

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Федорова Анжелика Игоревна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Самостоятельная работа	18
6	Выводы	21
7	Список литературы	22

Список иллюстраций

4.1	Создание каталога lab08 и файла lab8-1.asm	8
4.2	Копирование файла in_out.asm	8
4.3	Заполнение файла lab8-1.asm	9
4.4	Запуск файла lab8-1	9
4.5	Коррекция кода в файла lab8-1.asm	10
4.6	Вторичный запуск файла lab8-1	10
4.7	Вторичное редактирование файла lab8-1.asm	11
4.8	Еще один запуск файла lab8-1	11
4.9	Создание lab8-2.asm	12
4.10	Создание lab8-2.asm	12
4.11	Запуск программы lab8-2	13
4.12	Создание файла lab8-3.asm	13
4.13	Ввод кода программы	14
4.14	Ввод кода программы	14
4.15	Редактирования кода программы для умножения аргументов . .	15
4.16	Запуск отредактированного файла lab8-3	15
5.1	Создание файла lab8-4.asm	18
5.2	Написание кода для программы	19
5.3	Проверка работы программы	19

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 $x = x_1, x_2, \dots, x_n$, 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

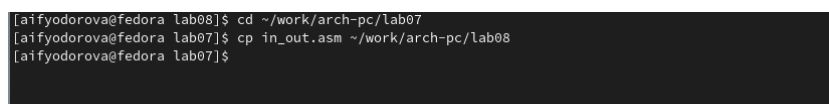
Я создаю каталог `~/work/arch-pc/lab08` с помощью команды `mkdir` и создаю файл `lab8-1.asm` с помощью `touch` (рис. fig:001).



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

Рис. 4.1: Создание каталога `lab08` и файла `lab8-1.asm`

Также я должна скопировать файл `in_out.asm` в данную директорию, чтобы в дальнейшем подключить ее. (рис. fig:002)



```
[aifyodorova@fedora lab08]$ cd ~/work/arch-pc/lab07
[aifyodorova@fedora lab07]$ cp in_out.asm ~/work/arch-pc/lab08
[aifyodorova@fedora lab07]$
```

Рис. 4.2: Копирование файла `in_out.asm`

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


```

lab8-1.asm [-M--] 11 L: 1*25 26/ 30] +(498 / 638b) 0010 0x00A [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 iprintf ; Вывод значения 'N'
loop label ; ecx=ecx-1 и если ecx не '0'
; переход на label
call quit

```

Рис. 4.3: Заполнение файла lab8-1.asm

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

```

[aiFYodorova@fedora lab08]$ nasm -f elf lab8-1.asm
[aiFYodorova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[aiFYodorova@fedora lab08]$ ./lab8-1
Введите N: 56
56
55
54
53
52
51
50
49
48
47
46
45
44
43
42
41
40
39
38
37

```

Рис. 4.4: Запуск файла lab8-1

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

```

lab8-1.asm  [-М--]  9 L:  1+24  25/ 31]  +(484 / 648b) 0010 0x00A
#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
mov [N],ecx
mov eax,[N]
call sprintf ; Вывод значения 'N'
loop label ; ecx=ecx-1 и если 'ecx' не '0'
; Переход на 'label'
call quit

```

Рис. 4.5: Коррекция кода в файла lab8-1.asm

Теперь снова транслирую .asm файл в объектный файл и запускаю программу.(рис. fig:006)

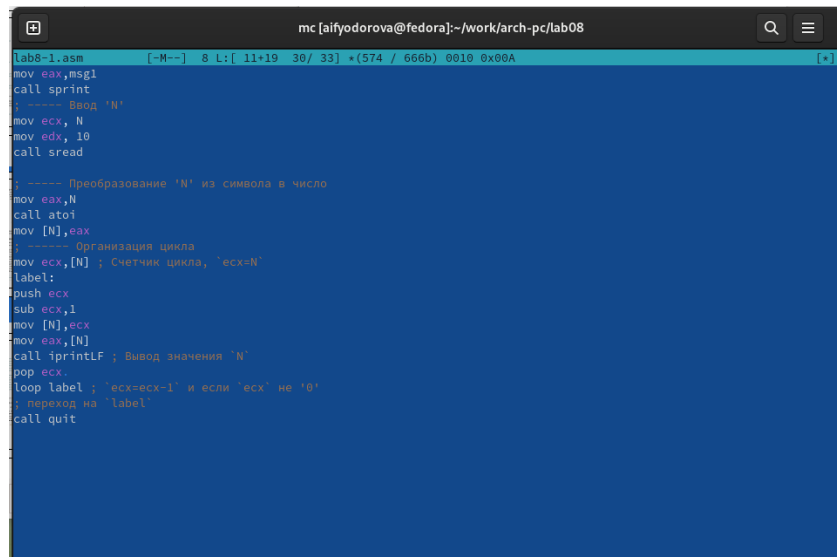
```

[aifedorov@fedora lab8]$ nasm -f elf lab8-1.asm
[aifedorov@fedora lab8]$ ld -o lab8-1 lab8-1.o
[aifedorov@fedora lab8]$ ./lab8-1
Введите N: 10
7
5
3
1
[aifedorov@fedora lab8]$

```

Рис. 4.6: Вторичный запуск файла lab8-1

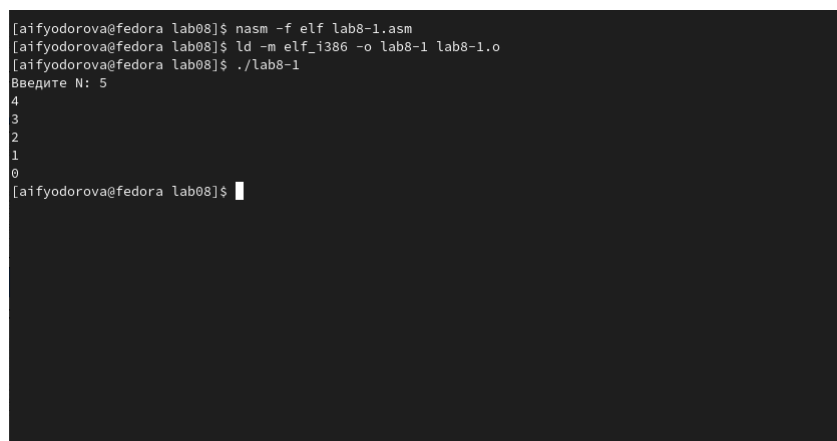
В данном случае видно, что регистр ecx принимает только положительные нечетные значения и на выходе пользователь получает только их. Значит, число всех циклов в программе не соответствует числу, введенному с клавиатуры. Теперь я снова редактирую код по имеющимся в материалах указаниям, добавив строку “push ecx” в секции label. (рис. fig:007)



```
lab8-1.asm [-M--] 8 L: [ 11:19 30/ 33] *(574 / 666b) 0010 0x00A [*]  
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  
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. 4.7: Вторичное редактирование файла lab8-1.asm

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



```
[aifyodorova@fedora lab08]$ nasm -f elf lab8-1.asm  
[aifyodorova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o  
[aifyodorova@fedora lab08]$ ./lab8-1  
Введите N: 5  
4  
3  
2  
1  
0  
[aifyodorova@fedora lab08]$
```

Рис. 4.8: Еще один запуск файла lab8-1

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.

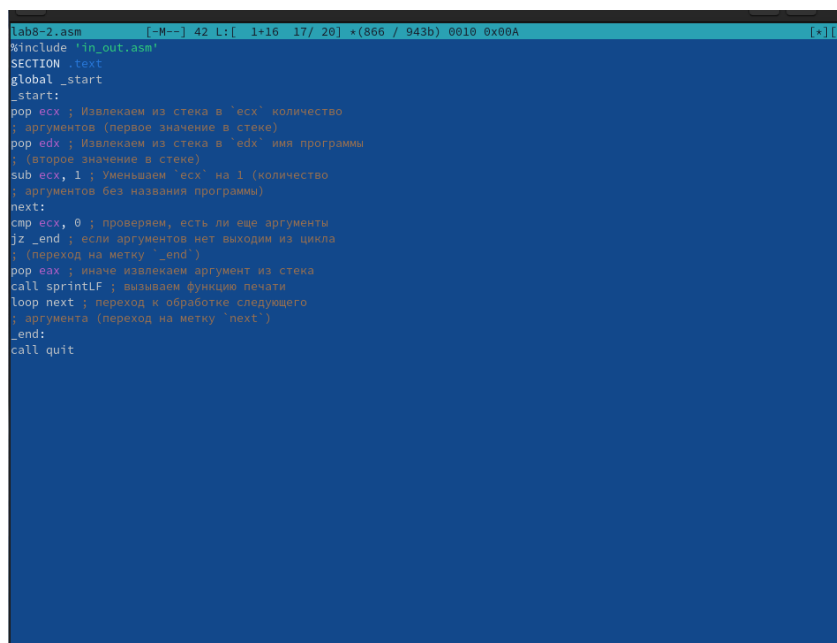
Далее я создаю файл lab8-2.asm.(рис. fig:009)

A terminal window with a dark background. The title bar shows the user 'aifyodorova@fedora' and the directory '~/work/arch-pc/lab08'. The command history shows the user navigating to the directory and creating a file named 'lab8-2.asm' using the 'touch' command.

```
aifyodorova@fedora: ~/work/arch-pc/lab08
[aifyodorova@fedora ~]$ cd ~/work/arch-pc/lab08
[aifyodorova@fedora lab08]$ touch lab8-2.asm
[aifyodorova@fedora lab08]$
```

Рис. 4.9: Создание lab8-2.asm

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

An assembly code editor window with a blue background. The title bar shows the filename 'lab8-2.asm' and some statistics. The code includes a macro call, a section declaration, and assembly instructions with comments in Russian.

```
lab8-2.asm [-M--] 42 L: [ 1+16 17/ 20] *(866 / 943b) 0010 0x00A [*][X]
#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. 4.10: Создание lab8-2.asm

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


```

#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. 4.13: Ввод кода программы

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

```

[aifyodorova@fedora lab08]$ nasm -f elf lab8-3.asm
[aifyodorova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[aifyodorova@fedora lab08]$ ./main 12 13 7 10 5
bash: ./main: Нет такого файла или каталога
[aifyodorova@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[aifyodorova@fedora lab08]$

```

Рис. 4.14: 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)

```

lab8-3.asm [----] 32 L: [ 1+15 16/ 32] *(680 /1678b) 1077 0x435 [x]
#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
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

```

Рис. 4.15: Редактирования кода программы для умножения аргументов

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

```

[aiFYodorova@fedora lab08]$ nasm -f elf lab8-3.asm
[aiFYodorova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[aiFYodorova@fedora lab08]$ ./lab8-3 1 2 3
Результат: 6
[aiFYodorova@fedora lab08]$ mc
[aiFYodorova@fedora lab08]$ ./lab8-3 4 5
Результат: 20
[aiFYodorova@fedora lab08]$ touch lab8-4.asm
[aiFYodorova@fedora lab08]$

```

Рис. 4.16: 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 сл
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)



```
[aifyodorova@fedora report]$ touch lab8-4.asm  
[aifyodorova@fedora report]$
```

Рис. 5.1: 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)

```

lab8-4.asm [-M--] 26 L: [ 1+22 23/ 31] * (1198/1576b) 0010 0x00A [*] [X]
#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 ebx, 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 iprintf ; печать результата
call quit ; завершение программы

```

Рис. 5.2: Написание кода для программы

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

```

[aiifyodorova@fedora lab08]$ nasm -f elf lab8-4.asm
[aiifyodorova@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[aiifyodorova@fedora lab08]$ ./lab8-4 1 2
Результат: 27
[aiifyodorova@fedora lab08]$ ./lab8-4 2 3
Результат: 57

```

Рис. 5.3: 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 ebx, чтобы результат записался при сле
mul ebx ; ebx = 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