

# **Отчет по лабораторной работе №5**

**Архитектура компьютера**

Николенко Анна Николаевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Основы работы с тс . . . . .	9
4.2	Структура программы на языке ассемблера NASM . . . . .	10
4.3	Подключение внешнего файла in_out.asm . . . . .	12
<b>5</b>	<b>Выполнение заданий для самостоятельной работы</b>	<b>16</b>
<b>6</b>	<b>Выводы</b>	<b>23</b>

# Список иллюстраций

4.1	Открытый Midnight Commander . . . . .	9
4.2	Перемещение между директориями . . . . .	9
4.3	Создание каталога lab05 . . . . .	9
4.4	Перемещение между директориями . . . . .	10
4.5	Создание файла . . . . .	10
4.6	Открытие файла для редактирования . . . . .	11
4.7	Ввод кода, сохранение и выход . . . . .	11
4.8	Открытие файла для просмотра . . . . .	12
4.9	Компиляция файла и передача на обработку компоновщику . . . . .	12
4.10	Скачанный файл . . . . .	13
4.11	Копирование файла . . . . .	13
4.12	Копирование файла . . . . .	14
4.13	Редактирование файла . . . . .	14
4.14	Редактирование файла (замена подпрограммы <code>sprintLF</code> на <code>sprint</code> ) . . . . .	15
4.15	Исполнение файла . . . . .	15
5.1	Копирование файла . . . . .	16
5.2	Редактирование файла . . . . .	17
5.3	Исполнение файла . . . . .	17
5.4	Копирование файла . . . . .	18
5.5	Редактирование файла . . . . .	18
5.6	Исполнение файла . . . . .	18

## Список таблиц

# 1 Цель работы

Цель работы заключапется в приобретении практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера mov и int.

## 2 Задание

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

### 3 Теоретическое введение

Midnight Commander — один из файловых менеджеров с текстовым интерфейсом типа Norton Commander для UNIX-подобных операционных систем. MC позволяет просматривать структуру каталогов и выполнять основные операции по управлению файловой системой/ 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 (учетверённое 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.

Для определения строк принято использовать директиву DB в связи с особенностями хранения данных в оперативной памяти. Инструкция языка ассембле-

ра `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 `int n` предназначена 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]).

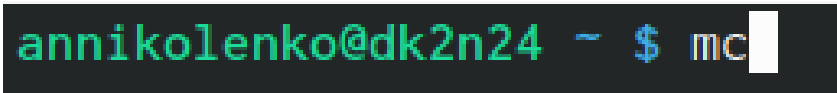


Рис. 4.1: Открытый Midnight Commander

Перехожу в каталог ~/work/study/2023-2024/Архитектура Компьютера/arch-рс, используя файловый менеджер mc (рис. [4.2]).

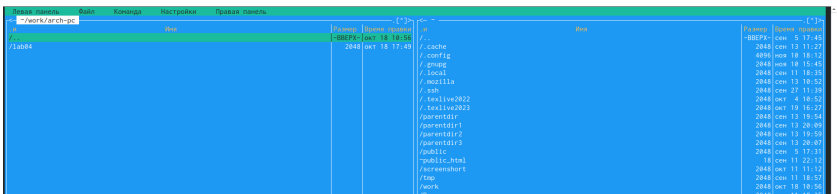


Рис. 4.2: Перемещение между директориями

С помощью функциональной клавиши F7 создаю каталог lab05 и перехожу в него (рис. [4.3]).

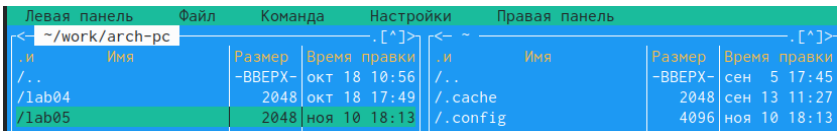


Рис. 4.3: Создание каталога lab05

Перехожу в этот каталог (рис. [4.4]).

Левая панель				Правая панель			
Файл		Команда		Настройки		Правая панель	
~/work/arch-pc/lab05		[< ^>		[< ^>		[< ^>	
Имя	Размер	Время	Правки	Имя	Размер	Время	Правки
./	-	ВВЕРХ	ноя 10 18:21	./local	2048	сен 11 18:35	
./				./mozilla	2048	сен 13 10:52	
				./ssh	2048	сен 27 11:39	

Рис. 4.4: Перемещение между директориями

В строке ввода создаю файл с помощью команды touch lab5-1.asm, в нем буду работать (рис. [4.5]).

Левая панель				Правая панель			
Файл		Команда		Настройки		Правая панель	
~/work/arch-pc/lab05		[< ^>		[< ^>		[< ^>	
Имя	Размер	Время	Правки	Имя	Размер	Время	Правки
./	-ВВЕРХ-	ноя 10 18:21		./local	2048	сен 11 18:35	
				./mozilla	2048	сен 13 10:52	
				./ssh	2048	сен 27 11:39	
				./texlive2022	2048	окт 4 10:52	
				./texlive2023	2048	окт 19 16:27	
				/parentdir	2048	сен 13 19:54	
				/parentdir1	2048	сен 13 20:09	
				/parentdir2	2048	сен 13 19:59	
				/parentdir3	2048	сен 13 20:07	
				/public	2048	сен 5 17:31	
				~public_html	18	сен 11 22:12	
				/screenshot	2048	окт 11 11:12	
				/tmp	2048	сен 11 18:57	
				/work	2048	окт 18 10:56	
				/Видео	2048	сен 11 18:35	
				/Документы	2048	сен 11 18:35	
				/Загрузки	2048	ноя 10 18:09	
				/Изображения	2048	сен 11 18:54	
				/Музыка	2048	сен 11 18:35	
				/Общедоступные	2048	сен 11 18:35	
				/Рабочий стол	2048	сен 11 18:35	
				/Шаблоны	2048	сен 11 18:35	
				/лаба 1 продолжение	2048	сен 13 11:50	
				.Xauthority	0	ноя 8 11:49	
-ВВЕРХ-				.Xauthority			
2048G / 2048G (100%)				2048G / 2048G (100%)			
Совет: Формат списка файлов может быть изменен; наберите "map mc" для деталей.							
annikolenko@dk2n24 ~/work/arch-pc/lab05 \$ touch lab5-1.asm							

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]).

Левая панель		Файл		Команда		Настройки		Правая панель			
< ~ /work/arch-pc/lab05						.[^>		< ~		.[^>	
Имя		Размер	Время правки	Имя		Размер	Время правки	Имя		Размер	Время правки
./		-ВВЕРХ-	ноя 10 18:21	./local		2048	сен 11 18:35				
lab5-1.asm		0	ноя 10 18:22	./mozilla		2048	сен 13 10:52				
				./		2048	сен 27 11:30				

Рис. 4.6: Открытие файла для редактирования

Ввожу в файл код программы для запроса строки у пользователя. Далее выхожу из файла (F10), предварительно сохранив изменения (F2). (рис. [4.7]).

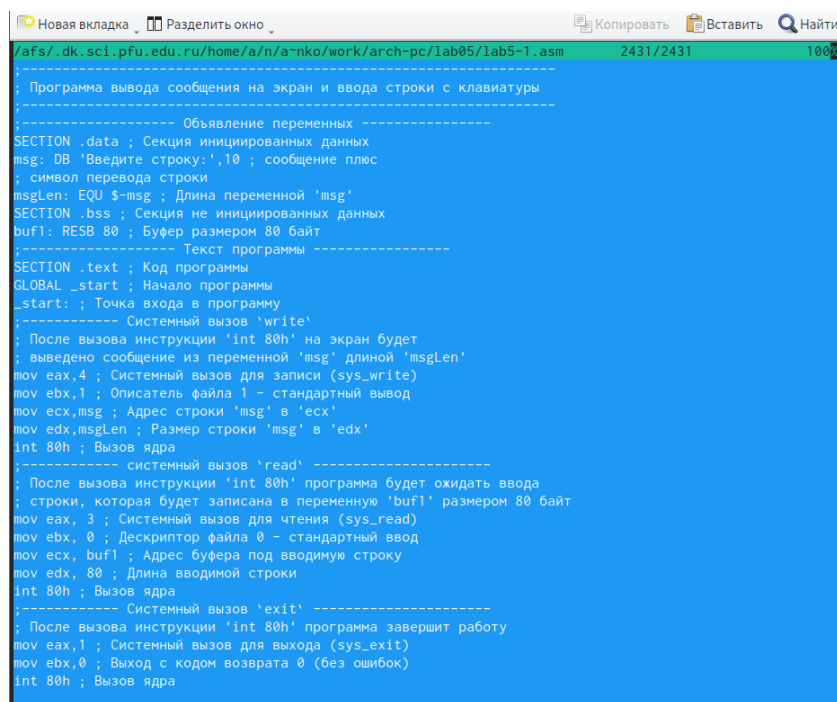
```

Новая вкладка  Разделить окно  Копировать  Вставить  Найти
lab5-1.asm  [-M--] 38 L:[ 1+ 9 10/ 35] *(712 /2431b) 0010 0x00A  [*][X]
;~~~~~
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;~~~~~
;~~~~~ Объявление переменных ~~~~~
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
;~~~~~ Текст программы ~~~~~
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
;~~~~~ Системный вызов 'write' ~~~~~
; После вызова инструкции 'int 80h' на экран будет
; выведено сообщение из переменной 'msg' длиной 'msglen'
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описание файла 1 - стандартный вывод
mov ecx,msg ; Адрес строки 'msg' в 'ecx'
mov edx,msglen ; Размер строки 'msg' в 'edx'
int 80h ; Вызов ядра
;~~~~~ Системный вызов 'read' ~~~~~
; После вызова инструкции 'int 80h' программа будет ожидать ввода
; строки, которая будет записана в переменную 'buf1' размером 80 байт
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
;~~~~~ Системный вызов 'exit' ~~~~~
; После вызова инструкции 'int 80h' программа завершит работу
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: Ввод кода, сохранение и выход

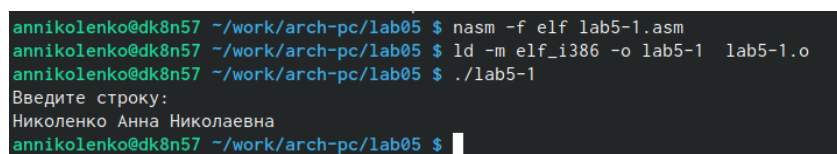
С помощью функциональной клавиши F3 открываю файл для просмотра, чтобы проверить, содержит ли файл текст программы (рис. [4.8]).



```
-----  
; Программа вывода сообщения на экран и ввода строки с клавиатуры  
-----  
----- Объявление переменных -----  
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  
----- Текст программы -----  
SECTION .text ; Код программы  
GLOBAL _start ; Начало программы  
_start: ; Точка входа в программу  
----- Системный вызов 'write' -----  
; После вызова инструкции 'int 80h' на экран будет  
; выведено сообщение из переменной 'msg' длиной 'msgLen'  
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  
----- системный вызов 'read' -----  
; После вызова инструкции 'int 80h' программа будет ожидать ввода  
; строки, которая будет записана в переменную 'buf1' размером 80 байт  
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  
----- Системный вызов 'exit' -----  
; После вызова инструкции 'int 80h' программа завершит работу  
mov eax,1 ; Системный вызов для выхода (sys_exit)  
mov ebx,0 ; Выход с кодом возврата 0 (без ошибок)  
int 80h ; Вызов ядра
```

Рис. 4.8: 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`. Создался исполняемый файл `lab5-1`. На запрос ввожу свою ФИО (рис. [4.9]).



```
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1.asm  
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1 lab5-1.o  
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ./lab5-1  
Введите строку:  
Николенко Анна Николаевна  
annikolenko@dk8n57 ~/work/arch-pc/lab05 $
```

Рис. 4.9: Компиляция файла и передача на обработку компоновщику

## 4.3 Подключение внешнего файла `in_out.asm`

Скачиваю файл `in_out.asm` со страницы курса в ТУИС. Он сохранился в каталог “Загрузки” (рис. [4.10]).

Правая панель			
<- ~/Загрузки .[^]>			
.и	Имя	Размер	Время правки
/..		-ВВЕРХ-	ноя 11 10:28
5,1		31649	сен 13 18:09
hello(1).asm		338	окт 18 17:16
hello.asm		338	ноя 10 16:04
in_out.asm		3942	ноя 11 10:50
report.docx		202588	ноя 10 17:42
Тема 6. Матер~х занятий.doc		104448	ноя 10 17:06

Рис. 4.10: Скачанный файл

С помощью функциональной клавиши F5 копирую файл in\_out.asm из каталога Загрузки в созданный каталог lab05 (рис. [4.11]).

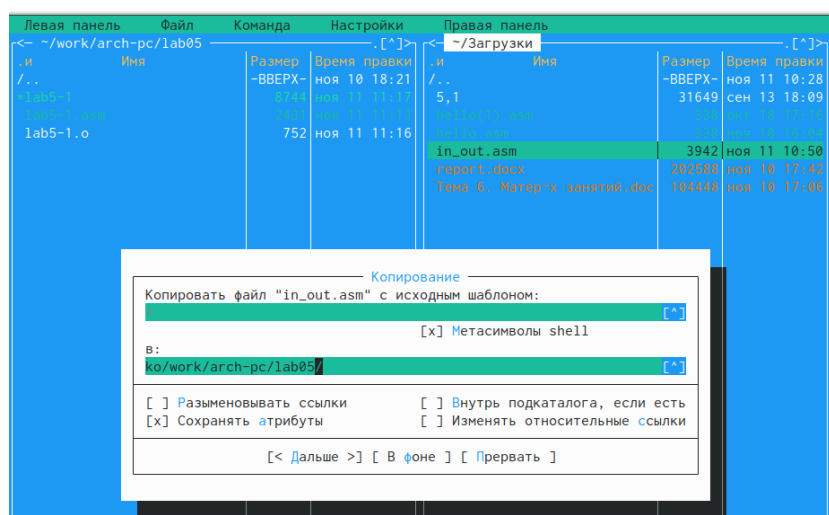


Рис. 4.11: Копирование файла

С помощью функциональной клавиши F6 создаю копию файла lab5-1.asm с именем lab5-2.asm (рис. [4.12]).

Перемещение

Переместить файл "lab5-1.asm" с исходным шаблоном:

[x] Метасимволы shell

В:

ko/work/arch-pc/lab05/lab5-2.asm

<input type="checkbox"/> Разыменовывать ссылки <input checked="" type="checkbox"/> Сохранять атрибуты	<input type="checkbox"/> Внутрь подкаталога, если есть <input type="checkbox"/> Изменять относительные ссылки
--	--

Рис. 4.12: Копирование файла

Изменяю содержимое файла lab5-2.asm во встроенном редакторе mcedit, чтобы в программе использовались подпрограммы из внешнего файла in\_out.asm (рис. [4.13]).

```
lab5-2.asm [-M--] 41 L: [ 1+16 17/ 17] *(1224/1224b) <EOF> [*][X]
; Программа вывода сообщения на экран и ввода строки с клавиатуры
%include "in_out.asm" ; подключение внешнего файла
SECTION .data ; Секция инициализированных данных
msg: DB "Введите строку: ", 0h ; сообщение
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, msg ; запись адреса выводимого сообщения в 'EAX'
call sprintf ; вызов подпрограммы печати сообщения
mov ecx, buf1 ; запись адреса переменной в 'ECX'
mov edx, 80 ; запись длины вводимого сообщения в 'EDX'
call sread ; вызов подпрограммы ввода сообщения
call quit ; вызов подпрограммы завершения
```

Рис. 4.13: Редактирование файла

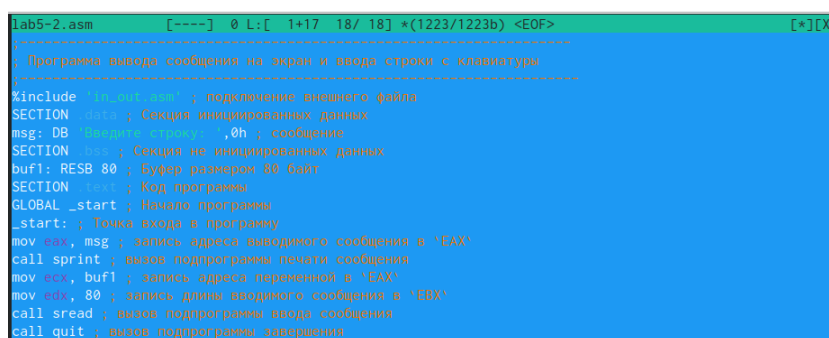
Транслирую текст программы файла в объектный файл командой `nasm -f elf lab5-2.asm` (рис. [??]). Создался объектный файл lab5-2.o. Выполняю компоновку объектного файла с помощью команды `ld -m elf_i386 -o lab5-2 lab5-2.o` Создался исполняемый файл lab5-2. Запускаю исполняемый файл, чтобы проверить его работу (рис. [??]).

```
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2.asm
```

```
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ld -m
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ./lab
Введите строку:
Николенко Анна Николаевна
annikolenko@dk8n57 ~/work/arch-pc/lab05 $
```

Открываю файл lab5-2.asm для редактирования в mcedit функциональной клавишей F4. Изменяю в нем подпрограмму `sprintf` на `sprint`. Сохраняю изменения

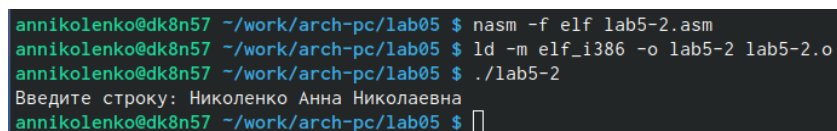
(рис. [4.14]).



```
lab5-2.asm [----] 0 L: [ 1+17 18/ 18] *(1223/1223b) <EOF> [*][X]
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;-----
%include "lab5-2.inc" ; подключение внешнего файла
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 ecx, 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
call quit ; вызов подпрограммы завершения
```

Рис. 4.14: Редактирование файла (замена подпрограммы sprintLF на sprint)

Транслирую файл, выполняю компоновку созданного объектного файла, запускаю новый исполняемый файл (рис. [4.15]).



```
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2.asm
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-2 lab5-2.o
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ./lab5-2
Введите строку: Николенко Анна Николаевна
annikolenko@dk8n57 ~/work/arch-pc/lab05 $
```

Рис. 4.15: Исполнение файла

Разница между первым исполняемым файлом и вторым в том, что запуск первого запрашивает ввод с новой строки, а при запуске второго, запрашивает ввод без переноса на новую строку, потому что в этом заключается различие между подпрограммами sprintLF и sprint.

## 5 Выполнение заданий для самостоятельной работы

Создаю копию файла lab5-1.asm с именем lab5-1-1.asm с помощью функциональной клавиши F5 (рис. [5.1]).

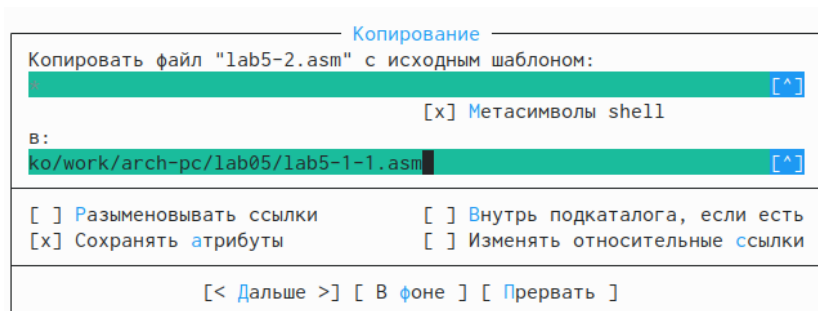


Рис. 5.1: Копирование файла

Предварительно открыв созданный файл, изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку (без использования внешнего файла in\_out.asm) (рис. [5.2]).



```

lab5-1-1.asm      [----]  0 L: [ 1+40  41/ 41] *(2571/2571b) <EOF>      [*][X]
; Программа вывода сообщения на экран и ввода строки с клавиатуры
;----- Объявление переменных -----
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
;----- Текст программы -----
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
;----- Системный вызов 'write' -----
; После вызова инструкции 'int 80h' на экран будет
; выведено сообщение из переменной 'msg' длиной 'msgLen'
mov eax, 4 ; Системный вызов для записи (sys_write)
mov ebx, 1 ; Описание файла 1 — стандартный вывод
mov ecx, msg ; Адрес строки 'msg' в 'ecx'
mov edx, msgLen ; Размер строки 'msg' в 'edx'
int 80h ; Вызов ядра
;----- Системный вызов 'read' -----
; После вызова инструкции 'int 80h' программа будет ожидать ввода
; строки, которая будет записана в переменную 'buf1' размером 80 байт
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
;----- Системный вызов 'exit' -----
; После вызова инструкции 'int 80h' программа завершит работу
mov eax, 4 ; Системный вызов для записи
mov ebx, 1 ; Описание файла
mov ecx, buf1
mov edx, buf1
int 80h
mov eax, 1 ; Системный вызов для выхода (sys_exit)
mov ebx, 0 ; Выход с кодом возврата 0 (без ошибок)
int 80h ; Вызов ядра

```

Рис. 5.2: Редактирование файла

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. [5.3]).

```

annikolenko@dk8n57 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1-1.asm
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1-1 lab5-1-1.o
annikolenko@dk8n57 ~/work/arch-pc/lab05 $ ./lab5-1-1
Введите строку:
Николенко Анна Николаевна
Николенко Анна Николаевна
annikolenko@dk8n57 ~/work/arch-pc/lab05 $

```

Рис. 5.3: Исполнение файла

Создаю копию файла lab5-2.asm под названием lab5-2-1.asm. (рис. [5.4]).

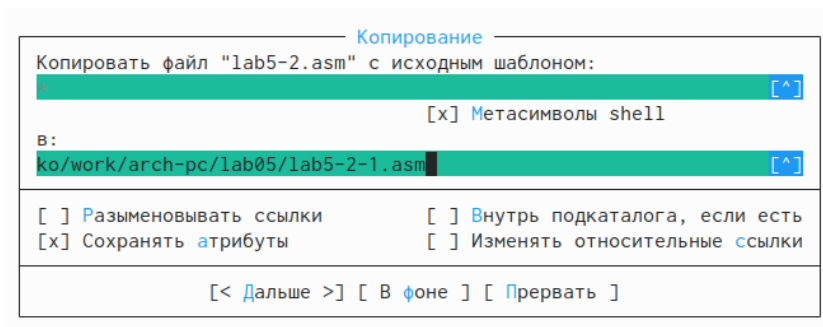


Рис. 5.4: Копирование файла

Предварительно открыв созданный файл, изменяю программу так, чтобы кроме вывода приглашения и запроса ввода, она выводила вводимую пользователем строку с использованием под-программ из внешнего файла in\_out.asm (рис. [5.4]).

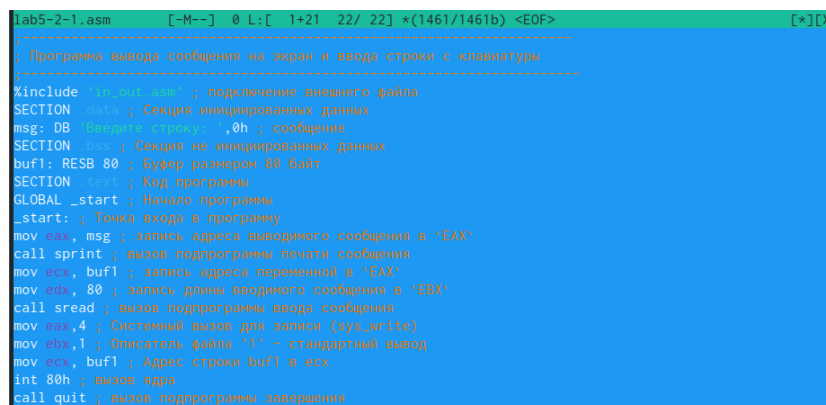


Рис. 5.5: Редактирование файла

Создаю исполняемый файл и проверяю его работу (рис. [5.6]).

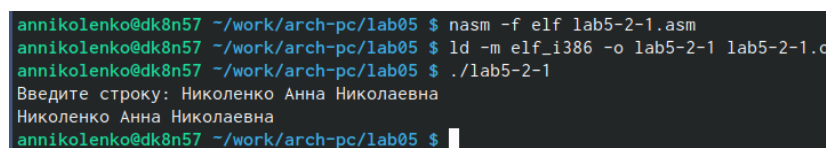


Рис. 5.6: Исполнение файла

#Листинги написанных программ

## 1. Программа вывода сообщения на экран и ввода строки с клавиатуры

```
;----- ; Программа вывода сообщения на
экран и ввода строки с клавиатуры ;-----
;----- Объявление переменных ----- SECTION .data ; Секция иници-
цированных данных msg: DB 'Введите строку:',10 ; сообщение плюс ; символ
перевода строки msgLen: EQU $-msg ; Длина переменной 'msg' SECTION .bss
; Секция не иницированных данных buf1: RESB 80 ; Буфер размером 80 байт
;----- Текст программы ----- SECTION .text ; Код программы GLOBAL
_start ; Начало программы _start: ; Точка входа в программу ;----- Системный
вызов write ; После вызова инструкции 'int 80h' на экран будет ; выведено со-
общение из переменной 'msg' длиной 'msgLen' 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 ;----- системный вызов read ----- ; После вызова
инструкции 'int 80h' программа будет ожидать ввода ; строки, которая будет
записана в переменную 'buf1' размером 80 байт 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 ;----- Системный вызов exit ----- ; После вызова
инструкции 'int 80h' программа завершит работу mov eax,1 ; Системный вызов
для выхода (sys_exit) mov ebx,0 ; Выход с кодом возврата 0 (без ошибок) int 80h ;
Вызов ядра
```

## 2. внешнего файла in\_out.asm

```
;----- slen ----- ; Функция вычисления длины сообщения slen:
push ebx
mov ebx, eax
nextchar:
cmp byte [eax], 0
```

jz finished

inc eax

jmp nextchar

finished: sub eax, ebx pop ebx

ret

;----- sprint ----- ; Функция печати сообщения ; входные данные:

mov eax, sprint: push edx push ecx push ebx push eax call slen

mov edx, eax

pop eax

mov ecx, eax

mov ebx, 1

mov eax, 4

int 80h

pop ebx

pop ecx

pop edx

ret

;----- sprintLF ----- ; Функция печати сообщения с переводом строки

; входные данные: mov eax, sprintLF: call sprint

push eax

mov eax, 0AH

push eax

mov eax, esp

call sprint

pop eax

pop eax

ret

;----- sread ----- ; Функция считывания сообщения

3. Программа вывода сообщения на экран и ввода строки с клавиатуры с использованием файла in\_out.asm

;----- ; Программа вывода сообщения на экран и ввода строки с клавиатуры ;-----  
%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 sprintLF ; вызов подпрограммы печати сообщения mov ecx, buf1 ; запись адреса переменной в EAX mov edx, 80 ; запись длины вводимого сообщения в EBX 56 Демидова А. В. Архитектура ЭВМ call sread ; вызов подпрограммы ввода сообщения call quit ; вызов подпрограммы завершения

4. Программа вывода сообщения на экран и ввода строки с клавиатуры без переноса на новую строку с использованием файла in\_out.asm (замена подпрограммы sprintLF на sprint)

;----- ; Программа вывода сообщения на экран и ввода строки с клавиатуры ;-----  
%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 sprint ; вызов подпрограммы печати сообщения mov ecx, buf1 ; запись адреса переменной в EAX mov edx, 80 ; запись длины вводимого сообщения в

EBX 56 Демидова А. В. Архитектура ЭВМ call sread ; вызов подпрограммы ввода сообщения call quit ; вызов подпрограммы завершения

5. Код программы из пункта 1 самостоятельной работы:

SECTION .data ; Секция инициированных данных msg: DB 'Введите строку:',10  
msgLen: EQU \$-msg ; Длина переменной 'msg' SECTION .bss ; Секция не иници-  
ированных 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

6. Код программы из пункта 3 самостоятельной работы:

%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

## 6 Выводы

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