

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 $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x_i передаются как аргументы. 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 $x = x_1, x_2, \dots, x_n$.

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).

4 Выполнение лабораторной работы

Я создаю каталог `~/work/arch-pc/lab08` с помощью команды `mkdir` и создаю файл `lab8-1-1.asm` с помощью `touch` (рис. fig:001).

```
File Actions Edit View Help
(kali@mkantoz)-[~]
$ cd work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh--pc/labs/lab08
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ touch lab8-1-1.asm
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$
```

Создание каталога *lab08* и файла *lab8-1-1.asm*

```
%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
```

Теперь я заполняю файл *lab8-1-1.asm* кодом из листинга 8.1 (рис. fig:003).

```
mc [kali@mkantoz]:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08 (on mkantoz)
File Actions Edit View Help
GNU nano 7.2 /home/kali/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08/lab8-1-1.asm *
#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 sprintf
; ----- 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 "ecx=ecx-1"
mov ecx,[N] ; Счетчик цикла, "ecx=N"
label:
sub ecx,1
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
; ----- Конец программы
```

Заполнение файла lab8-1-1.asm

Я создаю исполняемый файл и запускаю его. (рис. fig:004)

```
(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ nasm -f elf lab8-1-1.asm

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ./lab8-1-1

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ld -m elf_i386 -o lab8-1-1 lab8-1-1.o

Введите N: 56
```

Запуск файла lab8-1-1

Вижу, что программа выводит все числа по убыванию от введенного пользователем числа до единицы. Значит, программа совершает именно то количество циклов, соответствующее введенному с клавиатуры числу.

Я должен изменить программу согласно указанию в материале по лабораторной работе, добавив строку “sub ecx,1” в секции label. (рис. fig:005)


```
mc [kali@mkantoz]~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08 (on mkantoz)
File Actions Edit View Help
GNU nano 7.2 /home/kali/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08/lab8-1-1.asm *
; ----- Ввод 'N' -----
mov eax, msg1
call sprint
; ----- Ввод 'N' -----
mov ecx, N
mov edx, 10
call streadd
; ----- 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
mov [N], ecx
call iprintfLF ; Вывод значения 'N'
pop ecx
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Вторичное редактирование файла lab8-1-1.asm

Снова создаю исполняемый файл и запускаю его.(рис. fig:008)

```
(kali@mkantoz)~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08
$ nasm -f elf lab8-1-1.asm

(kali@mkantoz)~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08
$ ld -m elf_i386 -o lab8-1-1 lab8-1-1.o

(kali@mkantoz)~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08
$ ./lab8-1-1
Введите N: 5
4
3
2
1
0
```

Еще один запуск файла lab8-1-1

Теперь я вижу, что программа выдает числа из отрезка $[0, 4]$, то есть получается ровно 5 чисел, а значит программа производит то число циклов, которое было введено пользователем.

Далее я создаю файл lab8-2-2.asm.(рис. fig:009)

Создание lab8-2-2.asm

Заполняю данный файл кодом из листинга 8.2 (рис. fig:010)

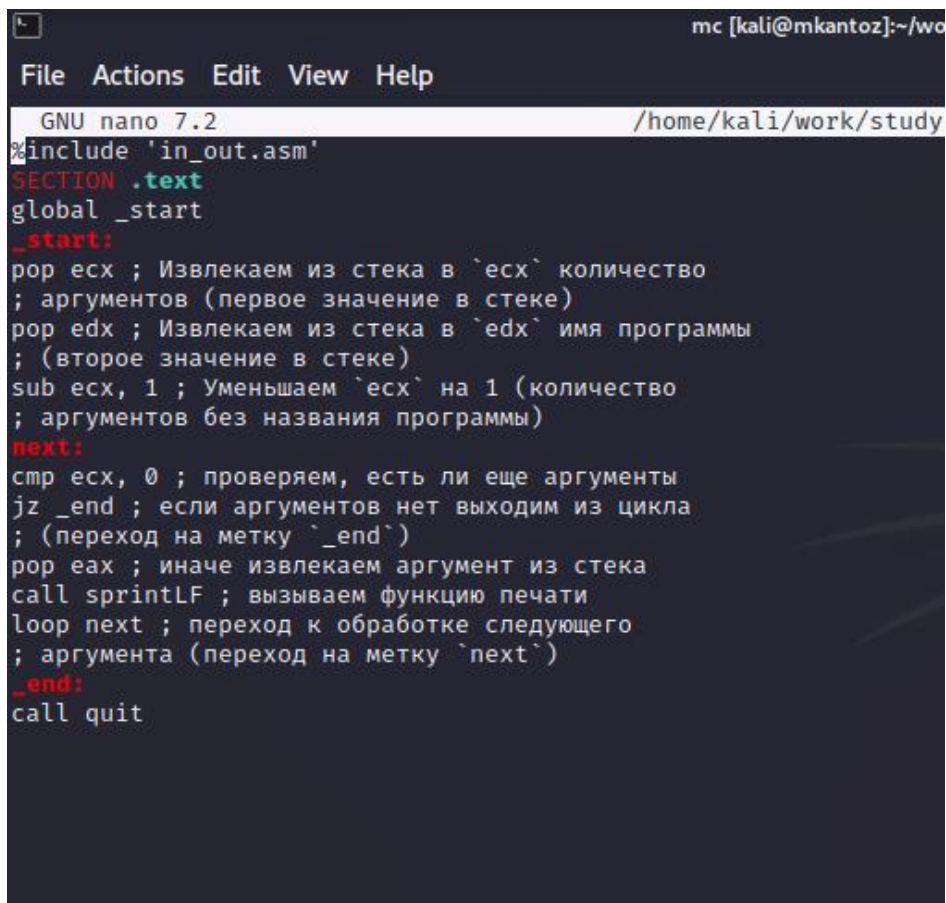
```
%include    'in_out.asm'

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
```



```
mc [kali@mkantoz]:~/wo
File Actions Edit View Help
GNU nano 7.2 /home/kali/work/study
%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
```

Создание lab8-2-2.asm

Теперь оттранслирую исходный файл в объектный и запущу его. (рис. fig:011)

```
(kali@mkantoz)-[~/.../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]  
$ nasm -f elf lab8-2-2.asm
```

```
(kali@mkantoz)-[~/.../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]  
$ ld -m elf_i386 -o lab8-2-2 lab8-2-2.o
```

```
4  
5  
6
```

Запуск программы lab8-2-2

Введя 3 аргумента согласно схеме в лабораторной работе, я вижу, что программа вывела все введенные мною аргументы. Значит, программа обработала все аргументы. Теперь я создаю новый файл lab8-3.asm (рис. fig:012)

```
(kali@mkantoz)-[~/.../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]  
$ nasm -f elf lab8-3.asm
```

```
(kali@mkantoz)-[~/.../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]  
$ ld -m elf_i386 -o lab8-3 lab8-3.o
```

Создание файла lab8-3.asm

Ввожу данный мне код данной мне программы. (рис. fig:013)

```
%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

```

Ввод кода программы

После трансляции запускаю программу смотрю на результат. (рис. fig:014)

```

(kali@mkantoz)~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ nasm -f elf lab8-3.asm

(kali@mkantoz)~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

./main 12 13 7 10 5
Результат: 47

```


Ввод кода программы

Вижу, что программа выводит сумму всех введенных аргументов. Теперь я меняю код программы, согласно указаниям, чтобы программа перемножала введенные аргументы. (рис. fig:015)

```
mc [kali@mkantoz]:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08 (on mkantoz)
File Actions Edit View Help
GNU nano 7.2 /home/kali/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08/lab8-3.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
```

Редактирования кода программы для умножения аргументов

Теперь запускаю программу, чтобы проверить ее действие. (рис. fig:016)

```
(kali@mkantoz)-[~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ nasm -f elf lab8-3.asm

(kali@mkantoz)-[~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

./lab8-3 1 2 3

Результат: 6
```

Запуск отредактированного файла lab8-3

Вижу, что программа работает исправно.

Исправленный код:

```
%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 произведений
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, eax ; перемещаем значение из eax в ebx
mov eax, esi ; перемещаем значение из esi в eax, чтобы результат записался
при следующей операции записался в eax.
mul ebx ; eax = eax*ebx
mov esi, eax ; перемещаем обратно в esi
\
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

```

5 Самостоятельная работа

Создаю файл lab8-4.asm. (рис. fig:017)



```

(kali@mkantoz) - [~/.../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ touch lab8-4.asm

```

Создание файла lab8-4.asm

В данной работе мне нужно реализовать функцию под номером 12(согласно моему варианту в прошлой лабораторной работе), то есть $f(x) = 15 \cdot x - 9$. Также, если при вводе дано несколько аргументов, программа должна вычислить сумму соответствующих им значений функции. Я заполняю файл lab8-4.asm. соответствующим кодом. (рис. fig:018)

```
mc [kali@mkantoz]:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08 (on mkantoz)
File Actions Edit View Help
GNU nano 7.2 /home/kali/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08/lab8-4.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
```

Написание кода для программы

Теперь я создаю исполняемый файл и проверяю работу своего кода. (рис. fig:019)

```
(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ touch lab8-4.asm

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ nasm -f elf lab8-4.asm

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ld -m elf_i386 -o lab8-4 lab8-4.o

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ./lab8-4 1 2
Результат: 27

(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab08]
$ ./lab8-4 2 3
Результат: 57
```

Проверка работы программы

Программа выдает верные значения.

Код программы:

```
%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
mov ebx, 15 ; перемещаем значение из esi в eax, чтобы результат записался
; при следующей операции записался в eax.
mul ebx ; eax = 15*eax
sub eax, 9 ; eax= 15*eax-9
add esi, eax ; 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 ; завершение программы

```

6 Выводы

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

7 Список литературы

Лабораторная работа №8

