

# Отчет по лабораторной работе №8

## Дисциплина: Архитектура компьютера

Зайцева Ульяна Владимировна

### Содержание

1	Цель работы .....	1
2	Задание .....	1
3	Теоретическое введение.....	1
4	Выполнение лабораторной работы.....	2
5	Выводы.....	8

### 1 Цель работы

Приобретение навыков 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

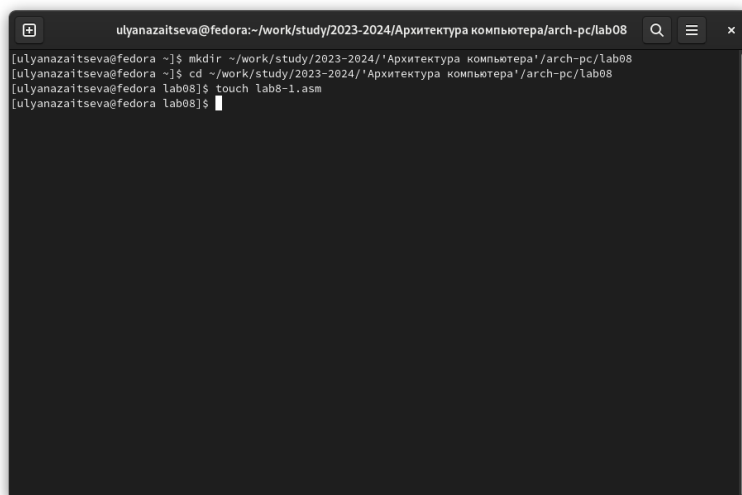
Команда `pop` извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр `esp`, после этого уменьшает значение регистра `esp` на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека 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

### 1. Реализация циклов в NASM

Перехожу в нужный мне каталог, создайте каталог для программ лабораторной работы № 8, перехожу в него и создаю файл `lab8-1.asm`(рис. ??).



```
ulyanazaitseva@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
[ulyanazaitseva@fedora ~]$ mkdir ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
[ulyanazaitseva@fedora ~]$ cd ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
[ulyanazaitseva@fedora lab08]$ touch lab8-1.asm
[ulyanazaitseva@fedora lab08]$
```

### *Создание рабочей папки и файла*

Ввожу в файл `lab8-1.asm` текст программы из листинга 8.1.(рис. ??) Создаю исполняемый файл и проверяю его работу.(рис. ??)

```
Открыть ▾ ⓘ • lab8-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08

;-----
; 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 printf
; ---- Ввод 'N'
mov ecx,N
mov edx,10
call read
; ---- 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 printf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

### Ввод текста программы листинга 8.1

```
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-1.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[ulyanazaitseva@fedora lab08]$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
[ulyanazaitseva@fedora lab08]$
```

### Проверка работы

Данная программа выводит числа от N до 1 включительно.

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

```

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 программы

Создаю файл и проверяю работу(рис. ??)

```

1 [ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-1.asm
2 [ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
3 [ulyanazaitseva@fedora lab08]$ ./lab8-1
4 Введите N: 6
5
6 5
7
8 3
9
10 1
11 [ulyanazaitseva@fedora lab08]$

```

### Проверка работы программы

В данном случае число проходов цикла не соответствует введенному значению.

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

```

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
call quit

```

### Изменения

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

```

1
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-1.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[ulyanazaitseva@fedora lab08]$ ./lab8-1
Введите N: 5
4
3
2
1
0
[ulyanazaitseva@fedora lab08]$

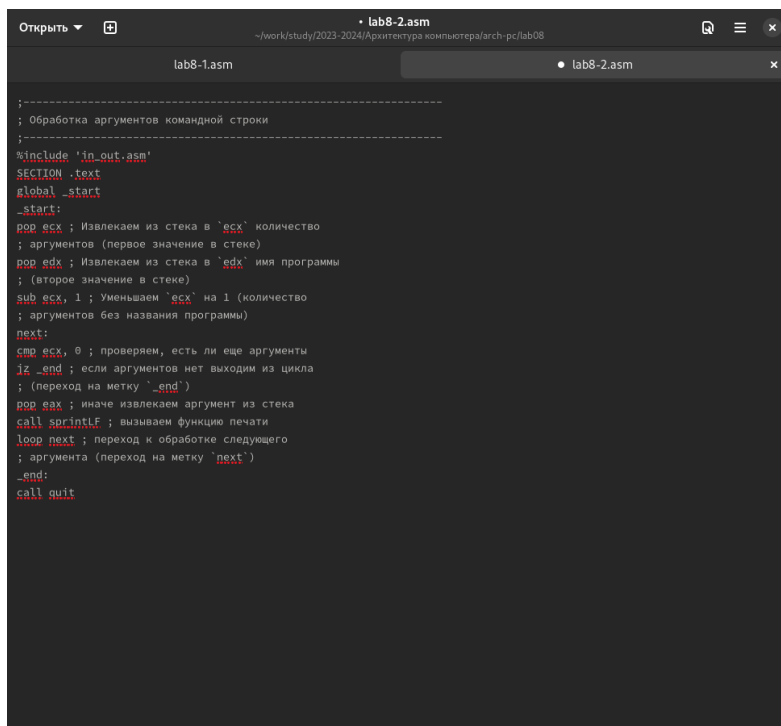
```

## Проверка работы программы

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

## 2. Обработка аргументов командной строки

Создаю файл lab8-2.asm и ввожу в него программу из листинга 2(рис. ??)



```

-----
; 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
; аргумента (переход на метку 'next')
_end:
call quit

```

## Листинг 2

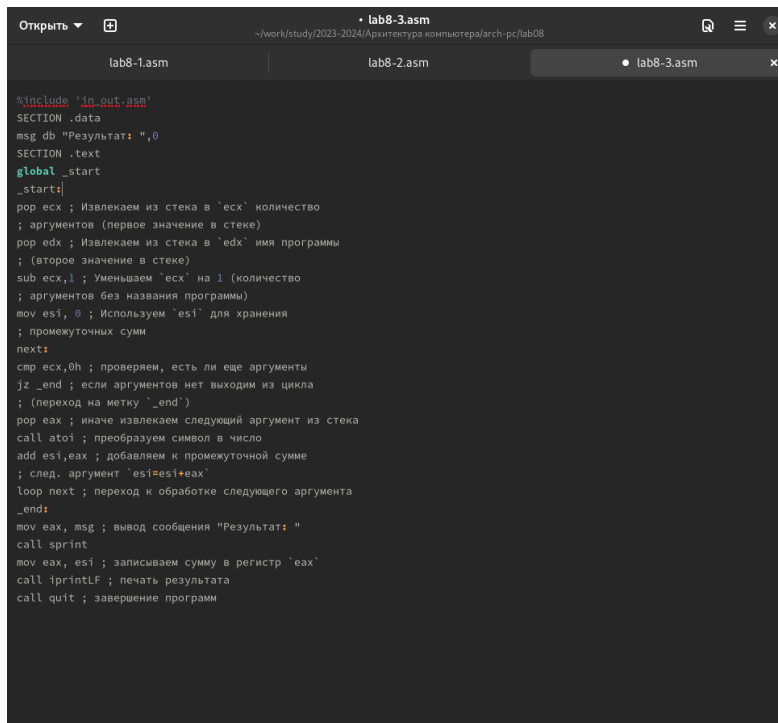
Создаю файл и проверяю работу(рис. ??)

```
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-2.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[ulyanazaitseva@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[ulyanazaitseva@fedora lab08]$
```

## Проверка

Программа вывела 4 аргумента, т.к. аргумент 2 не взят в кавычки программа считает “2” как отдельный аргумент из-за пробела.

Создаю файл lab8-3.asm и ввожу в него текст программы из листинга 8.3. (рис. ??).



```
lab8-3.asm
~/.work/study/2023-2024/Архитектура компьютера/arch-pclab08

lab8-1.asm lab8-2.asm lab8-3.asm 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
    ; 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 iprintf ; печать результата
    call quit ; завершение программы
```

## Листинг 3

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

```
2
аргумент 3
[ulyanazaitseva@fedora lab08]$ touch lab8-3.asm
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-3.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[ulyanazaitseva@fedora lab08]$ ./lab8-3 11 13 7
Результат: 31
[ulyanazaitseva@fedora lab08]$
```

## Проверка работы

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.(рис. ??).

```

; аргументов без названия программы)
mov esi, 1;
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 esi
mov esi, eax

```

*Изменяю текст программы*

Проверяю работу(рис. ??).

```

[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-3.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[ulyanazaitseva@fedora lab08]$ ./lab8-3 3 4 5
Результат: 60
[ulyanazaitseva@fedora lab08]$

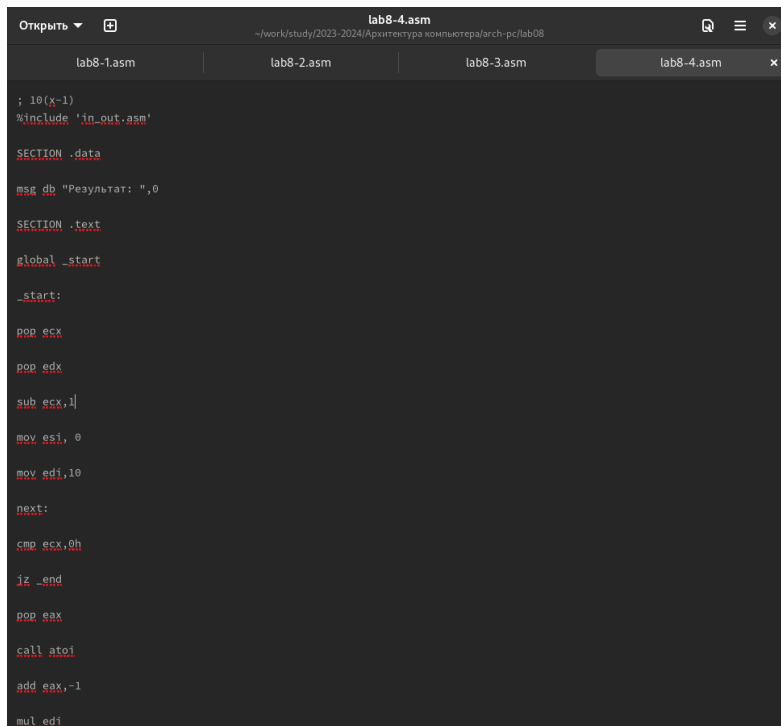
```

*Измененный листинг 8.3*

Программа работает исправно

### 3. Задание для самостоятельной работы

Пишу программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ .  
Мой вариант - 17. Выражение  $10(x-1)$ (рис. ??).



```
Открыть  lab8-4.asm  ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
lab8-1.asm  lab8-2.asm  lab8-3.asm  lab8-4.asm x

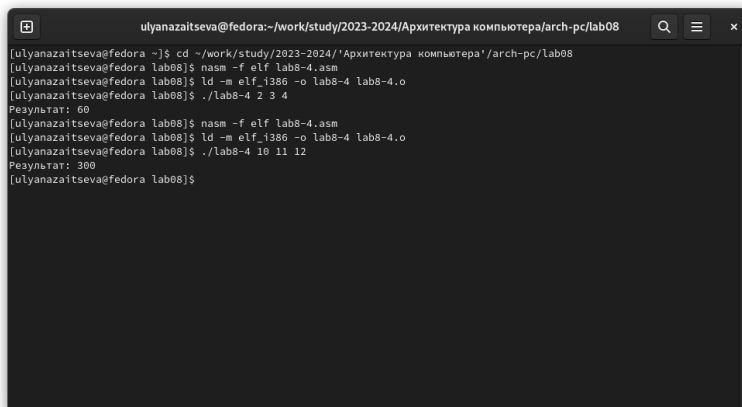
; 10(x-1)
%include 'in-out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start
_start:
    pop ecx
    pop edx
    sub ecx,1
    mov esi,0
    mov edi,10
next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    add eax,-1
    mul edi
```

## Программа 8-4

Компилирую и проверяю работу на нескольких x(рис. ??).



```
ulyanazaitseva@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
[ulyanazaitseva@fedora ~]$ cd ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab08
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-4.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[ulyanazaitseva@fedora lab08]$ ./lab8-4 2 3 4
Результат: 60
[ulyanazaitseva@fedora lab08]$ nasm -f elf lab8-4.asm
[ulyanazaitseva@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[ulyanazaitseva@fedora lab08]$ ./lab8-4 10 11 12
Результат: 380
[ulyanazaitseva@fedora lab08]$
```

## Проверка

Программа работает исправно

## 5 Выводы

При выполнении работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки