

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

по лабораторной работе № 5

дисциплина:      *Архитектура компьютера*

Студент: Сачковская София Александровна

Группа: НКАбд-06-25

МОСКВА

2025 г.

## **Содержание**

1. Цель работы
2. Теоретическое введение
3. Выполнение лабораторной работы
4. Выполнение самостоятельных заданий
5. Выводы

## **Список иллюстраций:**

<b>Рис 3.1.....</b>	<b>8</b>
<b>Рис 3.2.....</b>	<b>8</b>
<b>Рис 3.3.....</b>	<b>9</b>
<b>Рис 3.4.....</b>	<b>9</b>
<b>Рис 3.5.....</b>	<b>9</b>
<b>Рис 3.6.....</b>	<b>10</b>
<b>Рис 3.7.....</b>	<b>11</b>
<b>Рис 3.8.....</b>	<b>11</b>
<b>Рис 3.9.....</b>	<b>12</b>
<b>Рис 3.10.....</b>	<b>12</b>
<b>Рис 3.11.....</b>	<b>13</b>
<b>Рис 3.12.....</b>	<b>13</b>
<b>Рис 3.13.....</b>	<b>14</b>
<b>Рис 3.14.....</b>	<b>14</b>
<b>Рис 3.15.....</b>	<b>14</b>
<b>Рис 4.1.....</b>	<b>15</b>
<b>Рис 4.2.....</b>	<b>15</b>
<b>Рис 4.3.....</b>	<b>16</b>
<b>Рис 4.4.....</b>	<b>16</b>

## **1.Цель работы**

Приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера mov и int.

## 2. Теоретическое введение

### Основы работы с Midnight Commander

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. Для активации оболочки Midnight Commander достаточно ввести в командной строке *mc* и нажать клавишу *Enter*. В Midnight Commander используются функциональные клавиши F1 — F10, к которым привязаны часто выполняемые операции.

Следующие комбинации клавиш облегчают работу с Midnight Commander:

- **Tab** используется для переключения между панелями;
  - **↑ и ↓** используется для навигации, **Enter** для входа в каталог или открытия файла (если в файле расширений *mc.ext* заданы правила связи определённых расширений файлов с инструментами их запуска или обработки);
  - **Ctrl + u** (или через меню **Команда > Переставить панели**) меняет местами содержимое правой и левой панелей;
  - **Ctrl + o** (или через меню **Команда > Отключить панели**) скрывает или возвращает панели Midnight Commander, за которыми доступен для работы командный интерпретатор оболочки и выводимая туда информация.
  - **Ctrl + x + d** (или через меню **Команда > Сравнить каталоги**) позволяет сравнить содержимое каталогов, отображаемых на левой и правой панелях.
- Дополнительную информацию о Midnight Commander можно получить по команде `man mc`

## Структура программы на языке ассемблера NASM

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. Синтаксис директив определения данных следующий:

*<имя> DB <операнд> [, <операнд>] [, <операнд>]*

Для объявления неинициированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти.

## Элементы программирования

### Описание инструкции `mov`

Инструкция языка ассемблера `mov` предназначена для дублирования данных источника в приёмнике. В общем виде эта инструкция записывается в виде

*`mov dst,src`*

Здесь операнд `dst` — приёмник, а `src` — источник. В качестве операнда могут выступать регистры (`register`), ячейки памяти (`memory`) и непосредственные значения (`const`).

**ВАЖНО!** Переслать значение из одной ячейки памяти в другую нельзя, для этого необходимо использовать две инструкции `mov`:

*`mov eax, x`*

*`mov y, eax`*

Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Использование следующих примеров приведет к ошибке:

- *`mov al,1000h`* — ошибка, попытка записать 2-байтное число в 1-байтный регистр;
- *`mov eax,cx`* — ошибка, размеры операндов не совпадают.

### Описание инструкции `int`

Инструкция языка ассемблера `int` предназначена для вызова прерывания с указанным номером. В общем виде она записывается в виде

*`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).

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 этот 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`.

### **Системные вызовы для обеспечения диалога с пользователем**

Простейший диалог с пользователем требует наличия двух функций — вывода текста на экран и ввода текста с клавиатуры. Простейший способ вывести строку на экран — использовать системный вызов `write`. Этот системный вызов имеет номер 4, поэтому перед вызовом инструкции `int` необходимо поместить значение 4 в регистр `eax`. Первым аргументом `write`, помещаемым в регистр `ebx`, задаётся дескриптор файла. Для вывода на экран в качестве дескриптора файла нужно указать 1 (это означает «стандартный вывод», т. е. вывод на экран). Вторым аргументом задаётся адрес выводимой строки (помещаем его в регистр `ecx`, например, инструкцией `mov ecx, msg`). Строка может иметь любую длину. Последним аргументом (т.е. в регистре `edx`) должна задаваться максимальная длина выводимой строки. Для ввода строки с клавиатуры можно использовать аналогичный системный вызов `read`. Его аргументы — такие же, как у вызова `write`, только для «чтения» с клавиатуры используется файловый дескриптор 0 (стандартный ввод). Системный вызов `exit` является обязательным в конце любой программы на языке ассемблер. Для обозначения конца программы перед вызовом инструкции `int 80h` необходимо поместить в регистр `eax` значение 1, а в регистр `ebx` код завершения 0.

### 3.Выполнение лабораторной работы

1. Откроем Midnight Commander

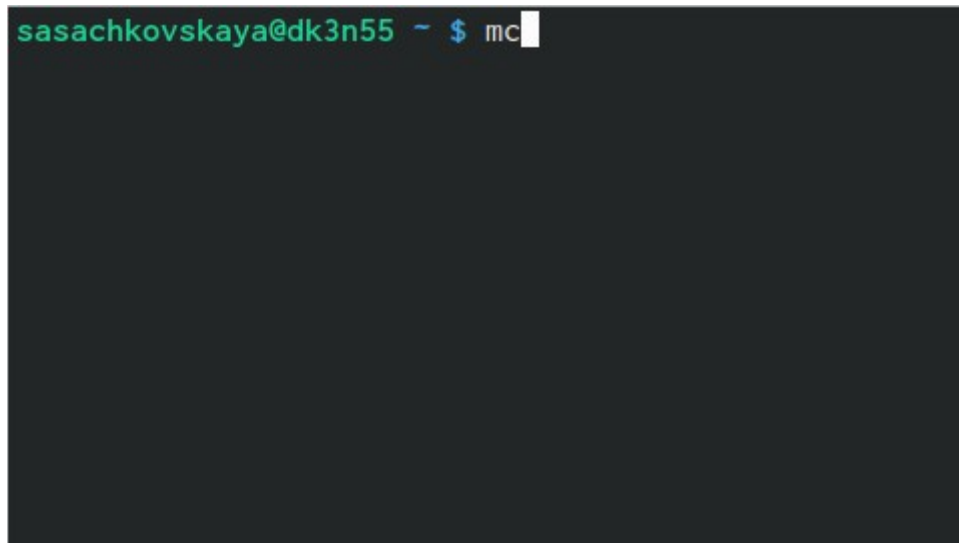


Рис.3.1

2.Пользуясь клавишами ↑ , ↓ и Enter перейдём в каталог ~/work/arch-pc созданный при выполнении лабораторной работы №4 (рис. 3.2).

Левая панель	Файл	Команда	Настройки	Г
< ~/work/arch-pc .[^]>				
.и	Имя	Размер	Дата правки	
/..		-ВВЕРХ-	окт 24 17:58	
/lab04		2048	окт 24 18:42	

Рис.3.2

3. С помощью функциональной клавиши F7 создадим папку lab05 (рис. 3.3) и перейдём в созданный каталог.

Левая панель	Файл	Команда	Настройки
< ~ /work/arch-pc .[^]>			
.и	Имя	Размер	Дата правки
/..		-ВВЕРХ-	окт 24 17:58
/lab04		2048	окт 24 18:42
/lab05		2048	ноя 6 18:25

Рис.3.3.

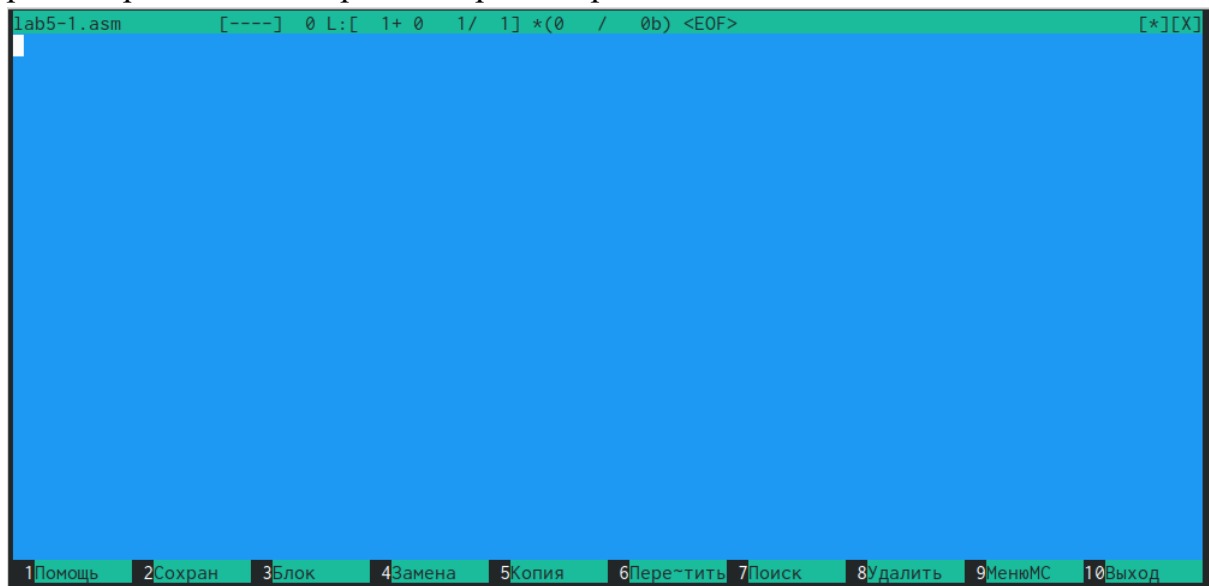
4. Пользуясь строкой ввода и командой touch создадим файл lab5-1.asm (рис. 3.4).

Левая панель	Файл	Команда	Настройки	Правая панель
< ~ /work/arch-pc/lab05 .[^]>				
.и	Имя	Размер	Дата правки	.и
/..		-ВВЕРХ-	ноя 6 18:29	/..
				/.cache
				/.config
				/.gnupg
				/.local
				/.mozilla
				/.pki
				/.ssh
				/public
				~public_
				/work
				/Видео
				/Докумен
				/Загрузк
				/Изображ
				/Музыка
-ВВЕРХ-				-ВВЕРХ-
2048G / 2048G (100%)				
Совет: Устали от этих сообщений? Отключите их в меню Настройки/В				
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 \$ touch lab5-1.asm				
1Помощь	2Меню	3Просмотр	4Правка	5Копия
				6Перенос

Левая панель		Файл	Команда	Настройки	Пр
<-		~/work/arch-pc/lab05	.[^]>		
Имя		Размер	Дата правки		
/..		-ВВЕРХ-	ноя	6	18:29
lab5-1.asm		0	ноя	6	18:31

**Рис.3.4.,Рис.3.5.**

5. С помощью функциональной клавиши F4 откроем файл lab5-1.asm для редактирования во встроенном редакторе.



**Рис.3.6.**

6. Введем текст программы из листинга 5.1.(Рис.3.7)Сохраним изменения и закроем файл(Рис.3.8).

```

lab5-1.asm      [-M--] 20 L:[ 1+34 35/ 35] *(2431/2431b) <EOF>
;
; 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.3.7.

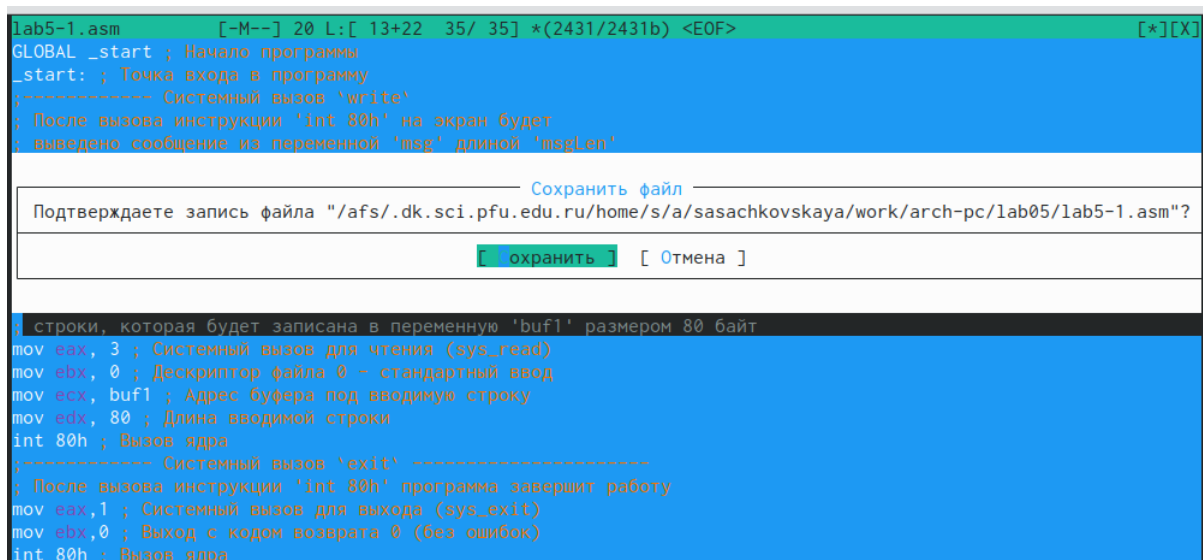


Рис.3.8.

7. С помощью функциональной клавиши F3 откроем файл lab5-1.asm для просмотра. Убедимся, что файл содержит текст программы.

```

/afs/.dk.sci.pfu.edu.ru/home/s/a/sasachkovskaya/work/arch-pc/lab05/lab5-1.asm
;-----
; 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.3.9.**

8. Оттранслируем текст программы lab5-1.asm в объектный файл. Выполним компоновку объектного файла и запустим получившийся исполняемый файл. Программа выводит строку 'Введите строку:' и ожидает ввода с клавиатуры. На запрос введу моё ФИО.

```

sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ nasm -f elf lab5-1.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-1 lab5-1.o
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ./lab5-1
Введите строку:
Сачковская София
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ █

```

**Рис.3.10.**

9. Скачаем файл in\_out.asm со страницы курса в ТУИС.

10. Подключаемый файл in\_out.asm должен лежать в том же каталоге, что и файл с программой, в которой он используется. В одной из панелей mc откроем каталог с файлом lab5-1.asm. В другой панели каталог со скачанным файлом in\_out.asm (для перемещения между панелями используем Tab). Скопируем файл in\_out.asm в каталог с файлом lab5-1.asm с помощью функциональной клавиши F5 (рис. 3.11).

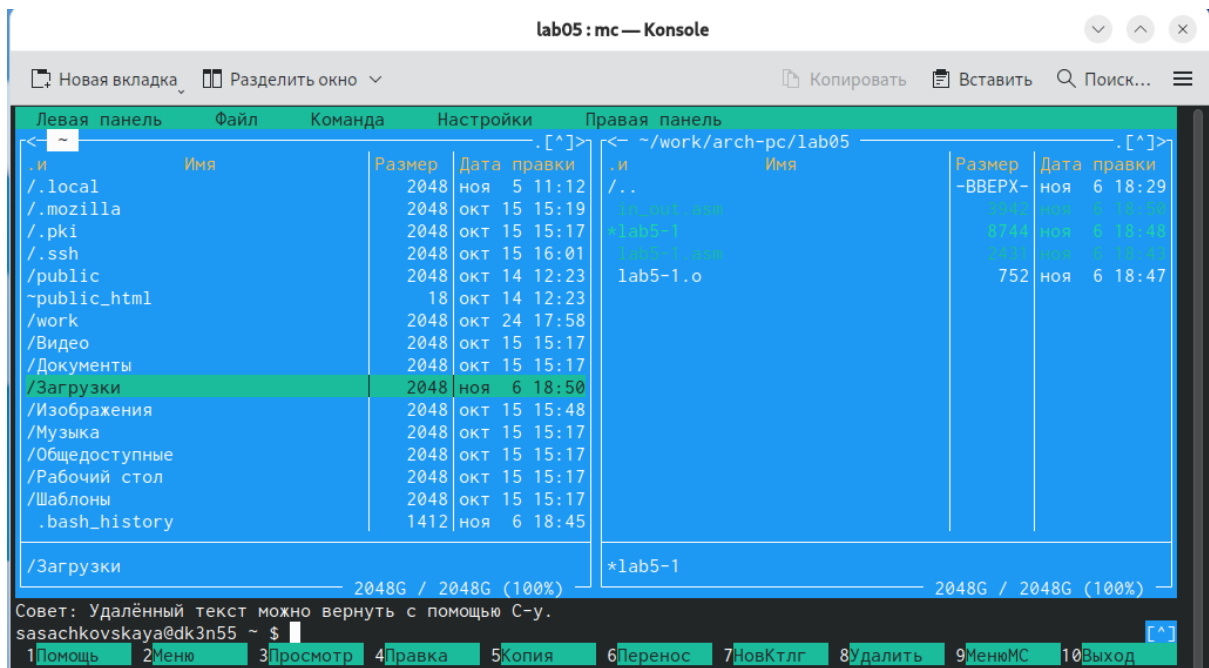


Рис.3.11.

11. С помощью функциональной клавиши F6 создадим копию файла lab5-1.asm с именем lab5-2.asm. Выделим файл lab5-1.asm, нажмём клавишу F6, введём имя файла lab5-2.asm и нажмём клавишу Enter (рис. 3.12.).

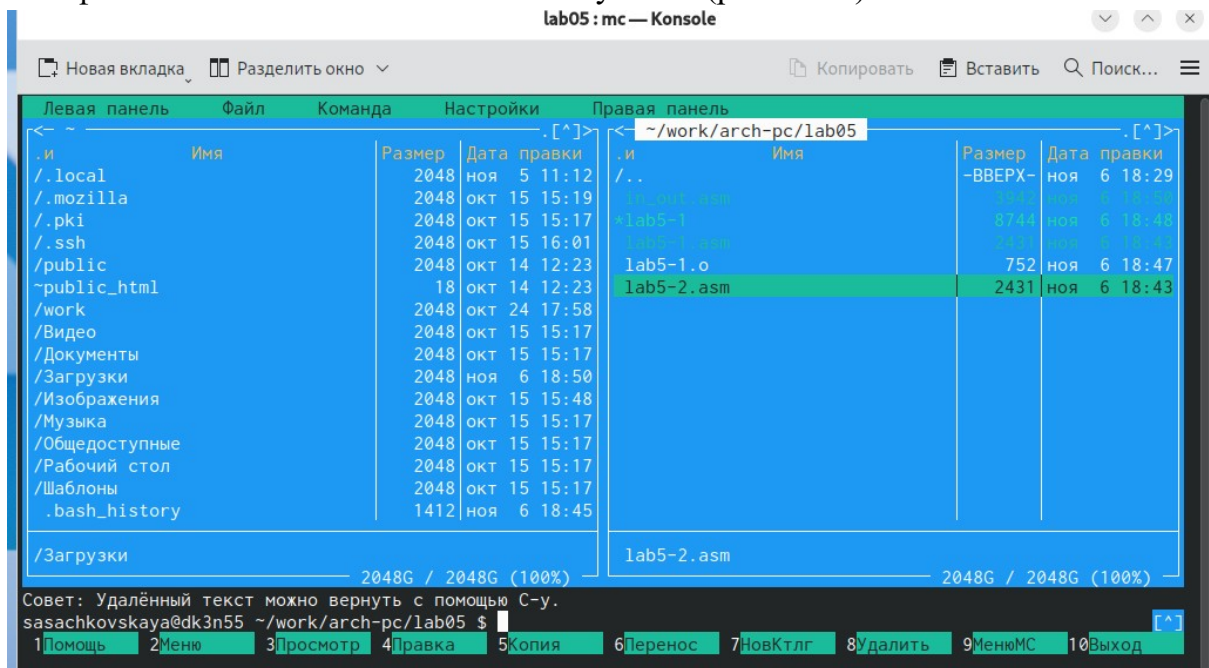


Рис.3.12.

12. Исправим текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in\_out.asm (используем подпрограммы sprintLF, sread и quit) в соответствии с листингом 5.2. Создадим исполняемый файл и проверим его работу.

```
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 sprintLF ; 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

sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-2 lab5-2.o
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ./lab5-2
Введите строку:
Сачковская София
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $
```

Рис 3.13.,3.14.

13. В файле lab5-2.asm заменим подпрограмму sprintLF на sprint. Создадим исполняемый файл и проверим его работу. Разница в том, что при выводе на экран sprintLF добавляет к сообщению символ перевода строки

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ nasm -f elf lab5-2.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-2 lab5-2.o
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ./lab5-2
Введите строку: Сачковская София
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $
```

Рис 3.15.

Вывод:Выполнили лабораторную работу. Изучили новые команды и горячие клавиши

## 4.Выполнение самостоятельной работы

1. Создайте копию файла lab5-1.asm. Внесите изменения в программу (без использования внешнего файла in\_out.asm), так чтобы она работала по следующему алгоритму:

- вывести приглашение типа “Введите строку:”;
- ввести строку с клавиатуры;
- вывести введенную строку на экран.

Левая панель				Правая панель			
Файл		Команда		Настройки		Файл	
< ~/work/arch-pc/lab05						< ~/work/arch-pc/lab05	
.и		Имя		Размер		Имя	
/..		-ВВЕРХ-		Дата правки		/..	
in_out.asm		3942		ноя 6 18:29		in_out.asm	
*lab5-1		8744		ноя 6 18:40		*lab5-1	
lab5-1.asm		2431		ноя 6 18:43		lab5-1.asm	
lab5-1.o		752		ноя 6 18:47		lab5-1.o	
*lab5-2		9892		ноя 6 19:25		*lab5-2	
lab5-2.asm		1222		ноя 6 19:24		lab5-2.asm	
lab5-2.o		1312		ноя 6 19:25		lab5-2.o	
lab5-3.asm		2431		ноя 6 18:43		lab5-3.asm	

Рис 4.1

2. Получите исполняемый файл и проверьте его работу. На приглашение ввести строку введите свою фамилию.

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ nasm -f elf lab5-3.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-3 lab5-3.o
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ./lab5-3
Введите строку:
Сачковская
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $
```

Рис 4.2

3. Создайте копию файла lab5-2.asm. Исправьте текст программы с использованием подпрограмм из внешнего файла in\_out.asm, так чтобы она работала по следующему алгоритму:

- вывести приглашение типа “Введите строку:”;
- ввести строку с клавиатуры;
- вывести введенную строку на экран.

Левая панель	Файл	Команда	Настройки	П
<	~/work/arch-pc/lab05			.[^]>
.и	Имя	Размер	Дата	правки
/..		-ВВЕРХ-	ноя 6 18:29	
in.out.asm		3942	ноя 6 18:50	
*lab5-1		8744	ноя 6 18:48	
lab5-1.asm		2431	ноя 6 18:43	
lab5-1.o		752	ноя 6 18:47	
*lab5-2		9092	ноя 6 19:25	
lab5-2.asm		1222	ноя 6 19:24	
lab5-2.o		1312	ноя 6 19:25	
*lab5-3		8744	ноя 6 19:30	
lab5-3.asm		2431	ноя 6 18:43	
lab5-3.o		752	ноя 6 19:30	
lab5-4.asm		1222	ноя 6 19:24	

Рис 4.3

4. Создайте исполняемый файл и проверьте его работу

```

sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ nasm -f elf lab5-4.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ld -m elf_i386 -o lab5-4 lab5-4.o
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $ ./lab5-4
Введите строку: Сачковская
sasachkovskaya@dk3n55 ~/work/arch-pc/lab05 $

```

Рис 4.4

Вывод: Я выполнила самостоятельные задания лабораторной работы. Получила навыки использования Midnight Commander

## **5.Выводы**

Я приобрела практические навыки работы в Midnight Commander. Освоила инструкции языка ассемблера mov и int.