Лабораторная работа №10. Понятие подпрограммы. Отладчик GDB.

Дисциплина: Архитектура ЭВМ

Плескачева Елизавета Андреевна

Содержание

# 1 Цель работы

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

Создадим каталог и файл для выполнения лабораторной работы

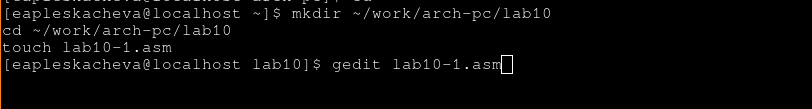


Рис. 1: создадние каталога и файла

Введем листинг 10.1 в lab10-1.asm

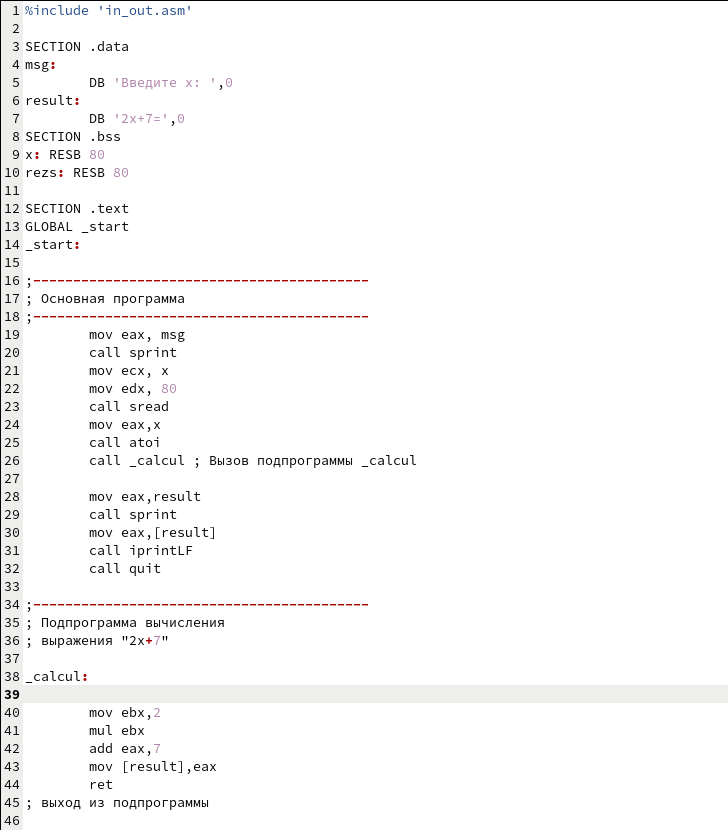


Рис. 2: Листинг 10.1 в gedit

Скомпилируем и запустим программу.

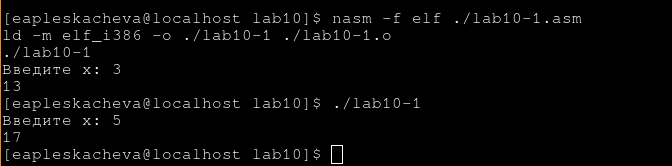


Рис. 3: Запуск программы lab10-1

Программа выводит 13 и 7 для 3 и 5, потому что функция 2x + 7

## 1.1 Создание подпрограммы

Создадим подпрограмму, которая будет вычислять g(x) = 3x - 1

Добавим ее вниз нашего кода и в вычисления

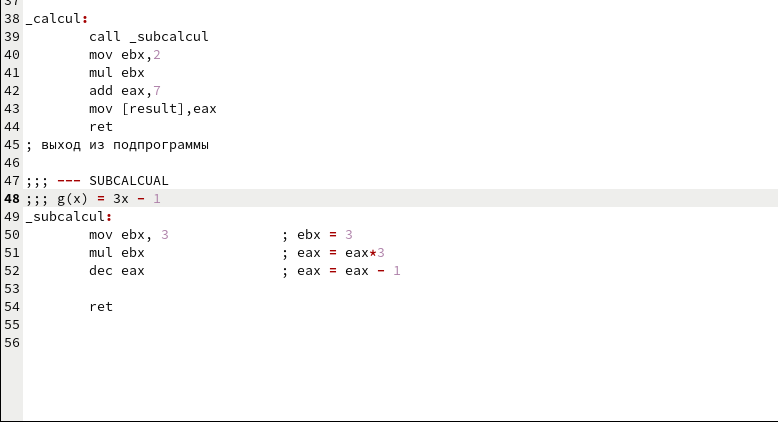


Рис. 4: Добавление подпрограммы

Снова запустим и посмотрим, как изменился результат

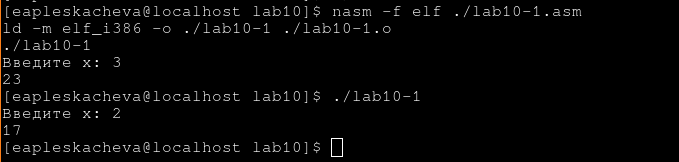


Рис. 5: Запуск измененной программы lab10-1

f(g(x)) = 2(3x - 1) + 7 = 6x + 5 f(g(3)) = 18 + 5 = 23 f(g(2)) = 12 + 5 = 17

Результат верный

## 1.2 Отладка с помощью GDB

Скопируем листинг 10.2 в lab10-2.asm, скомпилируем и запустим программу.

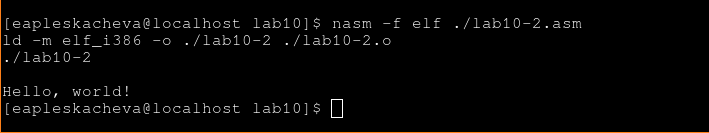


Рис. 6: Вывод lab10-2.asm

Программа выводит на экран сообщение

Теперь создадим листинг для lab10-2 и откроем исполняемый файл через GDB

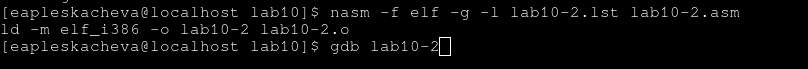


Рис. 7: Создание листинга и добавление отладочной информации, открытие через GDB

Открыв программу в gdb запустим ее, написав run

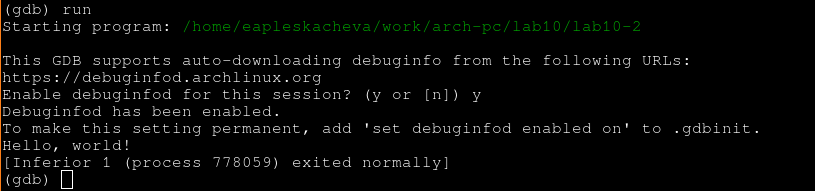


Рис. 8: Запуск программы внутри GDB

Поставим точку остановки на \_start и запустим программу

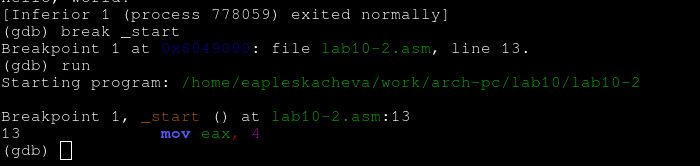


Рис. 9: Точка остановы на start

Теперь програмаа запустилась, но остановилась на \_start

### 1.2.1 ПРосмотр дизасемблированного кода

Напишем dissassemble \_start что бы посомотреть дизассемблированный код

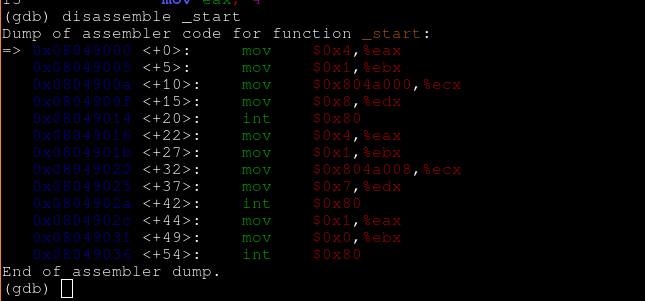


Рис. 10: Просмотр дизассемблированного кода

Изменим отображение на intel

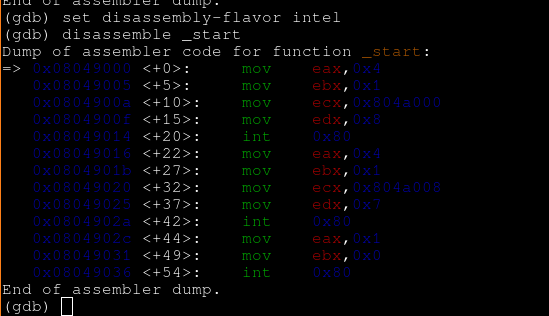


Рис. 11: Изменение отображения на синтаксис intel

Теперь программа выглядит как на NASM

### 1.2.2 Режим псевдографики

Напишем layout asm layout regs и run что бы включить режим псевдографики и запустить прогроамм

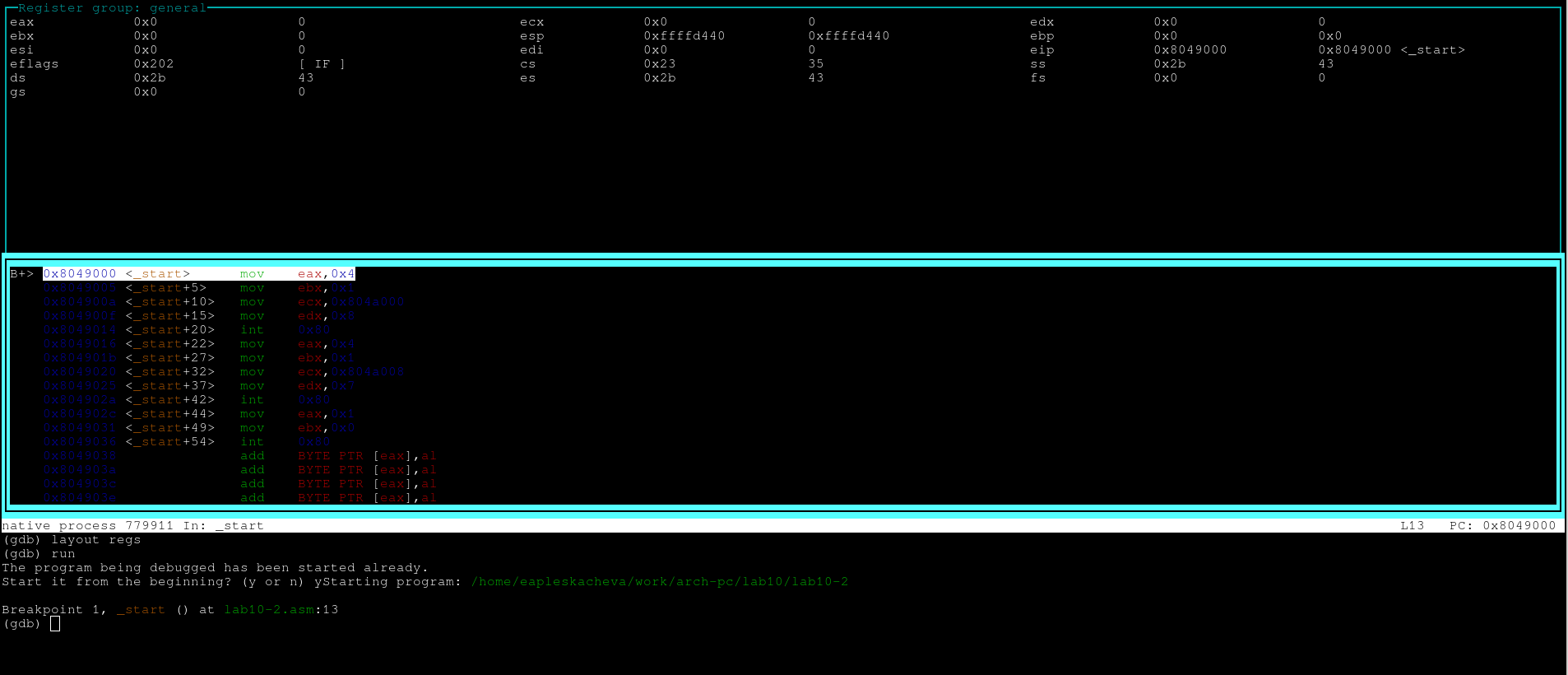


Рис. 12: Включение псевдографики

### 1.2.3 Информация о точках остановки

Просмотрим информацию о поставленных точках остановки, написав i b

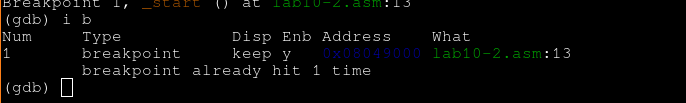


Рис. 13: Информация об установленных точках

Поставим точку остановки на 0x8049031

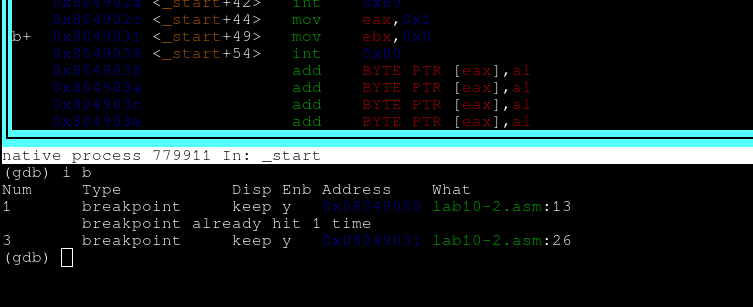


Рис. 14: Вывод точек, после добавления новой

Просмотрим информацию о точках остановки и увидим там новую точку

### 1.2.4 Работа с данными программы

Вывведем содеримое переменной msg1

Рис. 15: Вывод msg1

Рис. 15: Вывод msg1

Выведем содержимое msg2 по адресу

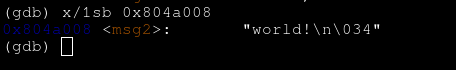


Рис. 16: Вывод содержимого msg2 по адресу

### 1.2.5 Изменение данных в памяти

Заменим “H” на “h” в msg1

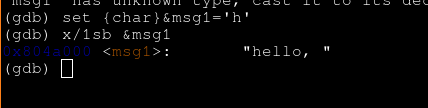


Рис. 17: замена символа в msg1

Изменим любой символ в msg2

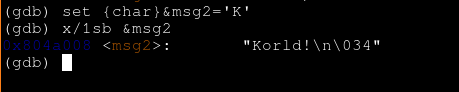


Рис. 18: Замена W на K в msg2

### 1.2.6 Изменение содержимого регистра

Изменим значение ebx на ‘2’ и на 2

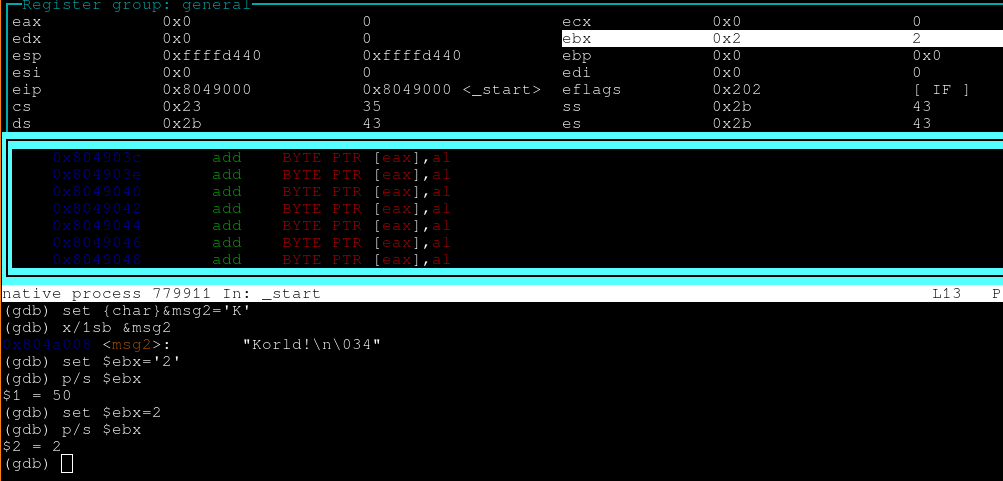


Рис. 19: Изменение содержимого eax

В начале программа выводит код символа 2, а потом число 2

Продолжим исполнение программы и завершим ее

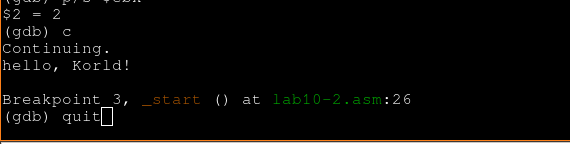


Рис. 20: Завершение программы

Заметим, что выводимое сообщенеи изменилось

## 1.3 Обработка аргументов коммандной строки

Скопируем lab9-2.asm в lab10-3.asm создадим исполняемый файл с отладочной информацией и откроем программу через gdb добавив аргументы.

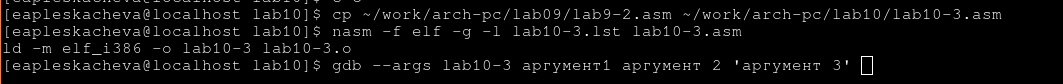


Рис. 21: Копирование и открытие прогарммы с введенными аргументами

Поставим точку остановы на \_start и запустим прогармму

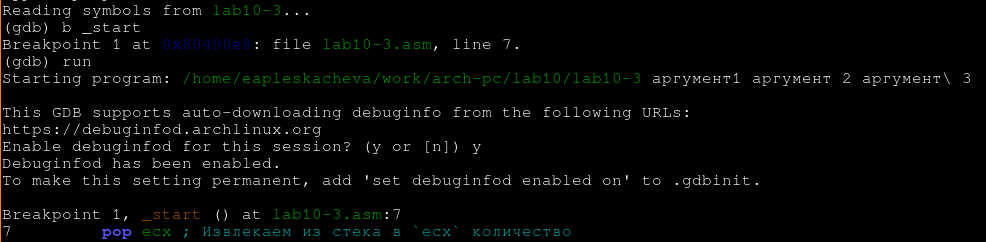


Рис. 22: Запуск lab10-3 с точкой остановы

Выведем значене esp

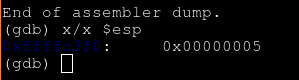


Рис. 23: Значение esp

В esp лежит 5, потому что аргументов 5

Просмотрим остальное содержиме стека



Рис. 24: Просмотр содержимого стека

Мы увеличиваем, значение адресса на 4 потому что столько занимет размер указателя на аргумент из стека.

Продолжим программу и завершим ее

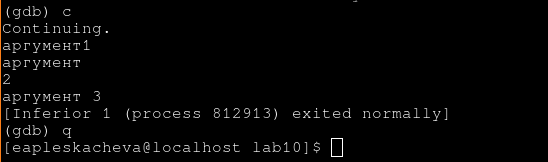


Рис. 25: Завершение программы

# 2 Задания для самостоятельной рабоыт

## 2.1 Написание подпрограммы

Изменим программу из самостоятельно лабораторной 9, представив функцию как подпрограмму.

Напишем эту программу в низу кода

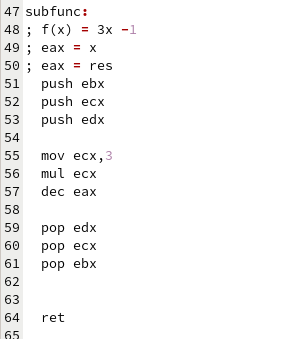


Рис. 26: Подпрограмма

Запустим и проверим правильность программы

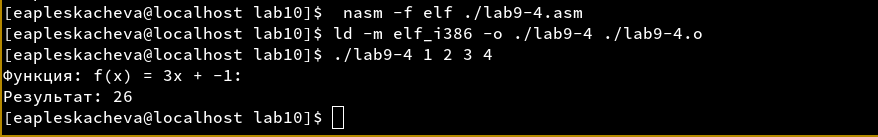


Рис. 27: Измененная lab9-4.asm

Результат верный

## 2.2 Поиск ошибки в программа через GDB

Введем в lab10-4.asm код из листинга 10.3, скомпилируем его и запустим через GDB

Так как происходит умножение, то после исполнения mul в eax должен оказаться результат вычислени. Там должно лежать (3+2)\*4 = 20

Пройдемся до исполнения этой части программы

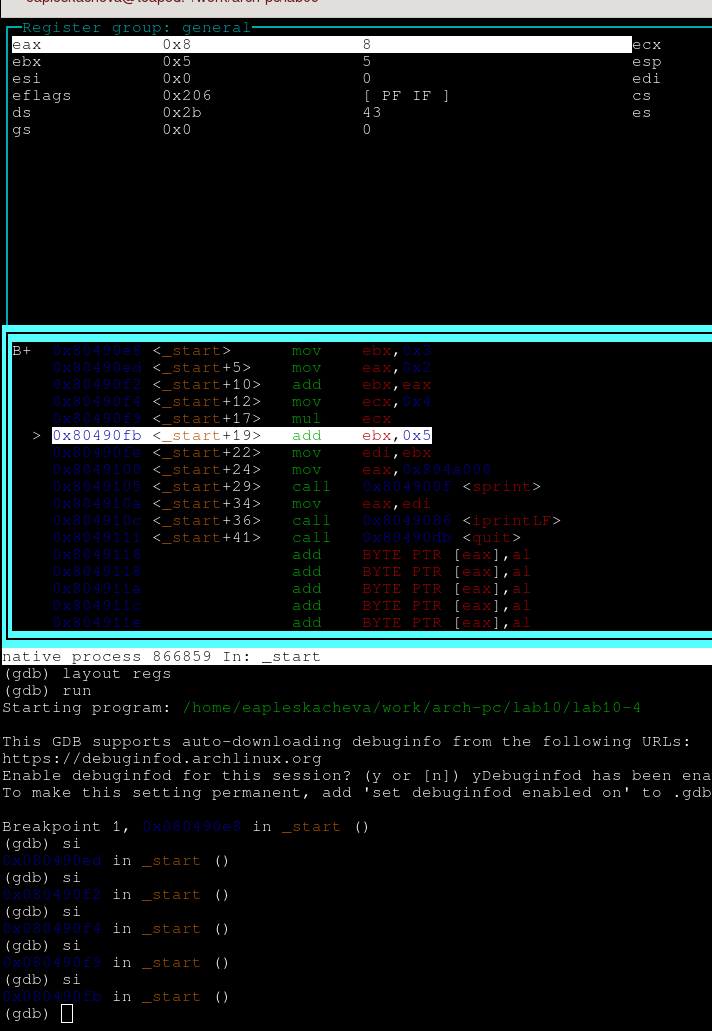


Рис. 28: Наблюдение за изменением eax

После исполнения умножения в eax лежит 8. Это потому что мы умножаем eax на ecx, а сложение производилось в ebx

Изменим код так, что бы программа работала верно

Рис. 29: Изменение неправильного кода

Рис. 29: Изменение неправильного кода

Теперь суммирование будет в eax

Снова запустим GDB и дойдем до того же момента.



Рис. 30: Просмотр eax измененой программы

Теперь в eax лежит 20

Будем просматривать прогармум дальше

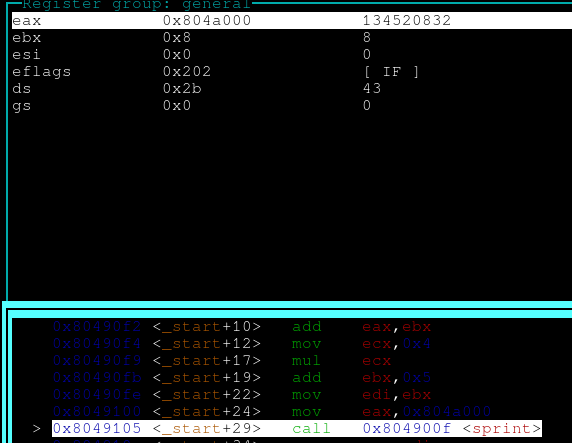


Рис. 31: Дальнейший просмотр программы

Заметим, что в eax записывается сообщение для вывода, а само значение никуда не сохраняется, а так же 5 прибавляется к ebx, в котором до этого лежало 3, поэтому теперь там 8

Изменим програму так что бы она работала корректно

Рис. 32: Изменение программы

Рис. 32: Изменение программы

Запустим программу теперь и понаблюдаем за ее исполнением

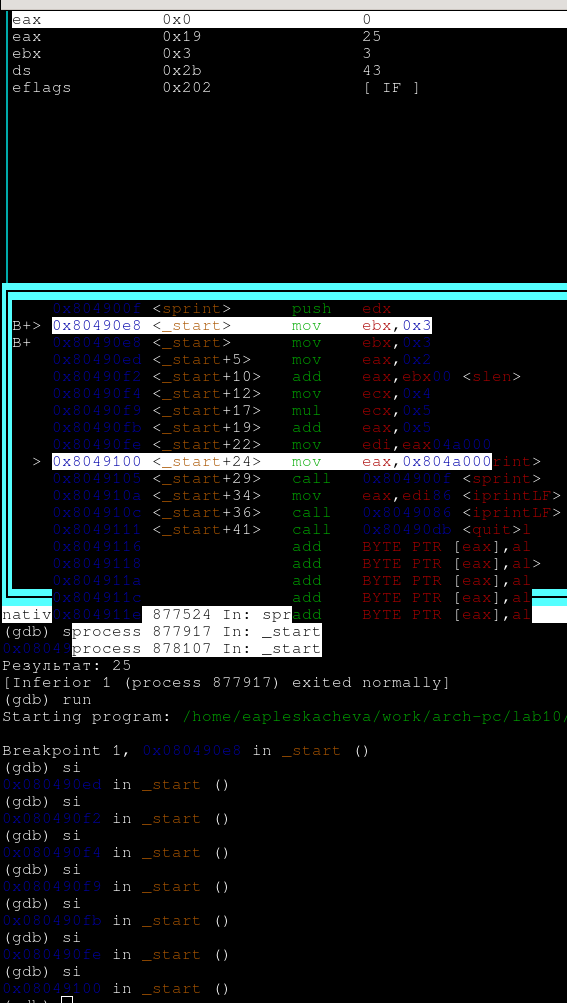


Рис. 33: Просмотр eax

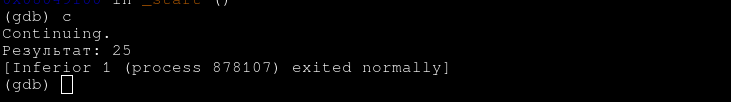


Рис. 34: Завершенеи программы

Теперь программа вычисляет значение правильно, на этом ее можно завершить

# 3 Выводы

Мы приобрели навыки написания программ с использованием подпрограмм и ознакомились с процессом отладки через программу GDB и научились пользоваться его основными возможностями.