Лабораторная работа №10

Дисциплина: Архитектура компьютера

Ким Ангелина Павловна

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

Создали каталог для выполнения лабораторной работы №10, переходим в него и создаем файл lab10-1.asm (рис. 1)

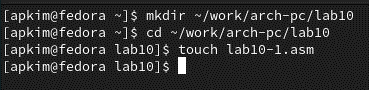


Рис. 1: Создание каталога

Введем в файл lab10-1.asm текст программы из листинга 10.1. (рис. 2)

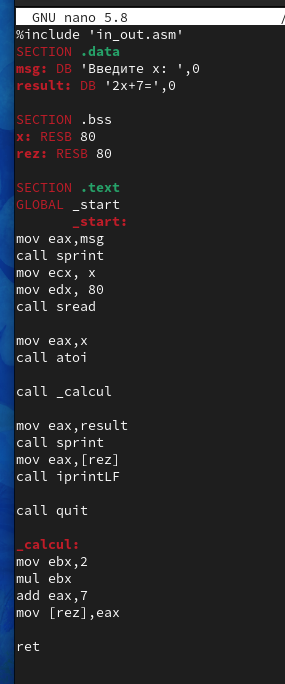


Рис. 2: Текст программы из листинга 10.1

Создаем исполняемый файл и проверяем его работу (рис. 3)

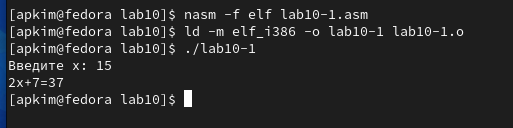


Рис. 3: Создание исполняемого файла

Изменяем текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul (рис. 4)

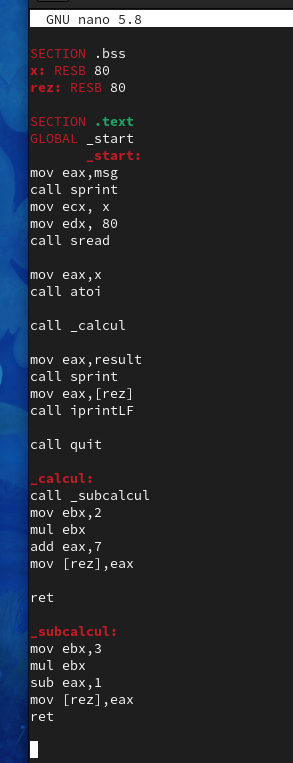


Рис. 4: Текст измененной программы

Создаем исполняемый файл и проверяем его работу (рис. 5)

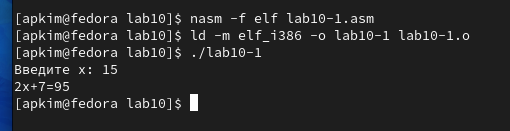


Рис. 5: Создание исполняемого файла

Далее создаем файл lab10-2.asm, вносим туда текст программы из листинга 10.2, создаем исполняемый файл, для работы с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом “-g”. Загружаем исполняемый файл в отладчик GDB. Проверили работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. 6)

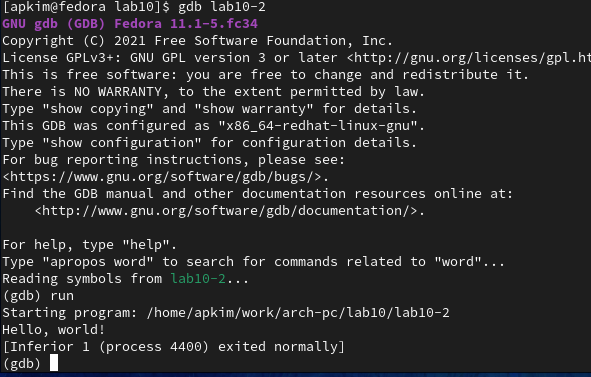


Рис. 6: Исполняемый файл в отладчик GDB

Установили брейкпоинт на метку \_start (рис. 7)

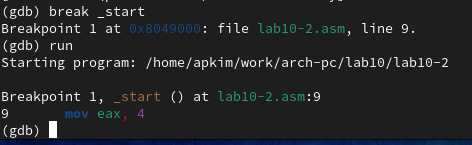


Рис. 7: Установка брейкпоинта

Посмотрели дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 8)

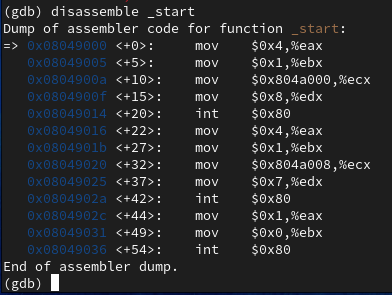


Рис. 8: Дисассимилированный код

Переключились на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel.(рис. 9)

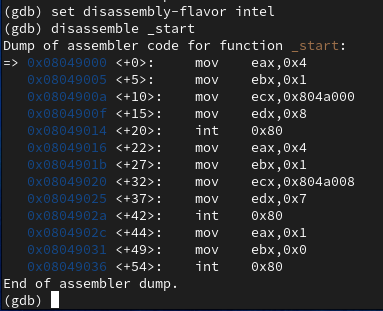


Рис. 9: Команда set disassembly-flavor intel

Различия отображения синтаксиса машинных кодов в режимах ATT и Intel: в АТТ перед адресом регистра ставится $, а перед названием регистра %, сначала записывается адрес, а потом - регистр. В Intel сначала регистр, а потом адрес, и перед ними ничего не ставится. Далее переходим на режим псевдографики для более удобного анализа программы. (рис. 10)

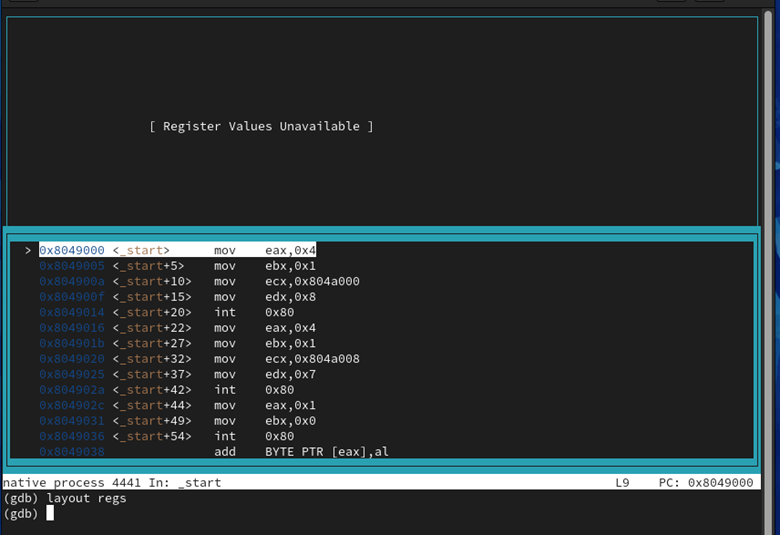


Рис. 10: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяем это с помощью команды info breakpoints. (рис. 11)

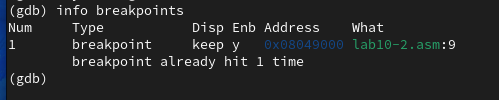


Рис. 11: Команда info breakpoints

Далее определили адрес предпоследней инструкции и установили точку останова (рис. 12)

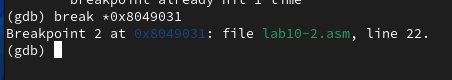


Рис. 12: Адрес предпоследней инструкции

Посмотрели информацию о всех установленных точках останова (рис. 13)

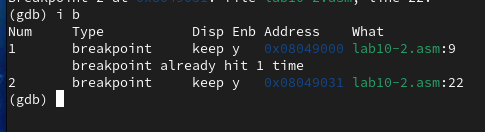


Рис. 13: Информация о всех установленных точках останова

Выполняем 5 инструкций с помощью команды stepi и проследим за изменением значения регистров (рис. 14)

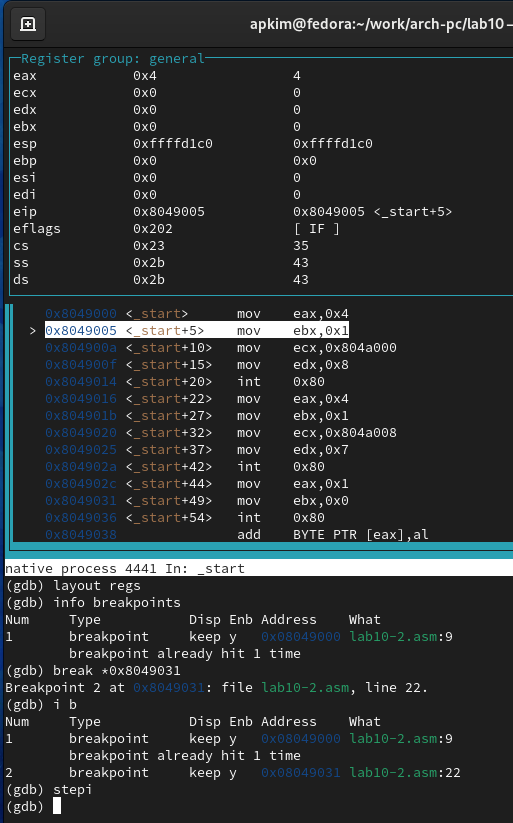


Рис. 14: Команда stepi (1)

Изменяются значения регистров: eax, ecx, edx, ebx (рис. 15)

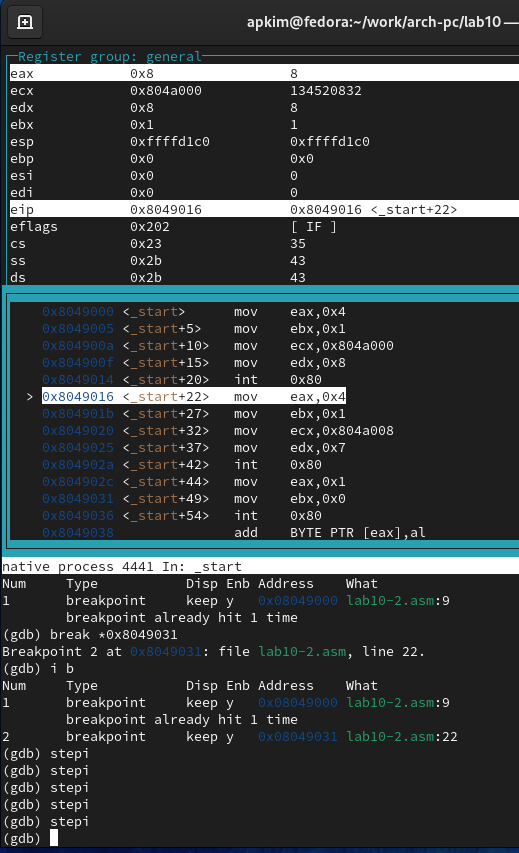


Рис. 15: Команда stepi (2)

Посмотрели содержимое регистров с помощью команды info registers (рис. 16)

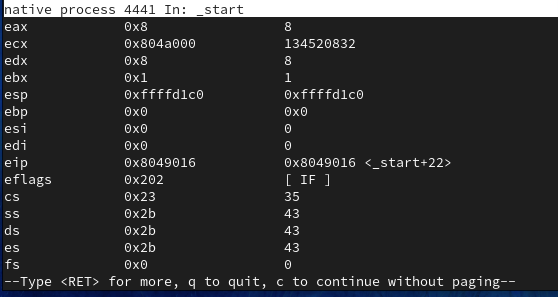


Рис. 16: Содержимое регистров

Далее посмотрели значение переменной msg1 по имени (рис. 17)

Рис. 17: Значение переменной msg1

Рис. 17: Значение переменной msg1

Посмотрели значение переменной msg2 по адресу. Адрес переменной определили по дизассемблированной инструкции. Посмотрели инструкцию mov ecx, msg2 которая записывает в регистр ecx адрес переменной msg2. (рис. 18)

Рис. 18: Значение переменной msg2

Рис. 18: Значение переменной msg2

Далее изменяем первый символ переменной msg1 (рис. 19)

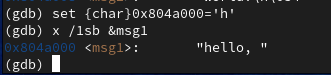


Рис. 19: Первый символ переменной msg1

Заменяем первый символ во второй переменной msg2 (рис. 20)

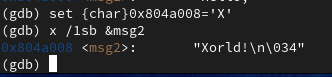


Рис. 20: Первый символ переменной msg2

Чтобы посмотреть значения регистров используется команда print /F . Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.(рис. 21)

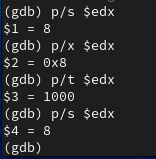


Рис. 21: Значения регистра edx

С помощью команды set изменим значение регистра ebx (рис. 22)

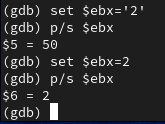


Рис. 22: Команда set

Завершили выполнение программы с помощью команды continue и выходим из GDB (рис. 23)

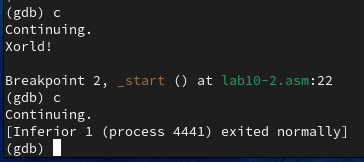


Рис. 23: Завершение программы

Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm (рис. 24)

Рис. 24: Копия файла

Рис. 24: Копия файла

Создаем исполняемый файл (рис. 25)

Рис. 25: Создание исполняемого файла

Рис. 25: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузили исполняемый файл в отладчик, указав аргументы. Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследовали расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установили точку останова перед первой инструкцией в программе и запустили ее (рис. 26)

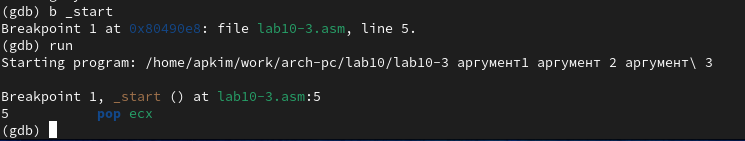


Рис. 26: Загрузка исполняемого файла в отладчик

Адрес вершины стека храниться в регистре esp и по этому адресу распола- гается число равное количеству аргументов командной строки (включая имя программы) (рис. 27)

Рис. 27: Регистр esp

Рис. 27: Регистр esp

Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрели остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] - второго и т.д. Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.) потому что в теле цикла next 4 строки кода. (рис. 28)

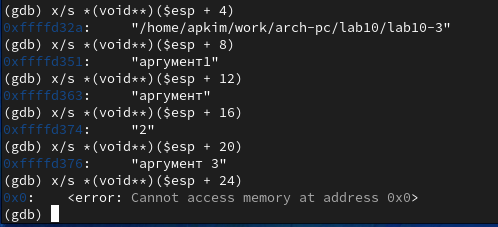


Рис. 28: Позиции стека

Задание для самостоятельной работы. 1 задание. Текст программы из лабораторной работы №9, реализовав вычисление значения функции как подпрограмму. (рис. 29)

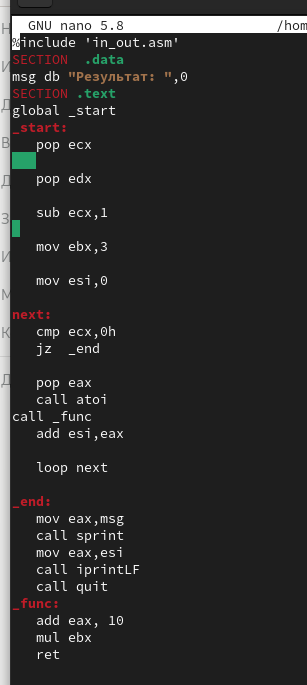


Рис. 29: Текст программы с подпрограммой

Создаем исполняемый файл и проверяем его работу. (рис. 30)

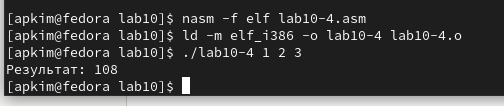


Рис. 30: Создание исполняемого файла

Создаем файл lab10-5.asm (рис. 31)

Рис. 31: Создание файла

Рис. 31: Создание файла

Вводим в него текст программы из листинга 10.3 (рис. 32)

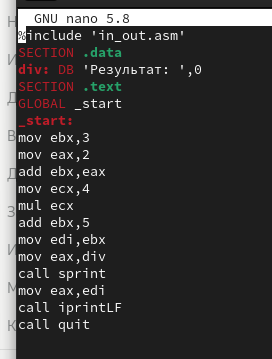


Рис. 32: Текст программы из листинга 10.3

Создаем исполняемый файл и проверяем его. (рис. 33)

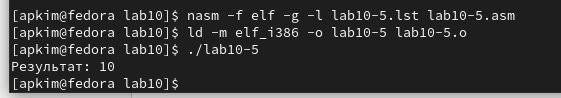


Рис. 33: Создание исполняемого файла

Запустили файл в отладчик GDB. Установили точку останова, запустили код, включили режим псевдографики. (рис. 34)

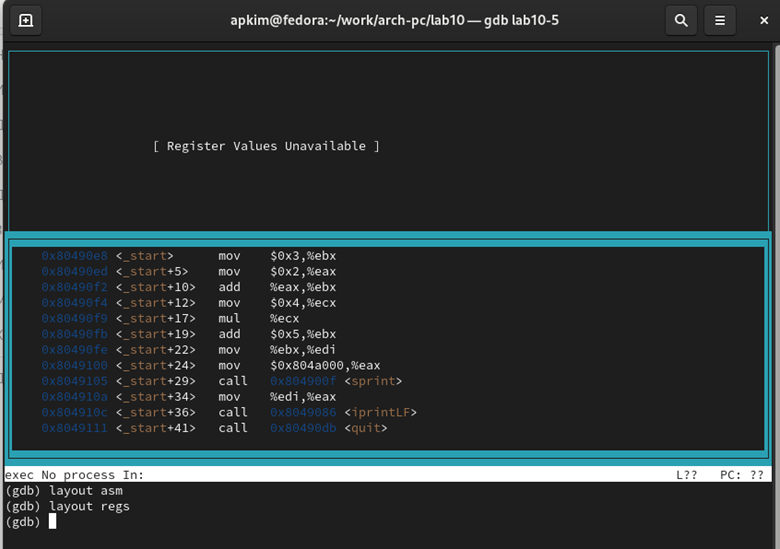


Рис. 34: Режим псевдографики

Пошагово проходим строчки кода (рис. 35)

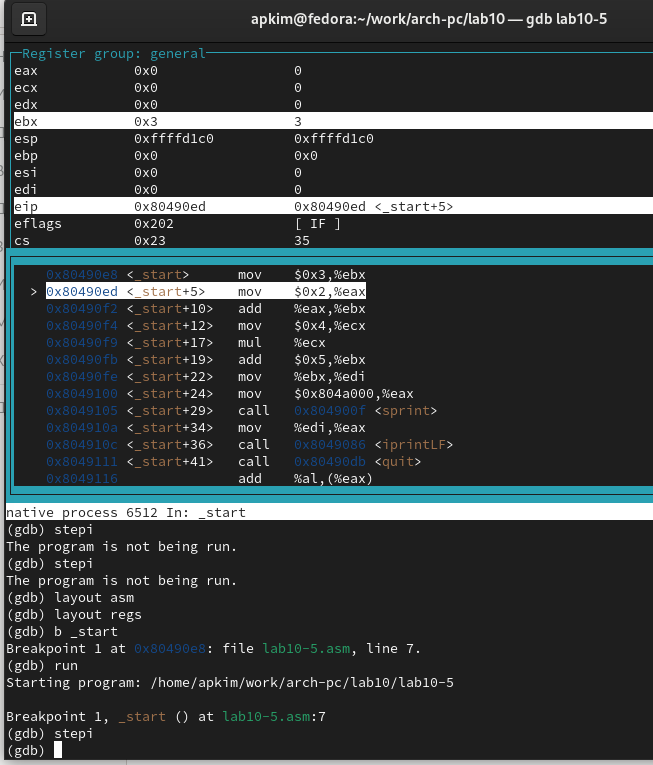


Рис. 35: Строчки кода (1)

(рис. 36)

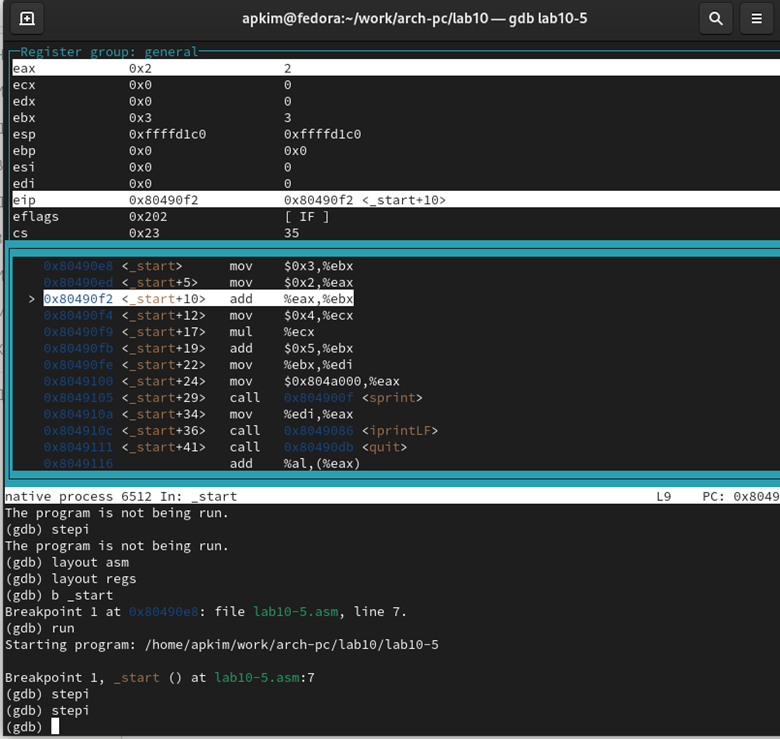


Рис. 36: Строчки кода (2)

(рис. 37)

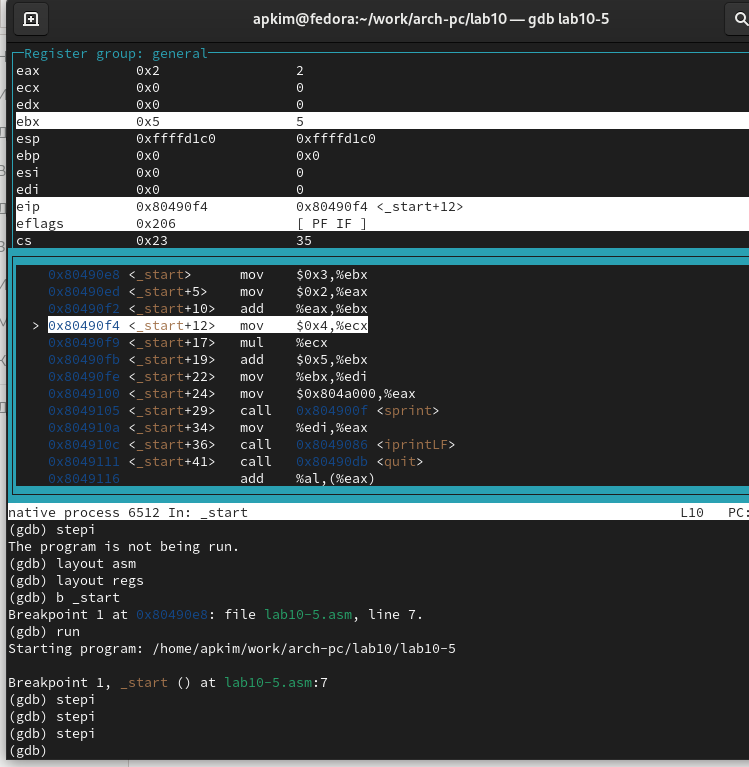


Рис. 37: Строчки кода (3)

(рис. 38)

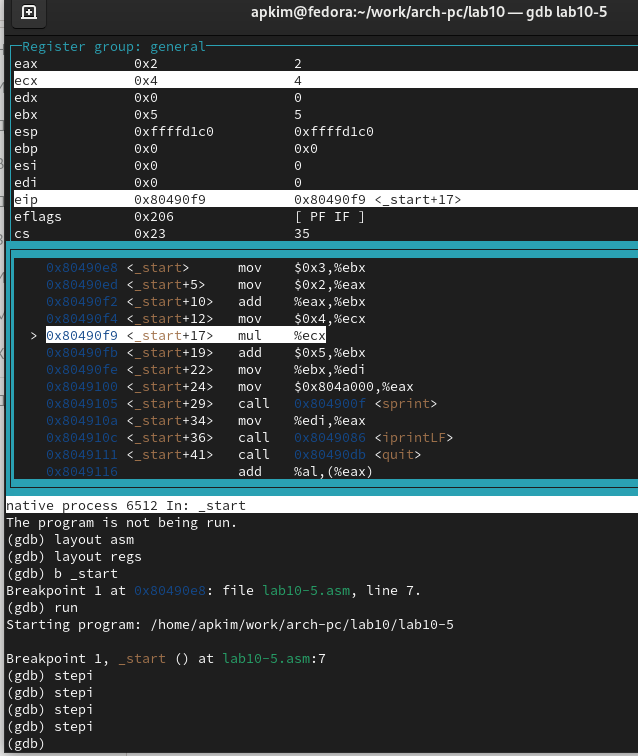


Рис. 38: Строчки кода (4)

(рис. 39)

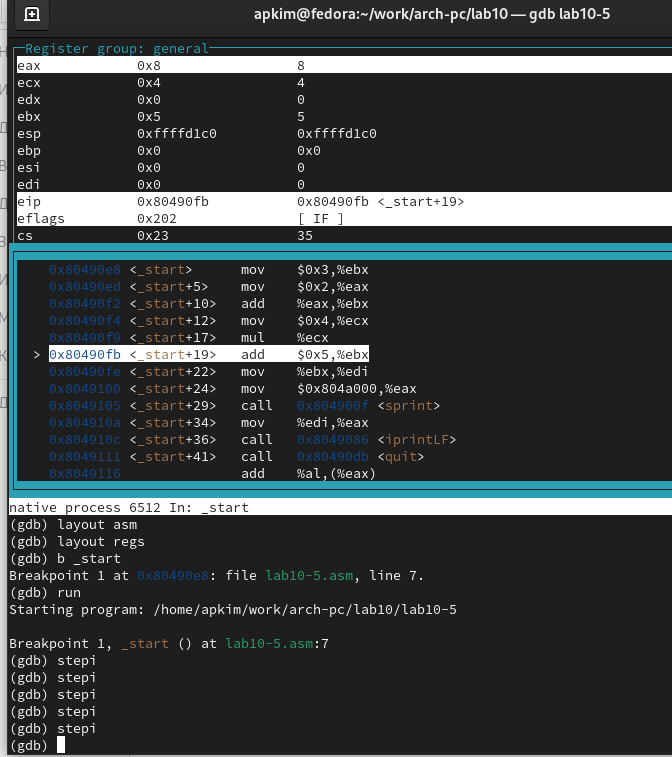


Рис. 39: Строчки кода (5)

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 1)

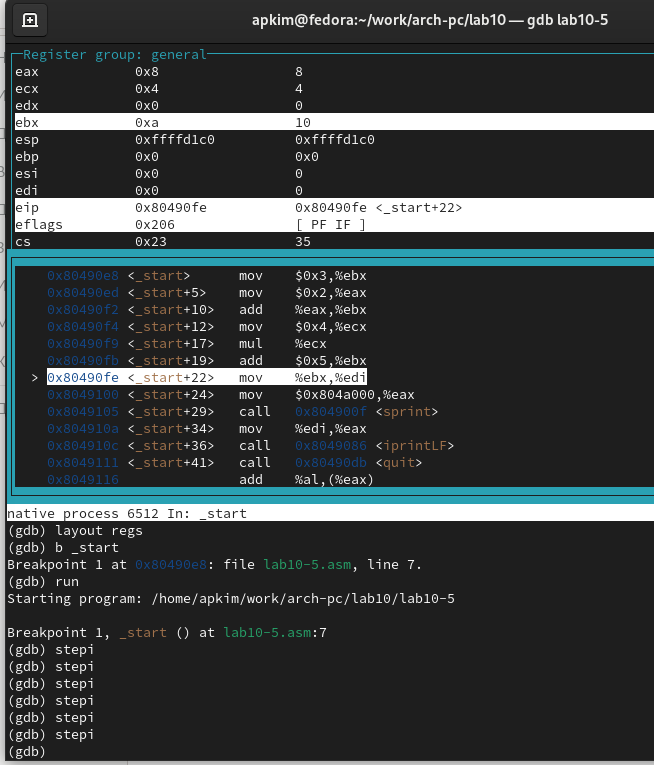


Рис. 40: Строчки кода (6)

(рис. 41)

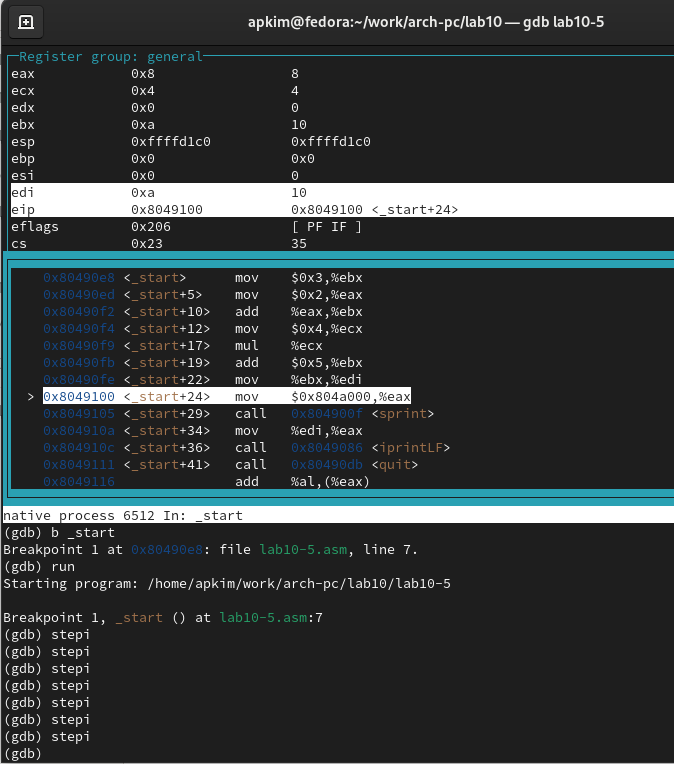


Рис. 41: Строчки кода (7)

(рис. 42)

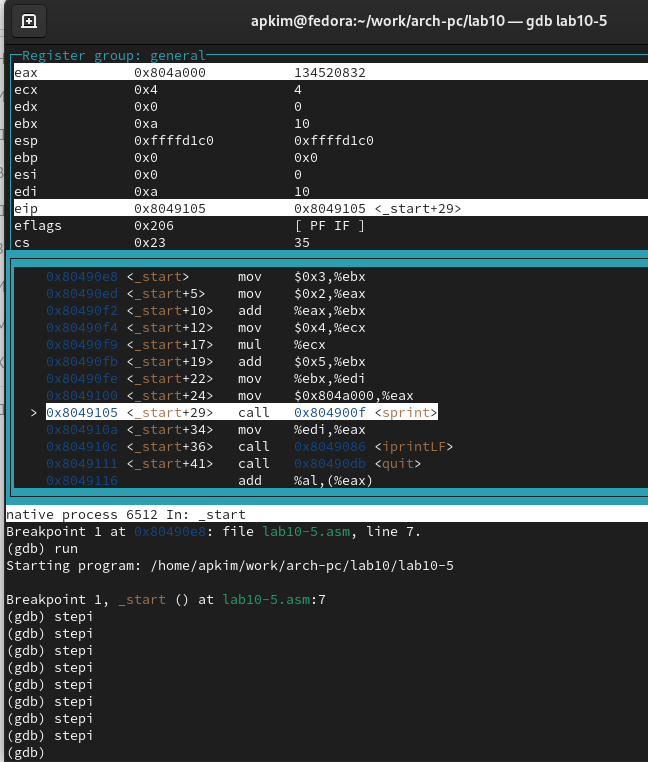


Рис. 42: Строчки кода (8)

Исправленный код (рис. 43)

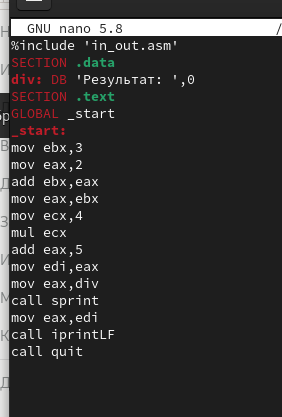


Рис. 43: Исправленный код

Запускаем исполняемый файл и проверяем его работу (рис. 44)

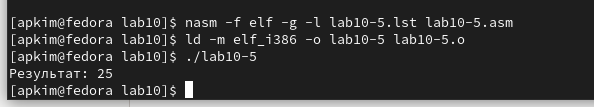


Рис. 44: Исполняемый файл

# 3 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм, ознакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы