Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Коровкин Никита Михайлович

Содержание

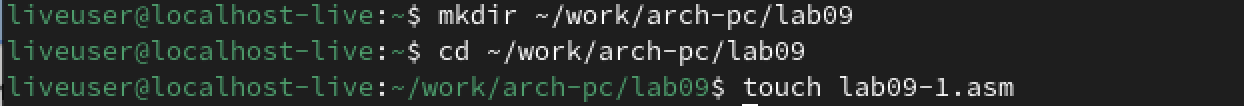
# 1 Цель работы

Ознакомиться с понятием подпрограмм в Ассемблере и научиться использовать подпрограммы на практике. Ознакомиться с отладчиком gdb и научиться использовать его

|

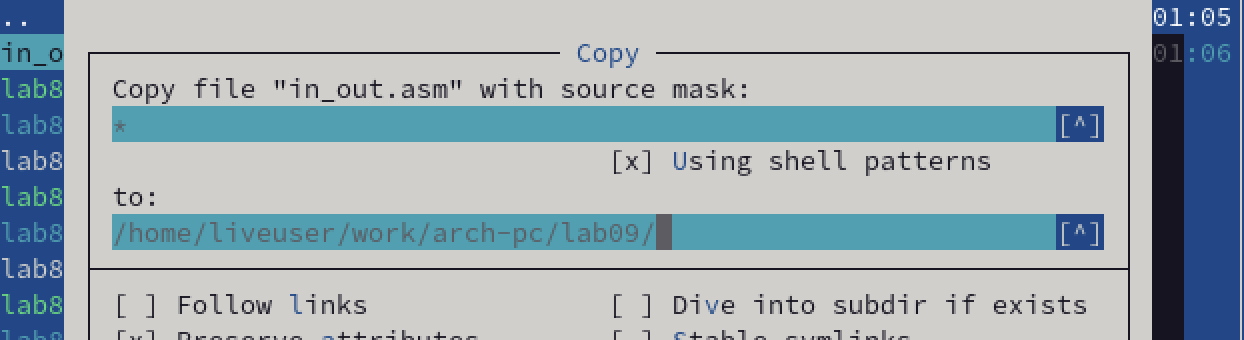
# 2 Выполнение лабораторной работы

Сперва создадим рабочую директорию и первый файл: с которым мы будем работать.(рис.1)



Создание рабочей директории и файла lab9-1.asm

Далее подключим in\_out.asm, перенеся его из папки прошлой лабораторной работы.(рис.2)



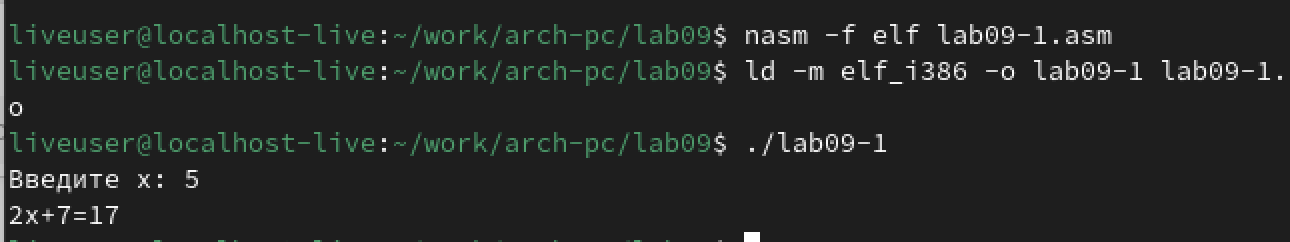
Копирование файла in\_out.asm в рабочую папку

Откроем файл и вставим код из первого листинга.(рис.3)



Вставляем код из первого листинга

Теперь соберем файл и запустим его.(рис.4)



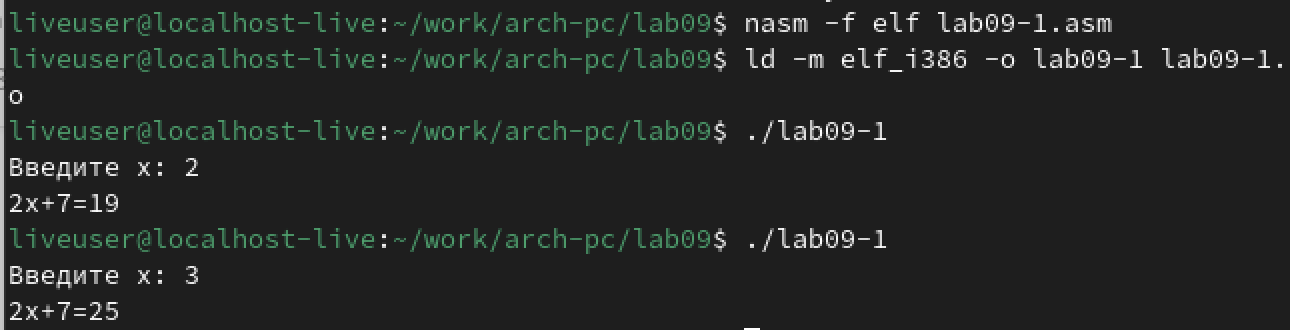
Запуск файла

Затем изменим файл, чтобы в подпрограмме была ещё одна подпрограмма, вычисляющая значение g(x), которая будет передавать значение в первую подпрограмму, которая бы уже вычислила значение f(g(x)).(рис.5)



Изменение кода файла lab9-1.asm

Запустим программу еще раз.(рис.6)



Повторный запуск файла lab9-1.asm

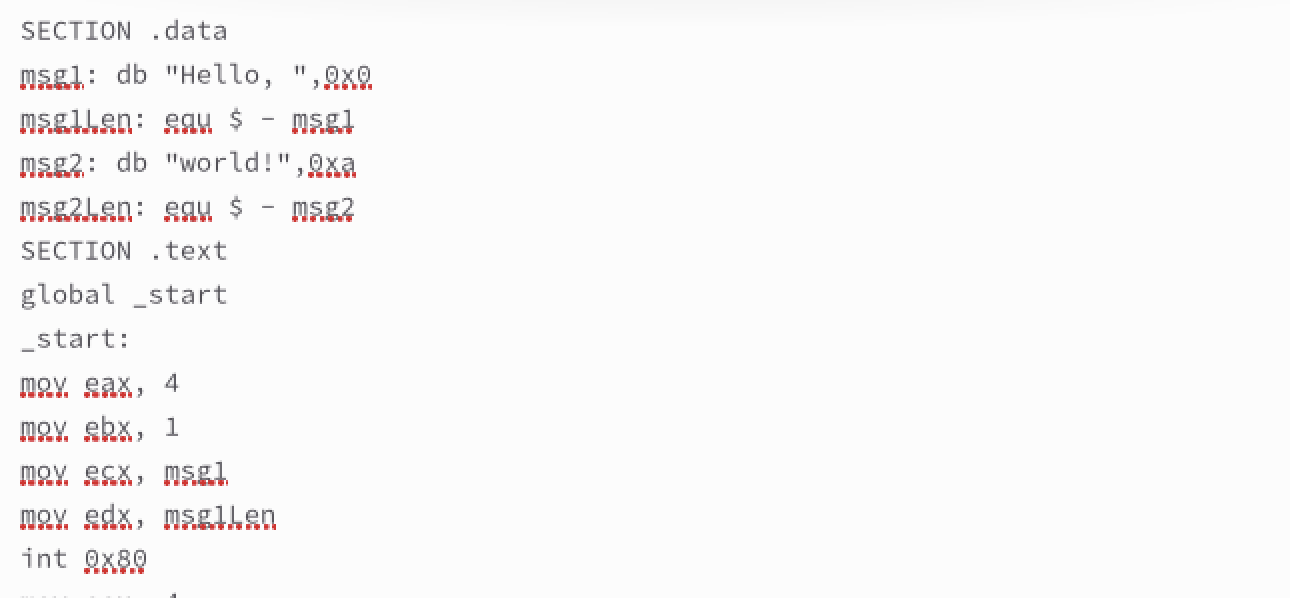
Все работает верно.

Теперь создадим второй файл.(рис.7)

Создание второго файла

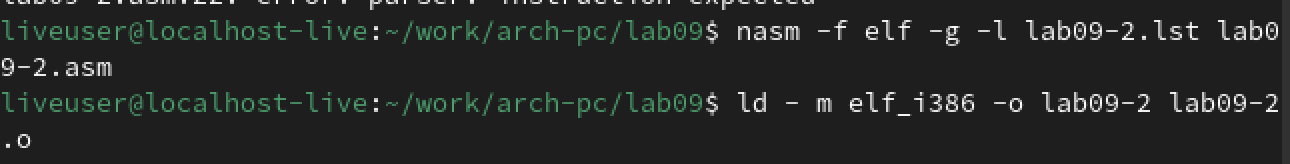
Создание второго файла

Вставим туда код из листинга.(рис.8)



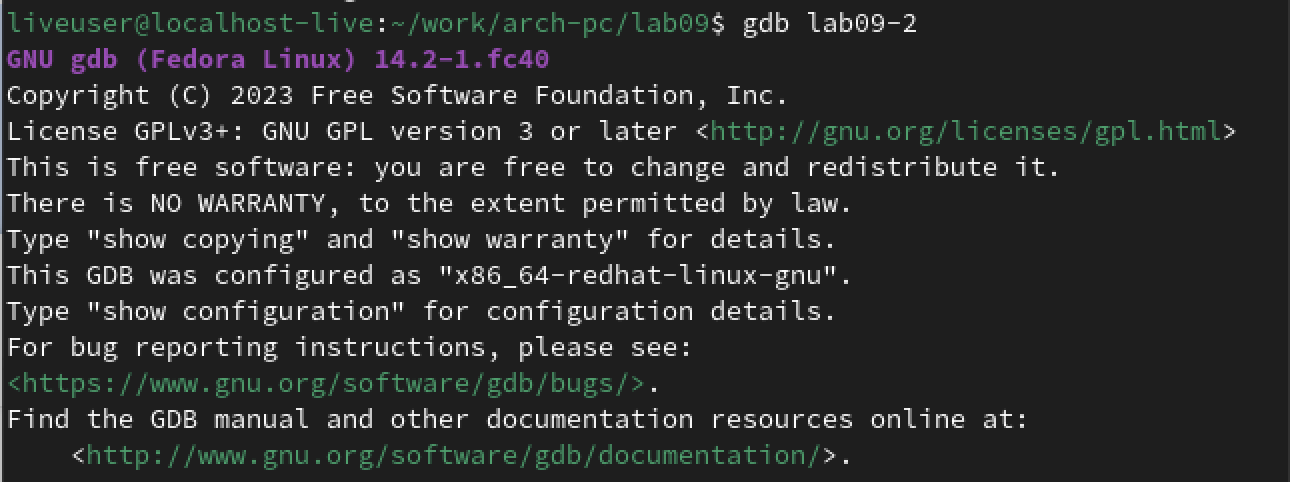
Изменение файла lab9-2.asm

Запустим файл.(рис.9)



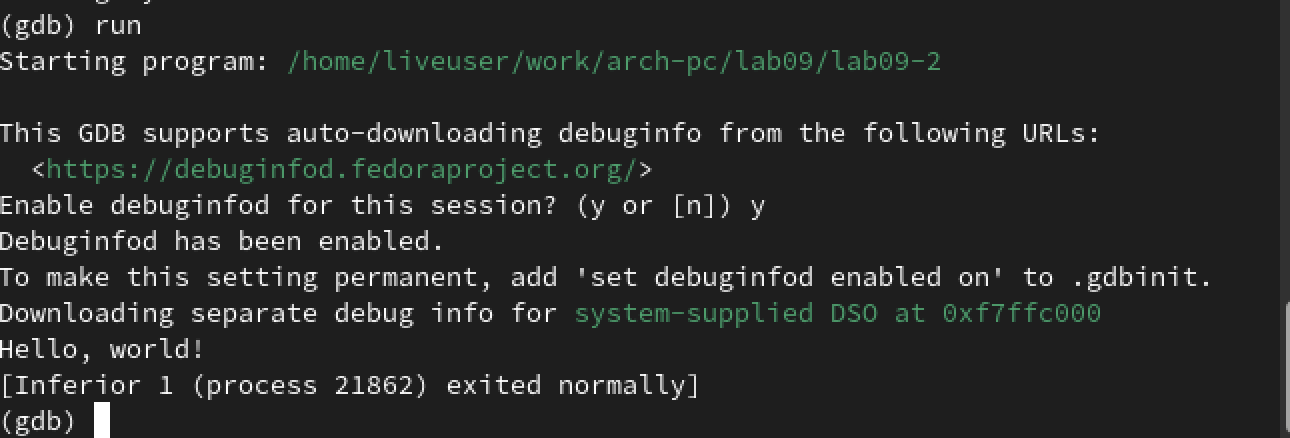
Запуск файла lab9-2.asm

Теперь загрузим программу в gdb.(рис.10)



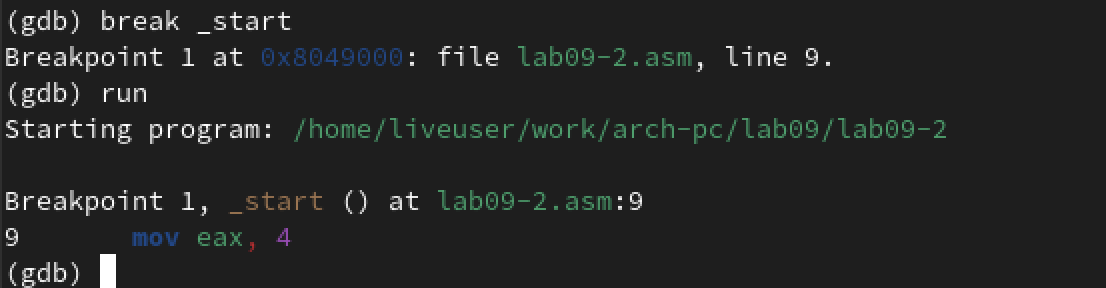
Загрузка программы в gdb

Запустим ее в отладчике.(рис.11)



Запуск программы в отладчике

После запуска создадим брейкпоинт на метке \_start с помощью команды break.(рис.12)



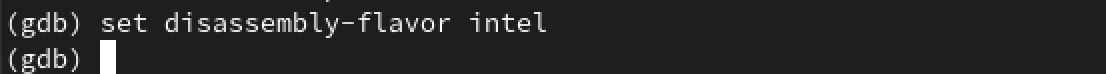
Создание брейкпоинта

С помощью команды disassemble дизассемблируем метку.(рис.13)



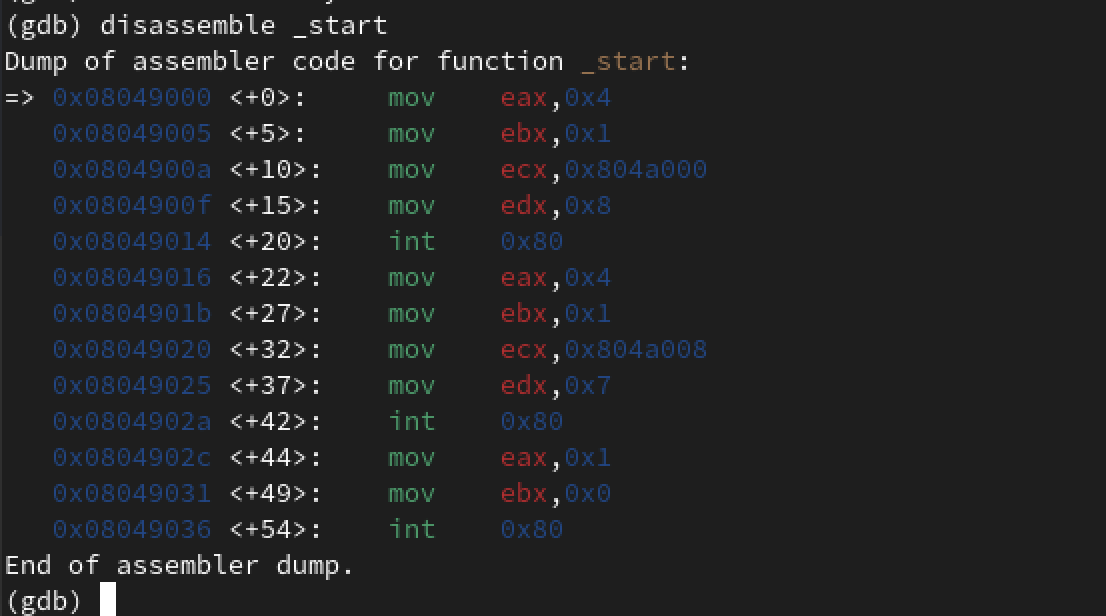
Дизассемблирование

При помощи следующей команды переключаем синтаксис вывода на intel(рис.14)



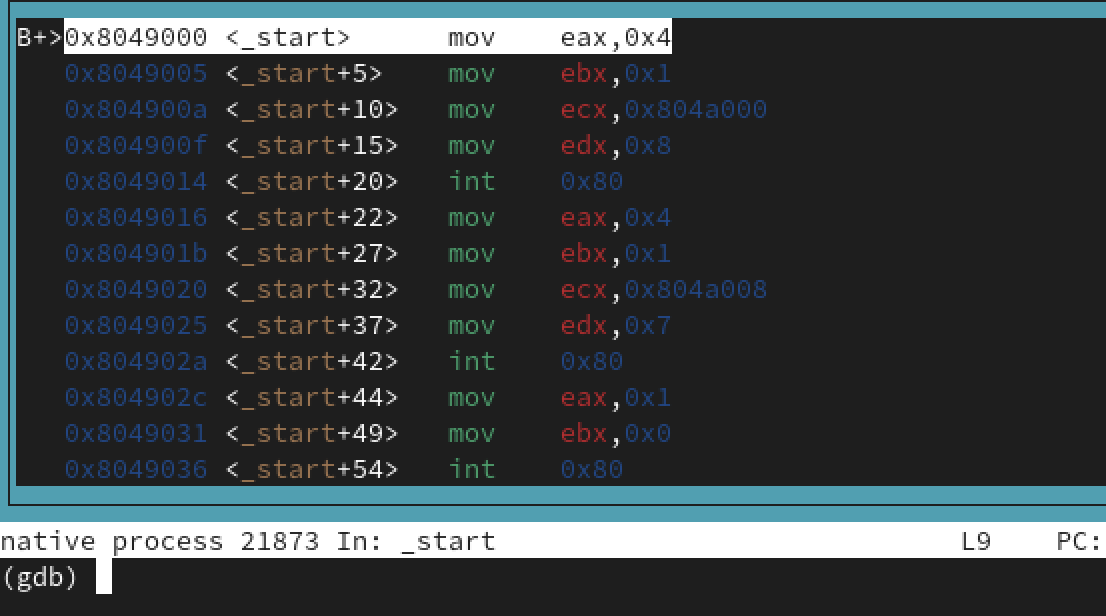
Переключение синтаксиса

Дизассемблиурем программу еще раз(рис.15)



Повторное дизассемблирование программы

Теперь включаем графическое отображение кода.(рис.16)



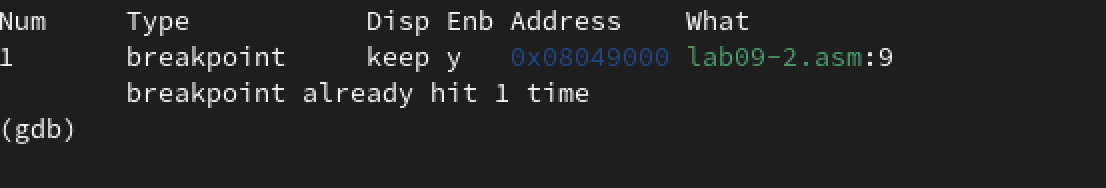
Графическое отображение кода

Включаем графическое отображение значений регистров.(рис.17)



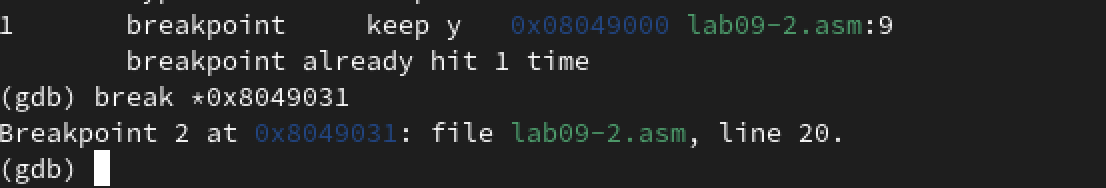
Графическое отображение регистров

Выведем отображение информации о имеющихся брейкпоинтах.(рис.18)



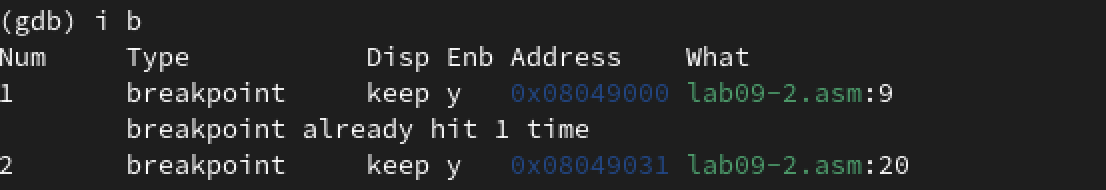
Отображение информации о брейкпоинтах

А теперь создадим брейкпоинт самостоятельно.(рис.19)



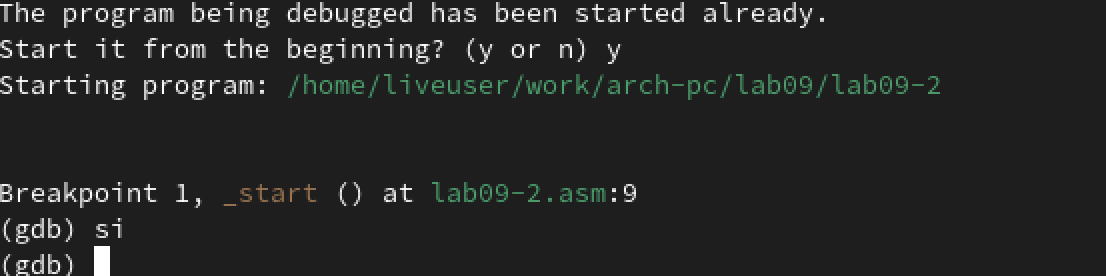
Создание брейкпоинта по адресу

Выведем информацию о брейкпоинтах еще раз.(рис.20)

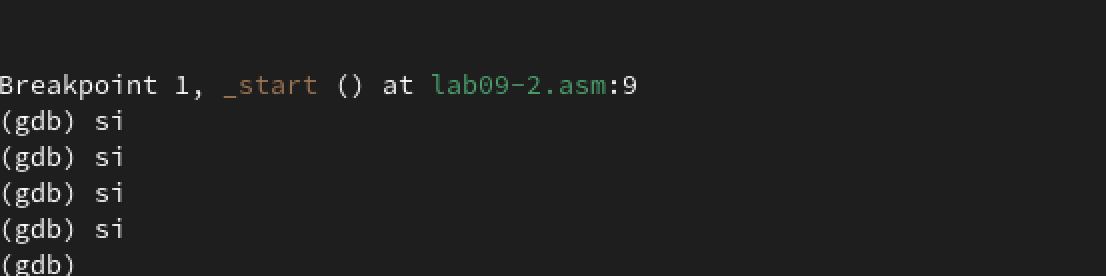


Повторных вывод информации о брейкпоинтах

Воспользуемся командой si для построчного выполнения кода. Задействуем ее 5 раз.(рис.21-22)

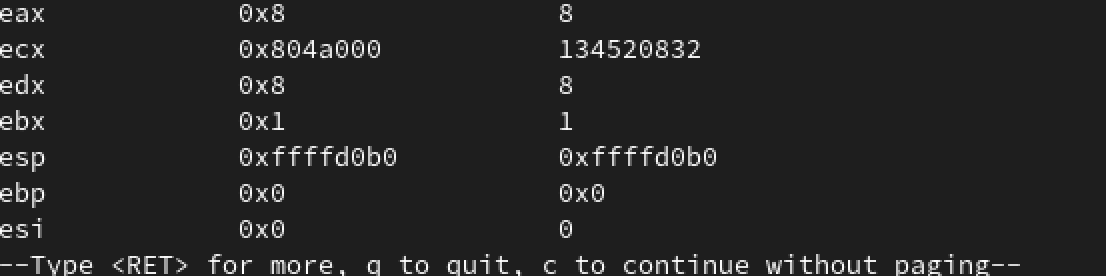


Построчное выполнение кода



Построчное выполнение кода

Как видим, поменялись значения регистров eax, ecx, edx и ebx. Теперь выведем информацию о значениях регистров(рис.23)

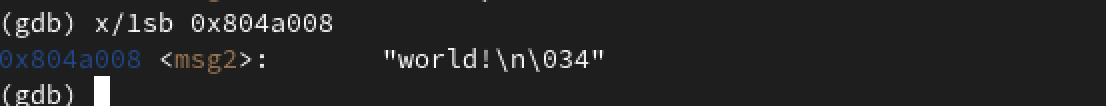


Изменение значений регистров

Теперь выведем значение переменной по имени и по адресу.(рис.24-25)

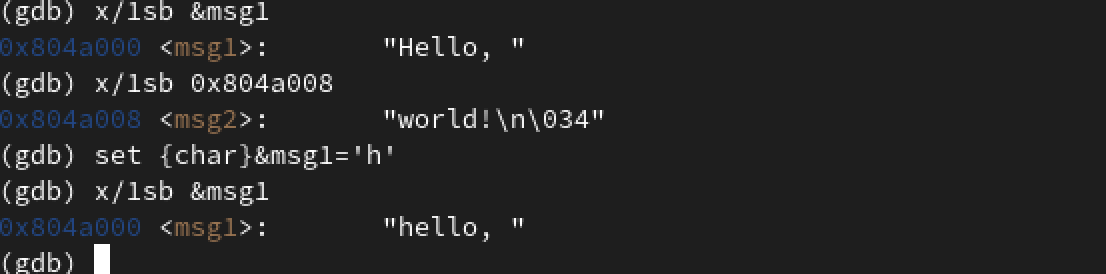


Вывод значения переменной по имени



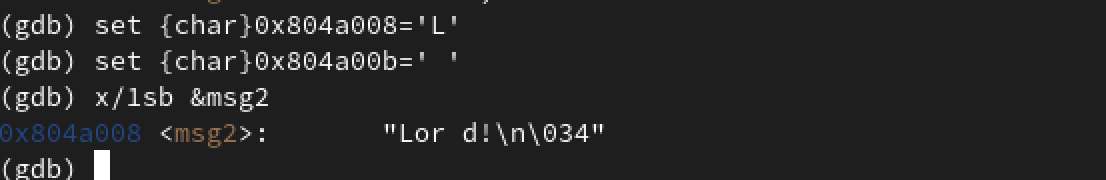
Вывод значения переменной по адресу

После этого изменим первый символ переменной.(рис.26)



Изменение первого символа

А затем изменим несколько символов переменной, обращаясь по адресу.(рис.27)



Изменение нескольких символов

Теперь предстоит вывести значение регистра в изначальном, двоичном и шестнадцатиричном виде(рис.28)



Вывод значения регистра в разном виде

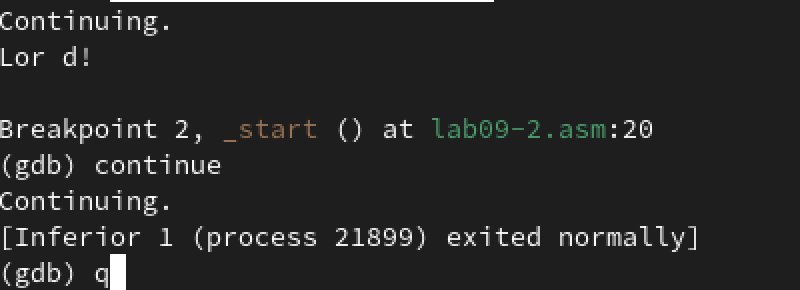
Изменим значение регистра.(рис.29)



Изменение значения регистра

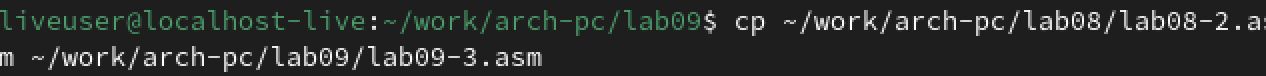
Нетрудно заметить, что в регистр записались разные значения, так как в одном случае мы записываем туда число - в другом строку.

Завершаем работу программы с помощью continue и выходим из отладчика.(рис.30)



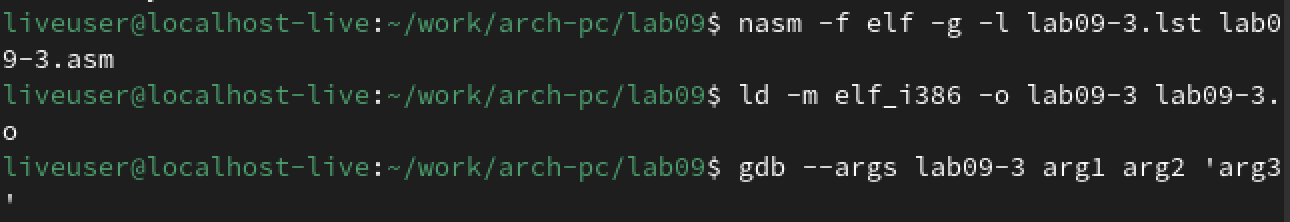
Завершение работы

Теперь скопируем файл из предыдущей лабораторной работы.(рис.21)



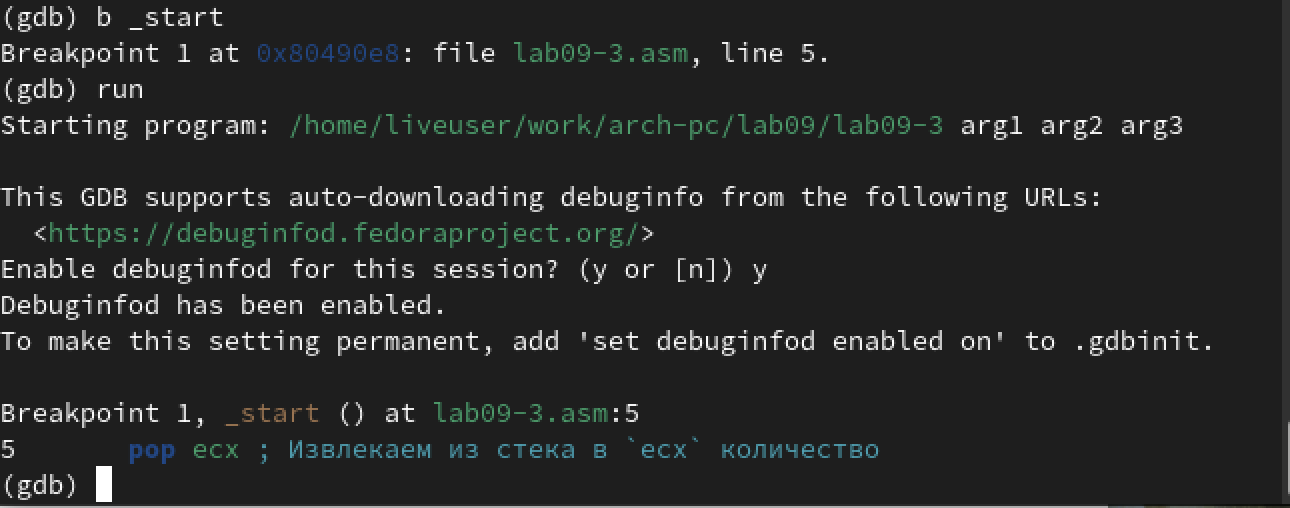
Копирование предыдущей лабораторной работы

Мы так же соберем его и загрузим в gdb.(рис.32)



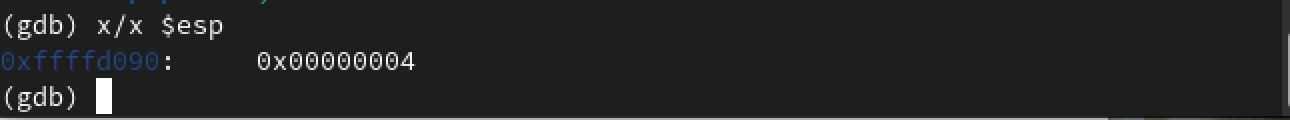
Загрузка файла предыдущей работы в gdb

Теперь создаем брейкпоинт и запускаем программу.(рис.33)



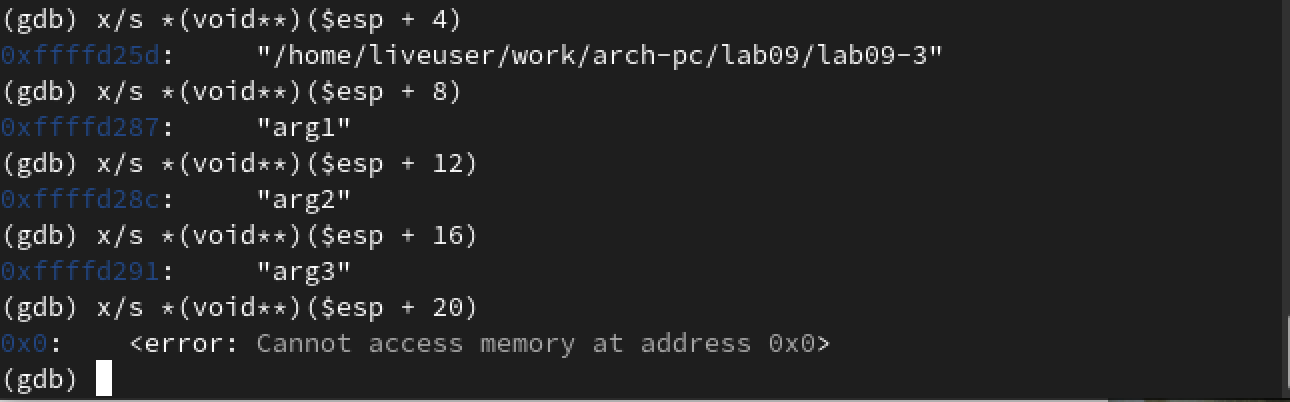
Запуск программы

Затем выведем значение регистра esp, где хранятся данные о стеке.(рис.34)



Вывод значения регистра

Теперь нужно вывести значение всех элементов стека.(рис.35)

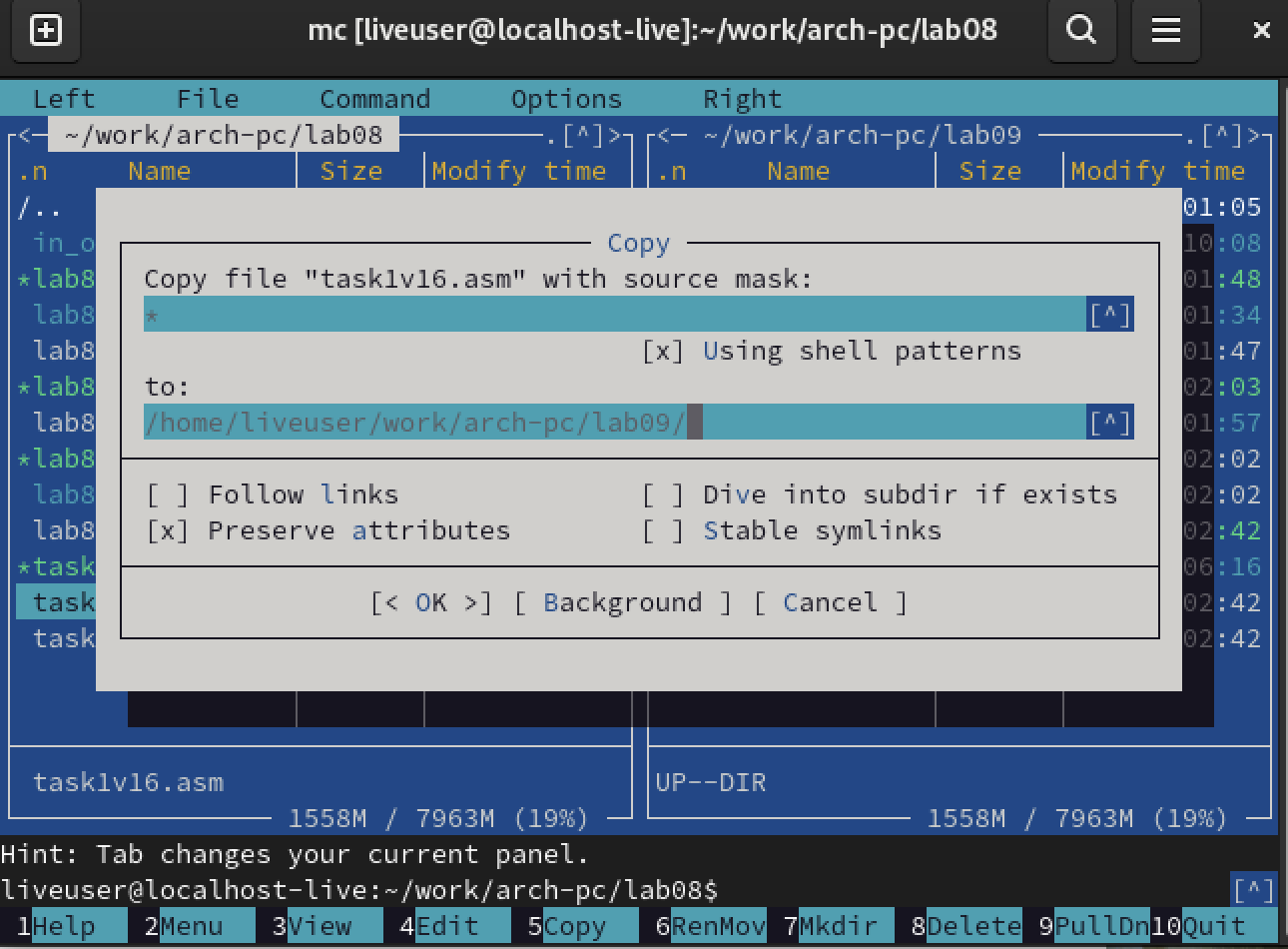


Вывод значений всех элементов стека

Можно заметить, что для вывода каждого элемента нужно менять значения адреса с шагом на 4. Это связано с тем, что под каждый элемент выделяется 4 байта.

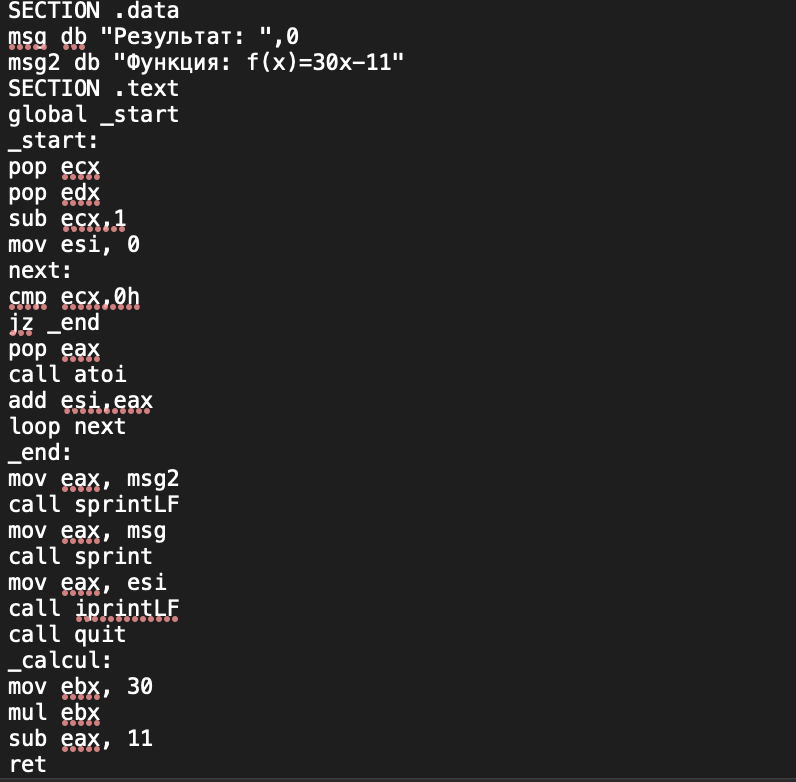
#Выполнение самостоятельной работы

Первым делом копируем файл задания из прошлой лабораторной работы.(рис.36)



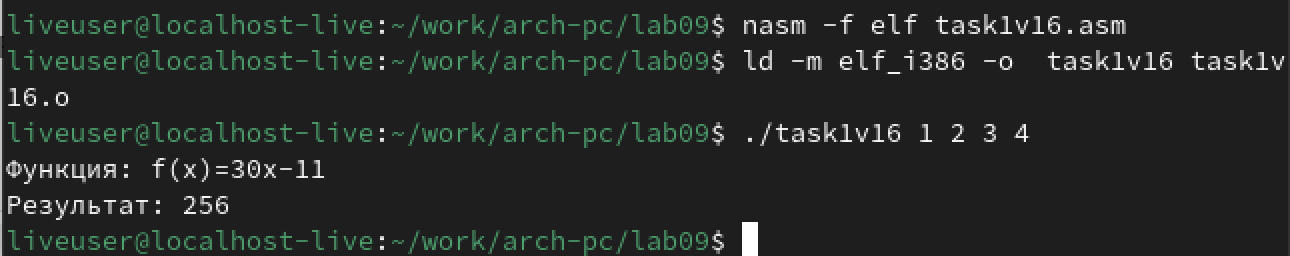
Копируем файл из предыдущей работы

Его код выглядит так.(рис.37)



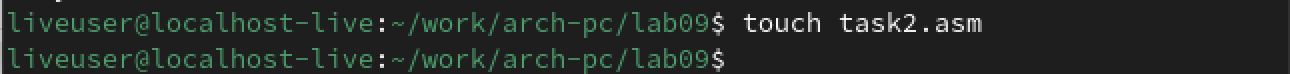
Код в файле

Теперь соберем код и проверим.(рис.38)



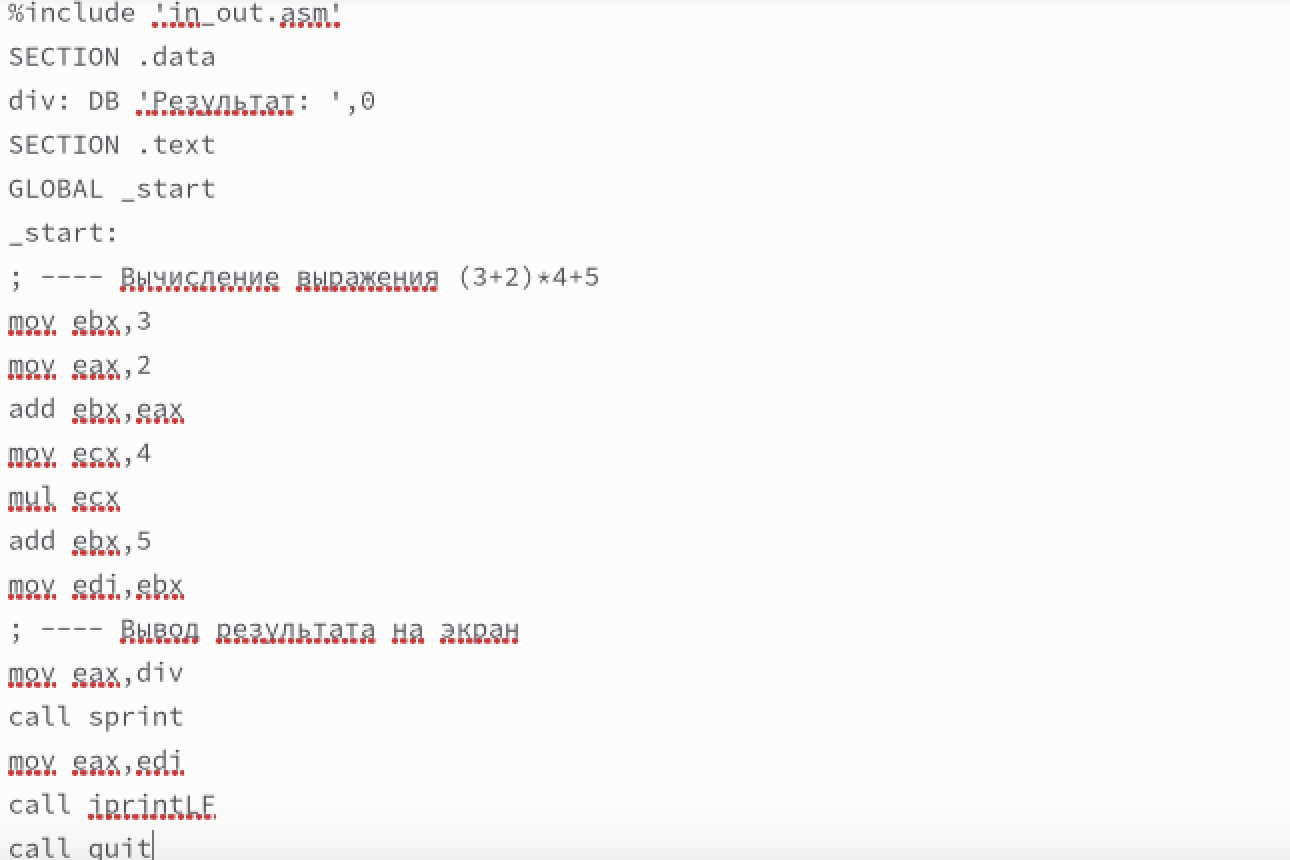
Запуск кода

Создадим второй файл.(рис.39)



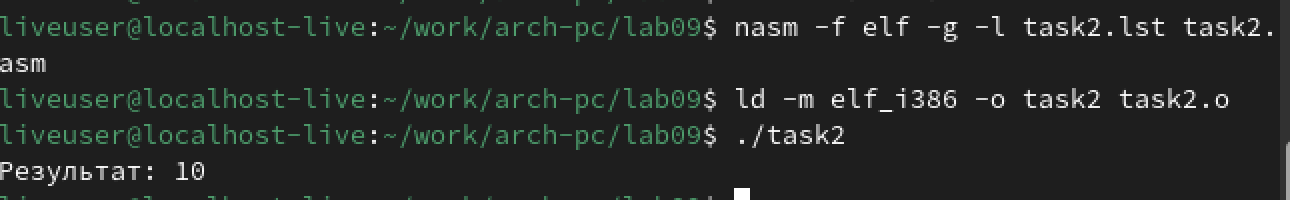
Второй файл

Вставляем в него код из листинга.(рис.40)



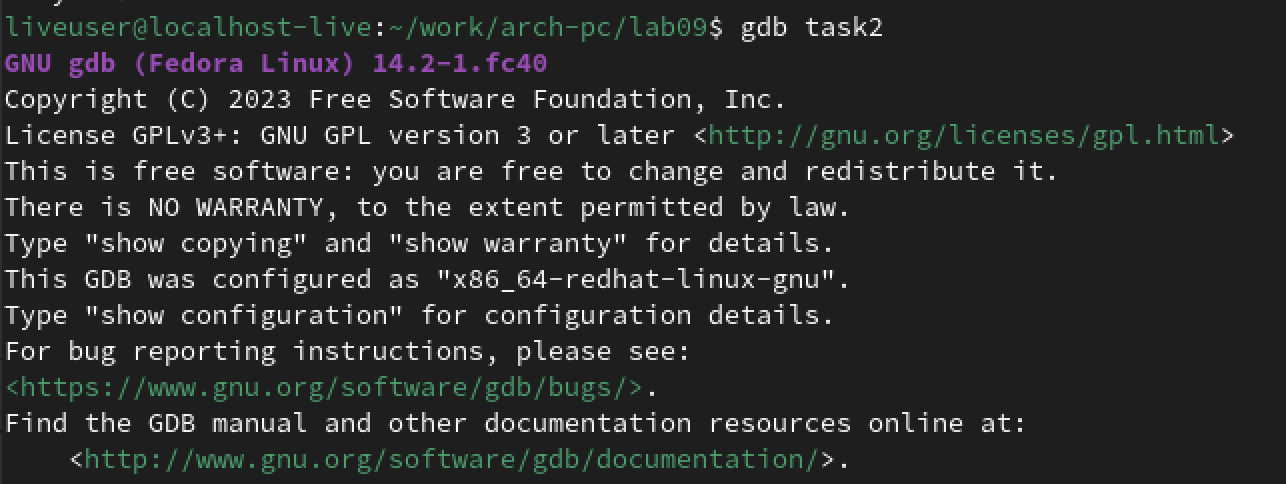
Вставляем код из листинга

Запускаем файл.(рис.41)



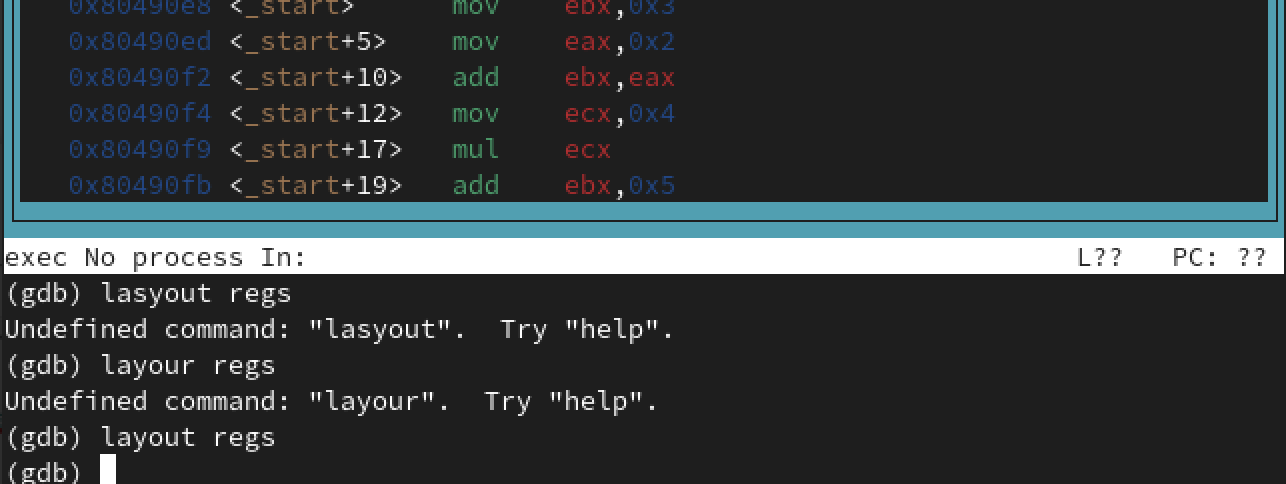
Запуск второго файла

Вставим наш файл в gdb(рис.42)



Вставляем файл в gdb

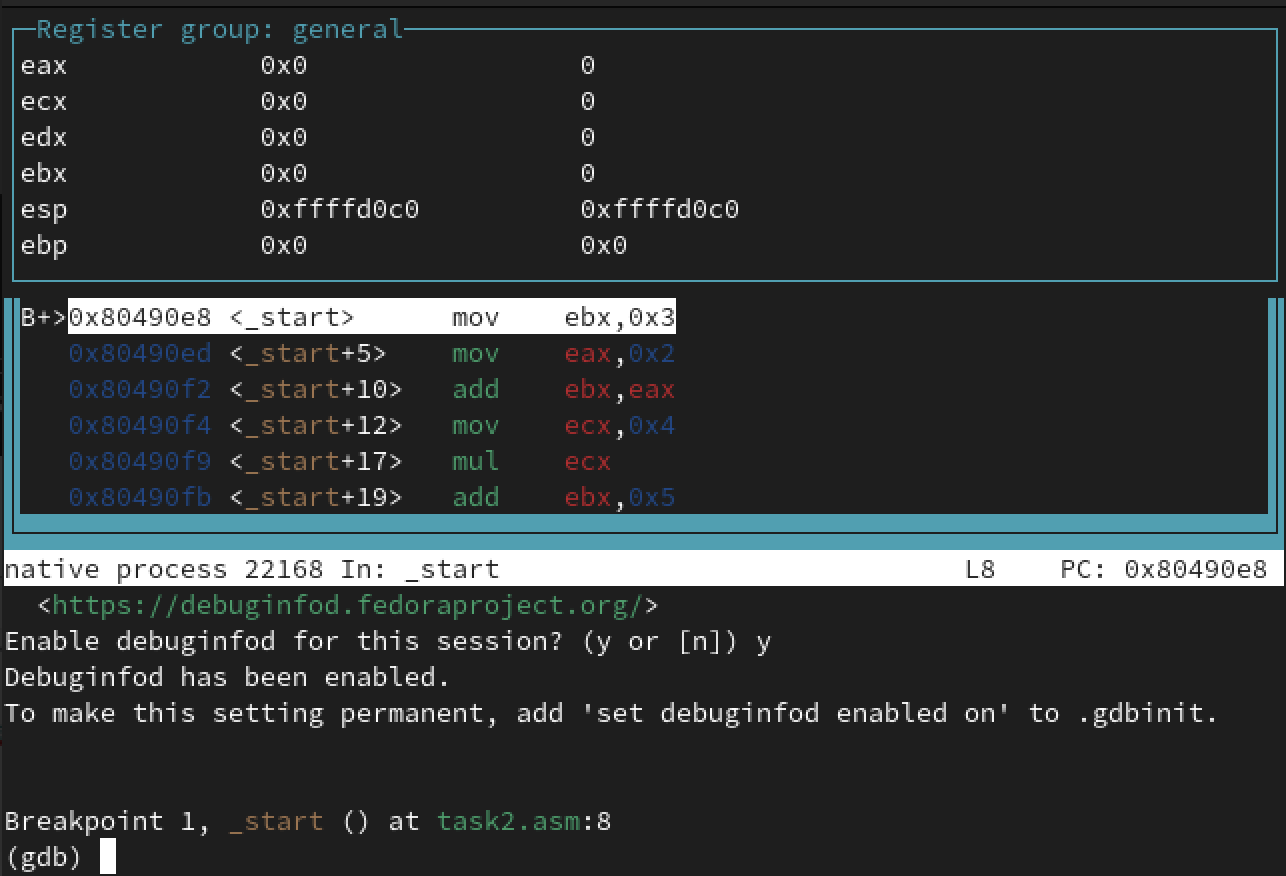
Переключаем на синтаксис и включаем графическое отображение.(рис.43)



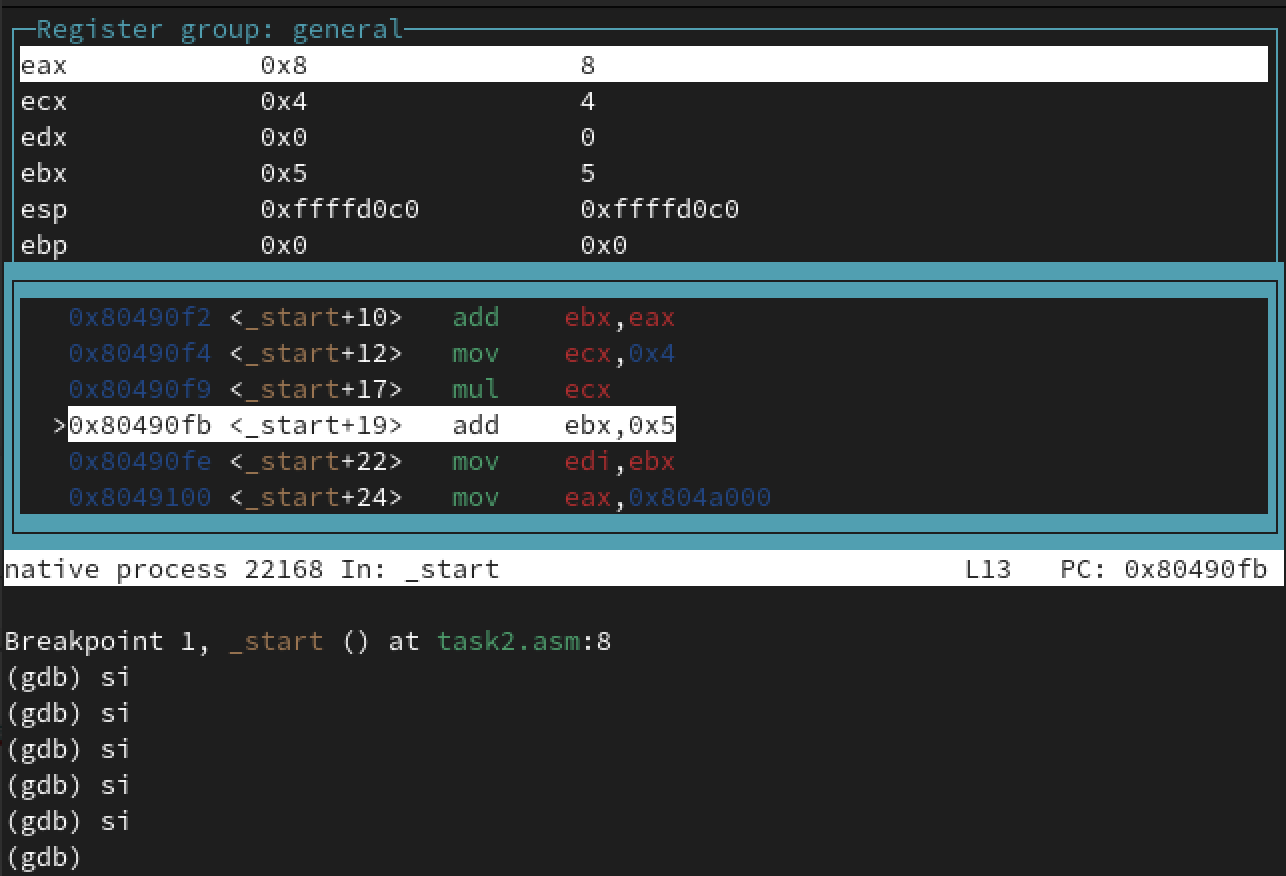
Переключение синтаксиса и включение графического отображения

Затем устанавливаем брейкпоинт(рис.44) 

Начинаем выполнение кода.(рис.45-46)



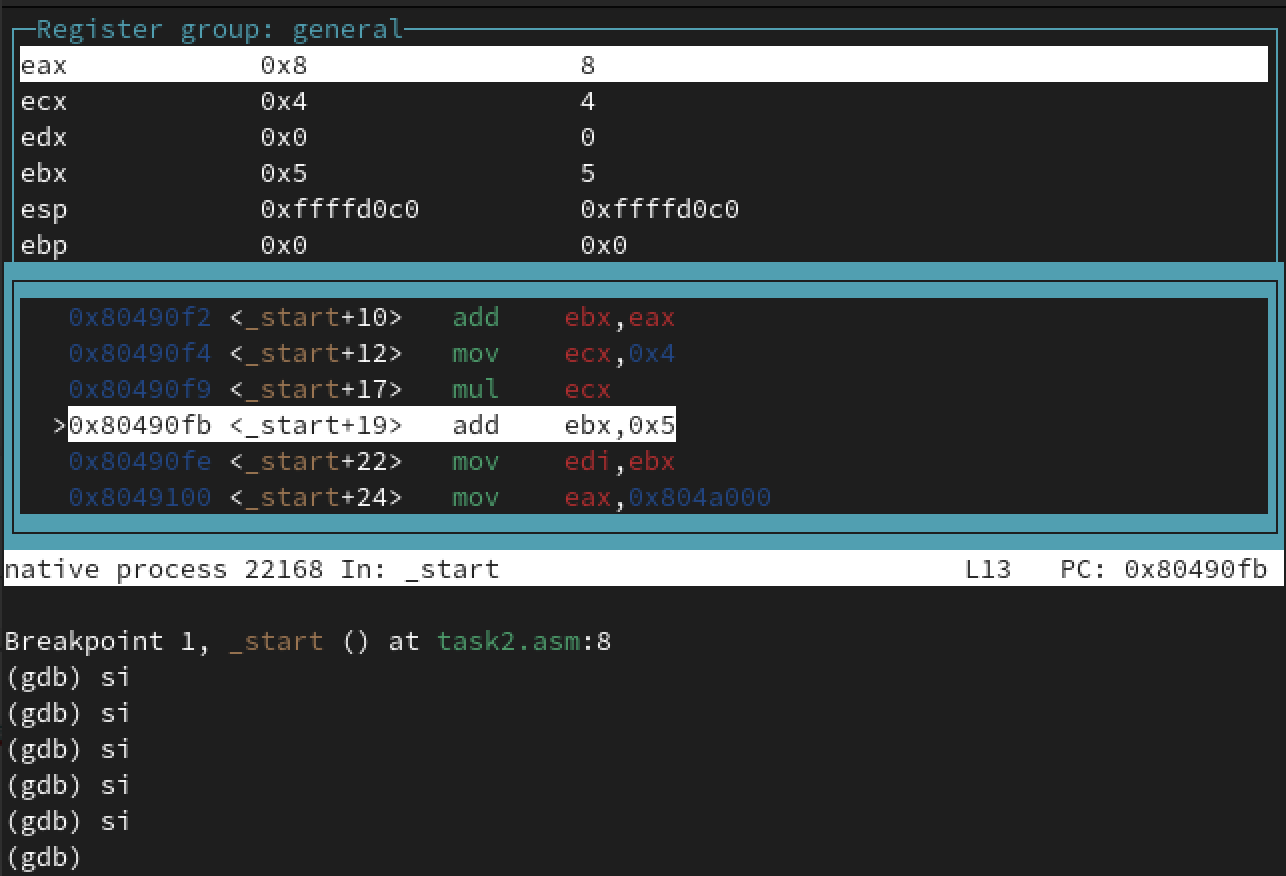
Выполнение кода



Выполнение кода

Нетрудно заметить,что мы должны были умножить значение регистра ebx, но сделали это с eax.

Отредактируем код и запустим снова.(рис.47)



Повторный запуск

Все вывелось верно. Работа сделана правильно.

# 3 Выводы

В результате выполнения данной лабораторной работы мы научились использовать подпрограммы, узнали как пользоваться отладчиком и изучили его функции.