Шаблон отчёта по лабораторной работе номер 9

Архитектура компьютеровт

Волчкова Елизавета Дмитриевна

Содержание

# Цель работы

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

# Задание

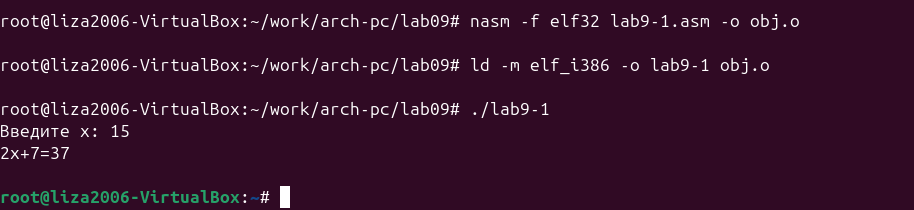
1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции 𝑓(𝑥) как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее

# Теоретическое введение

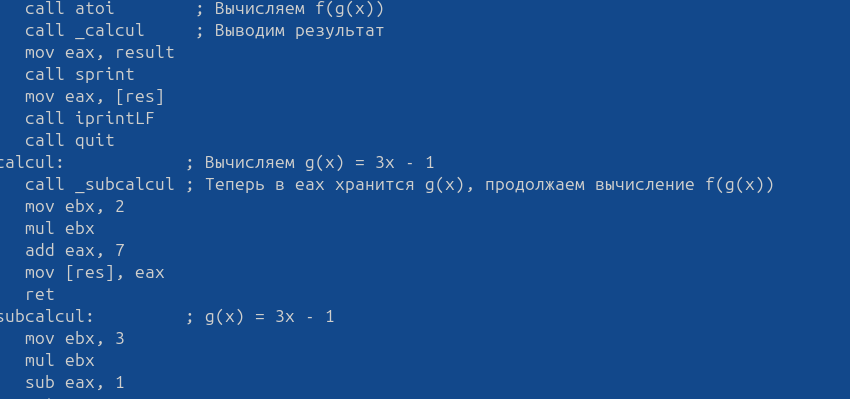
Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают пре- рывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Более подробно про Unix см. в [@tanenbaum\_book\_modern-os\_ru; @robbins\_book\_bash\_en; @zarrelli\_book\_mastering-bash\_en; @newham\_book\_learning-bash\_en].

# Выполнение лабораторной работы

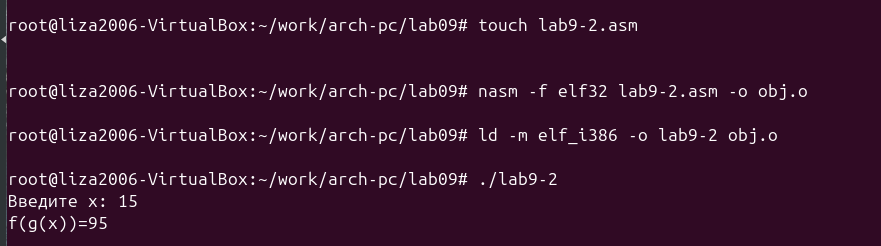
1. Создала каталог для выполнения лабораторной работы № 9, перешла в него и со- здала файл lab09-1.asm
2. В качестве примера рассмотрела программу вычисления арифметического выражения 𝑓(𝑥) = 2𝑥 + 7 с помощью подпрограммы \_calcul. В данном примере 𝑥 вводилось с клавиатуры, а само выражение вычисляется в подпрограмме. Затем внимательно изучила текст программы (Листинг 9.1). Листинг 9.1. П 1



Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).] После следующей инструкции call \_calcul, которая передает управление подпрограмме \_calcul, выполнила инструкции подпрограммы:



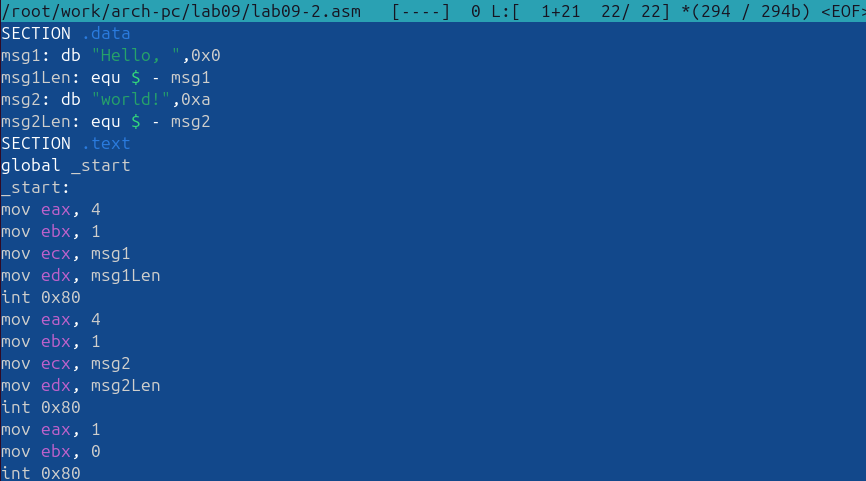
Инструкция ret является последней в подпрограмме и ее исполнение приводит к воз- вращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму.

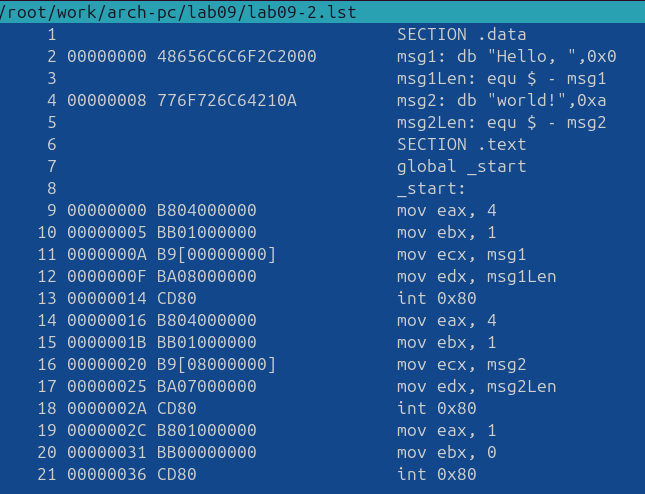


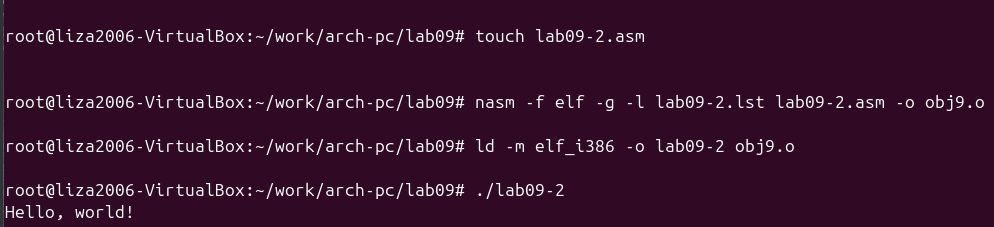
Последние строки программы реализуют вывод сообщения (call sprint), результата вы- числения (call iprintLF) и завершение программы (call quit).

Ввела в файл lab09-1.asm текст программы из листинга 9.1. Создала исполняемый файл и проверила его работу.

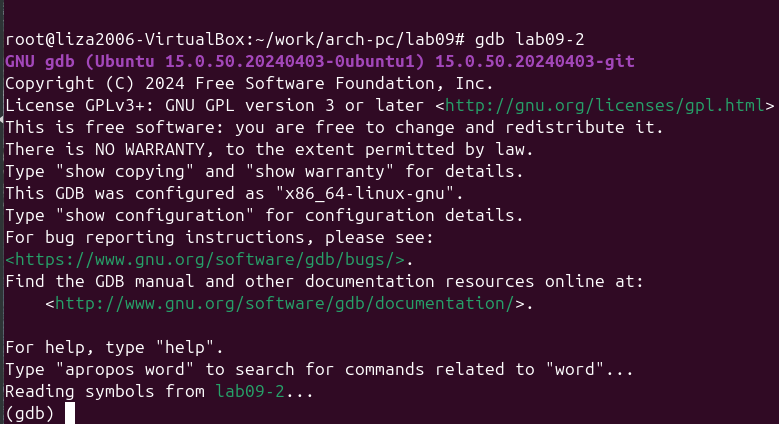
Потом изменила текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения 𝑓(𝑔(𝑥)), где 𝑥 вводится с клавиатуры, 𝑓(𝑥) = 2𝑥 + 7, 𝑔(𝑥) = 3𝑥 − 1. Т.е. 𝑥 передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение 𝑔(𝑥), результат возвращается в \_calcul и вычисляется выражение 𝑓(𝑔(𝑥)). Результат возвращается в основную программу для вывода результата на экран. 9.4.2. Отладка программам с помощью GDB Сначала создала файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):

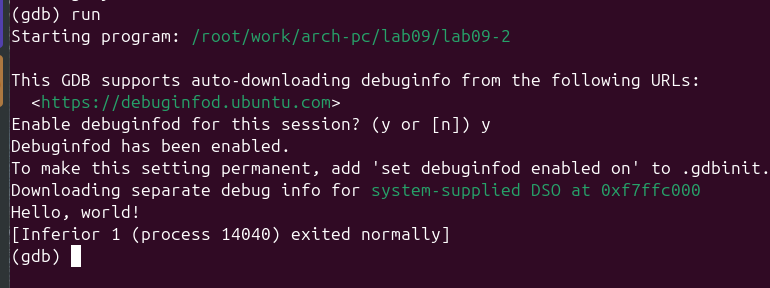






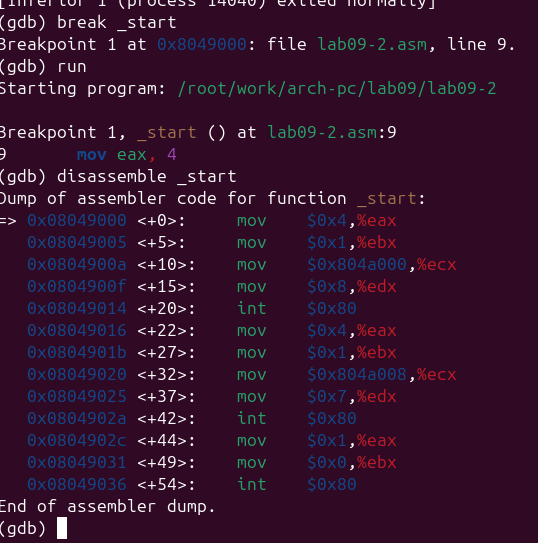
Получила исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавила отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.





Загрузила исполняемый файл в отладчик gdb: user@dk4n31:~$ gdb lab09-2 Проверила работу программы, запустив ее в оболочке GDB с помощью команды run (со- кращённо r): (

Hello, world! [Inferior 1 (process 10220) exited normally] (gdb) Для более подробного анализа программы установила брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустила её. (gdb) break \_start Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) run Starting program: ~/work/arch-pc/lab09/lab09-2 Breakpoint 1, \_start () at lab09-2.asm:12 12 mov eax, 4



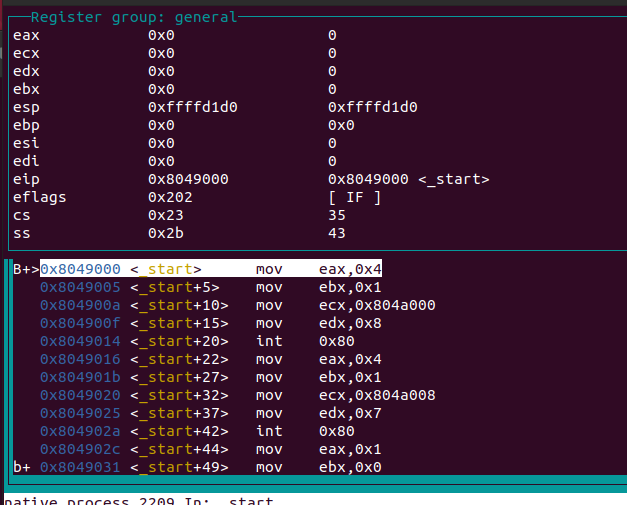
Посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (gdb) disassemble \_start Переключилась на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (gdb) set disassembly-flavor intel (gdb) disassemble \_start Перечислила различия отображения синтаксиса машинных команд в режимах ATT и Intel. Далее включила режим псевдографики для более удобного анализа программы (рис. 9.2): (gdb) layout asm (gdb) layout regs

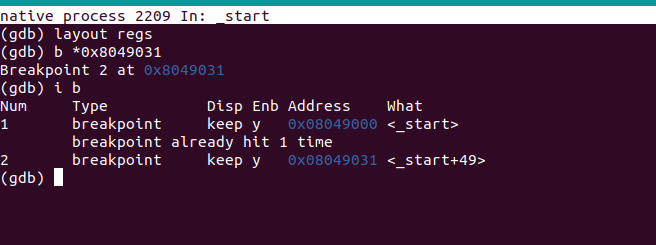
В этом режиме есть три окна:

• В верхней части видны названия регистров и их текущие значения; • В средней части виден результат дисассимилирования программы; • Нижняя часть доступна для ввода команд.

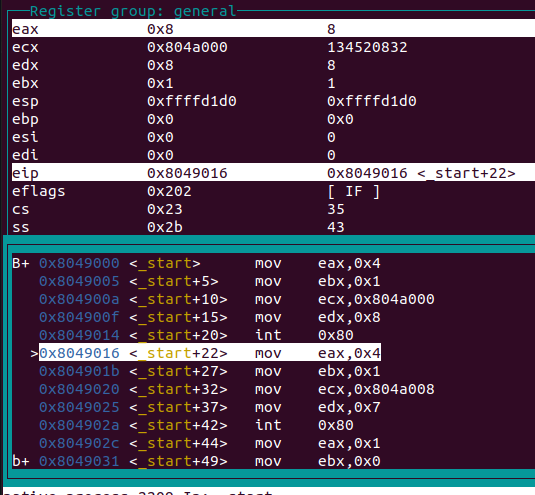
9.4.2.1. Добавление точек останова Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»:

На предыдущих шагах была установлена точка останова по имениметки (\_start). Про- верьте это с помощью команды info breakpoints (кратко i b):



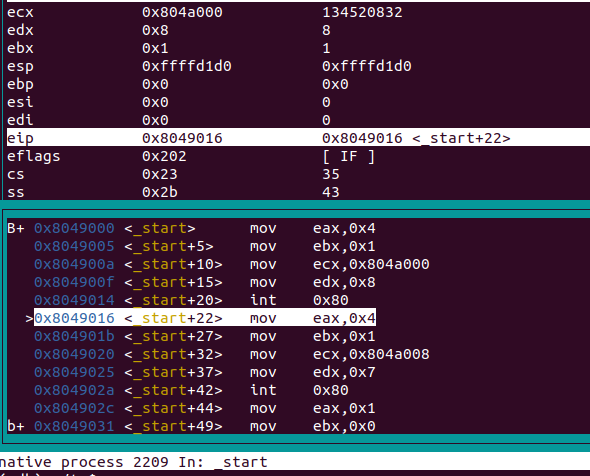


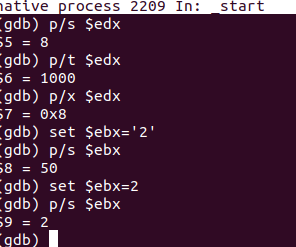
Посмотрела значение переменной msg1 по имени и посмотрела значение переменной msg2 по адресу.



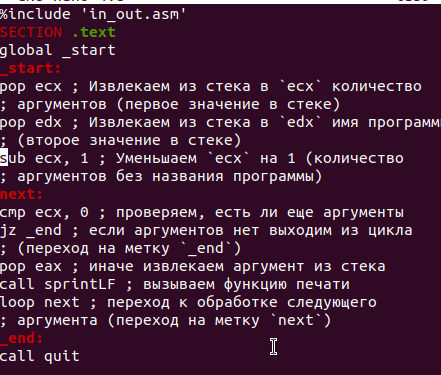


Вывел в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.





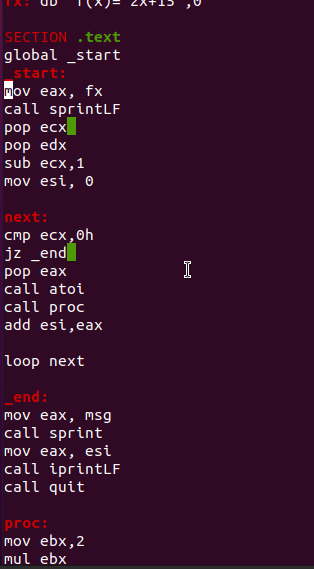
Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки. Создал исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузил исполняемый файл в отладчик, указав аргументы.

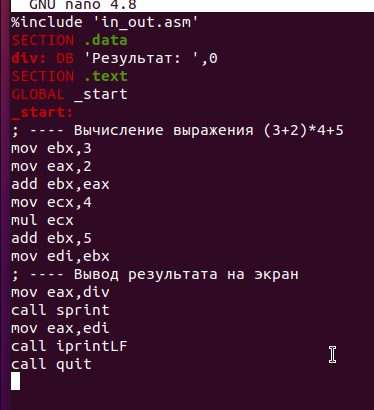
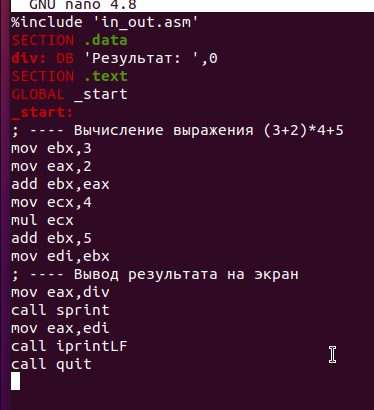
Затем для начала установила точку останова перед первой инструкцией в программе, запустила ее.

При условии, что дрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Число аргументов равно 5 – это имя программы lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

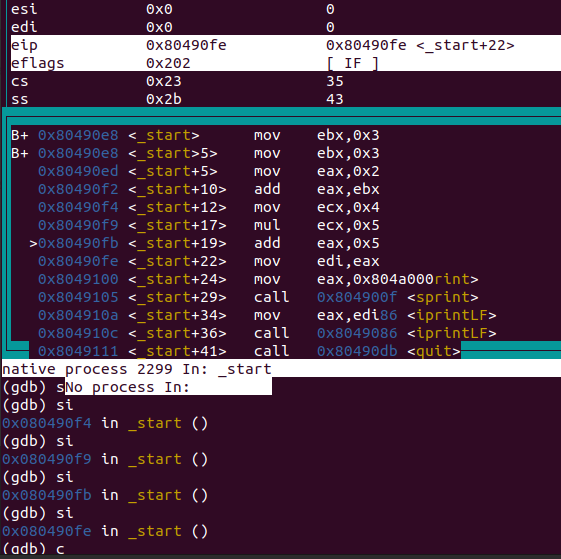
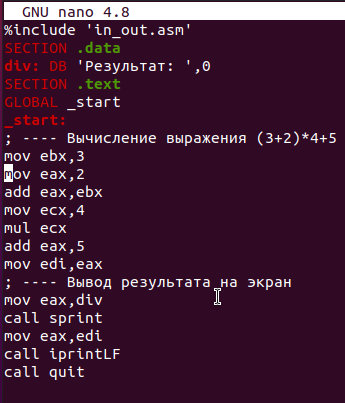
Посмотрела остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.



Приведенный ниже листинг программы вычисляет выражение ( 3+2 ) ∗ 4+5 .

Программа дает неверный результат., проверив это я использовала отладчик GDB для анализа изменений значений регистров и определения ошибки. 

Узнала, что орядок аргументов в инструкции add был перепутан и что при завершении работы, вместо eax, значение отправлялось в edi. Поэтому вот исправленнный код в файле task-2.asm



Установила точку останова перед первой инструкцией в программе и запустила ее.

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрела остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по

# Выводы

Целью работы было - приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями, сделав данные задания, я познакомилась с написанием программ при помощи GDB.

# 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. 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,
17. — 1120 с. — (Классика Computer Science) ::: {#refs} :::