

Российский университет дружбы народов  
Факультет физико-математически и естественных наук

Отчёт  
по лабораторной работе №7  
по дисциплине:  
архитектура компьютеров и операционные  
системы

Студент: Аманов Рустам Марксович: НКАбд 01-23

№ ст. Билета: 1032234130

МОСКВА  
2023 Г

# Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выполнение заданий для самостоятельной работы	18
6	Выводы	21
	Список литературы	22

## Список иллюстраций

4.1	Создание файла.....	Ошибка! Закладка не определена.
4.2	Текстфайла lab8-1.asm.....	8
4.3	Работа файла.....	8
4.4	Измененный текст программы.....	9
4.5	Созданный файл с ошибкой.....	9
4.6	Измененный текст программы.....	10
4.7	Работа файла.....	10
4.8	Название рисунка.....	11
4.9	Новый текст файла.....	11
4.10	Работа файла.....	12
4.11	Создание файла.....	12
4.12	Текст файла.....	13
4.13	Работа файла.....	13
4.14	Измененный текст файла.....	14
4.15	Работа файла.....	15
5.1	Создание файла.....	15
5.2	Работа файла.....	15

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

Напишите программу, которая находит сумму значений функции  $f(x)$  для  $x = 1, 2, \dots, n$ , т.е. программа должна выводить значение  $f(1) + f(2) + \dots + f(n)$ . Значения передаются как аргументы. Вид функции  $f(x)$  выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах  $n = 1, 2, \dots$ .

### 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. На рис. 8.1 показана схема организации стека в процессоре. 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). 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. push -10 ; Поместить -10 в стек push ebx ; Поместить значение регистра ebx в стек push [buf] ; Поместить значение переменной buf в стек push word [ax] ; Поместить в стек слово по адресу в ax 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. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и

остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. Примеры: `pop eax` ; Поместить значение из стека в регистр `eax pop [buf]` ; Поместить значение из стека в `buf pop word[si]` ; Поместить значение из стека в слово по адресу в `si` 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. 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, типичная структура которого имеет следующий вид: `mov ecx, 100` ; Количество проходов `NextStep:.....`; тело цикла... `loop NextStep` ; Повторить `ecx` раз от метки `NextStep` Инструкция `loop` выполняется в два этапа. Сначала из регистра `ecx` 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`.

## 4 Выполнение лабораторной работы

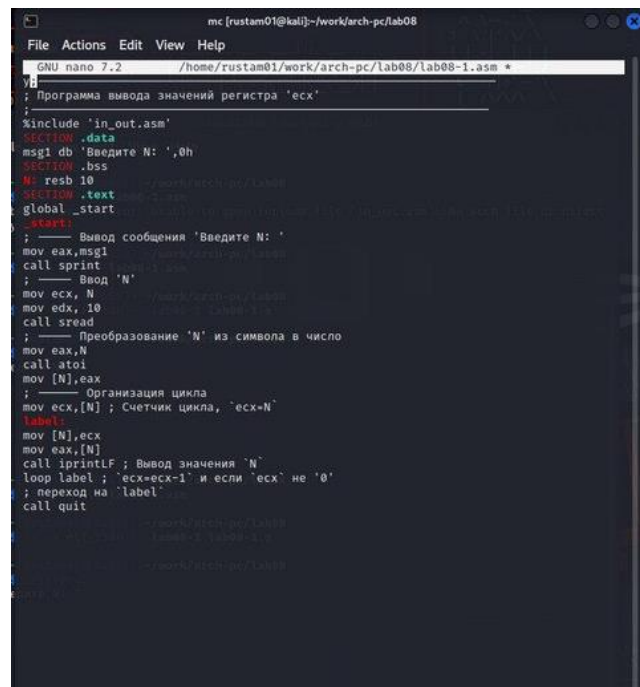
- 1) Создаю папку, файл в этой папке, проверяю его создание



```
rustam01@kali: ~/work/arch-pc/lab08
File Actions Edit View Help
(rustam01@kali)~$ mkdir -p ~/work/arch-pc/lab08
(rustam01@kali)~$ cd ~/work/arch-pc/lab08
(rustam01@kali)~/work/arch-pc/lab08$ touch lab08-1.asm
(rustam01@kali)~/work/arch-pc/lab08$ ls
lab08-1.asm
(rustam01@kali)~/work/arch-pc/lab08$
```

Рис. 4.1: Создание файла

2) Ввожу в файл lab8-1.asm текст программы из листинга 8.1



```
mc [rustam01@kali]~/work/arch-pc/lab08
File Actions Edit View Help
GNU nano 7.2 /home/rustam01/work/arch-pc/lab08/lab08-1.asm *
; 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
; ----- 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 iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 4.2: Текст файла lab8-1.asm

3) Создание исполняемого файла и проверка его работы



```
(rustam01@kali)~/work/arch-pc/lab08
$ nasm -f elf lab08-1.asm
(rustam01@kali)~/work/arch-pc/lab08
$ ld -m elf_i386 -o lab08-1 lab08-1.o
(rustam01@kali)~/work/arch-pc/lab08
$ ./lab08-1
Введите N: 7
7
6
5
4
3
2
1
```

Рис. 4.3: Работа файла

4) Изменение текста программы, добавляю изменение значение регистра ecx в цикле



```

mc [rustam01@kali]~/work/arch-pc/lab08
File Actions Edit View Help
GNU nano 7.2 /home/rustam01/work/arch-pc/lab08/lab08-1.asm
; Программа вывода значений регистра 'ecx'
;
#include "in_out.asm"
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
;
; 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 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:
push ecx ;
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'
pop ecx ;
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit

```

Рис. 4.4: Измененный текст программы

- 5) Создаю файл и запускаю его. Значения идут с приблизительно 4350000000 и работает некорректно. Работу заскринить не получилось

```

(rustam01@kali)~/work/arch-pc/lab08
$ nasm -f elf lab08-1.asm

(rustam01@kali)~/work/arch-pc/lab08
$ ld -m elf_i386 -o lab08-1 lab08-1.o

(rustam01@kali)~/work/arch-pc/lab08
$ ./lab08-1
Введите N: 

```

Рис. 4.5: Созданный файл с ошибкой

- 6) Изменяю текст программы, добавив изменение значение регистра ecx в цикле

```
mc [rustam01@kali]~/work/arch-pc/lab08
File Actions Edit View Help
/home/rustam01/work/arch-pc/lab08/lab08-1.asm
; Программа вывода значений регистра 'ecx'
;
#include 'in_out.asm'
;
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
;
; 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 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:
push ecx
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 4.6: Измененный текст программы

#### 7) Работа программы и создание файла

```
rustam01@kali: ~/work/arch-pc/lab08
File Actions Edit View Help
$ nasm -f elf lab08-1.asm
(rustam01@kali)~/work/arch-pc/lab08
$ ld -m elf_i386 -o lab08-1 lab08-1.o
(rustam01@kali)~/work/arch-pc/lab08
$ ./lab08-1
Введите N: 6
5
4
3
2
1
0
```

Рис. 4.7: Работа файла

Значения идут от N-1 до 0. Их количество соответствует N, введенному с клавиатуры

#### 8) Создаю файл lab8-2.asm

```
(rustam01@kali)-[~/work/arch-pc/lab08]
$ touch lab08-2.asm

(rustam01@kali)-[~/work/arch-pc/lab08]
$ ls
in_out.asm  lab08-1  lab08-1.asm  lab08-1.asm.save  lab08-1.o  lab08-2.asm
```

Рис. 4.8: Название рисунка

- 9) Вношу изменения в текст программы добавив команды push и pop для сохранения значения счетчика цикла loop

```
mc [rustam01@kali]-/work/arch-pc/lab08
File Actions Edit View Help
GNU nano 7.2 /home/rustam01/work/arch-pc/lab08/lab08-2.asm
#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: Новый текст файла

- 10) Создание файла и его работа

```
(rustam01@kali)-[~/work/arch-pc/lab08]
$ touch lab08-2.asm

(rustam01@kali)-[~/work/arch-pc/lab08]
$ ls
in_out.asm  lab08-1  lab08-1.asm  lab08-1.asm.save  lab08-1.o  lab08-2.asm

(rustam01@kali)-[~/work/arch-pc/lab08]
$ nasm -f elf lab08-2.asm

(rustam01@kali)-[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab08-2 lab08-2.o

(rustam01@kali)-[~/work/arch-pc/lab08]
$ ./lab08-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.10: Работа файла

Программой было обработано 4 аргумента

11) Создаю файл lab8-3.asm

```
(rustam01@kali)~[/work/arch-pc/lab08]
$ touch lab08-3.asm

(rustam01@kali)~[/work/arch-pc/lab08]
$ ls
in_out.asm  lab08-1.asm  lab08-1.o  lab08-2.asm  lab08-3.asm
lab08-1     lab08-1.asm.save  lab08-2     lab08-2.o
```

Рис. 4.11: Создание файла

12) Ввожу в него текст программы из листинга 8.3

```
mc [rustam01@kali]:~/work/arch-pc/lab08
File Actions Edit View Help
GNU nano 7.2 /home/rustam01/work/arch-pc/lab08/lab08-3.asm
#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 файла

### 13) Создание исполняемого файла и его работа

```
(rustam01@kali)~[/work/arch-pc/lab08]
$ nasm -f elf lab08-3.asm

(rustam01@kali)~[/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab08-3 lab08-3.o

(rustam01@kali)~[/work/arch-pc/lab08]
$ ./lab08-3 12 13 7 10 5
Результат: 47

(rustam01@kali)~[/work/arch-pc/lab08]
$
```

Рис. 4.13: Работа файла

### 14) Меняю текст файла, чтоб он выводил произведение

```
mc [rustam01@kali]~[/work/arch-pc/lab08]
File Actions Edit View Help
GNU nano 7.2 /home/rustam01/work/arch-pc/lab08/lab08-3.asm
#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,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')
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    mul esi ; добавляем к промежуточной сумме
; след. аргумент 'esi=esi+eax'
    mov 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. 4.14: Измененный текст файла

```
%include 'in_out.asm' SECTION
.data msg db "Результат: ",0
SECTION .text global _start
_start:
; Извлекаем из стека в `ecx` количество ; аргументов (первое
; значение в стеке) mov ecx, [esp+4]
; 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:
mov ecx, 0h ; проверяем, есть ли еще аргументы jz _end ; если
; аргументов нет выходим из цикла
; (переход на метку `_end`) mov eax, [esp+ecx*4] ; иначе извлекаем следующий
; аргумент из стека call atoi ; преобразуем символ в число mul esi,
; добавляем к промежуточной сумме ; след. аргумент `esi=esi+eax` mov
; 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 ; завершение программы
```

#### 15) Создание и работа файла



```
(rustam01@kali) - [~/work/arch-pc/lab08]
$ nasm -f elf lab08-3.asm

(rustam01@kali) - [~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab08-3 lab08-3.o

(rustam01@kali) - [~/work/arch-pc/lab08]
$ ./lab08-3 1 2 3 4 5 6
Результат: 720
```

Рис. 4.15: Работа файла

## 5 Выполнение заданий для самостоятельной работы

1) Создаю файл lab8-4.asm

```
(rustam01@kali) - [~/work/arch-pc/lab08]
$ touch lab08-4.asm
(rustam01@kali) - [~/work/arch-pc/lab08]
$ mcedit lab08-4.asm
```

Рис. 5.1: Создание файла

2) Создание и работа файла

```
(rustam01@kali) - [~/work/arch-pc/lab08]
$ nasm -f elf lab08-4.asm
(rustam01@kali) - [~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab08-4 lab08-4.o
(rustam01@kali) - [~/work/arch-pc/lab08]
$ ./lab08-4 1 2 3 4 5 6
Функция:  $f(x) = 2x + 15$ 
Результат: 132
(rustam01@kali) - [~/work/arch-pc/lab08]
$
```

Рис. 5.2: Работа файла

текст файла:

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0 msg1 db "Функция:  $f(x) =$ 
2x + 15",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 mov ebx, 2 mul
ebx add eax, 15 add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax` loop next ; переход к обработке
следующего аргумента
_end:
mov eax, msg1 call sprintf mov eax, msg ; вывод сообщения
"Результат: " call sprintf mov eax, esi ; записываем сумму в
регистр `eax` call sprintf ; печать результата call quit ;
завершение программы

```

## 6 Выводы

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

## Список литературы