**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

Студент: Барханоева Раяна Магометовна

Ст.билет: 1032252468

Группа: НКАбд-01-25

МОСКВА

2025 г

**Содержание**

[1. Цель задачи 3](#__RefHeading___Toc1252_1947014752)

[2. Выполнение работы 4](#__RefHeading___Toc1254_1947014752)

[3. Самостоятельная работа 16](#__RefHeading___Toc1256_1947014752)

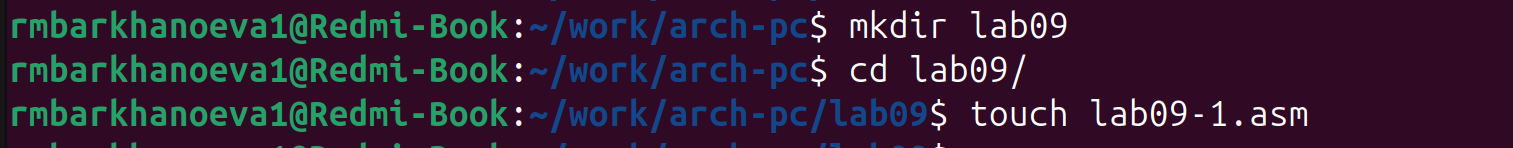
[Вывод 23](#__RefHeading___Toc1258_1947014752)

# 1. Цель задачи

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

# 2. Выполнение работы

Каталог (рисунок 1).

 Рисунок 1.

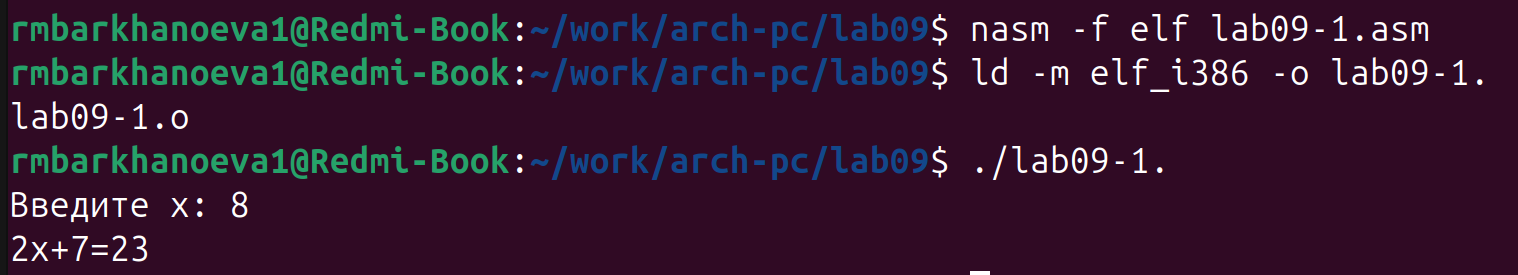
Создали каталог и файл lab09-1.asm

Программа (рисунок 2).

Рисунок 2.

Программа вычисления арифметического выражения 𝑓(𝑥) = 2𝑥 + 7 с помощью подпрограммы \_calcul.

Тест программы (рисунок 3).

Рисунок 3.

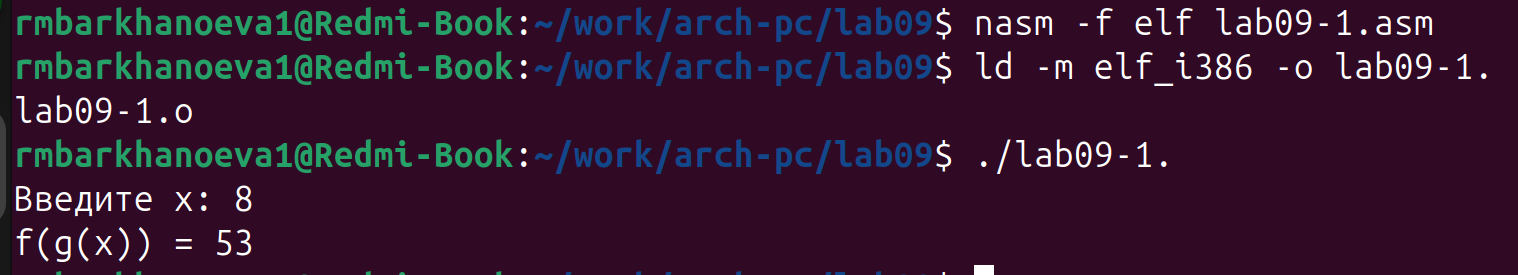
Скомпелировали программу и запустили. Ввели число и посчитали формулу.

Программа (рисунок 4).

Рисунок 4.

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).

Тест программы (рисунок 5).

Рисунок 5.

Скомпелировали и запустили программу.

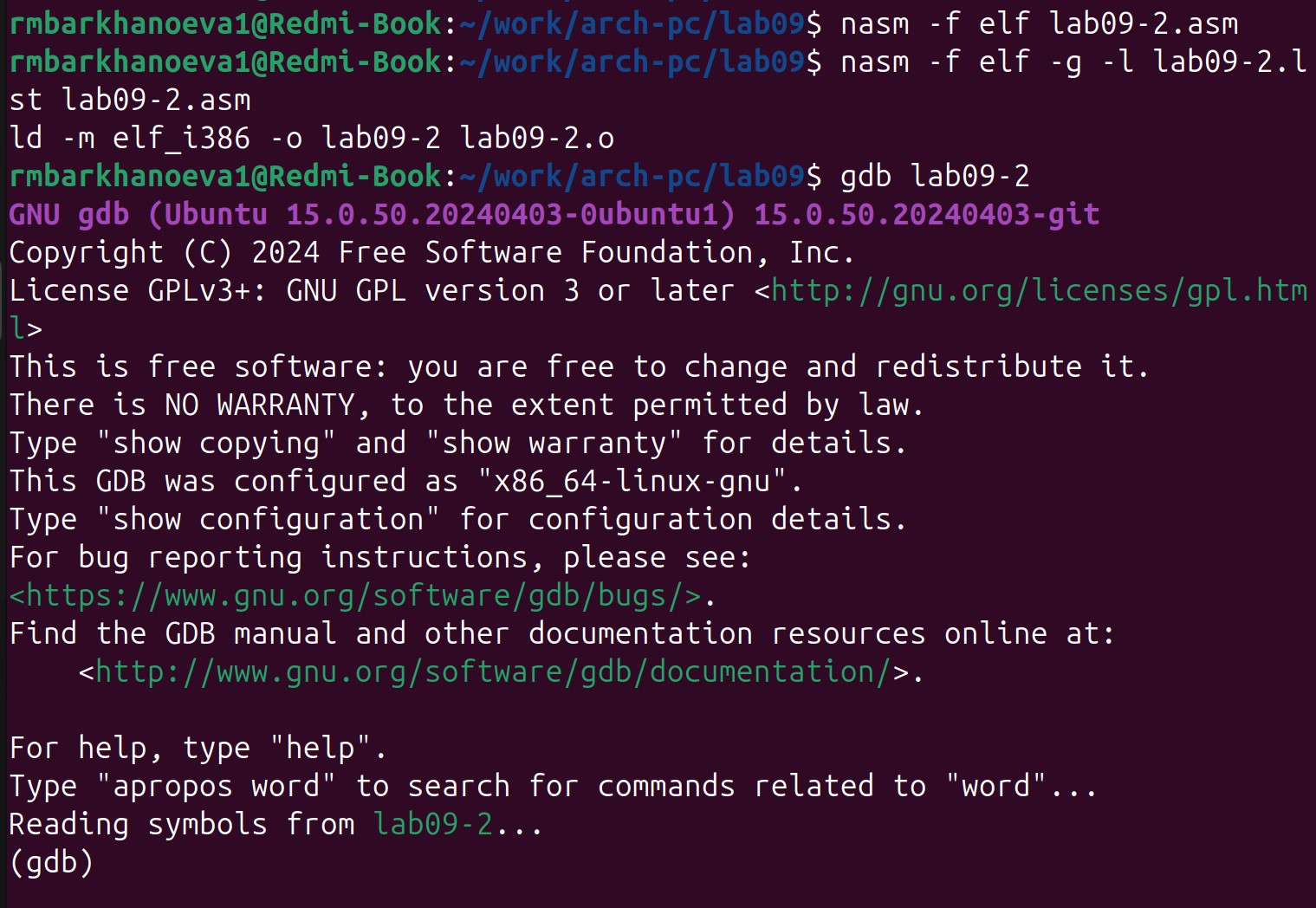
Программа (рисунок 6).

Рисунок 6.

Создала файл lab09-2.asm, с текстом программы из Листинга 9.2. (Программа печати

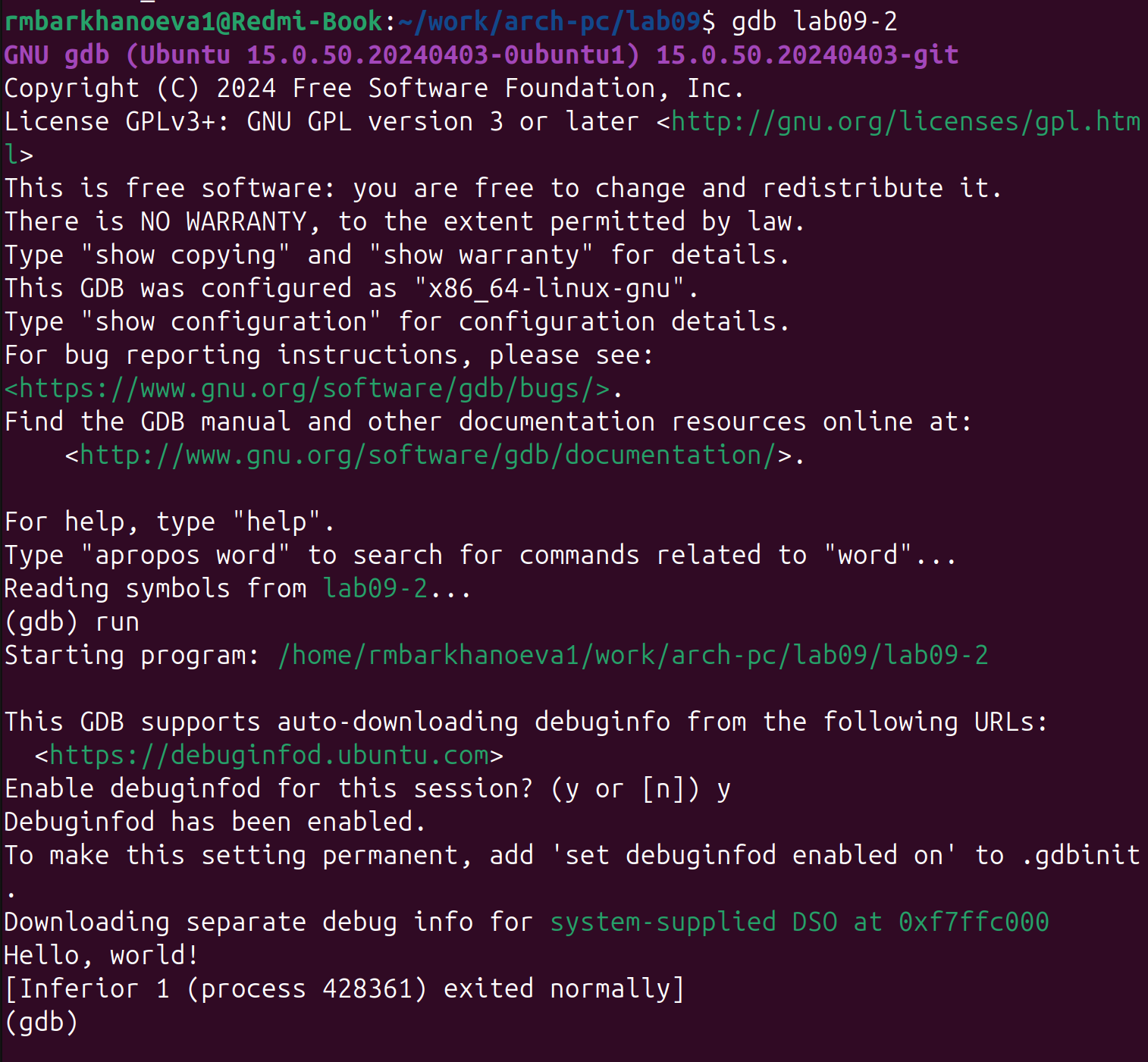
сообщения Hello world!)

Тест и запуск GDB (рисунок 7).

Рисунок 7.

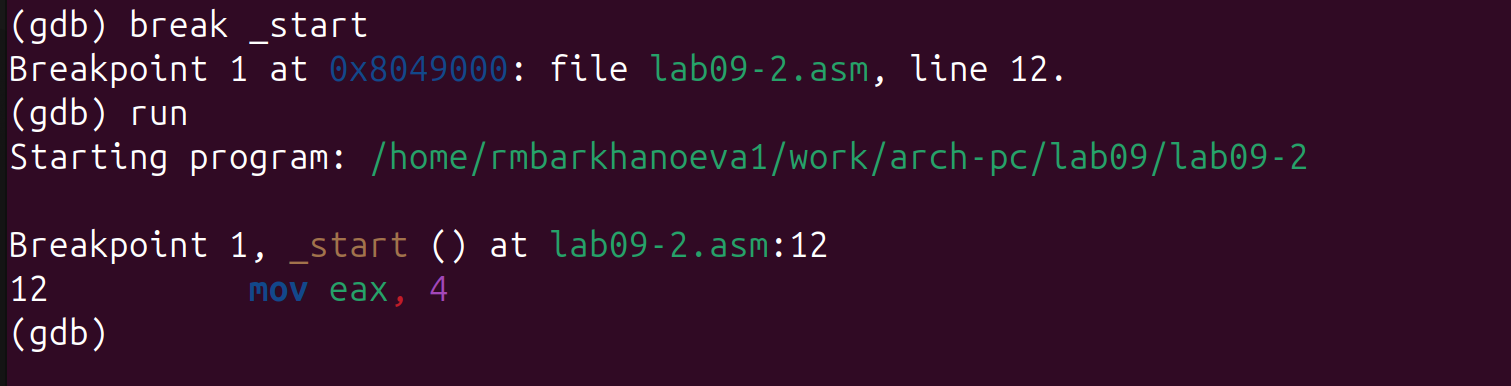
Скомпелировали и запустили программу. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Загрузили исполняемый файл в отладчик gdb.

GDB (рисунок 8).

Рисунок 8.

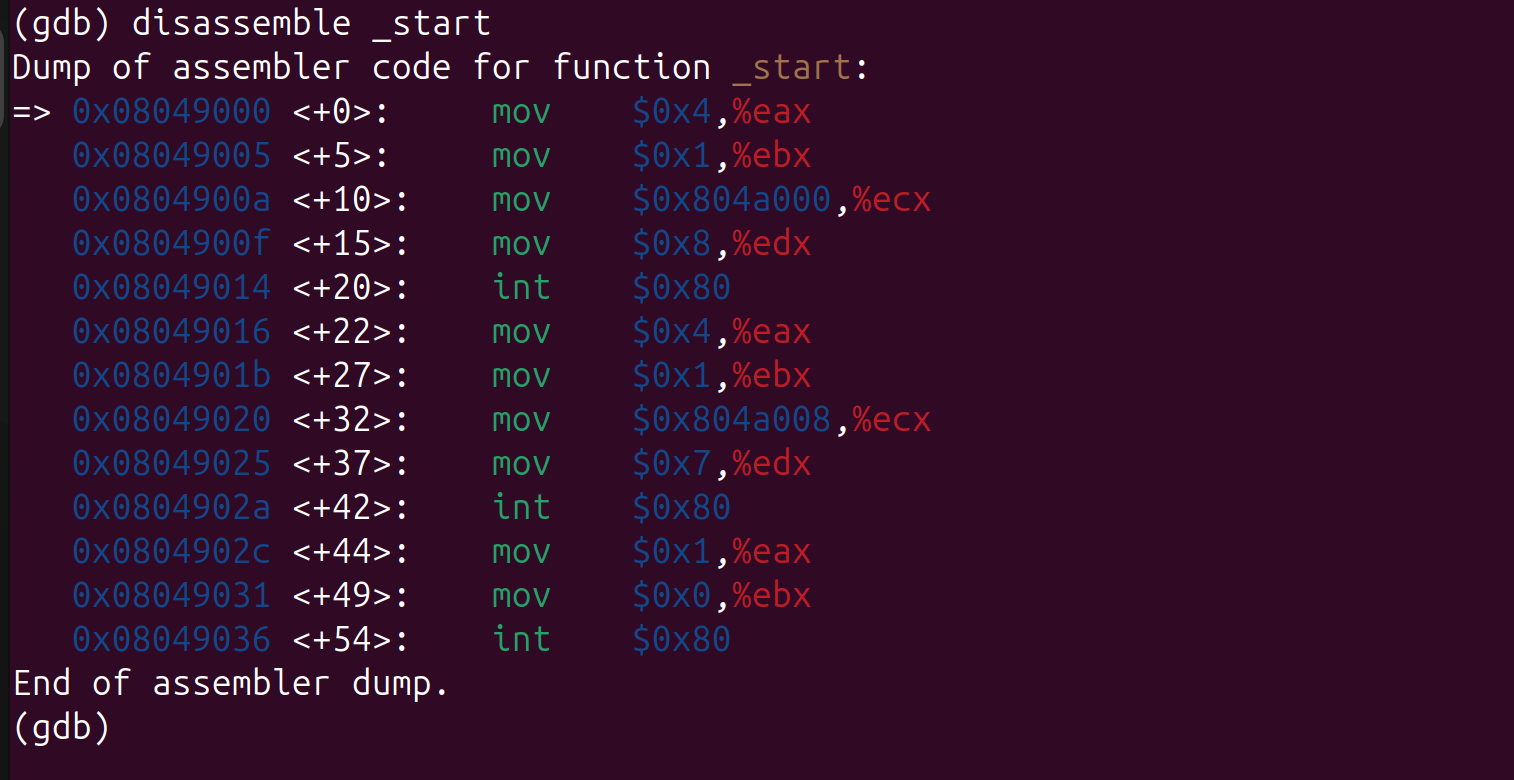
Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run. Видим Hello World.

Брейкпоинт (рисунок 9).

Рисунок 9.

Для более подробного анализа программы установили брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы и запустили программу

Код программы (рисунок 10).

Рисунок 10.

Посмотр дисассимилированного кода программы с помощью команды disassemble начиная с метки \_start.

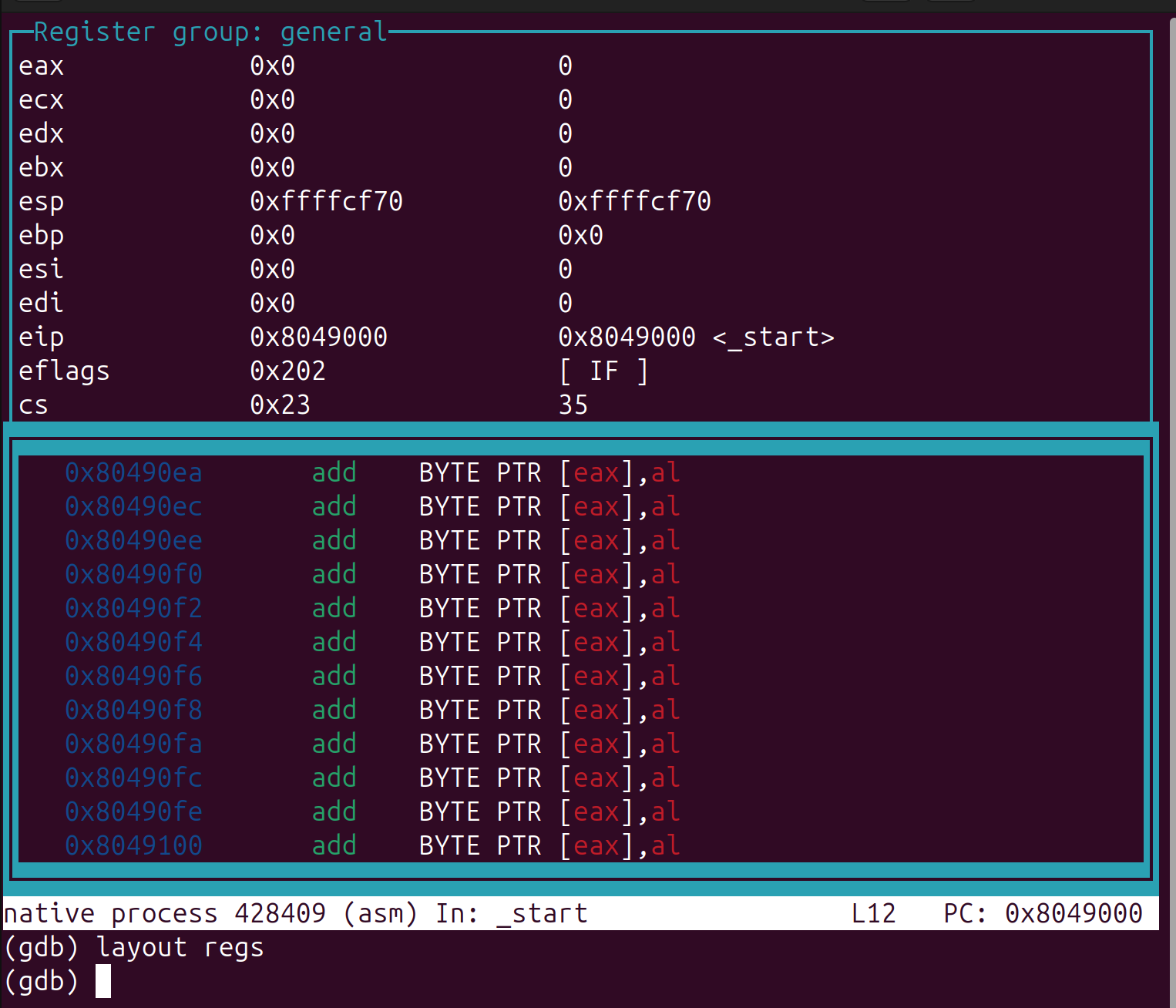
Код программы (рисунок 11).

Рисунок 11.

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

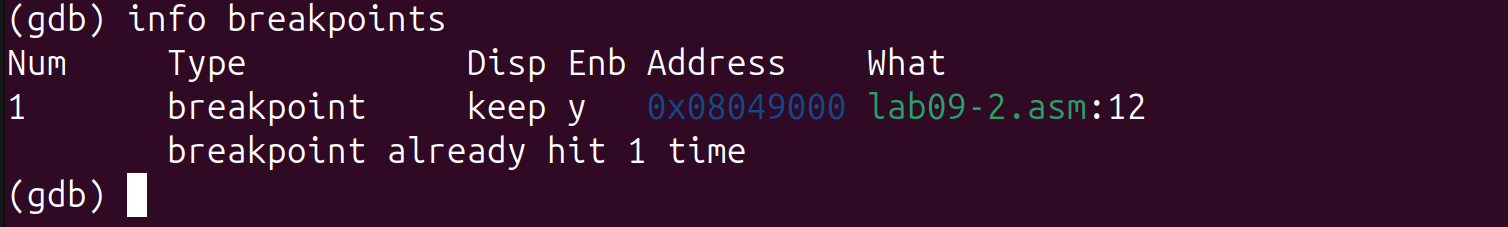
Различия отображения ATT и Intel в порядке операндов, регистрах, размерах данных.

Псевдографика (рисунок 12).

Рисунок 12.

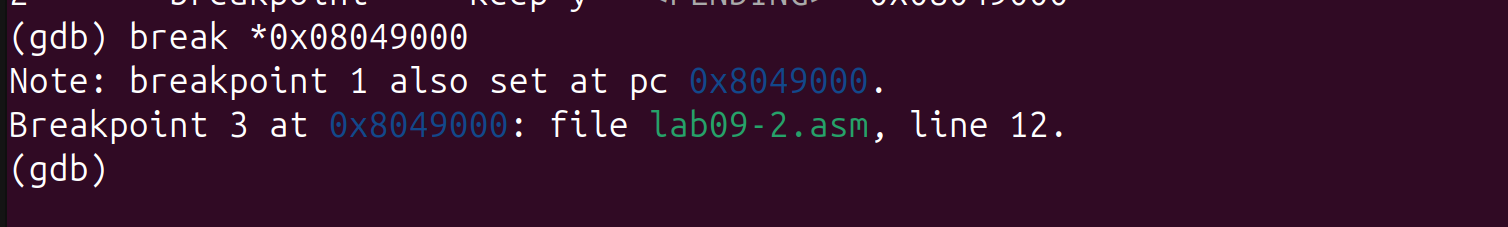
Переключислся в режим псевдографики для более удобного анализа программы.

Точка (рисунок 13).

Рисунок 13.

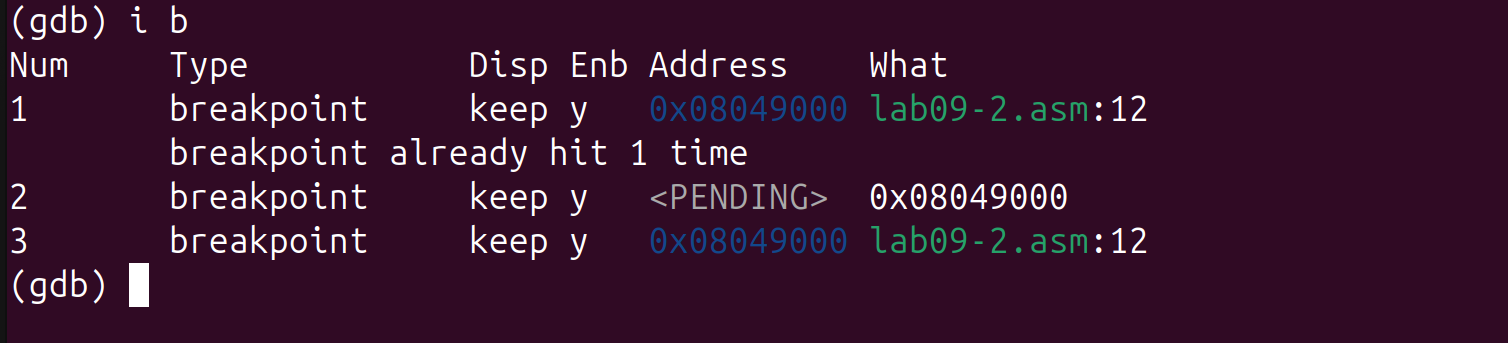
Установил с помощью команды info breakpoints точку остановы.

Точка (рисунок 14).

Рисунок 14.

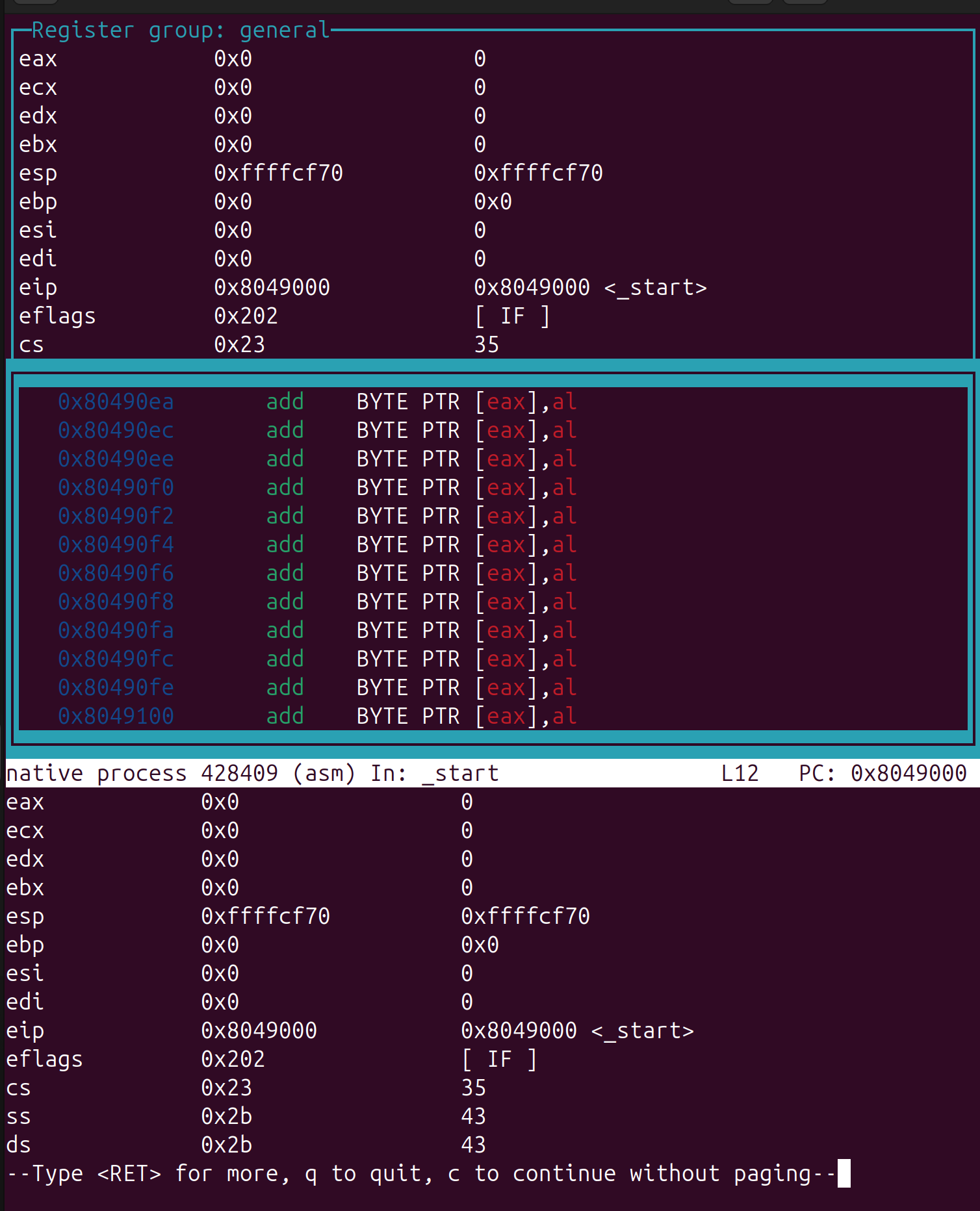
Установила еще одну точку остановы по адресу инструкции.

Информация (рисунок 15).

Рисунок 15.

Информацию о всех установленных точках останова.

Регистр (рисунок 16).

Рисунок 16.

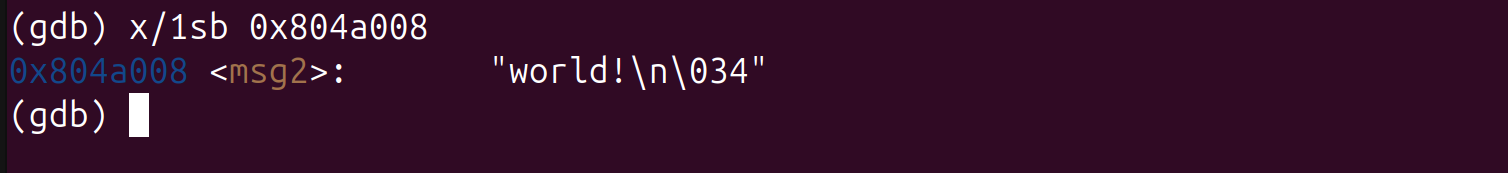
Просмотр содержимого регистров с помощью команды info registers.

Имя (рисунок 17).

Рисунок 17.

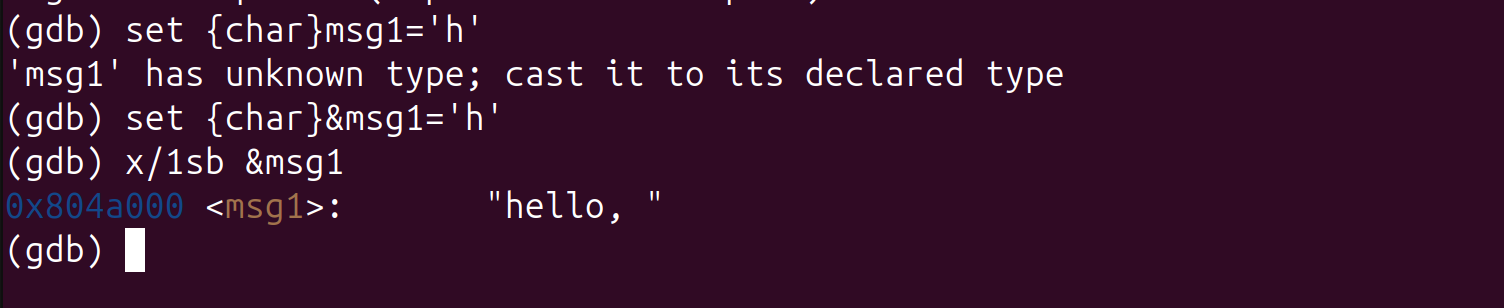
Посмотрела значение переменной msg1 по имени.

Адрес (рисунок 18).

Рисунок 18.

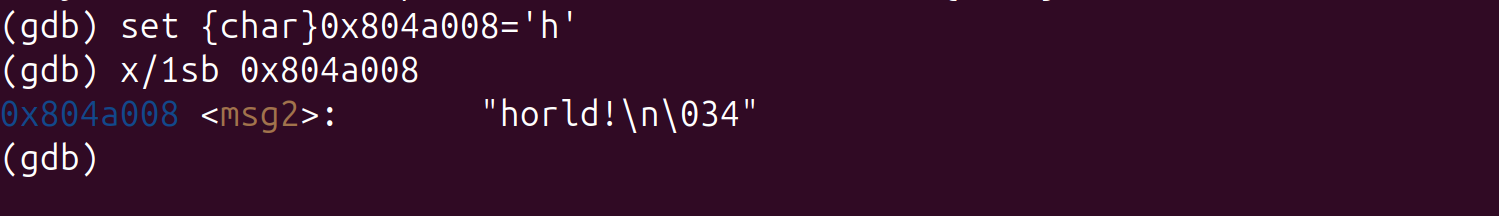
Посмотрела значение переменной msg2 по адресу.

Изменение (рисунок 19).

Рисунок 19.

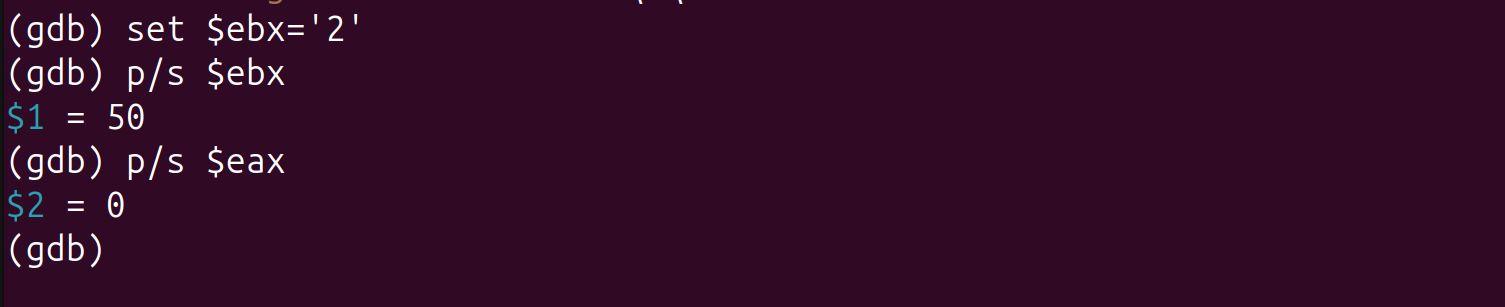
Изменение первого символа переменной msg1.

Изменение (рисунок 20).

Рисунок 20.

С помощью команды set изменила значение регистра.

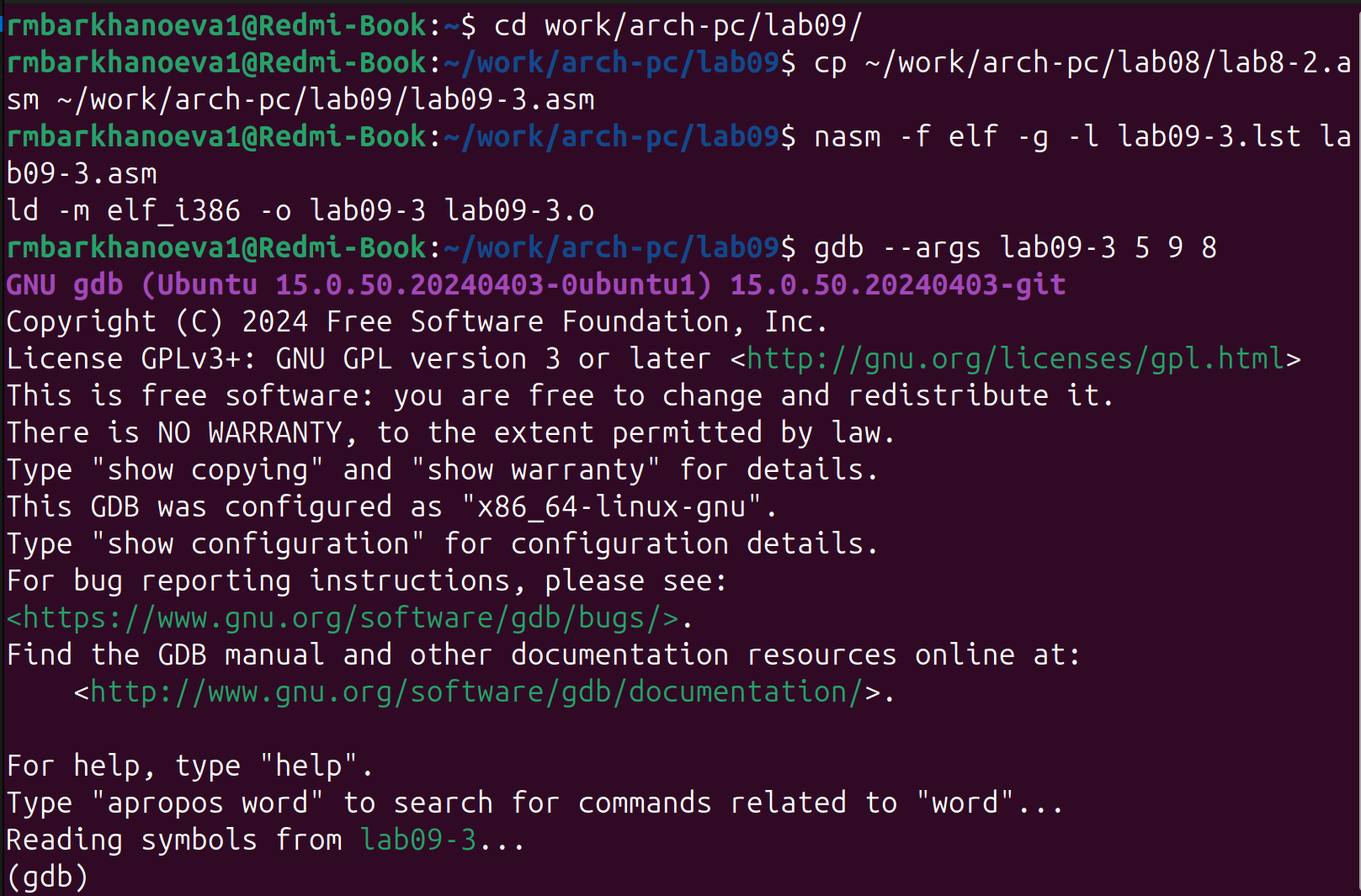
Изменение (рисунок 21).

Рисунок 21.

С помощью команды set изменила значение регистра ebx.

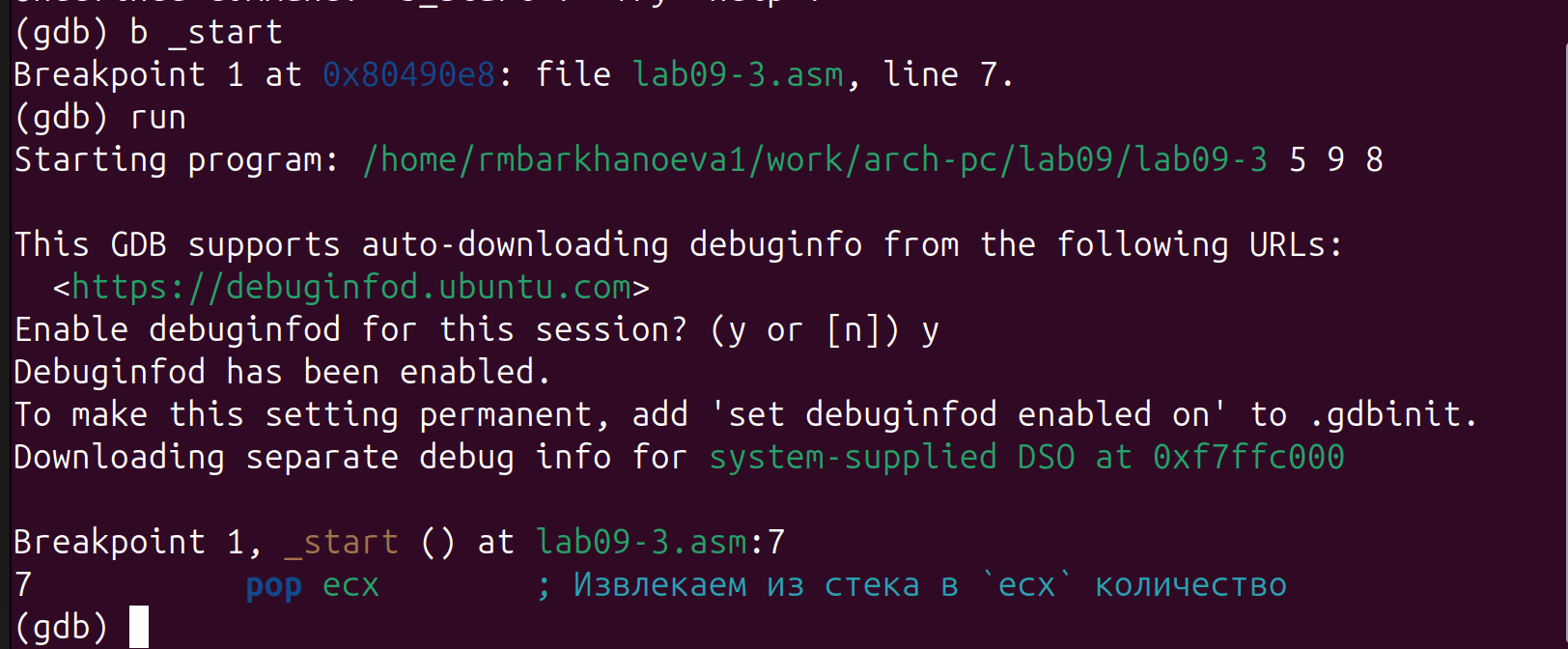
p/s $ebx в GDB выводит значение регистра EBX не как число, а как **адрес C-строки**, поэтому GDB пытается прочитать память по этому адресу и показать строку до символа \0; если в EBX лежит обычное число, а не адрес строки, вывод будет мусорным или пустым, в отличие от p $ebx, который всегда показывает числовое значение регистра.

Файлы (рисунок 22).

Рисунок 22.

С помощью команды cp скопировала файл lab8-2.asm в lab09-3.asm. Создала исполняемый файл. Загрузила исполняемый файл в отладчик, указав аргументы

Точка (рисунок 23).

Рисунок 23.

Установили точку остановы и запустили ее.

Адрес (рисунок 24).

Рисунок 24.

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы).

Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.), потому что в 32-битном режиме x86 размер одного слова (word) и любого адреса/целого регистра составляет **4 байта**, стек выровнен по 4 байтам, и каждая операция push или pop помещает или извлекает **4-байтовое значение**, поэтому каждый следующий элемент в стеке находится на смещении, кратном 4.

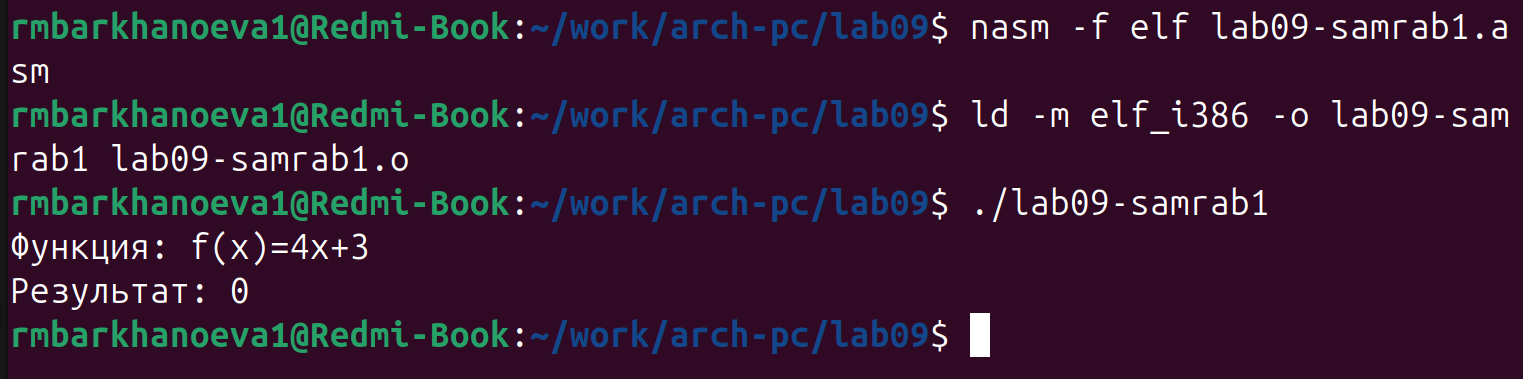
# 3. Самостоятельная работа

Программа (рисунок 25).

Рисунок 25.

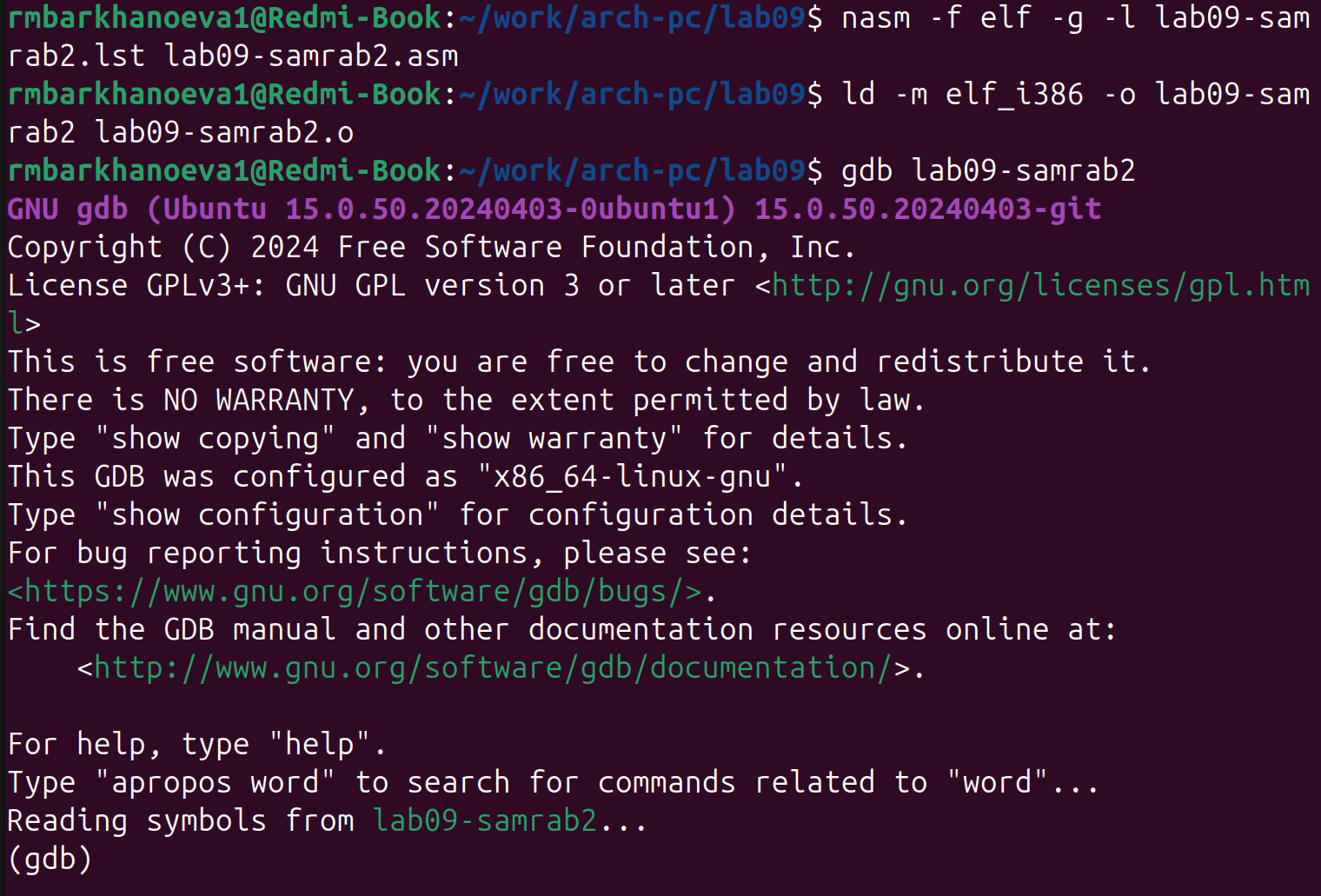
Программа реализовывает вычисление значения функции 𝑓(𝑥) как подпрограмму.

Тест программы (рисунок 26).

Рисунок 26.

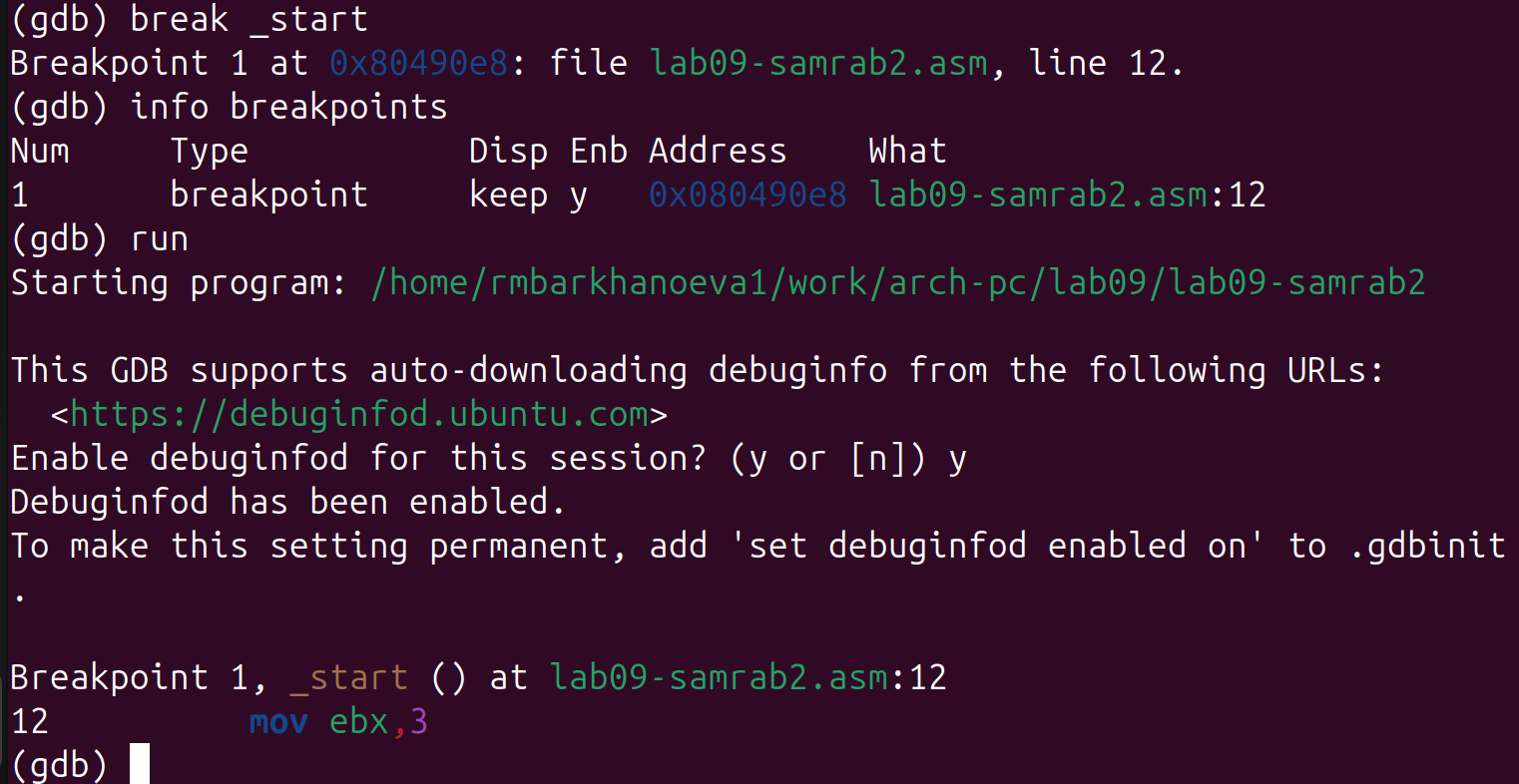
Компиляция и тест программы.

Оболочка (рисунок 27).

Рисунок 27.

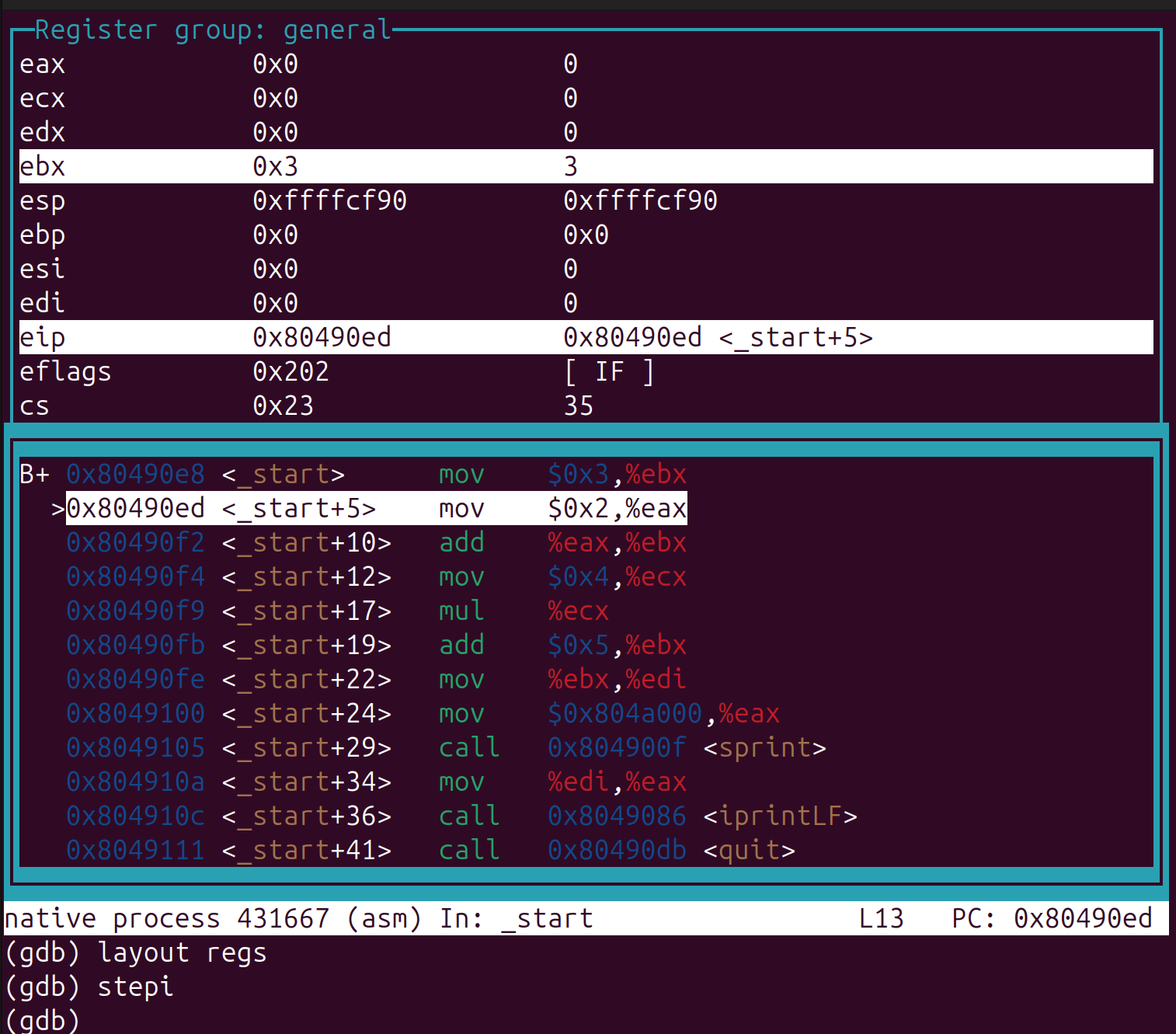
Запускаем оболчку gdb

Поинт (рисунок 28).

Рисунок 28.

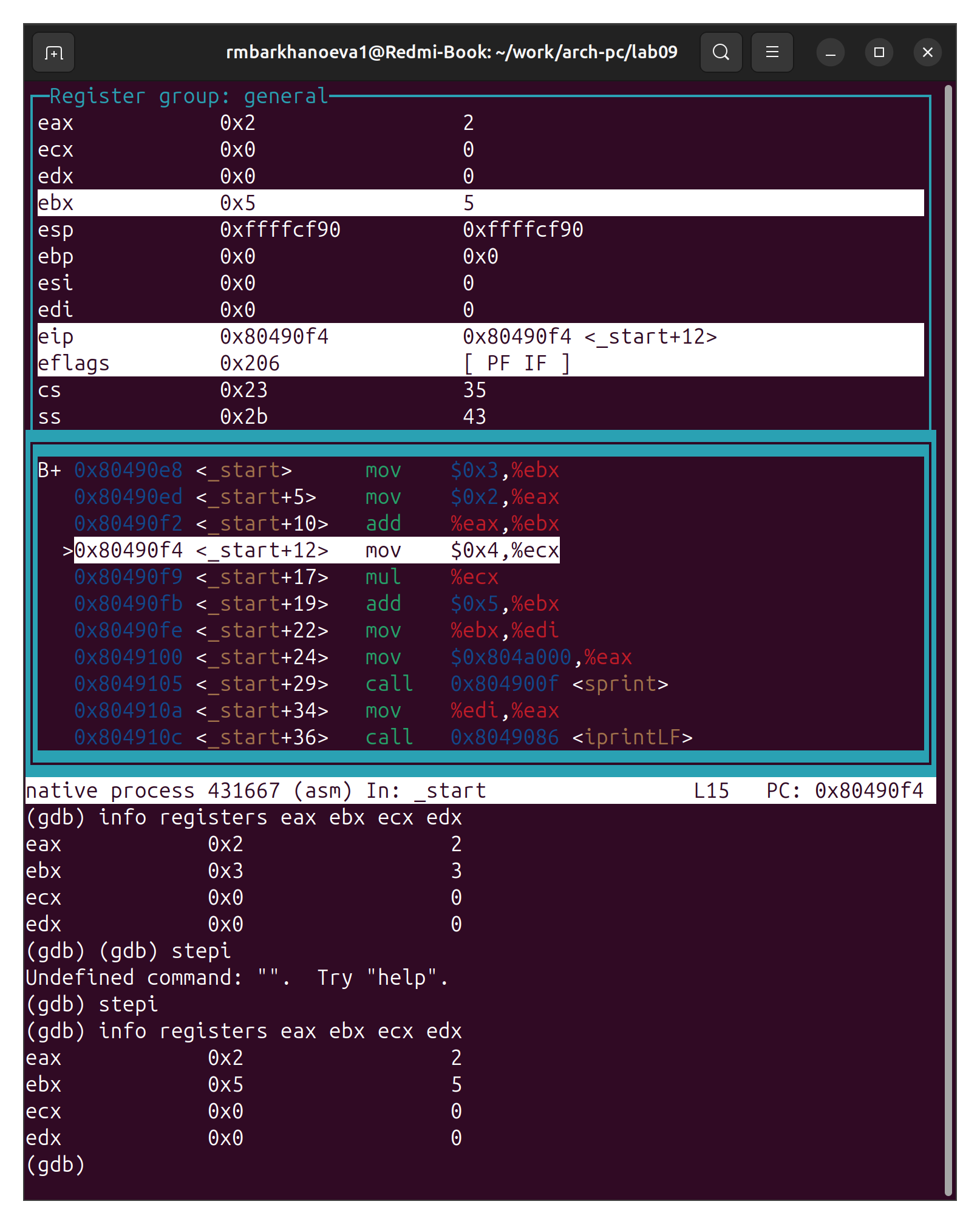
Устанавлваем брейкпоинт на \_start и запускаем программу run

Регистры (рисунок 29).

Рисунок 29.

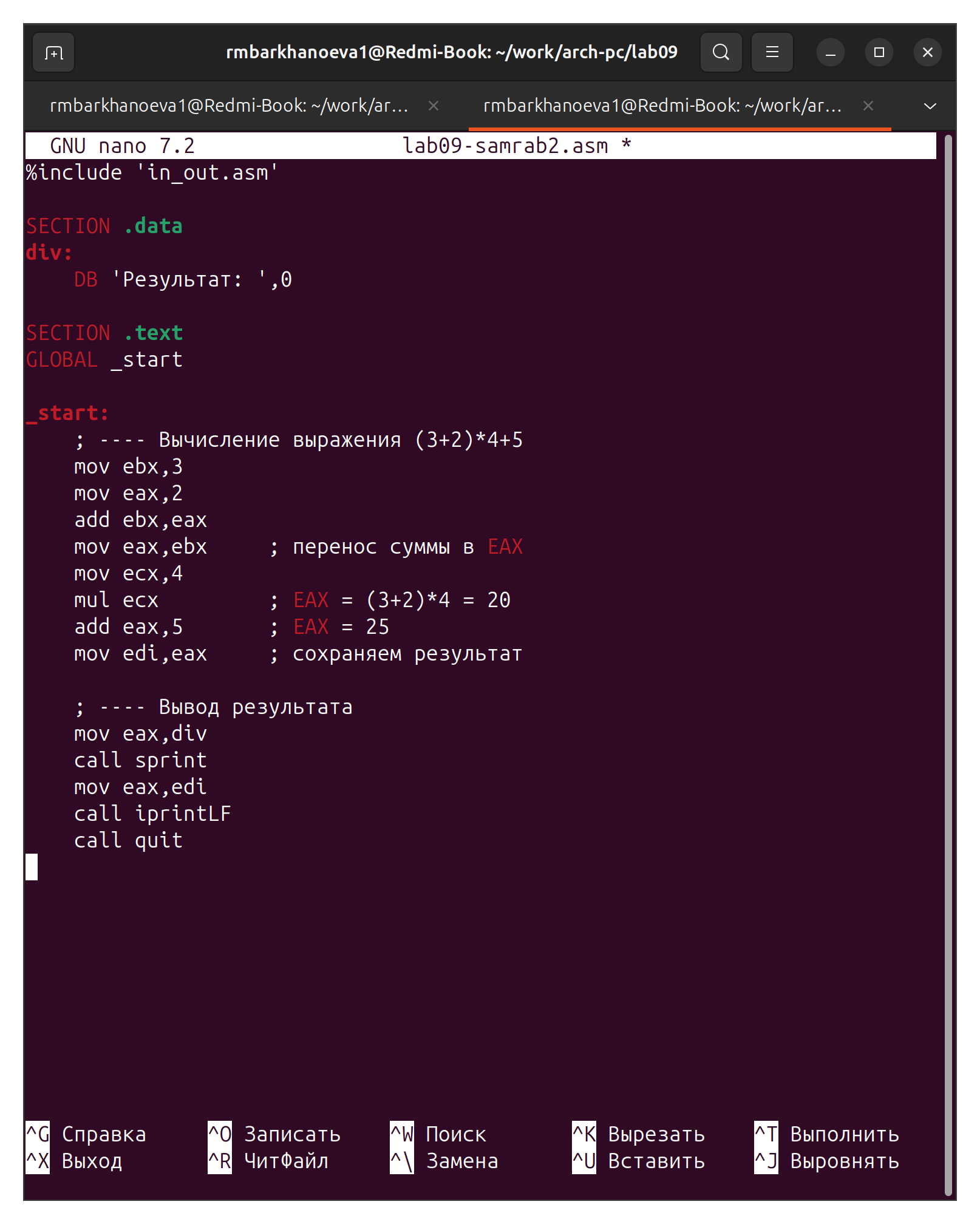
Смотрим значения регистров на старте. Переходим к пошаговому выполнению интсрукций stepi.

Регистры (рисунок 30).

Рисунок 30.

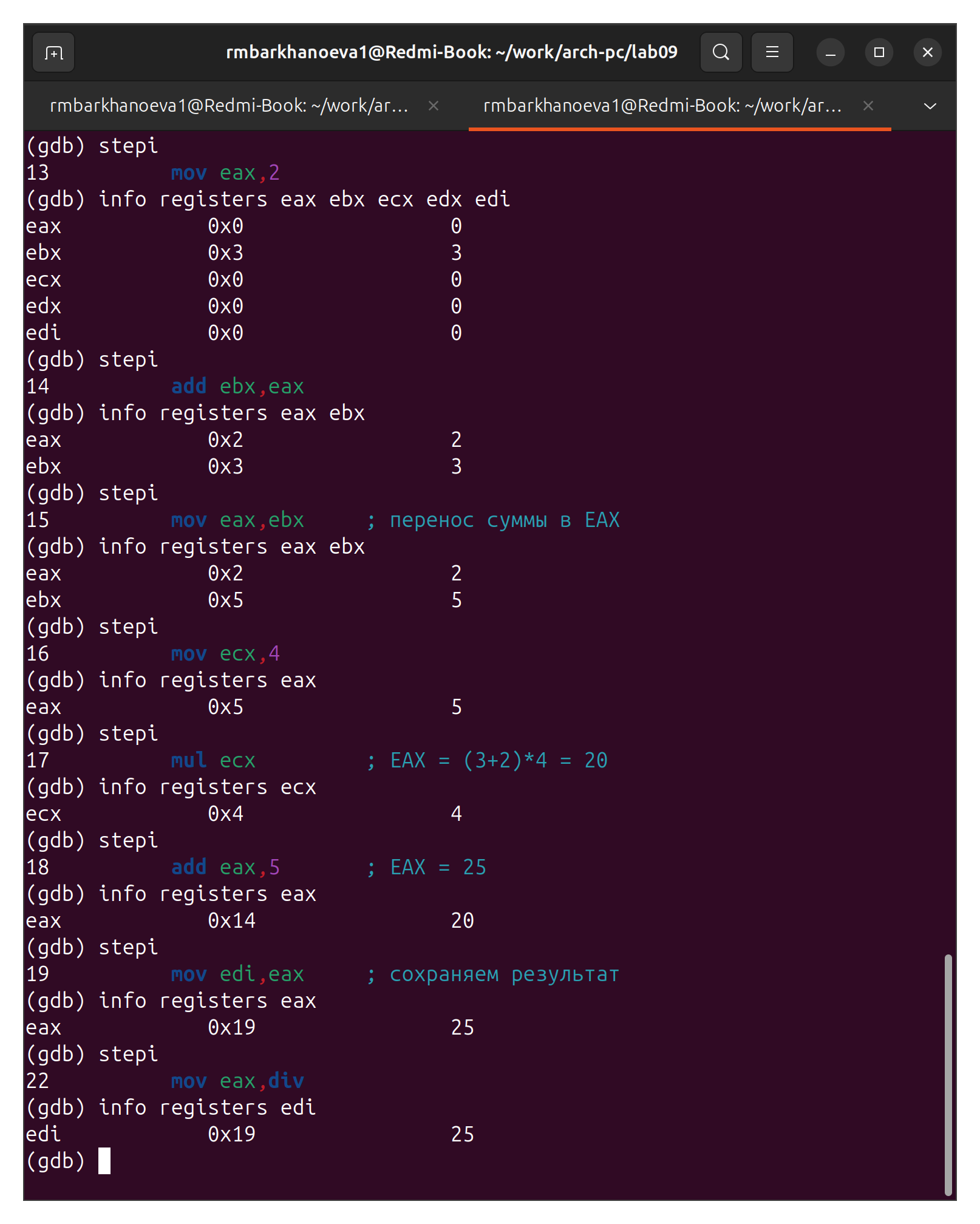
После каждой инструкции проверяем регистры. Ошибка. После add ebx,eax → EBX = 5, но EAX = 2. Инструкция mul ecx умножает **EAX**, а ECX = 0 → результат = 0. Поэтому весь результат будет неверный.

Программа (рисунок 31).

Рисунок 31.

В исходном файле добавляем **перенос суммы в EAX перед умножением** и созраняем результат.

Инструкции (рисунок 32).

Рисунок 32.

Сохранили код и теперь пошагово выполняем интсрукции.

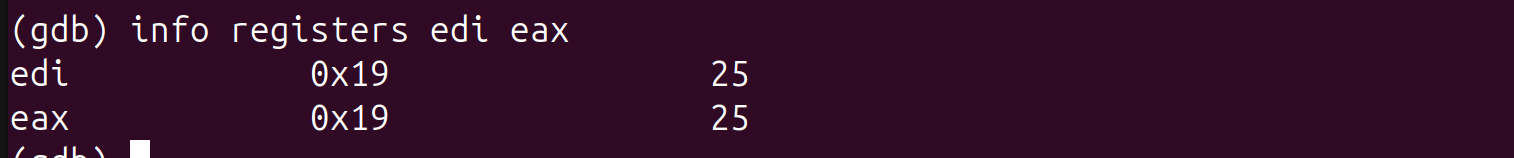
После add ebx,eax → EBX = 5

После mov eax,ebx → EAX = 5

После mul ecx → EAX = 20

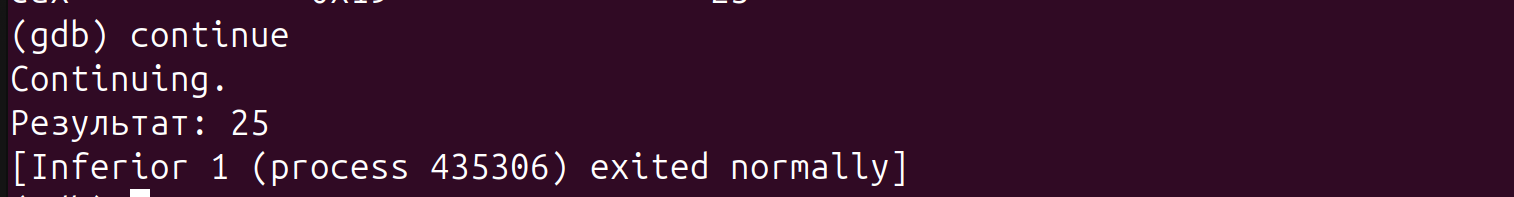
После add eax,5 → EAX = 25

Проверка (рисунок 33).

Рисунок 33.

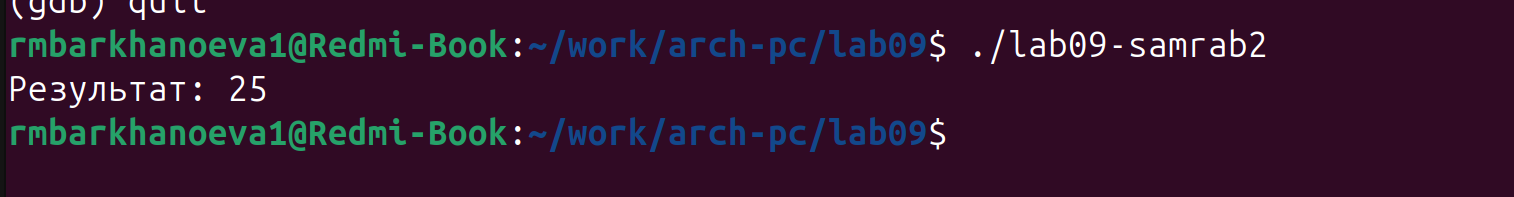
Проверили.

Завершение (рисунок 34).

Рисунок 34.

Завершение программы и вывод результата.

Вывод (рисунок 35).

Рисунок 35.

# Вывод

В ходе лабораторной работы была изучена работа с 32-битным ассемблером x86, вычисление арифметического выражения с использованием регистров общего назначения, а также пошаговая отладка программы в GDB; было закреплено понимание порядка выполнения инструкций, влияния каждой команды на значения регистров, принципов работы инструкции mul, особенностей хранения данных в регистрах и стеке, выравнивания стека по 4 байта, а также получены практические навыки использования команд stepi, info registers и print для анализа и поиска ошибок, в результате чего программа была корректно отлажена и выдала правильный результат вычислений.