

# **Отчёт по лабораторной работе №5**

***Дисциплина: Архитектура компьютера***

Байрамова Гюльсабах Акифовна НММбд-01-24

# 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>
3.1	Основы работы с Midnight Commander. . . . .	7
3.2	Структура программы на языке ассемблера NASM .. . . .	9
3.3	Элементы программирования. . . . .	12
3.3.1	Описание инструкции mov. . . . .	12
3.3.2	Описание инструкции int. . . . .	14
3.3.3	Системные вызовы для обеспечения диалога с пользователем	14
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>16</b>
4.1	Задания для самостоятельной работы . . . . .	22
<b>5</b>	<b>Выводы</b>	<b>25</b>
	<b>Список литературы</b>	<b>26</b>

# Список иллюстраций

3.1	Окно Midnight Commander . . . . .	7
4.1	Окно Midnight Commander . . . . .	16
4.2	Каталог ~/work/arch-pc/ в Midnight Commander. . . . .	17
4.3	Создание папки lab05 . . . . .	17
4.4	Создание файла lab5-1.asm . . . . .	18
4.5	Выбор редактора . . . . .	18
4.6	Редактор mcedit . . . . .	19
4.7	Текст программы . . . . .	19
4.8	Проверка . . . . .	20
4.9	Создание исполняемого файла . . . . .	20
4.10	Работа исполняемого файла . . . . .	20
4.11	Перемещение файла in_out.asm . . . . .	20
4.12	Создание копии файла lab5-1.asm... . . . .	21
4.13	Текст программы . . . . .	21
4.14	Создание и работа исполняемого файла . . . . .	22
4.15	Создание и работа исполняемого файла . . . . .	22
4.16	Создание копии файла lab5-1.asm... . . . .	23
4.17	Измененный текст программы . . . . .	23
4.18	Создание и работа исполняемого файла . . . . .	23
4.19	Измененный текст программы . . . . .	24
4.20	Создание и работа исполняемого файла . . . . .	24

## Список таблиц

3.1	Функциональные клавиши Midnight Commander . . . . .	8
3.2	Примеры . . . . .	10
3.3	Директивы для объявления неиницированных данных... . . .	11
3.4	Варианты использования mov с разными операндами . . . . .	13

# 1 Цель работы

Приобретение практических навыков работы в *Midnight Commander*. Освоение инструкций языка ассемблера `movi int` .

## 2 Задание

1) Выполнение лабораторной работы

1) Подключение внешнего файла in\_out.asm

2) Задания для самостоятельной работы

# 3 Теоретическое введение

## 3.1 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. 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** (рис. 3.1).

В Midnight Commander используются функциональные клавиши **F1—F10**, к которым привязаны часто выполняемые операции (табл. 3.1).

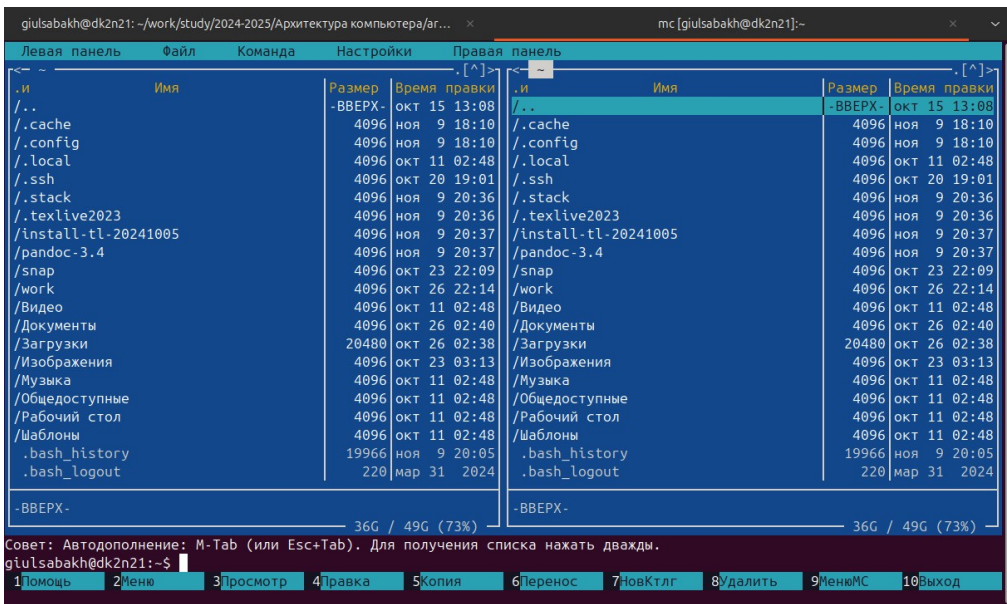


Рис. 3.1: Окно Midnight Commander

Таблица 3.1: Функциональные клавиши Midnight Commander

Кла- виша	Назначение
F1	вызов контекстно-зависимой подсказки
F2	вызов меню, созданного пользователем
F3	просмотр файла, на который указывает подсветка в активной панели
F4	вызов встроенного редактора для файла, на который указывает подсветка в активной панели
F5	копирование файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй панели
F6	перенос файла или группы отмеченных файлов из каталога, отображаемого в активной панели, в каталог, отображаемый на второй панели
F7	создание подкаталога в каталоге, отображаемом в активной панели
F8	удаление файла (подкаталога) или группы отмеченных файлов
F9	вызов основного меню программы
F10	выход из программы

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 расширения .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 ко-



мандный 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.

## 3.2 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`)).

Таким образом, общая структура программы имеет следующий вид:

**SECTION .data** ; Секция содержит переменные, для

... ; которых задано начальное значение

**SECTION .bss** ; Секция содержит переменные, для

... ; которых не задано начальное значение

**SECTION .text** ; Секция содержит код программы

**GLOBAL \_start**

`_start:` ; Точка входа в программу

... ; Текст программы

**mov eax,1** ; Системный вызов для выхода (`sys_exit`)

**mov ebx,0** ; Выход с кодом возврата 0 (без ошибок)

**int 80h** ; Вызов ядра

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 (учетверенное 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>]

Таблица 3.2: Примеры

Пример	Пояснение
a db 10011001b	определяем переменную размером 1 байт с начальным значением, заданным в двоичной системе счисления (на двоичную систему счисления указывает также буква b в конце числа) подсказки
b db '!'	определяем переменную в 1 байт, инициализируемую символом !
c db "Hello"	определяем строку из 5 байт
d dd -345d	определяем переменную размером 4 байта с начальным значением, заданным в десятичной системе счисления (на десятичную систему указывает буква d decimal ) в конце числа)

Пример	Пояснение
h dd 0f1ah	определяем переменную размером 4 байта с начальным значением, заданным в шестнадцатеричной системе счисления ( - hexadecimal )

Для объявления неиницированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти. Примеры их использования приведены в табл. 3.3.

Таблица 3.3: Директивы для объявления неиницированных данных

Директива	Назначение директивы	Аргумент	Назначение аргумента
resb	Резервирование заданного числа однобайтовых ячеек	string resb 20	По адресу с меткойstring будет расположен массив из 20 однобайтовых ячеек (хранение строки символов)
resw	Резервирование заданного числа двухбайтовых ячеек (слов)	count resw 256	По адресу с меткойcount будет расположен массив из 256 двухбайтовых слов

Директива	Назначение директивы	Аргумент	Назначение аргумента
resd	Резервирование заданного числа четырёхбайтовых ячеек (двойных слов)	x resd 1	По адресу с меткой будет расположено одно двойное слово (т.е. 4 байта для хранения большого числа)

## 3.3 Элементы программирования

### 3.3.1 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). В табл. 3.4 приведены варианты использования `mov` с разными операндами.

Таблица 3.4: Варианты использования mov с разными операндами

Тип операндов	Примеры	Пояснение
mov <reg>,<reg>	mov eax,ebx	пересылает значение регистра ebx в регистр eax
mov <reg>,<mem>	mov cx,[eax]	пересылает в регистр cx значение из памяти, указанной в eax
mov <mem>,<reg>	mov rez,ebx	пересылает в переменную rez значение из регистра ebx
mov <reg>,<const>	mov eax,403045h	пишет в регистр eax значение 403045h
mov <mem>,<const>	mov byte[rez],0	записывает в переменную rez значение 0

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

**mov eax, x**

**mov y, eax**

Также необходимо учитывать то, что размер операндов приемника и источника должны совпадать. Использование слудующих 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.

### 3.3.2 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 `ecx`, `edx`. Если системная функция должна вернуть значение, то она помещает его в регистр `eax`.

### 3.3.3 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 необходимо поместить

значение 4 в регистре `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.

## 4 Выполнение лабораторной работы

Откроем Midnight Commander с помощью клавиш  $\uparrow$  и  $\downarrow$  перейдем в каталог `'~/work/arch-pc/`, созданный при выполнении лабораторной работы №4 (рис. 4.1, 4.2)

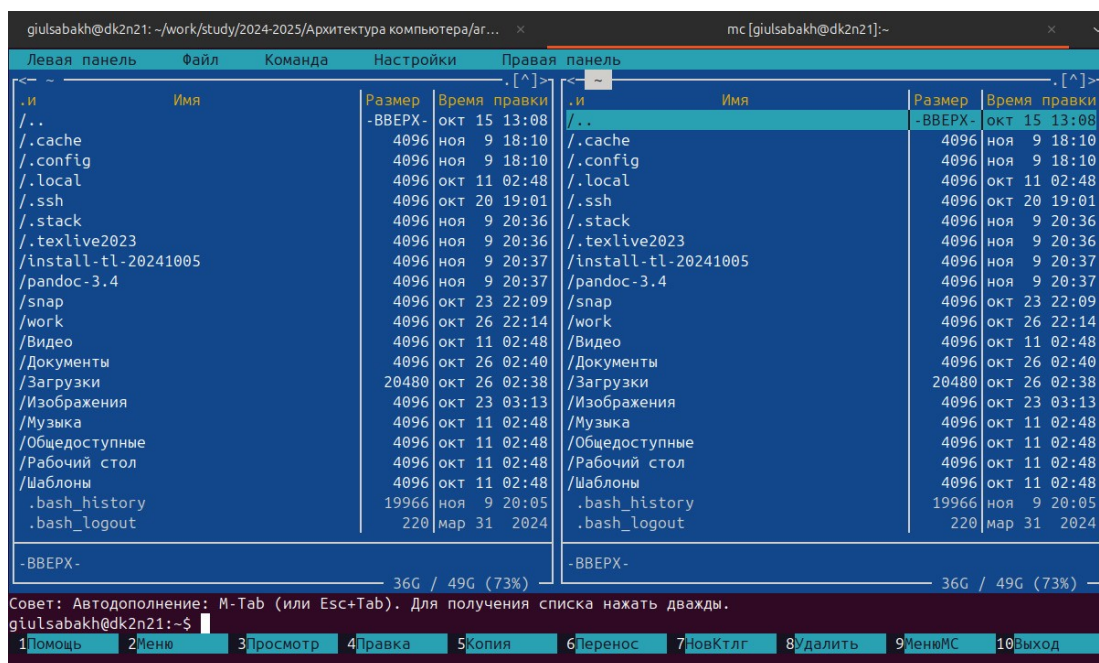


Рис. 4.1: Окно Midnight Commander



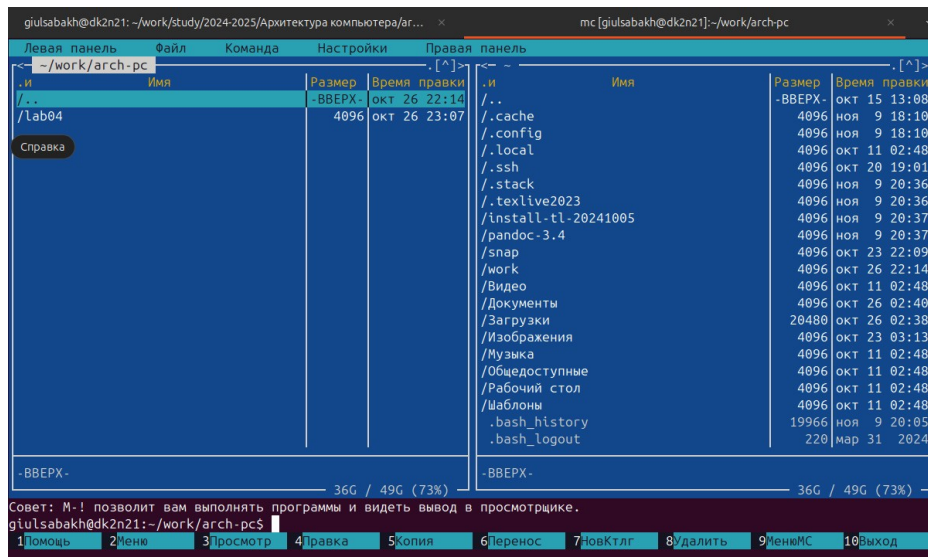


Рис. 4.2: Каталог `~/work/arch-pc/` в Midnight Commander

С помощью функциональной клавиши `F7` создадим папку `lab05` и, перейдя в неё, с помощью команды `touch` создадим файл `lab5-1.asm` (рис. 4.3, 4.4)

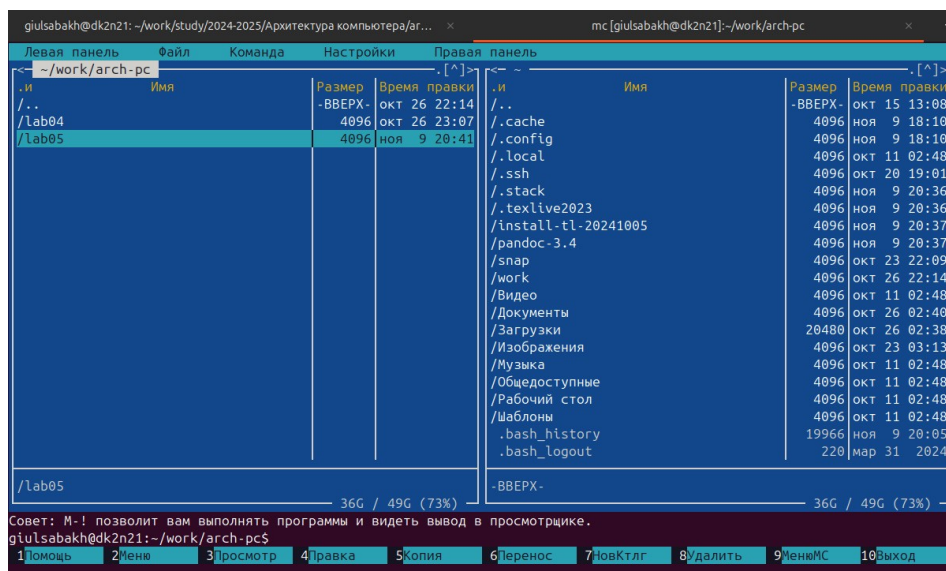


Рис. 4.3: Создание папки `lab05`

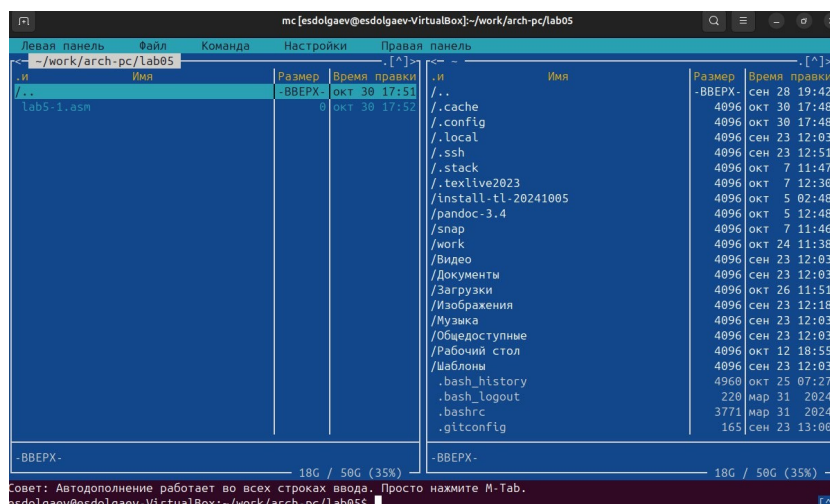


Рис. 4.4: Создание файла lab5-1.asm

С помощью функциональной клавиши **F4** откроем файл `lab5-1.asm` для редактирования во встроенном редакторе. Как правило в качестве встроенного редактора Midnight Commander используются редакторы `nano` или `mcedit` (рис. 4.5, 4.6).

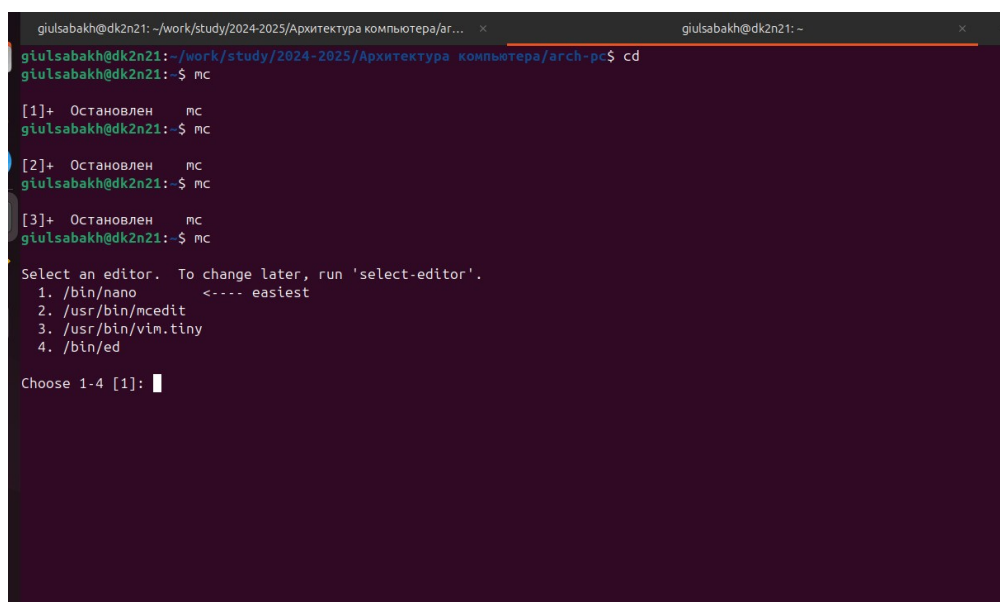


Рис. 4.5: Выбор редактора

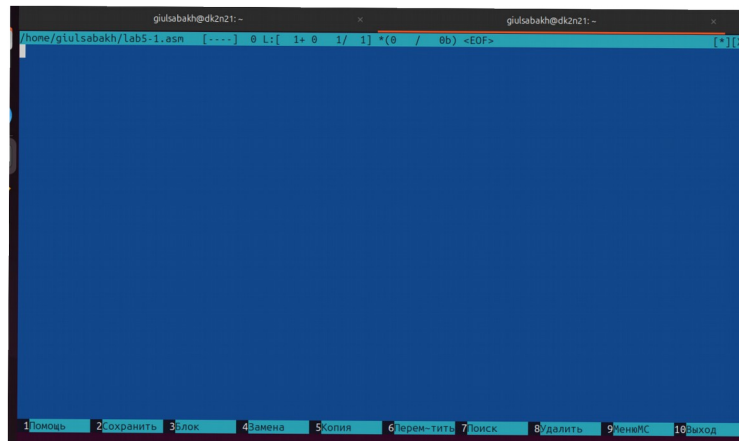


Рис. 4.6: Редактор mcedit

Введём текст программы, сохраним изменения и закроем файл(рис. 4.7).

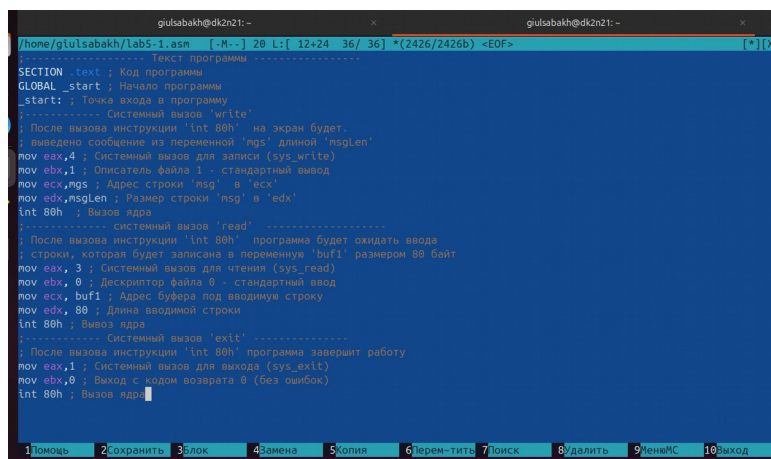


Рис. 4.7: Текст программы

С помощью функциональной клавиши F5 откроем файл lab5-1.asm для просмотра. Убедимся, что файл содержит текст программы(рис. 4.8).

```

/home/giulsabakh/work/arch-pc/lab05/lab5-1.asm 2067/2426 85%
; 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 ; Файловый дескриптор 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 ; Файловый дескриптор 0 - стандартный ввод
mov ecx,buf1 ; Адрес буфера под вводимую строку
mov edx,80 ; Длина вводимой строки
int 80h ; Вызов ядра
1Помощь 2Разверн 3Выход 4Hex 5Перейти 6 7Поиск 8Исходный 9Формат 10Выход

```

Рис. 4.8: Проверка

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 введём ФИО(рис. 4.9, 4.10).

*lab5-1	8744	ноя 10 00:1	
lab5-1.asm	2426	ноя 10 00:03	/.config
lab5-1.o	752	ноя 10 00:09	/.local /.ssh

Рис. 4.9: Создание исполняемого файла

```

giulsabakh@dk2n21:~$ cd ~/work/arch-pc/lab05
giulsabakh@dk2n21:~/work/arch-pc/lab05$ ./lab5-1
Введите строку:
Байрамова Гюльсабах

```

Рис. 4.10: Работа исполняемого файла

Далее, скачаем файл\_in\_out.asm со страницы курса в ТУИС и переместим его в папку, где находится файл с программой(рис. 4.11).

```

giulsabakh@dk2n21:~/work/arch-pc/lab05$ cd
giulsabakh@dk2n21:~$ cd ~/Загрузки
giulsabakh@dk2n21:~/Загрузки$ mv in_out.asm ~/work/arch-pc/lab05
mv: не удалось выполнить stat для 'in_out.asm': Нет такого файла или каталога
giulsabakh@dk2n21:~/Загрузки$ mkdir in_out.asm
giulsabakh@dk2n21:~/Загрузки$ mv in_out.asm ~/work/arch-pc/lab05
giulsabakh@dk2n21:~/Загрузки$ cd
giulsabakh@dk2n21:~$ cd ~/work/arch-pc/lab05
giulsabakh@dk2n21:~/work/arch-pc/lab05$ ls
in_out.asm lab5-1 lab5-1.asm lab5-1.o
giulsabakh@dk2n21:~/work/arch-pc/lab05$

```

Рис. 4.11: Перемещение файла in\_out.asm

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

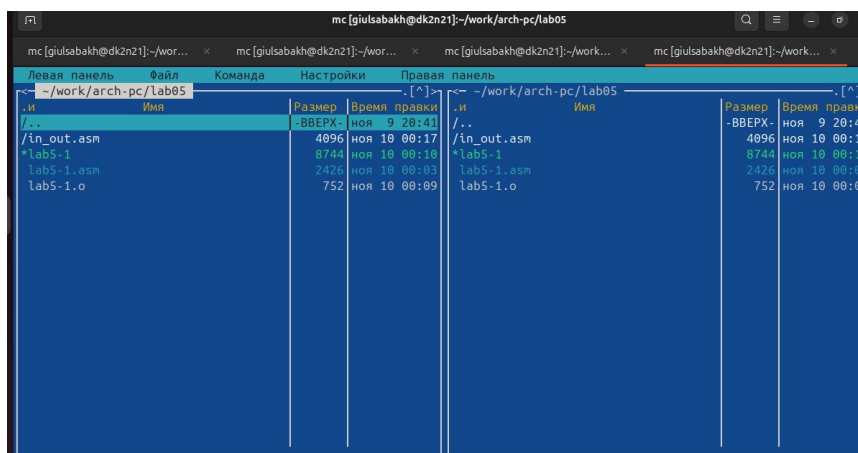


Рис. 4.12: Создание копии файла lab5-1.asm

Исправим текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in\_out.asm. Создадим исполняемый файл и проверим его работу(рис. 4.13).

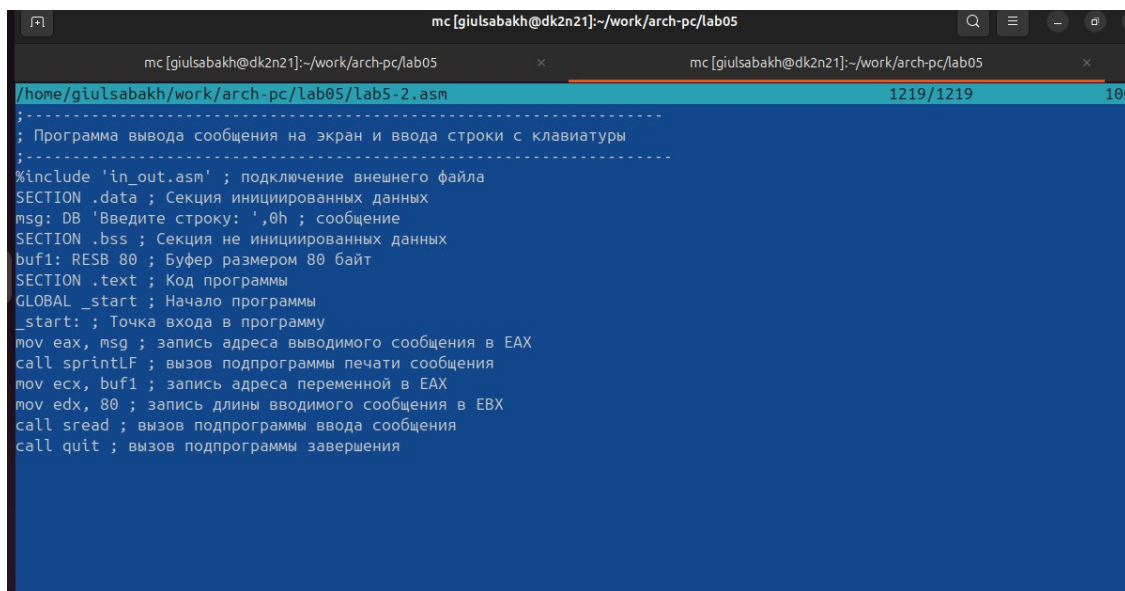
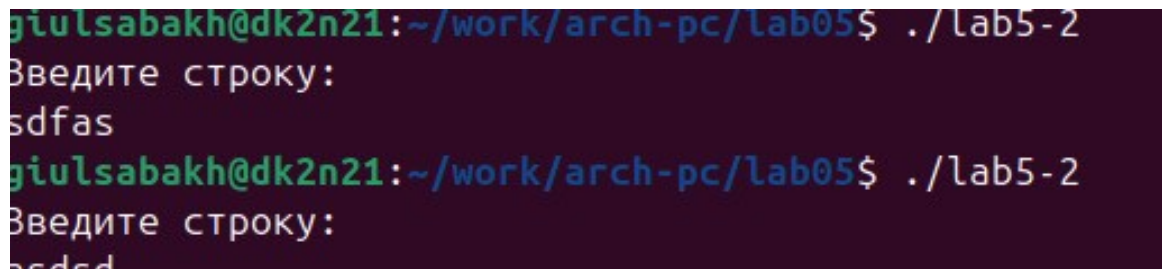


Рис. 4.13: Текст программы

В файле lab5-2.asm заменим подпрограмму sprintLF на sprint. Создадим исполняемый файл и проверим его работу(рис. 4.15).



```
giulsabakh@dk2n21:~/work/arch-pc/lab05$ ./lab5-2
Введите строку:
sdfas
giulsabakh@dk2n21:~/work/arch-pc/lab05$ ./lab5-2
Введите строку:
asdsd
```

Рис. 4.15: Создание и работа исполняемого файла

Теперь вводимая строка выводится ниже строки 'Введите строку:'.

## 4.1 Задания для самостоятельной работы

Создадим копию файла lab5-1.asm . Внесём изменения в программу (без использования внешнего файла in\_out.asm ), так чтобы она работала по следующему алгоритму(рис. 4.16, 4.17):

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

Получим исполняемый файл и проверим его работу. На приглашение ввести строку введите свою фамилию(рис. 4.18).



Имя	Размер	Время правки	Имя	Размер	Время правки
./in_out.asm	4096	ноя 9 20:41	./in_out.asm	4096	ноя 9 20:41
*lab5-1	8744	ноя 10 00:10	*lab5-1	8744	ноя 10 00:10
lab5-1.asm	2426	ноя 10 00:03	lab5-1.asm	2426	ноя 10 00:03
lab5-1.o	752	ноя 10 00:29	lab5-1.o	752	ноя 10 00:29
*lab5-11	8748	ноя 10 00:48	*lab5-11	8748	ноя 10 00:48
lab5-11.asm	2426	ноя 10 00:45	lab5-11.asm	2426	ноя 10 00:45
lab5-11.o	752	ноя 10 00:46	lab5-11.o	752	ноя 10 00:46
*lab5-2	8744	ноя 10 00:10	*lab5-2	8744	ноя 10 00:10
lab5-2.asm	1219	ноя 10 00:25	lab5-2.asm	1219	ноя 10 00:25
*lab5-21	8744	ноя 10 00:10	*lab5-21	8744	ноя 10 00:10
lab5-21.asm	1219	ноя 10 00:53	lab5-21.asm	1219	ноя 10 00:53

Рис. 4.16: Создание копии файла lab5-1.asm

```

GNU nano 7.2          Новый буфер *
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
Сохранить изменённый буфер?
Y Да
N Нет

```

Рис. 4.17: Измененный текст программы

```

giulsabakh@dk2n21: ~/work/arch-pc/lab05
mc [giulsabakh@dk2n21:~/work/arch-pc/lab05$ ./lab5-11
Введите строку:
Байрамова Гюльсабах
giulsabakh@dk2n21:~/work/arch-pc/lab05$ ./lab5-11
Введите строку:
dsff
giulsabakh@dk2n21:~/work/arch-pc/lab05$

```

Рис. 4.18: Создание и работа исполняемого файла

Создадим копию файла lab5-2.asm . Исправим текст программы с использование подпрограмм из внешнего файла in\_out.asm , так чтобы она работала по следующему алгоритму(рис. 4.19):

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

Создадим исполняемый файл и проверьте его работу(рис. 4.19).

```

GNU nano 7.2                                     Новый буфер *
;-----;
; 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 sprintfLF ; вызов подпрограммы печати сообщения
mov ecx, buf1 ; запись адреса переменной в EAX
mov edx, 80 ; запись длины вводимого сообщения в EBX

```

Рис. 4.19: Измененный текст программы

```

дгнґзәрәкүм@qкзу5т:-\молк\алсү-bc\ґәрөзз
зқґ
рвєџн1є с1вokλ:
дгнґзәрәкүм@qкзу5т:-\молк\алсү-bc\ґәрөзз ґ\ґәрз-5т
рззр: ґ\ґәрз-5т: һє1 1ґкoлo фәннүғ нун кә1ғулoғ
дгнґзәрәкүм@qкзу5т:-\молк\алсү-bc\ґәрөзз ґ\ґәрз-5т

ґқ: һєвoзвoжнo һәнн1н ґәрз-5т'o: һє1 1ґкoлo фәннүғ нун кә1ғулoғ
дгнґзәрәкүм@qкзу5т:-\молк\алсү-bc\ґәрөзз ґқ -w єґтґз8є -o ґәрз-5т ґәрз-5т'o

```

Рис. 4.20: Создание и работа исполняемого файла



## 5 Выводы

В ходе выполнения данной лабораторной работы я приобрёл практические навыки работы в *Midnight Commander* и освоил инструкции языка ассемблера и `int` .

## **Список литературы**