Лабораторная работа №5

Дисциплина: архитектура компьютера

Ларина Наталья Денисовна

Содержание

[1 Цель работы 1](#_Toc150424815)

[2 Задание 1](#_Toc150424816)

[3 Теоретическое введение 1](#_Toc150424817)

[4 Выполнение лабораторной работы 2](#_Toc150424818)

[4.1 Основы работы в mc 2](#_Toc150424819)

[4.2 Структура программы на языке ассемблера NASM 5](#_Toc150424820)

[4.3 Подключение внешнего файла 6](#_Toc150424821)

[4.4 Выполнение заданий для самостоятельной работы 9](#_Toc150424822)

[5 Выводы 12](#_Toc150424823)

[6 Список литературы 12](#_Toc150424824)

# 1 Цель работы

Приобретение практических навыков работы в Midnight Commander, освоение инструкций языка ассемблера mov и int.

# 2 Задание

1. Основы работы с mc
2. Структура программы на языке ассемблера NASM
3. Подключение внешнего файла
4. 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). Для объявления инициированных данных в секции .data используются директивы DB, DW, DD, DQ и DT, которые резервируют память и указывают, какие значения должны храниться в этой памяти: - DB (define byte) — определяет переменную размером в 1 байт; - DW (define word) — определяет переменную размеров в 2 байта (слово); - DD (define double word) — определяет переменную размером в 4 байта (двойное слово); - DQ (define quad word) — определяет переменную размером в 8 байт (учетве- рённое слово); - DT (define ten bytes) — определяет переменную размером в 10 байт. Директивы используются для объявления простых переменных и для объявления массивов. Для определения строк принято использовать директиву DB в связи с особенностями хранения данных в оперативной памяти. Инструкция языка ассемблера mov предназначена для дублирования данных источника в приёмнике.

mov dst,src

Здесь операнд dst — приёмник, а src — источник. В качестве операнда могут выступать регистры (register), ячейки памяти (memory) и непосредственные значения (const). Инструкция языка ассемблера intпредназначена для вызова прерывания с указанным номером.

int n

Здесь n — номер прерывания, принадлежащий диапазону 0–255. При программировании в Linux с использованием вызовов ядра sys\_calls n=80h (принято задавать в шестнадцатеричной системе счисления).

# 4 Выполнение лабораторной работы

## 4.1 Основы работы в mc

Открываю Midnight Commander и ввожу в терминал mc (рис. [1](#fig:001)).

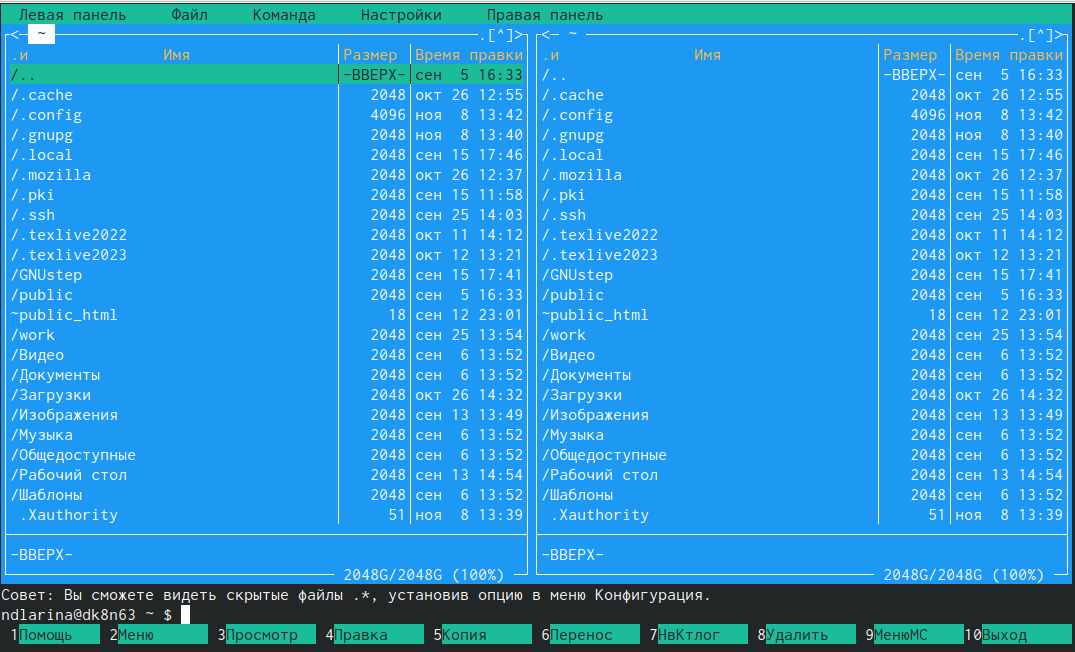


Рис. 1: Открытый mc

Далее перехожу в каталог ~/work/study/2022-2023/Архитектура Компьютера/arch-pc, используя файловый менеджер mc (рис. [2](#fig:002))

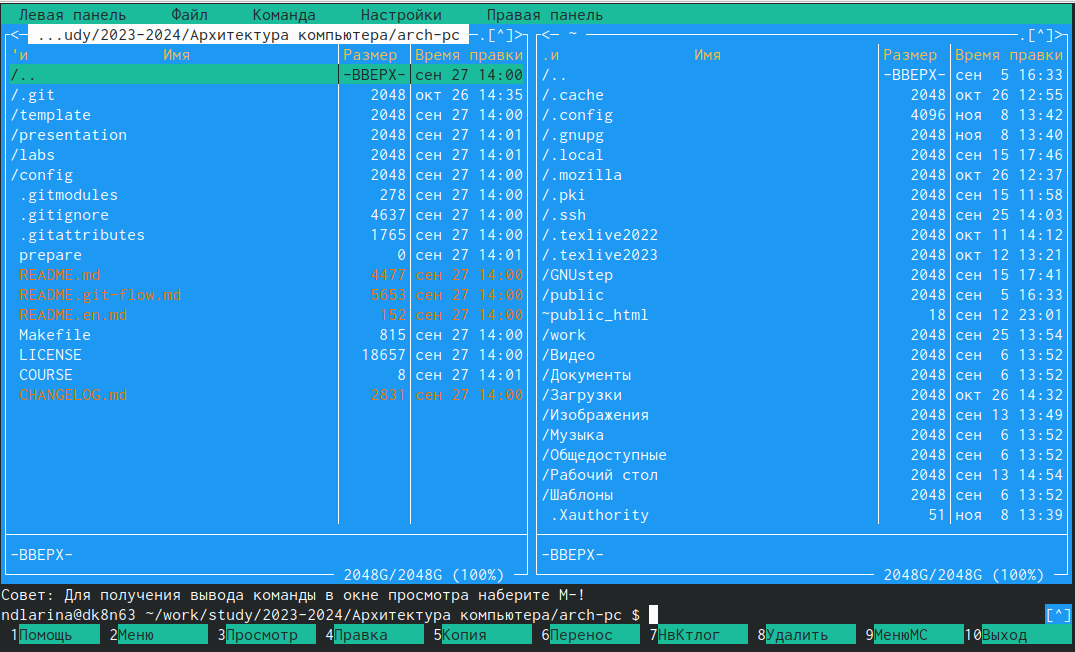


Рис. 2: Перемещение между директориями

Затем создаю каталог lab05 с помощью функциональной клавиши F7 (рис. [3](#fig:003)).

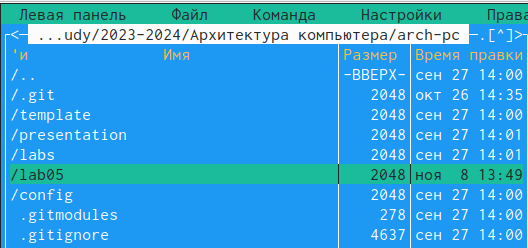


Рис. 3: Создание каталога

Перехожу в созданный каталог (рис. [4](#fig:004)).

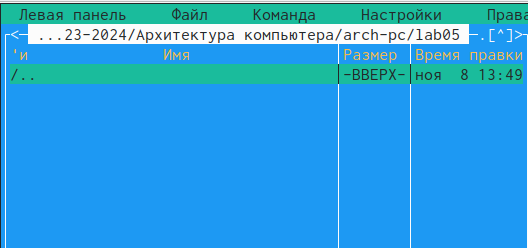


Рис. 4: Перемещение между директориями

В строке ввода прописываю команду touch lab5-1.asm, чтобы создать файл, в котором буду работать (рис. [5](#fig:005)).

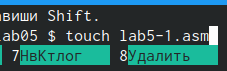


Рис. 5: Создание файла

## 4.2 Структура программы на языке ассемблера NASM

Открываю созданный файл для редактирования в редакторе nano с помощью функциональной клавиши F4. Ввожу в файл код программы для запроса строки у пользователя. Далее выхожу из файла (Ctrl+X), сохраняя изменения (Y, Enter) (рис. [6](#fig:006)).

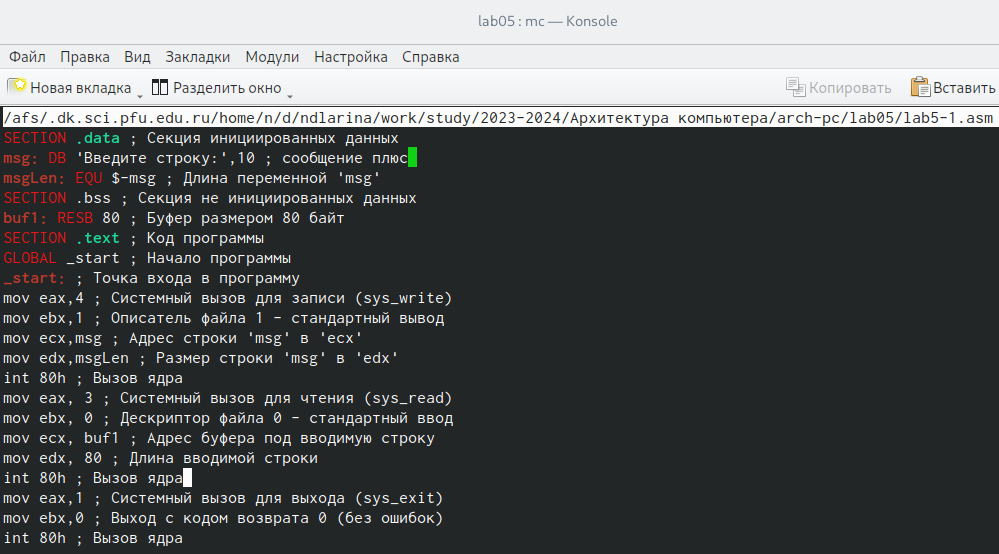


Рис. 6: Открытый отредактированный файл

Затем с помощью функциональной клавиши F3 открываю файл для просмотра, чтобы проверить, содержит ли файл текст программы (рис. [7](#fig:007)).

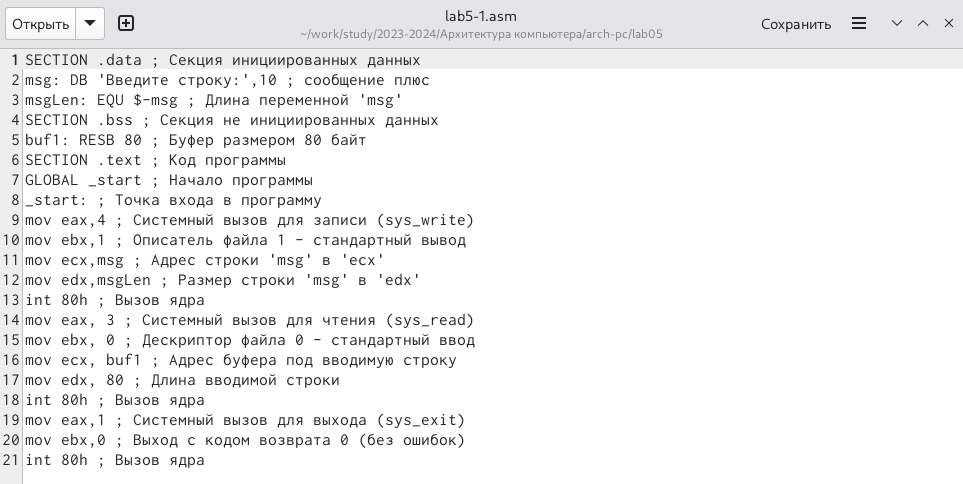


Рис. 7: Открытие файла для просмотра

Транслирую текст программы файла в объектный файл командой nasm -f elf lab5-1.asm. Создался объектный файл lab5-1.o (рис. [8](#fig:008)).

Figure 8: Компиляция файла

Рис. 8: Компиляция файла

Выполняю компоновку объектного файла с помощью команды ld -m elf\_i386 -o lab5-1 lab5-1.o (рис. [9](#fig:009)). Создался исполняемый файл lab5-1.

Figure 9: Передача на обработку компоновщику

Рис. 9: Передача на обработку компоновщику

Далле запускаю исполняемый файл. Программа выводит строку “Введите строку:” и ждет ввода с клавиатуры, я ввожу свои ФИО, на этом программа заканчивает свою работу (рис. [10](#fig:010)).

Figure 10: Исполнение файла

Рис. 10: Исполнение файла

## 4.3 Подключение внешнего файла

Скачиваю файл in\_out.asm со страницы курса в ТУИС, который сохранился в каталог “Загрузки” (рис. [11](#fig:011)).

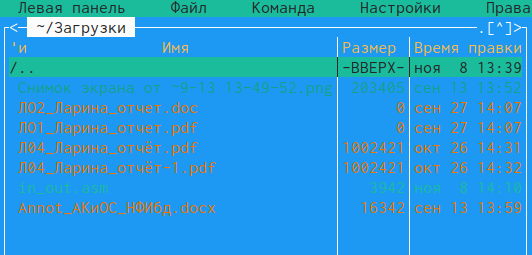


Рис. 11: Скачанный файл

Копирую файл in\_out.asm из каталога Загрузки в созданный каталог lab05 с помощью функциональной клавиши F5 (рис. [12](#fig:012)).

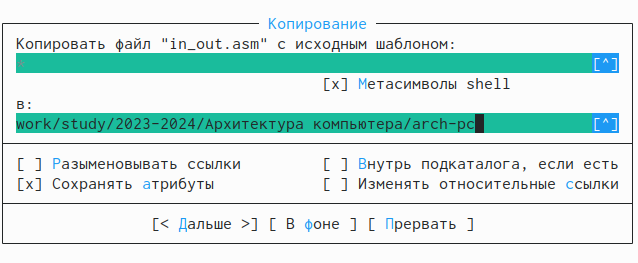


Рис. 12: Копирование файла

Затем с помощью функциональной клавиши F5 копирую файл lab5-1 в тот же каталог, но с другим именем, для этого в появившемся окне mc прописываю имя для копии файла (рис. [13](#fig:013)).

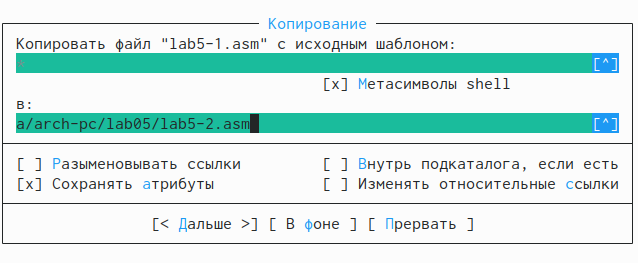


Рис. 13: Копирование файла

Изменяю содержимое файла lab5-2.asm во встроенном редакторе nano (рис. [14](#fig:014)), чтобы в программе использовались подпрограммы из внешнего файла in\_out.asm.

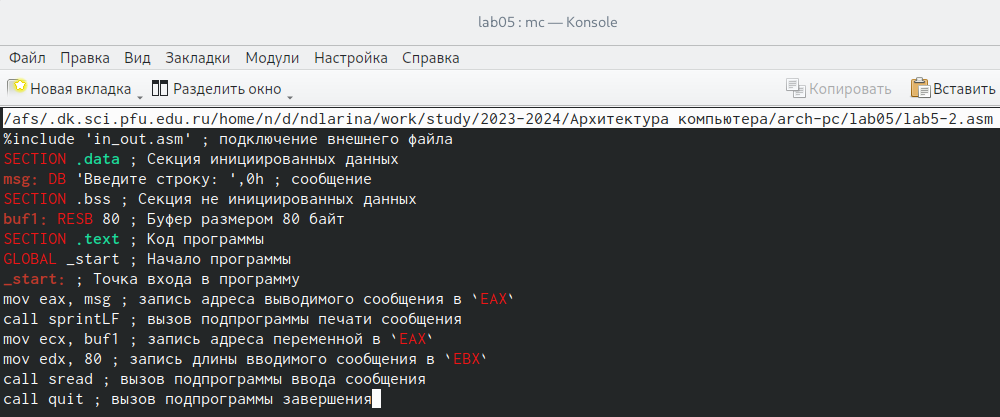


Рис. 14: Редактирование файла

Транслирую текст программы файла в объектный файл командой nasm -f elf lab5-2.asm. Создался объектный файл lab5-2.o. Выполняю компоновку объектного файла с помощью команды ld -m elf\_i386 -o lab5-2 lab5-2.o Создался исполняемый файл lab5-2. Запускаю исполняемый файл (рис. -[15](#fig:015)).

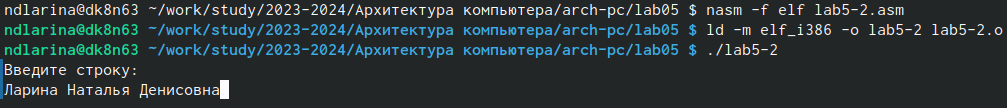


Рис. 15: Исполнение файла

Далее открываю файл lab5-2.asm для редактирования в nano функциональной клавишей F4. Изменяю в нем подпрограмму sprintLF на sprint. Сохраняю изменения и открываю файл для просмотра, чтобы проверить сохранение действий (рис. [16](#fig:016)).

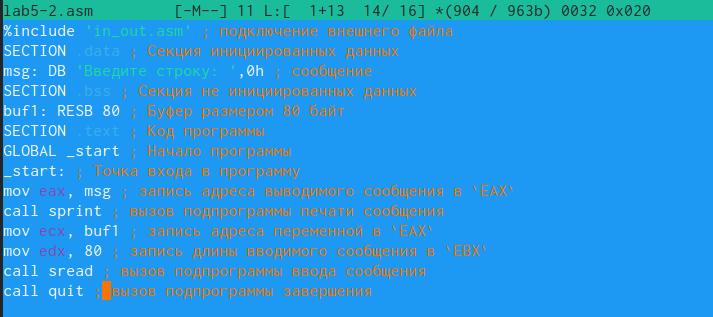


Рис. 16: Отредактированный файл

Снова транслирую файл, выполняю компоновку созданного объектного файла, запускаю новый исполняемый файл (рис. [17](#fig:017)).

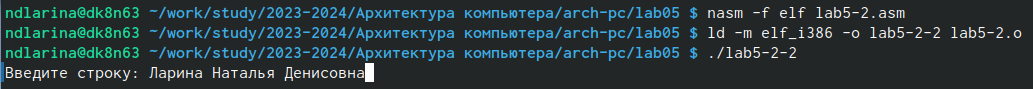


Рис. 17: Исполнение файла

Разница между первым исполняемым файлом lab5-2 и вторым lab5-2-2 в том, что запуск первого запрашивает ввод с новой строки, а программа, которая исполняется при запуске второго, запрашивает ввод без переноса на новую строку, потому что в этом заключается различие между подпрограммами sprintLF и sprint.

## 4.4 Выполнение заданий для самостоятельной работы

1. Создаю копию файла lab5-1.asm с именем lab5-1-1.asm с помощью функциональной клавиши F5 (рис. [18](#fig:018)).

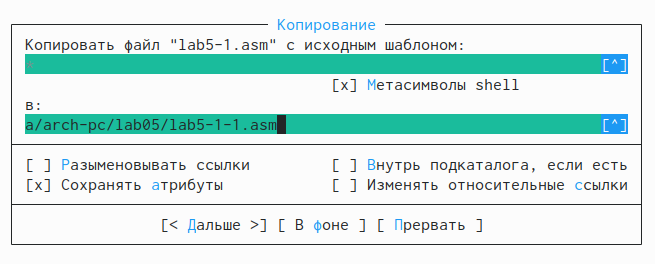


Рис. 18: Копирование файла

Открываю созданный файл для редактирования с помощью функциональной клавиши F4. Изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку (рис. [19](#fig:019)).

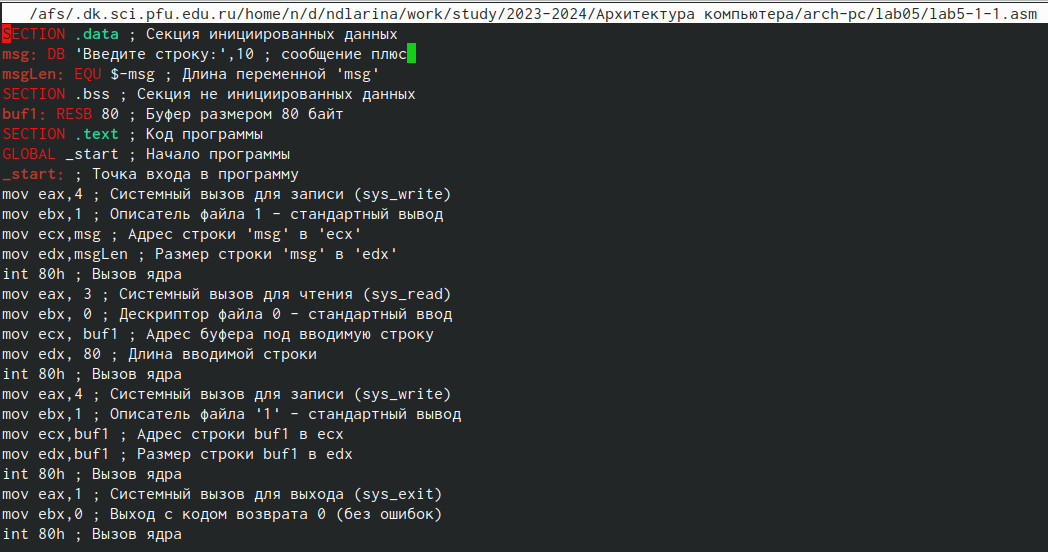


Рис. 19: Редактирование файла

1. Создаю объектный файл lab5-1-1.o, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-1-1, запускаю полученный исполняемый файл. Программа запрашивает ввод, ввожу свои ФИО, далее программа выводит введенные мною данные (рис. [20](#fig:020)).

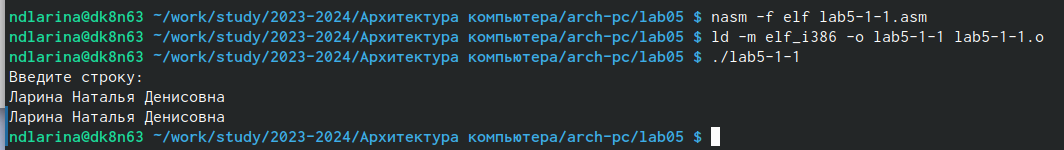


Рис. 20: Исполнение файла

Код программы из пункта 1:

SECTION .data ; Секция инициированных данных  
msg: DB 'Введите строку:',10  
msgLen: EQU $-msg ; Длина переменной 'msg'  
SECTION .bss ; Секция не инициированных данных  
buf1: RESB 80 ; Буфер размером 80 байт  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
mov eax,4 ; Системный вызов для записи (sys\_write)  
mov ebx,1 ; Описатель файла 1 - стандартный вывод  
mov ecx,msg ; Адрес строки 'msg' в 'ecx'  
mov edx,msgLen ; Размер строки 'msg' в 'edx'  
int 80h ; Вызов ядра  
mov eax, 3 ; Системный вызов для чтения (sys\_read)  
mov ebx, 0 ; Дескриптор файла 0 - стандартный ввод  
mov ecx, buf1 ; Адрес буфера под вводимую строку  
mov edx, 80 ; Длина вводимой строки  
int 80h ; Вызов ядра  
mov eax,4 ; Системный вызов для записи (sys\_write)  
mov ebx,1 ; Описатель файла '1' - стандартный вывод  
mov ecx,buf1 ; Адрес строки buf1 в ecx  
mov edx,buf1 ; Размер строки buf1  
int 80h ; Вызов ядра  
mov eax,1 ; Системный вызов для выхода (sys\_exit)  
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)  
int 80h ; Вызов ядра

1. Создаю копию файла lab5-2.asm с именем lab5-2-1.asm с помощью функциональной клавиши F5 (рис. [21](#fig:021)).

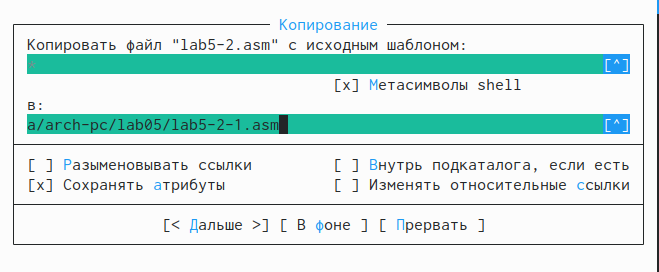


Рис. 21: Копирование файла

С помощью функциональной клавиши F4 открываю созданный файл для редактирования. Изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку (рис. [22](#fig:022)).

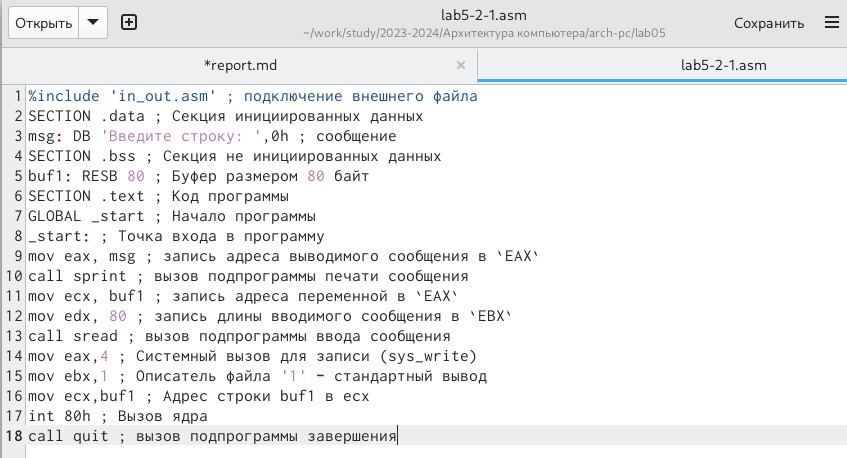


Рис. 22: Редактирование файла

1. Создаю объектный файл lab5-2-1.o, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-2-1, запускаю полученный исполняемый файл. Программа запрашивает ввод без переноса на новую строку, ввожу свои ФИО, далее программа выводит введенные мною данные (рис. [23](#fig:023)).

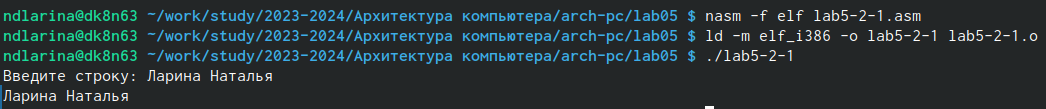


Рис. 23: Исполнение файла

Код программы из пункта 3:

%include 'in\_out.asm'  
SECTION .data ; Секция инициированных данных  
msg: DB 'Введите строку: ',0h ; сообщение  
SECTION .bss ; Секция не инициированных данных  
buf1: RESB 80 ; Буфер размером 80 байт  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
mov eax, msg ; запись адреса выводимого сообщения в `EAX`  
call sprint ; вызов подпрограммы печати сообщения  
mov ecx, buf1 ; запись адреса переменной в `EAX`  
mov edx, 80 ; запись длины вводимого сообщения в `EBX`  
call sread ; вызов подпрограммы ввода сообщения  
mov eax,4 ; Системный вызов для записи (sys\_write)  
mov ebx,1 ; Описатель файла '1' - стандартный вывод  
mov ecx,buf1 ; Адрес строки buf1 в ecx  
int 80h ; Вызов ядра  
call quit ; вызов подпрограммы завершения

# 5 Выводы

В ходе выполнения данной лабораторной работы я освоила инструкции языка ассемблера mov и int, а также приобрела практические навыки работы в Midnight Commander.

# 6 Список литературы

1. [Лабораторная работа №5](https://esystem.rudn.ru/mod/resource/view.php?id=1030553)