Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Павличенко Родион Андреевич

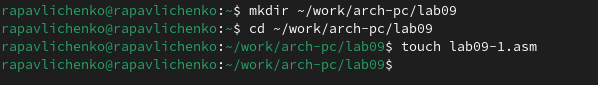
Содержание

# 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

Для начала выполнения работы необходимо создать рабочую папку и файл lab9-1.asm :



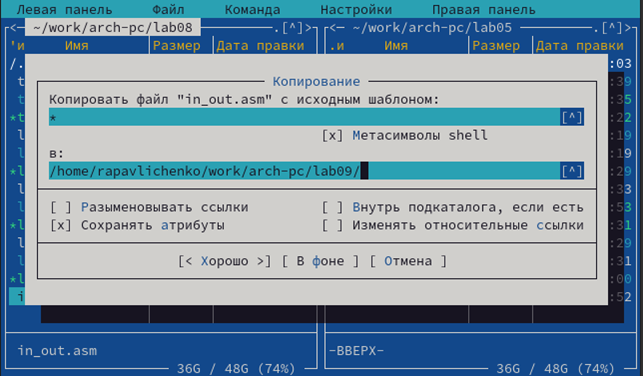
Создание рабочей директории и файла lab9-1.asm

Далее, запустим Midnight commander :

Запуск Midnight commander

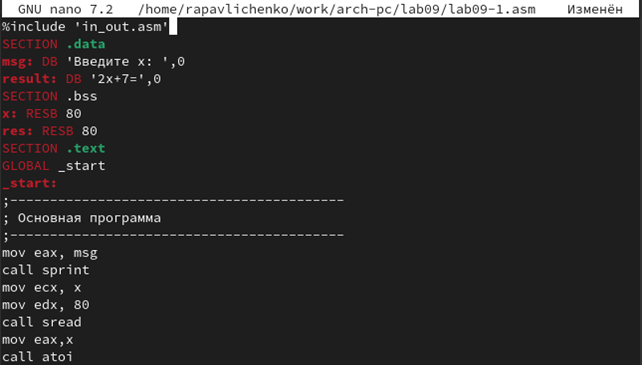
Запуск Midnight commander

Скопируем файл in\_out.asm из директории прошлой работы :



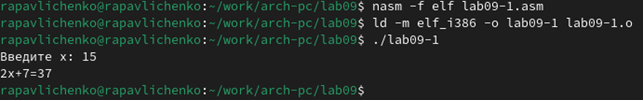
Копирование файла in\_out.asm в рабочую директорию

Вставим в файл lab9-1.asm код из листинга 9.1 :



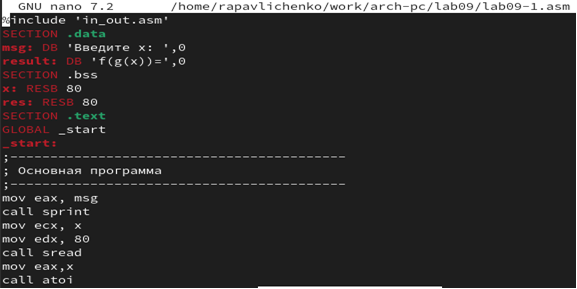
Вставка кода из файла листинга 9.1

Соберём программу и посмотрим на вывод :



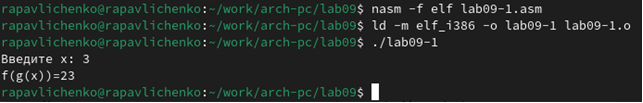
Сборка программы из файла lab9-1.asm и её запуск

Теперь изменим файл так, чтобы внутри подпрограммы была ещё одна подпрограмма, вычисляющая значение g(x) и чтобы она передавала значение в первую подпрограмму, которая бы уже вычислила значение f(g(x)) :



Изменение файла lab9-1.asm

Соберём программу и проверим её работу :



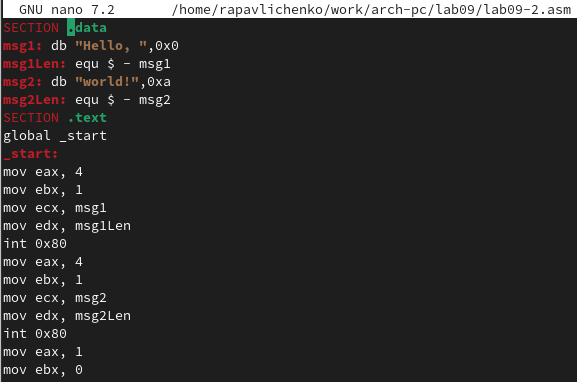
Повторная сборка программы из файла lab9-1.asm и её запуск

Создадим новый файл lab9-2.asm:

Создание второго файла: lab9-2.asm

Создание второго файла: lab9-2.asm

Вставим в него код из листинга 9.2 :



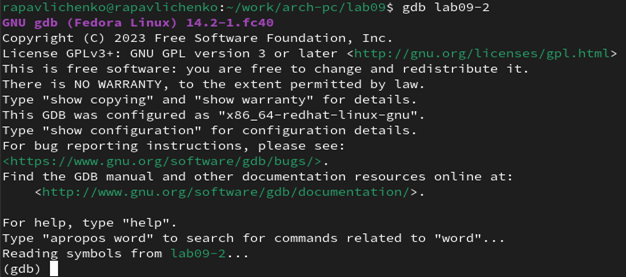
Запись кода из листинга 9.2 в файл lab9-2.asm

Соберём программу следующим образом (с использованием аргумента -g) :

Сборка программы из файла lab9-2.asm

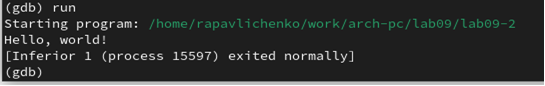
Сборка программы из файла lab9-2.asm

Теперь загрузим её в gdb :



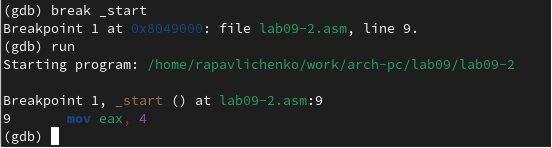
Загрузка программы lab9-2.asm в gdb

Запустим её в отладчике с помощью команды run :



Запуск программы в отладчике

Создадим брейкпоинт на метке \_start с помощью команды break :



Создание брейкпоинта

С помощью команды disassemble дизассемблируем её :



Дизассемблирование программы

Переключим синтаксис вывода на intel :

Переключение на синтаксис intel

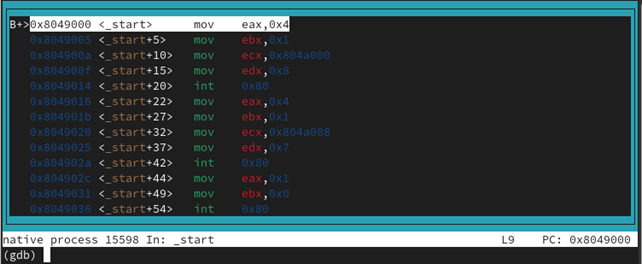
Переключение на синтаксис intel

Повторно дизассемблируем программу :



Повторное дизассемблирование

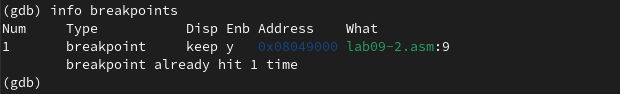
Включим графическое отображения кода :



Внешний вид интерфейса

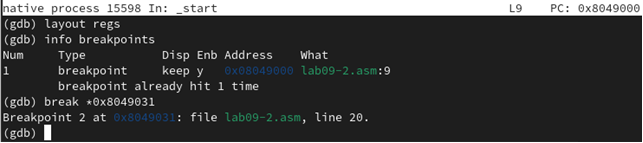
Теперь включеним графическое отображение значений регистров :

 Выведем инормацию о всех брейкпоинтах :



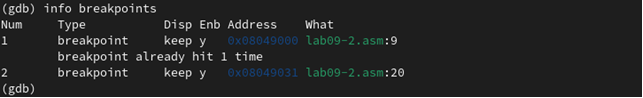
Вывод информации о брейкпоинтах

Попробуем теперь создать брейкпоинт по адресу:



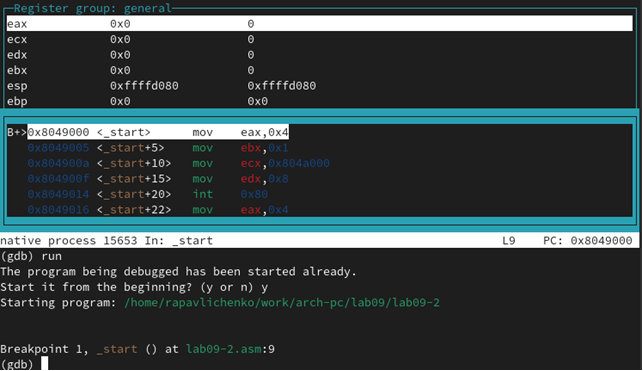
Создание брейкпоинта по адресу

Повторно выведем информацию о брейкпоинтах :

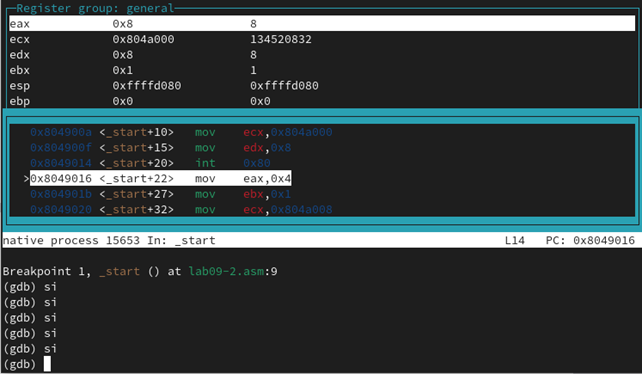


Повторный вывод информации о брейкпоинтах

Теперь 5 раз выполним команду si для построчного выполнения кода :



Выполнение следующей команды в коде программы



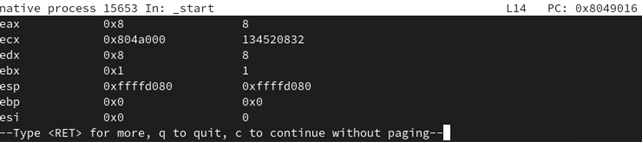
Выполнение следующей команды в коде программы (2)

Как видим, поменялись значения регистров eax, ecx, edx и ebx. Теперь выведем информацию о значениях регистров :

Вывод значений регистров

Вывод значений регистров

Вот, что нам выводится :



Значения регистров

Попробуем вывести значени переменной по имени :

Вывод значения переменной по имени

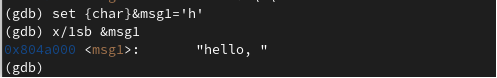
Вывод значения переменной по имени

Теперь попробуем вывести значени переменной по адресу :

Вывод значения переменной по адресу

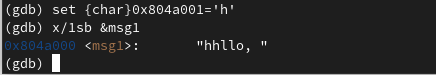
Вывод значения переменной по адресу

Теперь изменим первый символ переменной :



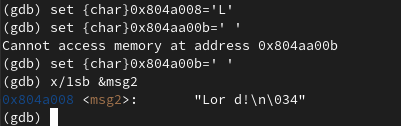
Изменение первого символа переменной по имени и вывод переменной

А теперь изменим второй символ переменной, уже обратясь по адресу :



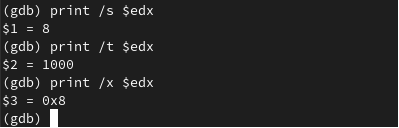
Изменение второго символа переменной по адресу и вывод переменной

Теперь изменим несколько символов второй переменной :



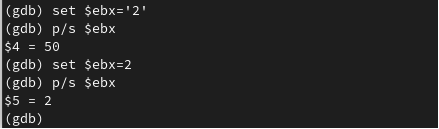
Исменение нескольких символов второй переменной по адресу и вывод переменной

Теперь попробуем вывести значение регистра в строковом, двоичном и шестнадцатиричном виде :



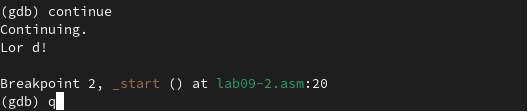
Вывод значения регистра в строковом, двоичном и шестнадцатиричном виде

Попробуем теперь изменить значение регистра :



Изменение значения регистра

Как видим, в регистр записались разные значения. Это связано с тем, что в одном случае мы записываем в него число, а в другом случае - строку. Завершим работу программы с помощью continue (чтобы продолжить выполнение) и выйтем из отладчика :



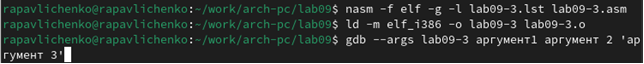
Завершение работы программы

Скопируем файл из прошлой работы :

Копирование файла из прошлой работы

Копирование файла из прошлой работы

Соберём его и вгрузим в gdb :



Сборка программы и вгрузка в gdb

Создадим брейкпоинт и запустим программу :



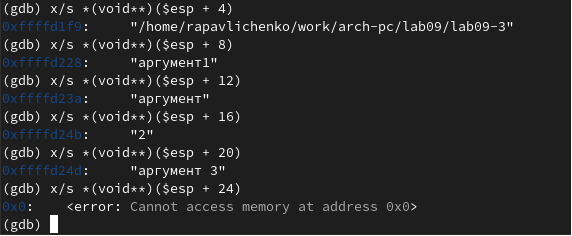
Создание брейкпоинта и запуск программы

Теперь выведем значение регистра esp, где хранятся данные о стеке :

Вывод значения регистра esp

Вывод значения регистра esp

Теперь выведем значение всех элементов стека :

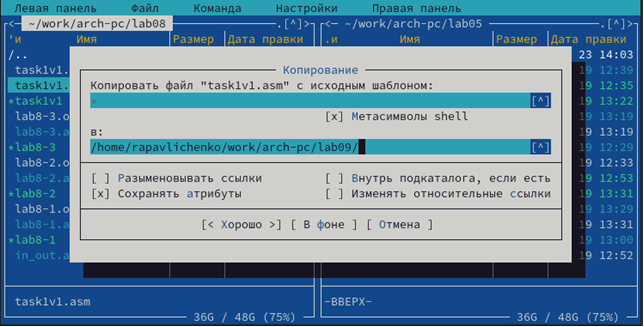


Вывод всех значений в стеке

Как видим, для вывода каждого элемента стека нам нужно менять значение адреса с шагом 4. Это связано с тем, что именно с шагом 4 располагаются данные в стеке, ведь под каждый элемент выделяется 4 байта

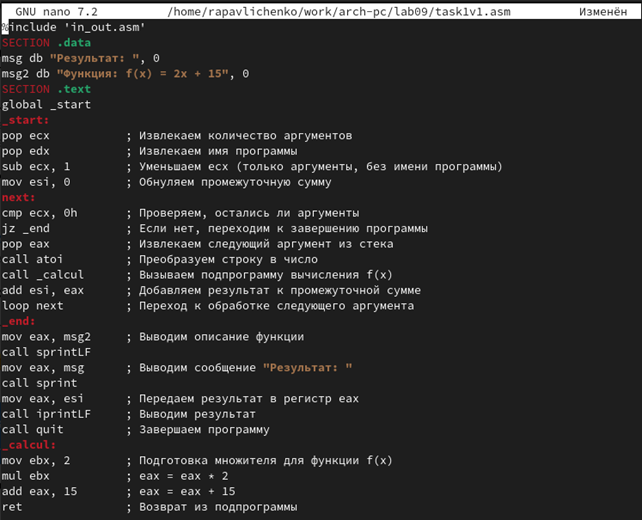
# 3 Выполнение задания для самостоятельной работы

Скопируем файл первого задания прошлой самостоятельной работы :



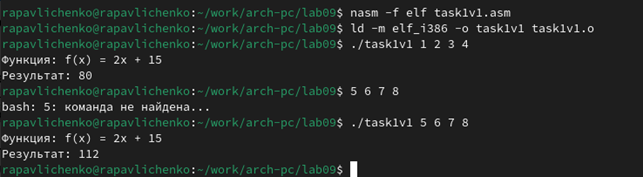
Копирование первого файла самостоятельной работы из прошлой работы

Нам нужно переписать его так, чтобы он использовал для авчисления выражения подпрограмму :



Редактирование кода

Соберём его и проверим корректность выполнения :



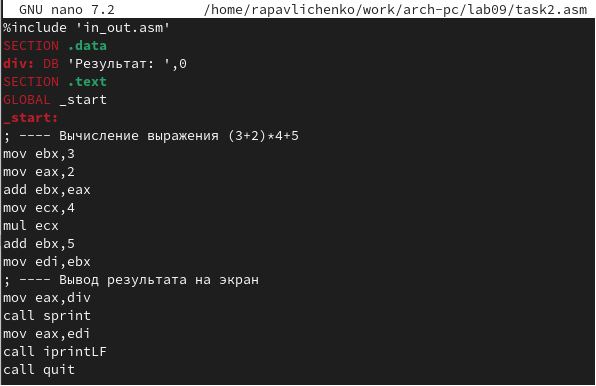
Сборка и проверка работы программы

Создадим файл второго задания самостоятельной работы :

Создание файла второго задания самостоятельной работы

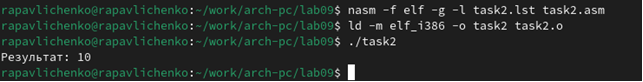
Создание файла второго задания самостоятельной работы

Вставим в него код из листинга 9.3 :

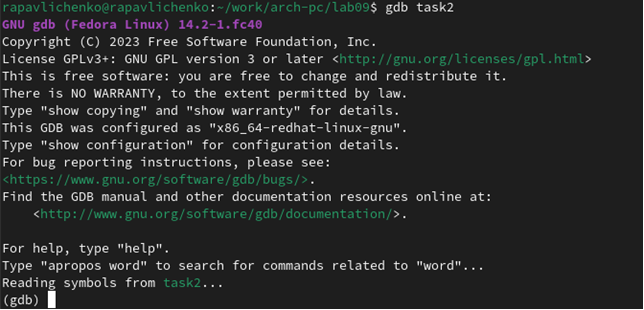


Вставка кода из листинга 9.3

Соберём его и запустим:



Сборка программы

Как видим, код считает значение выражения неправильно. Загрузим его в gdb : 

Переключим его на синтаксис intel :

Переключение на синтаксис intel

Переключение на синтаксис intel

Включим графическое отображение кода :

Включение графического отображения кода и выполнения команд

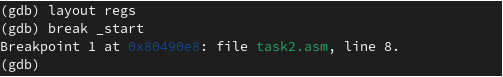
Включение графического отображения кода и выполнения команд

Включеним графическое отображение значений регистров :

Включение графического отображения значений регистров

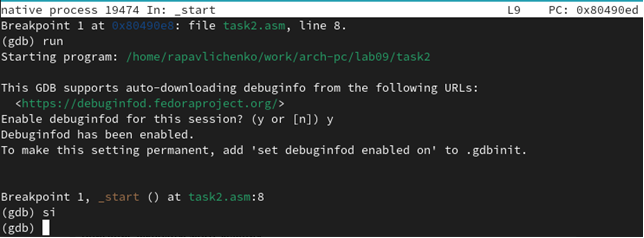
Включение графического отображения значений регистров

Установим брейкпоинт на \_start :

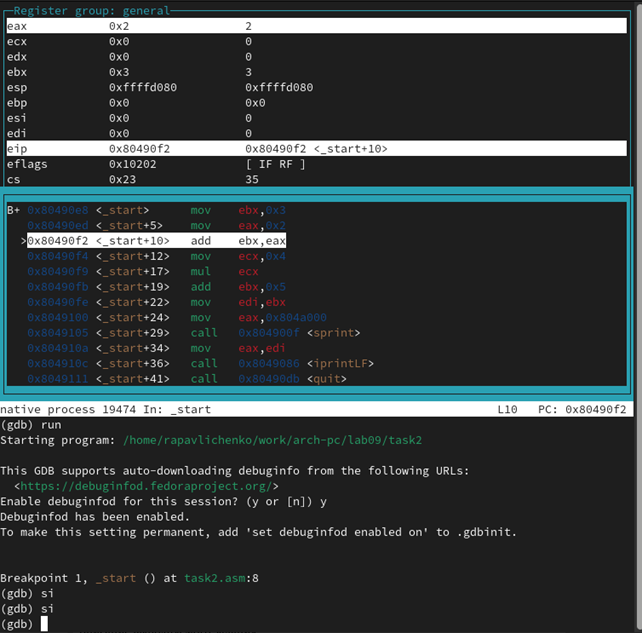


Установка брейкпоинта

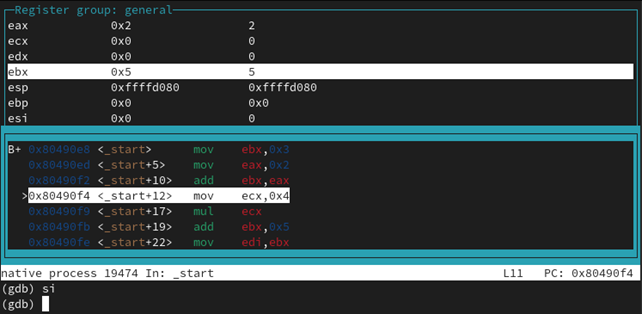
И начнём построчно выполнять код :



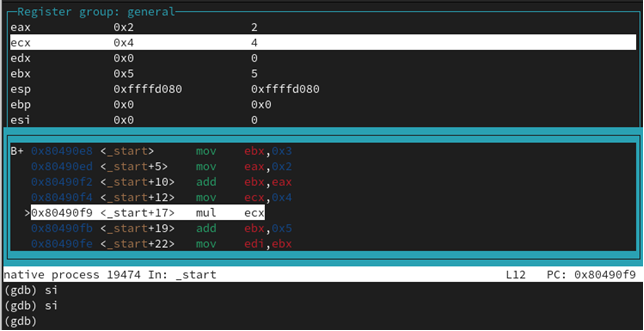
Значение всех регистров на 1 шаге



Значение всех регистров на 2 шаге



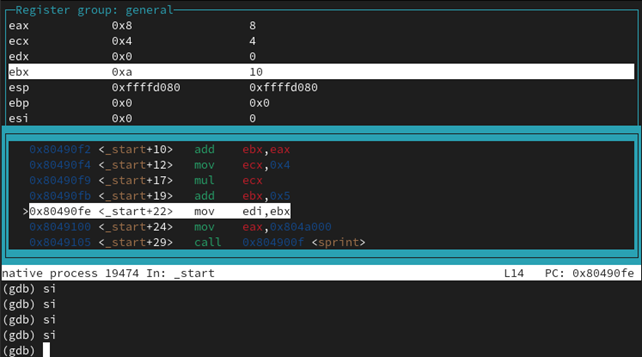
Значение всех регистров на 3 шаге



Значение всех регистров на 4 шаге

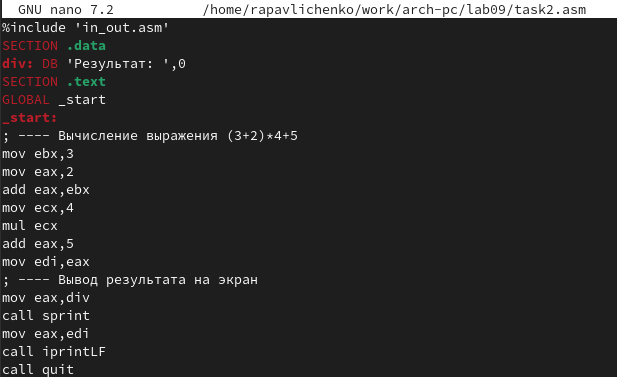


Значение всех регистров на 5 шаге



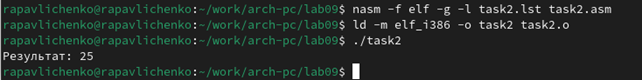
Значение всех регистров на 6 шаге

Как видим, мы должны были умножить значение регистра ebx, но умножили регистр eax. Нам необходимо все результаты хранить в регистре eax. Изменим код :



Редактирование кода

И проверим корректность его выполнения :



Сборка кода и проверка выполнения

Как видим, теперь код работает корректно

# 4 Выводы

В результате выполнения лабораторной работы были получены представления о работе подпрограмм, а также было реализовано несколько программ, использующих подпрограммы. Также, были получены навыки работы с базовым функионалом gdb, и с помощью gdb была отловлена ошибка в коде программы