

Архитектура компьютера

Отчёт по лабораторной работе №5

Лю Сяо

1 Цель работы

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

2 Задание

1. Откройте Midnight Commander user@dk4n31:~\$ mc
2. Пользуясь клавишами ↑, ↓ и Enter перейдите в каталог ~/work/arch-pc созданный при выполнении лабораторной работы №4.
3. С помощью функциональной клавиши F7 создайте папку lab05 и перейдите в созданный каталог.
4. Пользуясь строкой ввода и командой touch создайте файл lab5-1.asm.
5. С помощью функциональной клавиши F4 откройте файл lab5-1.asm для редактирования во встроенном редакторе. Как правило в качестве встроенного редактора Midnight Commander используется редакторы nano или mcedit.
6. Введите текст программы из листинга 5.1 (можно без комментариев), сохраните изменения и закройте файл.
7. С помощью функциональной клавиши F3 откройте файл lab5-1.asm для просмотра. Убедитесь, что файл содержит текст программы.
8. Оттранслируйте текст программы lab5-1.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл. выводит строку 'Введите строку:' и ожидает ввода с клавиатуры. На запрос введите Ваши ФИО.
9. Скачайте 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.
11. С помощью функциональной клавиши F6 создайте копию файла lab5-1.asm с именем lab5-2.asm. Выделите файл lab5-1.asm, нажмите клавишу F6, введите имя файла lab5-2.asm и нажмите клавишу Enter
12. Исправьте текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in_out.asm (используйте подпрограммы sprintf, fread и quit) в соответствии с листингом 5.2. Создайте исполняемый файл и проверьте его работу.

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. Для активации оболочки Midnight Commander достаточно ввести в командной строке `mc` и нажать клавишу Enter (рис. 5.1). В Midnight Commander используются функциональные клавиши F1 — F10, к которым привязаны часто выполняемые операции (табл. 5.1).

Таблица 5.1. Функциональные клавиши Midnight Commander

Функциональные клавиши	Выполняемое действие
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 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` и на странице проекта [3].

Программа на языке ассемблера NASM, как правило, состоит из трёх секций: секция кода программы (SECTION .text), секция инициализированных (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 ; 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 инициализированных 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 (четверное 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: DB [] [,]

Таблица 5.2. Примеры

Пример	Пояснение
a db 10011001b	определяем переменную a размером 1 байт с начальным значением, заданным в двоичной системе счисления (на двоичную систему счисления указывает также буква b (binary) в конце числа)
b db '!'	определяем переменную b в 1 байт, инициализируемую символом !
c db "Hello"	определяем строку из 5 байт
d dd -345d	определяем переменную d размером 4 байта с начальным значением, заданным в десятичной системе счисления (на десятичную систему указывает буква d (decimal) в конце числа)
h dd 0f1ah	определяем переменную h размером 4 байта с начальным значением, заданным в шестнадцатеричной системе счисления (h — hexadecimal)

Для объявления неинициализированных 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. 5.3

Таблица 5.3. Резервирование заданного числа

Директива	Пояснение
resb 20	Резервирование заданного числа однобайтовых ячеек (слов)
resw 256	Резервирование заданного числа двухбайтовых ячеек (слов)
resd 1	Резервирование заданного числа четырехбайтовых ячеек (двойных слов)

По адресу с меткой string будет расположен массив из 20 однобайтовых ячеек (хранение строки символов)

По адресу с меткой count будет расположен массив из 256 двухбайтовых слов

По адресу с меткой x будет расположено одно двойное слово (т.е. 4 байта для хранения большого числа)

5.2.3. Элементы программирования

5.2.3.1. 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. 5.4 приведены варианты использования mov с разными операндами

Тип операндов	Пример	Пояснение
mov , mov eax,ebx		пересылает значение регистра ebx в регистр eax
mov , mov cx,[eax]		пересылает в регистр cx значение из памяти, указанной в eax
mov , mov rez,ebx		пересылает в переменную rez значение из регистра ebx
mov , mov eax,403045h		пишет в регистр eax значение 403045h
mov , 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`: `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.

5.2.3.2. 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 какойлибо 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`.

5.2.3.3. 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 `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.

4 Выполнение лабораторной работы

1) Открываю Midnight Commander с помощью команды `mc`

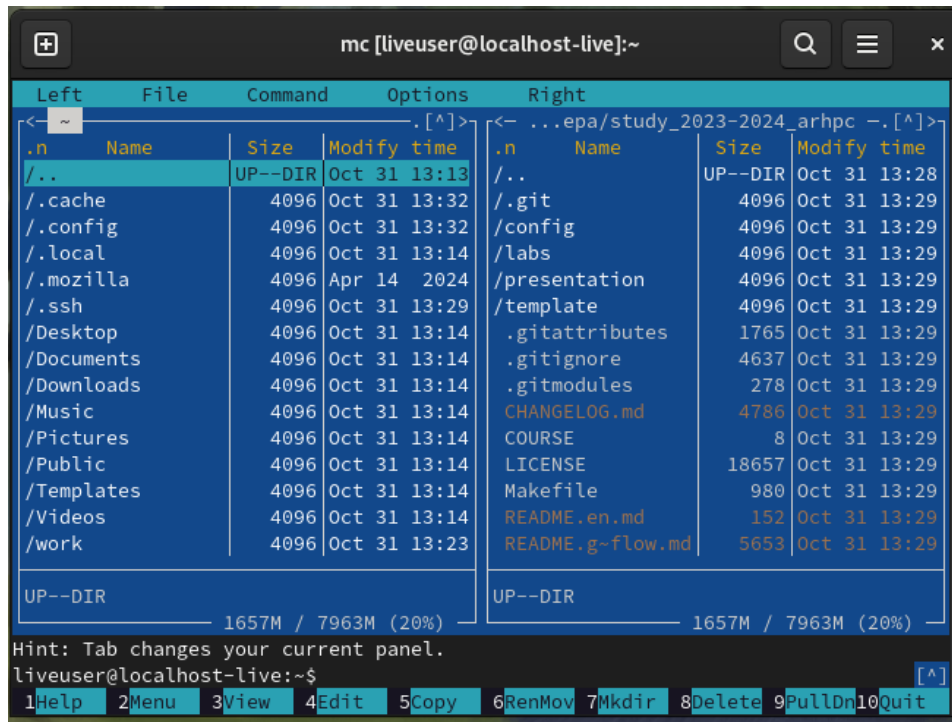


Figure 1: 1. Midnight Commander

2) с помощью клавиш \uparrow , \downarrow и Enter перехожу в каталог ~/work/arch-rc созданный при выполнении лабораторной работы №4 (рисунки 2 и 3)

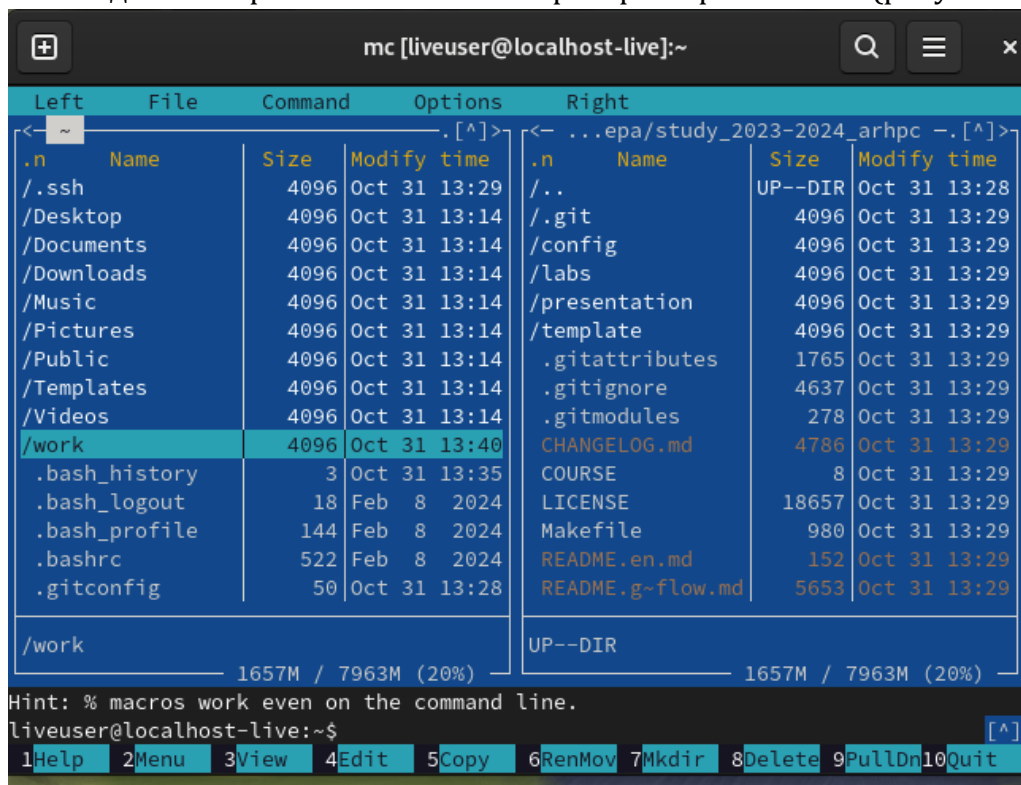


Figure 2: 2. Переход в work

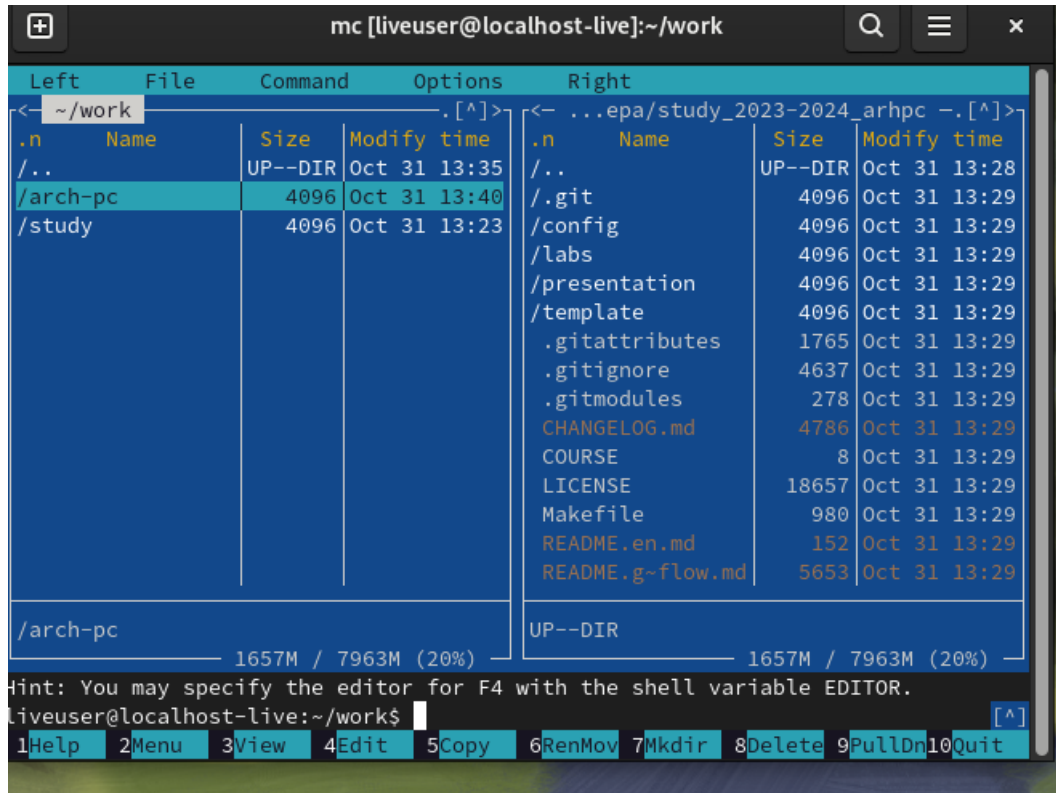


Figure 3: 3. Переход в arch-pc

3) С помощью F7 создаю папку lab05

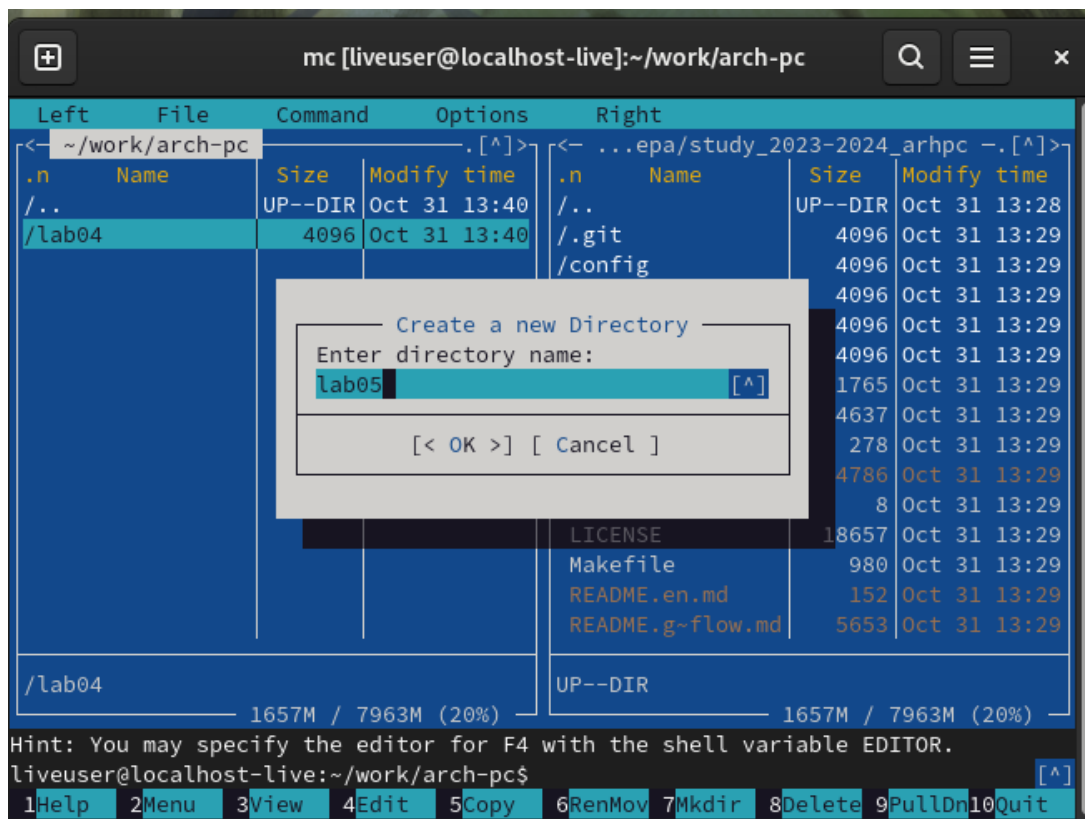


Figure 4: 4. Создание папки lab05

4) Перехожу в созданный каталог и с помощью touch создаю lab5-1.asm

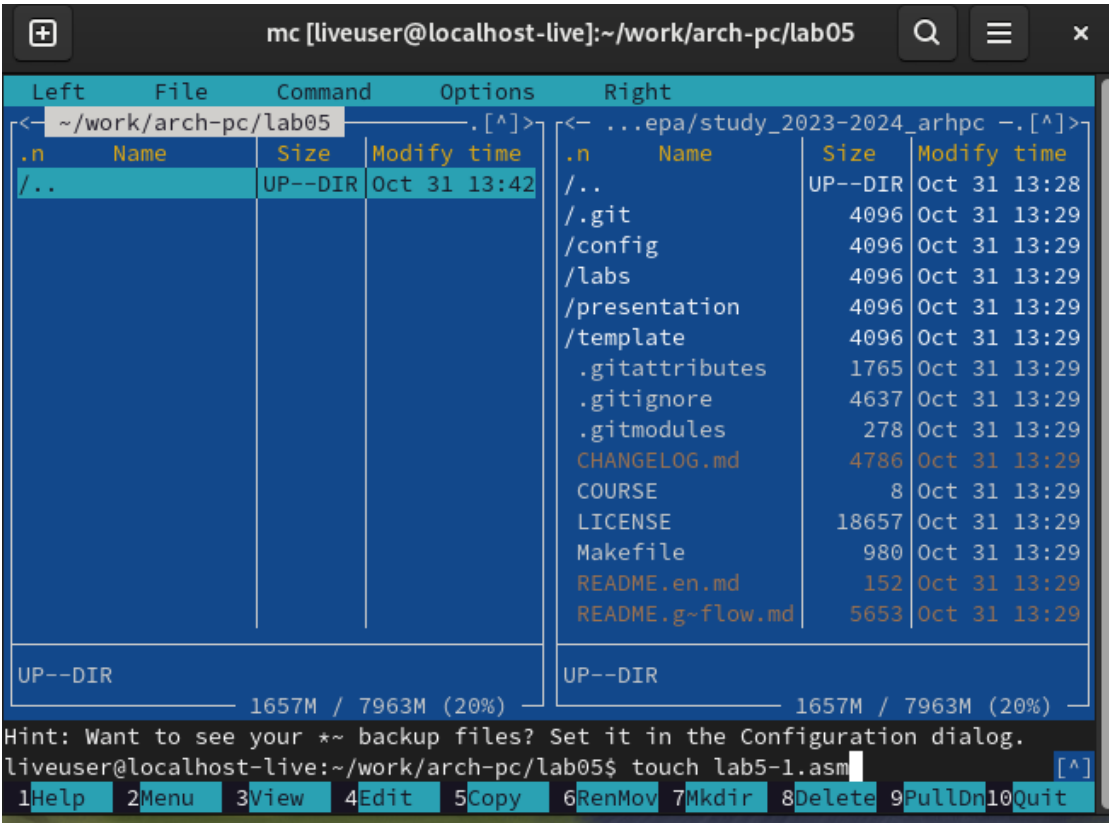


Figure 5: 5. lab05
Созданный файл

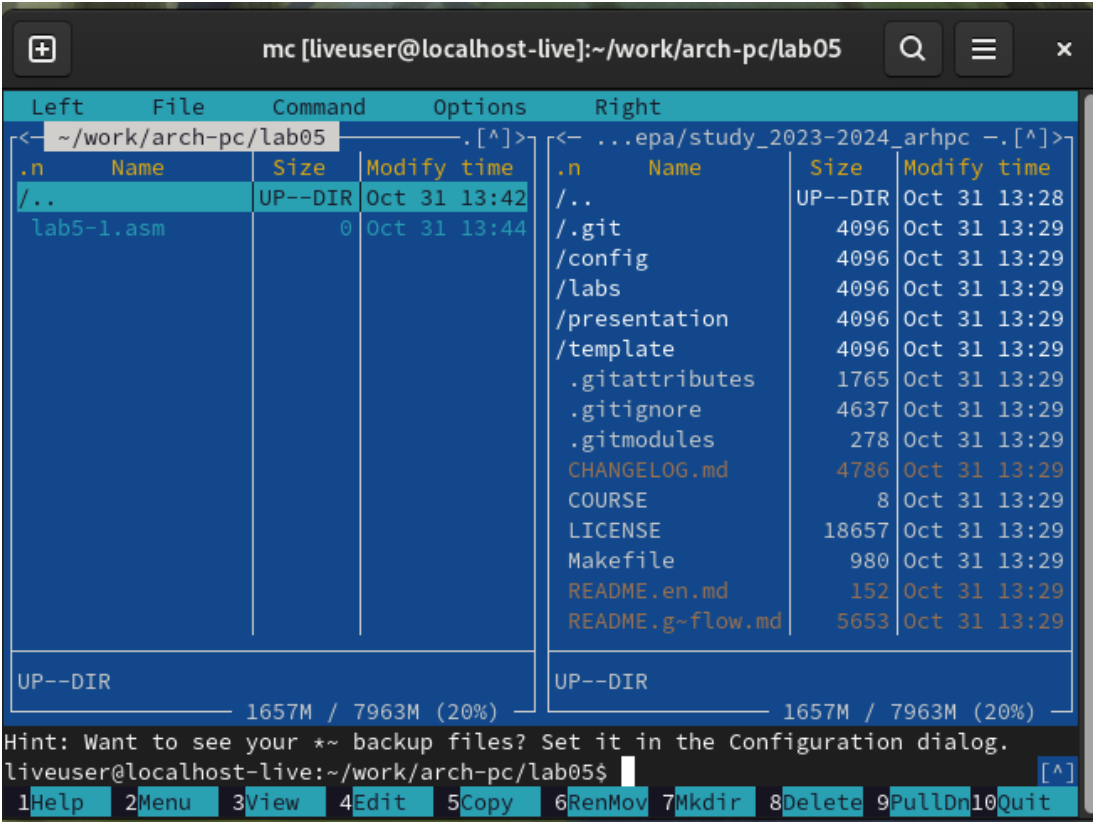
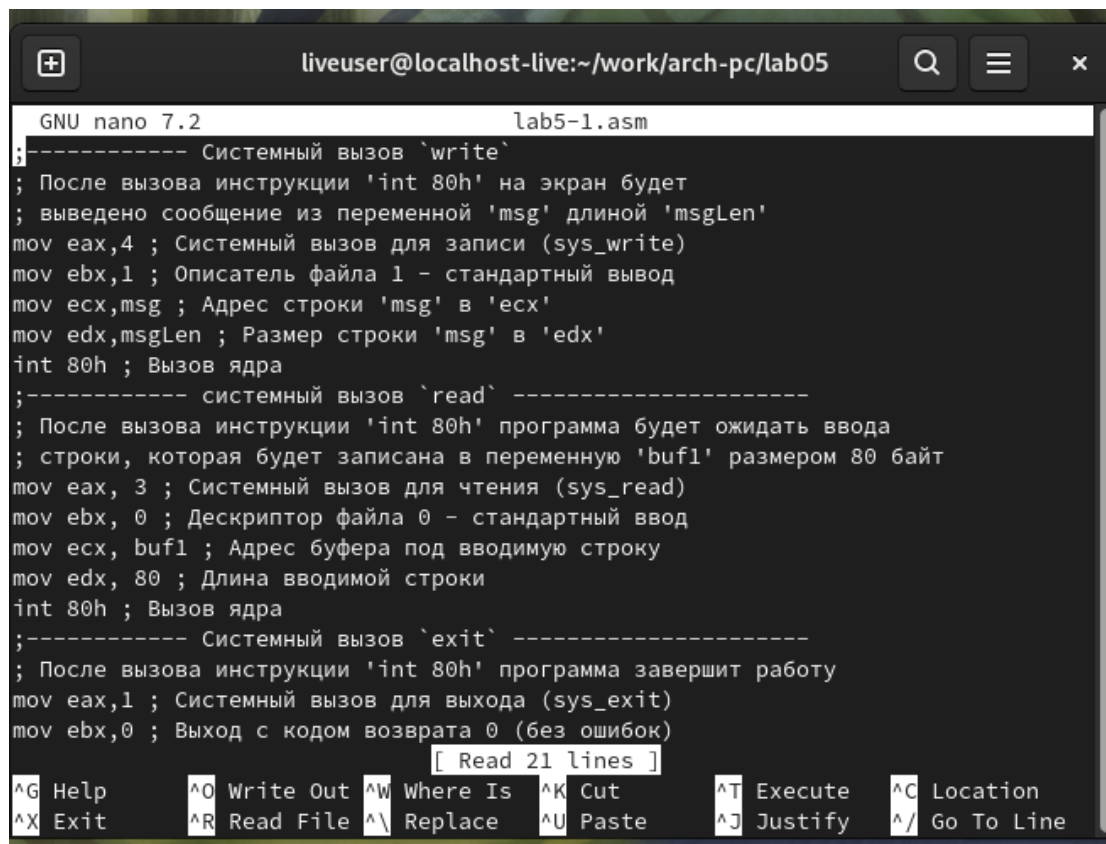


Figure 6: 6. Созданный файл

5) С помощью F4 открываю файл lab5-1.asm во встроенном редакторе, в моём случае - nano, и копирую туда код из задания лабораторной работы, сохраняю изменения



```
GNU nano 7.2 lab5-1.asm
;----- 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)
[ Read 21 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Figure 7: 7. Открытие файла в nano

6) С помощью F3 открываю файл и смотрю на сохраненные изменения


```
mc [liveuser@localhost-live]:~/work/arch-pc/lab05
/home/liveuser/work/arch-pc/lab05/lab5-1.asm 1448/2433 59%
;-----
; 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
1Help 2UnWrap 3Quit 4Hex 5Goto 6 7Search 8Raw 9Format10Quit
```

Figure 8: 7.1 Просмотр файла

7) Оттранслирую в объектный файл и выполняю компоновку файла (8-9)

```
Hint: Tab changes your current panel.
liveuser@localhost-live:~/work/arch-pc/lab05$ nasm -f elf lab5-1.asm
1Help 2Menu 3View 4Edit 5Copy 6RenMov 7Mkdir 8Delete 9PullDn10Quit
```

Figure 9: 8. Оттранслирую файл

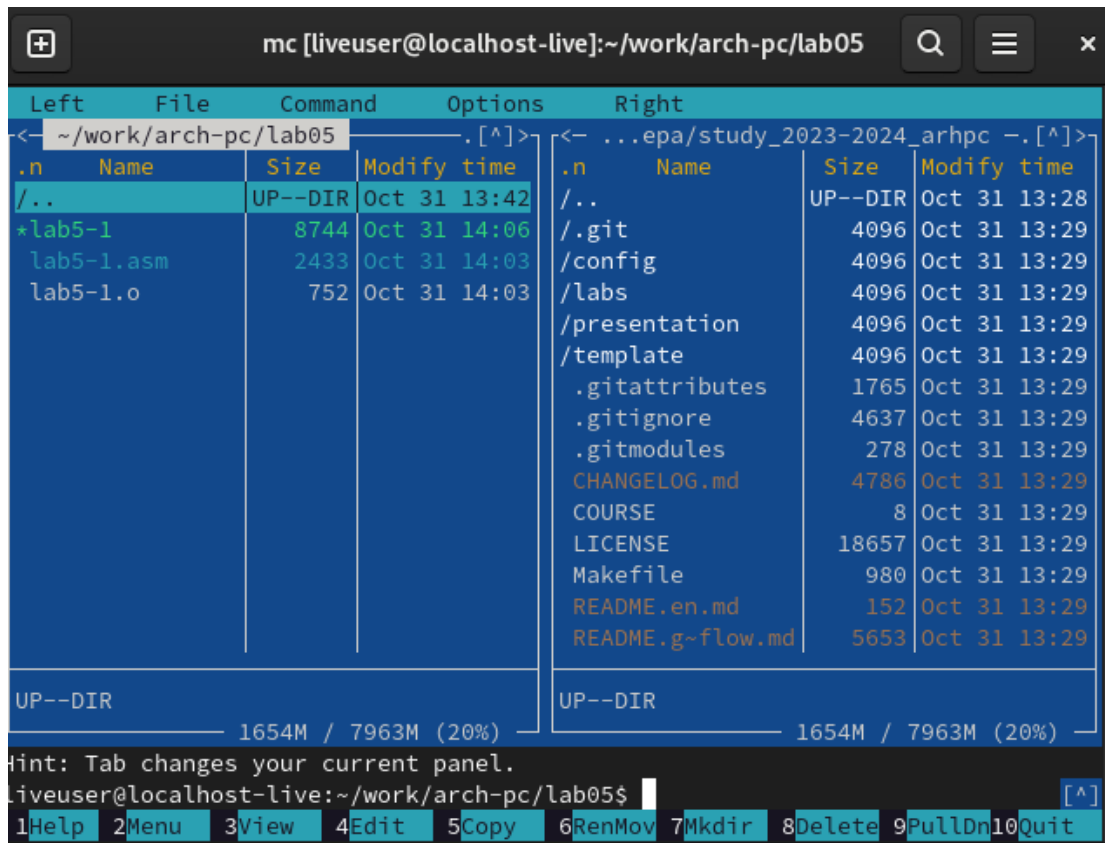


Figure 10: 9. Компоную файл

8) Работа файла

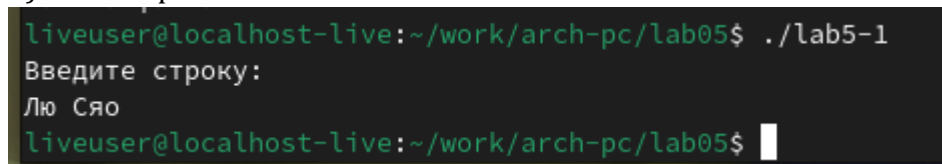


Figure 11: 10. Работа файла

9) Скачиваю in_out.asm и открываю во втором окне

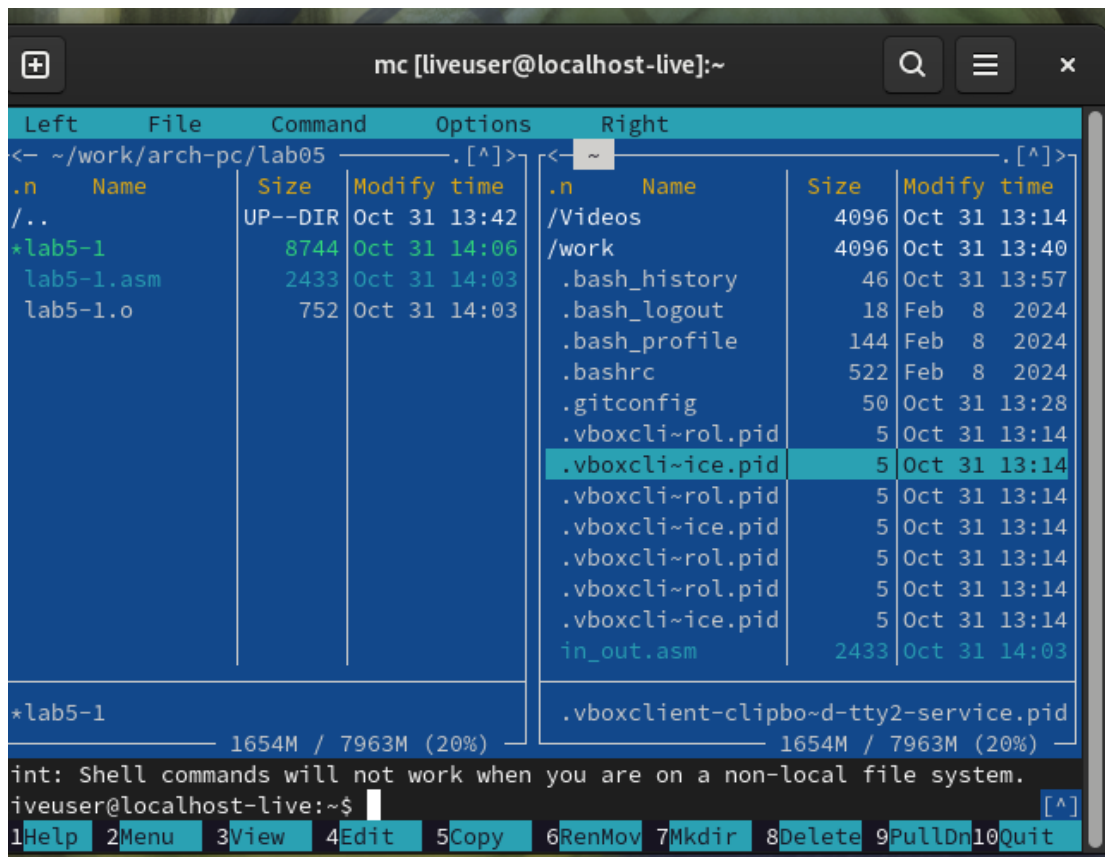


Figure 12: 11. два окна

10) С помощью F5 копирую файл в нужную папку

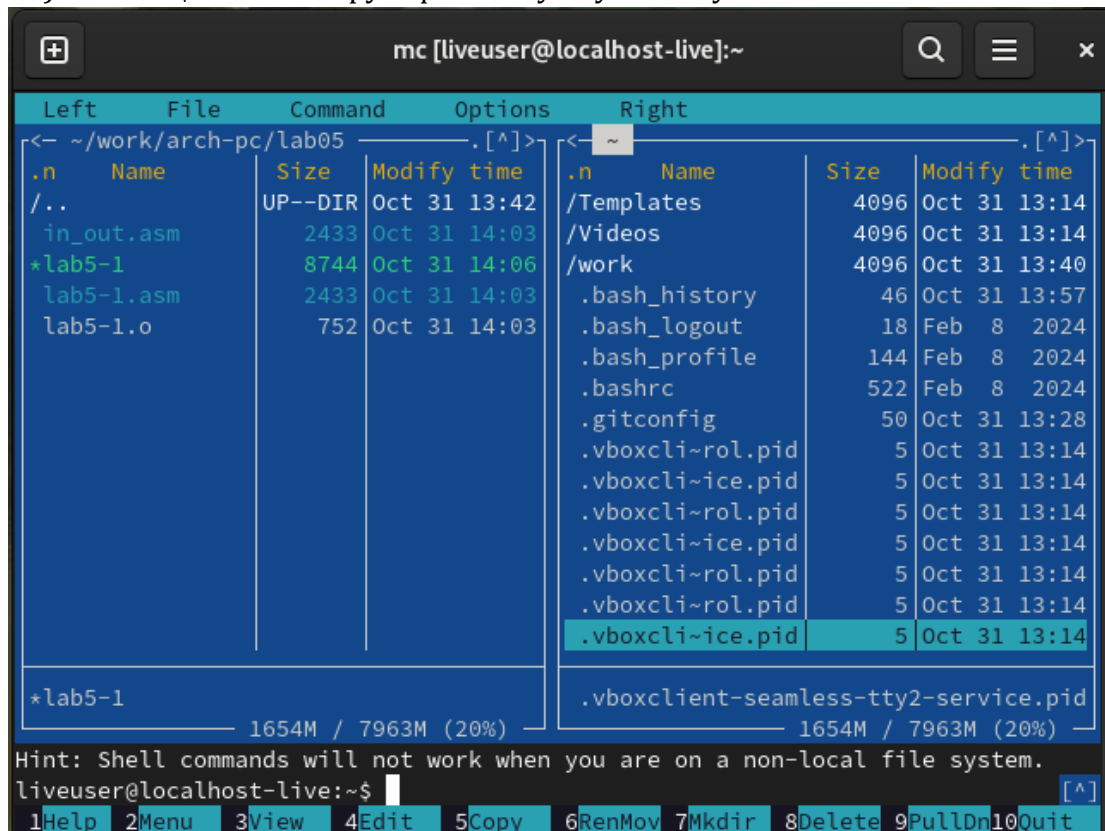


Figure 13: 12. Копирование файла

11) С помощью F6 создаю копию файла lab5-1.asm с именем lab5-2.asm. (13-14)

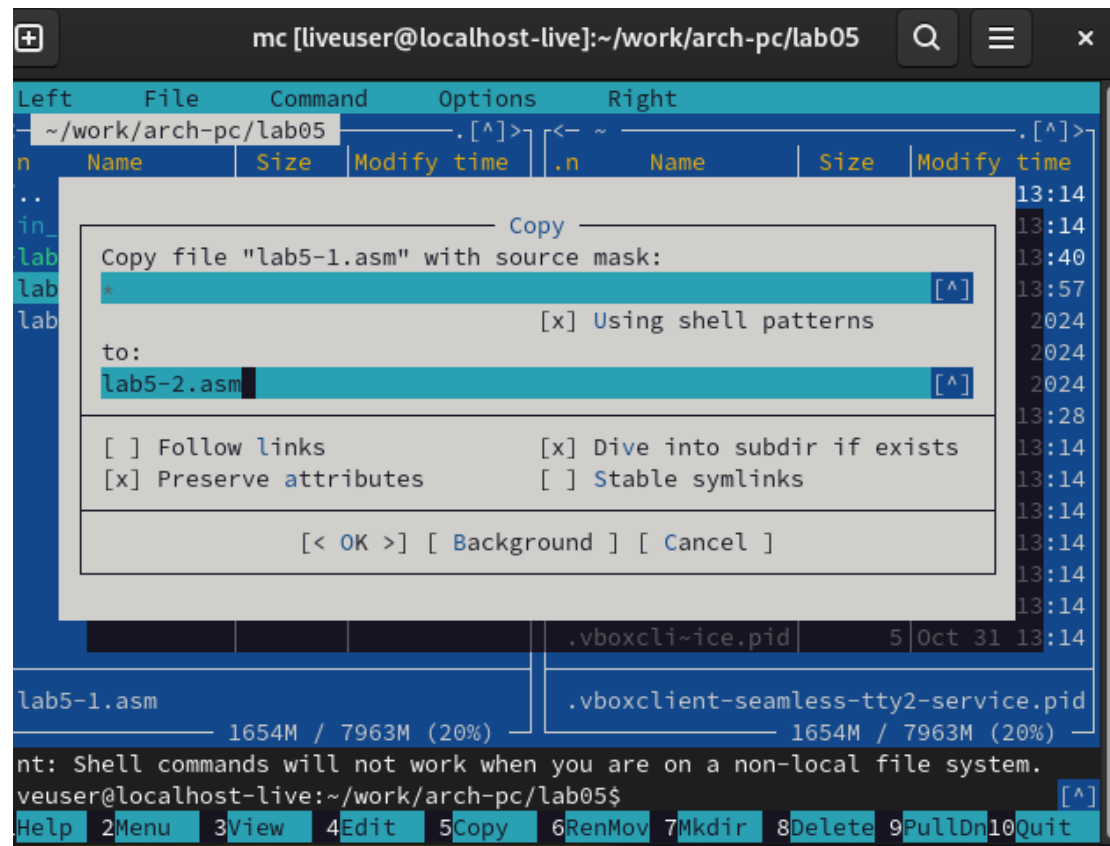


Figure 14: 13. Копирование файла

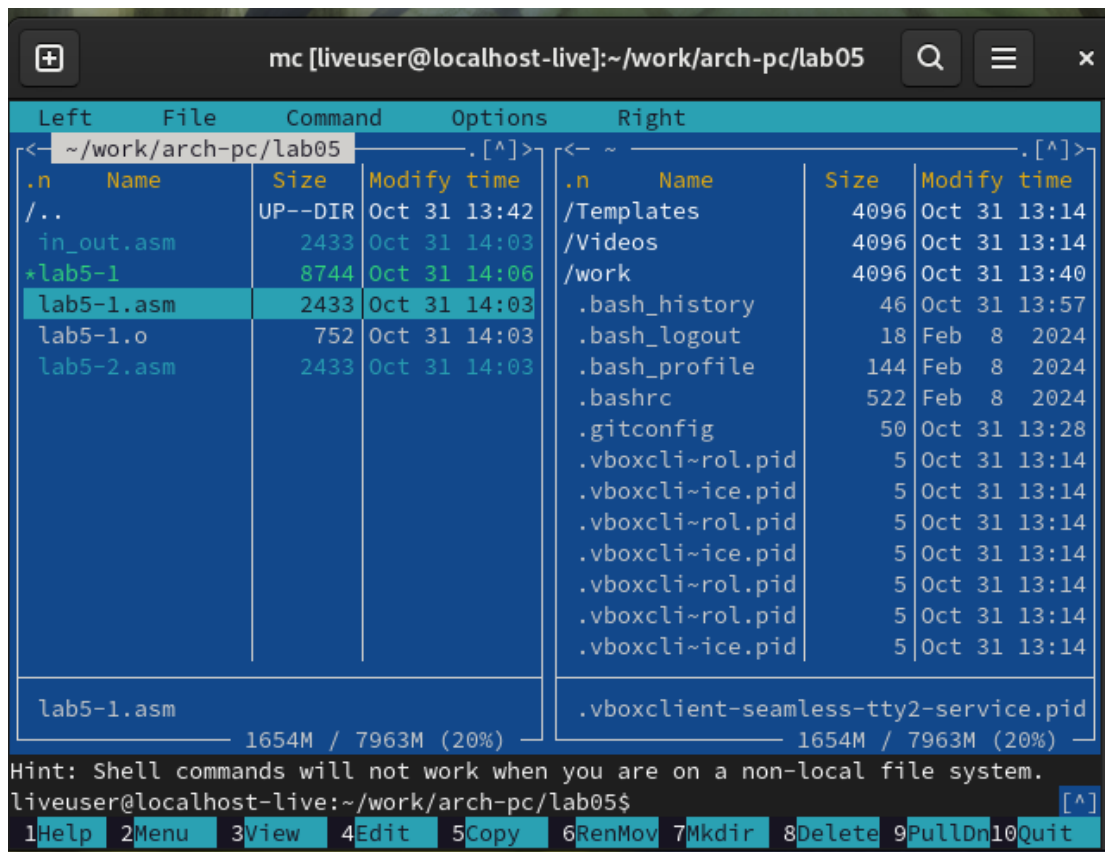


Figure 15: 14. Скопированный файл
12) Открываю lab5-2.asm для редактирования в mcedit

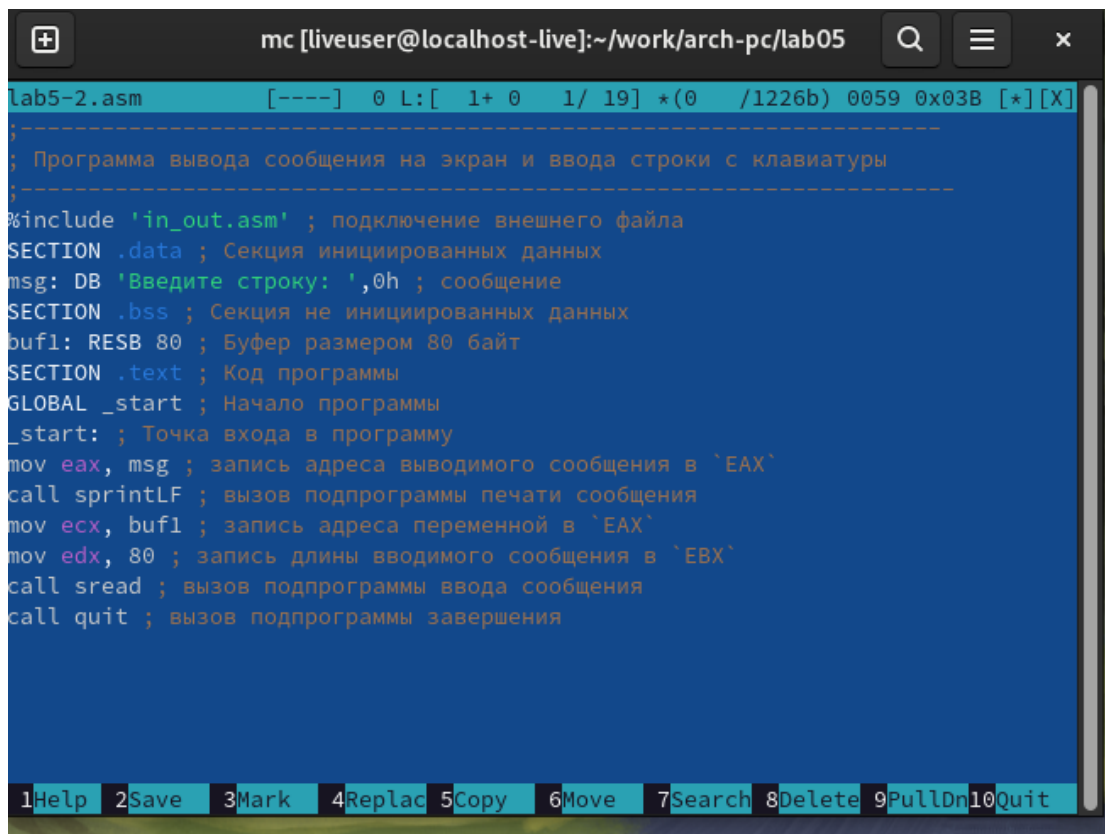


Figure 16: 15. Редактирование файла
13) Оттранслирую в объектный файл и выполняю компоновку файла (16-17)

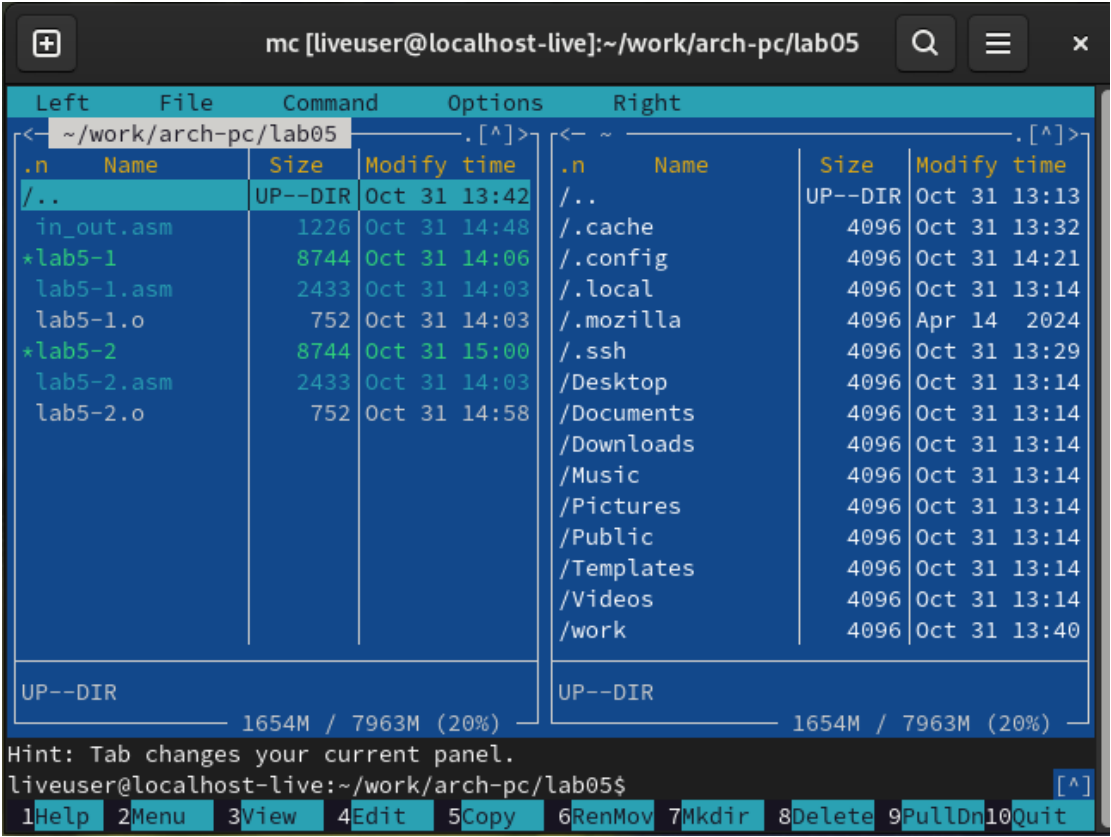


Figure 17: 16. Созданный объектный файл

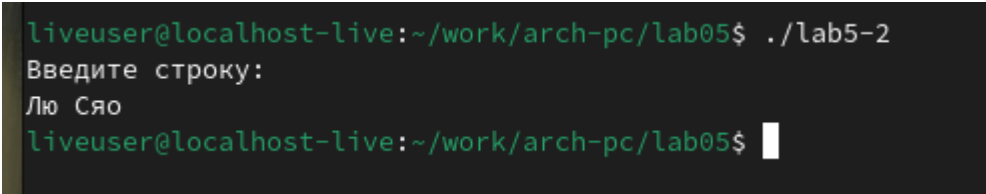


Figure 18: 17. Создание и работы исполняемого файла

Выполнение заданий для самостоятельной работы

1) Копирую lab5-1.asm как lab5-1.1.asm

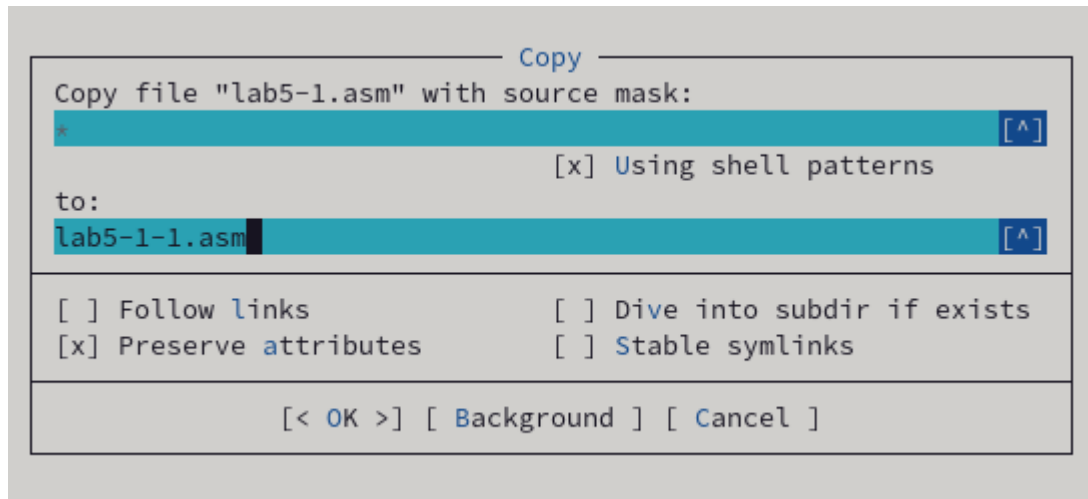


Figure 21: 20. Копирование файла
2) Открываю его с помощью mcedit

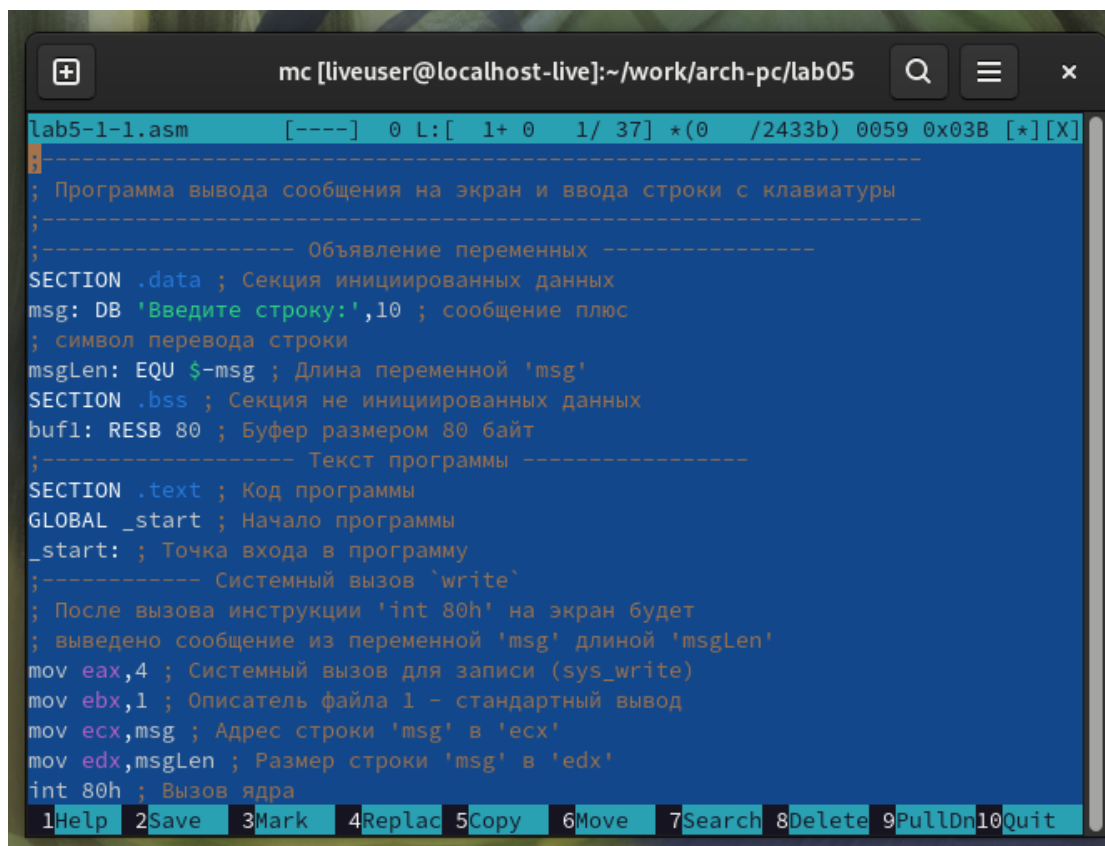


Figure 22: 21. Открытый файл
3) Вношу изменения так, чтобы программа возвращала введённое значение код программы:
SECTION .data ; Секция иницированных данных msg: DB 'Введите строку:',10 ; сообщение плюс ; символ перевода строки msgLen: EQU \$-msg ; Длина переменной 'msg' SECTION .bss ; Секция не иницированных данных buf1: RESB 80 ; Буфер размером 80 байт 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 ; mov ebx, 1 ; 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

```
lab5-1-1.asm [----] 0 L: [ 22+21 43/ 43] *(2/53/2/530) <EOF> [*][X]
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
mov eax, 4 ; Системный вызов для чтения (sys_read)
mov ebx, 1 ; Дескриптор файла 0 - стандартный ввод
mov ecx, buf1 ; Адрес буфера под вводимую строку
mov edx, buf1 ; 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

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit
```

Figure 23: 22. Код исправленной программы

4) Создаю исполняемый файл и запускаю программу

```
liveuser@localhost-live:~/work/arch-pc/lab05$ ./lab5-1-1
Введите строку:
Лю Сяо
Лю Сяо
liveuser@localhost-live:~/work/arch-pc/lab05$
```

Figure 24: 23. Работа файла

5) Создаю копию файла lab5-2.asm. Исправляю текст программы с использованием подпрограмм из внешнего файла in_out.asm код программы: %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 `mov eax, 4 ; mov ebx, 1 ; mov ecx, buf1 ; int 80h; call quit ;` вызов подпрограммы завершения

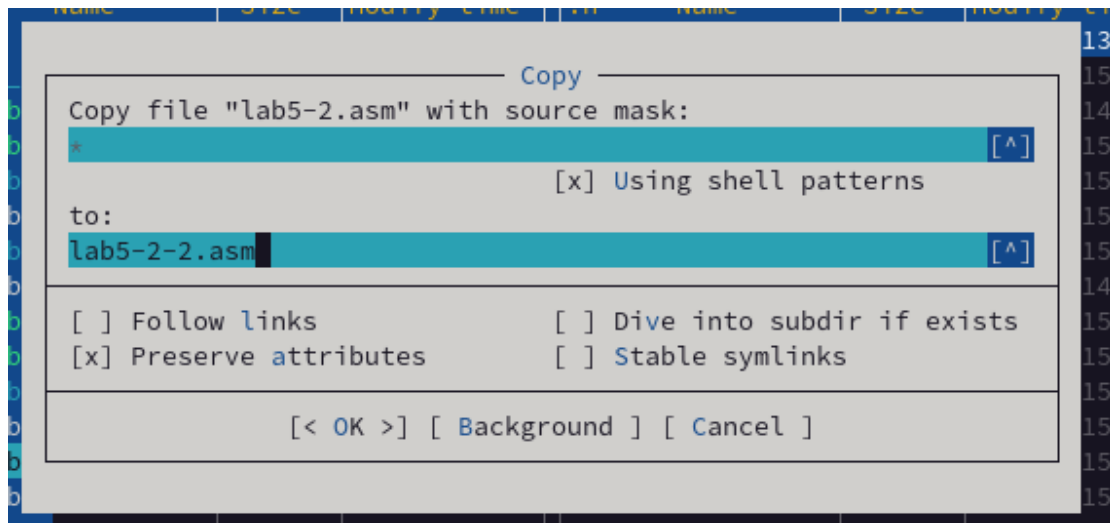


Figure 25: 24. Копирование файла

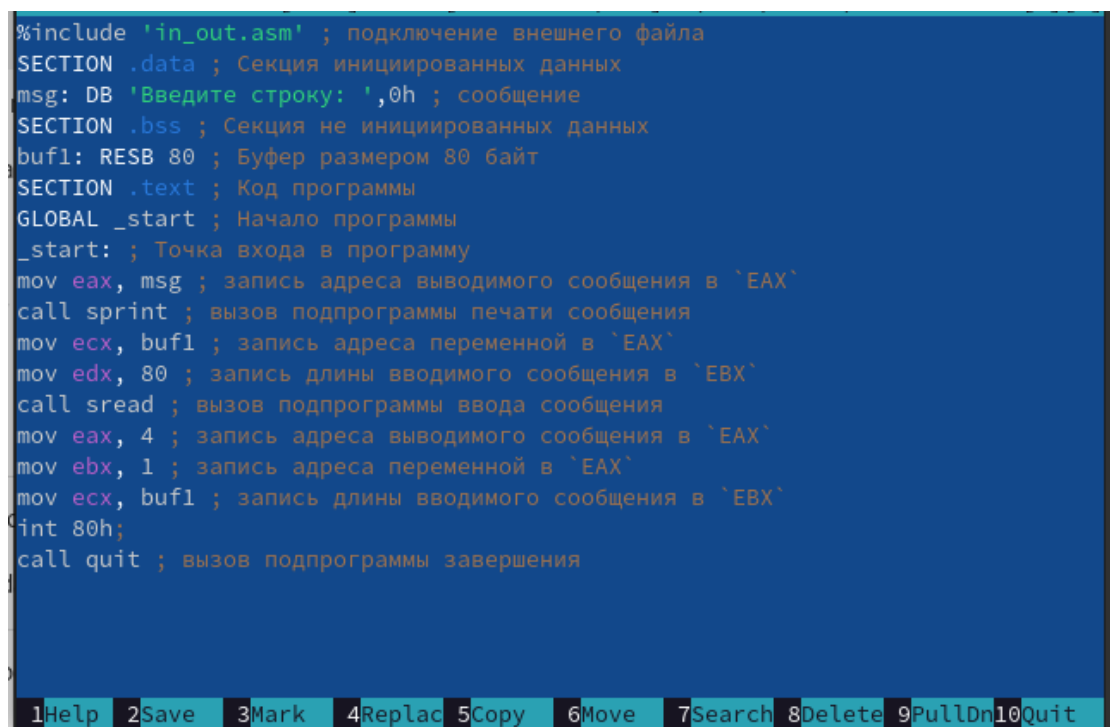


Figure 26: 25. Отредактированный файл

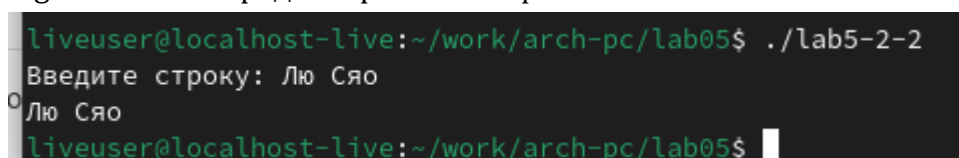


Figure 27: 26. Работа файла

Список литературы