Отчёт по лабораторной работе 9

Понятие подпрограммы. Отладчик GDB.

Татьяна Соколова НММбд-03-24

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

Сначала я создала новую папку, чтобы выполнять лабораторную работу номер 9, и перешла в нее. Затем я создала файл с именем lab9-1.asm.

В качестве примера, я рассмотрела 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)

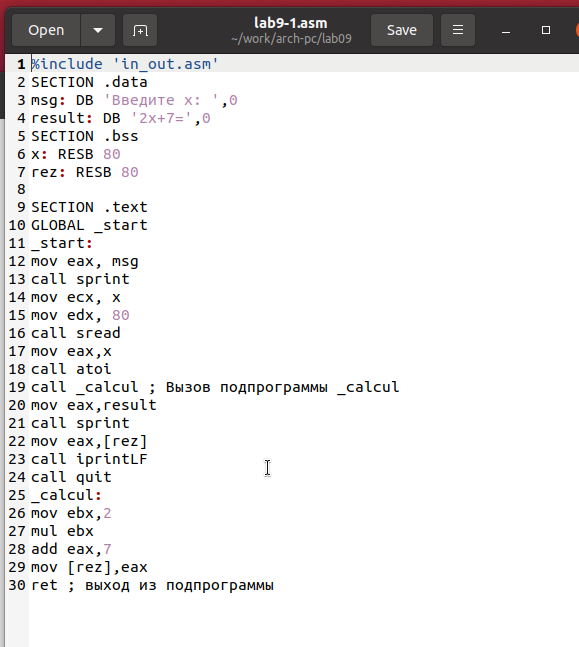


Рис. 1: Программа lab9-1.asm

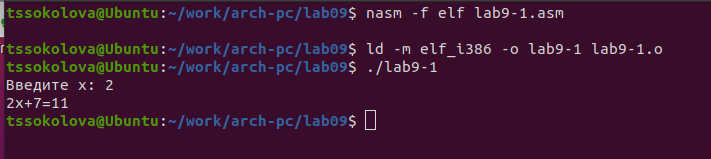


Рис. 2: Запуск программы lab9-1.asm

После этого я внесла 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)

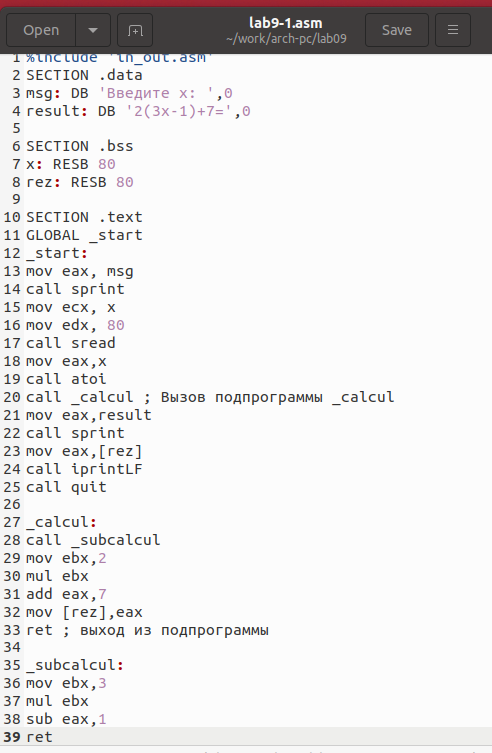


Рис. 3: Программа lab9-1.asm

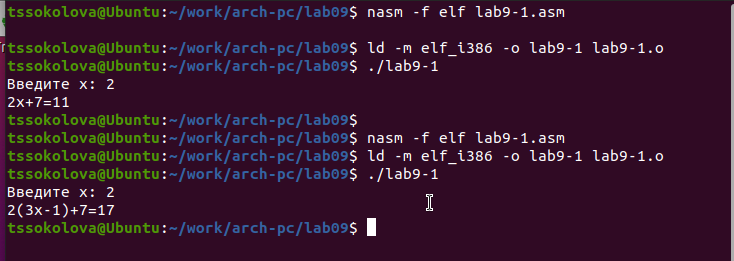


Рис. 4: Запуск программы lab9-1.asm

## 2.2 Отладка программам с помощью GDB

Я создала 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)

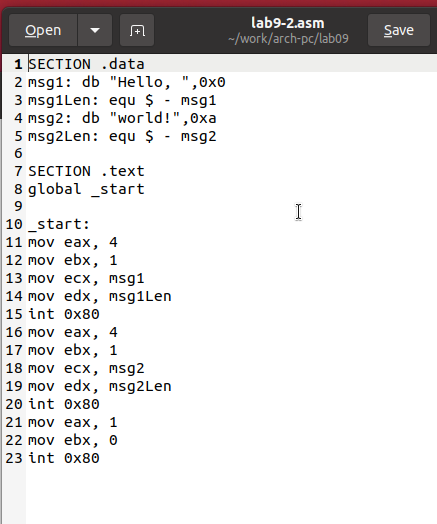


Рис. 5: Программа lab9-2.asm

После этого я скомпилировала файл и получила исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, я использовала ключ “-g”. Затем я загрузила полученный исполняемый файл в отладчик GDB и проверила 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 установила точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустила ее. Затем я просмотрела дизассемблированный код программы.(рис. 7) (рис. 8)

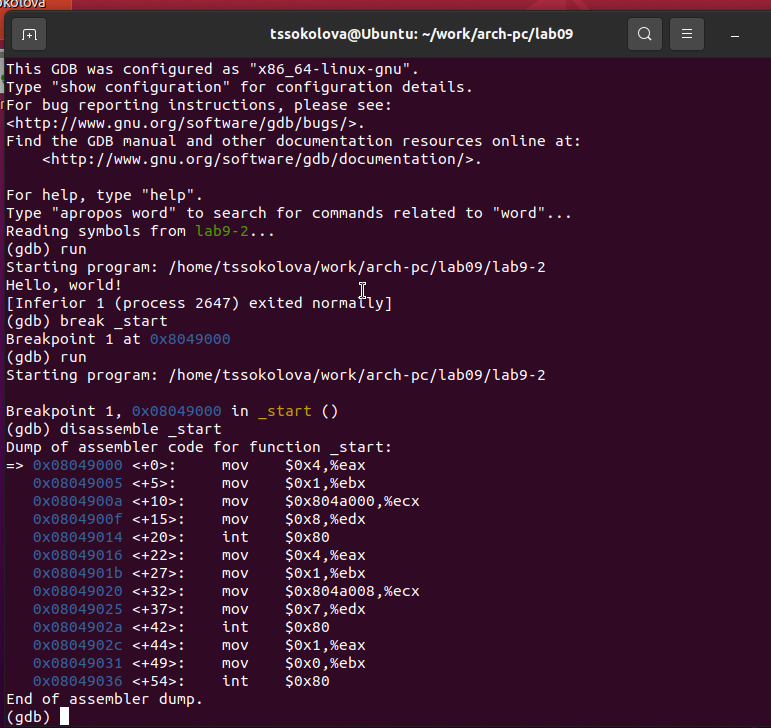


Рис. 7: Дизассемблированный код

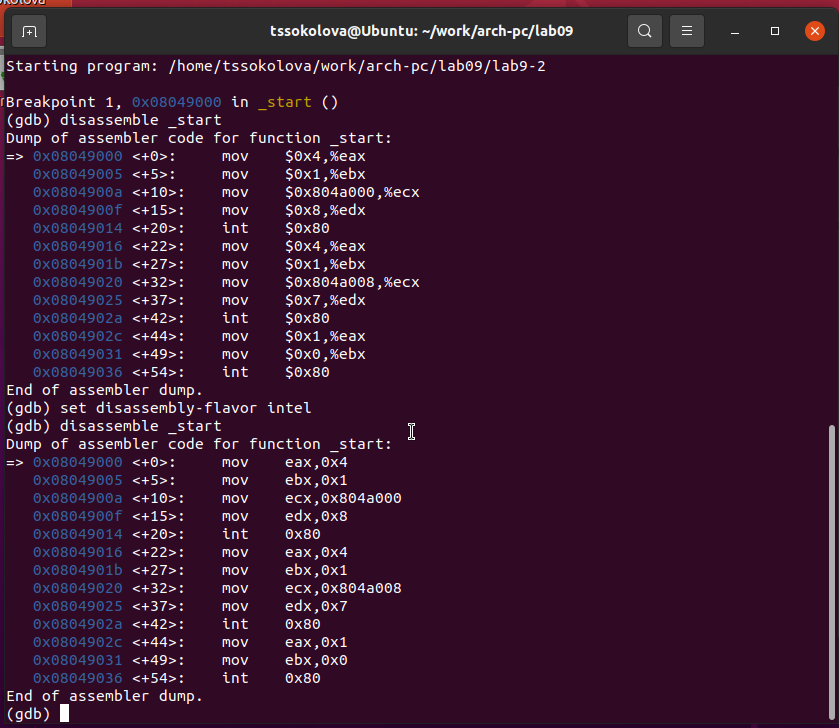


Рис. 8: Дизассемблированный код в режиме интел

Чтобы проверить точку остановки по имени метки “\_start”, я использовала команду “info breakpoints” или “i b”. Затем я установила еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0”. (рис. 9)

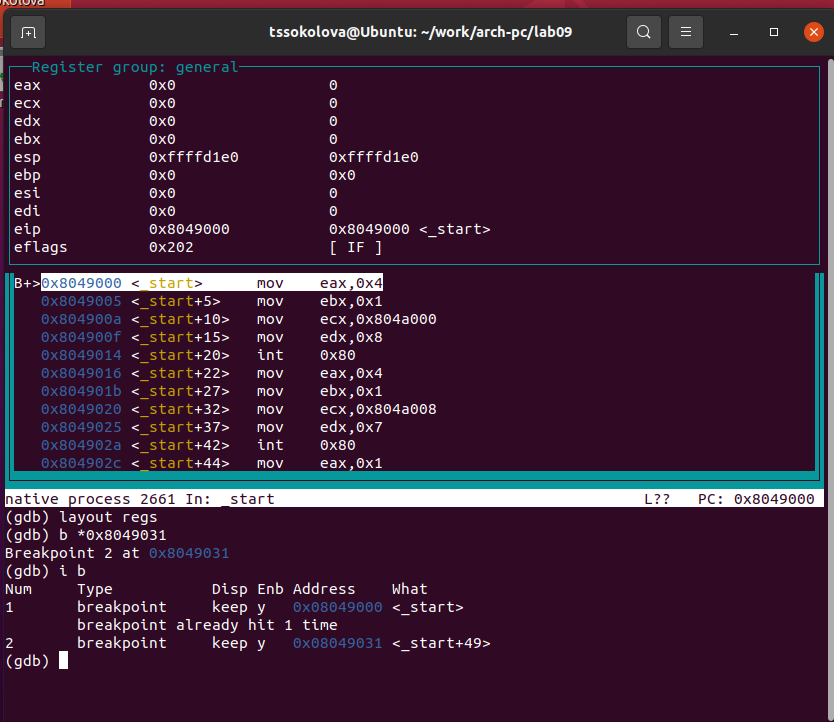


Рис. 9: Точка остановки

В отладчике GDB у меня была возможность просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Я выполнила 5 инструкций с помощью команды ‘stepi’ (сокращенно ‘si’) и отслеживала изменение значений регистров. (рис. 10) (рис. 11)

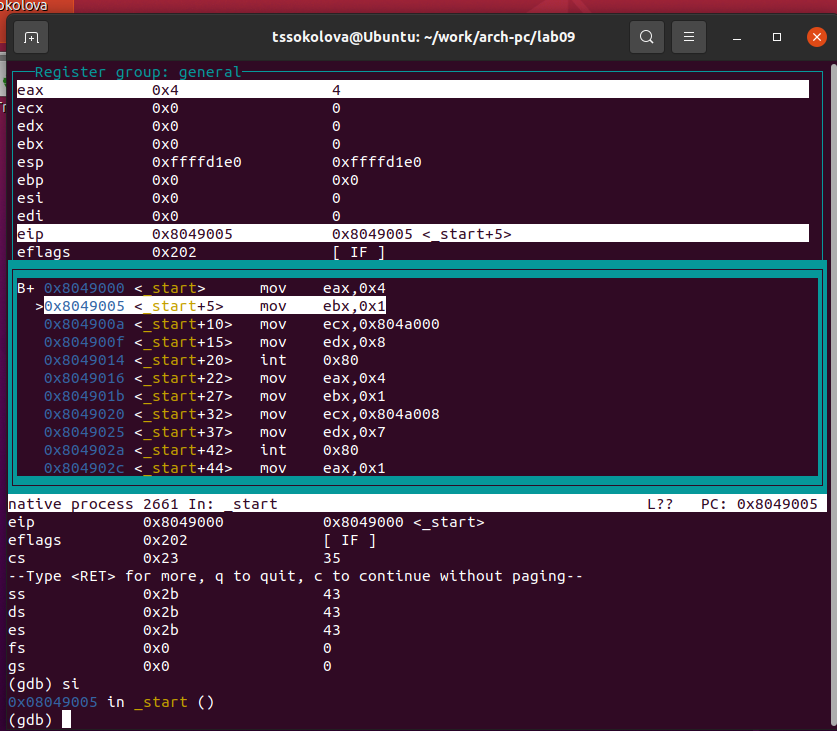


Рис. 10: Изменение регистров

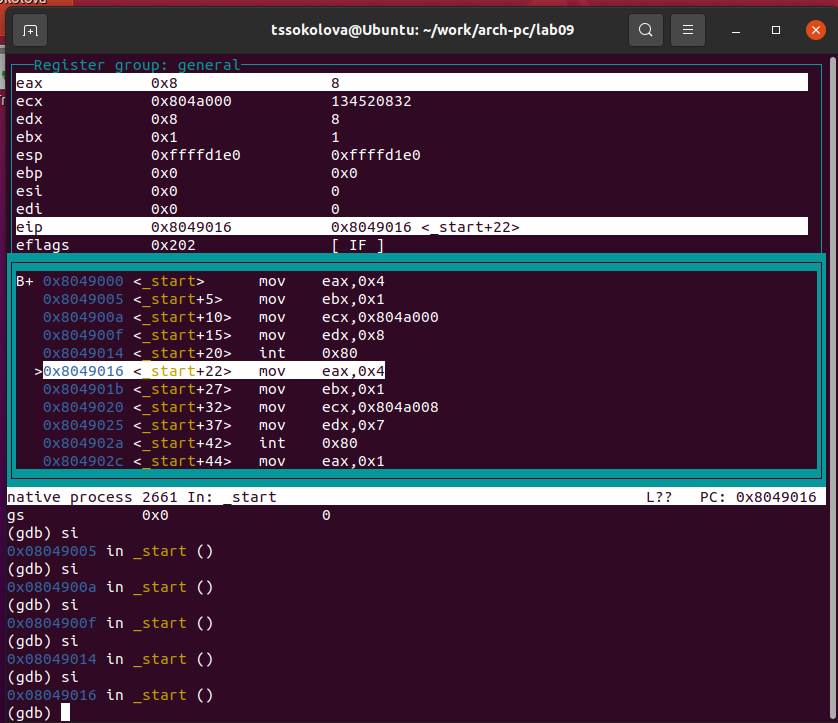


Рис. 11: Изменение регистров

Я также просмотрела значение переменной msg1 по имени и получила нужные данные. Чтобы изменить значение регистра или ячейки памяти, я использовала команду ‘set’, указав имя регистра или адрес в качестве аргумента. Я изменила первый символ переменной msg1. (рис. 12) (рис. 13)

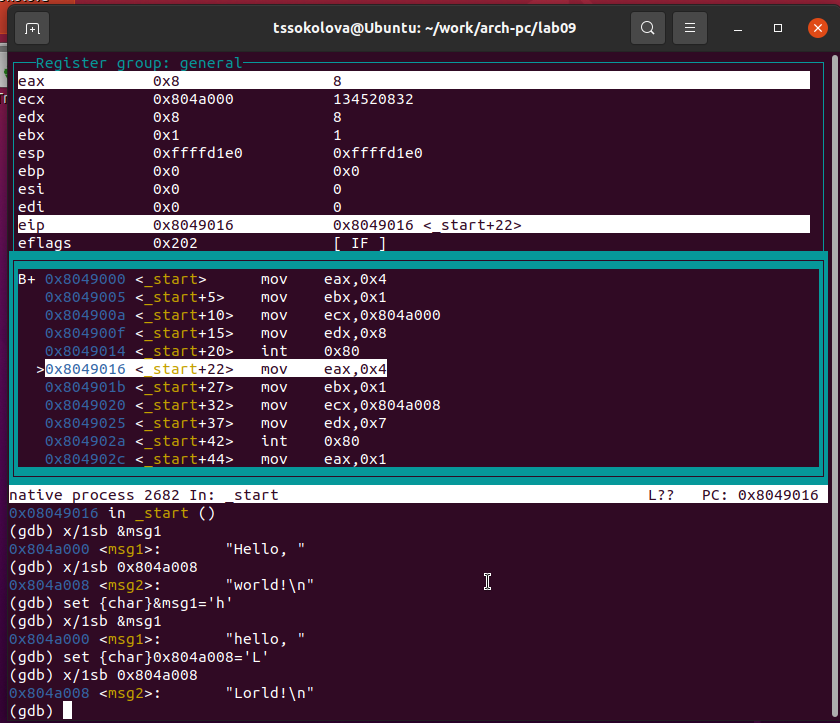


Рис. 12: Изменение значения переменной

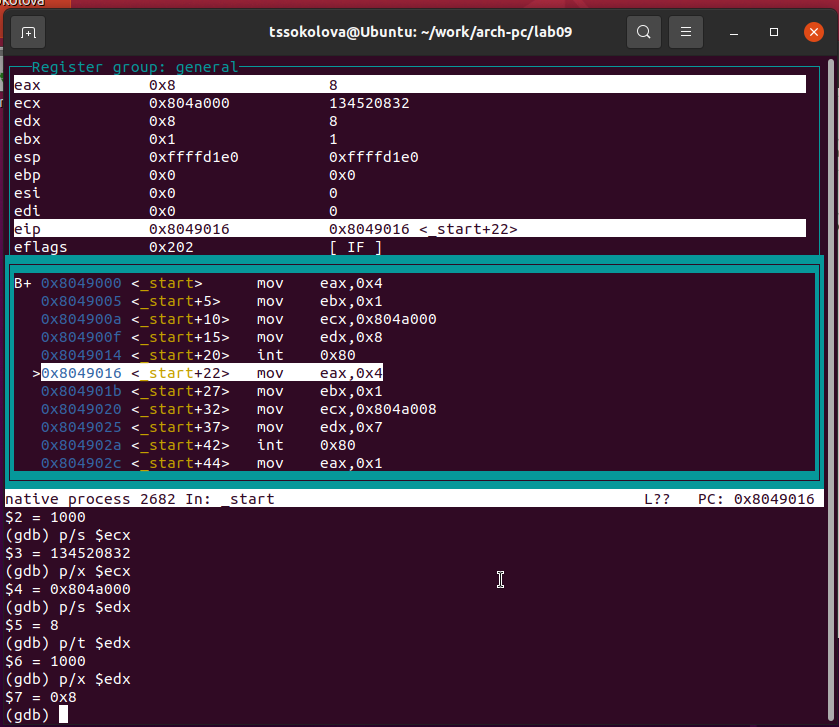


Рис. 13: Вывод значения регистра

Также, с помощью команды ‘set’, я изменила значение регистра ebx на нужное значение.(рис. 14)

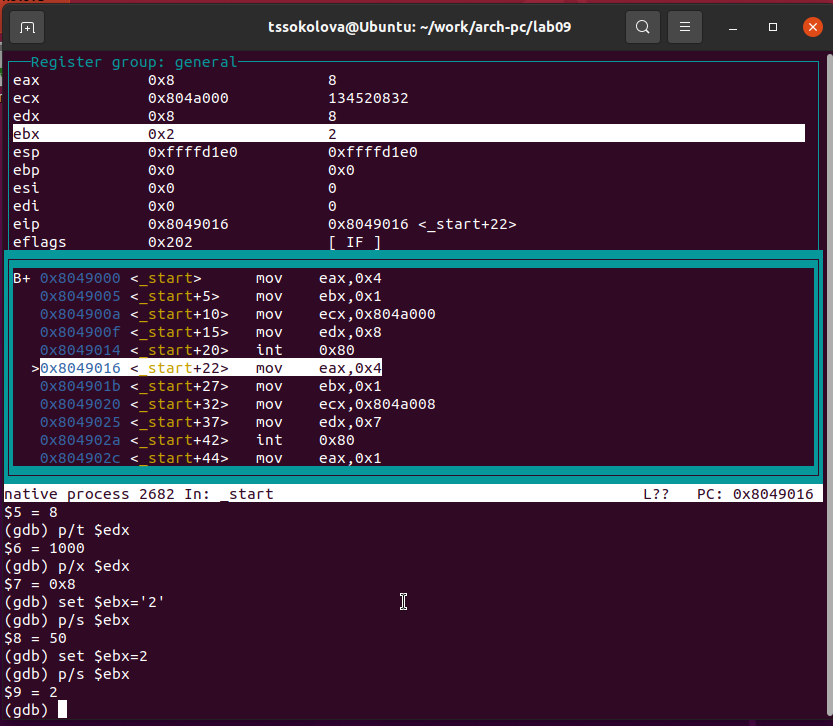


Рис. 14: Вывод значения регистра

Я скопировала файл lab8-2.asm, который был создан во время выполнения лабораторной работы №8. Этот файл содержит программу для вывода аргументов командной строки. Затем я создала исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в отладчик GDB, я использовала ключ –args и загрузила исполняемый файл в отладчик с указанными аргументами. Я установила точку останова перед первой инструкцией программы и запустила ее.

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 я увидела, что количество аргументов равно 5, включая имя программы lab9-3 и сами аргументы: аргумент1, аргумент2 и ‘аргумент 3’.

Я также просмотрела 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)

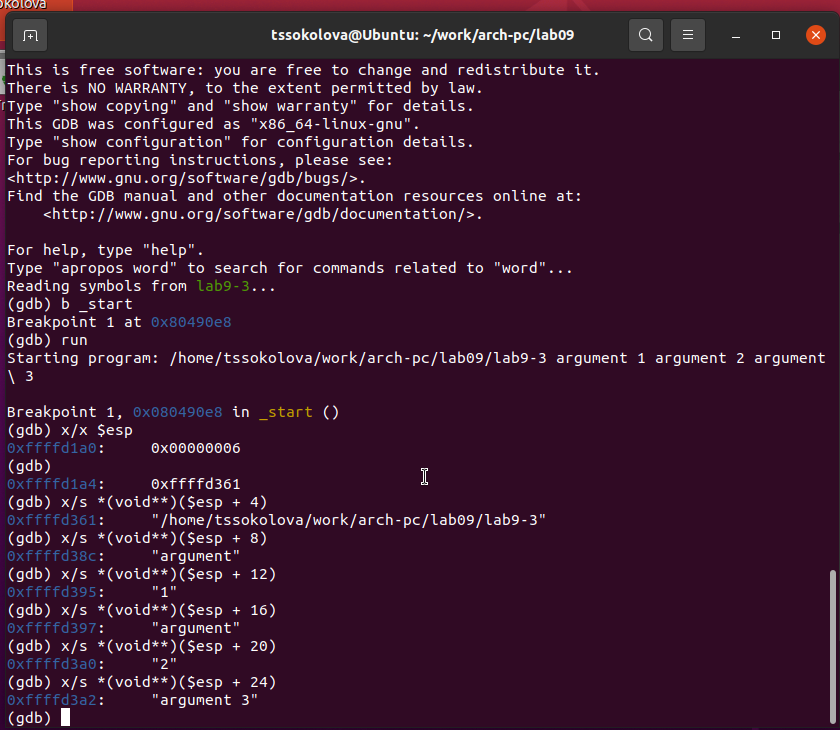


Рис. 15: Вывод значения регистра

## 2.3 Задание для самостоятельной работы

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

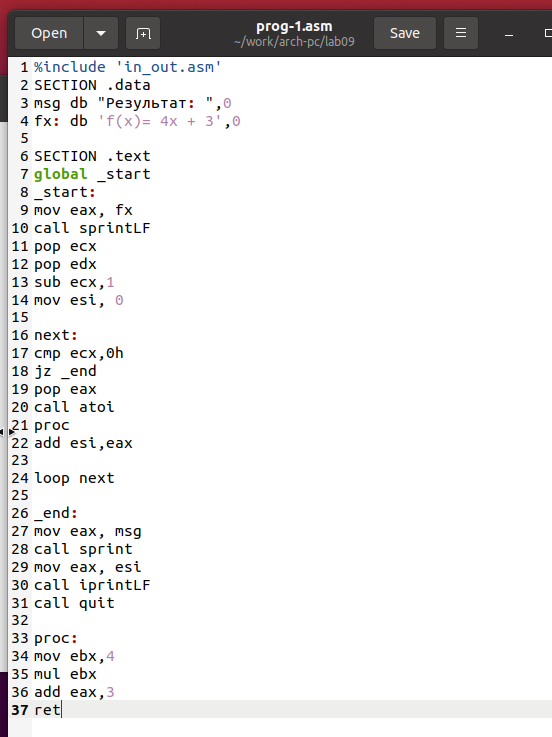


Рис. 16: Программа prog-1.asm

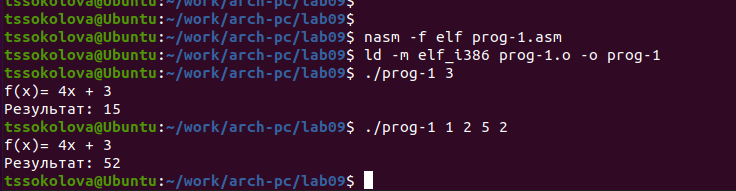


Рис. 17: Запуск программы prog-1.asm

В листинге представлена программа для вычисления выражения . Однако, при запуске программы, я обнаружила, что она дает неверный результат. Чтобы разобраться в причинах, я провела анализ изменений значений регистров с помощью отладчика GDB.

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

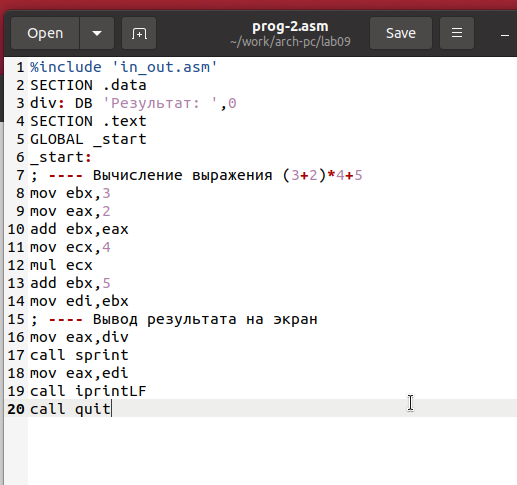


Рис. 18: Код с ошибкой

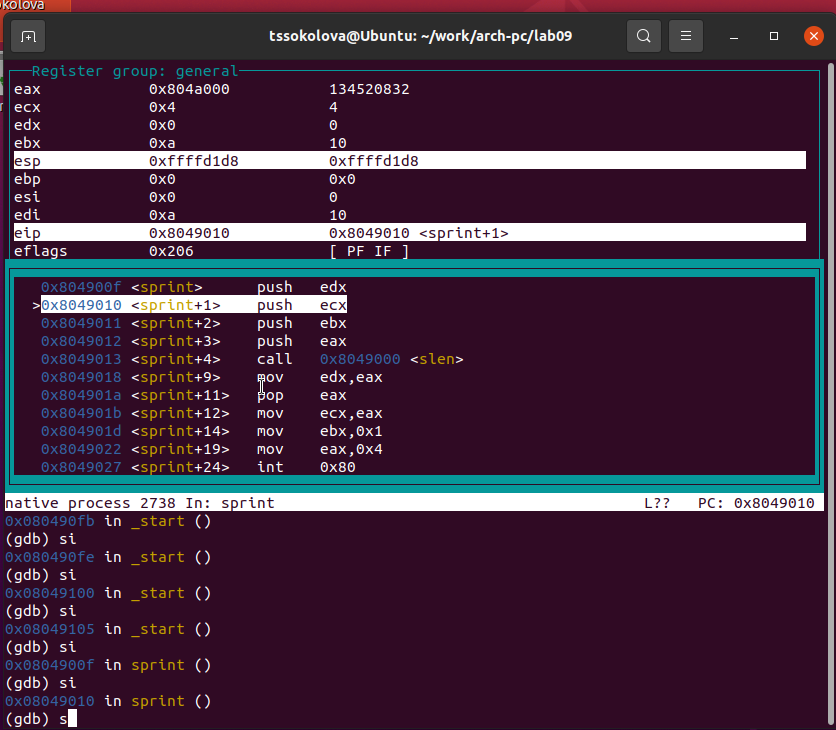


Рис. 19: Отладка

Отмечу, 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)

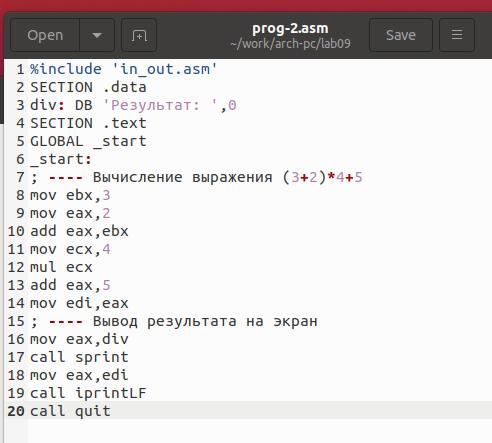


Рис. 20: Код исправлен

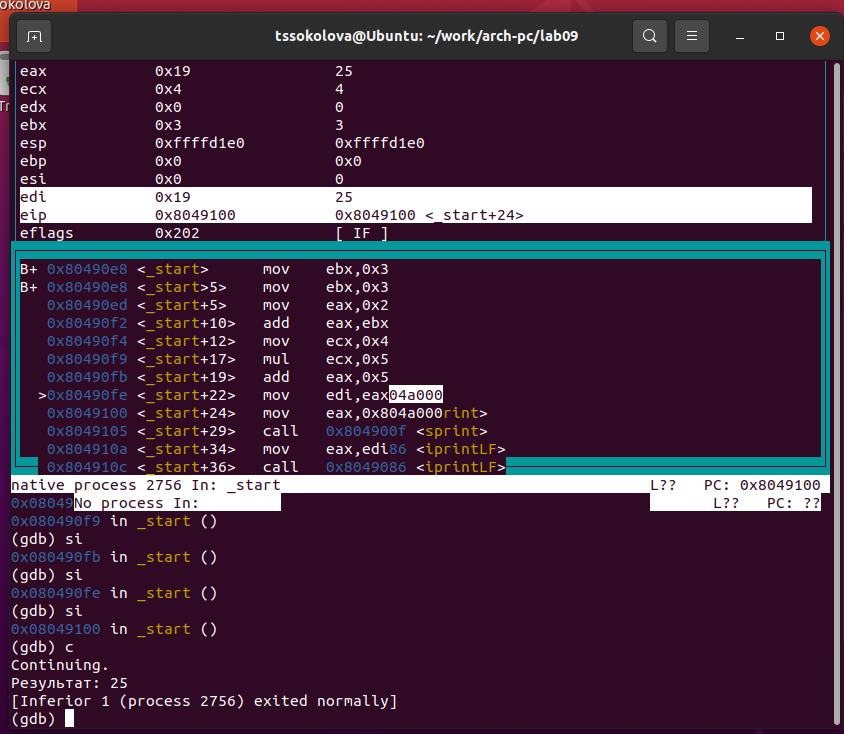


Рис. 21: Проверка работы

# 3 Выводы

Освоили работy с подпрограммами и отладчиком.