

Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютера

Осина Виктория Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	13
4.3	Выполнение задания для самостоятельной работы	16
4.3.1	Проверка	18
5	Выводы	20
6	Список литературы	21

Список иллюстраций

4.1	Создание каталога lab08 и файла lab8-1.asm в нем	9
4.2	Копирование файла in_out.asm	9
4.3	Проверка, что файл находится в нужном каталоге	10
4.4	Открытие файла в редакторе	10
4.5	Ввод текста программы в файл	10
4.6	Создание исполняемого файла и его запуск	11
4.7	Изменение текста программы	11
4.8	Создание исполняемого файла и его запуск	12
4.9	Изменение текста программы	12
4.10	Создание исполняемого файла и его запуск	13
4.11	Создание файла lab8-2.asm и его открытие в редакторе	13
4.12	Ввод текста программы в файл	13
4.13	Создание исполняемого файла и его запуск	14
4.14	Создание файла lab8-3.asm и его открытие в редакторе	14
4.15	Ввод текста программы в файл	14
4.16	Создание исполняемого файла и его запуск	15
4.17	Изменение текста программы	15
4.18	Создание исполняемого файла и его запуск	15
4.19	Создание файла var7.asm	16
4.20	Ввод текста программы в файл	17
4.21	Создание исполняемого файла и его запуск	17
4.22	Второй набор значений	18

Список таблиц

1 Цель работы

Целью данной лабораторной работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Выполнение задания для самостоятельной работы

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.

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.

Аналогично команде записи в стек существует команда pusha, 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.

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

4.1 Реализация циклов в NASM

Создаю каталог lab08 для программ лабораторной работы №8, перехожу в него и создаю файл lab8-1.asm, проверяю, что файл создан (рис. 4.1)

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

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

Перед работой с программами копирую файл in_out.asm в каталог и проверяю, что файл находится в нужном каталоге (рис. 4.2) (рис. 4.3)

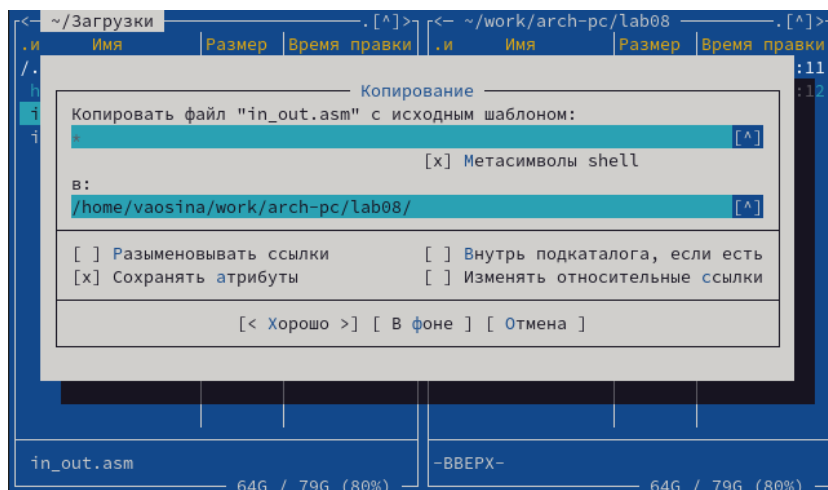


Рис. 4.2: Копирование файла in_out.asm

```

<- ~/work/arch-pc/lab08 _____.[^]>

```

.и	Имя	Размер	Время правки
/..		-ВВЕРХ-	ноя 28 12:11
	in_out.asm	3942	ноя 11 14:25
	lab8-1.asm	0	ноя 28 12:12

Рис. 4.3: Проверка, что файл находится в нужном каталоге

Открываю lab8-1.asm в редакторе и ввожу в него текст программы, которая выводит значение регистра ecx (рис. 4.4) и (рис. 4.5).

```
[vaosina@fedora lab08]$ gedit lab8-1.asm
```

Рис. 4.4: Открытие файла в редакторе

```

Открыть  lab8-1.asm
~/work/arch-pc/lab08
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1 db 'Введите N: ',0h
5
6 SECTION .bss
7 N: resb 10
8
9 SECTION .text
10 global _start
11 _start:
12 ; ----- Вывод сообщения 'Введите N: '
13 mov eax,msg1
14 call sprint
15 ; ----- Ввод 'N'
16 mov ecx, N
17 mov edx, 10
18 call sread
19 ; ----- Преобразование 'N' из символа в число
20 mov eax,N
21 call atoi
22 mov [N],eax
23 ; ----- Организация цикла
24 mov ecx,[N] ; Счетчик цикла, 'ecx=N'
25 label:
26 mov [N],ecx
27 mov eax,[N]
28 call iprintf ; Вывод значения 'N'
29 loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
30 ; переход на 'label'
31 call quit

```

Рис. 4.5: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его (рис. 4.6). В результате выводятся цифры от N до 1.

```

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

```

Рис. 4.6: Создание исполняемого файла и его запуск

Изменяю текст программы добавив значение регистра ecx в цикле (рис. 4.7)

```

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

```

Рис. 4.7: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.8). Теперь программа работает некорректно и цикл не прерывается. Число проходов цикла не соответствует значению N, введенному с клавиатуры.

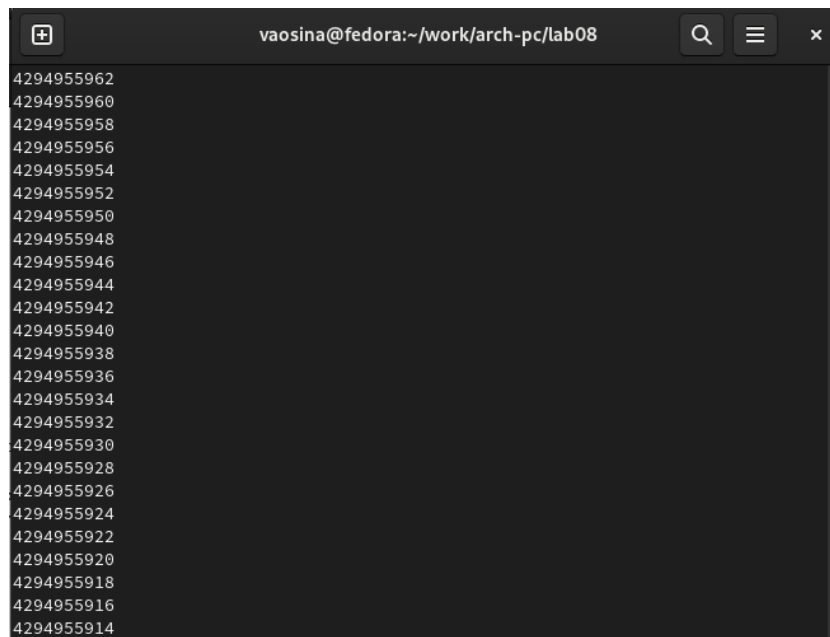


Рис. 4.8: Создание исполняемого файла и его запуск

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Поэтому вношу изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop`. (рис. 4.9).

```

25 label:
26 push ecx ; добавление значения ecx в стек
27 sub ecx,1
28 mov [N],ecx
29 mov eax,[N]
30 call iprintLF
31 pop ecx ; извлечение значения ecx из стека
32 loop label

```

Рис. 4.9: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.10) В результате выводят цифры от $N-1$ до 0 включительно. Число проходов цикла соответствует значению N , введенному с клавиатуры.

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

Рис. 4.10: Создание исполняемого файла и его запуск

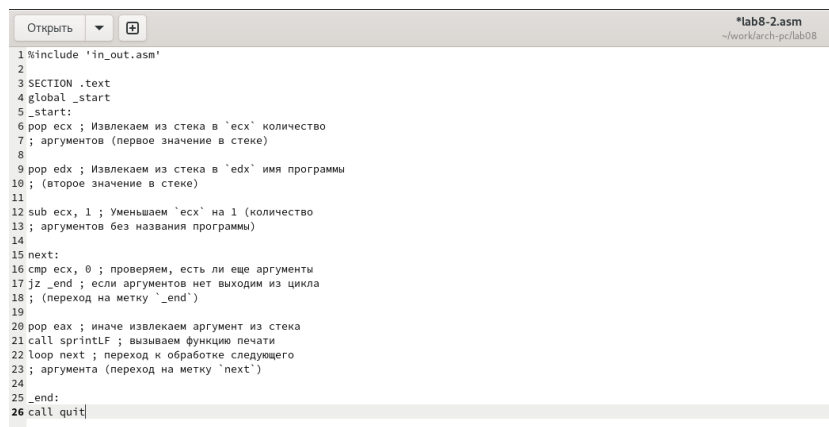
4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и открываю его в редакторе (рис. 4.11).

```
[vaosina@fedora lab08]$ touch lab8-2.asm
[vaosina@fedora lab08]$ gedit lab8-2.asm
```

Рис. 4.11: Создание файла lab8-2.asm и его открытие в редакторе

Ввожу в файл текст программы, которая выводит на экран аргументы командной строки (рис. 4.12).



```
Открыть  *lab8-2.asm
~/work/arch-pc/lab08

1 %include 'in_out.asm'
2
3 SECTION .text
4 global _start
5 _start:
6 pop ecx ; Извлекаем из стека в 'ecx' количество
7 ; аргументов (первое значение в стеке)
8
9 pop edx ; Извлекаем из стека в 'edx' имя программы
10 ; (второе значение в стеке)
11
12 sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество
13 ; аргументов без названия программы)
14
15 next:
16 cmp ecx, 0 ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку '_end')
19
20 pop eax ; иначе извлекаем аргумент из стека
21 call sprintf ; вызываем функцию печати
22 loop next ; переход к обработке следующего
23 ; аргумента (переход на метку 'next')
24
25 _end:
26 call quit
```

Рис. 4.12: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его.(рис. 4.13). Командой было обработано 4 аргумента, так как они разделены пробелом, поэтому, если аргумент

содержит пробел, его надо заключать в кавычки

```
[vaosina@fedora lab08]$ nasm -f elf lab8-2.asm
[vaosina@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[vaosina@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[vaosina@fedora lab08]$
```

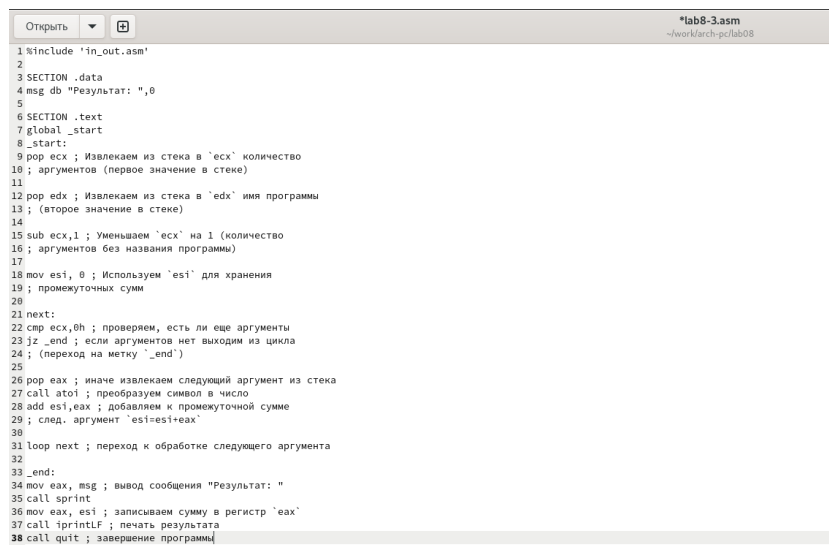
Рис. 4.13: Создание исполняемого файла и его запуск

Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и открываю его в редакторе (рис. 4.14).

```
[vaosina@fedora lab08]$ touch lab8-3.asm
[vaosina@fedora lab08]$ gedit lab8-3.asm
```

Рис. 4.14: Создание файла lab8-3.asm и его открытие в редакторе

Ввожу в файл текст программы, которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. 4.15)



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8 _start:
9 pop ecx ; Извлекаем из стека в 'ecx' количество
10 ; аргументов (первое значение в стеке)
11
12 pop edx ; Извлекаем из стека в 'edx' имя программы
13 ; (второе значение в стеке)
14
15 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
16 ; аргументов без названия программы)
17
18 mov esi, 0 ; Используем 'esi' для хранения
19 ; промежуточных сумм
20
21 next:
22 cmp ecx,0h ; проверяем, есть ли еще аргументы
23 jz _end ; если аргументов нет выходим из цикла
24 ; (переход на метку '_end')
25
26 pop eax ; иначе извлекаем следующий аргумент из стека
27 call atoi ; преобразуем символ в число
28 add esi,eax ; добавляем к промежуточной сумме
29 ; след. аргумент 'esi=esi+eax'
30
31 loop next ; переход к обработке следующего аргумента
32
33 _end:
34 mov eax, msg ; вывод сообщения "Результат: "
35 call sprint
36 mov eax, esi ; записываем сумму в регистр 'eax'
37 call iprintf ; печать результата
38 call quit ; завершение программы
```

Рис. 4.15: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его. (рис. 4.16) и (рис. 4.17) Программа работает корректно.

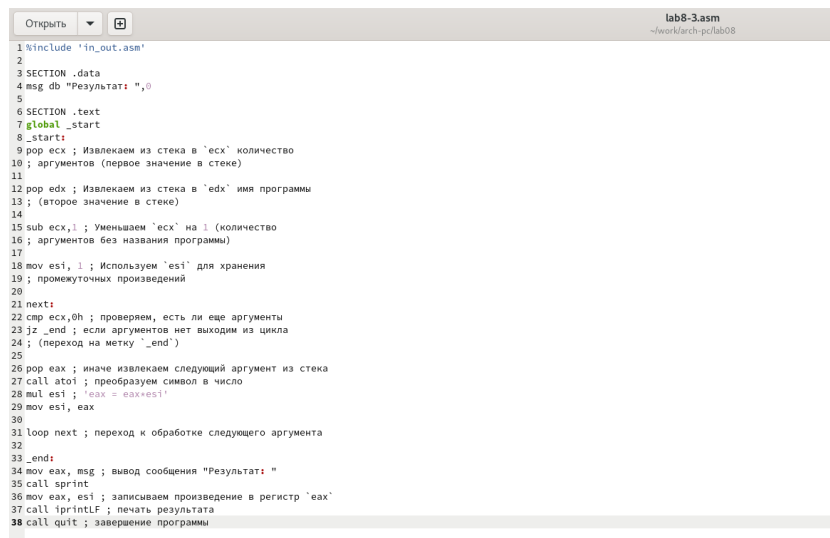
```

[vaosina@fedora lab08]$ nasm -f elf lab8-3.asm
[vaosina@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[vaosina@fedora lab08]$ ./lab8-3 5 5 4 6 1 1 1
Результат: 23
[vaosina@fedora lab08]$

```

Рис. 4.16: Создание исполняемого файла и его запуск

Изменяю текст программы так, чтобы вместо суммы чисел выводилось произведение аргументов командной строки



```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8 _start:
9 pop ecx ; Извлекаем из стека в 'ecx' количество
10 ; аргументов (первое значение в стеке)
11
12 pop edx ; Извлекаем из стека в 'edx' имя программы
13 ; (второе значение в стеке)
14
15 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
16 ; аргументов без названия программы)
17
18 mov esi, 1 ; Используем 'esi' для хранения
19 ; промежуточных произведений
20
21 next:
22 cmp ecx,0h ; проверяем, есть ли еще аргументы
23 jz _end ; если аргументов нет выходим из цикла
24 ; (переход на метку '_end')
25
26 pop eax ; иначе извлекаем следующий аргумент из стека
27 call atoi ; преобразуем символ в число
28 mul esi ; 'eax = eax*esi'
29 mov esi, eax
30
31 loop next ; переход к обработке следующего аргумента
32
33 _end:
34 mov eax, msg ; вывод сообщения "Результат: "
35 call sprintf
36 mov eax, esi ; записываем произведение в регистр 'eax'
37 call iprintf ; печать результата
38 call quit ; завершение программы

```

Рис. 4.17: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.18) Программа работает корректно.

```

[vaosina@fedora lab08]$ nasm -f elf lab8-3.asm
[vaosina@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[vaosina@fedora lab08]$ ./lab8-3 9 10 8
Результат: 720
[vaosina@fedora lab08]$

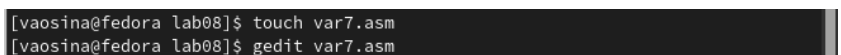
```

Рис. 4.18: Создание исполняемого файла и его запуск

4.3 Выполнение задания для самостоятельной работы

Необходимо написать 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. В соответствии с моим вариантом (вариант7), вид функции $3(x + 2)$.

Создаю файл `var7.asm` в каталоге `~/work/arch-pc/lab08` и открываю его в редакторе (рис. 4.19)



```
[vaosina@fedora lab08]$ touch var7.asm  
[vaosina@fedora lab08]$ gedit var7.asm
```

Рис. 4.19: Создание файла `var7.asm`

Ввожу в файл текст программы (рис. 4.20).


```

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

```

Рис. 4.20: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его. (рис. 4.21) и (рис. 4.22) Проверяю работу программы на нескольких наборах $x = x_1, x_2, \dots, x_n$. Программа работает корректно

```

[vaosina@fedora lab08]$ nasm -f elf var7.asm
[vaosina@fedora lab08]$ ld -m elf_i386 -o var7 var7.o
[vaosina@fedora lab08]$ ./var7 1 2 3
Результат: 36
[vaosina@fedora lab08]$

```

Рис. 4.21: Создание исполняемого файла и его запуск

```
[vaosina@fedora lab08]$ ./var7 8 3 6
Результат: 69
```

Рис. 4.22: Второй набор значений

4.3.1 Проверка

““ $3(x+2)$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$ $3(1+2) + 3(2+2) + 3(3+2)$ $33 + 34 + 3*5$ $9 + 12 + 15$ 36
 $3(x+2)$, $x_1 = 8$, $x_2 = 3$, $x_3 = 6$ $3(8+2) + 3(3+2) + 3(6+2)$ $310 + 35 + 3*8$ $30 + 15 + 24$ 69

```
### Текст программы
```%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:

pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi, 3

next:

cmp ecx,0h
jz _end
pop eax
call atoi
```

```
add eax, 2
mul edi
add esi,eax
loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

## **5 Выводы**

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

## **6 Список литературы**

### **1. Архитектура ЭВМ**