

Отчёт по лабораторной работе № 8

Дисциплина: архитектура компьютера

Кайнова Алина Андреевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	13
4.3	Выполнение заданий для самостоятельной работы	17
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Ввод текста программы	9
4.3	Создание и запуск исполняемого файла	9
4.4	Изменение текста программы	10
4.5	Создание и запуск исполняемого файла	11
4.6	Изменение текста программы	12
4.7	Создание и запуск исполняемого файла	13
4.8	Ввод текста программы	14
4.9	Создание и запуск исполняемого файла	14
4.10	Ввод текста программы	15
4.11	Создание и запуск исполняемого файла	15
4.12	Изменение текста программы	16
4.13	Создание и запуск исполняемого файла	16
4.14	Текст программы	18
4.15	Запуск исполняемого файла и проверка его работы	19

1 Цель работы

Научиться писать программы с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Выполнение заданий 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. 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. 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. 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. 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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для данной лабораторной работы и файл lab8-1.asm

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

Рис. 4.1: Создание каталога и файла

Ввожу в созданный файл текст программы из листинга 8.1


```

lab8-1.asm
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: Ввод текста программы

Создаю исполняемый файл и проверяю его работу

```

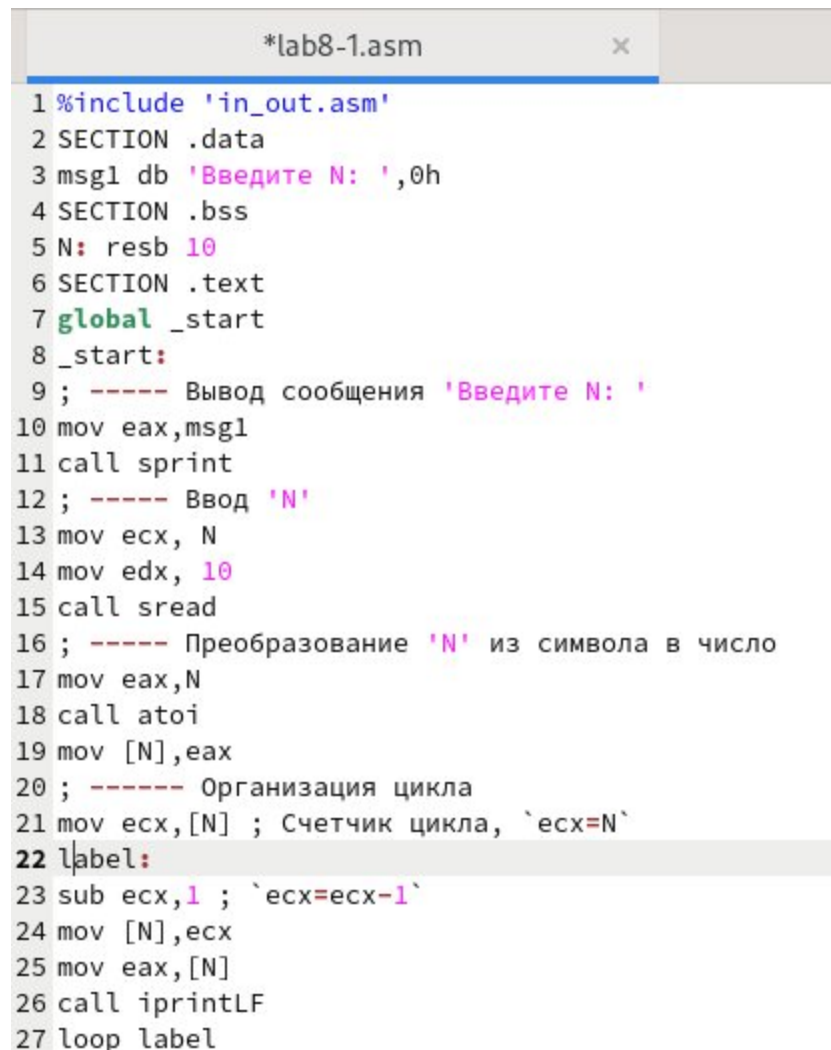
[aakayjnova@fedora lab08]$ nasm -f elf lab8-1.asm
[aakayjnova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[aakayjnova@fedora lab08]$ ./lab8-1
Введите N: 4
4
3
2
1

```

Рис. 4.3: Создание и запуск исполняемого файла

Программа выводит числа от N до 1 включительно.

Изменяю текст программы, изменив значение регистра ecx в цикле



```
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 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу

```
[aakayjnova@fedora lab08]$ nasm -f elf lab8-1.asm
[aakayjnova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[aakayjnova@fedora lab08]$ ./lab8-1
Введите N: 6
5
3
1
```

Рис. 4.5: Создание и запуск исполняемого файла

Регистр `ecx` принимает нечётные значения в цикле. Число проходов не соответствует введённому значению `N`.

Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счётчика цикла `loop`

```

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

```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу

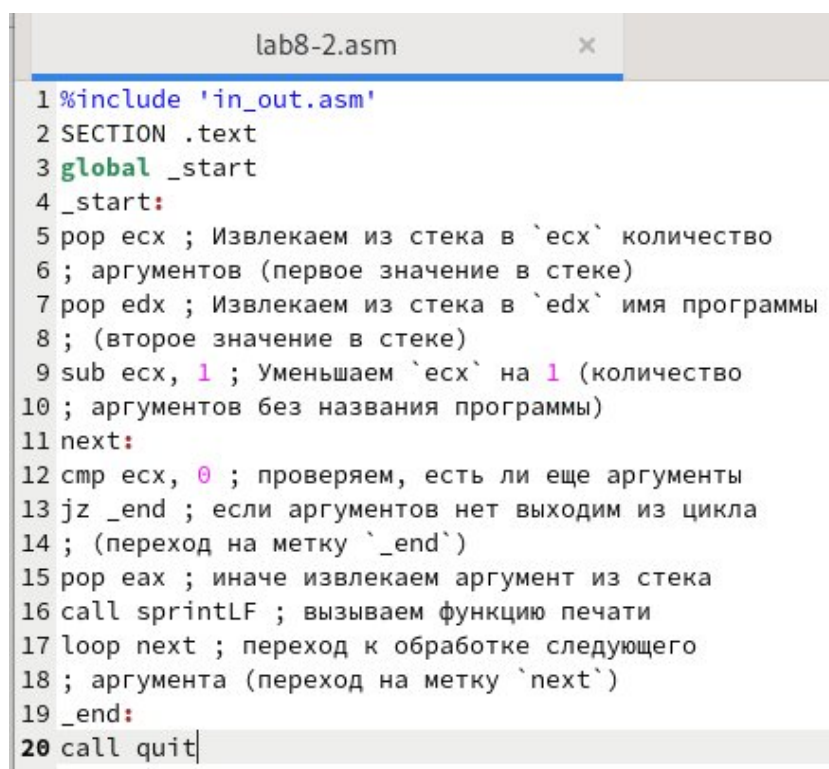
```
[aakayjnova@fedora lab08]$ nasm -f elf lab8-1.asm
[aakayjnova@fedora lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[aakayjnova@fedora lab08]$ ./lab8-1
Введите N: 6
5
4
3
2
1
0
```

Рис. 4.7: Создание и запуск исполняемого файла

В данном случае число проходов цикла соответствует введённому с клавиатуры значению и выводит числа от N-1 до 0 включительно.

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в данном каталоге и ввожу в него текст из программы листинга 8.2



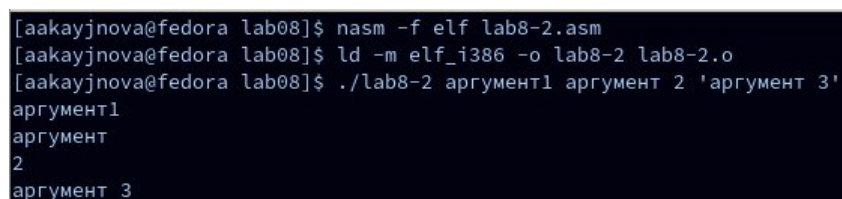
```

lab8-2.asm
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintLF ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit

```

Рис. 4.8: Ввод текста программы

Создаю исполняемый файл и проверяю его работу



```

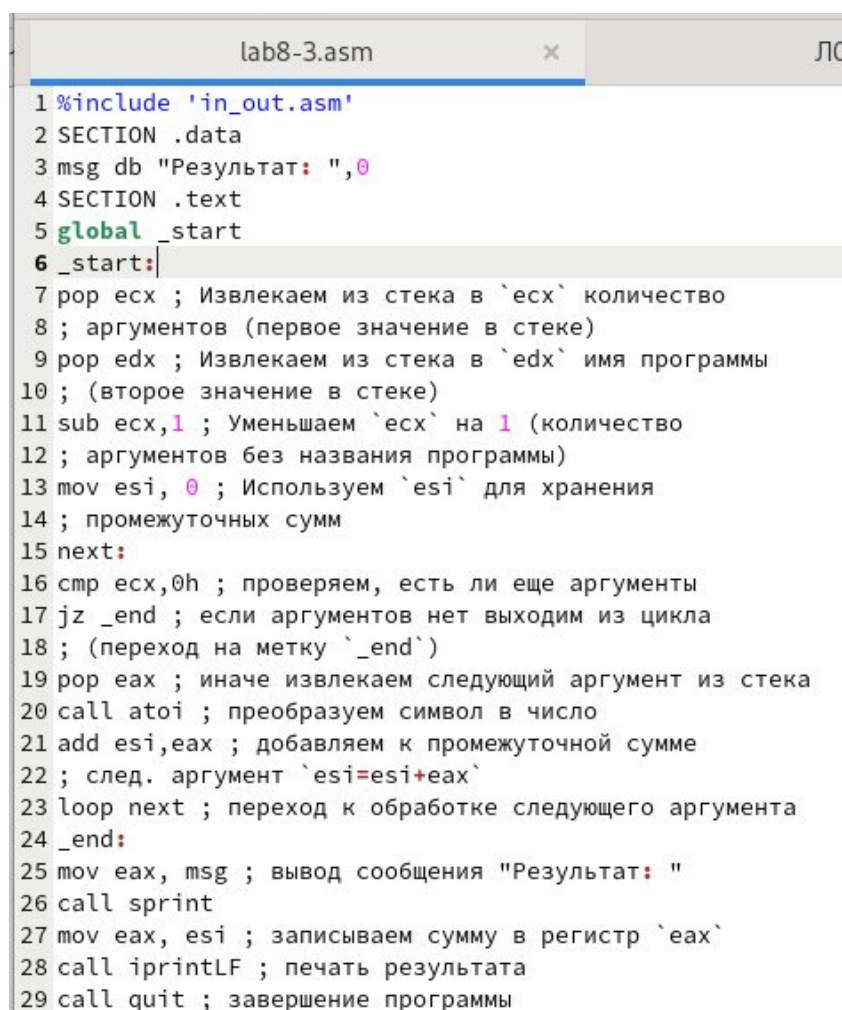
[aakayjnova@fedora lab08]$ nasm -f elf lab8-2.asm
[aakayjnova@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[aakayjnova@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 4.9: Создание и запуск исполняемого файла

Программа выводит 4 аргумента, так как аргумент 2 не взят в кавычки, поэтому из-за пробела программа считывает 2 как отдельный аргумент.

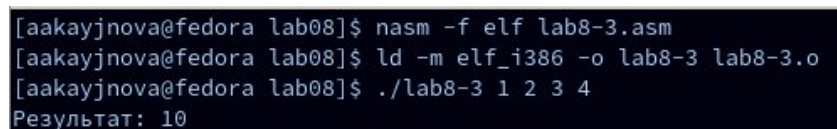
Создаю файл lab8-3.asm в данном каталоге и ввожу в него текст программы из листинга 8.3



```
lab8-3.asm x ЛС
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi,eax ; добавляем к промежуточной сумме
22 ; след. аргумент `esi=esi+eax`
23 loop next ; переход к обработке следующего аргумента
24 _end:
25 mov eax, msg ; вывод сообщения "Результат: "
26 call sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

Рис. 4.10: Ввод текста программы

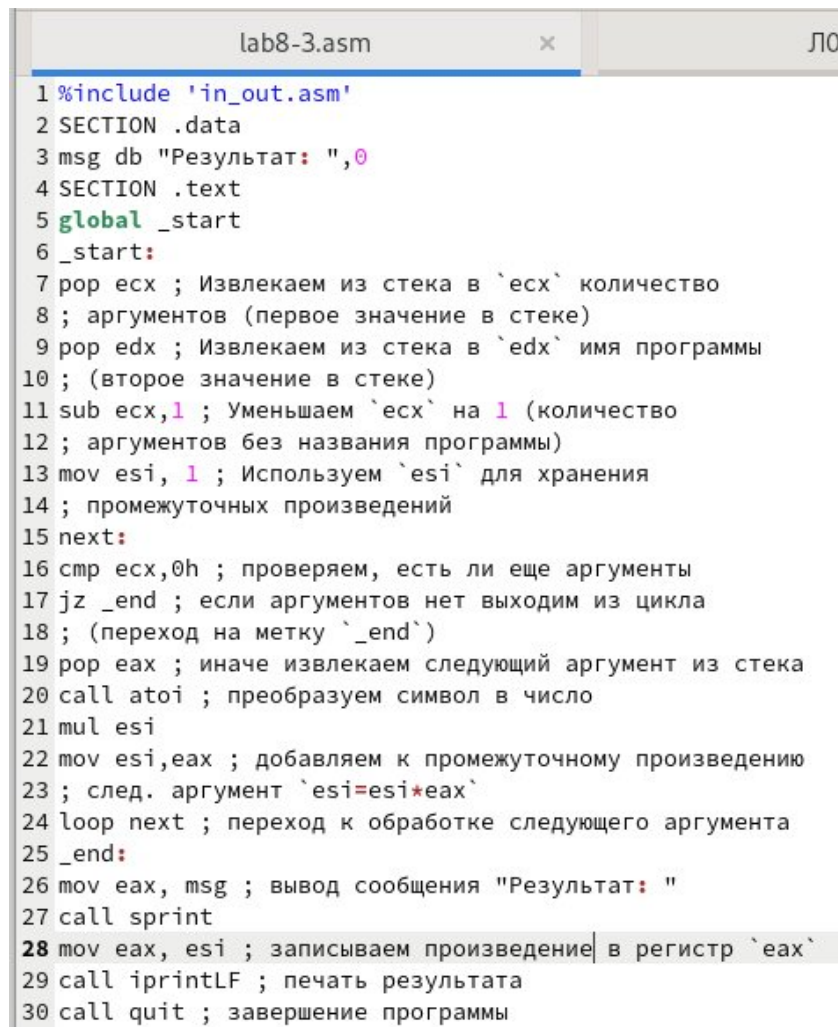
Создаю исполняемый файл и запускаю его, указав аргументы



```
[aakaynova@fedora lab08]$ nasm -f elf lab8-3.asm
[aakaynova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[aakaynova@fedora lab08]$ ./lab8-3 1 2 3 4
Результат: 10
```

Рис. 4.11: Создание и запуск исполняемого файла

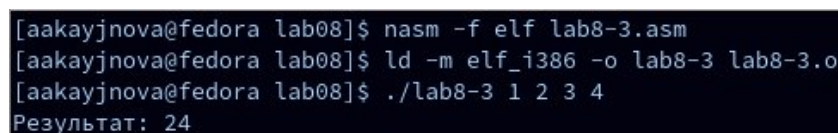
Изменяю текст программы для вычисления произведения аргументов команд-
ной строки



```
lab8-3.asm x ЛО
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в `ecx` количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 1 ; Используем `esi` для хранения
14 ; промежуточных произведений
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi
22 mov esi,eax ; добавляем к промежуточному произведению
23 ; след. аргумент `esi=esi*eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем произведение в регистр `eax`
29 call iprintLF ; печать результата
30 call quit ; завершение программы
```

Рис. 4.12: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы

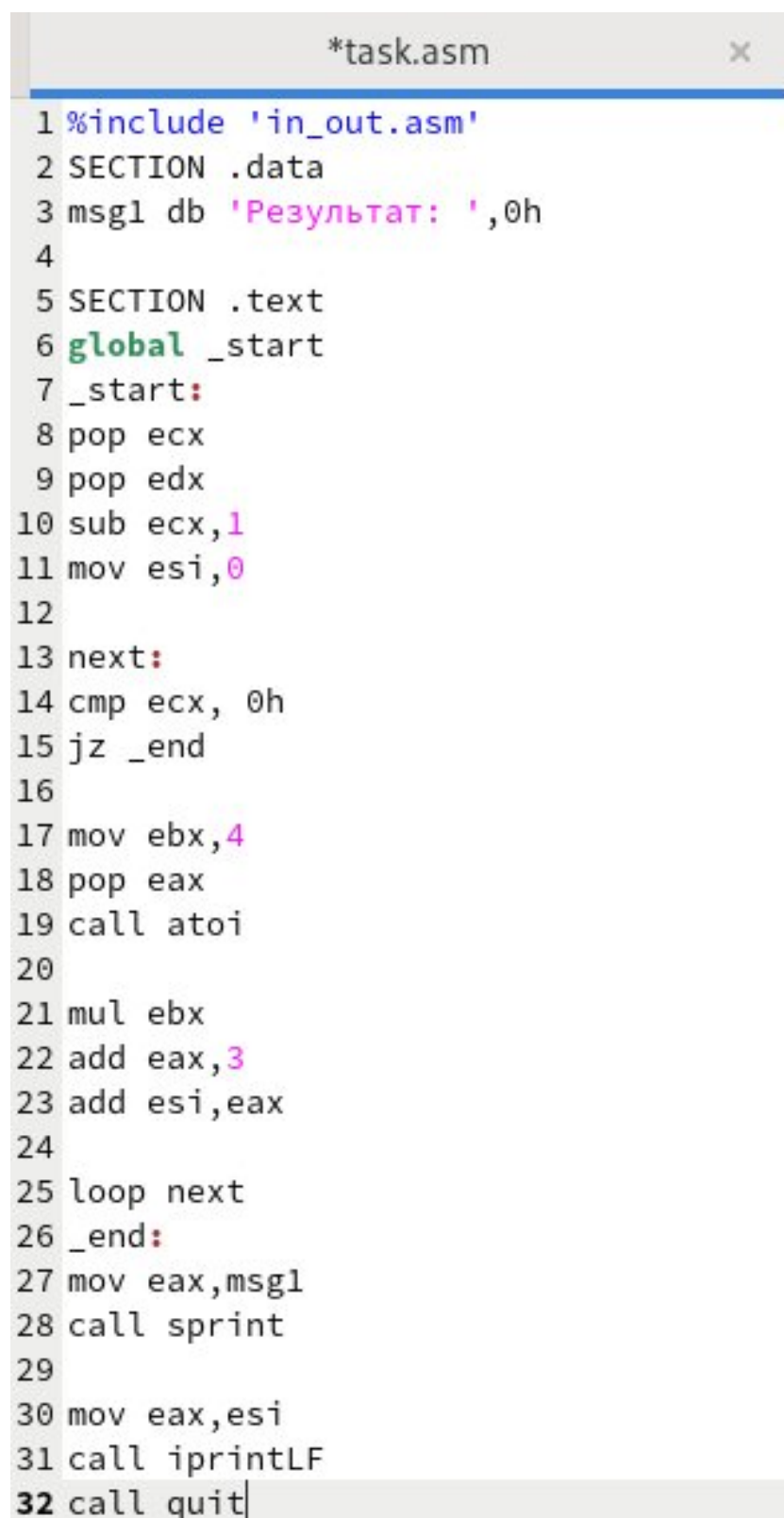


```
[aakayjnova@fedora lab08]$ nasm -f elf lab8-3.asm
[aakayjnova@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[aakayjnova@fedora lab08]$ ./lab8-3 1 2 3 4
Результат: 24
```

Рис. 4.13: Создание и запуск исполняемого файла

4.3 Выполнение заданий для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x)=4*x+3$ в соответствии с моим вариантом № 5 для значений $x=x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы.



```
*task.asm
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Результат: ',0h
4
5 SECTION .text
6 global _start
7 _start:
8 pop ecx
9 pop edx
10 sub ecx,1
11 mov esi,0
12
13 next:
14 cmp ecx, 0h
15 jz _end
16
17 mov ebx,4
18 pop eax
19 call atoi
20
21 mul ebx
22 add eax,3
23 add esi,eax
24
25 loop next
26 _end:
27 mov eax,msg1
28 call sprint
29
30 mov eax,esi
31 call iprintLF
32 call quit
```

Рис. 4.14: Текст программы

Создаю исполняемый файл и проверяю его работу на нескольких наборах $x=x_1, x_2, \dots, x_n$

```
[aakaynova@fedora lab08]$ nasm -f elf task.asm
[aakaynova@fedora lab08]$ ld -m elf_i386 -o task task.o
[aakaynova@fedora lab08]$ ./task 1 2 3
Результат: 33
[aakaynova@fedora lab08]$ ./task 100 101 102
Результат: 1221
```

Рис. 4.15: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

5 Выводы

В ходе данной лабораторной работы мы приобрели навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

- [illegible]

• • •

• • •