ОТЧЕТ по лабораторной работе №9

дисциплина: Архитектура компьютера

Студент: Идрисов Д.А.

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 Задание

1. Изучение подпрограмм в ассемблере
2. Изучение отладчика GDB
3. Изучение примеров программ и процесса отладки
4. Выполнение заданий для самостоятельной работы

# 3 Теоретическое введение

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

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

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

В рамках примера рассмотрим программу, которая вычисляет арифметическое выражение с использованием подпрограммы calcul. В данном примере значение переменной вводится с клавиатуры, а само выражение вычисляется внутри подпрограммы.

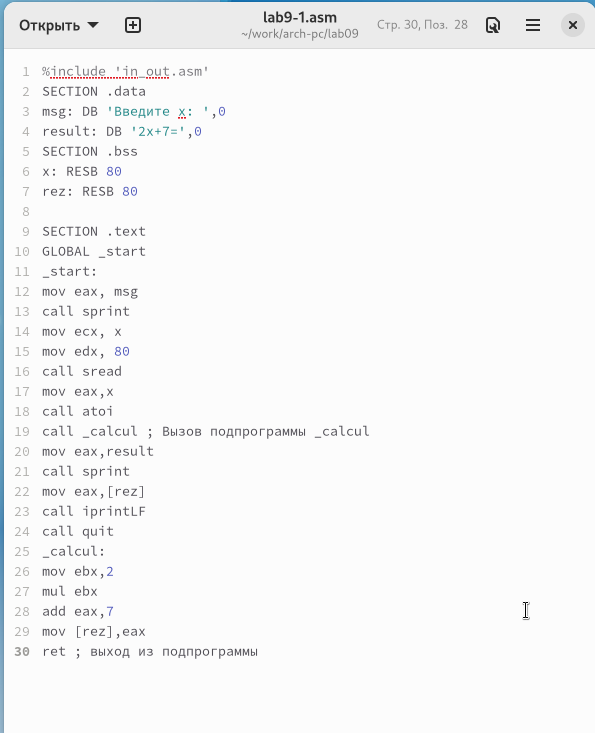


Figure 1: Изменение кода

Первые строки программы отвечают за вывод сообщения на экран с использованием функции sprint, чтение данных, введенных с клавиатуры с помощью функции sread, и преобразование введенных данных из символьного в числовой формат с помощью функции atoi.

После инструкции call \_calcul, которая передает управление подпрограмме \_calcul, выполнение программы переходит к инструкциям, содержащимся внутри подпрограммы.

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.

Внесены изменения в текст программы, добавлена подпрограмма subcalcul внутри подпрограммы calcul для вычисления выражения , где значение вводится с клавиатуры, , .

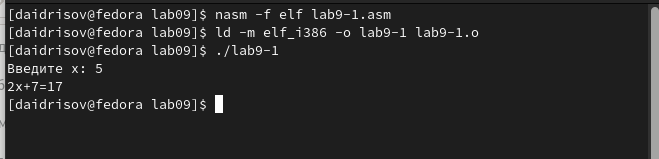


Figure 2: Запуск программы

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

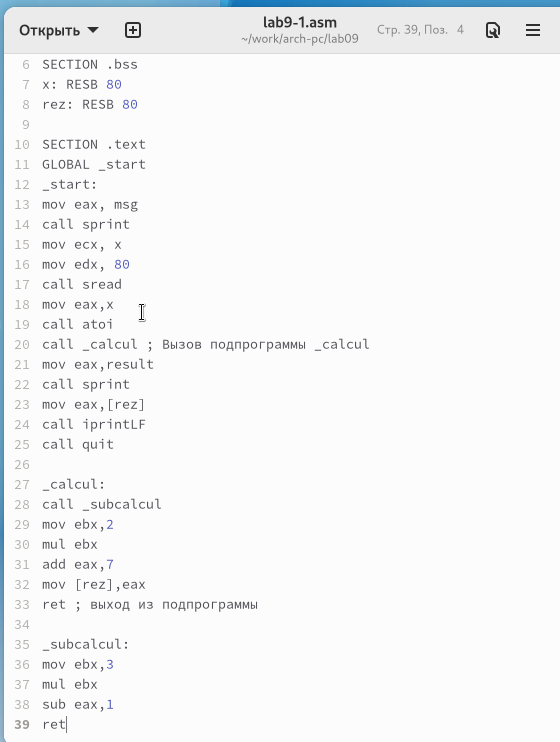


Figure 3: Изменение кода

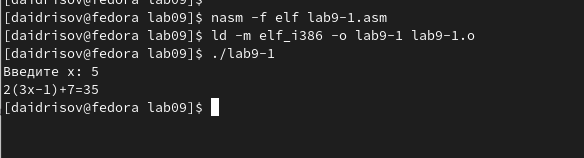


Figure 4: Запуск программы

## 4.2 Отладка программам с помощью GDB

Я создал файл с именем lab9-2.asm, в котором содержится текст программы из Листинга 9.2, реализующей функцию печати сообщения “Hello world!”.



Figure 5: Изменение кода

После компиляции получил исполняемый файл. Чтобы использовать отладчик GDB, я добавил отладочную информацию к исполняемому файлу, указав ключ “-g” при компиляции.

Затем я загрузил исполняемый файл в отладчик GDB и проверил его работу, запустив программу с помощью команды “run” (или “r” в сокращенной форме).

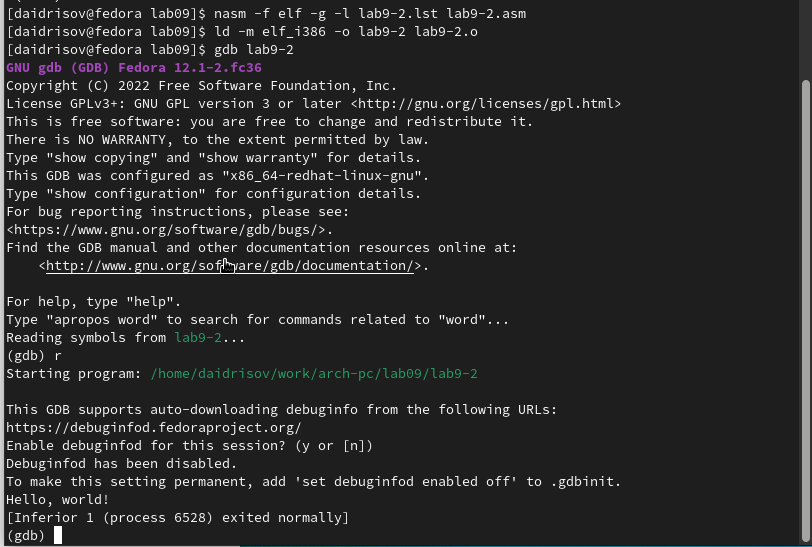


Figure 6: Запуск программы в отладчике

Для более детального анализа программы я установил точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустил ее. Затем я просмотрел дизассемблированный код программы.

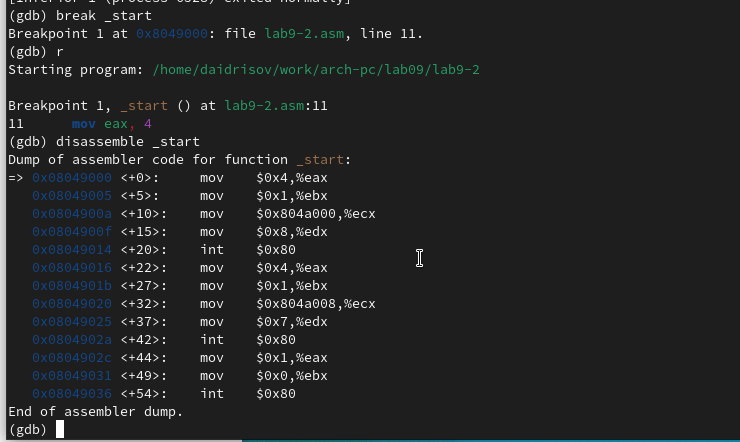


Figure 7: Дизассимилированный код

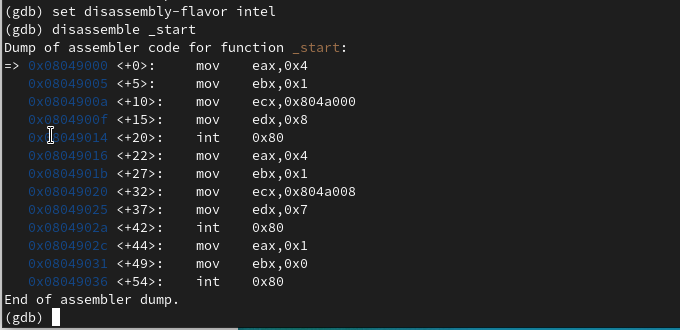


Figure 8: Дизассимилированный код в режиме интел

Чтобы установить точку остановки, я использовал команду “break” (или “b” в сокращенной форме). Типичным аргументом для этой команды может быть номер строки программы, имя метки или адрес. Чтобы избежать путаницы с номерами, перед адресом ставится знак “\*“.

На предыдущих шагах я уже установил точку остановки по имени метки “\_start” и проверил это с помощью команды “info breakpoints” (или “i b” в сокращенной форме). Затем я установил еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0”.

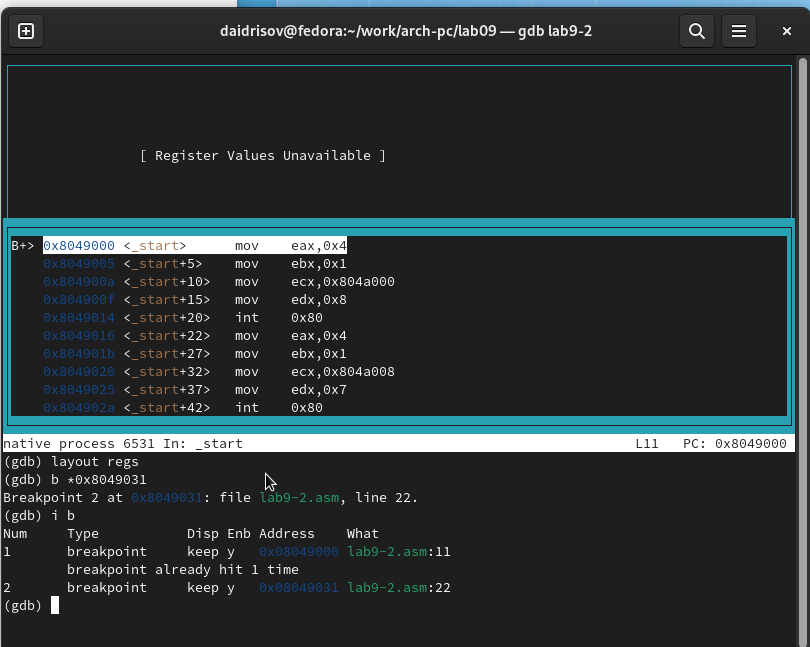


Figure 9: Точка остановки

Я использовал отладчик, который позволяет просматривать содержимое ячеек памяти и регистров, а также вносить в них изменения при необходимости. Я выполнил 5 инструкций с помощью команды stepi (или si) и следил за изменениями значений регистров.

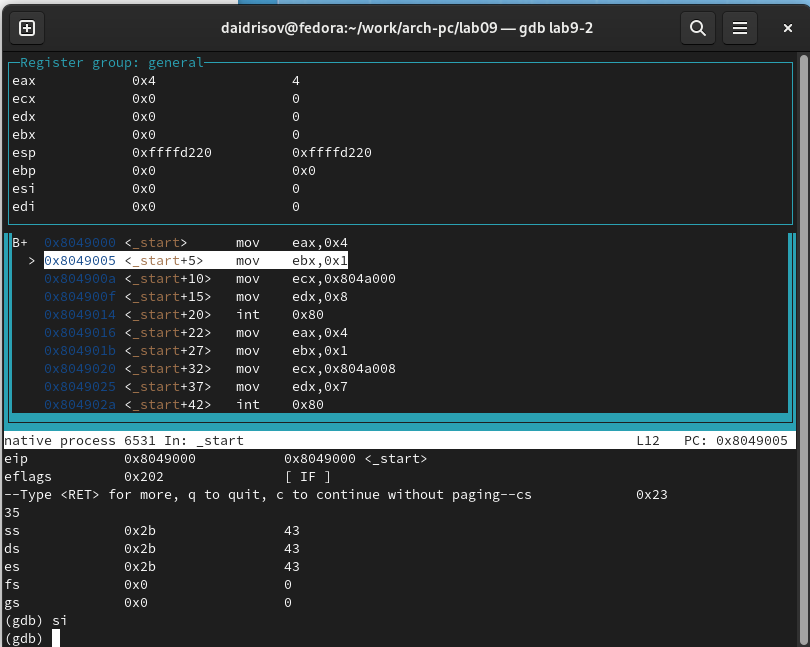


Figure 10: Изменение регистров

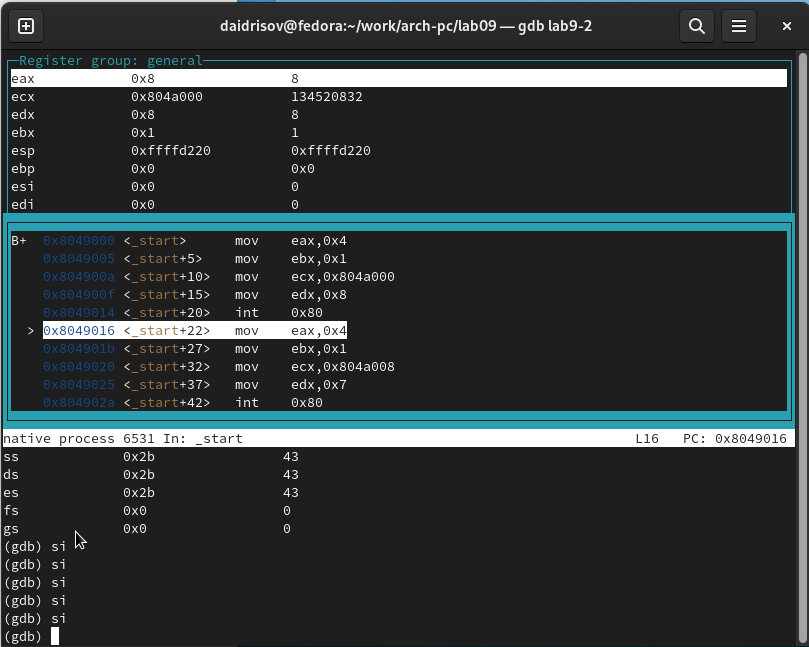


Figure 11: Изменение регистров

Далее я просмотрел значение переменной msg1, обратившись к ней по имени. Также я посмотрел значение переменной msg2, обратившись к ней по адресу.

Для изменения значения регистра или ячейки памяти я использовал команду set, указав в качестве аргумента имя регистра или адрес. Я изменил первый символ переменной msg1.

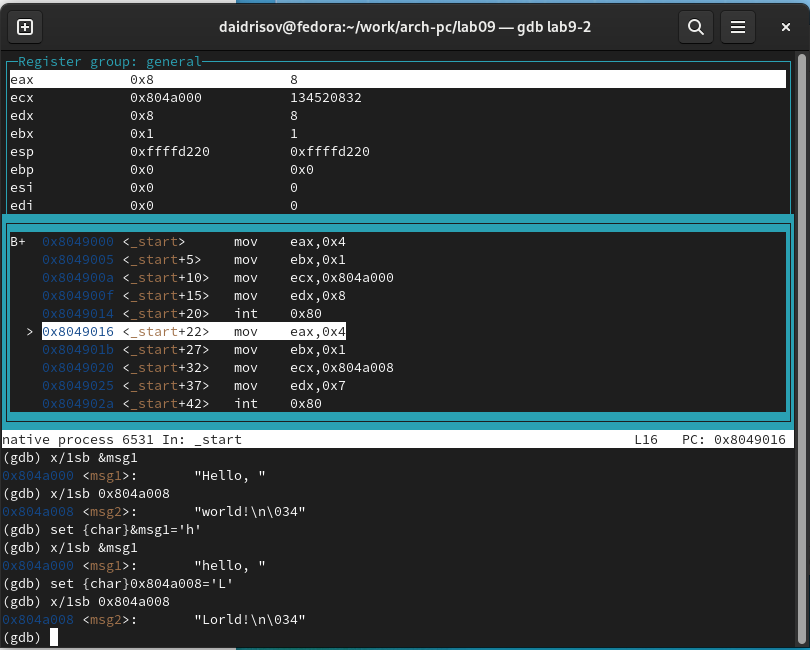


Figure 12: Изменение значения переменной

Для вывода значения регистра edx в различных форматах (шестнадцатеричном, двоичном и символьном) я использовал соответствующие команды.

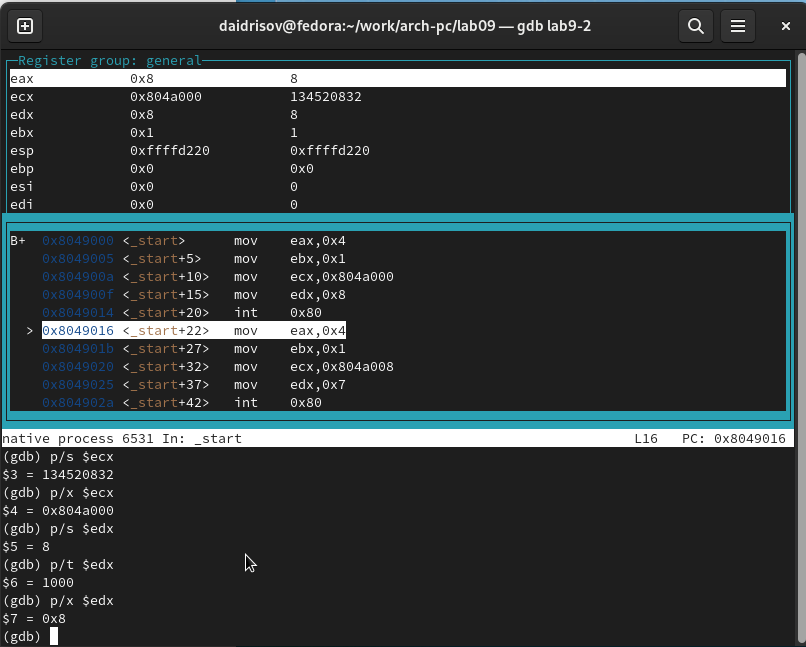


Figure 13: Вывод значения регистра

С помощью команды set также изменил значение регистра ebx.

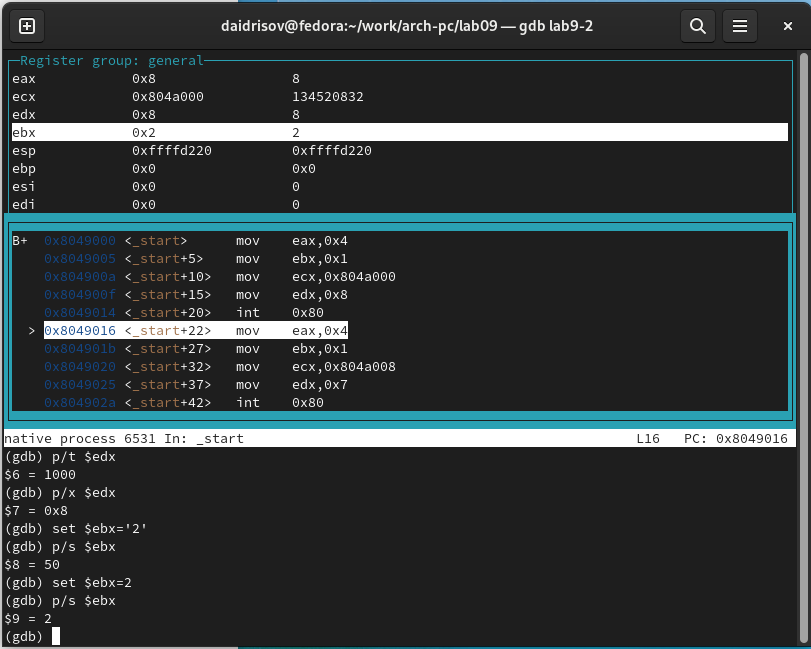


Figure 14: Вывод значения регистра

Я скопировал файл lab8-2.asm, созданный в ходе выполнения лабораторной работы №8, который выводит аргументы командной строки, и создал исполняемый файл. Для загрузки программы с аргументами в отладчик GDB использовал ключ –args, указав соответствующие аргументы. Затем установил точку остановки перед первой инструкцией в программе и запустил ее.

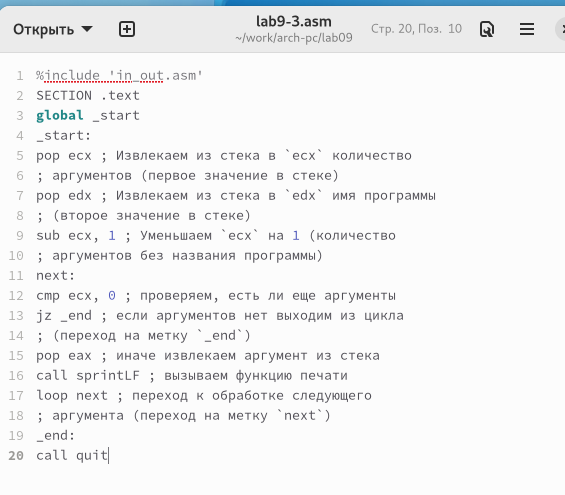


Figure 15: Изменение кода

Адрес вершины стека хранится в регистре esp, и по этому адресу располагается число, равное количеству аргументов командной строки, включая имя программы. В данном случае число аргументов равно 5: имя программы lab9-3 и аргументы: аргумент1, аргумент2 и ‘аргумент 3’.

Я также просмотрел остальные позиции стека. По адресу [esp+4] находится адрес в памяти, где хранится имя программы, по адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] - второго и так далее.

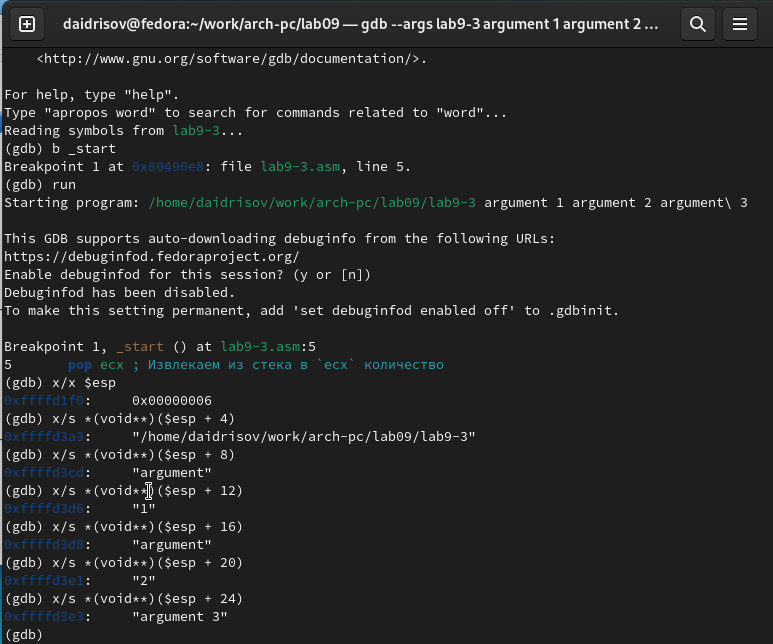


Figure 16: Вывод значения регистра

Объясню, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12]). Это связано с тем, что шаг равен размеру переменной, который составляет 4 байта.

## 4.3 Задание для самостоятельной работы

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

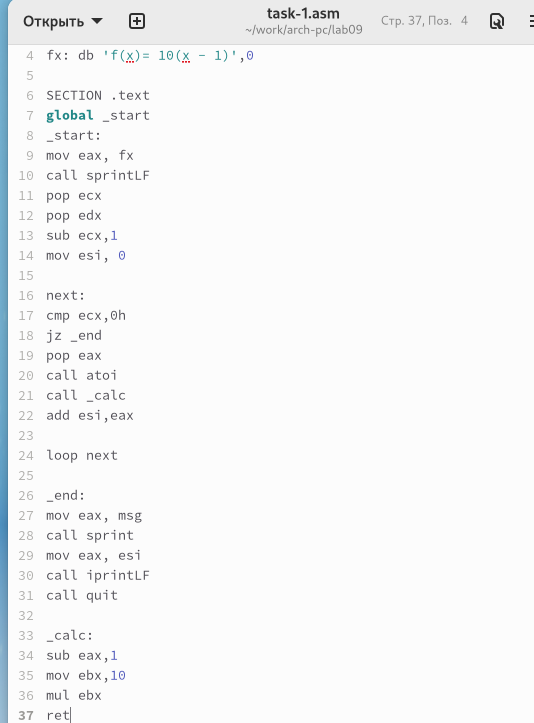


Figure 17: Изменение кода

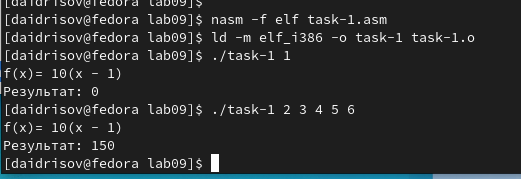


Figure 18: Запуск программы

Ниже приведен исправленный листинг программы, который вычисляет выражение . Однако, при запуске, программа дает неверный результат. Я решил использовать отладчик GDB для анализа изменений значений регистров и определения ошибки.

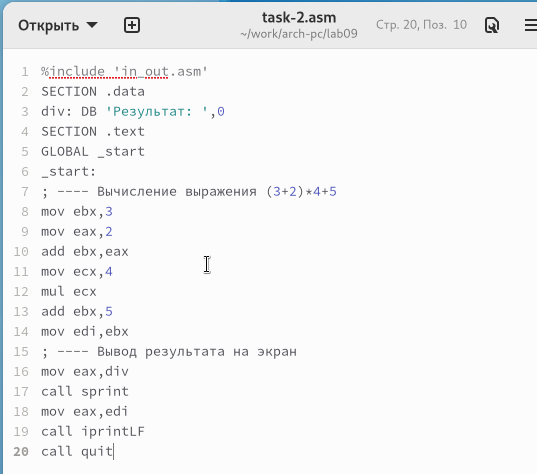


Figure 19: Код с ошибкой

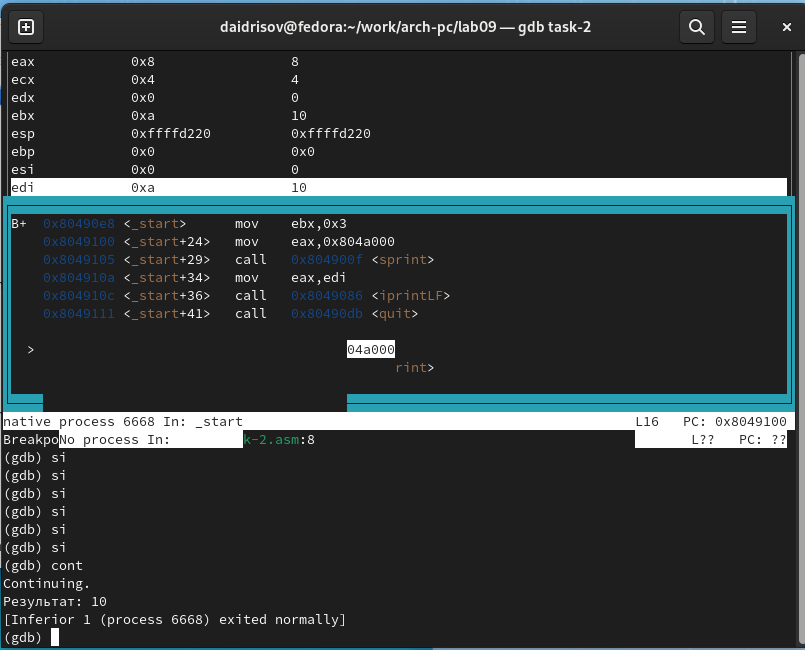


Figure 20: Отладка

В процессе отладки 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:

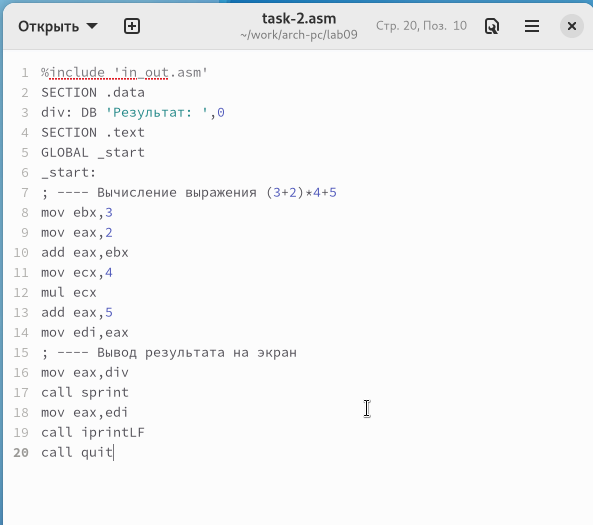


Figure 21: Код исправлен

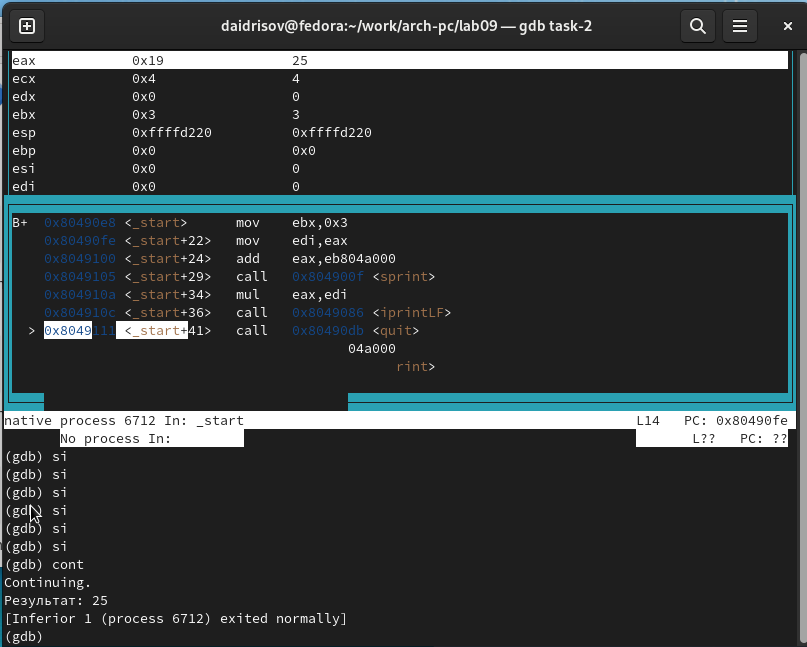


Figure 22: Проверка работы

# 5 Выводы

Освоили работy с подпрограммами и отладчиком.