Отчет по лабораторной работе 10

Понятие подпрограммы. Отладчик GDB.

Гисматуллин Артём Вадимович НПИбд-01-22

Содержание

# 1 Цель работы

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

# 2 Задание

1. Рассмотрите программу вычисления выражения f(x) = 2х + 7 с помощью подпрограммы. Добавьте в нее подпрограмму вычисления выражения f(g(x)).
2. Проведите отладку программы листинга 10.2. Выполните соответствующие задания по изучению процесса отладки (точки останова, работа с данными программами в gdb, обработка аргументов).
3. Преобразовать программу из лабораторной работы №9, реализовав вычисления как подпрограмму.
4. С помощью отладчика gdb определить ошибку программы листинга 10.3.

# 3 Теоретическое введение

## 3.1 Понятие об отладке

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

• обнаружение ошибки;

• поиск её местонахождения;

• определение причины ошибки;

• исправление ошибки.

Можно выделить следующие типы ошибок:

• синтаксические ошибки — обнаруживаются во время трансляции исходно- го кода и вызваны нарушением ожидаемой формы или структуры языка;

• семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата;

• ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнару- жить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местона- хождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запус- ке программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

## 3.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы;

• использование специальных программ-отладчиков.

Отладчики позволяют управлять ходом выполнения программы, контролиро- вать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каж- дой строчки, чтобы программист мог проверить значения переменных и вы- полнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

• Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной про- граммистом);

• Watchpoint — точка просмотра (выполнение программы приостанавли- вается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

## 3.3 Основные возможности отладчика GDB

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX- подобных системах и умеет производить отладку многих языков программиро- вания. GDB предлагает обширные средства для слежения и контроля за выпол- нением компьютерных программ. Отладчик не содержит собственного графи- ческого пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графиче- ских надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что проис- ходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её пове- дение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

### 3.3.1 Понятие подпрограммы

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом пере- ход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы.

Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

# 4 Выполнение лабораторной работы

1. Создаем каталог для программам лабораторной работы No 10, перейдим в него и создаем файл lab10-1.asm. (рис. 1)

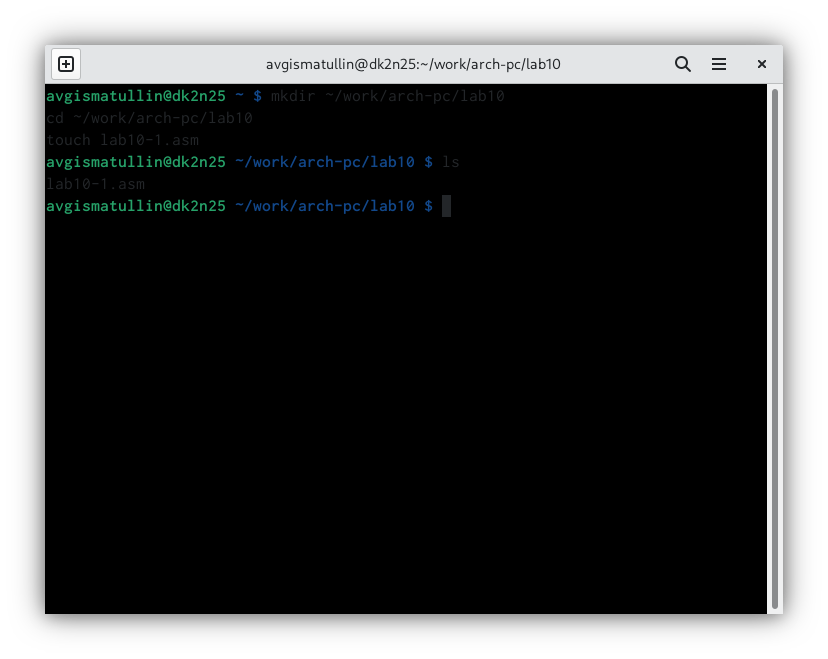


Рис. 1: Командная строка. каталог ~/work/arch-pc/lab10

1. Ввели в файл lab10-1.asm текст программы листинга, создали исполняемый файл и посмотрели результат программы. (рис. 2)

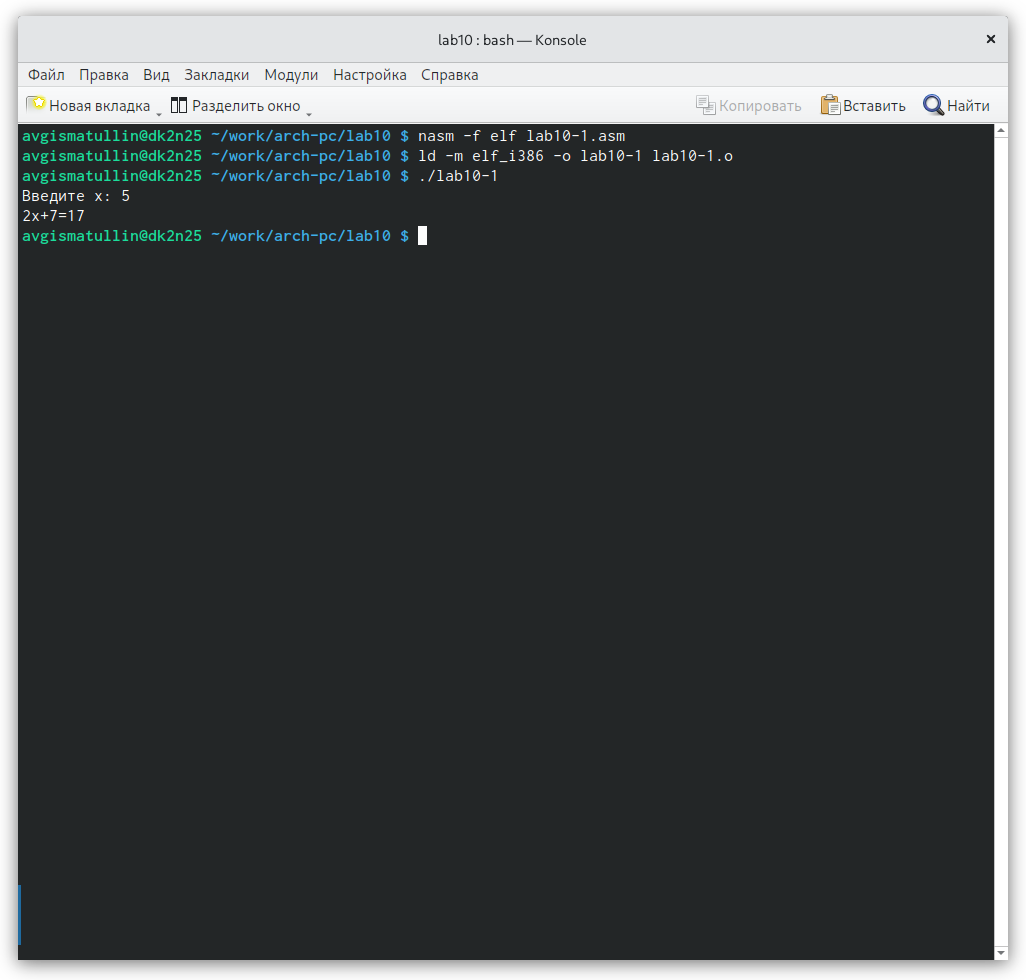


Рис. 2: Командная строка. Выполнение программы lab10-1.asm

Далее добавляем подпрограмму для вычисления функции g(x) = 3x - 1, и выполняем программу. Результат: (рис. 3)

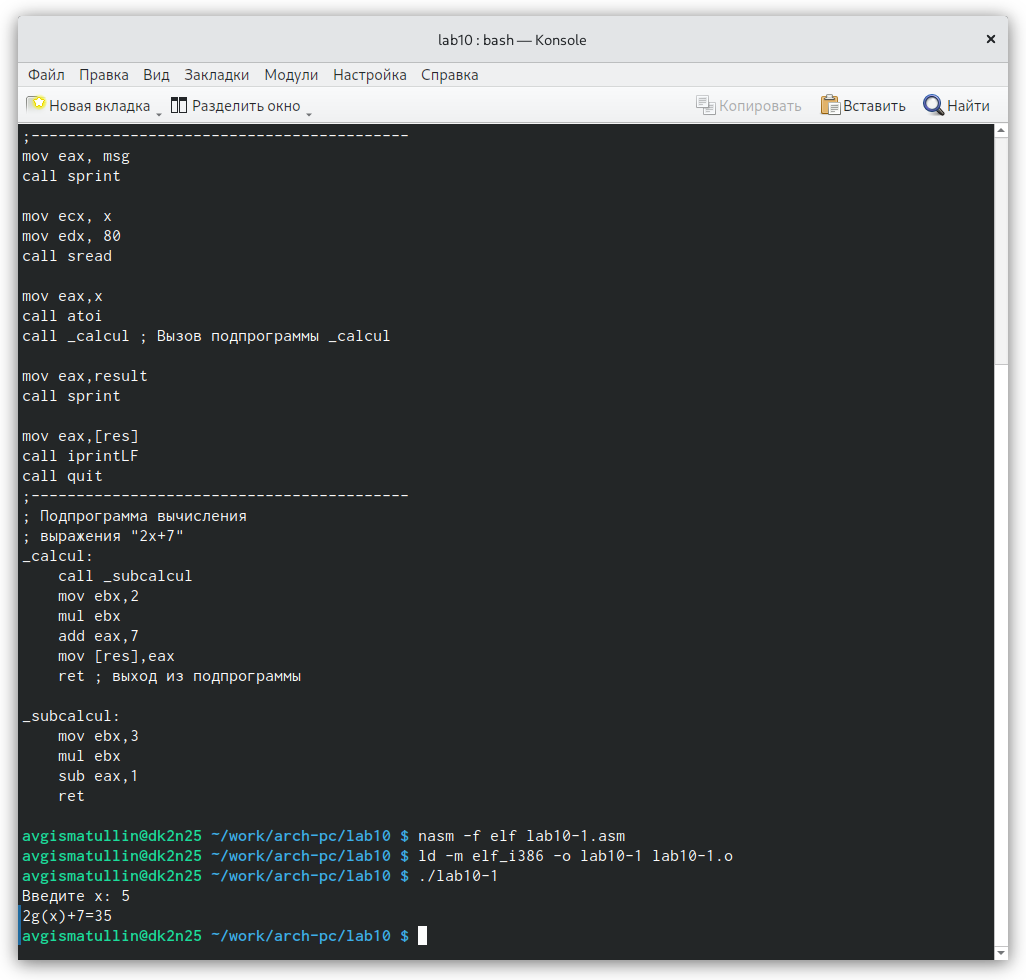


Рис. 3: Командная строка. Выполнение измененной программы lab10-1.asm

1. Создаем файл lab10-2.asm с текстом программы из Листинга 10.2, получаем исполняемый файл с ключом ‘-g’. (рис. 4)

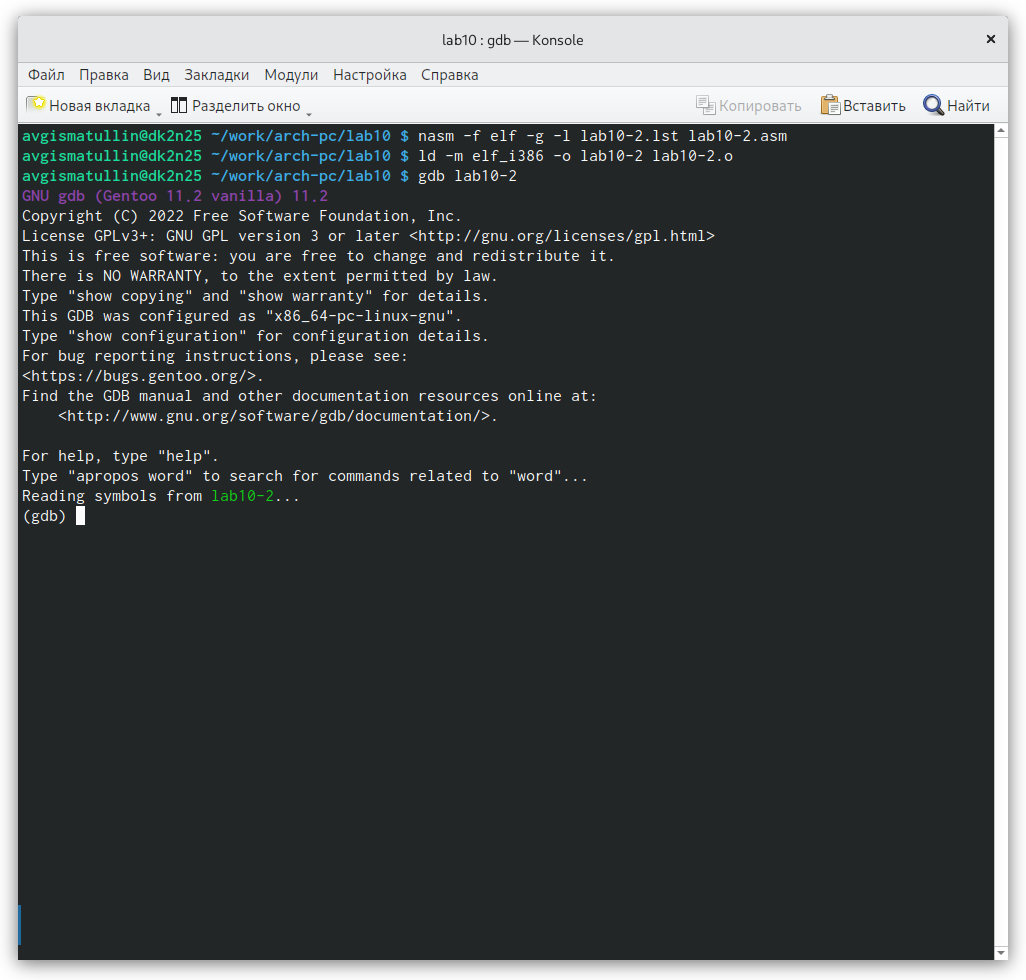


Рис. 4: Командная строка. Отладка программы lab10-2.asm

Проверяем работу программы, запустив ее в оболочке GDB с помощью коман- ды run, устанавливаем брейкпоинт на метку \_start, посмотрели дисассимилированный код программы с помощью команды disassemble (рис. 5)

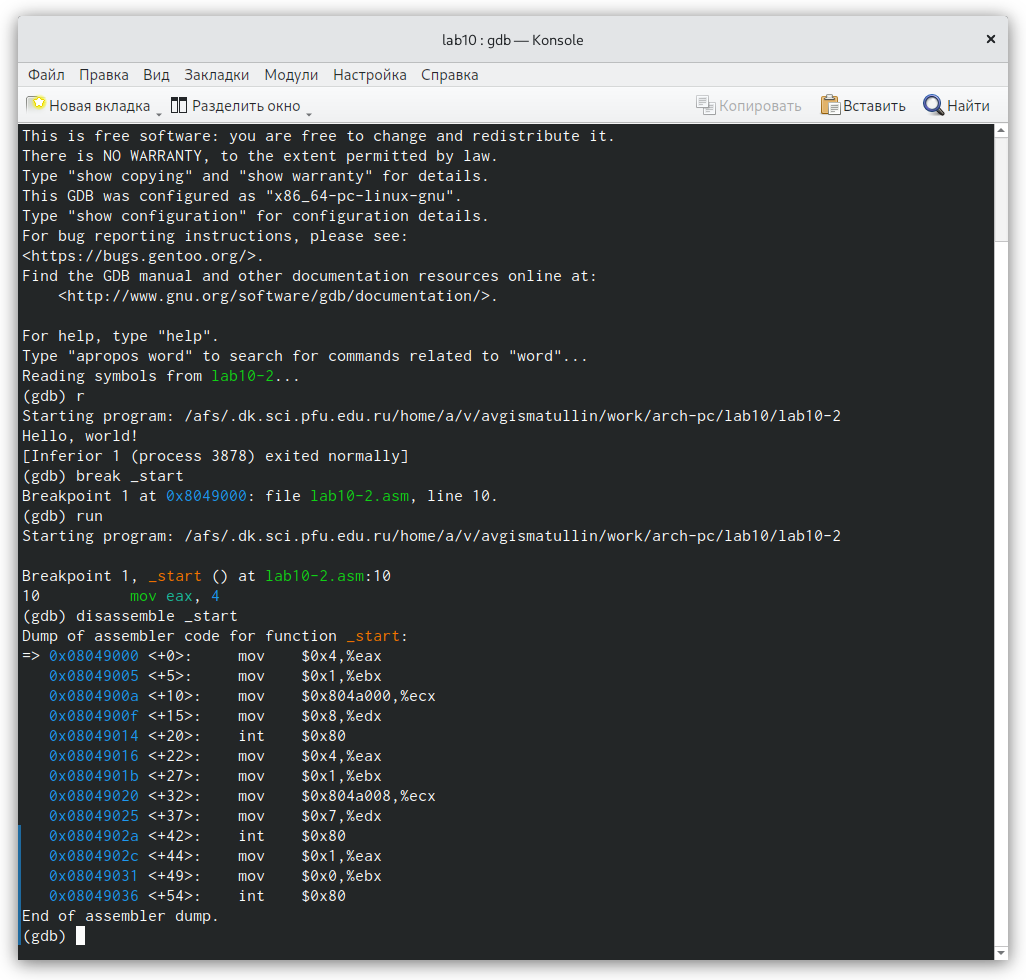


Рис. 5: Командная строка. Отладка программы lab10-2.asm 2

Затем переключаем на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 6)

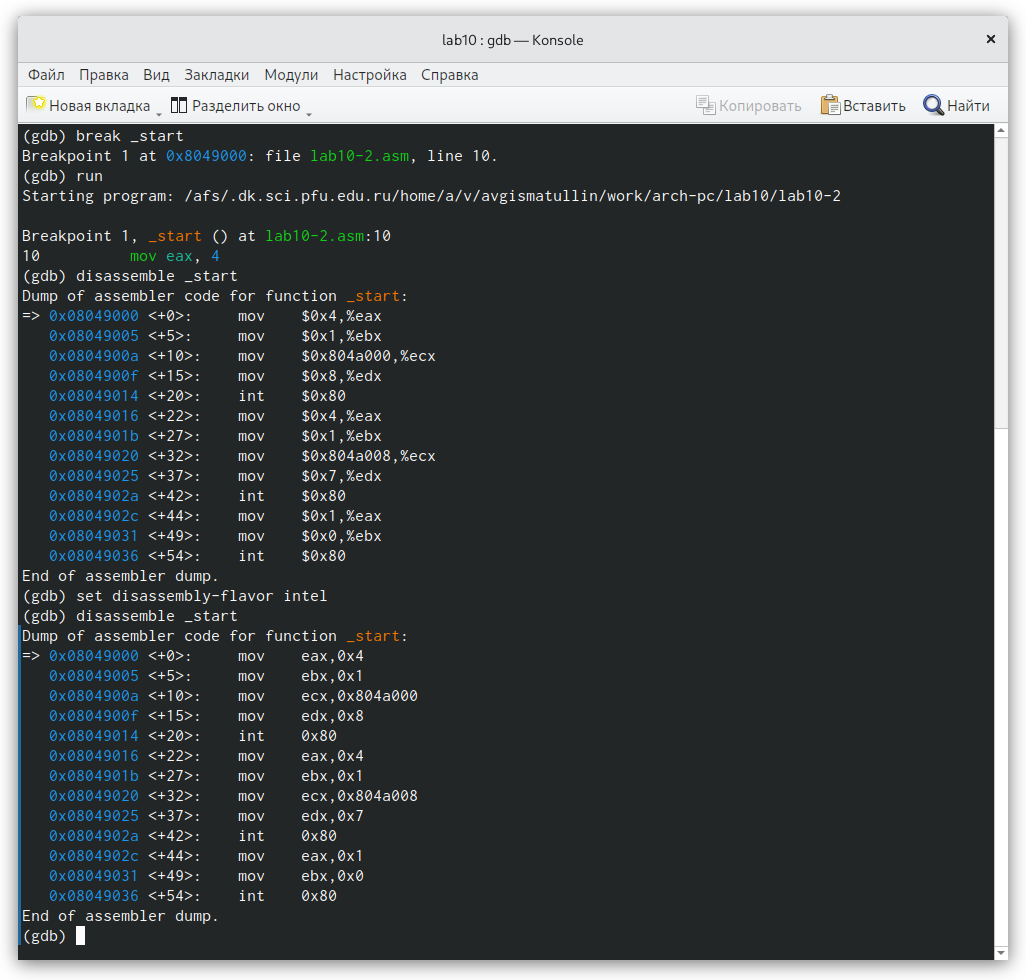


Рис. 6: Командная строка. Intel синтаксис

Различия отображения синтаксиса машинных команд в режимах ATT и Intel отличаются тем, что название регистра и адрес в режиме Intel меняются местами и изменяются так, что читаемость листинга отладки становится лучше.

Затем включаем режим псевдографики для более удобного анализа программы: (рис. 7)

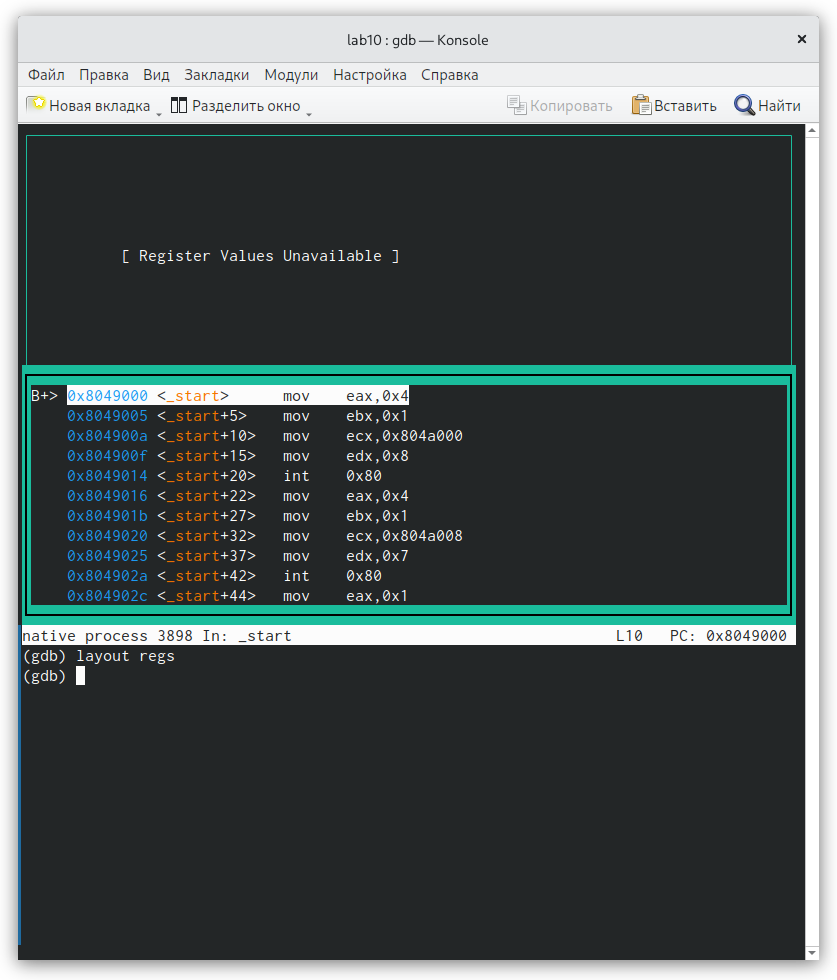


Рис. 7: Командная строка. Режим псевдографики

1. Далее устанавливаем еще одну точку останова по адресу инструкции, смотрим информацию о всех установленных точках останова, а после начинаем работу с данными программы, выполняя инструкции командой stepi (si): (рис. 8)

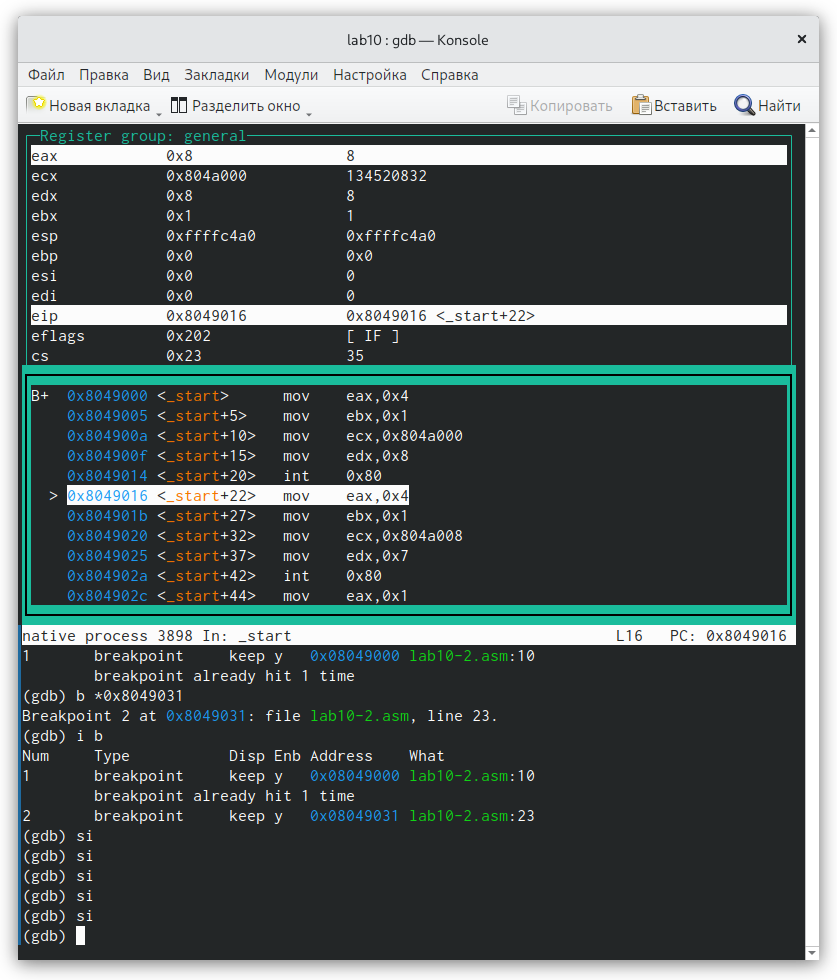


Рис. 8: Командная строка. Выполнение инструкций.

Можно проследить, как изменяются значения регистров eax, ebx, ecx, edx, eip и др.

1. После просмотра содержимого регистров, смотрим значение переменной msg1. (рис. 9)

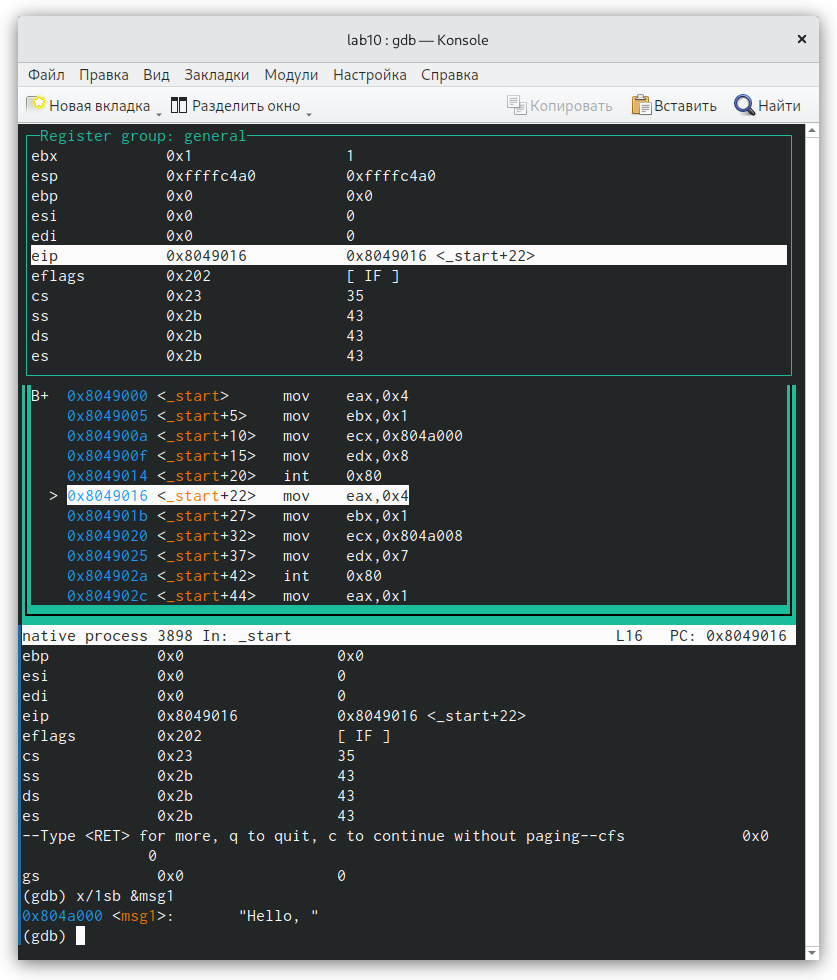


Рис. 9: Командная строка. Просмотр значения переменной

1. Смотрим значение переменной msg1 по адресу, изменяем значение для этой переменной, смотрим значение переменной msg2 и проводим с ним изменения первого символа: (рис. 8)

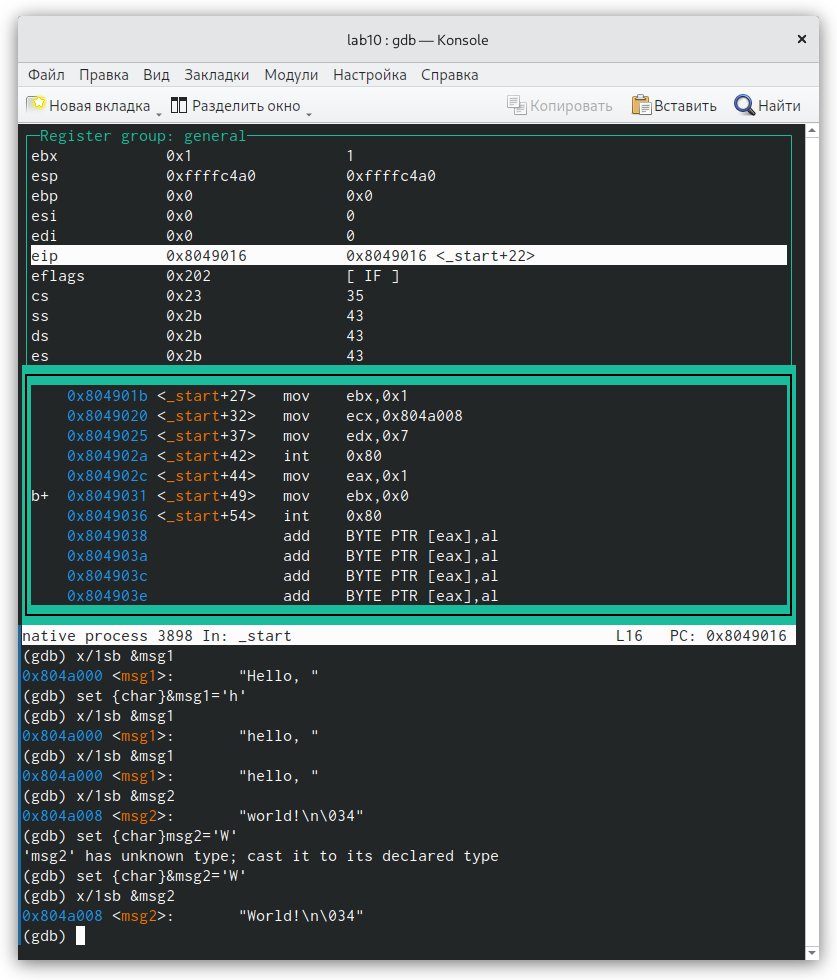


Рис. 10: Командная строка. Просмотр и изменение значения переменных

1. Далее выводим в различных форматах значение регистров eax и edx. (рис. 11)

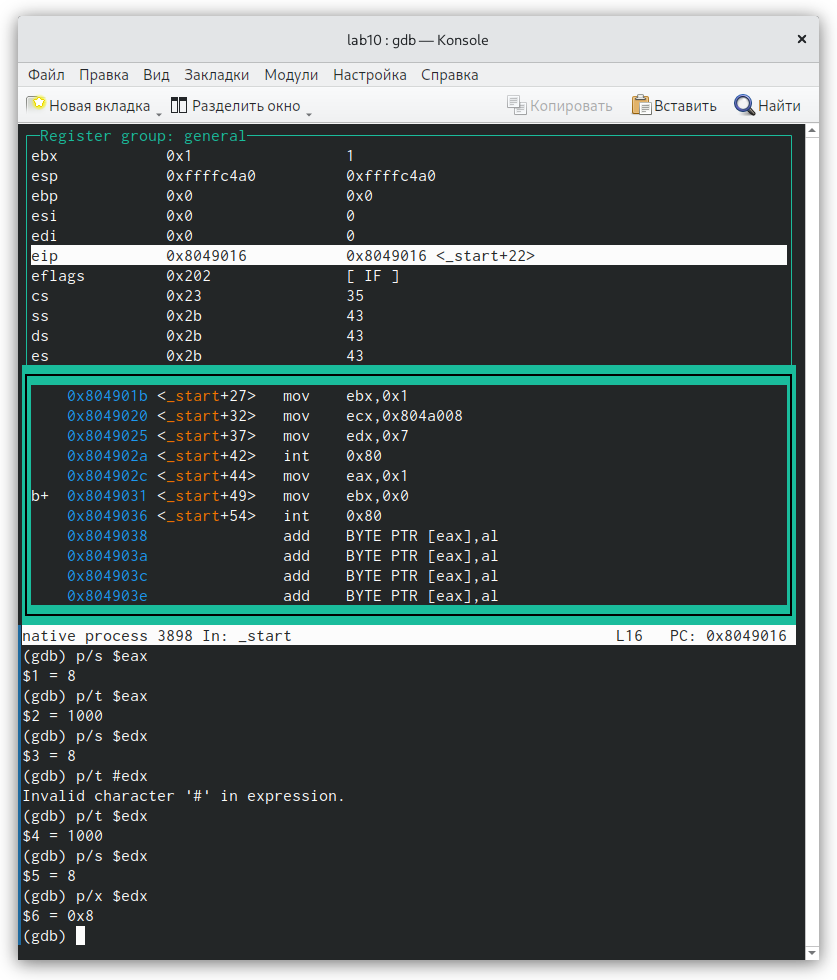


Рис. 11: Командная строка. Значения регистров eax и edx

Более того, значения регистров можно изменять с помощью команды set: (рис. 12)

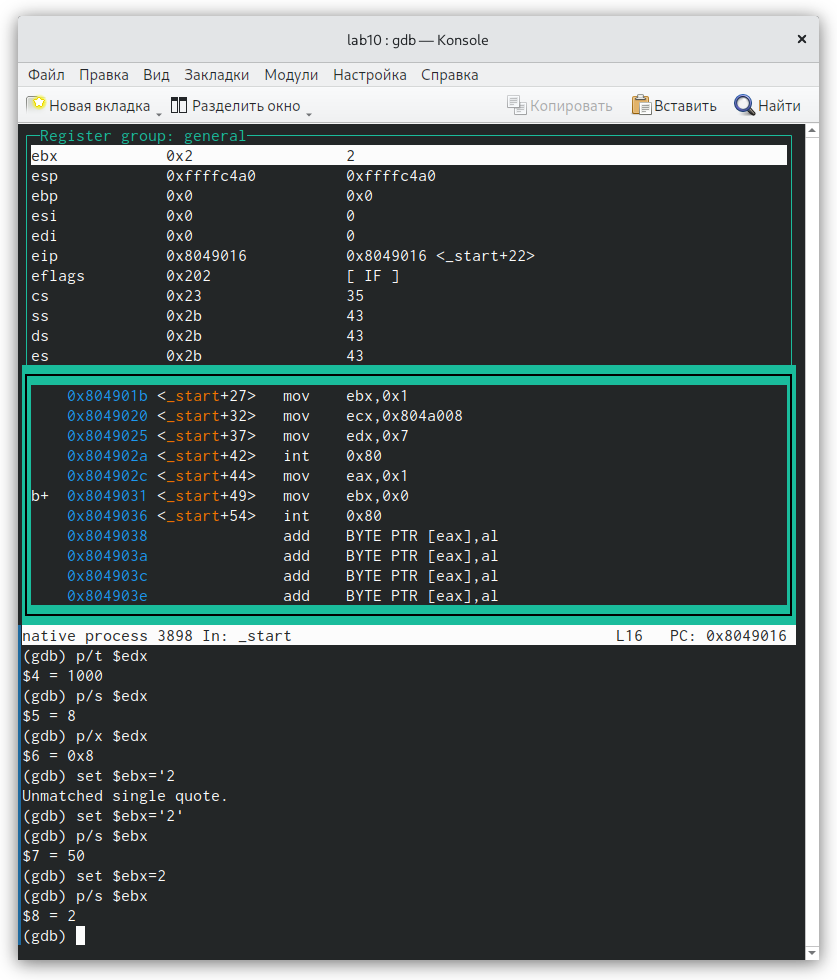


Рис. 12: Командная строка. Изменение знчаения регистра ebx

Разница между выводом p/s для ‘2’ и 2 заключается в том, что для данных символов возвращается число из таблицы символов Unicode представляющее их позицию, а строковые объекты и числа имеют разные значения.

1. Завершив выполнение программы, копируем файл lab9-2.asm в lab10-3.asm и запускаем отладчик в программе с аргументами. Устанавливаем брэйкпоинт и смотрим количество аргументов командной строки: (рис. 13)

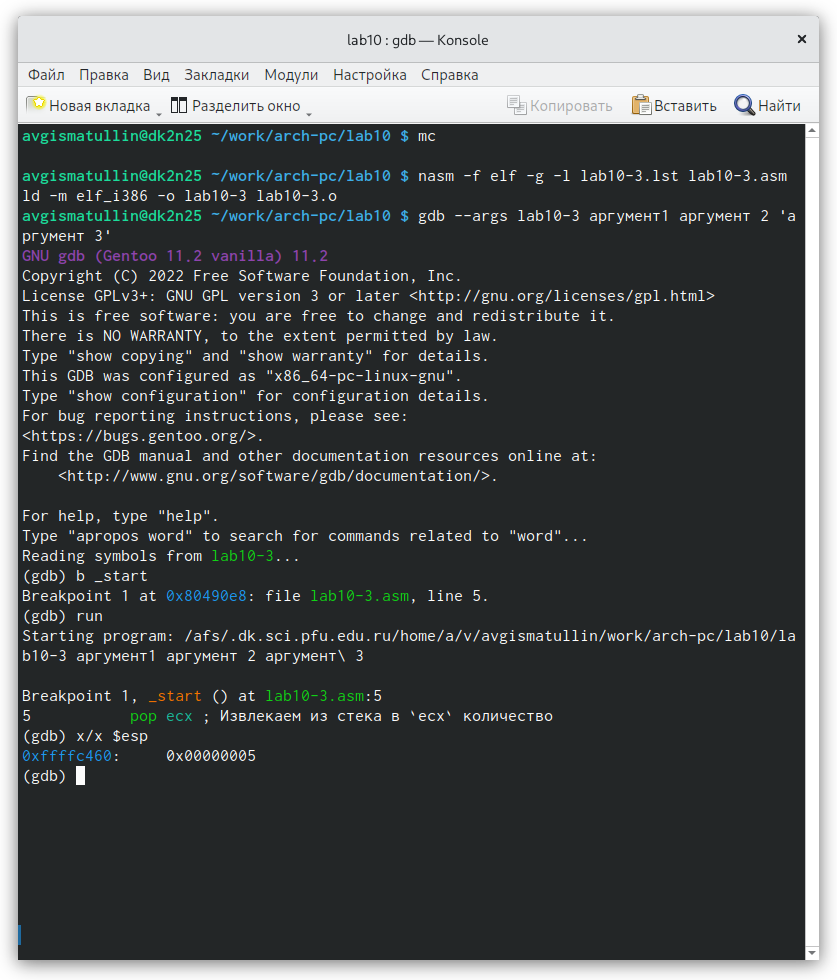


Рис. 13: Командная строка. Запуск в gdb lab10-3.asm

1. C помощью команды x/s \*(void\*\*)($esp + n) можно посмотреть, какие аргументы находятся в стеке: (рис. 14)

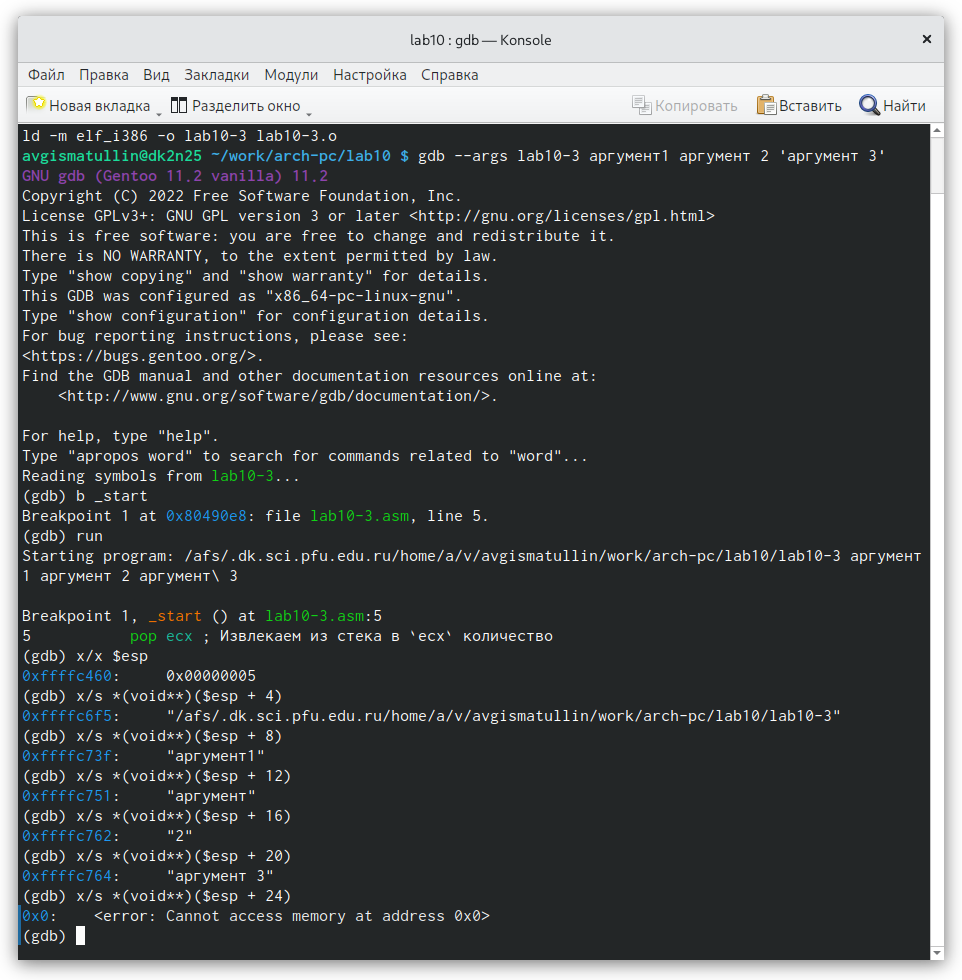


Рис. 14: Командная строка. Просмотр содержимого очереди

Примечательно следующее: адрес выглядит, как [$esp + n], где n кратно 4. Это вызвано тем, что n - число байт, на которое смещен элемент относительно вершины стека.

# 5 Выполнение задания для самостоятельной работы

Суть задания для самостоятельной работы заключается в том, чтобы преобразовать программу из лабораторной работы №9, реализовав вычисление значения функции как подпрограмму, а также изучить листинг 10.3 с помощью отладчика gdb и определить в нем ошибку.

1. Создадим файл lab10-4.asm для создания подпрограммы для функции. Скопируем программу из файла lab9-4.asm, цель которой была вычисление сумм значений функций f(x) = 15x + 2. Перепишем ее, добавив подпрограмму. Результат получился следующим: (рис. 15)

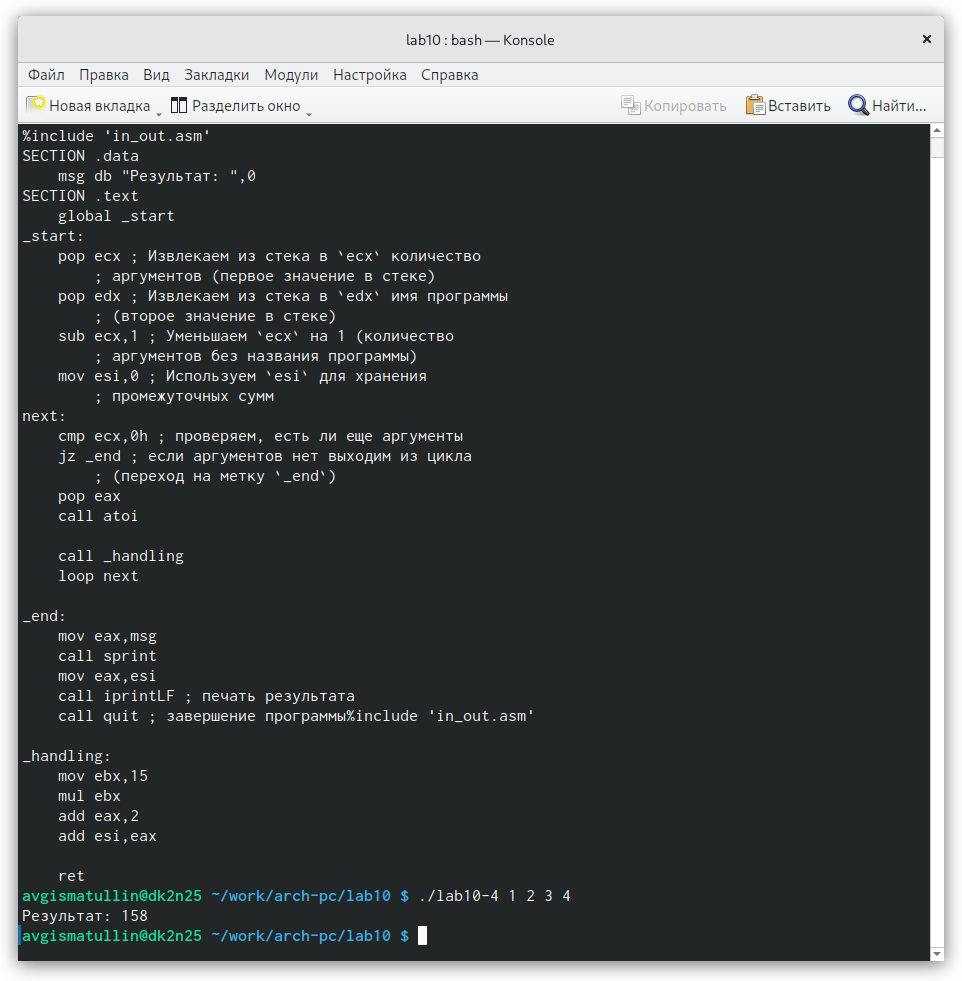


Рис. 15: Командная строка. Программа lab10-4.asm

Ответ получился верным.

1. Скопируем программу листинга 10.3 в файл lab10-5.asm, запустим его в gdb и проследим за изменением значений регистров. (рис. 16)

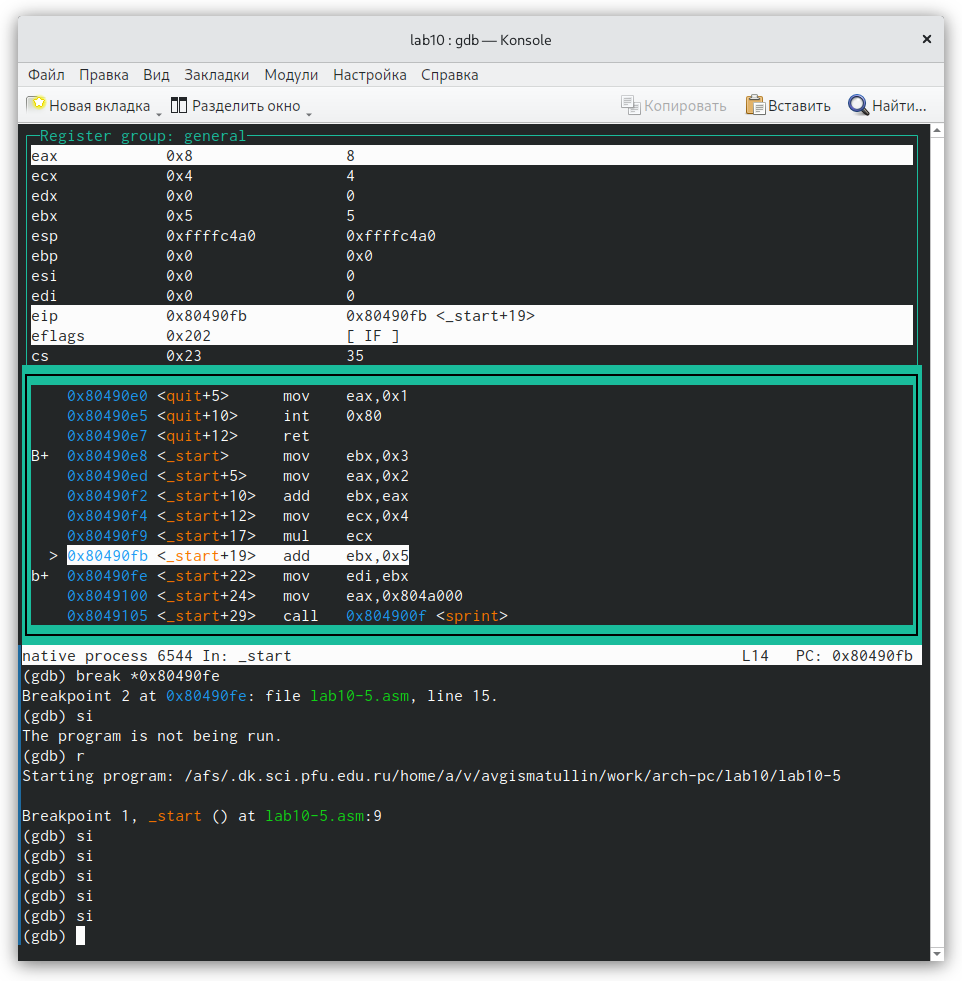


Рис. 16: Командная строка. Отладка программы lab10-5.asm

Здесь сразу можно обратить внимание, что значение из регистра eax прибавляется к ebx, а не наоборот. Ввиду этого на данный момент значение в eax равно 8. Проходим далее: (рис. 17)

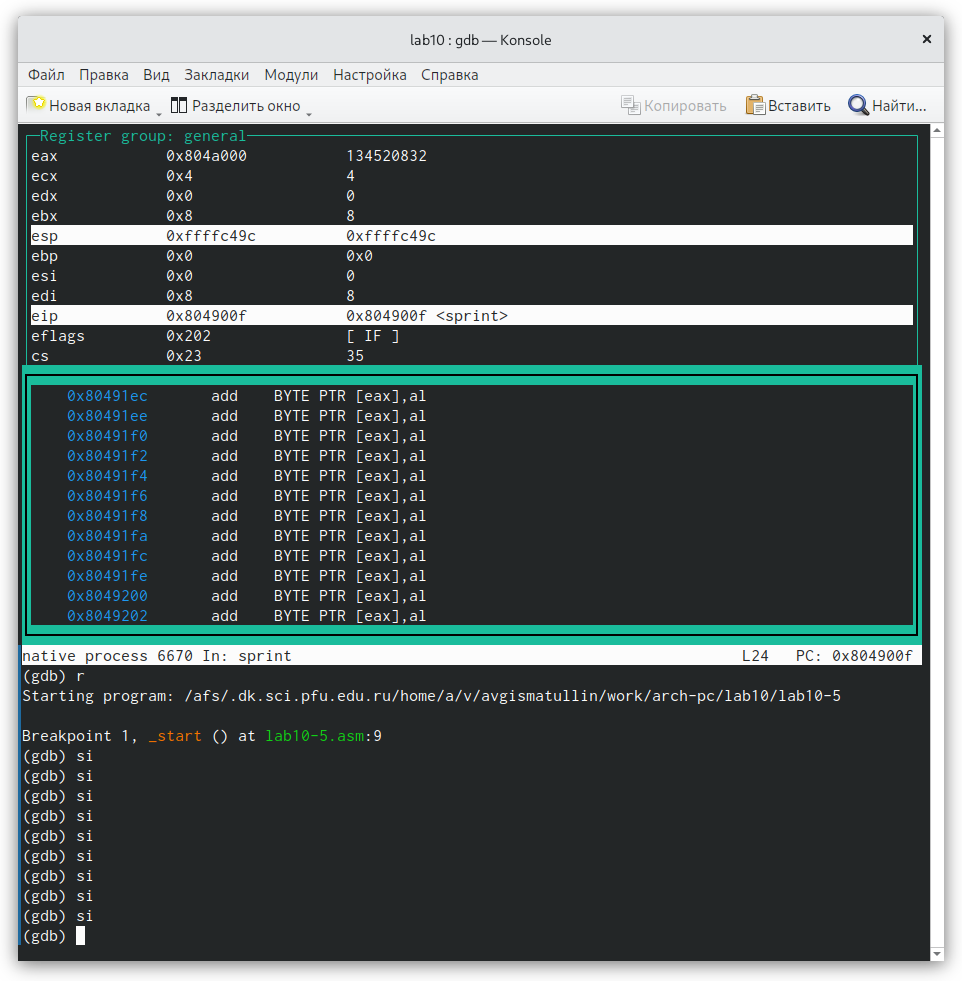


Рис. 17: Командная строка. Отладка программы lab10-5.asm 2

Здесь тоже можно сделать вывод, что значения прибавляются не к тому регистру, а edi запоминает число из ebx, что тоже неверно. Основываясь на всех этих недочетах, я переписал программу следующим образом: (рис. 18)

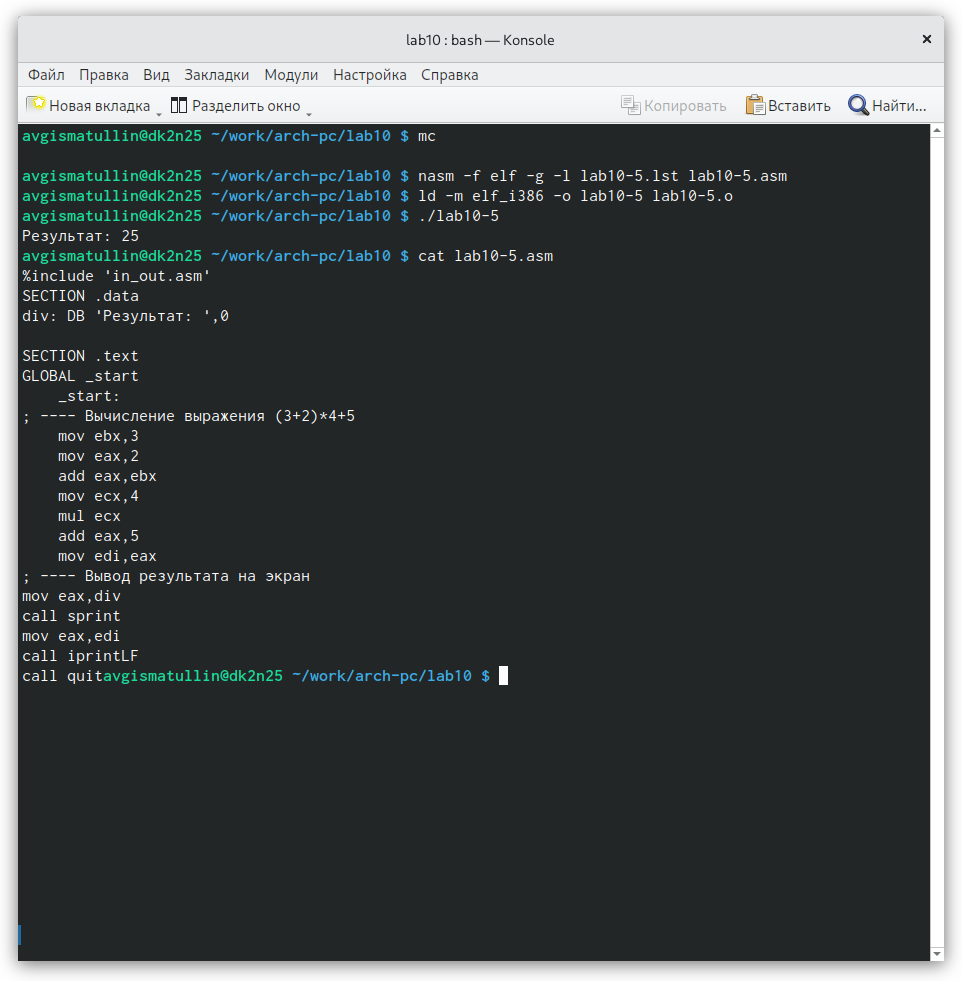


Рис. 18: Командная строка. Выполнение верной программы lab10-5.asm

# 6 Выводы

В ходе работы я приобрел навыки написания программ с использованием подпрограмм, познакомился с методами отладки при помощи GDB и его основными возможностями.

# Список литературы