

# **Отчёт по лабораторной работе №5**

**Дисциплина: Архитектура Компьютера**

Дарина Андреевна Куокконен

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Открытие тс . . . . .	8
4.2	Перемещение между директориями . . . . .	9
4.3	Создание каталога . . . . .	9
4.4	Перемещение между директориями . . . . .	10
4.5	Открытый файл . . . . .	10
4.6	Редактирование файла . . . . .	11
4.7	Открытие файла . . . . .	12
4.8	Компиляция файла, передача компоновщику . . . . .	12
4.9	Исполнение файла . . . . .	12
4.10	Скаченный файл . . . . .	13
4.11	Копирование файла . . . . .	13
4.12	Редактирование файла . . . . .	14
4.13	Исполнение файла . . . . .	14
4.14	Редактирование файла . . . . .	15
4.15	Исполнение файла . . . . .	15
4.16	Копирование файла . . . . .	16
4.17	Редактирование файла . . . . .	16
4.18	Исполнение файла . . . . .	17
4.19	Копирование файла . . . . .	18
4.20	Редактирование файла . . . . .	19
4.21	Исполнение файла . . . . .	19

# 1 Цель работы

Целью моей лабораторной работы - приобретение практических навыков работы в Midnight Commander, а также освоение инструкций языка ассемблера `mov` и `int`.

## 2 Задание

1. Основы работы с тс
2. Структура программы на языке ассемблера NASM
3. Подключение внешнего файла
4. Выполнение заданий для самостоятельной работы

### 3 Теоретическое введение

Midnight Commander (или просто tc) — это программа, которая позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой, т.е. tc 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 размером 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 (четырёх- 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 `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).

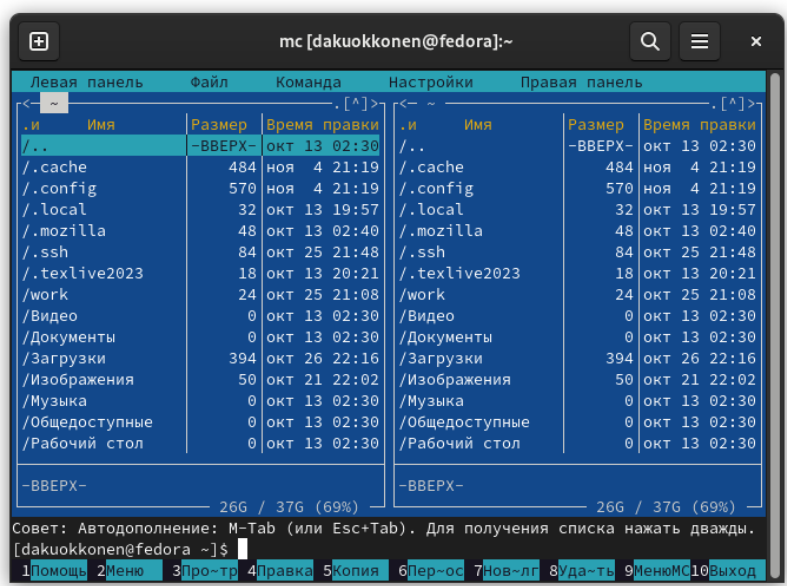


Рис. 4.1: Открытие mc

Перехожу в ранее созданный мной каталог, используя файловый менеджер mc (рис. 4.2).



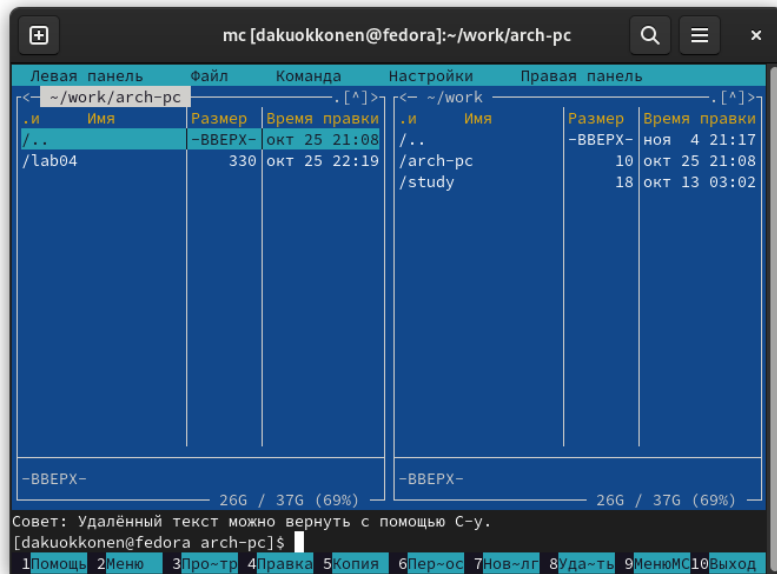


Рис. 4.2: Перемещение между директориями

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

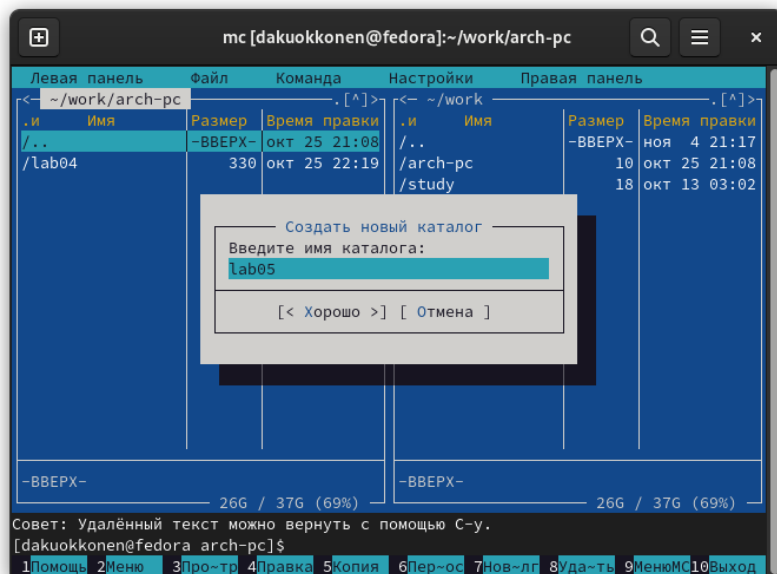


Рис. 4.3: Создание каталога

Перехожу в созданный каталог(рис. 4.4).

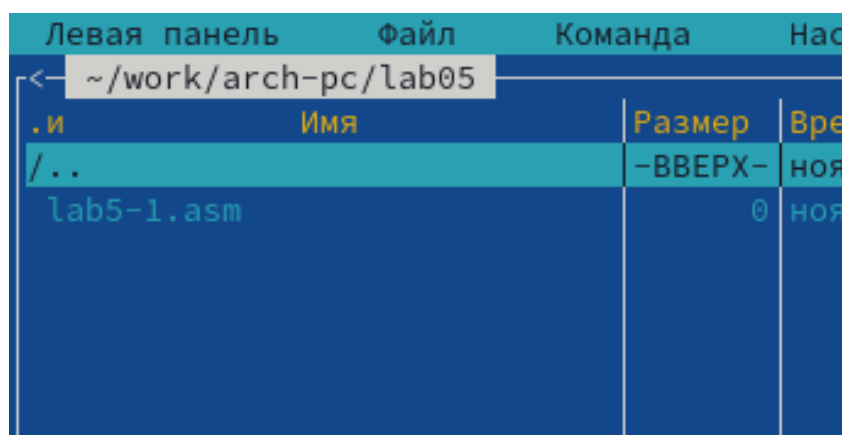


Рис. 4.4: Перемещение между директориями

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. 4.5).

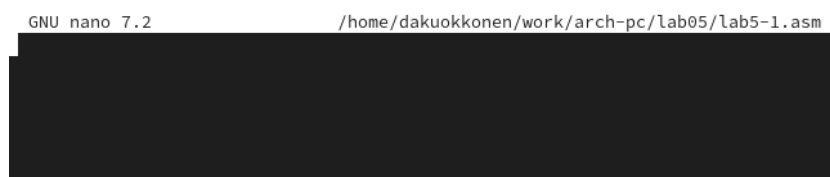


Рис. 4.5: Открытый файл

Ввожу в файл код программы для запроса строки у пользователя и выхожу с файла(рис. 4.6).

```

GNU nano 7.2 /home/dakuokkonen/work/arch-pc
SECTION .data ; Секция инициализированных данных
msg: DB 'Введите строку:',10 ; сообщение плюс
; символ перевода строки
msgLen: EQU $-msg ; Длина переменной 'msg'
SECTION .bss ; Секция не инициализированных 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,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

Рис. 4.6: Редактирование файла

С помощью функциональной клавиши F3 открываю файл для просмотра, чтобы проверить, содержит ли файл текст программы(рис. 4.7).

```

/home/dakuokkonen/work/arch-pc/lab05/lab5-1.asm
SECTION .data ; Секция инициированных данных
msg: DB 'Введите строку:',10 ; сообщение плюс
; символ перевода строки
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,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

Рис. 4.7: 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 lab5-1.o` (рис. 4.8).

```

[dakuokkonen@fedora lab05]$ nasm -f elf lab5-1.asm
[dakuokkonen@fedora lab05]$ ld -m elf_i386 -o lab5-1 lab5-1.o
[dakuokkonen@fedora lab05]$

```

Рис. 4.8: Компиляция файла, передача компоновщику

Запускаю исполняемый файл, ввожу свои ФИО в строке “Введите строку”. На этом программа заканчивает свою работу(рис. 4.9).

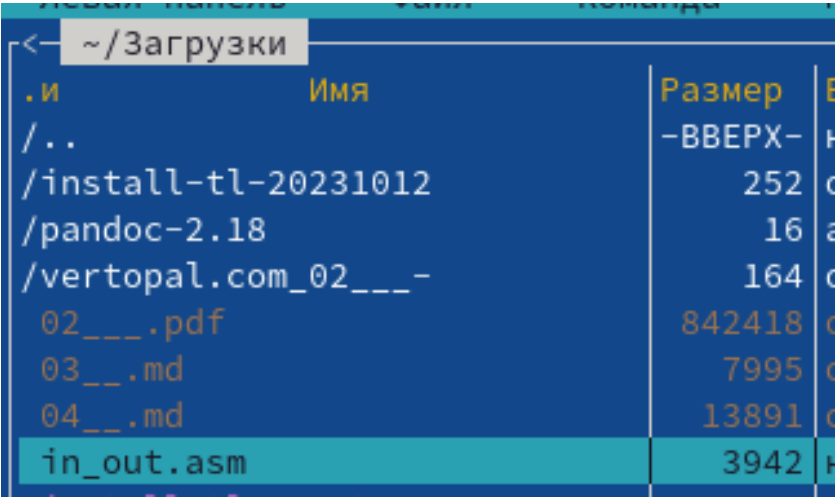
```

[dakuokkonen@fedora lab05]$ ./lab5-1
Введите строку:
Куокконен Дарина Андреевна

```

Рис. 4.9: Исполнение файла

4.3) Подключение внешнего файла Скачиваю файл in\_out.asm со страницы курса в ТУИС. Он сохранился в каталог “Загрузки”(рис. 4.10).



.и	Имя	Размер	Время
/..	-ВВЕРХ-		
/install-tl-20231012		252	0
/pandoc-2.18		16	а
/vertopal.com_02___-		164	о
02___-pdf		842418	о
03__-md		7995	о
04__-md		13891	о
in_out.asm		3942	н

Рис. 4.10: 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. 4.11).

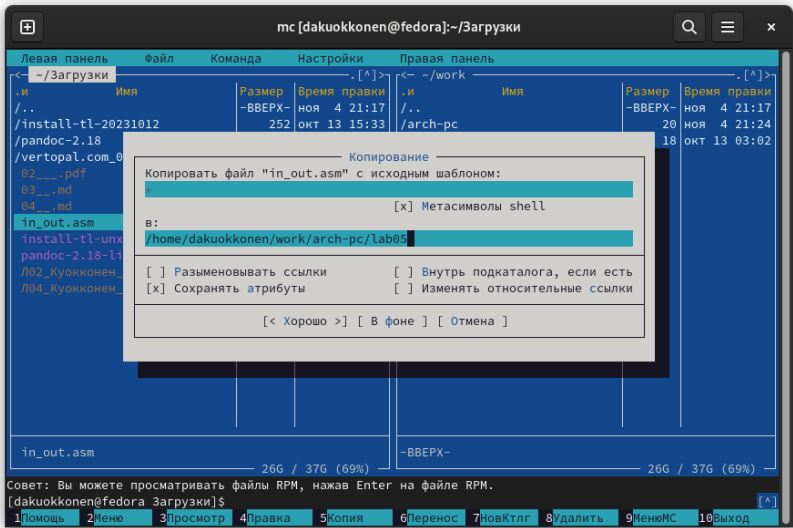
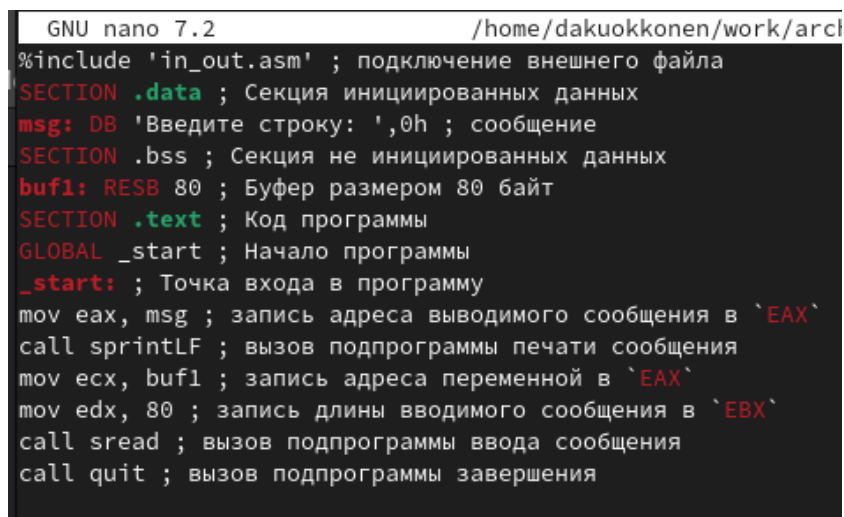


Рис. 4.11: Копирование файла

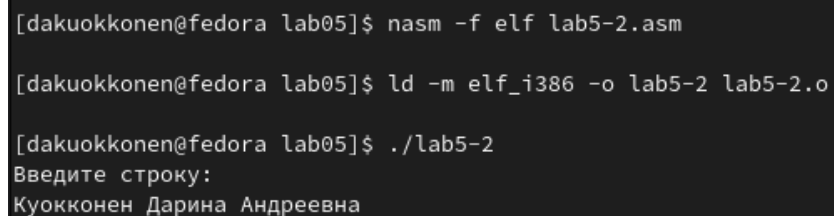
Изменяю содержимое файла, чтобы в программе использовались подпрограммы из внешнего файла `in_out.asm` (рис. 4.12).



```
GNU nano 7.2 /home/dakuokkonen/work/arc1
#include 'in_out.asm' ; подключение внешнего файла
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 sprintf ; 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
call quit ; вызов подпрограммы завершения
```

Рис. 4.12: Редактирование файла

Транслирую текст программы и выполняю компоновку объектного файла. Затем, запускаю исполняемый файл (рис. 4.13).



```
[dakuokkonen@fedora lab05]$ nasm -f elf lab5-2.asm
[dakuokkonen@fedora lab05]$ ld -m elf_i386 -o lab5-2 lab5-2.o
[dakuokkonen@fedora lab05]$ ./lab5-2
Введите строку:
Куокконен Дарина Андреевна
```

Рис. 4.13: Исполнение файла

Открываю файл `lab5-2.asm` для редактирования в `nano`. Изменяю в нем подпрограмму `sprintf` на `sprint` (рис. 4.14).

```
GNU nano 7.2 /home/dakuokkonen/work/arch-pc/1
#include 'in_out.asm' ; подключение внешнего файла
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 sprintf ; 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
call quit ; вызов подпрограммы завершения
```

Рис. 4.14: Редактирование файла

Транслирую файл, выполняю компоновку созданного объектного файла, запускаю файл(рис. 4.15).

```
[dakuokkonen@fedora lab05]$ nasm -f elf lab5-2.asm
[dakuokkonen@fedora lab05]$ ld -m elf_i386 -o lab5-2-2 lab5-2.o
[dakuokkonen@fedora lab05]$ ./lab5-2-2
Введите строку: Куокконен Дарина Андреевна
[dakuokkonen@fedora lab05]$
```

Рис. 4.15: Исполнение файла

Разница между первым и вторым файлами заключается в том, что запуск первого запрашивает ввод с новой строки, а программа, которая выполняется при запуске второго, запрашивает ввод без переноса на новую строку, потому что в этом заключается различие между подпрограммами `sprintf` 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. 4.16).

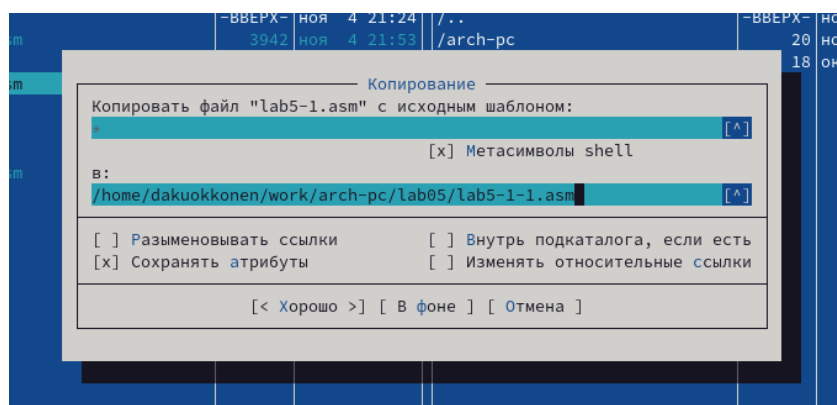


Рис. 4.16: Копирование файла

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. 4.17).

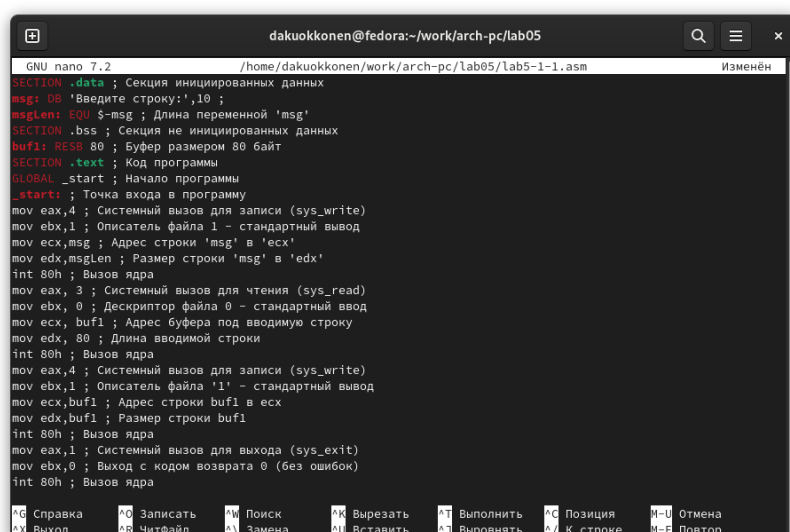


Рис. 4.17: Редактирование файла

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. 4.18).



```
[dakuokkonen@fedora lab05]$ nasm -f elf lab5-1-1.asm
[dakuokkonen@fedora lab05]$ ld -m elf_i386 -o lab5-1-1 lab5-1-1.o
[dakuokkonen@fedora lab05]$ ./lab5-1-1
Введите строку:
Куокконен Дарина Андреевна
Куокконен Дарина Андреевна
[dakuokkonen@fedora lab05]$
```

Рис. 4.18: 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

3. Создаю копию файла lab5-2.asm с именем lab5-2-1.asm с помощью функциональной клавиши F5(рис. 4.19).

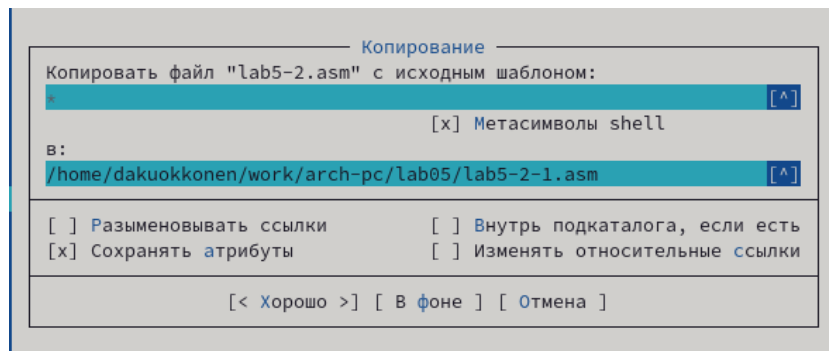


Рис. 4.19: Копирование файла

Открываю файл, изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку(рис. 4.20).

```

#include 'in_out.asm' ;
SECTION .data ; Секция иницированных данных
msg: DB 'Введите строку: ',0h ; сообщение
SECTION .bss ; Секция не иницированных 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

```

Рис. 4.20: 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. 4.21).

```

[dakuokkonen@fedora lab05]$ nasm -f elf lab5-2-1.asm
[dakuokkonen@fedora lab05]$ ld -m elf_i386 -o lab5-2-1 lab5-2-1.o
[dakuokkonen@fedora lab05]$ ./lab5-2-1
Введите строку: Куокконен Дарина Андреевна
Куокконен Дарина Андреевна
[dakuokkonen@fedora lab05]$

```

Рис. 4.21: Исполнение файла

Код программы из пункта 3:

```
%include 'in_out.asm' ;
```

```
SECTION .data ; Секция иницированных данных
```

```
msg: DB 'Введите строку:',0h ; сообщение
```

```
SECTION .bss ; Секция не иницированных 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