

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Петрова Алевтина Александровна

Содержание

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания для самостоятельной работы.

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов:

- условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия.
- безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

Безусловный переход выполняется инструкцией `jmp`. Инструкция `str` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `str` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

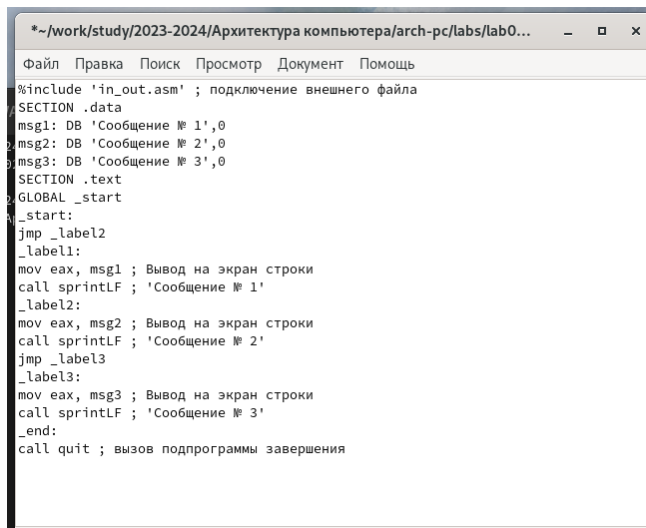
Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл `lab7-1.asm`. (рис.1).



```
[petrovkina1002@fedora ~]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/Labs/Lab07/Lab7
[petrovkina1002@fedora ~]$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/Labs/Lab07/Lab7
[petrovkina1002@fedora lab7]$ touch lab7-1.asm
[petrovkina1002@fedora lab7]$
```

Figure 1: рис.1 Создание файлов для лабораторной работы

Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 2).

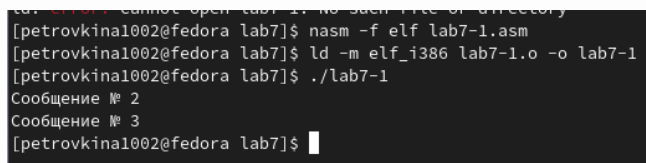


```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab0...
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label3
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Figure 2: рис.2 Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 3).

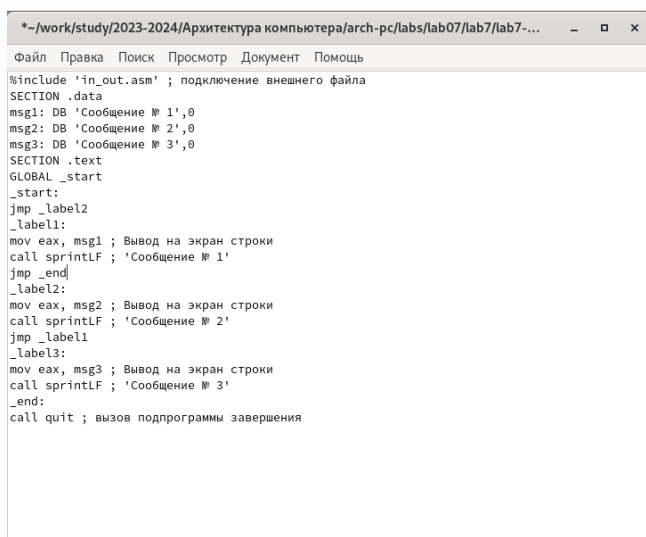


```
[petrovkina1002@fedora lab7]$ nasm -f elf lab7-1.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 lab7-1.o -o lab7-1
[petrovkina1002@fedora lab7]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[petrovkina1002@fedora lab7]$
```

Figure 3: рис.3 Запуск программного кода

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4).



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7/lab7-...
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

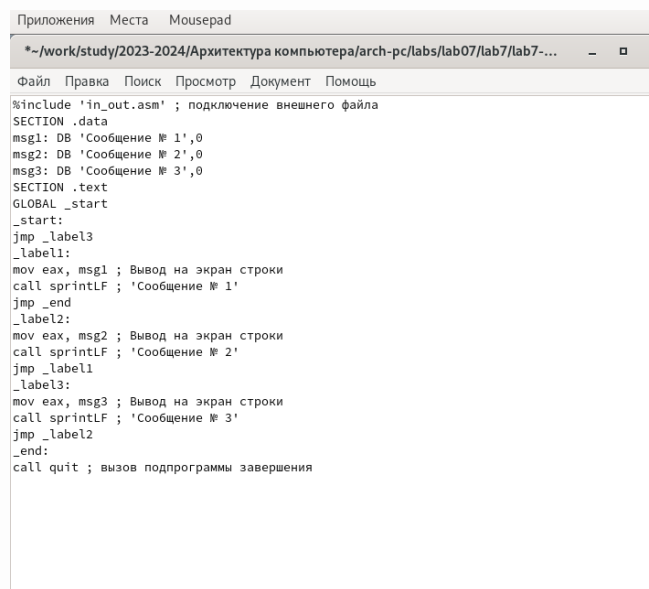
Figure 4: рис.4 Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 5).

```
[petrovkina1002@fedora lab7]$ mousepad lab7-1.asm
[petrovkina1002@fedora lab7]$ nasm -f elf lab7-1.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 lab7-1.o -o lab7-1
[petrovkina1002@fedora lab7]$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Figure 5: рис.5 Создание исполняемого файла

Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. 6).



```
Приложения Места Mousepad
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7/lab7-...
Файл Правка Поиск Просмотр Документ Помощь
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

Figure 6: рис.6 Изменение текста программы

чтобы вывод программы был следующим: (рис. 7).

```
[petrovkina1002@fedora lab7]$ mousepad lab7-1.asm
[petrovkina1002@fedora lab7]$ nasm -f elf lab7-1.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 lab7-1.o -o lab7-1
[petrovkina1002@fedora lab7]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[petrovkina1002@fedora lab7]$
```

Figure 7: рис.7 Вывод программы

Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создаю файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. (рис. 8).

```
[petrovkina1002@fedora lab7]$ touch lab7-2.asm
[petrovkina1002@fedora lab7]$
```

Figure 8: рис.8 Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 9).

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7/lab7-2.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
```

Figure 9: рис.9 Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 10).

```
[petrovkina1002@fedora lab7]$ mousepad lab7-2.asm
[petrovkina1002@fedora lab7]$ nasm -f elf lab7-2.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 lab7-2.o -o lab7-2
[petrovkina1002@fedora lab7]$ ./lab7-2
Введите B: 10
Наибольшее число: 50
[petrovkina1002@fedora lab7]$ nasm -f elf lab7-2.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 lab7-2.o -o lab7-2
[petrovkina1002@fedora lab7]$ ./lab7-2
Введите B: 100
Наибольшее число: 100
[petrovkina1002@fedora lab7]$
```

Figure 10: рис.10 Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 11).

```
[petrovkina1002@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[petrovkina1002@fedora lab7]$
```

Figure 11: рис.11 Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 12).

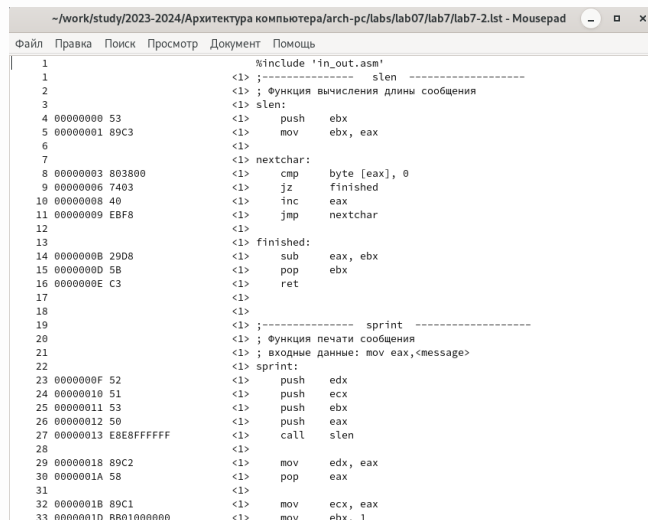


Figure 12: рис.12 Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 13).

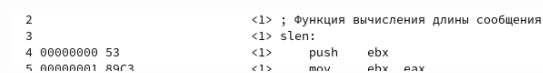


Figure 13: рис.13 Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны сообщения” - комментарий к коду, не имеет адреса и машинного кода.

“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.

“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. 14).

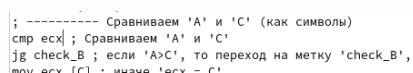


Figure 14: рис.14 Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 15).

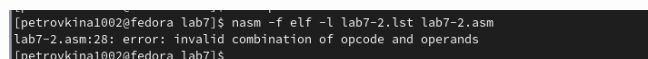
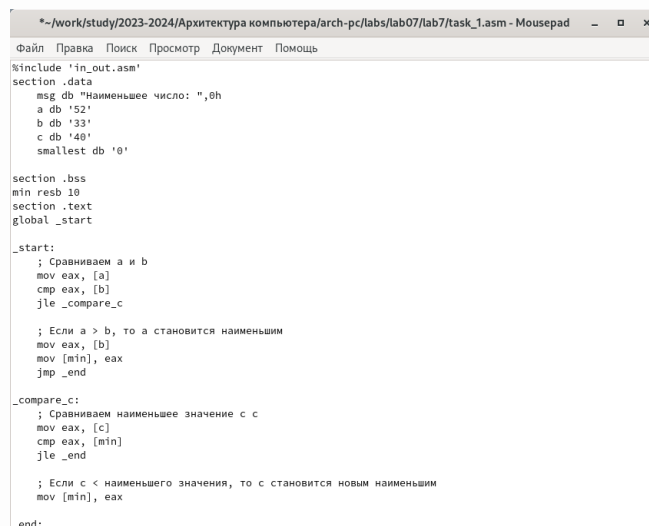


Figure 15: рис.15 Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки:инструкция `mov` (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

4.3 Задания для самостоятельной работы

1. Пишу программу нахождения наименьшей из 3 целочисленных переменных `a`, `b` и `c`. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант под номером 8, поэтому мои значения - 41, 62 и 35. (рис. 16).



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7/task_1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%include 'in_out.asm'
section .data
    msg db "Наименьшее число: ",0h
    a db '52'
    b db '33'
    c db '40'
    smallest db '0'

section .bss
    min resb 10
section .text
    global _start

_start:
    ; Сравниваем a и b
    mov eax, [a]
    cmp eax, [b]
    jle _compare_c

    ; Если a > b, то a становится наименьшим
    mov eax, [b]
    mov [min], eax
    jmp _end

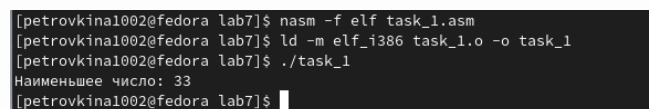
_compare_c:
    ; Сравниваем наименьшее значение с c
    mov eax, [c]
    cmp eax, [min]
    jle _end

    ; Если c < наименьшего значения, то c становится новым наименьшим
    mov [min], eax

_end:
```

Figure 16: рис.16 Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. 17).



```
[petrovkina1002@fedora lab7]$ nasm -f elf task_1.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 task_1.o -o task_1
[petrovkina1002@fedora lab7]$ ./task_1
Наименьшее число: 33
[petrovkina1002@fedora lab7]$
```

Figure 17: рис.17 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm' section .data msg db "Наименьшее число:",0h a
db '52' b db '33' c db '40' smallest db '0'
```

```
section .bss min resb 10 section .text global _start
```

```
_start: ; Сравниваем a и b mov eax, [a] cmp eax, [b] jle _compare_c
```

```
; Если a > b, то a становится наименьшим
```

```
mov eax, [b]
```

```
mov [min], eax
```

```
jmp _end
```

```
_compare_c: ; Сравниваем наименьшее значение с c mov eax, [c] cmp  
eax, [min] jle _end
```

```
; Если c < наименьшего значения, то c становится новым наименьшим
```

```
mov [min], eax
```

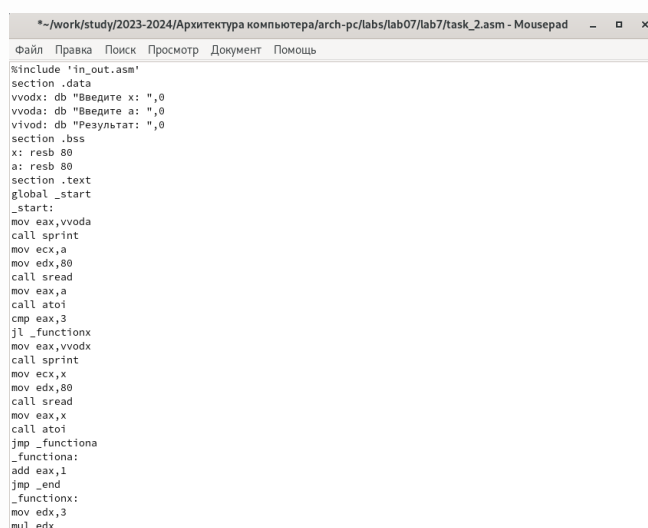
```
_end: ; Выводим наименьшее значение mov eax, msg call sprint ; Вы-  
вод сообщения 'Наименьшее число:' mov edx,[min] call iprintLF ; Вы-  
вод 'min(A,B,C)' call quit
```

1. Пишу программу, которая для введенных с клавиатуры значе-
ний x и a вычисляет значение и выводит результат вычислений
заданной для моего варианта функции $f(x)$:

$3a$, при $a < 3$

$x + 1$, при $a \geq 3$

(рис. 18).



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7/task_2.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
#include "in_out.asm"
section .data
vvdxd: db "Введите x: ",0
vvoda: db "Введите a: ",0
vivod: db "Результат: ",0
section .bss
x: resb 80
a: resb 80
section .text
global _start
_start:
mov eax,vvoda
call sprint
mov ecx,a
mov edx,80
call sread
mov eax,a
call atoi
cmp eax,3
jl _functionx
mov eax,vvdxd
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
jmp _functiona
_functiona:
add eax,1
jmp _end
_functionx:
mov edx,3
mul edx
```

Figure 18: рис.18 Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (1, 4), (2, 6). (рис. 19).


```
[petrovkina1002@fedora lab7]$ nasm -f elf task_2.asm
[petrovkina1002@fedora lab7]$ ld -m elf_i386 task_2.o -o task_2
[petrovkina1002@fedora lab7]$ ./task_2
Введите x: 4
Введите a: 1
Результат: 2
[petrovkina1002@fedora lab7]$ ./task_2
Введите x: 2
Результат: 6
[petrovkina1002@fedora lab7]$
```

Figure 19: рис.19 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm' section .data vvodx: db "Введите x:",0 vvoda: db
"Введите a:",0 vivod: db "Результат:",0 section .bss x: resb 80 a: resb 80
section .text global _start _start: mov eax,vvoda call sprint mov ecx,a
mov edx,80 call sread mov eax,a call atoi cmp eax,3 jl _functionx mov
eax,vvodx call sprint mov ecx,x mov edx,80 call sread mov eax,x call atoi
jmp _functiona _functiona: add eax,1 jmp _end _functionx: mov edx,3
mul edx jmp _end _end: mov ecx,eax mov eax,vivod call sprint mov
eax,ecx call iprintLF call quit
```

5 Выводы

По итогам данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

6 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., Lupin C. A. 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).