Отчёт по лабораторной работе № 10

Архитектура компьютера

Рыжов Егор

Содержание

# 1 Цель работы

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

# 2 Задание

1. Реализовать подпрограммы в NASM.
2. Выполнить отладку программам с помощью GDB.
3. Отработать добавление точек останова.
4. Поработа с данными программы в GDB.
5. Отработать обработку аргументов командной строки в GDB.
6. Выполнить задание для самостоятельной работы.

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

1. Создали каталог для выполнения лабораторной работы № 10, перешли в него и создали файл lab10-1.asm: (рис. 1)

Рис. 1: .

Рис. 1: .

1. В качестве примера рассмотрели программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучили текст программы (Листинг 10.1).

Введите в файл lab10-1.asm текст программы из листинга 10.1. (рис. 2) Создайте исполняемый файл и проверьте его работу.(рис. 3)

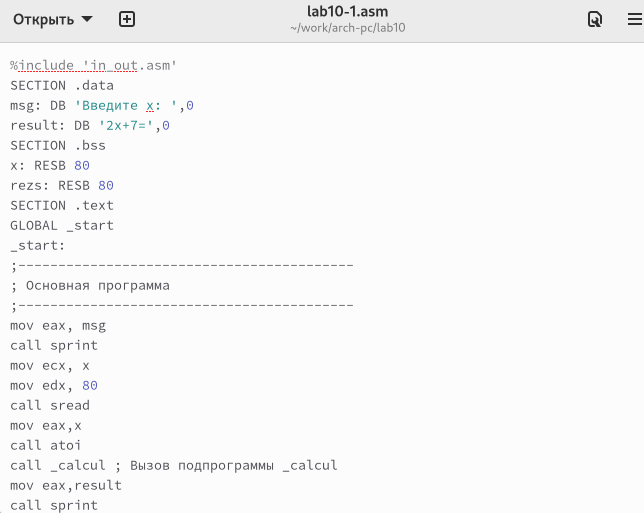


Рис. 2: .

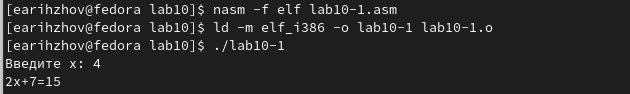


Рис. 3: .

Изменили текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. Т.е. x передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение g(x), результат возвращается в \_calcul и вычисляется выражение f(g(x)). Результат возвращается в основную программу для вывода результата на экран. (рис. 4), (рис. 5)

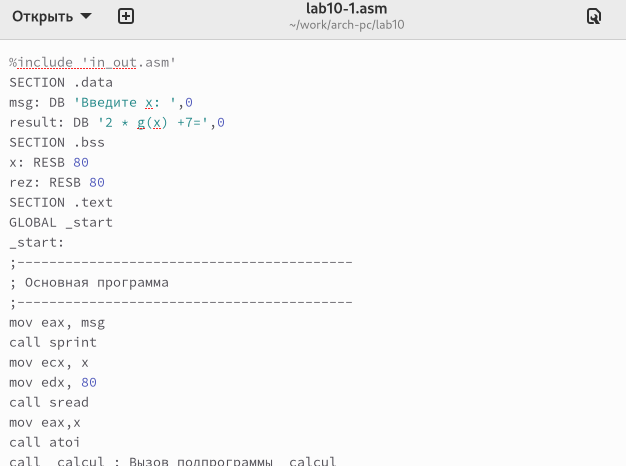


Рис. 4: .

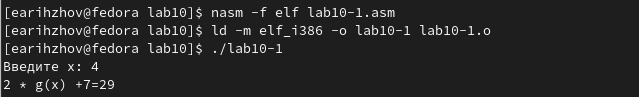


Рис. 5: .

## 3.2 Отладка программам с помощью GDB

Создали файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 6), (рис. 7)

Рис. 6: .

Рис. 6: .

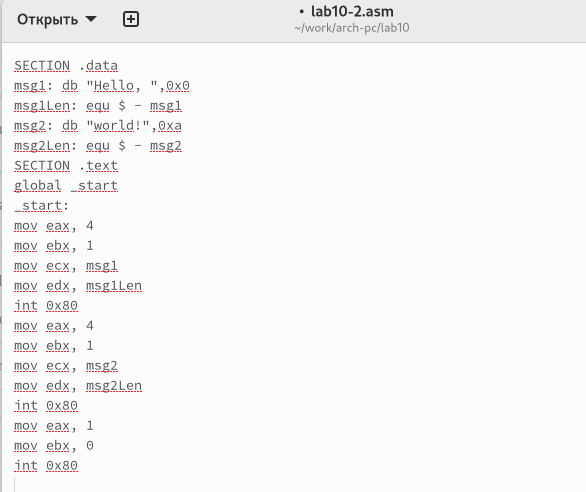


Рис. 7: .

Получили исполняемый файл. Для работы с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом ‘-g’.Загрузили исполняемый файл в отладчик gdb.(рис. 8)

Рис. 8: .

Рис. 8: .

Загрузили исполняемый файл в отладчик gdb. Проверили работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r):(рис. 9)

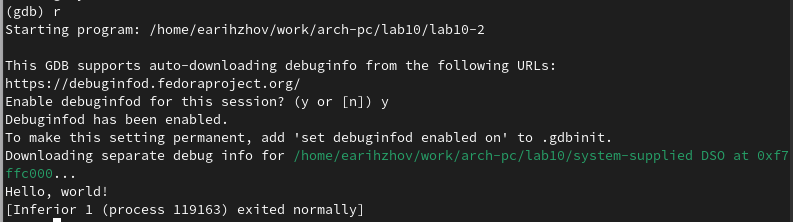


Рис. 9: .

Для более подробного анализа программы установили брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустили её.(рис. 10)

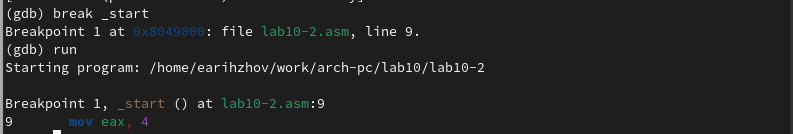


Рис. 10: .

Посмотрели дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. (рис. 11)

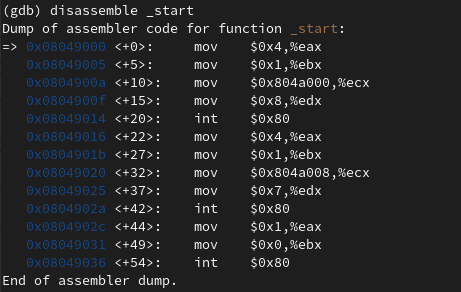


Рис. 11: .

Переключились на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. (рис. 12)

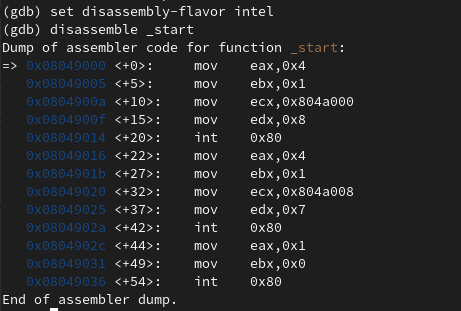


Рис. 12: .

Различия отображения синтаксиса машинных команд в режимах ATT и Intel: в АТТ перед адресом регистра ставится $, а перед названием регистра %, сначала записывается адрес, а потом - регистр. В Intel сначала регистр, а потом адрес, и перед ними ничего не ставится.

Включили режим псевдографики для более удобного анализа программы.(рис. 13)

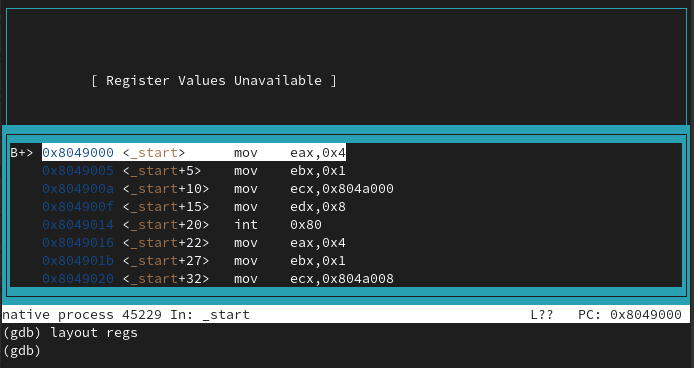


Рис. 13: .

## 3.3 Добавление точек останова

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверили это с помощью команды info breakpoints (кратко i b). (рис. 14)

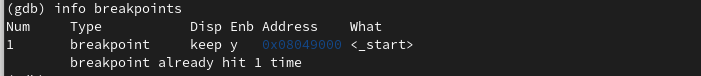


Рис. 14: .

Установили еще одну точку останова по адресу инструкции. Адрес инструкции увидели в средней части экрана в левом столбце соответствующей инструкции. Определили адрес предпоследней инструкции (mov ebx,0x0) и установили точку останова. (рис. 15)

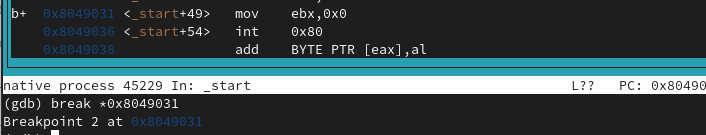


Рис. 15: .

Посмотрели информацию о всех установленных точках останова: (рис. 16)

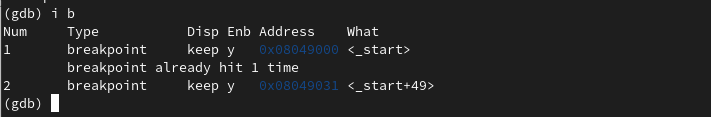


Рис. 16: .

## 3.4 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнили 5 инструкций с помощью команды stepi (или si) и проследили за изменением значений регистров. (рис. 17), (рис. 18)

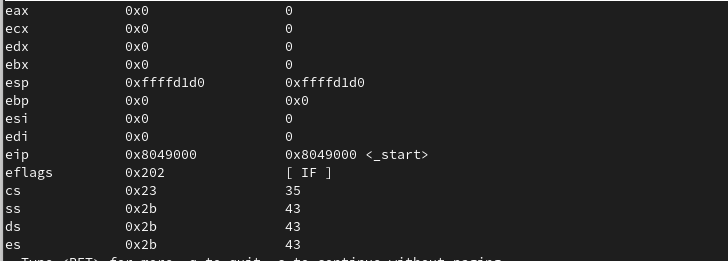


Рис. 17: .

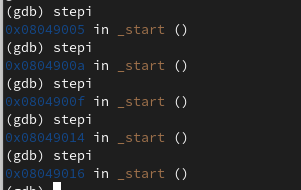


Рис. 18: .

Изменяются значения регистров: eax, ecx, edx, ebx.

Посмотрели содержимое регистров с помощью команды info registers (или i r). (рис. 19)

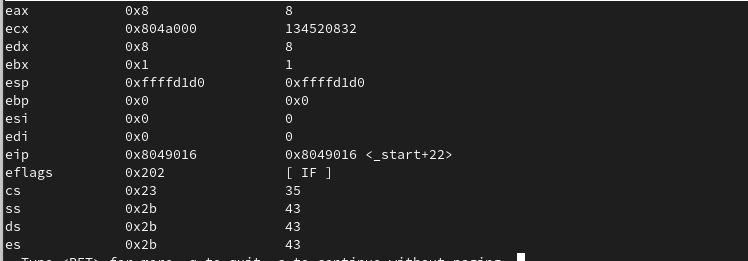


Рис. 19: .

Для отображения содержимого памяти можно использовать команду x , которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU . С помощью команды x & также можно посмотреть содержимое переменной. Посмотрели значение переменной msg1 по имени. (рис. 20)

Рис. 20: .

Рис. 20: .

Посмотрели значение переменной msg2 по адресу. Адрес переменной определили по дизассемблированной инструкции. Посмотрели инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2. (рис. 21)

Рис. 21: .

Рис. 21: .

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных. Изменили первый символ переменной msg1. (рис. 22)

Рис. 22: .

Рис. 22: .

Замените первый символ во второй переменной msg2. (рис. 23)

Рис. 23: .

Рис. 23: .

Чтобы посмотреть значения регистров используется команда print /F . Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 24)

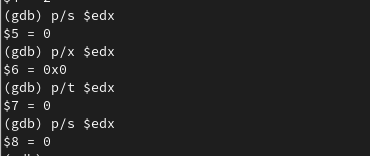


Рис. 24: .

С помощью команды set измените значение регистра ebx: (рис. 25)

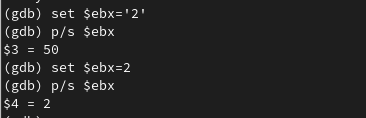


Рис. 25: .

Разница вывода комманд p/s $ebx:

Завершили выполнение программы с помощью команды continue (сокращенно c) и вышли из GDB с помощью команды quit (сокращенно q). (рис. 26), (рис. 27)

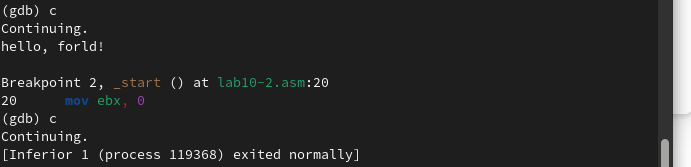


Рис. 26: .

Рис. 27: .

Рис. 27: .

## 3.5 Обработка аргументов командной строки в GDB

Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm: (рис. 28)

Рис. 28: .

Рис. 28: .

Создали исполняемый файл. (рис. 29)

Рис. 29: .

Рис. 29: .

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузили исполняемый файл в отладчик, указав аргументы: (рис. 30)

Рис. 30: .

Рис. 30: .

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследовали расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установили точку останова перед первой инструкцией в программе и запустили ее. (рис. 31)

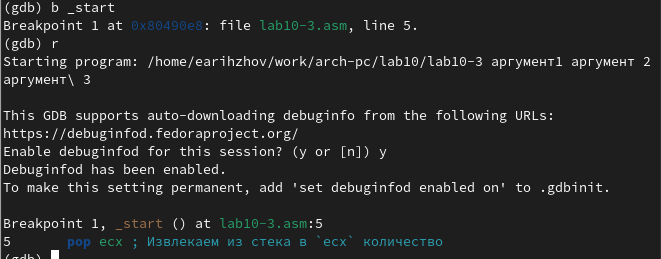


Рис. 31: .

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): (рис. 32)

Рис. 32: .

Рис. 32: .

Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрели остальные позиции 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 т.д. (рис. 33)

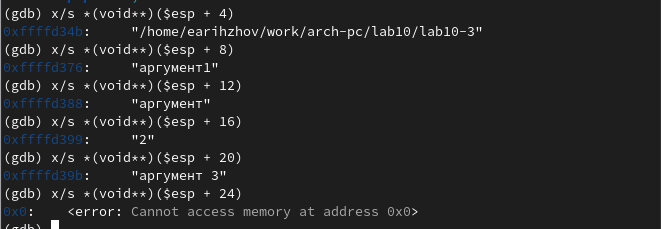


Рис. 33: .

Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.), потому что в теле цикла 4 строки кода.

## 3.6 Задание для самостоятельной работы

1. Преобразовали программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. 34), (рис. 35), (рис. 36)

Рис. 34: .

Рис. 34: .

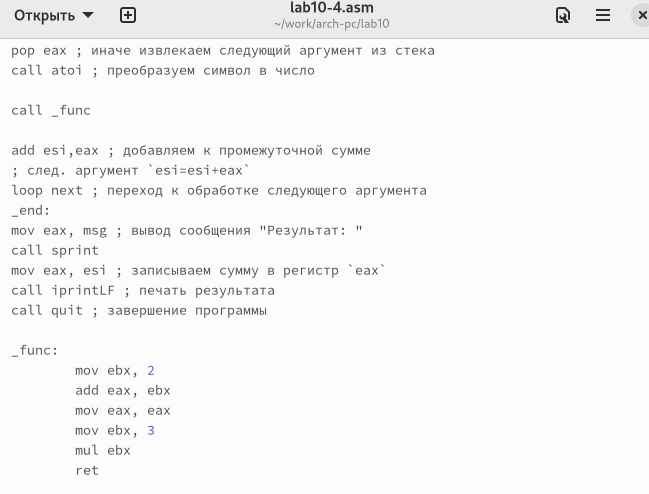


Рис. 35: .

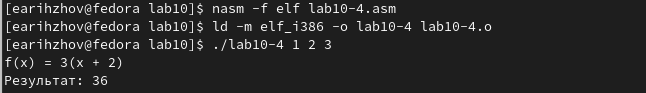


Рис. 36: .

1. В листинге 10.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5.

Создали файл (рис. 37), записали туда код листинга (рис. 38), создали исполняющий файл (рис. 39), при запуске обнаружили вывод неверного результата (рис. 40).

Рис. 37: .

Рис. 37: .

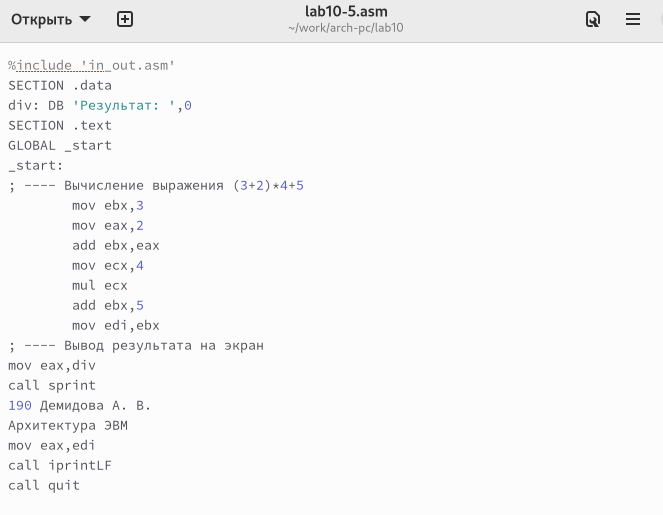


Рис. 38: .

Рис. 39: .

Рис. 39: .

Рис. 40: .

Рис. 40: .

Запустили файл в отладчике GDB (рис. 41), установили точку останова (рис. 42), запустили код (рис. 43), включили режим псевдографики (рис. 44), пошагово прошли все строчки кода (рис. 45), (рис. 46),(рис. 47), (рис. 48), (рис. 49), (рис. 50), (рис. 51), (рис. 52), обнаружили ошибку: вместо регистра ebx на 4 умножался eax, а 5 прибавлялась не к произведению, а только к ebx, исправили её (рис. 53), проверили результат работы программы (рис. 54).

Рис. 41: .

Рис. 41: .

Рис. 42: .

Рис. 42: .

Рис. 43: .

Рис. 43: .

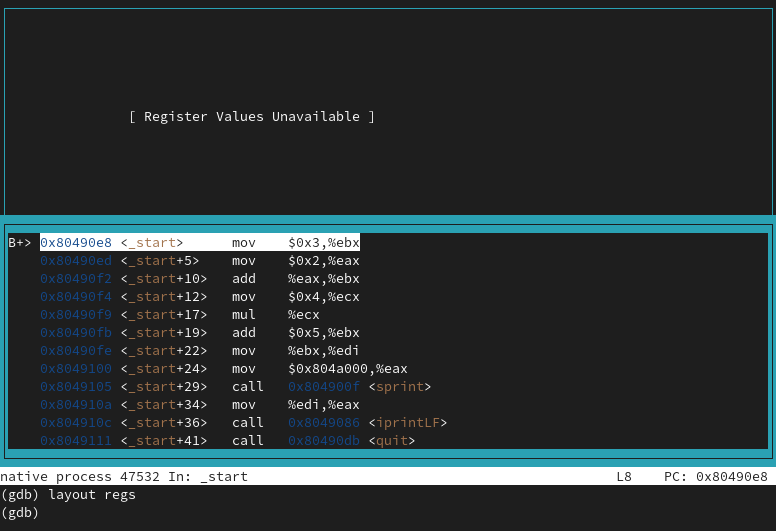


Рис. 44: .

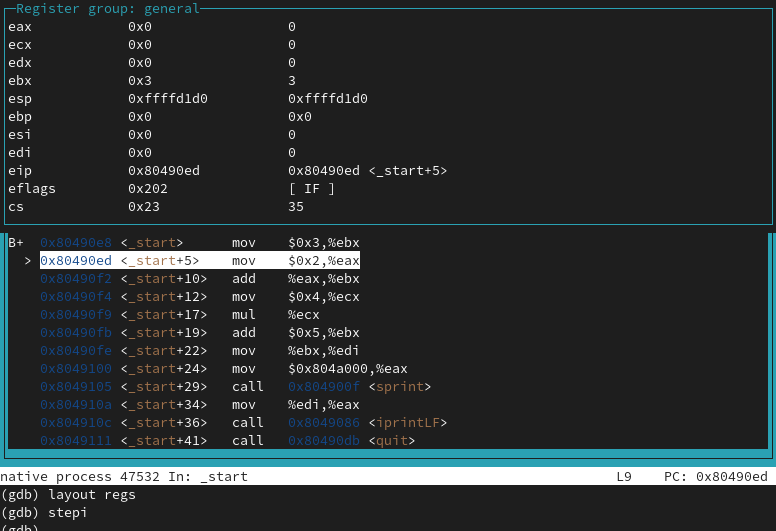


Рис. 45: .

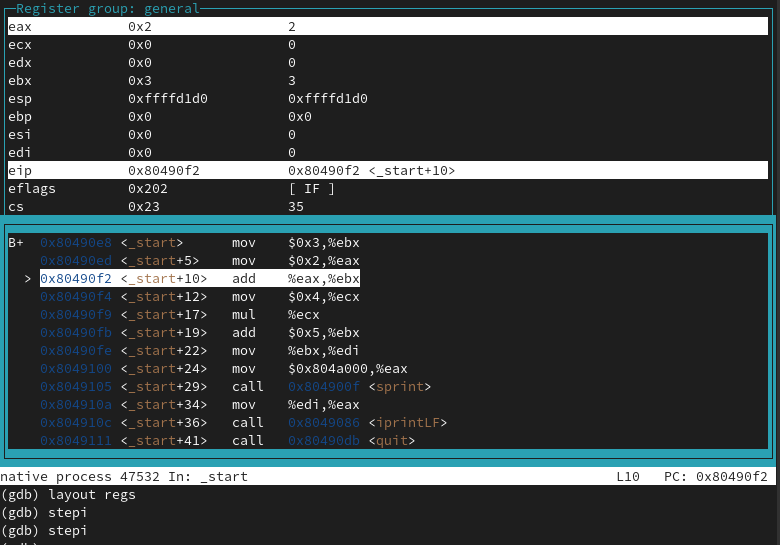


Рис. 46: .

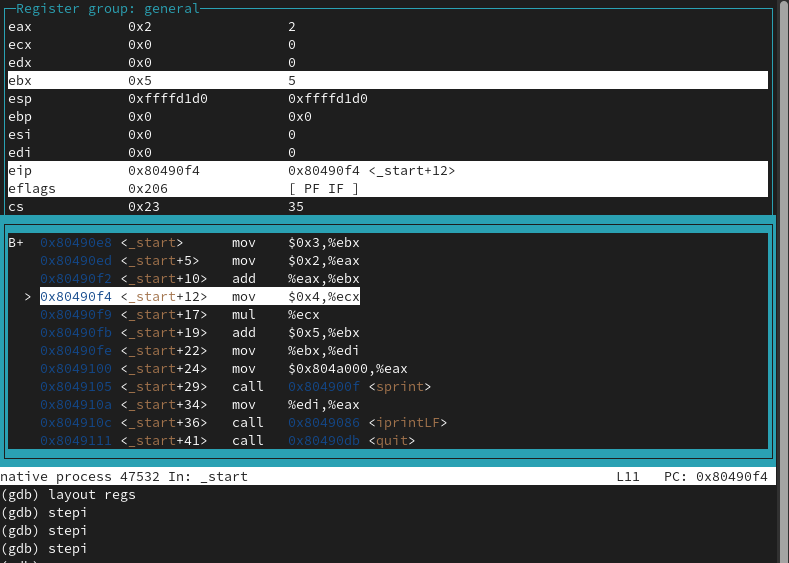


Рис. 47: .

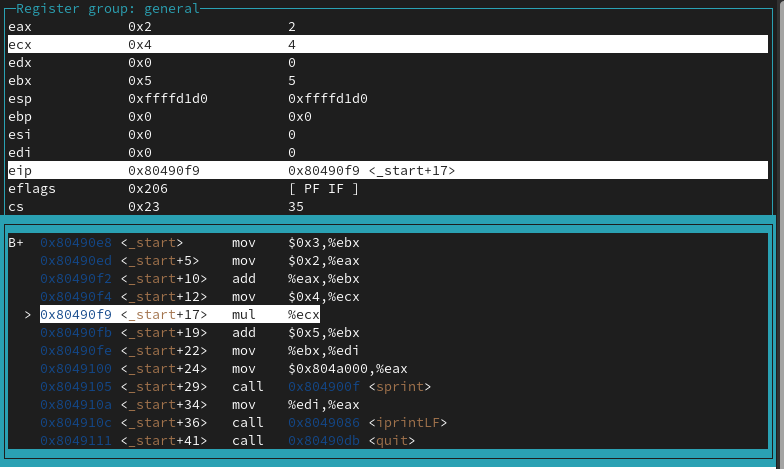


Рис. 48: .

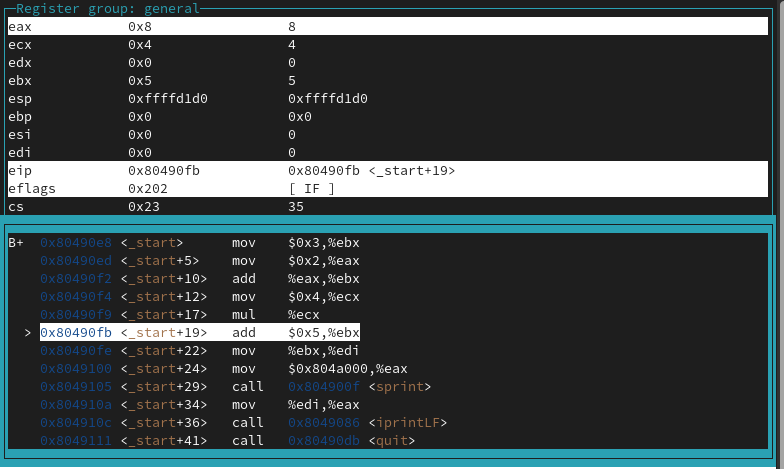


Рис. 49: .

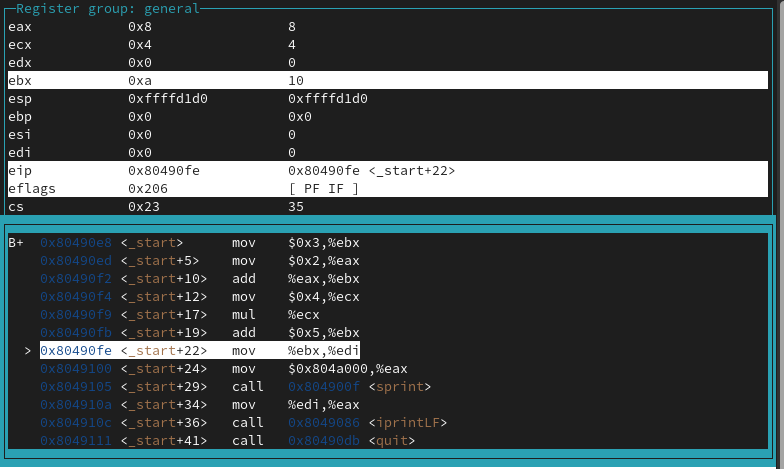


Рис. 50: .

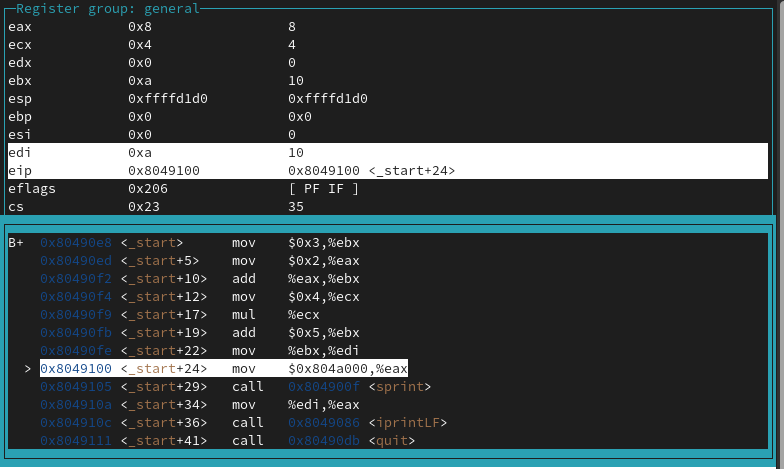


Рис. 51: .

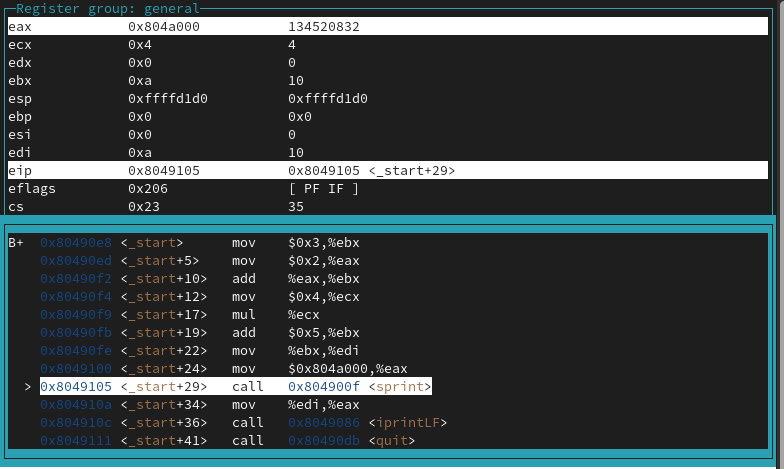


Рис. 52: .

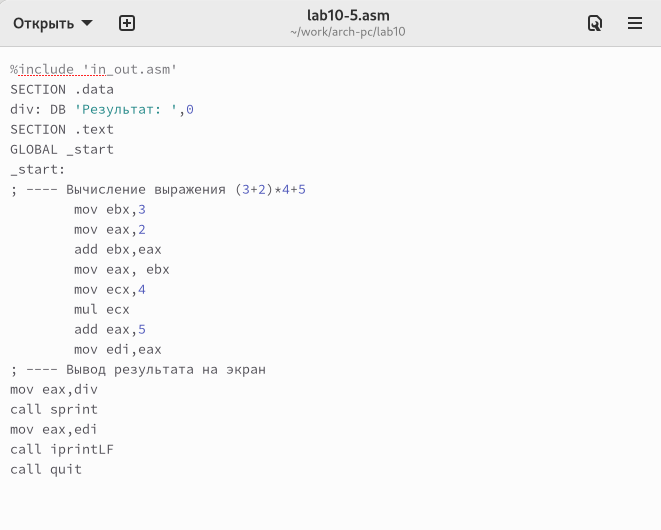


Рис. 53: .

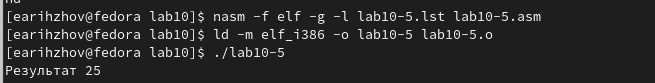


Рис. 54: .

# 4 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм, ознакомились с методами отладки при помощи GDB и его основными возможностями.