

Отчёт по лабораторной работе №5

Дисциплина: Архитектура Компьютера

Егор Витальевич Кузьмин

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Midnight Commander	8
4.2	Перемещение	9
4.3	Создание каталога	9
4.4	Перемещение	10
4.5	Создание файла	10
4.6	Редактирование файла	10
4.7	Открытие файла для просмотра	11
4.8	Компиляция, обработка и запуск исполняемого файла	11
4.9	Копирование файла	12
4.10	Копирование файла	13
4.11	Редактирование файла	13
4.12	Компиляция, обработка и запуск исполняемого файла	14
4.13	Редактирование файла	14
4.14	Запуск исполняемого файла	15
4.15	Создание копии файла	15
4.16	Редактирование файла	16
4.17	Компиляция, обработка и запуск исполняемого файла	16
4.18	Создание копии файла	17
4.19	Отправка файлов	17
4.20	Компиляция, обработка и запуск исполняемого файла	18

1 Цель работы

Целью данной работы является приобретение практического опыта работы с Midnight Commander, освоение инструкций языка ассемблера mov и int.

2 Задание

- 0. Общее ознакомление с Midnight Commander
- 1. Основы работы с Midnight Commander
- 2. Подключение внешнего файла
- 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). 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: 1) DB (define byte) — определяет переменную размером в 1 байт; 2) DW (define word) — определяет переменную размером в 2 байта (слово); 3) DD (define double word) — определяет переменную размером в 4 байта (двойное слово); 4) DQ (define quad word) — определяет переменную размером в 8 байт (четверённое слово); 5) 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. Для объявления неинициированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти. Инструкция языка ассемблера mov

предназначена для дублирования данных источника в приёмнике. Здесь операнд `dst` — приёмник, а `src` — источник. В качестве операнда могут выступать регистры (`register`), ячейки памяти (`memory`) и непосредственные значения (`const`). Переслать значение из одной ячейки памяти в другую нельзя, для этого необходимо использовать две инструкции `mov`. Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером. После вызова инструкции `int 80h` выполняется системный вызов какой-либо функции ядра Linux. При этом происходит передача управления ядру операционной системы. Чтобы узнать, какую именно системную функцию нужно выполнить, ядро извлекает номер системного вызова из регистра `eax`. Поэтому перед вызовом прерывания необходимо поместить в тот регистр нужный номер. Кроме того, многим системным функциям требуется передавать какие-либо параметры. По принятым в ОС Linux правилам эти параметры помещаются в порядке следования в остальные регистры процессора: `ebx`, `ecx`, `edx`. Если системная функция должна вернуть значение, то она помещает его в регистр `eax`.

4 Выполнение лабораторной работы

4.1) Основы работы с Midnight Commander

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

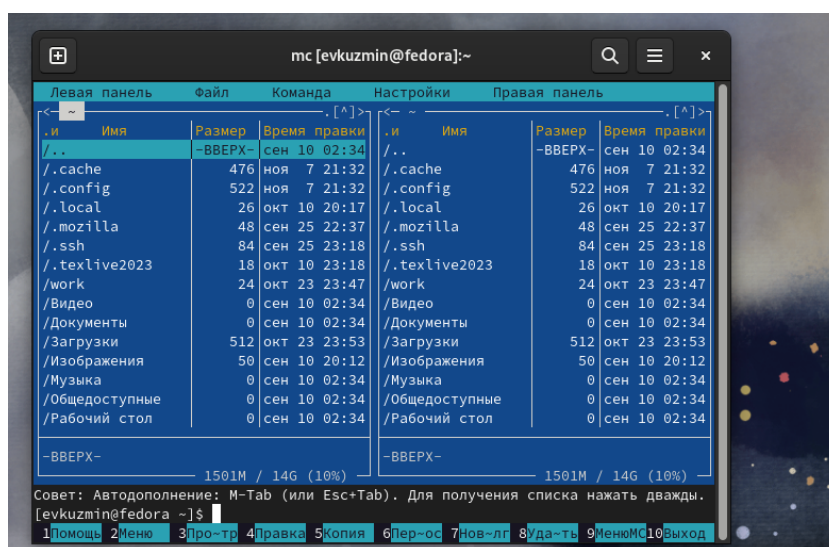


Рис. 4.1: Midnight Commander

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

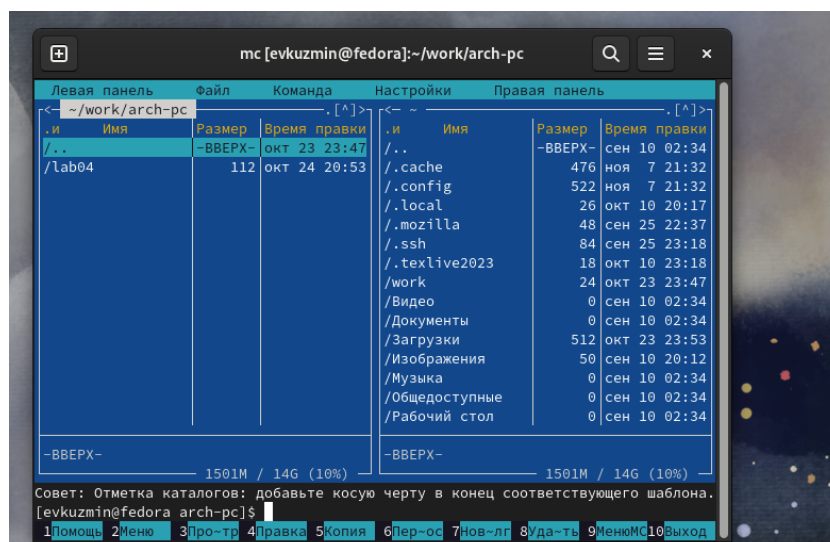


Рис. 4.2: Перемещение

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

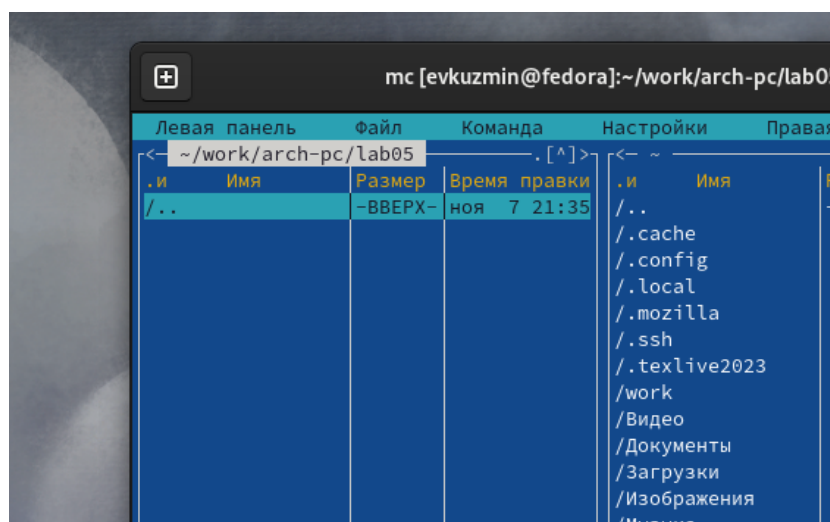


Рис. 4.3: Создание каталога

Перехожу в созданный каталог. (рис. 4.4).

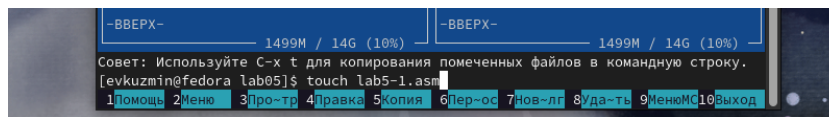


Рис. 4.4: Перемещение

В строке ввода прописываю команду `touch lab5-1.asm`, дабы создать соответствующий файл. (рис. 4.5).

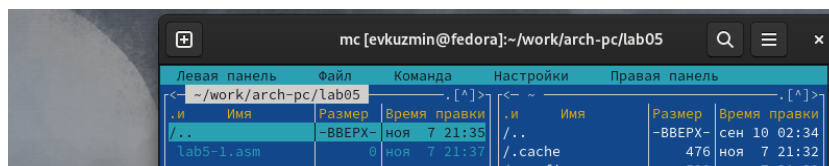


Рис. 4.5: Создание файла

С помощью функциональной клавиши F4 открываю созданный файл в режиме правки. Ввожу в файл код программы для запроса строки у пользователя, затем сохраняю изменения и выхожу из файла. (рис. 4.6).

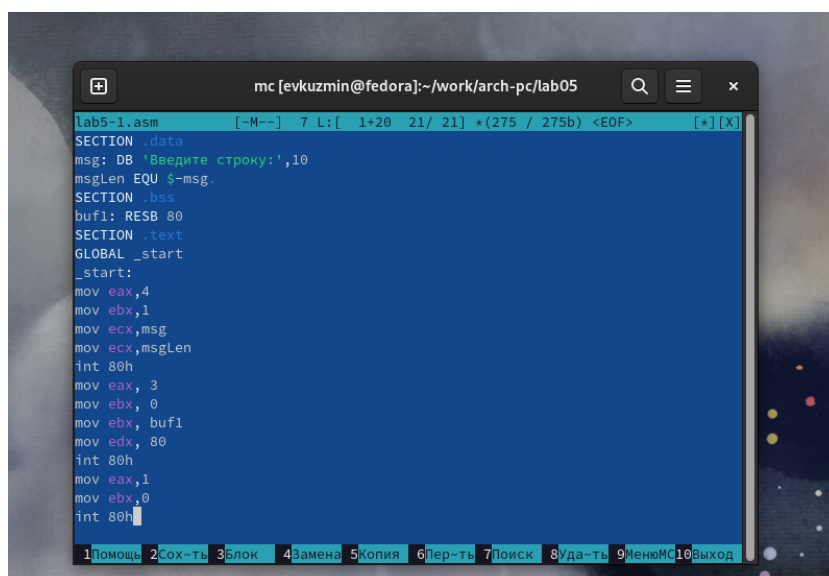


Рис. 4.6: Редактирование файла

Для проверки открываю файл с помощью функциональной клавиши F3 в ре-

жиме просмотра (рис. 4.7).

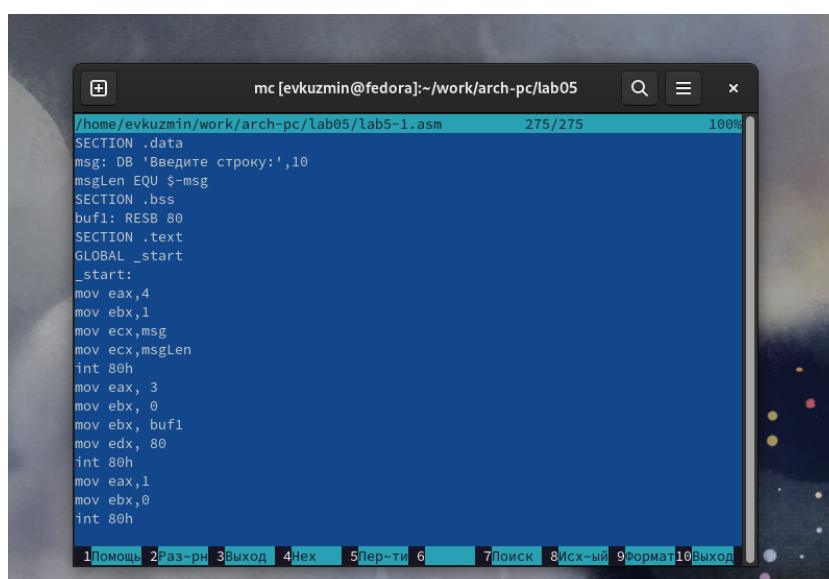


Рис. 4.7: Открытие файла для просмотра

Транслирую текст программы файла в объектный файл командой `nasm -f elf lab5-1.asm`. Создался объектный файл `lab5-1.o`. Выполняю компоновку объектного файла с помощью команды `ld -m elf_i386 -o lab5-1 lab5-1.o`. Создался исполняемый файл `lab5-1`. Запускаю исполняемый файл. Программа выводит “Введите строку.” и ждет ввода с клавиатуры, я ввожу свои ФИО, на этом программа заканчивает свою работу. (рис. 4.8).

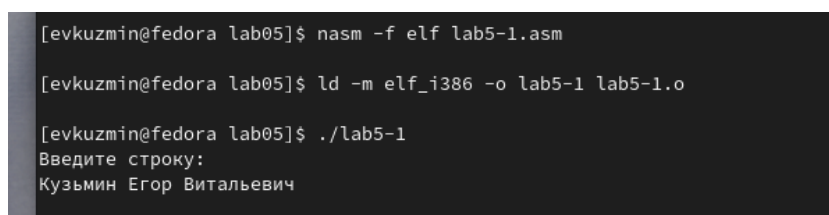


Рис. 4.8: Компиляция, обработка и запуск исполняемого файла

4.2) Подключение внешнего файла

Скачиваю файл `in_out.asm` со страницы курса в ТУИС. С помощью функциональной клавиши F5 копирую файл `in_out.asm` из каталога “Загрузки”, куда он

скачался, в созданный каталог lab05. (рис. 4.9).

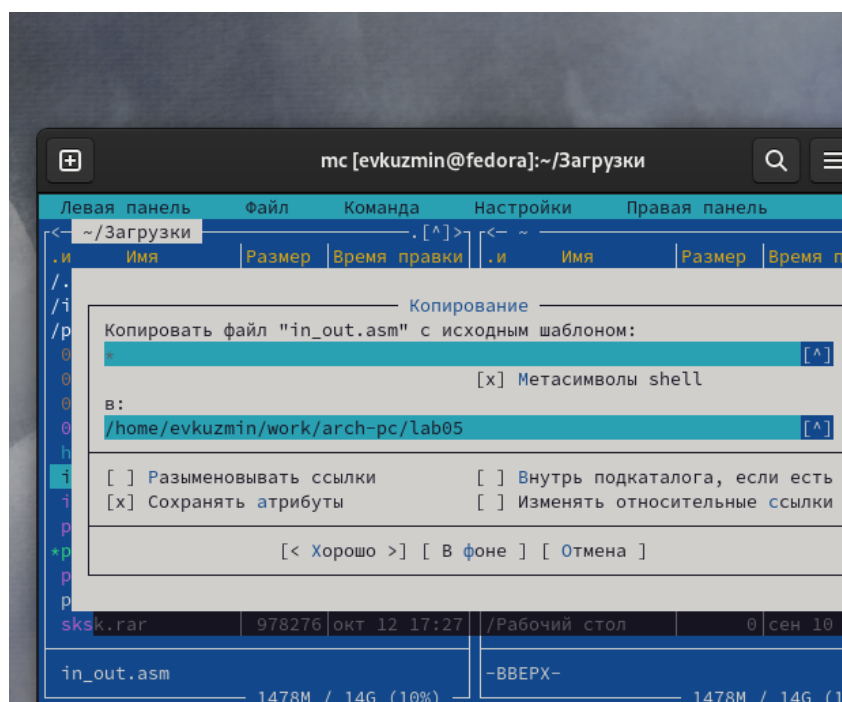


Рис. 4.9: Копирование файла

Далее копирую файл lab5-1 в тот же каталог, но с другим именем, для этого в окне mc прописываю путь к каталогу и новое имя файла. (рис. 4.10).

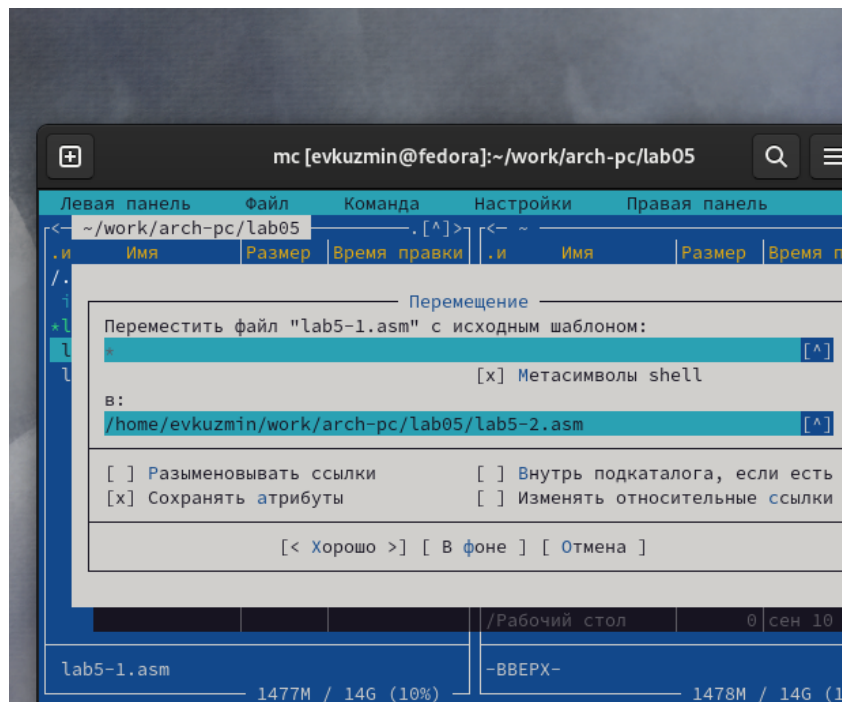


Рис. 4.10: Копирование файла

Изменяю содержимое файла lab5-2.asm во встроенном редакторе, чтобы в программе также использовался подключенный внешний файл in_out.asm. (рис. 4.11).

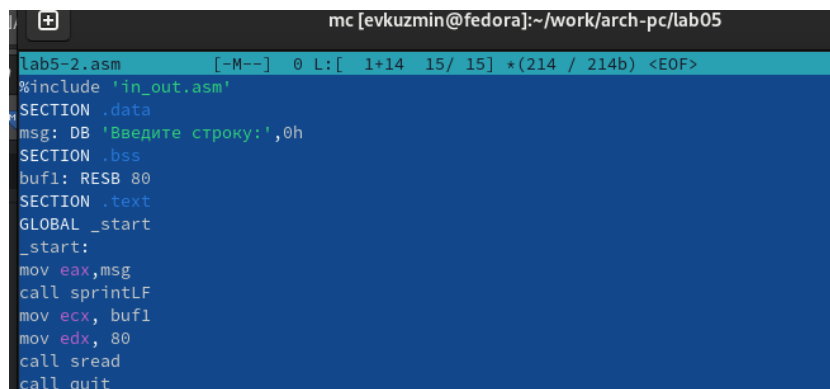


Рис. 4.11: Редактирование файла

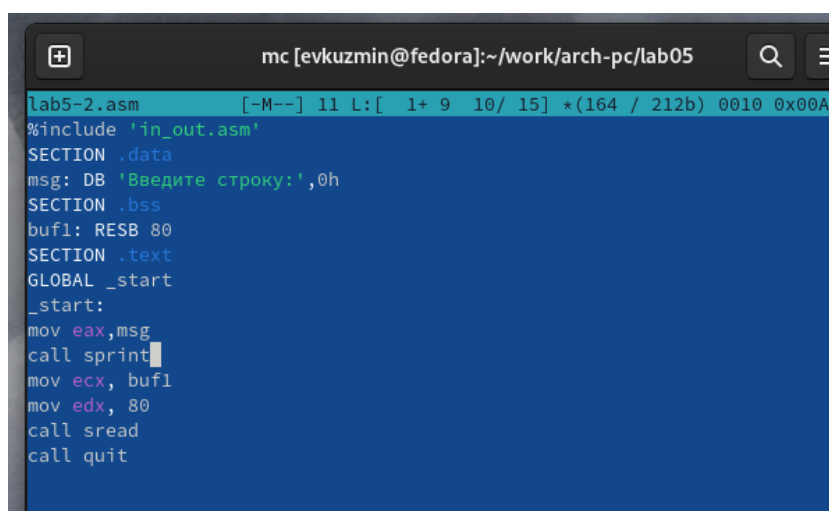
Транслирую текст программы файла в объектный файл командой `nasm -f elf lab5-2.asm`. Создался объектный файл lab5-2.o. Выполняю компоновку объектного

файла с помощью команды `ld -m elf_i386 -o lab5-2 lab5-2.o` Создался исполняемый файл `lab5-2`. Запускаю его. (рис. 4.12).

```
[evkuzmin@fedora lab05]$ nasm -f elf lab5-2.asm
[evkuzmin@fedora lab05]$ ld -m elf_i386 -o lab5-2 lab5-2.o
[evkuzmin@fedora lab05]$ ./lab5-2
Введите строку:
Кузьмин Егор Витальевич
```

Рис. 4.12: Компиляция, обработка и запуск исполняемого файла

Открываю файл `lab5-2.asm` в режиме редактирования, меняю в нем `sprintLF` на `sprint`. (рис. 4.13).



```
lab5-2.asm [-M--] 11 L: [ 1+ 9 10/ 15] *(164 / 212b) 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. 4.13: Редактирование файла

Сохраняю изменения и транслирую файл, выполняю компоновку созданного объектного файла, в конце концов запускаю новый исполняемый файл. (рис. 4.14).

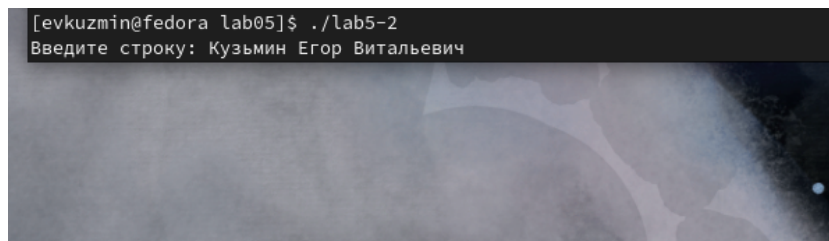


Рис. 4.14: Запуск исполняемого файла

Разница между первой и второй версией исполняемого файла состоит в том, что запуск первой запрашивает ввод с новой строки, а программа, которая исполняется при запуске второй, запрашивает ввод без переноса на новую строку, ибо в этом и заключается различие между подпрограммами `sprintLF` и `sprint`.

4.3) Выполнение заданий для самостоятельной работы

1. Создаю копию файла `lab5-1.asm` с именем `lab5-1-a.asm`. (рис. 4.15).

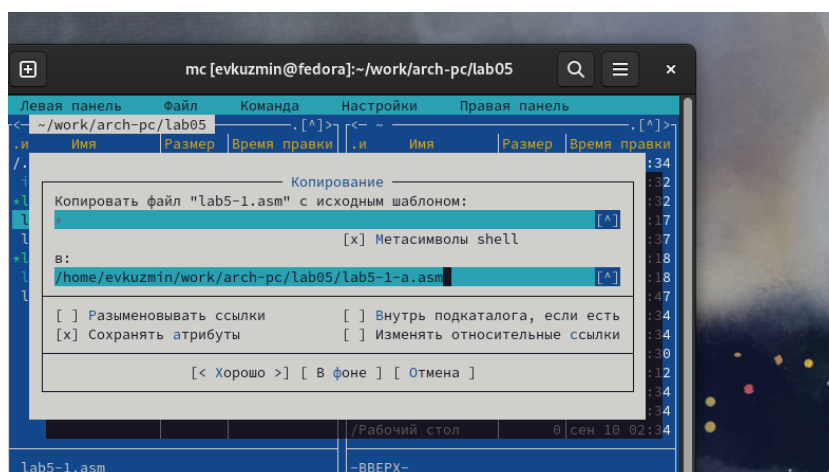


Рис. 4.15: Создание копии файла

Затем открываю созданный файл в режиме правки. Изменяю программу так, чтобы кроме вывода “Введите строку” и запроса ввода, она выводила вводимую пользователем строку. (рис. 4.16)

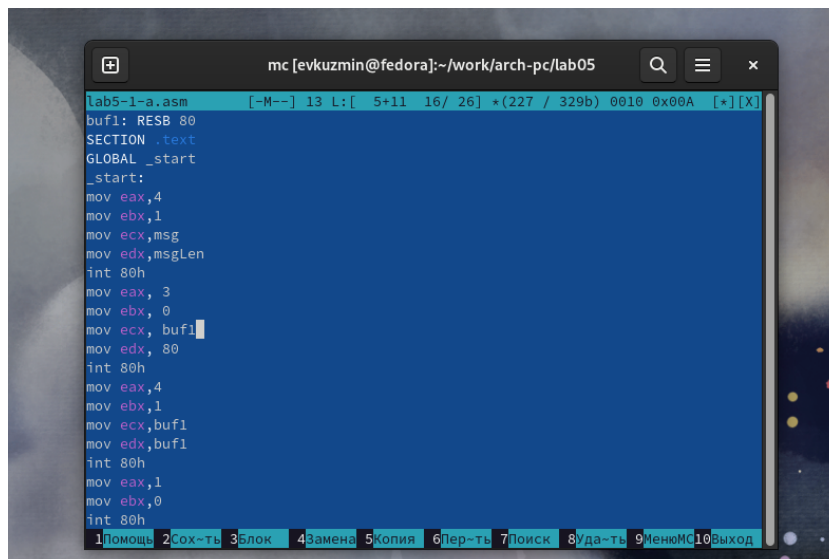


Рис. 4.16: Редактирование файла

2. Создаю объектный файл lab5-1-а.о, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-1-а, запускаю полученный исполняемый файл. Программа запрашивает ввод, ввожу свои ФИО, далее программа выводит введенные мною данные.(рис. 4.17)

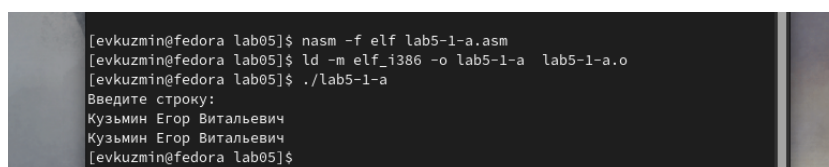


Рис. 4.17: Компиляция, обработка и запуск исполняемого файла

3. Создаю копию файла lab5-2.asm с именем lab5-2-а.asm. (рис. 4.18)

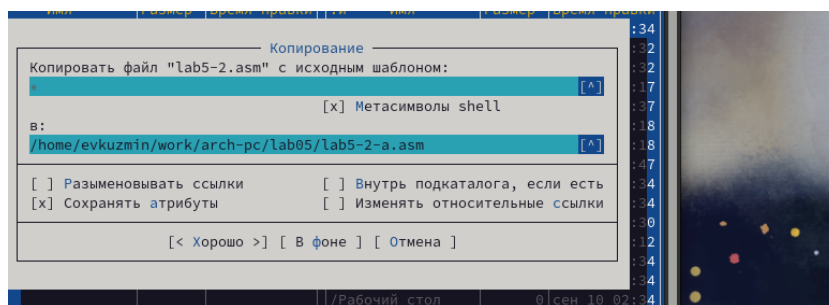


Рис. 4.18: Создание копии файла

Открываю созданный файл для редактирования. Изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку. (рис. 4.19)

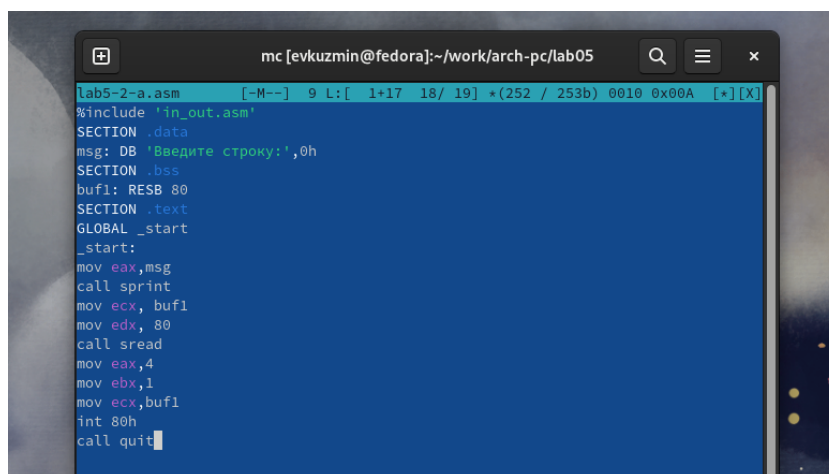
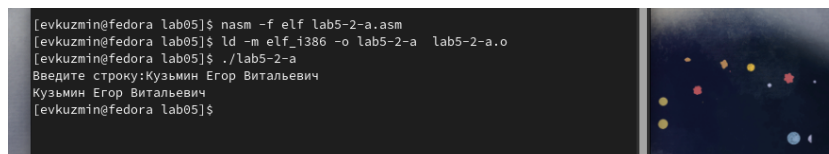


Рис. 4.19: Отправка файлов

4. Создаю объектный файл lab5-2-a.o, отдаю его на обработку компоновщику, получаю исполняемый файл lab5-2-a, запускаю полученный исполняемый файл. Затем программа запрашивает ввод без переноса на новую строку, ввожу свои ФИО, далее снова выводятся введенные мною данные. (рис. 4.20)



```
[evkuzmin@fedora lab05]$ nasm -f elf lab5-2-a.asm
[evkuzmin@fedora lab05]$ ld -m elf_i386 -o lab5-2-a lab5-2-a.o
[evkuzmin@fedora lab05]$ ./lab5-2-a
Введите строку:Кузьмин Егор Витальевич
Кузьмин Егор Витальевич
[evkuzmin@fedora lab05]$
```

Рис. 4.20: Компиляция, обработка и запуск исполняемого файла

5 Выводы

При выполнении лабораторной работы я приобрёл практический опыт работы с Midnight Commander, освоил инструкции языка ассемблера.

Список литературы

Архитектура ЭВМ