# Отчёт по лабораторной работе №6

Дисциплина: Архитектура компьютера

Павлова Татьяна Юрьевна

## Содержание



## 1 Цель работы

Целью данной работы является освоение арифметических инструкций языка ассемблера NASM

## 2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий для самостоятельной работы

## 3 Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти.

• Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx. • Непосредственная адресация – значение операнда задается непосредственно в команде, Например: mov ax,2. • Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию. Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации

производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

# 4 Выполнение лабораторной работы

##Символьные и численные данные в NASM

Создайте каталог для программам лабораторной работы № 6, перейдите в него и создайте файл lab6-1.asm: mkdir ~/work/arch-pc/lab06 cd ~/work/arch-pc/lab06 touch lab6-1.asm (рис. 1).


*Рис. 1: Создание каталога и файла*

Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр eax.

ВАЖНО! Для корректной работы программы подключаемый файл in_out.asm должен лежать в том же каталоге, что и файл с текстом программы. Перед созданием исполняемого файла создайте копию файла in_out.asm в каталоге ~/work/arch-pc/lab06 (рис. 2).

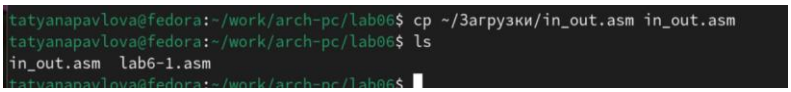


*Рис. 2: Создание копии файла в нужном каталоге*

Введите в файл lab6-1.asm текст программы из листинга 6.1. В данной программе в регистр eax записывается символ 6 (mov eax,‘6’), в регистр ebx символ 4 (mov ebx,‘4’). Далее к значению в регистре eax прибавляем значение регистра ebx (add eax,ebx, результат сложения запишется в регистр eax). Далее выводим результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра eax в переменную buf1 (mov [buf1],eax), а затем запишем адрес переменной buf1 в регистр eax (mov eax,buf1) и вызовем функцию sprintLF.

Создайте исполняемый файл и запустите его. nasm -f elf lab6-1.asm ld -m elf_i386 -o lab6-1 lab6-1.o ./lab6-1 (рис. 3).


*Рис. 3: Запуск файла*

В данном случае при выводе значения регистра eax мы ожидаем увидеть число 10. Однако результатом будет символ j. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда add eax,ebx запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j.

Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 6.1) следующим образом: замените строки mov eax,‘6’ mov ebx,‘4’ на строки mov eax,6 mov ebx,4 (рис. 4).

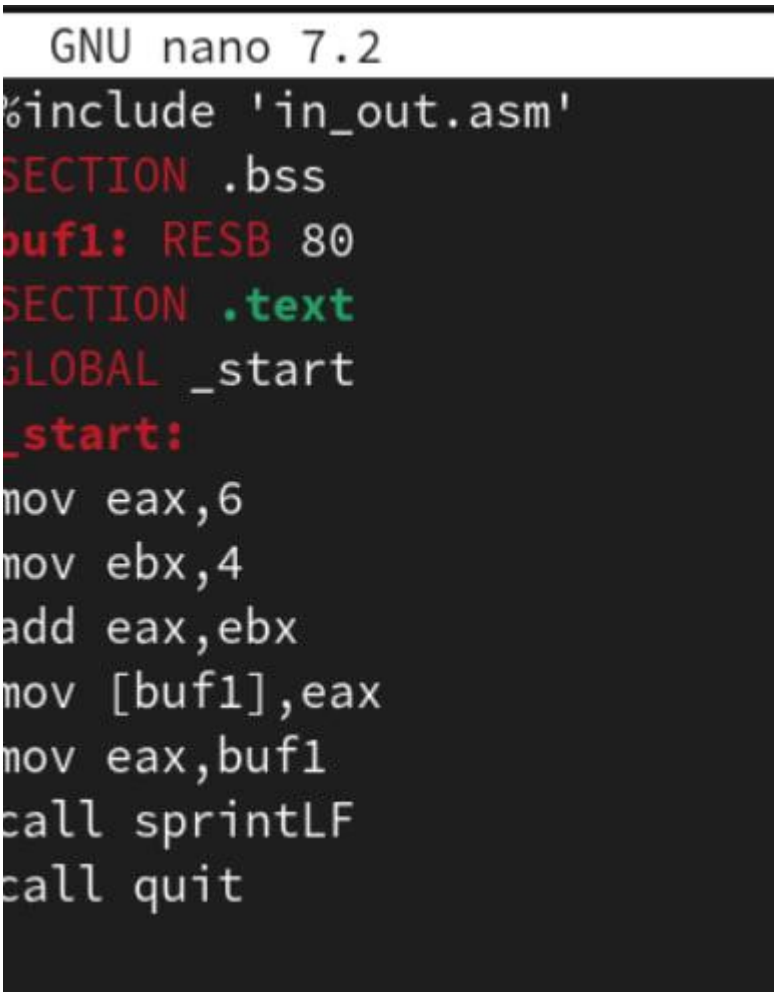


*Рис. 4: Измененный файл*

Создайте исполняемый файл и запустите его (рис. 5).

```
tatyanapavlova@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ./lab6-1
```

*Рис. 5: Запуск файла*

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Это символ перевода строки, этот символ не отображается при выводе на экран.

Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 6.1 с использованием этих функций.

Создайте файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введите в него текст программы из листинга 6.2. touch ~/work/arch-pc/lab06/lab6-2.asm (рис. 6).

```
tatyanapavlova@fedora:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-2.asm
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ls
in_out.asm  lab6-1  lab6-1.asm  lab6-1.o  lab6-2.asm
```

*Рис. 6: Создание файла*

Создайте исполняемый файл и запустите его. (рис. 7).

```
tatyanapavlova@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ./lab6-2
106
```

*Рис. 7: Запуск файла*

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда add складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

Аналогично предыдущему примеру изменим символы на числа. Замените строки mov eax,‘6’ mov ebx,‘4’ на строки mov eax,6 mov ebx,4 (рис. 8).

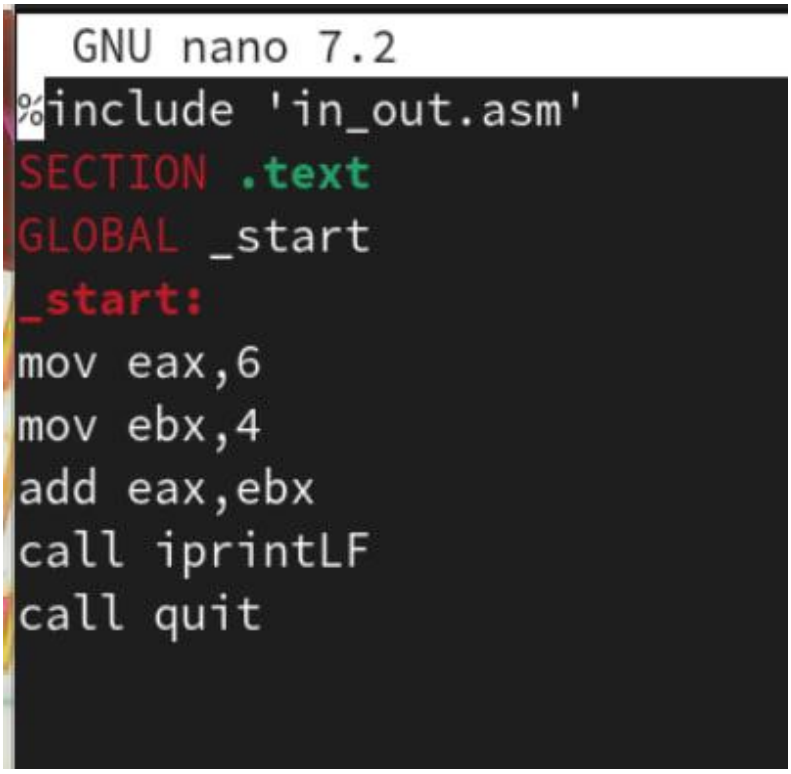


*Рис. 8: Измененный файл*

Создайте исполняемый файл и запустите его. (рис. 9).


*Рис. 9: Запуск файла*

Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10.

## 4.1 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 * 2 + 3)/3$.

Создайте файл lab6-3.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/lab6-3.asm (рис. 10).

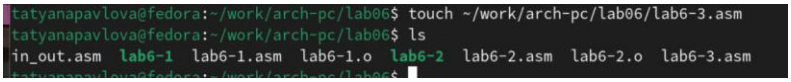


*Рис. 10: Создание файла*

Внимательно изучите текст программы из листинга 6.3 и введите в lab6-3.asm. (рис. 11).

```
 GNU nano 7.2                                          /home/ta
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения
call iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
```

*Рис. 11: Ввод текста в файл*

Создайте исполняемый файл и запустите его. Результат работы программы должен быть следующим: user@dk4n31:~$ ./lab6-3 Результат: 4 Остаток от деления: 1 user@dk4n31:~$ (рис. 12).

```
tatyanapavlova@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
```

*Рис. 12: Результат запуска файла*

Измените текст программы для вычисления выражения $f(x)$ = (4 ∗ 6 + 2)/5. Создайте исполняемый файл и проверьте его работу. (рис. 13).

```
tatyanapavlova@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
tatyanapavlova@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
```

*Рис. 13: Результат запуска измененного файла*

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: • вывести запрос на введение № студенческого билета • вычислить номер варианта по формуле: ($Sn$ mod 20) + 1, где $Sn$ – номер студенческого билета (В данном случае $a$ mod $b$ – это остаток от деления $a$ на

$b$). • вывести на экран номер варианта. В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция atoi из файла in_out.asm.

Создайте файл variant.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/variant.asm (рис. 14).


*Рис. 14: Создание файла*

Внимательно изучите текст программы из листинга 6.4 и введите в файл variant.asm. (рис. 15).

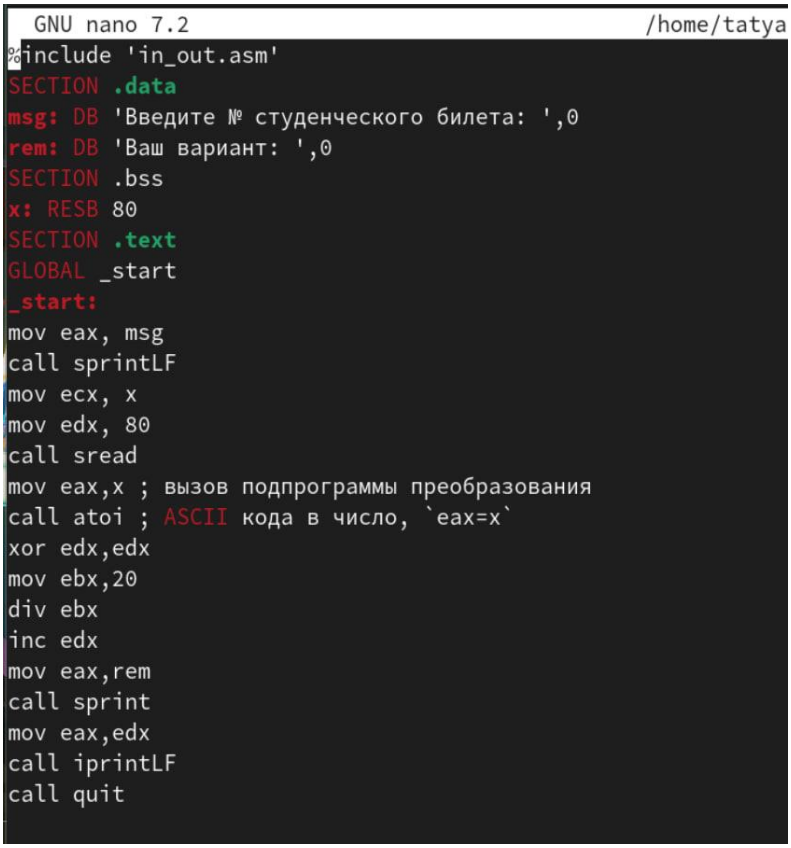


*Рис. 15: Ввод текста в файл*

Создайте исполняемый файл и запустите его. Проверьте результат работы программы вычислив номер варианта аналитически. (рис. 16).

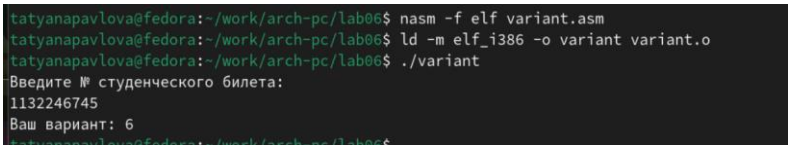


*Рис. 16: Результат работы программы*

## 4.2 Ответы на вопросы

Включите в отчет по выполнению лабораторной работы ответы на следующие вопросы:

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

За вывод сообщения “Ваш вариант” отвечают строки кода: mov eax,rem call sprint

2. Для чего используется следующие инструкции? mov ecx, x mov edx, 80 call sread

Инструкция mov ecx, x используется, чтобы положить адрес вводимой строки x в регистр ecx mov edx, 80 - запись в регистр edx длины вводимой строки call sread - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры

3. Для чего используется инструкция “call atoi”?

“call atoi” используется для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

За вычисления варианта отвечают строки: xor edx,edx ; обнуление edx для корректной работы div mov ebx,20 ; ebx = 20 div ebx ; eax = eax/20, edx - остаток от деления inc edx ; edx = edx + 1

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

При выполнении инструкции “div ebx” остаток от деления записывается в регистр edx

6. Для чего используется инструкция “inc edx”?

Инструкция “inc edx” увеличивает значение регистра edx на 1

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

За вывод на экран результатов вычислений отвечают строки: mov eax,edx call iprintLF

# 5 Выполнение заданий для самостоятельной работы

Создаю файл lab6-4.asm с помощью утилиты touch (рис. 17).


*Рис. 17: Создание файла*

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения $x$^^3/2 + 1. Это выражение было под вариантом 15. (рис. 18).

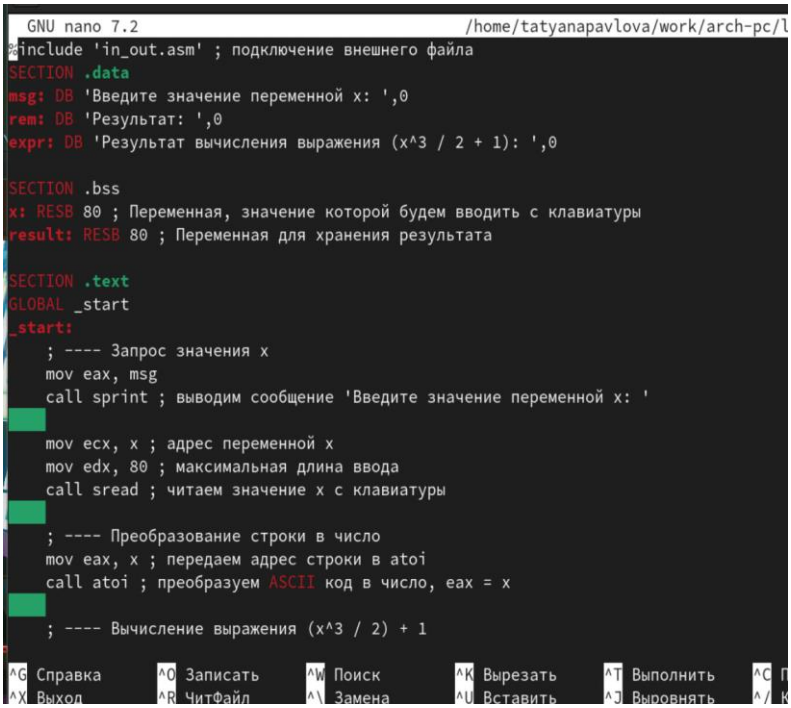


*Рис. 18: Редактированный файл*

Создаю и запускаю исполняемый файл.

При вводе значения 2, вывод — 5. (рис. 19).

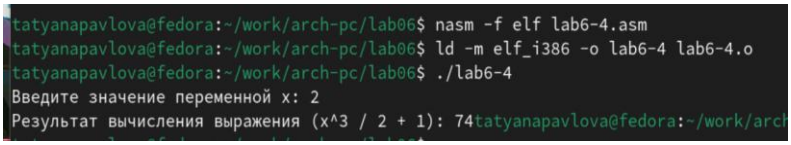


*Рис. 19: Результат работы программы*

При вводе значения 5, вывод — 63,5. (рис. 20).

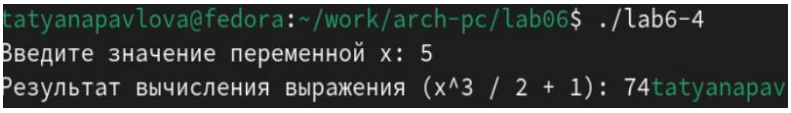


*Рис. 20: Результат работы программы*

# 6 Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера 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).