

Отчёт по лабораторной работе 8

Архитектура компьютера

Хулер Александрович Оюн

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация циклов в NASM	6
2.2	Задание для самостоятельной работы	16
3	Выводы	19

Список иллюстраций

2.1	Код программы lab8-1.asm	7
2.2	Компиляция и запуск программы lab8-1.asm	8
2.3	Код программы lab8-1.asm	9
2.4	Компиляция и запуск программы lab8-1.asm	10
2.5	Код программы lab8-1.asm	11
2.6	Компиляция и запуск программы lab8-1.asm	12
2.7	Код программы lab8-2.asm	13
2.8	Компиляция и запуск программы lab8-2.asm	13
2.9	Код программы lab8-3.asm	14
2.10	Компиляция и запуск программы lab8-3.asm	14
2.11	Код программы lab8-3.asm	15
2.12	Компиляция и запуск программы lab8-3.asm	16
2.13	Код программы lab8-4.asm	17
2.14	Компиляция и запуск программы lab8-4.asm	18

Список таблиц

1 Цель работы

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

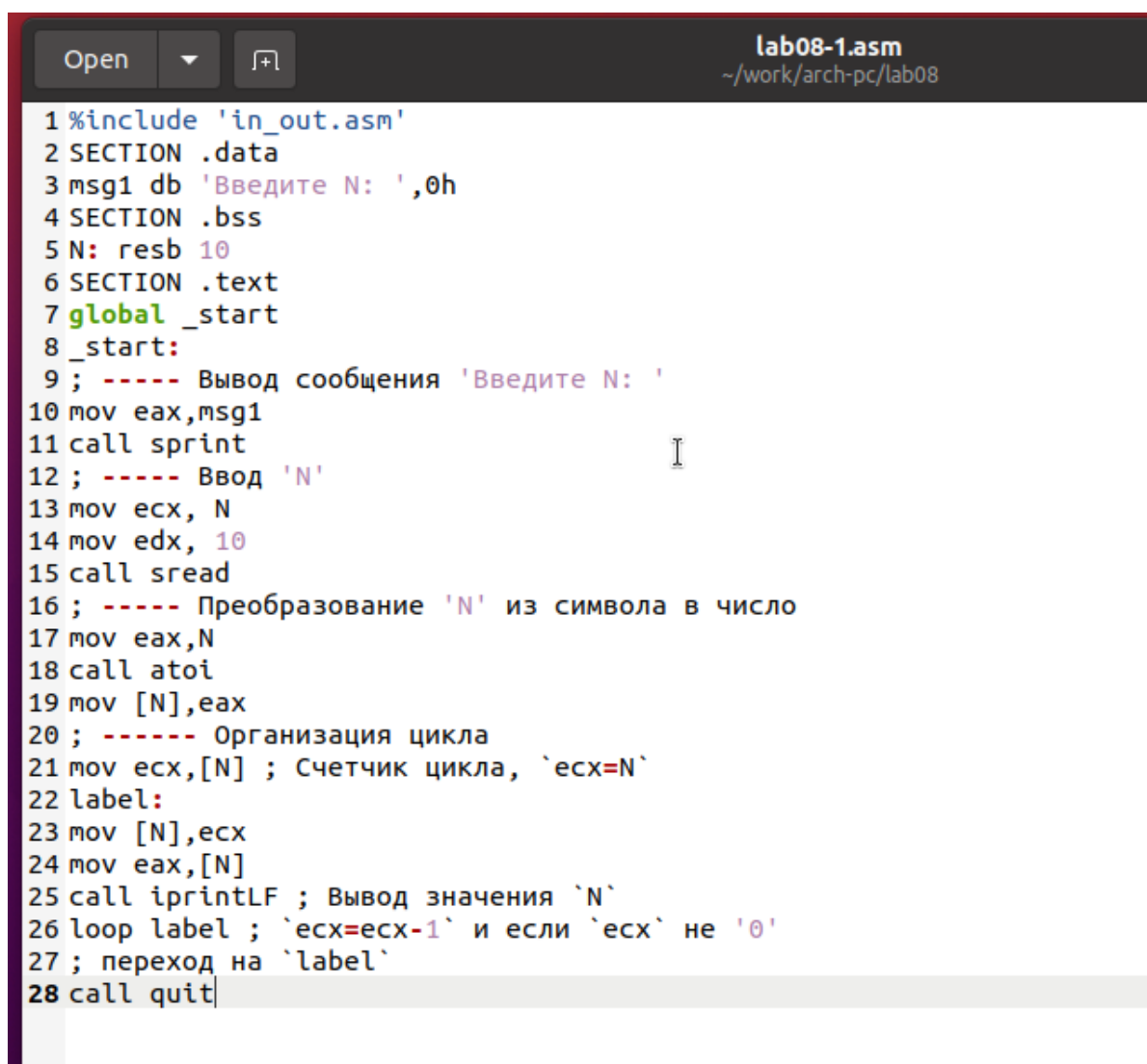
2 Выполнение лабораторной работы

2.1 Реализация циклов в NASM

Был организован каталог для выполнения лабораторного задания №8, в котором также был сформирован файл с наименованием lab8-1.asm.

Когда вы используете команду `loop` в NASM для создания циклических структур, важно учитывать, что она использует регистр `ecx` как счетчик, автоматически декрементируя его на один с каждым проходом цикла. Для наглядности рассмотрим пример кода, который демонстрирует значение регистра `ecx`.

В файл lab8-1.asm был введен код из примера 8.1. После этого была собрана исполняемая версия и проведена ее проверка.



```
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. 2.1: Код программы lab8-1.asm

```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-1.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-1.o -o lab08-1
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-1
Введите N: 8
8
7
6
5
4
3
2
1
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.2: Компиляция и запуск программы lab8-1.asm

В данном случае видно, что использование регистра `ecx` в команде `loop` может стать причиной ошибочного поведения программы. Я изменил код, изменив обработку значения регистра `ecx` во время цикла.

Теперь программа входит в бесконечный цикл, если `N` нечетное, и выводит только нечетные числа, если `N` четное.


```
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
28 ; переход на `label`
29 call quit
```

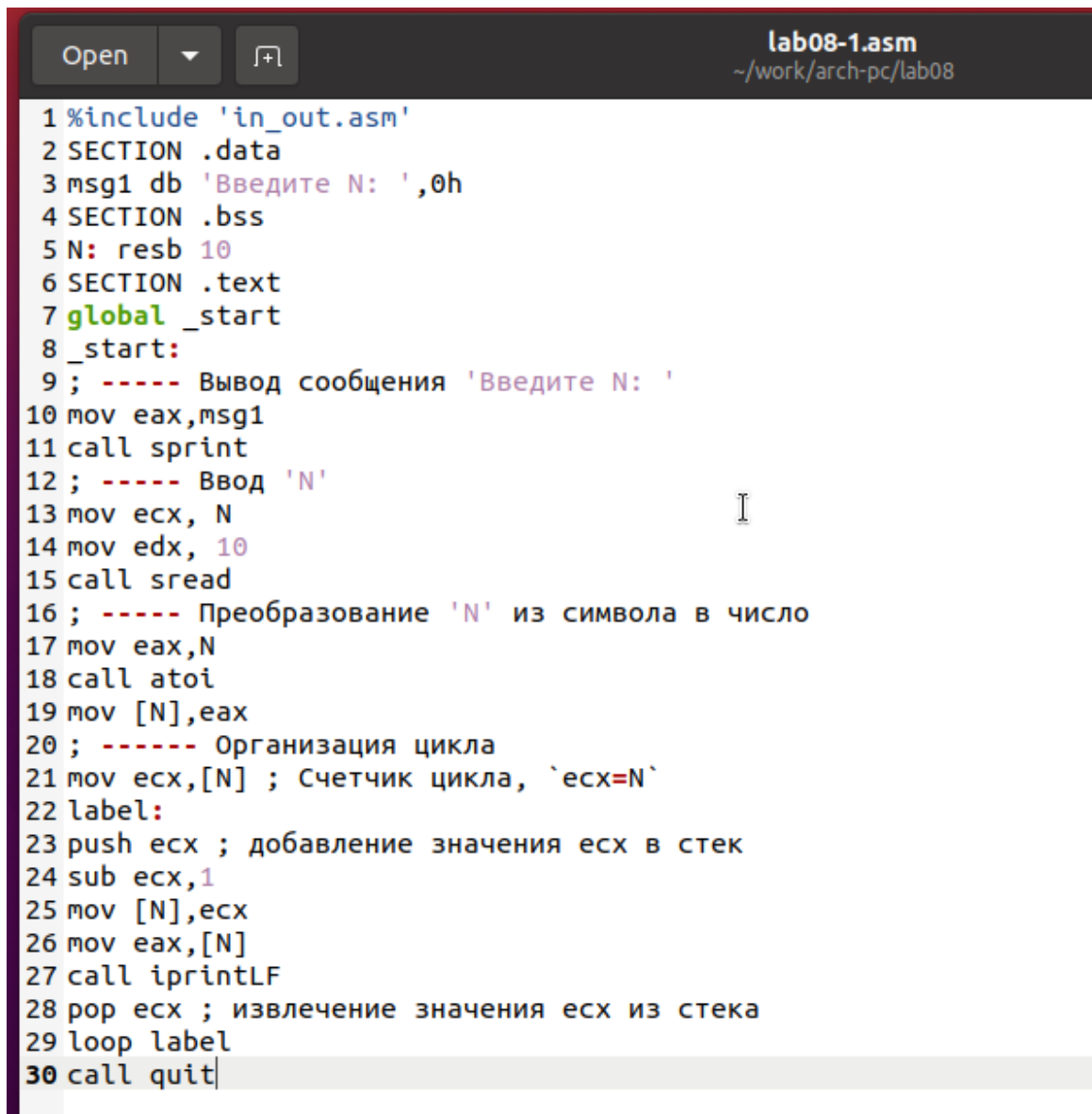
Рис. 2.3: Код программы lab8-1.asm

```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-1.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-1.o -o lab08-1
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-1
Введите N: 8
7
5
3
1
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.4: Компиляция и запуск программы lab8-1.asm

Для корректного использования регистра `ecx` в цикле и обеспечения правильной работы программы можно применить стек. Я модифицировал код, добавив инструкции `push` и `pop`, чтобы сохранить значение счетчика цикла `loop` в стеке.

Была сформирована исполняемая версия и осуществлена ее проверка. Программа отображает числа от $N-1$ до 0, где число итераций соответствует величине N .



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

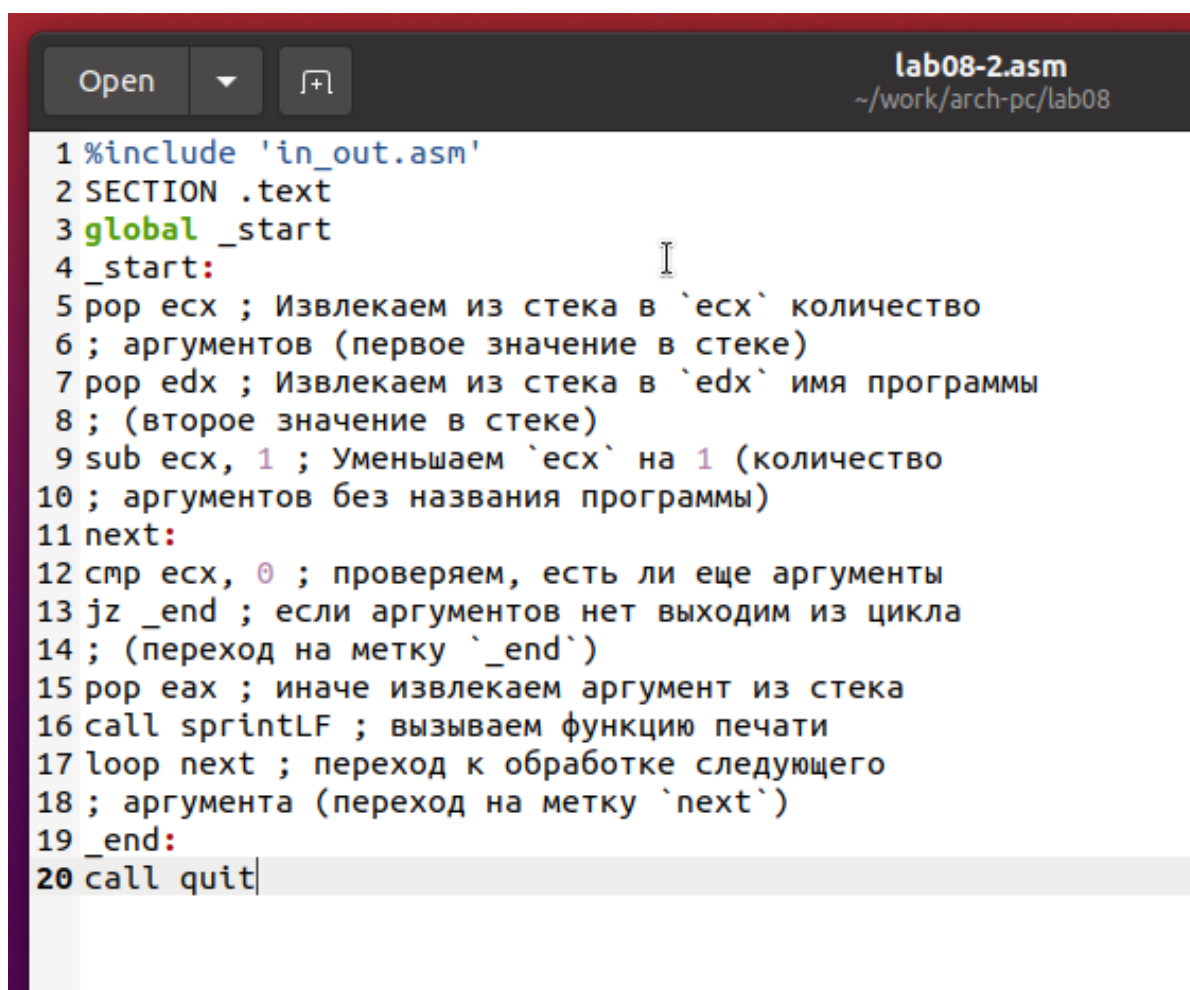
Рис. 2.5: Код программы lab8-1.asm

```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-1.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-1.o -o lab08-1
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-1
Введите N: 8
7
6
5
4
3
2
1
0
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.6: Компиляция и запуск программы lab8-1.asm

Я создал файл с именем lab8-2.asm в папке ~/work/arch-pc/lab08 и занес в него код, взятый из примера 8.2.

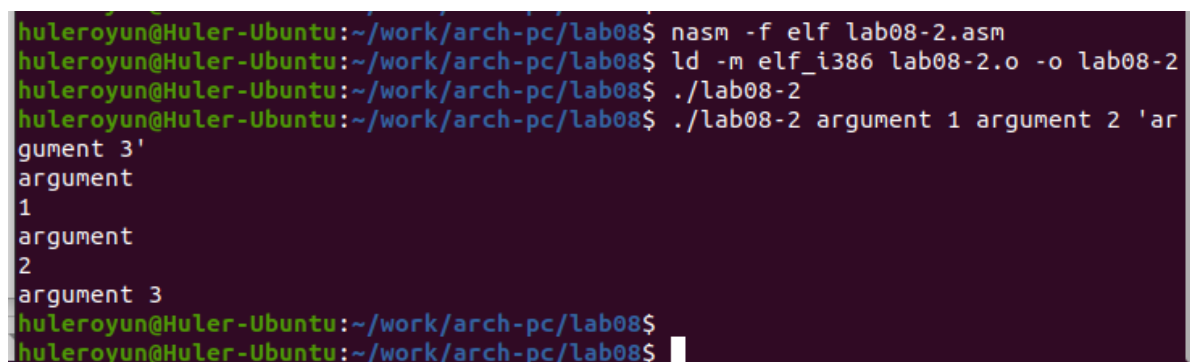
После этого я собрал исполняемый файл из исходного кода и запустил его с параметрами. В итоге программа успешно обработала пять переданных ей параметров. Под параметрами понимаются элементы, разделяемые пробелами, которые могут быть текстом или числами.



```
lab08-2.asm
~/work/arch-pc/lab08

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 sprintf ; вызываем функцию печати
17     loop next ; переход к обработке следующего
18     ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 2.7: Код программы lab8-2.asm

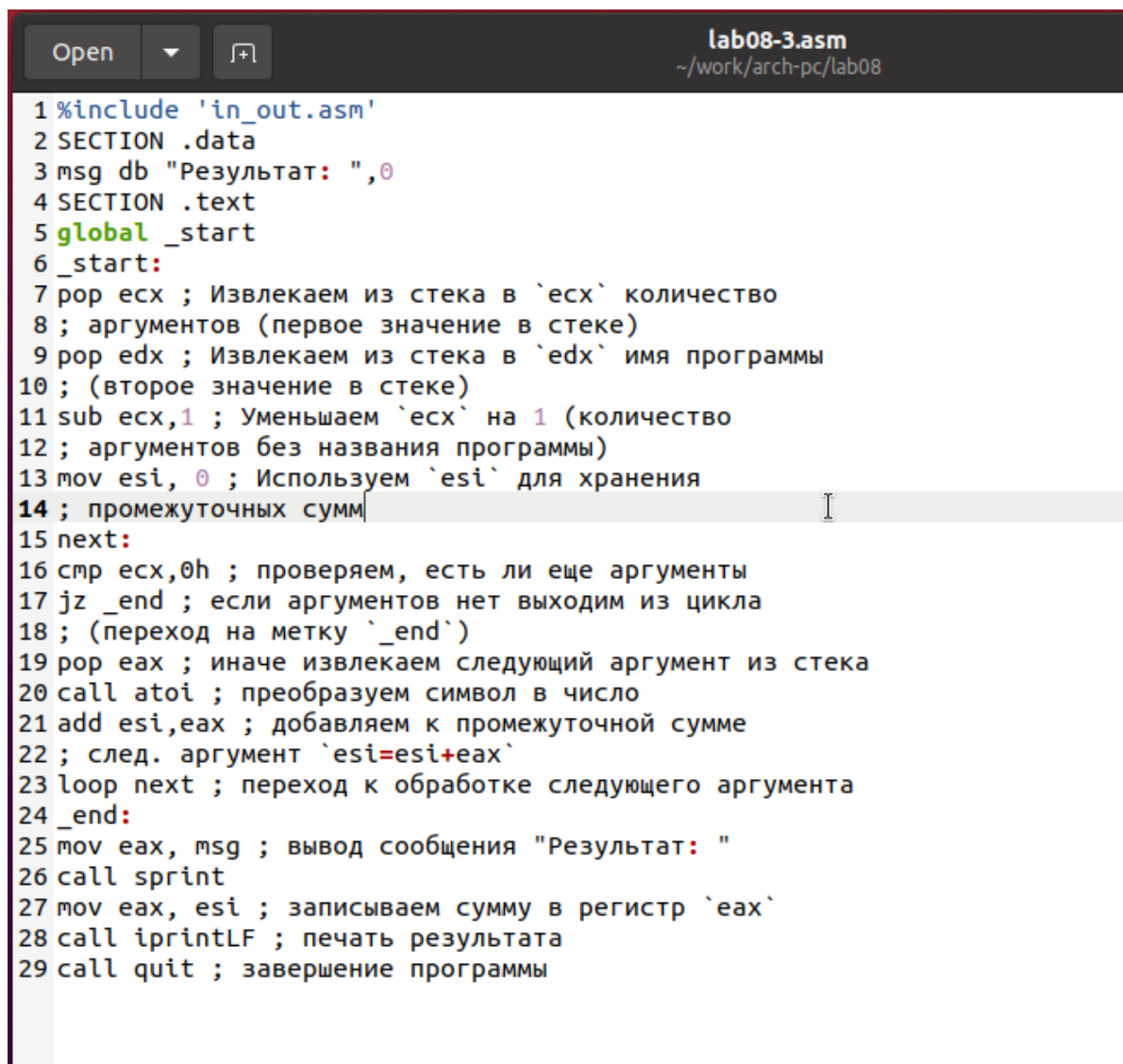


```
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-2.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-2.o -o lab08-2
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-2
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-2 argument 1 argument 2 'ar
argument 3'
argument
1
argument
2
argument 3
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.8: Компиляция и запуск программы lab8-2.asm

Теперь давайте рассмотрим другой пример программы, задачей которой яв-

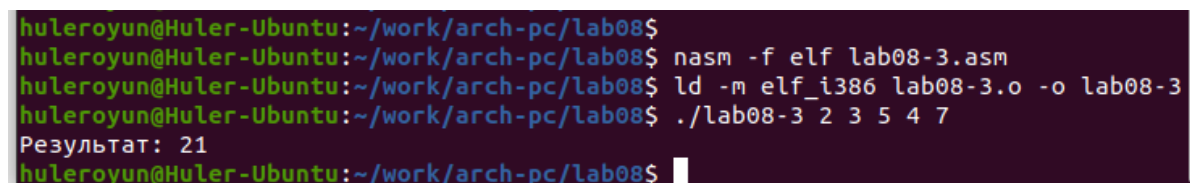
ляется вывод на экран суммы чисел, передаваемых в неё в качестве параметров.



```
lab08-3.asm
~/work/arch-pc/lab08

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
```

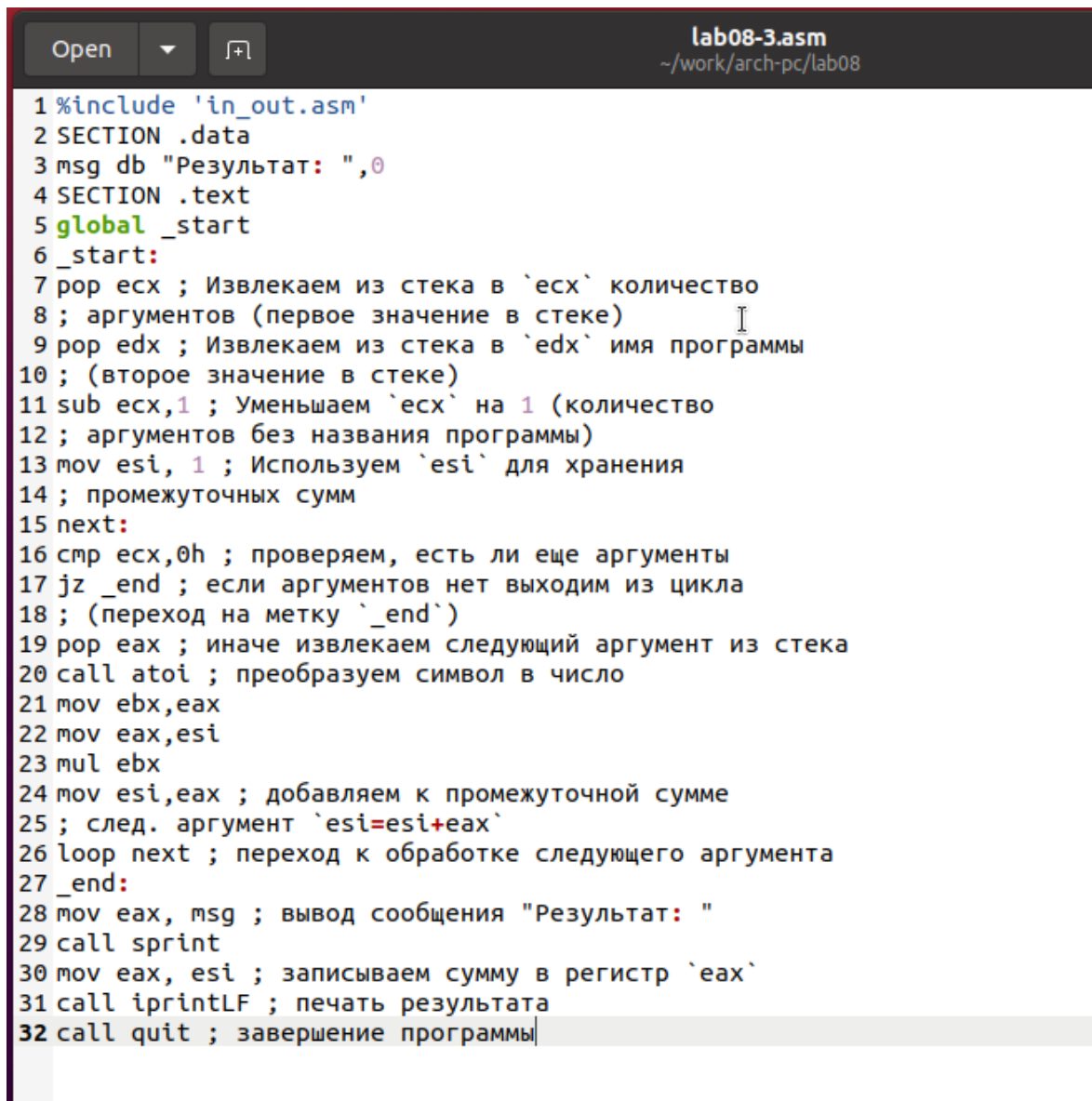
Рис. 2.9: Код программы lab8-3.asm



```
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-3.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-3.o -o lab08-3
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-3 2 3 5 4 7
Результат: 21
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.10: Компиляция и запуск программы lab8-3.asm

Я внес изменения в код из примера 8.3 таким образом, чтобы программа теперь вычисляла произведение значений, переданных через командную строку.



```
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 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 mov ebx,eax
22 mov eax,esi
23 mul ebx
24 mov esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент `esi=esi+eax`
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax, msg ; вывод сообщения "Результат: "
29 call sprint
30 mov eax, esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программы
```

Рис. 2.11: Код программы lab8-3.asm

```

huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-3.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-3.o -o lab08-3
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-3 2 3 5 4 7
Результат: 21
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-3.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-3.o -o lab08-3
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-3 2 3 5 4 7
Результат: 840
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab08$

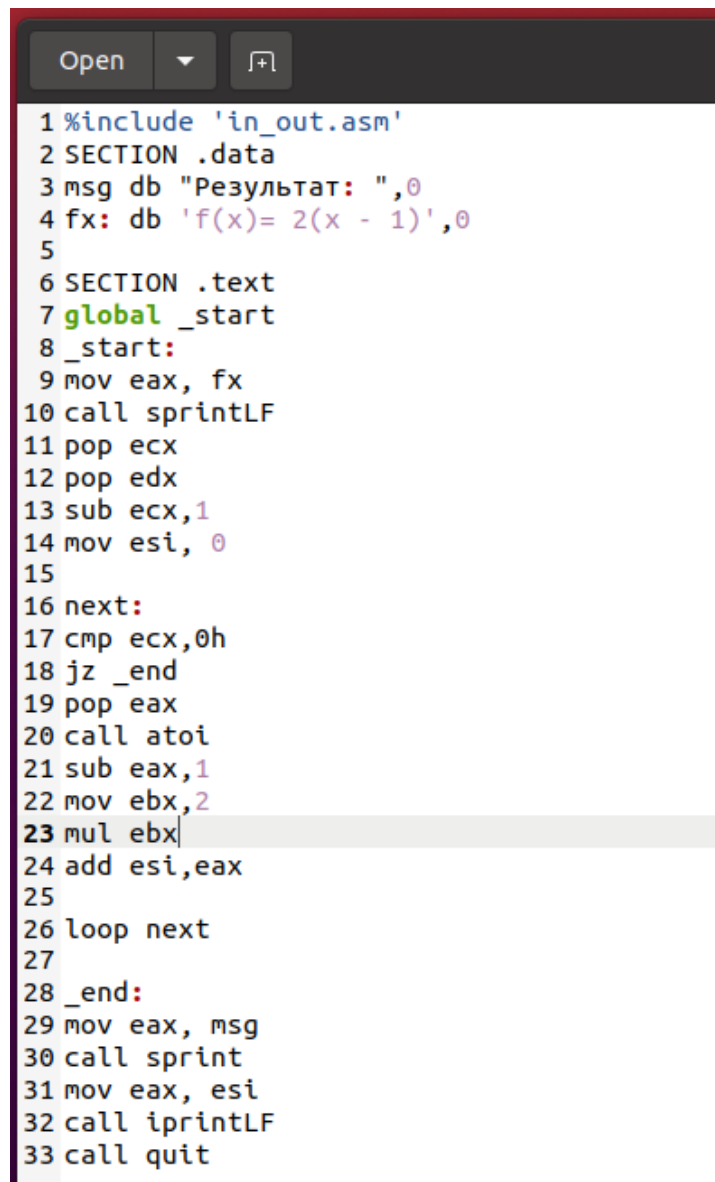
```

Рис. 2.12: Компиляция и запуск программы lab8-3.asm

2.2 Задание для самостоятельной работы

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 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 .

Мой вариант 4: $f(x) = 2(x - 1)$



```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 2(x - 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
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 sub eax,1
22 mov ebx,2
23 mul ebx
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprintf
31 mov eax, esi
32 call iprintLF
33 call quit

```

Рис. 2.13: Код программы lab8-4.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(1) = 0, f(2) = 2$

Затем подал несколько аргументов и получил сумму значений функции.

```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab08-4.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab08-4.o -o lab08-4
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-4
f(x)= 2(x - 1)
Результат: 0
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-4 1
f(x)= 2(x - 1)
Результат: 0
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-4 2
f(x)= 2(x - 1)
Результат: 2
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$ ./lab08-4 2 3 6 5 4 7
f(x)= 2(x - 1)
Результат: 42
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.14: Компиляция и запуск программы lab8-4.asm

3 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `asm`.