Отчёт по лабораторной работе 9

Архитектура компьютера

Соловьев Серафим

Содержание

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Выполнение лабораторной работы

Я организовал папку для проведения лабораторного занятия № 9 и переместился в неё. После этого я создал файл с именем lab9-1.asm.

Давайте рассмотрим в качестве примера программу, задачей которой является расчёт арифметической формулы , используя для этого вспомогательную функцию calcul. В этом случае значение подаётся через клавиатуру, а расчёт формулы происходит внутри вспомогательной функции.

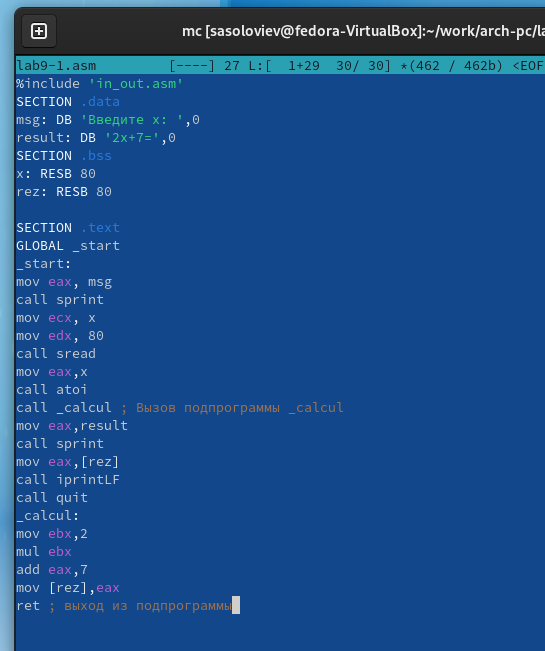


Figure 1: Код программы lab9-1.asm

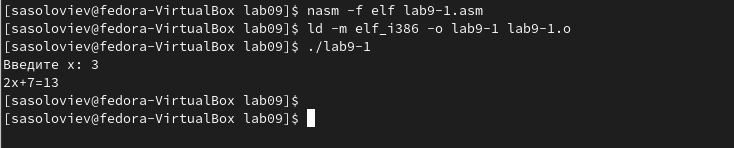


Figure 2: Компиляция и запуск программы lab9-1.asm

Затем я внес некоторые корректировки в код программы, включив дополнительную функцию subcalcul внутри calcul для расчёта формулы , при этом значение по-прежнему вводится через клавиатуру, а функции и обрабатываются внутри этих функций.

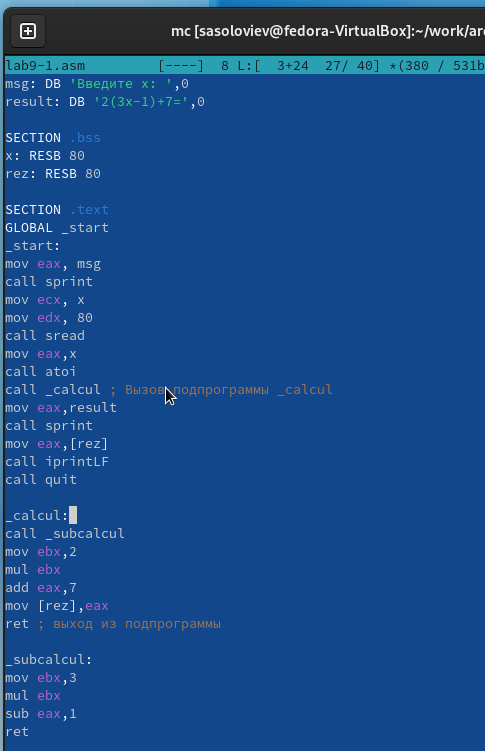


Figure 3: Код программы lab9-1.asm

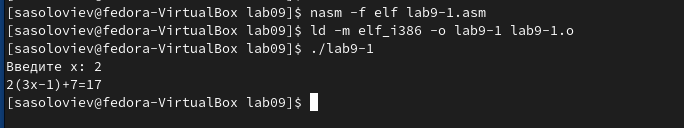


Figure 4: Компиляция и запуск программы lab9-1.asm

Кроме того, я подготовил файл lab9-2.asm, содержащий код программы из Примера 9.2, который демонстрирует программу для вывода сообщения “Hello world!” на экран.



Figure 5: Код программы lab9-2.asm

Я добавил отладочную информацию с помощью ключа ‘-g’ для возможности работы с отладчиком GDB.

После этого я загрузил исполняемый файл в отладчик GDB и проверил функционирование программы, активировав её командой ‘run’ (или ‘r’).

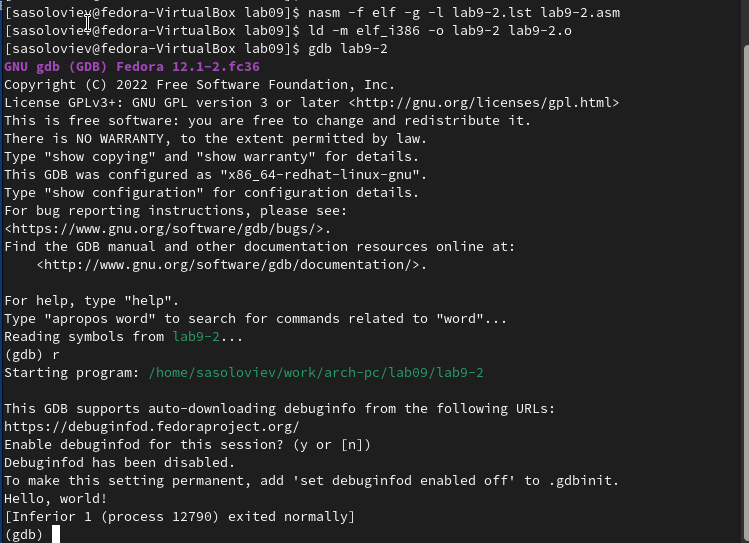


Figure 6: Компиляция и запуск программы lab9-2.asm в отладчике

Для тщательного анализа программы я установил точку останова на метке ‘start’, с которой стартует исполнение любой программы на ассемблере, и запустил программу для наблюдения. После этого я осмотрел дизассемблированный код программы, чтобы понять её структуру и работу.

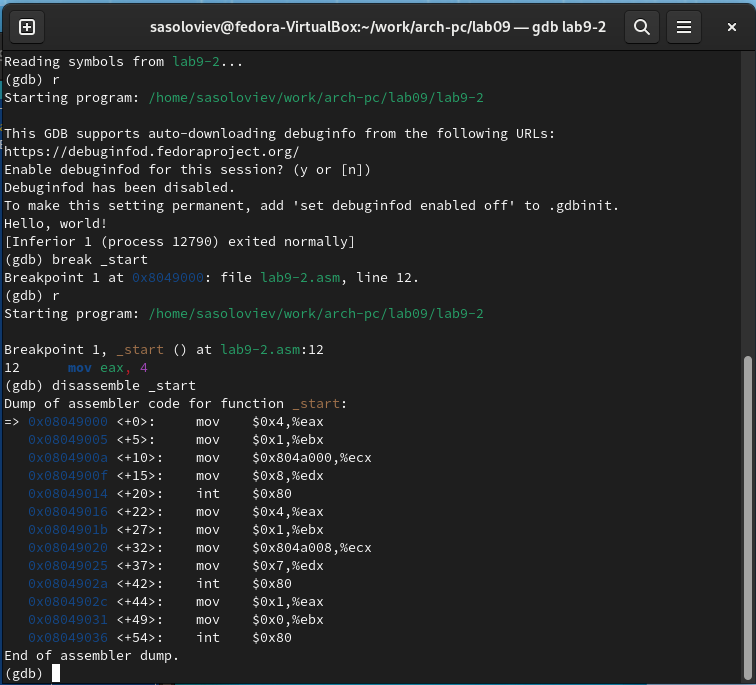


Figure 7: Дизассемблированный код

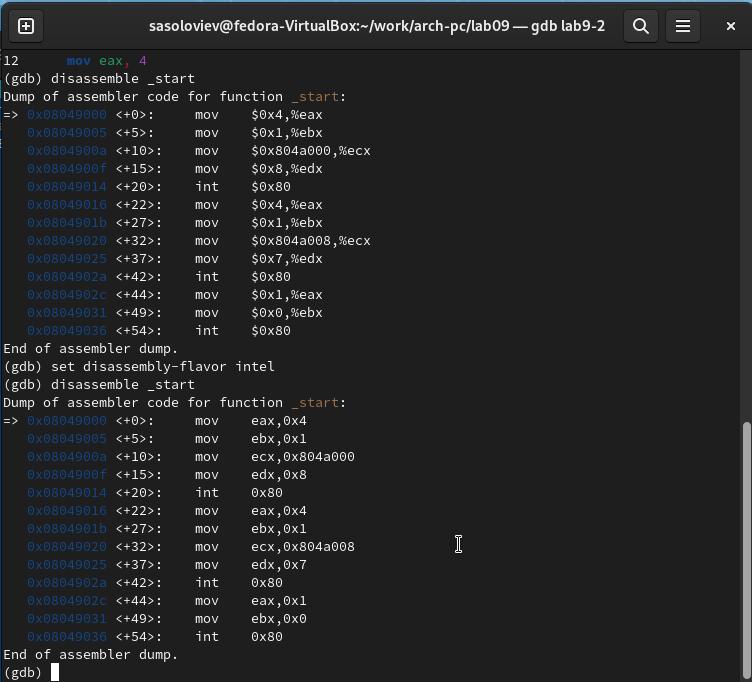


Figure 8: Дизассемблированный код в режиме интел

Чтобы проверить наличие брейкпоинта с меткой ’\_start’, я применил команду ‘info breakpoints’ (или ‘i b’). После этого я задал еще один брейкпоинт на адресе предпоследней команды ‘mov ebx, 0x0’.

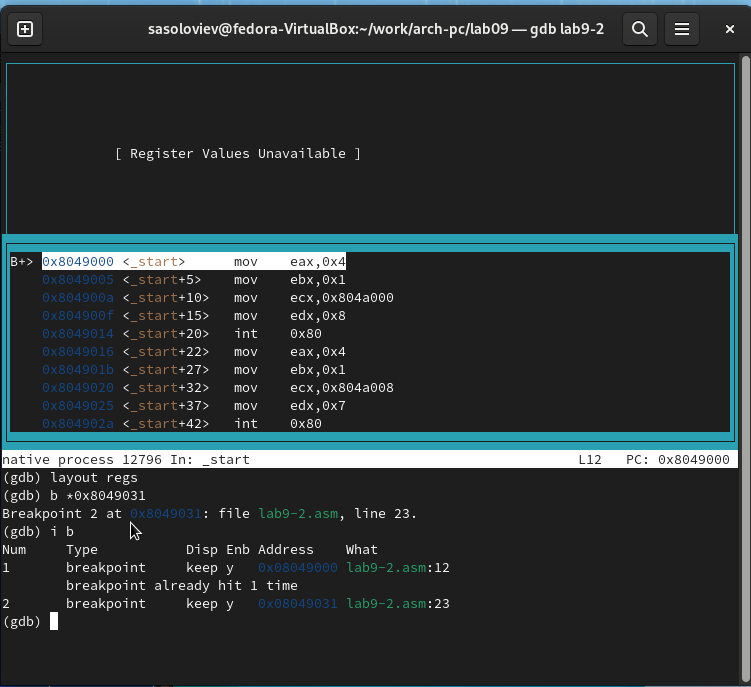


Figure 9: Точка остановки

Используя отладчик GDB, я мог наблюдать и редактировать содержимое памяти и регистров. Я выполнил пять шагов командой ‘stepi’ (или ‘si’), следя за изменениями в регистрах.

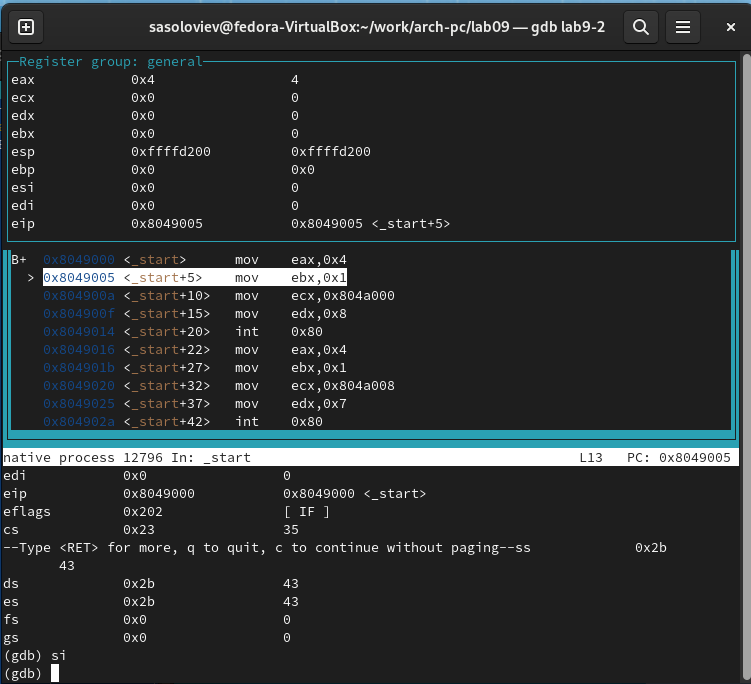


Figure 10: Изменение регистров

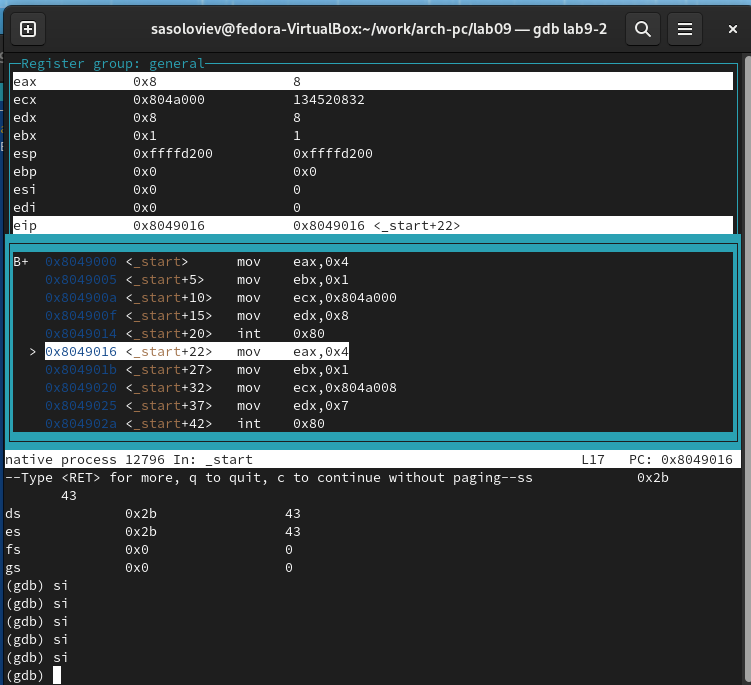


Figure 11: Изменение регистров

Чтобы просмотреть значение переменной msg1, я воспользовался соответствующей командой для извлечения необходимой информации.

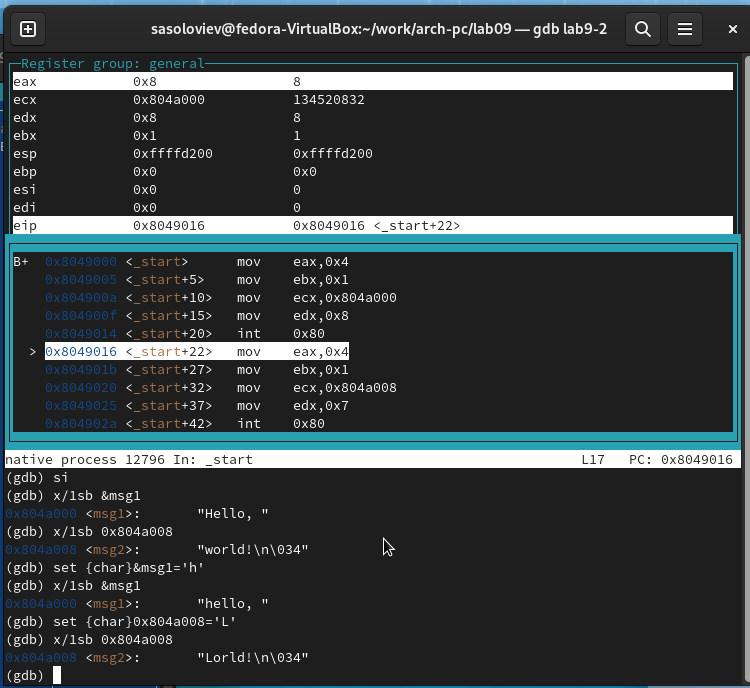


Figure 12: Изменение значения переменной

Я также использовал команду ‘set’ для модификации значений в регистрах или ячейках памяти, указывая при этом нужный регистр или адрес. Мне удалось изменить первый символ переменной msg1.

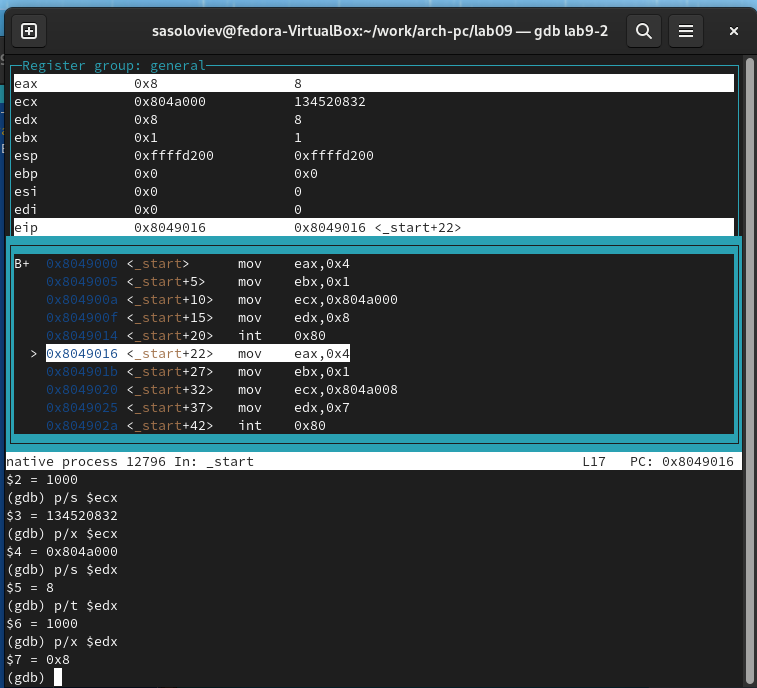


Figure 13: Вывод значения регистра

С помощью команды ‘set’ я изменил значение регистра ebx на требуемое.

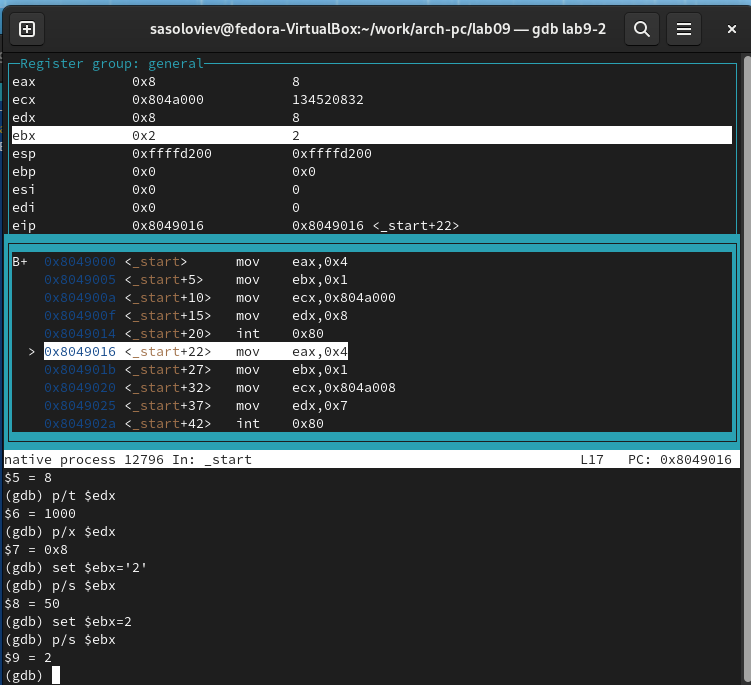


Figure 14: Вывод значения регистра

Я скопировал файл lab8-2.asm, созданный в рамках лабораторной работы №8, который содержит код программы для вывода аргументов командной строки, и сформировал из него исполняемый файл.

Для запуска программы с аргументами в GDB я использовал опцию –args, загрузив исполняемый файл с заданными аргументами в отладчик.

Я установил брейкпоинт перед выполнением первой команды программы и начал ее выполнение.

Адрес вершины стека, содержащий количество аргументов командной строки (включая название программы), находится в регистре ESP. По этому адресу расположено число, показывающее количество аргументов. В моем случае было видно, что их пять, включая название программы lab9-3 и аргументы: аргумент1, аргумент2 и ‘аргумент 3’.

Я также осмотрел другие записи стека. По адресу [ESP+4] расположен указатель на имя программы в памяти. Адреса первого, второго и последующих аргументов находятся по адресам [ESP+8], [ESP+12] и так далее, с шагом в 4 байта, поскольку каждый следующий адрес отстоит на 4 байта от предыдущего ([ESP+4], [ESP+8], [ESP+12]).

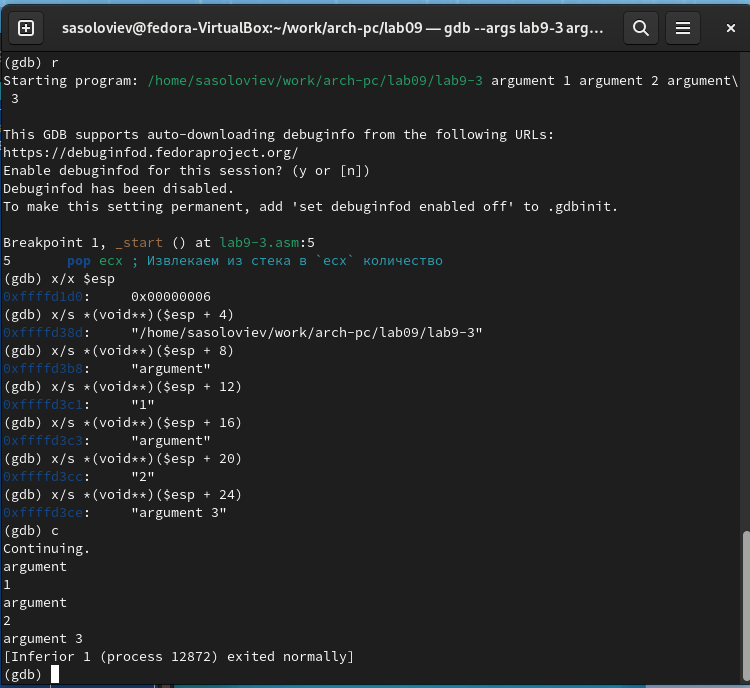


Figure 15: Вывод значения регистра

## 2.1 Задание для самостоятельной работы

Модифицировал код из восьмой лабораторной работы (Первое задание для индивидуального выполнения), создав подпрограмму для расчета значения функции f(x).



Figure 16: Код программы lab9-4.asm

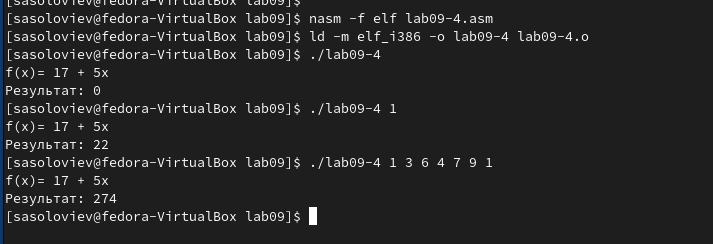


Figure 17: Компиляция и запуск программы lab9-4.asm

В представленном коде описан алгоритм для расчета формулы . Однако его исполнение приводит к некорректному итогу. Я выявил это, наблюдая за изменениями в регистрах при помощи отладчика GDB.

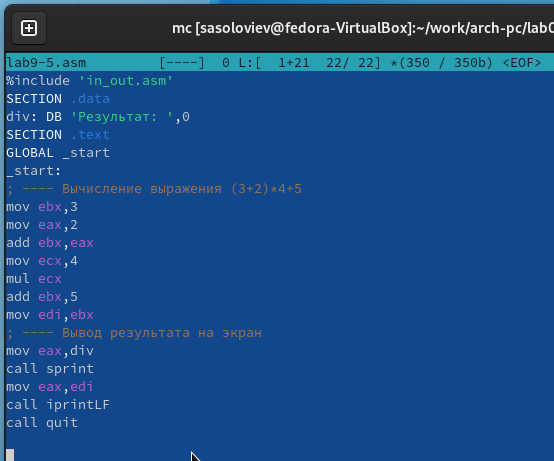


Figure 18: Код программы lab9-5.asm с ошибкой

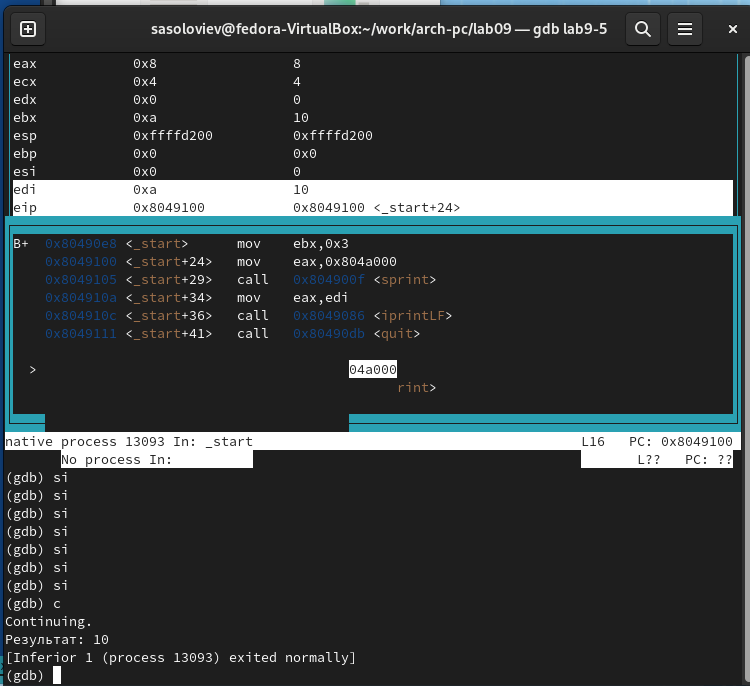


Figure 19: Отладка

Ошибка заключалась в неверном порядке аргументов команды add и в том, что в конце исполнения программы значение ebx переносится в edi вместо eax.

Исправленный код программы

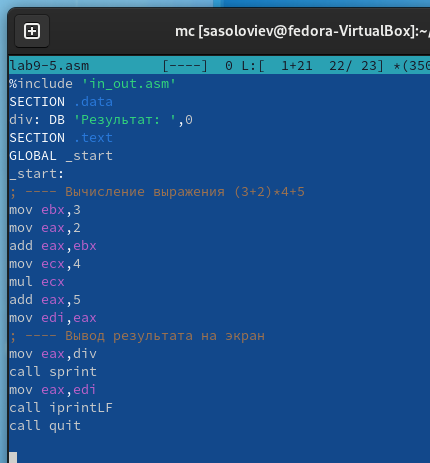


Figure 20: Код программы lab9-5.asm исправлен

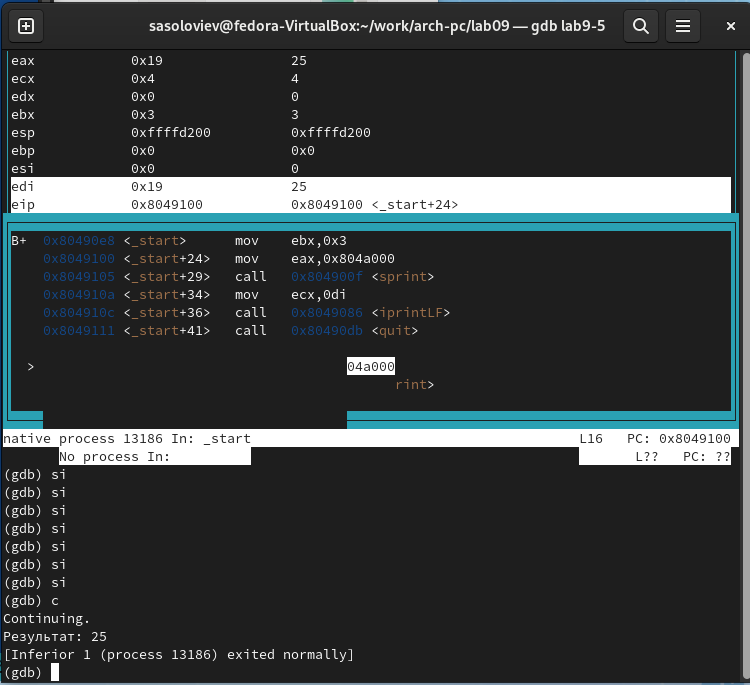


Figure 21: Проверка работы

# 3 Выводы

Освоили работy с подпрограммами и отладчиком.