

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

по лабораторной работе № 9

дисциплина: Архитектура компьютера

Студент: Ужаков Магомед

Группа: НПИбд-02-22

**МОСКВА**

2022 г.

### Цель работы:

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

### Порядок выполнения лабораторной работы:

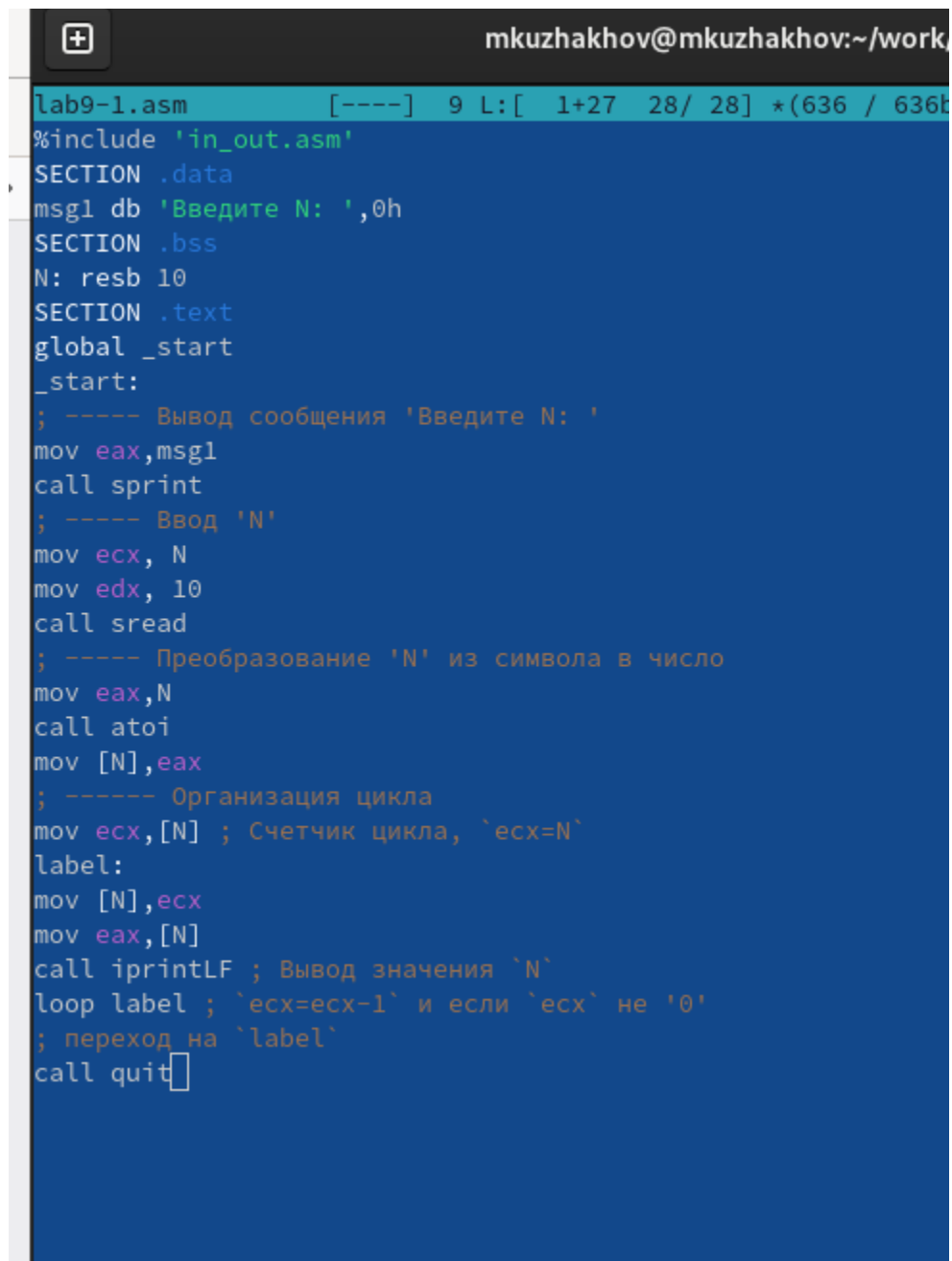
#### Реализация циклов в NASM.

Создадим каталог для программ лабораторной работы №9, перейдем в него и создадим нужный файл (рис. 1).

```
[mkuzhakhov@mkuzhakhov ~]$ mkdir ~/work/  
arch-pc/ study/  
[mkuzhakhov@mkuzhakhov ~]$ mkdir ~/work/arch-pc/lab09  
[mkuzhakhov@mkuzhakhov ~]$ cd ~/work/arch-pc/lab09  
[mkuzhakhov@mkuzhakhov lab09]$ touch lab9-1.asm  
[mkuzhakhov@mkuzhakhov lab09]$ mcedit lab9-1.asm  
  
[mkuzhakhov@mkuzhakhov lab09]$ █
```

рис. 1. Создание файла lab9-1.asm

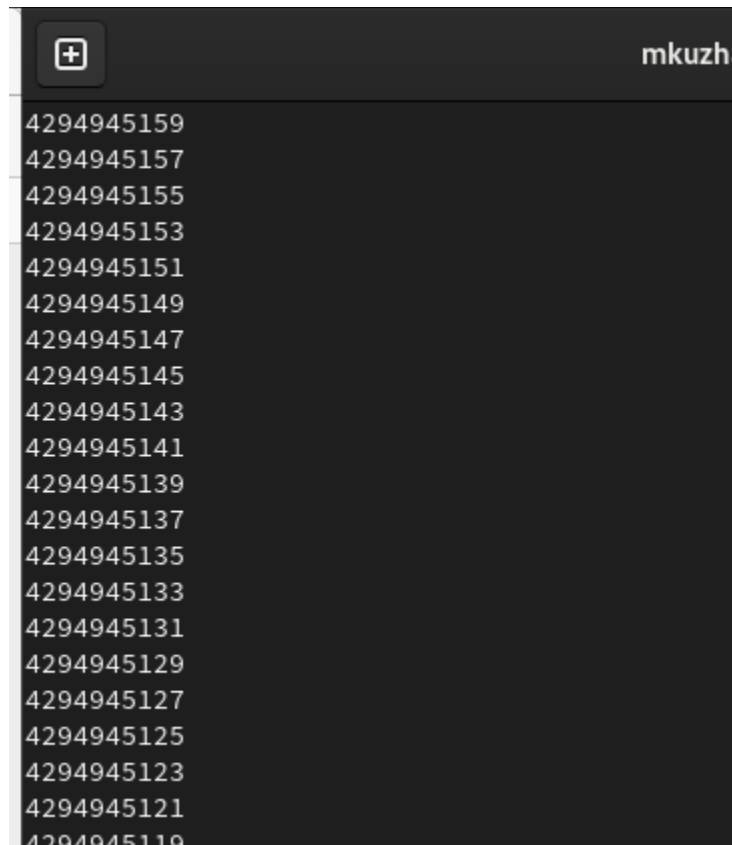
При реализации циклов в NASM с использованием инструкции loop необходимо помнить о том, что эта инструкция использует регистр ecx в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра ecx (рис. 2).



```
lab9-1.asm [----] 9 L: [ 1+27 28/ 28] *(636 / 636b
#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, `ecx=N`
label:
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

рис. 2. Текст программы lab9-1

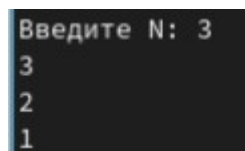
Создадим исполняемый файл и проверим его работу (рис. 3).



*рис. 3. Результат работы программы lab9-1*

Данный пример показывает, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Изменим текст программы добавив изменение значение регистра `ecx` в цикле по следующему примеру

Создадим исполняемый файл и проверим его работу



*рис. 5. Результат работы измененной программы lab9-1*

Как видим, все работает. Регистр `ecx` принимает все значения от `N` до 1 включительно, что соответствует числу проходов цикла, введенному с клавиатуры.

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесем изменения в текст

программы по примеру, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop

Создадим исполняемый файл и проверим его работу

Как видим, программа работает корректно, число проходов цикла соответствует значению N, введенному с клавиатуры.

### **Обработка аргументов командной строки.**

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы. При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов. Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки (рис. 8). Создадим в каталоге лабораторной работы №9 файл lab9-2

Затем создадим исполняемый файл и запустим программу, указав следующие аргументы (рис. 9).

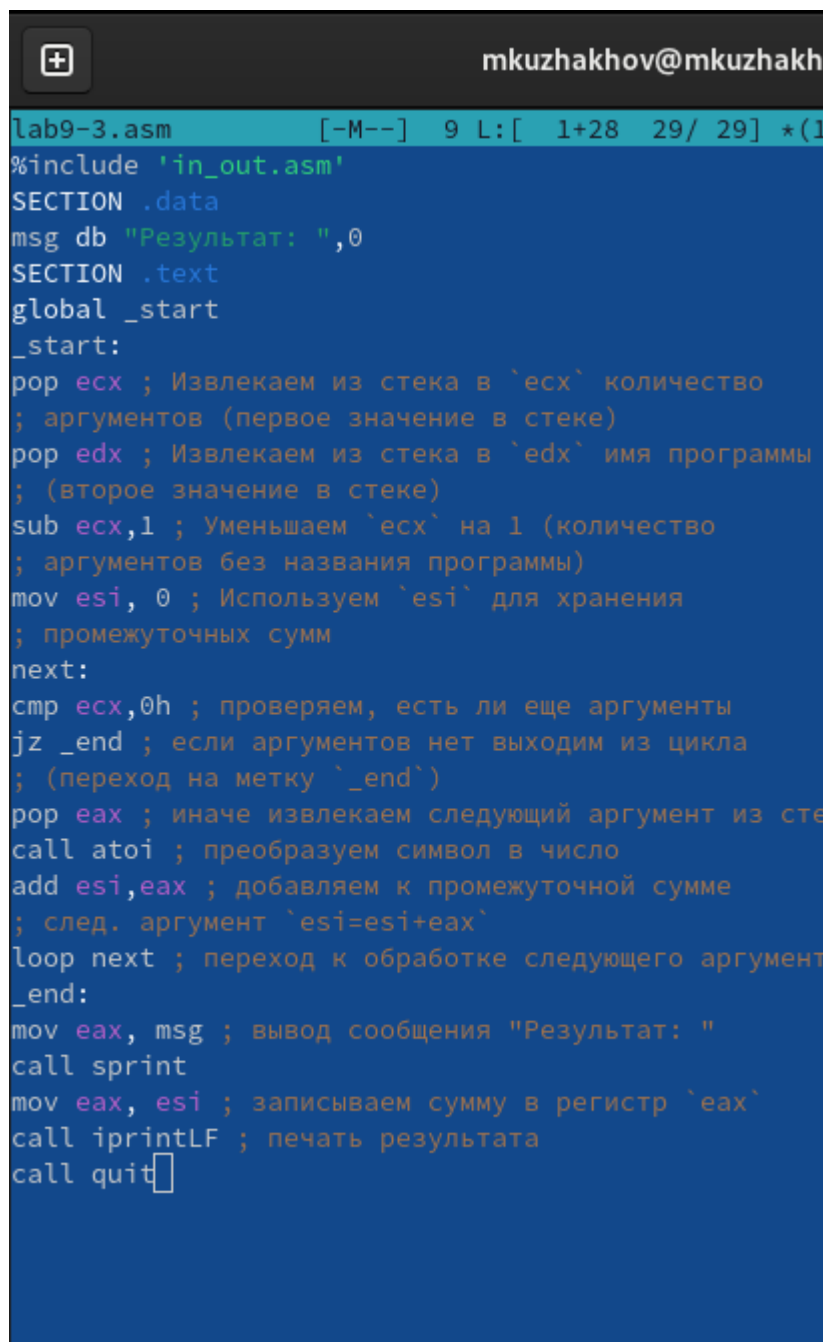
```
in_out.asm lab9-1 lab9-1.asm lab9-1.o lab9-2.asm
[mkuzhakhov@mkuzhakhov lab09]$ mcedit lab9-2.asm

[mkuzhakhov@mkuzhakhov lab09]$ nasm -f elf lab9-2.asm
[mkuzhakhov@mkuzhakhov lab09]$ ld -m elf_i386 lab9-2.o -o lab9-2
[mkuzhakhov@mkuzhakhov lab09]$ ./lab9-2
[mkuzhakhov@mkuzhakhov lab09]$ ./lab9-2 arg1 arg 2 'ahu3'
arg1
arg
2
ahu3
[mkuzhakhov@mkuzhakhov lab09]$
```

*рис. 9. Результат работы программы lab9-2*

Как видим, программа восприняла “аргумент” и “2” как отдельные аргументы, в то время как ‘аргумент 3’ как один. Соответственно программой было обработано 4 аргумента.

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл lab9-3.asm в том же каталоге и введем в него следующий текст программы (рис. 10).



```
lab9-3.asm [-M--] 9 L:[ 1+28 29/ 29] *(1
%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
```

рис. 10. Текст программы lab9-3

Затем создадим исполняемый файл и запустим его, указав аргументы (рис. 11).

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

*рис. 11. Результат работы программы lab9-3*

Как видим, все работает корректно.

Изменим строку

*add esi,ecx*

на

*move ebx, eax*

*mov eax, esi*

*mul ecx*

*mov esi, eax*

а также присвоим esi значение 1, чтобы программа выводила произведение аргументов командной строки и запустим ее

### **Вывод:**

Во время выполнения лабораторной работы были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки.