Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Камбунду Паулине

Содержание

# 1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

# 2 Выполнение лабораторной работы

## 2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. fig. 1).

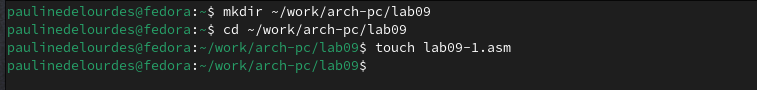


Рис. 1: Создаем каталог с помощью команды mkdir и файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. fig. 2).

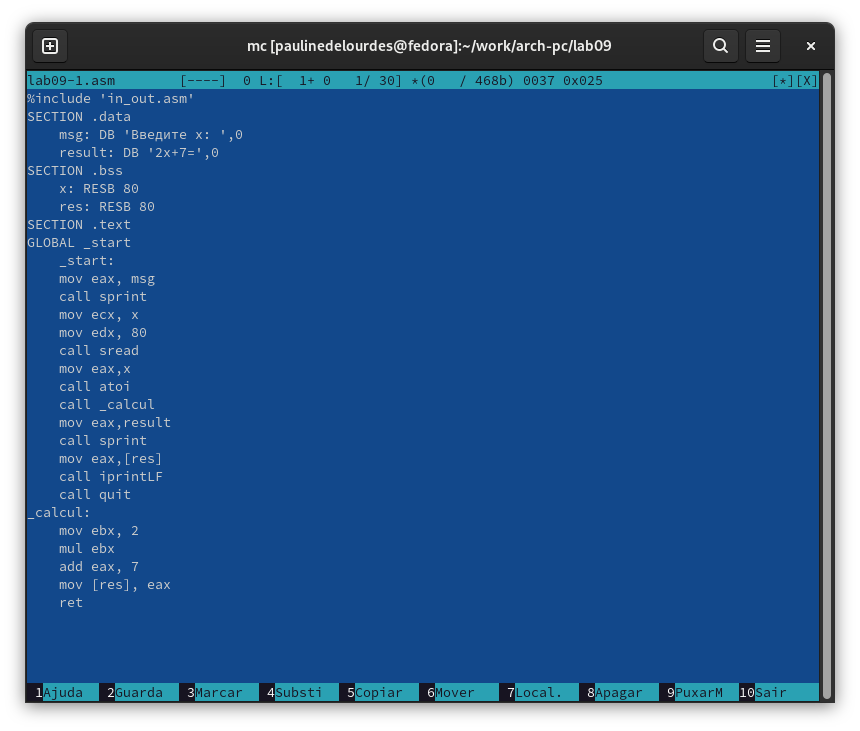


Рис. 2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. fig. 3).

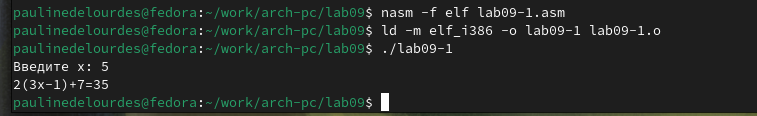


Рис. 3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. fig. 4).

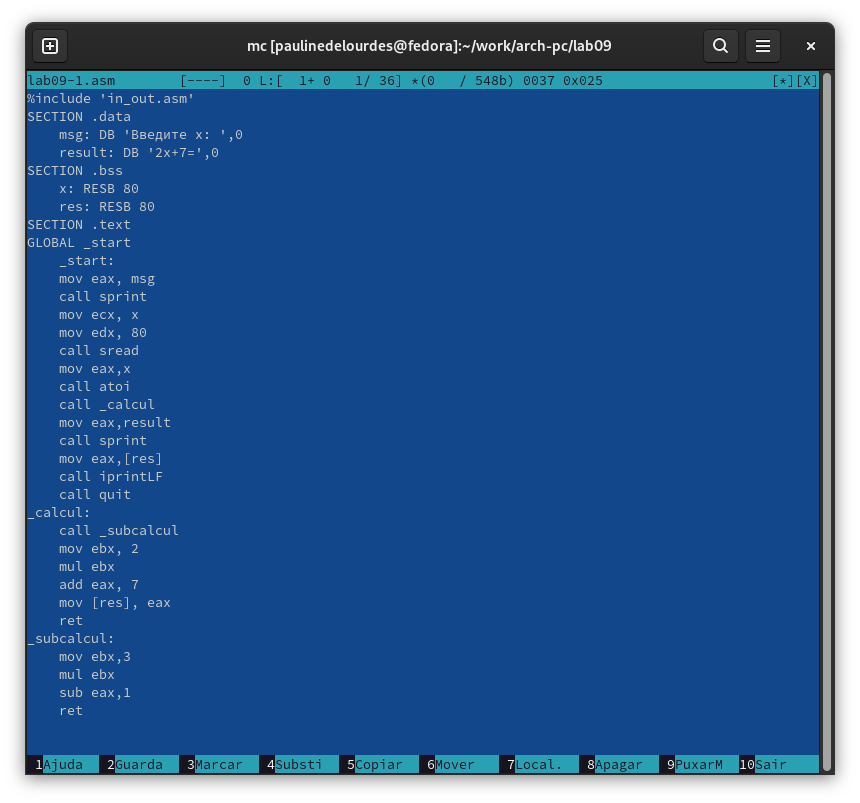


Рис. 4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. fig. 5).

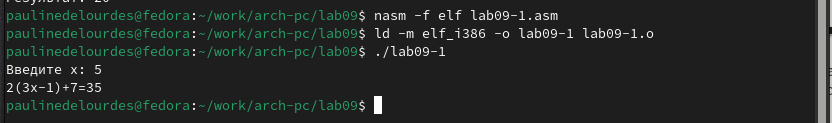


Рис. 5: Запускаем файл и смотрим на его работу

## 2.2 Отладка программам с помощью GDB

Создаем новый файл в каталоге(рис. fig. 6).

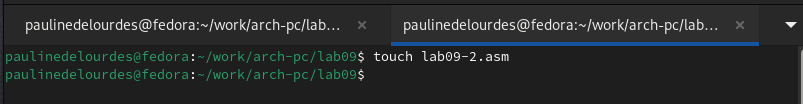


Рис. 6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. fig. 7).

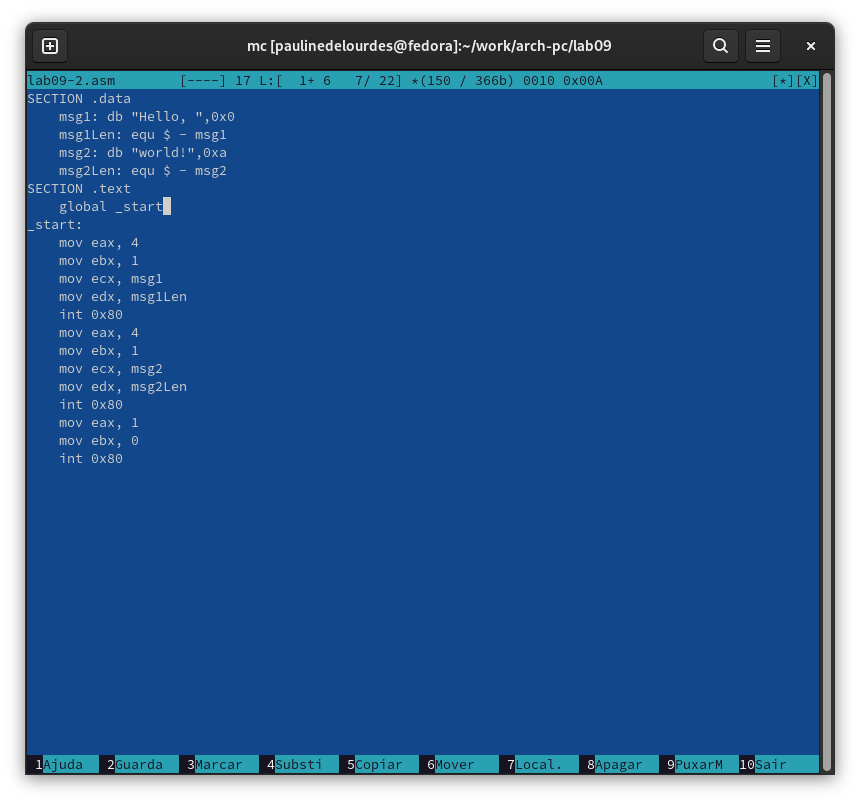


Рис. 7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. fig. 8).

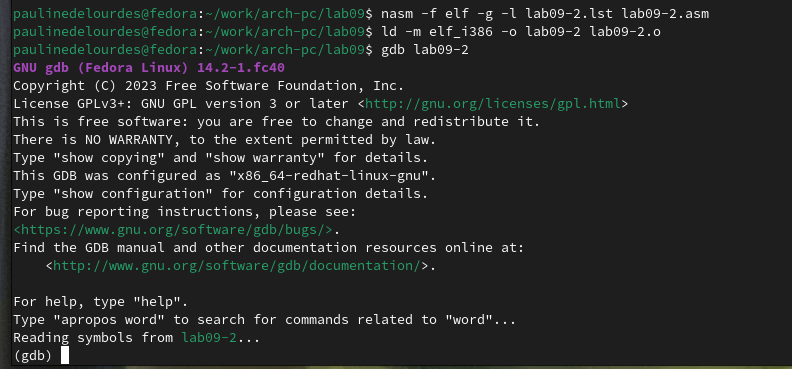


Рис. 8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. fig. 9).

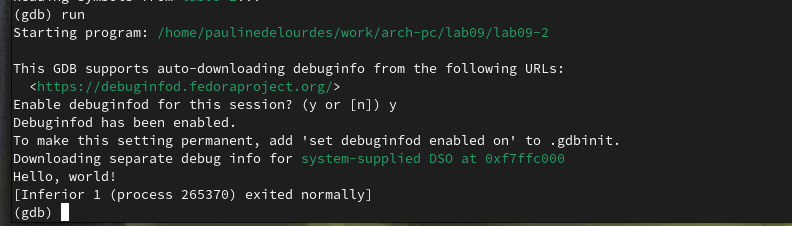


Рис. 9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку \_start и запускаем программу (рис. fig. 10).

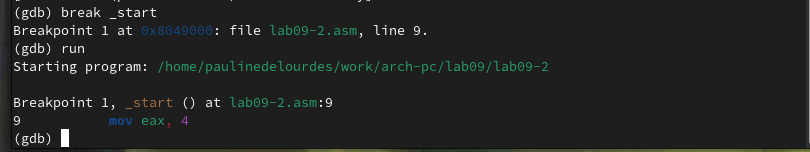


Рис. 10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start(рис. fig. 11).

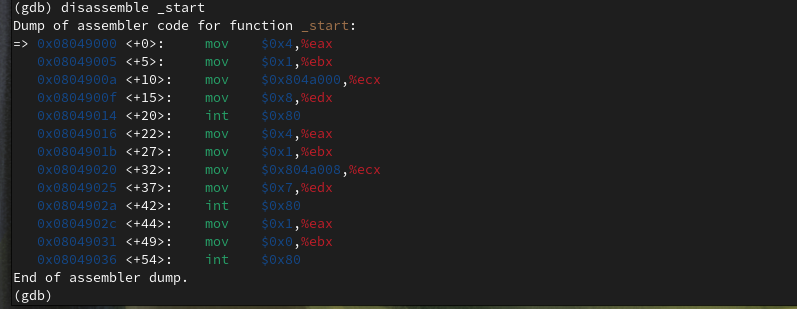


Рис. 11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel’овским синтаксисом (рис. fig. 12).

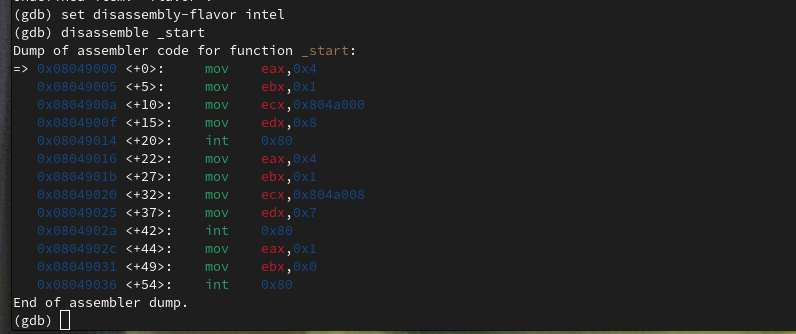


Рис. 12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1.Порядок операндов: В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом “$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "$”.

5.Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. fig. 13).

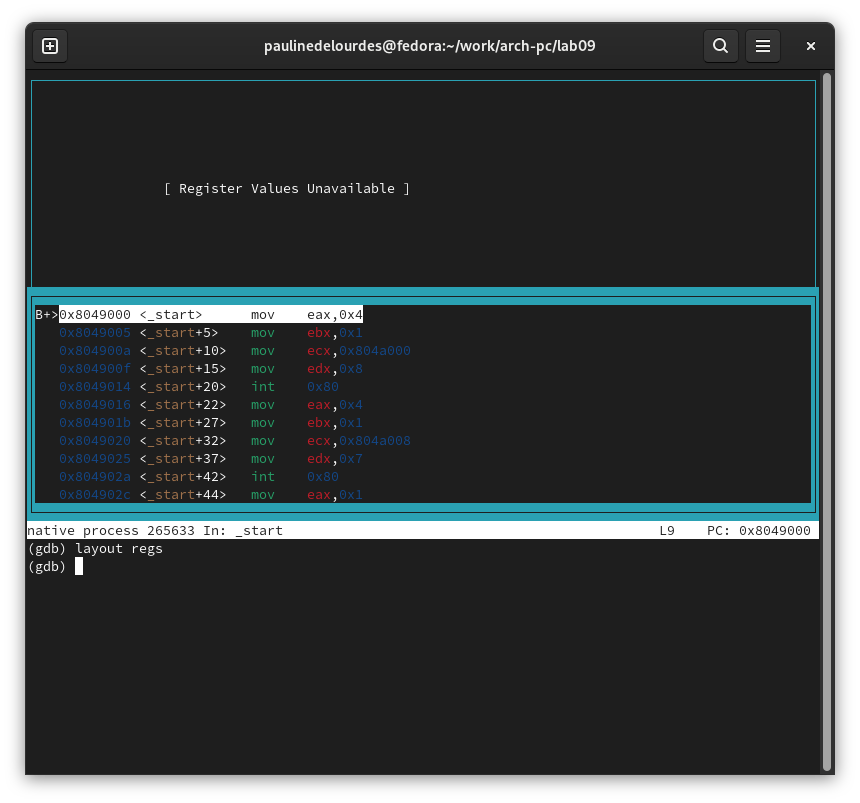


Рис. 13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. 14).

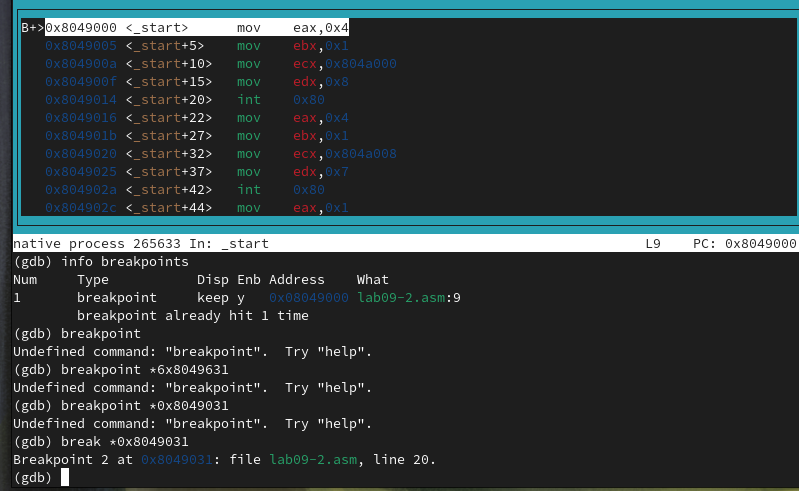


Рис. 14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. fig. 15).

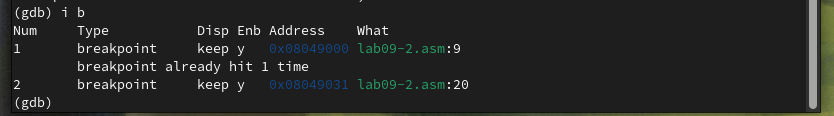


Рис. 15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. fig. 16).

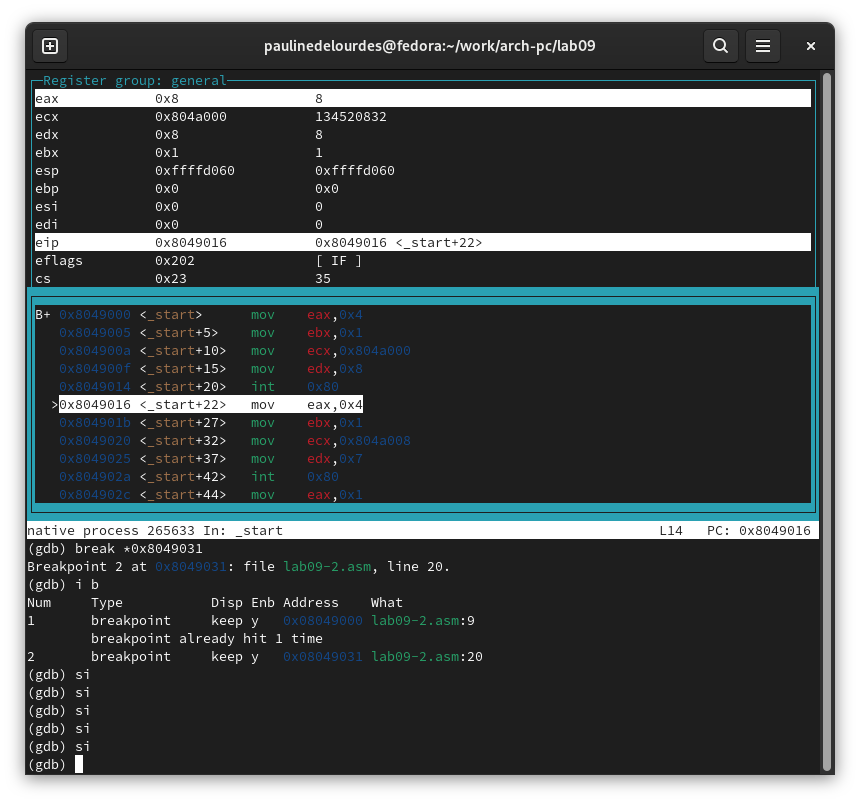


Рис. 16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx,eax, eip.

Смотрим значение переменной msg1 по имени (рис. fig. 17).

Смотрим значение переменной

Рис. 17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. fig. 18).

Смотрим значение переменной

Рис. 18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. fig. 19).

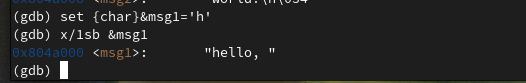


Рис. 19: Меняем символ

Изменим первый символ переменной msg2 (рис. fig. 20).

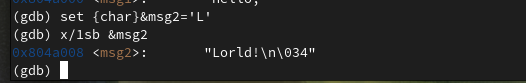


Рис. 20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. fig. 21).

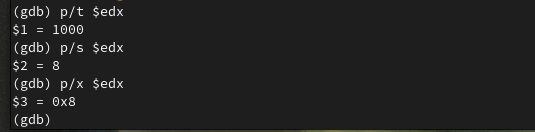


Рис. 21: Смотрим значение регистра

Изменяем регистор ebx (рис. fig. 22).

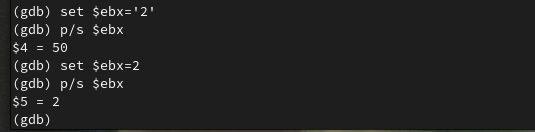


Рис. 22: Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. fig. 23).

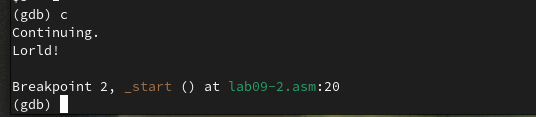


Рис. 23: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. fig. 24).

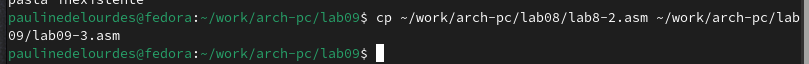


Рис. 24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. fig. 25).

Создаем и запускаем в отладчике файл

Рис. 25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. fig. 26).

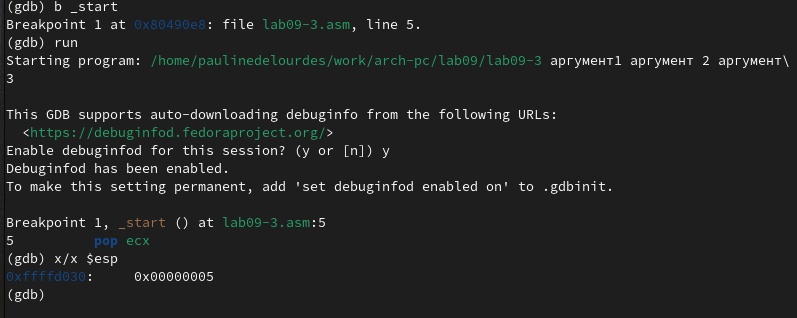


Рис. 26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. fig. 27).

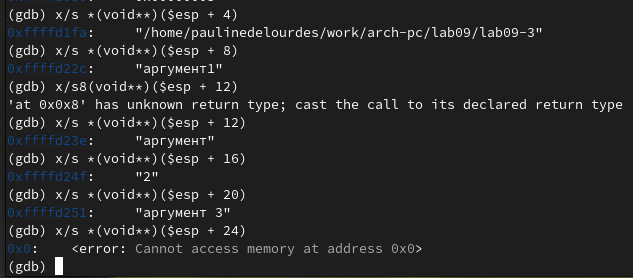


Рис. 27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

###Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. fig. 28).

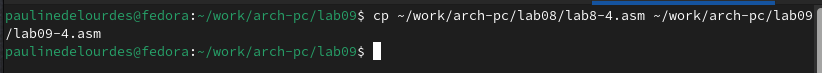


Рис. 28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. fig. 29).

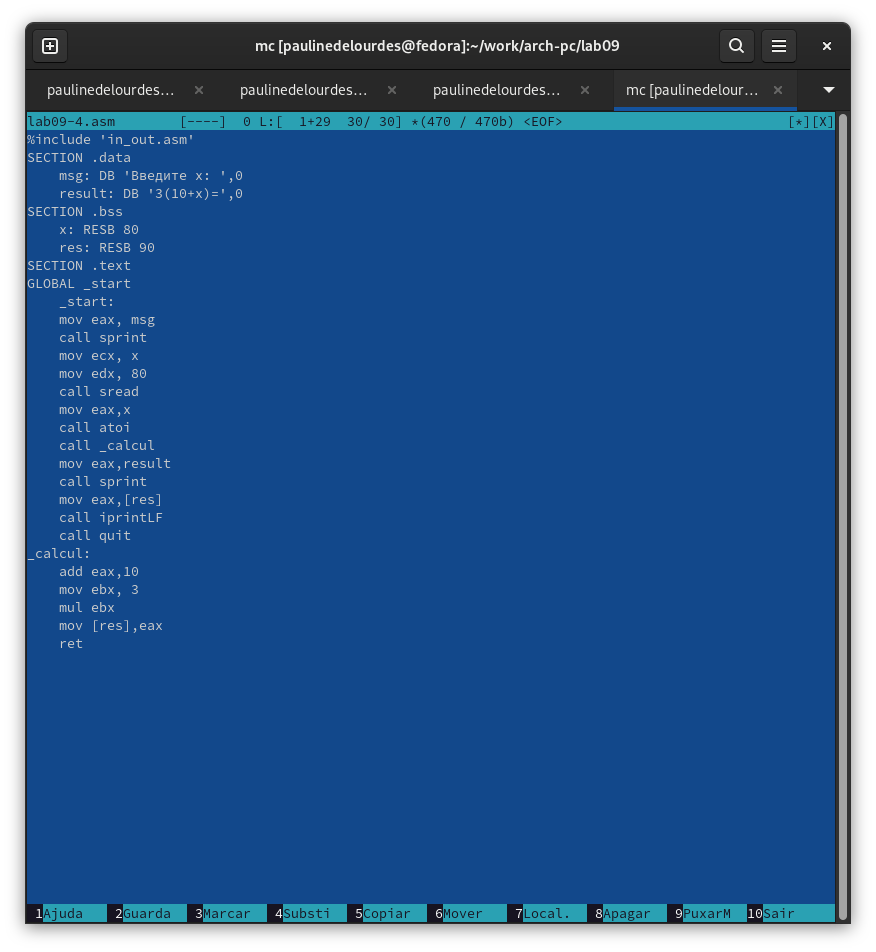


Рис. 29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. fig. 30).

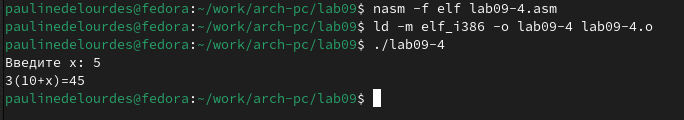


Рис. 30: Проверяем работу программы

###Задание 2

Создаем новый файл в дирректории (рис. fig. 31).

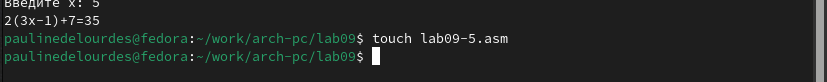


Рис. 31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. fig. 32).

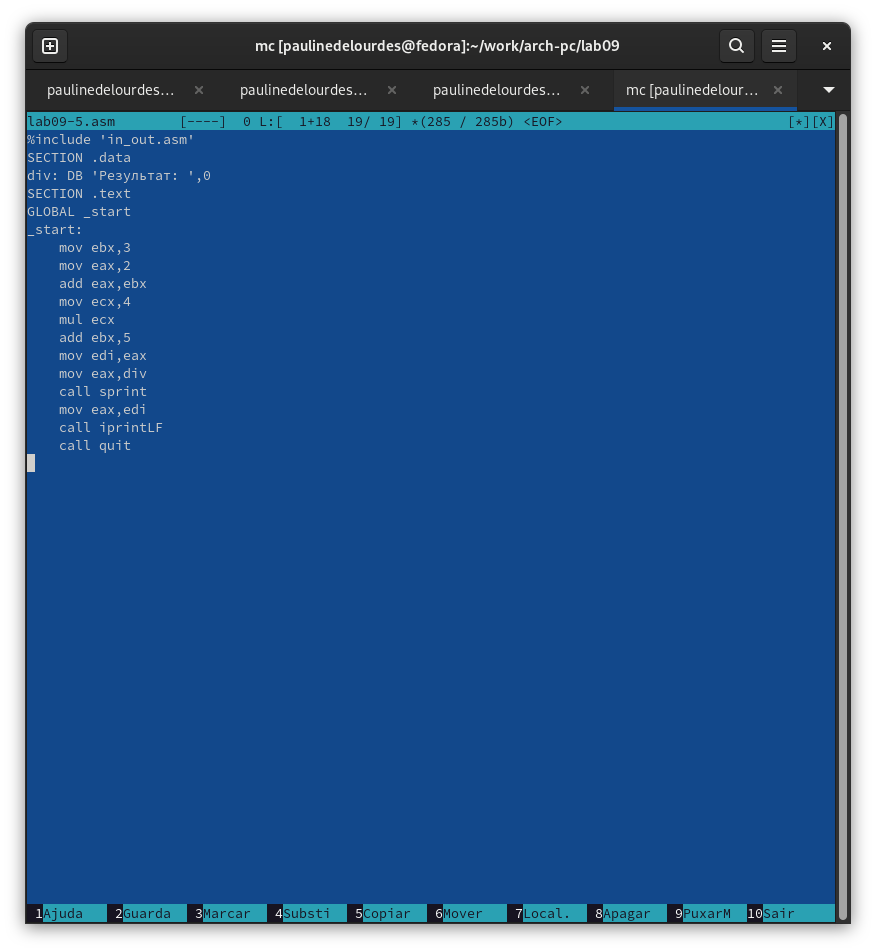


Рис. 32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. fig. 33).

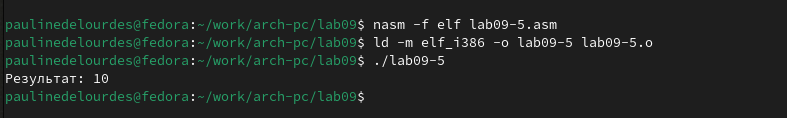


Рис. 33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение решистров командой si (рис. fig. 34).

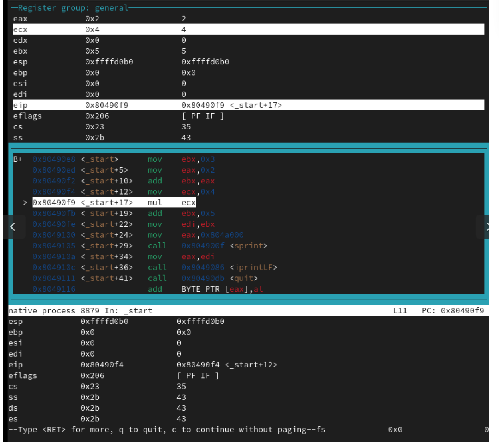


Рис. 34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. fig. 35).

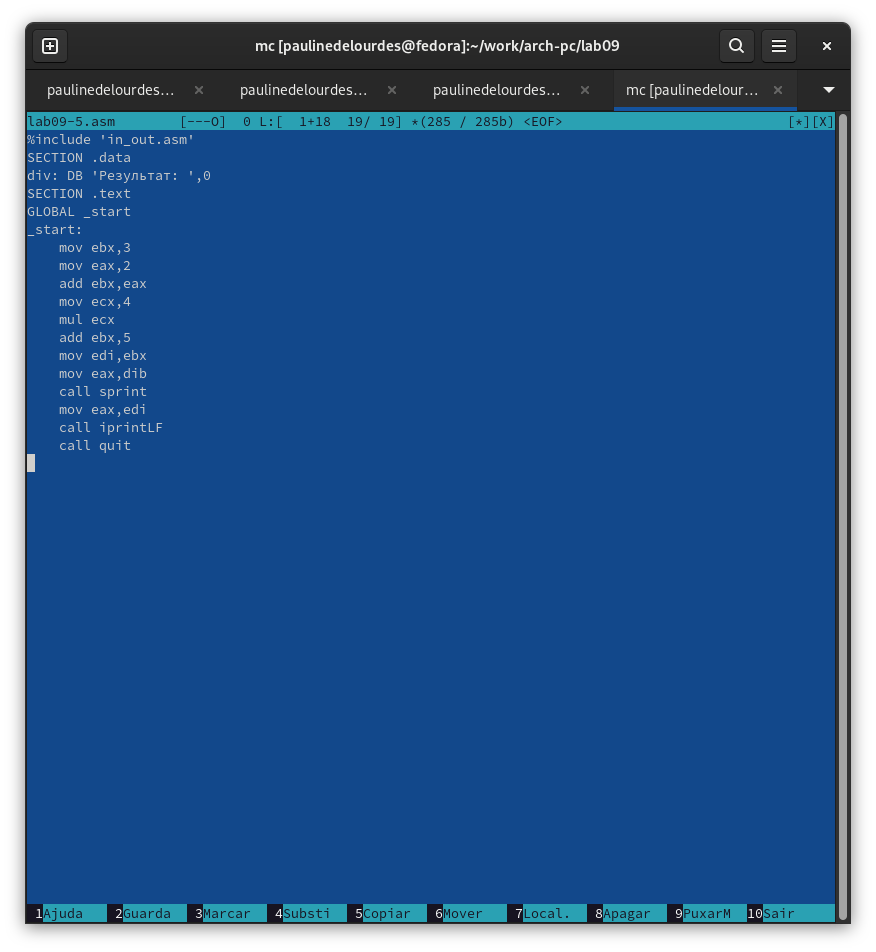


Рис. 35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. fig. 36).

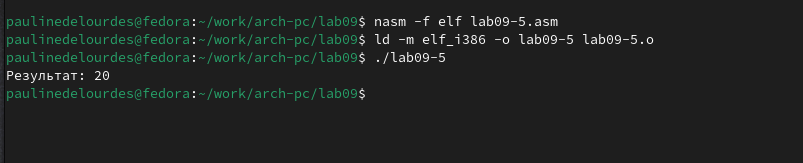


Рис. 36: Создаем и запускаем файл(работает корректно)

# 3 Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.