Отчёт по лабораторной работе 9

Архитектура компьютера

Ушаков Данила Алексеевич

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 Выполнение лабораторной работы

## 2.1 Реализация подпрограмм в NASM

Для начала я создал новую директорию, в которой планировал выполнять лабораторную работу номер 9, и перешел в нее. Затем я создал файл с именем lab9-1.asm.

В качестве 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.(рис. 1) (рис. 2)

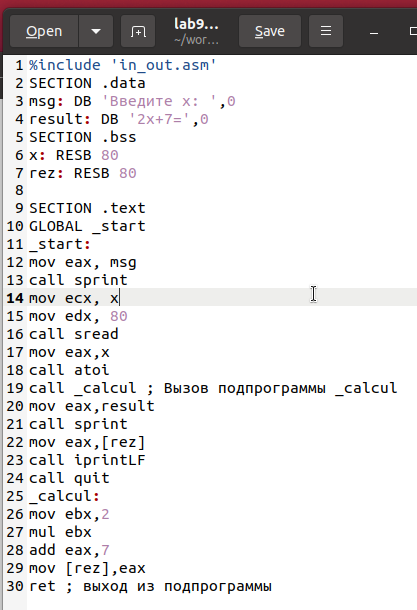


Рис. 1: Код программы lab9-1.asm

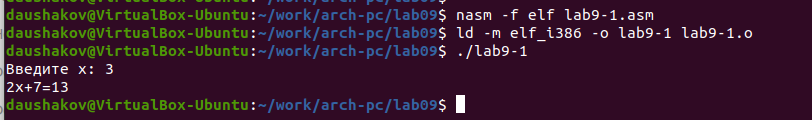


Рис. 2: Компиляция и запуск программы lab9-1.asm

После этого я внес изменения в текст программы, добавив подпрограмму subcalcul внутрь подпрограммы calcul. Это позволяет вычислить составное выражение , где значение также вводится с клавиатуры. Функции определены следующим образом: , .(рис. 3) (рис. 4)

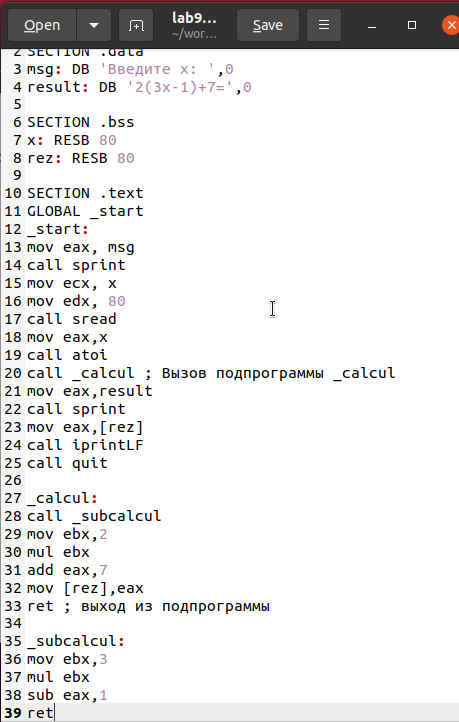


Рис. 3: Код программы lab9-1.asm

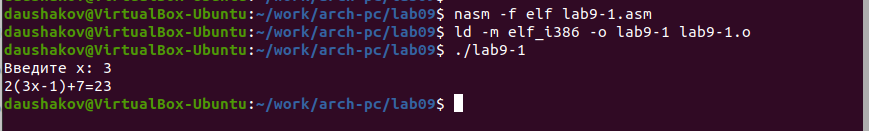


Рис. 4: Компиляция и запуск программы lab9-1.asm

## 2.2 Отладка программам с помощью GDB

Я создал файл с именем lab9-2.asm, в котором содержится текст программы из Листинга 9.2. Эта программа отвечает за печать сообщения “Hello world!”.(рис. 5)

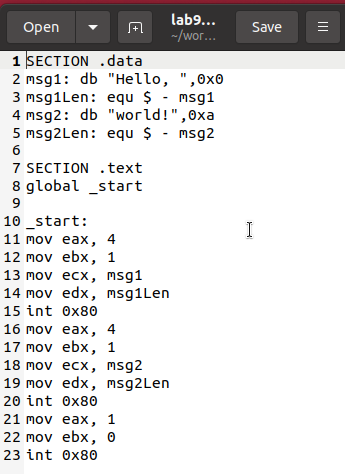


Рис. 5: Код программы lab9-2.asm

После этого я скомпилировал файл и получил исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, использовал ключ “-g”.

Затем я загрузил полученный исполняемый файл в отладчик GDB и проверил его работу, запустив программу с помощью команды “run” или “r”. (рис. 6)

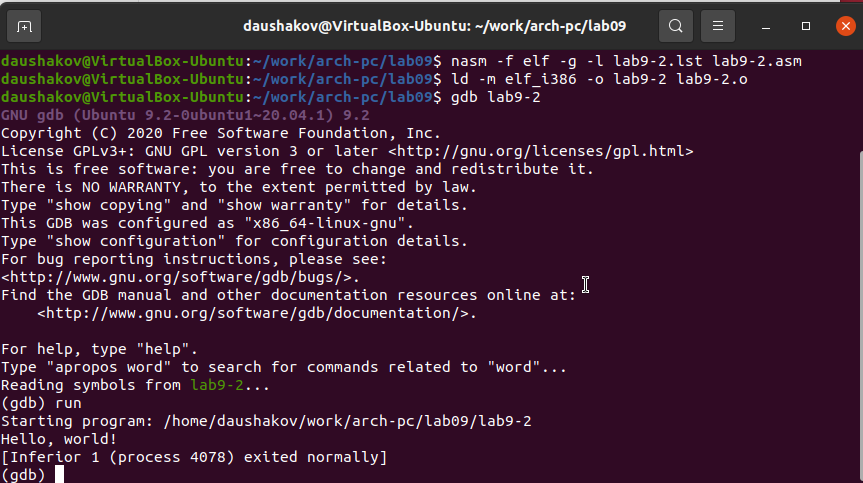


Рис. 6: Компиляция и запуск программы lab9-2.asm в отладчике

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.(рис. 7) (рис. 8)

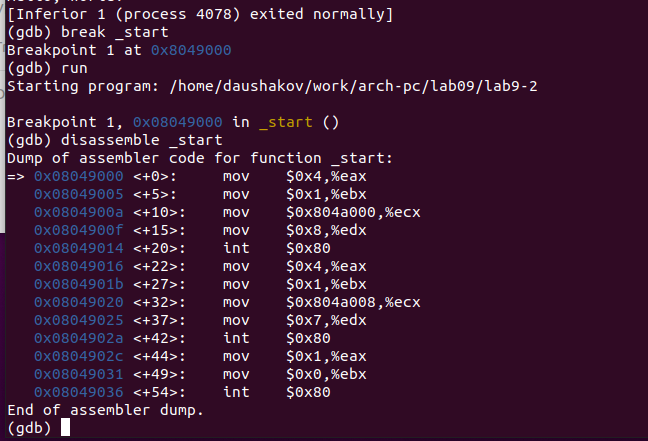


Рис. 7: Дизассемблированный код

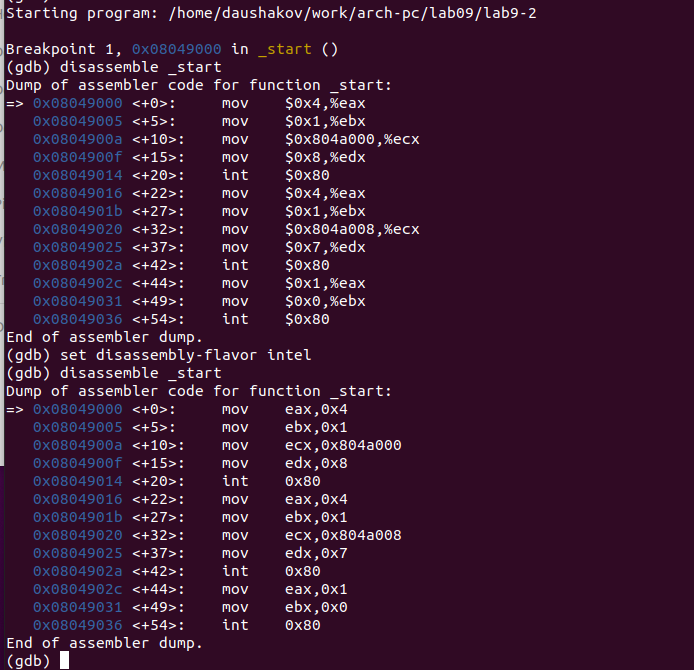


Рис. 8: Дизассемблированный код в режиме интел

### 2.2.1 Точки остановки

Чтобы проверить точку остановки по имени метки “\_start”, я использовал команду “info breakpoints” или “i b”. Затем установил еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0” (рис. 9)

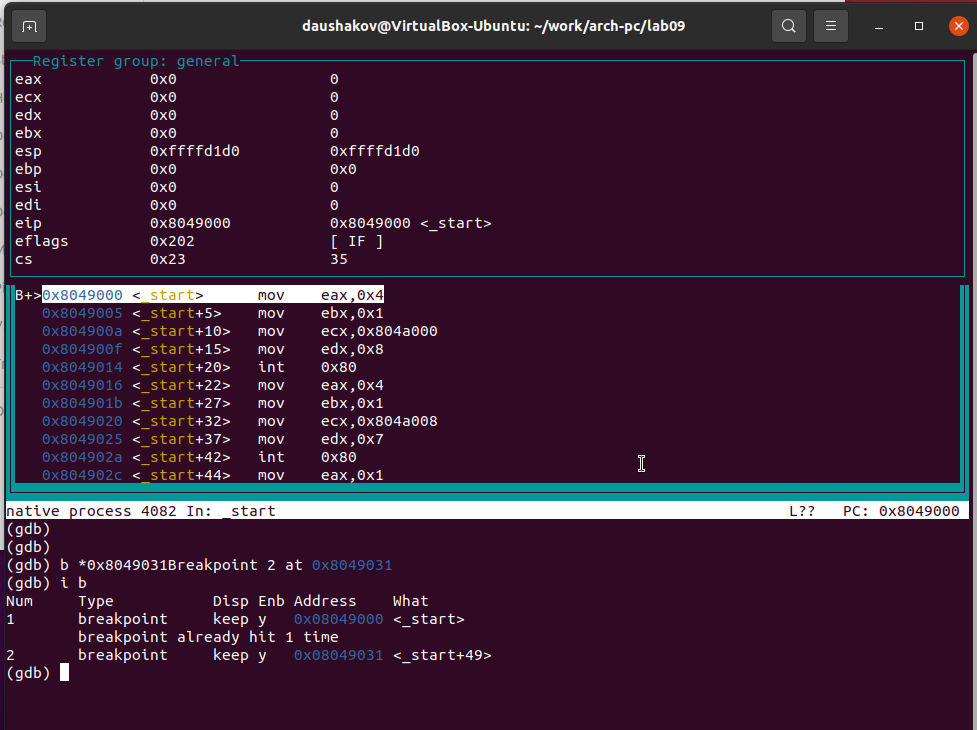


Рис. 9: Точка остановки

### 2.2.2 Работа с данными программы в GDB

В отладчике GDB можно просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Выполнил 5 инструкций с помощью команды ‘stepi’ (сокращенно ‘si’) и отследил изменение значений регистров. (рис. 10) (рис. 11)

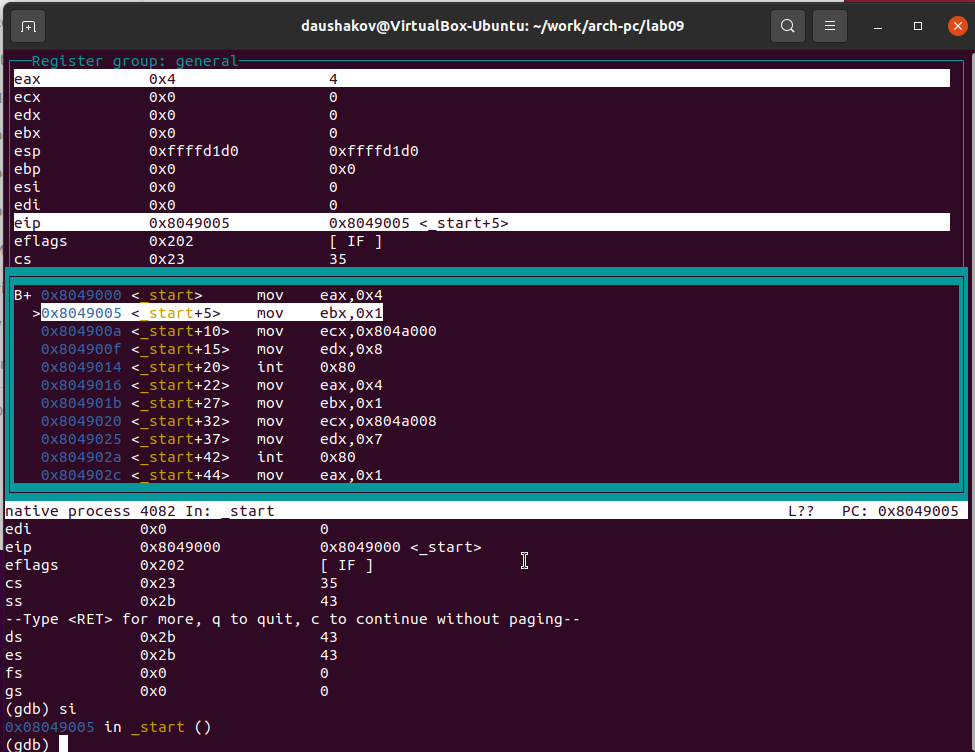


Рис. 10: Изменение регистров

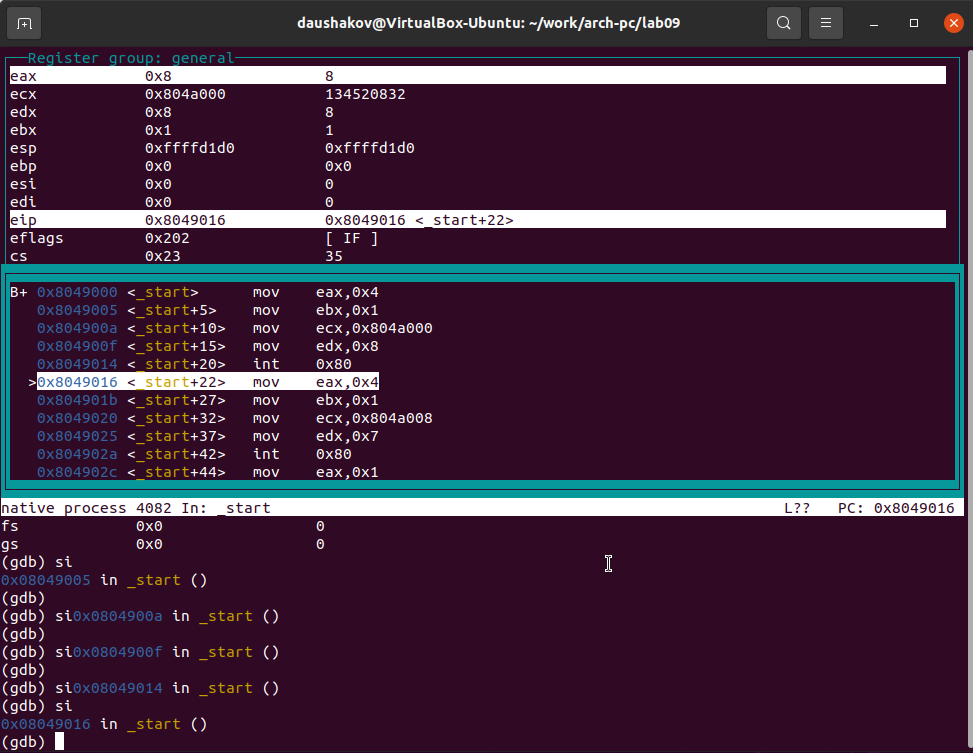


Рис. 11: Изменение регистров

Просмотрел значение переменной msg1 по имени и получил нужные данные.

Просмотрел значение переменной msg1 по имени и получил нужные данные.

Для изменения значения регистра или ячейки памяти использовал команду set, указав имя регистра или адрес в качестве аргумента. Изменил первый символ переменной msg1. (рис. 12)

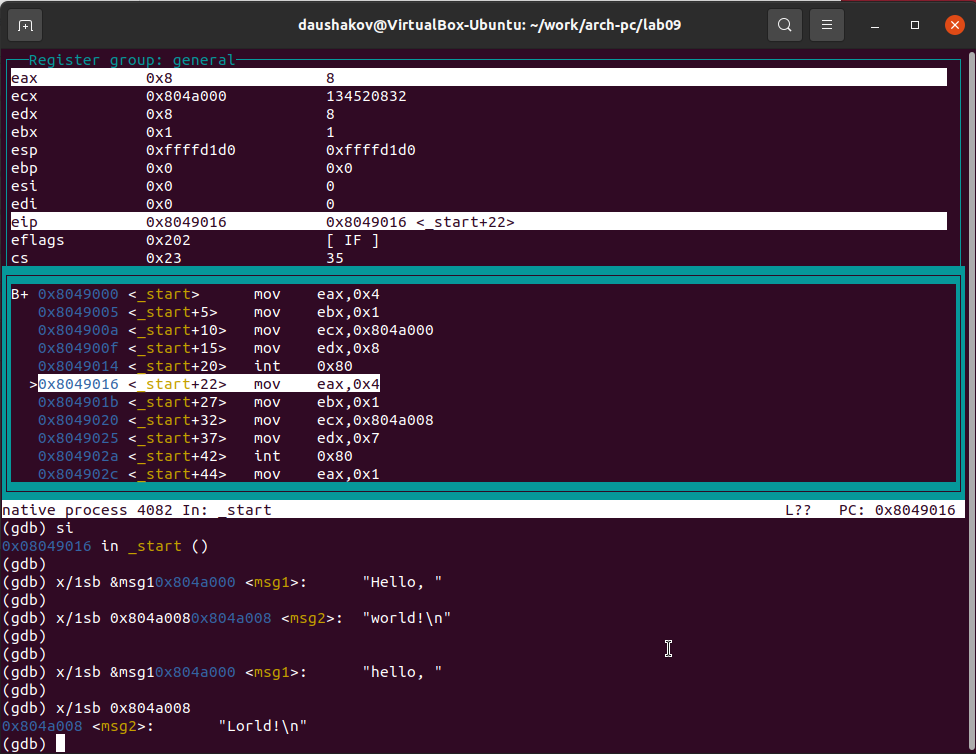


Рис. 12: Изменение значения переменной

Для изменения значения регистра или ячейки памяти использовал команду set, указав имя регистра или адрес в качестве аргумента. Изменил первый символ переменной msg1.(рис. 13)

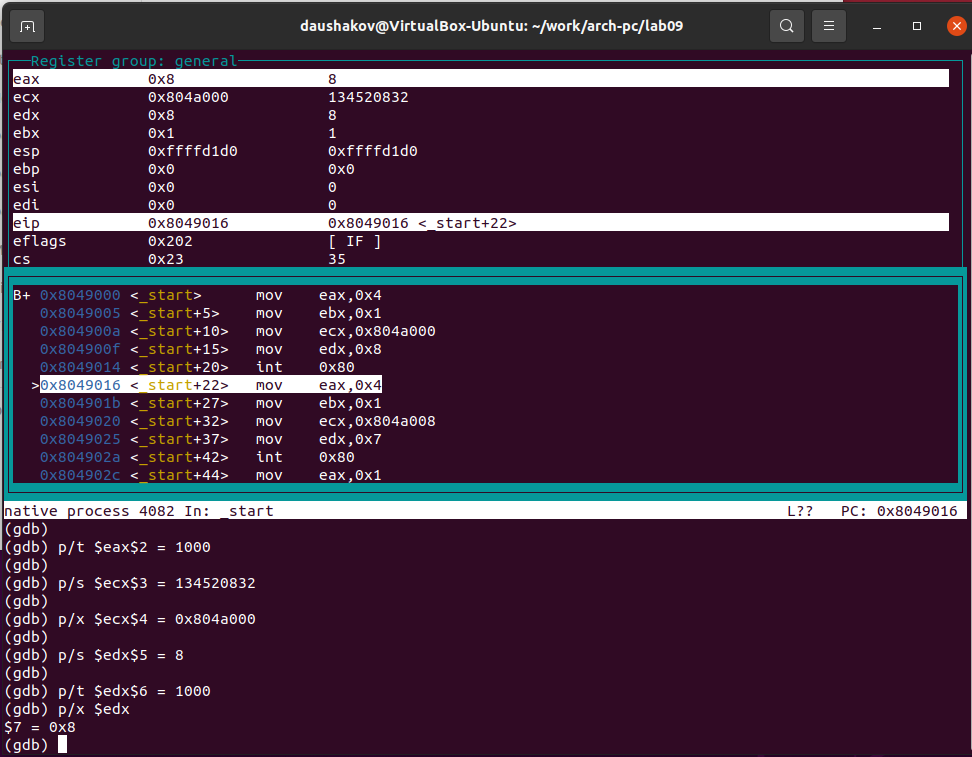


Рис. 13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx на нужное значение. (рис. 14)

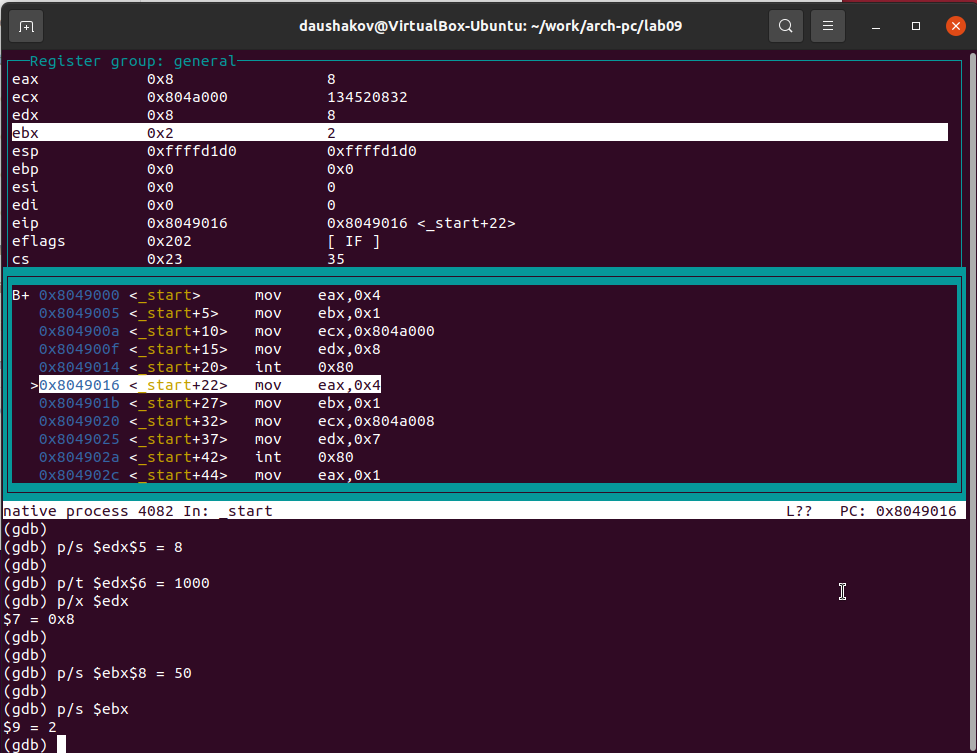


Рис. 14: Вывод значения регистра

### 2.2.3 Обработка аргументов командной строки в GDB

Скопировал файл lab8-2.asm, созданный во время выполнения лабораторной работы №8, который содержит программу для вывода аргументов командной строки. Создал исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в gdb использовал ключ –args и загрузил исполняемый файл в отладчик с указанными аргументами.

Установил точку останова перед первой инструкцией программы и запустил ее.

Адрес вершины стека, содержащий количество аргументов командной строки (включая имя программы), хранится в регистре esp. По этому адресу находится число, указывающее количество аргументов. В данном случае видно, что количество аргументов равно 5, включая имя программы lab9-3 и сами аргументы: аргумент1, аргумент2 и ‘аргумент 3’.

Просмотрел остальные позиции стека. По адресу [esp+4] находится адрес в памяти, где располагается имя программы. По адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] - второго и так далее. (рис. 15)

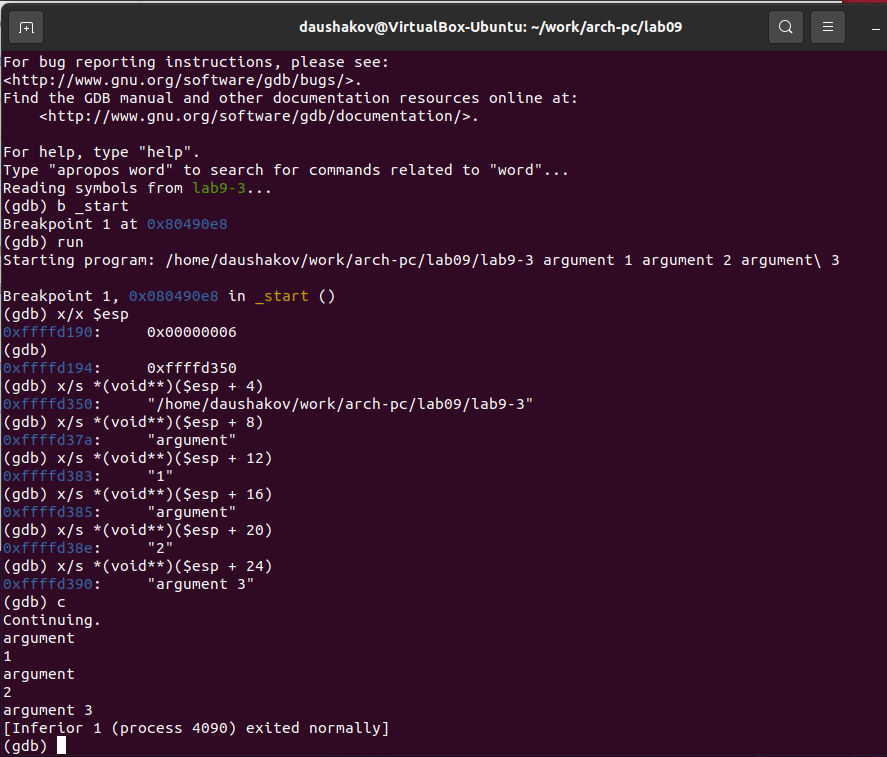


Рис. 15: Вывод значения регистра

Шаг изменения адреса равен 4, так как каждый следующий адрес на стеке находится на расстоянии 4 байт от предыдущего ([esp+4], [esp+8], [esp+12]).

## 2.3 Задание для самостоятельной работы

Преобразовал программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. 16) (рис. 17)

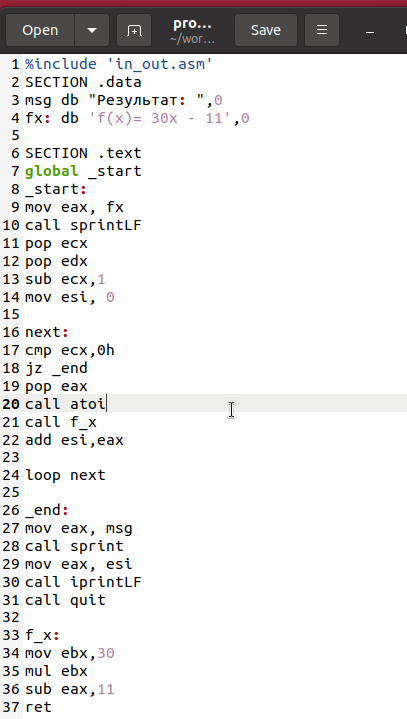


Рис. 16: Код программы prog-1.asm

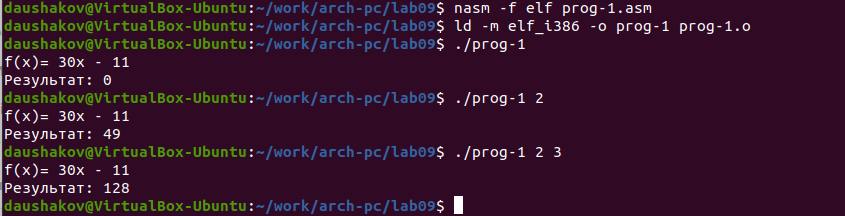


Рис. 17: Компиляция и запуск программы prog-1.asm

В листинге приведена программа вычисления выражения . При запуске данная программа дает неверный результат. Проверил это, анализируя изменения значений регистров с помощью отладчика GDB.

Определил ошибку - перепутан порядок аргументов у инструкции add. Также обнаружил, что по окончании работы в edi отправляется ebx вместо eax.(рис. 18)

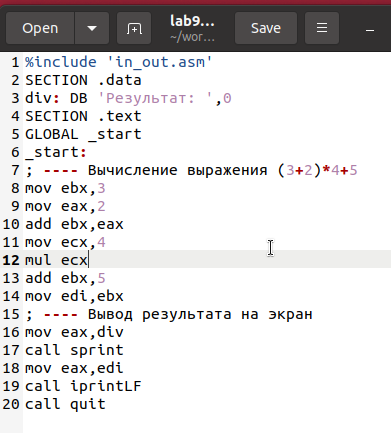


Рис. 18: Код с ошибкой

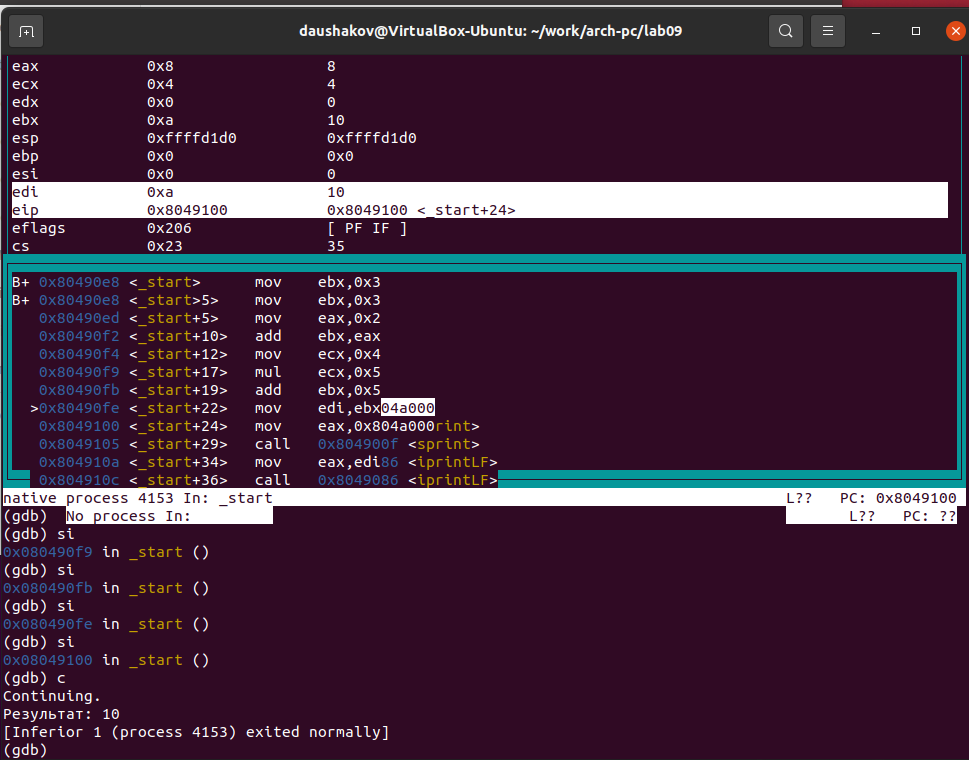


Рис. 19: Отладка

Отмечу, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax (рис. 19)

Исправленный код программы (рис. 20) (рис. 21)

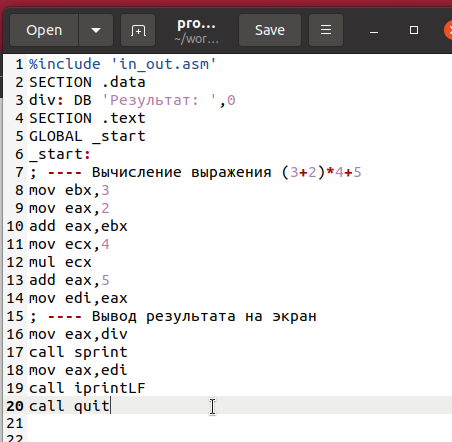


Рис. 20: Код исправлен

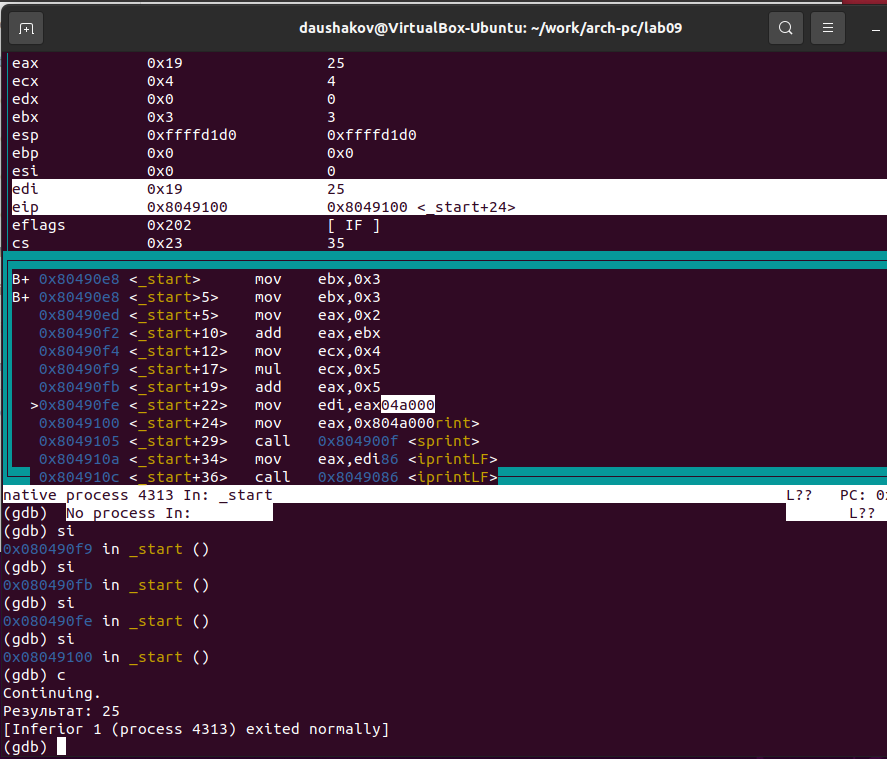


Рис. 21: Проверка работы

# 3 Выводы

Освоили работy с подпрограммами и отладчиком.