Отчёт по лабораторной работе 9

Понятие подпрограммы. Отладчик GDB.

Гадаборшев Заур Закреевич НПИбд-01-23

Содержание

# 1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. 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.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

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

Я создал каталог для выполнения лабораторной работы №9 и перешел в него.

В качестве примера рассмотрим программу, которая вычисляет арифметическое выражение с использованием подпрограммы calcul. В этом примере значение вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

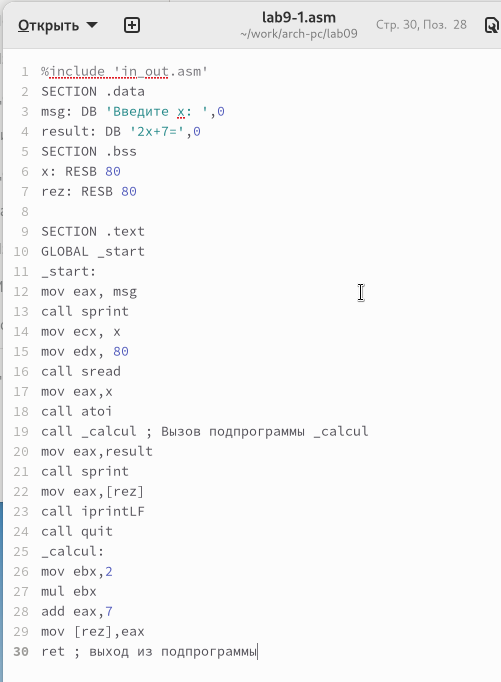


Figure 1: Программа в файле lab9-1.asm

Первые строки программы отвечают за вывод сообщения на экран (с помощью вызова sprint), чтение данных, введенных с клавиатуры (с помощью вызова sread) и преобразование введенных данных из символьного вида в численный (с помощью вызова atoi).

После инструкции call \_calcul, которая передает управление подпрограмме \_calcul, будут выполнены инструкции, содержащиеся в подпрограмме.

Инструкция ret является последней в подпрограмме и ее выполнение приводит к возврату в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму.

Последние строки программы реализуют вывод сообщения (с помощью вызова sprint), вывод результата вычисления (с помощью вызова iprintLF) и завершение программы (с помощью вызова quit).

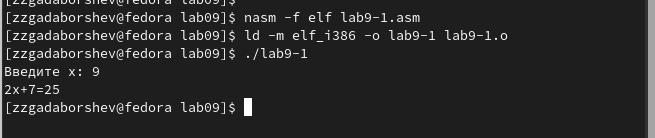


Figure 2: Запуск программы lab9-1.asm

Изменил текст программы, добавив подпрограмму subcalcul в подпрограмму calcul, для вычисления выражения , где вводится с клавиатуры, .

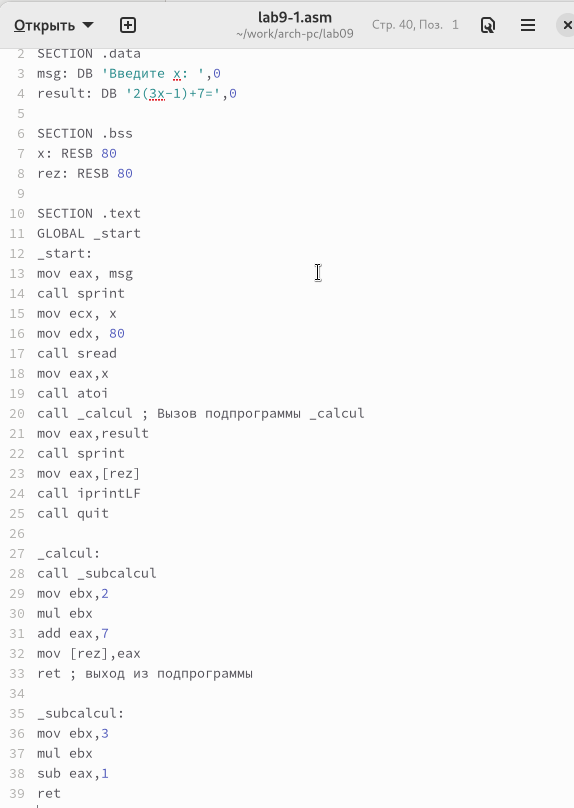


Figure 3: Программа в файле lab9-1.asm

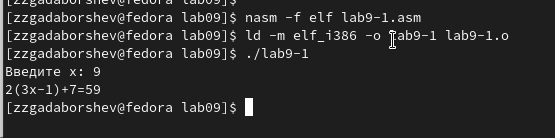


Figure 4: Запуск программы lab9-1.asm

## 3.2 Отладка программам с помощью GDB

Создал файл lab9-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!).

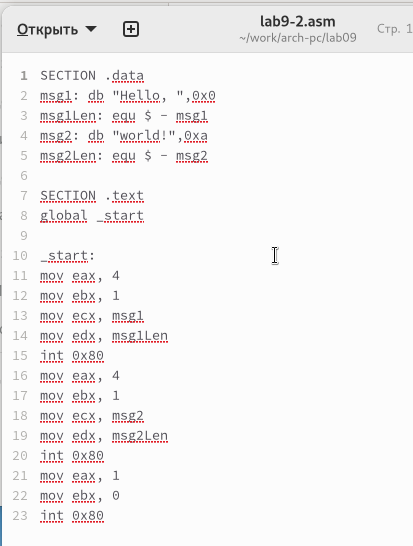


Figure 5: Программа в файле lab9-2.asm

Получил исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

Загрузил исполняемый файл в отладчик gdb. Проверил работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r).

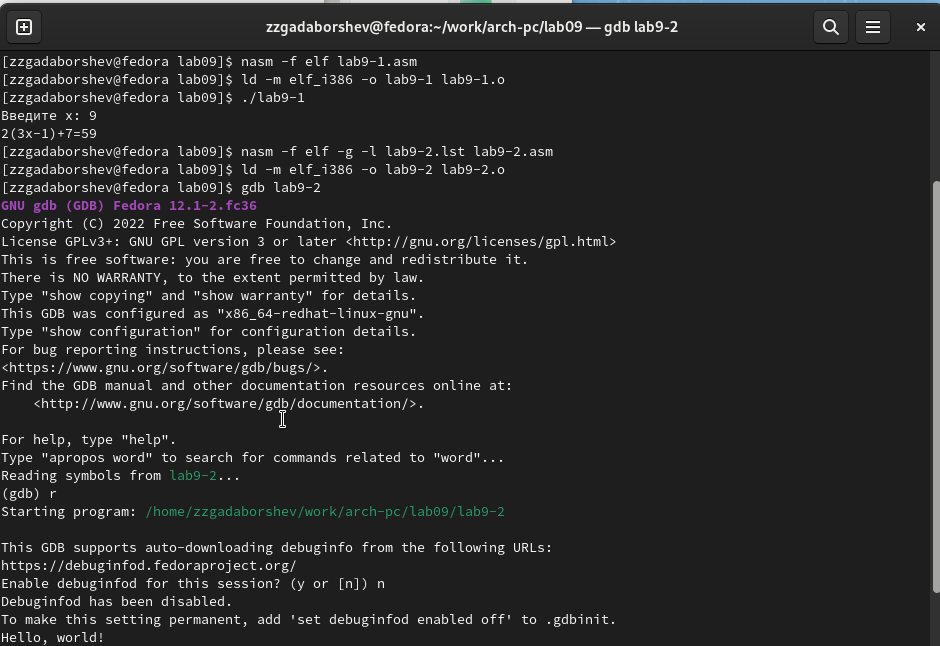


Figure 6: Запуск программы lab9-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы.

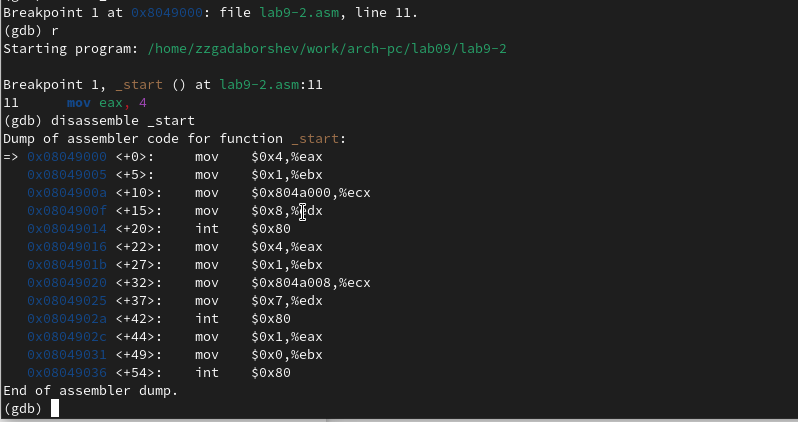


Figure 7: Дизассимилированный код

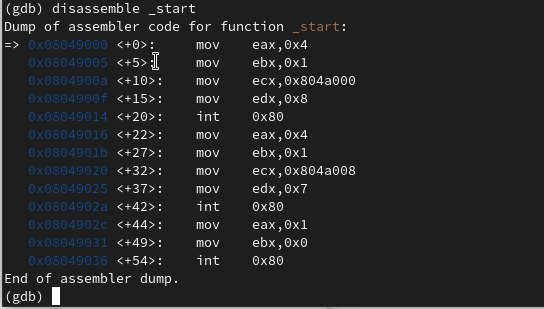


Figure 8: Дизассимилированный код в режиме интел

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»

На предыдущих шагах была установлена точка остановки по имени метки (\_start). Проверил это с помощью команды info breakpoints (кратко i b). Установил еще одну точку остановки по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определил адрес предпоследней инструкции (mov ebx,0x0) и установил точку.

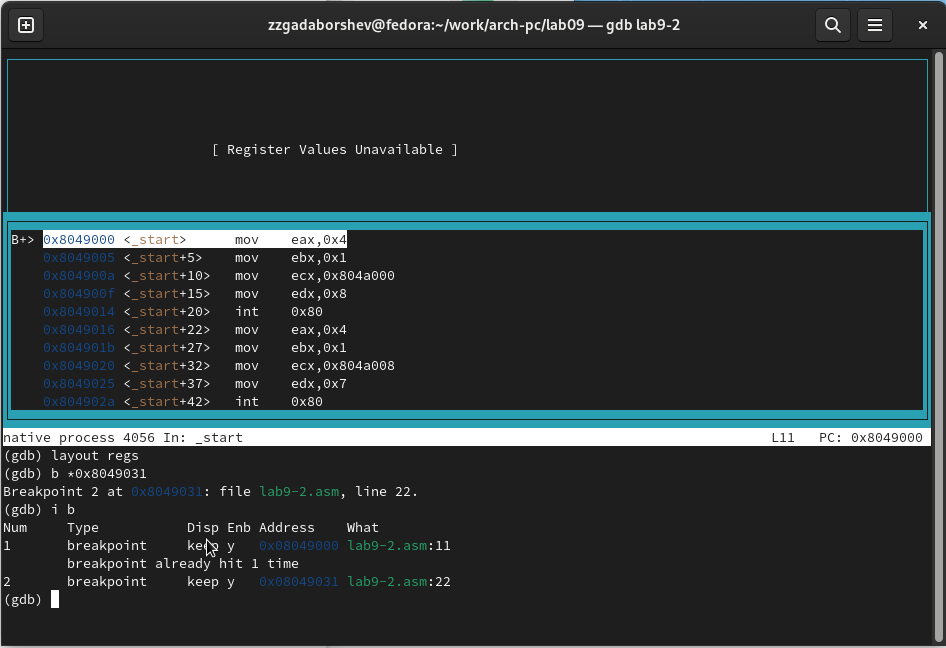


Figure 9: Точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнил 5 инструкций с помощью команды stepi (или si) и проследил за изменением значений регистров.

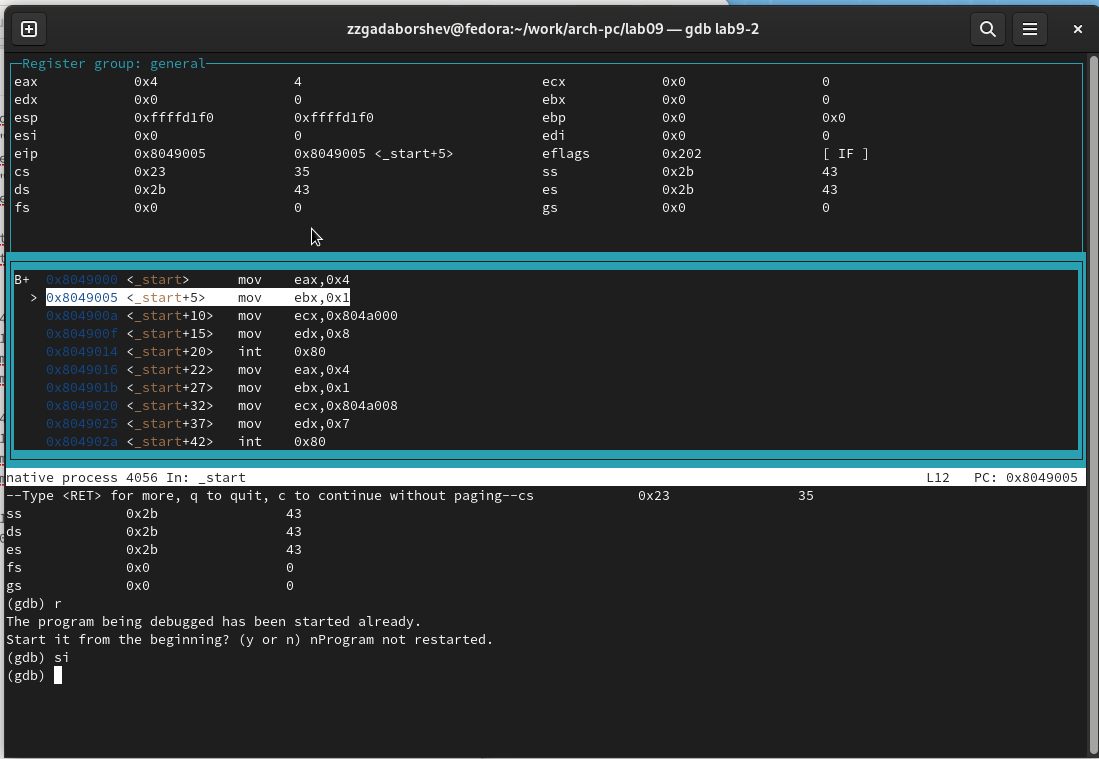


Figure 10: Изменение регистров

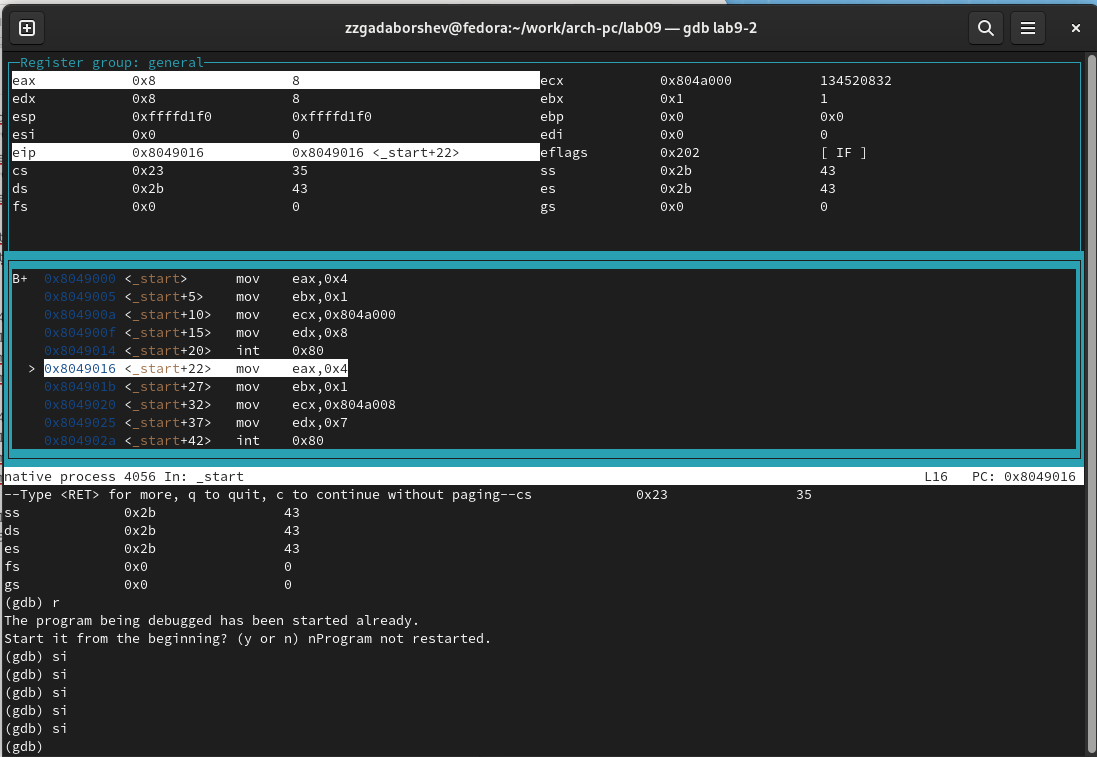


Figure 11: Изменение регистров

Посмотрел значение переменной msg1 по имени. Посмотрел значение переменной msg2 по адресу.

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Изменил первый символ переменной msg1.

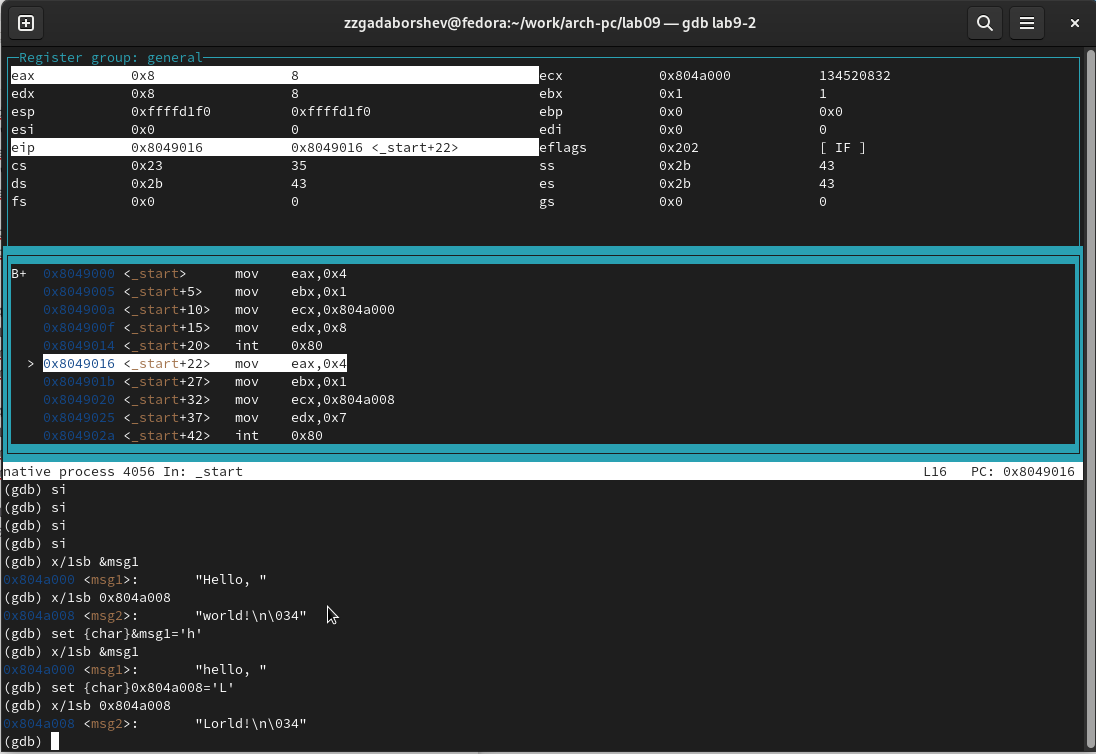


Figure 12: Изменение значения переменной

Вывел в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

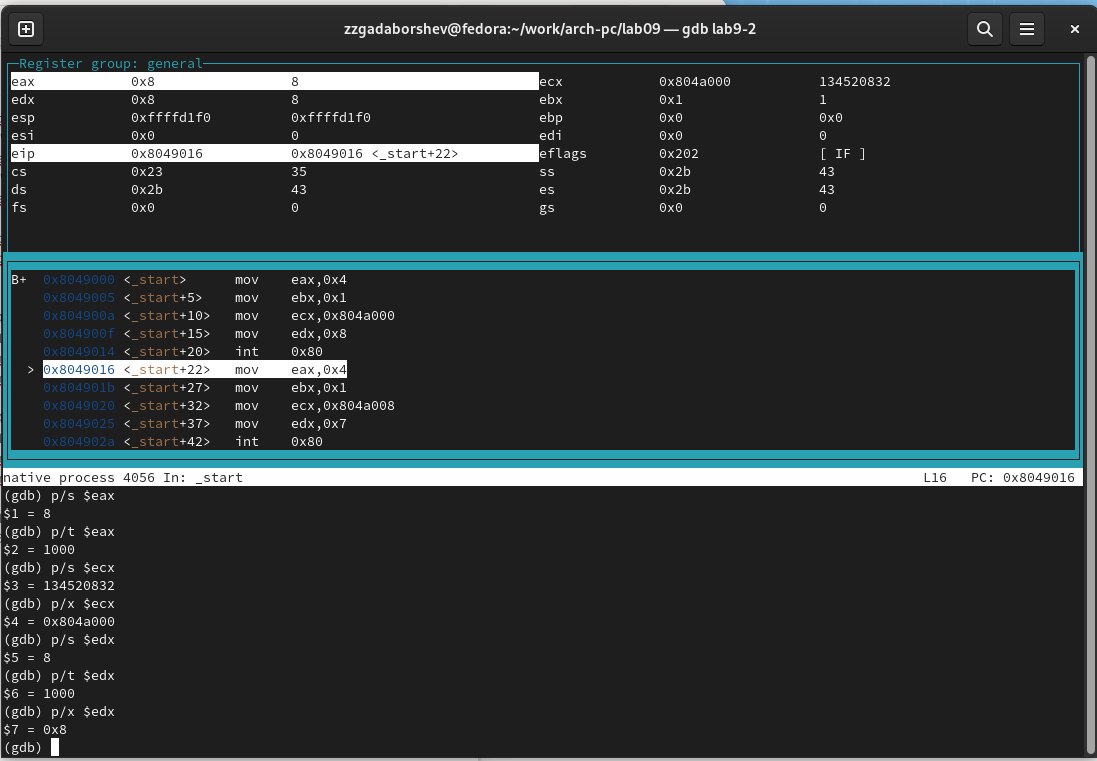


Figure 13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx

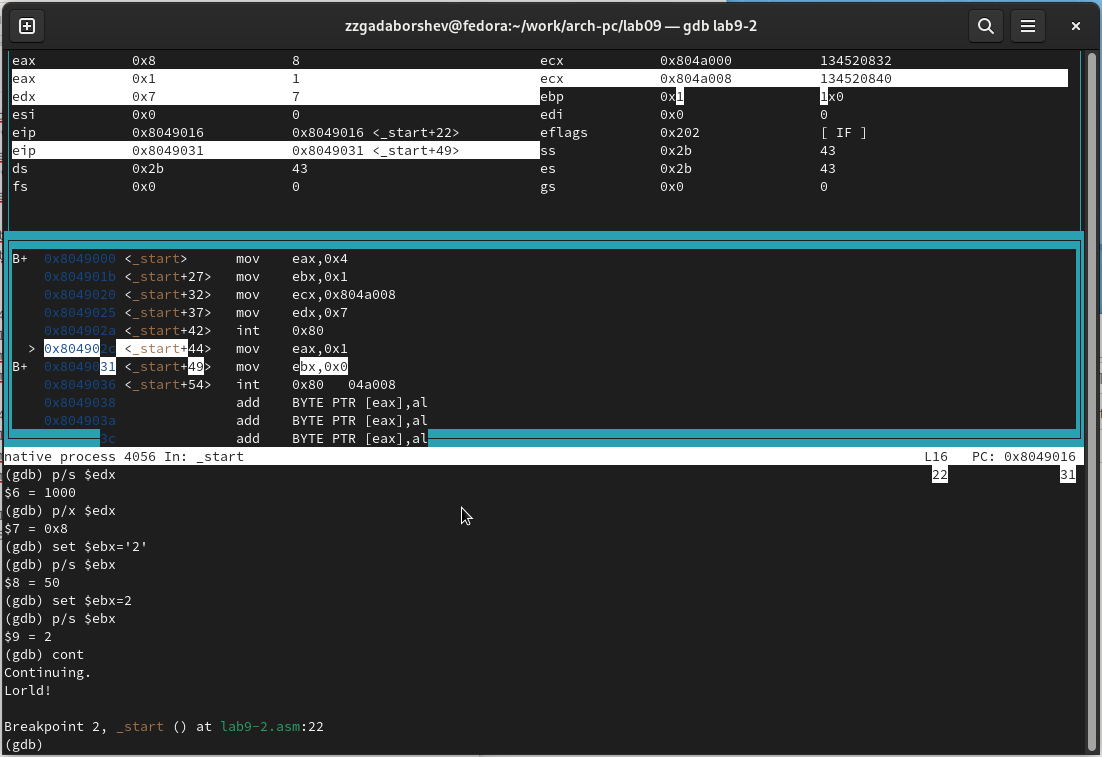


Figure 14: Вывод значения регистра

Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки. Создал исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузил исполняемый файл в отладчик, указав аргументы.

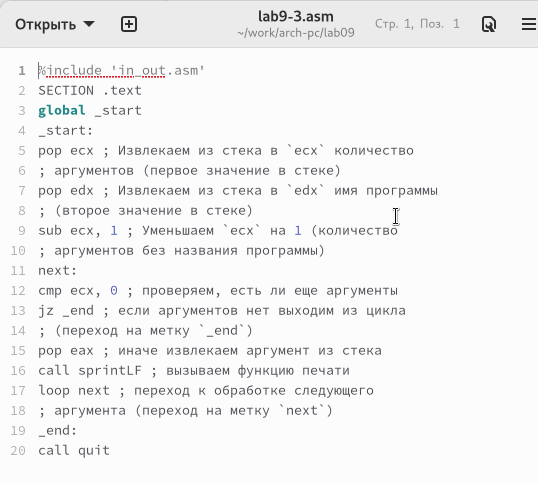


Figure 15: Программа в файле lab9-3.asm

Для начала установил точку останова перед первой инструкцией в программе и запустил ее.

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрел 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.

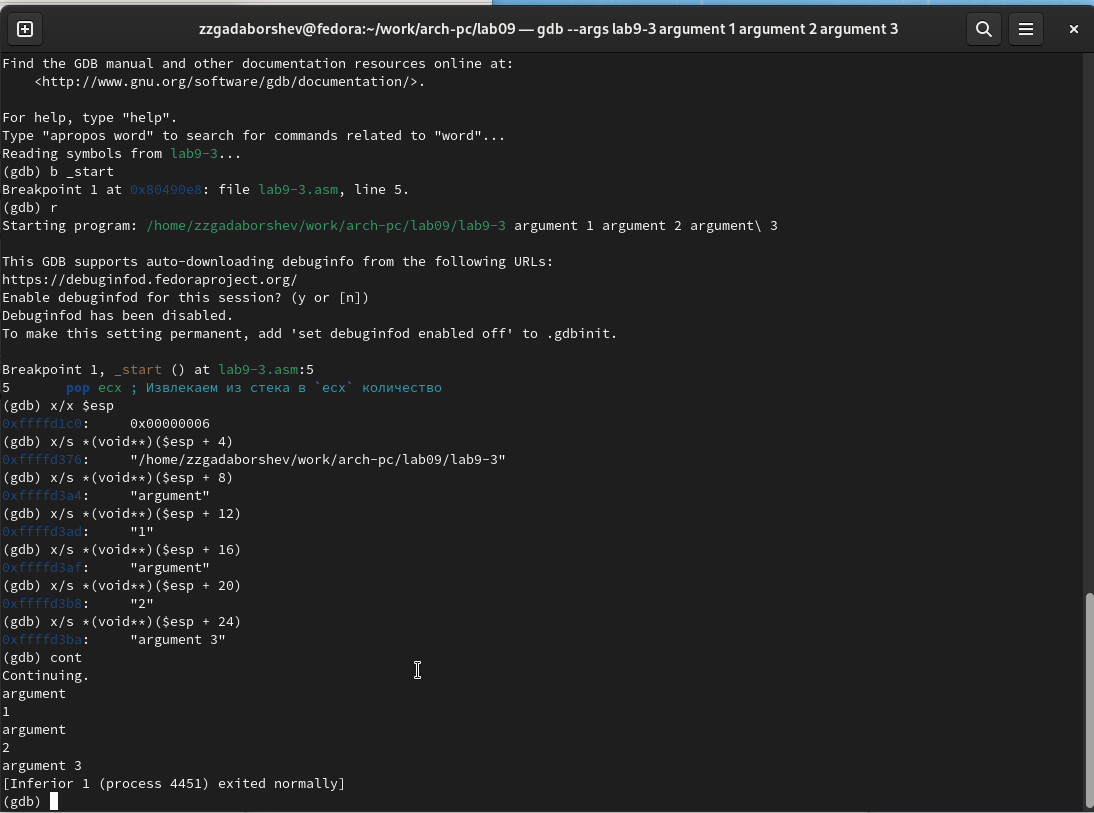


Figure 16: Вывод значения регистра

Объясню, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] - шаг равен размеру переменной - 4 байтам.

## 3.3 Задание для самостоятельной работы

Я переписал программу из лабораторной работы №8, чтобы вычислить значение функции f(x) в виде подпрограммы.

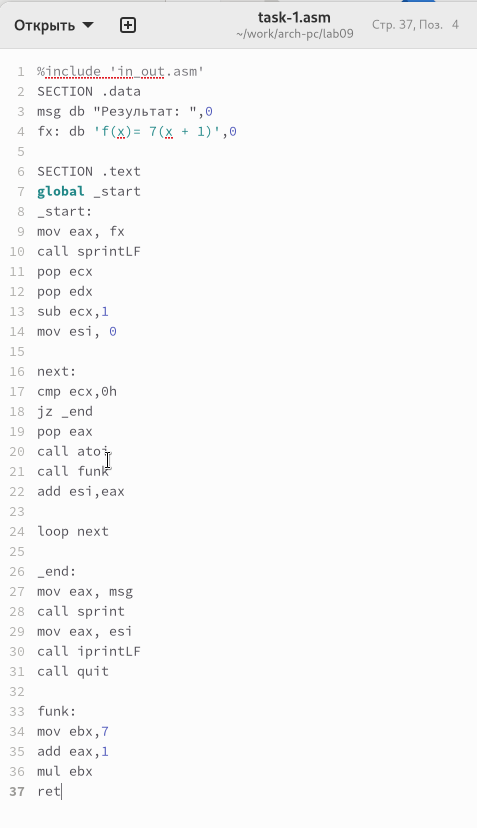


Figure 17: Программа в файле task-1.asm

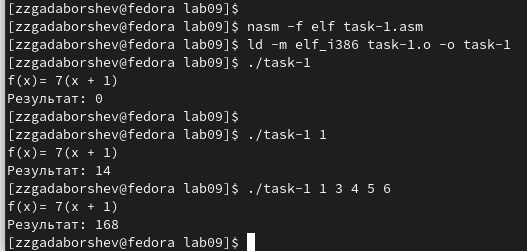


Figure 18: Запуск программы task-1.asm

Приведенный ниже листинг программы вычисляет выражение . Однако, при запуске, программа дает неверный результат. Я проверил это и решил использовать отладчик GDB для анализа изменений значений регистров и определения ошибки.

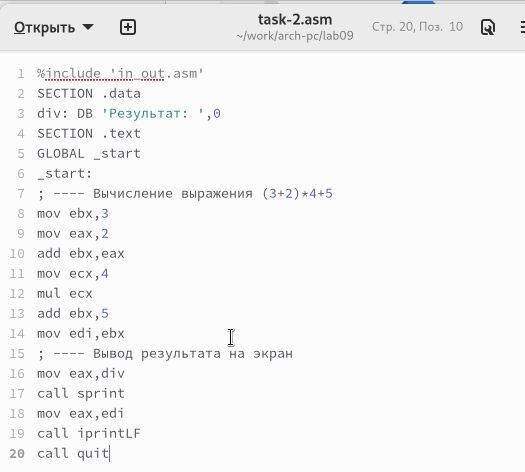


Figure 19: Код с ошибкой в файле task-2.asm

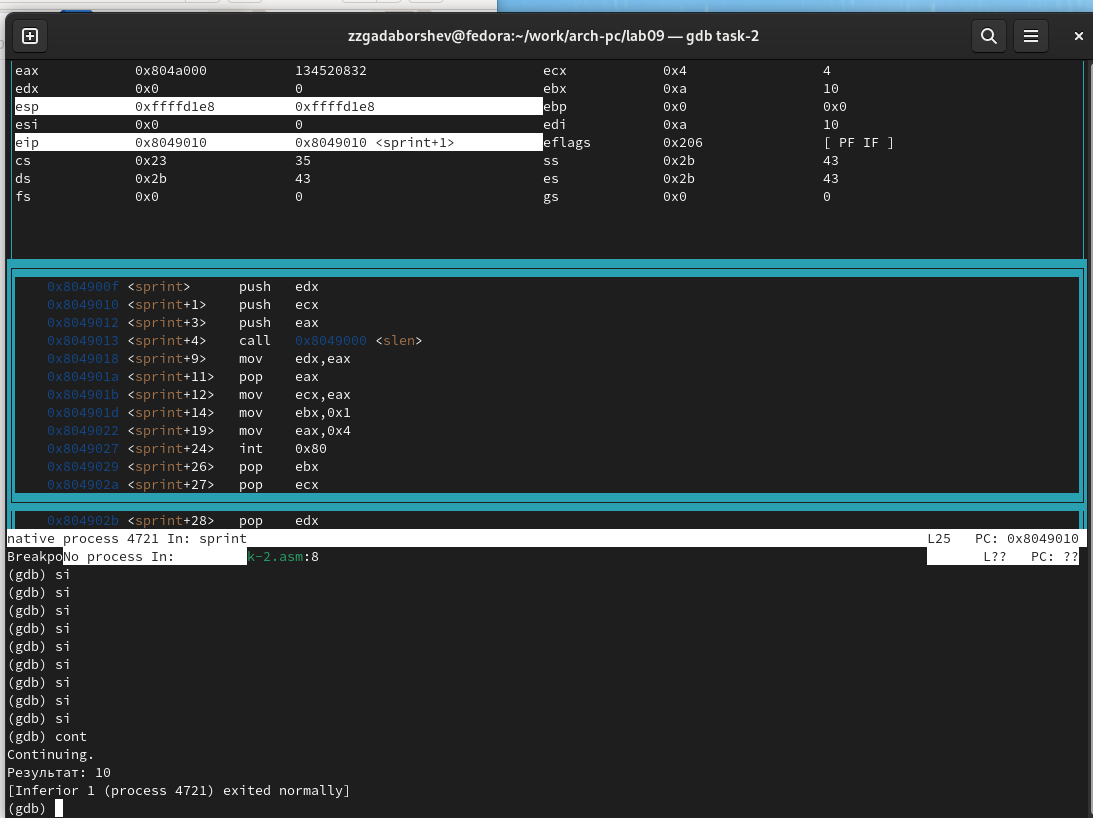


Figure 20: Отладка task-2.asm

Я заметил, что порядок аргументов в инструкции add был перепутан и что при завершении работы, вместо eax, значение отправлялось в edi. Вот исправленный код программы:

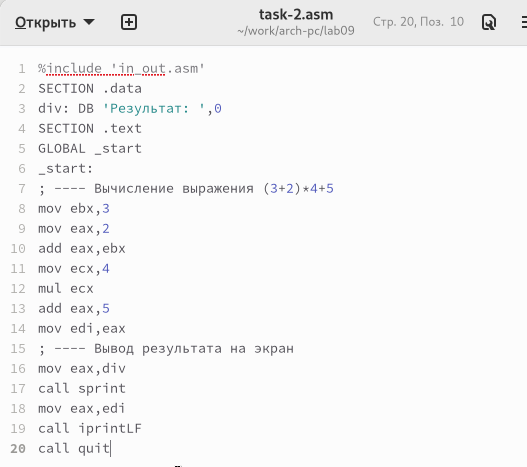


Figure 21: Код исправлен в файле task-2.asm

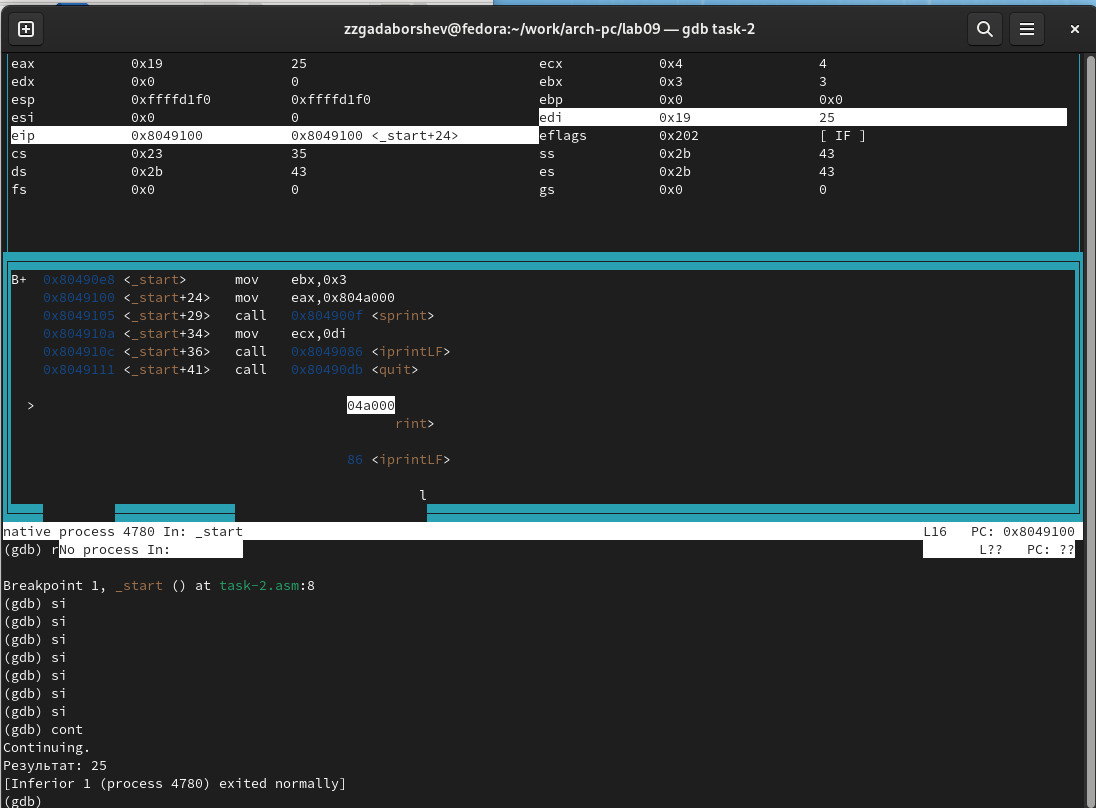


Figure 22: Проверка работы task-2.asm

# 4 Выводы

Освоили работy с подпрограммами и отладчиком.