**Oтчёт по лабораторной работе №5**

**Дисциплина: архитектура компьютера**

**Панявкина Ирина Васильевна**

**НКАбд-04-24**

Содержание

[1 Цель работы 3](#__RefHeading___Toc301_744737701)

[2 Задание 3](#__RefHeading___Toc303_744737701)

[3 Теоретическое введение 3](#__RefHeading___Toc305_744737701)

[4 Выполнение лабораторной работы 4](#__RefHeading___Toc307_744737701)

[4.1 Основы работы с mc 4](#__RefHeading___Toc309_744737701)

[4.2 Структура программы на языке ассемблера NASM 7](#__RefHeading___Toc313_744737701)

[4.3 Подключение внешнего файла 10](#__RefHeading___Toc315_744737701)

[4.4 Выполнение заданий для самостоятельной работы 14](#__RefHeading___Toc317_744737701)

[5 Выводы 18](#__RefHeading___Toc321_744737701)

[Список литературы 18](#__RefHeading___Toc323_744737701)

# 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.

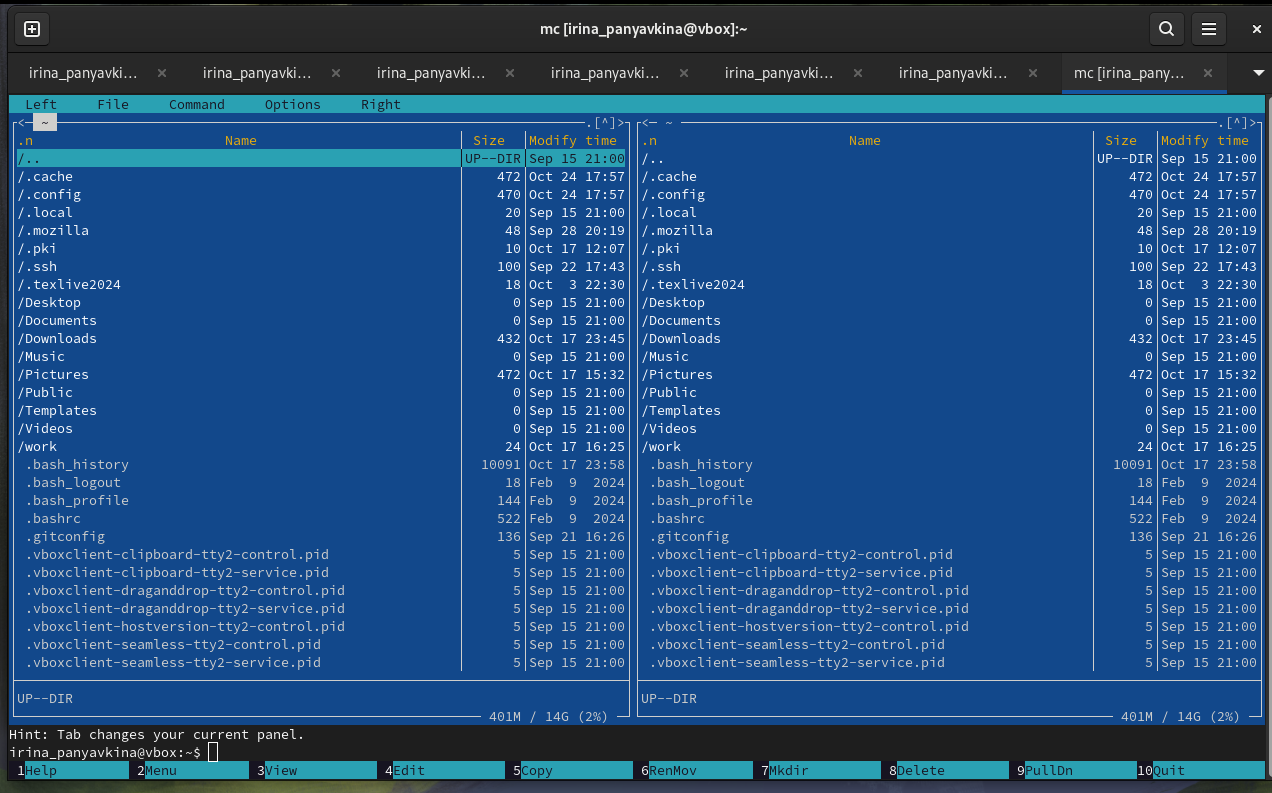
int n

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

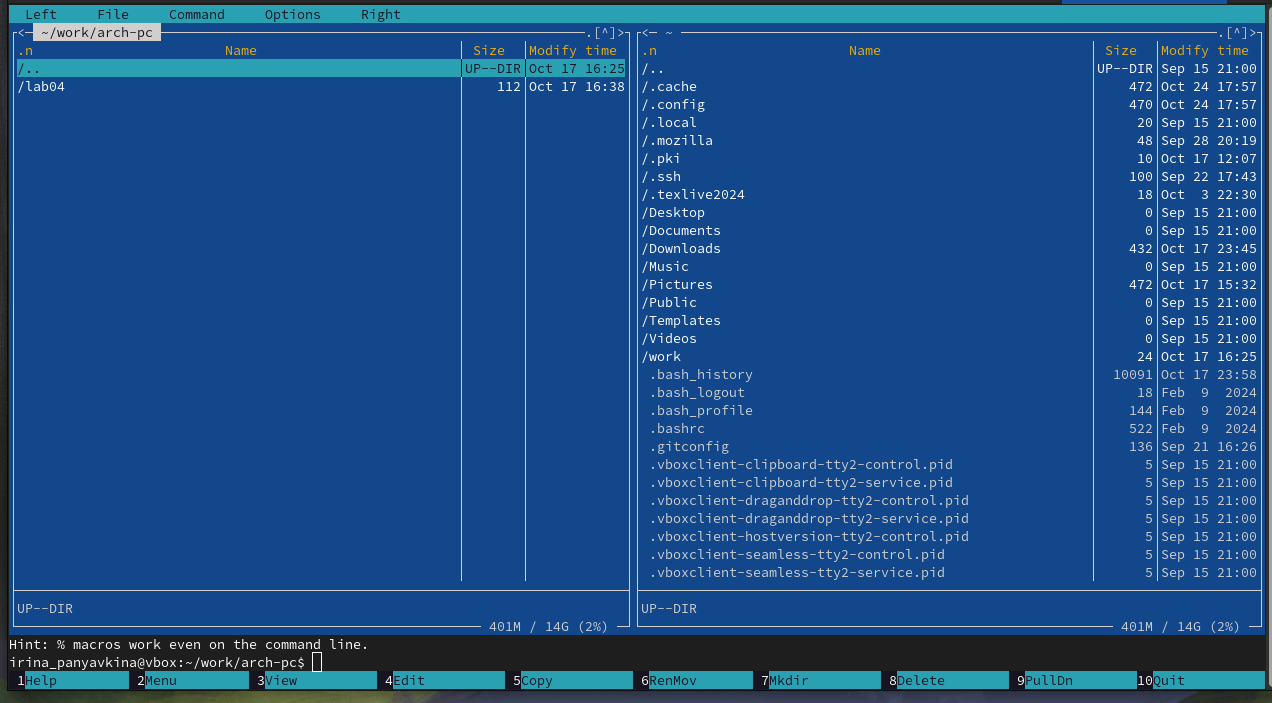
## 4.1 Основы работы с mc

Открываю Midnight Commander, введя в терминал mc (рис. 4.1).



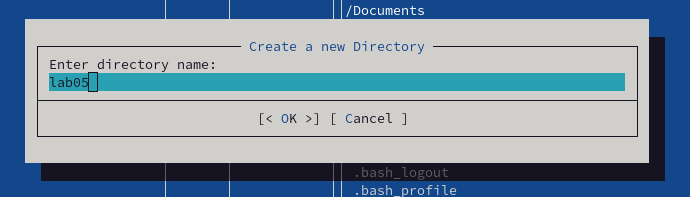
*Рис. 4.1: Открытый mc*

Перехожу в каталог ~/work/arch-pc/, используя файловый менеджер mc (рис. 4.2)



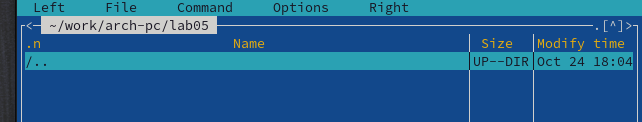
*Рис. 4.2: Перемещение между директориями*

С помощью функциональной клавиши F7 создаю каталог lab05 (рис. 4.3).



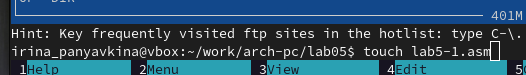
*Рис. 4.3: Создание каталога*

Перехожу в созданный каталог (рис. 4.4).



*Рис. 4.4: Перемещение между директориями*

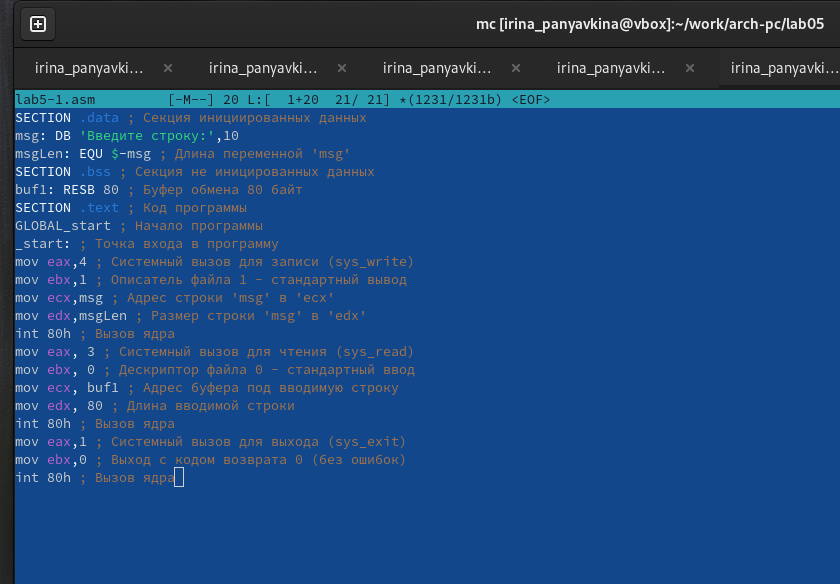
В строке ввода прописываю команду touch lab5-1.asm, чтобы создать файл, в котором буду работать (рис. 4.5).



*Рис. 4.5: Создание файла*

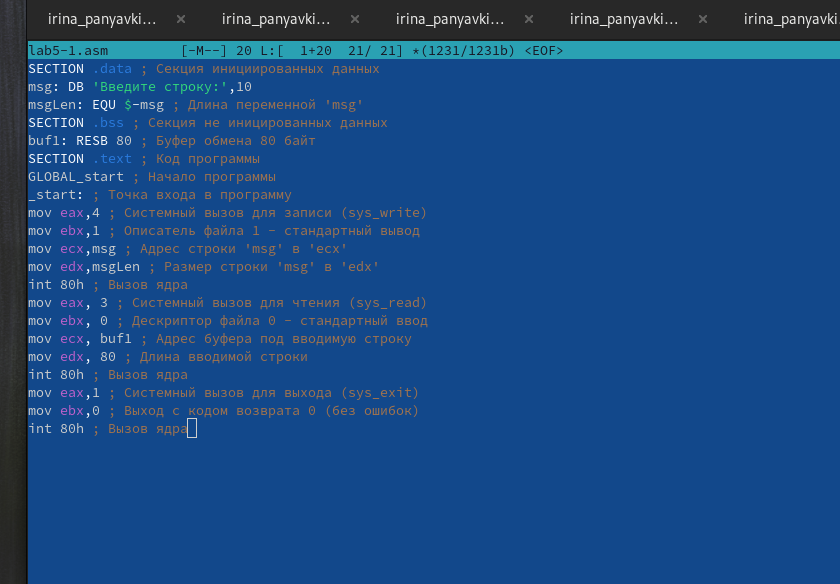
## 4.2 Структура программы на языке ассемблера NASM

С помощью функциональной клавиши F4 открываю созданный файл для редактирования в редакторе mcedit (рис. 4.6).



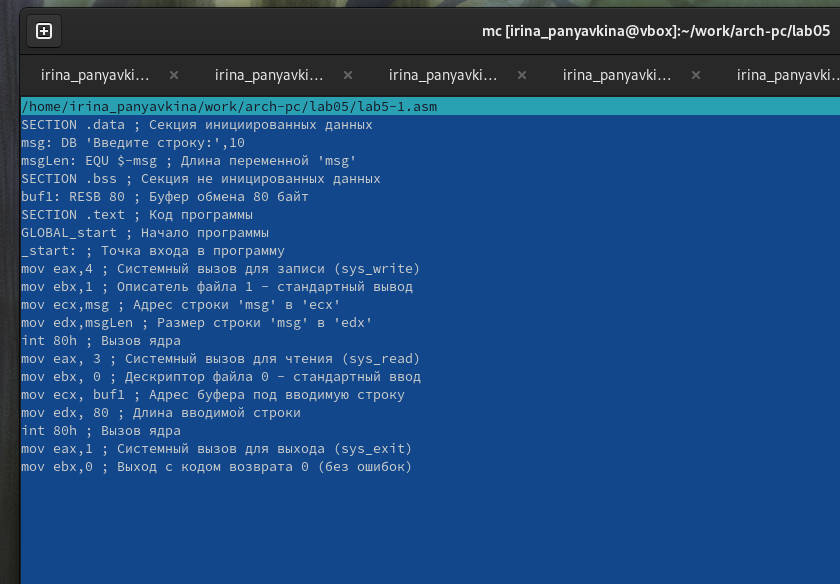
*Рис. 4.6: Открытие файла для редактирования*

Ввожу в файл код программы для запроса строки у пользователя (рис. 4.7). Далее выхожу из файла (Ctrl+X), сохраняя изменения (Y, Enter).



*Рис. 4.7: Редактирование файла*

С помощью функциональной клавиши F3 открываю файл для просмотра, чтобы проверить, содержит ли файл текст программы (рис. 4.8).



*Рис. 4.8: Открытие файла для просмотра*

Транслирую текст программы файла в объектный файл командой nasm -f elf lab5-1.asm. Создался объектный файл lab5-1.o. Выполняю компоновку объектного файла с помощью команды ld -m elf\_i386 -o lab5-1 lab5-1.o (рис. 4.9). Создался исполняемый файл lab5-1.



*Рис. 4.9: Компиляция файла и передача на обработку компоновщику*

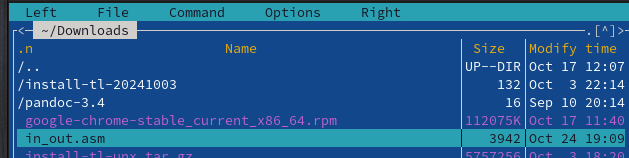
Запускаю исполняемый файл. Программа выводит строку “Введите строку:” и ждет ввода с клавиатуры, я ввожу свои ФИО, на этом программа заканчивает свою работу (рис. 4.10).



*Рис. 4.10: Исполнение файла*

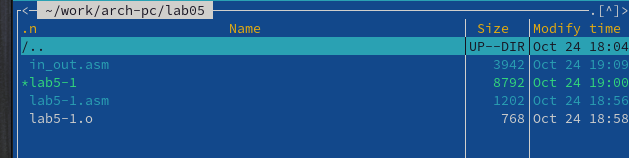
## 4.3 Подключение внешнего файла

Скачиваю файл in\_out.asm со страницы курса в ТУИС. Он сохранился в каталог “Загрузки” (рис. 4.11).



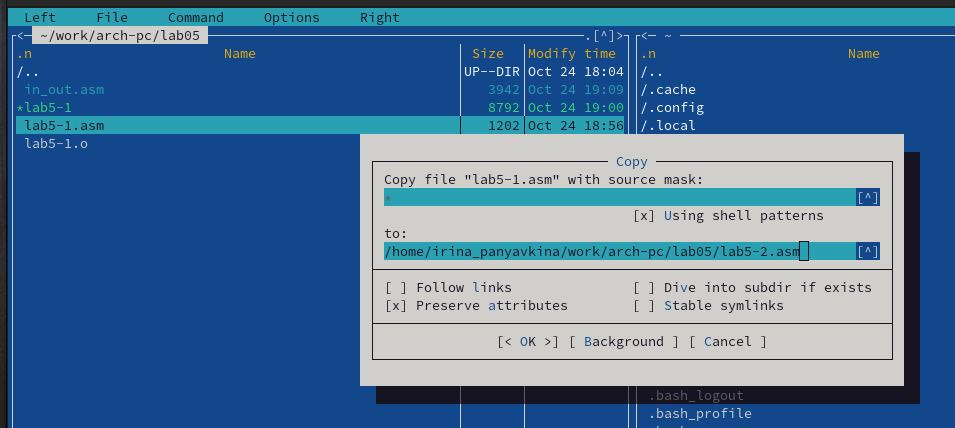
*Рис. 4.11: Скачанный файл*

С помощью функциональной клавиши F5 копирую файл in\_out.asm из каталога Загрузки в созданный каталог lab05 (рис. 4.12).



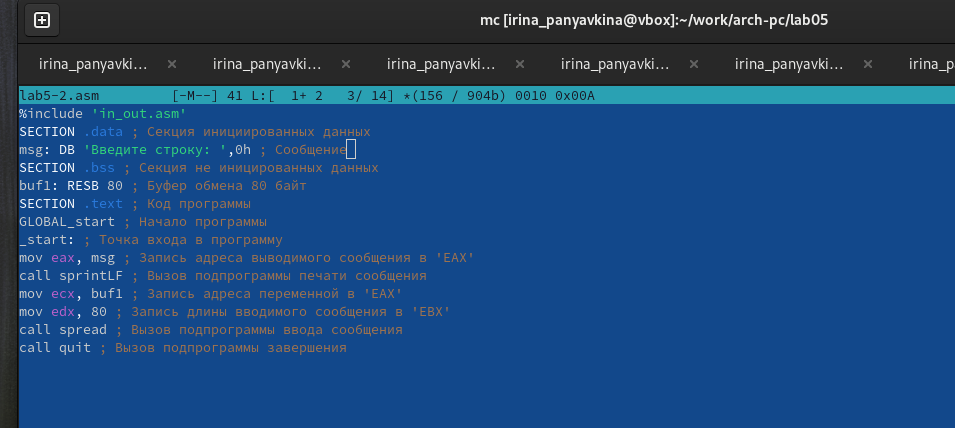
*Рис. 4.12: Копирование файла*

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. 4.13).



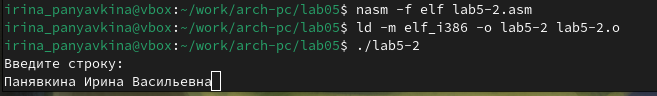
*Рис. 4.13: Копирование файла*

Изменяю содержимое файла lab5-2.asm во встроенном редакторе mcedit (рис. 4.14), чтобы в программе использовались подпрограммы из внешнего файла in\_out.asm.



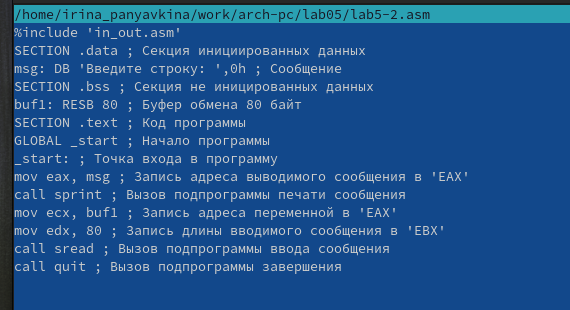
*Рис. 4.14: Редактирование файла*

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. 4.15).



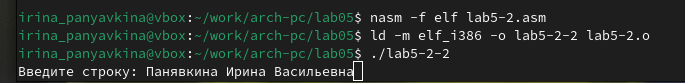
*Рис. 4.15: Исполнение файла*

Открываю файл lab5-2.asm для редактирования в mcedit функциональной клавишей F4. Изменяю в нем подпрограмму sprintLF на sprint. Сохраняю изменения и открываю файл для просмотра, чтобы проверить сохранение действий (рис. 4.16).



*Рис. 4.16: Отредактированный файл*

Снова транслирую файл, выполняю компоновку созданного объектного файла, запускаю новый исполняемый файл (рис. 4.17).

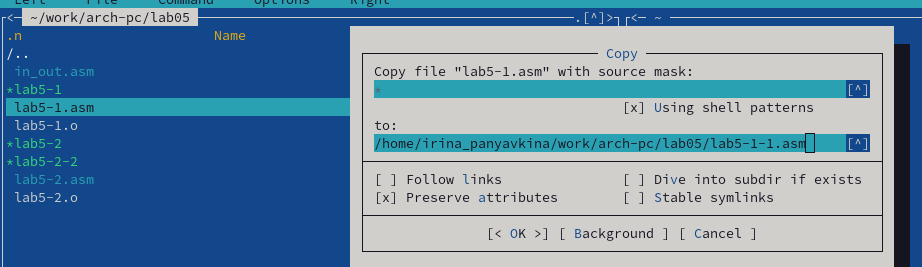


*Рис. 4.17: Исполнение файла*

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.

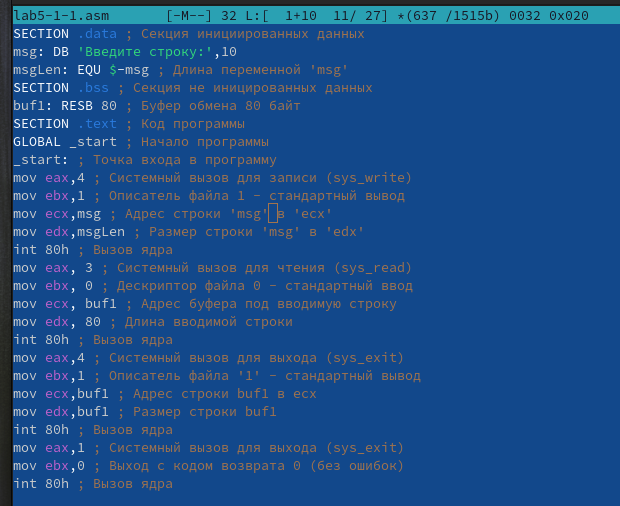
## 4.4 Выполнение заданий для самостоятельной работы

1.         Создаю копию файла lab5-1.asm с именем lab5-1-1.asm с помощью функциональной клавиши F5 (рис. 4.18).



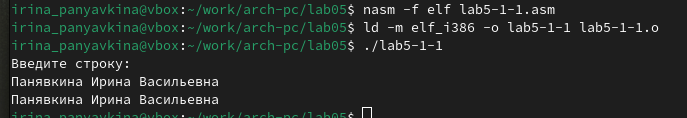
*Рис. 4.18: Копирование файла*

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. 4.19).



*Рис. 4.19: Редактирование файла*

2.         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. 4.20).

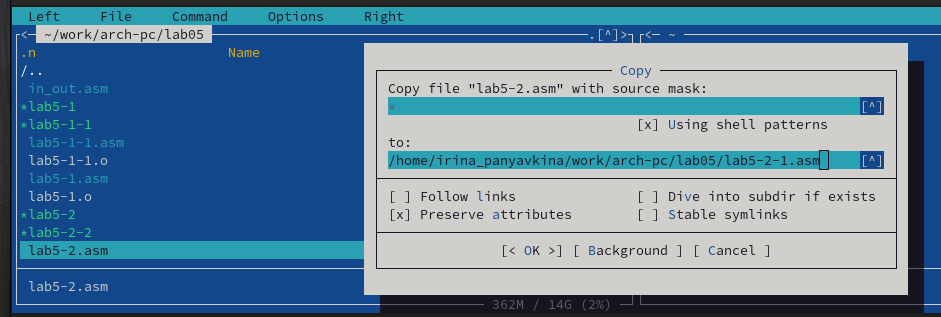


*Рис. 4.20: Исполнение файла*

Код программы из пункта 1:

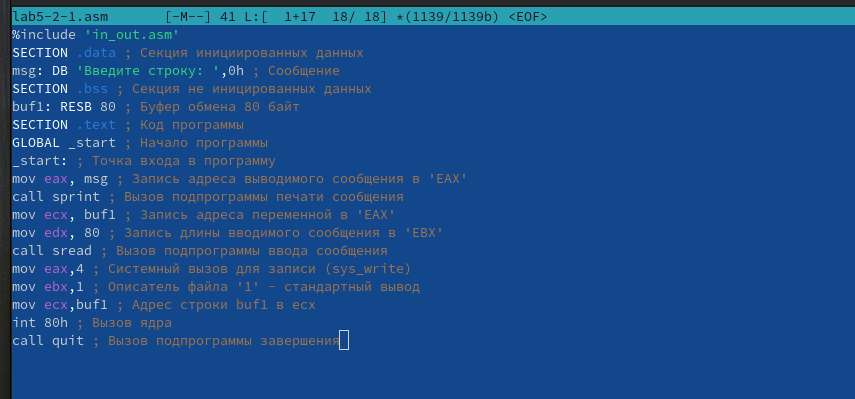
SECTION .data ; Секция инициированных данных  
msg: DB 'Введите строку:',10  
msgLen: EQU $-msg ; Длина переменной 'msg'  
SECTION .bss ; Секция не инициированных данных  
buf1: RESB 80 ; Буфер размером 80 байт  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа 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

3.         Создаю копию файла lab5-2.asm с именем lab5-2-1.asm с помощью функциональной клавиши F5 (рис. 4.21).



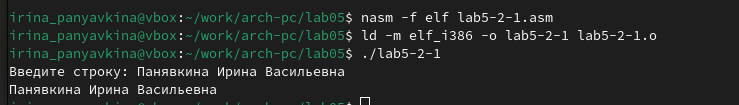
*Рис. 4.21: Копирование файла*

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. 4.22).



*Рис. 4.22: Редактирование файла*

4.         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. 4.23).



*Рис. 4.23: 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.

# 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/Learningbash-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. — 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>.

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).