

Отчёт по лабораторной работе 8

Архитектура компьютера

Ушаков Данила Алексеевич

Содержание

| | | |
|----------|-------------------------------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 2.1 | Реализация циклов в NASM | 6 |
| 2.2 | Обработка аргументов командной строки | 11 |
| 2.3 | Задание для самостоятельной работы | 15 |
| 3 | Выводы | 18 |

Список иллюстраций

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

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

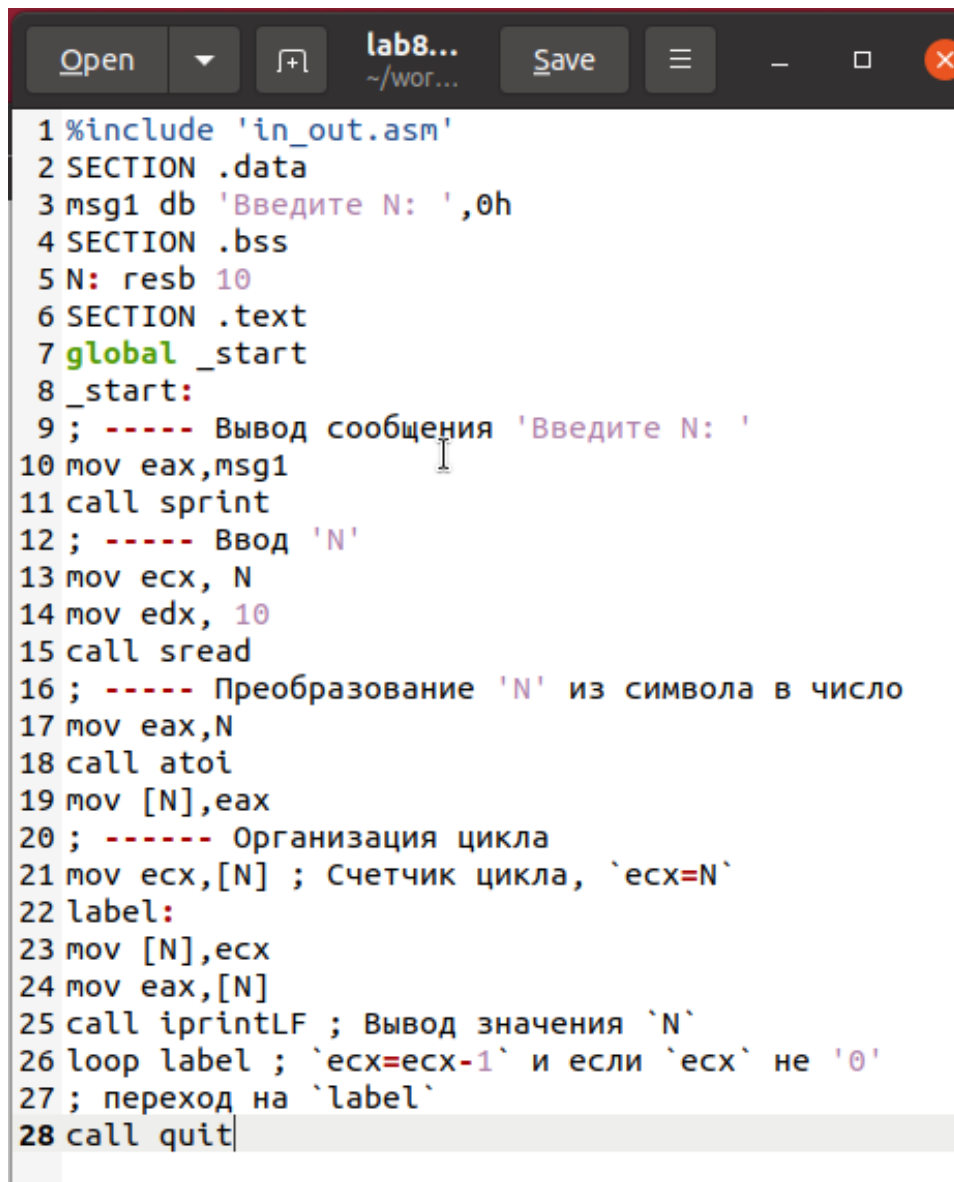
2.1 Реализация циклов в NASM

Был создан каталог для выполнения лабораторной работы № 8, а также создан файл с именем lab8-1.asm.

При использовании инструкции `loop` в NASM для реализации циклов, необходимо помнить о следующем: данная инструкция использует регистр `ecx` в качестве счетчика и на каждой итерации уменьшает его значение на единицу.

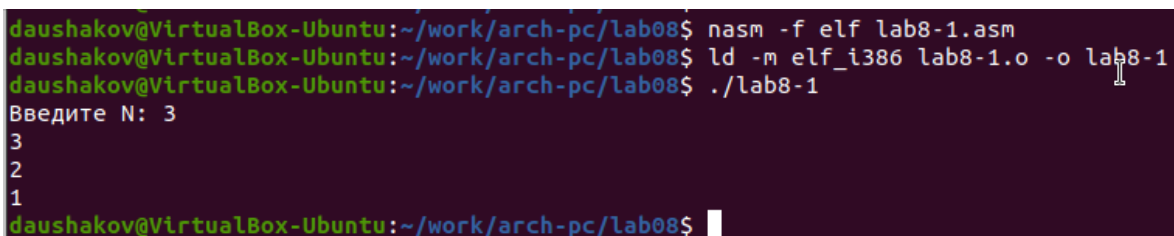
Для лучшего понимания этого процесса, рассмотрим пример программы, которая выводит значение регистра `ecx`.

Написал в файл lab8-1.asm текст программы из листинга 8.1. (рис. 2.1) Создал исполняемый файл и проверил его работу. (рис. 2.2)



```
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

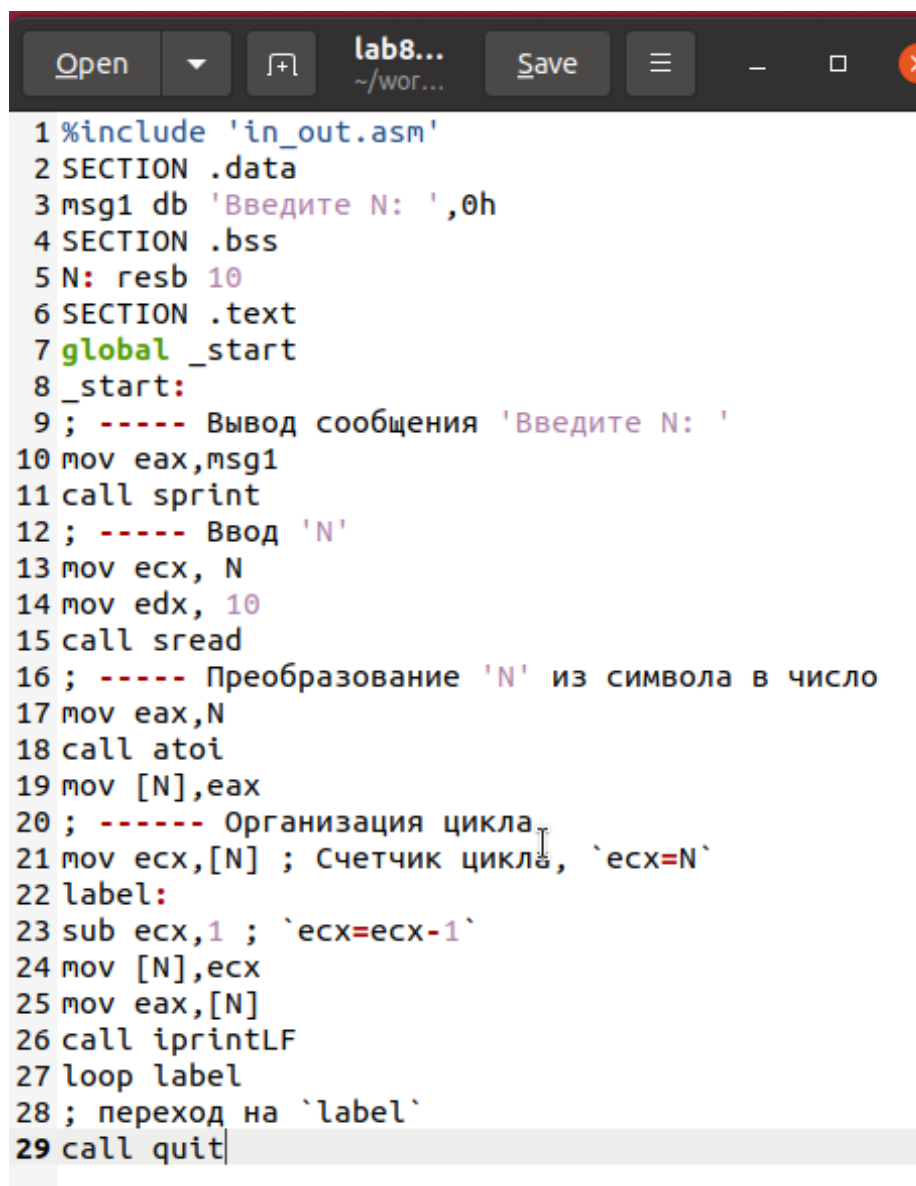


```
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

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

В данном примере демонстрируется, что использование регистра `ecx` в инструкции `loop` может привести к неправильной работе программы. В тексте программы были внесены изменения, которые включают изменение значения регистра `ecx` внутри цикла. (рис. 2.3)

Программа запускает бесконечный цикл при нечетном значении `N` и выводит только нечетные числа при четном значении `N`. (рис. 2.4)



```
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
```

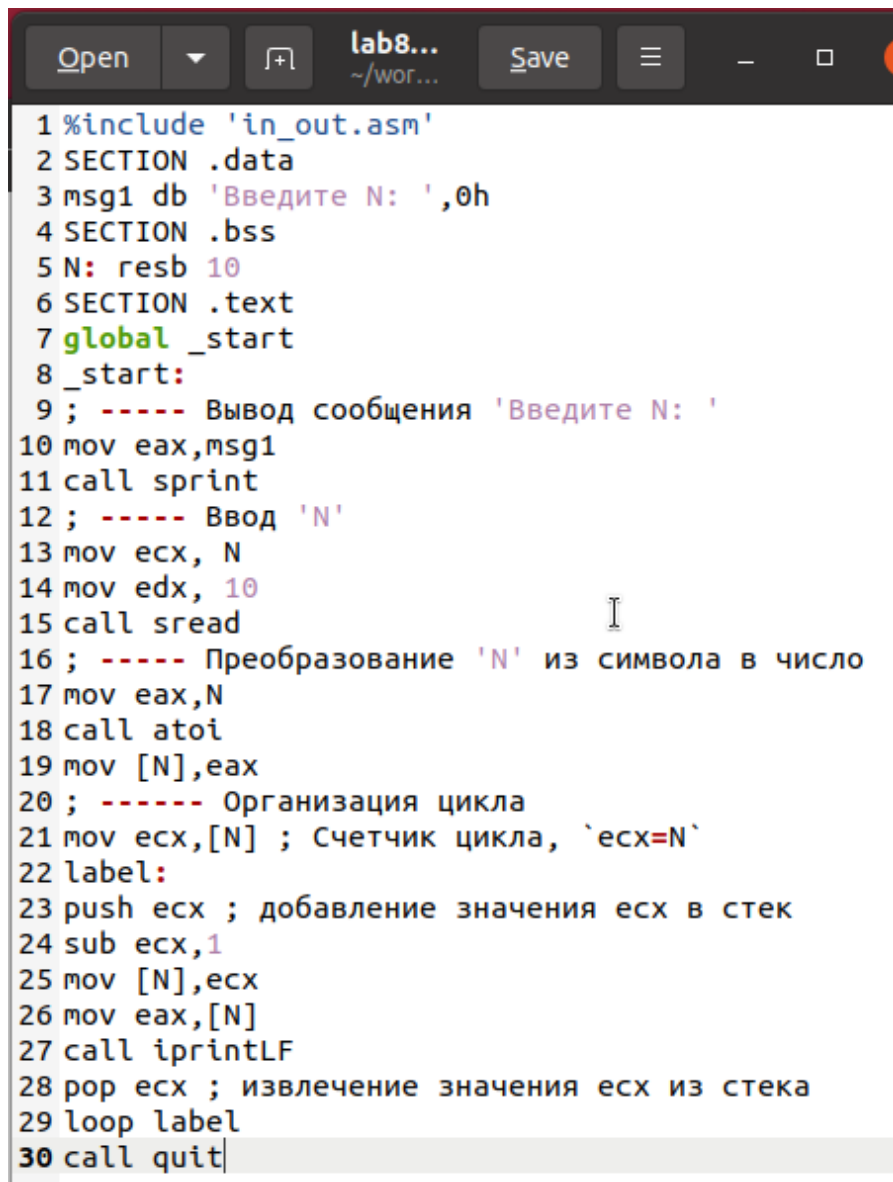
Рис. 2.3: Код программы lab8-1.asm


```
4294939236
4294939234
4294939232
^C
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

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

Для того чтобы использовать регистр `ecx` в цикле и обеспечить корректность работы программы, можно применить стек. Внесены изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop` в стеке. (рис. 2.5)

Был создан исполняемый файл и проверена его работа. Программа выводит числа от $N-1$ до 0, где количество проходов цикла соответствует значению N . (рис. 2.6)



```
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

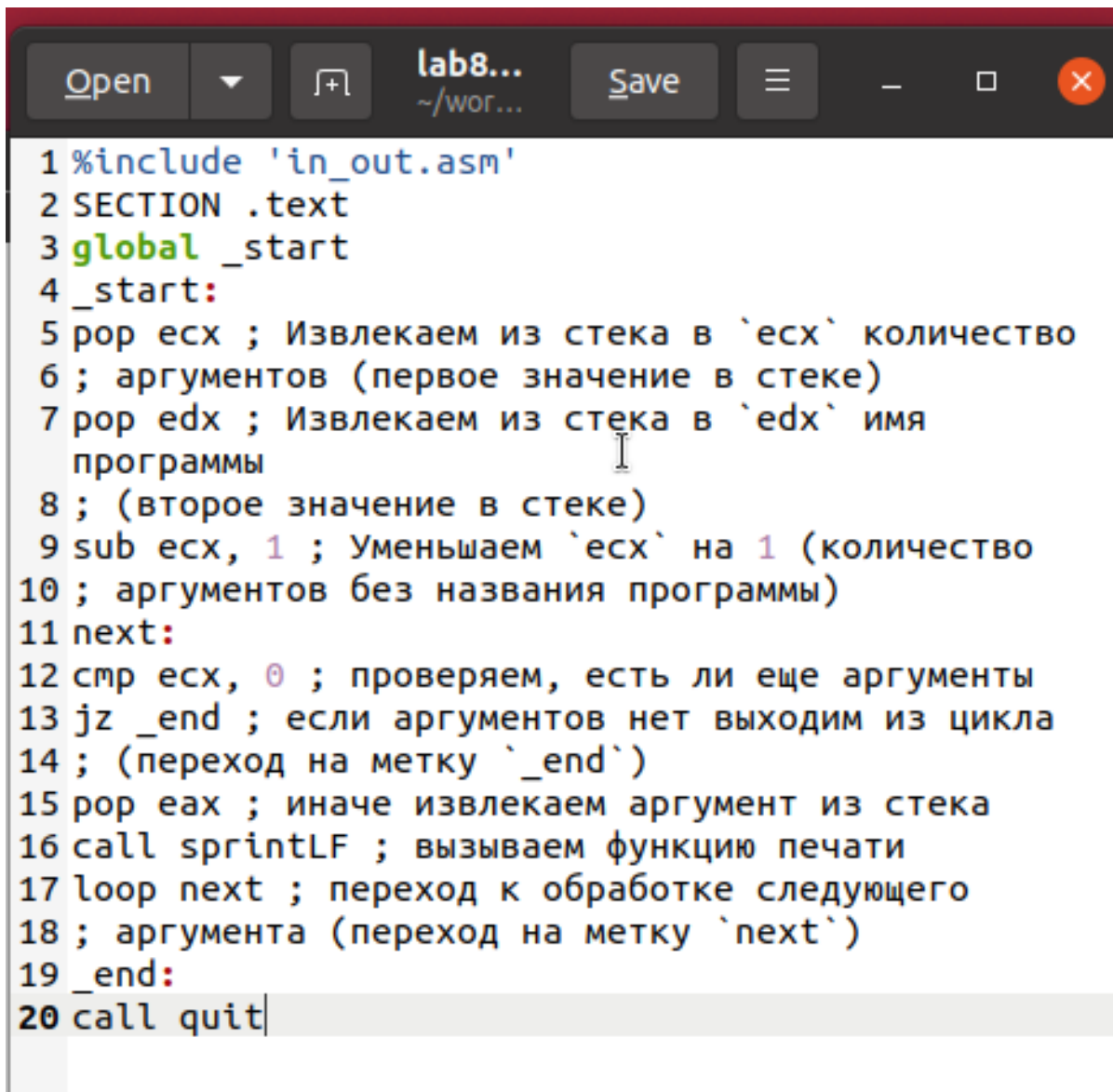
```
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
2
1
0
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
2
1
0
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

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

2.2 Обработка аргументов командной строки

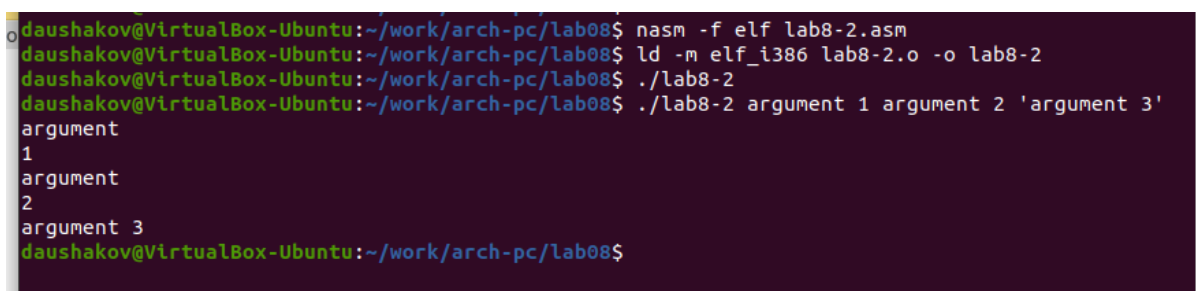
Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввел в него текст программы из листинга 8.2. (рис. 2.7)

Создал исполняемый файл и запустил его, указав аргументы. Программа обработала 5 аргументов. Аргументами считаются слова/числа, разделенные пробелом. (рис. 2.8)



```
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
```

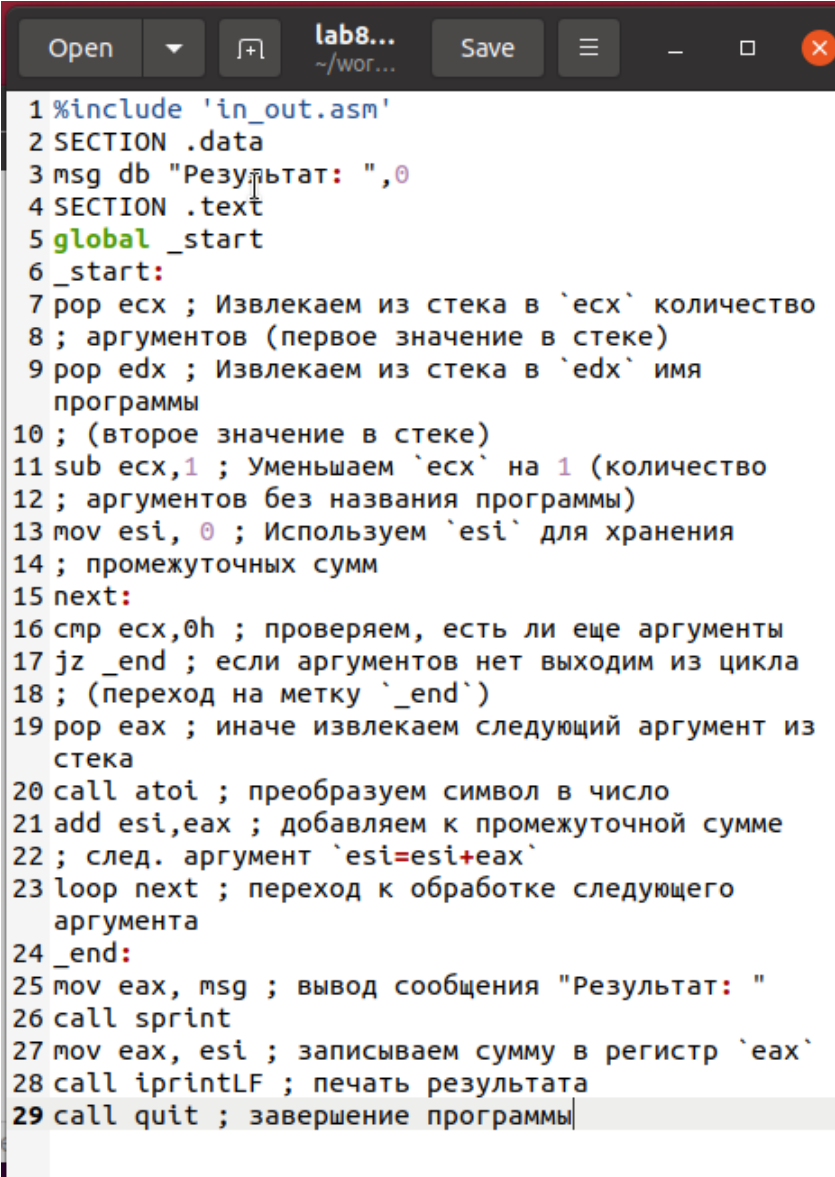
Рис. 2.7: Код программы lab8-2.asm



```
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-2 argument 1 argument 2 'argument 3'
argument
1
argument
2
argument 3
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

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

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. (рис. 2.9) (рис. 2.10)



```
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

```

daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 10
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$

```

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

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 2.11) (рис. 2.12)

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; 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

```
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4
Результат: 24
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

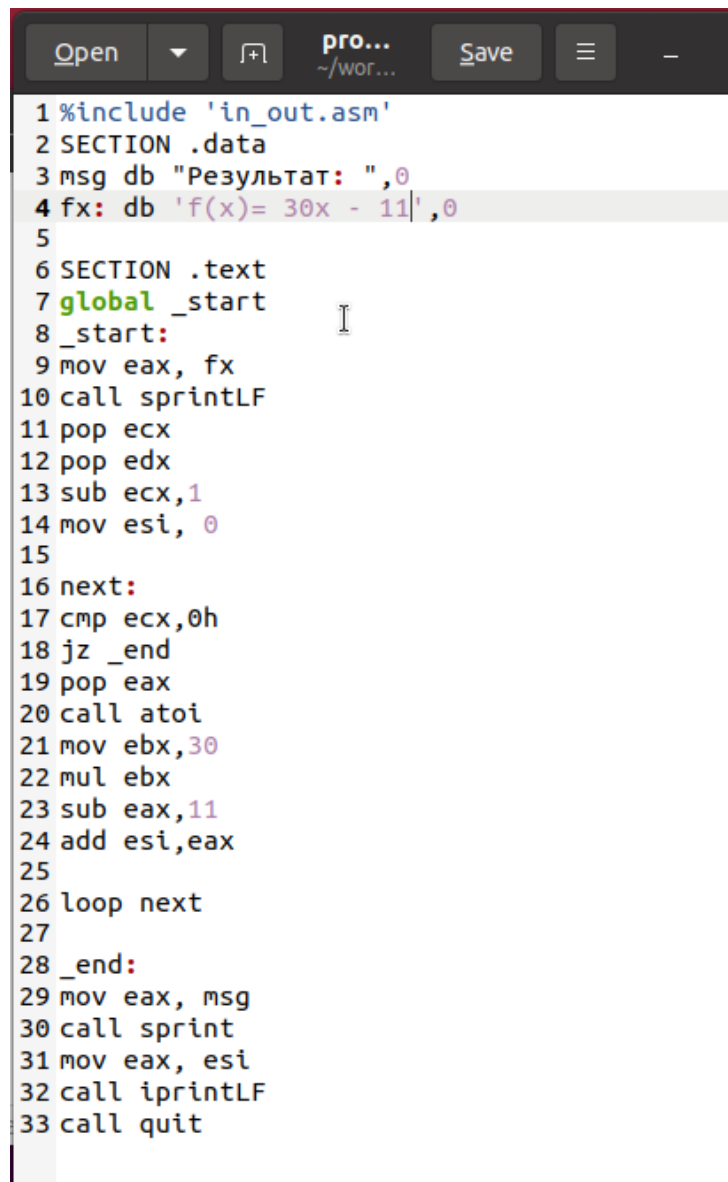
Рис. 2.12: Компиляция и запуск программы lab8-3.asm

2.3 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 . (рис. 2.13) (рис. 2.14)

Мой вариант 16:

$$f(x) = 30x - 11$$



```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 30x - 11|',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 mov ebx,30
22 mul ebx
23 sub eax,11
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 prog.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(1) = 5, f(2) = 17$

Затем подал несколько аргументов и получил сумму значений функции.


```
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ nasm -f elf prog.asm
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 prog.o -o prog
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./prog
f(x)= 30x - 11
Результат: 0
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./prog 1
f(x)= 30x - 11
Результат: 19
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$ ./prog 1 3 4 5 6
f(x)= 30x - 11
Результат: 515
daushakov@VirtualBox-Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.14: Компиляция и запуск программы prog.asm

3 Выводы

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