

Лабораторная работа 4

**Создание и процесс обработки программ на языке ассемблера
NASM**

Головина Мария Игоревна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	13
5	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Создание каталога для работы с программами на языке ассемблера NASM	13
4.2	Созданный каталог для лабораторной работы	13
4.3	Создание текстового файла	14
4.4	Созданный текстовый файл	14
4.5	Компиляция текста программы	14
4.6	Компиляция исходного файла hello.asm в obj.o	15
4.7	Передача объектного файла на обработку компоновщику	15
4.8	Выполнение команды <code>ld -m elf_i386 obj.o -o main</code>	16
4.9	Выполнение созданного файла	16
4.10	Создание копии файла hello.asm с именем lab4.asm	16
4.11	Текст программы	17
4.12	Выполнение полученного текста программы lab4.asm	17
4.13	Загрузка файлов на Github	18

Список таблиц

1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

1. Создать каталог для работы с программами на языке ассемблера NASM.
2. Перейти в созданный каталог.
3. Создать текстовый файл с именем hello.asm.
4. Открыть этот файл с помощью текстового редактора и ввести необходимый текст.
5. Скомпилировать написанный текст программы «Hello World».
6. Выполнить команду: `nasm -o obj.o -f elf -g -l list.lst hello.asm`.
7. Передать объектный файл на обработку компоновщику.
8. Выполнить команду: `ld -m elf_i386 obj.o -o main`.
9. Запустить на выполнение созданный исполняемый файл.

Задание для самостоятельной работы

1. Создать копию файла hello.asm с именем lab4.asm.
2. Внести изменения в текст программы, чтобы вместо Hello world! на экран выводилась строка с фамилией и именем.
3. Оттранслировать полученный текст программы lab4.asm в объектный файл. Выполнить компоновку объектного файла и запустить получившийся исполняемый файл.
4. Скопировать файлы hello.asm и lab4.asm в локальный репозиторий и загрузить файлы на Github.

3 Теоретическое введение

Основные принципы работы компьютера Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) входят следующие устройства: арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; регистры — сверхбыстрая оперативная память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между

регистрами и памятью, преобразование (арифметические или логические операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита.

В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ):

- RAX, RCX, RDX, RBX, RSI, RDI — 64-битные;
- EAX, ECX, EDX, EBX, ESI, EDI — 32-битные;
- AX, CX, DX, BX, SI, DI — 16-битные;
- АН, AL, СН, CL, DH, DL, ВН, BL — 8-битные (половинки 16-битных регистров).

Например, АН (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX.

Таким образом, можно отметить, что вы можете написать в своей программе, например, такие команды (mov – команда пересылки данных на языке ассемблера):

```
mov ax, 1  
mov eax, 1
```

Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1. Другим важным узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает,

что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы. Программа состоит из машинных команд, которые указывают, какие операции и над какими данными (или операндами), в какой последовательности необходимо выполнить. Набор машинных команд определяется устройством конкретного процессора. Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. В самом общем виде он заключается в следующем:

1. формирование адреса в памяти очередной команды;
2. считывание кода команды из памяти и её дешифрация;
3. выполнение команды; 4. переход к следующей команде.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы.

Ассемблер и язык ассемблера Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл програм-

мист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора. Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются:

- для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);
- для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. метка перед командой связана с адресом данной команды. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

Процесс создания и обработки программы на языке ассемблера

В процессе создания ассемблерной программы можно выделить четыре шага:

- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`.
- Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`.
- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение `map`.
- Запуск программы. Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной

работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

4 Выполнение лабораторной работы

1. Создали каталог для работы с программами на языке ассемблера NASM (рис. 4.1 Создание каталога для работы с программами на языке ассемблера NASM).



```
migolovina@dk3n62 - migolovina
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $
```

Рис. 4.1: Создание каталога для работы с программами на языке ассемблера NASM

2. Перешли в созданный каталог (рис. 4.2 Созданный каталог для лабораторной работы).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.2: Созданный каталог для лабораторной работы

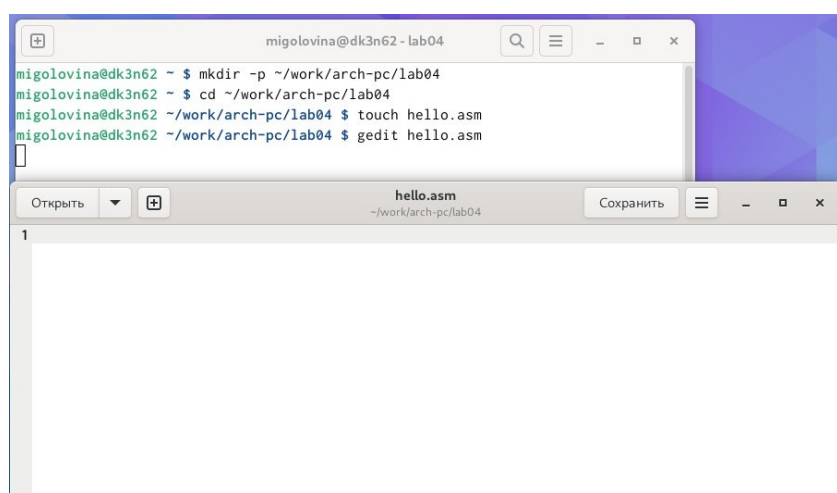
3. 3 Создали текстовый файл с именем hello.asm (рис. 4.3 Создание текстового файла).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.3: Создание текстового файла

4. Открыли этот файл с помощью текстового редактора и ввели необходимый текст (рис. 4.4 Созданный текстовый файл).

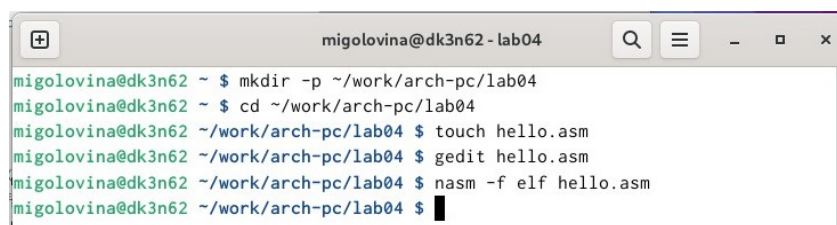


```
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
```

The text editor window shows a single line with the number 1, indicating the cursor is at the beginning of the first line of the file.

Рис. 4.4: Созданный текстовый файл

5. Компилировали написанный текст программы «Hello World» (рис. 4.5 Компиляция текста программы).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.5: Компиляция текста программы

6. Выполнили команду: `nasm -o obj.o -f elf -g -l list.lst hello.asm` (рис. 4.6 Компиляция исходного файла `hello.asm` в `obj.o`).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hel
lo.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.6: Компиляция исходного файла `hello.asm` в `obj.o`

7. Передали объектный файл на обработку компоновщику (рис. 4.7 Передача объектного файла на обработку компоновщику).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hel
lo.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

8. Выполнили команду: `ld -m elf_i386 obj.o -o main` (рис. 4.8 Выполнение ко-манды `ld -m elf_i386 obj.o -o main`).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hel
lo.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.8: Выполнение команды `ld -m elf_i386 obj.o -o main`

9. Запустили на выполнение созданный исполняемый файл (рис. 4.9 Выполнение созданного файла).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~ $ mkdir -p ~/work/arch-pc/lab04
migolovina@dk3n62 ~ $ cd ~/work/arch-pc/lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ touch hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hel
lo.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ./hello
Hello world!
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.9: Выполнение созданного файла

Задание для самостоятельной работы

1. Создали копию файла `hello.asm` с именем `lab4.asm` (рис. 4.10 Создание копии файла `hello.asm` с именем `lab4.asm`).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.10: Создание копии файла `hello.asm` с именем `lab4.asm`

2. Внесли изменения в текст программы (рис. 4.11 Текст программы).

```
SECTION .data
    hello:          DB 'Головина Мария',10

    helloLen:       EQU $-hello

SECTION .text
    GLOBAL _start
_start:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h

    mov eax,1
    mov ebx,0
    int 80h
```

Рис. 4.11: Текст программы

3. Оттранслировали полученный текст программы lab4.asm в объектный файл. Выполнили компоновку объектного файла и запустили получившийся исполняемый файл (рис. 4.12 Выполнение полученного текста программы lab4.asm).



```
migolovina@dk3n62 - lab04
migolovina@dk3n62 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ gedit lab4.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst lab4.asm
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
migolovina@dk3n62 ~/work/arch-pc/lab04 $ ./lab4
Головина Мария
migolovina@dk3n62 ~/work/arch-pc/lab04 $
```

Рис. 4.12: Выполнение полученного текста программы lab4.asm

4. Скопировали файлы hello.asm и lab4.asm в локальный репозиторий и загрузили файлы на Github (рис. 4.13 Загрузка файлов на Github).

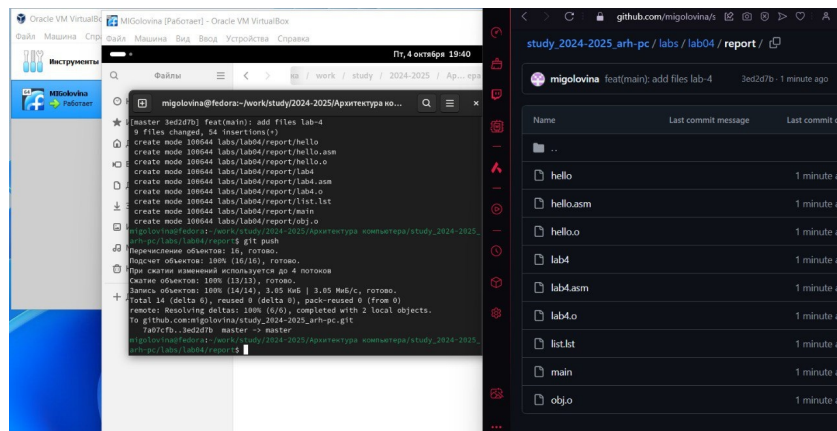


Рис. 4.13: Загрузка файлов на Github

5 Выводы

Освоили процедуры компиляции и сборки программ, написанных на ассемблере NASM.

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 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, 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. — 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).