РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

дисциплина: Архитектура компьютера

Студент: Нурмухаметова Каролина Марселевна

Группа: НКАбд-05-25

МОСКВА

2025 г.

Оглавление

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм.
Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

2.1. Реализация подпрограмм в NASM

Создадим каталог для выполнения лабораторной работы № 9, перейдём в него и создадим файл lab09-1.asm:

```
kmnurmukhametov@dk6n01 ~ $ mkdir ~/work/arch-pc/lab09
kmnurmukhametov@dk6n01 ~ $ cd ~/work/arch-pc/lab09
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ touch lab09-1.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $
```

Рис. 2.1.1 — Создание каталога

В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы _calcul.
В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Создадим файл lab09-1.asm с текстом листинга 9.1, оттранслируем и запустим:

```
1  %include 'in_out.asm'
2  SECTION .data
3     msg: DB 'Введите x: ',0
4     result: DB '2x+7=',0
5
6  SECTION .bss
7     x: RESB 80
8     res: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12    _start:
13
14 ;----------------------------------------
15 ; Основная программа
16 ;----------------------------------------
17
18    mov eax, msg
19    call sprint
20
21    mov ecx, x
22    mov edx, 80
23    call sread
24
25    mov eax,x
26    call atoi
27
28    call _calcul ; Вызов подпрограммы _calcul
29
30    mov eax,result
31    call sprint
32    mov eax,[res]
33    call iprintLF
34
35    call quit
36
37 ;----------------------------------------
38 ; Подпрограмма вычисления
39 ; выражения "2x+7"
40
41    _calcul:
42        mov ebx,2
43        mul ebx
44        add eax,7
45        mov [res],eax
46
47        ret ; выход из подпрограммы
```

Рис. 2.1.2 — Текст программы lab09-1.asm

```
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=13
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $
```

Рис. 2.1.3 — Вывод программы lab09-1.asm

Изменим текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x+ 7, g(x) = 3x − 1. Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul. Для этого создадим копию файла с именем lab09-1-1.asm:

```
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB 'f(g(x))=',0

SECTION .bss
    x: RESB 80
    res: RESB 80

SECTION .text
GLOBAL _start

_start:
    ;-------------------------------------------
    ; Основная программа
    ;-------------------------------------------
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul       ; Вызов подпрограммы _calcul
    mov [res], eax
    mov eax, result
    call sprint
    mov eax,[res]
    call iprintLF
    call quit

    ;-------------------------------------------
    ; Подпрограмма вычисления f(g(x))
    ;-------------------------------------------
    _calcul:
        push eax
        call _subcalcul    ; Вызов подпрограммы _subcalcul для g(x)
        mov ebx, eax
        mov eax, 2
        mul ebx
        add eax, 7
        pop ebx
        ret
    ;-------------------------------------------
    ; Вложенная подпрограмма вычисления g(x)
    ; Вход: EAX содержит x
    ; Выход: EAX содержит g(x)
    ;-------------------------------------------
    _subcalcul:
        mov ebx, 3
        mul ebx
        sub eax, 1
        ret

```

Рис. 2.1.4 — Текст программы lab09-1-1.asm

```
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ touch lab09-1-1.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1-1.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1-1 lab09-1-1.o
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ./lab09-1-1
Введите x: 3
f(g(x))=23
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $
```

Рис. 2.1.5 — Вывод программы lab09-1-1.asm

2.2. Отладка программ с помощью GDB

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2.:

```
1  SECTION .data
2      msg1: db "Hello, ",0x0
3      msg1Len: equ $ - msg1
4
5      msg2: db "world!",0xa
6      msg2Len: equ $ - msg2
7
8  SECTION .text
9      global _start
10
11 _start:
12     mov eax, 4
13     mov ebx, 1
14     mov ecx, msg1
15     mov edx, msg1Len
16     int 0x80
17
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg2
21     mov edx, msg2Len
22     int 0x80
23
24     mov eax, 1
25     mov ebx, 0
26     int 0x80
```

Рис. 2.2.1 — Текст программы lab09-2.asm

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

```
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ touch lab09-2.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 2.2.2 — Трансляция программы lab09-2.asm

Загрузим исполняемый файл в отладчик gdb:

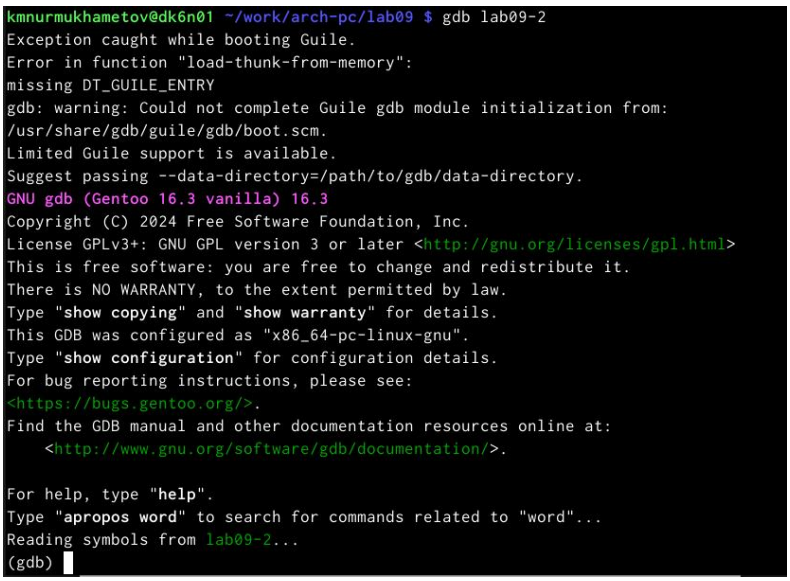

Рис. 2.2.3 — Загрузка файла в отладчик gdb

Проверим работу программы, запустив её в оболочке GDB с помощью команды run.

Рис. 2.2.4 — Результат программы

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её.

Рис. 2.2.5 — Запуск программы

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start.

Рис. 2.2.6 — Выполнение команды disassemble

Переключим на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel.

```
(gdb) set disassembly-flavor intel
```

Рис. 2.2.7 — Использование команды

Перечислим различия отображения синтаксиса машинных команд в режимах ATT и Intel: они заключаются в порядке операндов и оформлении команд. В синтаксисе Intel используется порядок "назначение-источник", тогда как в AT&T применяется обратный порядок "источник-назначение". Кроме того, в AT&T-синтаксисе регистры предваряются символом %, а непосредственные значения - символом $.
Включим режим псевдографики для более удобного анализа программы с помощью layout asm (активирует режим разделенного экрана с отображением ассемблерного кода) и layout regs.

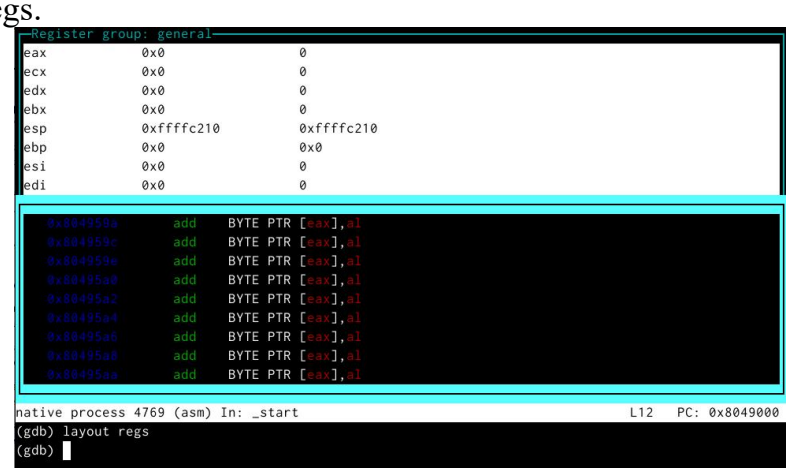

Рис. 2.2.8 — Использование команд

2.3. Добавление точек останова

Установить точку останова можно командой break (кратко b). С помощью команды info breakpoints (кратко i b) можно проверить установленные точки останова:

```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:12
        breakpoint already hit 1 time
(gdb)
```

Рис. 2.3.1 — Использование команды

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0), установим точку останова и посмотрим информацию о всех установленных точках останова:

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 25.
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:12
        breakpoint already hit 1 time
2       breakpoint     keep y   0x08049031 lab09-2.asm:25
(gdb)
```

Рис. 2.3.2 — Установка точки остановы и просмотр информации о всех таких точках

2.4. Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотрим содержимое регистров с помощью команды info registers.

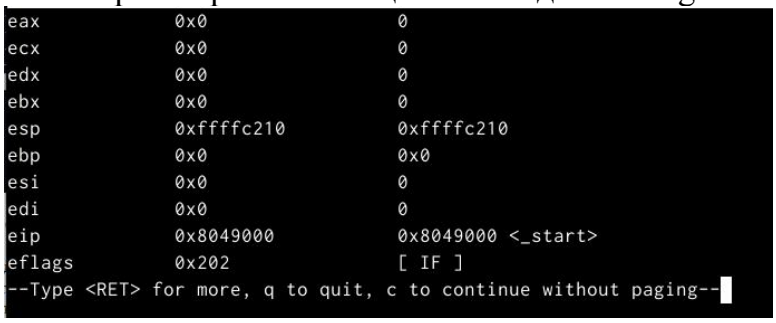

Рис. 2.4.1 — Использование программы i r

Для отображения содержимого памяти можно использовать команду x, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU. С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени и значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции.

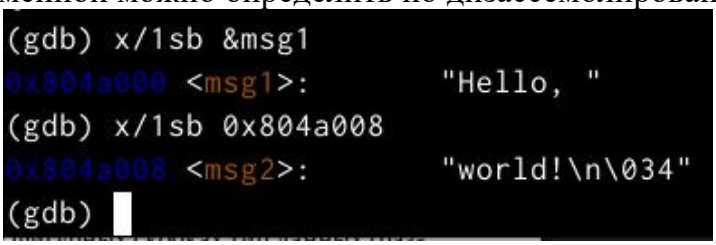

Рис. 2.4.2 — Использование команд

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных. Изменим 1-й символ переменной msg1 и 1-й символ переменной msg2:

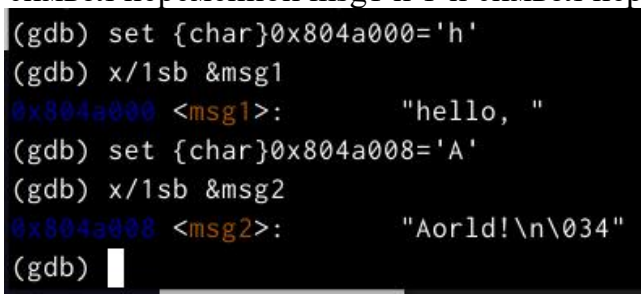

Рис. 2.4.3 — Изменение символов

С помощью команды set изменим значение регистра ebx :

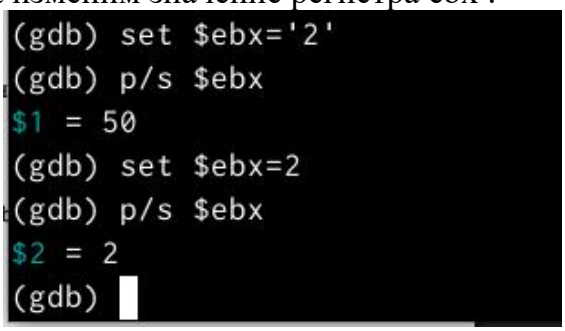

Рис. 2.4.4 — Изменение регистра

Кавычки ' ' заставляют GDB интерпретировать значение как символ и хранить его ASCII-код, тогда как без кавычек значение трактуется как чистое число.

2.5. Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm и создадим исполняемый файл:

```
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $
```

Рис. 2.5.1 — Копирование и создание файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args. Загрузим исполняемый файл в отладчик, указав аргументы

```
kmnurmukhametov@dk6n01 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
Exception caught while booting Guile.
Error in function "load-thunk-from-memory":
missing DT_GUILE_ENTRY
gdb: warning: Could not complete Guile gdb module initialization from:
/usr/share/gdb/guile/gdb/boot.scm.
Limited Guile support is available.
Suggest passing --data-directory=/path/to/gdb/data-directory.
GNU gdb (Gentoo 16.3 vanilla) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)
```

Рис. 2.5.2 — Выполнение загрузки файла

При запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим её.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 11.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmnurmukhametov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:11
11          pop ecx    ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.5.3 — Установка точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находиться имя программы, по адресу [esp+8] хранить адрес первого аргумента, по адресу [esp+12] – второго и т.д.

```
(gdb) x/x $esp
0xffffc1d0:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffc443:     "/afs/.dk.sci.pfu.edu.ru/home/k/m/kmnurmukhametov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffc48f:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffc4a1:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffc4b2:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffc4b4:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.5.4 — Просмотр позиций стека

Объясним, почему шаг изменения адреса равен 4: Шаг изменения адреса равен 4 байтам, потому что в данной архитектуре используется 32-битная система с размером машинного слова 4 байта.

3 Задание для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму, для этого создадим и оттранслируем файл lab09-sam.asm:

```
%include "in_out.asm"

SECTION .data
    msg db "Результат для функции f(x)=10(x-1): ",0

SECTION .text
    global _start

; Подпрограмма для вычисления f(x) = 10*(x-1)
; Вход: eax = x
; Выход: eax = f(x)
f:
    sub eax, 1    ; Вычислить x-1
    mov ebx, 10   ; Загрузить константу 10 в ebx
    imul eax, ebx ; Умножить (x-1) на 10 (результат в eax)
    ret           ; Вернуться из подпрограммы

_start:
    pop ecx       ; ecx = argc (количество аргументов)
    pop edx       ; edx = argv[0] (имя программы)
    sub ecx, 1    ; Пропустить имя программы в argv
    mov esi, 0    ; esi будет накапливать общую сумму

next:
    cmp ecx, 0h   ; Проверить, все ли аргументы обработаны
    jz _end       ; Если да, перейти к завершению
    pop eax       ; Получить следующий аргумент (строку) из стека
    call atoi     ; Преобразовать строку в целое число x (результат в eax)

    ; Вызов подпрограммы f
    call f

    add esi, eax

    loop next     ; Перейти к следующей итерации цикла

_end:
    ; Вывод результата
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintLF
    call quit

```

Рис. 3.1.1 — Текст самостоятельного задания

```
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-sam lab09-sam.o
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-sam 1 2 3 4
Результат для функции f(x)=10(x-1): 60
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 3.1.2 — Результат выполнения самостоятельного задания

2. В листинге 9.3 приведена программа вычисления выражения (3 + 2) * 4 + 5. При запуске данная программа даёт неверный результат. Создадим файл lab09-4.asm, впишем в него текст листинга 9.3

```
%include 'in_out.asm'

SECTION .data
    div: DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:
    ; Вычисление выражения (3 + 2) * 4 + 5
    mov eax, 3
    mov ebx, 2
    add eax, ebx
    mov ebx, 4
    mul ebx

    mov ebx, 5
    add eax, ebx

    mov edi, eax
    call sprint
    call iprintLF
    call quit

```

Рис. 3.2.1 — Текст листинга

Исправим программу.

```
%include 'in_out.asm'

SECTION .data
    div: DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:
    ; Вычисление выражения (3 + 2) * 4 + 5
    mov ebx, 3
    mov eax, 2
    add ebx,eax      ; ebx = 3 + 2 = 5
    mov eax,ebx
    mov ecx, 4
    mul ecx           ; eax = 5 * 4 = 20
    mov ebx,eax
    add ebx,5    ; ebx = 20 + 5 = 25
    mov edi,ebx
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
    call quit

```

Рис. 3.2.2 — Текст исправленной программы

```
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-4
Результат: 25
kmnurmukhametov@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 3.2.3 — Запуск исправленной программы

4 Выводы

Приобрела навыки написания программ с использованием подпрограмм. Ознакомилась с методами отладки при помощи GDB и его основными возможностями.

5 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, 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. — БХВ-Петербург, 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).