Отчёт по лабораторной работе №5

Дисциплина: Архитектура Компьютера

Дарина Андреевна Куокконен

Содержание

# 1 Цель работы

Целью моей лабораторной работы - преобретение практических навыков работы в Midnight Commander, а также 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 тс) — это программа, которая позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой, т.е. тс 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,srt

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

**int** n

Здесь п - номер прерывания, принадлежащий диапазону 0-255. При програм-мировании 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. [1](#fig:001)).

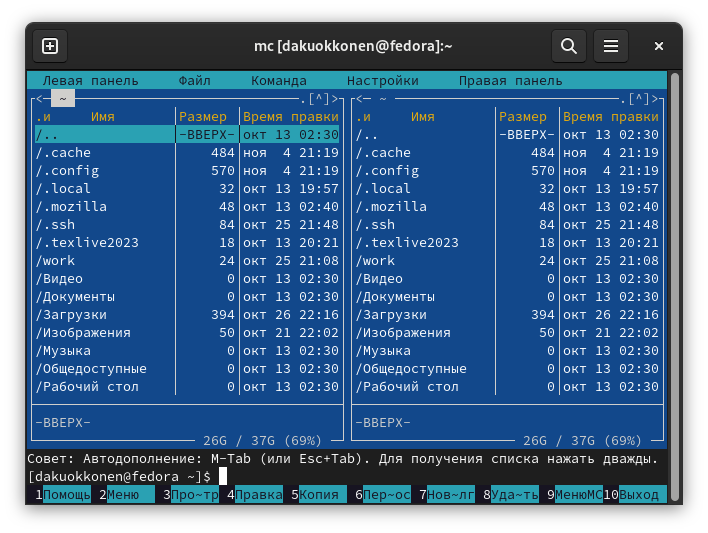


Figure 1: Открытие mc

Перехожу в раннее созданный мной каталог, используя файловый менеджер mc (рис. [2](#fig:002)).

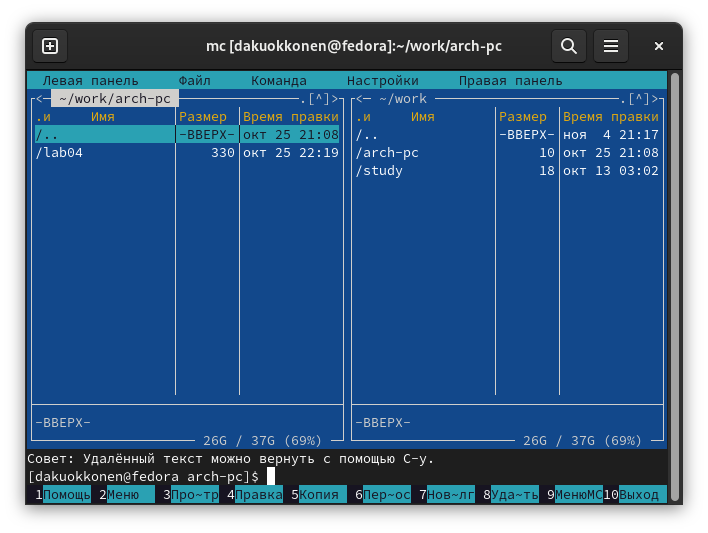


Figure 2: Перемещение между директориями

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

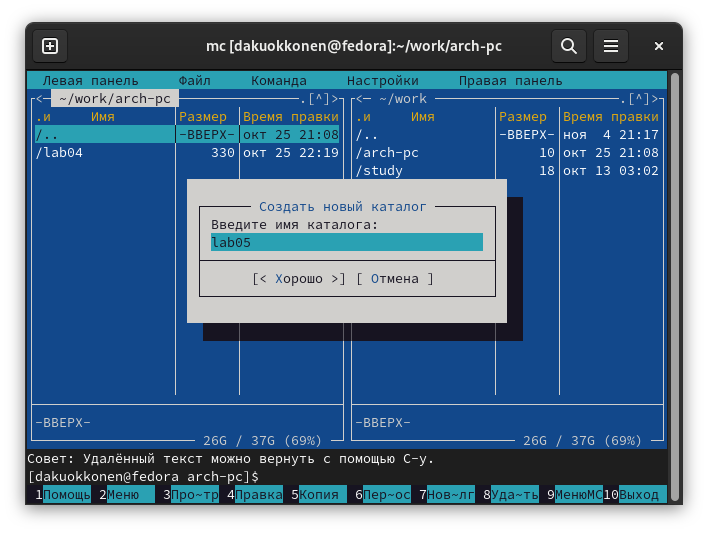


Figure 3: Создание каталога

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

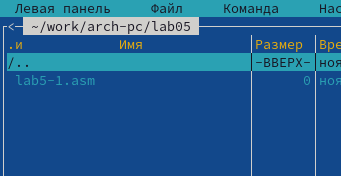


Figure 4: Перемещение между директориями

4.2) Структура программы на языке ассемблера NASM В строке ввода прописываю команду touch lab5-1.asm, чтобы создать файл, в котором буду работать и открываю его с помощью функциональной клавиши F4 в редакторе nano(рис. [5](#fig:005)).

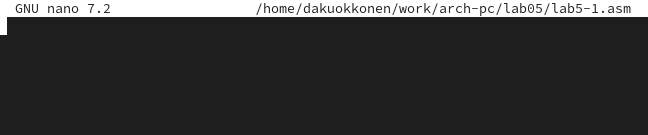


Figure 5: Открытый файл

Ввожу в файл код программы для запроса строки у пользователя и выхожу с файла(рис. [6](#fig:006)).

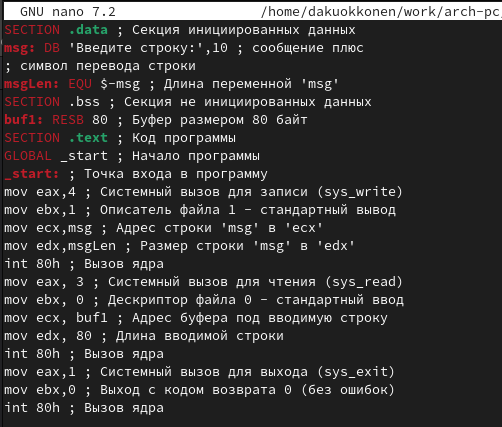


Figure 6: Редактирование файла

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

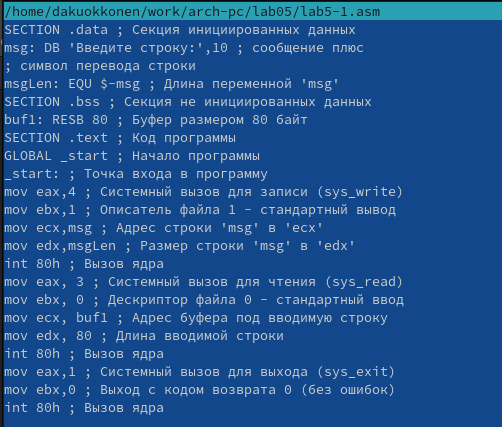


Figure 7: Открытие файла

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 lab6-1.o (рис. [8](#fig:008)).

Figure 8: Компиляция файла, передача компоновщику

Figure 8: Компиляция файла, передача компоновщику

Запускаю исполяемый файл, ввожу свои ФИО в строке “Введите строку”. На этом программа заканчивает свою работу(рис. [9](#fig:009)).

Figure 9: Исполнение файла

Figure 9: Исполнение файла

4.3) Подключение внешнего файла Скачиваю файл in\_out.asm со страницы курса в ТУИС. Он сохранился в каталог “Загрузки”(рис. [10](#fig:010)).

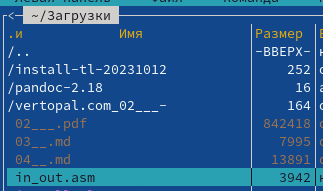


Figure 10: Скаченный файл

Копирую файл in\_out\_asm из каталога Загрузки в созданный каталог lab05. Проделываю тоже самое с файлом lab5-1, изменяя имя файла на lab5-2.asm(рис. [11](#fig:011)).

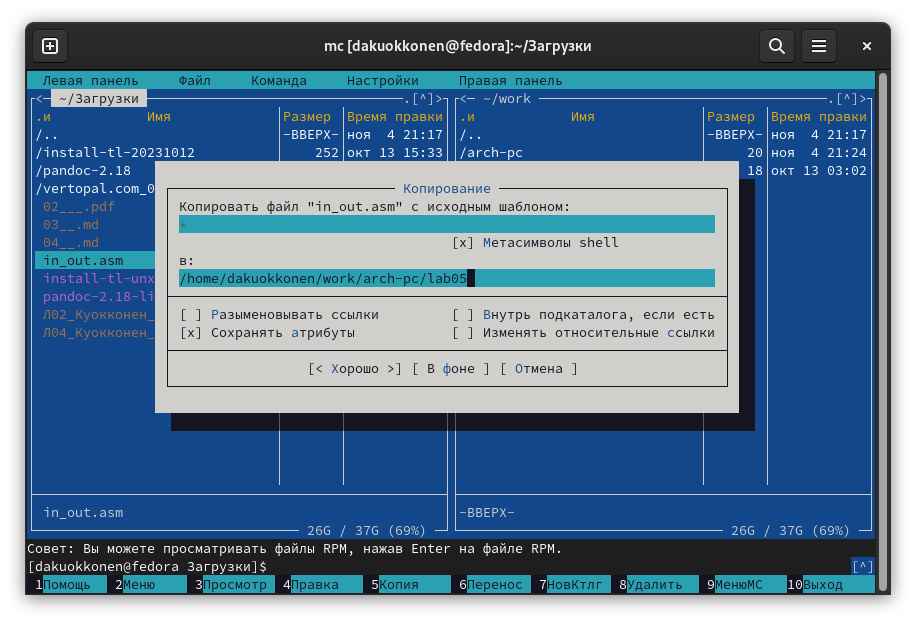


Figure 11: Копирование файла

Изменяю содержимое файла, чтобы в программе использовались подпрограммы из внешнего файла in\_out.asm(рис. [12](#fig:012)).

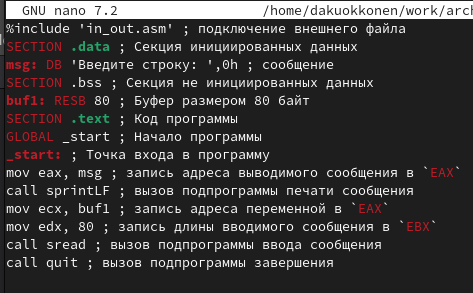


Figure 12: Редактирование файла

Транслирую текст программы и выполняю компоновку объектного файла. Затем, запускаю исполняемый файл(рис. [13](#fig:013)).

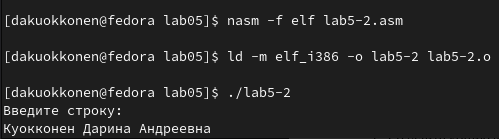


Figure 13: Исполнение файла

Открываю файл lab5-2.asm для редактирования в nano. Изменяю в нем подпрограмму sprintLF на sprint(рис. [14](#fig:014)).

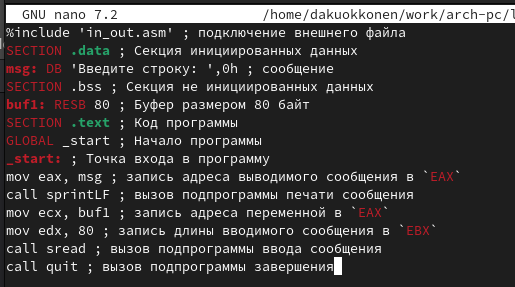


Figure 14: Редактирование файла

Транслирую файл, выполняю компоновку созданного объектного файла, запускаю файл(рис. [15](#fig:015)).

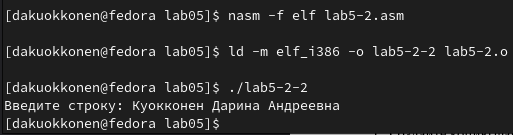


Figure 15: Исполнение файла

Разница между первым и вторым файлами заключается 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. [16](#fig:016)).

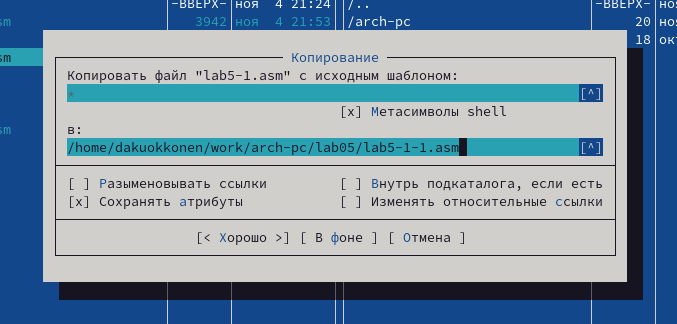


Figure 16: Копирование файла

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. [17](#fig:017)).

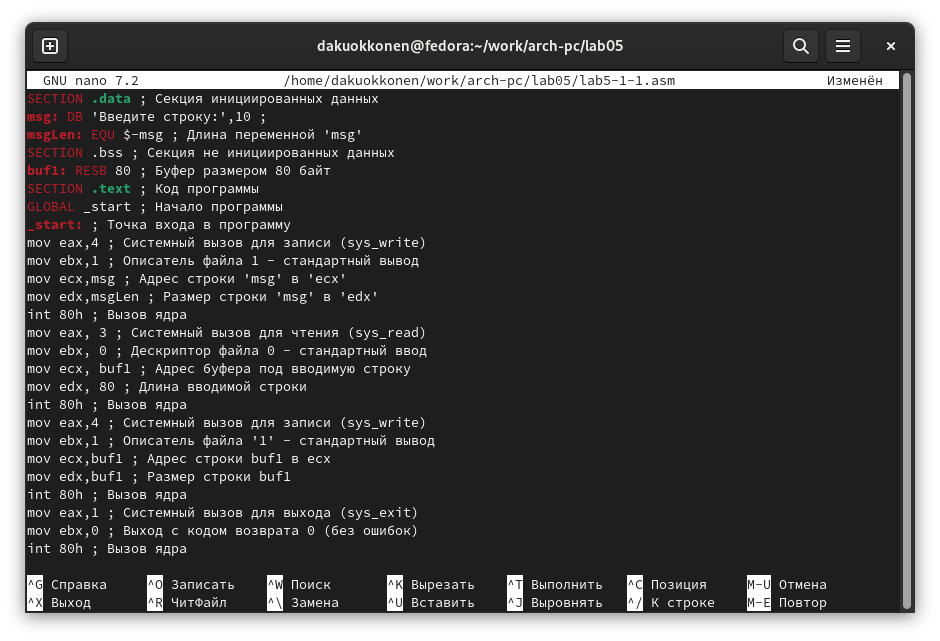


Figure 17: Редактирование файла

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. [18](#fig:018)).

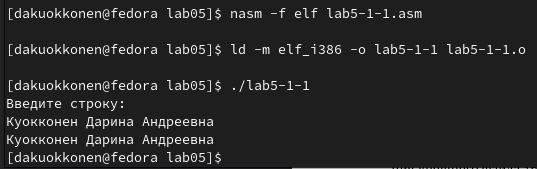


Figure 18: Исполнение файла

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. [19](#fig:019)).

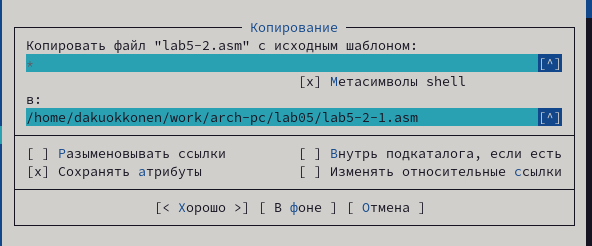


Figure 19: Копирование файла

Открываю файл, изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку(рис. [20](#fig:020)).

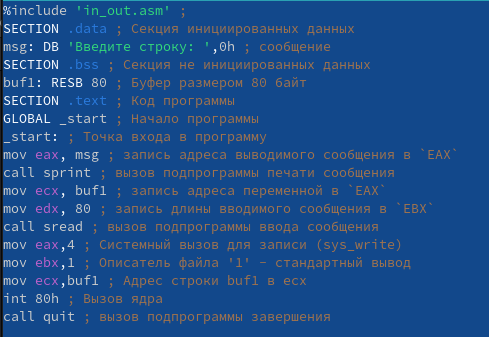


Figure 20: Редактирование файла

Создаю объектный файл lab5-2-1.o, отдаю его на обработку компоновщику. Запускаю полученный исполняемый файл. Вижу, что программа запрашивает ввод без переноса на новую строку, ввожу свои ФИО, далее программа выводит введенные мною данные(рис. [21](#fig:021)).

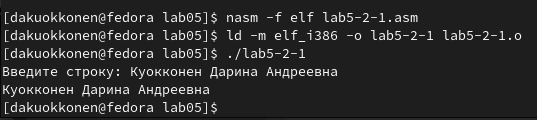


Figure 21: Исполнение файла

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 приобрела практические навыки работы в Mignight Commander, освоила инструкции языка ассемблера mov и int.

# Список литературы

1. [Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089085/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.%20%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B%20%D1%81%20Midnight%20Commander%20%28%29.%20%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B%20%D0%BD%D0%B0%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5%20%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0%20NASM.%20%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D1%8B%D0%B5%20%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2%D1%8B%20%D0%B2%20%D0%9E%D0%A1%20GNU%20Linux.pdf)