Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Сидельников Андрей Владимирович

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

Cоздал каталог для выполнения лабораторной работы № 9 и перешел в него. Затем создал файл lab9-1.asm. (рис. [1](#fig:001)).

Figure 1: Создание файла

Figure 1: Создание файла

Вписываю листинг9.1 в созданный файл (рис. [2](#fig:002)).

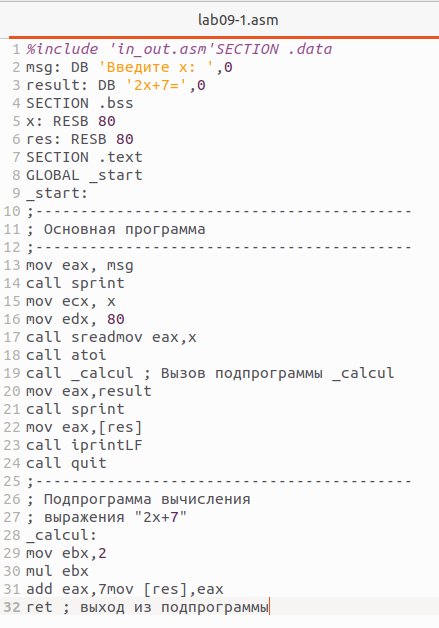


Figure 2: Листинг9.1

Запускаю программу из листинга9.1 (рис. [3](#fig:003)).

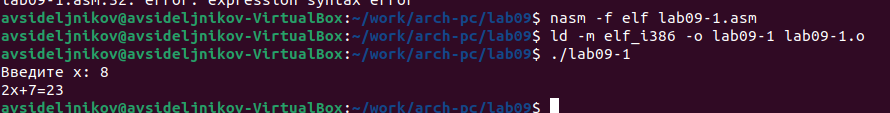


Figure 3: Запуск программы

Изменил текст программы (рис. [4](#fig:004)).

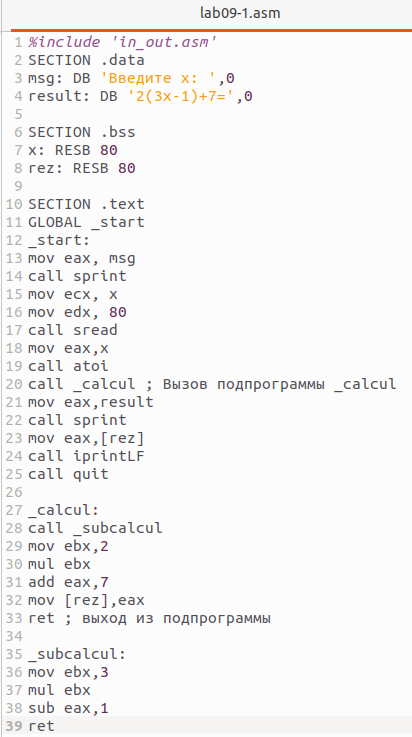


Figure 4: Изменённый текст программы

Запускаю изменённую программу (рис. [5](#fig:005)).

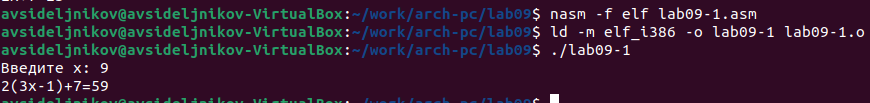


Figure 5: Запуск программы

Создал файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. [6](#fig:006)).

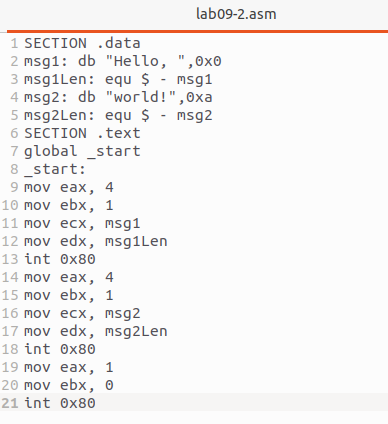


Figure 6: Листинг 9.2 в созданном файле

Получил исполняемый файл и добавил отладочную информацию с помощью ключа ‘-g’ для работы с GDB.Загрузил исполняемый файл в отладчик GDB и проверил работу программы, запустив ее с помощью команды ‘run’ (рис. [7](#fig:007)).

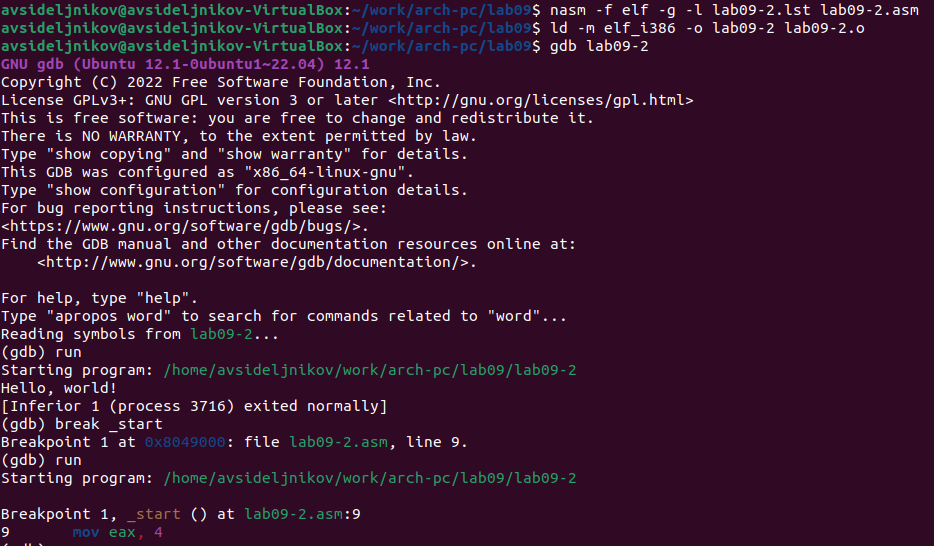


Figure 7: Запуск программы lab09-2.asm в отладичике

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. [8](#fig:008)).



Просмотр с помощью команды disassemble \_start

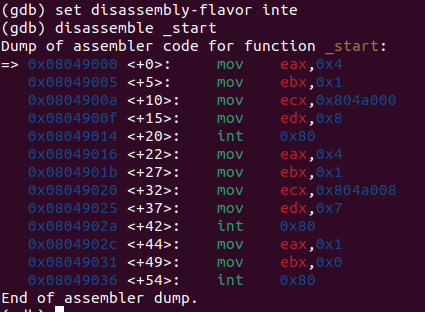


Figure 8: Дизассемблированный код в режиме intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

В Порядке операндов:

В ATT в начале идёт исходный операнд, затем целевой операнд. Например: movl %eax, %ebx (переместить данные из eax в ebx).

В Intel используется целевой операнд, затем исходный операнд. Например: mov ebx, eax.

Для проверки точки остановки по имени метки ’\_start’, использовал команду ‘info breakpoints’ . Затем установил еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции ‘mov ebx, 0x0’ (рис. [9](#fig:009)).

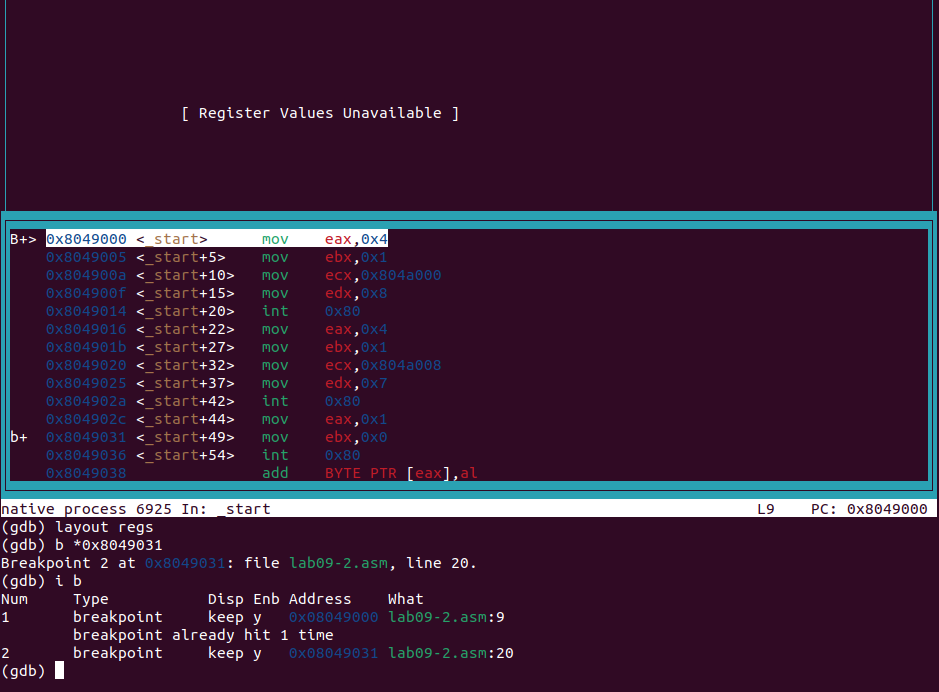


Figure 9: Точка установки

Выполнил 5 инструкций с помощью команды stepi и проследил за измением значений регистров (рис. [10](#fig:010)) (рис. [11](#fig:011)).

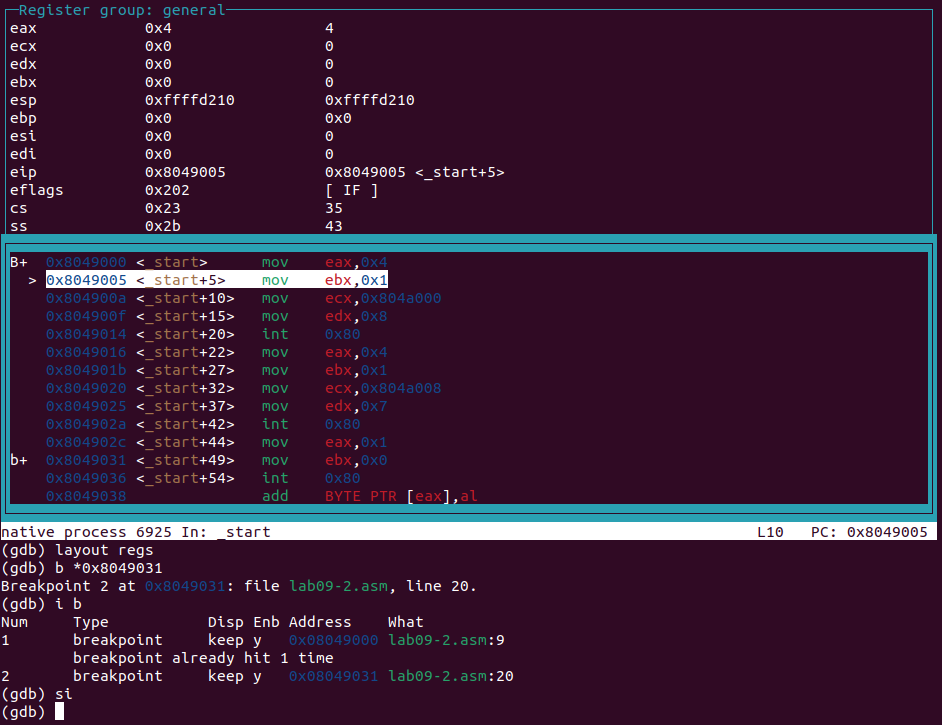


Figure 10: Изменение регистров

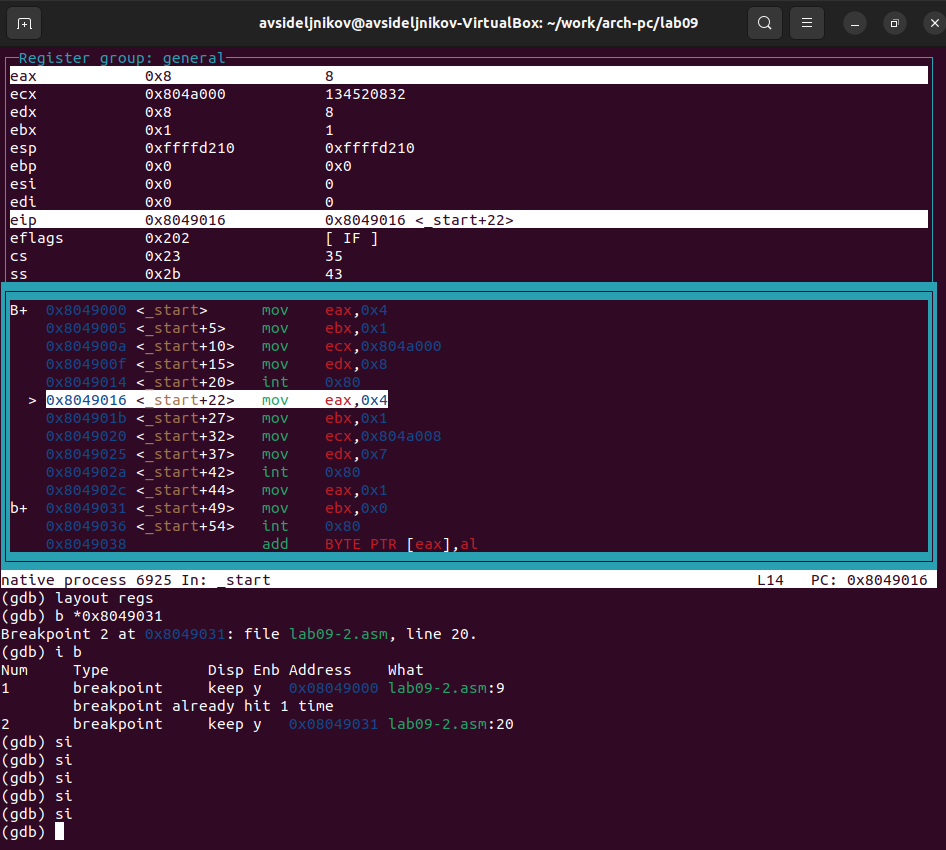


Figure 11: Изменение регистров

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](#fig:012)).

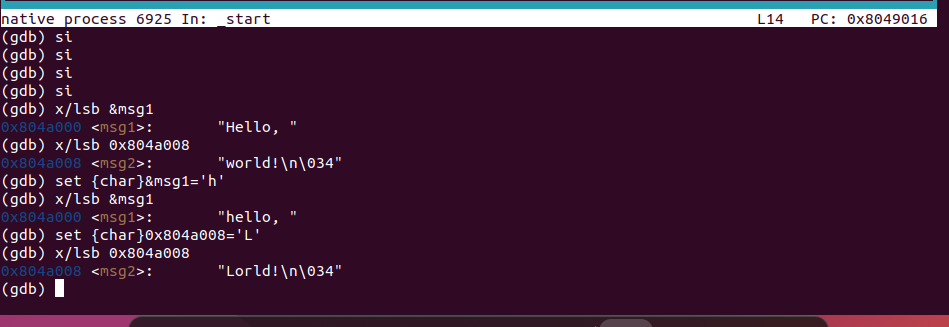


Figure 12: Изменение значение переменной

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](#fig:013))

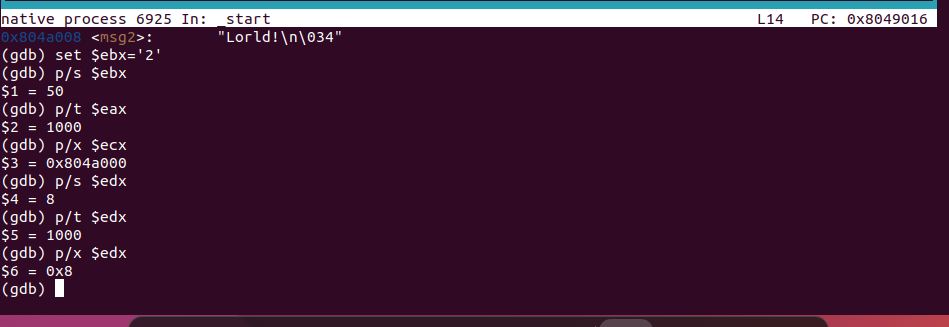


Figure 13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx на нужное значение (рис. [14](#fig:014))

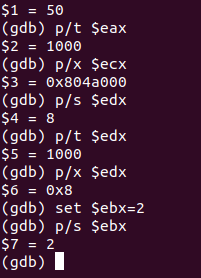


Figure 14: Вывод значения регистра

Копирую файл созданный при выполнении лабораторной работы №8 и создаю исполняемый файл

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](#fig:015)).

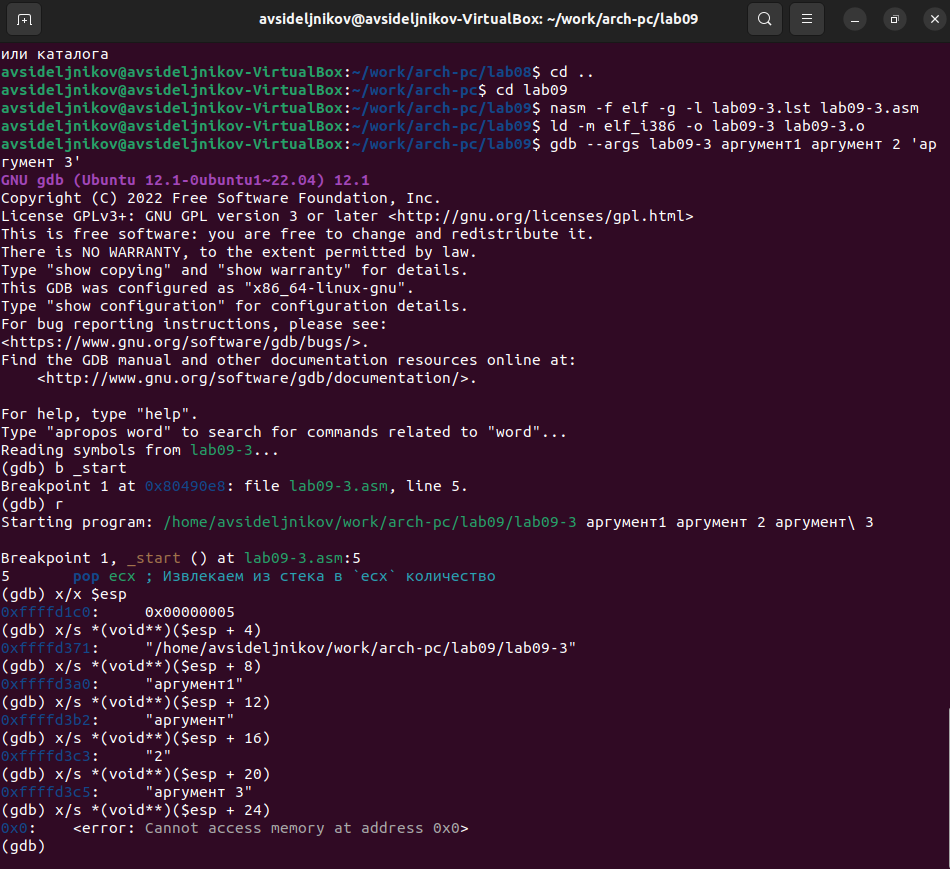


Figure 15: Вывод значений регистра

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])

# 3 Самостоятельная работа

Преобразовал программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму (рис. [16](#fig:016)) (рис. [17](#fig:017)).

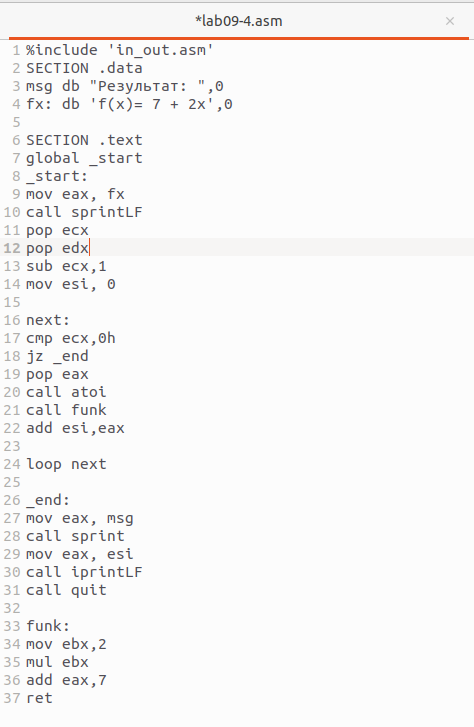


Figure 16: Программа в файле lab09-4.asm

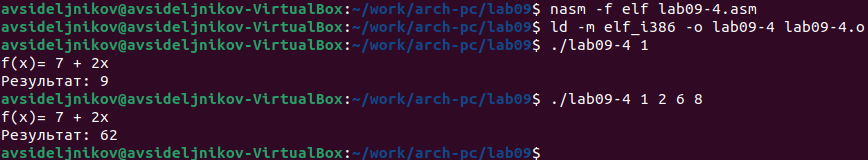


Figure 17: Запуск программы lab09-4.asm

В листинге приведена программа вычисления выражения (3+2)\*4+5. 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](#fig:018)).

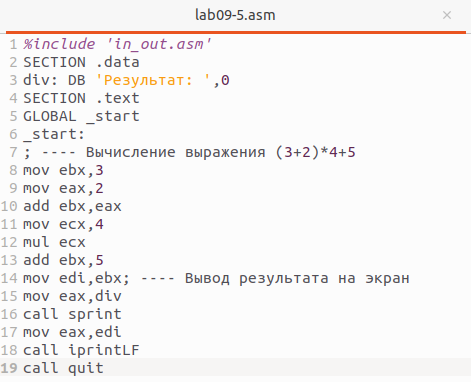


Figure 18: Код с ошибкой

Перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax (рис. [19](#fig:019)).

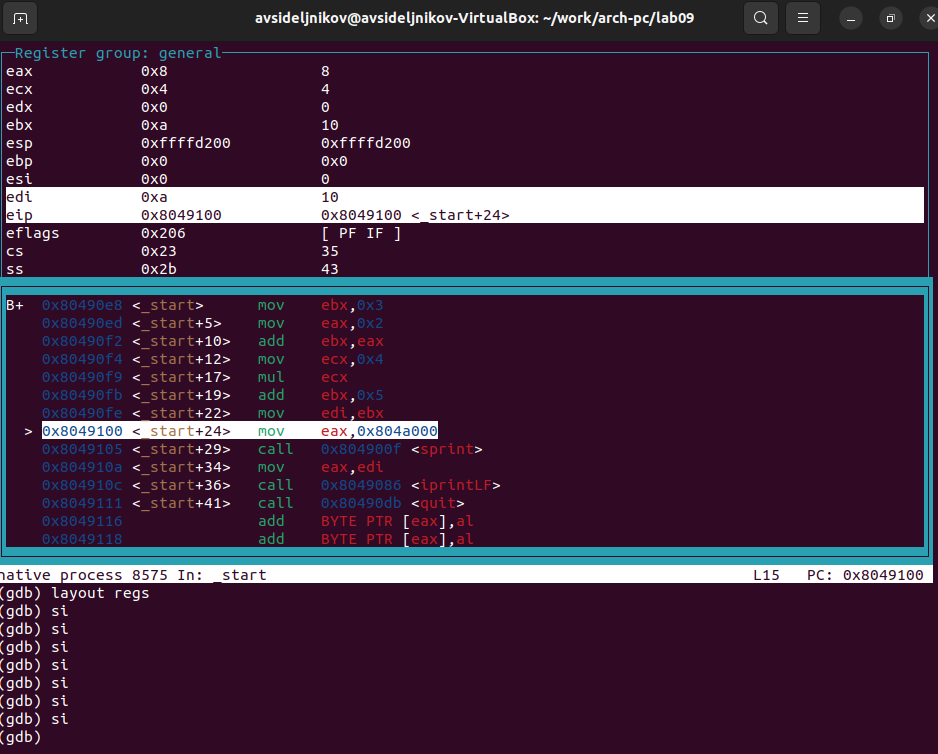


Figure 19: Отладка

Исправленный код программы (рис. [20](#fig:020)) (рис. [21](#fig:021)).

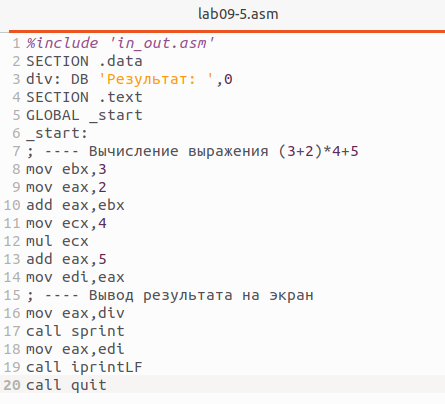


Figure 20: Изменённая программа

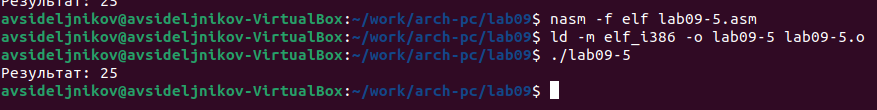


Figure 21: Проверка программы

# 4 Выводы

Я освоил работy с подпрограммами и отладчиком.