Лабораторная работа №9

Архитектура компьютера. Понятие подпрограммы. Отладчик GDB

Сафиуллина Айлина Саяровна

Содержание

# 1 Цель работы

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

# 2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

• обнаружение ошибки;

• поиск её местонахождения;

• определение причины ошибки;

• исправление ошибки.

Можно выделить следующие типы ошибок:

• синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;

• семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата;

• ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её поведение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g. Для продолжения остановленной программы используется команда continue (c) (gdb) с [аргумент]. Выполнение программы будет происходить до следующей точки останова. Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создайте файл lab09-1.asm(рис. 1).

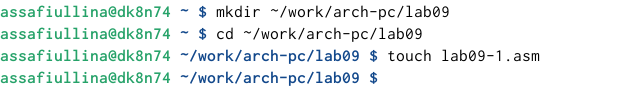


Рис. 1: Каталог lab09

В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме.(рис. 2).

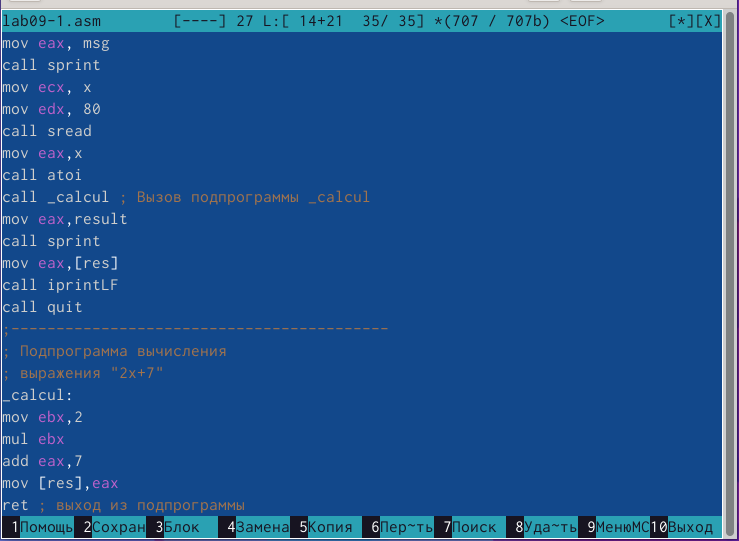


Рис. 2: Ввод текста из листинга 9.1

Создадим исполняемый файл и проверим его работу (рис. 3).

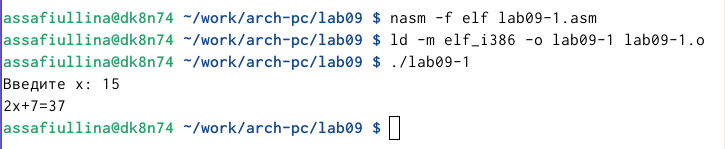


Рис. 3: Запуск исполняемого файла

Изменим текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. Т.е. x передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение g(x), результат возвращается в \_calcul и вычисляется выражение f(g(x)). Результат возвращается в основную программу для вывода результата на экран.(рис. 4).

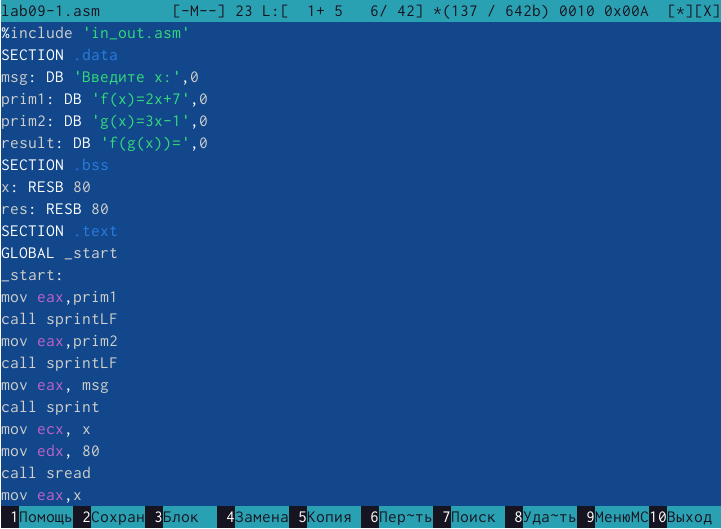


Рис. 4: Изменение текста программы

Создадим исполняемый файл и проверим его работу(рис. 5).

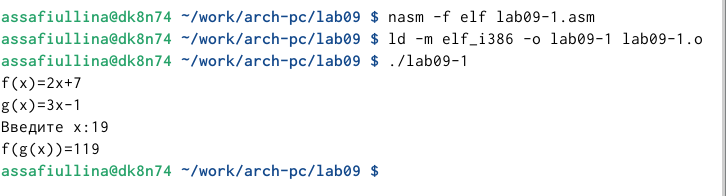


Рис. 5: Запуск исполняемого файла

## 3.2 Отладка программ с помощью GDB

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2.(Программа печати сообщения Hello world!): (рис. 6).

Рис. 6: Создание файла lab09-2.asm

Рис. 6: Создание файла lab09-2.asm

(рис. 7).

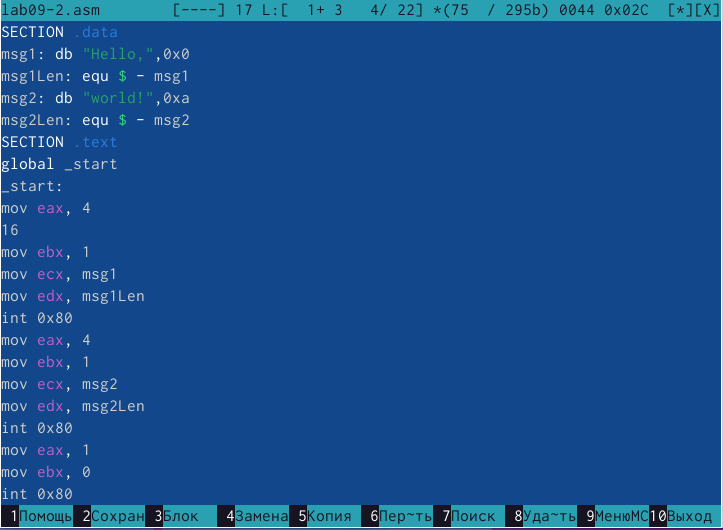


Рис. 7: Ввод текста из листинга 9.2.

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. 8).

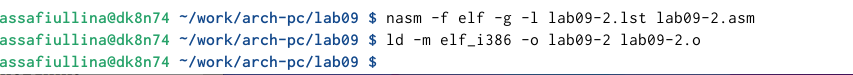


Рис. 8: Получение исполняемый файл

Загрузим исполняемый файл в отладчик gdb (рис. 9).

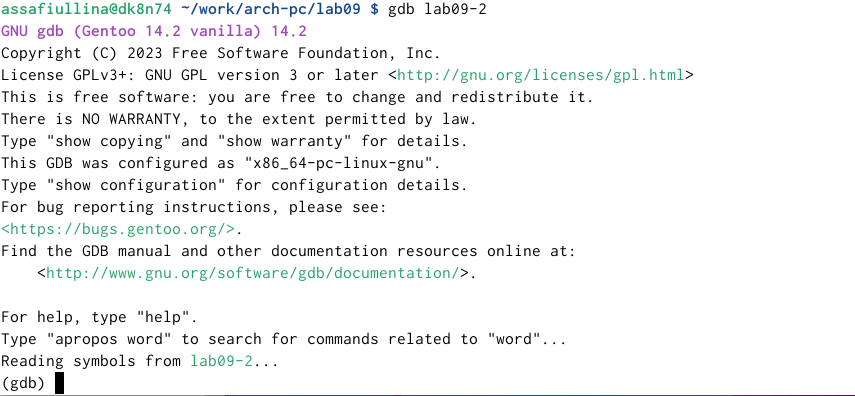


Рис. 9: Отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r)(рис. 10).

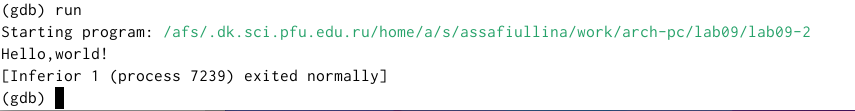


Рис. 10: run

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её(рис. 11).

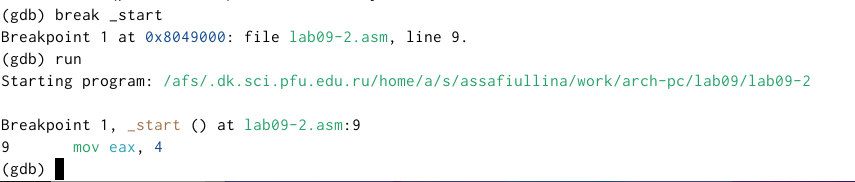


Рис. 11: Установка брейкпоинта

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start(рис. 12).

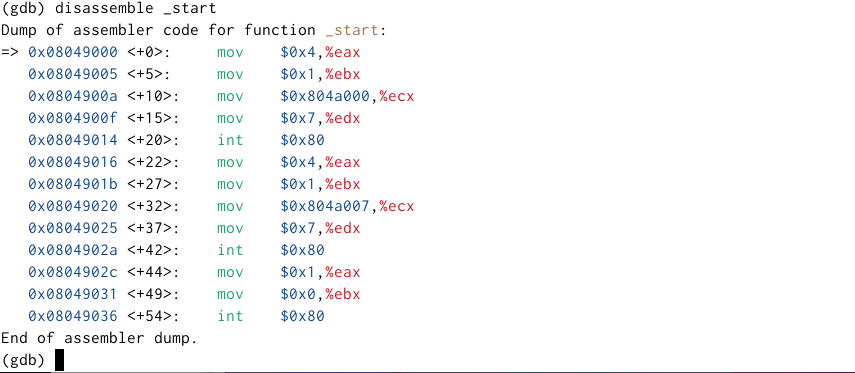


Рис. 12: Дисассимилированный код

Переключимся на отображение команд с Intel’овским синтаксисом,введя команду set disassembly-flavor intel(рис. 13).

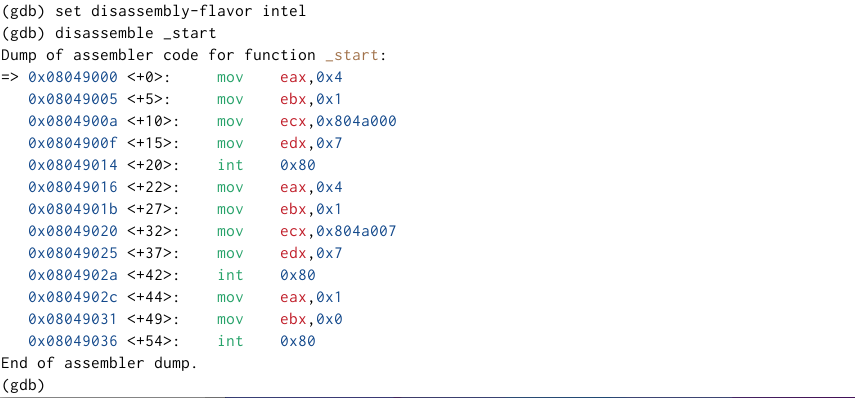


Рис. 13: Intel’овский синтаксис

Различия отображения синтаксиса машинных команд в режимах ATT и Intel заключаются в командах. В диссамилированном отображении в командах используюся “%” и “$” , а в Intel этих символов нет. На это отображение удобнее смотреть. Включим режим псевдографики для более удобного анализа програм- мы с помощью комманд: (gdb) layout asm (gdb) layout regs (рис. 14).

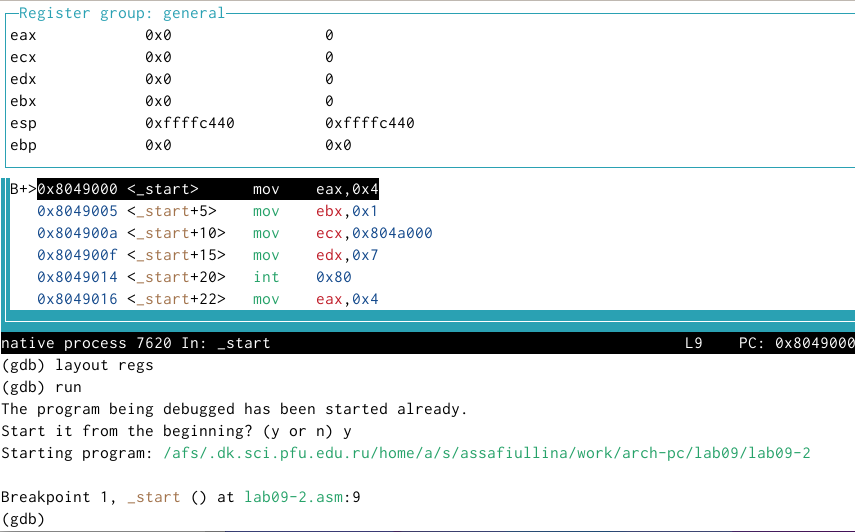


Рис. 14: Режим псевдографики

В этом режиме есть три окна:

• В верхней части видны названия регистров и их текущие значения;

• В средней части виден результат дисассимилирования программы;

• Нижняя часть доступна для ввода команд.

## 3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды info breakpoints (кратко i b)(рис. 15).

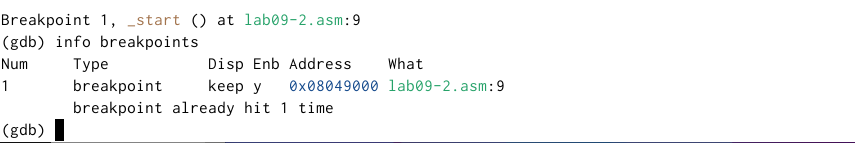


Рис. 15: Информация о точках останова

Установим еще одну точку останова по адресу инструкции (рис. 16).

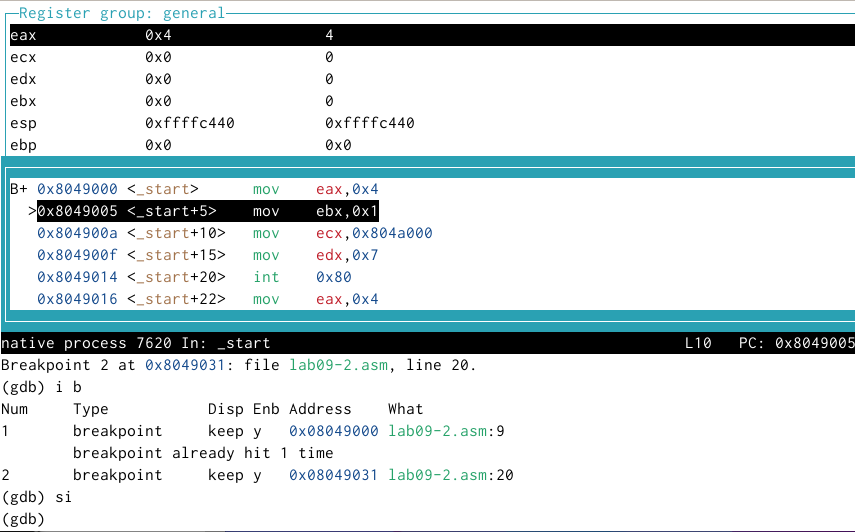


Рис. 16: Установка точки останова

Посмотрим информацию о всех установленных точках останова (рис. 17).

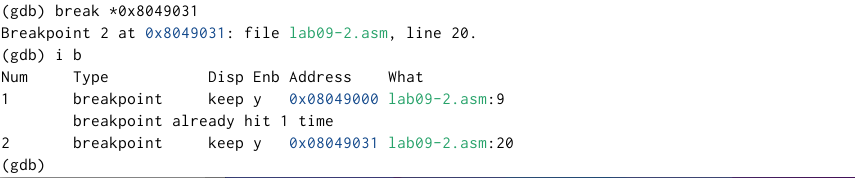


Рис. 17: Информация об установленных точках останова

## 3.4 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.С помощью команды si посмотрим регистры и изменим их.

(рис. 18).

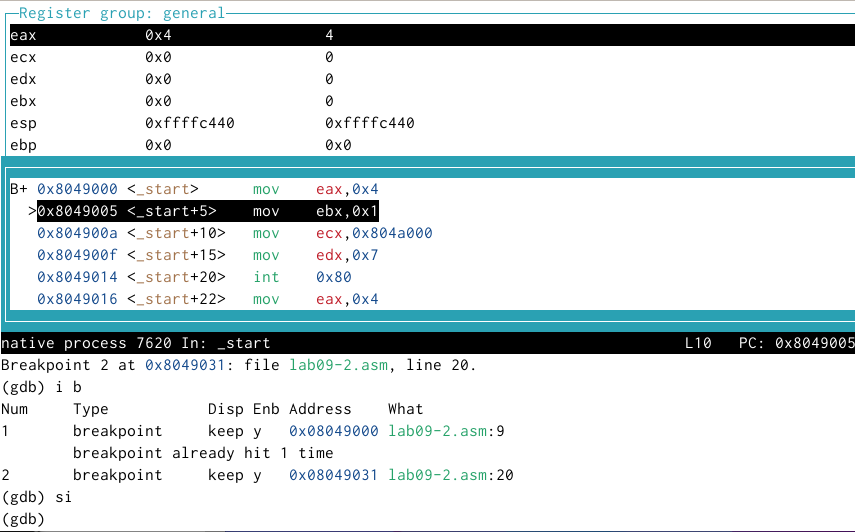


Рис. 18: Команда si

Для отображения содержимого памяти можно использовать команду x, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU . С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени (рис. 19).



Рис. 19: Значение переменной msg1

Посмотрим значение переменной msg2 по адресу (рис. 20).



Рис. 20: Значение переменной msg2

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес.При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных. Изменим первый символ переменной msg1(рис. 21).



Рис. 21: Изменение значения переменной msg1

Изменим символ переменной msg2(рис. 22).



Рис. 22: Изменение значения переменной msg2

С помощью команды set изменим значение регистра ebx(рис. 23).

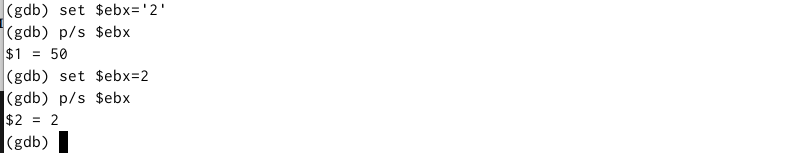


Рис. 23: Изменение значения регистра ebx

Команда выводит два разных значения, потому что в первый раз мы вносим значение 2, а во второй - регистр равен двум, поэтому значения отличаются. Завершим выполнение программы с помощью команды continue (со- кращенно c) или stepi (сокращенно si) и выйдем из GDB с помощью команды quit (сокращенно q) (рис. 24).

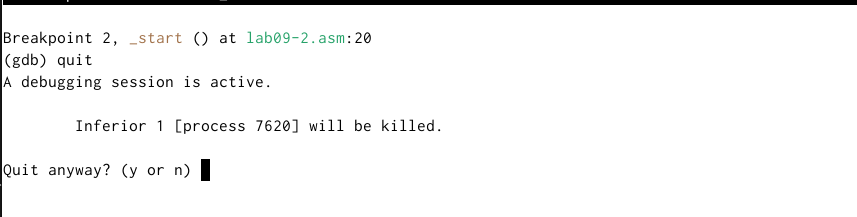


Рис. 24: Завершение и выход из программы

## 3.5 Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm (рис. 25).

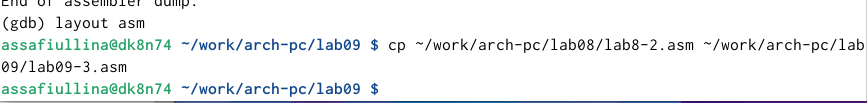


Рис. 25: Копирование файла lab8-2.asm

Создадим исполняемый файл (рис. 26).

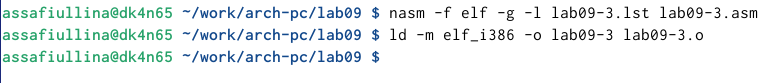


Рис. 26: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использо- вать ключ –args. Загрузим исполняемый файл в отладчик, указав аргументы((рис. 27).

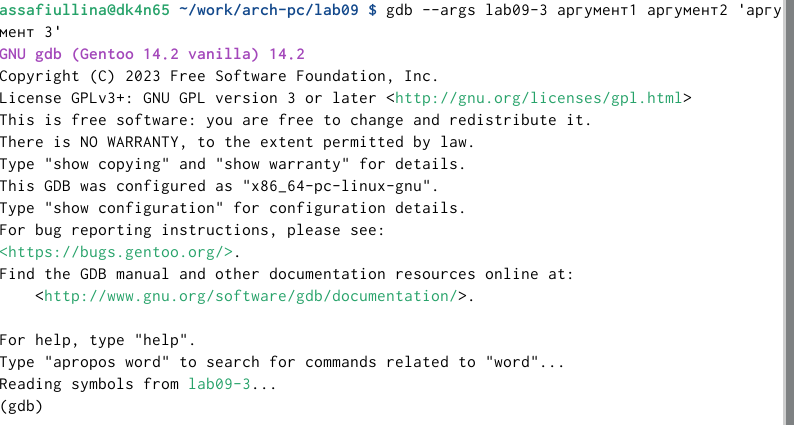


Рис. 27: Загрузка исполняемого файла в отладчик

При запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 28).

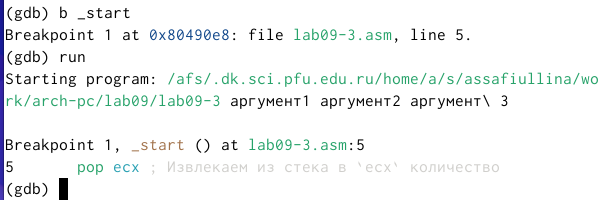


Рис. 28: Точка останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)(рис. 29).



Рис. 29: Адрес вершины стека

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека(рис. 30).

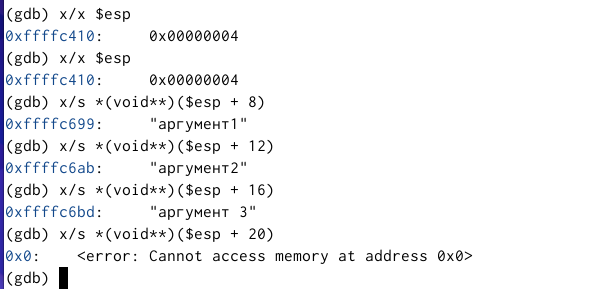


Рис. 30: Позиции стека

Элементы расположены с интервалом в 4 единицы, потому что стек может хранить до 4 байт, и для того, чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

## 3.6 Задание для самостоятельной работы

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы).(рис. 31).

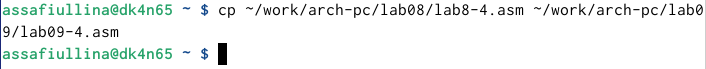


Рис. 31: Файл lab09-4.asm

Реализуем вычисление значения функции f(x) как подпрограмму(рис. 32).

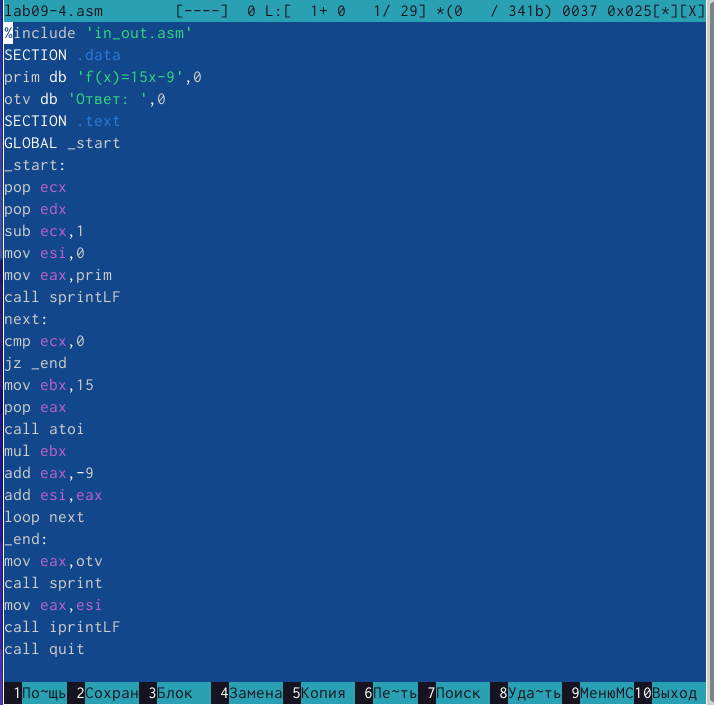


Рис. 32: Измененная программа

Создадим исполняемый файл и запустим его. (рис. 33).

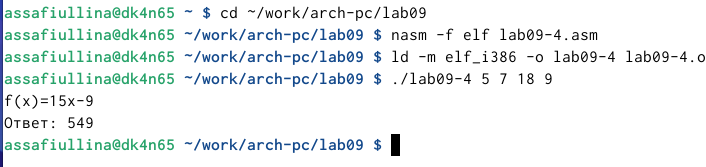


Рис. 33: Запуск исполняемого файла

Создадим файл для решения №2 самостоятельной работы (рис. 34).

Рис. 34: Создание файла lab09-5.asm

Рис. 34: Создание файла lab09-5.asm

В листинге 9.3 приведена программа вычисления выражения (3+2)\*4+5. При запуске данная программа дает неверный результат.(рис. 35).

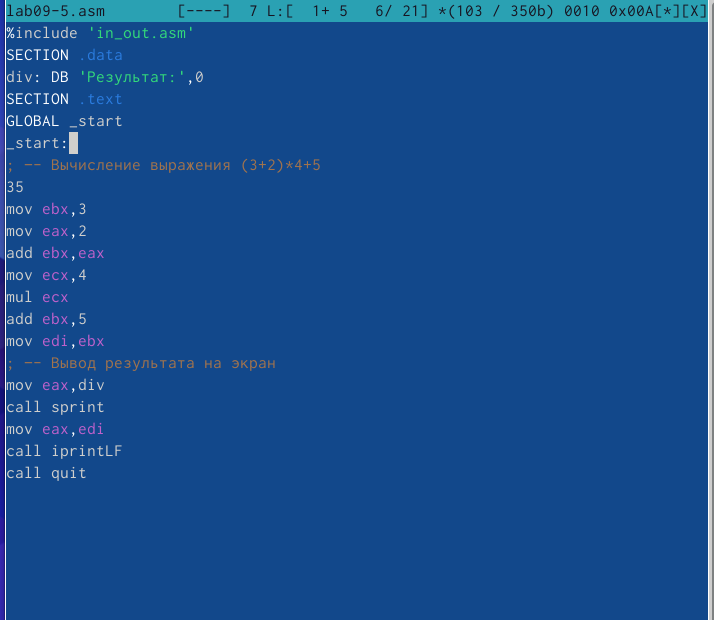


Рис. 35: Ввод текста из листинга 9.3

Проверим, что программа выводит неправильный ответ.(рис. 36)

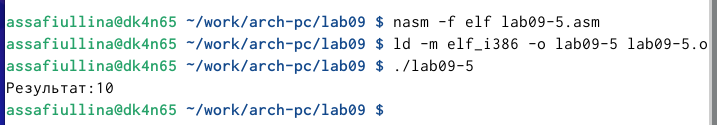


Рис. 36: Неправильный ответ программы

С помощью отладчика GDB запустим программу(рис. 37)

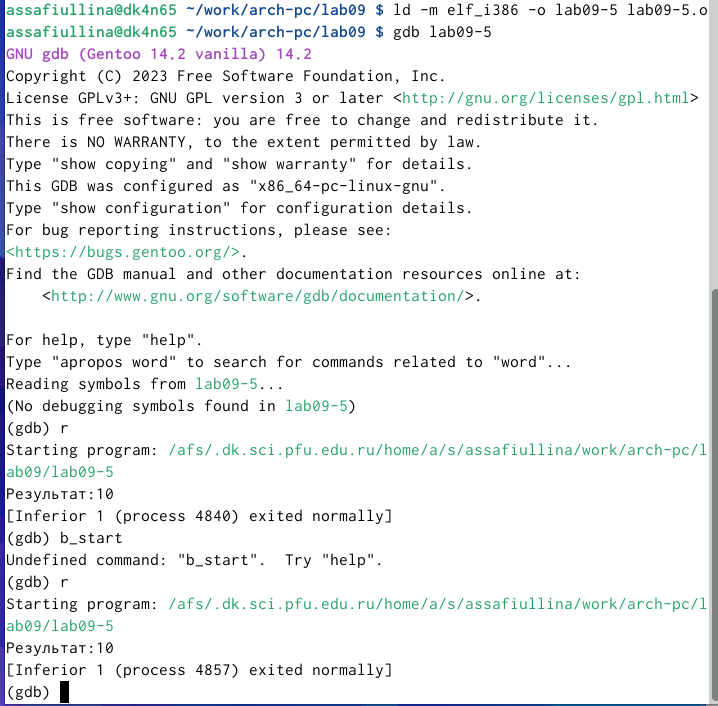


Рис. 37: Запуск программы с помощью отладчика GDB

Проанализировав изменения значений регистров, понятно, что некоторые регистры стоят не на своих местах.(рис. 38)

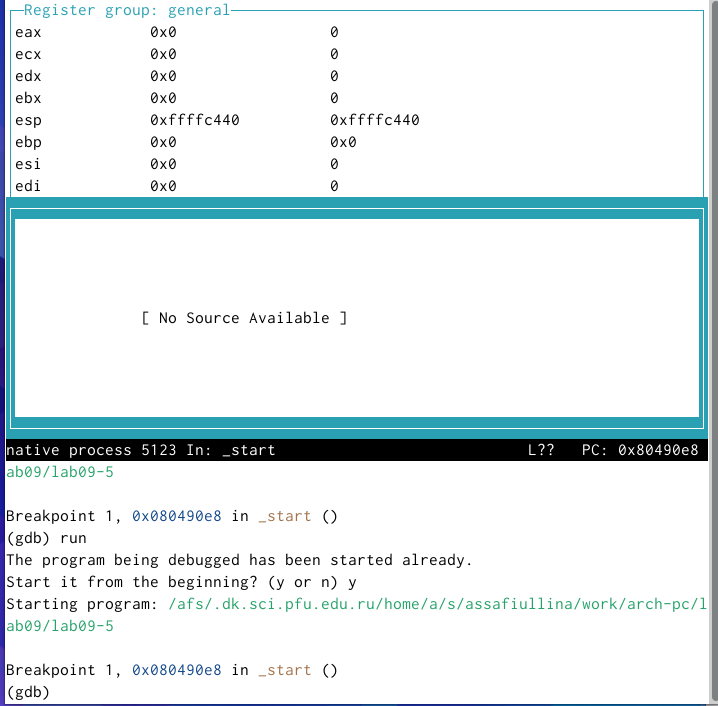


Рис. 38: Анализ регистров

Исправив регистры, запустим программу(рис. 39)

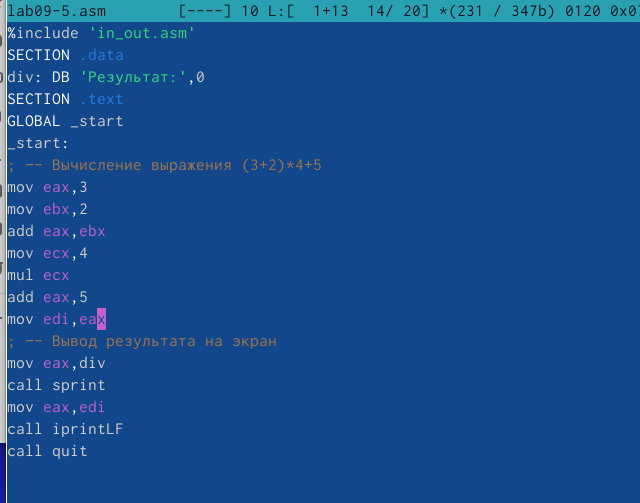
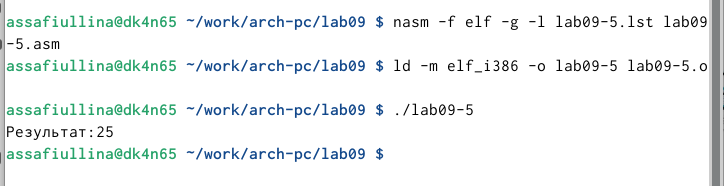


Рис. 39: Запуск исправленной программы

Теперь программа действительно выводит правильный ответ. Программа работает верно.(рис. **¿fig:040?**)

 # Выводы

В процессе выполнения лабораторной работы я получила навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB, его основными возможностями.