

Лабораторная работа №5

Архитектура ОС

Гаязов Рузаль Ильшатovich НКАбд-04-24

9 ноября 2023

Цель лабораторной работы

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

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.

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 (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: DB [,] [,] Для объявления неиницированных данных в секции .bss используются директивы resb, resw, resd и другие, которые сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти.

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 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.

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.

Выполнение лабораторной работы

Открыл Midnight Commander (рис. [-@fig:001]).

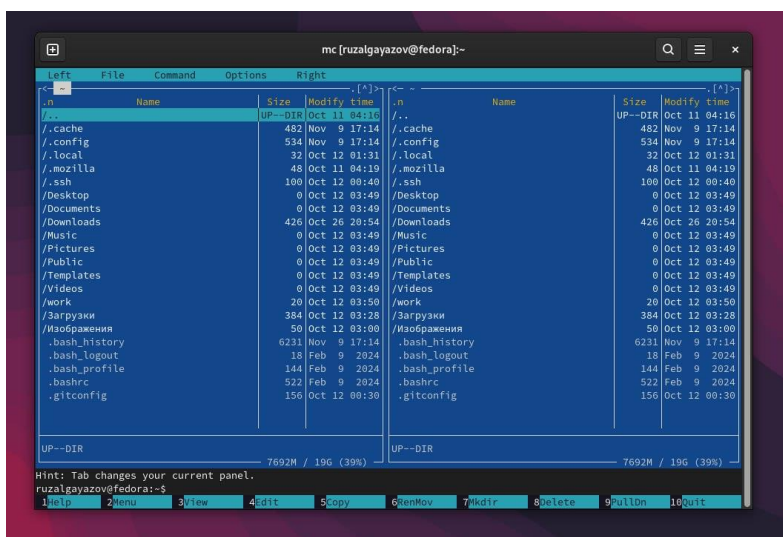


Рис.1

Создал папку `lab05` и перехожу в нее (рис. [-@fig:002]).

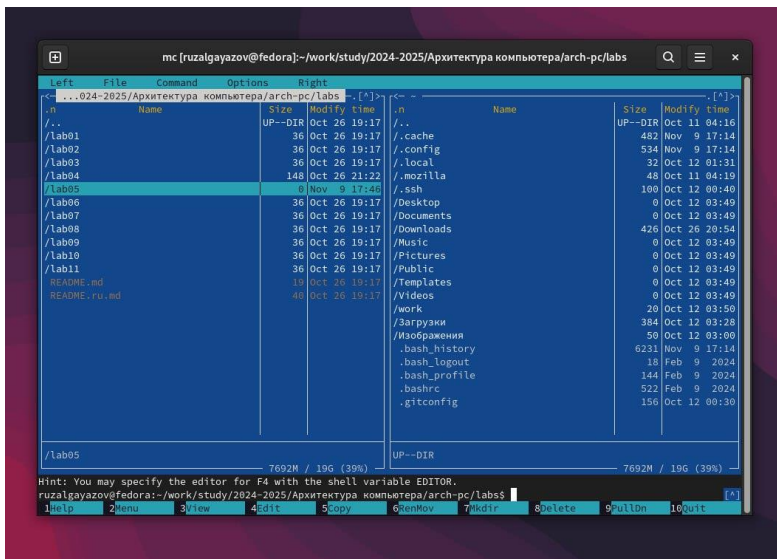


Рис.2

Пользуясь строкой ввода и командой touch создал файл lab5-1.asm (рис. [-@fig:003]).

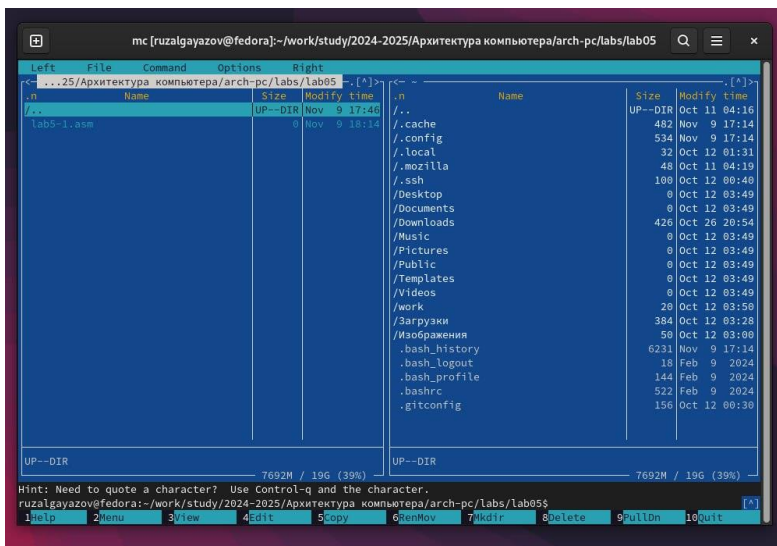


Рис.3

С помощью функциональной клавиши F4 открыл файл lab5-1.asm для редактирования во встроенном редакторе. Ввел текст программы из листинга 5.1, сохранил изменения и закрыл файл. (рис. [-@fig:004]).

Скачал файл in_out.asm со страницы курса в ТУИС. (рис. [-@fig:007]).

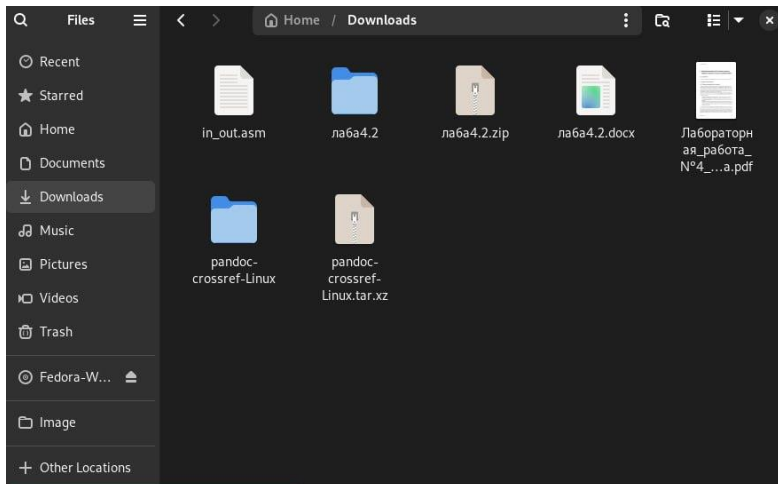


Рис.7

Скопировал файл in_out.asm в каталог с файлом lab5-1.asm с помощью функциональной клавиши F5 (рис. [-@fig:008]).

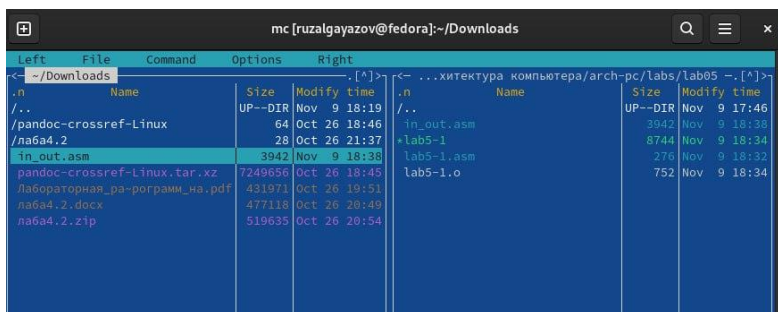


Рис.8

С помощью функциональной клавиши F6 создал копию файла lab5-1.asm с именем lab5-2.asm.(рис. [-@fig:009]).

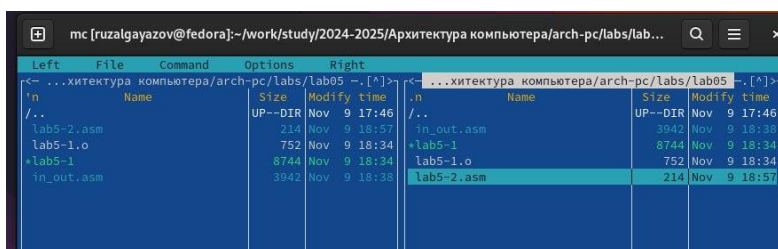


Рис.9

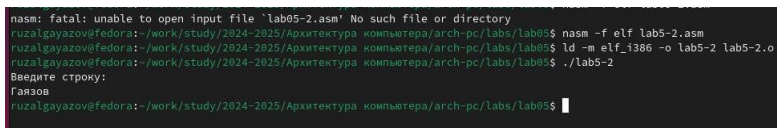
Исправил текст программы в файле lab5-2.asm с использованием подпрограмм из внешнего файла in_out.asm (используйте подпрограммы sprintLF, sread и quit) в соответствии с листингом 5.2.(рис. [-@fig:010]).



```
mc [ruzalgayazov@fedora]:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab5-2.asm
/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab5-2.asm
#include "in_out.asm"
SECTION .data
msg: DB 'Введите строку:',0h
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,msg
call sprintf
mov ecx, buf1
mov edx, 80
call sread
call quit
```

Рис.10

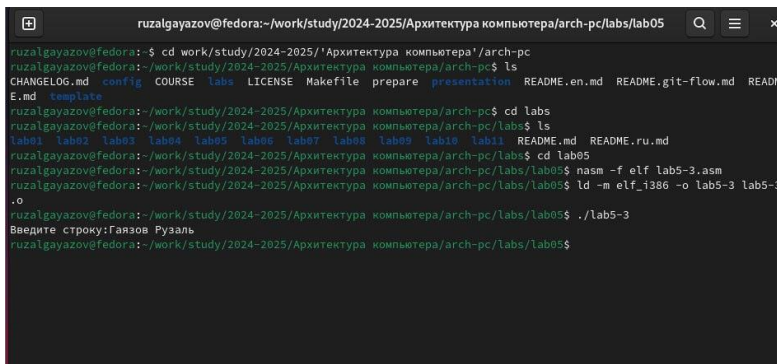
Создал исполняемый файл и проверил его работу.(рис. [-@fig:011]).



```
nasm: fatal: unable to open input file 'lab5-2.asm' No such file or directory
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ nasm -f elf lab5-2.asm
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ld -m elf_i386 -o lab5-2 lab5-2.o
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ./lab5-2
Введите строку:
Гаязов
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$
```

Рис.11

В файле lab5-2.asm заменил подпрограмму sprintf на sprint. Создал исполняемый файл и проверил его работу. Вывод: ввод строки во втором случае начался сразу же после надписи 'Введите строку'(рис. [-@fig:012]).



```
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ cd work/study/2024-2025/Архитектура компьютера/arch-pc
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc$ ls
CHANGELOG.md config COURSE labs LICENSE Makefile prepare presentation README.en.md README.git-flow.md README.ru.md template
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd labs
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs$ ls
lab01 lab02 lab03 lab04 lab05 lab06 lab07 lab08 lab09 lab10 lab11 README.md README.ru.md
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs$ cd lab05
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ nasm -f elf lab5-3.asm
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ld -m elf_i386 -o lab5-3 lab5-3.o
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ./lab5-3
Введите строку:Гаязов Рузаль
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$
```

Рис.12

Задания для самостоятельной работы

Создал два новых файла lab5-1s.asm и lab5-2s.asm(рис. [-@fig:013]).

mc [ruzalgayazov@fedora]:~/work/study/2024-2025/A				
Left	File	Command	Options	Right
< ...хитектура компьютера/arch-pc/labs/lab05 -.[^]>				
'n	Name		Size	Modify time
/..		UP--DIR		Nov 9 17:46
	report.md		5819	Oct 26 19:17
	lab5-3.o		1312	Nov 9 19:09
	lab5-3.asm		212	Nov 9 19:07
*lab5-3			9092	Nov 9 19:10
	lab5-2s.asm		306	Nov 9 19:28
	lab5-2.o		1312	Nov 9 19:02
	lab5-2.asm		214	Nov 9 18:57
	lab5-1s.asm		422	Nov 9 19:24
	lab5-1.o		752	Nov 9 18:34
	lab5-1.asm		281	Nov 9 19:18
*lab5-1			8744	Nov 9 18:34
	in_out.asm		3942	Nov 9 18:38

Рис.13

lab5-1s.asm(рис. [-@fig:014]).


```

SECTION .data
msg: DB 'Введите строку:',10
msgLen: EQU $-msg

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, msgLen
    int 80h

    mov eax, 3
    mov ebx, 0
    mov ecx, buf1
    mov edx, 80
    int 80h

    mov eax, 4
    mov ebx, 1
    mov ecx, buf1
    mov edx, 80
    int 80h

    mov eax, 1
    mov ebx, 0

```

Рис.14

lab5-2s.asm(рис. [-@fig:015]).

```

#include 'in_out.asm'

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

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:
    mov eax, msg
    call print

    mov eax, buf1
    mov ebx, 80
    call read

    mov eax, buf1
    call print

    mov eax, 1
    xor ebx, ebx
    int 80h

```

Рис.15

Создал исполняемый файл и проверил его работу lab5-1s.asm(рис. [-@fig:016]).

```

ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ nasm -f elf lab5-1s.asm
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ld -m elf_i386 -o lab5-1s
ld: no input files
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ld -m elf_i386 -o lab5-1s lab5-1s.o
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ./lab5-1s
Введите строку:
Гаязов Рузаль Ильшатович
Гаязов Рузаль Ильшатович
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$

```

Рис.16

Создал исполняемый файл и проверил его работу lab5-2s.asm(рис. [-@fig:016]).

```

ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ nasm -f elf lab5-2s.asm
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ld -m elf_i386 -o lab5-2s lab5-2s.o
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$ ./lab5-2s
Введите строку:
Гаязов Рузаль
Гаязов Рузаль
ruzalgayazov@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab05$

```

Рис.17

Вывод

Я приобрел практические навыки работы в Midnight Commander. Освоил инструкции языка ассемблера mov и int.

Список источников

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://midnightcommander.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 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — 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,
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. — БХВПетербург, 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. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).