Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Налобин Михаил Дмитриевич

Содержание

# 1 Цель работы

Приобрести навык написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Ход работы

Создали каталог lab09 и в данном каталоге файл lab9-1.asm (рис. [1](#fig:001)).

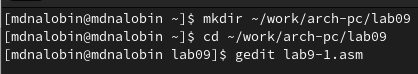


Figure 1: Создание каталога lab09

Переписали в него текст из Листинга 9.1. и, создав исполняемый файл, запустили lab9-1 (рис. [2](#fig:002) и рис. [3](#fig:003)).

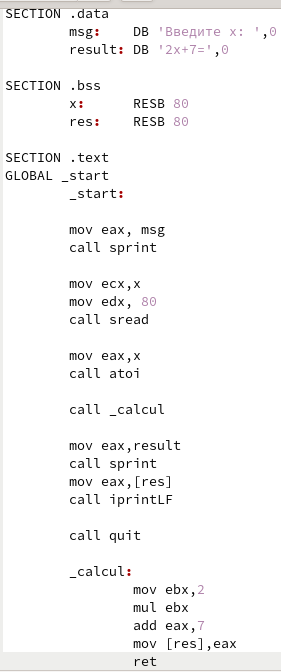


Figure 2: Код программы lab9-1.asm

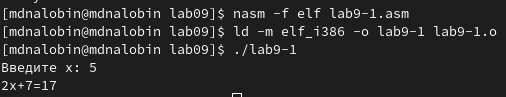


Figure 3: Запуск программы lab9-1

После чего изменили текст файла lab9-1.asm, добавив еще одну подпрограмму, создали исполняемый файл и запустили его (рис. [4](#fig:004)).

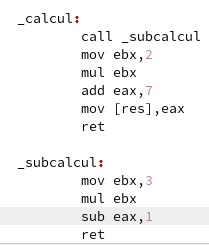


Figure 4: Измененная часть кода программы lab9-1.asm

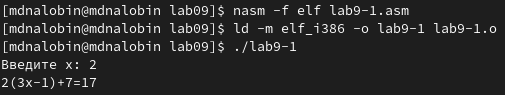


Figure 5: Запуск измененной программы lab9-1

Создали файл lab9-2.asm и заполнили его текстом из Листинга 9.2. (рис. [6](#fig:006) и рис. [7](#fig:007)).

Figure 6: Создание lab9-2.asm

Figure 6: Создание lab9-2.asm

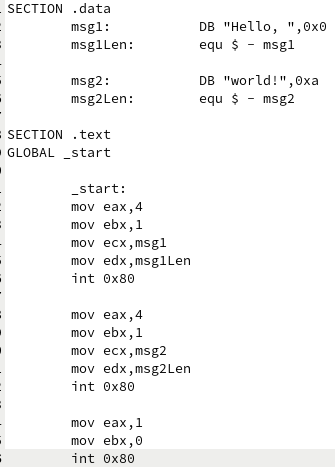


Figure 7: Код программы lab9-2.asm

Создаем исполняемый файл с применением ключа -g и загружаем его в отладчик gdb. Уже в самой оболочке проверяем работу программы и для более подробного анализа устанавливаем брейкпоинт на метку \_start и запускаем снова (рис. [8](#fig:008) и рис. [9](#fig:009)).

Figure 8: Загрузка файла lab9-2 в отладчик

Figure 8: Загрузка файла lab9-2 в отладчик

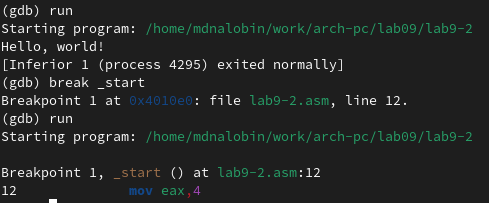


Figure 9: Проверка работы без брейкпоинта и с ним

Далее смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. [10](#fig:010)).



Figure 10: Просмотр дисасимилированного кода программы

Затем переключили на отображение команд с синтаксисом intel, введя set disassembly-flavor intel, и снова применяем команду disassemble. Различие между синтаксисами ATT и Intel заключается в конструкции: у ATT первый после команды идет источник, а после приёмник; у Intel - Приёмник, источник (рис. [11](#fig:011)).



Figure 11: Вид дисасимилированного кода программы с Intel’овским синтаксисом

Включили режим псевдографики для более удобного анализа программы (рис. [12](#fig:012)).

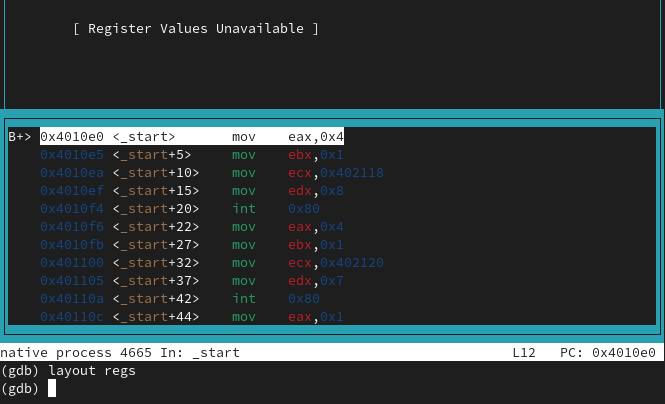


Figure 12: Режим псевдографики

Проверили установленную точку останова командой info breakpoints (кратко i b), после установили еще одну точку останова и проверили это (рис. [13](#fig:013)).

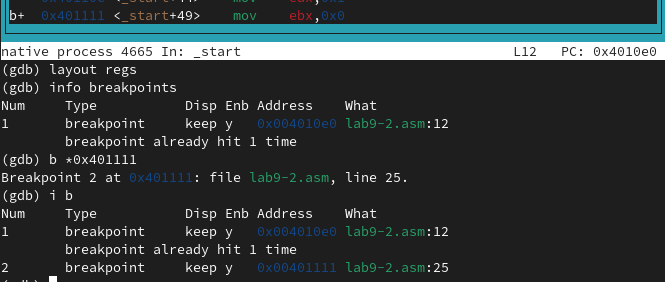


Figure 13: Проверка и установка точек останова

Выполнили 5 инструкций с использованием команды stepi, проследив за изменениями регистров, и заметили это у регистров eax, ecx и edx на рис. 14-2, 14-4 и 14-5 соответственно (рис. [14](#fig:014), рис. [15](#fig:015), рис. [16](#fig:016), рис. [17](#fig:017) и рис. [18](#fig:018)).

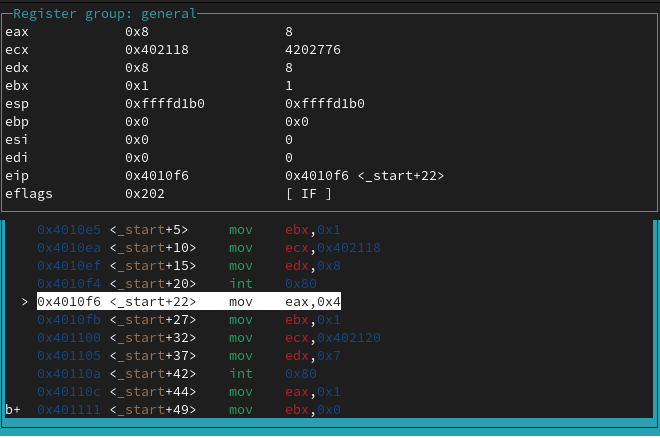


Figure 14: Просмотр 6 строчки после \_start

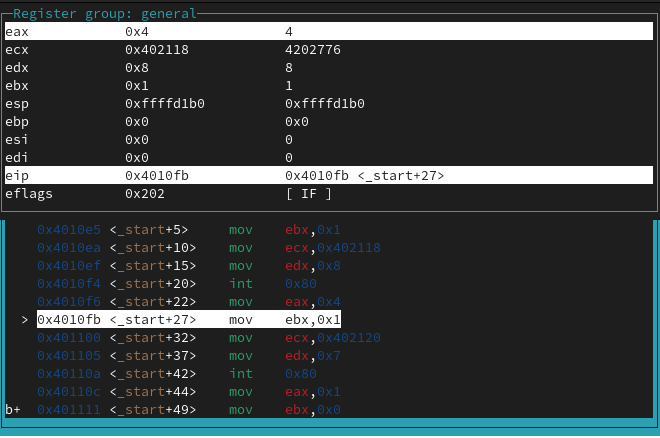


Figure 15: Просмотр 7 строчки после \_start

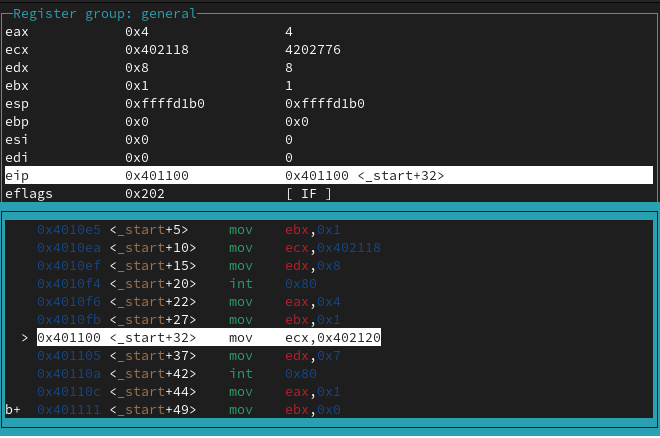


Figure 16: Просмотр 8 строчки после \_start

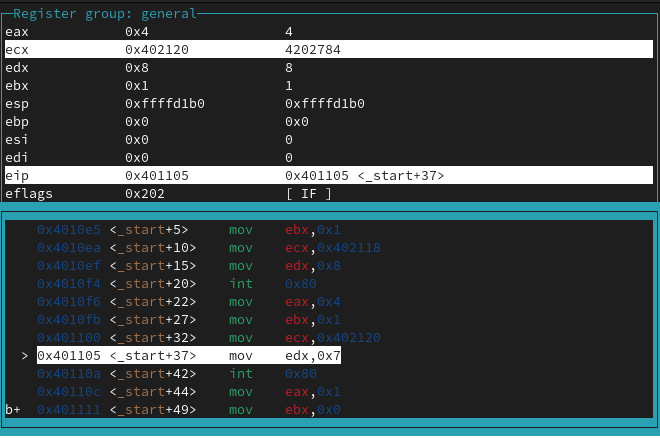


Figure 17: Просмотр 9 строчки после \_start

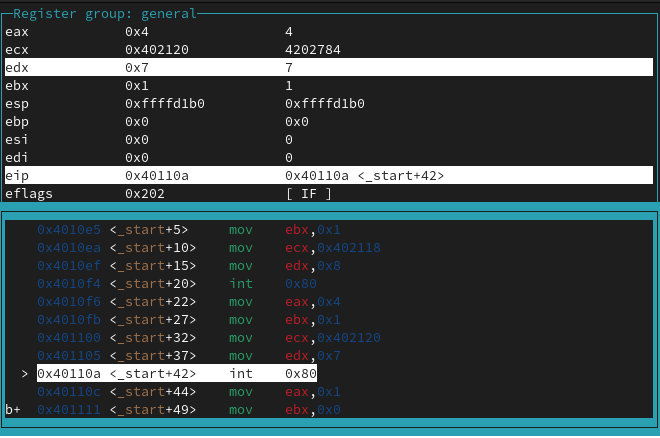


Figure 18: Просмотр 10 строчки после \_start

Посмотрели содержимое регистров с помощью команды info refisters (кратко i r) (рис. [19](#fig:019)).

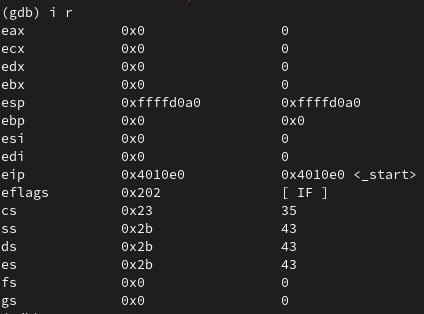


Figure 19: Просмотр содержимого регистров

Посмотрели значение переменной msg1 по имени и msg2 по адресу (рис. [20](#fig:020)).

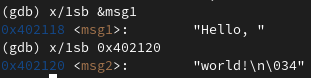


Figure 20: Просмотр содержимого msg1 и msg2

Изменили командой set первого символа в msg1 и для втрого в msg2 (рис. [21](#fig:021) и рис. [22](#fig:022)).

Figure 21: Изменение первого символа msg1

Figure 21: Изменение первого символа msg1

Figure 22: Изменение второго символа msg2

Figure 22: Изменение второго символа msg2

Вывели в различных форматах значение регистра edx (рис. [23](#fig:023)).

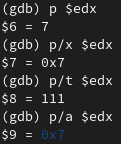


Figure 23: Вывод значения регистра edx в разных форматах

С помощью команды set изменили значение ebx на символ 2, далее изменили значение на число 2. Так как в первом случае символ, а во втором число, то и выводится разное значение (рис. [24](#fig:024) и рис. [25](#fig:025)).

Figure 24: Вывод символа 2 в регистре edx

Figure 24: Вывод символа 2 в регистре edx

Figure 25: Вывод числа 2 в регистре edx

Figure 25: Вывод числа 2 в регистре edx

Далее завершаем выполнение команды с помощью continue и выходис из GDB с помощью quit.

Копируем файл lab8-2.asm в наш каталог с именем lab9-3.asm и создаем исполняемый файл (рис. [26](#fig:026) и рис. [27](#fig:027)).

Figure 26: Копирование lab8-2.asm

Figure 26: Копирование lab8-2.asm

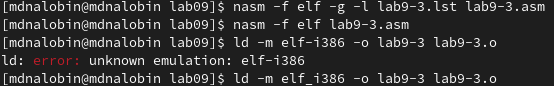


Figure 27: Создание исполняемого файла для lab9-3

С указанием аргументов загрузили исполняемый файл в отладчик, затем для исследования стека установили точку основы и запустили ее (рис. [28](#fig:028)).

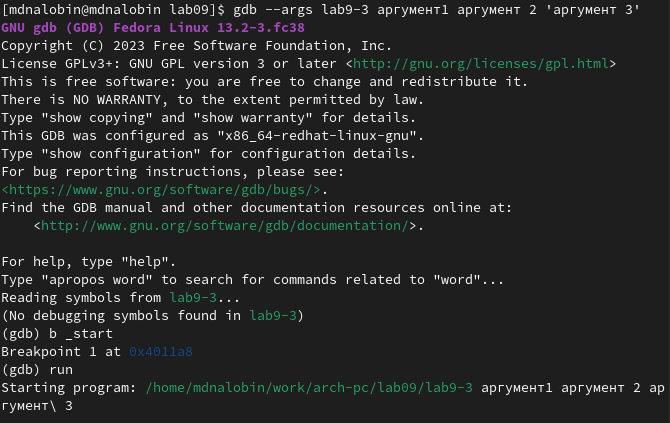


Figure 28: Исследование стека с помощью отладки

Просматриваем регистр esp, ибо в нем хранится число аргументов командой строки, включая имя программы и просматриваем позиции стека, изменяя шаг изменения адреса на 4 по причине того, что элемент стека занимает 4 байта (рис. [29](#fig:029) и рис. [30](#fig:030))

Figure 29: Просмотр регистра esp

Figure 29: Просмотр регистра esp

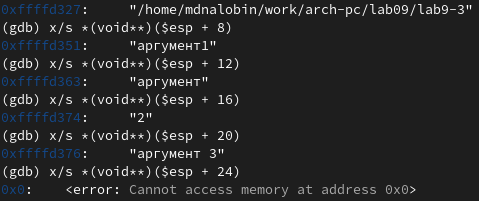


Figure 30: Просмотр позиций стека

##Самостоятельная работа

Копируем sr.asm из каталога lab08 в lab09 и изменяем таким образом, чтобы вычисление значения функции f(x) было подпрограммой (рис. [31](#fig:031) и рис. [32](#fig:032))

Figure 31: Копирование sr.asm

Figure 31: Копирование sr.asm

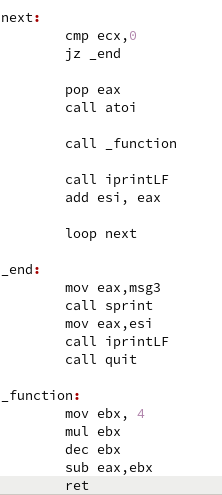


Figure 32: Код преобразованной программы sr.asm

Затем создаем исполняем файл и проверяем на правильную работу (рис. [33](#fig:033)).

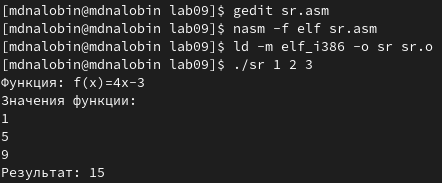


Figure 33: Выполнение программы sr.asm

Создаем файл lab9-4.asm, записываем в него содержимое Листинга 9.3., создаем исполняемый файл и проверяем работу (рис. [34](#fig:034) и рис. [35](#fig:035))

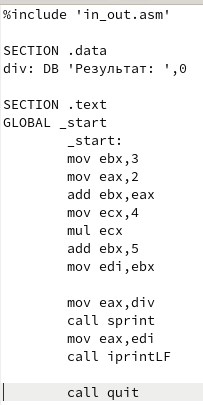


Figure 34: Код программы lab9-4.asm

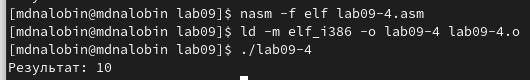


Figure 35: Выполнение команды lab9-4

С помощью отладчика GDB анализируем изменения значений регистров, чтобы определить место ошибок. В итоге обнаруживаем ошибку *в строке с mul ecx, потому что до этого мы прибавляем к ebx*, но команда mul может умножать лишь на eax, поэтому должны из строчки add ebx,eax получить add eax,ebx. *Следующая ошибка кроется в строчке add ebx,5, ведь ранее проводили операции над eax и именно eax несет в себе значение уже выполненных действий* (рис. [36](#fig:036) и рис. [37](#fig:037))

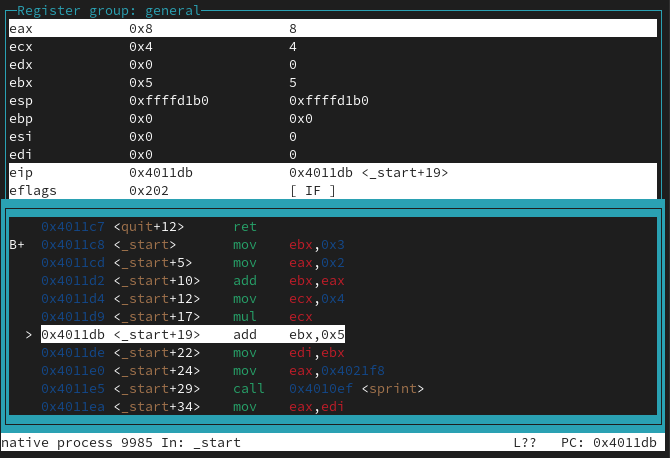


Figure 36: Первая ошибка в lab9-4.asm

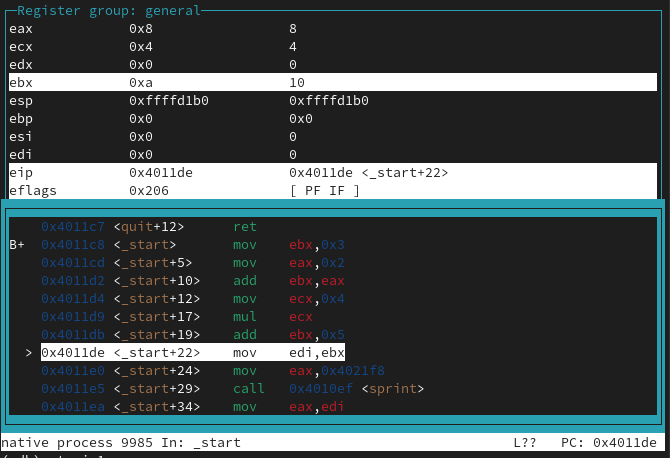


Figure 37: Первая ошибка в lab9-4.asm

В этом этапе исправляем все найденные ошибки, создаем исполняемый файл и проверяем работу(рис. [38](#fig:038) и рис. [39](#fig:039))

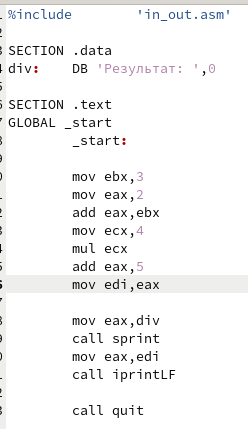


Figure 38: Код измененной программы lab9-4.asm

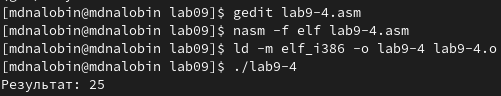


Figure 39: Выполнение измененной команды lab9-4

# 3 Выводы

В ходе данной длительной лабораторной работы приобрели навык, с помощью которого сможем писать программы с использованием подпрограмм, и познакомились с методами отладки с использованием GDB и его основными возможностями.

:::