

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров

Чувакина Мария Владимировна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM.....	8
4.2	Обработка аргументов командной строки	14
4.3	Задание для самостоятельной работы	18
5	Выводы	21
6	Список литературы	22

Список иллюстраций

4.1 Создание файлов для лабораторной работы	8
4.2 Ввод текста из листинга 8.1	9
4.3 Запуск исполняемого файла.....	10
4.4 Изменение текста программы.....	11
4.5 Запуск обновленной программы	12
4.6 Изменение текста программы.....	13
4.7 Запуск исполняемого файла.....	14
4.8 Ввод текста программы из листинга 8.2.	14
4.9 Запуск исполняемого файла.....	15
4.10 Ввод текста программы из листинга 8.3.	16
4.11 Запуск исполняемого файла.....	16
4.12 Изменение текста программы.....	17
4.13 Запуск исполняемого файла.....	17
4.14 Текст программы.....	18
4.15 Запуск исполняемого файла и проверка его работы	19

Список таблиц

1 Цель работы

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. Задание 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.

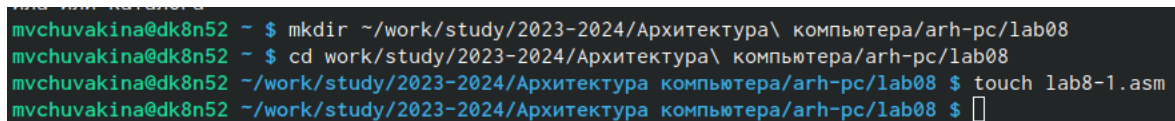
Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент 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

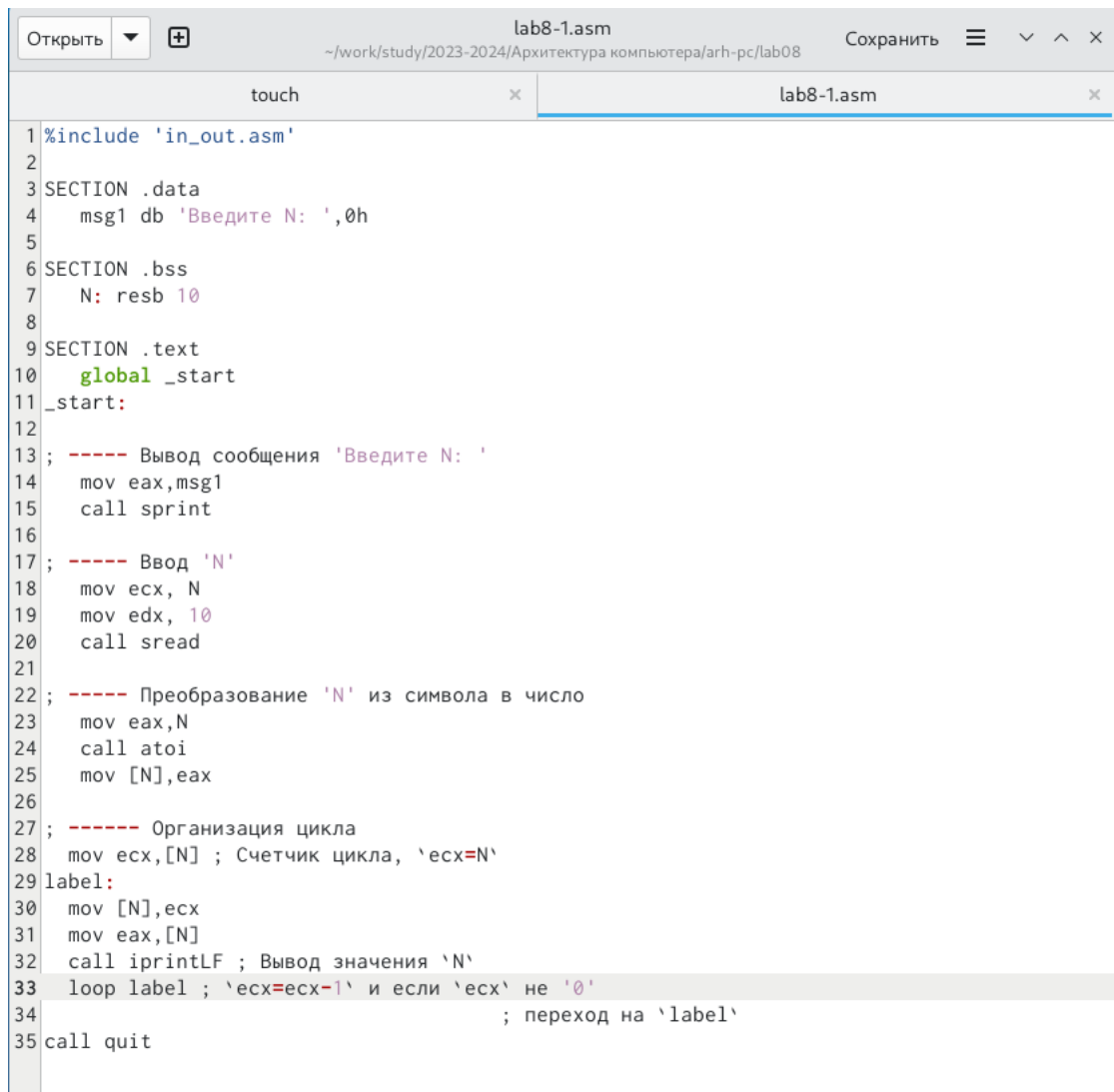
Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. 4.1).



```
mvchuvakina@dk8n52 ~$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab08
mvchuvakina@dk8n52 ~$ cd work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab08
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ touch lab8-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. 4.2).



```
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
13 ; ----- Вывод сообщения 'Введите N: '
14     mov eax,msg1
15     call sprint
16
17 ; ----- Ввод 'N'
18     mov ecx, N
19     mov edx, 10
20     call sread
21
22 ; ----- Преобразование 'N' из символа в число
23     mov eax,N
24     call atoi
25     mov [N],eax
26
27 ; ----- Организация цикла
28     mov ecx,[N] ; Счетчик цикла, `ecx=N`
29 label:
30     mov [N],ecx
31     mov eax,[N]
32     call iprintLF ; Вывод значения 'N'
33     loop label ; `ecx=ecx-1` и если `ecx` не `0`
34                                     ; переход на `label`
35 call quit
```

Рис. 4.2: Ввод текста из листинга 8.1

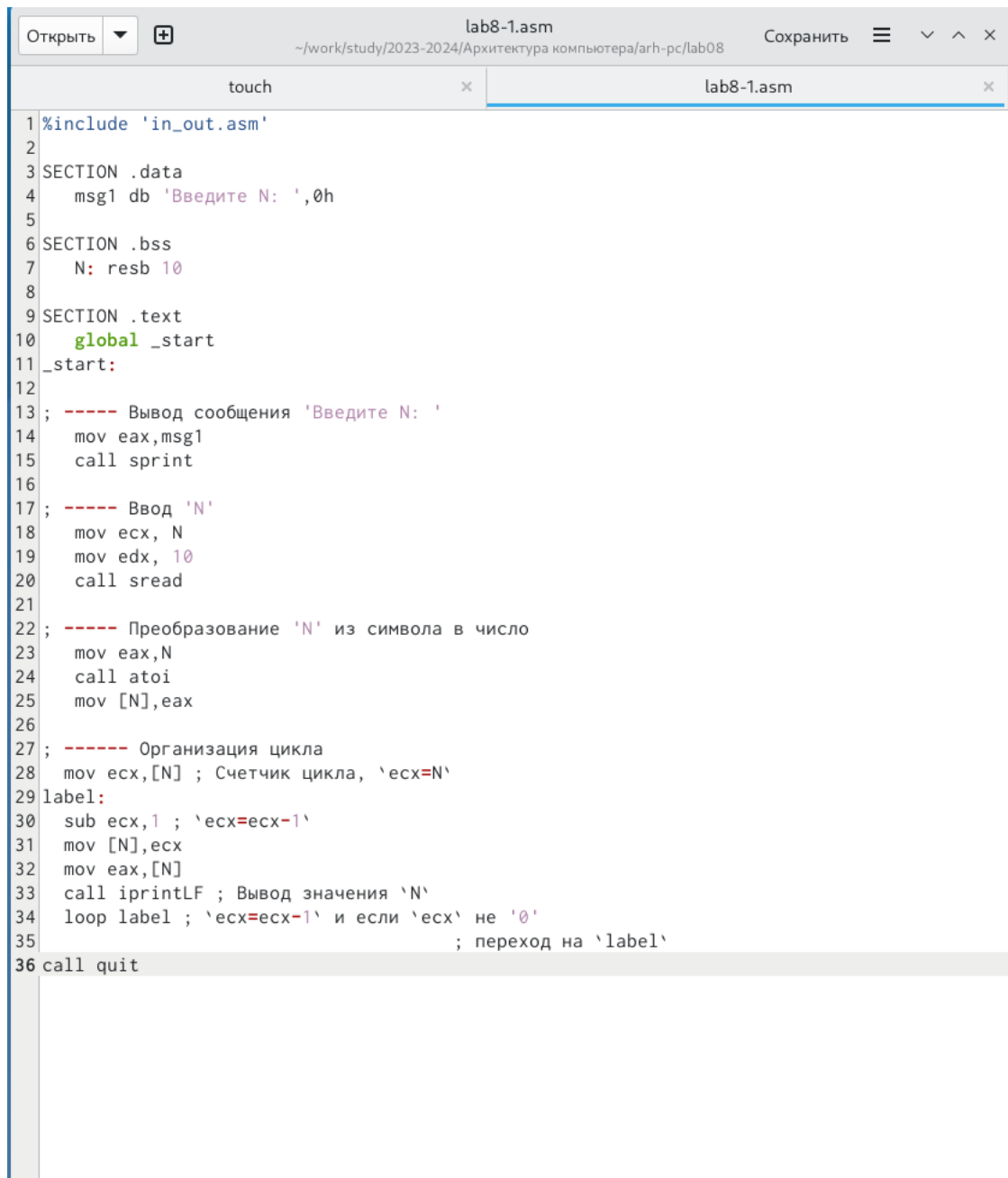
Создаю исполняемый файл и проверяю его работу. (рис. 4.3).

```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ nasm -f elf lab8-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $
```

Рис. 4.3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра есх в цикле. (рис. 4.4).



```
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
13 ; ----- Вывод сообщения 'Введите N: '
14     mov eax,msg1
15     call sprint
16
17 ; ----- Ввод 'N'
18     mov ecx, N
19     mov edx, 10
20     call sread
21
22 ; ----- Преобразование 'N' из символа в число
23     mov eax,N
24     call atoi
25     mov [N],eax
26
27 ; ----- Организация цикла
28     mov ecx,[N] ; Счетчик цикла, `ecx=N`
29 label:
30     sub ecx,1 ; `ecx=ecx-1`
31     mov [N],ecx
32     mov eax,[N]
33     call iprintLF ; Вывод значения `N`
34     loop label ; `ecx=ecx-1` и если `ecx` не `0`
35                                     ; переход на `label`
36 call quit
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.5).

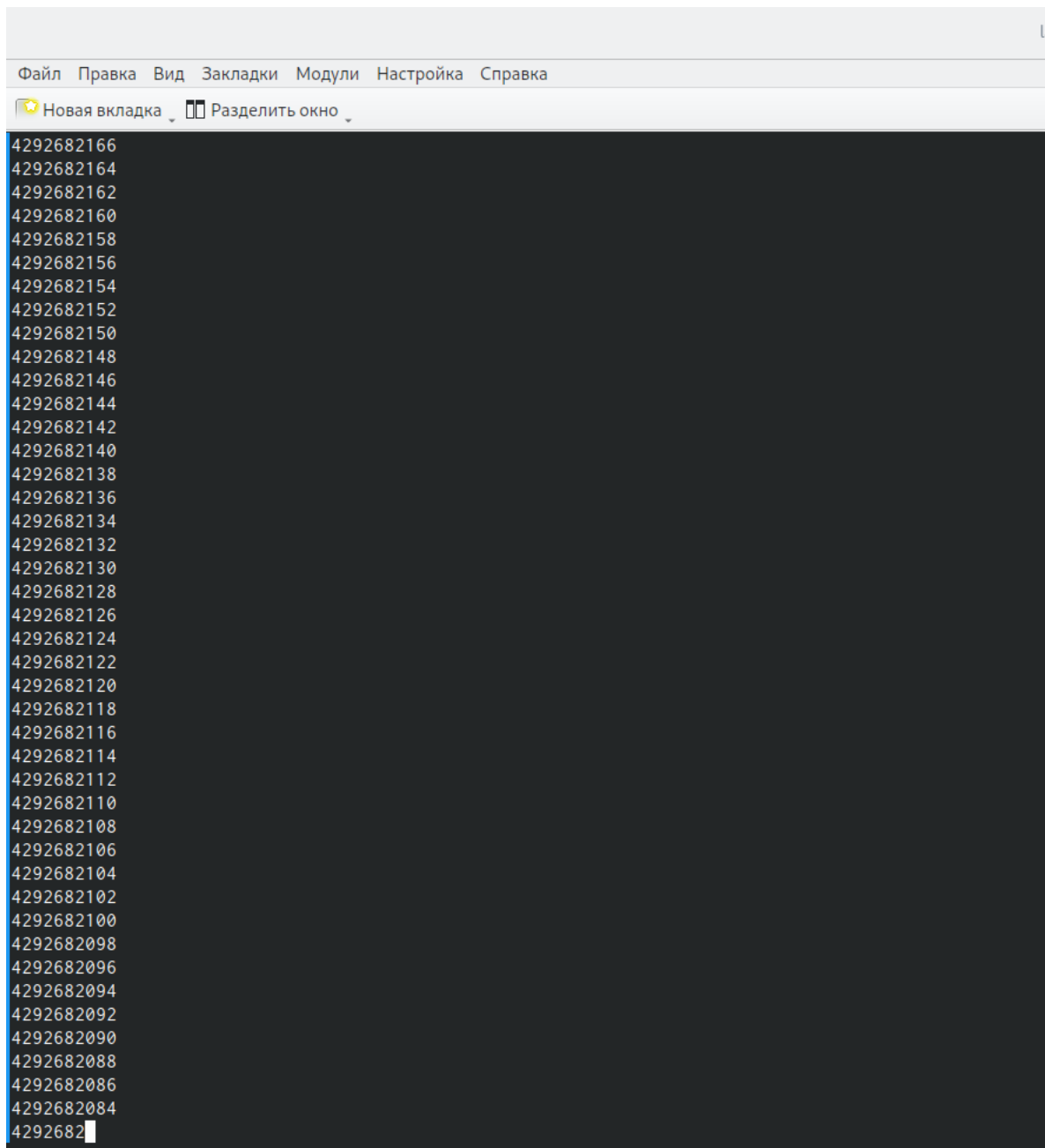


Рис. 4.5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению.

Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop`. (рис. 4.6).

```
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
13 ; ----- Вывод сообщения 'Введите N: '
14     mov eax,msg1
15     call sprint
16
17 ; ----- Ввод 'N'
18     mov ecx, N
19     mov edx, 10
20     call sread
21
22 ; ----- Преобразование 'N' из символа в число
23     mov eax,N
24     call atoi
25     mov [N],eax
26
27 ; ----- Организация цикла
28     mov ecx,[N] ; Счетчик цикла, 'ecx=N'
29 label:
30     push ecx ; добавление значения ecx в стек
31     sub ecx,1 ; 'ecx=ecx-1'
32     mov [N],ecx
33     mov eax,[N]
34     call iprintLF ; Вывод значения 'N'
35     pop ecx ; извлечение значения ecx из стека
36     loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
37                                     ; переход на 'label'
38 call quit
```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. 4.7).

```

mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ nasm -f elf lab8-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $

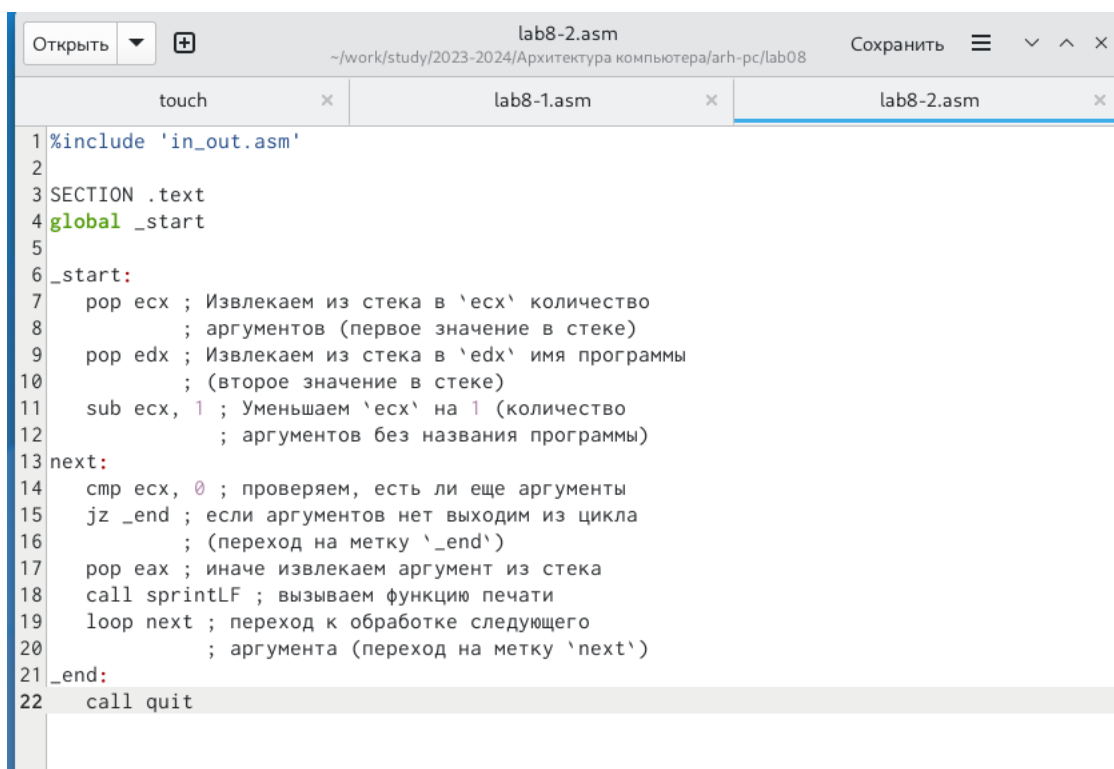
```

Рис. 4.7: Запуск исполняемого файла

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

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arh-pc/lab08 и ввожу в него текст программы из листинга 8.2. (рис. 4.8).



```

1 %include 'in_out.asm'
2
3 SECTION .text
4 global _start
5
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 next:
14     cmp ecx, 0 ; проверяем, есть ли еще аргументы
15     jz _end ; если аргументов нет выходим из цикла
16             ; (переход на метку '_end')
17     pop eax ; иначе извлекаем аргумент из стека
18     call sprintf ; вызываем функцию печати
19     loop next ; переход к обработке следующего
20              ; аргумента (переход на метку 'next')
21 _end:
22     call quit

```

Рис. 4.8: Ввод текста программы из листинга 8.2

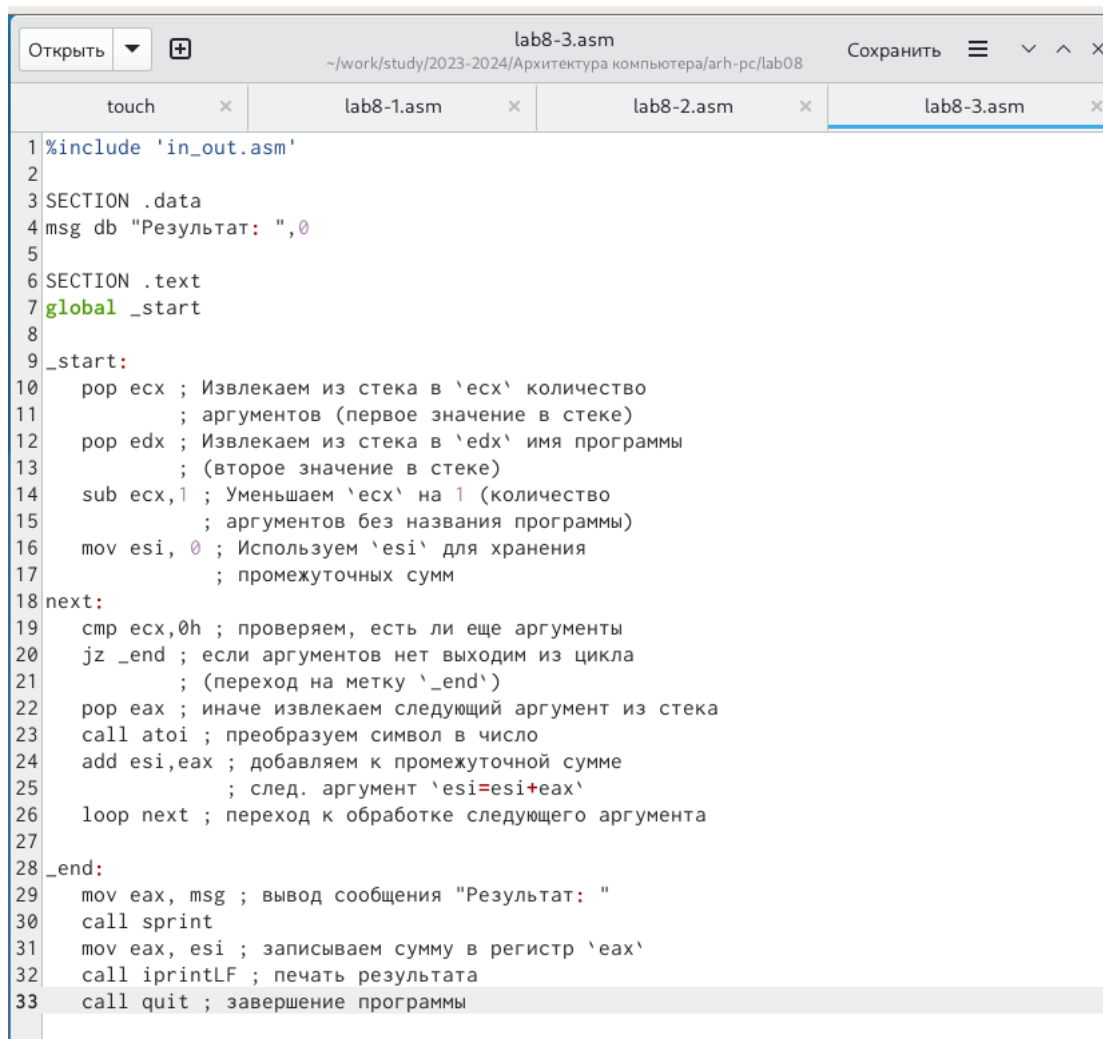
Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. 4.9).

```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ touch lab8-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ gedit lab8-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ nasm -f elf lab8-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08 $
```

Рис. 4.9: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа считывает “2” как отдельный аргумент.

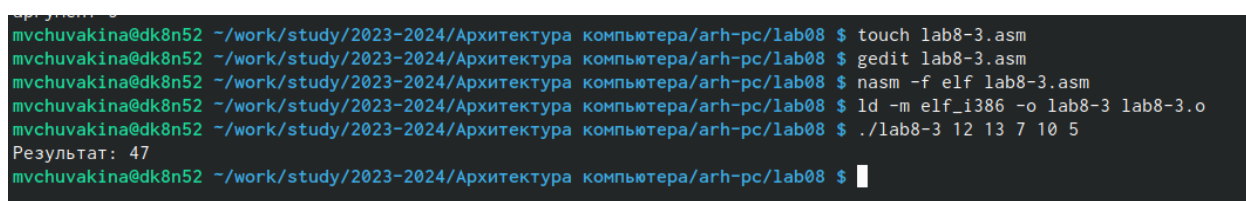
Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/arh-pc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. 4.10).



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

Рис. 4.10: Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.11).



```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ touch lab8-3.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ gedit lab8-3.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ nasm -f elf lab8-3.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $
```

Рис. 4.11: Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 4.12).

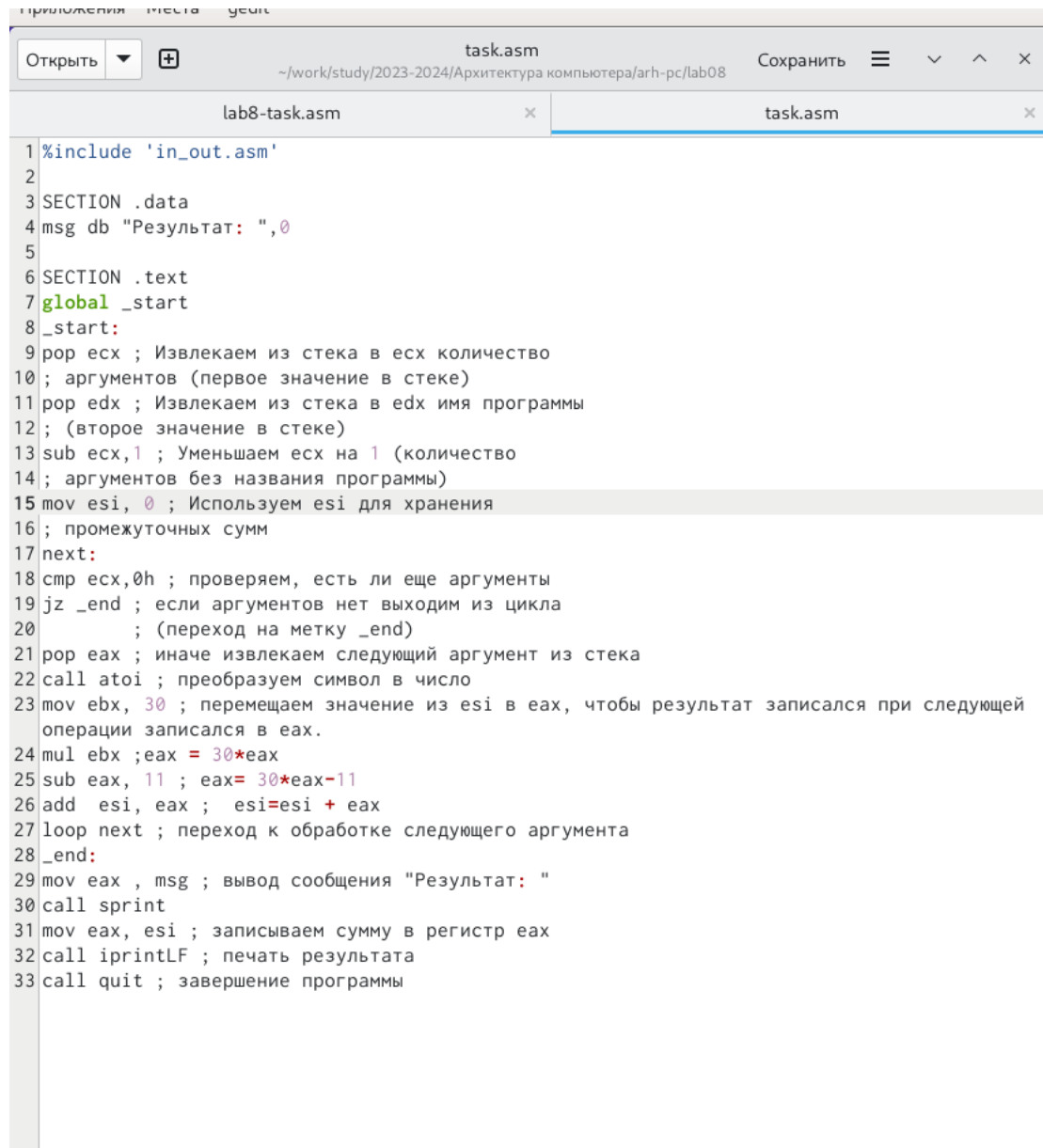
Рис. 4.12: Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 4.13).

Рис. 4.13: Запуск исполняемого файла

4.3 Задание для самостоятельной работы

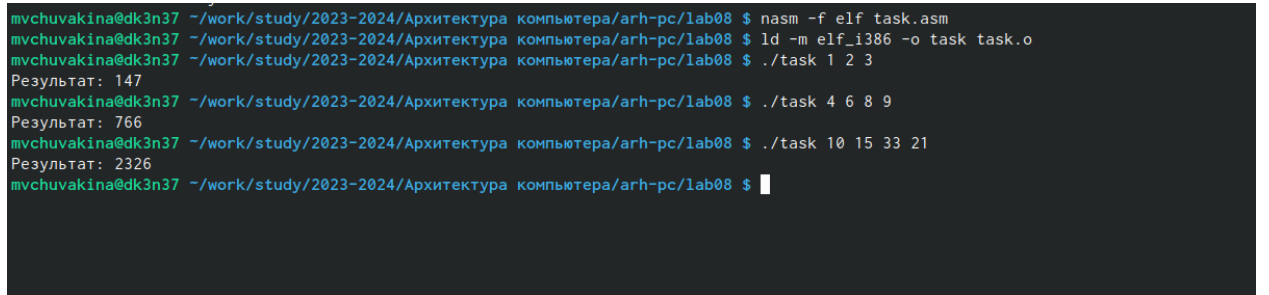
Пишу текст программы, которая находит сумму значений функции $f(x) = 30 \cdot x - 11$ в соответствии с моим номером варианта (16) для $x = x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы. (рис. 4.14).



```
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 pop edx ; Извлекаем из стека в edx имя программы
12 ; (второе значение в стеке)
13 sub ecx,1 ; Уменьшаем ecx на 1 (количество
14 ; аргументов без названия программы)
15 mov esi, 0 ; Используем esi для хранения
16 ; промежуточных сумм
17 next:
18 cmp ecx,0h ; проверяем, есть ли еще аргументы
19 jz _end ; если аргументов нет выходим из цикла
20 ; (переход на метку _end)
21 pop eax ; иначе извлекаем следующий аргумент из стека
22 call atoi ; преобразуем символ в число
23 mov ebx, 30 ; перемещаем значение из esi в ebx, чтобы результат записался при следующей
24 ; операции записался в ebx.
25 mul ebx ; ebx = 30*ebx
26 sub ebx, 11 ; ebx = 30*ebx - 11
27 add esi, ebx ; esi = esi + ebx
28 loop next ; переход к обработке следующего аргумента
29 _end:
30 mov eax, msg ; вывод сообщения "Результат: "
31 call sprint
32 mov eax, esi ; записываем сумму в регистр eax
33 call iprintLF ; печать результата
34 call quit ; завершение программы
```

Рис. 4.14: Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. (рис. 4.15).



```
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ nasm -f elf task.asm
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ld -m elf_i386 -o task task.o
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ./task 1 2 3
Результат: 147
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ./task 4 6 8 9
Результат: 766
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $ ./task 10 15 33 21
Результат: 2326
mvchuvakina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/argh-pc/lab08 $
```

Рис. 4.15: Запуск исполняемого файла и проверка его работы

Программа работает корректно. Текст программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx ; Извлекаем из стека в ecx количество
```

```
; аргументов (первое значение в стеке)
```

```
pop edx ; Извлекаем из стека в edx имя программы
```

```
; (второе значение в стеке)
```

```
sub ecx,1 ; Уменьшаем ecx на 1 (количество
```

```
; аргументов без названия программы)
```

mov esi, 0 ; Используем esi для хранения

; промежуточных сумм

next:

cmp ecx, 0h ; проверяем, есть ли еще аргументы

jz _end ; если аргументов нет выходим из цикла

 ; (переход на метку _end)

pop eax ; иначе извлекаем следующий аргумент из стека

call atoi ; преобразуем символ в число

mov ebx, 30 ; перемещаем значение из esi в eax, чтобы результат записался при
следующей операции записался в eax.

mul ebx ; eax = 30*eax

sub eax, 11 ; eax= 30*eax-11

add esi, eax ; esi=esi + eax

loop next ; переход к обработке следующего аргумента

_end:

mov eax, msg ; вывод сообщения "Результат: "

call sprint

mov eax, esi ; записываем сумму в регистр eax

call iprintLF ; печать результата

call quit ; завершение программы

5 Выводы

Благодаря данной лабораторной работе я приобрела навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ.

6 Список литературы

1. GDB:TheGNUProjectDebugger.—URL:<https://www.gnu.org/software/gdb/>.
2. GNUBashManual.—2016.—URL:<https://www.gnu.org/software/bash/manual/>.
3. MidnightCommanderDevelopmentCenter.—2021.—URL:<https://midnight-commander.org/>.
4. NASMAssemblyLanguageTutorials.—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/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. TheNASMdocumentation.—2021.—URL:<https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. КолдаевВ.Д.,ЛупинС.А.АрхитектураЭВМ.—М.:Форум,2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. НовожиловО.П.АрхитектураЭВМисистем.—М.:Юрайт,2016.
12. Расширенныйассемблер:NASM.—2021.—
URL:<https://www.opennet.ru/docs/RUS/nasm/>.
13. РобачевскийА.,НемнюгинС.,СтесикО.ОперационнаясистемаUNIX.—2-е изд. — БХВПетербург, 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.

20

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).