Отчёт по лабораторной работе 9

Архитектура компьютера

Тумуреева Галина Аркадьевна

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

1. Я создала папку для выполнения лабораторной работы номер девять, затем перешла в неё и сформировала файл lab9-1.asm.
2. Давайте рассмотрим пример программы, которая вычисляет арифметическую функцию с использованием вспомогательной подпрограммы calcul. В этом случае значение мы получаем через ввод с клавиатуры, а расчёт самой функции происходит внутри подпрограммы.

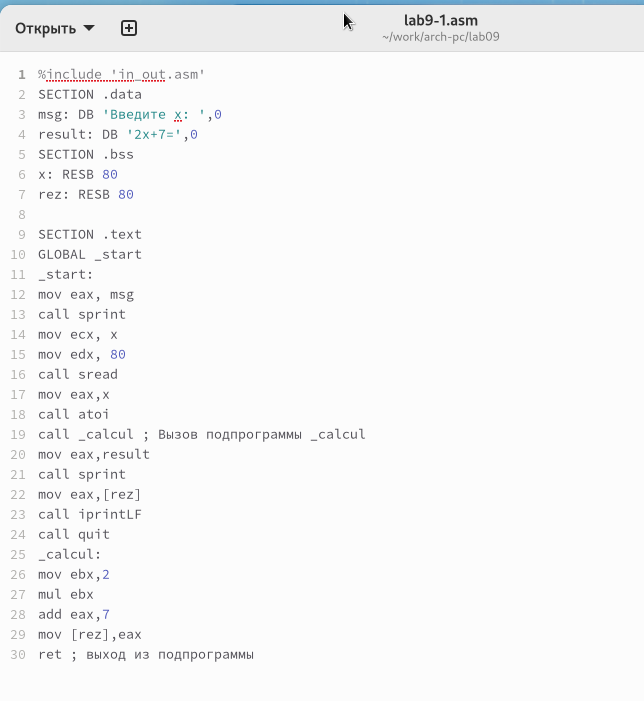


Figure 1: Программа в файле lab9-1.asm

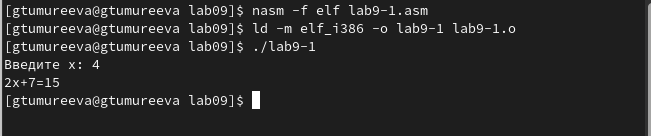


Figure 2: Запуск программы lab9-1.asm

1. Я внесла изменения в код программы, добавив в подпрограмму calcul дополнительную подпрограмму subcalcul. Это позволило мне вычислить составное выражение , где также вводится через клавиатуру, а функции заданы как и .

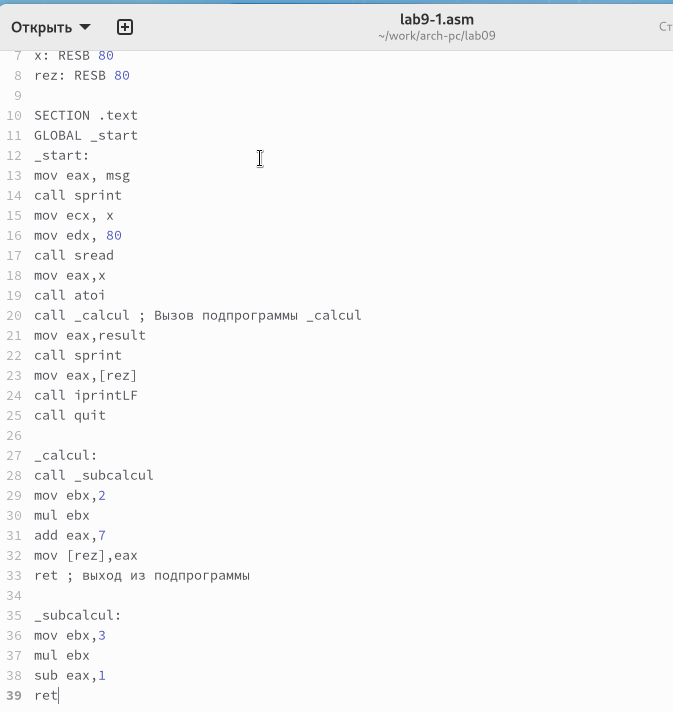


Figure 3: Программа в файле lab9-1.asm

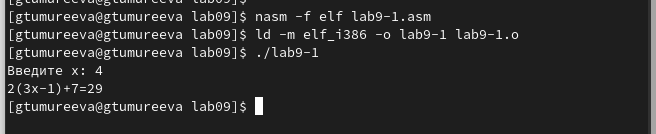


Figure 4: Запуск программы lab9-1.asm

1. Я создала файл lab9-2.asm, в который вписала код программы из Листинга 9.2, который выводит на экран сообщение “Hello world!”.

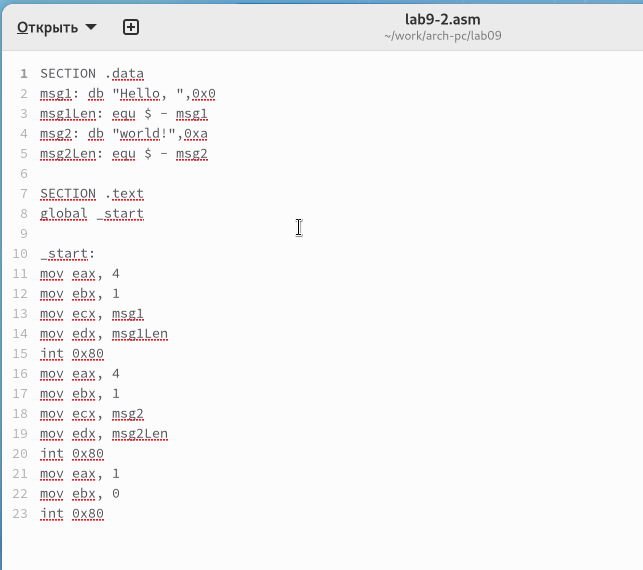


Figure 5: Программа в файле lab9-2.asm

После этого я получила исполняемый файл. Чтобы использовать отладчик GDB, мне нужно было добавить в исполняемый файл отладочную информацию. Для этого я скомпилировала программу с ключом ‘-g’. Затем я загрузила исполняемый файл в отладчик gdb и проверила, как работает программа, выполнив её в среде GDB с использованием команды run (или просто r).

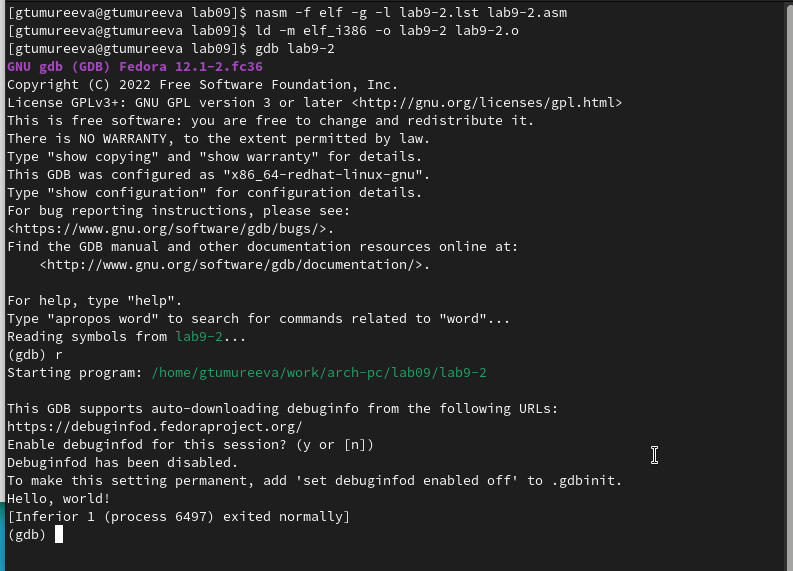


Figure 6: Запуск программы lab9-2.asm в отладчике

Чтобы более детально разобраться в программе, я поставила точку останова у метки \_start, с которой начинается любая программа на ассемблере, и запустила её. Затем я взглянула на дизассемблированный код.

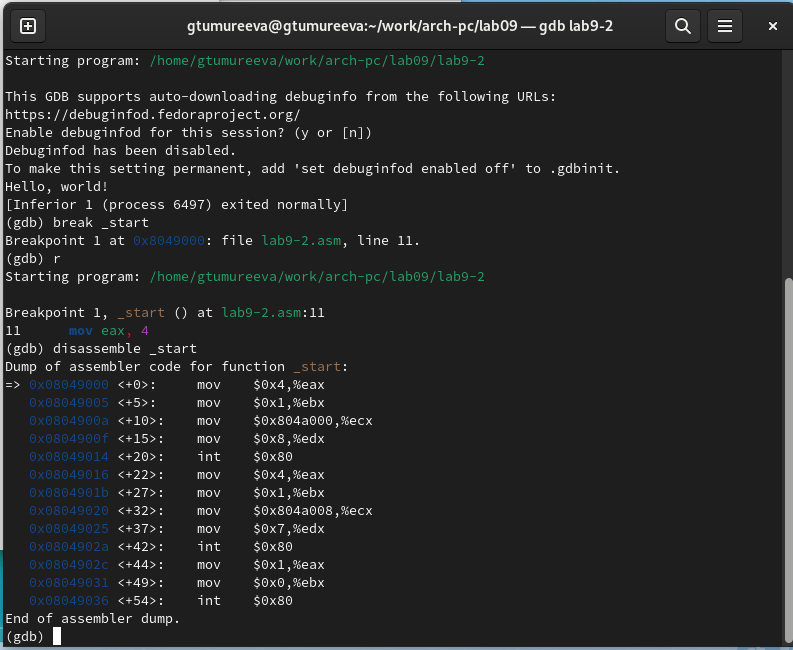


Figure 7: Дизассимилированный код

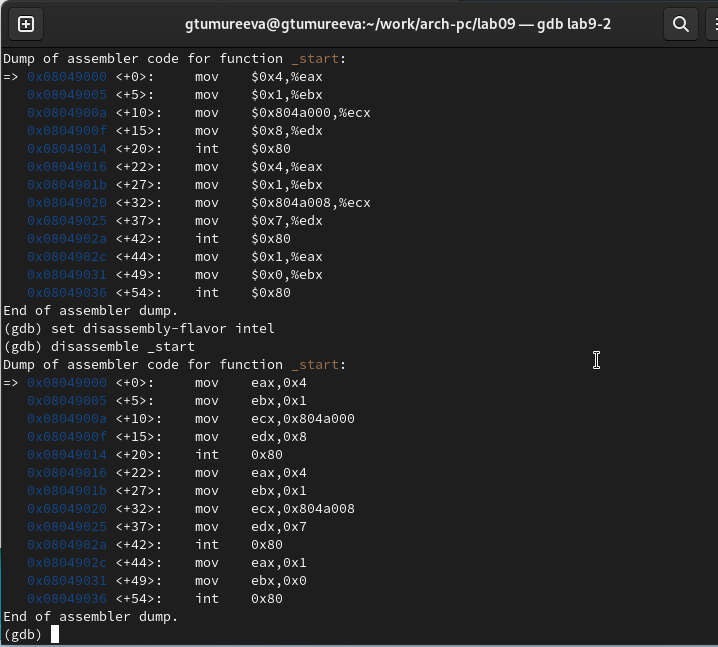


Figure 8: Дизассимилированный код в режиме интел

На предыдущем этапе я уже разместила брейкпоинт с именем \_start и проверила это, используя команду info breakpoints, или просто i b. После этого я установила ещё одну точку останова на адрес определённой инструкции, который можно было найти в середине экрана, в левой колонке напротив соответствующей инструкции. Я выбрала адрес предпоследней инструкции (mov ebx,0x0) и поставила там брейкпоинт.

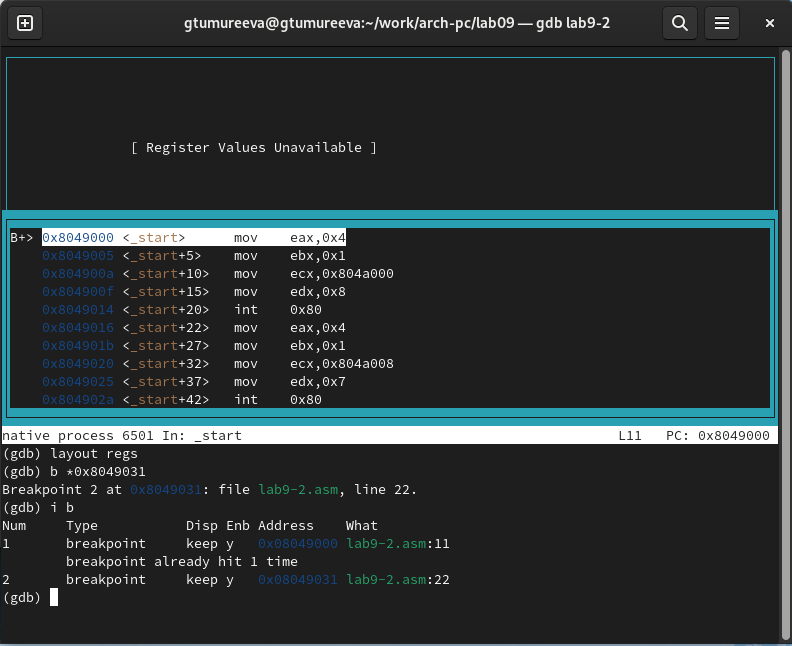


Figure 9: Точка остановки

Отладчик предоставляет возможность просмотра содержимого ячеек памяти и регистров, и при необходимости я могу вручную поменять значения регистров или переменных. Я выполнила пять инструкций с помощью команды stepi (или si) и наблюдала за тем, как меняются значения в регистрах.

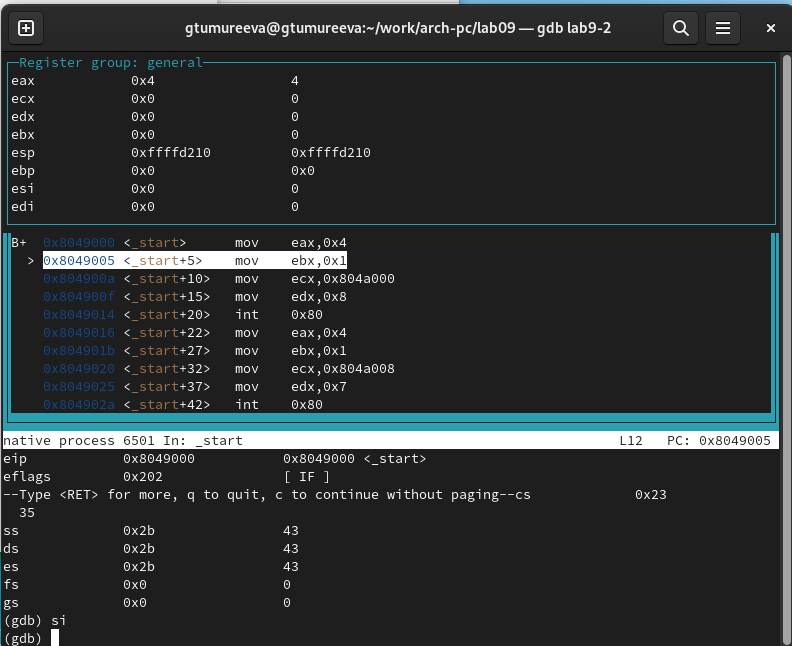


Figure 10: Изменение регистров

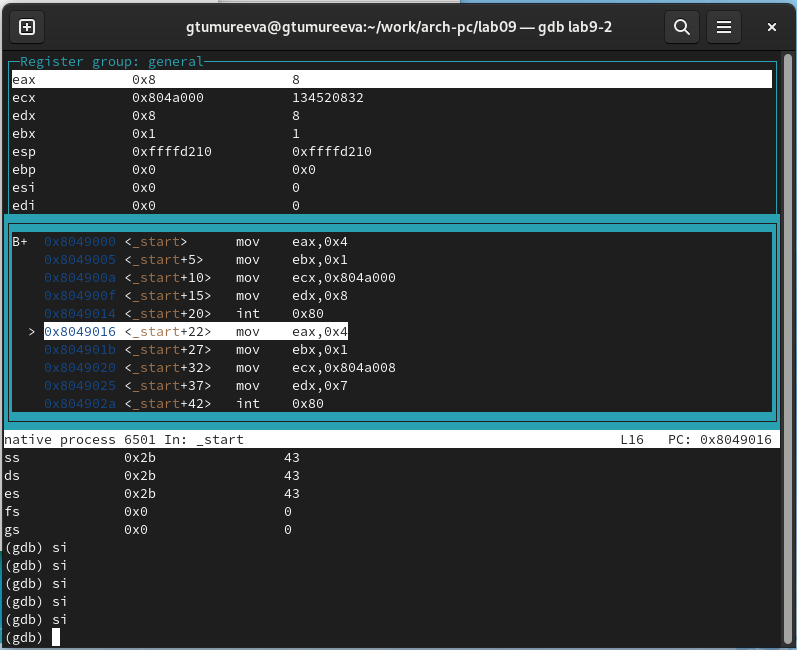


Figure 11: Изменение регистров

Я проверила значение переменной msg1 по её имени и значение переменной msg2, обратившись к ней по адресу.

Чтобы изменить значение регистра или ячейки памяти, я использовала команду set, указав ей имя регистра или адрес в качестве аргумента. Я изменила первый символ в переменной msg1.

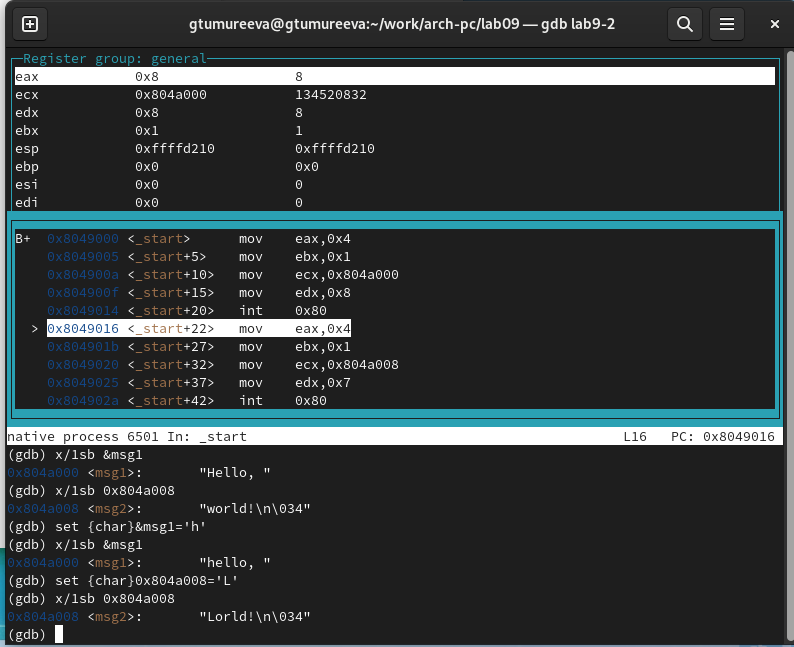


Figure 12: Изменение значения переменной

Также я вывела значение регистра edx в разных форматах: в шестнадцатеричном, в двоичном и в символьном.

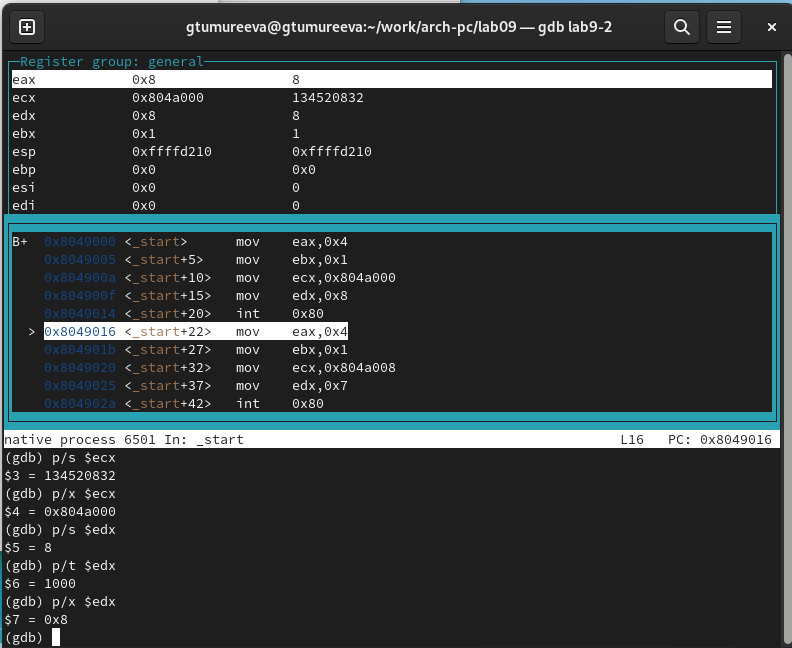


Figure 13: Вывод значения регистра

И далее я изменила значение регистра ebx, воспользовавшись командой set.

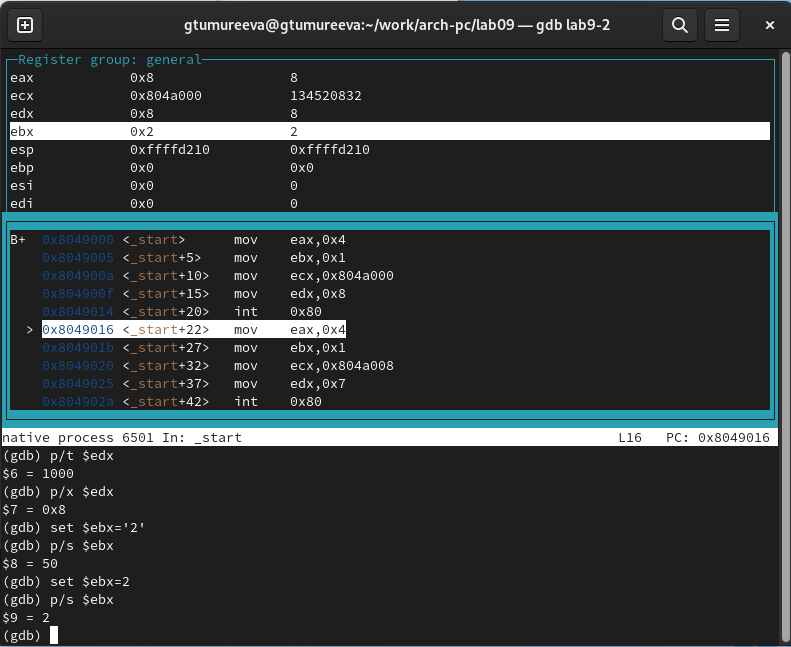


Figure 14: Вывод значения регистра

1. Я скопировала файл lab8-2.asm, который был создан в ходе выполнения восьмой лабораторной работы, содержащий программу для вывода аргументов командной строки на экран. Затем я сформировала из него исполняемый файл. Чтобы загрузить эту программу в отладчик gdb вместе с аргументами, мне понадобилось использовать ключ –args. После этого я успешно загрузила исполняемый файл в отладчик, не забыв указать необходимые аргументы.

Первым делом я установила точку останова до выполнения первой инструкции программы и запустила её.

Важно отметить, что адрес вершины стека находится в регистре esp, и именно по этому адресу расположено значение, показывающее количество аргументов командной строки, включая само имя программы. В моем случае, число аргументов составило пять: имя программы lab9-3 и четыре аргумента - аргумент1, аргумент2 и ‘аргумент 3’.

Я также исследовала другие значения в стеке: по адресу [esp+4] находится адрес в памяти, где расположено имя программы, по адресу [esp+8] - адрес первого аргумента, по адресу [esp+12] - второго, и так далее.

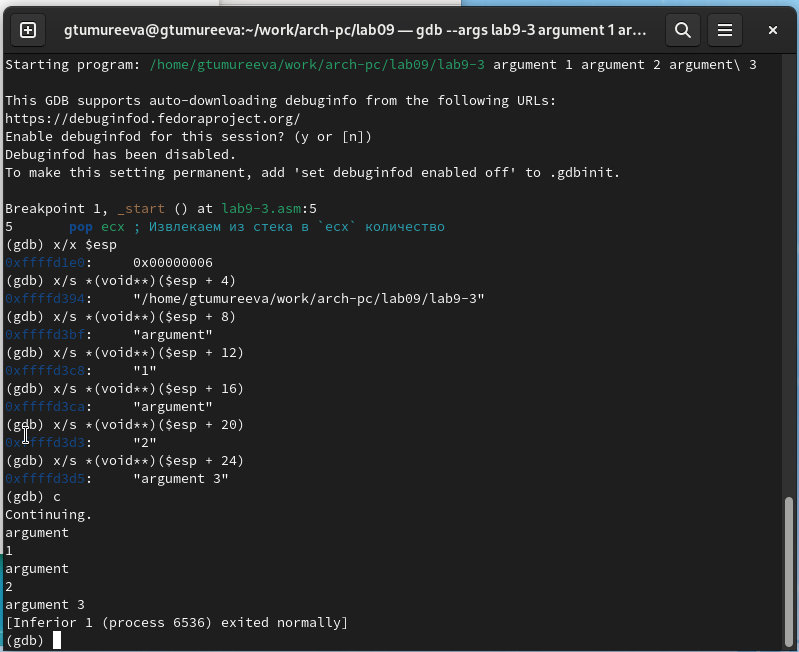


Figure 15: Вывод значения регистра

Шаг изменения адреса в стеке составляет 4 байта ([esp+4], [esp+8], [esp+12]). Это связано с тем, что размер каждой переменной, хранящейся в стеке, равен четырем байтам.

1. Я модифицировала программу из восьмой лабораторной работы (первое задание для индивидуального выполнения), включив в нее подпрограмму для расчета функции f(x).

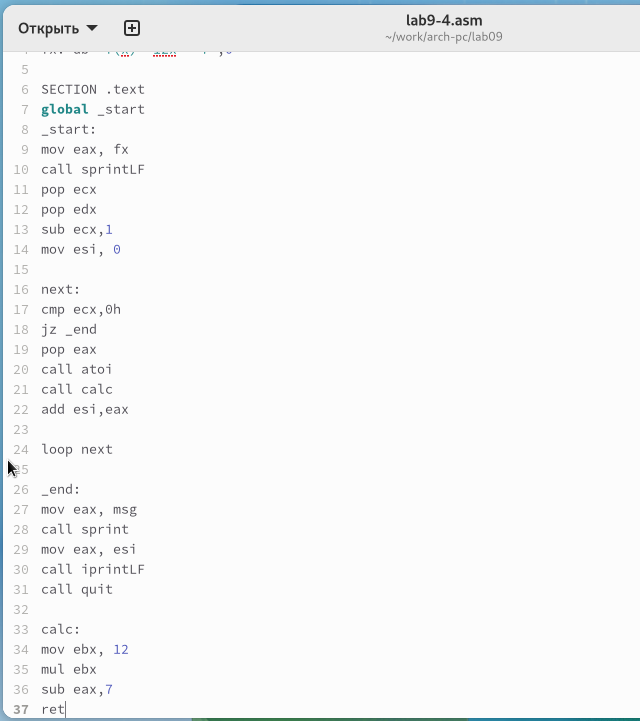


Figure 16: Программа в файле lab9-4.asm

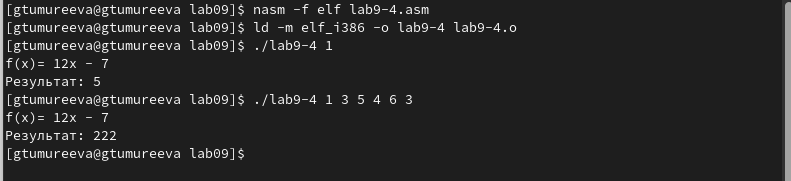


Figure 17: Запуск программы lab9-4.asm

1. В листинге приведена программа вычисления выражения . При запуске данная программа дает неверный результат. Проверил это. С помощью отладчика GDB, анализируя изменения значений регистров, определю ошибку и исправлю ее.

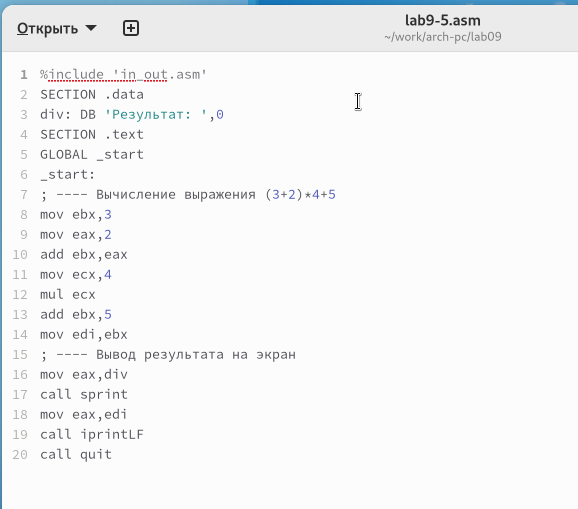


Figure 18: Код с ошибкой

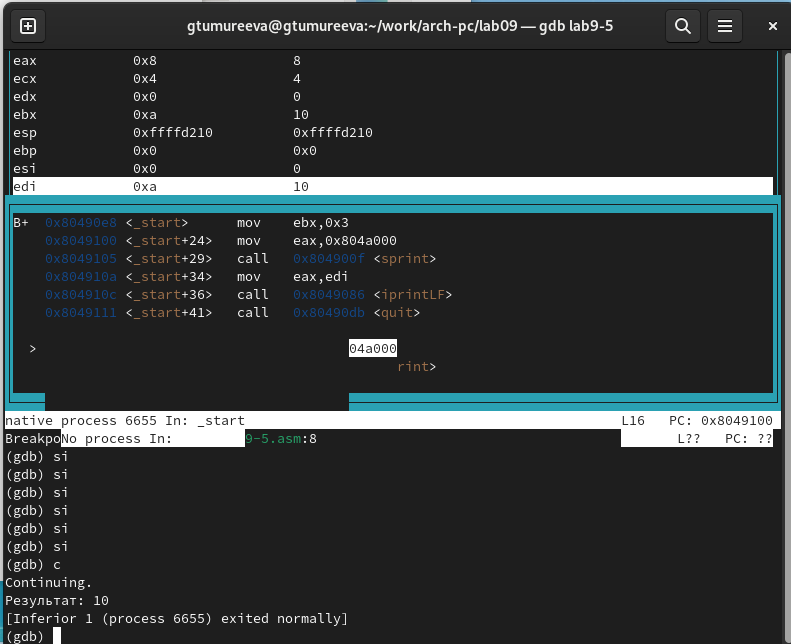


Figure 19: Отладка

Я обнаружила, что в инструкции add были перепутаны местами аргументы, и после завершения работы значение регистра ebx ошибочно передавалось в edi вместо ожидаемого eax. Эту ошибку мне предстоит исправить.

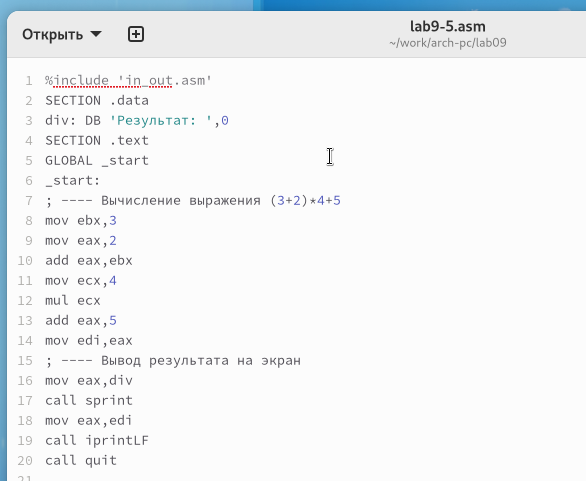


Figure 20: Код исправлен

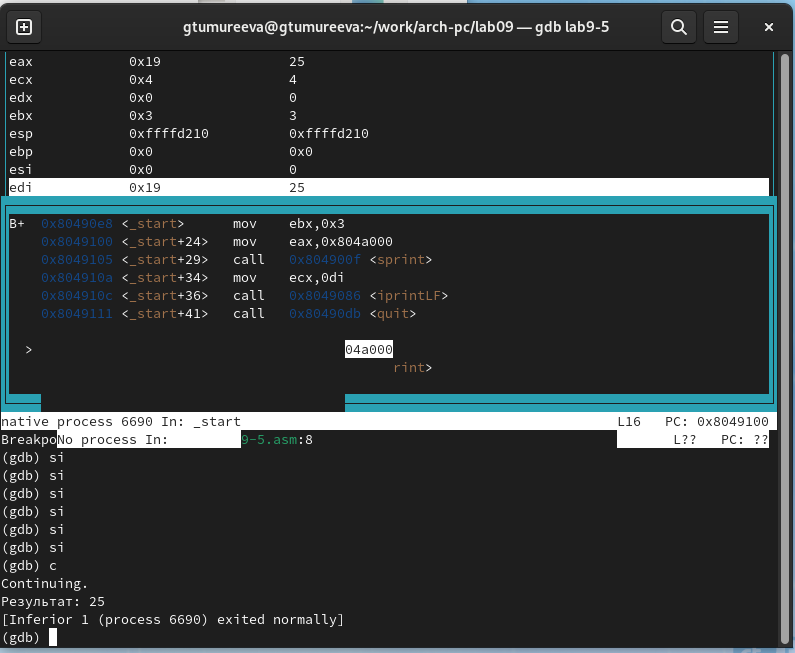


Figure 21: Проверка работы

# 3 Выводы

Освоили работy с подпрограммами и отладчиком.