

# **Лабораторная работа №8**

**Дисциплина: Архитектура компьютера**

Савостин Олег

# 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>9</b>
4.1	Реализация циклов в NASM. . . . .	9
4.2	Обработка аргументов командной строки . . . . .	12
<b>5</b>	<b>Выводы</b>	<b>19</b>
	<b>Список литературы</b>	<b>20</b>

# Список иллюстраций

4.1	Новый каталог и файл. . . . .	9
4.2	Текст из Листинга 8.1 . . . . .	9
4.3	Исполняемый файл Листинг 8.1 . . . . .	10
4.4	Изменение текста кода . . . . .	10
4.5	Создание и запуск измененного исполняемого файла . . . . .	11
4.6	Добавление рор есх . . . . .	11
4.7	Корректно работающий lab8-1.asm . . . . .	12
4.8	Создание нового файла . . . . .	12
4.9	Текст из Листинга 8.2 в новом файле . . . . .	13
4.10	Запуск исполняемого файла Листинга 8.2 . . . . .	13
4.11	Создание третьего файла . . . . .	13
4.12	Текст из Листинга 8.3 . . . . .	14
4.13	Запуск третьего исполняемого файла . . . . .	14
4.14	Измененный файл Листинга 8.3 . . . . .	15
4.15	Вычисление произведения аргументов . . . . .	16
4.16	Код программы . . . . .	17
4.17	Запуск исполняемого файла. . . . .	17

## **Список таблиц**

# 1 Цель работы

Целью данной лабораторной работы является 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. Выполнение заданий лабораторной 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

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

Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

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

Аналогично команде записи в стек существует команда рора, которая восстанавливает из стека все регистры общего назначения, и команда popf для переме-

щения значений из вершины стека в регистр флагов.

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



## 4 Выполнение лабораторной работы

### 4.1 Реализация циклов в NASM.

Сперва, создаю каталог и файл для лабораторной работы.(рис. 4.1).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mkdir lab08
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd lab08
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ touch lab8-1.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.1: Новый каталог и файл.

Вставляю в файл текст из Листинга 8.1 (рис. 4.2).

```
lab8-1.asm [----] 0 L: [ 1+ 0 1/ 29] *(0 / 637b) 0037 0x025 [*] [X]
#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. 4.2: Текст из Листинга 8.1

Создаю исполняемый файл и проверяю его работу. Он считывает числа с N до 1 (рис. 4.3).

```
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 4
4
3
2
1
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.3: Исполняемый файл Листинг 8.1

Изменяю текст файла, добавив изменение значение регистра eax в цикле (рис. 4.4).

```
lab8-1.asm      [----] 0 L:[ 1+ 6 7/ 30] *(106 / 627b) 0103[*][X]
#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:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label ; `ecx=ecx-1` и если `ecx` не `0`
; переход на `label`
call quit
```

Рис. 4.4: Изменение текста кода

Создаю исполняемый файл. При запуске исполняемого файла, код считывает с N до 0, но затем начинается счет с 4294967294.(рис. 4.5).

```

savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 1
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
4294967270
4294967268
4294967266
4294967264
4294967262
4294967260
4294967258
4294967256
4294967254
4294967252
4294967250
4294967248

```

Рис. 4.5: Создание и запуск измененного исполняемого файла

Чтобы сохранить корректность программы, использую команду `ror` (рис. 4.6).

```

lab8-1.asm [----] 0 L: [ 1+14 15/ 32] *(256 / 644b) 0099 0x063
#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:
push ecx
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit

```

Рис. 4.6: Добавление `ror ecx`

Создаю исполняемый файл. Все работает корректно, однако, он теперь считает числа с N до 0 (рис. 4.7).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-1.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.7: Корректно работающий lab8-1.asm

## 4.2 Обработка аргументов командной строки

Создаю новый файл (рис. 4.8) и записываю в него текст из Листинга 8.2 (рис. 4.9).

```
$ touch lab8-2.asm
```

Рис. 4.8: Создание нового файла

```

lab8-2.asm      [----]  0 L: [ 1+ 0  1/ 22] *(0  / 945b) 0010 0x00A  [*]
%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

```

Рис. 4.9: Текст из Листинга 8.2 в новом файле

Теперь запускаю файл с тремя аргументами. Обработались все три, однако, аргумент 2 был выписан по другому. Это произошло, так как он засчитал этот аргумент как два разных элемента. Если записать его как третий аргумент, то он будет выведен идентично ему. (рис. 4.10).

```

savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab08/CODES$ nasm -f elf lab8-2.asm
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab08/CODES$ ld -m elf_i386 -o lab8-2 lab8-2.o
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab08/CODES$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab08/CODES$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 4.10: Запуск исполняемого файла Листинга 8.2

Создаю третий файл(рис. 4.11) и ввожу в него код из Листинга 8.3(рис. 4.12).

```

touch lab8-3.asm

```

Рис. 4.11: Создание третьего файла

```

lab8-3.asm      [----]  0 L: [ 1+ 0  1/ 31] *(0  /1430b) 0037 0x025  [*]
#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

```

Рис. 4.12: Текст из Листинга 8.3

Создаю исполняемый файл и проверяю на его работу. (рис. 4.13).

```

savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3
Результат: 0
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3
Результат: 6
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 15
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$

```

Рис. 4.13: Запуск третьего исполняемого файла

Изменяю код так, чтобы вместо вычисления суммы, он вычислял произведение аргументов (рис. 4.14).

```

lab8-3.asm [----] 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,1
mov eax,1

next:
cmp ecx,0h ;
jz _end.

pop eax.
call atoi
mov ebx,eax
mov eax,esi
mul ebx
mov esi,eax
loop next

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

```

Рис. 4.14: Измененный файл Листинга 8.3

Создаю исполняемый файл и проверяю. Все готово(рис. 4.15).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf lab8-3.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3
Результат: 1
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3
Результат: 6
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 120
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$
```

Рис. 4.15: Вычисление произведения аргументов

КОД

%include 'in\_out.asm'

SECTION .data

msg db "Результат:",0

SECTION .text

global \_start

\_start:

pop ecx

pop edx

sub ecx,1

mov esi,1

mov eax,1

next:

cmp ecx,0h ;

jz \_end

pop eax

call atoi

mov ebx,eax

mov eax,esi

mul ebx

mov esi,eax

loop next

\_end:



```
mov eax, msg
```

```
call sprint
```

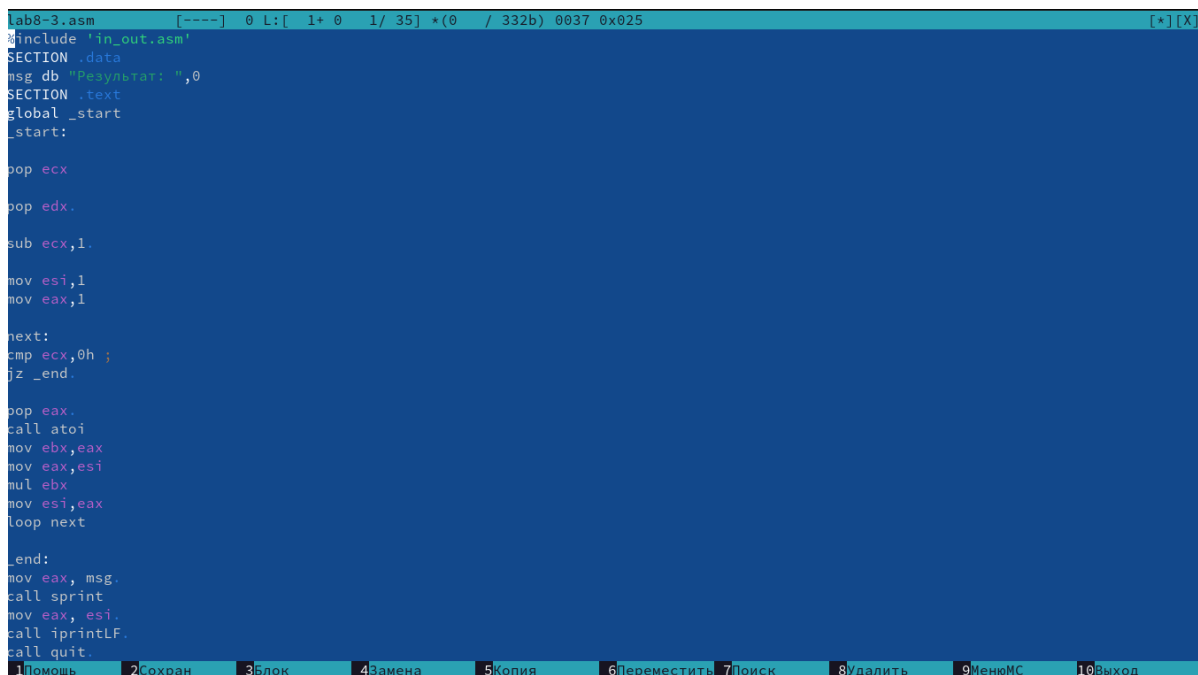
```
mov eax, esi
```

```
call iprintLF
```

```
call quit
```

##Задание для самостоятельной работы

У меня 13 вариант, я сделал программу, вычисляющая сумму всех  $f(x)=12x-7$  (рис. 4.16).



```
lab8-3.asm [----] 0 L: [ 1+ 0 1/ 35] *(0 / 332b) 0037 0x025 [*] [X]
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:

    pop ecx
    pop edx
    sub ecx,1

    mov esi,1
    mov eax,1

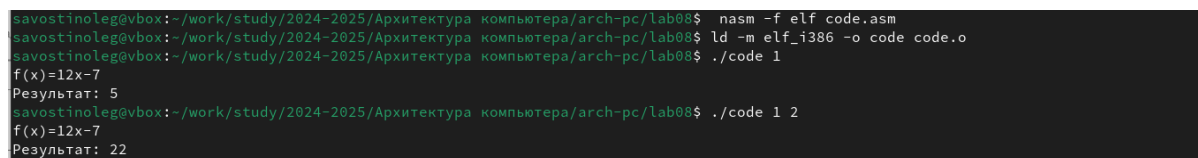
next:
    cmp ecx,0h ;
    jz _end

    pop eax
    call atoi
    mov ebx,eax
    mov eax,esi
    mul ebx
    mov esi,eax
    loop next

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

Рис. 4.16: Код программы

Проверяю на его правильность. Программа работает (рис. 4.17).



```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ nasm -f elf code.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ld -m elf_i386 -o code code.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./code 1
f(x)=12x-7
Результат: 5
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08$ ./code 1 2
f(x)=12x-7
Результат: 22
```

Рис. 4.17: Запуск исполняемого файла.

КОД

```

%include 'in_out.asm'
SECTION .data
funct db "f(x)=12x-7",0
msg db "Результат:",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov eax,funct
call sprintLF
next:
cmp ecx,0h ;
jz _end
mov ebx,12
pop eax
call atoi
mul ebx
sub eax,7
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

## **5 Выводы**

В ходе данной работы я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки

# Список литературы

Лабораторная работа №8 ::: {#refs} :::