

Отчёт по лабораторной работе 5

Архитектура компьютера

Нилова Кристина Артуровна

Содержание

1	Цель работы	5
2	Задания	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Основы работы в Midnight Commander	8
4.2	Подключение внешнего файла in_out.asm	12
4.3	Задание для самостоятельной работы	14
5	Выводы	18

Список иллюстраций

4.1	Создание каталога	8
4.2	Создание файла lab05-1.asm	9
4.3	Заполнение файла lab05-1.asm	10
4.4	Просмотр файла lab05-1.asm	11
4.5	Компиляция текста программы lab05-1.asm	11
4.6	Копирование файла	12
4.7	Заполнение файла lab05-2.asm	13
4.8	Компиляция текста программы lab05-2.asm	13
4.9	Заполнение файла lab05-2.asm	14
4.10	Компиляция текста программы lab05-2.asm	14
4.11	Заполнение файла lab05-3.asm	15
4.12	Компиляция текста программы lab05-3.asm	15
4.13	Заполнение файла lab05-4.asm	16
4.14	Компиляция текста программы lab05-4.asm	16

Список таблиц

1 Цель работы

Целью работы является приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера `mov` и `int`.

2 Задания

1. Изучение Midnight Commander
2. Примеры программ с использованием внешнего файла `in_out.asm`
3. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

Midnight Commander (или просто mc) — это программа, которая позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой, т.е. mc является файловым менеджером. Midnight Commander позволяет сделать работу с файлами более удобной и наглядной.

Программа на языке ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (SECTION .text), секция инициированных (известных во время компиляции) данных (SECTION .data) и секция неинициализированных данных (тех, под которые во время компиляции только отводится память, а значение присваивается в ходе выполнения программы) (SECTION .bss).

Инструкция языка ассемблера mov предназначена для дублирования данных источника в приёмнике. В общем виде эта инструкция записывается в виде `mov dst,src` Здесь операнд `dst` — приёмник, а `src` — источник

Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером. В общем виде она записывается в виде `int n` Здесь `n` — номер прерывания, принадлежащий диапазону 0–255

4 Выполнение лабораторной работы

4.1 Основы работы в Midnight Commander

Я открыла Midnight Commander и перешла в каталог `~/work/arch-рс`. Затем я создала новый каталог под названием `lab05`. рис. [4.1]

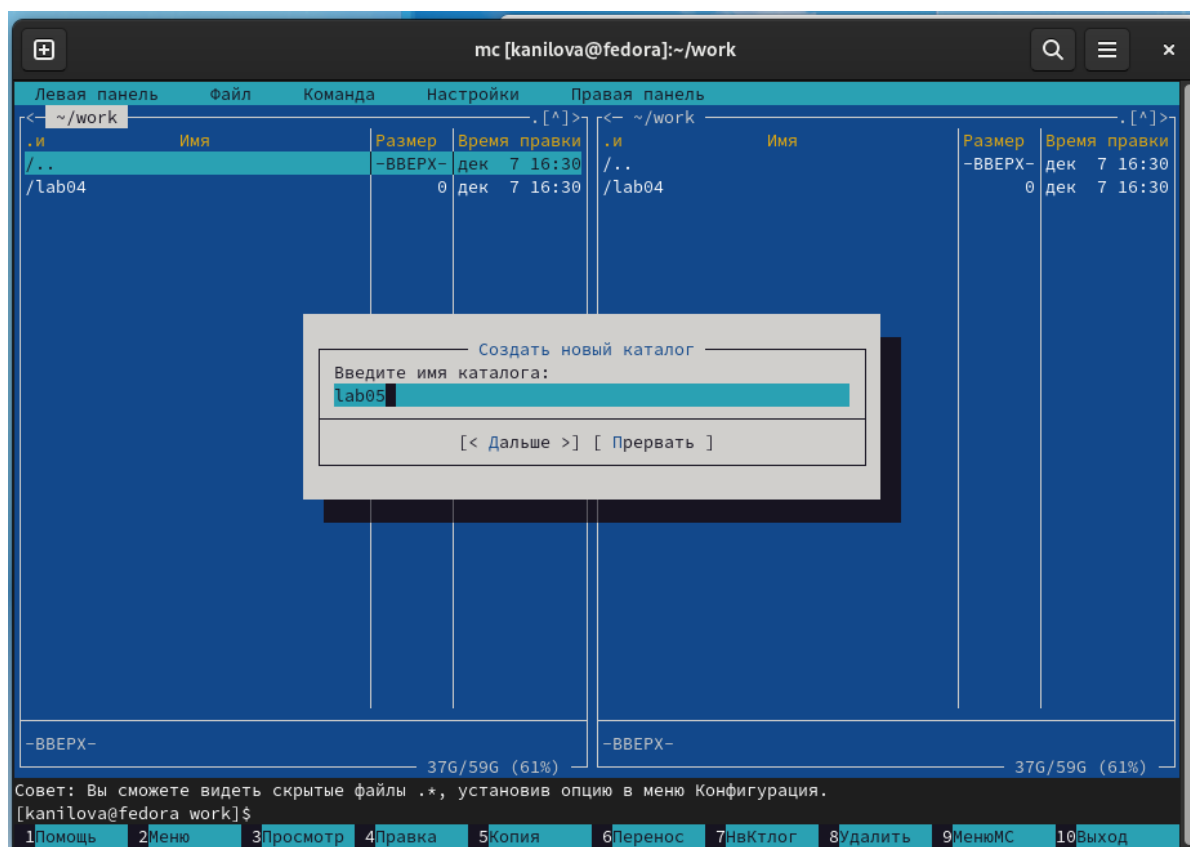


Рис. 4.1: Создание каталога

Внутри каталога `lab05` я создала файл с именем `lab05-1.asm`. рис. [4.2]

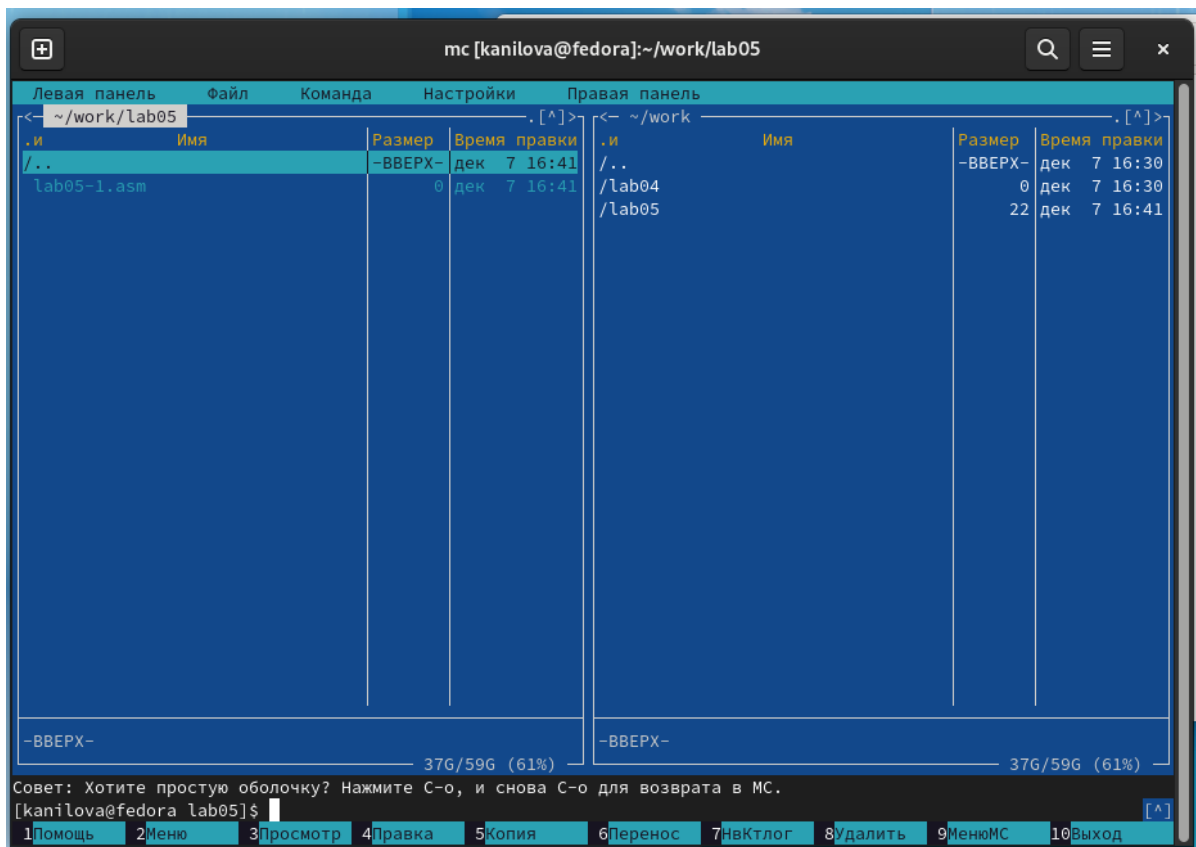
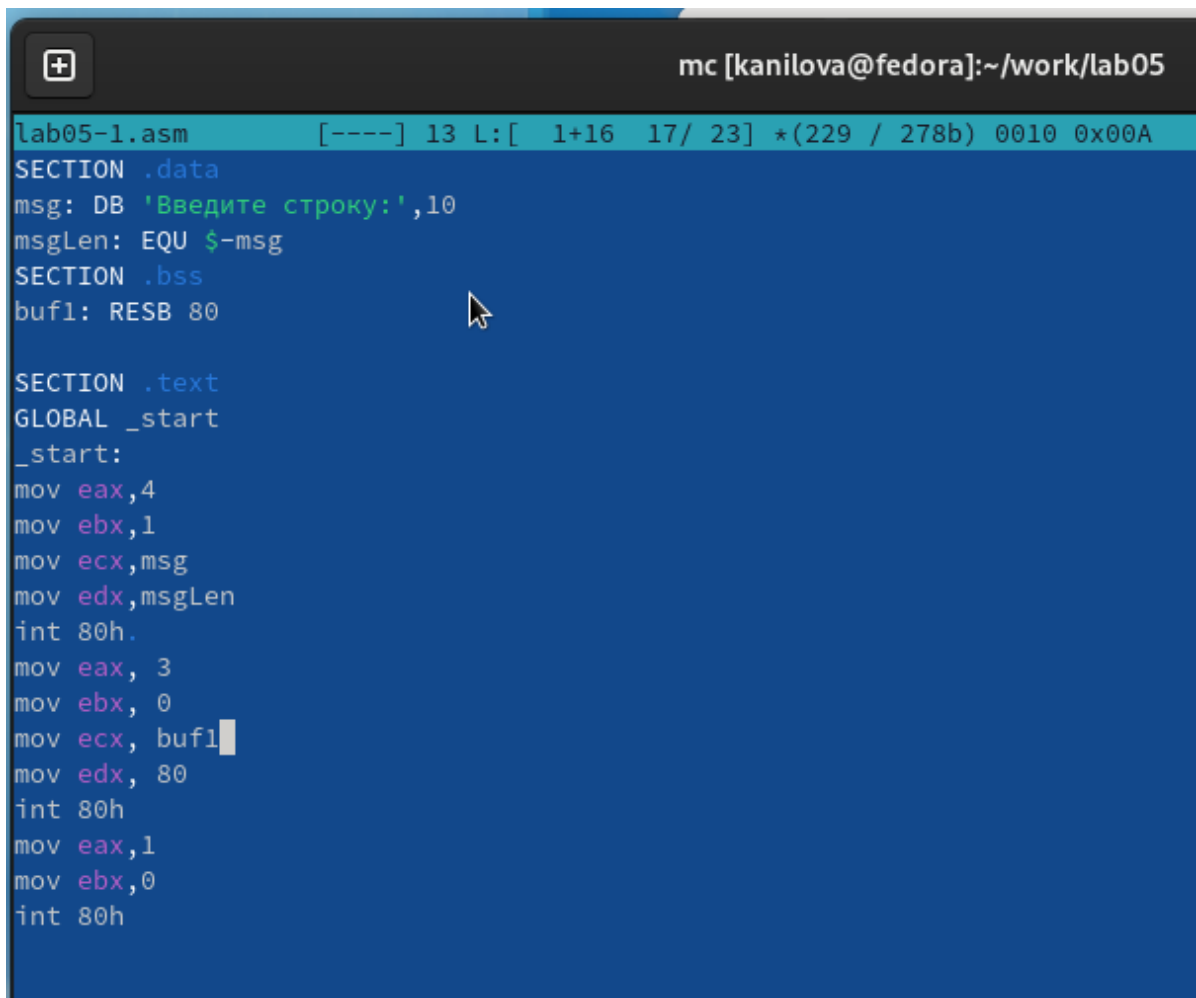


Рис. 4.2: Создание файла lab05-1.asm

Я открыла файл lab05-1.asm для редактирования и написала в нем код программы. рис. [4.3]

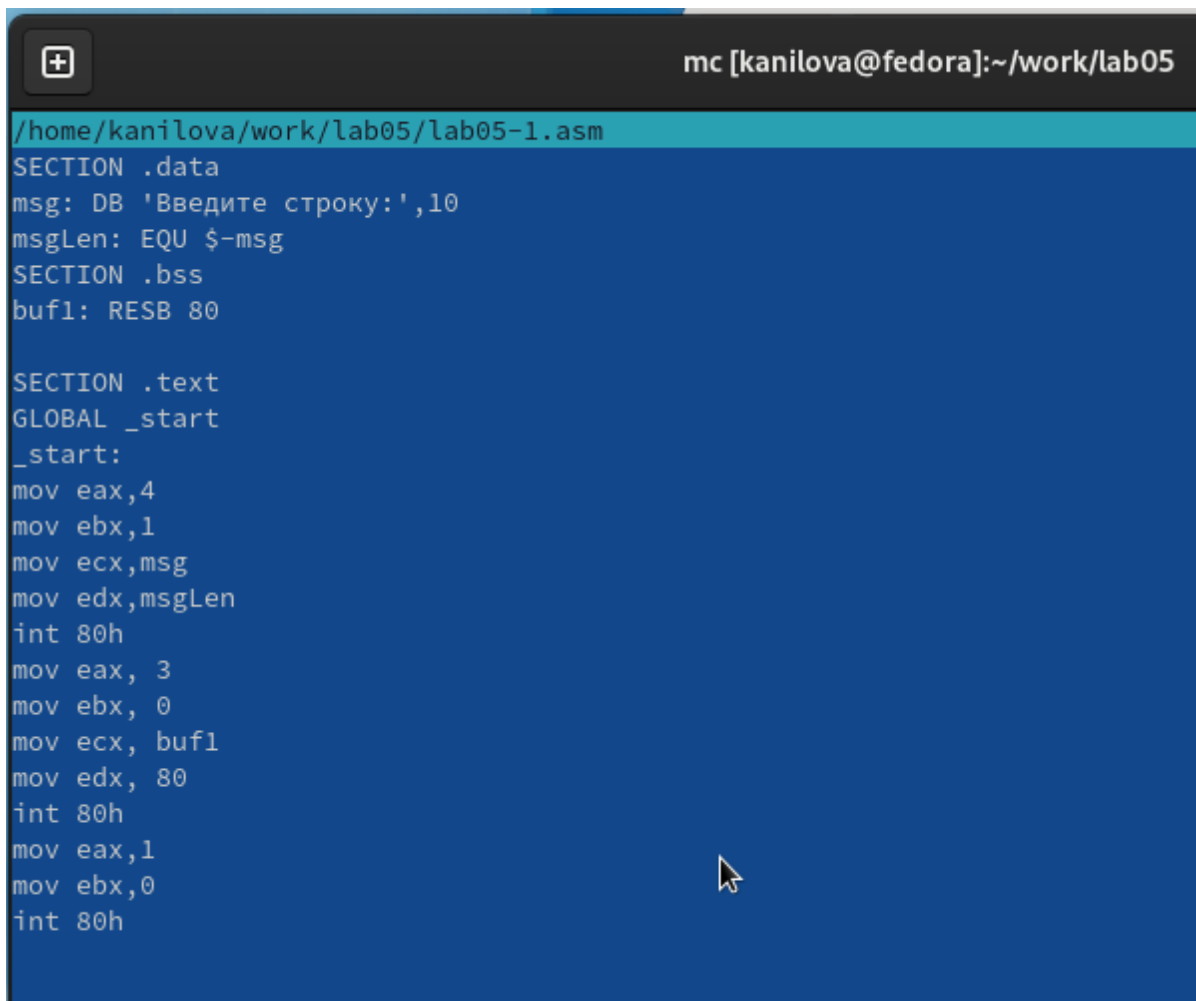


```
mc [kanilova@fedora]:~/work/lab05
lab05-1.asm [----] 13 L: [ 1+16 17/ 23] *(229 / 278b) 0010 0x00A
SECTION .data
msg: DB 'Введите строку:',10
msgLen: EQU $-msg
SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:
mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h
mov eax, 3
mov ebx, 0
mov ecx, buf1
mov edx, 80
int 80h
mov eax,1
mov ebx,0
int 80h
```

Рис. 4.3: Заполнение файла lab05-1.asm

После этого я открыла файл для просмотра и убедилась, что он содержит написанный мной код. рис. [4.4]

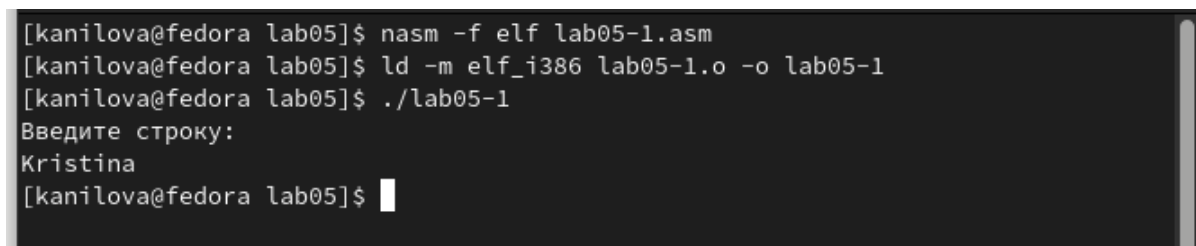


```
mc [kanilova@fedora]:~/work/lab05
/home/kanilova/work/lab05/lab05-1.asm
SECTION .data
msg: DB 'Введите строку:',10
msgLen: EQU $-msg
SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:
mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h
mov eax, 3
mov ebx, 0
mov ecx, buf1
mov edx, 80
int 80h
mov eax,1
mov ebx,0
int 80h
```

Рис. 4.4: Просмотр файла lab05-1.asm

Затем я скомпилировала программу и проверила ее работу, получив исполняемый файл. рис. [4.5]



```
[kanilova@fedora lab05]$ nasm -f elf lab05-1.asm
[kanilova@fedora lab05]$ ld -m elf_i386 lab05-1.o -o lab05-1
[kanilova@fedora lab05]$ ./lab05-1
Введите строку:
Kristina
[kanilova@fedora lab05]$
```

Рис. 4.5: Компиляция текста программы lab05-1.asm

4.2 Подключение внешнего файла in_out.asm

Далее, я скачала файл с именем in_out.asm и добавила его в рабочий каталог. Затем я скопировала файл lab05-1.asm и создала копию с именем lab05-2.asm. рис. [4.6]

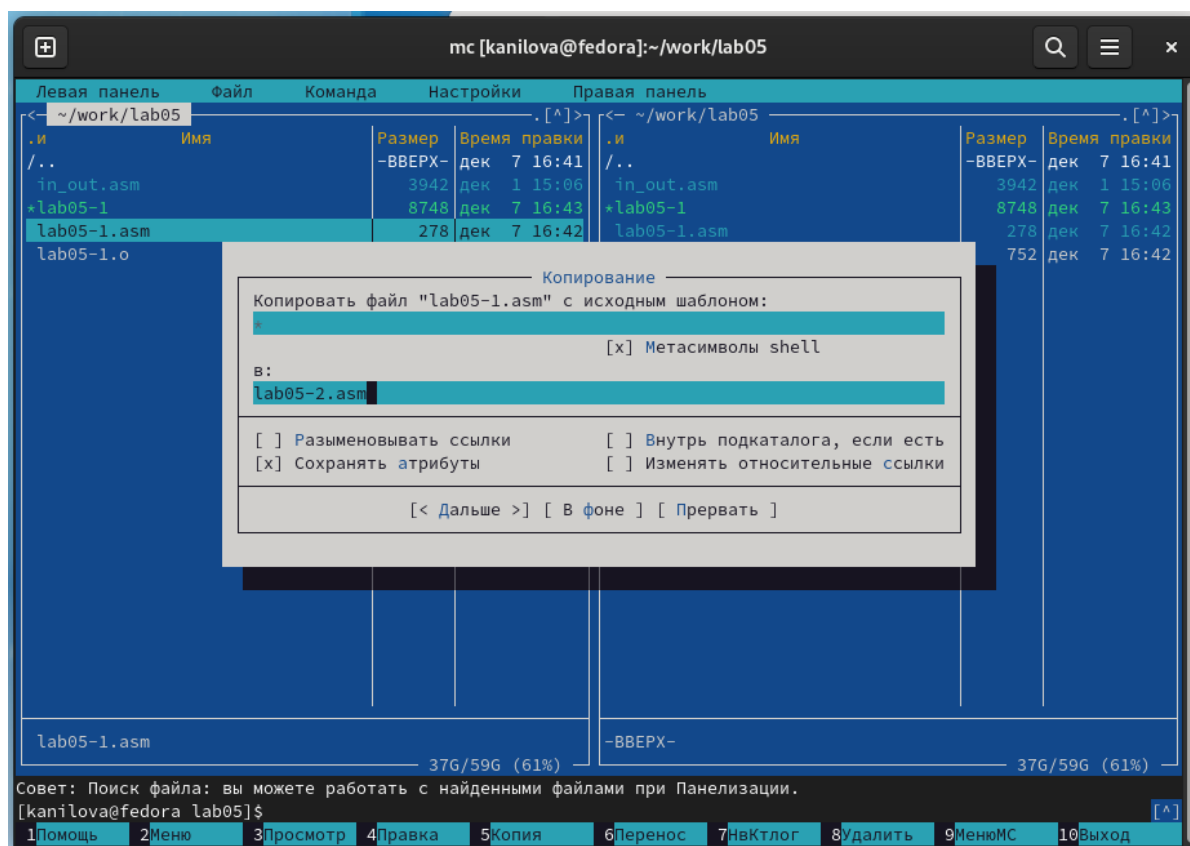
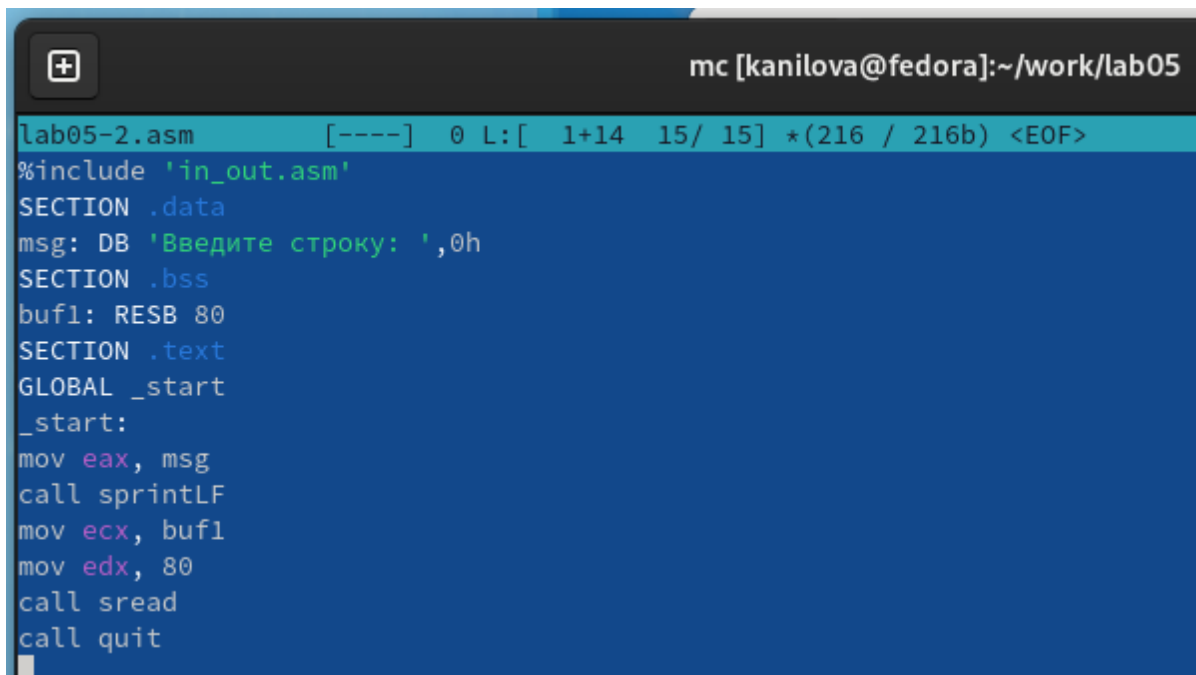


Рис. 4.6: Копирование файла

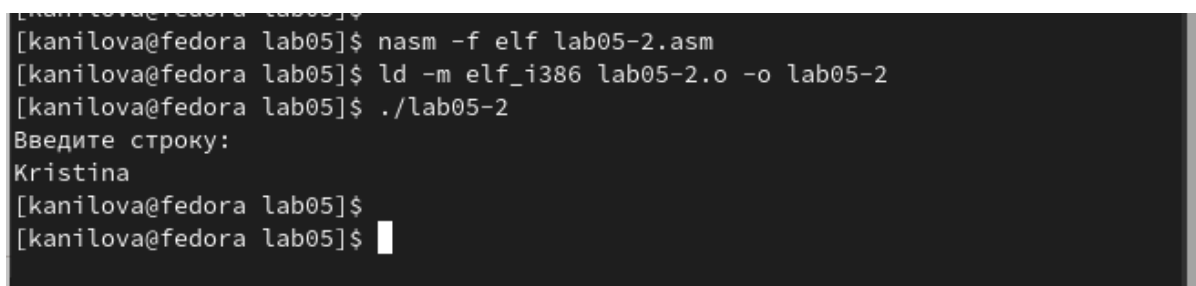
Я написала код программы в файле lab05-2.asm. рис. [4.7]



```
mc [kanilova@fedora]:~/work/lab05
lab05-2.asm [----] 0 L: [ 1+14 15/ 15] *(216 / 216b) <EOF>
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите строку: ',0h
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF
mov ecx, buf1
mov edx, 80
call sread
call quit
```

Рис. 4.7: Заполнение файла lab05-2.asm

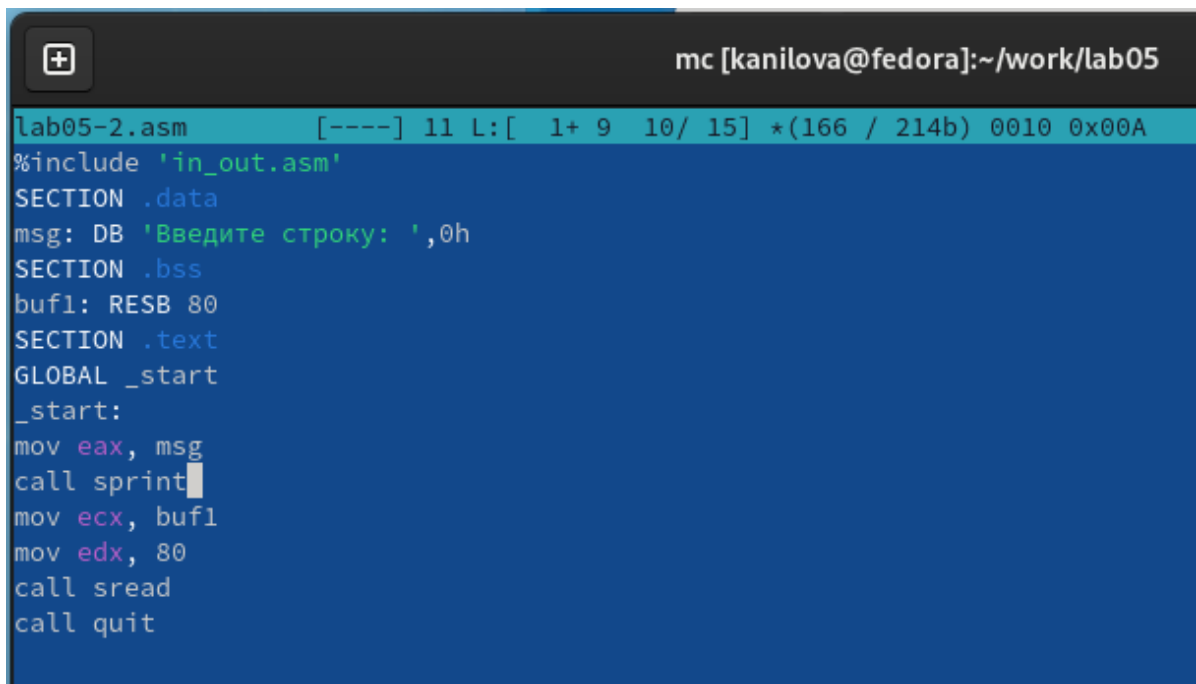
После этого я скомпилировала программу и проверила ее запуск. рис. [4.8]



```
[kanilova@fedora lab05]$ nasm -f elf lab05-2.asm
[kanilova@fedora lab05]$ ld -m elf_i386 lab05-2.o -o lab05-2
[kanilova@fedora lab05]$ ./lab05-2
Введите строку:
Kristina
[kanilova@fedora lab05]$
[kanilova@fedora lab05]$
```

Рис. 4.8: Компиляция текста программы lab05-2.asm

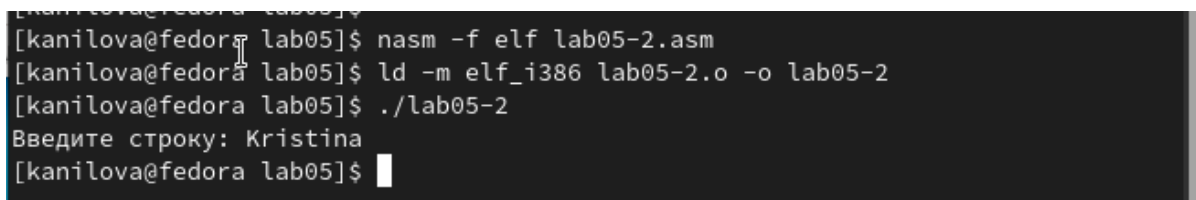
В файле lab05-2.asm я внесла изменения, заменив подпрограмму sprintLF на sprint. Это позволило строке вывода не завершаться символом перехода на новую строку. рис. [4.9].



```
mc [kanilova@fedora]:~/work/lab05
lab05-2.asm [-----] 11 L: [ 1+ 9 10/ 15] *(166 / 214b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите строку: ',0h
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, buf1
mov edx, 80
call sread
call quit
```

Рис. 4.9: Заполнение файла lab05-2.asm

Затем я снова собрала исполняемый файл. Теперь после вывода строки она не будет завершаться символом перехода на новую строку. рис. [4.10].

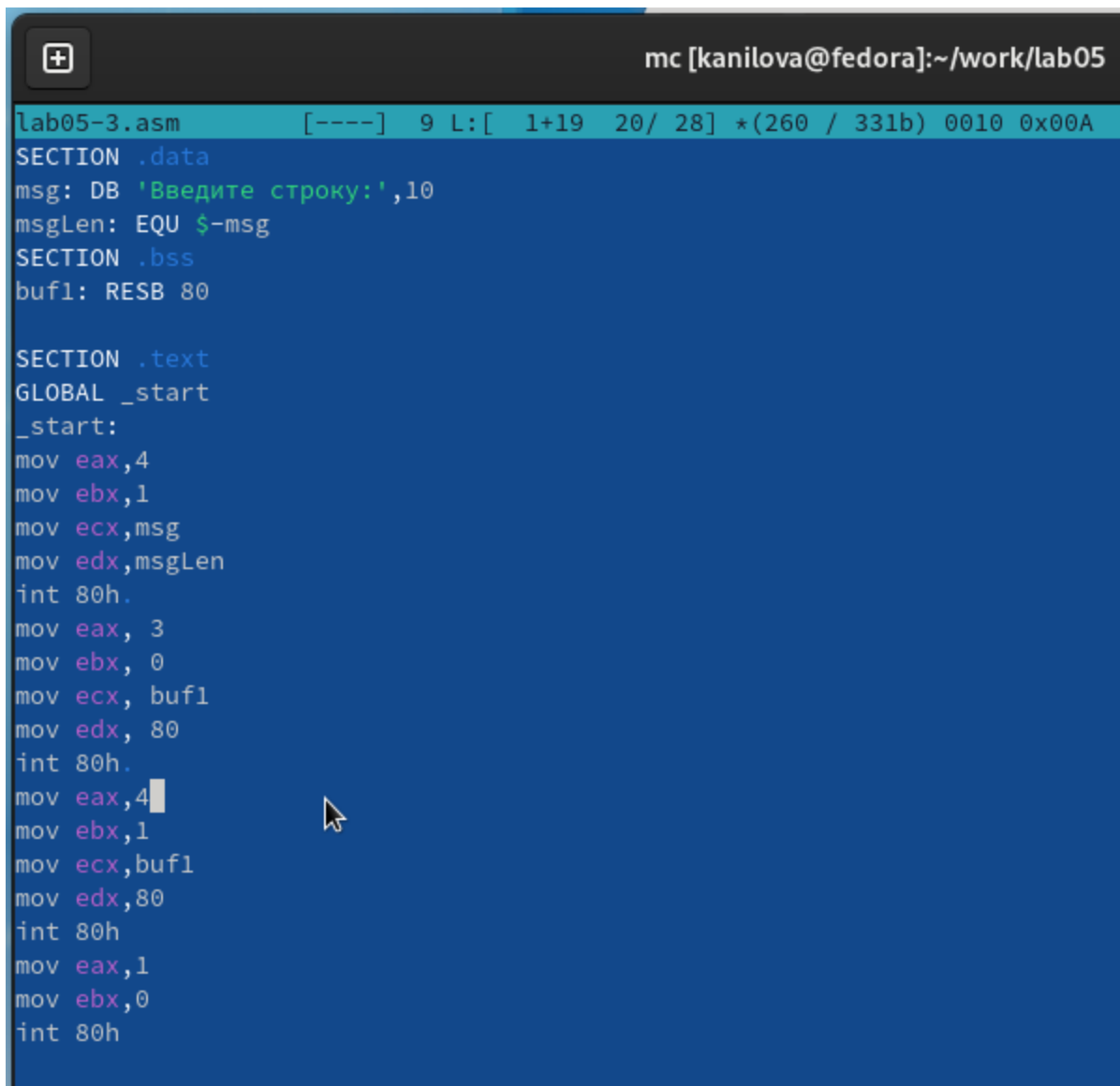


```
[kanilova@fedora lab05]$ nasm -f elf lab05-2.asm
[kanilova@fedora lab05]$ ld -m elf_i386 lab05-2.o -o lab05-2
[kanilova@fedora lab05]$ ./lab05-2
Введите строку: Kristina
[kanilova@fedora lab05]$
```

Рис. 4.10: Компиляция текста программы lab05-2.asm

4.3 Задание для самостоятельной работы

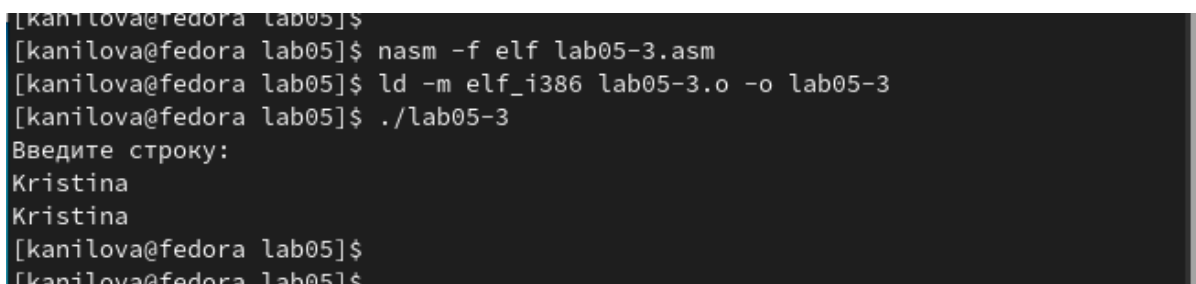
Я решила скопировать программу lab05-1.asm и внести изменения в код, чтобы программа выводила приглашение вроде “Введите строку:”, затем считывала строку с клавиатуры и выводила ее на экран. рис. [-4.11][4.12]



```
mc [kanilova@fedora]:~/work/lab05
lab05-3.asm [----] 9 L: [ 1+19 20/ 28] *(260 / 331b) 0010 0x00A
SECTION .data
msg: DB 'Введите строку:',10
msgLen: EQU $-msg
SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:
mov eax,4
mov ebx,1
mov ecx,msg
mov edx,msgLen
int 80h.
mov eax, 3
mov ebx, 0
mov ecx, buf1
mov edx, 80
int 80h.
mov eax,4
mov ebx,1
mov ecx,buf1
mov edx,80
int 80h
mov eax,1
mov ebx,0
int 80h
```

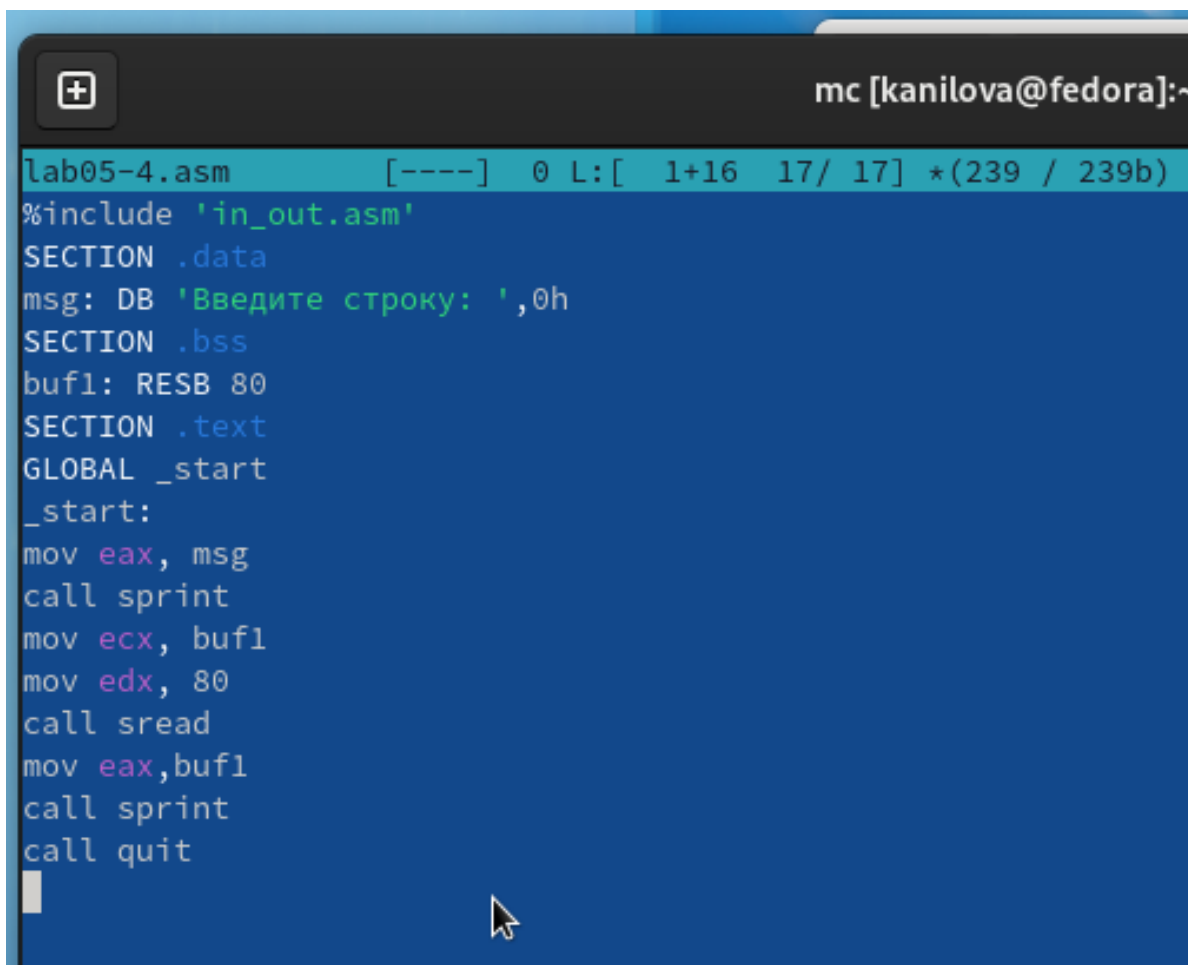
Рис. 4.11: Заполнение файла lab05-3.asm



```
[kanilova@fedora lab05]$
[kanilova@fedora lab05]$ nasm -f elf lab05-3.asm
[kanilova@fedora lab05]$ ld -m elf_i386 lab05-3.o -o lab05-3
[kanilova@fedora lab05]$ ./lab05-3
Введите строку:
Kristina
Kristina
[kanilova@fedora lab05]$
[kanilova@fedora lab05]$
```

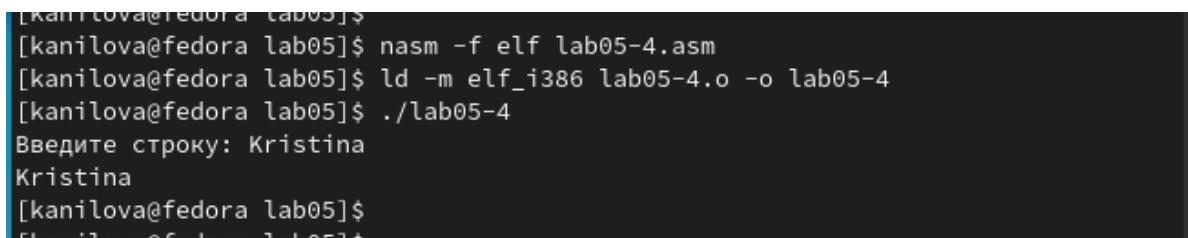
Рис. 4.12: Компиляция текста программы lab05-3.asm

Также я скопировала программу lab05-2.asm и внесла соответствующие изменения в код, чтобы программа тоже выводила приглашение вроде “Введите строку:”, считывала строку с клавиатуры и выводила ее на экран. рис. [-4.13][4.14]



```
lab05-4.asm [----] 0 L: [ 1+16 17/ 17] *(239 / 239b) .
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите строку: ',0h
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, buf1
mov edx, 80
call sread
mov eax, buf1
call sprint
call quit
```

Рис. 4.13: Заполнение файла lab05-4.asm



```
[kanilova@fedora lab05]$
[kanilova@fedora lab05]$ nasm -f elf lab05-4.asm
[kanilova@fedora lab05]$ ld -m elf_i386 lab05-4.o -o lab05-4
[kanilova@fedora lab05]$ ./lab05-4
Введите строку: Kristina
Kristina
[kanilova@fedora lab05]$
[kanilova@fedora lab05]$
```

Рис. 4.14: Компиляция текста программы lab05-4.asm

Отличие между этими двумя реализациями заключается в том, что файл `in_out.asm` уже содержит готовые подпрограммы для обеспечения ввода/вывода. Таким образом, нам остается только разместить данные в нужных регистрах и вызвать нужную подпрограмму с помощью инструкции `call`. Это упрощает кодирование и обеспечивает более гибкую работу с вводом и выводом данных.

5 Выводы

Научились писать базовые ассемблерные программы. Освоили ассемблерные инструкции `mov` и `int`.