# **Отчет по лабораторной работе №6**

# **Дисциплина: архитектура компьютера**

Даровских Александра Сергеевна

Содержание

[**1** **Цель работы** 1](#_Toc150034478)

[**2** **Задание** 1](#_Toc150034479)

[**3** **Теоретическое введение** 1](#_Toc150034480)

[**4** **Выполнение лабораторной работы** 2](#_Toc150034481)

[**4.1** **Основы работы с mc** 2](#_Toc150034482)

[**4.2** **Структура программы на языке ассемблера NASM** 3](#_Toc150034483)

[**4.3** **Подключение внешнего файла** 3](#_Toc150034484)

[**4.4** **Выполнение заданий для самостоятельной работы** 4](#_Toc150034485)

[**5** **Выводы** 6](#_Toc150034486)

[**6** **Список литературы** 6](#_Toc150034487)

**1 Цель работы**

Целью данной лабораторной работы является 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.

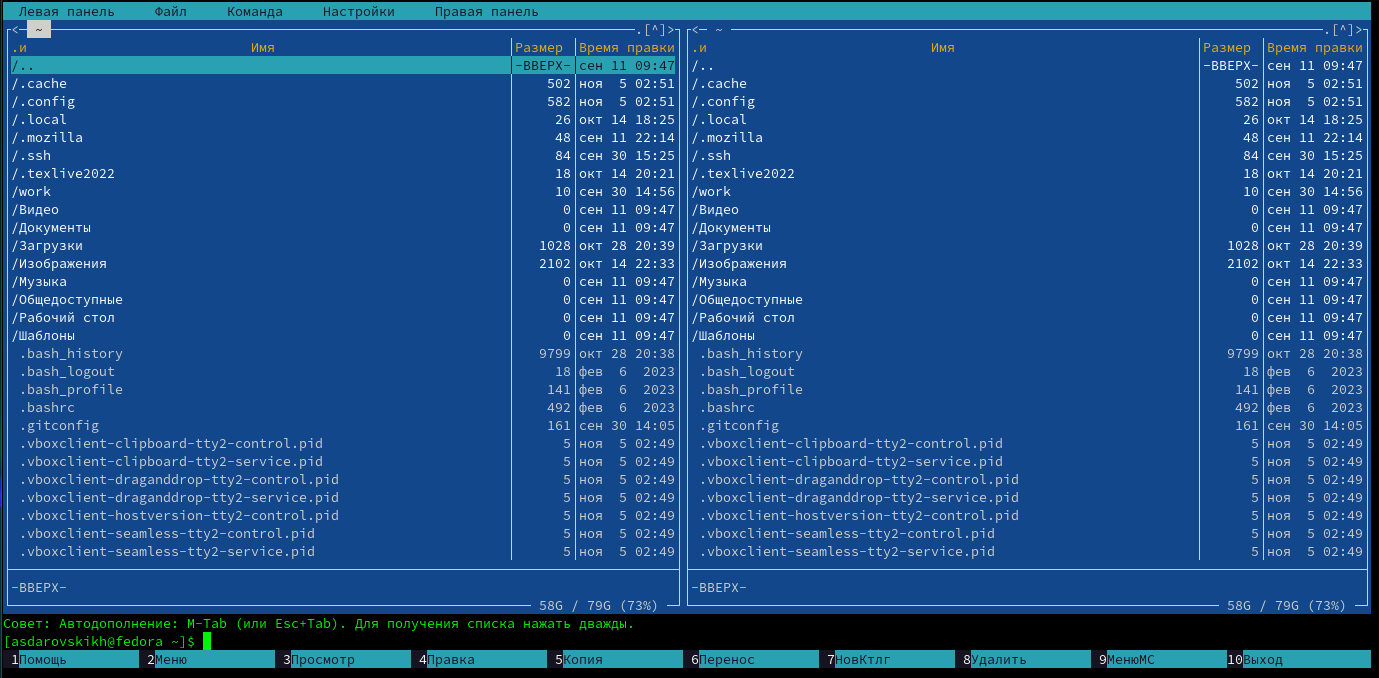
**int** n

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**

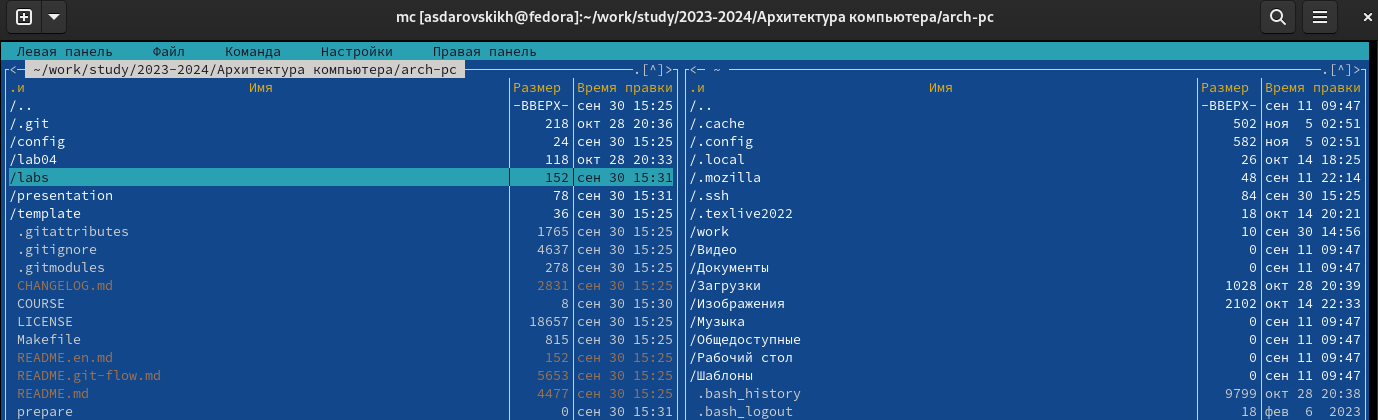
**4.1 Основы работы с mc**

Открываем Midnight Commander, введя в терминал mc (рис. 1).



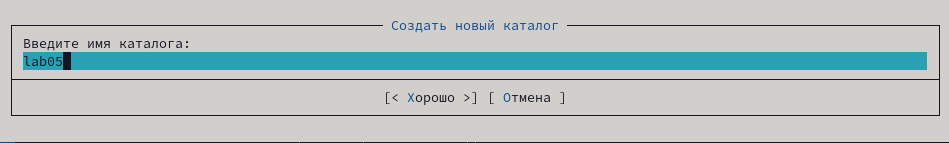
*Рис. 1: Открытый mc*

Переходим в каталог ~/work/study/2023-2024/Архитектура Компьютера/arch-pc, используя файловый менеджер mc (рис. 2)



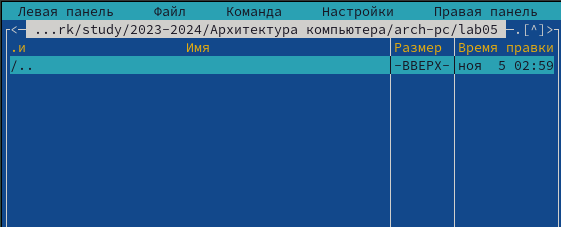
*Рис. 2: Перемещение между директориями*

С помощью функциональной клавиши F7 создаем каталог lab05 (рис. 3).



*Рис. 3: Создание каталога*

Переходим в созданный каталог (рис. 4).



*Рис. 4: Перемещение между директориями*

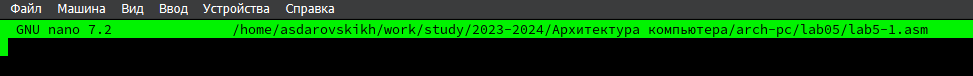
В строке ввода прописываем команду touch lab5-1.asm, чтобы создать файл, в котором будем работать (рис. 5).



*Рис. 5: Создание файла*

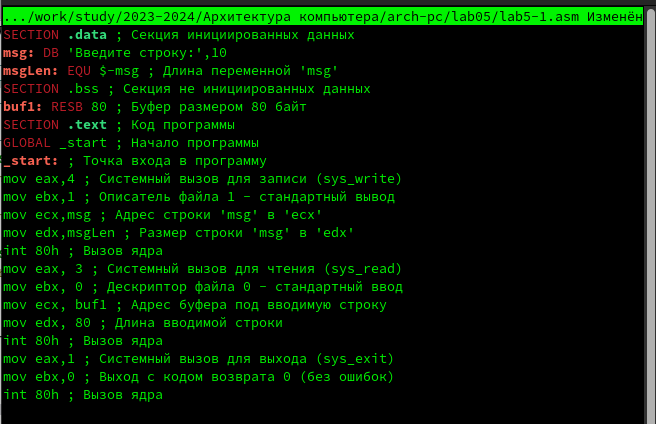
# **4.2 Структура программы на языке ассемблера NASM**

С помощью функциональной клавиши F4 открываем созданный файл для редактирования в редакторе nano (рис. 6).



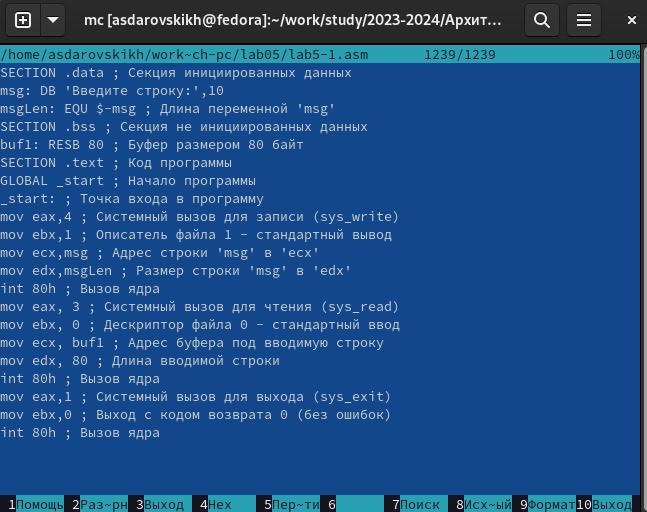
*Рис. 6: Открытие файла для редактирования*

Вводим в файл код программы для запроса строки у пользователя (рис. 7). Далее выходим из файла (Ctrl+X), сохраняя изменения (Y, Enter).



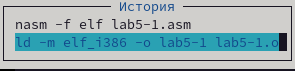
*Рис. 7: Редактирование файла*

С помощью функциональной клавиши F3 открываем файл для просмотра, чтобы проверить, содержит ли файл текст программы (рис. 8).



*Рис. 8: Открытие файла для просмотра*

Транслируем текст программы файла в объектный файл командой nasm -f elf lab6-1.asm. Создался объектный файл lab6-1.o. Выполняем компоновку объектного файла с помощью команды ld -m elf\_i386 -o lab6-1 lab6-1.o (рис. 9). Создался исполняемый файл lab6-1.



*Рис. 9: Компиляция файла и передача на обработку компоновщику*

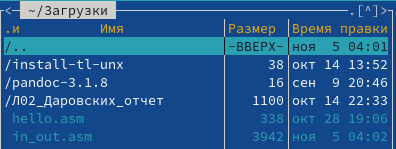
Запускаем исполняемый файл. Программа выводит строку “Введите строку:” и ждет ввода с клавиатуры, я ввожу свои ФИО, на этом программа заканчивает свою работу (рис. 10).



*Рис. 10: Исполнение файла*

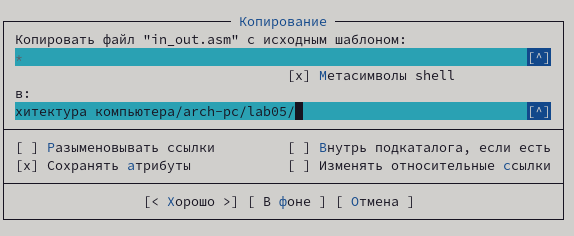
# **4.3 Подключение внешнего файла**

Скачиваем файл in\_out.asm со страницы курса в ТУИС. Он сохранился в каталог “Загрузки” (рис. 11).



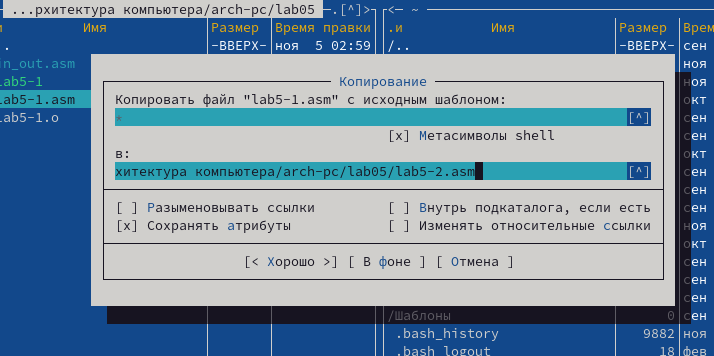
*Рис. 11: Скачанный файл*

С помощью функциональной клавиши F5 копируем файл in\_out.asm из каталога Загрузки в созданный каталог lab06 (рис. 12).



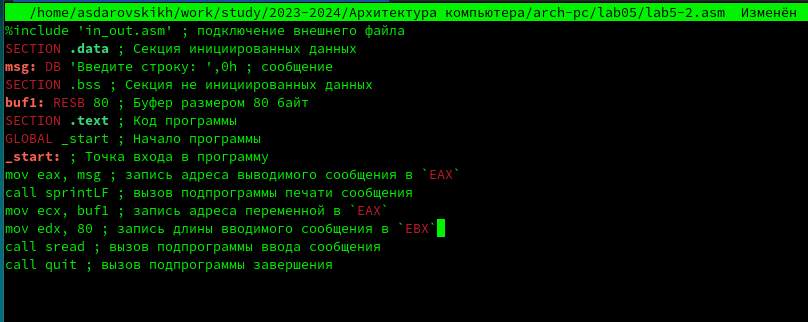
*Рис. 12: Копирование файла*

С помощью функциональной клавиши F5 копируем файл lab5-1 в тот же каталог, но с другим именем, для этого в появившемся окне mc прописываем имя для копии файла (рис. 13).



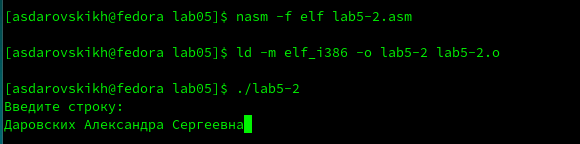
*Рис. 13: Копирование файла*

Изменяем содержимое файла lab5-2.asm во встроенном редакторе nano (рис. 14), чтобы в программе использовались подпрограммы из внешнего файла in\_out.asm.



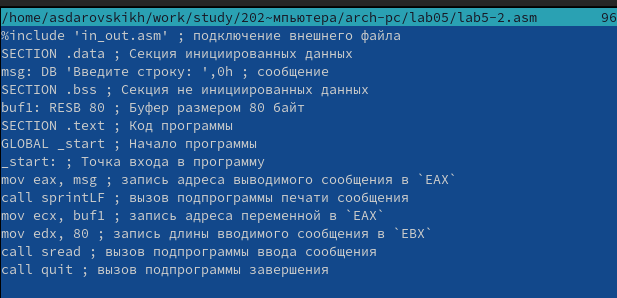
*Рис. 14: Редактирование файла*

Транслируем текст программы файла в объектный файл командой nasm -f elf lab6-2.asm. Создался объектный файл lab6-2.o. Выполняем компоновку объектного файла с помощью команды ld -m elf\_i386 -o lab6-2 lab6-2.o Создался исполняемый файл lab6-2. Запускаем исполняемый файл (рис. 15).



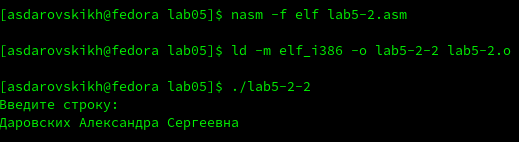
*Рис. 15: Исполнение файла*

Открываем файл lab5-2.asm для редактирования в nano функциональной клавишей F4. Изменяем в нем подпрограмму sprintLF на sprint. Сохраняем изменения и открываю файл для просмотра, чтобы проверить сохранение действий (рис. 16).



*Рис. 16: Отредактированный файл*

Снова транслируем файл, выполняем компоновку созданного объектного файла, запускаем новый исполняемый файл (рис. 17).

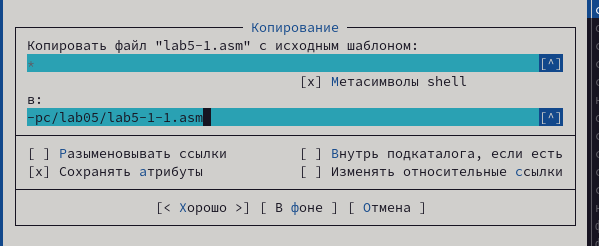


*Рис. 17: Исполнение файла*

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.

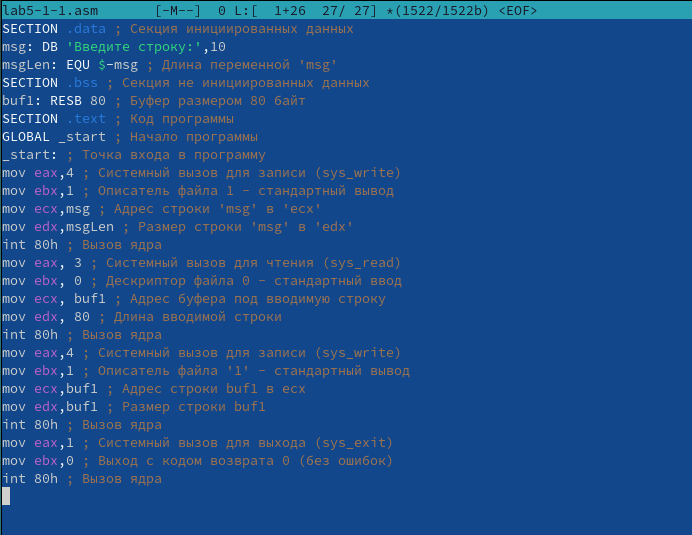
# **4.4 Выполнение заданий для самостоятельной работы**

1. Создаем копию файла lab5-1.asm с именем lab5-1-1.asm с помощью функциональной клавиши F5 (рис. 18).



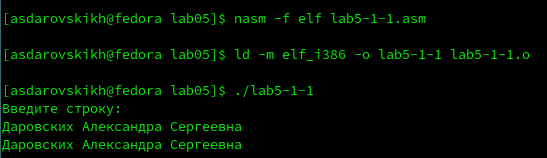
*Рис. 18: Копирование файла*

С помощью функциональной клавиши F4 открываем созданный файл для редактирования. Изменяем программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку (рис. 19).



*Рис. 19: Редактирование файла*

1. Создаем объектный файл lab5-1-1.o, отдаем его на обработку компоновщику, получаем исполняемый файл lab5-1-1, запускаем полученный исполняемый файл. Программа запрашивает ввод, вводим свои ФИО, далее программа выводит введенные мною данные (рис. 20).

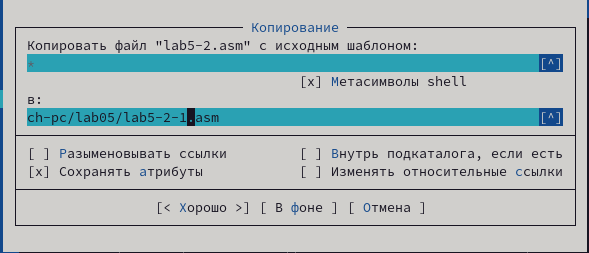


*Рис. 20: Исполнение файла*

Код программы из пункта 1:

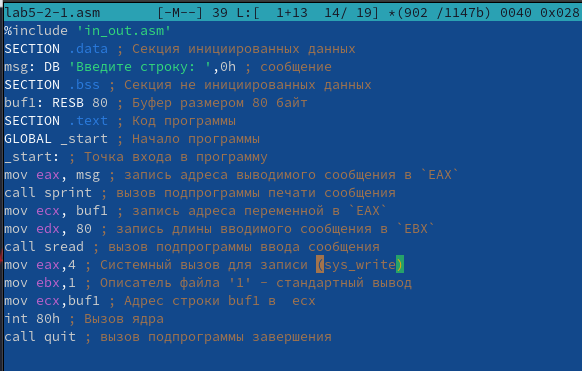
**SECTION** .data *; Секция инициированных данных*  
msg: DB 'Введите строку:',10  
msgLen: EQU $-msg *; Длина переменной 'msg'*  
**SECTION** .bss *; Секция не инициированных данных*  
buf1: RESB 80 *; Буфер размером 80 байт*  
**SECTION** .text *; Код программы*  
**GLOBAL** \_start *; Начало программы*  
\_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. Создаем копию файла lab5-2.asm с именем lab5-2-1.asm с помощью функциональной клавиши F5 (рис. 21).



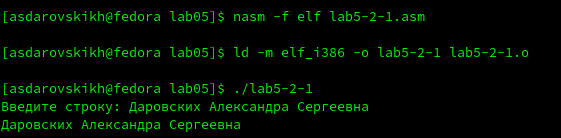
*Рис. 21: Копирование файла*

С помощью функциональной клавиши F4 открываем созданный файл для редактирования. Изменяем программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку (рис. 22).



*Рис. 22: Редактирование файла*

1. Создаем объектный файл lab6-2-1.o, отдаем его на обработку компоновщику, получаю исполняемый файл lab6-2-1, запускаю полученный исполняемый файл. Программа запрашивает ввод без переноса на новую строку, ввожу свои ФИО, далее программа выводит введенные мною данные (рис. 23).



*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**

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

# **6 Список литературы**

1. <https://esystem.rudn.ru/pluginfile.php/2089085/mod_resource/content/0/Лабораторная%20работа%20№5.%20Основы%20работы%20с%20Midnight%20Commander%20%28%29.%20Структура%20программы%20на%20языке%20ассемблера%20NASM.%20Системные%20вызовы%20в%20ОС%20GNU%20Linux.pdf>