Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Бережной Иван Александрович

Содержание

# 1 Цель работы

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

# 2 Задание

1. Реализация программ в NASM
2. Отладка программам с помощью GDB
3. Задание для самостоятельной работы

# 3 Выполнение лабораторной работы

## 3.1 Реализация программ в NASM

Создадим каталог для дальнейшего выполнения лабораторной работы. Перейдём в него и создадим файл lab09-1.asm (рис. [1](#fig:001)).

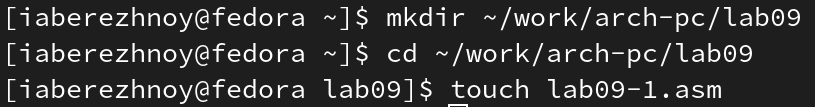


Figure 1: Создание рабочего каталога

Скопируем код из предложенного листинга 9.1. (рис. [2](#fig:002)), создадим исполняемый файл и проверим его работу (рис. [3](#fig:003)).

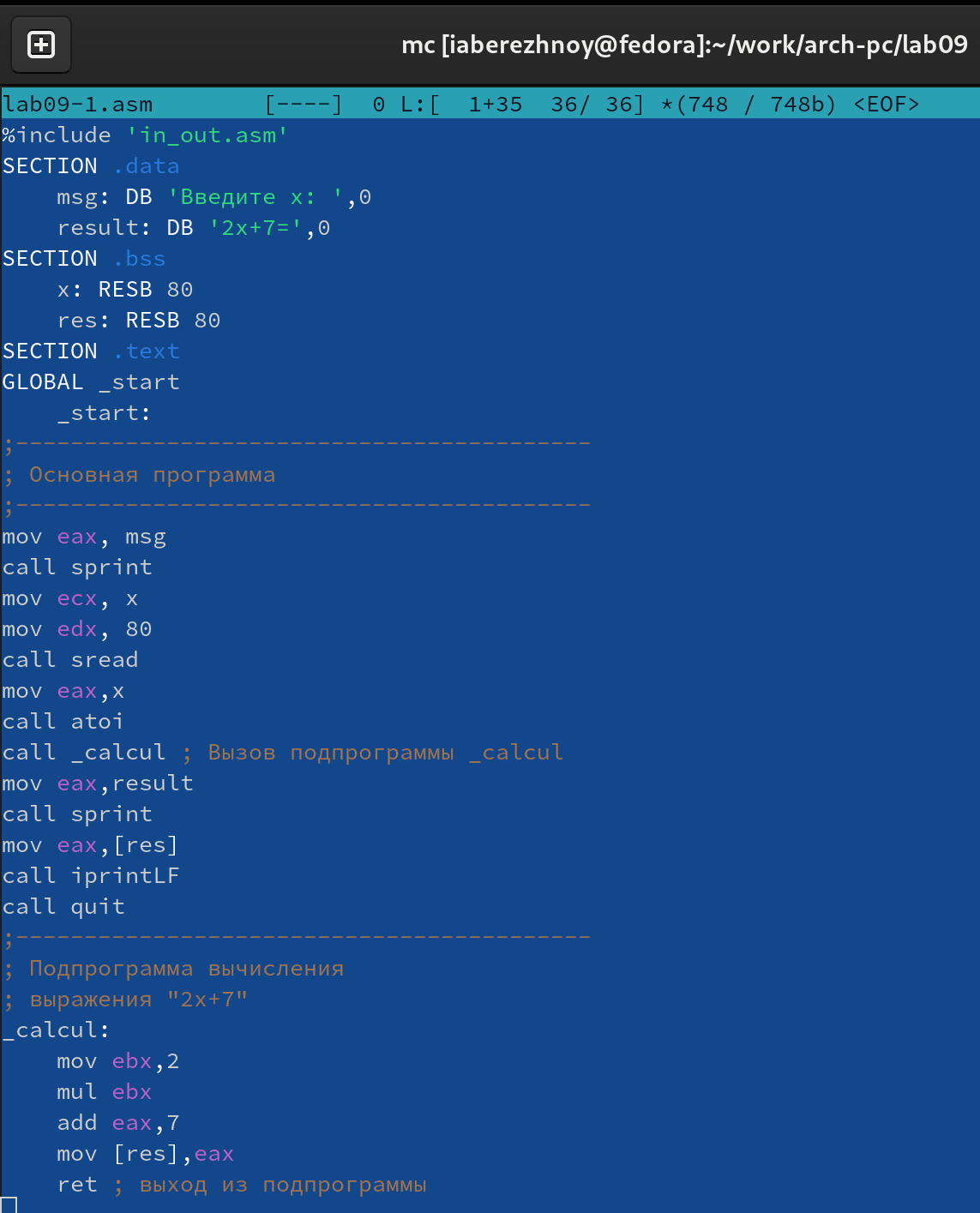


Figure 2: Копирование листинга в lab09-1.asm

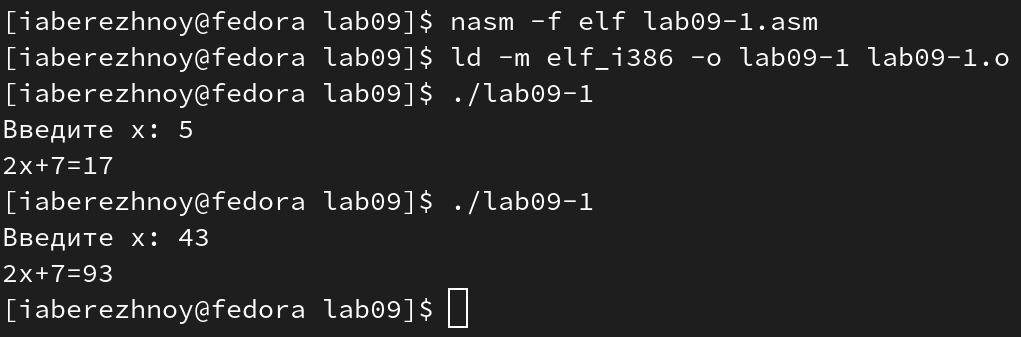


Figure 3: Проверка работы исполняемого файла lab09-1

Изменим текст программы, добавив подпрограмму, которая вычисляет значение выражения , где вводится с клавиатуры (рис. [4](#fig:004)). Создадим исполняемый файл и проверим его работу (рис. [5](#fig:005)).

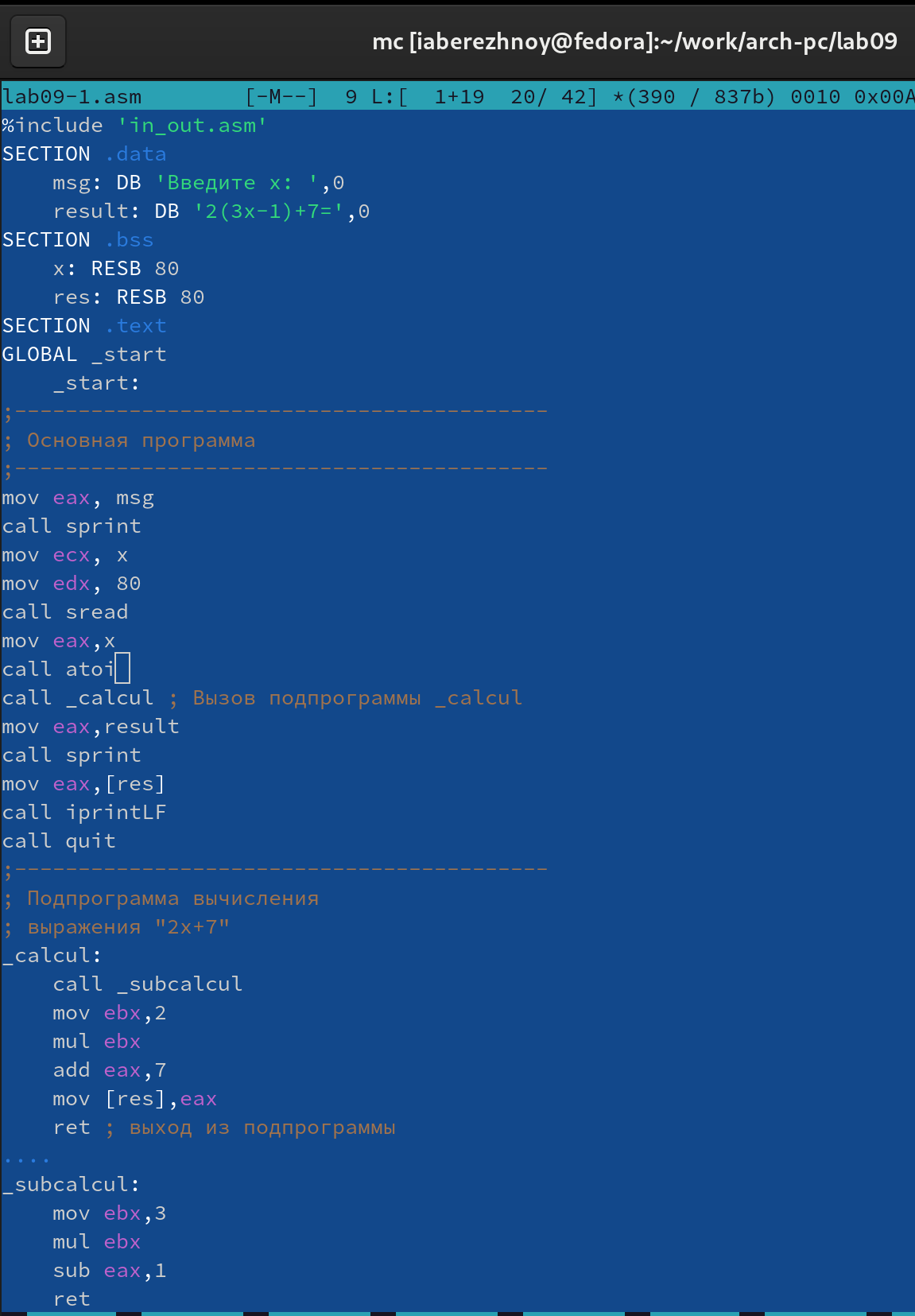


Figure 4: Изменение программы

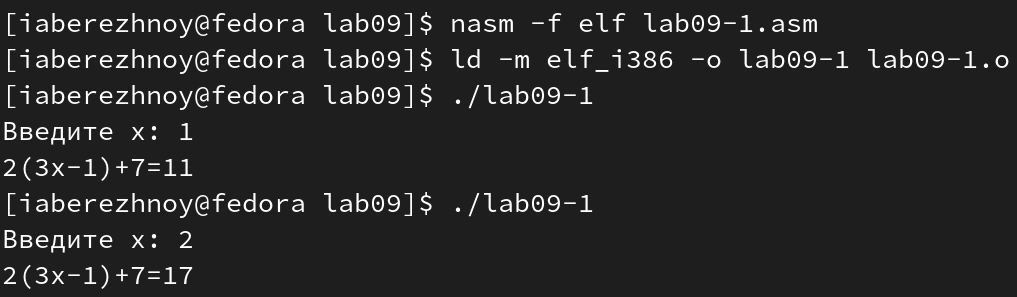


Figure 5: Проверка работы изменённой программы

## 3.2 Отладка программам с помощью GDB

Создадим файл lab09-2.asm и скопируем в него листинг 9.2. (рис. [6](#fig:006)), затем создадим исполняемый файл, загрузим его в отладчик GDB и проверим его работу (рис. [7](#fig:007)).

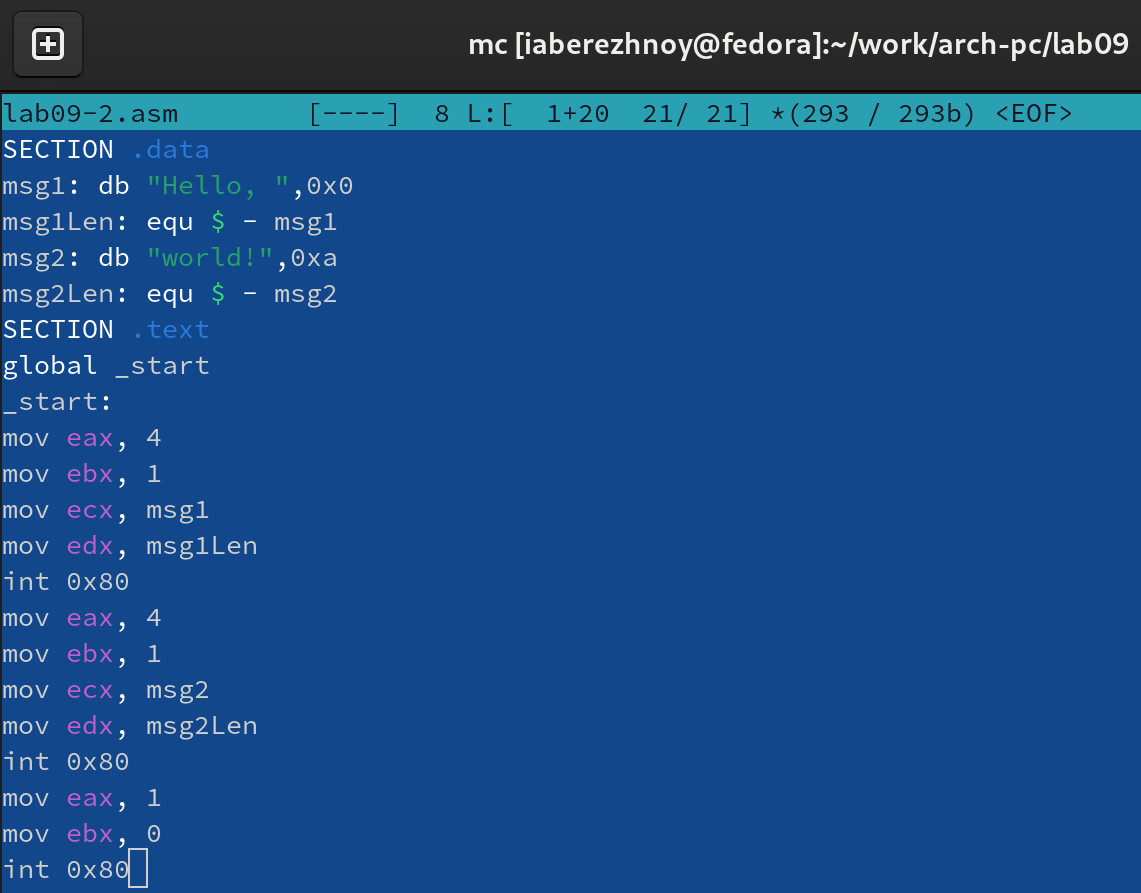


Figure 6: Копирование листинга в lab09-2.asm

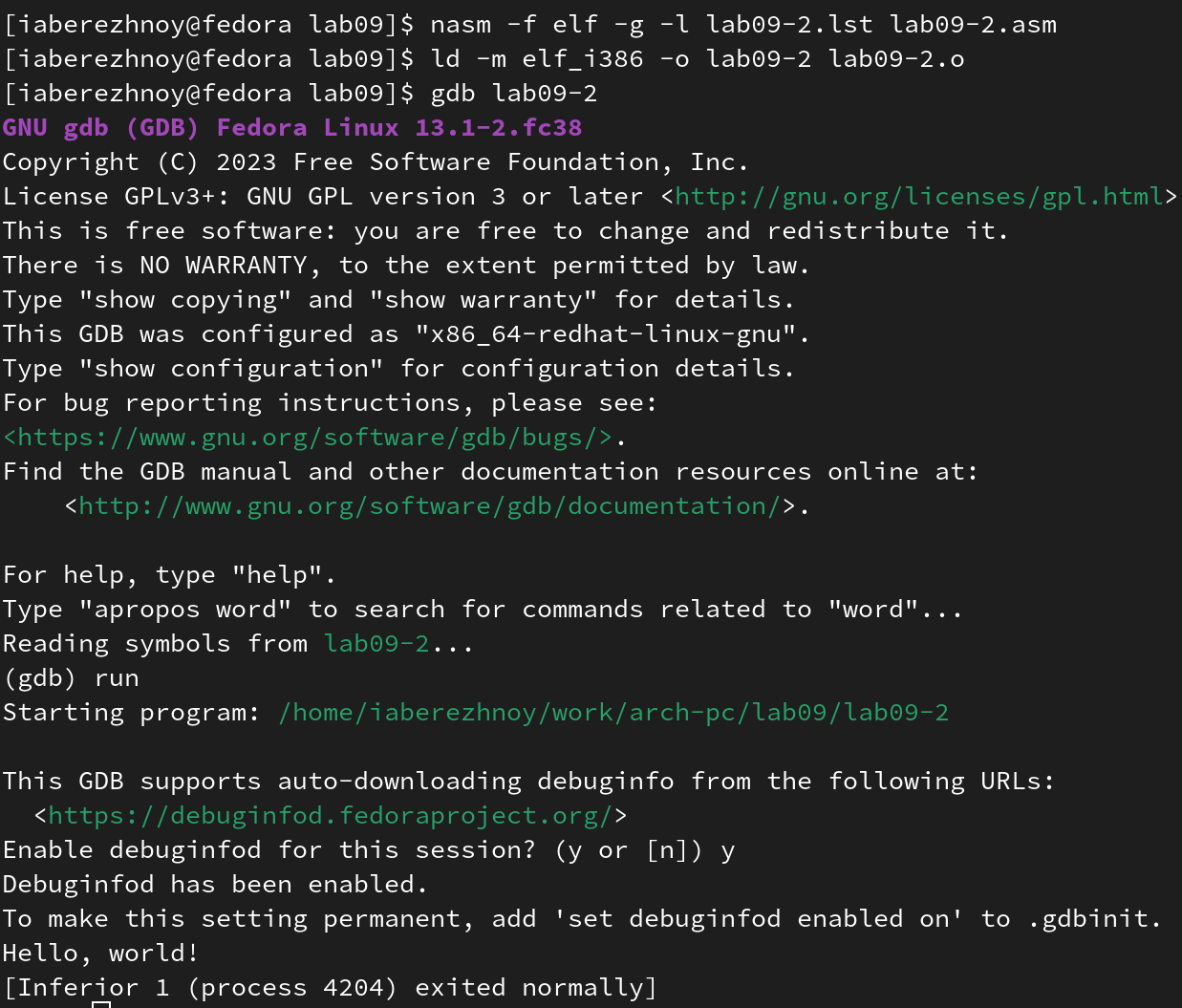


Figure 7: Загрузка исполняемого файла в отладчик

Установим точку останова командой break \_start и запустим программу (рис. [8](#fig:008)).

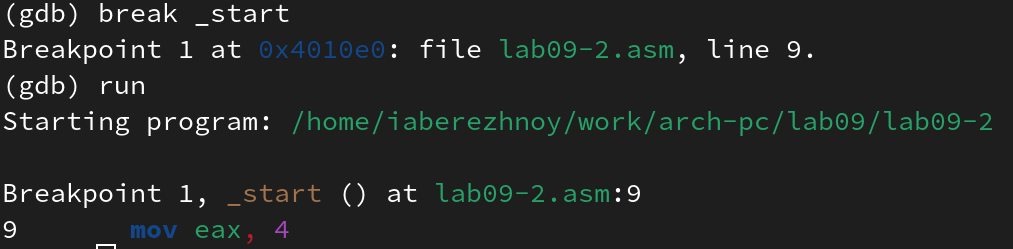


Figure 8: Установка брейкпоинта \_start

Посмотрим на дисассимилированный код программы командой disassemble, начиная с брейкпоинта. Теперь переключим отображение команд на Intel’овский синтаксис с помощью команды set disassembly-flavour intel, посмотрим и на это отображение (рис. [9](#fig:009)). Можем заметить различия: 1. В отображении ATT имена регистров начинаются с символов % или $, в отличии от отображения Intel, где перед регистрами символов нет; 2. В отображении ATT другой порядок регистров, нежели в отображении Intel.

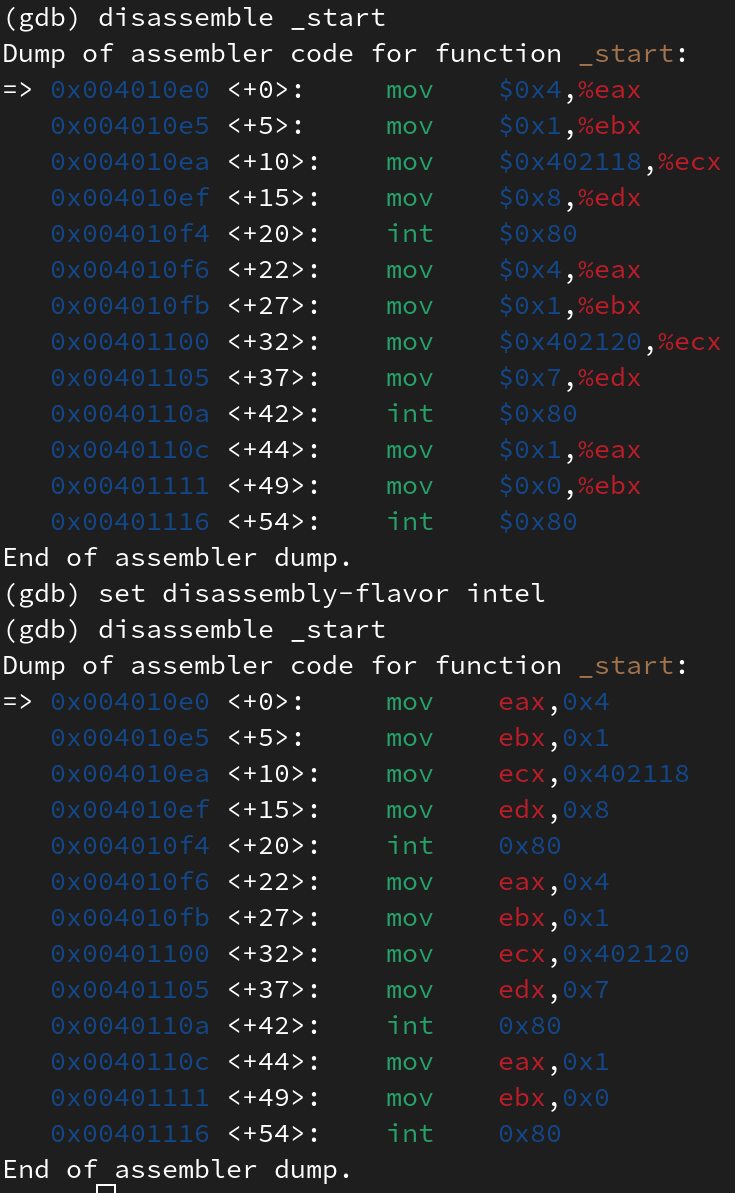


Figure 9: Сравнение отображений команд

Включим режим псевдографики для удобного анализа программы командами layout asm и layout regs (рис. [10](#fig:010)).

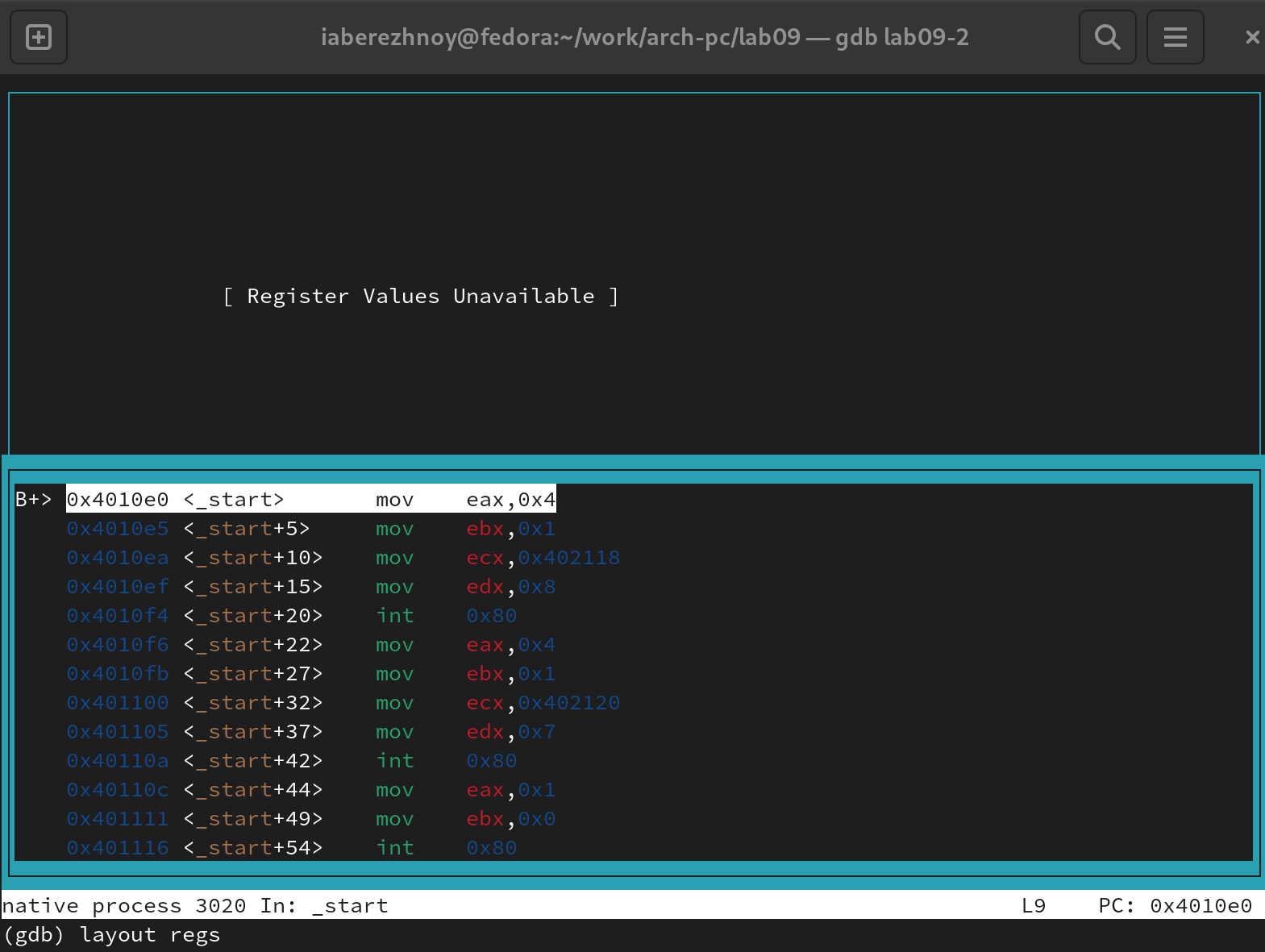


Figure 10: Включение режима псевдографики

Посмотрим, числится ли уже установленный нами брейкпоинт в списке точек останова командой info breakpoints (рис. [11](#fig:011)). Установим вторую точку останова по адресу инструкции. Для этого пишем команду break \* и после символа звёздочки без пробела вводим адрес нужной нам инструкции (а именно mov ebx,0x0). Снова посмотрим список точек останова (рис. [12](#fig:012)).

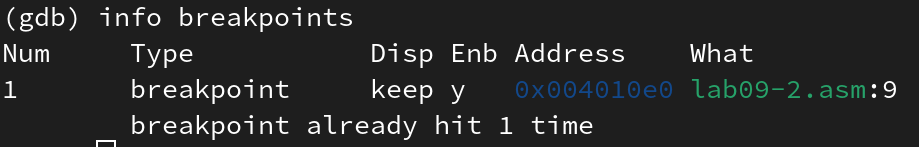


Figure 11: Просмотр списка точек останова

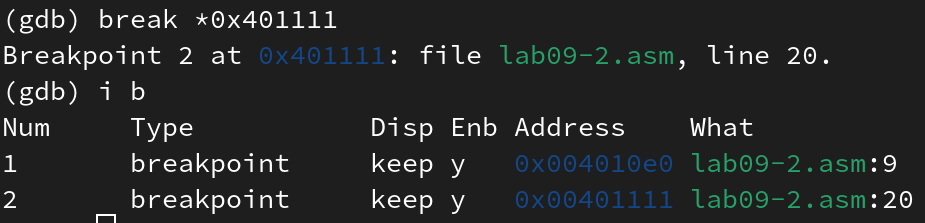


Figure 12: Установка брейкпоинта по адресу инструкции

Выполним команду si 5, которая позволит пошагово выполнять инструкции. Теперь посмотрим, какие регистры были изменены, сравнив их начальные значения (рис. [13](#fig:013)) с новыми (рис. [14](#fig:014)). Были изменены значения в регистрах eax, ebx, ecx, edx, eip, cs, ds, ss, eflags и es.

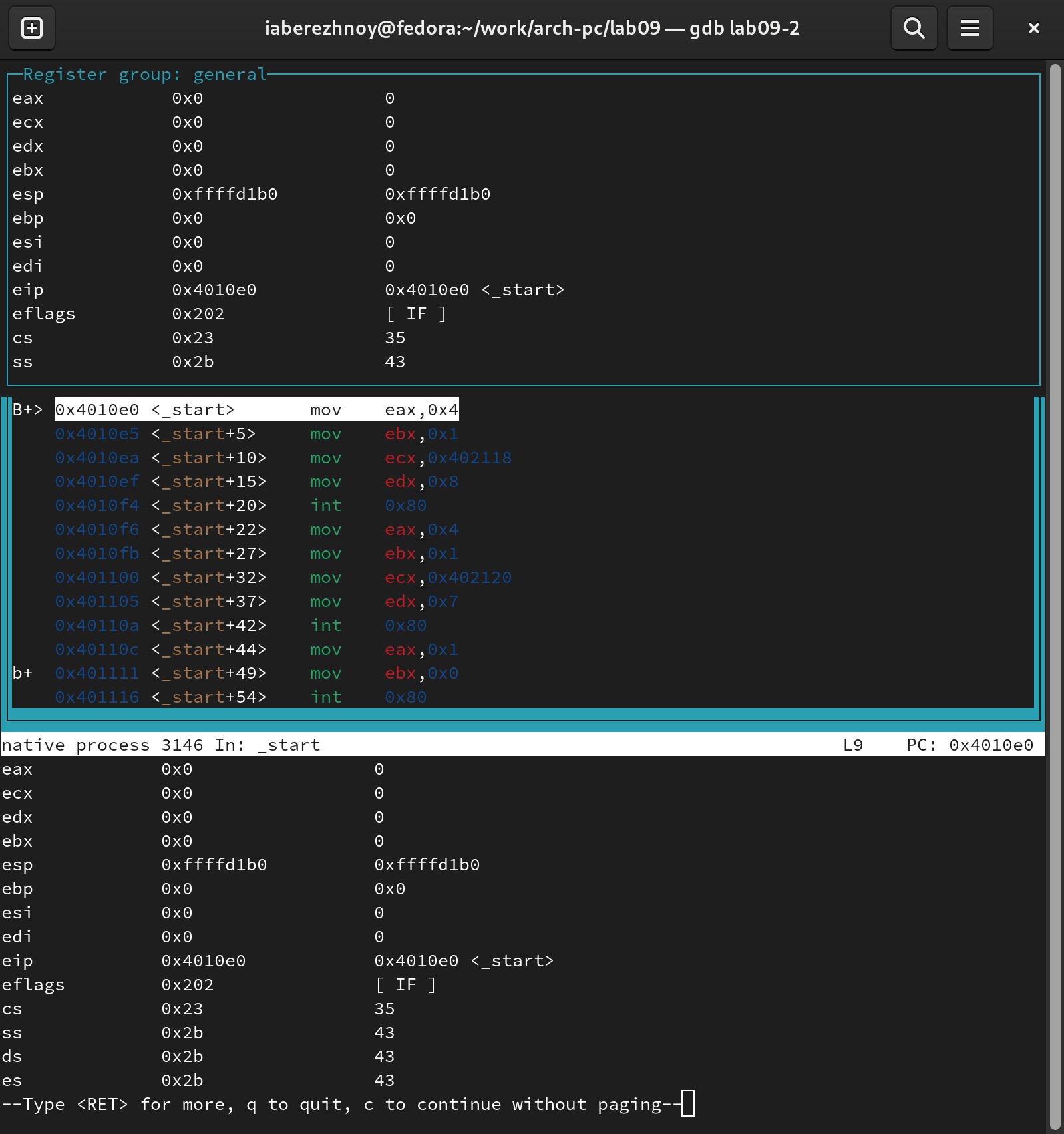


Figure 13: Начальные значения регистров

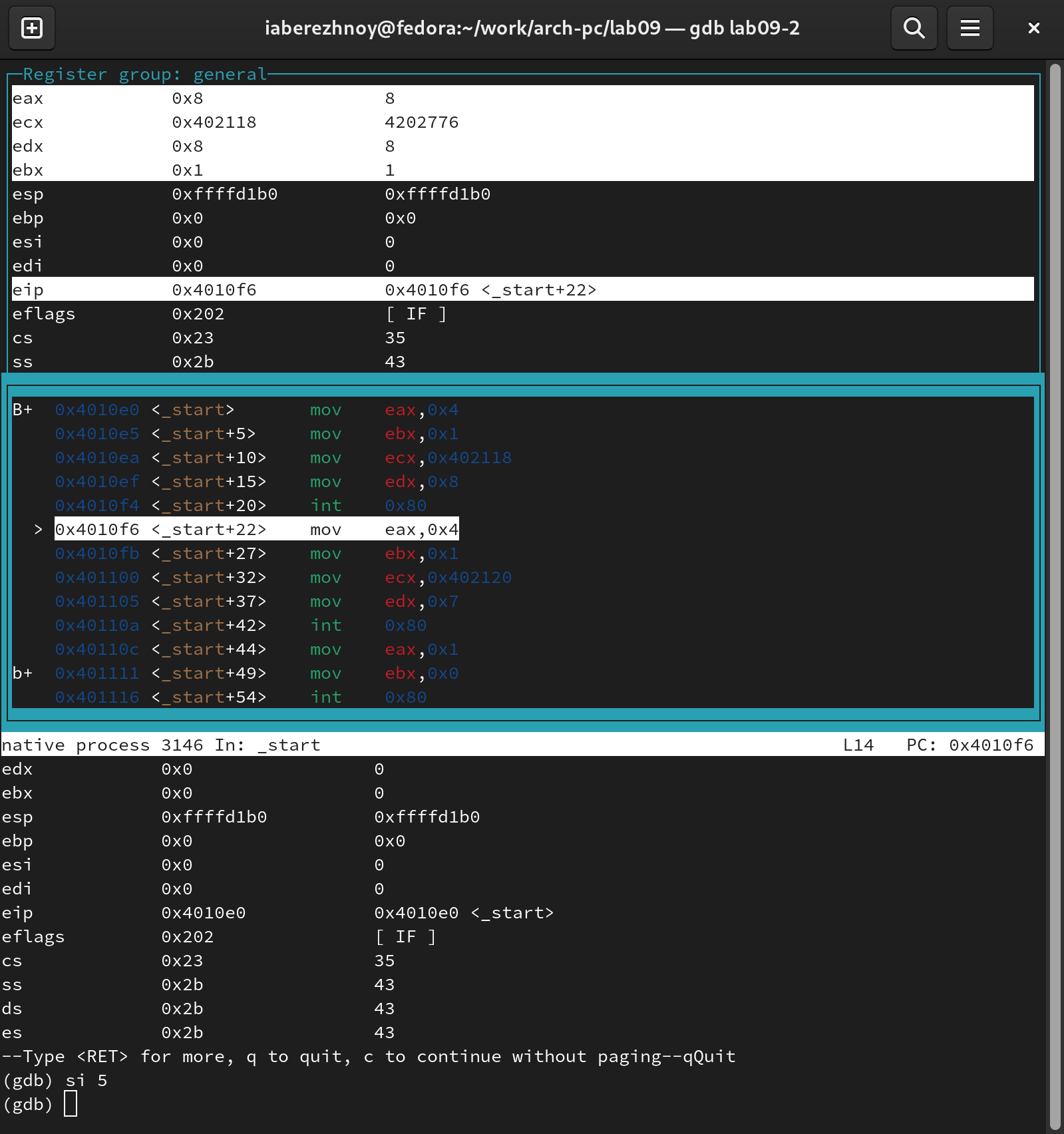


Figure 14: Новые значения регистров

Посмотрим значение переменной msg1 по имени с помощью команды x/1sb &msg1. Сделаем то же самое для переменной msg2, но уже по адресу (рис. [15](#fig:015)). Изменим первый символ переменной msg1 командой `set {char}msg1=‘h’ и посмотрим, сработали изменения. Также изменим первый символ и во второй переменной (рис. [16](#fig:016)).

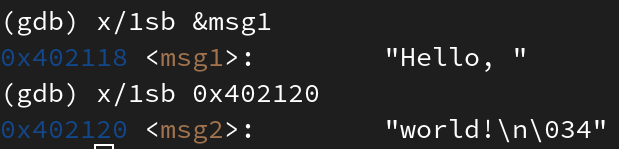


Figure 15: Просмотр содержимого переменных

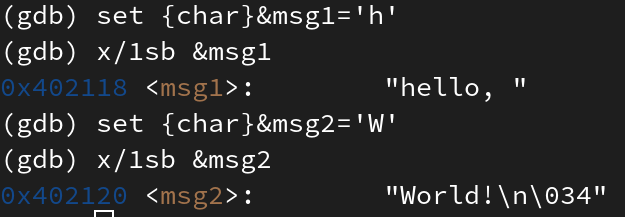


Figure 16: Замена символов в переменных

Выведем значение регистра edx в различных форматах командами p/F $<регистр> (рис. [17](#fig:017)).

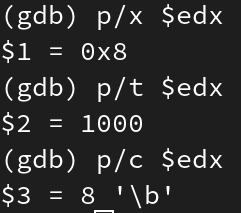


Figure 17: Вывод значения регистра

Командой set $ebx=<value> изменим значение регистра ebx на ‘2’, а затем на 2. Проверим значение после каждого его изменения (рис. [18](#fig:018)). Значения различаются, хотя мы ввели одинаковые числа. Тем не менее, в первом случае мы вводили число как символ, соответственно в качестве значения видим код этого числа (50). Во втором же случае мы вводим число как числовой тип данных, поэтому и получили то же самое число.

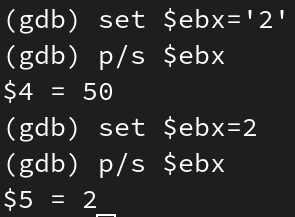


Figure 18: Изменение значения регистра

Завершим выполнение программы и выйдем из отладчика GDB.

Создадим файл lab09-3.asm и скопируем в него код из файла lab8-2.asm. Создадим исполняемый файл (рис. [19](#fig:019)). Теперь загрузим исполняемый файл в отладчик, попутно указав нужные аргументы (для этого также нужно использовать ключ --args (рис. [20](#fig:020)).

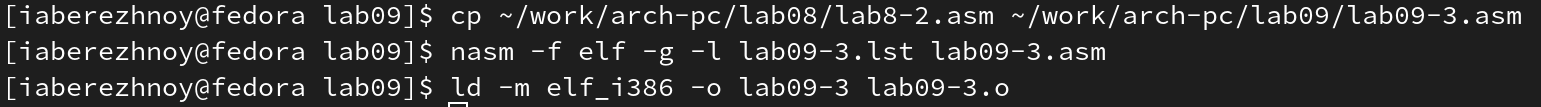


Figure 19: Создание файла lab09-3.asm

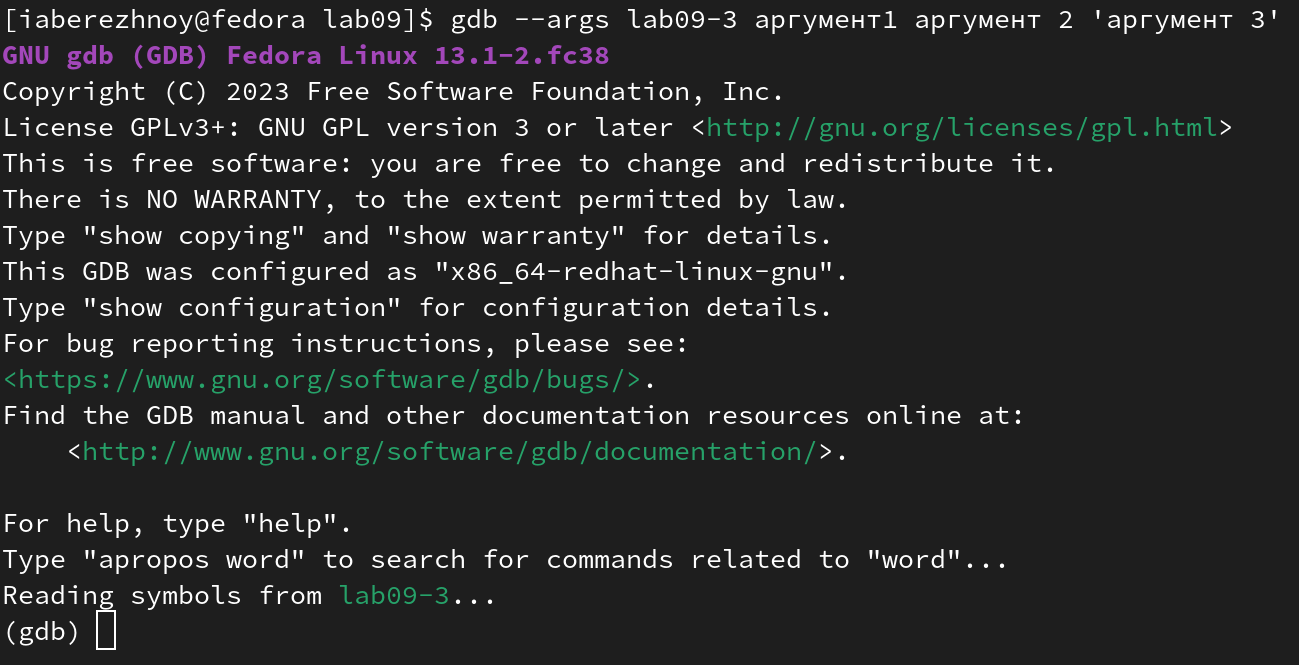


Figure 20: Загрузка исполняемого файла в отладчик с аргументами

Установим брейкпоинт и запустим программу (рис. [21](#fig:021)). Посмотрим позиции стека через регистр esp командой x/s \*(void\*\*)($esp + <value>) (рис. [22](#fig:022)). Шаг изменения равен 4, т.к. шаг - int, под который выделяется 4 байта памяти.

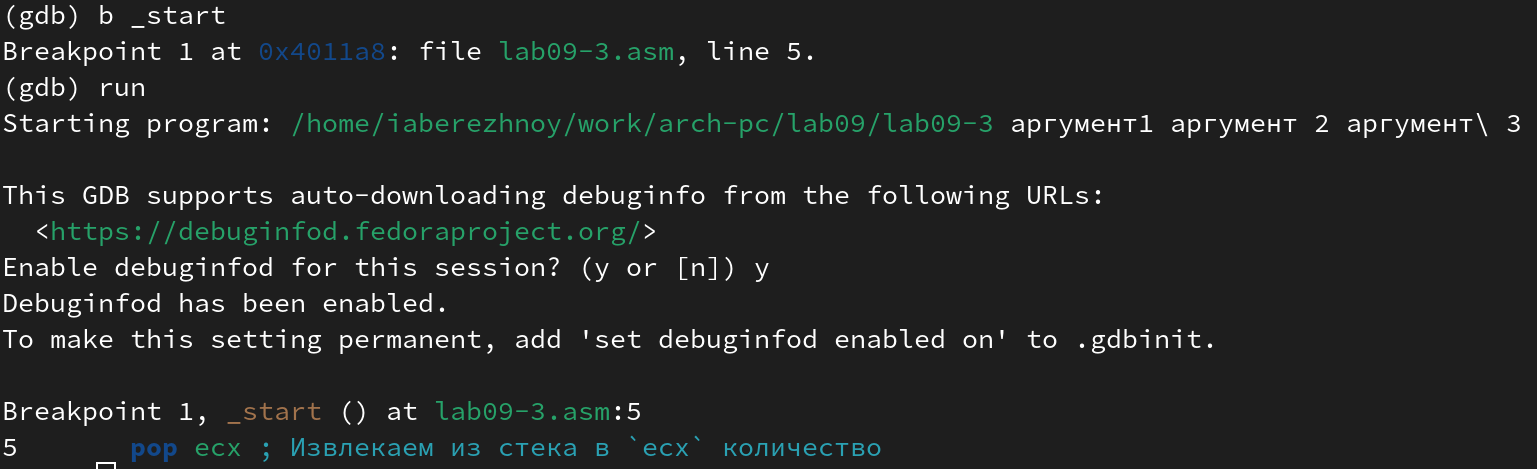


Figure 21: Установка брейкпоинта

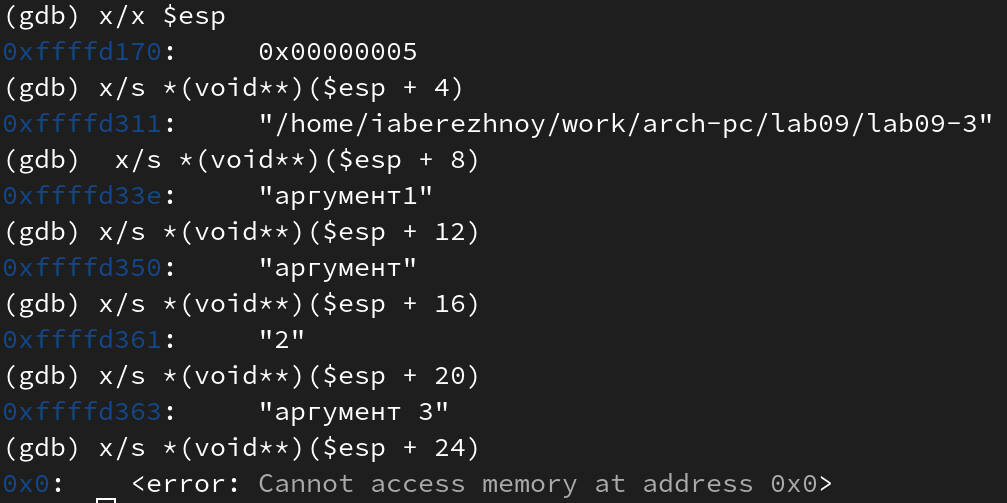


Figure 22: Просмотр позиций стека

## 3.3 Задание для самостоятельной работы

1. Скопируем код программы из задания для самостоятельной работы (лабораторная работы №8) в файл lab09-4.asm. Перепишем код так, чтобы вычисление функции происходило в подпрограмме (функция соответствует варианту 2 из лабораторной работы №8) (рис. [23](#fig:023)). Создадим исполняемый файл и проверим его работу (рис. [24](#fig:024)).

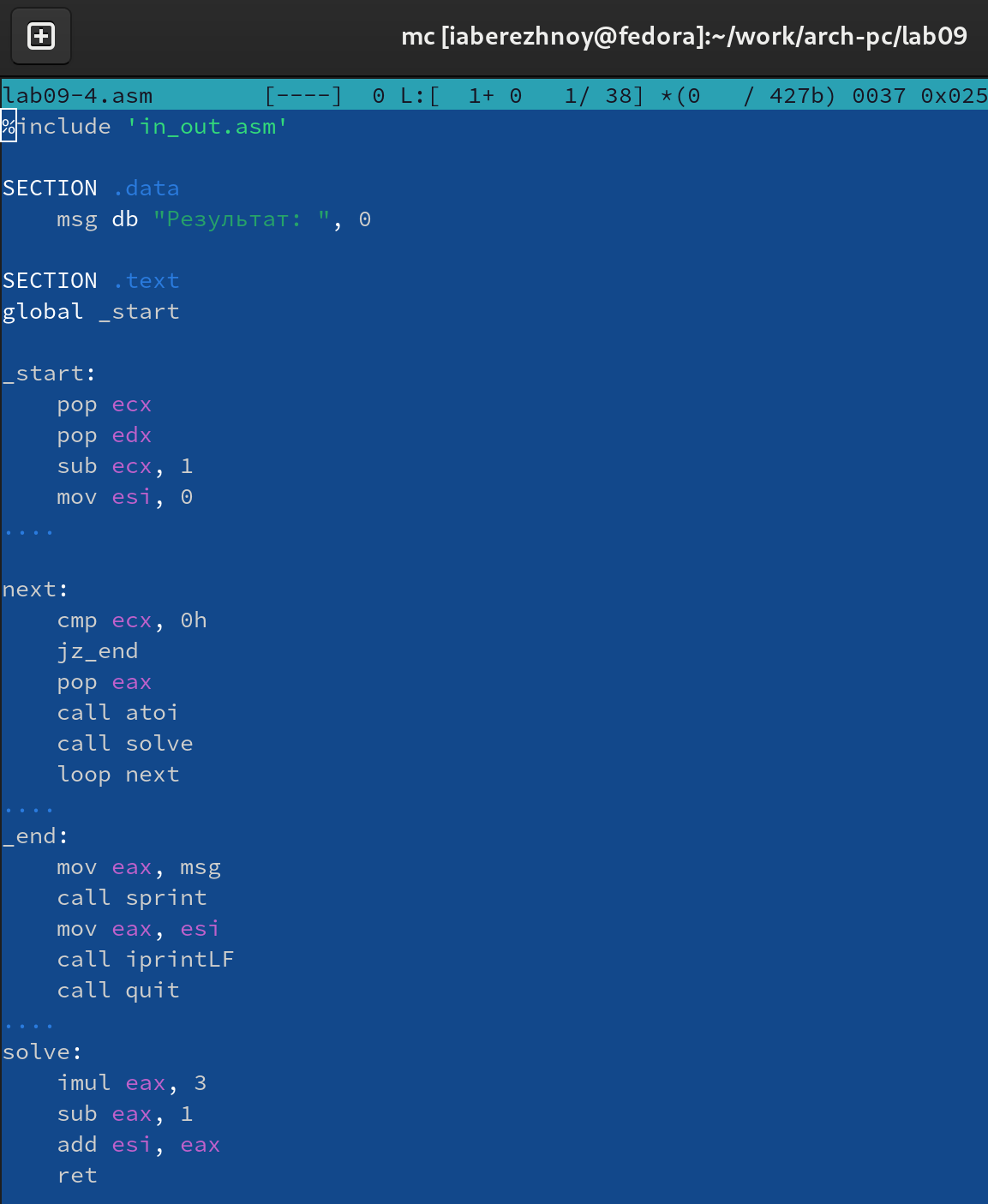


Figure 23: Написание программы

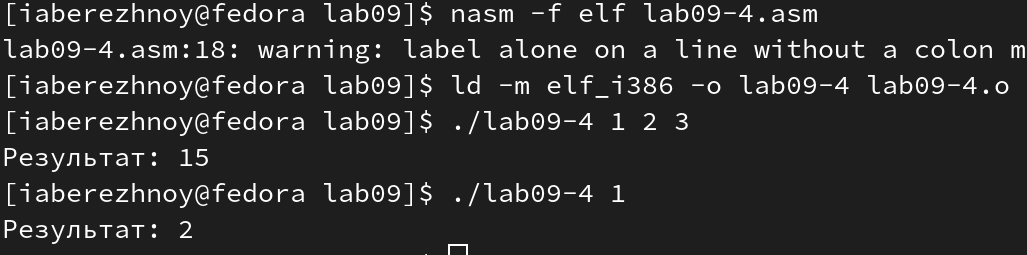


Figure 24: Тестирование программы

**Листинг 9.1. Программа нахождения значения функции с использованием подпрограммы**

%include 'in\_out.asm'  
  
SECTION .data  
 msg db "Результат: ", 0  
  
SECTION .text  
global \_start  
  
\_start:  
 pop ecx  
 pop edx  
 sub ecx, 1  
 mov esi, 0  
   
  
next:  
 cmp ecx, 0h  
 jz\_end  
 pop eax  
 call atoi  
 call solve  
 loop next  
   
\_end:  
 mov eax, msg  
 call sprint  
 mov eax, esi  
 call iprintLF  
 call quit  
   
solve:  
 imul eax, 3  
 sub eax, 1  
 add esi, eax  
 ret

1. Создадим файл lab09-5.asm и скопируем в него предложенный листинг 9.3 (рис. [25](#fig:025)).

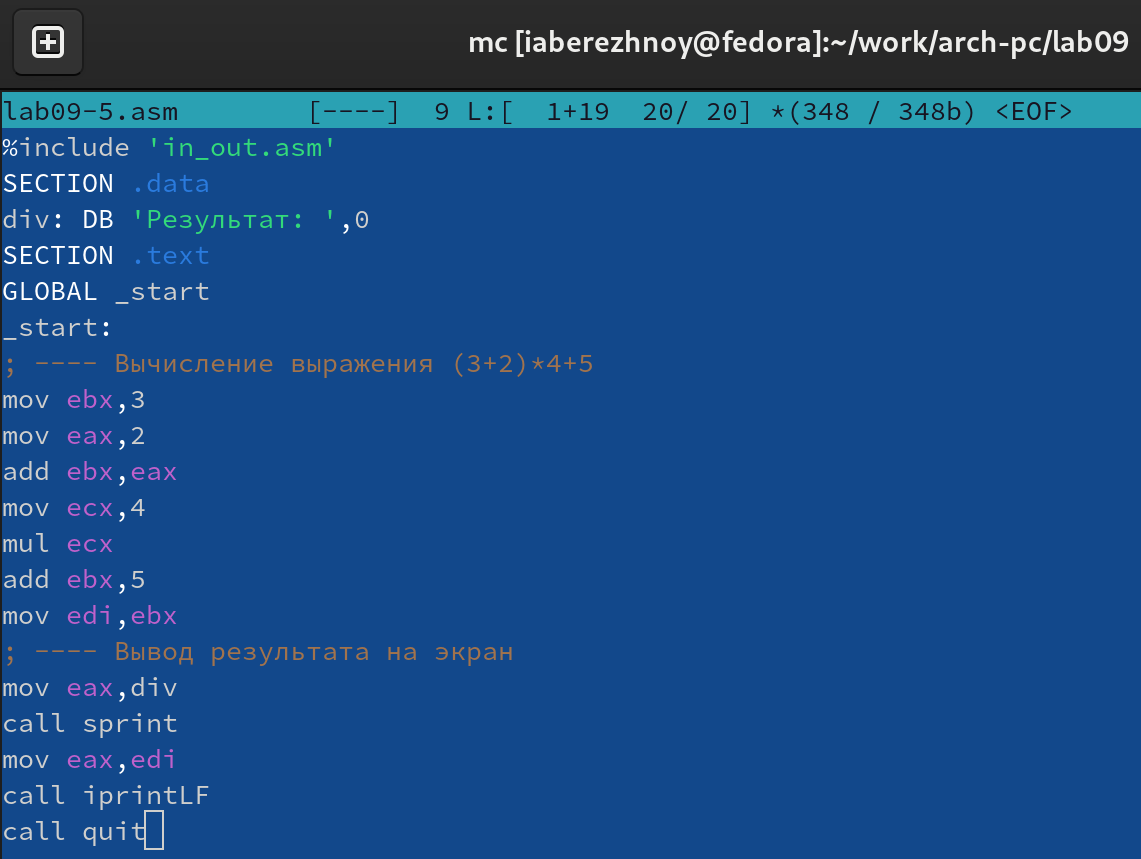


Figure 25: Копирование листинга в lab9-5.asm

Создадим исполняемый файл, загрузим его в отладчик GDB и проверим его работу (рис. [26](#fig:026)). Результат получился неверным. Установим несколько брейкпоинтов и пройдёмся по коду, учитывая изменения значений регистров (рис. [27](#fig:027) и рис. [28](#fig:028)).

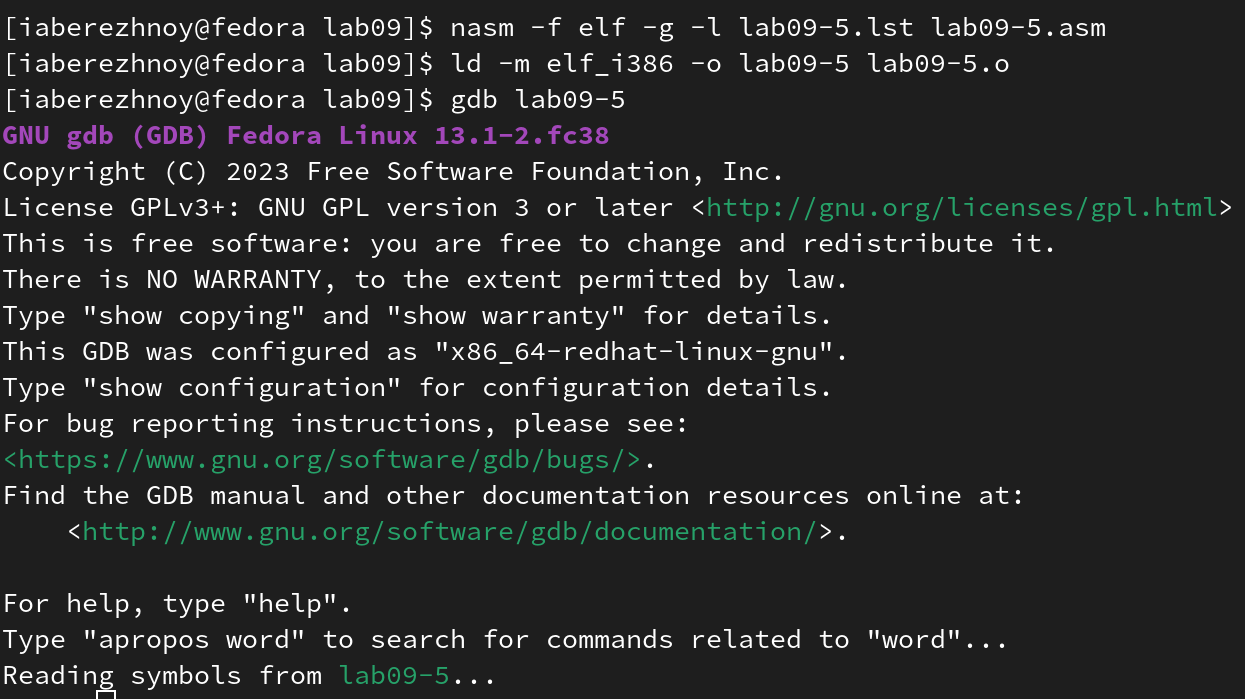


Figure 26: Создание исполняемого файла lab09-5.exe

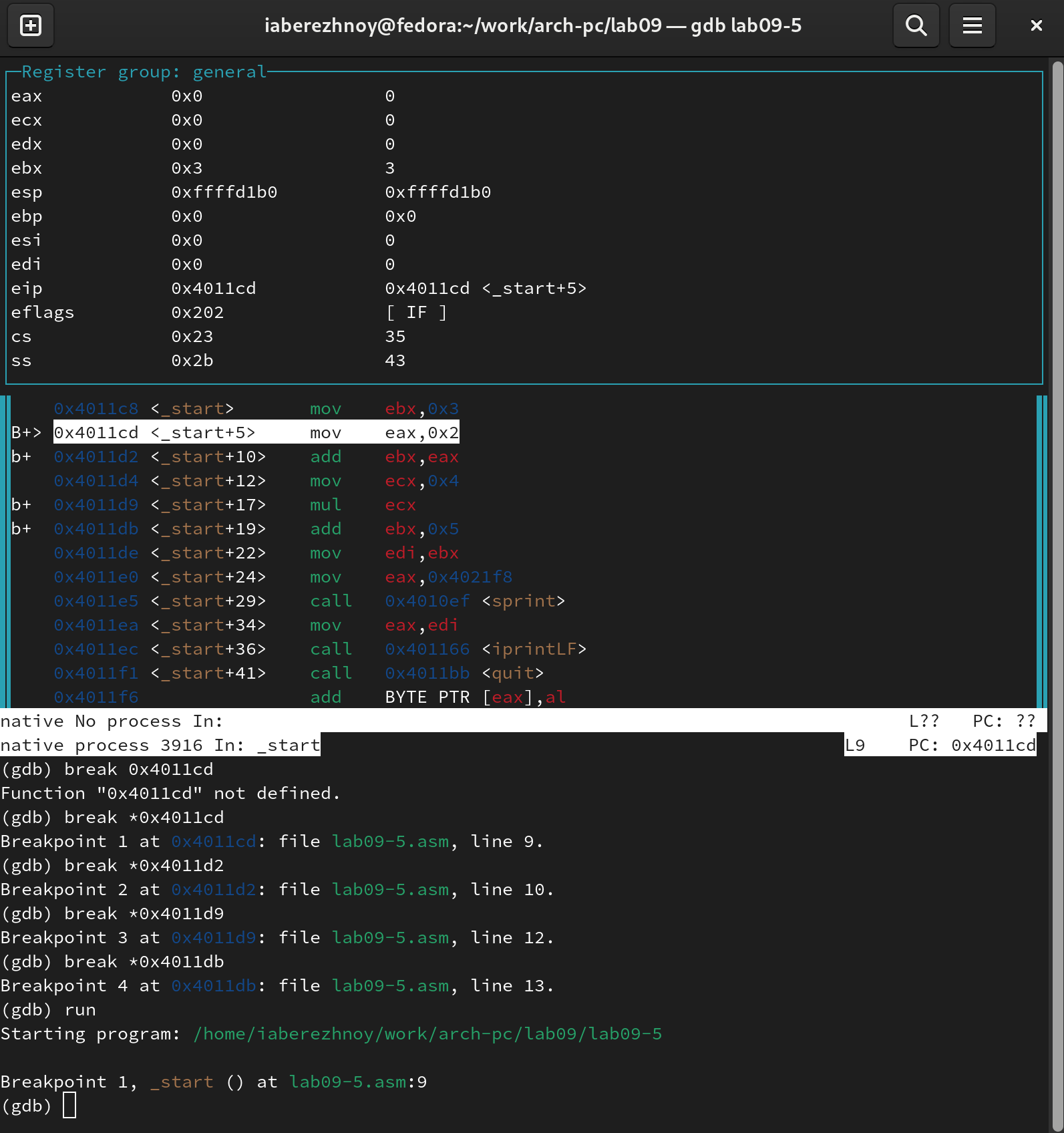


Figure 27: Начало проверки программы

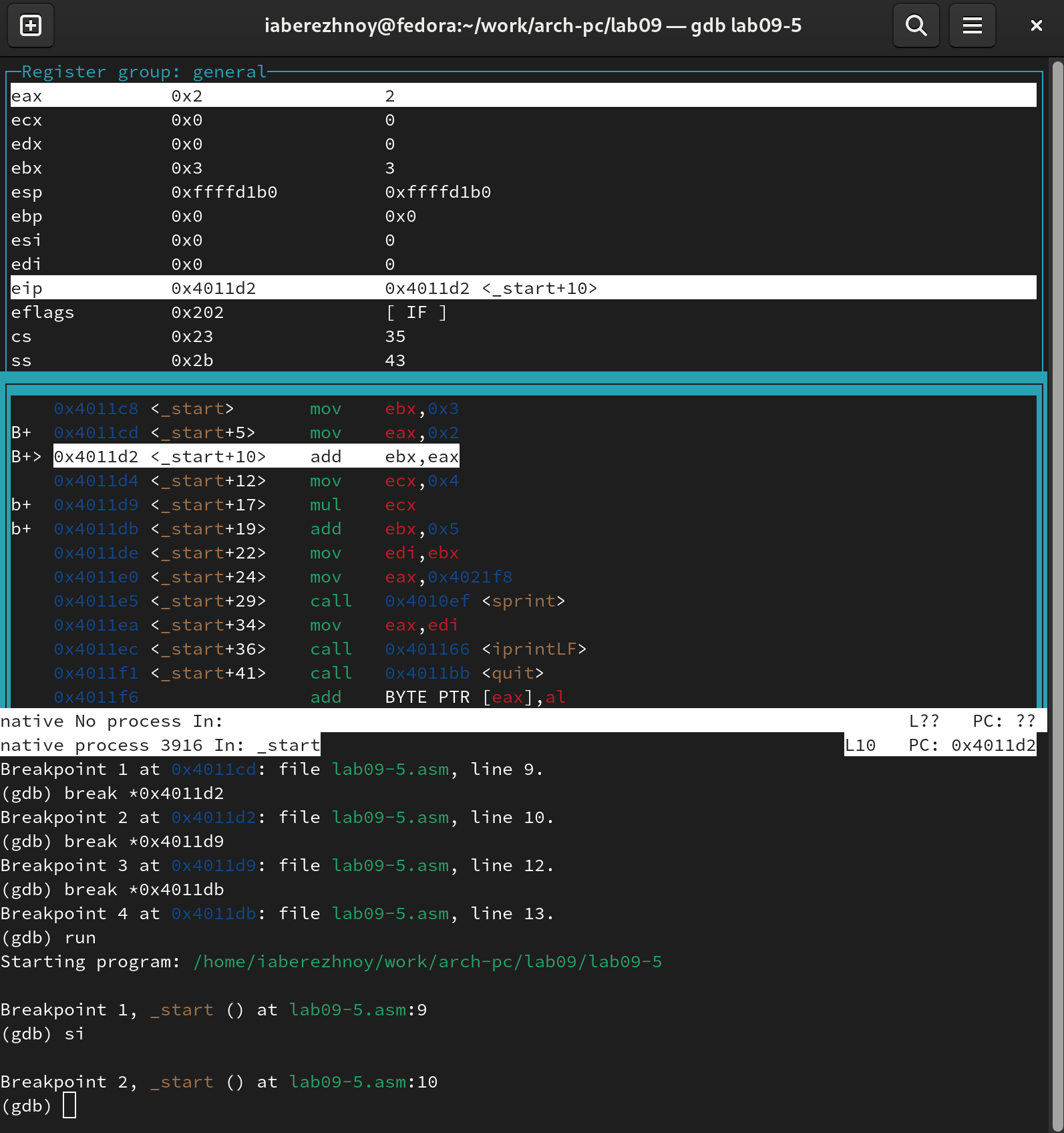


Figure 28: Конец проверки программы

Очевидно, что присутствуют несколько ошибок, которые мы исправим (рис. [29](#fig:029)). Проверим работу программы (рис. [30](#fig:030)).

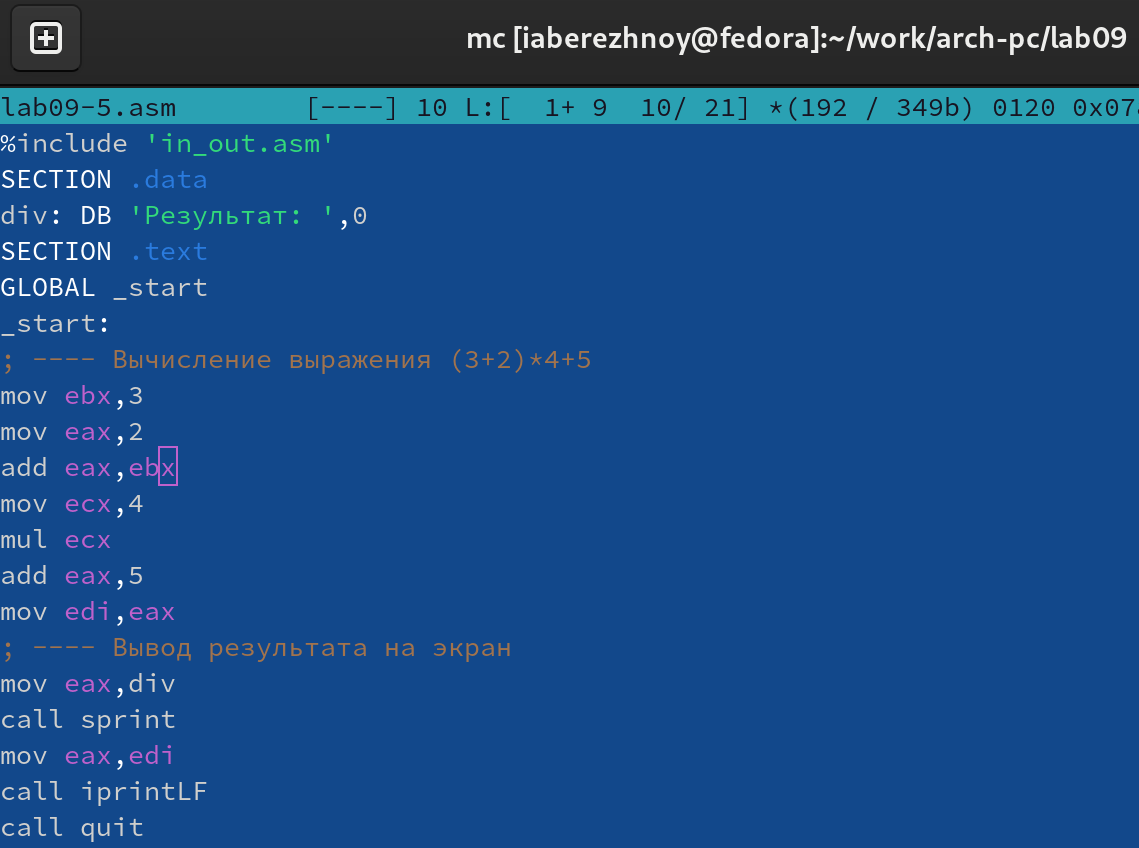


Figure 29: Исправление кода

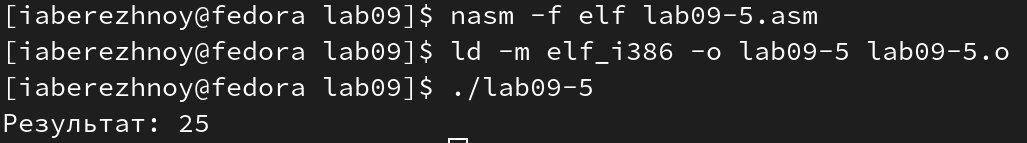


Figure 30: Проверка исправленного кода

**Листинг 9.2. Исправленная программа, находящая значение выражения**

%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 eax,ebx  
mov ecx,4  
mul ecx  
add eax,5  
mov edi,eax  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

# 4 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм и познакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

::: [Архитектура ЭВМ](https://esystem.rudn.ru/mod/resource/view.php?id=1030557)