**РОССИЙСКИЙ** **УНИВЕРСИТЕТ** **ДРУЖБЫ**

**НАРОДОВ**

**Факультет** **физико-математических** **и** **естественных**

**наук**

**направление:** **Компьютерные** **и** **информационные**

**науки**

**Лабораторная** **работа** **№9**

**дисциплина:** **Архитектура** **компьютеров** **и** **операционные** **системы**

**студент:** **Гробман** **Александр** **Евгеньевич**

**Группа:** **НКАбд-02-23**

**Цель** **работы**

Целью работы является приобретение навыков написания программ с использованием

подпрограмм. Знакомство с методами отладки при помощи GDB и его

основными возможностями.

**Задание**

1. Изучение подпрограмм в ассемблере

2. Освоение возможностей отладчика GDB

3. Рассмотрение примеров работы с отладчиком

4. Выполнение заданий для самостоятельной работы

**Теория**

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIXподобных

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.

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.

Если в программе встречается одинаковый участок кода,

его можно оформить в виде подпрограммы, а во всех нужных местах поставить

её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что

позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция

call, которая заносит адрес следующей инструкции в стек и загружает в регистр

eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем

начинается выполнение подпрограммы, которая, в свою очередь,

также может содержать подпрограммы.

Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес,

занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого

выполнение основной программы возобновится с инструкции, следующей за

инструкцией call

**Выполнение** **лабораторной** **работы**

1. **Реализация** **подпрограмм** **в** **NASM**

Создаём файл с именем lab9-1.asm, в котором реализуем программу для вычисления

арифметического выражения f(x) = 2x + 7 с использованием подпрограммы calcul. Для

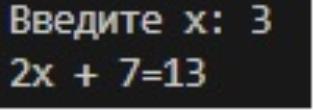
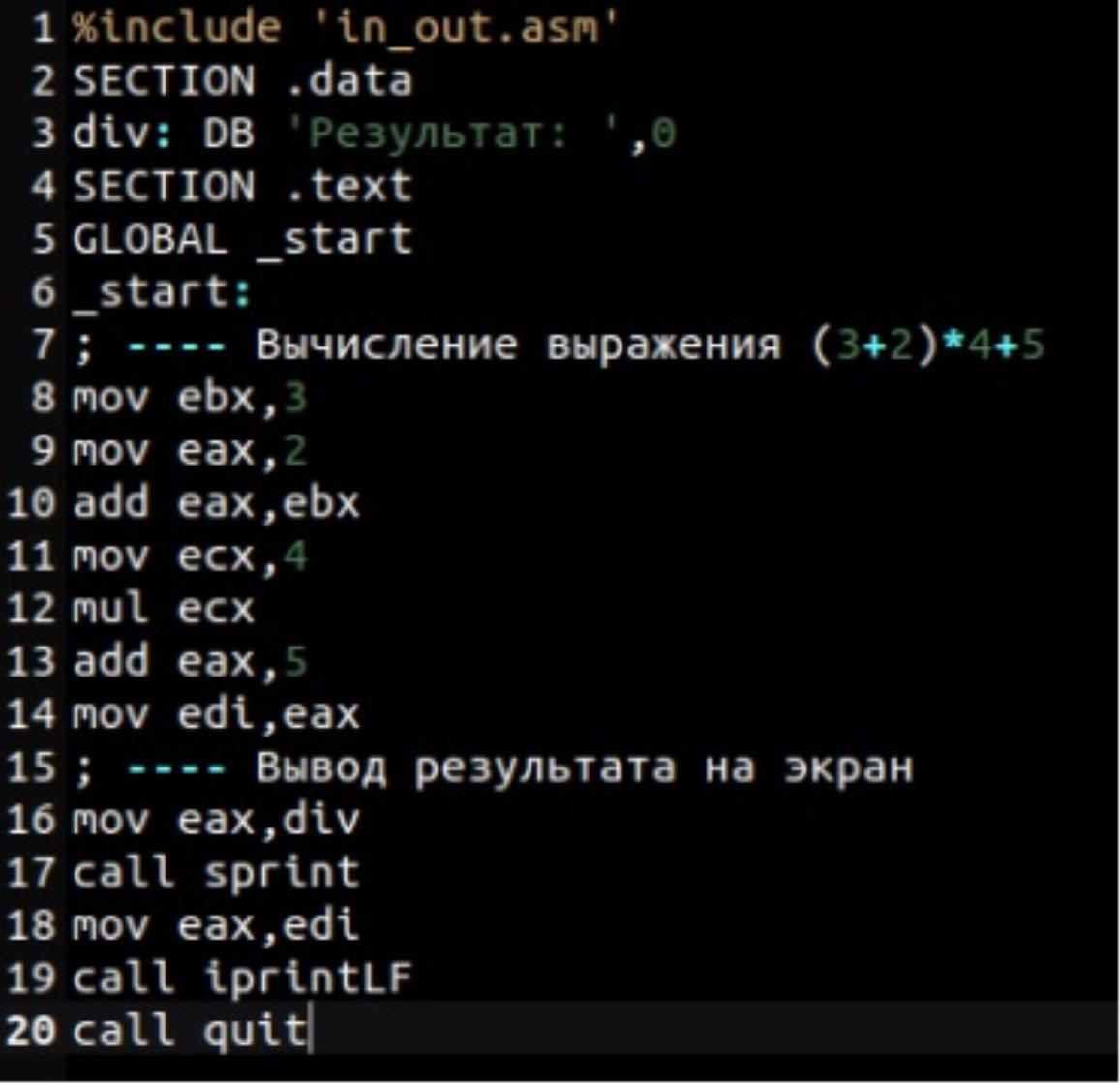
этого я ввожу значение переменной x с клавиатуры, а само выражение вычисляется

внутри подпрограммы.

Добавим подпрограмму subcalcul внутрь подпрограммы calcul. Это позволяет вычислить

составное выражение f(g(x)), где значение x также вводится с клавиатуры. Функции

определены следующим образом: f(x) = 2x + 7, g(x) = 3x - 1.

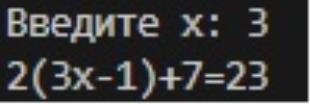
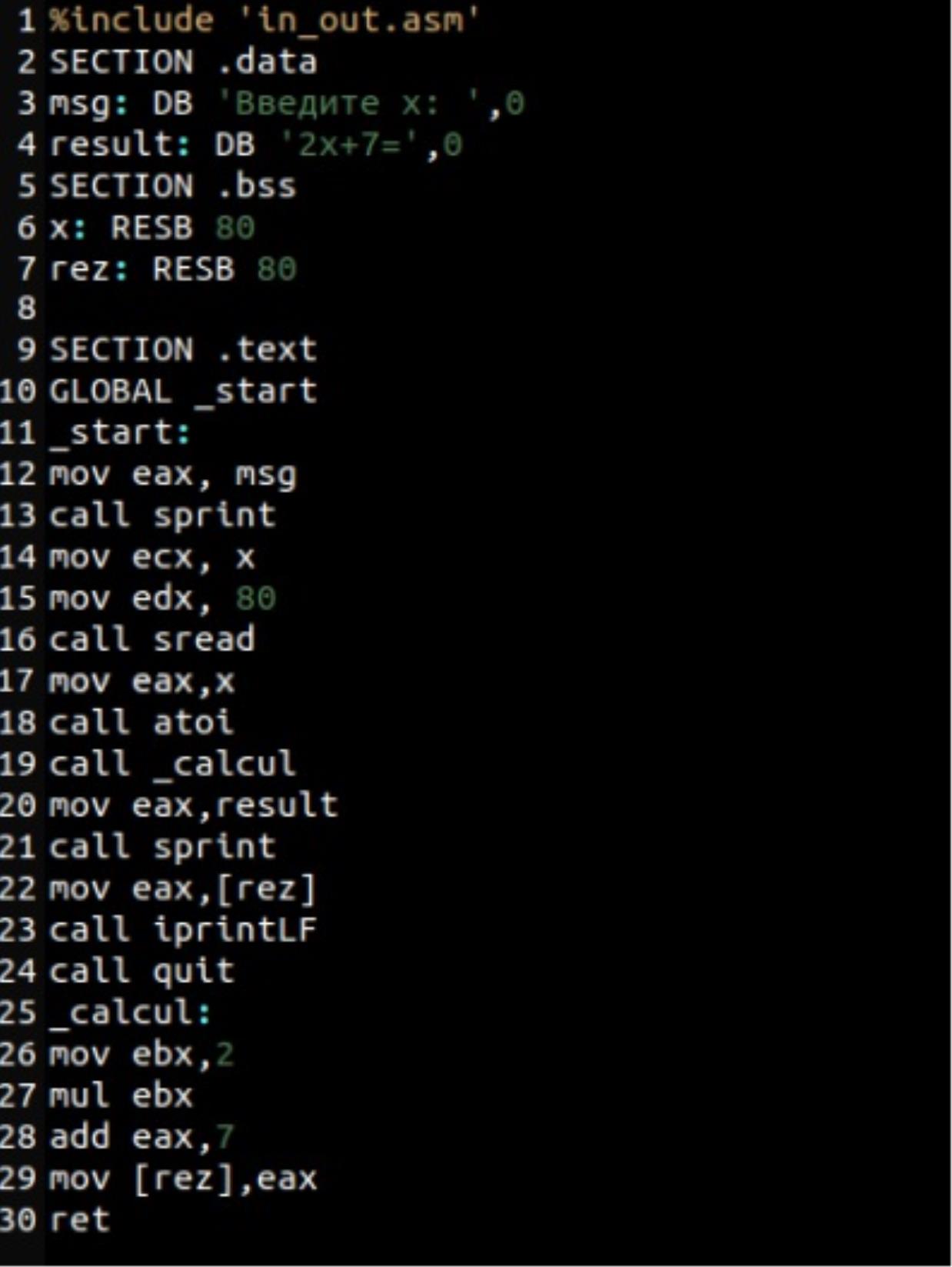


2. **Отладка** **программам** **с** **помощью** **GDB**

Также я создал файл с именем lab9-2.asm, в котором содержится программа

для вывода сообщения “Hello world!”. Я скомпилировал этот файл и получила

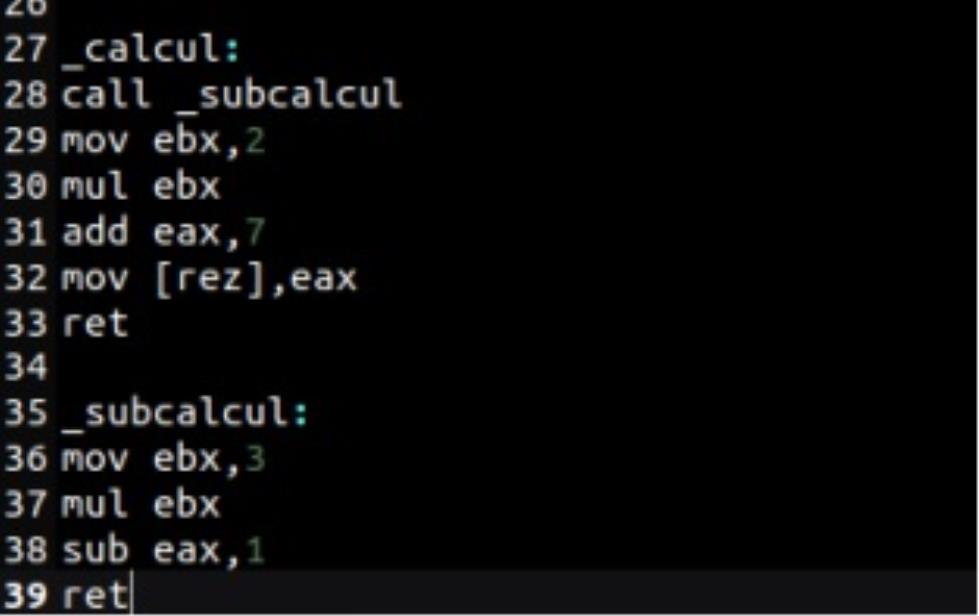
исполняемый файл.



Затем я загрузил полученный исполняемый файл в отладчик GDB и проверил его работу.

Чтобы получить более детальный анализ программы, я установил точку остановки на

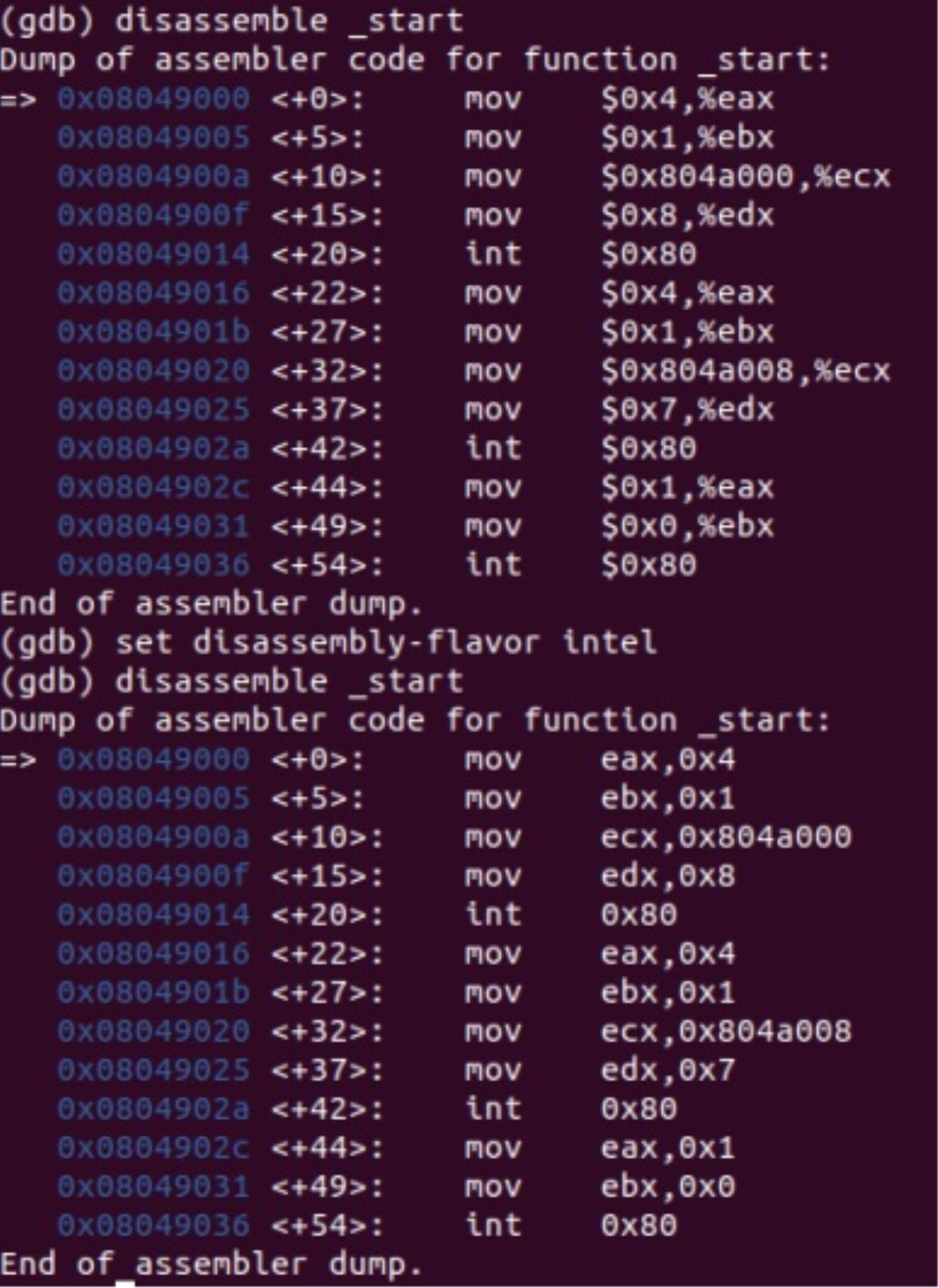
метке “start” и запустил ее.



Я установил еще одну точку остановки по адресу инструкции, определив адрес

предпоследней инструкции “mov ebx, 0x0”. Это помогло мне контролировать

выполнение программы и анализировать ее состояние в отладчике GDB.



Для отслеживания изменений значений регистров,я использовал команду ‘stepi’

(сокращенно ‘si’), которая позволяет выполнить одну инструкцию за раз. Это позволило

мне следить за состоянием программы и анализировать изменения регистров.

Для просмотра значения переменной msg1 по имени и получения нужных

данных, использовала соответствующую команду, предоставленную отладчиком GDB.

Еще одной полезной командой была команда set, которую я использовал для

изменения значения регистра или ячейки памяти. Я указывал имя регистра

или адрес в качестве аргумента команды set, и успешно изменял значения переменных

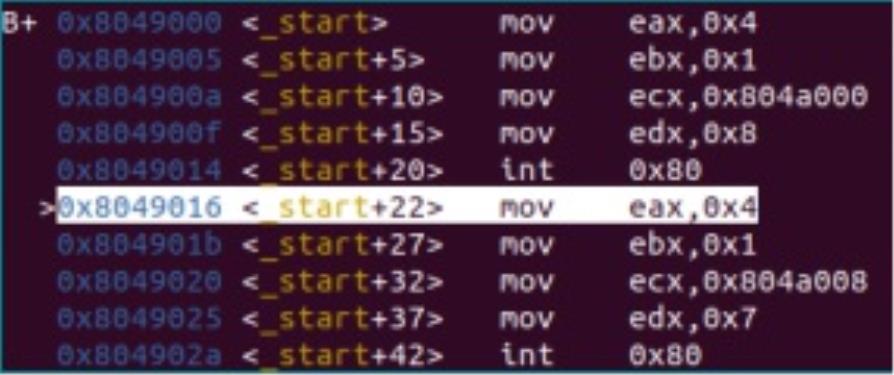
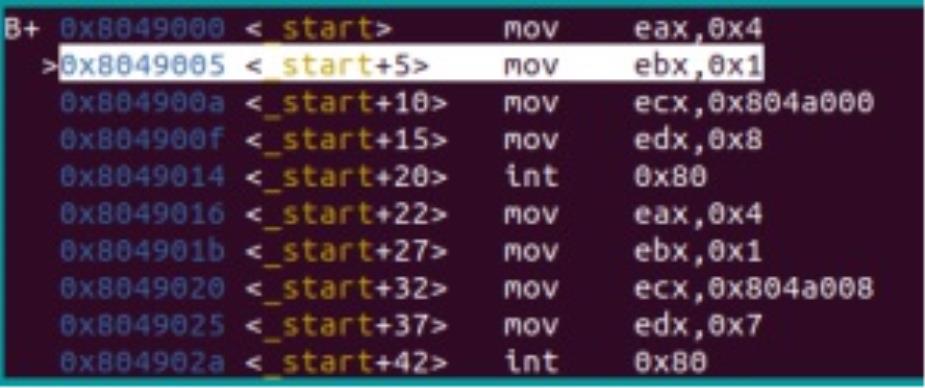
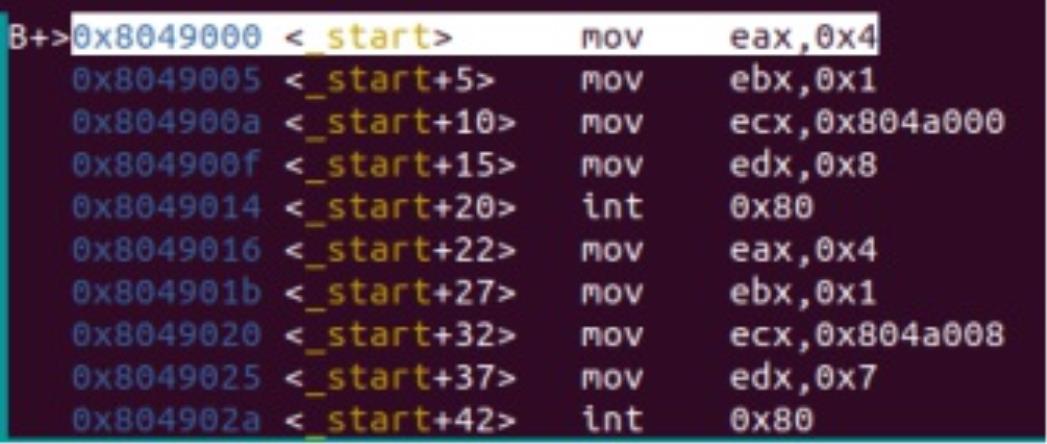
и регистров в процессе отладки программы.

В частности, я успешно изменила первый символ переменной msg1, что позволило мне

проверить поведение программы при изменении данных.

Также, с помощью команды set, я изменяю значение регистра ebx на нужное

значение, чтобы проверить влияние такой модификации на выполнение программы.



Для выполнения лабораторной работы, я решил использовать файл lab8-

2.asm. Этот файл содержит программу, которая выводит аргументы командной строки.

Для загрузки программы с аргументами в отладчик GDB, я использую ключ

–args и загрузил исполняемый файл в отладчик с указанными аргументами. Затем,

установил точку остановки перед первой инструкцией программы и запустил ее.

Далее, я просмотрел остальные позиции стека. По адресу [esp+4], нашёл адрес в

памяти, где располагается имя программы. По адресу [esp+8] хранится адрес первого

аргумента, по адресу [esp+12] - второго, и так далее. Шаг изменения

адреса равен 4 байта, так как каждый следующий адрес на стеке находится на

расстоянии 4 байт от предыдущего ([esp+4], [esp+8], [esp+12]).

3. **Задания** **для** **самостоятельной** **работы**

Я переписываю программу из лабораторной работы №8, задание №1, чтобы реализовать

вычисление значения функции f(x) как подпрограмму.

Однако, при запуске, программа дает неверный результат.

Я провел анализ изменений значений регистров с помощью отладчика GDB

и обнаружила ошибку: перепутан порядок аргументов у инструкции add. По окончанию

работы программы в регистр edi передается значение ebx вместо eax.

Я внес необходимые исправления в код программы, учитывая перепутанный порядок

аргументов у инструкции add и правильную передачу значения в

регистр edi по окончании работы программы. Это позволило исправить ошибку

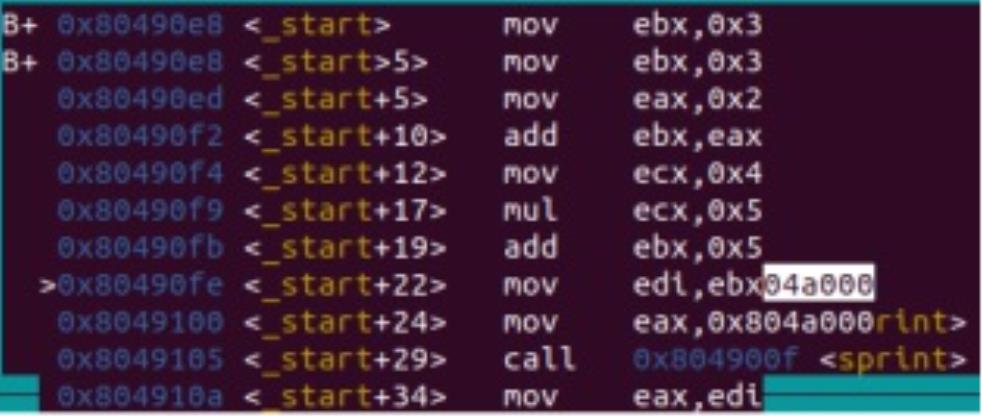
и получить правильный результат вычисления выражения

Я внес необходимые исправления в код программы, учитывая перепутанный порядок

аргументов у инструкции add и правильную передачу значения в

регистр edi по окончании работы программы. Это позволило исправить ошибку

и получить правильный результат вычисления выражения



5. **Вывод**

Я освоил работy с подпрограммами и отладчиком.

**Отправляем** **файлы** **на** **гитхаб.**

Ссылка на отчёт https://github.com/DaOneme/AEGrobman\_study\_2023-

2024\_arhpc/tree/main/Labs/Lab09 (https://github.com/DaOneme/AEGrobman\_study\_2023-

2024\_arhpc/tree/main/Labs/Lab09)

