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:Земцов Роман

Группа:НКАбд 02-25

МОСКВА

2025 г.

Оглавление

Цель работы

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

Теоретическое введение

2.1. Midnight Commander (mc)

Midnight Commander (mc) — это файловый менеджер для консоли, который упрощает работу с файлами. Он запускается командой mc и представляет собой две панели, между которыми можно копировать и перемещать файлы.

Основные клавиши, которые я использовал в работе:

F4 (Edit)	Открывает файл в текстовом редакторе.
F5 (Copy)	Копирует выделенный файл в каталог на другой панели.
F6 (Move/Rename)	Перемещает или переименовывает файл.
F7 (Mkdir)	Создает новый каталог.
F10 (Quit)	Выходит из mc.
Tab	Переключение между левой и правой панелями.
Ctrl + O	Скрывает/показывает панели mc, позволяя работать с командной строкой.

Таблица 1. Основные клавиши

2.2. Структура программы на NASM

Программа на ассемблере NASM, которую мы писали, состоит из нескольких секций.

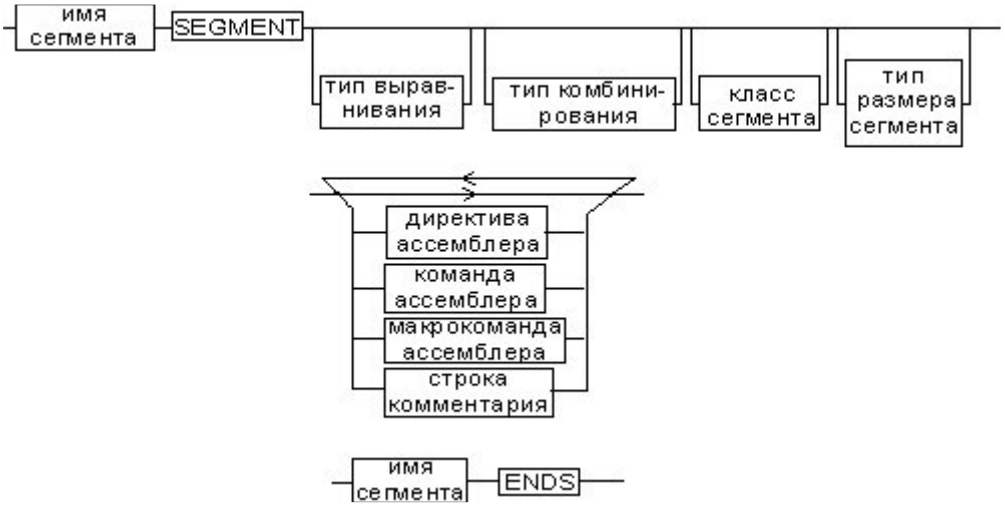


Рисунок 1. Синтаксис описания сегмента

Выполнение программы начинается с метки _start, которую нужно объявить глобальной с помощью GLOBAL _start.

2.3. Основные инструкции и системные вызовы

В работе я освоил две главные инструкции:

1. **mov (move):** Инструкция для копирования данных. Она имеет формат mov <приёмник>, <источник>. Например, mov eax, 4 копирует число 4 в регистр eax.
2. **int 80h (interrupt):** Инструкция, которая вызывает ядро Linux для выполнения системной функции (системного вызова).

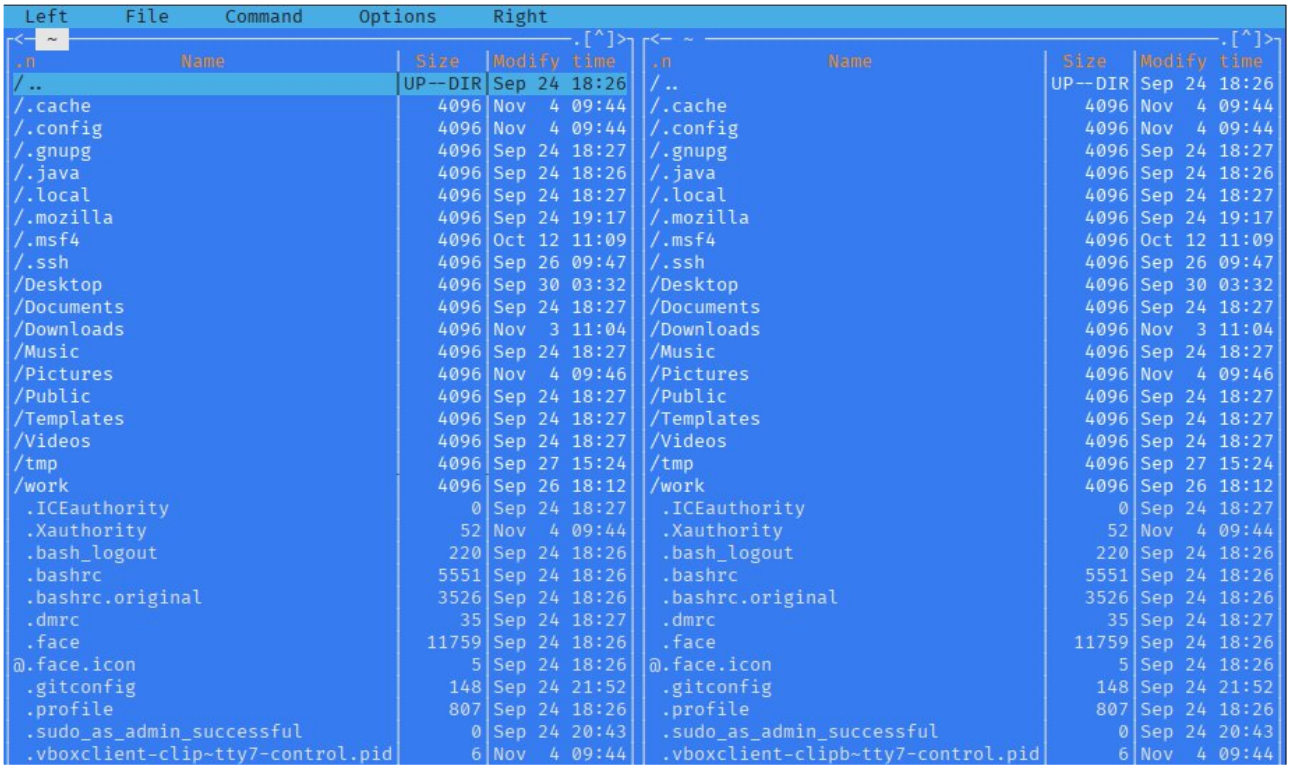
Чтобы ядро поняло, какую функцию мы хотим, мы перед int 80h должны поместить номер функции в регистр eax. Параметры для этой функции кладутся в регистры ebx, ecx, edx

Описание результатов выполнения лабораторной работы

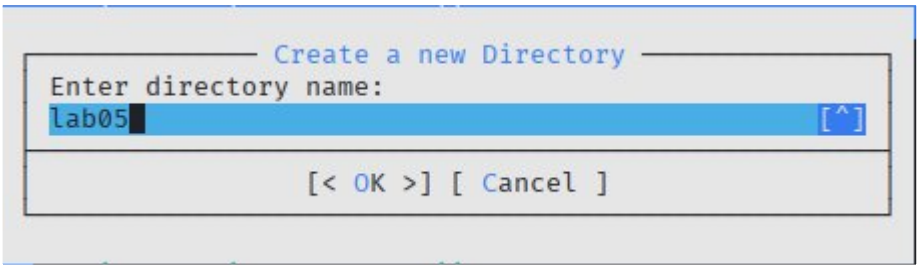
3.1. Задание 1. Работа в Midnight Commander и создание первой программы

Необходимо было запустить Midnight Commander, создать в нем рабочий каталог lab05, создать файл lab5-1.asm, отредактировать его, вписав программу из Листинга 5.1, а затем скомпилировать и запустить ее.

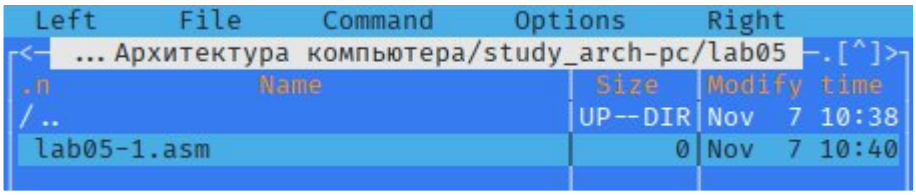
Я открыл терминал и ввел команду mc, чтобы запустить Midnight Commander.



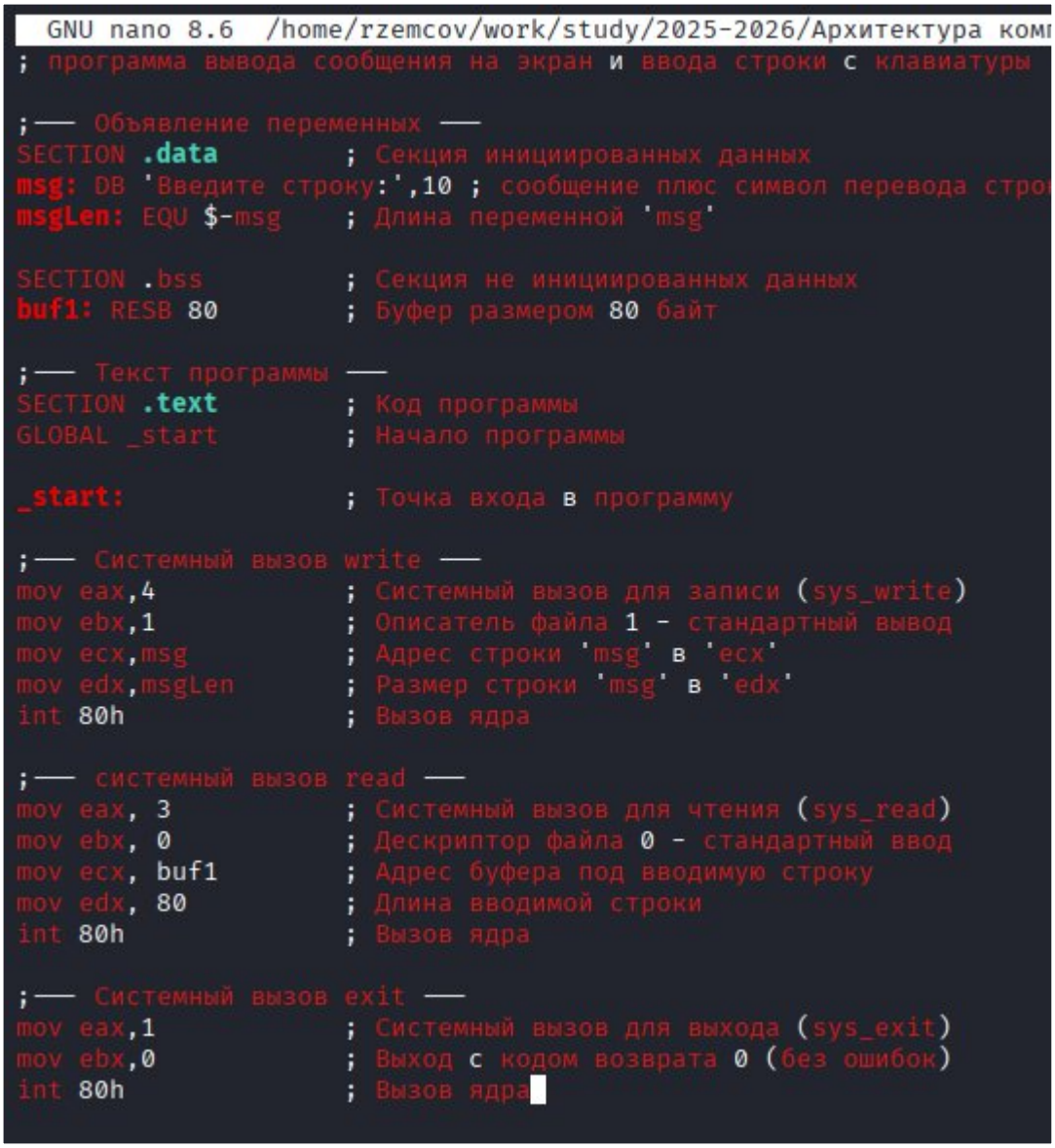
Используя навигацию, я перешел в каталог ~/work/study_arch-pc, который создал в прошлой лабораторной и нажал клавишу F7 для создания нового каталога и ввел имя lab05.



Зашел в созданный каталог **lab05** и в командной строке под панелями ввел **touch lab5-1.asm** и нажал **Enter**. Файл появился в панели



Выделив файл lab5-1.asm, я нажал F4, чтобы открыть его во встроенном редакторе (у меня это был mcedit).



Я сохранил файл (клавиша F2) и вышел из редактора (F10), нажав F3 на файле lab5-1.asm, я убедился, что его содержимое сохранилось.

```
/home/rzemcov/work/study/2025-2026/Архит~мпьютера/study_arch-pc/lab05/lab05-1.asm
; программа вывода сообщения на экран и ввода строки с клавиатуры

;── Объявление переменных ──
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

;── Текст программы ──
SECTION .text           ; Код программы
GLOBAL _start           ; Начало программы

_start:                 ; Точка входа в программу

;── Системный вызов write ──
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 ──
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 ──
mov eax,1               ; Системный вызов для выхода (sys_exit)
mov ebx,0               ; Выход с кодом возврата 0 (без ошибок)
int 80h                 ; Вызов ядра
```

Я скрыл панели mc с помощью Ctrl + O, чтобы получить доступ к командной строке выполнил трансляцию, компоновку и запуск программы.

```
rzemcov@rzemcov: ~/work/study/2025-2026/Архитектура компьютера/study_arch-pc/lab05
Session  Actions  Edit  View  Help
┌──(rzemcov㉿rzemcov)-[~]
└─$ cd ~/work/study/2025-2026/"Архитектура компьютера"/study_arch-pc/lab05/

┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ nasm -f elf lab05-1.asm

┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ ld -m elf_i386 -o lab05-1 lab05-1.o

┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ ./lab05-1
Введите строку:
Roman Zemsov
```

Программа вывела сообщение "Введите строку:". Я ввел свои ФИО и нажал Enter. Программа корректно завершила работу.

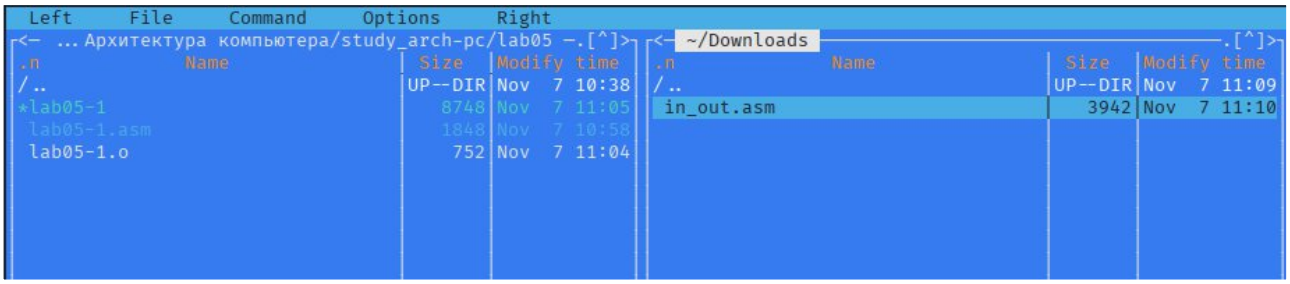
На этом этапе я научился базовым операциям в mc: навигации, созданию

каталогов (F7), созданию файлов (touch), редактированию (F4) и просмотру (F3). Я также собрал и запустил свою первую программу на NASM, используя системные вызовы write (для вывода), read (для ввода) и exit (для завершения).

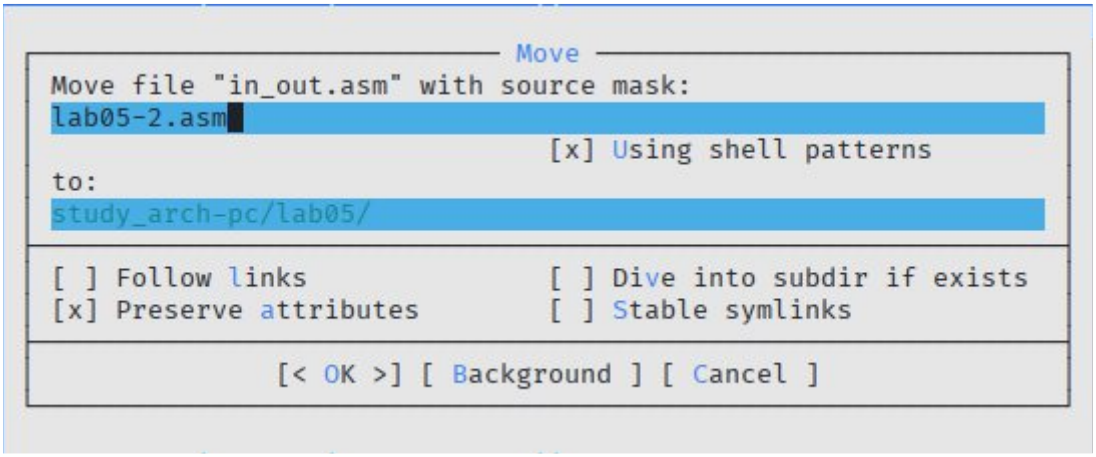
3.2. Задание 2. Подключение внешнего файла in_out.asm

Необходимо было скачать файл in_out.asm, скопировать его в рабочий каталог, создать копию lab5-1.asm с именем lab5-2.asm и модифицировать ее для использования подпрограмм из in_out.asm

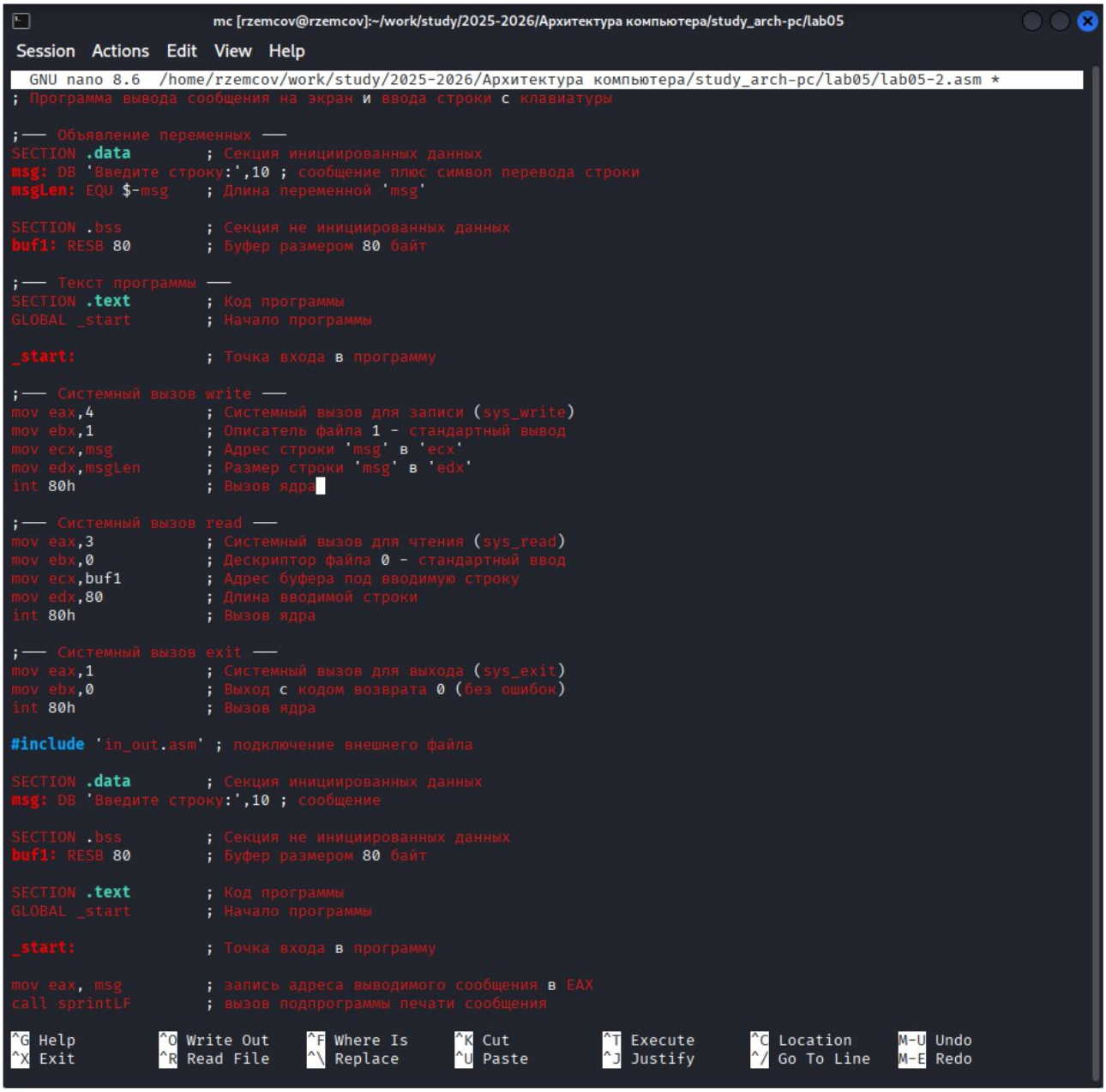
В одной панели mc я открыл свой рабочий каталог ~/work/arch-pc/lab05, а в другой панели mc (переключившись по Tab) я открыл каталог ~/Загрузки. Я выделил файл in_out.asm и нажал F5 (Копирование), подтвердив копирование в lab05.



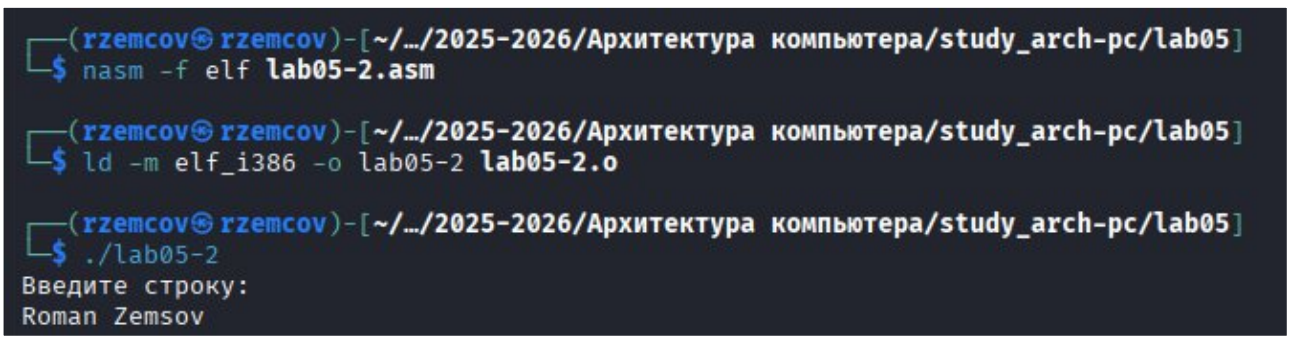
Далее я выделил файл lab5-1.asm в папке lab05, нажал F6 (Переименовать/Переместить) и ввел новое имя lab5-2.asm, чтобы создать копию.



Я открыл lab5-2.asm на редактирование (F4) и изменил код в соответствии с Листингом 5.2, используя директиву %include и вызовы call.

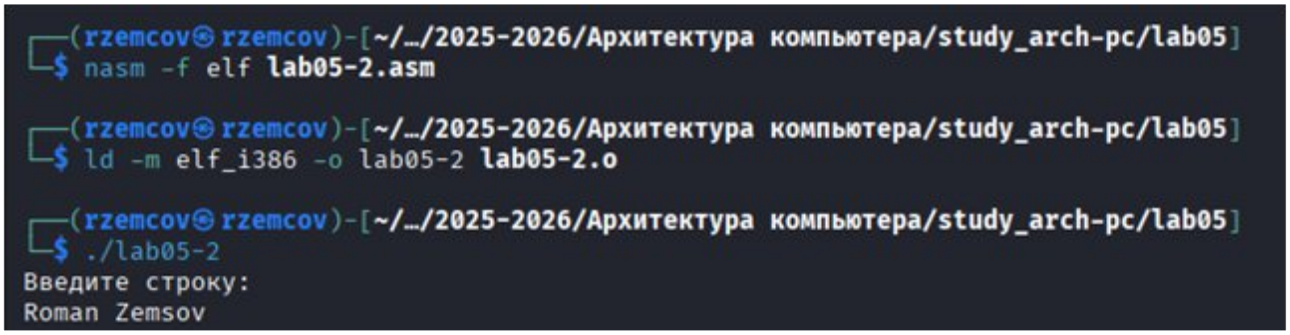


Я сохранил файл и вышел из редактора.Снова скрыв панели (Ctrl + O), я скомпилировал и запустил lab5-2.asm



Программа отработала так же, как и lab5-1.asm.

Я снова открыл lab5-2.asm и заменил call sprintLF на call sprint. Сохранил, скомпилировал и запустил.



Я научился использовать mc для операций с файлами между панелями (F5, F6). Использование in_out.asm сильно упрощает код: вместо 4-5 строк для каждого системного вызова теперь достаточно одной строки call.

Описание результатов выполнения заданий для самостоятельной работы

4.1. Задание 1. Вывод введенной строки (без in_out.asm)

Мне нужно создать копию lab5-1.asm. Модифицировать программу так, чтобы она сначала выводила приглашение, затем считывала строку с клавиатуры, а после этого выводила эту же введенную строку обратно на экран.

Я создал копию lab5-1.asm и назвал ее lab5-task-1.asm и открыл ее в редакторе (F4). После блока "системный вызов read" я добавил еще один "системный вызов write". Этот новый вызов использует в качестве адреса (ecx) буфер buf1, куда read сохранил введенную строку, и ту же длину 80 в edx.

```
        /home/rzemcov/work/study/2025-2026/Архитектура компьютера/
; Программа вывода сообщения на экран и ввода строки с клавиатур

;—— Объявление переменных ——
SECTION .data           ; Секция инициированных данных
msg: DB "Введите строку",10 ; сообщение плюс символ перевода стр
msgLen: EQU $-msg       ; Длина переменной 'msg'

SECTION .bss            ; Секция не инициированных 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 ——
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 ——
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 ——
mov eax,1               ; Системный вызов для выхода (sys_exit)
mov ebx,0               ; Выход с кодом возврата 0 (без ошибок)
int 80h                 ; Вызов ядра
```

Я скомпилировал и запустил программу. Она попросила ввести строку. Я ввел свою фамилию. Сразу после нажатия Enter, моя фамилия вывелась на следующей строке.

```
┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ nasm -f elf lab05-task-1.asm

┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ ld -m elf_i386 -o lab05-task-1 lab05-task-1.o

┌──(rzemcov㉿rzemcov)-[~/…/2025-2026/Архитектура компьютера/study_arch-pc/lab05]
└─$ ./lab05-task-1
Введите строку
Zemsov
```

Для вывода введенной строки я просто добавил еще один вызов sys_write (mov eax, 4), указав ему в ecx адрес буфера buf1, который до этого заполнил sys_read.

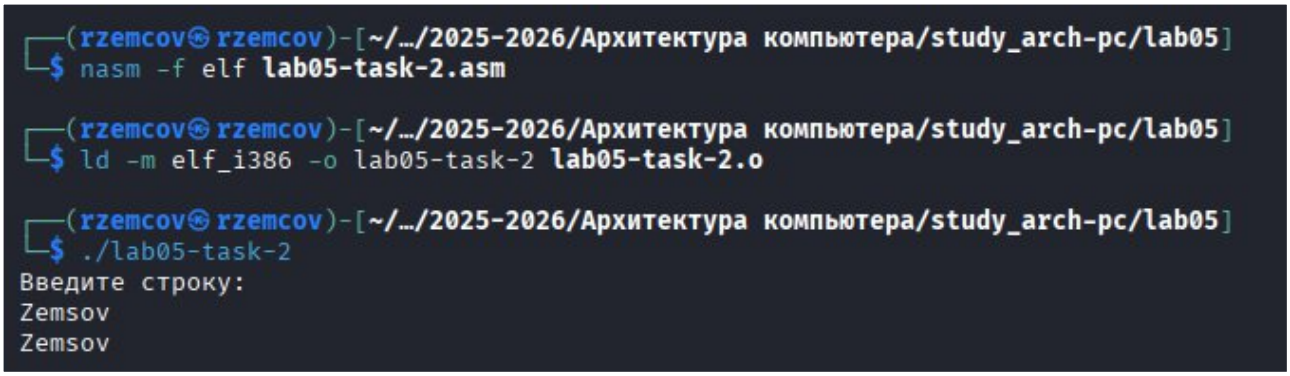
4.2. Задание 2. Вывод введенной строки (с in_out.asm)

Теперь реализуем тот же алгоритм (приглашение -> ввод -> вывод введенной строки), но с использованием подпрограмм из in_out.asm.

Я создал копию lab5-2.asm и назвал ее lab5-task-2.asm и открыл ее в редакторе (F4). осле call sread я добавил еще один вызов подпрограммы sprintLF. Перед этим вызовом я поместил в регистр eax адрес буфера buf1, так как sprintLF (как и sprint) ожидает в eax адрес того, что нужно напечатать.

```
/home/rzemcov/work/study/2025-2026/Архитектура компьютера/study_arch-pc/
; Задание для самостоятельной работы 2
; Вывод введенной строки с помощью in_out.asm

#include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg: DB 'Введите строку: ',0h ; Приглашение

SECTION .bss
buf1: RESB 80 ; Буфер для вводимой строки

SECTION .text
GLOBAL _start

_start:
; 1. Выводим приглашение
    mov eax, msg
    call sprintLF

; 2. Читаем ввод в buf1
; sread ожидает адрес в ECX и длину в EDX
    mov ecx, buf1
    mov edx, 80
    call sread

; 3. Выводим содержимое buf1 на экран
    mov eax, buf1 ; sprintLF ожидает адрес в EAX
    call sprintLF ; Печатаем то, что в буфере (с переводом строки)

; 4. Выход
    call quit
```

Я скомпилировал и запустил программу. Она отработала аналогично lab5-task-1.asm.



Эта задача решается еще проще с in_out.asm. Я просто добавил mov eax, buf1 и call sprintLF после call sread. Код получается намного чище и понятнее.

Вывод

Я получил практические навыки работы с файловым менеджером Midnight Commander: я научился создавать каталоги (F7), файлы (с помощью touch), редактировать их (F4), просматривать (F3), копировать (F5) и переименовывать (F6). Также я освоил базовую структуру программы на ассемблере NASM (секции .data, .bss, .text) и научился использовать ключевые инструкции:

- **mov**: для перемещения данных (адресов, констант) в регистры (eax, ebx, ecx, edx) для подготовки системных вызовов.
- **int 80h**: для вызова ядра Linux и выполнения системных функций, таких как sys_write, sys_read и sys_exit

Я также научился подключать внешние файлы (%include) и использовать подпрограммы (call), что значительно упрощает написание кода.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander.org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 5-е изд. — СПб. : Питер, 2018. — 1120 с. — (Классика Computer Science).