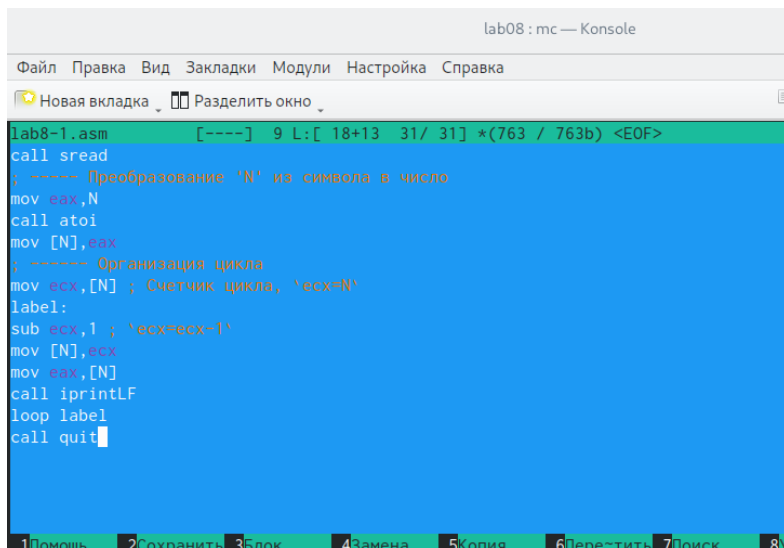


```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
6
5
4
3
2
1
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

### Запуск файла

Данная программа выводит числа от N до 1 включительно.

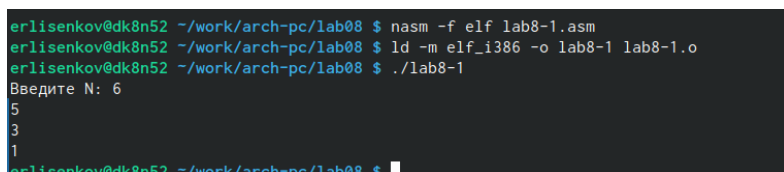
Изменяю текст программы, добавив изменение значения регистра ecx в цикле. (рис. ??).



```
lab8-1.asm [----] 9 L: [ 18+13 31/ 31] *(763 / 763b) <EOF>
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
call quit
```

### Изменение текста

Создаю исполняемый файл и проверяю его работу. (рис. ??).

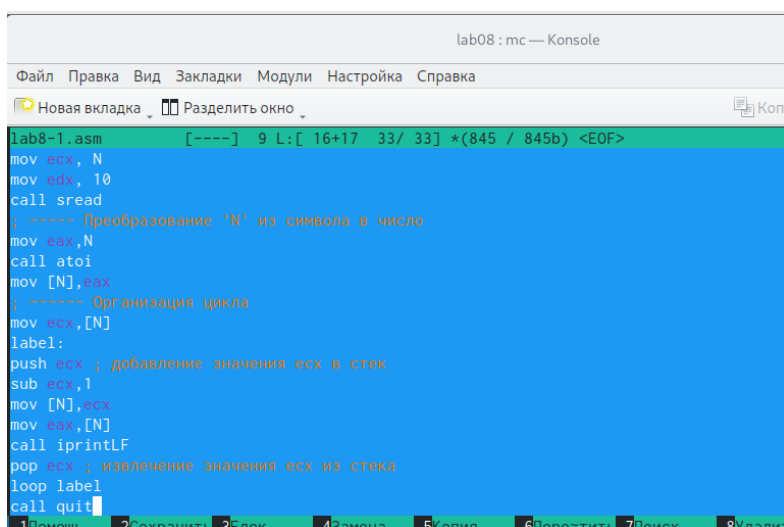


```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
1
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

### Запуск программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop. (рис. ??).



```
lab8-1.asm [----] 9 L: [ 16+17 33/ 33] *(845 / 845b) <EOF>
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]
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
call quit
```

### Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. ??).

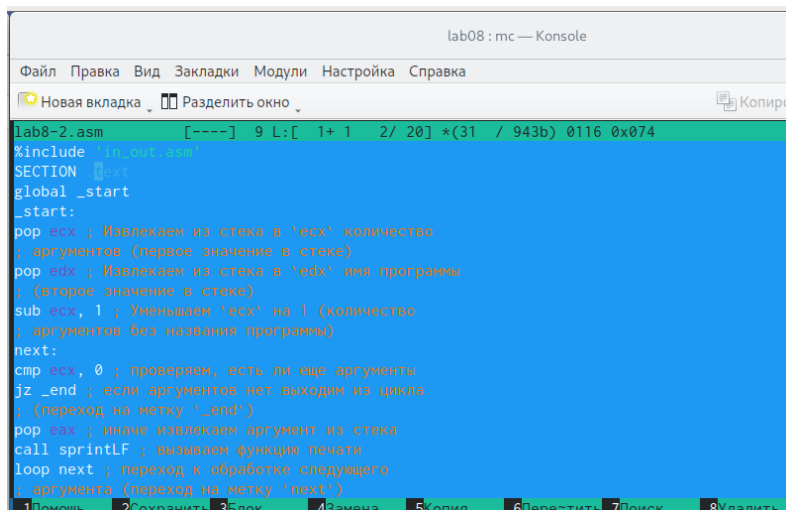
```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
4
3
2
1
0
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

### Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

## 4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.2. (рис. ??).



```
lab8-2.asm [----] 9 L: [ 1+ 1 2/ 20] *(31 / 943b) 0116 0x074
%include "ftn.inc.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:
; Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Переписать 7 Поиск 8 Удалить
```

### Ввод текста программы из листинга 8.2

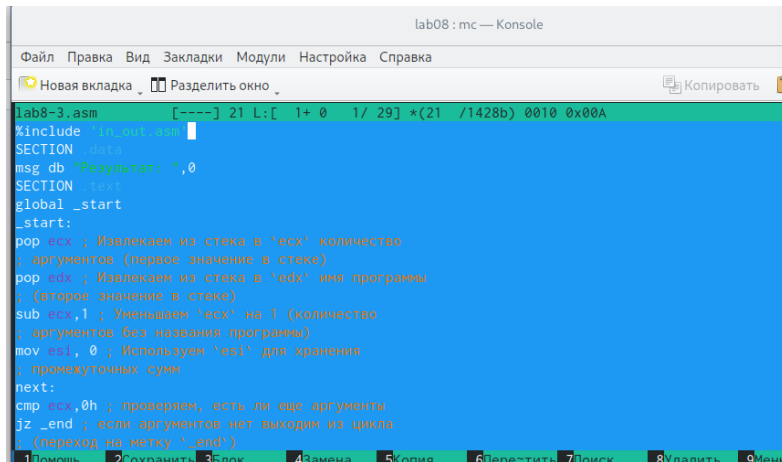
Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. ??).

```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-2
аргумент1
аргумент
2
аргумент 3
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

### Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличии от аргумента 3, поэтому из-за пробела программа считает “2” как отдельный аргумент.

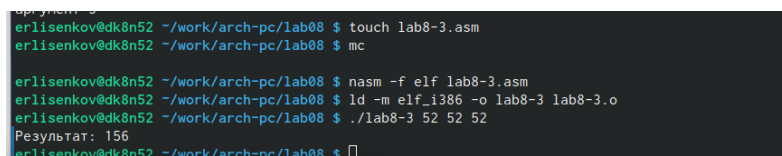
Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/archpc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. ??).



```
lab8-3.asm [----] 21 L: [ 1+ 0 1/ 29] *(21 /1428b) 0010 0x00A
%include "lab08.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')
1 Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Переписать 7 Поиск 8 Удалить 9 Меню
```

### Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. ??).

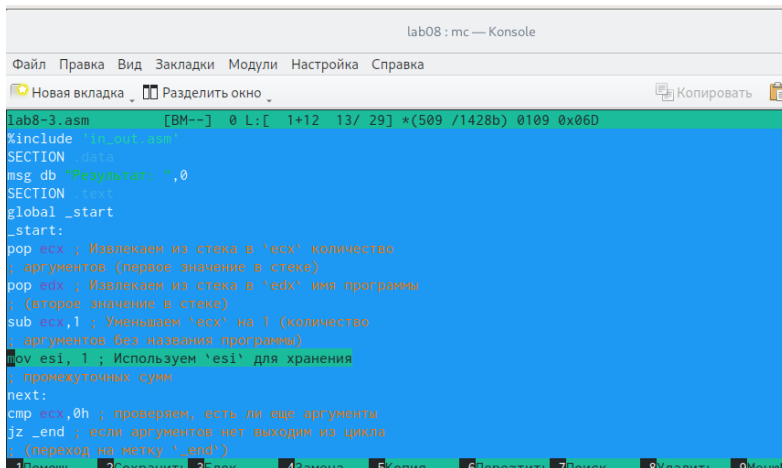


```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-3.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ mc

erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-3 52 52 52
Результат: 156
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

### Запуск исполняемого файла

Изменяю текст программы листинга 8.3 для выполнения программы. (рис. ??).



```
lab8-3.asm [BM--] 0 L: [ 1+12 13/ 29] *(509 /1428b) 0109 0x06D
%include "lab08.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 сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку '_end')
1 Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Переписать 7 Поиск 8 Удалить 9 Меню
```

### Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. ??).

```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-3 10 3 3
Результат: 17
```

*Запуск исполняемого файла*

### 4.3 Задание для самостоятельной работы

Пишу текст программы к своему варианту №2 (рис. ??).

```
ex.asm [----] 5 L: [ 1+11 12/ 26] *(156 / 301b) 0010 0
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,3
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,1
mul edi
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перетянуть 7
```

*Изменение программы*

Создаю исполняемый файл и проверьте его работу на нескольких наборах  $x = x_1, x_2, \dots, x_n$ . (рис. ??).

```
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ touch ex.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ mc
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf ex.asm
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o ex ex.o
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./ex 1 3 5
Результат: 36
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./ex 6 7 9 2
Результат: 84
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $ ./ex 11 43 63
Результат: 360
erlisenkov@dk8n52 ~/work/arch-pc/lab08 $
```

*Запуск исполняемого файла*

Замечу, что программа работает корректно!

Текст программы:

```
%include 'in_out.asm'
```

```
SECTION .data
msg db "Результат:",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,3
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,1
mul edi
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```



## 5 Выводы

Полученные знания являются актуальными для моего направления обучения. Они понадобятся мне в дальнейшей работе и помогут понимать работу компьютера.

## 6 Список литературы

Лабораторная работа №8. Программирование цикла. Обработка аргументов командной строки. Файл