Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Габралян Георгий Александрович

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Выполнение лабораторной работы

Создаём каталог для выполнения лабораторной работы номер 9, переходим в него и создаём файл lab09-1.asm. (рис. [1](#fig:001))

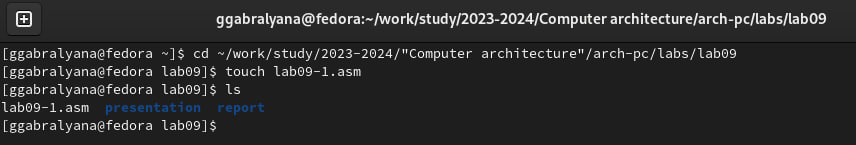


Figure 1: Создание файла lab09-1.asm

Рассматриваем программу вычисления арифметического выражения . Вводим текст этой программы в файл lab09-1.asm . (рис. [2](#fig:002))

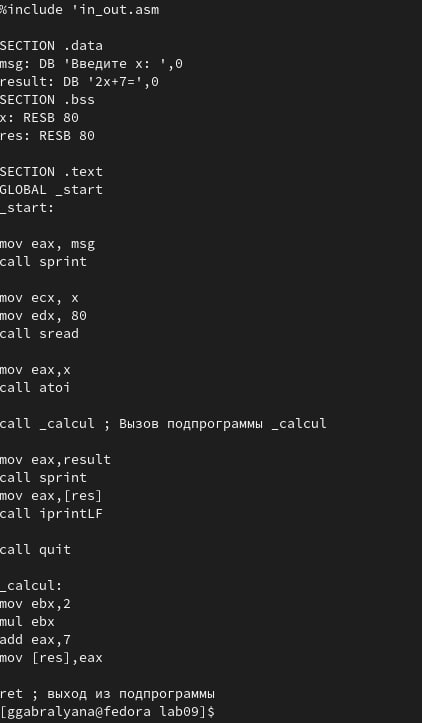


Figure 2: Текст lab09-1.asm

Создаём исполняемый файл и сразу проверяем его работу. (рис. [3](#fig:003))

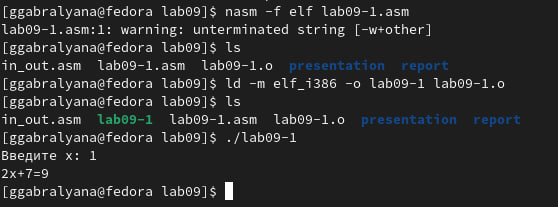


Figure 3: Создание и проверка работы файла lab09-1

Теперь изменяем текст программы, добавляя подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. (рис. [4](#fig:004))

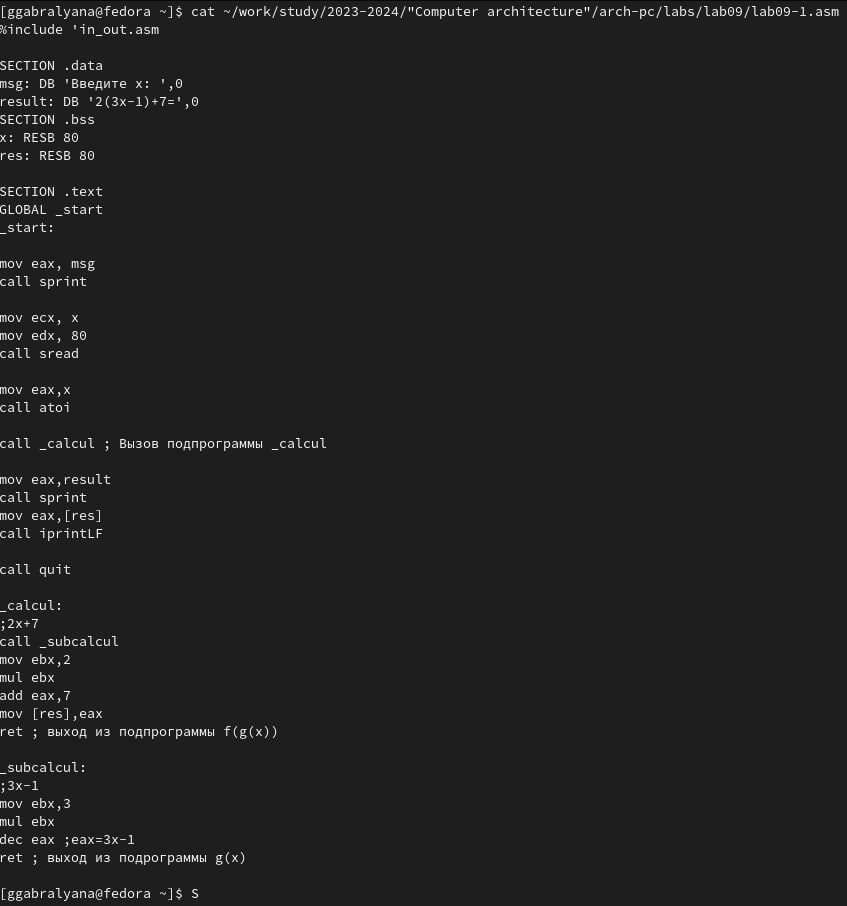


Figure 4: Изменённый текст lab09-1.asm

Создаём исполняемый файл, проверяем его работу. (рис. [5](#fig:005))

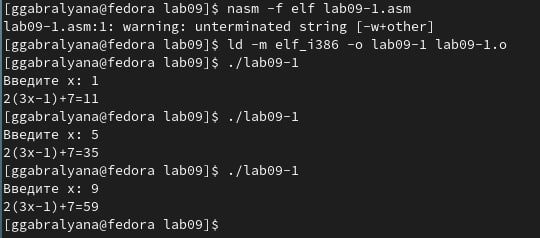


Figure 5: Создание и запуск изменённого файла lab09-1

Создаём файл lab09-2.asm, заполняем его текстом программы печати сообщения *Hello world!* (рис. [6](#fig:006))

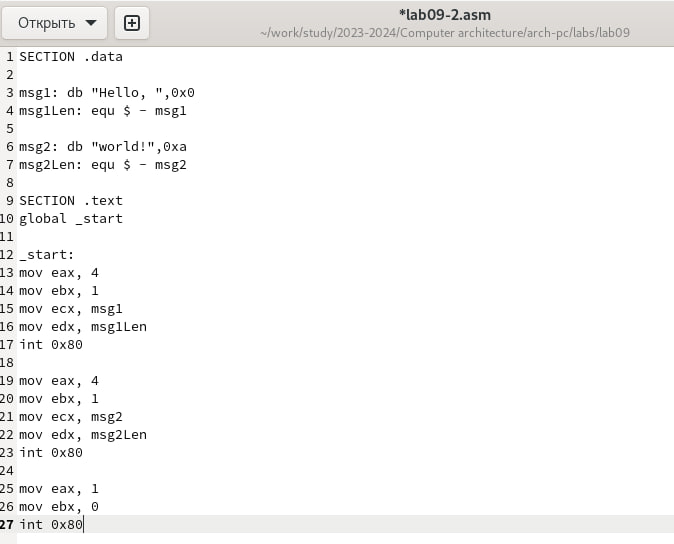


Figure 6: Текст lab09-2.asm

Получаем исполняемый файл. Для работы с GDB в исполняемый файл нужно добавить отладочную информацию, для этого используем ключ -g. (рис. [7](#fig:007))

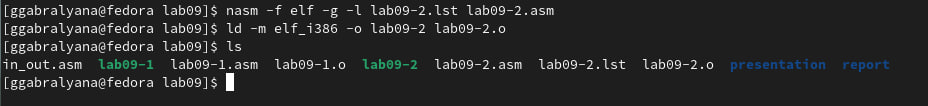


Figure 7: Создание исполняемого файла lab09-2 и файла листинга

Загружаем исполняемый файл в отладчик GDB. (рис. [8](#fig:008))

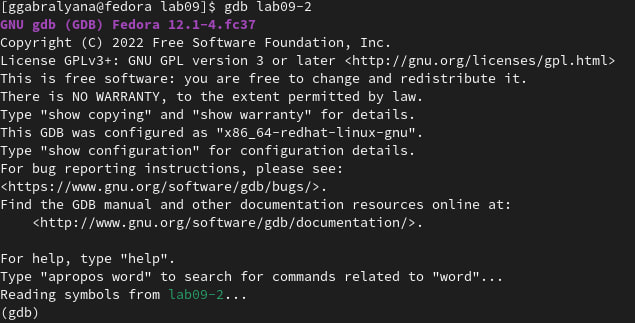


Figure 8: Отладчик GDB

Проверяем работу программы, запуская её в GDB с помощью команды run (рис. [9](#fig:009)).

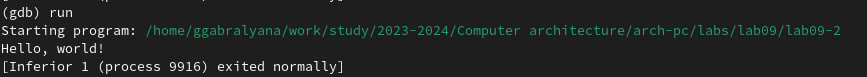


Figure 9: Запуск программы с помощью run

Для более детального анализа программы устанавливаем брейкпоинт на метку \_start, запускаем её. (рис. [10](#fig:010))

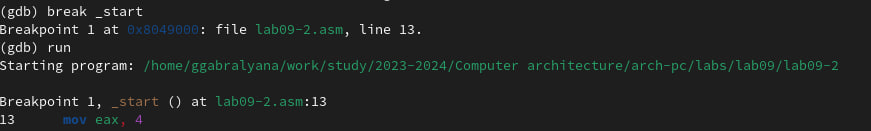


Figure 10: Установка брейкпоинта на метку \_start

Далее смотрим дисассимилированный код программы с помощью команды disassemble начиная с \_start. (рис. [11](#fig:011))



Figure 11: Дисассимилированный код программы

Переключаемся на отображение команд с синтаксисом Intel. (рис. [12](#fig:012))

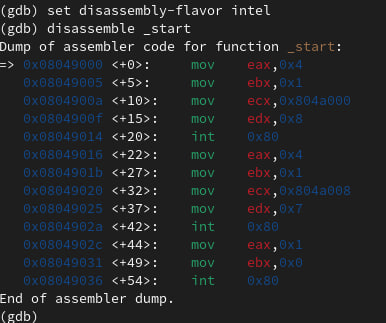


Figure 12: Синтаксис Intel

Как можно увидеть, в синтаксисе Intel после команды выводится регистр, куда помещается значение, и затем адрес помещаемого значения или число. В свою очередь, в синтаксисе ATT сначала выводится ссылка на заносимое в регистр значение, и только после неё регистр с символом %.

Включаем режим псевдографики для более удобного анализа программы с помощью layout asm и layout regs (рис. [13](#fig:013))

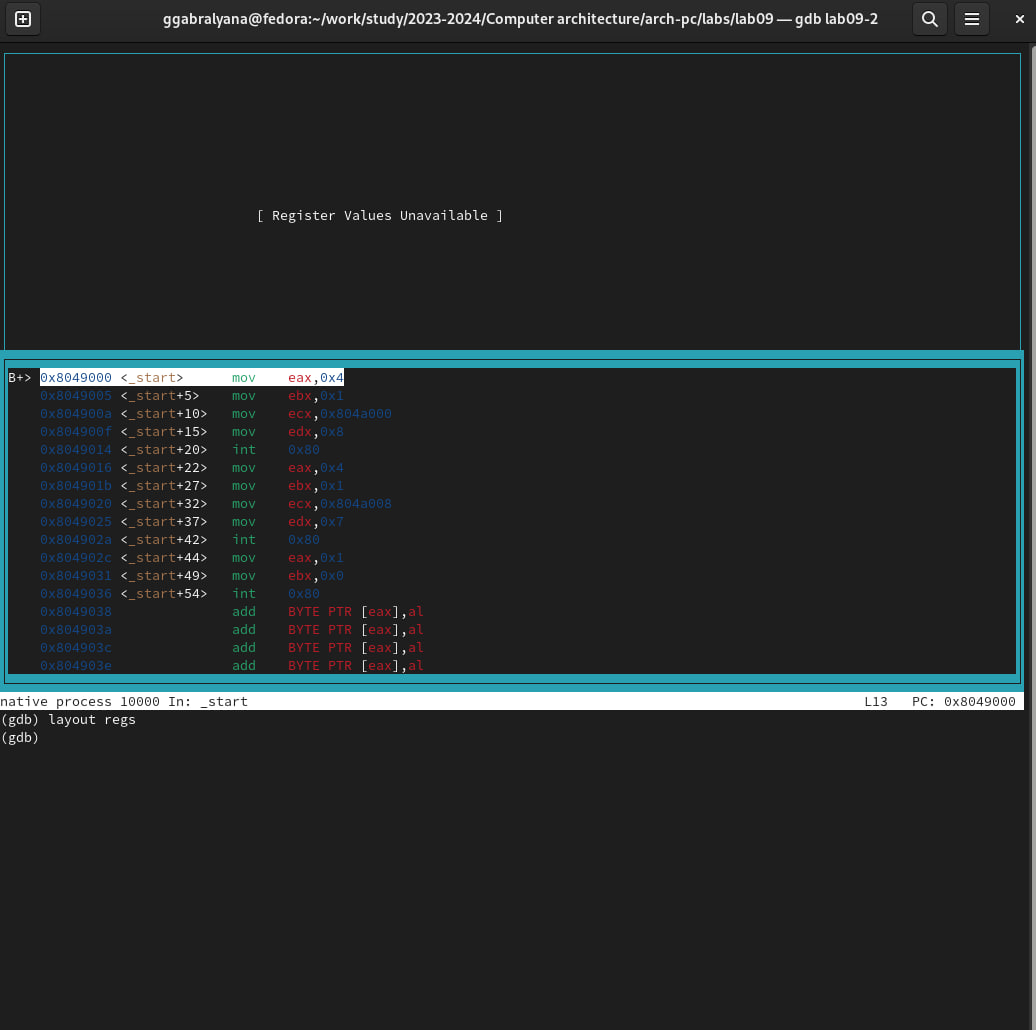


Figure 13: Режим псевдографики

Ранее была установлена точка останова по имени метки \_start. Проверяем это с помощью команды info breakpoints (рис. [14](#fig:014))

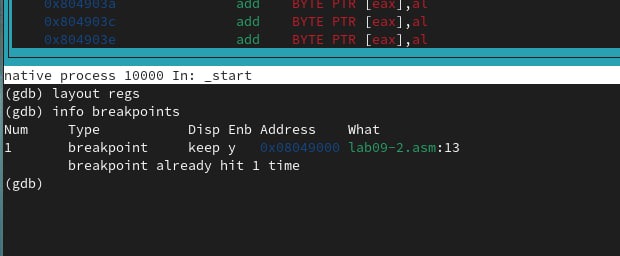


Figure 14: Проверка точки останова

Устанавливаем еще одну точку останова по адресу инструкции mov ebx,0x0. В нашем случае это адрес: 0x8049031. Затем смотрим информацию о всех точках останова (рис. [15](#fig:015)).

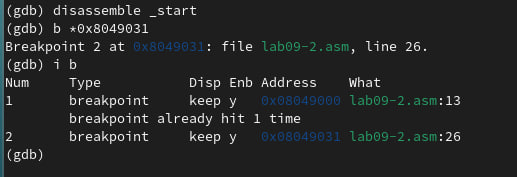


Figure 15: Установка точки останова по адресу

Выполняем 5 инструкций с помощью команды si и следим за изменениями в значениях регистров (рис. [16](#fig:016) - [20](#fig:020)).

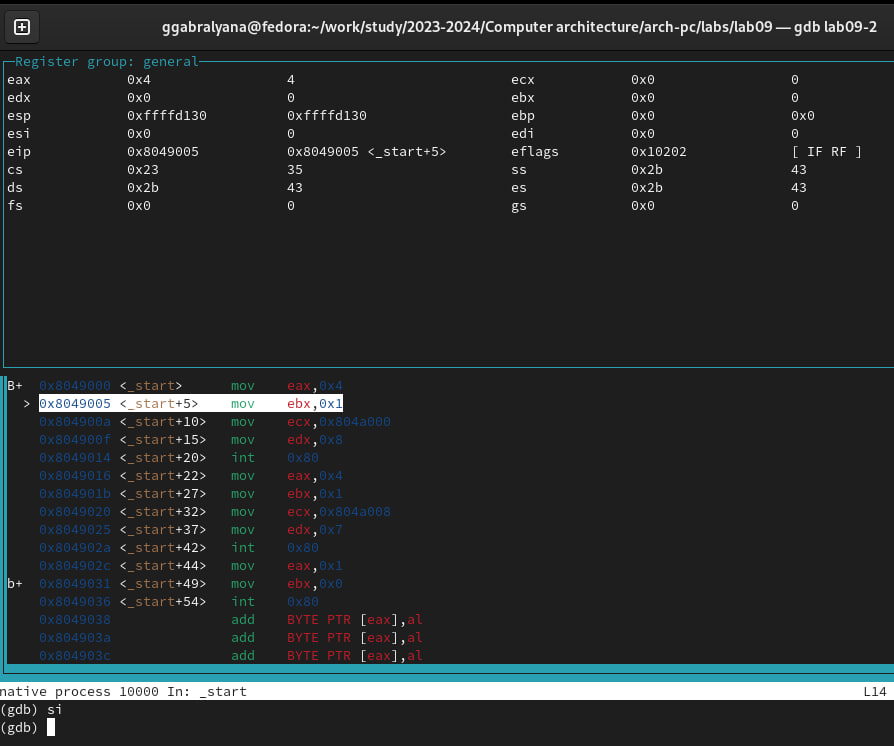


Figure 16: Инструкция stepi(1)

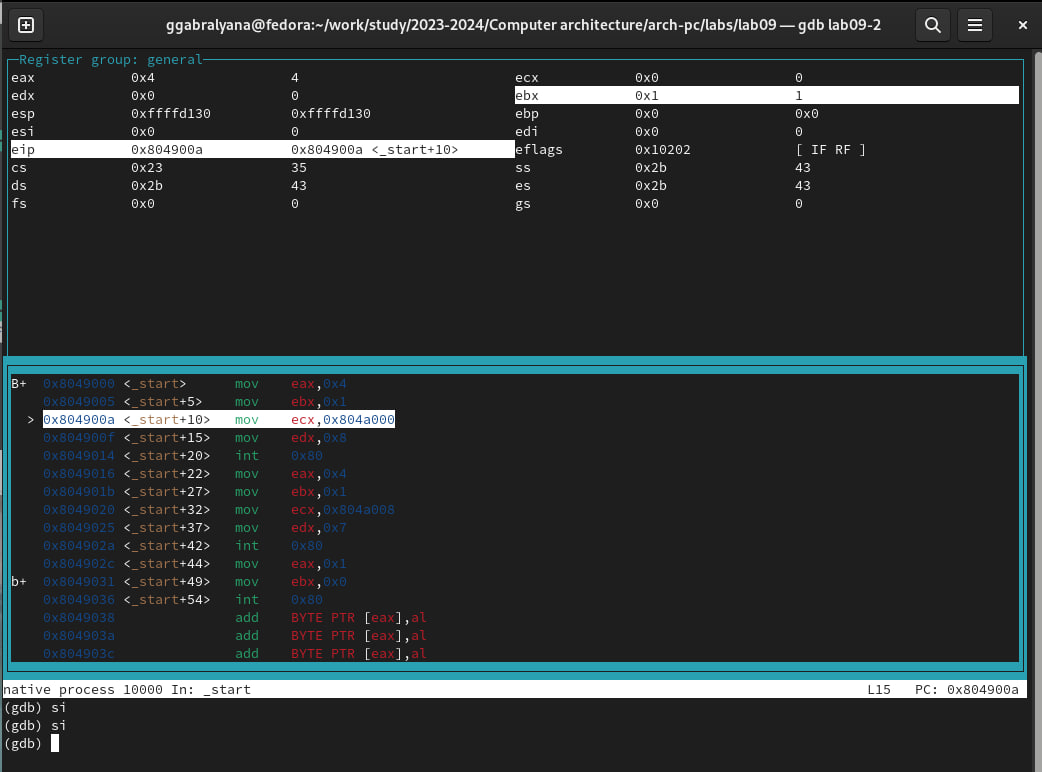


Figure 17: Инструкция stepi(2)

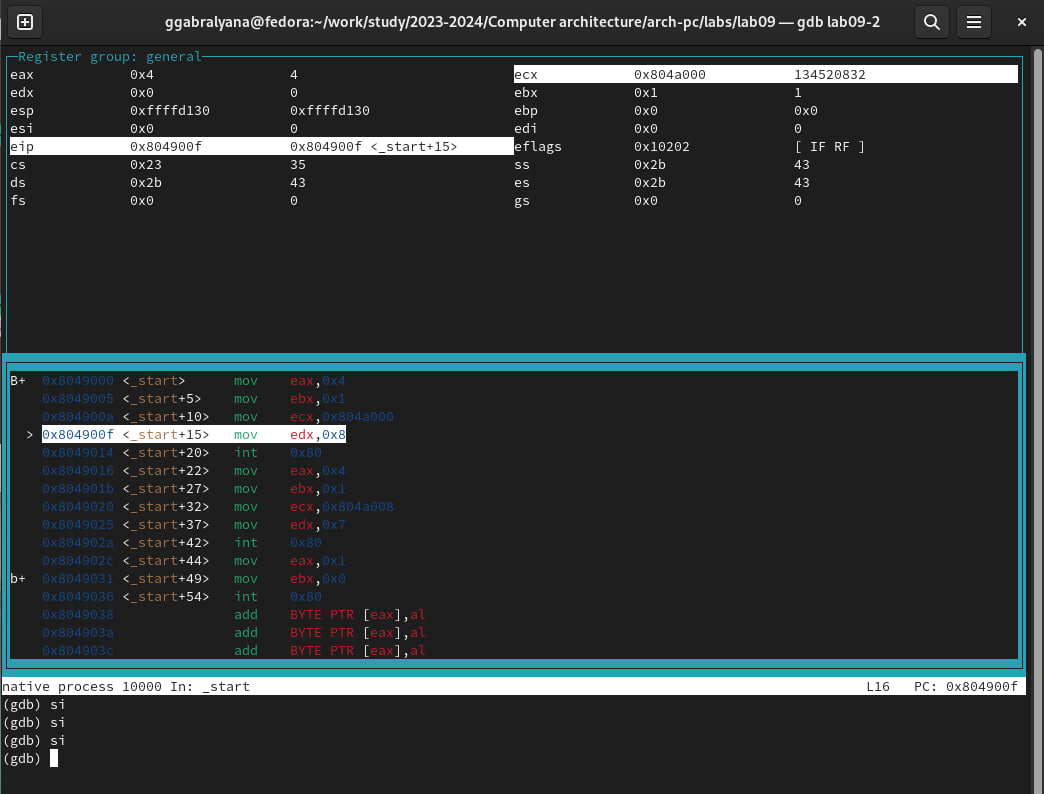


Figure 18: Инструкция stepi(3)

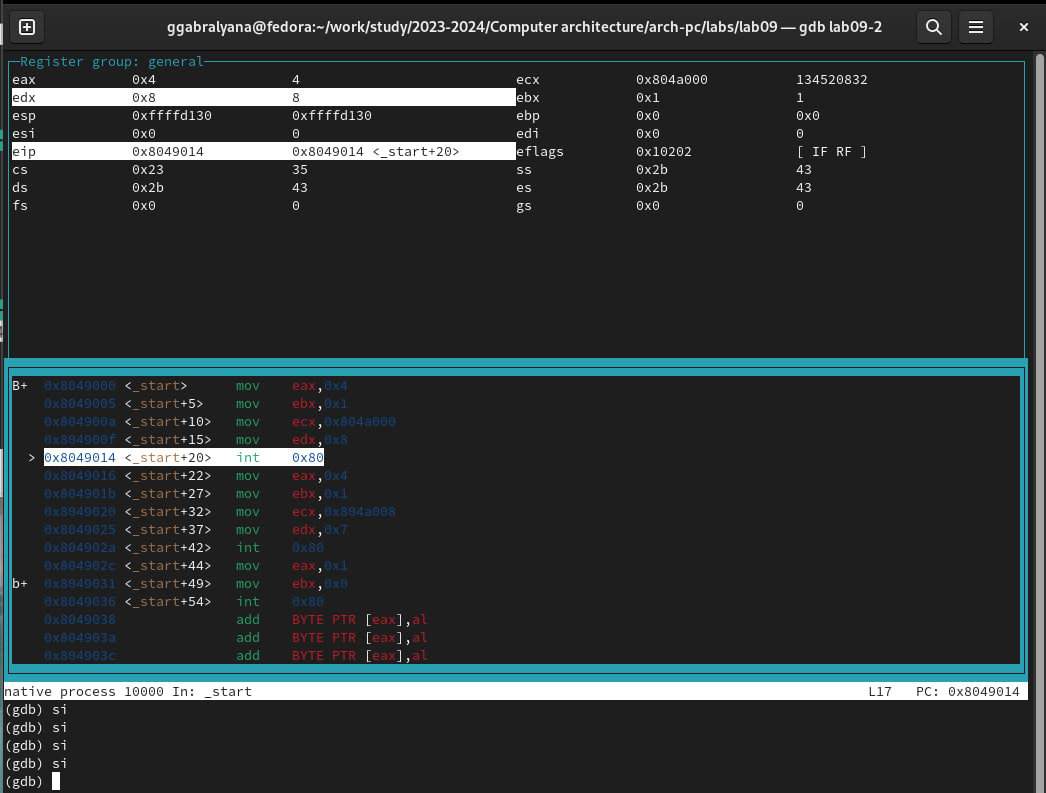


Figure 19: Инструкция stepi(4)

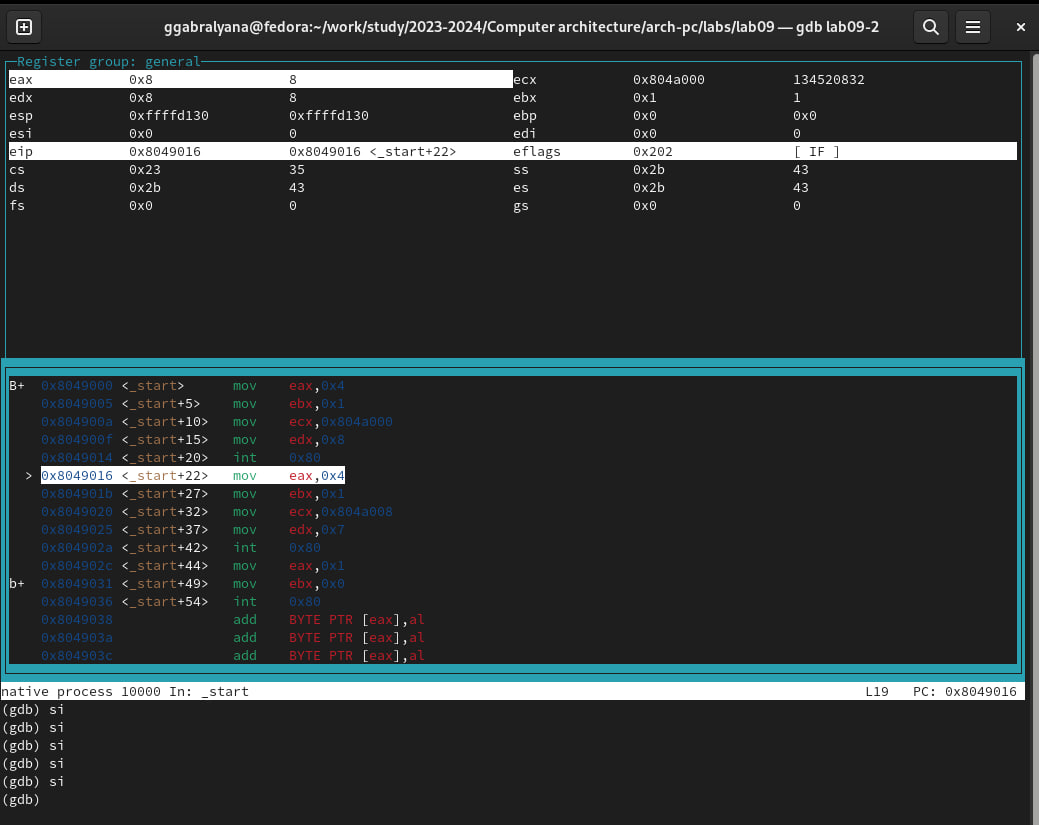


Figure 20: Инструкция stepi(5)

Как видно на верхней панели, изменились значения регистров eax, ecx, ebx и edx.

Теперь смотрим значение переменной msg1 по имени (рис. [21](#fig:021)).

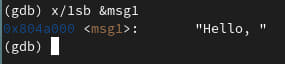


Figure 21: Значение переменной msg1 по имени

Выводится строка “Hello,”.

Теперь смотрим значение переменной msg2 по адресу. Адрес переменной определяем по дисассемблированной инструкции. (рис. [22](#fig:022)).

Figure 22: Значение переменной msg2 по адресу

Figure 22: Значение переменной msg2 по адресу

Далее изменяем первый символ в переменных msg1 и msg2(рис. [23](#fig:023)).

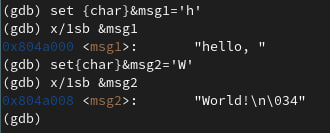


Figure 23: Изменение символов в переменных msg1 и msg2

Выводим в различных форматах значение регистра edx(рис. [24](#fig:024)).

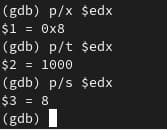


Figure 24: Вывод значения регистра edx

И теперь, используя команду set, изменяем значение регистра ebx (рис. [25](#fig:025)).

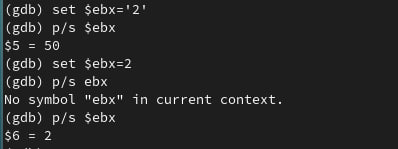


Figure 25: Изменение значения регистра ebx

Сначала мы занесли в регистр ebx символ ‘2’, вот почему после запроса p/s на вывод значения регистра на экране мы увидели код символа “2”, то есть 50. А затем, когда число 2 изначально было занесено в регистр, команда p/s $ebx вывела значение 2.

Завершаем выполнение программы с помощью c (рис. [26](#fig:026)).

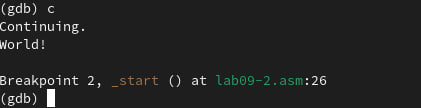


Figure 26: Завершение выполнения программы

И сразу после этого с помощью команды quit выходим из отладчика gdb.

Копируем файл lab8-2.asm, который был создан при выполнении лабораторной работы 8, в файл с именем lab09-3.asm и создаём исполняемый файл с ключом -g и файлом листинга (рис. [27](#fig:027)).

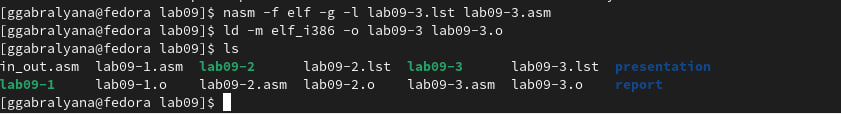


Figure 27: Создание исполняемого файла lab09-3

Для того чтобы загрузить в GDB программы с аргументами нужно использовать ключ --args. Загружаем файл в отладчик, указавыя аргументы (рис. [28](#fig:028)).

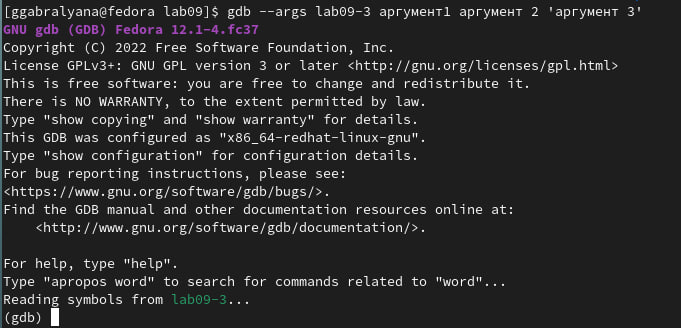


Figure 28: Загрузка программы lab09-3 в отладчике gdb

Сначала устанавливаем точку останова перед первой инструкцией в программе, запускаем её (рис. [29](#fig:029)).



Figure 29: Установка точки останова и запуск программы

Адрес вершины стека хранится в регистре esp и по нему располагается число, которое равно количеству аргументов командной строки (рис. [30](#fig:030)).

Figure 30: Содержимое регистра esp

Figure 30: Содержимое регистра esp

Число аргументов равно 5, то есть имя программы lab09-3 и сами аргументы: аргумент1, аргумент, 2 и аргумент 3.

Просматриваем остальные позиции (рис. [31](#fig:031)).

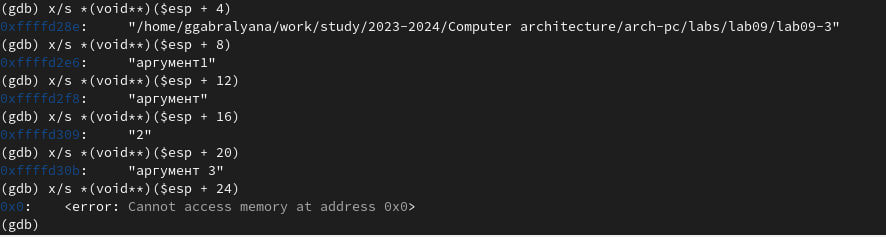


Figure 31: Содержимое стека

По адесу [esp+4] располагается адрес в памяти, где находиться имя программы. По адесу [esp+8] храниться адрес первого аргумента. По аресу [esp+12] – второго и так далее. Как можно заметить, шаг изменения равен 4. Шаг имеет такое значение, так как при добавлении значения каждого аргумента в стек, значение регистра esp увеличивается на 4.

# 3 Выполнение заданий для самостоятельной работы

**Задание №1**

Преобразовываем программу из лабораторной работы 8, реализовывая вычисление значения функции как подпрограмму.

В лабораторной работе 8 мы реализовывали вычисление в цикле следующей функции:

Теперь для вычисления значения этой функции в цикле мы вызываем вспомогательную подпрограмму. Создаём файл f1.asm и вписываем в него текст программы (рис. [32](#fig:032)).

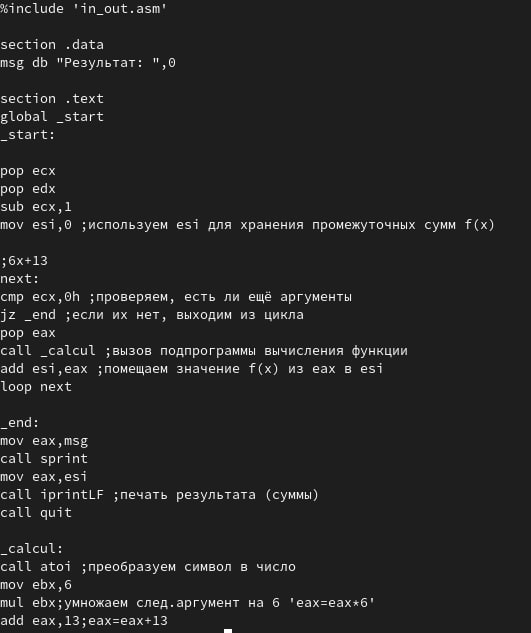


Figure 32: Текст f1.asm

Создаём исполняемый файл, проверяем его работу с теми же аргументами (рис. [33](#fig:033)).

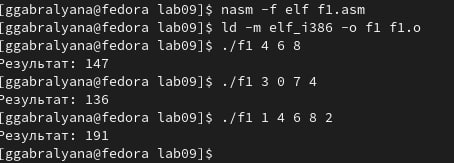


Figure 33: Создание и запуск исполняемого файла f1

Как видно, программа работает исправно.

**Задание №2**

Создаём файл f2.asm и вписываем в него текст программы вычисления выражения . Затем создаём исполняемый файл и запускаем его (рис. [34](#fig:034)).

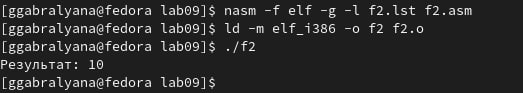


Figure 34: Запуск исполняемого файла f2

Как можно увидеть, при запуске программа действительно даёт неверный результат.

С помощью отладчика GDB анализируем изменения значений регистров, определяем ошибку и исправляем её.

Для начала загрузим исполняемый файл в отладчик (рис. [35](#fig:035)).

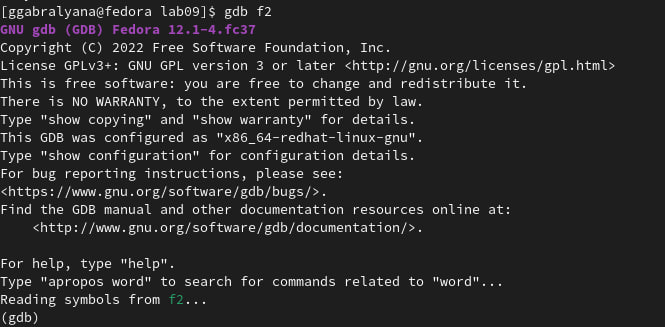


Figure 35: Закрузка файла f2 в отладчик gdb

Смотрим дисассимилированный код программы, начиная с метки \_start (рис. [36](#fig:036)).

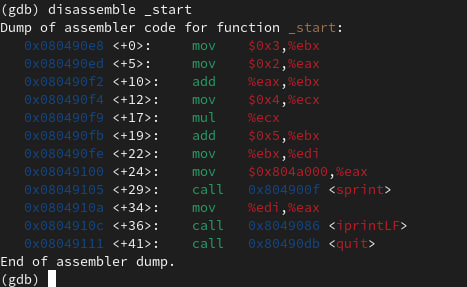


Figure 36: Дисассимилированный код f2

Переходим в режим псевдографики (рис. [37](#fig:037)).

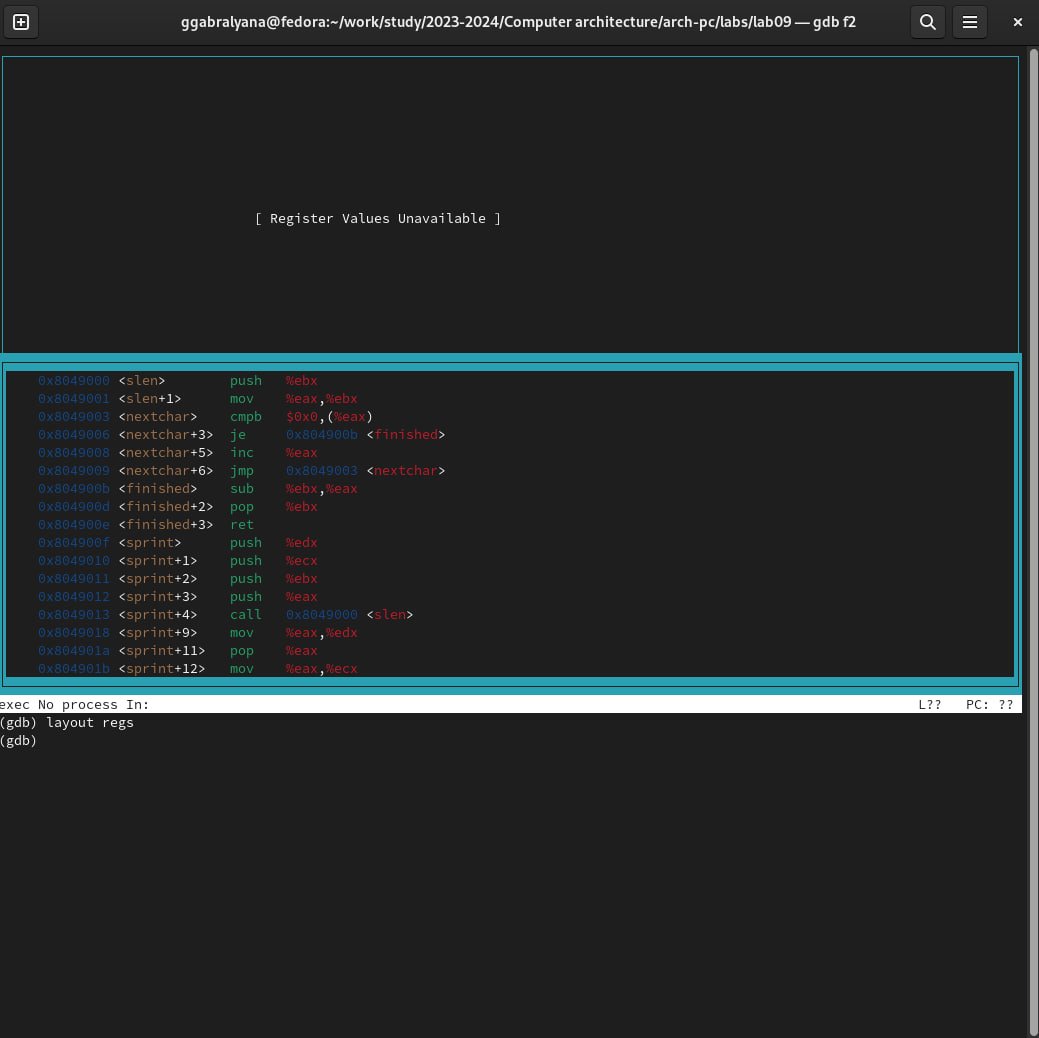


Figure 37: Режим псевдографики f2

Просматриваем значения регистров (рис. [38](#fig:038)).

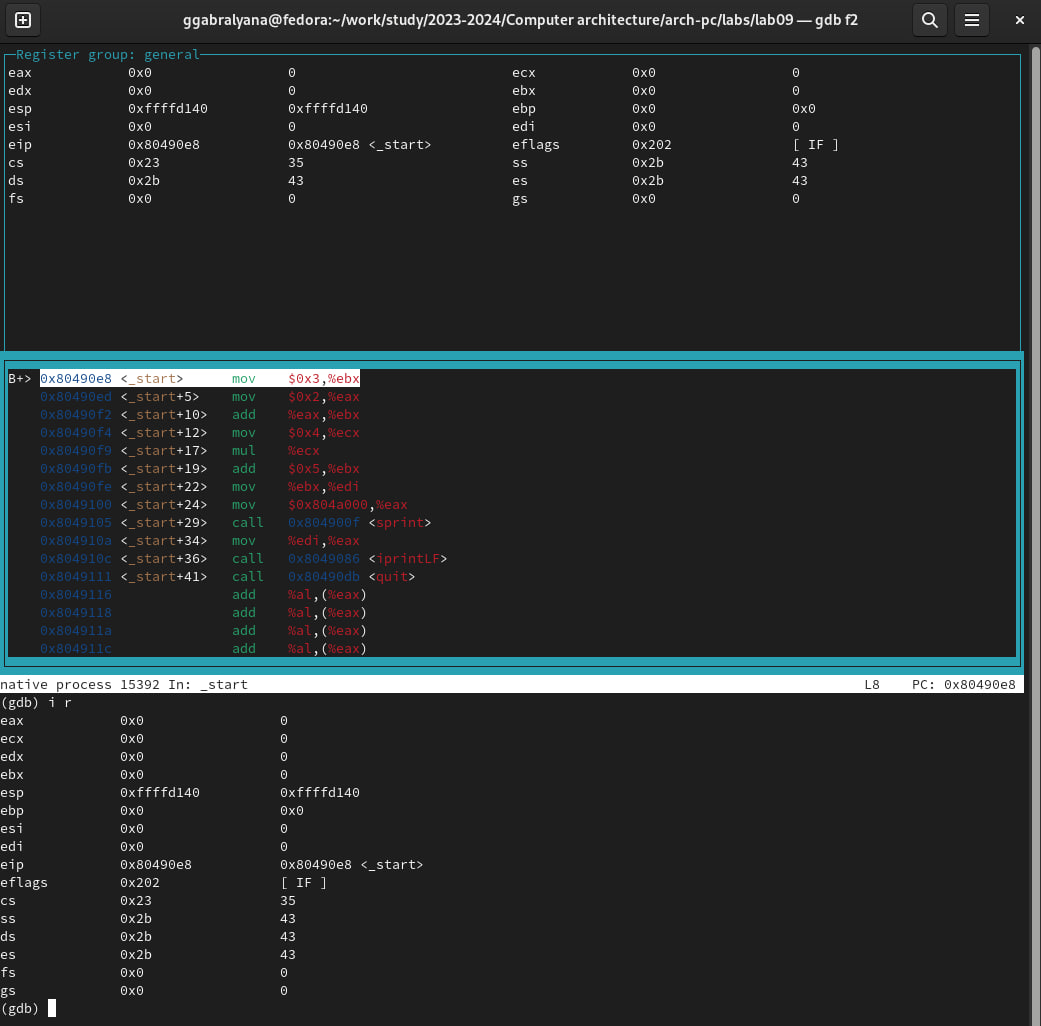


Figure 38: Просмотр значений регистров f2

Воспользовавшись командой break устанавливаем точку останова по адресу на инструкции add ebx,eax (рис. [39](#fig:039)).

Figure 39: Точка останова на инструкции add

Figure 39: Точка останова на инструкции add

Смотрим значения регистров с помощью команды i r (рис. [40](#fig:040)).

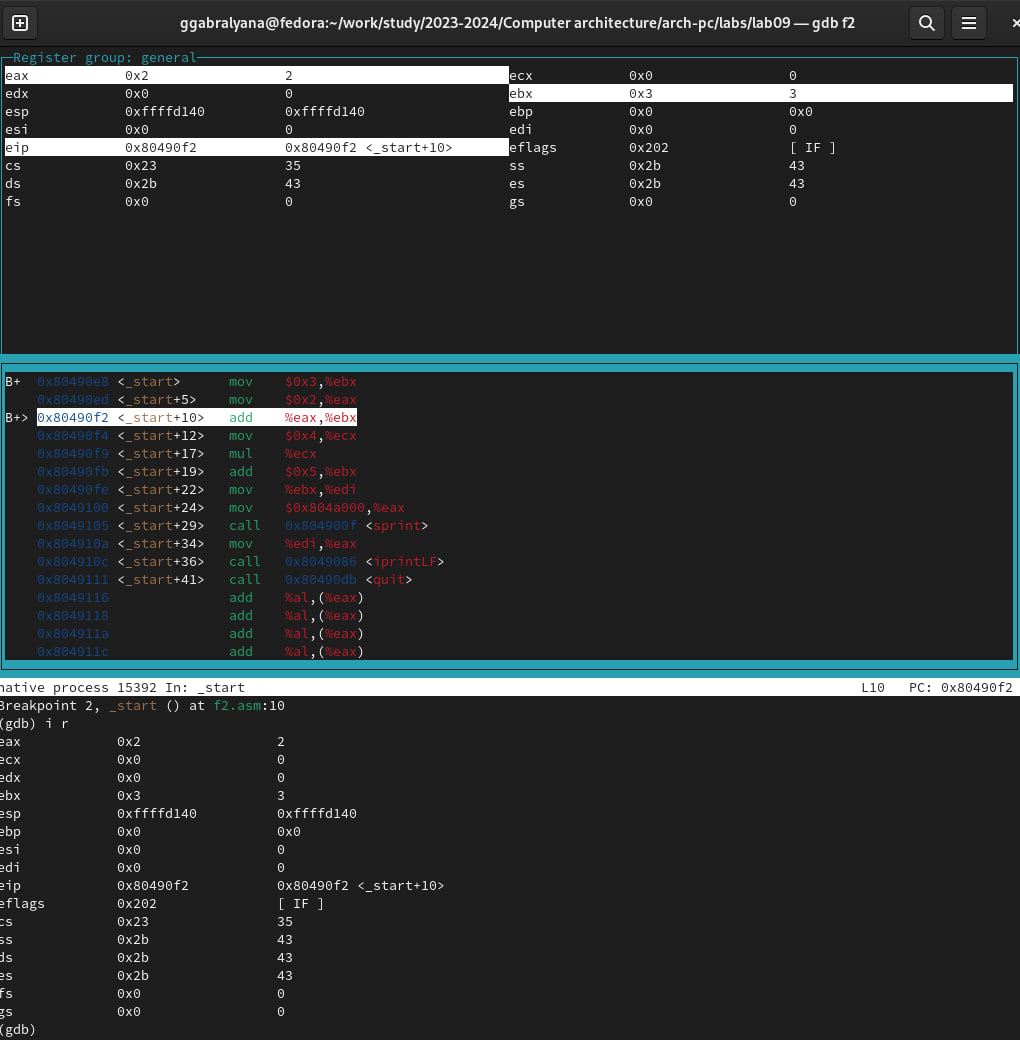


Figure 40: Просмотр значений регистров

Затем с помощью si переходим к следующей инструкции и следим за изменениями значений регистров (рис. [41](#fig:041)).

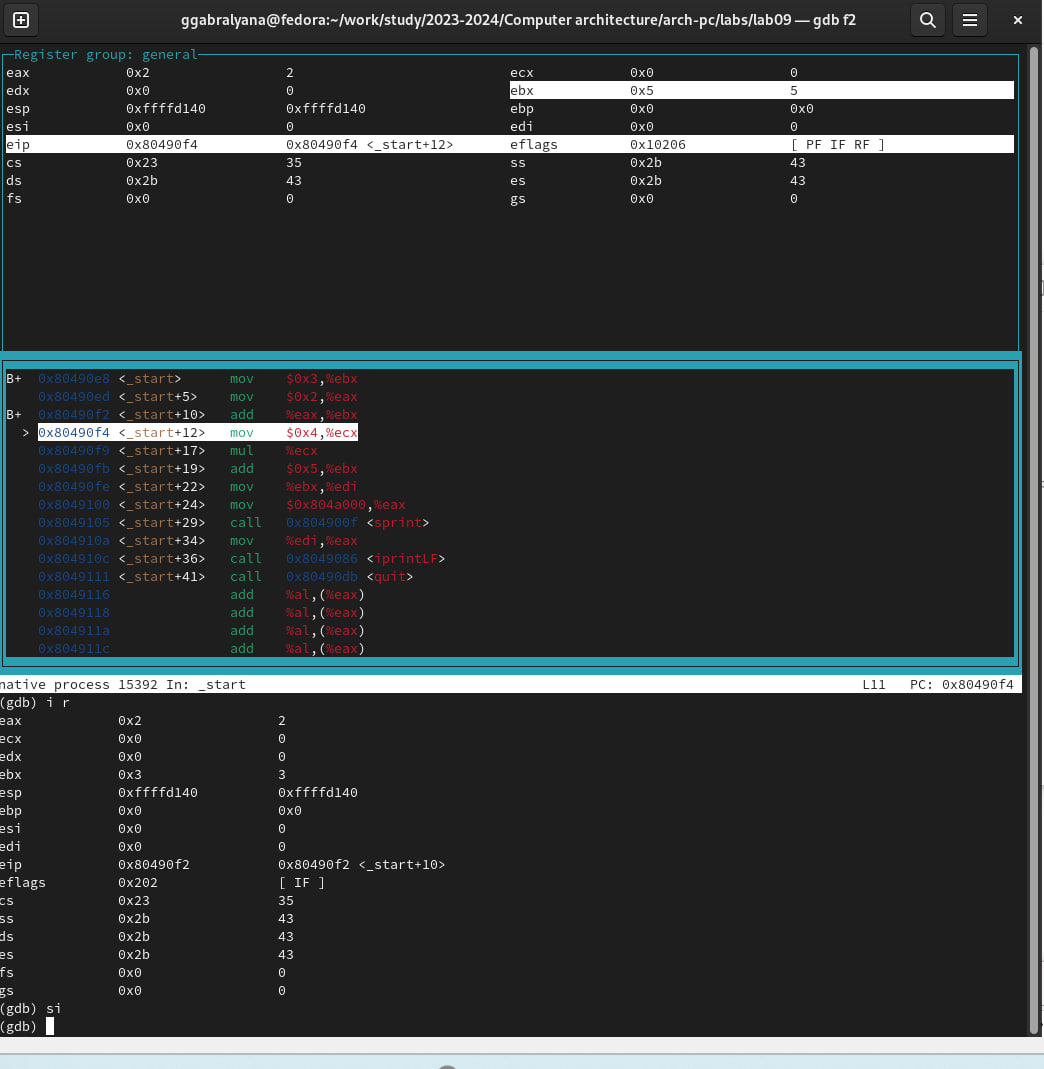


Figure 41: si

Результат суммы чисел и записался в регистр ebx. Это и могло послужить проблемой для дальнейшего вычисления произведения. Исправляем это, изменяя значение регистра eax и занеся в него значение с помощью set (рис. [42](#fig:042)).

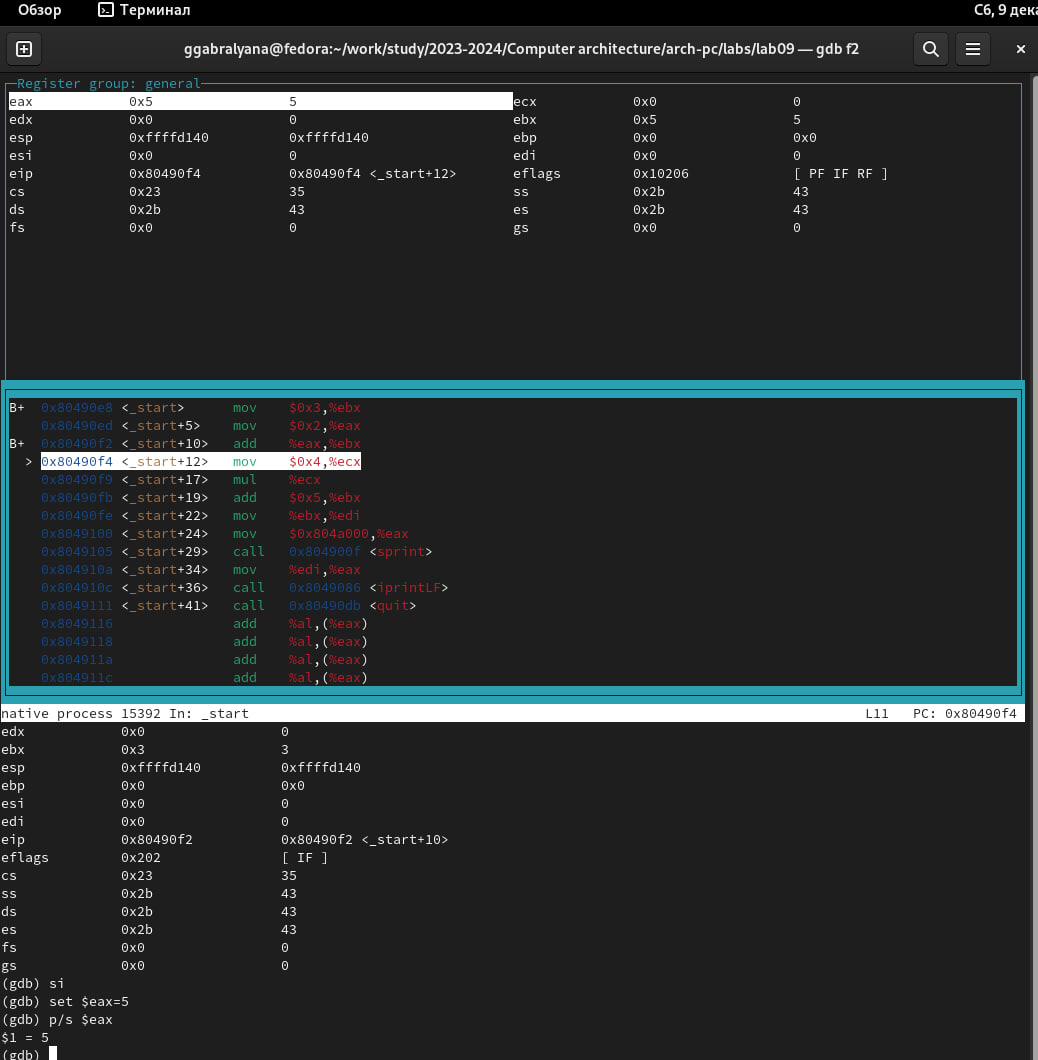


Figure 42: Изменение значения регистра eax

Далее переходим к следующей инструкции (рис. [43](#fig:043)).

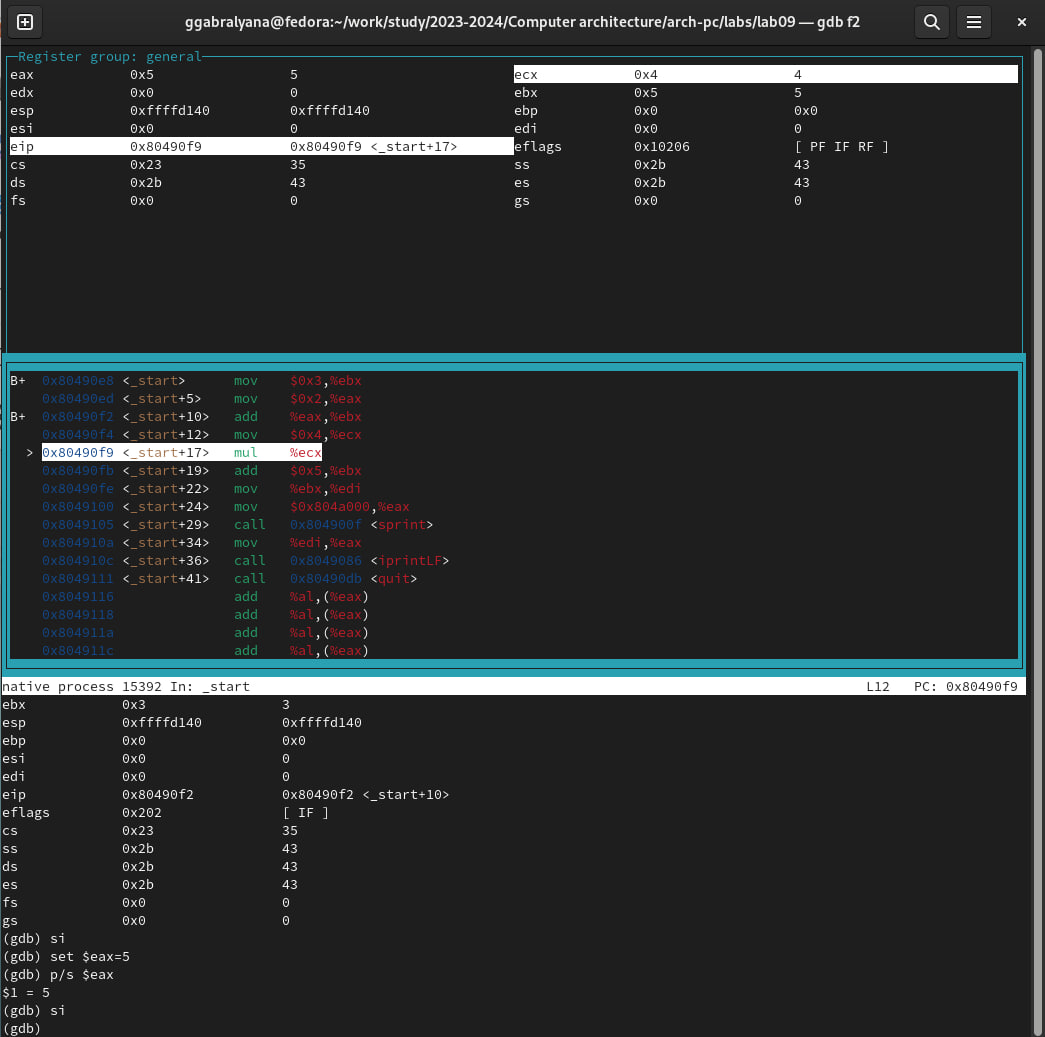


Figure 43: si

Мы поместили в регистр ecx значение для вычисления произведения. После произведения значения регистров будут следующими: (рис. [44](#fig:044)).

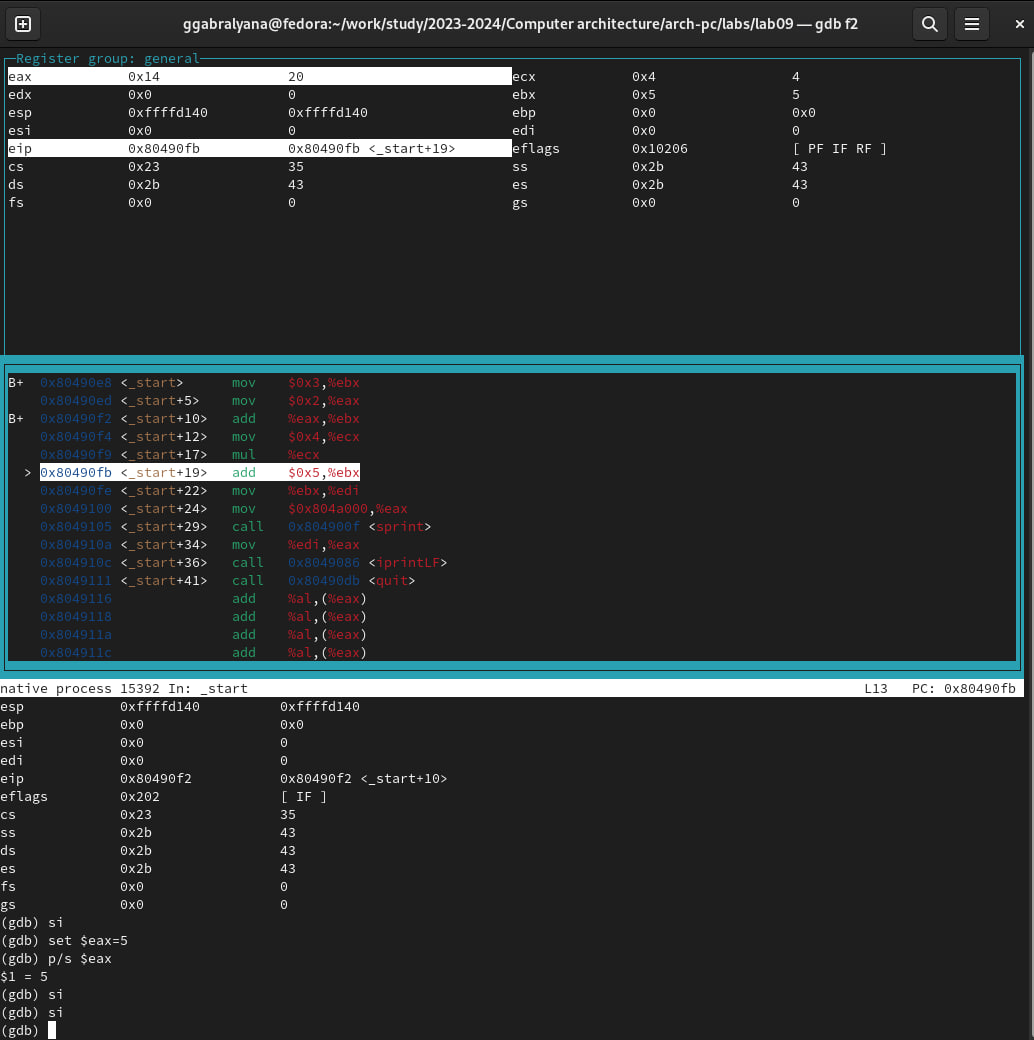


Figure 44: Результат вычисления

В результате в регистр eax было помещено значение произведения .

К этому значению нужно прибавить . В программе за результат отвечает регистр ebx. Поместим в него значение и запустим программу на вывод конечного результата(рис. [45](#fig:045)).

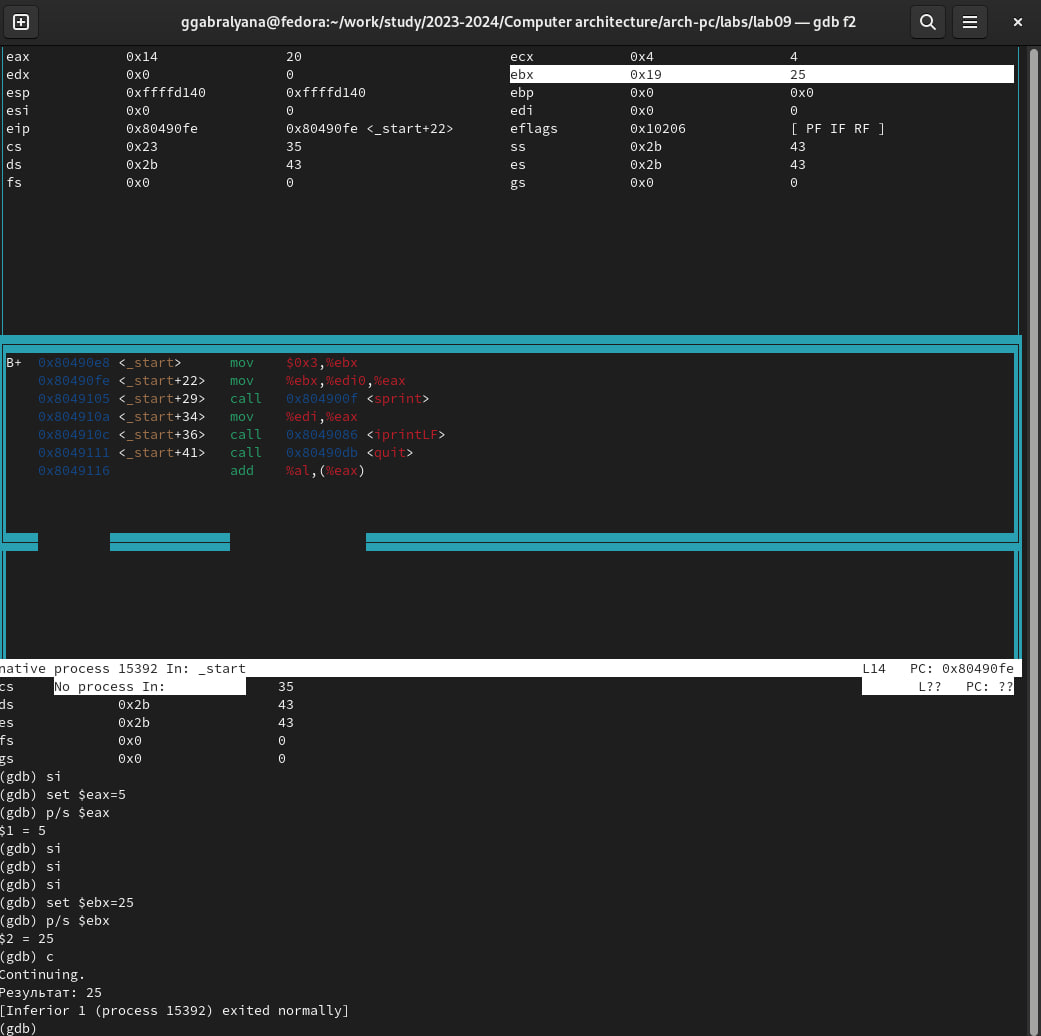


Figure 45: Завершение выполнения программы

Как видно, с учётом всех изменений программа выдаёт верный результат. Теперь изменяем код программы в файле f2.asm (рис. [46](#fig:046)).

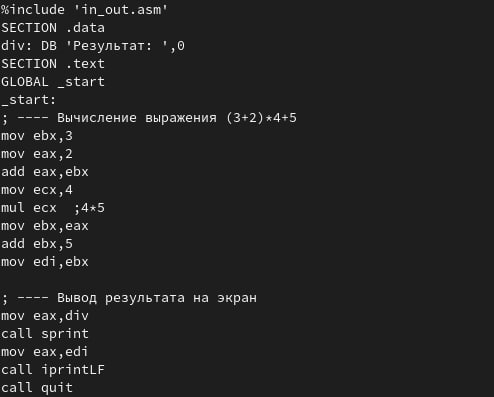


Figure 46: Исправленный текст программы в f2.asm

Теперь создаём исполняемый файл и проверяем корректность работы программы (рис. [47](#fig:047)).

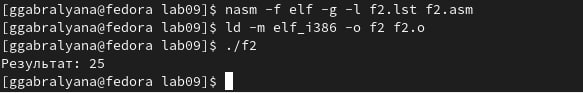


Figure 47: Проверка работы программы func2

Программа выдаёт верный результат.

# 4 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм. Также мы познакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,