Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Сырцева Анастасия Романовна

Содержание

# 1 Цель работы

Целью данной работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

* Ознакомление с подпрограммами
* Изучение отладки при помощи GDB

# 3 Выполнение лабораторной работы

Создаю папку для данной лабораторной работы. Перехожу в неё и создаю lab9-1.asm(рис. 1).

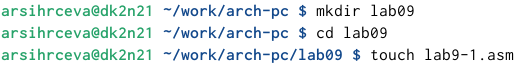


Рис. 1: Создание рабочего каталога и файла

Внимательно изучаю листинг программы с использованием вызова подпрограммы. Для корректной работы необходим внешний файл in\_out.asm. Копирую его в рабочую папку(рис. 2).

Рис. 2: Копирование внешнего файла

Рис. 2: Копирование внешнего файла

Ввожу в файл lab9-1.asm текст данной программы(рис. 3).

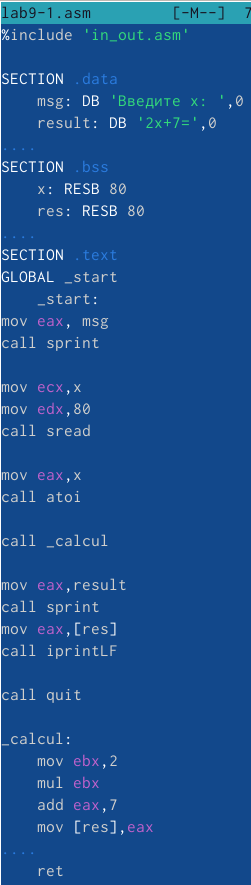


Рис. 3: Программа с использованием вызова подпрограммы

Создаю и запускаю исполняемый файл. Проверяю работу программы(рис. 4).

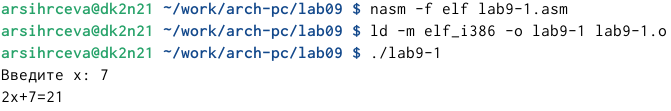


Рис. 4: Результат работы программы

Изменяю программу так, чтобы исходная функция была зависима от другой функции. Для этого добавляю ещё одну подпрограмму в исходную подпрограмму(рис. 5).

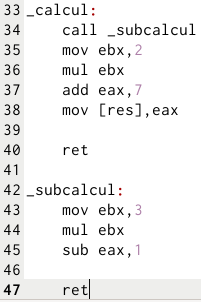


Рис. 5: Изменённая часть текста программы

Создаю и запускаю исполняемый файл изменённой программы(рис. 6).

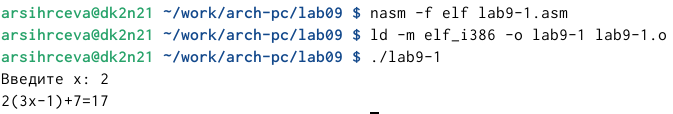


Рис. 6: Запуск и результат изменённой программы

Создаю новый файл lab9-2.asm( (рис. 7).

Рис. 7: Создание нового рабочего файла

Рис. 7: Создание нового рабочего файла

Ввожу в файл текст программы вывода сообщения Hello world!(рис. 8).

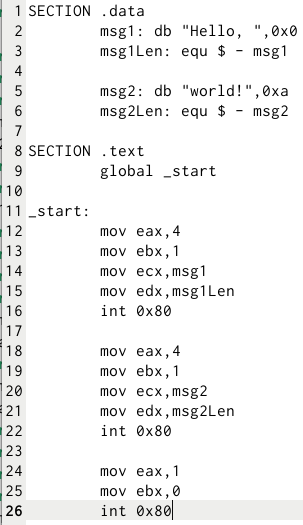


Рис. 8: Текст программы вывода сообщения

Создаю исполняемый файл для работы с GDB(рис. 9).

Рис. 9: Создание исполняемого файла

Рис. 9: Создание исполняемого файла

Загружаю исполняемый файл в отладчик GDB(рис. 10).

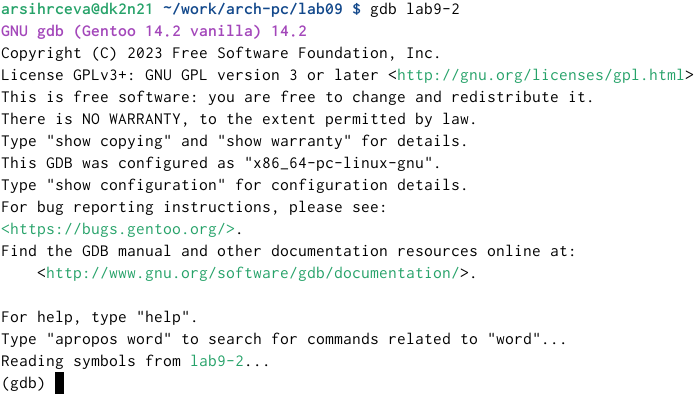


Рис. 10: Загрузка файла в отладчик

Проверяю работу программы, запустив её в оболочке GDB с помощью команды r(рис. 11).



Рис. 11: Запуск программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение программы. Запускаю её(рис. 12).

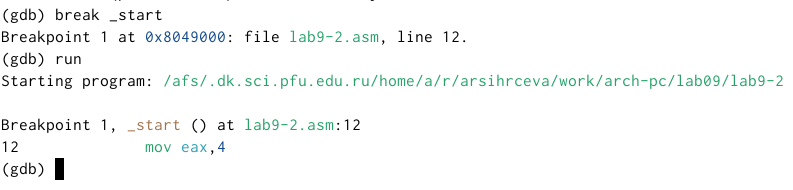


Рис. 12: Установка брейкпоинта и запуск программы

С помощью команды disassemble \_start изучаю дисассимилированный код программы(рис. 13).



Рис. 13: Просмотр дисассимилированного кода

Переключаюсь на отображение команд с intel’овским синтаксисом(рис. 14).

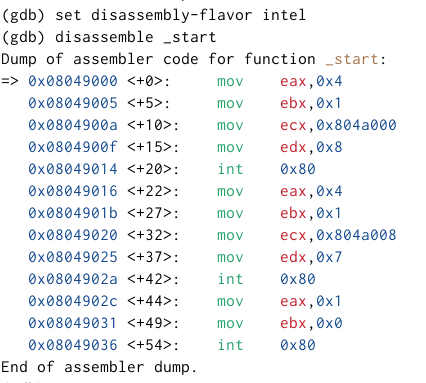


Рис. 14: Отображение команд с intel’овским синтаксисом

В дисассимилированном отображении используютя символы % и $ в командах, intel’овское же отображении они отсутствуют. Для удобства включаю режим псевдографики для более удобного анализа программы(рис. 15).

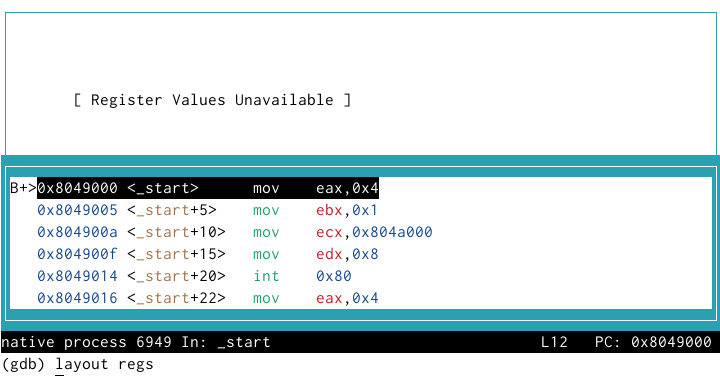


Рис. 15: Включение режима псевдографики

Проверяю наличие метки \_start(рис. 16).

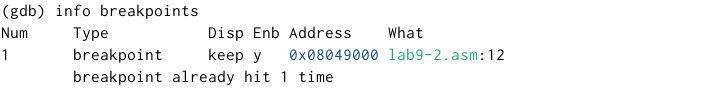


Рис. 16: Информация об установленных метках

Вручную ищу последнюю команду кода и смотрю её адрес(рис. 17).

Рис. 17: Адрес кода mov ebx,0x0

Рис. 17: Адрес кода mov ebx,0x0

Устанавливаю очку останова данной строчке(рис. 18).навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

Рис. 18: Точка останова

Рис. 18: Точка останова

Ещё раз просматриваю информацию о брейкпоинтах(рис. 19).

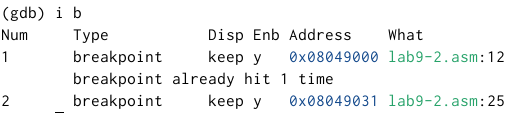


Рис. 19: Брейкпоинты

Выполняю команду stepi, изучаю значение регистров(рис. 20).

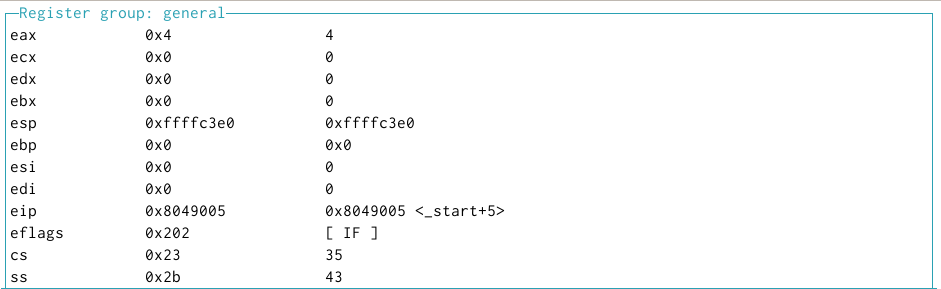


Рис. 20: Значение регистров в начальный момент

Пять раз выполняю инструкции с помощью данной команды(рис. 21).

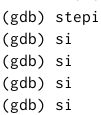


Рис. 21: Выполнение инструкций

Изучаю изменения значений регистров. Они поменялись у регистров eax, ecx, edx, ebx, eip(рис. 22).

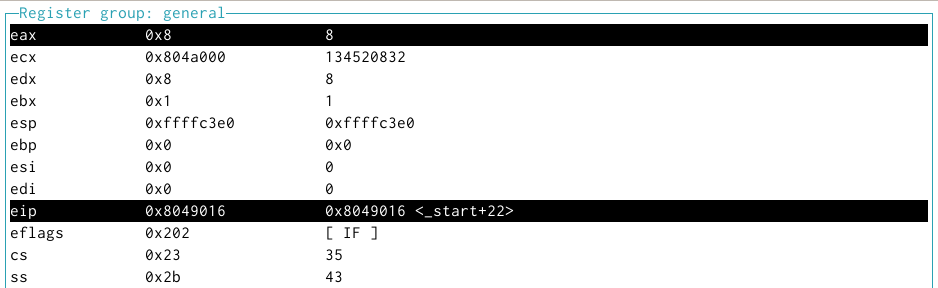


Рис. 22: Изменившиеся значения регистров

Просматриваю значение переменной msg1 по имени(рис. 23).

Рис. 23: Вывод значения msg1

Рис. 23: Вывод значения msg1

Вручную ищу адрес msg2(рис. 24).

Рис. 24: Адрес msg2

Рис. 24: Адрес msg2

Просматриваю значение переменной msg2 по адрессу(рис. 25).

Рис. 25: Значение msg2

Рис. 25: Значение msg2

Изменяю первый символ переменной msg1. Проверяю(рис. 26).

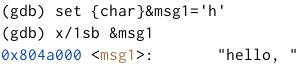


Рис. 26: Изменение значения msg1

Также изменяю первый символ переменной msg2. Проверяю(рис. 27).

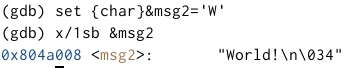


Рис. 27: Изменение значения msg2

Вывожу значение регистра edx в шестнадцатиричном формате(рис. 28), в двоичном формате (рис. 29), в символьном виде(рис. 30).

Рис. 28: Значение регистра edx в шестнадцатеричном формате

Рис. 28: Значение регистра edx в шестнадцатеричном формате

Рис. 29: Значение регистра edx в двоичном формате

Рис. 29: Значение регистра edx в двоичном формате

Рис. 30: Значение регистра edx в символьном виде

Рис. 30: Значение регистра edx в символьном виде

Изменяю значение регистра ebx с помощью команды set(рис. 31).

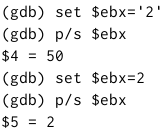


Рис. 31: Изменение значения регистра

Выводятся разные значения, так как в первый раз мы вносим значение 2, а во второй - регистр приравниваем к 2. Завершаю выполнение программы и выхожу из GDB. Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, изменяю его название на lab9-3.asm(рис. 32).

Рис. 32: Копирование нужного файла

Рис. 32: Копирование нужного файла

Создаю исполняемый файл. При запуске указываю ключ –args и аргументы(рис. 33).

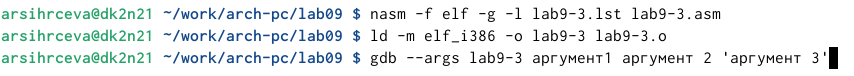


Рис. 33: Запуск исполняемого файла

Устонавливаю точку останова перед первой иструкцией(рис. 34).

Рис. 34: Установка точки останова

Рис. 34: Установка точки останова

Заускаю программу(рис. 35).



Рис. 35: Запуск программы

проверяю адрес вершины стека(рис. 36).

Рис. 36: Адрес вершины стека

Рис. 36: Адрес вершины стека

Убеждаюсь, что в нем хранится 5 аргументов Просматриваю остальные позиции стека(рис. 37), (рис. 38), (рис. 39), (рис. 40), (рис. 41), (рис. 42).

Рис. 37: Позиция стека с адресом имени программы

Рис. 37: Позиция стека с адресом имени программы

Рис. 38: Позиция стека с адресом первого аргумента

Рис. 38: Позиция стека с адресом первого аргумента

Рис. 39: Позиция стека с адресом второго аргумента

Рис. 39: Позиция стека с адресом второго аргумента

Рис. 40: Позиция стека с адресом третьего аргумента

Рис. 40: Позиция стека с адресом третьего аргумента

Рис. 41: Позиция стека с адресом четвёртого аргумента

Рис. 41: Позиция стека с адресом четвёртого аргумента

Рис. 42: Позиция стека с адресом пятого аргумента

Рис. 42: Позиция стека с адресом пятого аргумента

Шаг изменения адреса равен 4, так как стех хранит до 4 байт. Также компьютер использует новый стек для нормального сохранения данных.

#Самостоятельная работа

## 3.1 Задание 1

Копирую файл с самостоятельной работой из лабораторной работы №8 и переименовываю его. Изменяю текст программы так, чтобы значение функции вычислялось как подпрограмма(рис. 43).

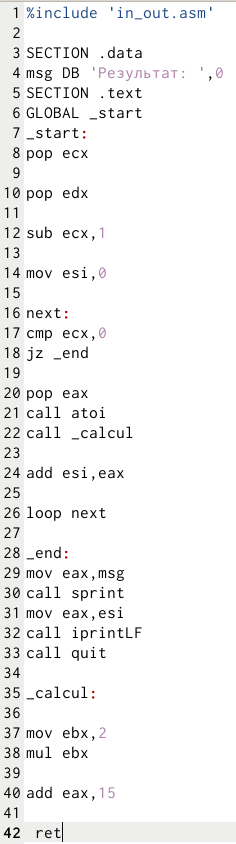


Рис. 43: Изменённый текст программы для вычисления суммы функций

Создаю и запускаю исполняемый файл. Проверяю его работу(рис. 44).

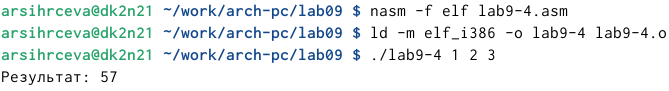


Рис. 44: Результат работы программы

## 3.2 Задание 2

Создаю новый файл для самостоятельной работы(рис. 45).

Рис. 45: Создание файла

Рис. 45: Создание файла

Ввожу текст программы для вычисления выражения(рис. 46).

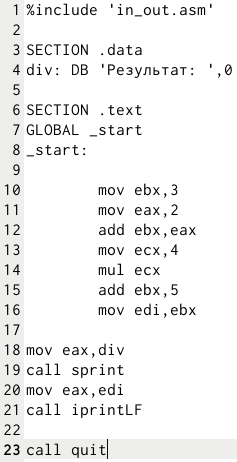


Рис. 46: Текст программы

Создаю и запускаю исполняемый файл, чтобы проверить его работу(рис. 47).

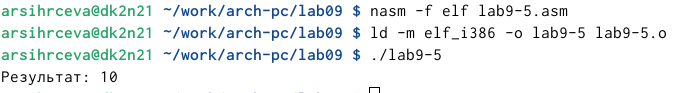


Рис. 47: Результат работы программы

Программа выводит неверный ответ. Вместо 25 - 10 Для поиска ошибки воспользуюсь отладчиком GDB. Загружаю исполняемый файл в данный отладчик, устанавливаю точку останова и запускаю программу(рис. 48).

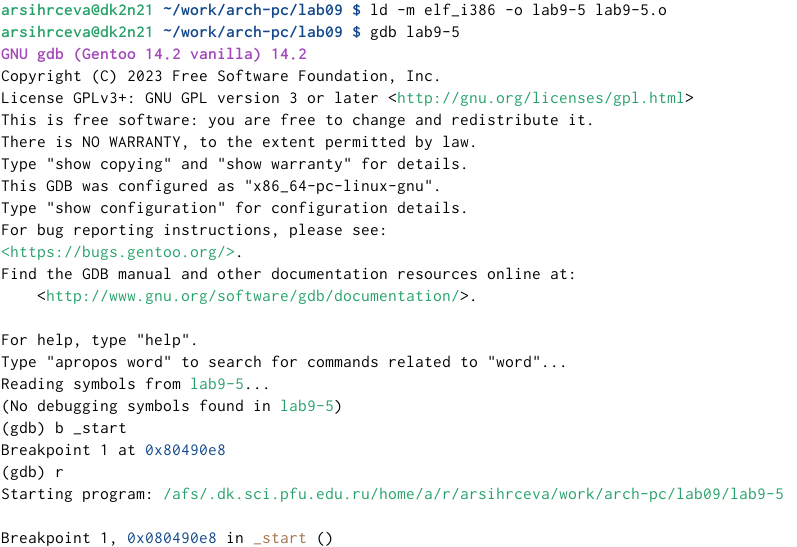


Рис. 48: Запуск программы в отладчике GDB

Включаю режим псевдографики, просматриваю значение регистров(рис. 49).

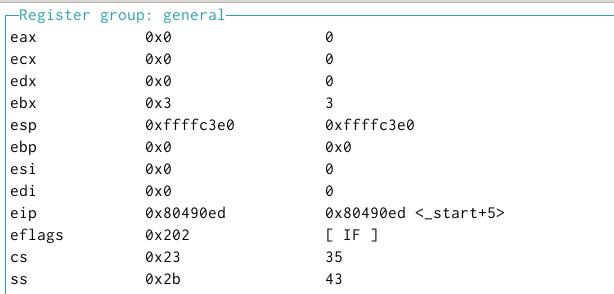


Рис. 49: Значение регистров

Пошагово выполняю инструкции с помощью команды steri. Нахожу ошибку в неправильном порядке операнд. Исправляю это. Создаю и запускаю исполняемый файл. Проверяю его работу(рис. 50).

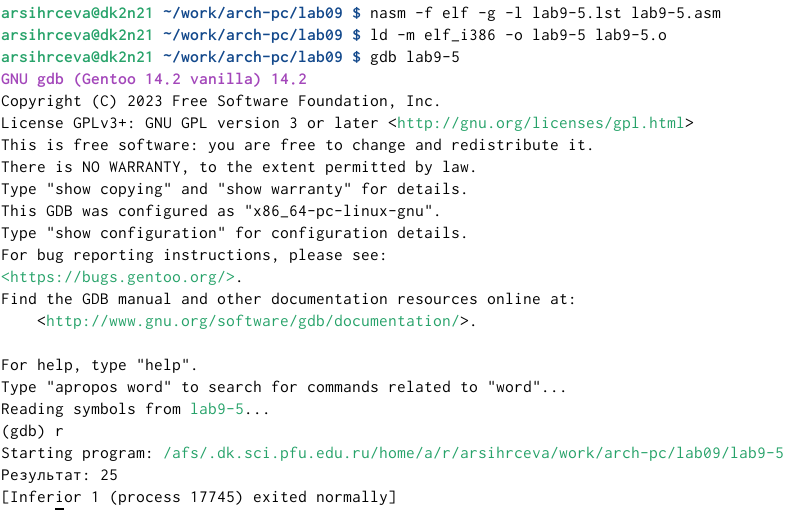


Рис. 50: Проверка работы исправленного кода

# 4 Выводы

Приобретены навыки написания программ с использованием подпрограмм. Я ознакомилась с методами отладки при помощи GDB и его основными возможностями.