

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Дмитрий Сергеевич Хохлов

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Самостоятельное задание	16
3	Выводы	19
	Список литературы	20

Список иллюстраций

2.1	Программа в файле lab8-1.asm	7
2.2	Запуск программы lab8-1.asm	8
2.3	Программа в файле lab8-1.asm	9
2.4	Запуск программы lab8-1.asm	10
2.5	Программа в файле lab8-1.asm	11
2.6	Запуск программы lab8-1.asm	12
2.7	Программа в файле lab8-2.asm	13
2.8	Запуск программы lab8-2.asm	13
2.9	Программа в файле lab8-3.asm	14
2.10	Запуск программы lab8-3.asm	14
2.11	Программа в файле lab8-3.asm	15
2.12	Запуск программы lab8-3.asm	16
2.13	Программа в файле lab8-4.asm	17
2.14	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

Сформировал директорию для программ, связанных с лабораторной № 8, и создал исходный файл с именем lab8-1.asm.

При использовании циклов в ассемблере NASM и применении команды `loop` следует учитывать, что она автоматически декрементирует счетчик, находящийся в регистре `ecx`. Возьмем для примера программу, демонстрирующую вывод текущего значения регистра `ecx`.

В файл lab8-1.asm внес код из примера под номером 8.1, представленного на иллюстрации [2.1]. Скомпилировал программу, получил исполняемый файл и осуществил его тестирование, результаты которого отражены на рисунке [2.2].

Open ▾ [+]

lab8-1.asm
~/work/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
```

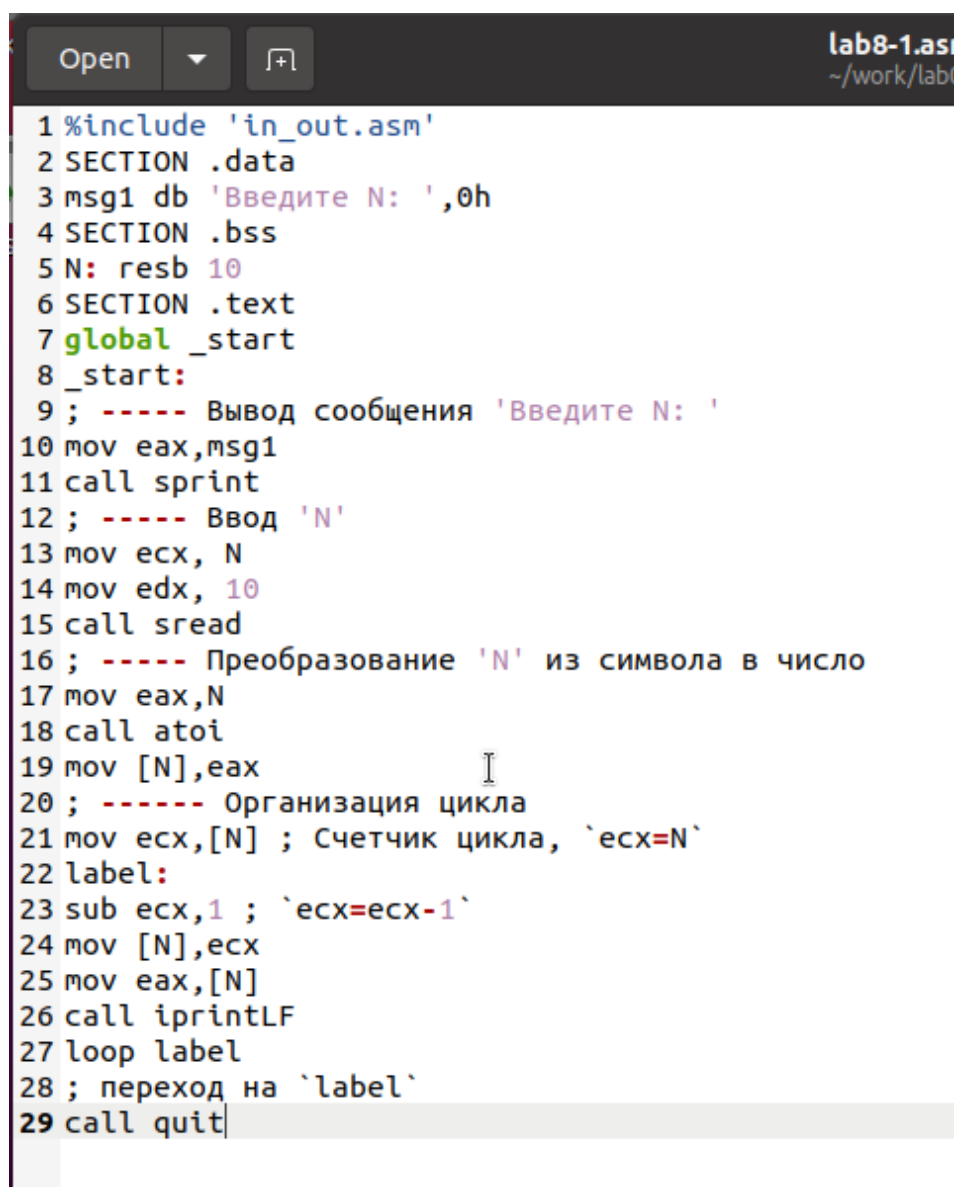
Рис. 2.1: Программа в файле lab8-1.asm

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-1.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-1
Введите N: 6
6
5
4
3
2
1
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
```

Рис. 2.2: Запуск программы lab8-1.asm

Приведенный пример иллюстрирует, что манипуляции с регистром `ecx` внутри цикла `loop` могут привести к ошибкам в работе программы. Внес корректировки в код программы, добавив действия по изменению значения регистра `ecx` в процессе итераций, как показано на рисунке [2.3].

Программа инициирует бесконечный цикл, если `N` имеет нечетное значение, и выводит исключительно нечетные числа, если `N` четное, как видно на рисунке [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
```

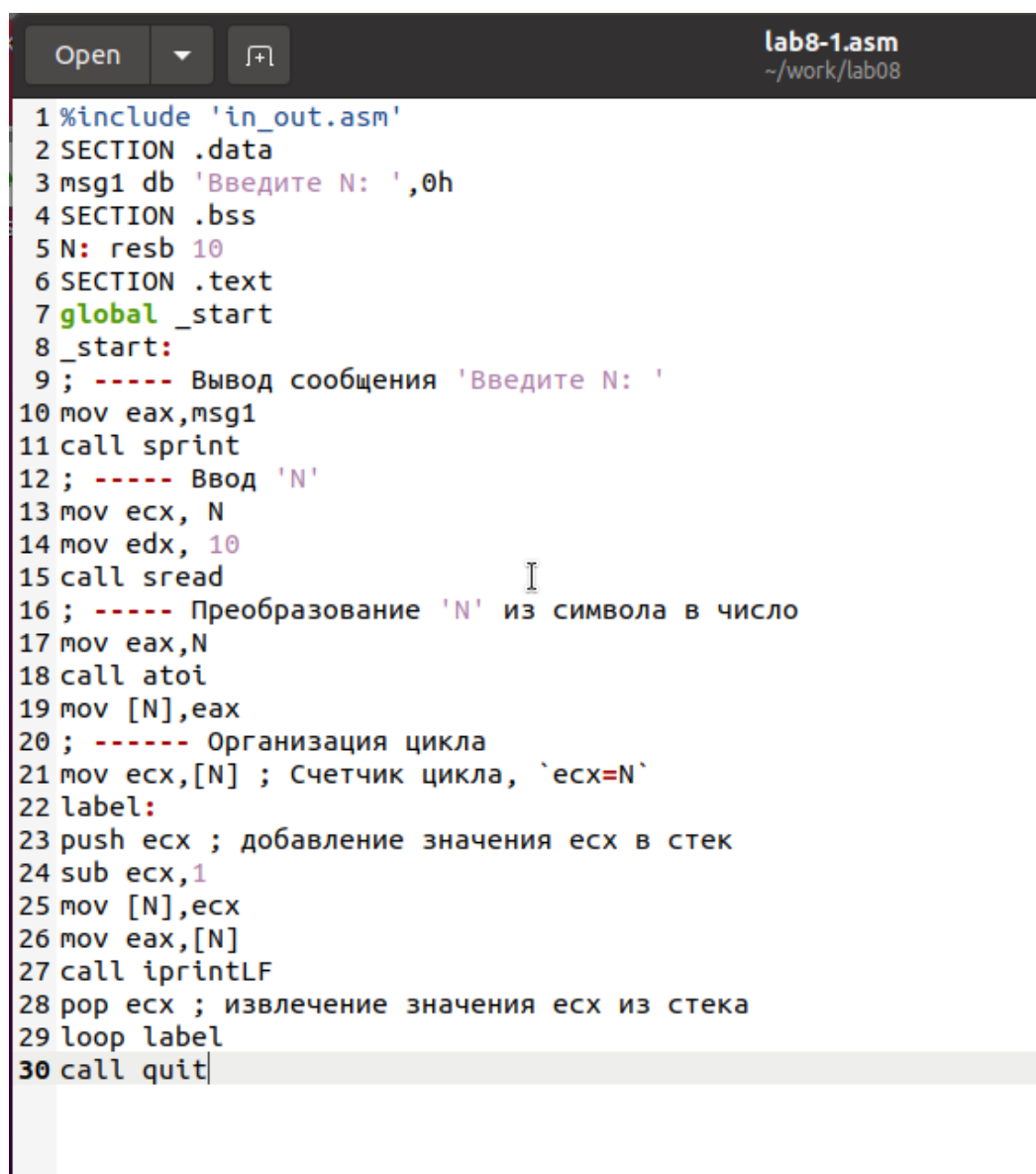
Рис. 2.3: Программа в файле lab8-1.asm

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-1.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-1
Введите N: 6
5
3
1
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
```

Рис. 2.4: Запуск программы lab8-1.asm

Для корректного использования регистра `ecx` в цикле и избегания ошибок можно применять стек. Произвел модификацию программного кода, добавив инструкции `push` и `pop` для сохранения и восстановления счетчика цикла, что отображено на рисунке [2.5].

Скомпилировал исполняемый файл и провел его тестирование. Программа последовательно выводит числа от $N-1$ до 0, с количеством итераций, равным N , что подтверждается изображением [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
```

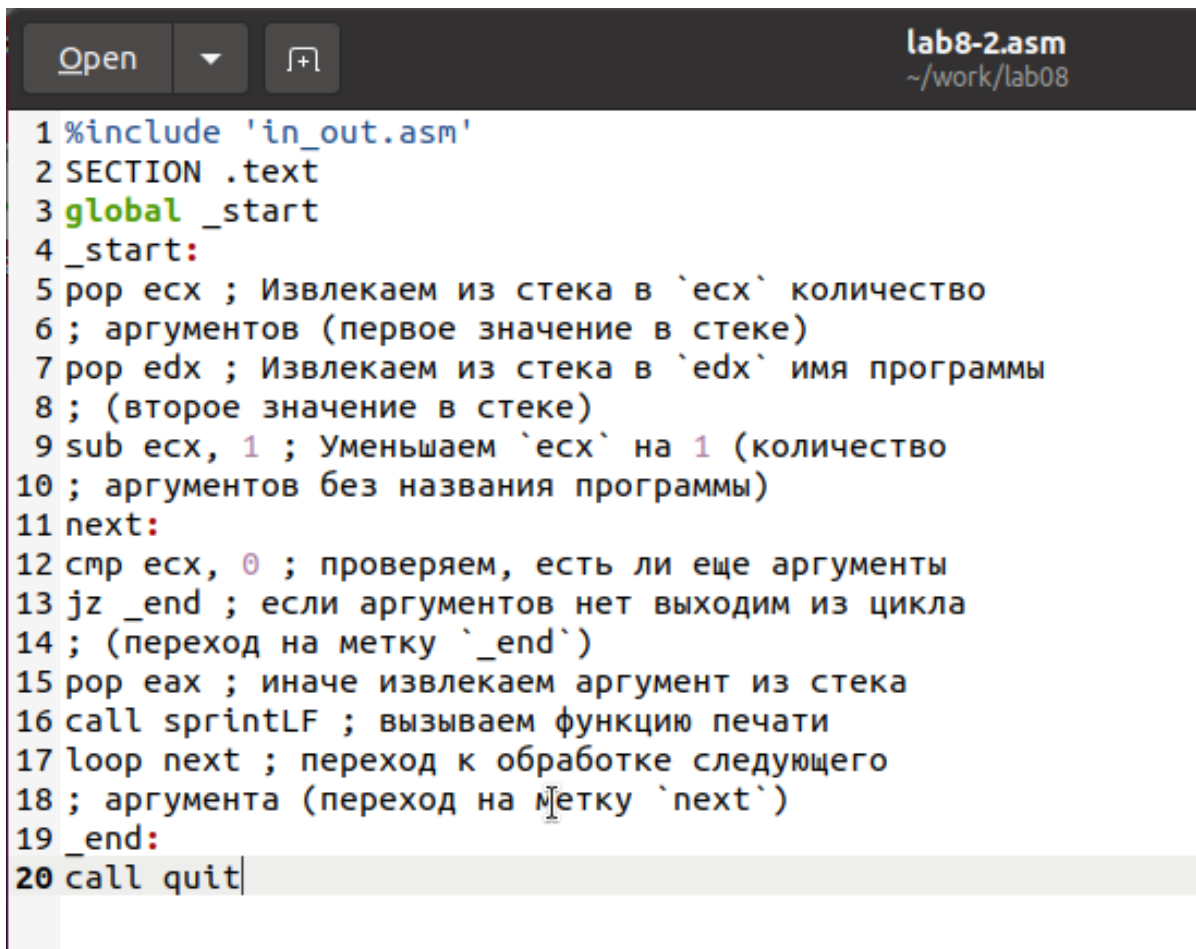
Рис. 2.5: Программа в файле lab8-1.asm

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nas -f elf lab8-1.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-1
Введите N: 6
5
4
3
2
1
0
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
```

Рис. 2.6: Запуск программы lab8-1.asm

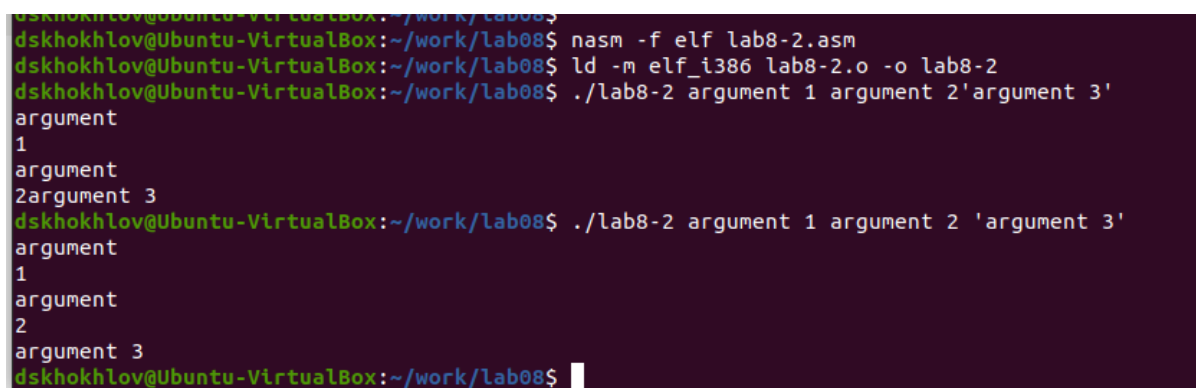
Сформировал файл lab8-2.asm в директории ~/work/arch-pc/lab08 и занес в него код из примера 8.2, как показано на иллюстрации [2.7].

Скомпилировал и выполнил скомпилированный файл, предоставив ему в качестве параметров несколько аргументов. В результате, программа успешно обработала пять аргументов, которые определяются как отдельные слова или числа, разделенные пробелами, (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: Программа в файле lab8-2.asm

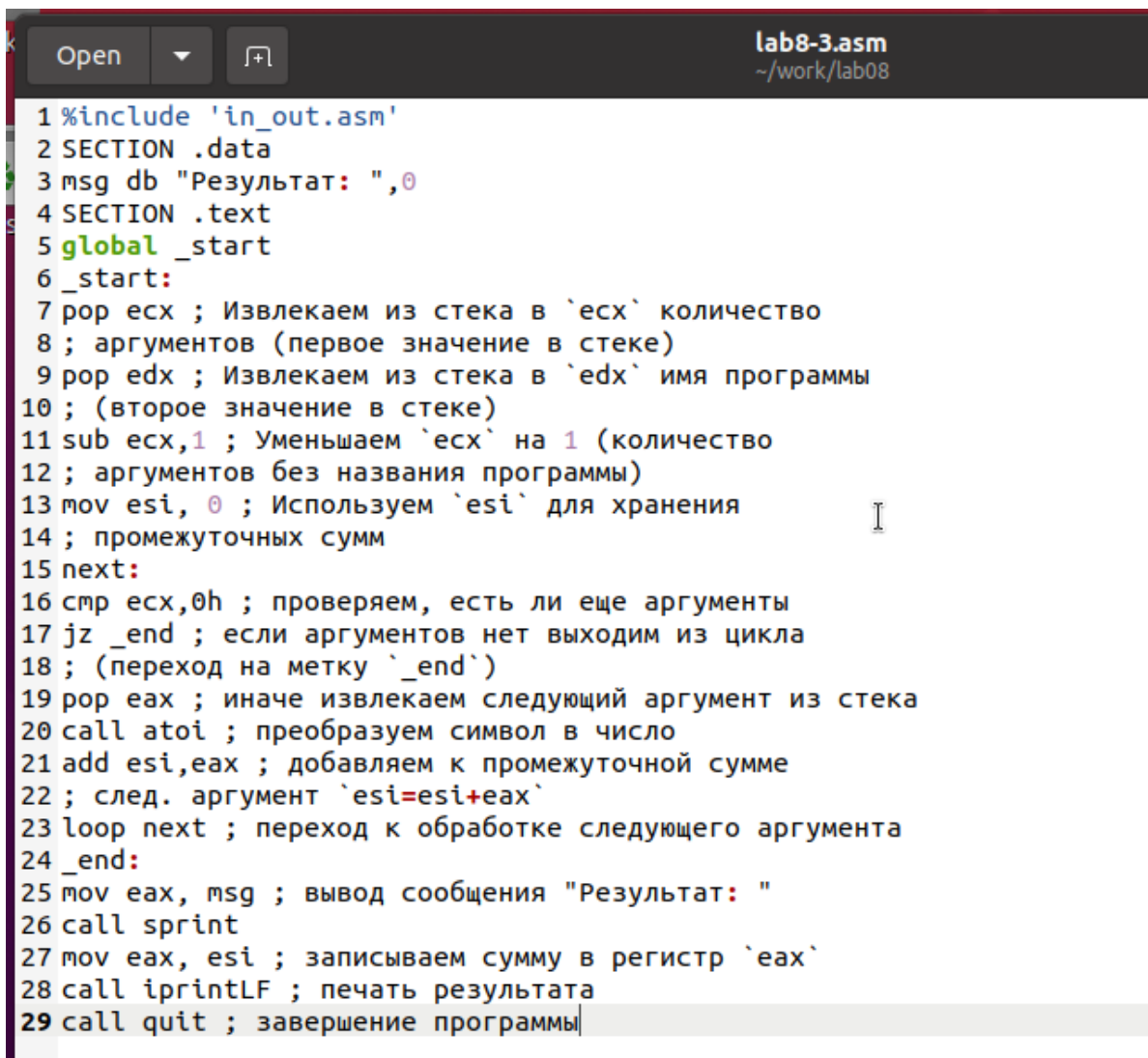


```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-2.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-2 argument 1 argument 2 'argument 3'
argument
1
argument
2
argument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

Рис. 2.8: Запуск программы lab8-2.asm

Теперь давайте обратим внимание на другой пример программы, которая

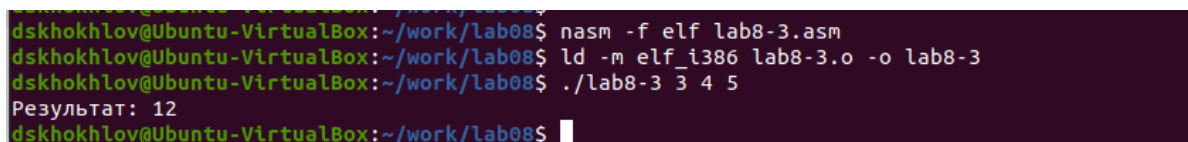
демонстрирует расчет суммы чисел, передаваемых в нее в виде аргументов командной строки, что визуализировано на рисунках [2.9] и [2.10].



```
lab8-3.asm
~/work/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
```

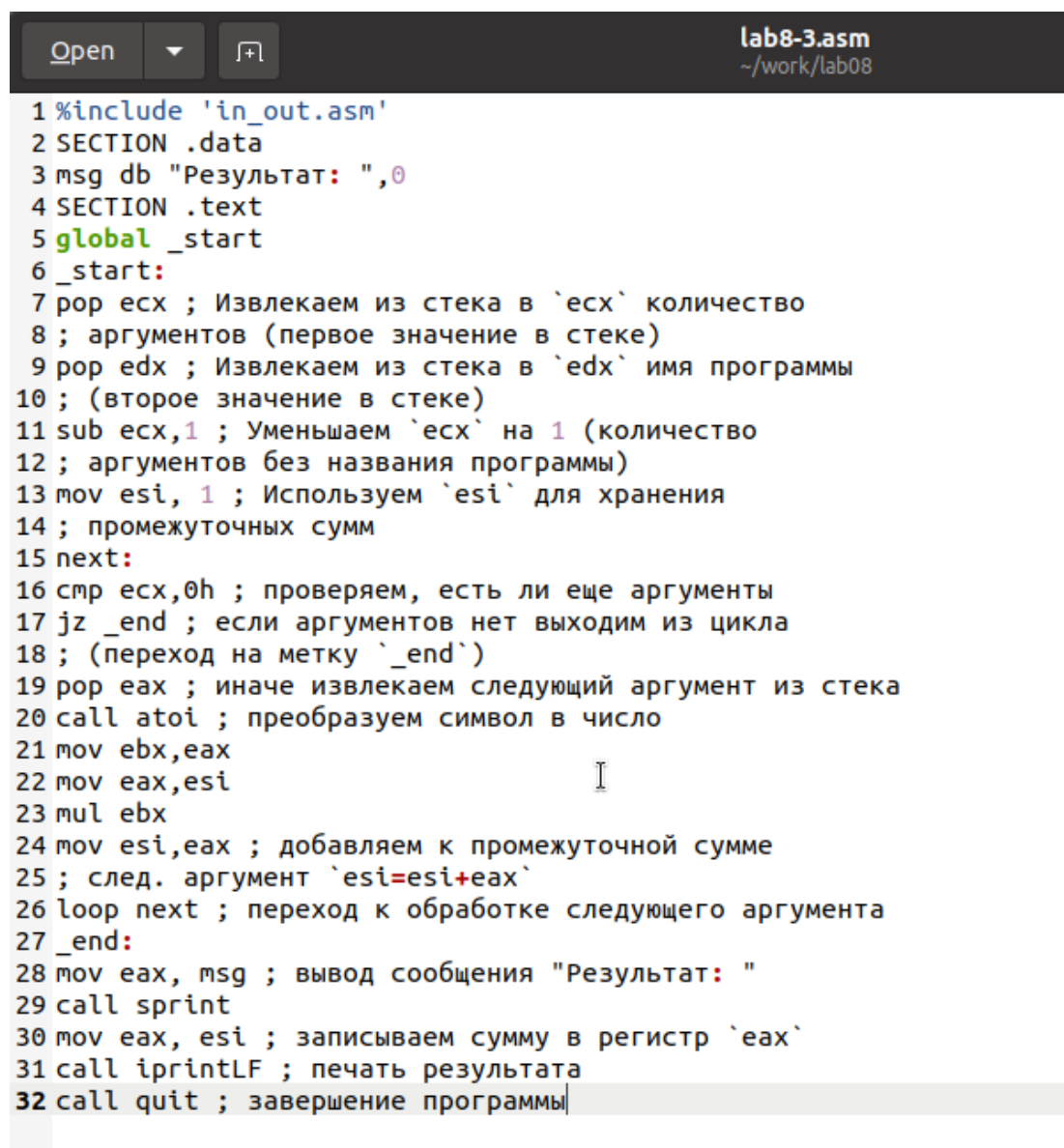
Рис. 2.9: Программа в файле lab8-3.asm



```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-3.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-3 3 4 5
Результат: 12
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
```

Рис. 2.10: Запуск программы lab8-3.asm

Модифицировал код из примера 8.3 так, чтобы программа теперь выполняла вычисление произведения значений, переданных в командной строке, что отражено на иллюстрациях [2.11] и [2.12].



```
lab8-3.asm
~/work/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, 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: Программа в файле lab8-3.asm

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-3.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-3 3 4 5
Результат: 60
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$
```

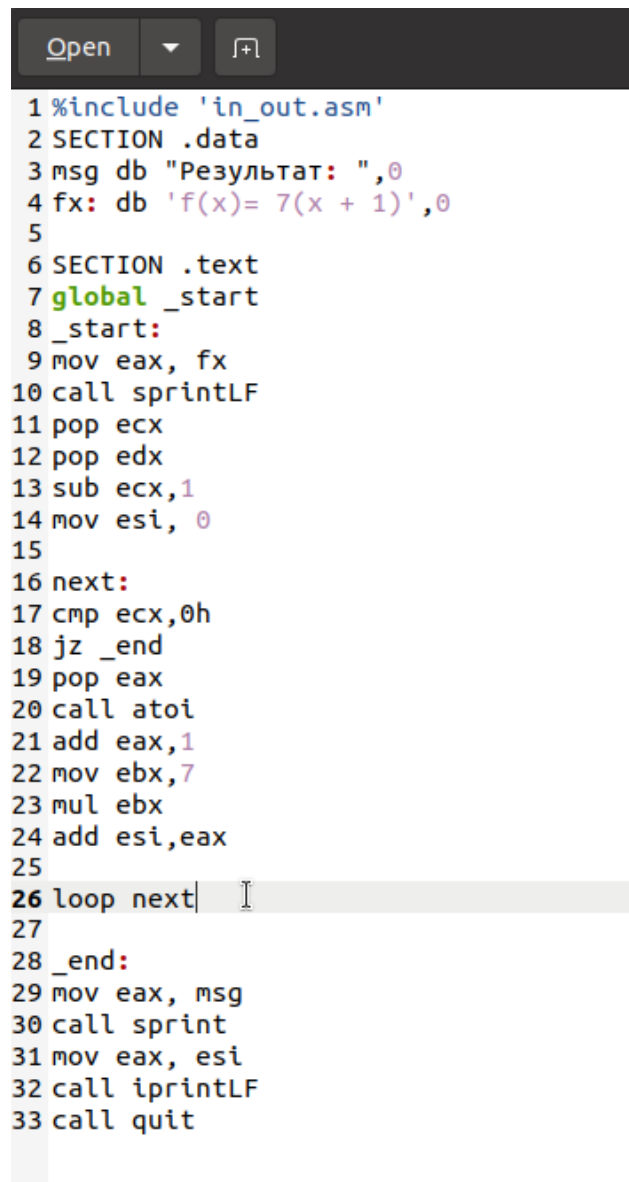
Рис. 2.12: Запуск программы lab8-3.asm

2.1 Самостоятельное задание

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])

для варианта 14

$$f(x) = 7(x + 1)$$



```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 7(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 add eax,1
22 mov ebx,7
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: Программа в файле lab8-4.asm

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(1) = 14, f(2) = 21$

Затем подал несколько аргументов и получил сумму значений функции.

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ nasm -f elf lab8-4.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ld -m elf_i386 lab8-4.o -o lab8-4
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-4 1
f(x)= 7(x + 1)
Результат: 14
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-4 2
f(x)= 7(x + 1)
Результат: 21
dskhokhlov@Ubuntu-VirtualBox:~/work/lab08$ ./lab8-4 3 6 5 7
f(x)= 7(x + 1)
Результат: 175
```

Рис. 2.14: Запуск программы lab8-4.asm

3 Выводы

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

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).