

Отчёт по лабораторной работе №7

Петлин Артём Дмитриевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Команды безусловного перехода	7
3.2	Команды условного перехода	8
3.2.1	Регистр флагов	8
3.2.2	Описание инструкции <code>cmpr</code>	8
3.2.3	Описание команд условного перехода.	9
3.3	Файл листинга и его структура	9
4	Выполнение лабораторной работы	11
4.1	Задание для самостоятельной работы	17
5	Выводы	21
	Список литературы	22

Список иллюстраций

Список таблиц

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. Напишите программу нахождения наименьшей из 3 целочисленных переменных a , b и c . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.
2. Напишите программу, которая для введенных с клавиатуры значений x и y вычисляет значение заданной функции $f(x, y)$ и выводит результат вычислений. Вид функции $f(x, y)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений x и y из 7.6.

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.

3.1 Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление:

`jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

В следующем примере рассмотрим использование инструкции `jmp`:

label:

```
... ;  
... ; команды  
... ;
```

```
jmp label
```

3.2 Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

3.2.1 Регистр флагов

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр – регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов.

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV

3.2.2 Описание инструкции `cmp`

Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения.

Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания:

```
cmp <операнд_1>, <операнд_2>
```

Команда `cmp`, так же как и команда вычитания, выполняет вычитание - , но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

3.2.3 Описание команд условного перехода.

Команда условного перехода имеет вид

```
j<мнемоника перехода> label
```

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

В табл. 7.3. представлены команды условного перехода, которые обычно ставятся после команды сравнения `cmp`. В их мнемокодах указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, `ja` и `jnbe`). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных `ax` и `bx` и если произведение превосходит размер байта, передает управление на метку `Error`.

```
mov al, a
mov bl, b
mul bl
jc Error
```

3.3 Файл листинга и его структура

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.

Ниже приведён фрагмент файла листинга.

```

10 00000000 B804000000 mov eax,4
11 00000005 BB01000000 mov ebx,1
12 0000000A B9[00000000] mov ecx,hello
13 0000000F BA0D000000 mov edx,helloLen
14
15 00000014 CD80 int 80h

```

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся.

Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

4 Выполнение лабораторной работы

```
petlin@fedora:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab07
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ls
lab7-1.asm  presentation  report
```

Переходим в каталог для лабораторной работы № 7 и создаём файл lab7-1.asm.

```
lab7-1.asm      [----] 44 L:[ 1+19 20/ 20] *(679 / 679b) <EOF>
%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 sprintLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$
```

Вводим в файл lab7-1.asm текст программы из листинга 7.1. Создаём исполняемый файл и запускаем его.

```

lab7-1.asm      [----] 44 L:[ 1+21 22/ 22] *(706 / 706b) <EOF>
#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 sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$

```

Изменяем текст программы в соответствии с листингом 7.2. Создаём исполняемый файл и запускаем его.

```

lab7-1.asm          [-----] 44 L:[ 1+22 23/ 23] *(721 / 721b) <EOF>
#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

```

```

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$

```

Изменяем текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим:

```

user@dk4n31:~$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
user@dk4n31:~$

```

Проверяем корректность работы.

```

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-2.asm

```

```

lab7-2.asm      [-M--] 57 L:[ 3+29 32/ 50] *(1136/1846b) 0010 0x00A
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
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax,msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-2
Введите B: 4
Наибольшее число: 50
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-2
Введите B: 87
Наибольшее число: 87
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$

```

Создаём файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучаем текст программы из листинга 7.3 и вводим в lab7-2.asm. Создаём исполняемый

файл и проверяем его работу для разных значений В.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ mcedit lab7-2.lst

lab7-2.lst      [----]  0 L:[ 1+ 2 3/225] *(147 /14560b) 0032 0x020
1               %include 'in_out.asm'
2               <1> ;----- slen -----
3               <1> ; Функция вычисления длины сообщения
4               <1> slen:.....
5               00000000 53               <1>     push     ebx.....
6               00000001 89C3            <1>     mov      ebx, eax.....
7               <1>.....
8               <1> nextchar:.....
9               00000003 803800          <1>     cmp      byte [eax], 0...
10              00000006 7403            <1>     jz       finished.....
11              00000008 40              <1>     inc      eax.....
12              00000009 EBF8            <1>     jmp      nextchar.....
13              <1>.....
14              <1> finished:
15              0000000B 29D8            <1>     sub      eax, ebx
16              0000000D 5B              <1>     pop      ebx.....
17              0000000E C3              <1>     ret.....
18              <1>.
19              <1> ;----- sprint -----
20              <1> ; Функция печати сообщения
21              <1> ; входные данные: mov eax,<message>
22              <1> sprint:
23              0000000F 52              <1>     push     edx
24              00000010 51              <1>     push     ecx
25              00000011 53              <1>     push     ebx
26              00000012 50              <1>     push     eax
27              00000013 E8E8FFFFFF      <1>     call     slen
28              <1>.....
29              00000018 89C2            <1>     mov      edx, eax
30              0000001A 58              <1>     pop      eax
31              <1>.....
32              0000001B 89C1            <1>     mov      ecx, eax
33              0000001D BB01000000      <1>     mov      ebx, 1
34              00000022 B804000000      <1>     mov      eax, 4
35              00000027 CD80            <1>     int      80h
36              <1>.
37              00000029 5B              <1>     pop      ebx
38              0000002A 59              <1>     pop      ecx
39              0000002B 5A              <1>     pop      edx
40              0000002C C3              <1>     ret
41              <1>.
42              <1>.
43              <1> ;----- sprintLF -----
44              <1> ; Функция печати сообщения с переводом строки
45              <1> ; входные данные: mov eax,<message>
```

Создаём файл листинга для программы из файла lab7-2.asm и открываем его с помощью текстового редактора.

Пояснение строк:

Строка 33: 0000001D-адрес в сегменте кода, BB01000000-машинный код, mov ebx,1-присвоение переменной ebx значения 1

Строка 34: 00000022-адрес в сегменте кода, B804000000-машинный код, mov eax,4-присвоение переменной eax значения 4

Строка 35 00000027-адрес в сегменте кода, CD80-машинный код, int 80h-вызов ядра.

```
lab7-2.asm [BM--] 10 L:[ 1+17 18/ 50] *(371 /1843b) 0010 0x00A
%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
    call sread
; ----- Преобразование 'B' из символа в число
    mov eax,B
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ mcedit lab7-2.lst
```



```

2          section .data
3          msg1 db 'Введите B: ',0h
3          00000012 00.....
4          00000013 D09DD0B0D0B8D0B1D0-      msg2 db "Наибольшее число: ",0h
4          0000001C BED0BBD18CD188D0B5-
4          00000025 D0B520D187D0B8D181-
4          0000002E D0BBD0BE3A2000.....
5          00000035 32300000          A dd '20'
6          00000039 35300000          C dd '50'
7          section .bss
8          00000000 <res Ah>          max resb 10
9          0000000A <res Ah>          B resb 10
10         section .text
11         global _start
12         _start:
13         ; ----- Вывод сообщения 'Введите B: '
14         000000E8 B8[00000000]      mov eax,msg1
15         000000ED E81DFFFFFF      call sprint
16         ; ----- Ввод 'B'
17         000000F2 B9[0A000000]      mov ecx,B
18         000000F7 E847FFFFFF      mov edx
19         *****
20         error: invalid combination of opcode and operands
21         call sread
22         ; ----- Преобразование 'B' из символа в число
23         000000FC B8[0A000000]      mov eax,B
24         call atoi ; Вызов подпрограммы перевода символа в число
25         00000101 E896FFFFFF      mov [B],eax ; запись преобразованного числа в 'B'
26         00000106 A3[0A000000]      ; ----- Записываем 'A' в переменную 'max'
27         mov ecx,[A] ; 'ecx = A'
28         0000010B 8B0D[35000000]      mov [max],ecx ; 'max = A'
29         0000011D 7F0C          ; ----- Сравниваем 'A' и 'C' (как символы)
30         cmp ecx,[C] ; Сравниваем 'A' и 'C'
31         0000011F 8B0D[39000000]      jg check_B ; если 'A>C', то переход на метку 'check_B',
32         00000125 890D[00000000]      mov ecx,[C] ; иначе 'ecx = C'
33         mov [max],ecx ; 'max = C'
34         ; ----- Преобразование 'max(A,C)' из символа в число

```

Открываем файл с программой lab7-2.asm и в инструкции с двумя операндами удаляем один операнд. Выполняем трансляцию с получением файла листинга. Мы получаем ошибку, однако файлы lab7-2 и lab7-2.lst все равно создаются. Открываем файл листинга и замечаем ошибку в строке, в которой мы удалили один операнд.

4.1 Задание для самостоятельной работы

Вариант №6

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-3.asm
```

```

lab7-3.asm      [-----]  7 L:[ 1+10 11/ 34] *(188 / 493b) 0010 0x00A
#include 'in_out.asm'
section .data
    msg1 db "Наименьшее число: ",0h
    A dd '79'
    B dd '83'
    C dd '41'
global _start
section .bss
    min resb 10
section .text
_start:
    mov ecx,[A]
    mov [min],ecx
    cmp ecx,[C]
    jl label1
    mov ecx,[C]
    mov [min],ecx
label1:
    mov eax,min
    call atoi
    mov [min],eax
    mov ecx,[min]
    cmp ecx,[B]
    jl fin
    mov ecx,[B]
    mov [min],ecx
fin:
    mov eax,msg1
    call sprint
    mov eax,[min]
    call iprintLF
    call quit

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-3
Наименьшее число: 41

```

Создаем файл lab7-3.asm, вписываем в него программу, которая будет искать наименьшее из трех чисел. Смотрим на результат выполнения программы и убеждаемся что все работает верно.

```

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$

```

```

lab7-4.asm      [-M--]  3 L:[ 1+46  47/ 47] *(708 / 708b) <EOF>
#include 'in_out.asm'
section .data
    msg1 db "Введите x: ",0h
    msg2 db "Введите a: ",0h
    msg3 db "Результат: ",0h
global _start
section .bss
    res resb 10
    X resb 10
    A resb 10
section .text
_start:
    mov eax,msg1
    call sprint
    mov ecx,X
    mov edx,10
    call sread
    mov eax,X
    call atoi
    mov [X],eax
    mov eax,msg2
    call sprint
    mov ecx,A
    mov edx,10
    call sread
    mov eax,A
    call atoi
    mov [A],eax ;
    cmp eax,[X]
    je labell
    mov ebx,5
    mov eax,[X]
    mul ebx
    mov [res],eax
    jmp fin
labell:
    mov ecx,[X]
    add eax,ecx
    mov [res],eax
    jmp fin
fin:
    mov eax,msg3
    call sprint
    mov eax,[res]
    call iprintLF
    call quit
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-4
Введите x: 2
Введите a: 2
Результат: 4
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-4
Введите x: 2
Введите a: 1
Результат: 10
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07$

```

Создаем файл lab7-4.asm, вписываем в него программу, которая будет выполнять действия описанные в данной нам функции. Смотрим на результат выполнения программы и убеждаемся что все работает верно.

5 Выводы

Мы изучили команды условного и безусловного переходов. Мы приобрели навыки написания программ с использованием переходов. Мы познакомились с назначением и структурой файла листинга.

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).