Лабораторная работа №10. Понятие подпрограммы. Отладчик GDB.

Дисциплина: Архитектура ЭВМ

Осокин Георгий Иванович НММбд-02-22

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы

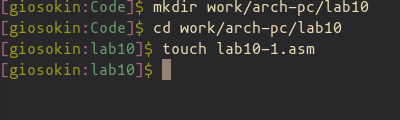


Рис. 1: создадние lab10

## 2.1 Листинг с примером выполнение подпрограммы

Скопируем листинг 10.1 и запустим его

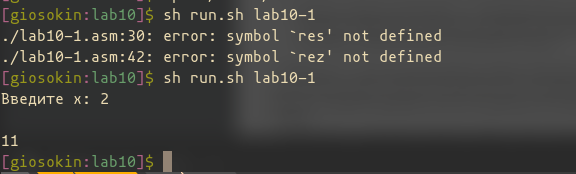


Рис. 2: Исполнение lab10-1.asm

Видим, что программа исполняется с ошибкой.

### 2.1.1 Исправление листинга

Найдем ошибки в коде и исправим их.

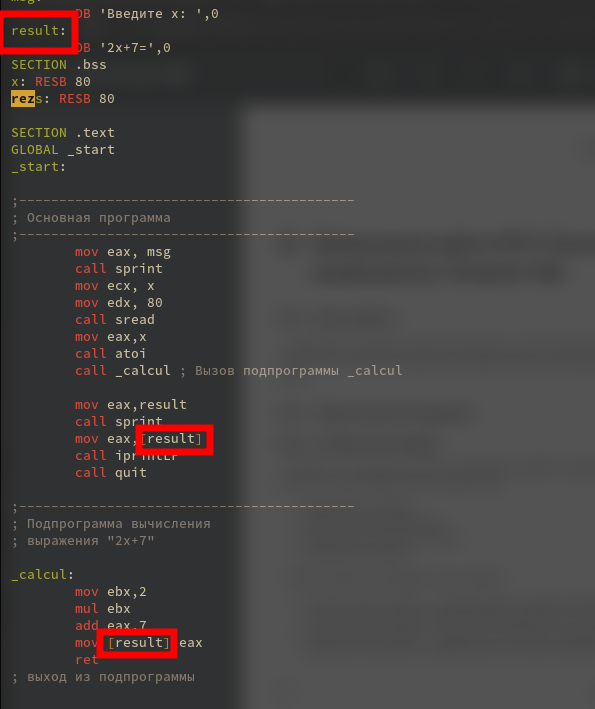


Рис. 3: Ошибки в листинге 10.1

Запустим код заново и проверим его

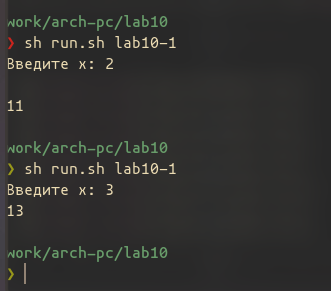


Рис. 4: Вывод исправленного lab10-1

Как видим, теперь код выполняется ожидаемо.

Добавим вниз кода подпрограмму

34 ;------------------------------------------  
 35 ; Подпрограмма вычисления  
 36 ; выражения "2x+7"  
 37   
 38 \_calcul:  
 39 call \_subcalcul  
 40 mov ebx,2  
 41 mul ebx  
 42 add eax,7  
 43 mov [result],eax  
 44 ret  
 45 ; выход из подпрограммы

Испполним его и посмотрим на результат

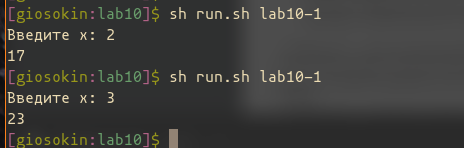


Рис. 5: Запуск измененного кода lab10-1

## 2.2 Отладка программы через GDB

Введем в файл lab10-2.asm код из листинга 10.2

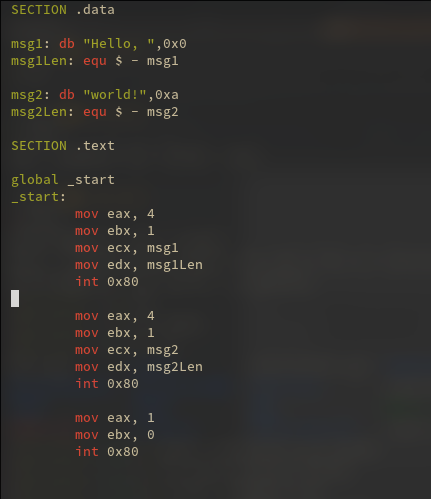


Рис. 6: Листинг 10.2

Создадим баш скрипт gen\_debug.sh что б дальше было удобнее работать с созданием исполняемых файлов с отладочной информацией

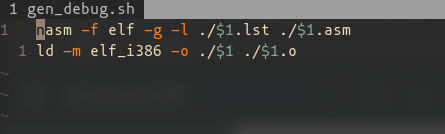


Рис. 7: Баш скрипт

Создадим с помощью этого скрипта исполняемый файл и откроем его через gdb введя gdb lab10-2

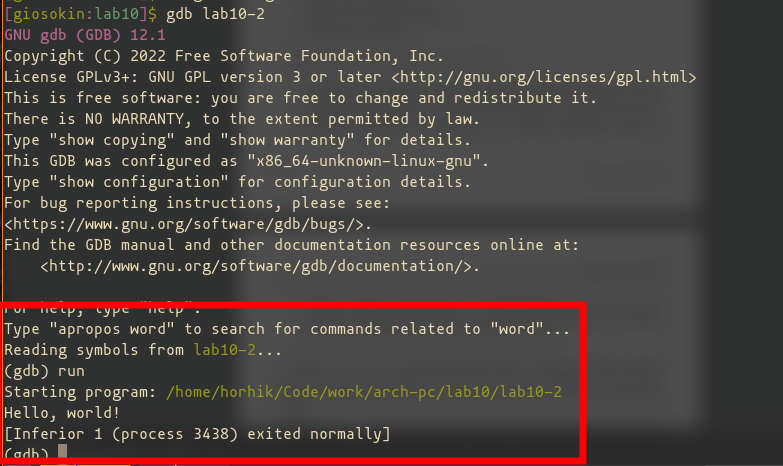


Рис. 8: Первое открытие GDB

Запустим run и увидим, что код исполнился и вывел Hello world на экран.

### 2.2.1 Точка остановы

Поставим отчку остановы на метке \_start и запустим программу

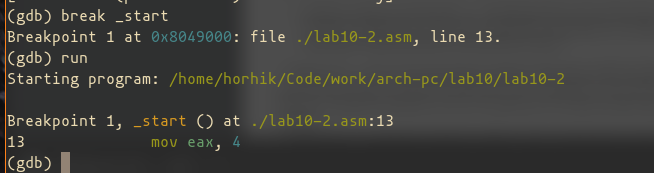


Рис. 9: Первая точка остановы

Исполнение программы остановилось на метке \_start

### 2.2.2 Вывод дисассемблированного кода в формате Intel

Выведем дизассемблированный код программы

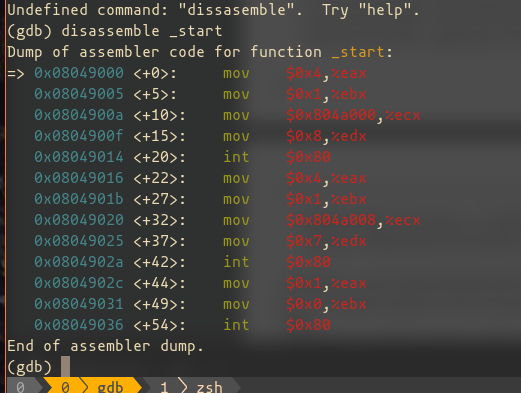


Рис. 10: Код в формате AT&T

Выведенный дизассемблированный код в формате AT&T

Изменим его на формат отображение от Intel



Рис. 11: Изменение отображения

### 2.2.3 GDB TUI

Включи режим псевдографики.

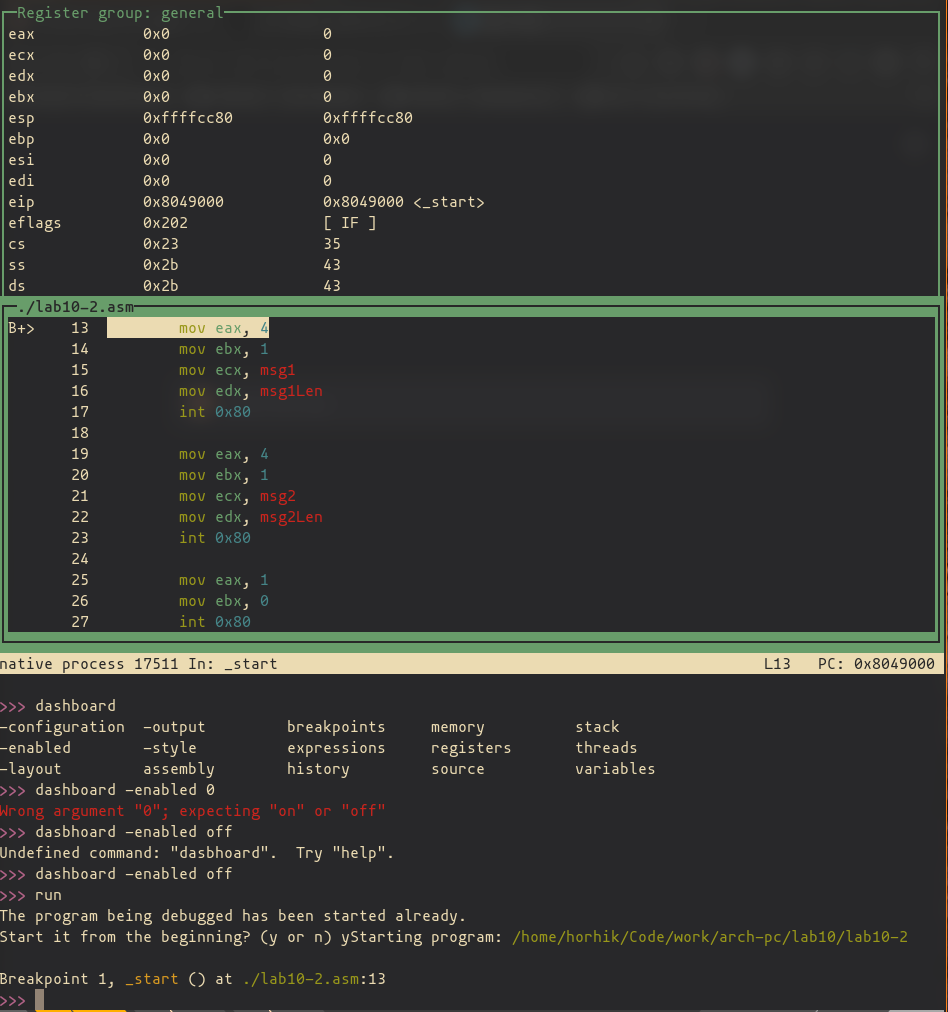


Рис. 12: Режим псевдографики

Также, мы можем переключаться между этими режимами, нажимая Ctrl+a Ctrl+a

### 2.2.4 Точка остановы по адресу

Поставим точку остановы на адресс 0x8049031

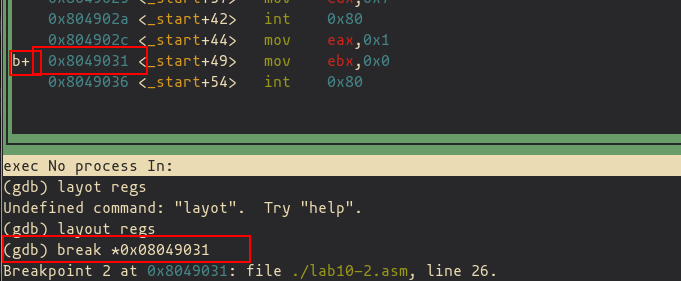


Рис. 13: Новая точка остановы

Выведем информацию о поставленных точках остановы

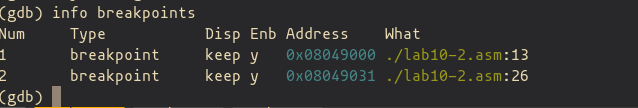


Рис. 14: Информация о точках остановы

Можем вывести информацию более короткой коммандой i b

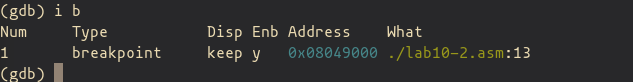


Рис. 15: i b

### 2.2.5 Просмотр содержимого регистров

Запустим программу заново и исполним комманду si 5 раз, наблюдая как меняются регистры.

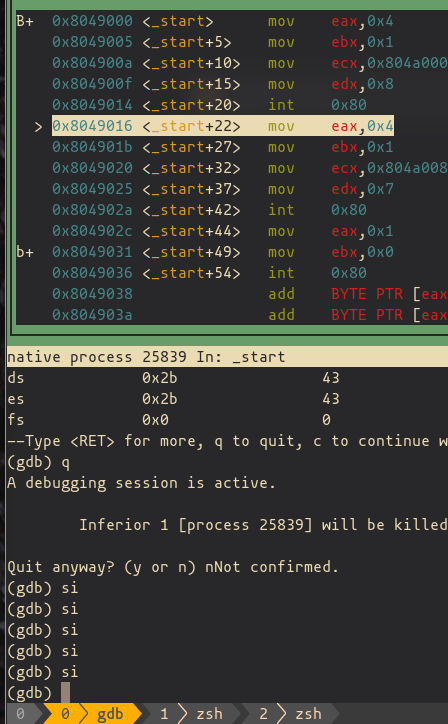


Рис. 16: использование stepi 5 раз

В конце мы видим, что в eax, ebx, ecx и edx уже не нулевые значения.

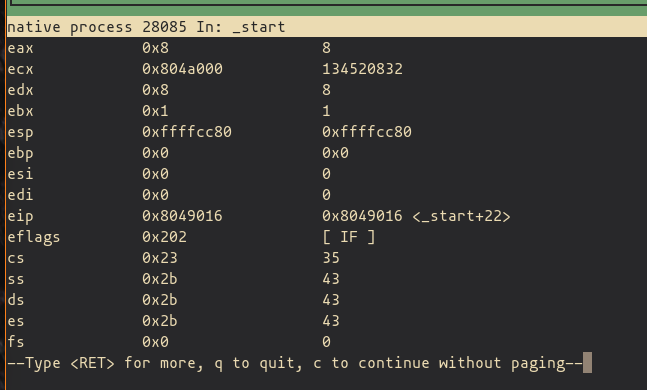


Рис. 17: Измененные регистры

Введем x/1sb &msg1 что бы посмотреть, какие данные лежат в msg1

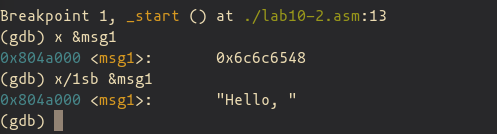


Рис. 18: Просмотр содержимого в переменной

ПРосмотрим содержимое &msg2 указав его аддрес в памяти

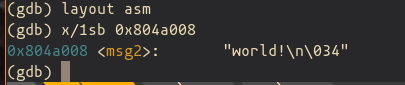


Рис. 19: Просмотр содержимого в переменной по адрессу

Просмотрим содержимое инструкции mov ecx, msg2, находящейся по адрессу 0x8049020

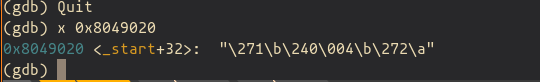


Рис. 20: Просмотр содержимого строчки по адрессу

### 2.2.6 Изменение значений в памяти

Изменим значение в msg1

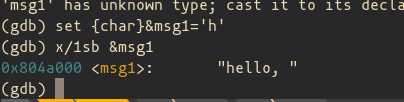


Рис. 21: Изменение переменной

У нас меняется только первая буква, потому что меняется только первый байт

Заменим две буквы в переменной &msg2

Мы сдвигаемся на один байт, поэтому +1

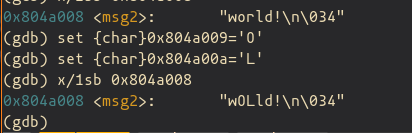


Рис. 22: Изменение двух символов

### 2.2.7 Просмотр значений регистров

Запустим программу заново и просмотрим как меняется регистр eax и выведем значение регистра.

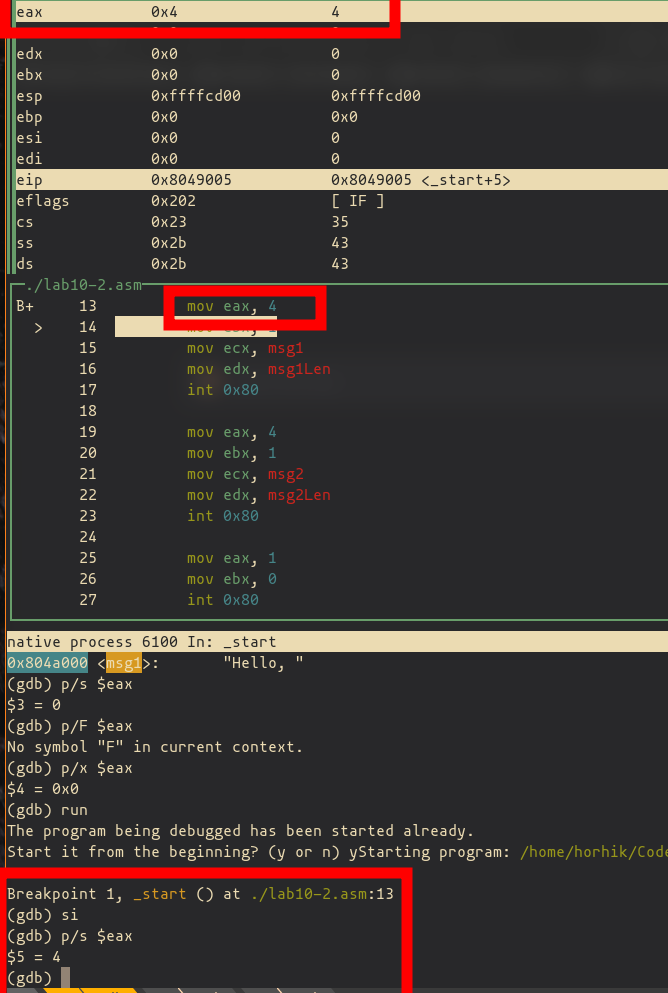


Рис. 23: Вывод значения регистра

Прсмотрим как меняется регистр edx и сделаем вывод в нескольких фоматах

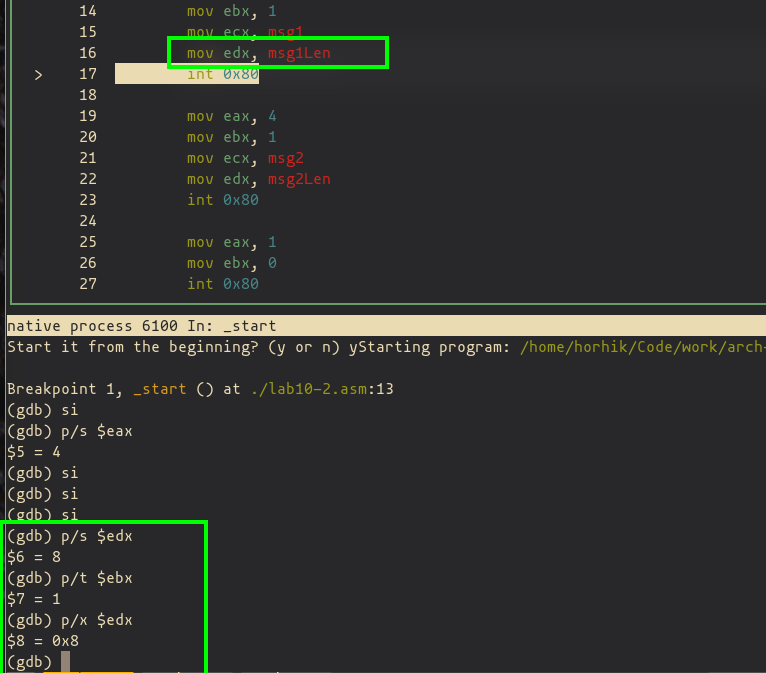


Рис. 24: Вывод значения регистра в разных форматах

### 2.2.8 Изменение значений регистров

Изменим значение регистра ebx с помощью коммагды set

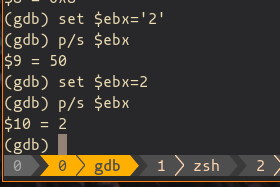


Рис. 25: Изменение ebx

Вывод разный, потому что в первом случае мы выводим не число два, а код числа два в таблице ASCII

### 2.2.9 Завершение программы

Введем c что бы продолжить выполнение программы.

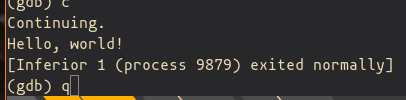


Рис. 26: Продолжение выполнения программы

Программа завершила свое выполнение. Можем выйти, нажав q

## 2.3 Обработка аргументов коммандной строки

Скопируем файл lab9-2.asm в lab10-4.asm

Откроем этот файл в gdb с флагом аргументов

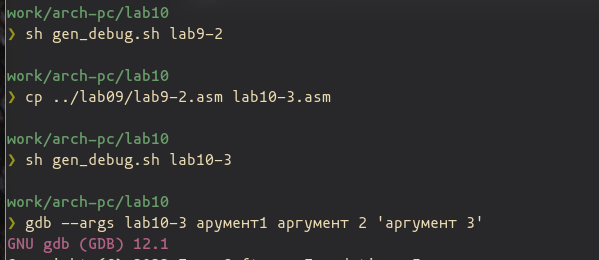


Рис. 27: GDB с флагом аргументов

Поставим точку остановы



Рис. 28: Точка остановы

Запустим и выведем значение регистра esp.

После выведем все введенные аргументы`

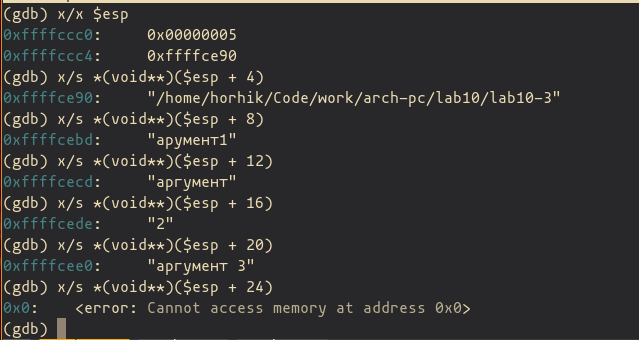


Рис. 29: Вывод значение регистра и аргументов

Как видем, когда аргументы закончились, вывелась ошибка

Шаг изменения адресса равен 4, потому что столько места зарезервированно на стеке на указатель введенного аргумента

# 3 Задания для самостоятельной работы

## 3.1 Вычисление значения функции как подпрограмма

В 9 лабораторной я уже выделил эту функцию как подпрограмму. Был создан файл my-function.asm

1 ; f(x) = 17 + 5x  
 2 ; eax = x  
 3 ; eax = res  
 4 magic\_function:  
 5   
 6 push ebx  
 7 push ecx  
 8 push edx  
 9   
 10 mov ecx,5  
 11 mul ecx  
 12 add eax, 17  
 13   
 14 pop edx  
 15 pop ecx  
 16 pop ebx  
 17   
 18   
 19 ret  
 20   
 21

Вызов этой функции в файле lab9-4.asm

30 call atoi ; преобразуем символ в число  
 31   
 32   
 33 call magic\_function  
 34   
 35 add esi,eax ; добавляем к промежуточной сумме  
 36

## 3.2 Анализ программы с сошибкой

Запустим код программы через GDB.

Так как умножение происходит в регистр EAX, будем ожидать, что перед умножением в EAX должно лежать (3+2) = 5

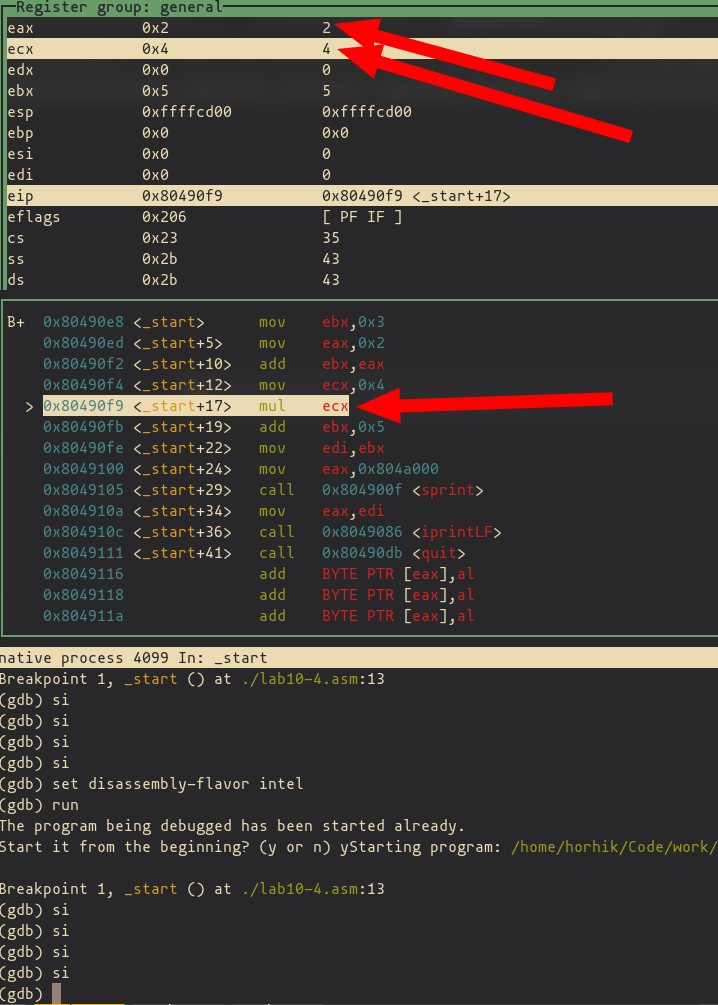


Рис. 30: Остановка перед mov

Как видим, в EAX не 5

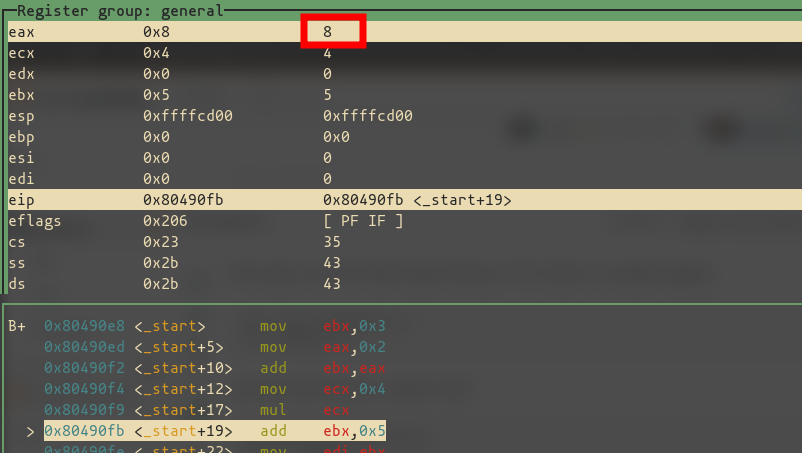


Рис. 31: После mov

Вычисление не те, потому что мы перемещаем в EAX 2 и после ничего с ним не делаем.

Мы можем заметить, что нам нужно изменить срочку со сложением ecx, eax на eax, ecx, что бы результат сложения хранился в EAX.



Рис. 32: Измененная строчка

Запустим заново

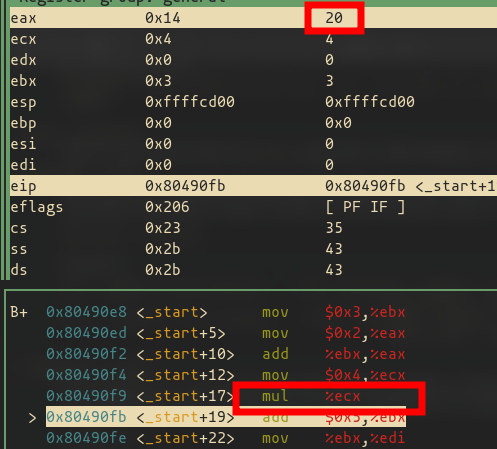


Рис. 33: Результат измененный программы после mul

Как видим, теперь результат верный

Продолжим дебажить.

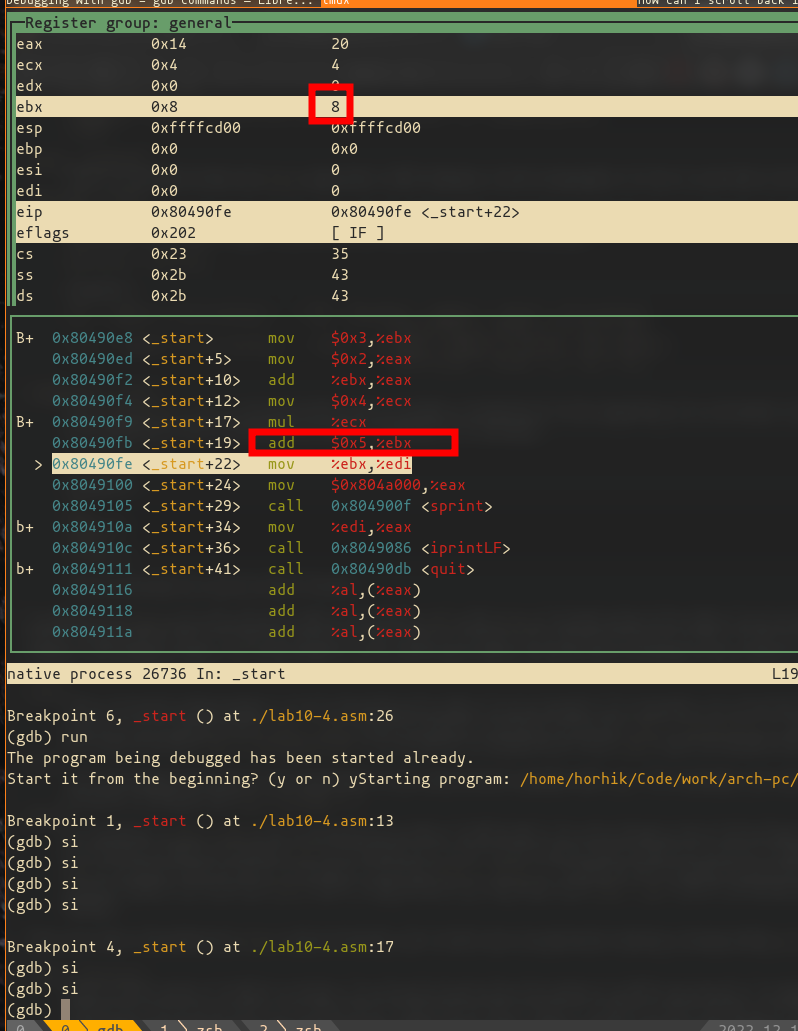


Рис. 34: Наблюдение изменений в регистре EBX

Как мы видим, число 5 добавилось в регистр EBX, а не в EAX.

А после это значение переносится в EDI. Обычно этот регист отвечает за ввод.

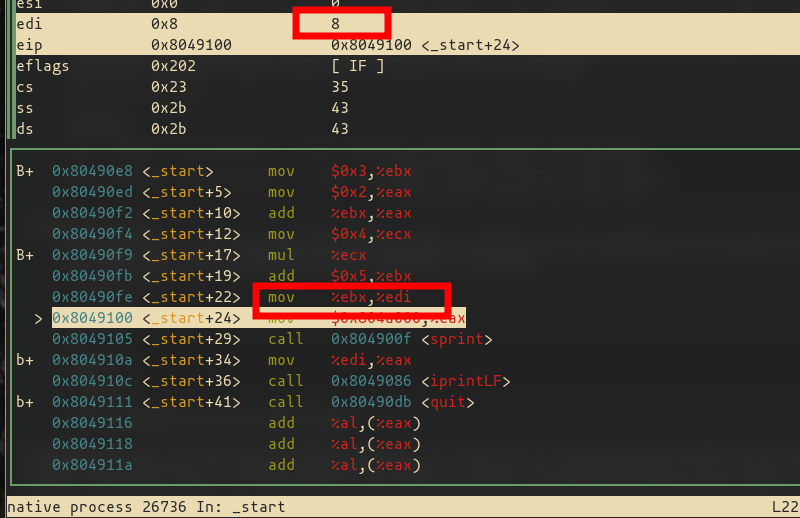


Рис. 35: Измененный EDI

После Содержимое EDI выводится на экран, поэтому следует предположить, что добавление к регистру EBX, а не к EAX - не то поведение программы, которое мы ожидаем.

Изменим программу, заменим ebx на eax.

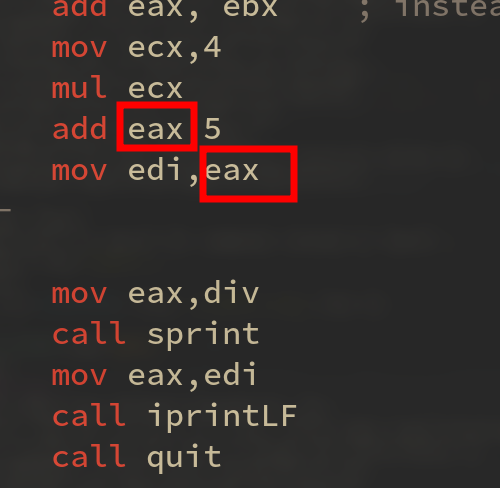


Рис. 36: Замена EBX на EAX

Запустим программу заново и посмотрим как теперь изменилось ее поведение.

Исполним continue, пропустив все точки остановы. На экран выводится 25 - ожидаемый результат

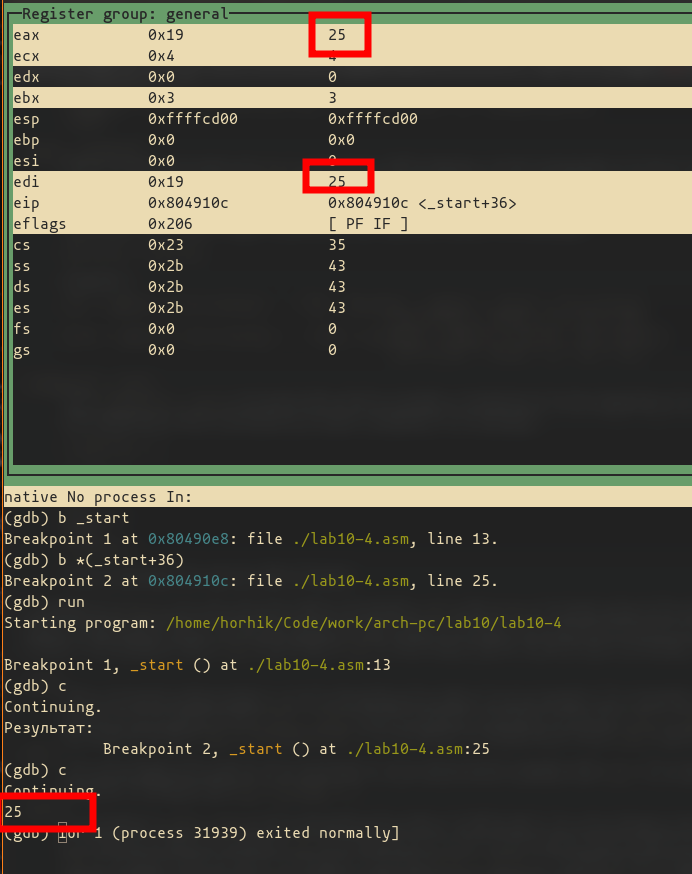


Рис. 37: Финальный результат

Теперь запустим программу не через GDB

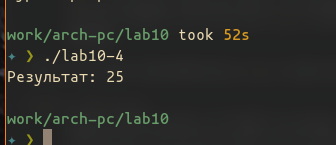


Рис. 38: Исполнение lab10-4

Вывелся правильный результат.

# 4 Выводы

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