Отчёт по лабораторной работе 9

Архитектура компьютеров

Рамиэль Сарханов

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

## 2.1 Реализация подпрограмм в NASM

Для выполнения лабораторной работы №9 я создал новую папку и перешел в нее. Далее был создан файл с именем lab9-1.asm.

В качестве примера я рассмотрел программу, которая вычисляет арифметическое выражение с использованием подпрограммы calcul. В этом примере значение переменной вводится с клавиатуры, а само выражение вычисляется внутри подпрограммы. (рис. 1) (рис. 2)

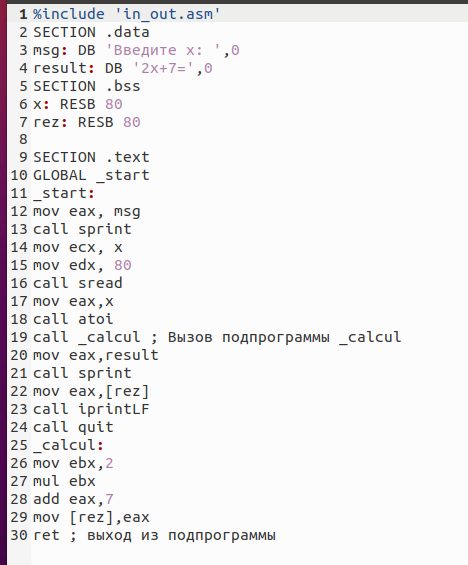


Рис. 1: Программа lab9-1.asm

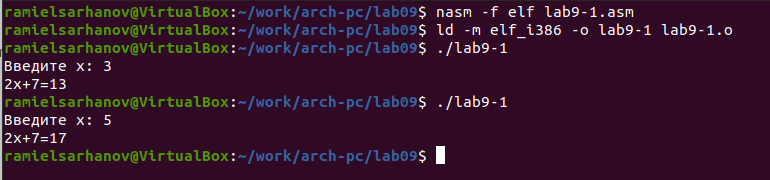


Рис. 2: Запуск программы lab9-1.asm

После этого я внес изменения в текст программы, добавив подпрограмму subcalcul внутрь подпрограммы calcul. Это позволило вычислить составное выражение , где значение также вводится с клавиатуры. Функции определены следующим образом: , . (рис. 3) (рис. 4)

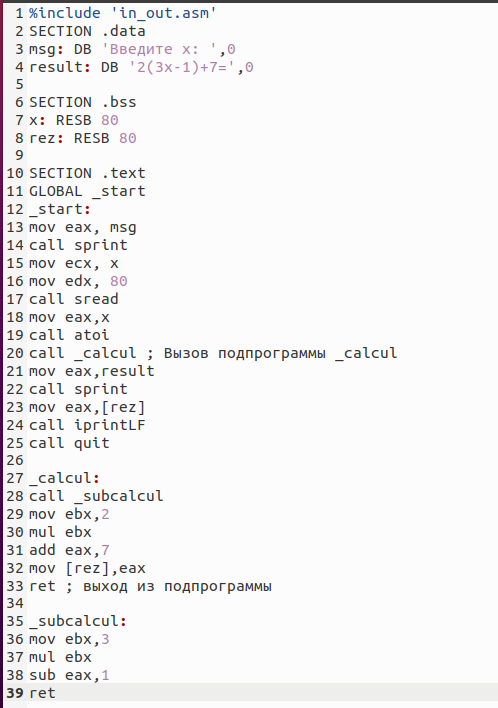


Рис. 3: Программа lab9-1.asm

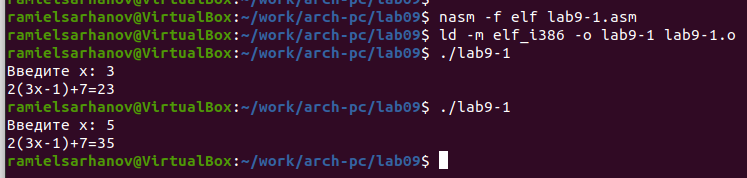


Рис. 4: Запуск программы lab9-1.asm

## 2.2 Отладка программы с помощью GDB

Я создал файл с названием lab9-2.asm, в котором содержится программа из Листинга 9.2. Эта программа отвечает за вывод сообщения “Hello world!” на экран. (рис. 5)

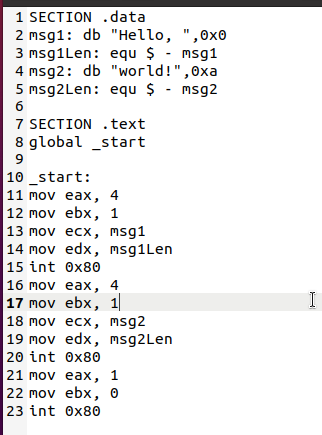


Рис. 5: Программа lab9-2.asm

Затем я скомпилировал файл и получил исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, я использовал ключ “-g”. После этого загрузил полученный исполняемый файл в отладчик GDB и проверил его работу, запустив программу с помощью команды “run” или “r”. (рис. 6)

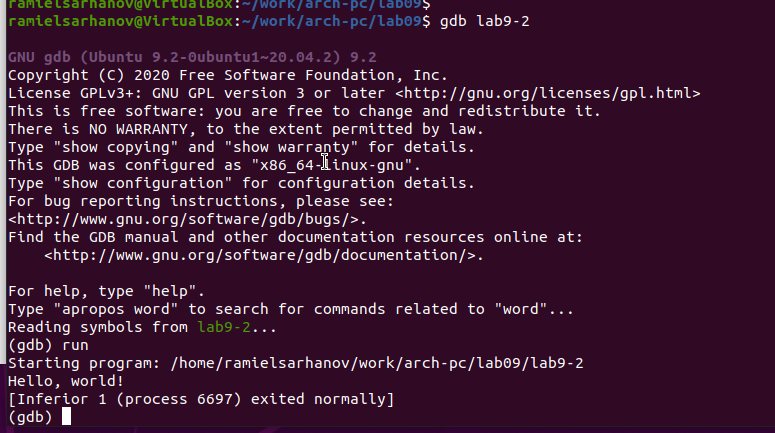


Рис. 6: Запуск программы lab9-2.asm в отладчике

Для более детального анализа программы я установил точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустил её. Затем я просмотрел дизассемблированный код программы. (рис. 7) (рис. 8)

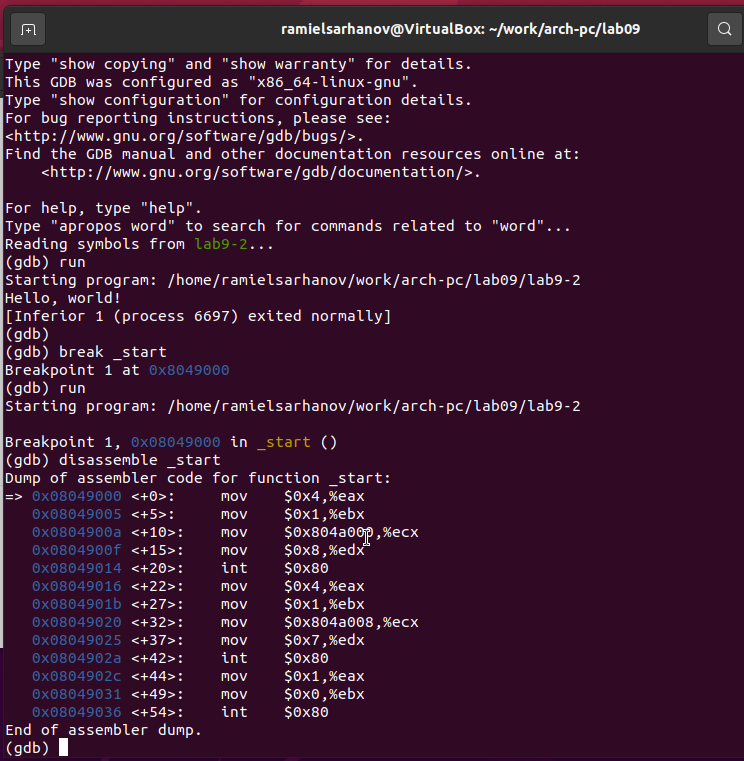


Рис. 7: Дизассемблированный код

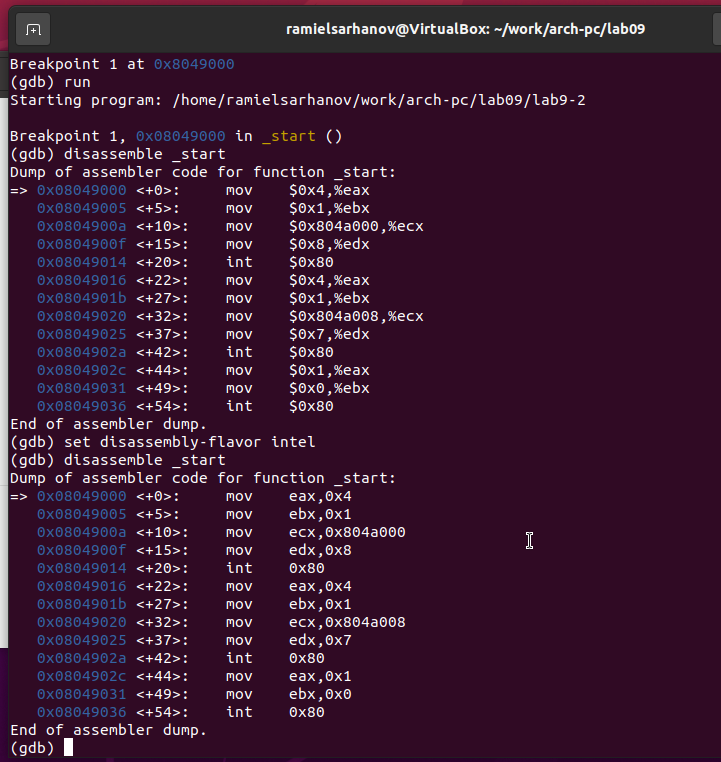


Рис. 8: Дизассемблированный код в режиме intel

Чтобы проверить точку остановки по имени метки “\_start”, я использовал команду “info breakpoints” или “i b”. После этого установил ещё одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0”. (рис. 9)

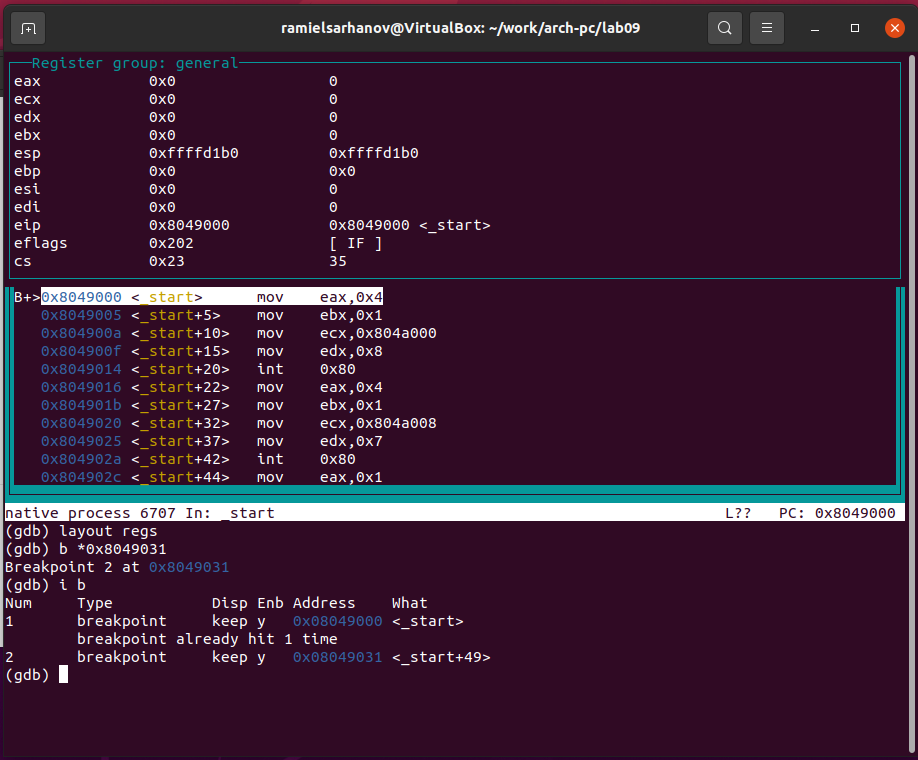


Рис. 9: Точка остановки

В отладчике GDB я имел возможность просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Я выполнил 5 инструкций с помощью команды stepi (сокращенно si) и отслеживал изменение значений регистров. (рис. 10) (рис. 11)

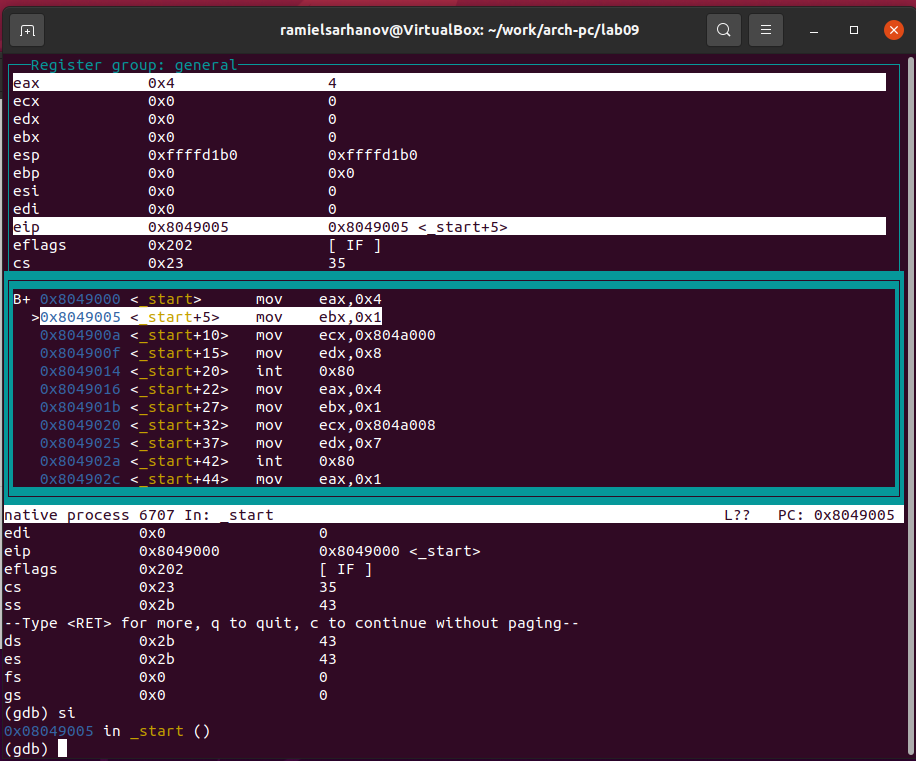


Рис. 10: Изменение регистров

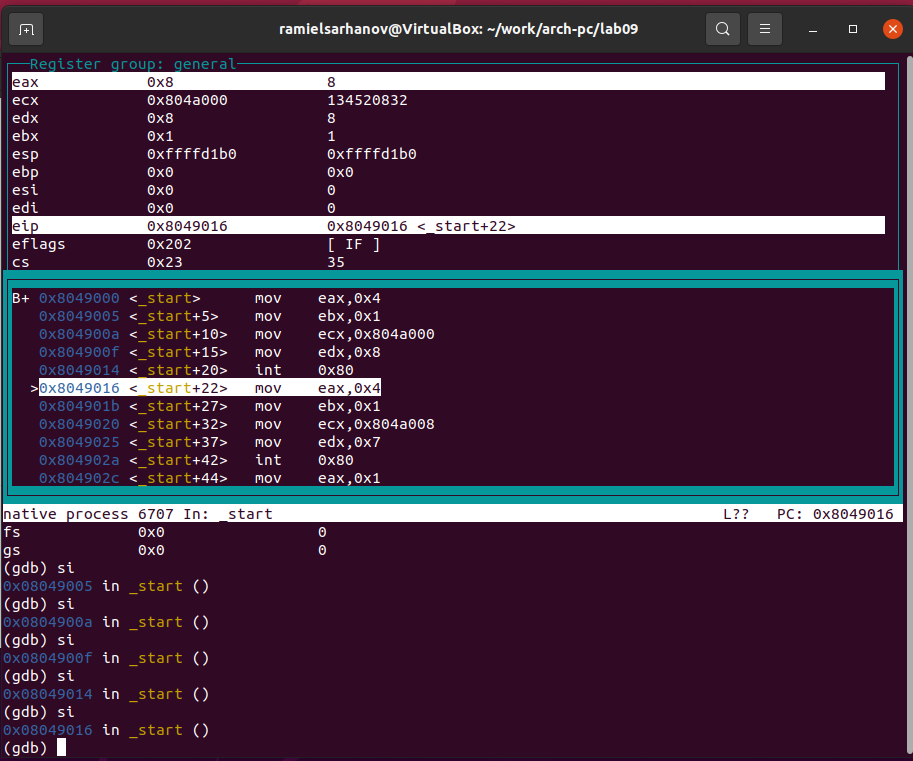


Рис. 11: Изменение регистров

Я также просмотрел значение переменной msg1 по имени и получил нужные данные. Для изменения значения регистра или ячейки памяти использовал команду set, указав имя регистра или адрес в качестве аргумента. Я изменил первый символ переменной msg1. (рис. 12) (рис. 13)

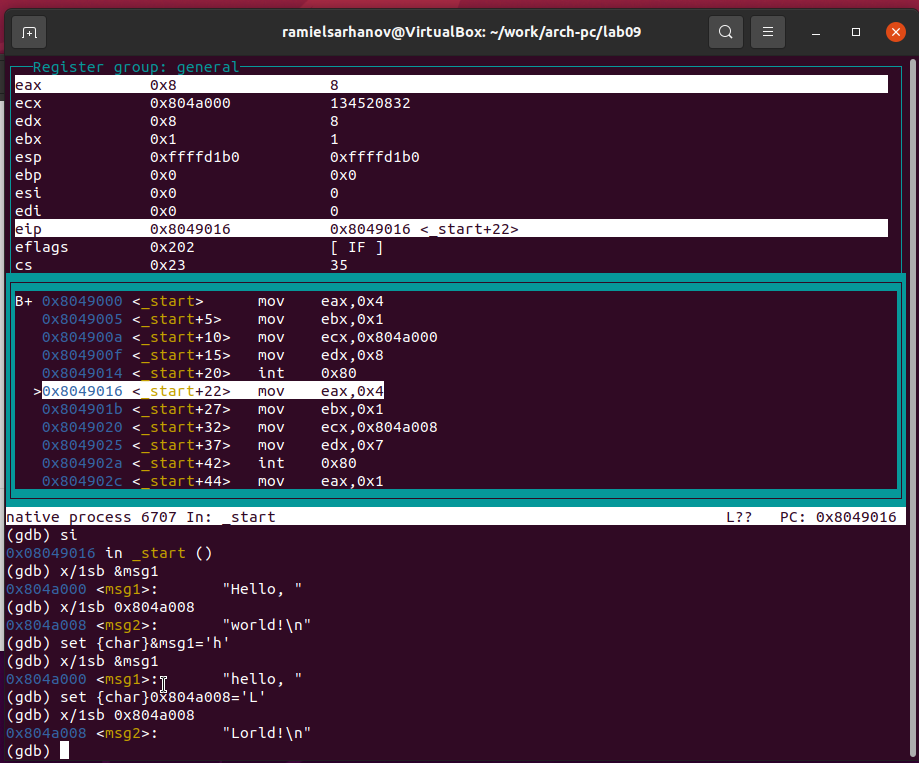


Рис. 12: Изменение значения переменной

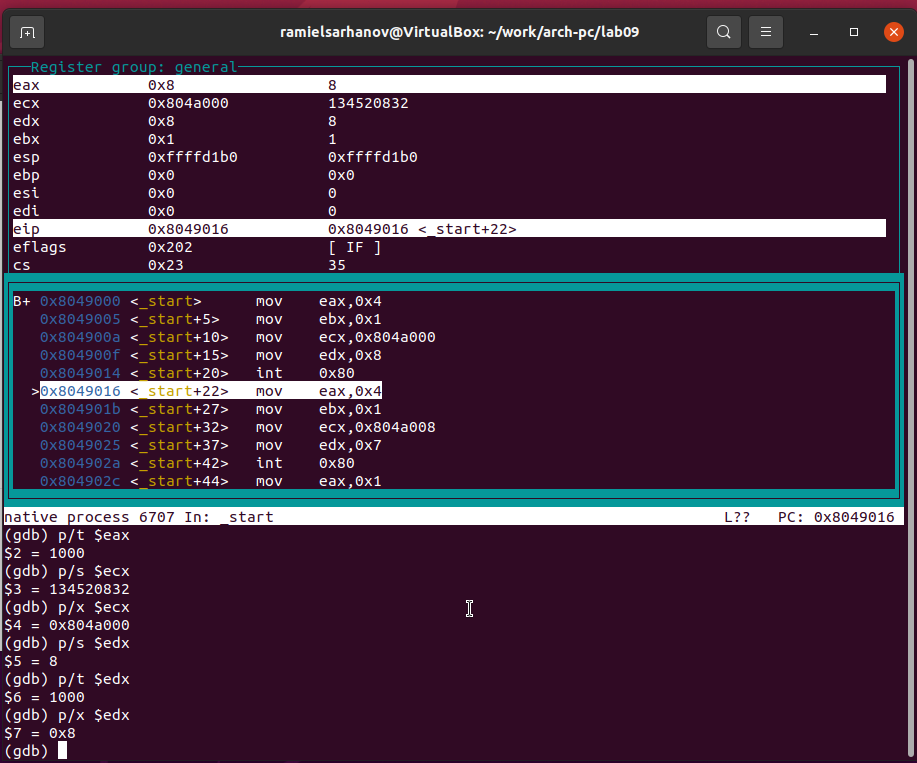


Рис. 13: Вывод значения регистра

Кроме того, с помощью команды set, я изменил значение регистра ebx на нужное значение. (рис. 14)

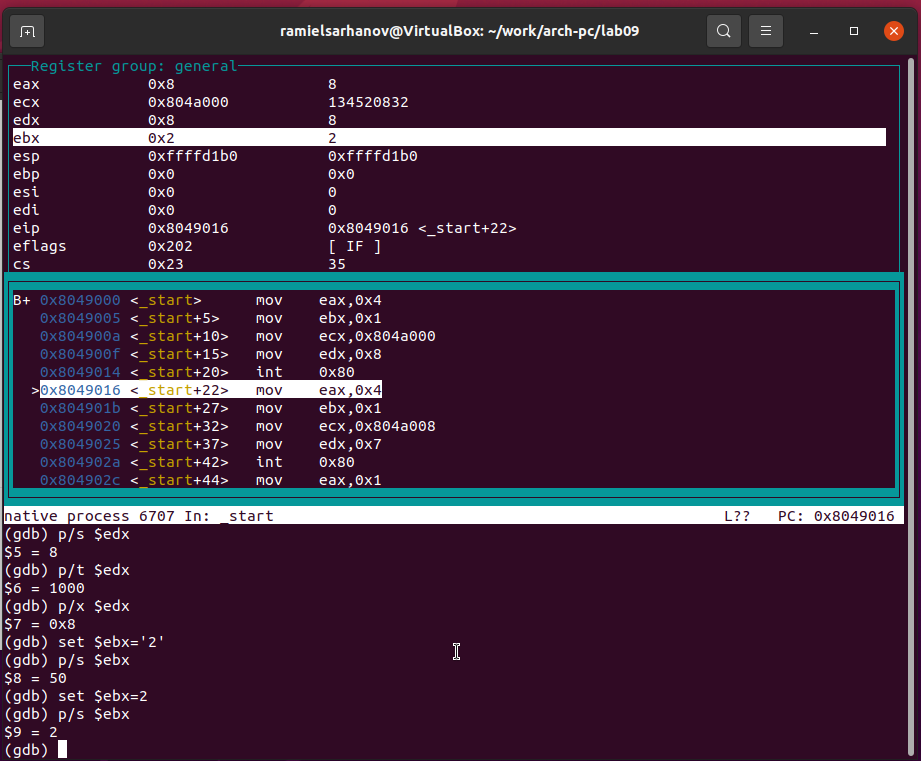


Рис. 14: Вывод значения регистра

Я скопировал файл lab8-2.asm, который был создан в ходе выполнения лабораторной работы №8. Этот файл содержит программу для вывода аргументов командной строки. После этого создал исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в отладчик GDB, использовал ключ –args и загрузил исполняемый файл в отладчик с указанными аргументами. Я установил точку останова перед первой инструкцией программы и запустил её.

Адрес вершины стека, где хранится количество аргументов командной строки (включая имя программы), находится в регистре esp. По этому адресу находится число, указывающее количество аргументов. В данном случае количество аргументов равно 5, включая имя программы lab9-3 и сами аргументы: аргумент1, аргумент2 и аргумент 3.

Я также просмотрел остальные позиции стека. По адресу [esp+4] находится адрес в памяти, где располагается имя программы. По адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] — второго и так далее. Шаг изменения адреса равен 4, так как каждый следующий адрес на стеке находится на расстоянии 4 байт от предыдущего ([esp+4], [esp+8], [esp+12]). (рис. 15)

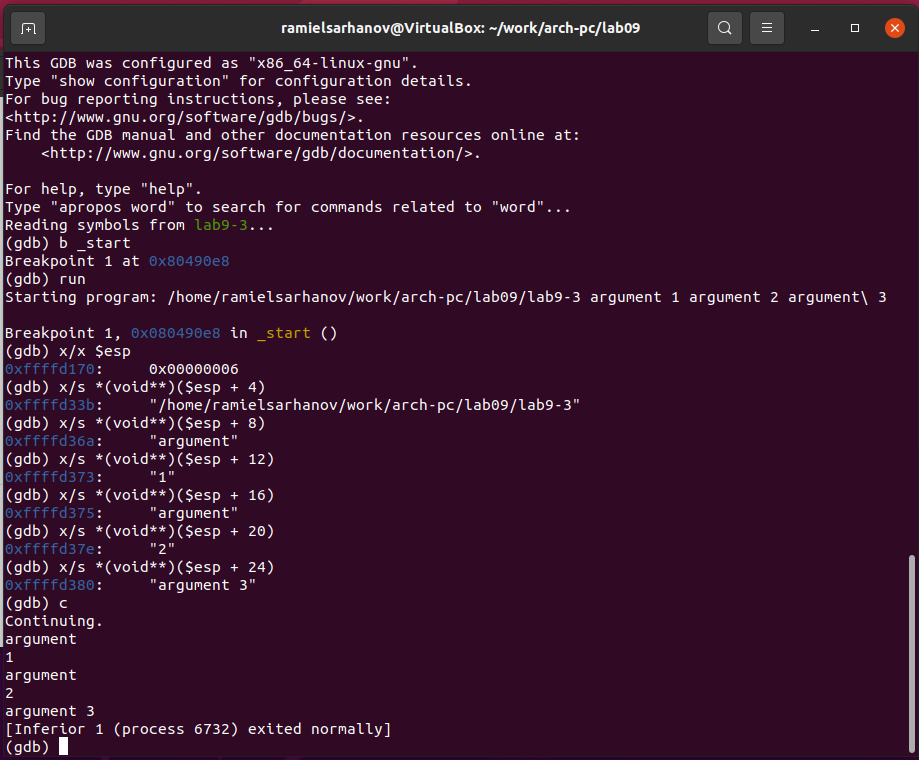


Рис. 15: Вывод значения регистра

## 2.3 Задание для самостоятельной работы

Я решил преобразовать программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), добавив вычисление значения функции в виде подпрограммы. (рис. 16) (рис. 17)

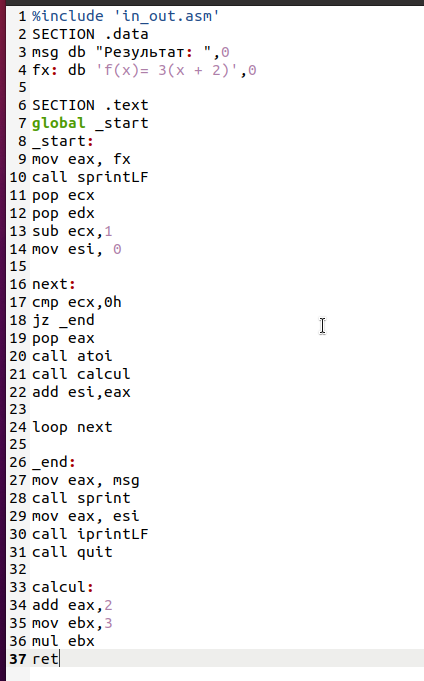


Рис. 16: Программа prog-1.asm

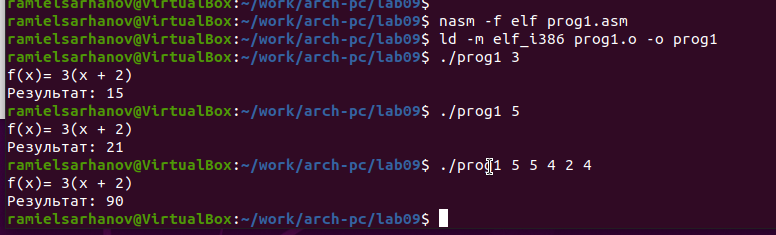


Рис. 17: Запуск программы prog-1.asm

В листинге представлена программа для вычисления выражения . Однако, при запуске программы я обнаружил, что она даёт неверный результат. Для выявления причин я провел анализ изменений значений регистров с помощью отладчика GDB.

В процессе анализа я обнаружил, что порядок аргументов у инструкции add был перепутан. Кроме того, я заметил, что по окончании работы программы значение ebx было отправлено в edi вместо eax. (рис. 18)

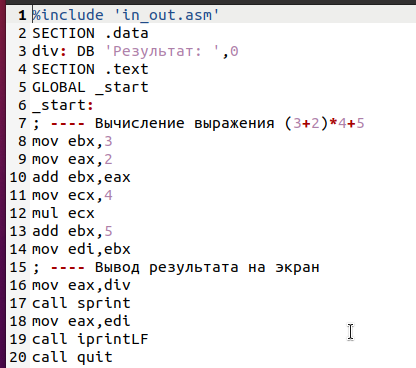


Рис. 18: Код с ошибкой

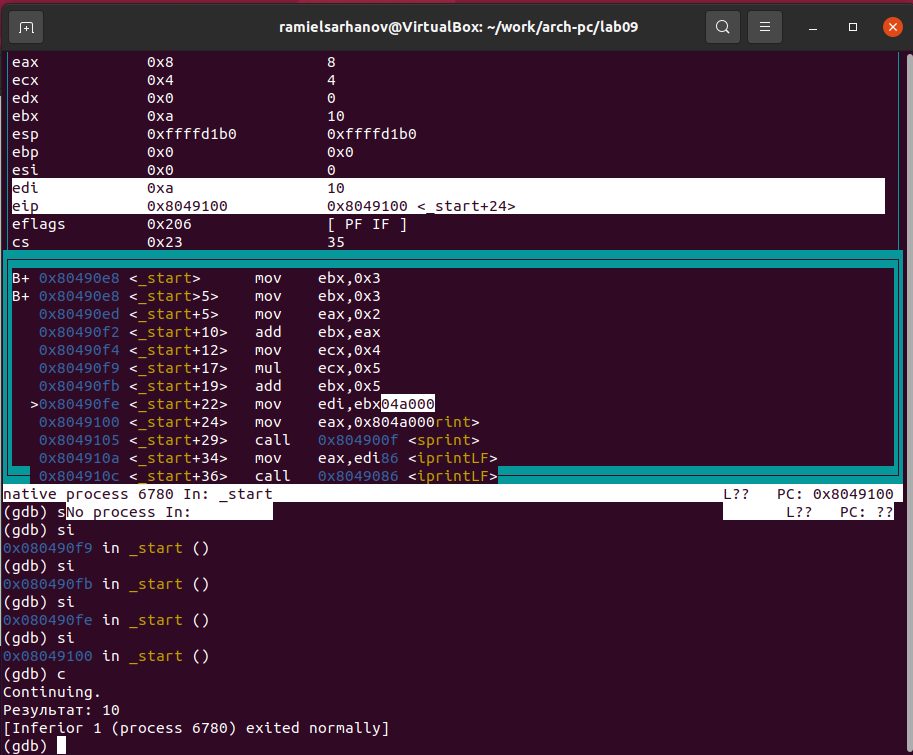


Рис. 19: Отладка

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax. (рис. 19)

Исправленный код программы (рис. 20) (рис. 21)

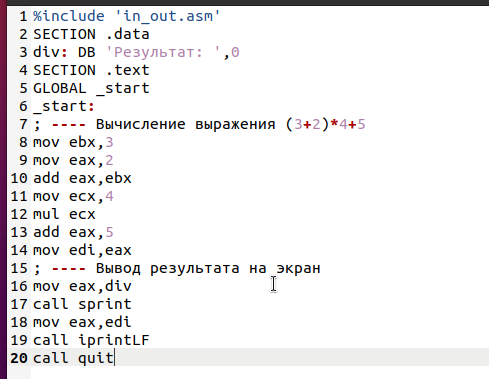


Рис. 20: Код исправлен

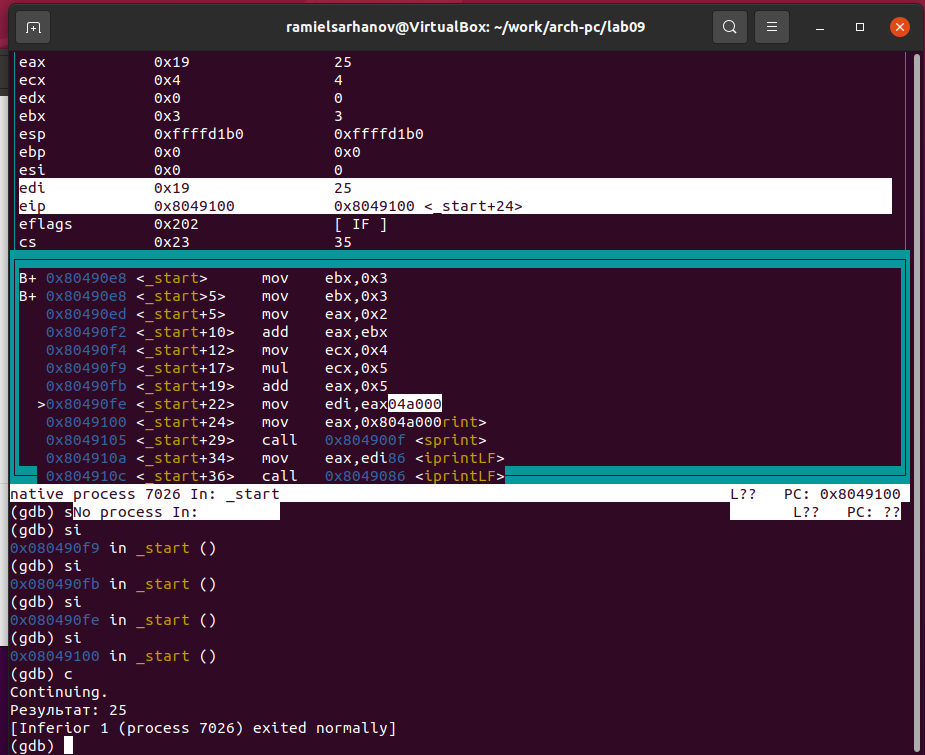


Рис. 21: Проверка работы

# 3 Выводы

В ходе работы я освоил работу с подпрограммами и отладчиком.