**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6.**

дисциплина: *Архитектура компьютеров*

Студент: Подхалюзина Виолетта Михайловна

Группа: НКАбд-04-24

**МОСКВА**

2024 г.

Оглавление

[1 Цель работы 3](#_Toc182660321)

[2 Введение 3](#_Toc182660322)

[3 Выполнение лабораторной работы 5](#_Toc182660323)

[3.1 Начало работы 5](#_Toc182660324)

[3.2 Самостоятельная работа 11](#_Toc182660325)

[4 Контрольные вопросы для самопроверки 13](#_Toc182660326)

[5 Список литературы 14](#_Toc182660327)

# Цель работы

Целью данной лабораторной работы является приобретение jсвоение арифметических инструкций языка ассемблера NASM.

# Введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти.

- Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx.

- Непосредственная адресация – значение операнда задается непосредственно в команде, Например: mov ax,2.

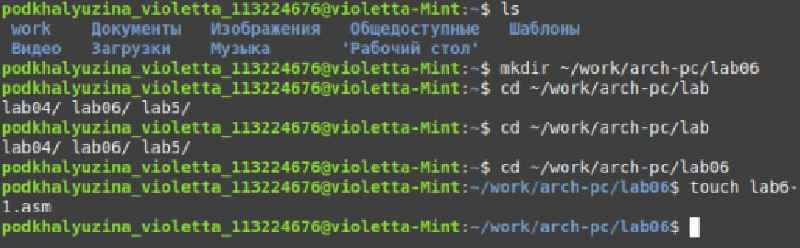
- Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

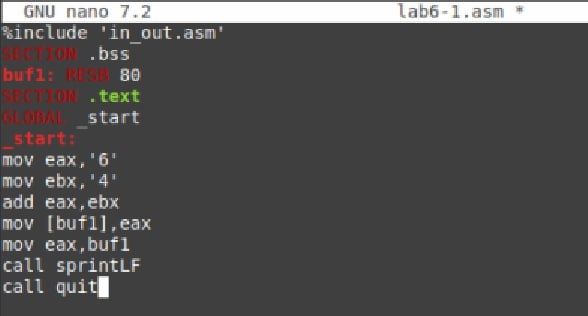
Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

# Выполнение лабораторной работы

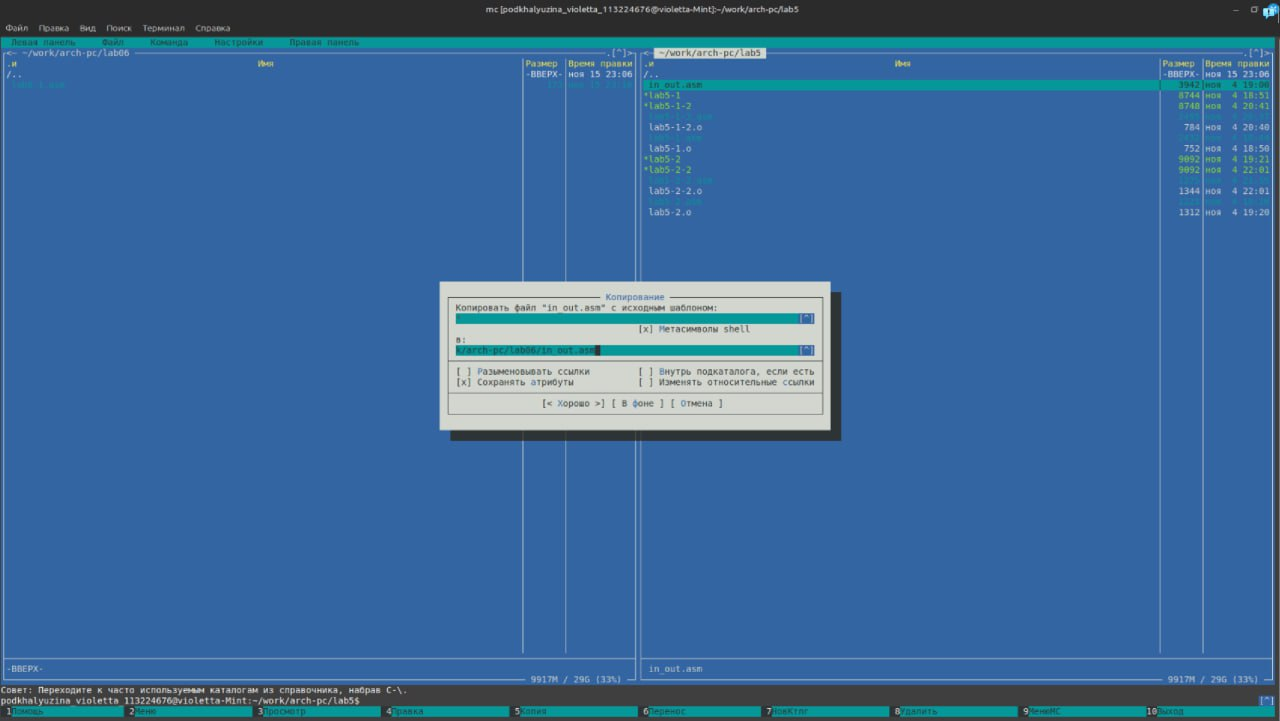
# Начало работы

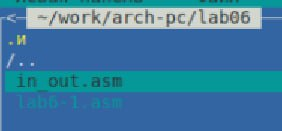
Я сделала каталог для программ лабораторной работы № 6, перешла в него и создала файл lab6-1.asm:

Открыла созданный файл lab6-1.asm, вставила в него программу вывода значения регистра eax и редактировала файл с помощью редактора nano



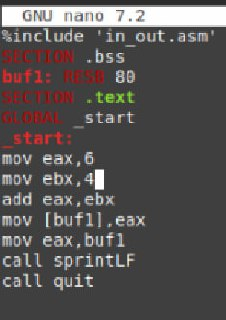
Скопировала файл in\_out.asm из папки lab5 в lab6 и проверила, всё ли прошло успешно.



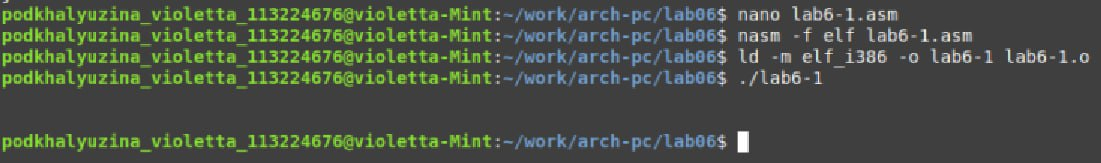


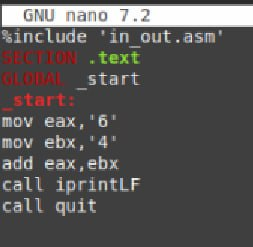
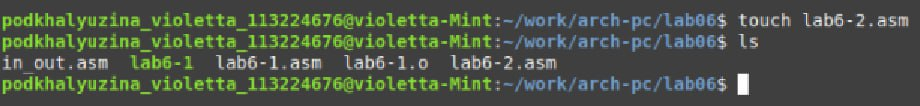
Создала исполняемый файл и запустила его

Далее изменила текст программы и вместо символов, записала в регистры числа

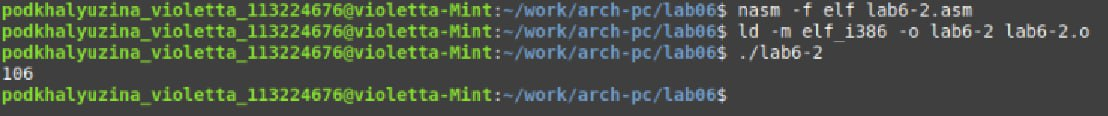


В таблице ascii коду 10 не принадлежит символа, поэтому вывелась пустая строка

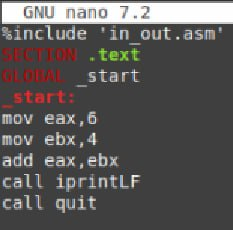
Я создала файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввела в него текст программы



После создала исполняемый файл и запустила его



Аналогично предыдущему примеру изменила символы на числа

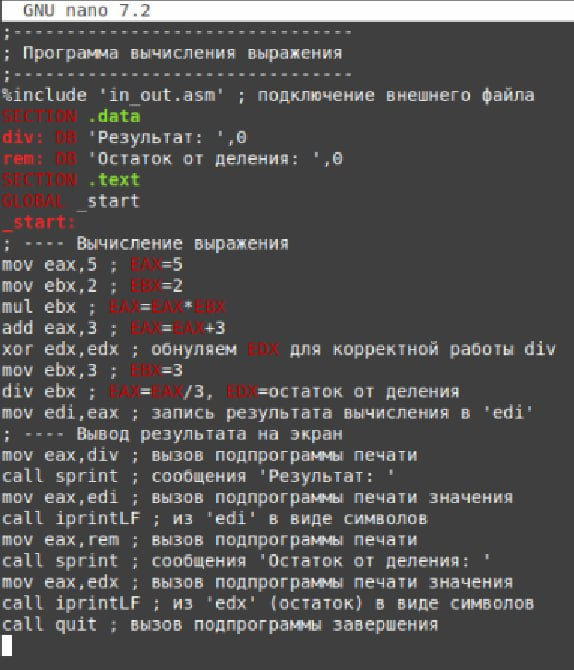
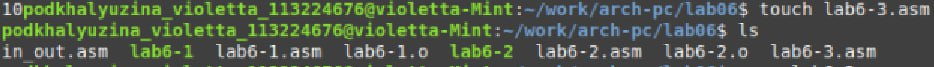


Далее смотрю на полученный резуль**т**ат

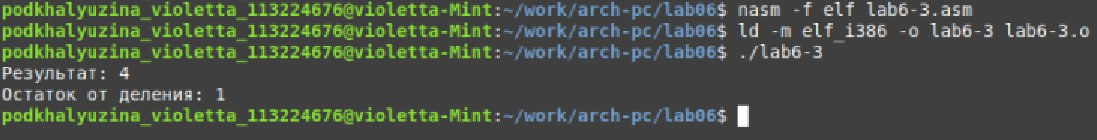
iprintLF делает отступ строки после вывода в отличии от iprint

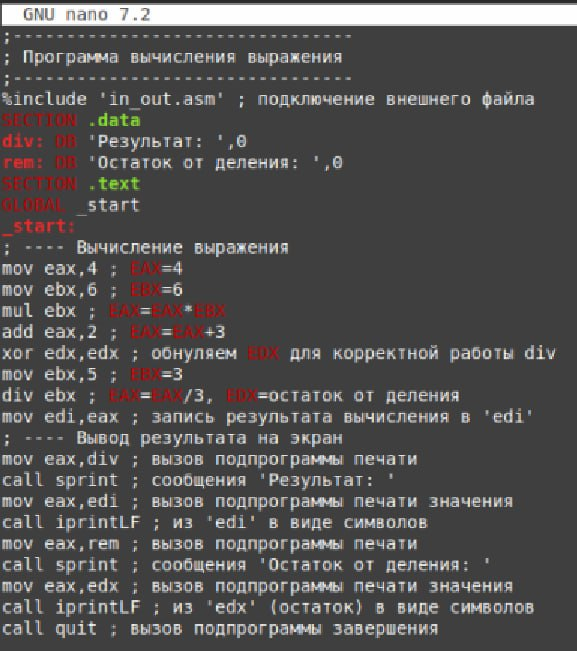


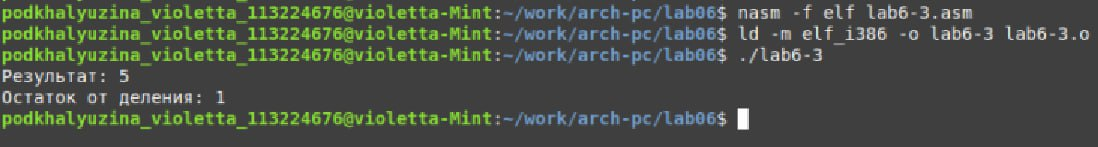
Далее я создала файл lab6-3.asm в каталоге ~/work/arch-pc/lab06

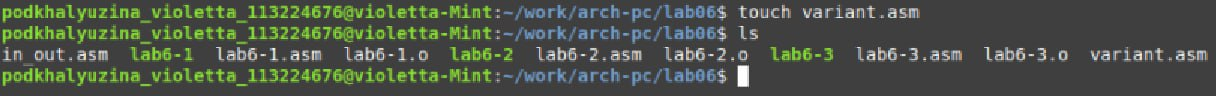


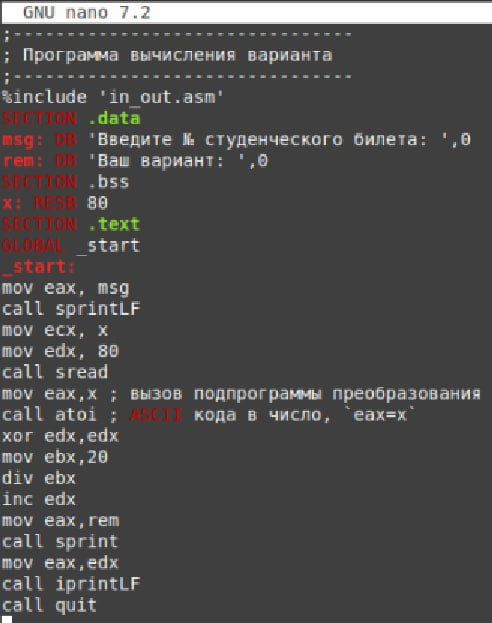
Далее создала исполняемый файл и запустила его

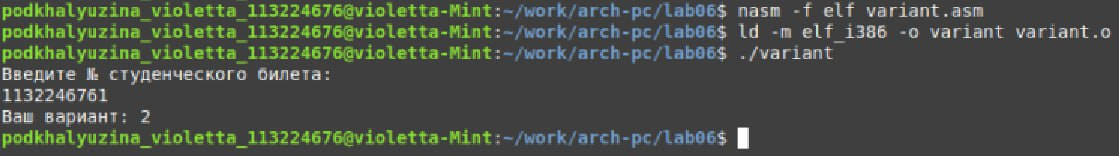
Изменила текст программы для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5. И создала исполняемый файл, проверила его работу

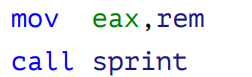


Далее я создала файл variant.asm в каталоге ~/work/arch-pc/lab06

Далее создала исполняемый файл и запустила его



1. За вывод сообщения “Ваш вариант” отвечают строки кода:



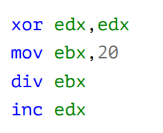
2.

1) Чтобы внести адрес вводимой строки x в регистр ecx mov edx, 80 - запись в регистр edx длины вводимой строки

2) Чтобы вызвать подпрограмму из внешнего файла, обеспечивающую ввод сообщения с клавиатуры

3) call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax

4) За вычисления варианта отвечают строки:



5. При выполнении инструкции div ebx остаток от деления записывается в регистр edx

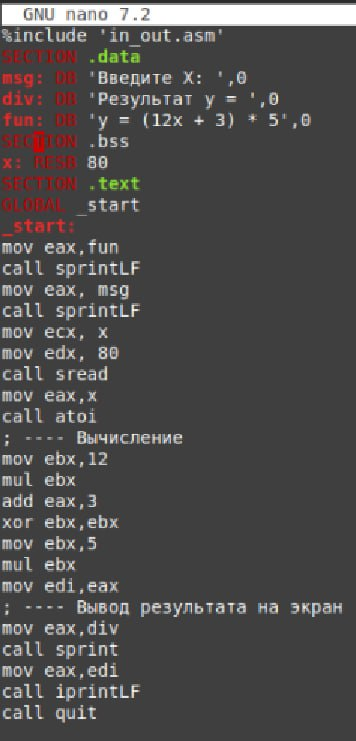
6. Инструкция inc edx увеличивает значение регистра edx на 1

7. За вывод на экран результатов вычислений отвечают строки:



# 3.2 Самостоятельная работа

Я написала программу вычисления выражения 𝑦 = 𝑓(𝑥). Программа должна выводит выражение для вычисления, выводит запрос на ввод значения 𝑥, вычисляет заданное выражение в зависимости от введенного 𝑥, выводить результат вычислений.





# Контрольные вопросы для самопроверки

1. Какой синтаксис команды сложения чисел?

add destination, source (например, add ax, bx)

2. Какая команда выполняет умножение без знака?

mul source

3. Какой синтаксис команды деления чисел без знака?

div source

4. Куда помещается результат при умножении двухбайтовых операндов?

В регистры DX:AX

5. Перечислите арифметические команды с целочисленными операндами и их назначение:

- ADD: сложение

- SUB: вычитание

- MUL: умножение без знака

- IMUL: умножение со знаком

- DIV: деление без знака. - IDIV: деление со знаком

- INC: увеличение на 1. - DEC: уменьшение на 1

- NEG: смена знака

6. Где находится делимое при целочисленном делении?

В регистре DX:AX (для 16 бит) или EDX:EAX (для 32 бит)

7. Куда помещаются неполное частное и остаток при делении целочисленных операндов?

Частное — в AX (или EAX), остаток — в DX (или EDX)

# 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).