

Отчет по лабораторной работе №6

Шубина София Антоновна

Содержание

1	Цель работы	5
2	Вывод	18
	Список литературы	19

Список иллюстраций

1.1	Создадим каталог и файл	5
1.2	Создаем файл и запускаем его	6
1.3	Изменяем текст программы	7
1.4	Запуск исполняемого файла	7
1.5	Ввод текста программы	8
1.6	Создание и запуск исполняемого файла	8
1.7	Создание и запуск исполняемого файла	9
1.8	Создание и запись исполняемого файла с функцией iprint	9
1.9	Изменение текста программы	11
1.10	Создание и проверка работы исполняемого файла	11
1.11	Создание файла в каталоге	12
1.12	Ввод текста программы	13
1.13	Создание и запуск исполняемого файла	13
1.14	Создание и запуск исполняемого файла	15
1.15	Ввод прирограммы	15

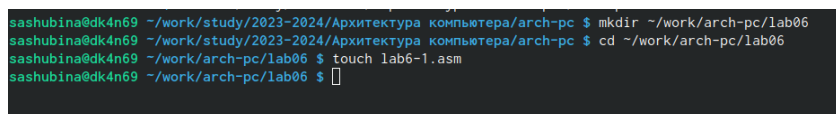
Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

#Порядок выполнения лабораторной работы

1. Создадим каталог для программ лабораторной работы No 6, перейдем в него и создадим файл lab6-1.asm: `mkdir ~/work/arch-pc/lab06 cd ~/work/arch-pc/lab06 touch lab6-1.asm` (рис 1.1)



```
sashubina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ mkdir ~/work/arch-pc/lab06
sashubina@dk4n69 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ cd ~/work/arch-pc/lab06
sashubina@dk4n69 ~/work/arch-pc/lab06 $ touch lab6-1.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $
```

Рис. 1.1: Создадим каталог и файл

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр `eax`. Введем в файл `lab6-1.asm` текст программы из листинга 6.1. В данной программе в регистр `eax` записывается символ `б` (`mov eax,'б'`), 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 `sprintf` 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 `sprintf`.

Листинг 6.1. Программа вывода значения регистра еах %include 'in_out.asm'
SECTION .bss buf1: RESB 80 SECTION .text GLOBAL _start _start: mov eax,'6' mov
ebx,'4' add eax,ebx mov [buf1],eax mov eax,buf1 call sprintLF call quit (рис ??)

```
lab6-1.asm [----] 9 L: [ 1+12 13/ 13] *(172 / 172b) <EOF>
#include "in_out.asm"
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, 6
mov ebx, 4
add eax, ebx
mov [buf1], eax
mov eax, buf1
call sprintLF
call quit
```

Создаем исполняемый файл и запускаем его. nasm -f elf lab6-1.asm ld -m elf_i386
-o lab6-1 lab6-1.o ./lab6-1 (рис 1.2)

```
sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-1
j
sashubina@dk4n69 ~/work/arch-pc/lab06 $
```

Рис. 1.2: Создаем файл и запускаем его

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. В данном случае при выводе значения регистра еах 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 еах сумму кодов – 01101010 (106), что в свою очередь является кодом символа j (см. таблицу ASCII в приложении). 3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы (Листинг 6.1) следующим образом: заменим строки
mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4 (рис 1.3)

```
lab6-2.asm [----] 11 L: [ 1+ 7 8/ 9] *(101 / 111b) 0010 0x00A
#include "in_out.asm"
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

Рис. 1.3: Изменяем текст программы

Создадим исполняемый файл и запустим его. (рис 1.4)

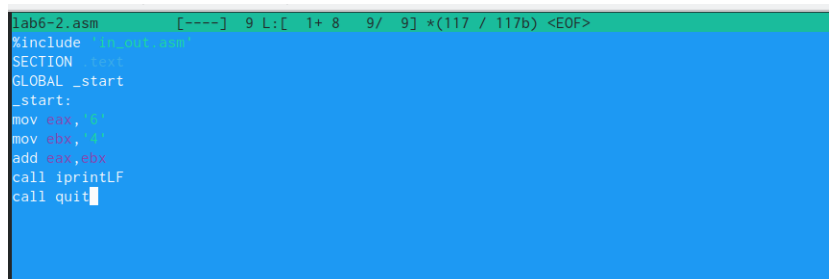
```
sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-1

sashubina@dk4n69 ~/work/arch-pc/lab06 $
```

Рис. 1.4: Запуск исполняемого файла

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определим какому символу соответствует код 10: LF, символ переноса строки. Отображается ли этот символ при выводе на экран? -нет, не отображается. Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпро-

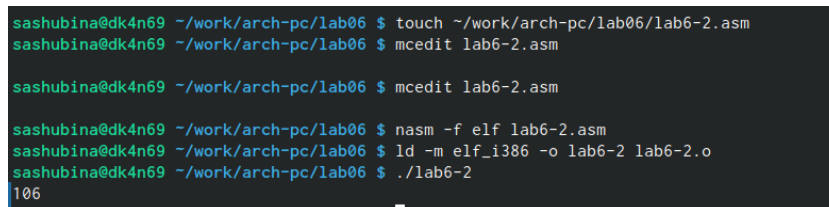
граммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 6.1 с использованием этих функций. Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введем в него текст программы из листинга 6.2. touch ~/work/arch-pc/lab06/lab6-2.asm Листинг 6.2. Программа вывода значения регистра eax %include 'in_out.asm' SECTION .text GLOBAL _start _start: mov eax,'6' mov ebx,'4' add eax,ebx call iprintLF call quit (рис 1.5)



```
lab6-2.asm [----] 9 L: [ 1+ 8 9/ 9] *(117 / 117b) <EOF>
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Рис. 1.5: Ввод текста программы

Создадим исполняемый файл и запустим его. nasm -f elf lab6-2.asm ld -m elf_i386 -o lab6-2 lab6-2.o ./lab6-2 (рис 1.6)



```
sashubina@dk4n69 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ mcedit lab6-2.asm

sashubina@dk4n69 ~/work/arch-pc/lab06 $ mcedit lab6-2.asm

sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-2
106
```

Рис. 1.6: Создание и запуск исполняемого 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 отличие от программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число. 5. Аналогично предыдущему примеру изменим символы на числа. Заменяем строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4 Создадим исполняемый файл и запустим его. (рис 1.7)


```

sashubina@dk4n69 ~/work/arch-pc/lab06 $ mcedit lab6-2.asm

sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-2
10

```

Рис. 1.7: Создание и запуск исполняемого файла

Какой результат будет получен при исполнении программы? Результат-10, при чем результат вывелся с новой строки. Заменяем функцию `iprintLF` на `iprint`. Создадим исполняемый файл и запустим его. Чем отличается вывод функций `iprintLF` и `iprint`? `iprintLF` - это вывод результата с новой строки `iprint` - результат выводится в прошлую строку (рис 1.8)

```

sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-2
10sashubina@dk4n69 ~/work/arch-pc/lab06 $

```

Рис. 1.8: Создание и запись исполняемого файла с функцией `iprint`

- В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $\boxed{x}(\boxed{x}) = (5 \boxed{x} 2 + 3)/3$. Создадим файл `lab6-3.asm` в каталоге `~/work/arch-pc/lab06`: `touch ~/work/arch-pc/lab06/lab6-3.asm`. Внимательно изучим текст программы из листинга 6.3 и введем в `lab6-3.asm`. Листинг 6.3. Программа вычисления выражения $\boxed{x}(\boxed{x}) = (5 \boxed{x} 2 + 3)/3$; ————— ; Программа вычисления выражения ; ————— `%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

Создадим исполняемый файл и запустим его. Результат работы программы
должен быть следующим:

user@dk4n31:~

./lab6-3

Результат: 4

Остаток от деления: 1

user@dk4n31:~\$ (рис @fig:010)

```
sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
sashubina@dk4n69 ~/work/arch-pc/lab06 $
```

Изменим текст про-

граммы для вычисления выражения $\text{X}(\text{X}) = (4 \times 6 + 2)/5$. Создадим исполняемый
файл и проверьте его работу. (рис 1.9)

```

lab6-3.asm          [-M--]  9 L: [ 1+16  17/ 29] *(552 /1365b) 0032 0x020
;-----
; Программа вычисления выражения
;-----
%include "lab6-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,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; 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. 1.9: Изменение текста программы

(рис 1.10)

```

sashubina@dk4n69 ~/work/arch-pc/lab06 $ mcedit lab6-3.asm

sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
sashubina@dk4n69 ~/work/arch-pc/lab06 $

```

Рис. 1.10: Создание и проверка работы исполняемого файла

- В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:
 - вывести запрос на введение No студенческого билета
 - вычислить номер варианта по формуле: $(\text{No} \bmod 20) + 1$, где No – номер студен-

ческого билета (В данном случае $\text{X} \bmod \text{X}$ – это остаток от деления X на X).

- 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 символ-ном 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`. Создадим файл `variant.asm` в каталоге `~/work/arch-pc/lab06`: `touch ~/work/arch-pc/lab06/variant.asm` (рис 1.11)

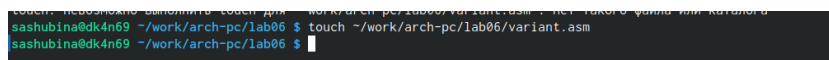


Рис. 1.11: Создание файла в каталоге

Внимательно изучим текст программы из листинга 6.4 и введем в файл `variant.asm`. Листинг 6.4. Программа вычисления варианта задания по номеру студенческого билета ;————— ; Программа вычисления варианта ;————— %include 'in_out.asm' SECTION .data msg: DB 'Введите No студенческого билета:',0 rem: DB 'Ваш вариант:',0 SECTION .bss x: RESB 80 SECTION .text GLOBAL _start _start: mov eax, msg call sprintLF mov ecx, x mov edx, 80 call sread mov eax,x ; вызов подпрограммы преобразования call atoi ; ASCII кода в число, eax=x xor edx,edx mov ebx,20 div ebx inc edx mov eax,rem call sprint mov eax,edx call iprintLF call quit (рис 1.12)

```

variant.asm      [----]  9 L:[ 1+27 28/ 28] *(617 / 617b) <EOF>
; Программа вычисления варианта
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите No студенческого билета: ',0
rem: DB 'Ваш вариант: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF
mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, 'eax=x'
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprint
mov eax, edx
call iprintLF
call quit

```

Рис. 1.12: Ввод текста программы

Создадим исполняемый файл и запустим его. Проверим результат работы программы вычислив номер варианта аналитически. (рис 1.13)

```

sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./variant
Введите No студенческого билета:
1132232885
Ваш вариант: 6
sashubina@dk4n69 ~/work/arch-pc/lab06 $

```

Рис. 1.13: Создание и запуск исполняемого файла

Включим в отчет по выполнению лабораторной работы ответы на следующие вопросы: 1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения 'Ваш вариант:?' rem: DB 'Ваш вариант:' в строке объявляется переменная, куда и записывается искомая строка mov eax, rem - помещаем строку в регистр eax call sprint - вызываем подпрограмму вывода из файла in_out.asm 2. Для чего ис-

пользуется следующие инструкции? `mov ecx, x`, `mov edx, 80`, `call sread`. Инструкции используются для того, чтобы ввести с клавиатуры строку отведённого размера (80) и поместить её по адресу `x`. Для этого `x` помещаем в регистр `ecx`, а длину строки (80) - в регистр `edx`. `call sread` - вызывает функции печати.

3. Для чего используется инструкция “`call atoi`”? Инструкция `call atoi` используется для преобразования символов в числа.

4. Какие строки листинга 6.4 отвечают за вычисления варианта? `mov eax, x` - поместили `x` в регистр `eax`, `call atoi` - преобразование символов в число, `xor edx, edx` - обнуляем `edx`, `mov ebx, 20` - поместили в регистр `ebx` число 20, `div ebx` - поделили число, лежащее в `eax`, на число, лежащее в `ebx`, `inc edx`; `edx + 1`.

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

6. Для чего используется инструкция “`inc edx`”? Инструкция `inc edx` используется для увеличения значения регистра `edx` на 1.

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений? `mov eax, edx`; помещаем результат вычислений в регистр `eax`, `call iprintLF`; выводим на экран содержимое регистра `eax`.

#Задания для самостоятельной работы

Написать программу вычисления выражения $x = x(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x , вычислять заданное выражение в зависимости от введенного x , выводить результат вычислений. Вид функции $x(x)$ выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x_1 и x_2 из 6.3. Мой вариант - №6 6: $x^3/2 + 1$, $x_1=2$, $x_2=5$. При выполнении задания преобразовывать (упрощать) выражения для $x(x)$ нельзя. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е. $5 : 2 = 2$) (рис 1.14)

```
sashubina@dk4n69 ~/work/arch-pc/lab06 $ nasm -f elf func.asm
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o func func.o
sashubina@dk4n69 ~/work/arch-pc/lab06 $ ./func
Введите x:
2
f(x)= 5
```

Рис. 1.14: Создание и запуск исполняемого файла

(рис 1.15)

```
; ~~~~~~
; вариант 6
; ~~~~~~
#include "in_out.asm"
SECTION .data
msg: DB "Введите x: ",0
rem: DB "f(x)= ",0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintf
mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, 'eax*x'
xor edx, edx

mov ebx, eax
mul ebx
mul ebx
mov ecx, 2
div ecx
inc eax
mov edx, eax

mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
```

Рис. 1.15: Ввод припрограммы

```
; ~~~~~~
; вариант 6
; ~~~~~~
```

```

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
rem: DB 'f(x)= ',0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start

_start:
mov eax, msg
call sprintf
mov ecx, x
mov edx, 80
call read

mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx,edx

mov ebx,eax
mul ebx
mul ebx
mov ecx,2
div ecx
inc eax
mov edx,eax

mov eax,rem
call sprintf
mov eax,edx

```



```
call iprintLF
```

```
call quit
```

2 Вывод

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/Learning-bash-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,
- 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)