Понятие подпрограммы. Отладчик GDB

Лабораторная работа №9

Приходько Иван Иванович

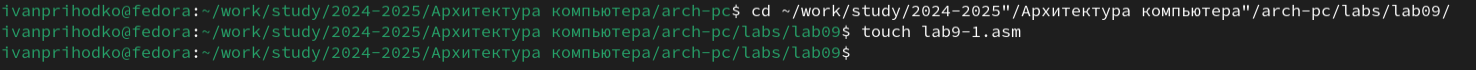
Содержание

# 1 Цель работы

Ознакомиться с понятием подпрограмм в Ассемблере и научиться использовать подпрограммы на практике. Ознакомиться с отладчиком gdb и научиться использовать его

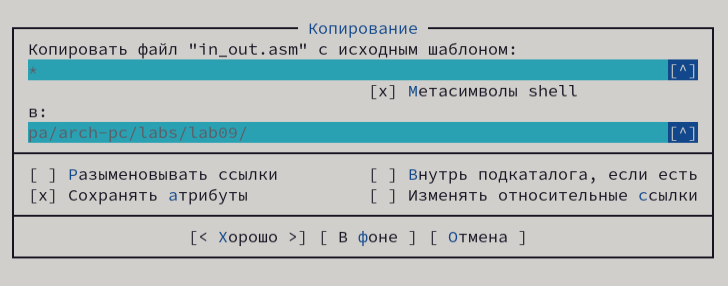
# 2 Выполнение лабораторной работы

Для начала выполнения работы необходимо создать файл lab9-1.asm (рис. 2.1).



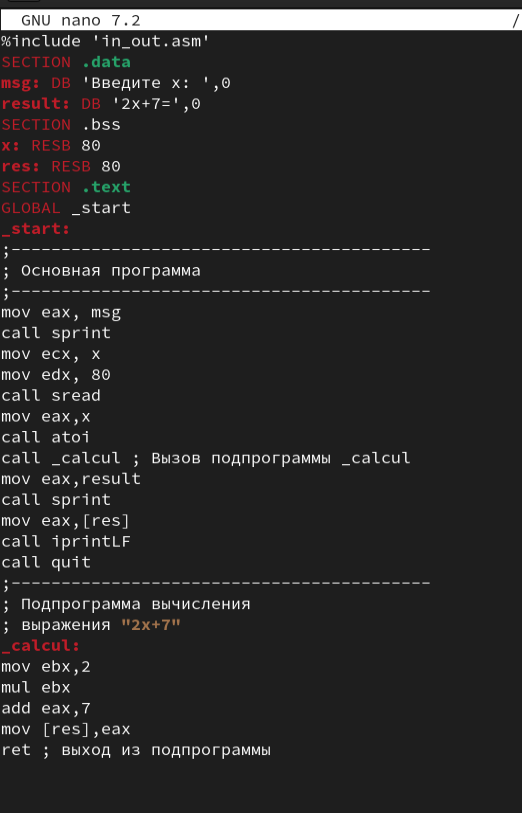
Создание lab9-1.asm

Скопируем файл in\_out.asm из директории прошлой работы (рис. 2.2).



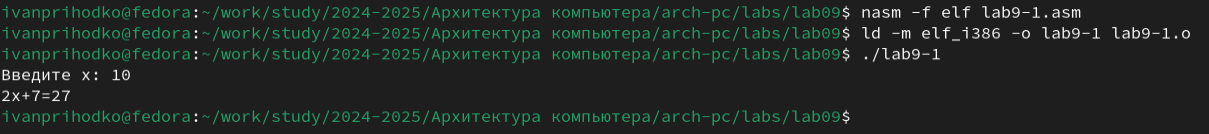
Копирование in\_out.asm

Вставим в файл lab9-1.asm код из листинга 9.1 (рис. 2.3).



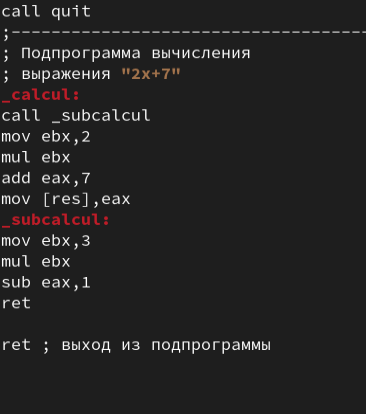
Вставка кода из файла листинга 9.1

Соберем и запустим файл (рис. 2.4).



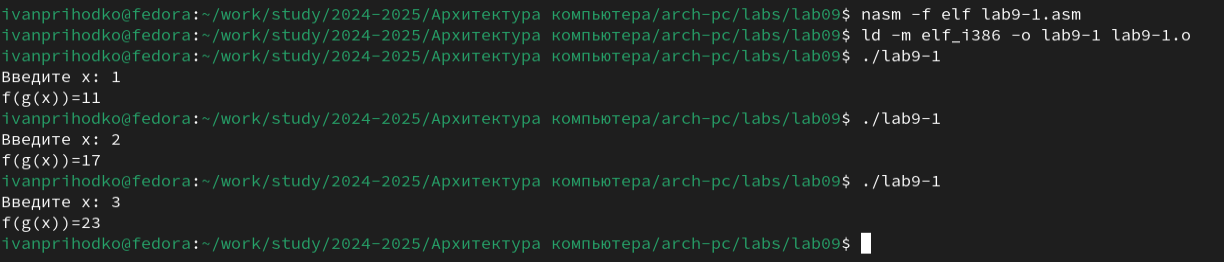
Сборка и запуск lab9-1.asm

Теперь изменим файл так, чтобы внутри подпрограммы была ещё одна подпрограмма, вычисляющая значение g(x) и чтобы она передавала значение в первую подпрограмму, которая бы уже вычислила значение f(g(x)) (рис. 2.5).



Изменение файла lab9-1.asm

Повторно соберем и запустим программу (рис. 2.6).

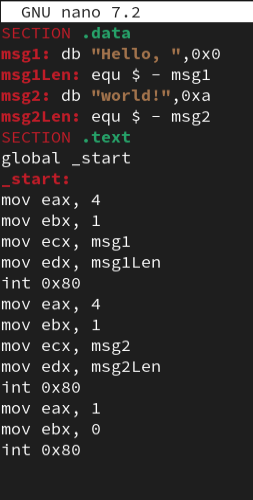


Повторная сборка и запуск lab9-1.asm

Создадим новый файл и вставим в него код из листинга 9.2 (рис. 2.7).

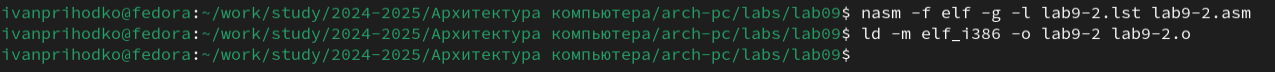
Создание lab9-2.asm

Создание lab9-2.asm



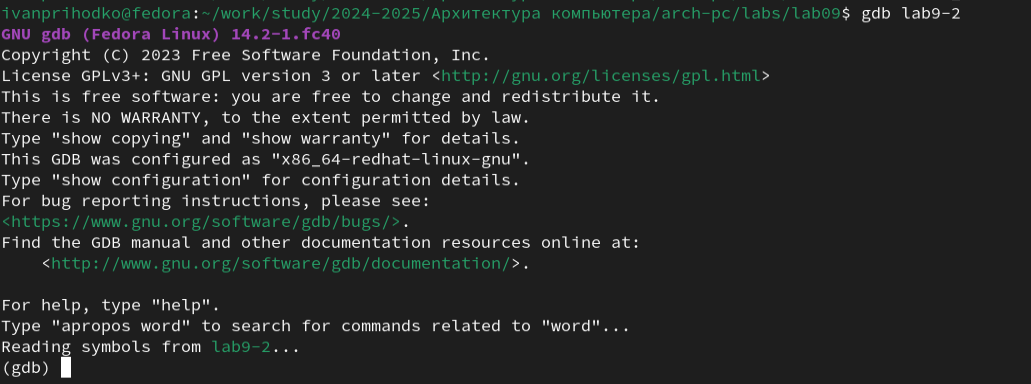
Редактирование lab9-2.asm

Соберём программу с использованием аргумента -g (рис. 2.9).



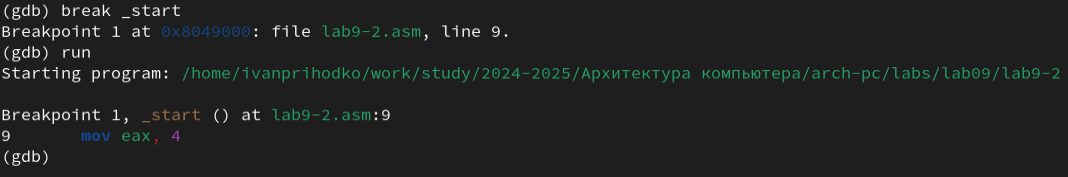
Сборка lab9-2.asm

Теперь загрузим её в gdb (рис. 2.10).



Загрузка программы lab9-2.asm в gdb

Создадим брейкпоинт на метке \_start с помощью команды break (рис. 2.11).



Создание брейкпоинта

С помощью команды disassemble дизассемблируем её (рис. 2.12).



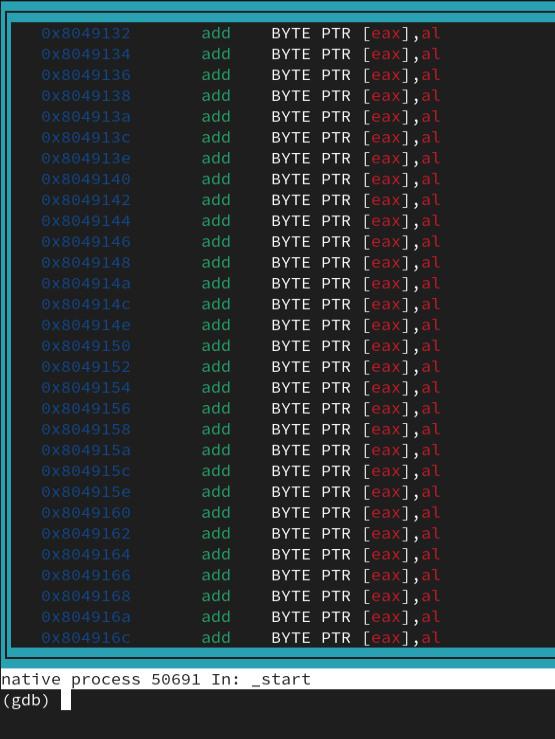
Дизассемблирование программы

Переключим синтаксис вывода на intel и повторно дизассемблируем программу (рис. 2.13).



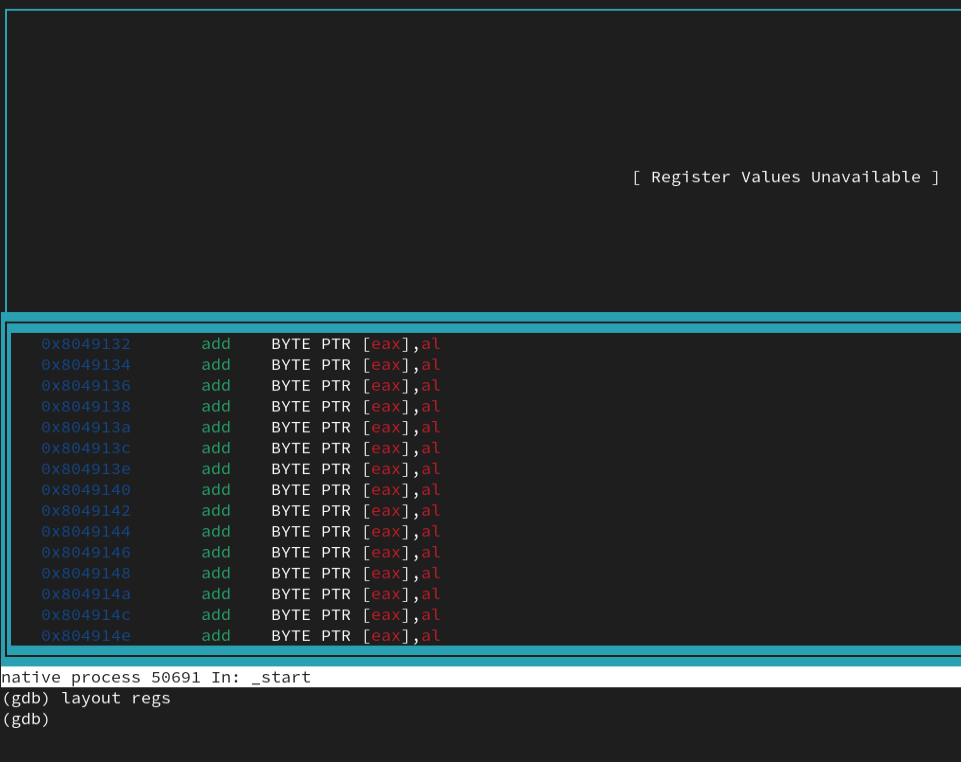
Переключение на синтаксис intel

Включим графическое отображения кода (рис. 2.14).



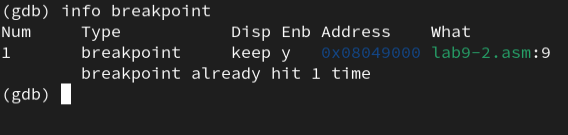
Внешний вид интерфейса

Включим графическое отображение значений регистров (рис. 2.15).



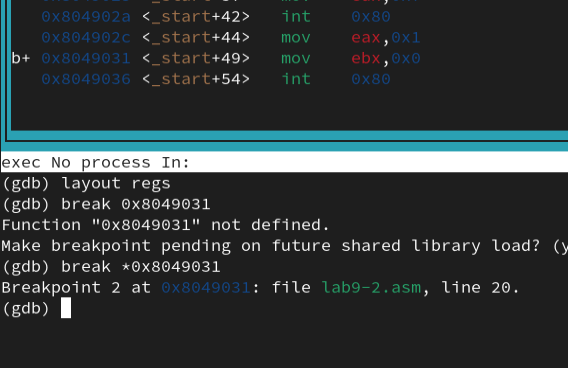
Включение графического отображения значений регистров

Выведем информацию о всех брейкпоинтах (рис. 2.16).



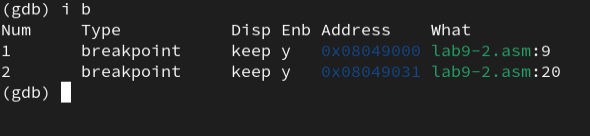
Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу (рис. 2.17).



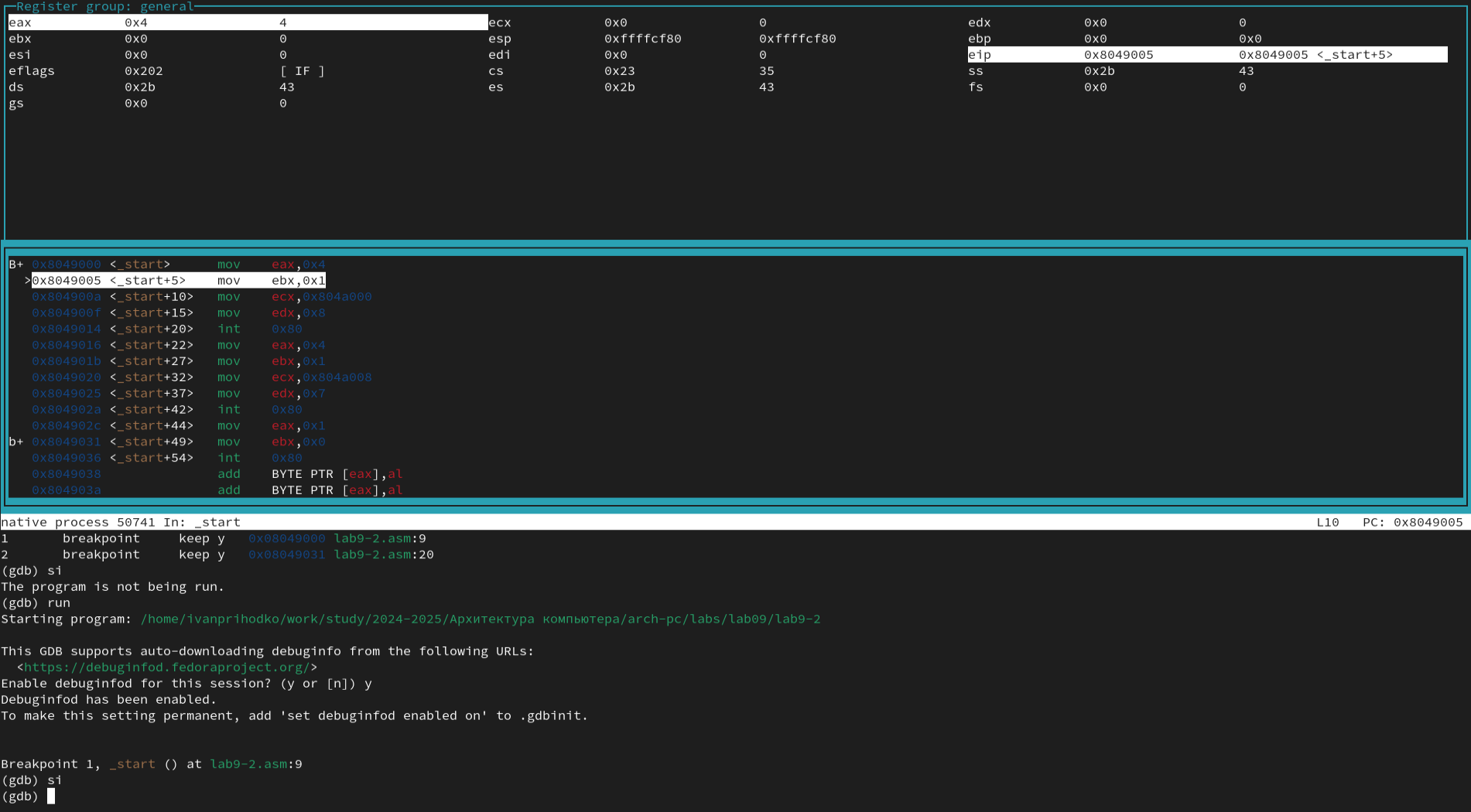
Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах (рис. 2.18).

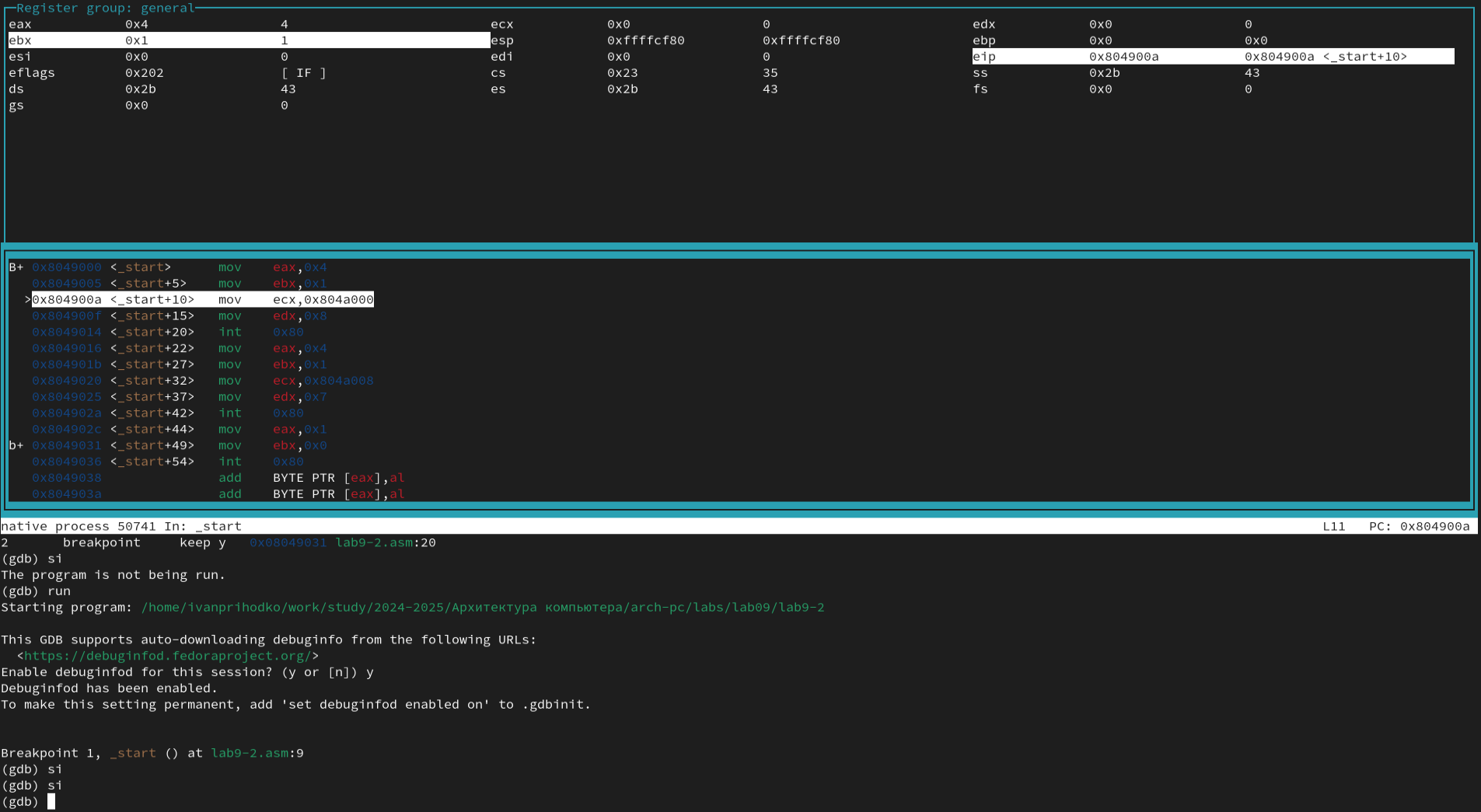


Повторный вывод информации о брейкпоинтах

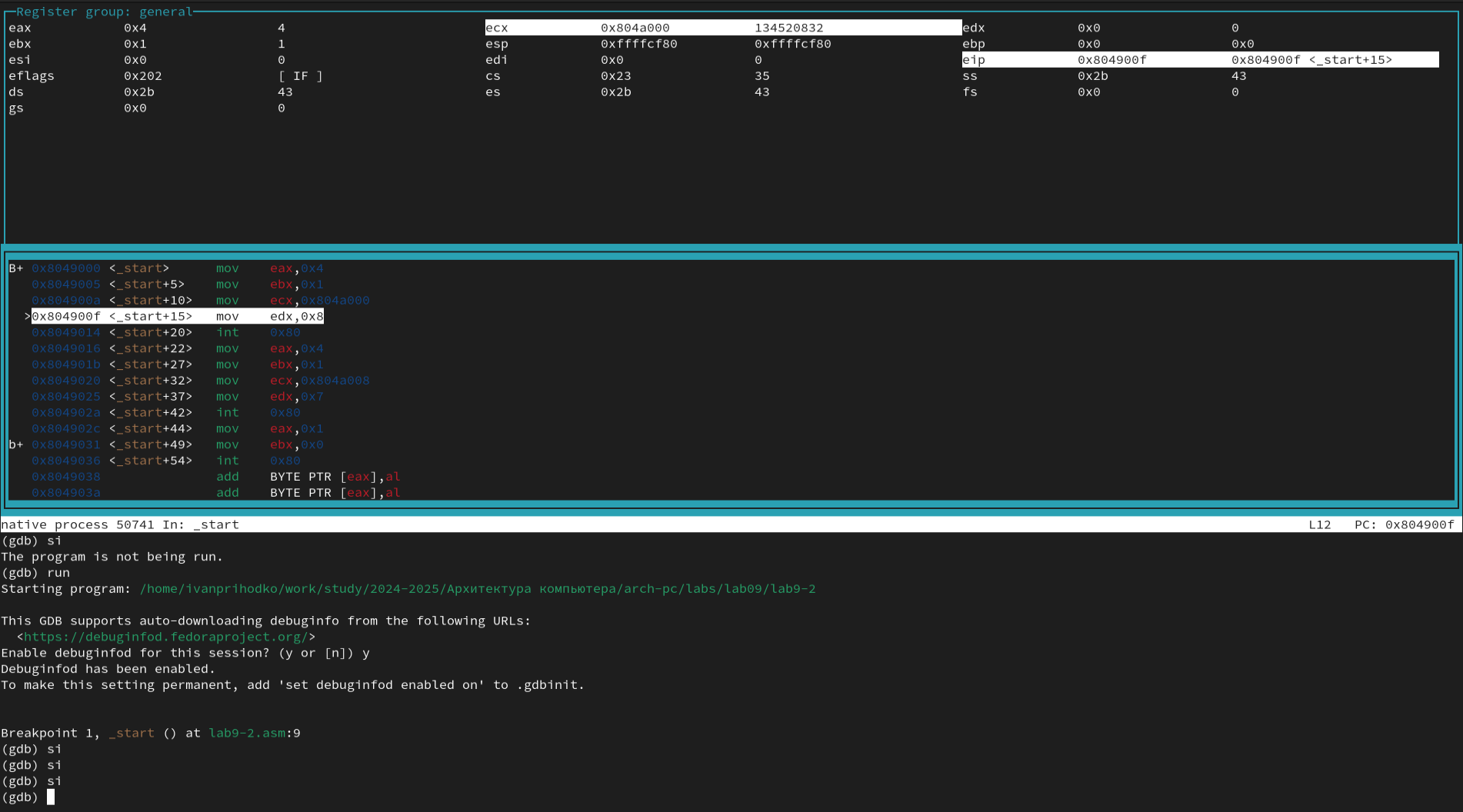
Теперь 5 раз выполним команду si для построчного выполнения кода (рис. 2.19-2.23).



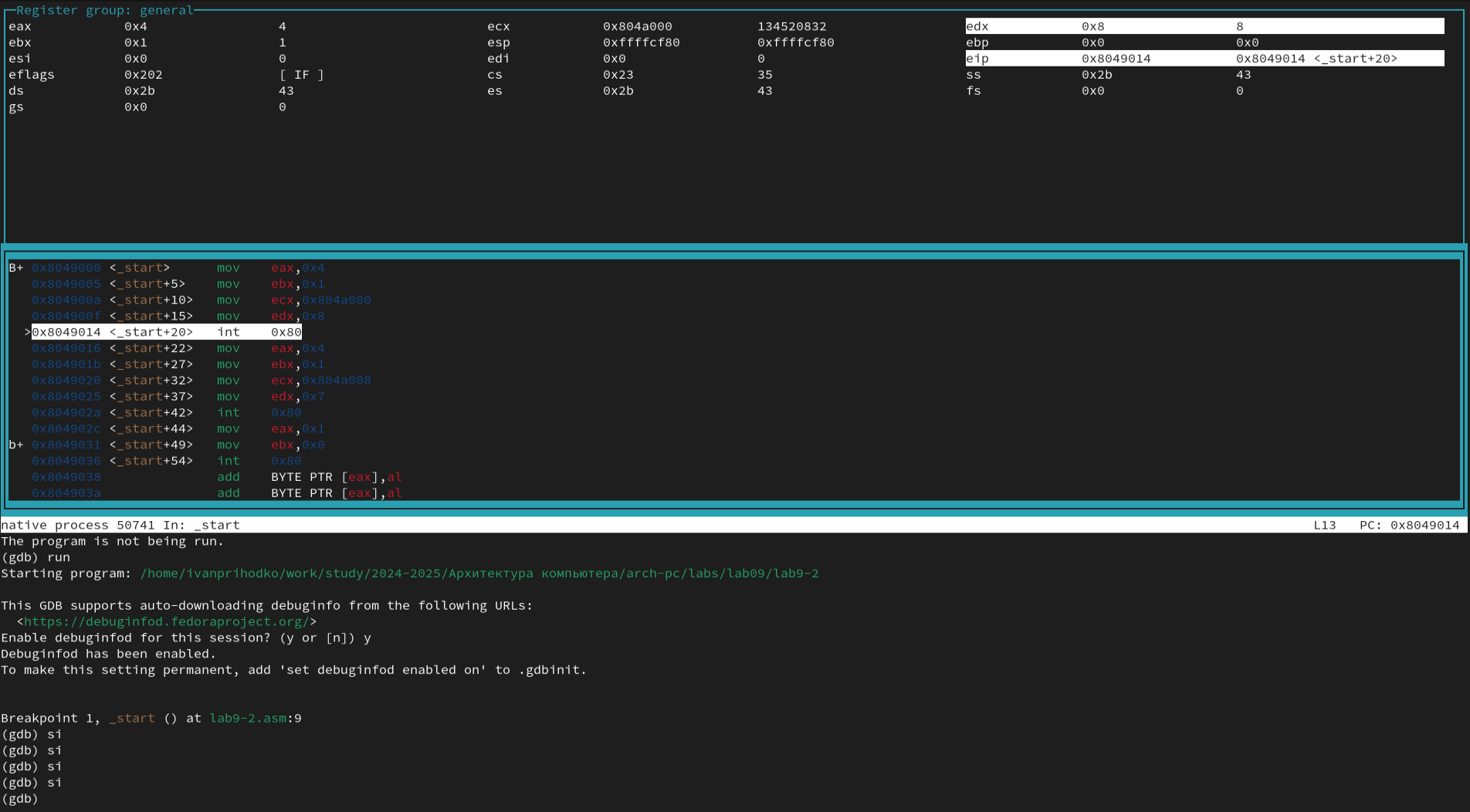
Выполнение команды в коде программы (1)



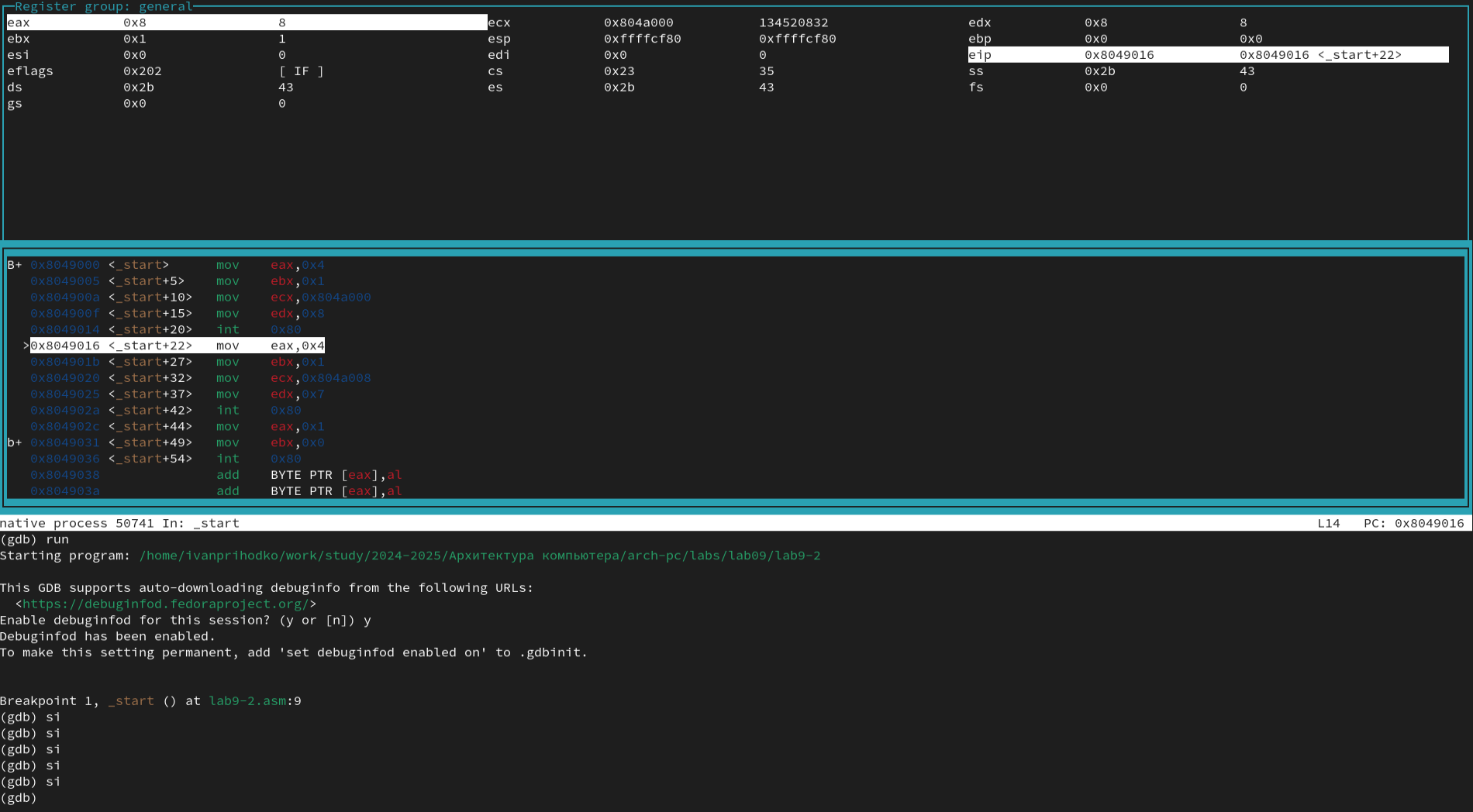
Выполнение команды в коде программы (2)



Выполнение команды в коде программы (3)



Выполнение команды в коде программы (4)

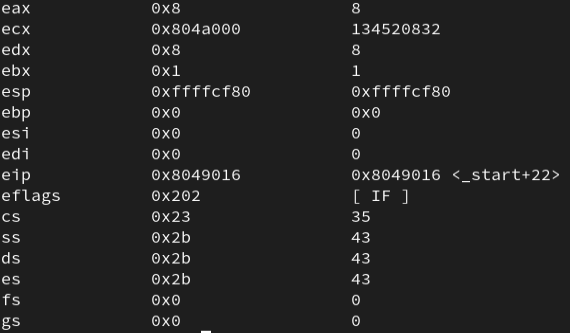


Выполнение команды в коде программы (5)

Как видим, поменялись значения регистров eax, ecx, edx и ebx. Теперь выведем информацию о значениях регистров (рис. 2.24).

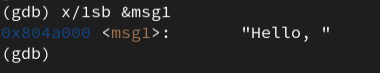
Вывод значений регистров

Вывод значений регистров



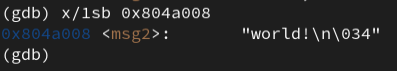
Значения регистров

Попробуем вывести значение переменной по имени (рис. 2.26).



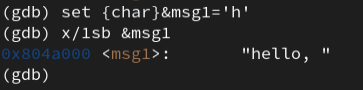
Вывод значения переменной по имени

Теперь попробуем вывести значение переменной по адресу (рис. 2.27).



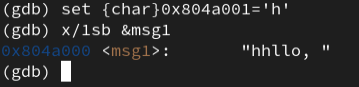
Вывод значения переменной по адресу

Теперь изменим первый символ переменной (рис. 2.28).



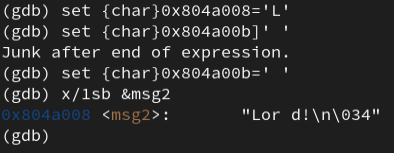
Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратившись по адресу (рис. 2.29).



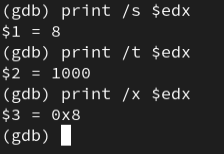
Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной (рис. 2.30).



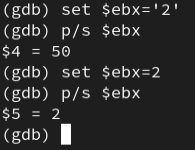
Изменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатиричном виде (рис. 2.31).



Вывод значения регистра в строковом, двоичном и шестнадцатиричном виде

Попробуем теперь изменить значение регистра (рис. 2.32).

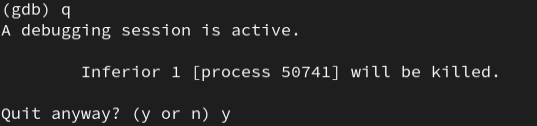


Изменение значения регистра

Как видим, в регистр записались разные значения. Это связано с тем, что в одном случае мы записываем в него число, а в другом случае - строку. Завершим работу программы (рис. 2.33-2.34).

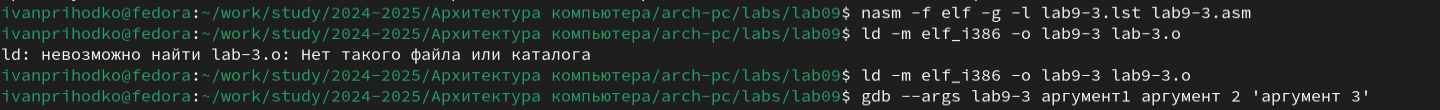


Завершение работы программы



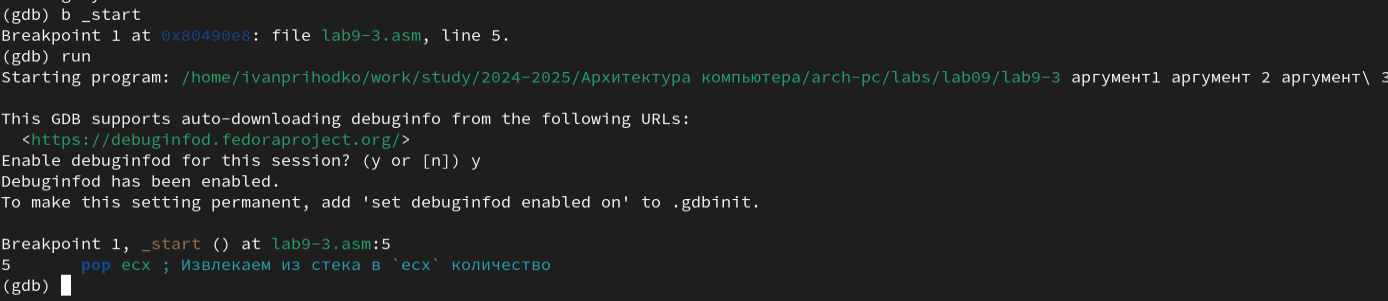
Завершение работы программы

Скопируем файл из прошлой работы, соберём его и выгрузим в gdb (рис. 2.35).



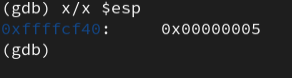
Сборка программы и выгрузка в gdb

Создадим брейкпоинт и запустим программу (рис. 2.36).



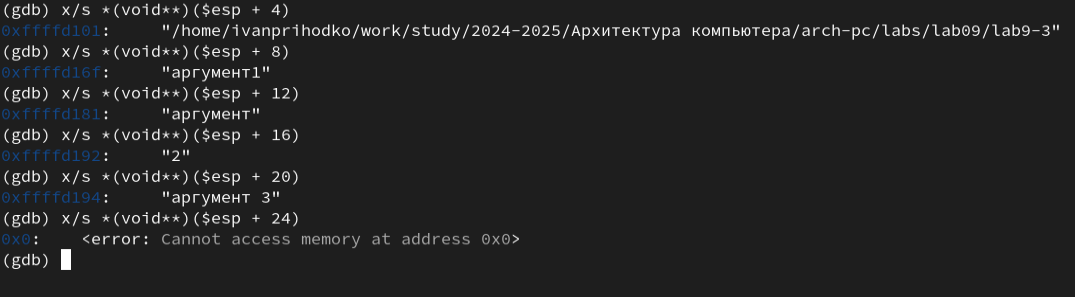
Создание брейкпоинта и запуск программы

Теперь выведем значение регистра esp, где хранятся данные о стеке (рис. 2.37).



Вывод значения регистра esp

Теперь выведем значение всех элементов стека (рис. 2.38).

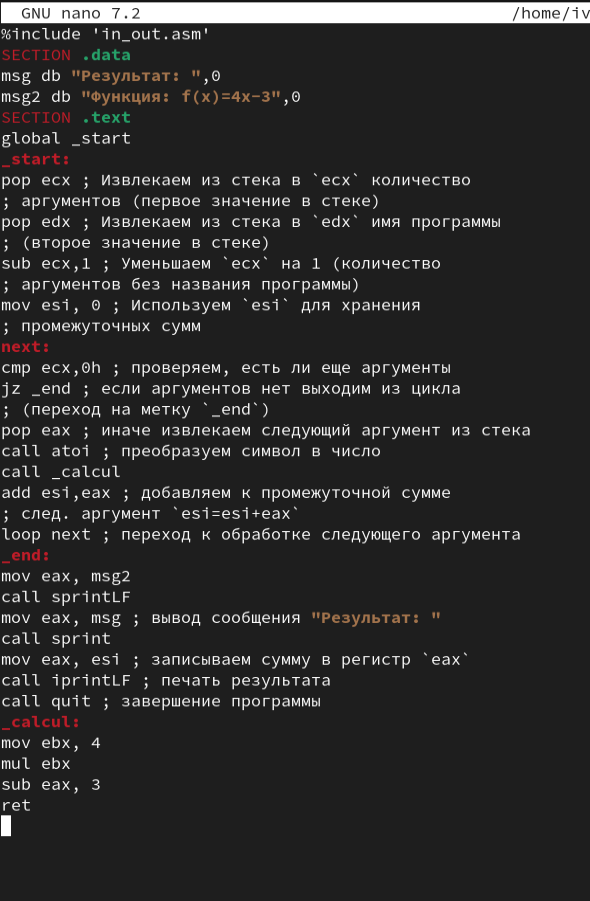


Вывод всех значений в стеке

Как видим, для вывода каждого элемента стека нам нужно менять значение адреса с шагом 4. Это связано с тем, что именно с шагом 4 располагаются данные в стеке, ведь под каждый элемент выделяется 4 байта.

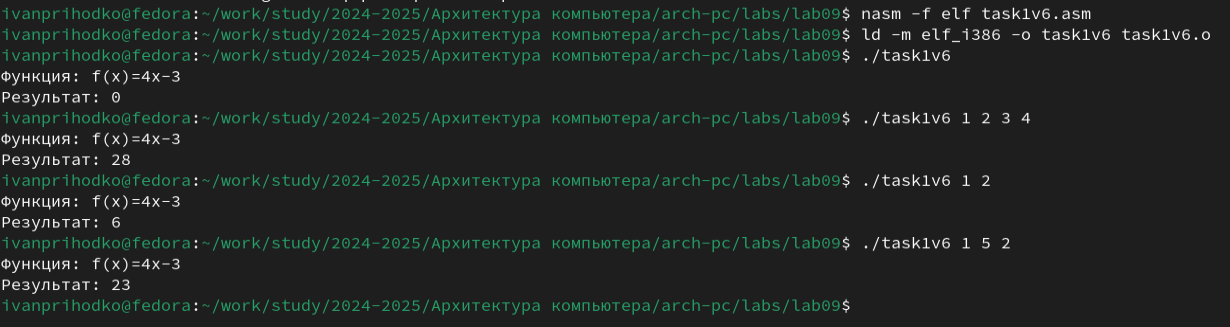
# 3 Выполнение задания для самостоятельной работы

Скопируем файл первого задания прошлой самостоятельной работы и перепишем его так, чтобы он использовал для вычисления выражения подпрограмму (рис. 3.1).



Редактирование кода

Соберем и запустим (рис. 3.2).



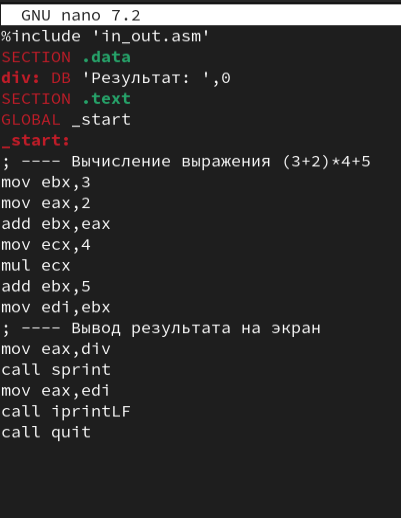
Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы (рис. 3.3).

Создание файла второго задания самостоятельной работы

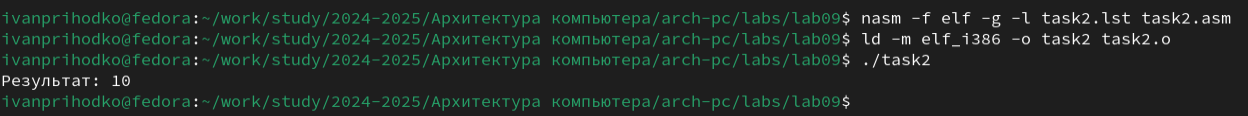
Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 (рис. 3.4).



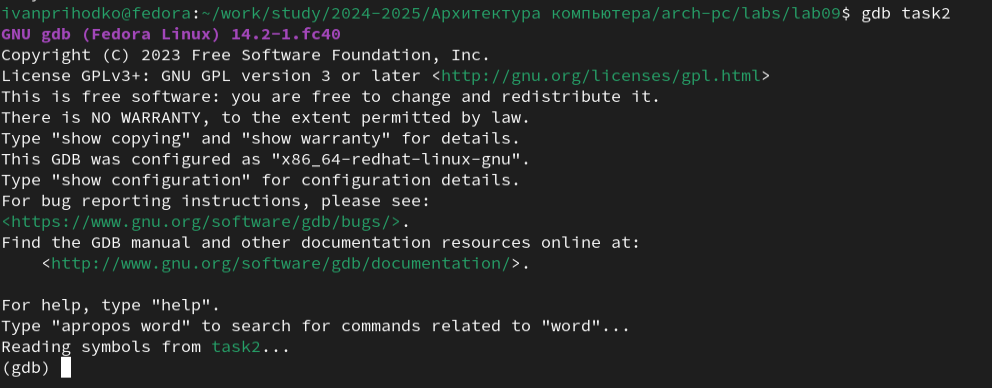
Вставка кода из листинга 9.3

Соберем его и запустим (рис. 3.5).



Сборка и запуск программы

Как видим, код считает значение выражения неправильно. Загрузим его в gdb (рис. 3.6).



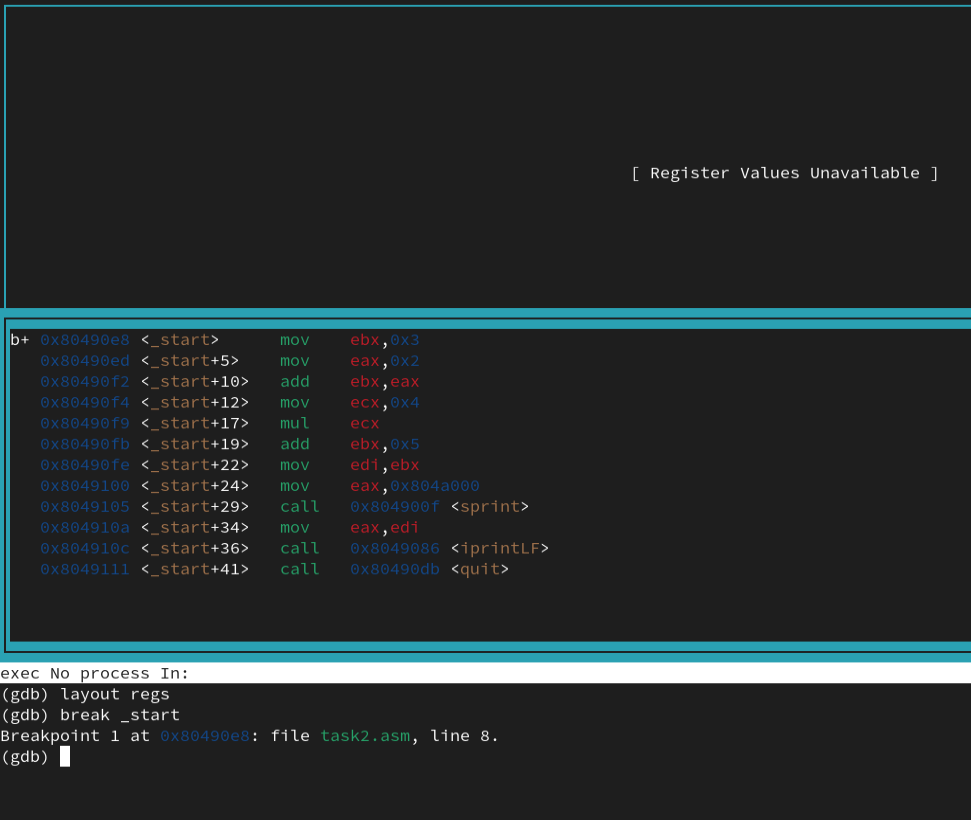
Выгрузка программы в gdb

Переключим его на синтаксис intel (рис. 3.7).

Переключение на синтаксис intel

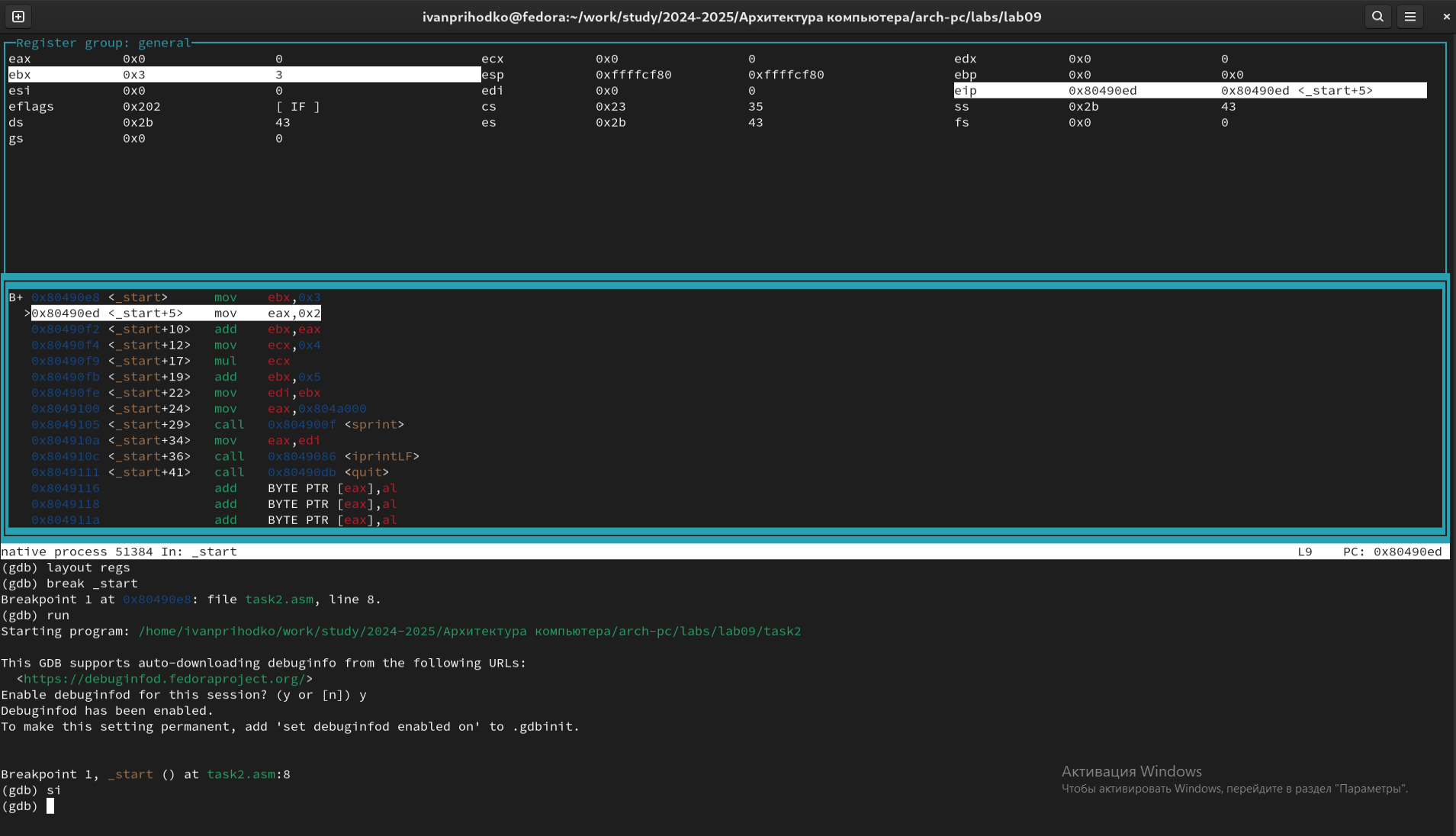
Переключение на синтаксис intel

Включим графическое отображение кода, отображение значений регистров и установим брейкпоинт на \_start (рис. 3.8).

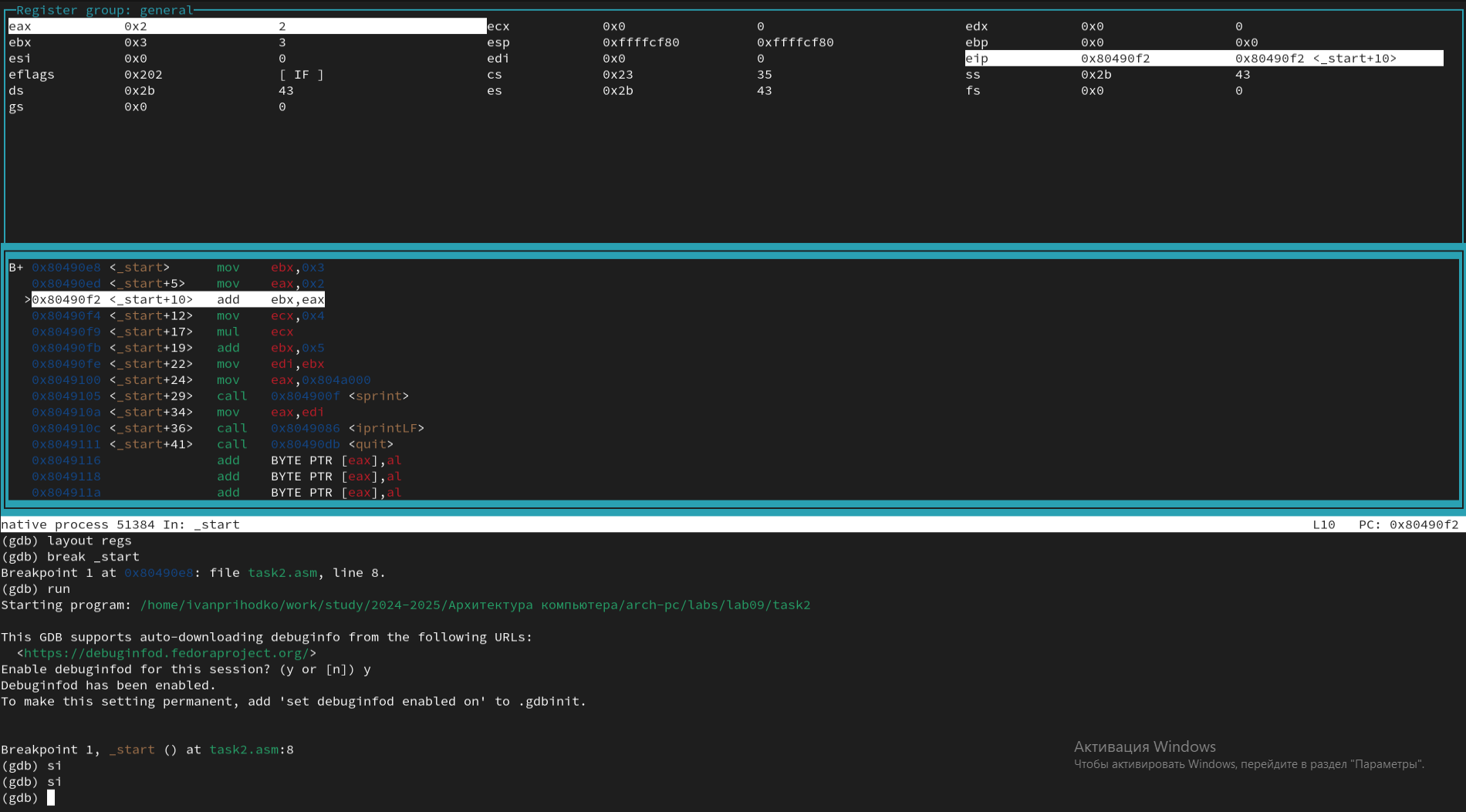


Включение графического отображения кода и выполнения команд

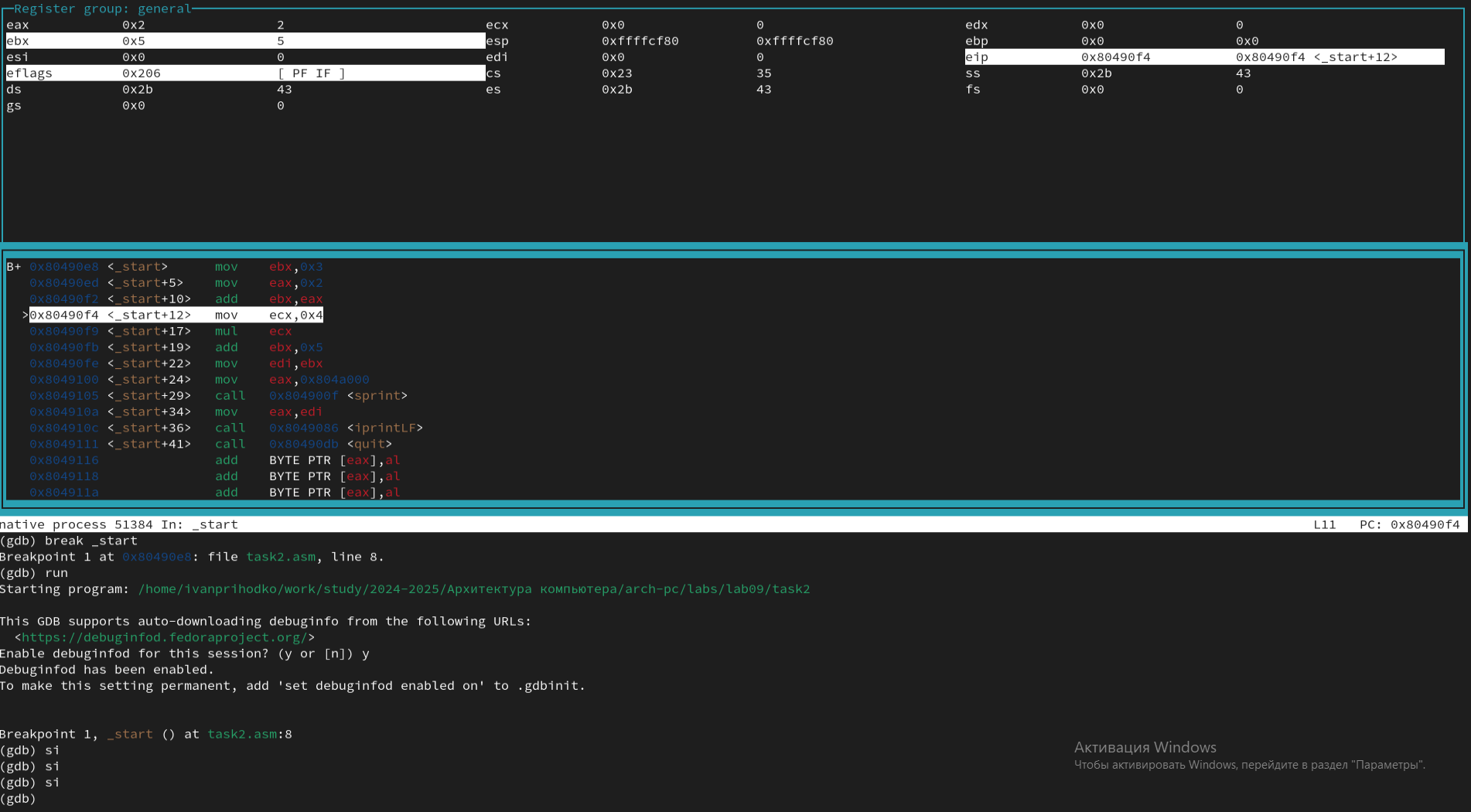
И начнём построчно выполнять код (рис. 3.9 - 3.14).



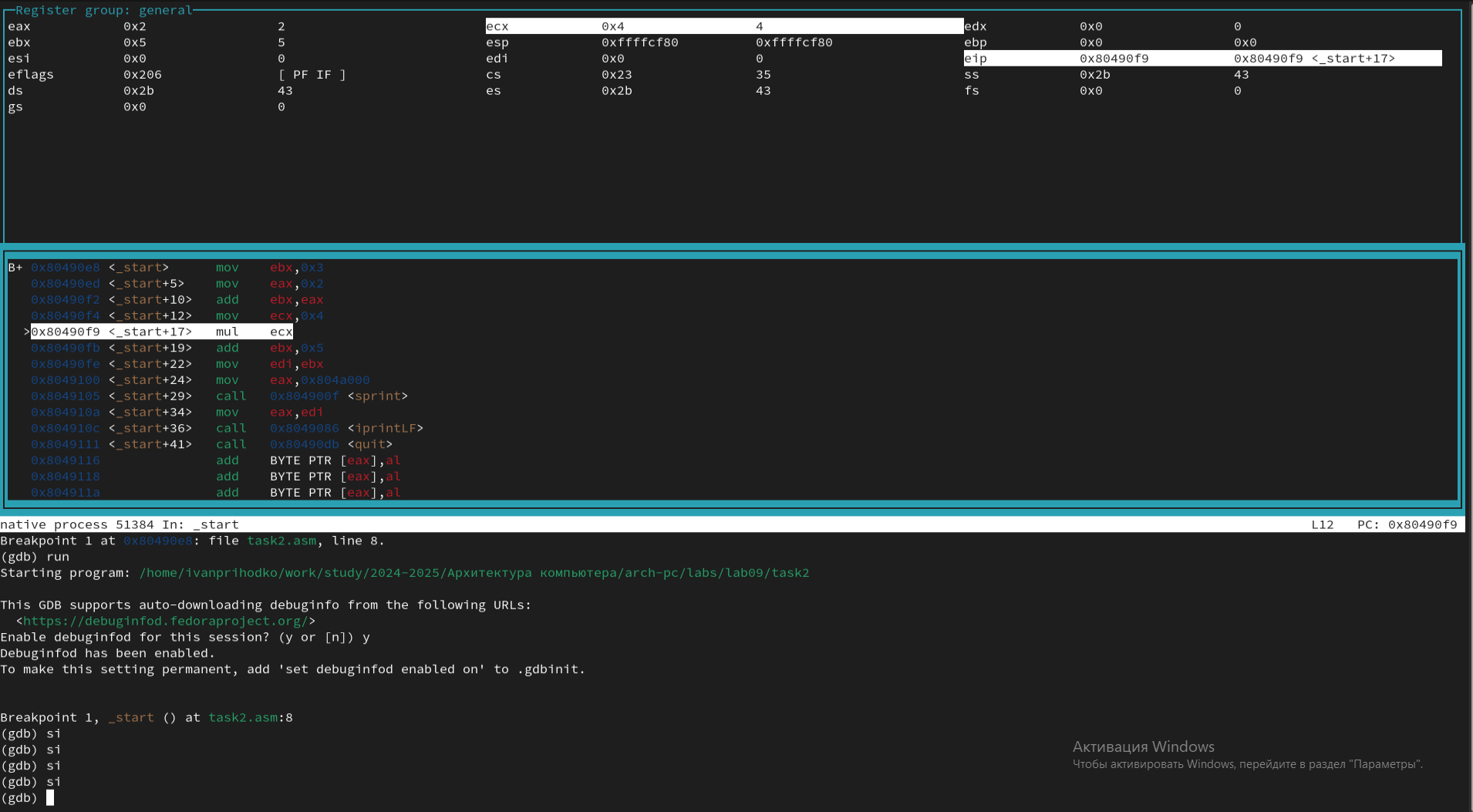
Значение всех регистров на 1 шаге



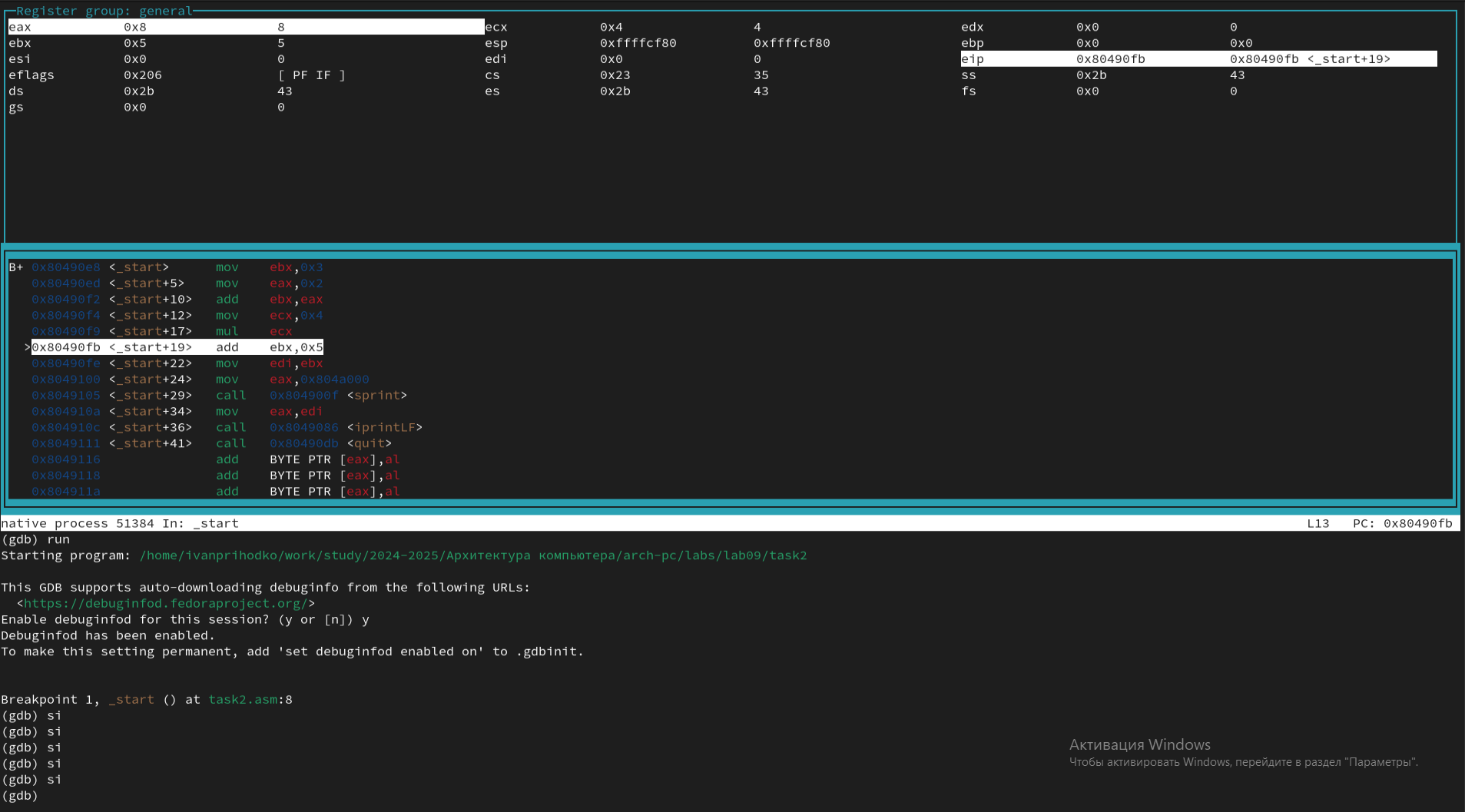
Значение всех регистров на 2 шаге



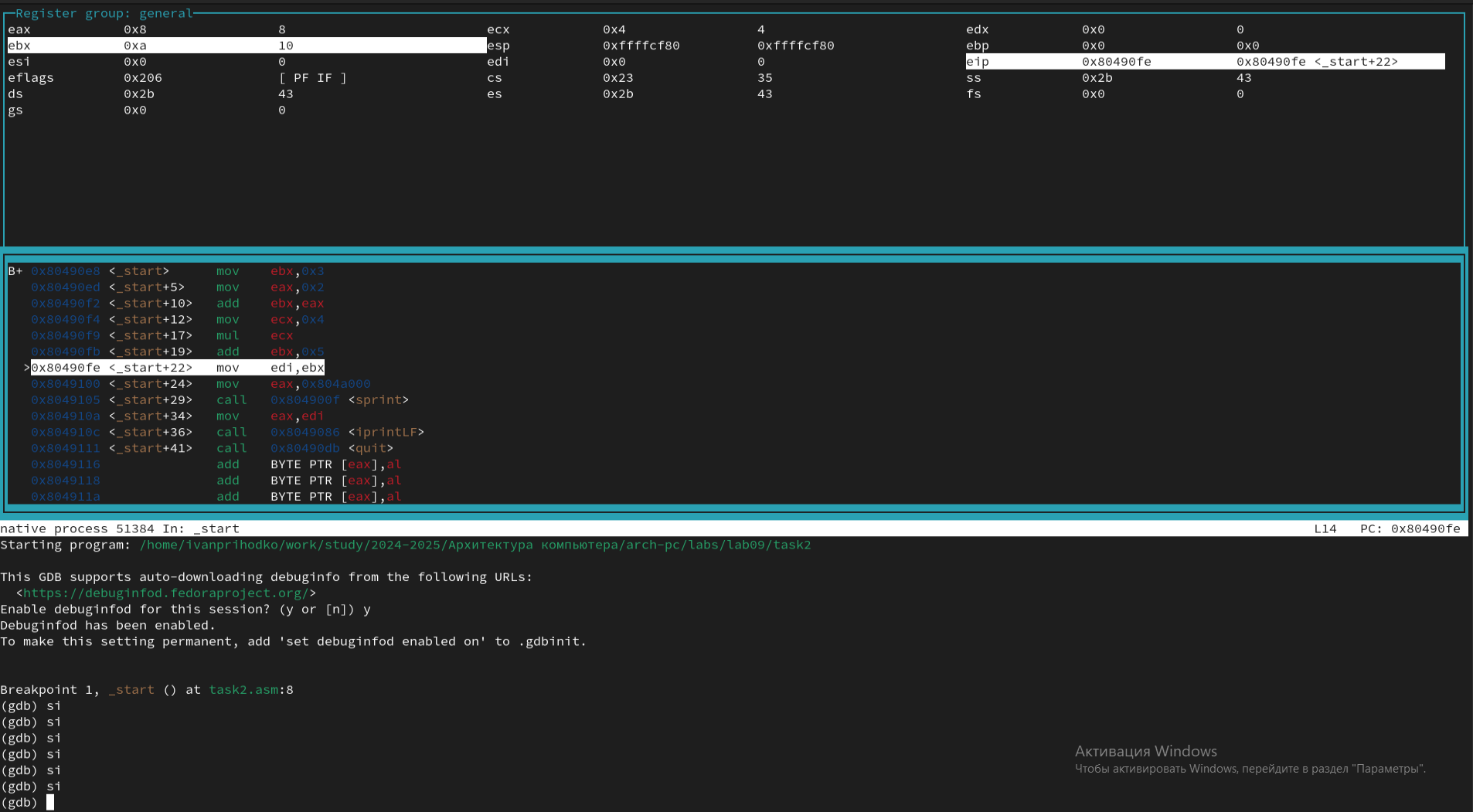
Значение всех регистров на 3 шаге



Значение всех регистров на 4 шаге

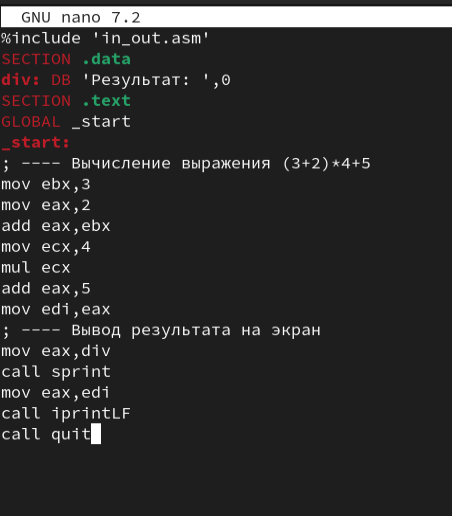


Значение всех регистров на 5 шаге



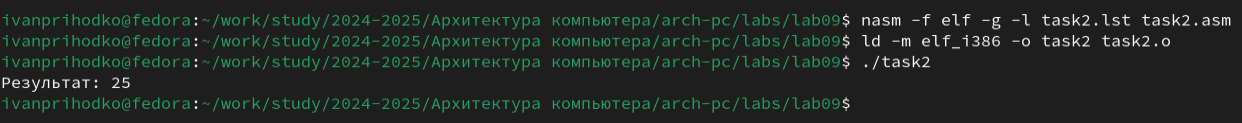
Значение всех регистров на 6 шаге

Как видим, мы должны были умножить значение регистра ebx, но умножили регистр eax. Нам необходимо все результаты хранить в регистре eax. Изменим код (рис. 3.15).



Редактирование кода

Проверим корректность его выполнения (рис. 3.16).



Сборка кода и проверка выполнения

# 4 Выводы

В результате выполнения лабораторной работы были получены представления о работе подпрограмм, а также было реализовано несколько программ, использующих подпрограммы. Также, были получены навыки работы с базовым функионалом gdb, и с помощью gdb была отловлена ошибка в коде программы.