Отчёт по лабораторной работе №5

Архитектура компьютера

Алехин Давид Андреевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Создать программу вывода сообщения на экран и ввода строки с клавиатуры
2. Создать программу вывода сообщения на экран и ввода строки с клавиатуры с использованием файла in_out.asm
3. Создать программу вывода сообщения на экран и ввода строки с клавиатуры и вывода сторки
4. Создать программу вывода сообщения на экран и ввода строки с клавиатуры с использованием файла in_out.asm и вывода строки

3 Теоретическое введение

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 непо- средственные значения (const). ВАЖНО! Переслать значение из одной ячейки памяти в другую нельзя, для этого необхо- димо 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 intпредназначена 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 систем- ного 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 по- рядке 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 — использо- вать 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 ассем- блер. 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

Открываю mc (Midnight Commander). (рис. 4.1).

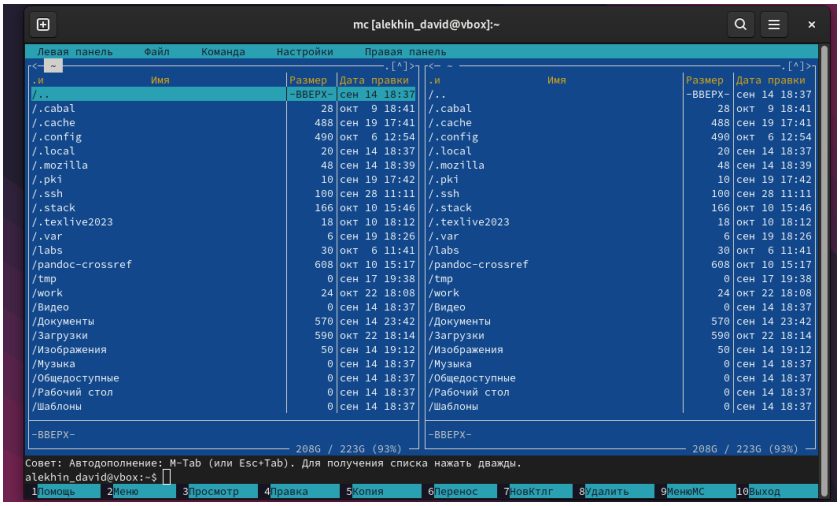

Рис. 4.1: Midnight Commander

Создаю каталог lab05 с файлом lab5-1.asm. (рис. 4.2).

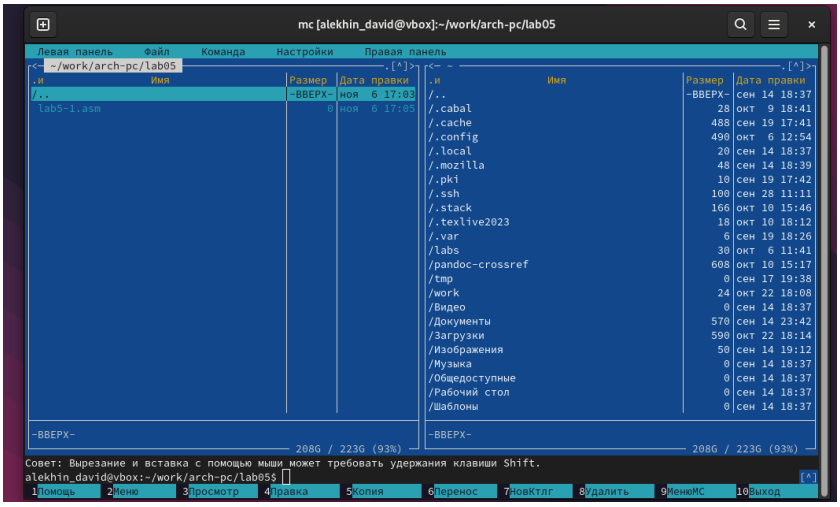

Рис. 4.2: lab05/lab5-1.asm

Ввожу в lab5-1.asm код команды “Программа вывода сообщения на экран и ввода строки с клавиатуры”. (рис. 4.3).

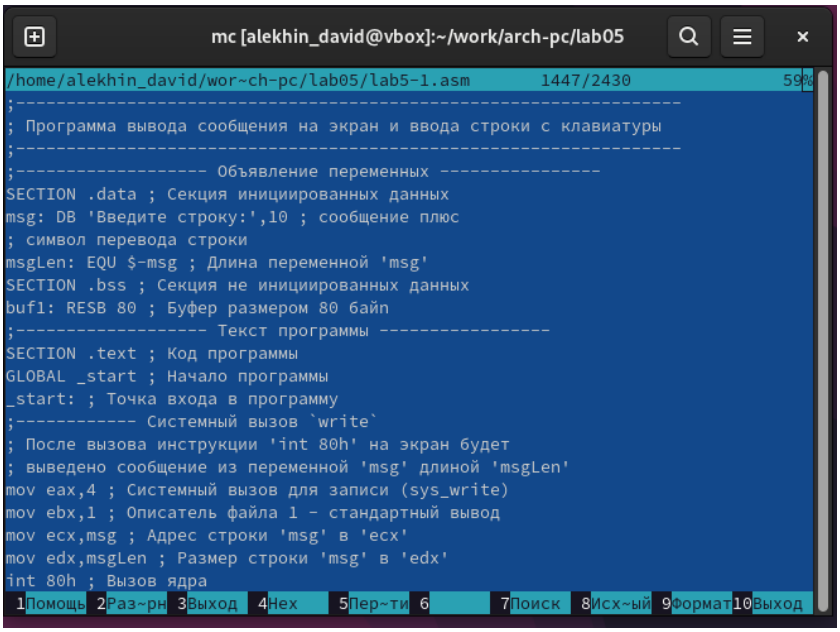

Рис. 4.3: Программа вывода сообщения на экран и ввода строки с клавиатуры

Компаную и запускаю файл lab5-1.asm. (рис. 4.4).

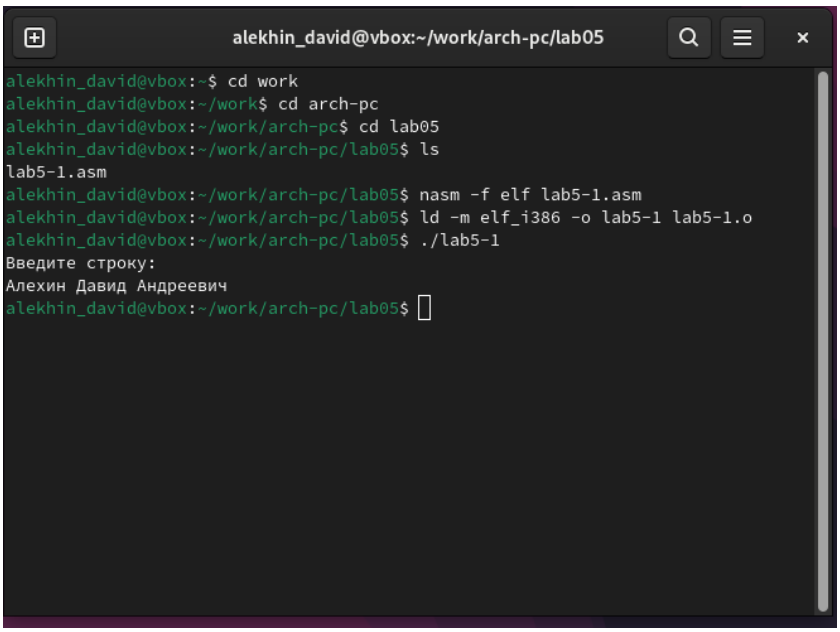

Рис. 4.4: Компаную и запускаю файл lab5-1.asm.

Скачиваю файл in_out.asm и добавляю его в каталог lab05. (рис. 4.5).

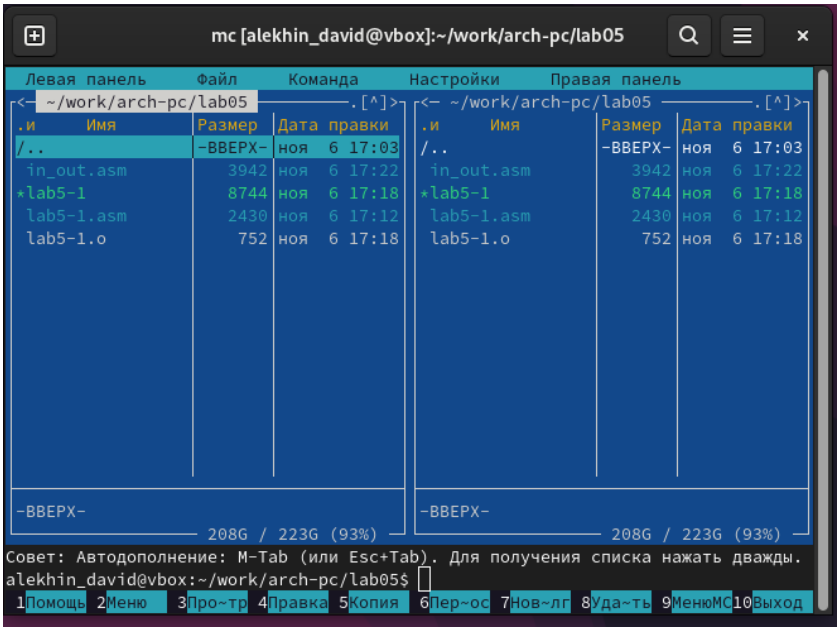

Рис. 4.5: lab05/in_out.asm

Создаю копию lab5-1.asm lab05-2.asm. (рис. 4.6).

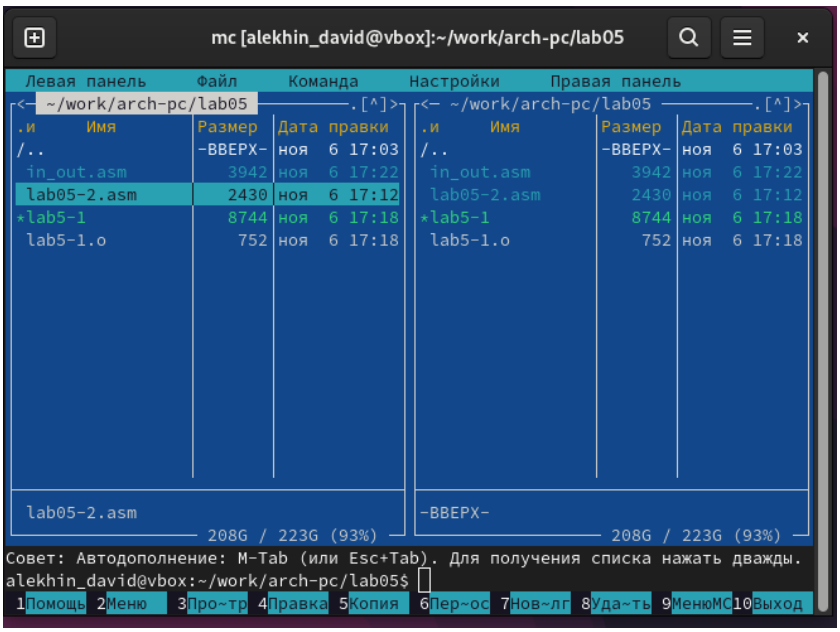

Рис. 4.6: lab05/lab05-2.asm

Ввожу в lab05-2.asm программу вывода сообщения на экран и ввода строки с клавиатуры с использованием файла in_out.asm, компаную и запускаю его. (рис. 4.7).

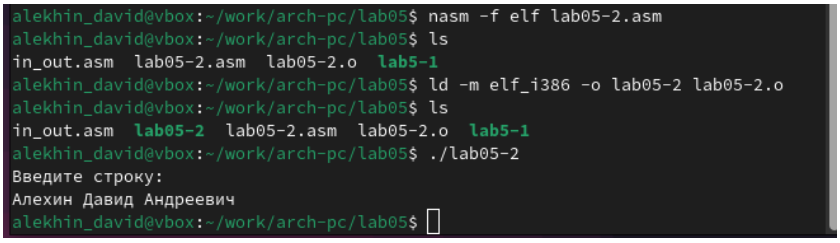

Рис. 4.7: Исполнение lab05-2.asm

В файле lab05-2.asm.2 (Копии программы lab05-2.asm) меняю sprintLF на sprint. Компаную и исполняю его. Разница между lab05-2.asm и lab05-2.asm.2 в том что lab05-2.asm с sprintLF переносит строку для ввода, а lab05-2.asm.2 с sprint не переносит. (рис. 4.8).

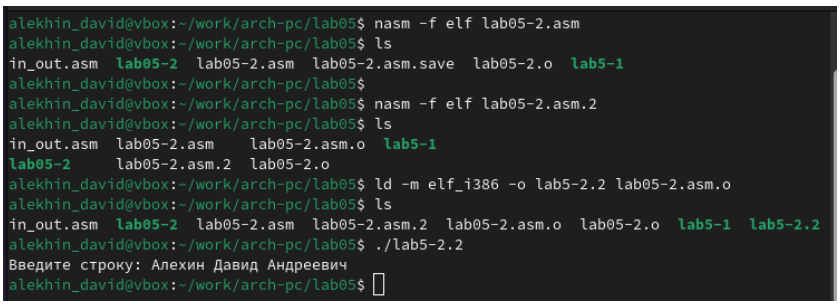

Рис. 4.8: Исполнение lab05-2.asm.2

Создаю файл lab05-1.1.asm, копию файла lab05-1.asm. (рис. 4.9).

Рис. 4.9: lab05/lab05-1.1.asm

Вожду туда код для вывода ранее введённой строки. (рис. 4.10).

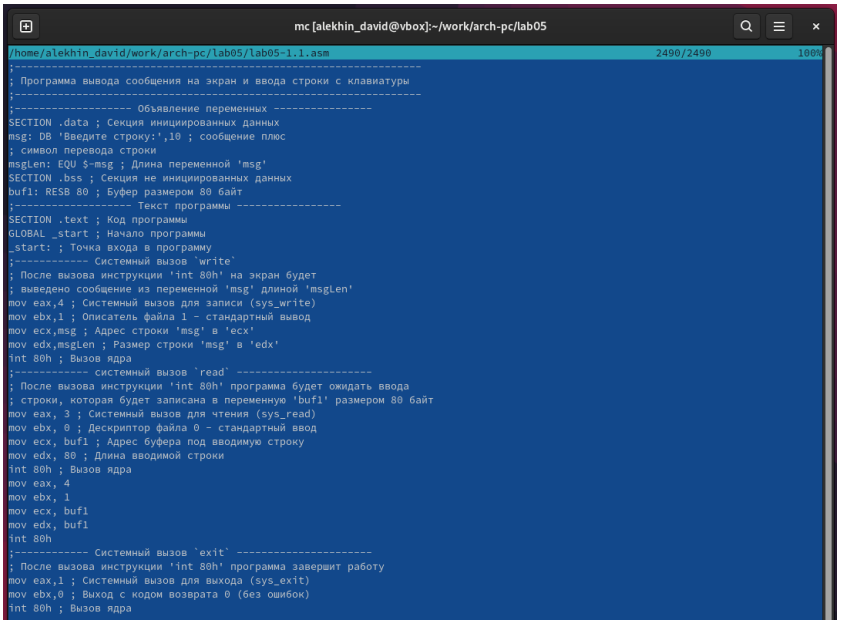

Рис. 4.10: Текст lab05-1.1.asm

Компаную и запускаю lab05-1.1.asm. (рис. 4.11).

Рис. 4.11: Запуск lab05-1.1.asm

Провожу ту же самую раблту с файлом lab05-2.asm. (рис. 4.12, 4.13, 4.14).

Рис. 4.12: Создание lab05-2.2.asm

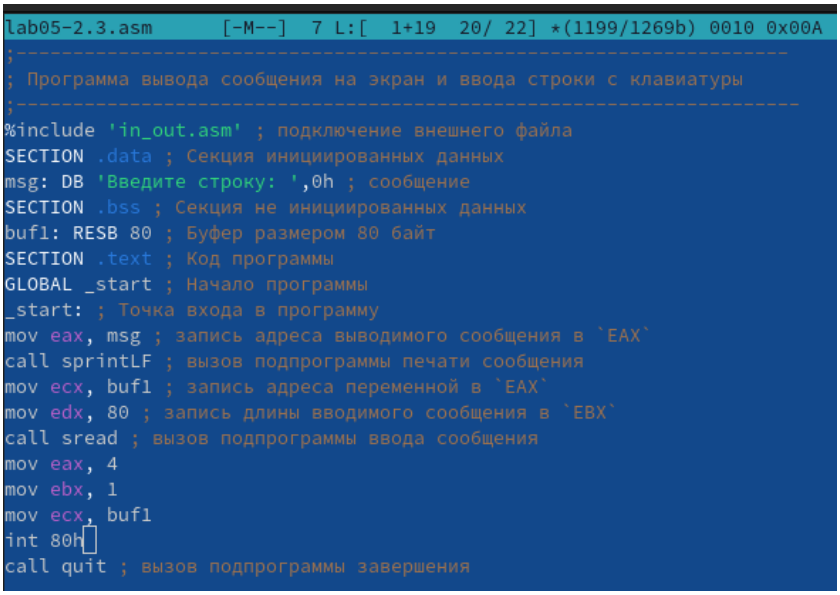

Рис. 4.13: Текст lab05-2.2.asm

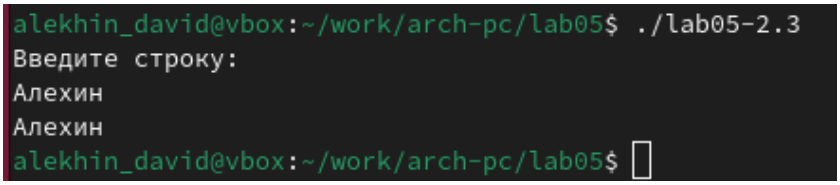

Рис. 4.14: Запуск lab05-2.2.asm

5 Выводы

После выполненя лабораторной работы я приобрёл практические навыки работы в Midnight Commander. Изучил основные инструкций языка ассемблера mov и int.

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,
11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. 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.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science).