

Отчет по лабораторной работе №8

Архитектура компьютера

Николенко Анна Николаевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	17
6	Листинги	18

Список иллюстраций

4.1	Создание каталога, переход в этот каталог и создание файла . . .	9
4.2	Редактирование файла	10
4.3	Создание исполняемого файла и его запуск	10
4.4	Редактирование файла	11
4.5	Создание исполняемого файла и его запуск	11
4.6	Редактирование файла	12
4.7	Создание исполняемого файла и его запуск	12
4.8	Создание файла	12
4.9	Редактирование файла	13
4.10	Создание исполняемого файла и его запуск	13
4.11	Создание файла	13
4.12	Редактирование файла	14
4.13	Создание исполняемого файла и его запуск	14
4.14	Редактирование файла	15
4.15	Создание исполняемого файла и его запуск	15
4.16	Программа суммы значений функции	16
4.17	Создание исполняемого файла и его запуск	16

Список таблиц

1 Цель работы

Цель работы заключается в приобретении навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Проверка работы программы вывода значений регистра ехх с добавлением различных изменений
2. Проверка работы программы, выводящая на экран аргументы командной строки
3. Проверка работы программы вычисления суммы и произведения аргументов командной строки
4. Задания по самостоятельной работе

3 Теоретическое введение

##Организация стека
Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO. Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основная функция стека - сохранение 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)) и дно (противоположный конец стека). 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).

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.

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 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.

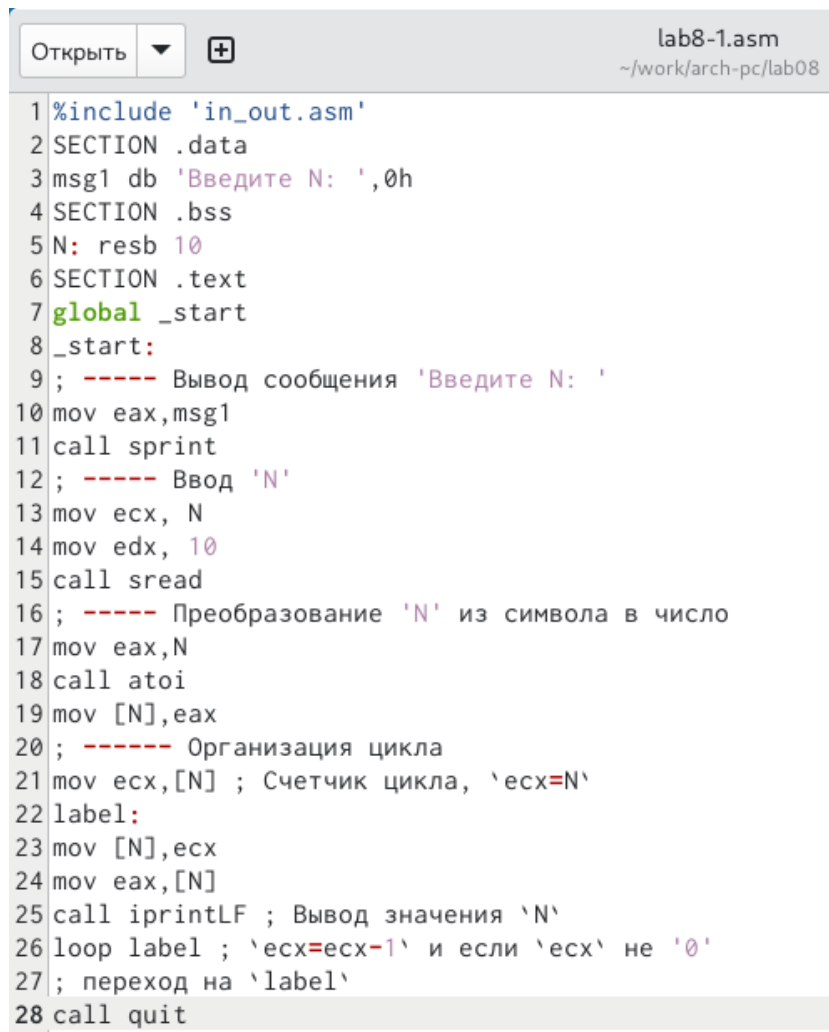
4 Выполнение лабораторной работы

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. [4.1]).

```
annikolenko@dk8n52 ~ $ mkdir ~/work/arch-pc/lab08  
annikolenko@dk8n52 ~ $ cd ~/work/arch-pc/lab08  
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 4.1: Создание каталога, переход в этот каталог и создание файла

Ввожу программу вывода значений регистра ехх в файл lab8-1.asm (рис. [4.2]).

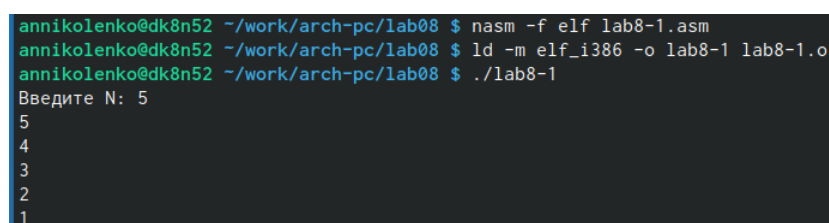


```
lab8-1.asm
~/work/arch-pc/lab08

1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; ----- Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
22 label:
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения 'N'
26 loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
27 ; переход на 'label'
28 call quit
```

Рис. 4.2: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. [4.3]).



```
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
```

Рис. 4.3: Создание исполняемого файла и его запуск

Изменяю текст программы в файле lab8-1.asm, добавив изменение значение

регистра ecx в цикле (рис. [4.4]).

```
22 label:
23 sub ecx,1 ; 'ecx=ecx-1'
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
28 call quit
```

Рис. 4.4: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. [4.5]).

```
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 4.5: Создание исполняемого файла и его запуск

Регистр ecx в цикле принимает нечетные значения, меньшие 10. Число проходов цикла не соответствует значению ☒ введенному с клавиатуры. Получаю результат отличный от ожидаемого: получаю N/2 значений

Вношу изменения в файл lab8-1.asm, добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop (рис. [4.6]).

```

22 label:
23 push ecx
24 sub ecx,1 ; 'ecx=ecx-1'
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx
29
30 loop label
31 call quit

```

Рис. 4.6: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. [4.7]).

```

annikolenko@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0

```

Рис. 4.7: Создание исполняемого файла и его запуск

Теперь ввелось N значений.

Создаю файл lab8-2.asm (рис. [4.8]).

```

annikolenko@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-2.asm

```

Рис. 4.8: Создание файла

Ввожу программу, выводящая на экран аргументы командной строки в файл lab8-2.asm (рис. [4.9]).

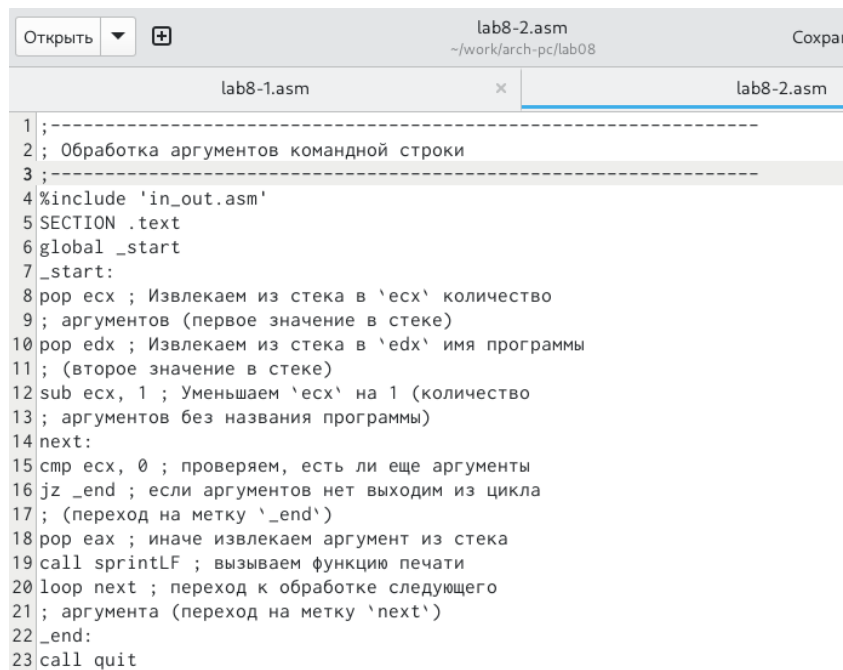


Рис. 4.9: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. [4.10]).

```

annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-2 arg1 arg2 arg3
arg1
arg2
arg3

```

Рис. 4.10: Создание исполняемого файла и его запуск

Все аргументы были обработаны программой.

Создаю файл lab8-3.asm (рис. [4.11]).

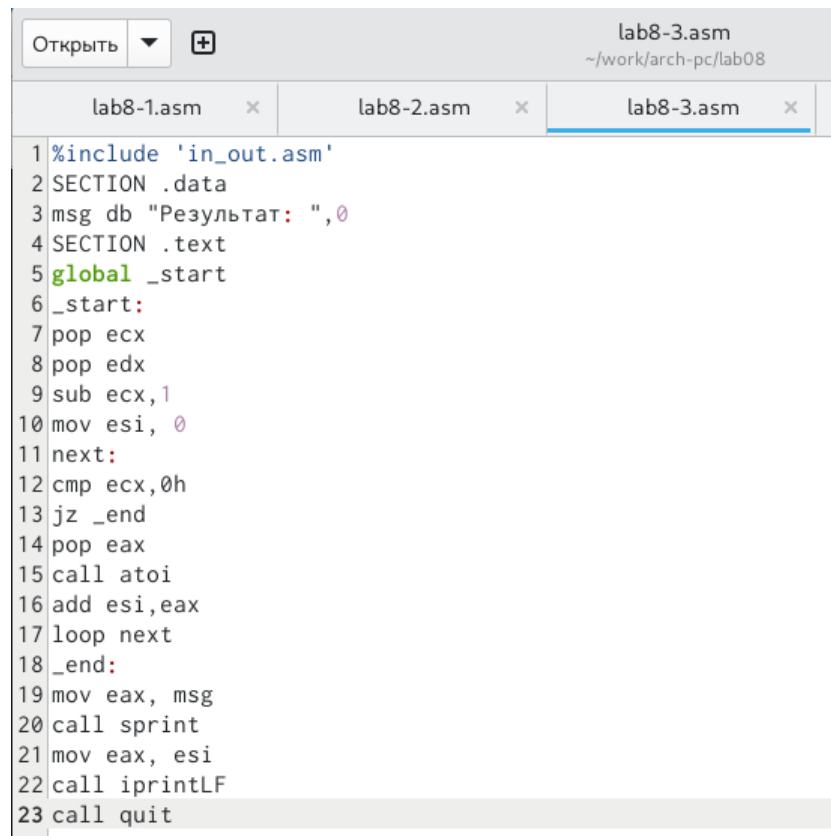
```

annikolenko@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-3.asm

```

Рис. 4.11: Создание файла

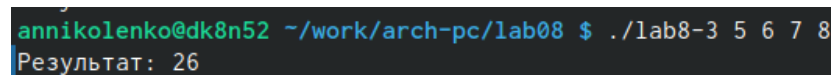
Ввожу программу вычисления суммы аргументов командной строки в файл lab8-3.asm (рис. [4.12]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 add esi,eax
17 loop next
18 _end:
19 mov eax, msg
20 call sprint
21 mov eax, esi
22 call iprintLF
23 call quit
```

Рис. 4.12: Редактирование файла

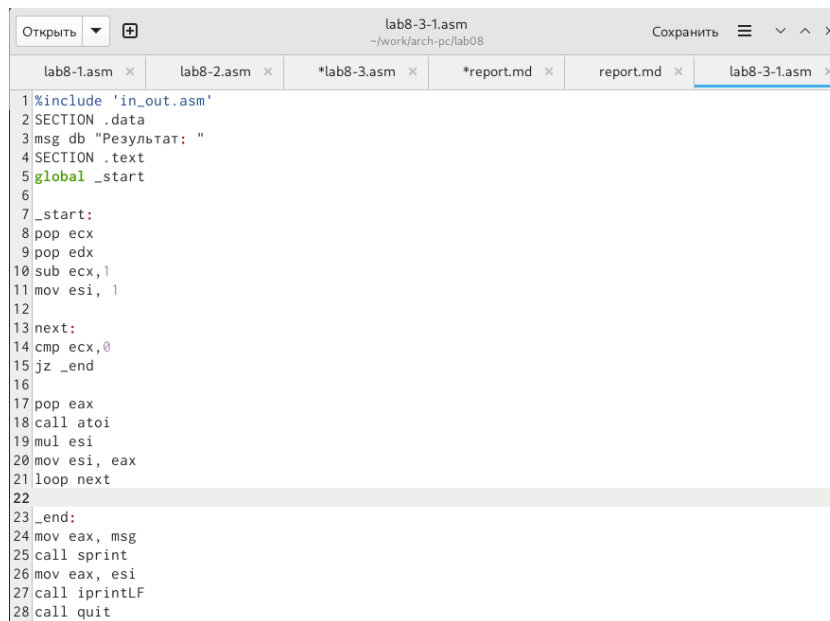
Создаю исполняемый файл и запускаю его (рис. [4.13]).



```
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-3 5 6 7 8
Результат: 26
```

Рис. 4.13: Создание исполняемого файла и его запуск

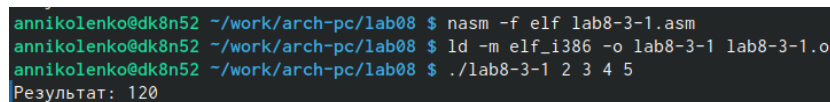
Вношу изменения в файл lab8-3.asm, чтобы она вычисляла произведение аргументов командной строки (рис. [4.14]).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: "
4 SECTION .text
5 global _start
6
7 _start:
8 pop ecx
9 pop edx
10 sub ecx,1
11 mov esi, 1
12
13 next:
14 cmp ecx,0
15 jz _end
16
17 pop eax
18 call atoi
19 mul esi
20 mov esi, eax
21 loop next
22
23 _end:
24 mov eax, msg
25 call sprint
26 mov eax, esi
27 call iprintLF
28 call quit
```

Рис. 4.14: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. [4.15]).



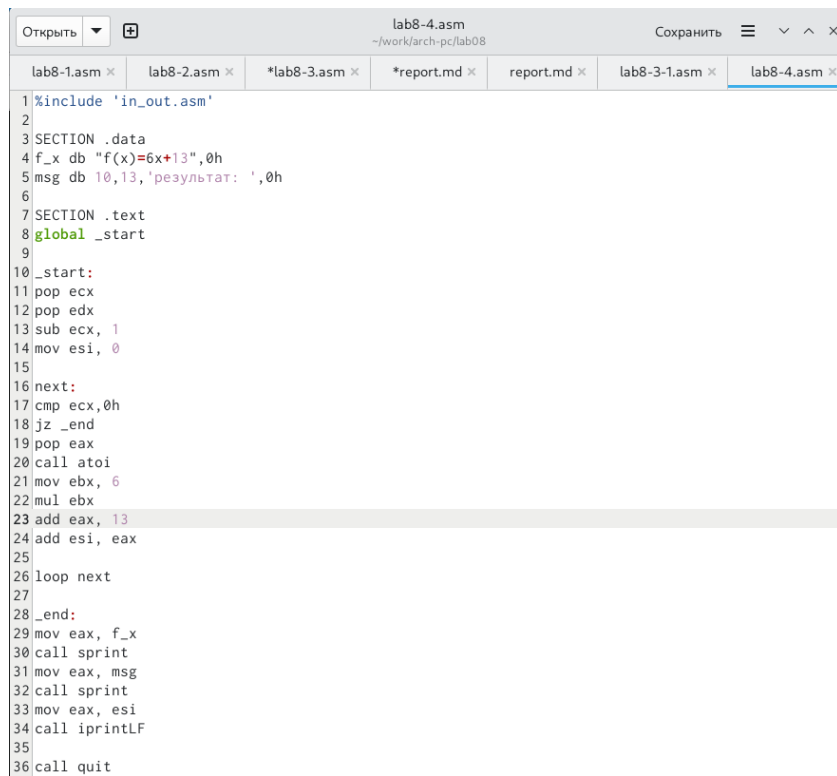
```
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3-1.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3-1 lab8-3-1.o
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-3-1 2 3 4 5
Результат: 120
```

Рис. 4.15: Создание исполняемого файла и его запуск

#Самостоятельная работа

Мой вариант: 15

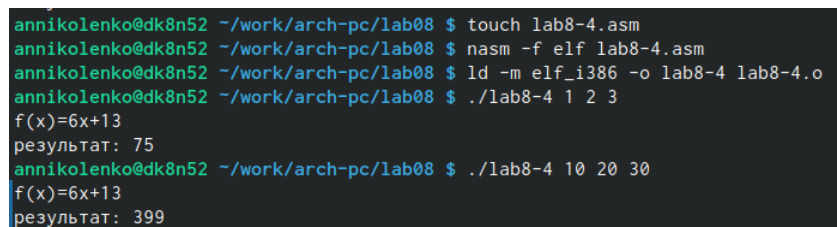
Пишу программу, которая находит сумму значений функции $f(x)$ (рис. [4.16]).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 f_x db "f(x)=6x+13",0h
5 msg db 10,13,'результат: ',0h
6
7 SECTION .text
8 global _start
9
10 _start:
11 pop ecx
12 pop edx
13 sub ecx, 1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 mov ebx, 6
22 mul ebx
23 add eax, 13
24 add esi, eax
25
26 loop next
27
28 _end:
29 mov eax, f_x
30 call sprint
31 mov eax, msg
32 call sprint
33 mov eax, esi
34 call iprintLF
35
36 call quit
```

Рис. 4.16: Программа суммы значений функции

Создаю исполняемый файл и запускаю его (рис. [4.17]).



```
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-4.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3
f(x)=6x+13
результат: 75
annikolenko@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-4 10 20 30
f(x)=6x+13
результат: 399
```

Рис. 4.17: Создание исполняемого файла и его запуск

5 Выводы

Были получены навыки по организации циклов и работе со стеком на языке NASM. <3

6 Листинги

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
f_x db "f(x)=6x+13",0h
```

```
msg db 10,13,'результат: ',0h
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx,0h
```

```
jz _end
```

```
pop eax
```

```
call atoi
```

```
mov ebx, 6
```

```
mul ebx
```

```
add eax, 13
add esi, eax

loop next

_end:
mov eax, f_x
call sprint
mov eax, msg
call sprint
mov eax, esi
call iprintLF

call quit
```