

**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**  
**по лабораторной работе № 8**

*дисциплина: Архитектура компьютера*

Студент: Бессонов Андрей Максимович

Группа: НКАбд - 01 - 25

**МОСКВА**

2025 г.

## **Содержание**

1. Цель работы	4
2. Теоретическое введение	4
3. Выполнение лабораторной работы	5
4. Выполнение самостоятельной работы	9
5. Выводы	11

## **Список иллюстраций**

Рис 3.1: 31	5
Рис 3.2: 32	6
Рис 3.3: 33	6
Рис 3.4: 34	7
Рис 3.5: 35	8
Рис 3.6: 36	8
Рис 3.7: 37	8
Рис 3.8: 38	9
Рис 3.9: 39	9
Рис 4.1: 41	10
Рис 4.2: 41	11

# **1. Цель работы**

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

## **2. Теоретическое введение**

### **2.1. Организация стека**

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. Для стека существует две основные операции:

- добавление элемента в вершину стека (push);
- извлечение элемента из вершины стека (pop).

#### **2.1.1. Добавление элемента в стек.**

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 operandов.

**2.1.2. Извлечение элемента из стека.** 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 operand, 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.

Аналогично команде записи в стек существует команда `popa`, которая восстанавливает из стека все регистры общего назначения, и команда `popf` для перемещения значений из вершины стека в регистр флагов.

2.2. Инструкции организации циклов Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `esx`. Наиболее простой является инструкция `loop`.

Инструкция `loop` выполняется в два этапа. Сначала из регистра `esx` вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.

### 3. Выполнение лабораторной работы

Создадим каталог для программам лабораторной работы № 8, перейдем в него и создадим файл `lab8-1.asm`.

```
ambessonov@fedora:~$ mkdir ~/work/arch-pc/lab08
ambessonov@fedora:~$ cd ~/work/arch-pc/lab08
ambessonov@fedora:~/work/arch-pc/lab08$ touch lab8-1.asm
```

Рис 3.1: 31

Введем в файл `lab8-1.asm` текст программы из листинга 8.1. Создадим исполняемый файл и проверим его работу.

The screenshot shows a terminal window with the following content:

```
%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 3.2: 32

The screenshot shows a terminal window with the following content:

```
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 13
13
12
11
10
9
8
7
6
5
4
3
2
1
ambessonov@fedora:~/work/arch-pc/lab08$
```

Рис 3.3: 33

Изменим текст программы добавим изменение значение регистра ecx в цикле.

Создадим исполняемый файл и проверим его работу.

Какие значения принимает регистр ecx в цикле?

Ответ: ecx принимает значения:  $5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 0xFFFFFFFF\dots$

Соответствует ли число проходов цикла значению  $N$  введенному с клавиатуры?

Ответ: нет, число проходов цикла не соответствует значению  $N$ .

```
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 1
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
```

Рис 3.4: 34

Внесем изменения в текст программы 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.

Соответствует ли в данном случае число проходов цикла значению  $N$  введенному с клавиатуры?

Ответ: да, число проходов цикла полностью соответствует значению  $N$ .

```
[1]+ Остановлен ./lab8-1
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 13
12
11
10
9
8
7
6
5
4
3
2
1
0
ambessonov@fedora:~/work/arch-pc/lab08$
```

Рис 3.5: 35

Создадим файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.2. Создадим исполняемый файл и запустим его, указав аргументы:

```
user@dk4n31:~$ ./lab8-2 аргумент1 аргумент2 'аргумент3'
```

Сколько аргументов было обработано программой?

Ответ: программа обработала 4 аргумента.

```
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент2 'аргумент3'

аргумент1
аргумент
2
аргумент 3
ambessonov@fedora:~/work/arch-pc/lab08$ █
```

Рис 3.6: 36

Создадим файл lab8-3.asm в каталоге ~/work/archpc/lab08 и введем в него текст программы из листинга 8.3. Создадим исполняемый файл и запустим его, указав аргументы.

```
ambessonov@fedora:~/work/arch-pc/lab08$ touch lab8-3.asm
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
ambessonov@fedora:~/work/arch-pc/lab08$
```

Рис 3.7: 37

Изменим текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

```
Открыть + • lab8-3.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
    pop ecx      ; Извлекаем количество аргументов
    pop edx      ; Извлекаем имя программы
    sub ecx,1    ; Уменьшаем количество аргументов на 1
    mov esi, 1    ; Используем `esi` для хранения произведения (начинаем с 1 для умножения)
next:
    cmp ecx,0h    ; проверяем, есть ли еще аргументы
    jz _end        ; если аргументов нет - выход
    pop eax        ; извлекаем следующий аргумент из стека
    call atoi      ; преобразуем символ в число
    imul esi, eax ; умножаем промежуточное произведение на 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 произведение в регистр `eax`
    call iprintLF ; печать результата
    call quit      ; завершение программы
```

Рис 3.8: 38

```
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
ambessonov@fedora:~/work/arch-pc/lab08$
```

Рис 3.9: 39

Вывод: выполнили задания лабораторной работы.

## 4. Выполнение самостоятельной работы

Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ , т.е. программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Значения  $x_i$  передаются как аргументы.

Создадим исполняемый файл и проверим его работу на нескольких наборах  $x=x_1, x_2, \dots, x_n$ .

```
Открыть ▾  [+] work.asm
~/work/arch-pc/lab08

%include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x)=6x+13",0
msg_result db "Результат: ",0
SECTION .text
global _start
_start:
_start:
pop ecx          ; Извлекаем количество аргументов
pop edx          ; Извлекаем имя программы
sub ecx, 1       ; Уменьшаем количество аргументов на 1
mov esi, 0        ; Используем esi для хранения суммы

; Вывод сообщения о функции
mov eax, msg_func
call sprintLF

next:
cmp ecx, 0        ; Проверяем, есть ли еще аргументы
jz _end           ; Если аргументов нет - выход

pop eax          ; Извлекаем аргумент (x) из стека
call atoi         ; Преобразуем в число

; Вычисляем f(x) = 6x + 13
mov ebx, eax      ; Сохраняем x в ebx
imul eax, 6       ; eax = 6x
add eax, 13        ; eax = 6x + 13

; Добавляем результат к общей сумме
add esi, eax      ; esi = esi + f(x)

loop next         ; Переход к следующему аргументу

_end:
; Вывод результата
mov eax, msg_result
call sprint
mov eax, esi      ; Записываем сумму в eax
call iprintLF     ; Печать результата
call quit         ; Завершение программы
```

Рис 4.1: 41

```
ambessonov@fedora:~/work/arch-pc/lab08$ touch work.asm
ambessonov@fedora:~/work/arch-pc/lab08$ nasm -f elf work.asm
ambessonov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o work work.o
ambessonov@fedora:~/work/arch-pc/lab08$ ./work 1
Функция: f(x)=6x+13
Результат: 19
ambessonov@fedora:~/work/arch-pc/lab08$ ./work 1 2 3
Функция: f(x)=6x+13
Результат: 75
ambessonov@fedora:~/work/arch-pc/lab08$ ./work 0 5 10
Функция: f(x)=6x+13
Результат: 129
ambessonov@fedora:~/work/arch-pc/lab08$
```

Рис 4.2: 42

Вывод: выполнили задания самостоятельной работы, отработали навыки, полученные в ходе лабораторной работы.

## 5. Выводы

Приобрели навыки написания программ с использованием циклов и обработкой аргументов командной строки.