Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Ларина Наталья Денисовна

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. Реализация подпрограмм в NASM.
2. Отладка программам с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

# 3 Теоретическое введение

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 исходного 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 пре- рывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить 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 местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторон- них графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm. (рис. [1](#fig:001))

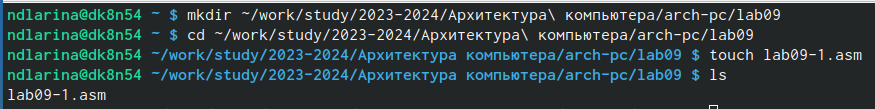


Рис. 1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. [2](#fig:002))

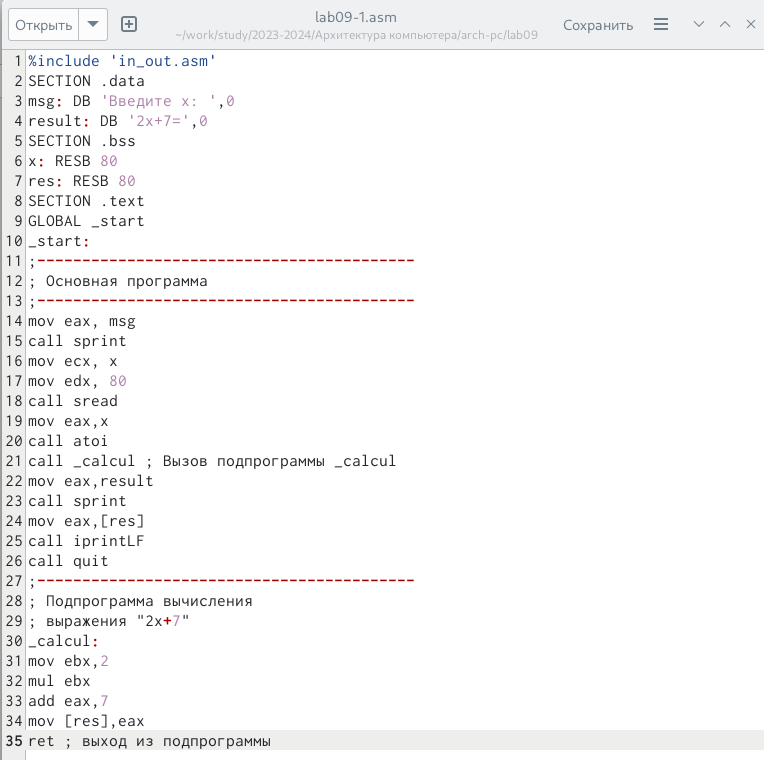


Рис. 2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. [3](#fig:003))

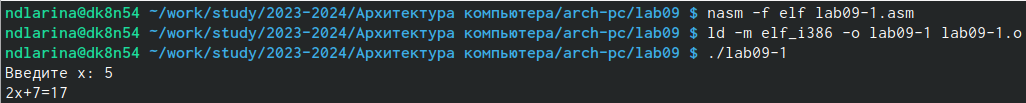


Рис. 3: Запуск исполняемого файла

Изменяю 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. (рис. [4](#fig:004))

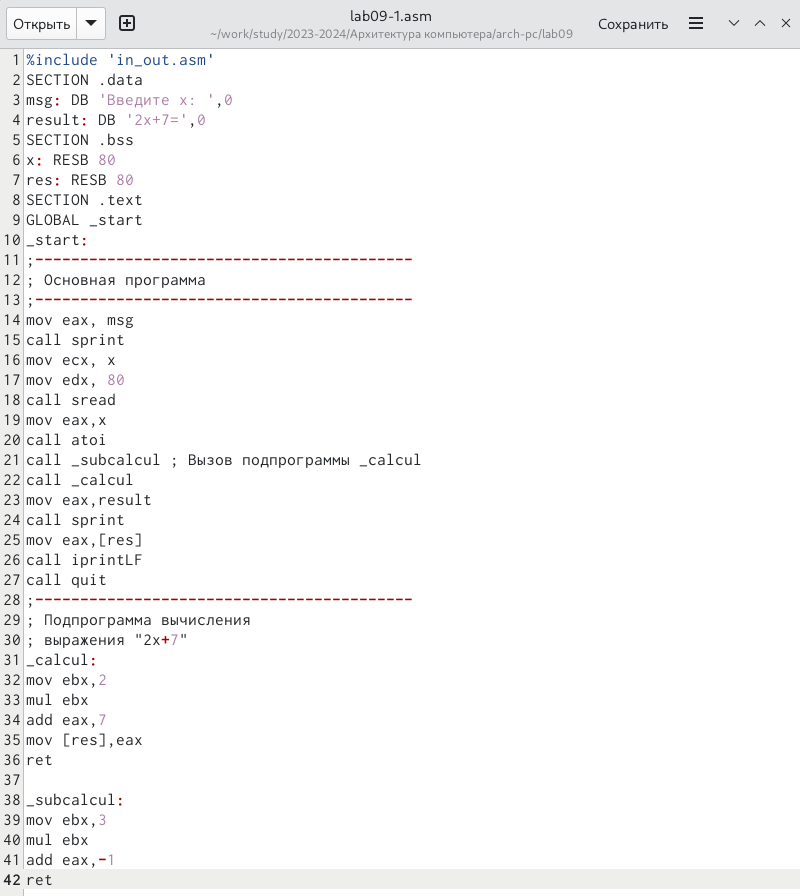


Рис. 4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. [5](#fig:005))

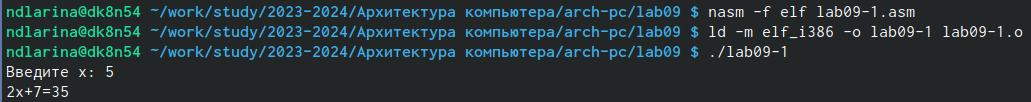


Рис. 5: Запуск исполняемого файла

## 4.2 Отладка программам с помощью GDB

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

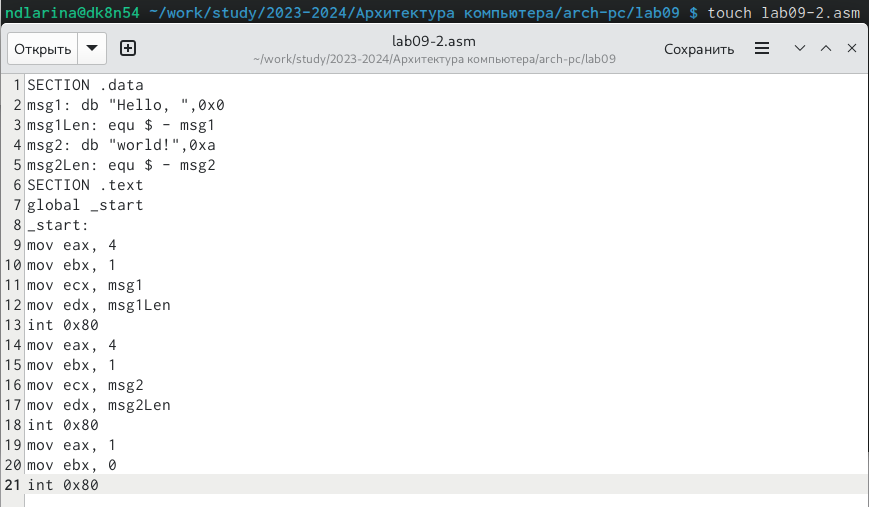


Рис. 6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом ‘-g’. (рис. [7](#fig:007))

Figure 7: Получение исполняемого файла

Рис. 7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. [8](#fig:008))

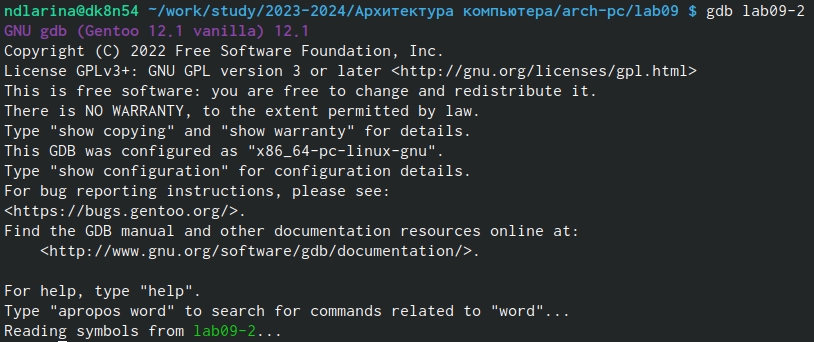


Рис. 8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. [9](#fig:009))

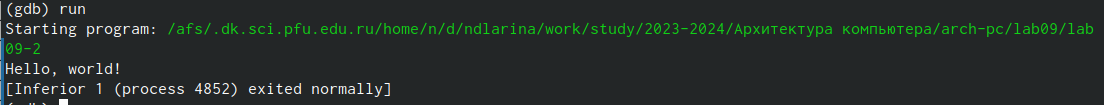


Рис. 9: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её. (рис. [10](#fig:010))

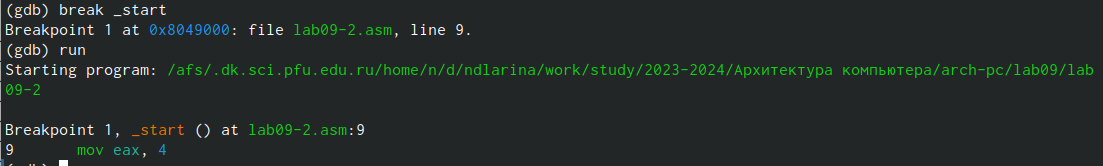


Рис. 10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel. (рис. [11](#fig:011))

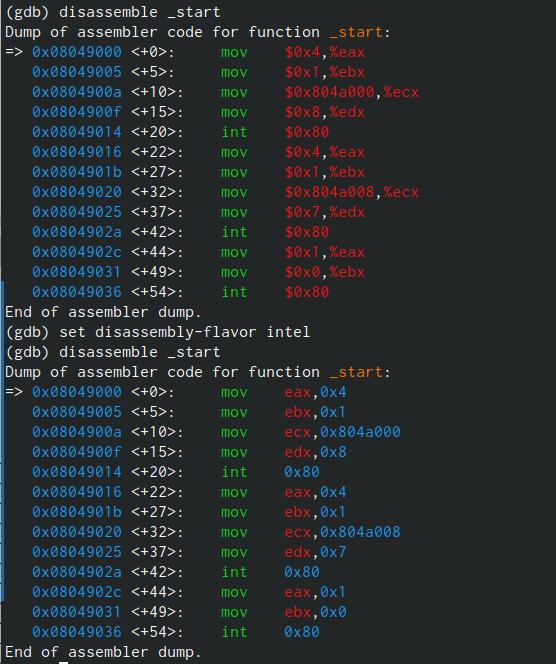


Рис. 11: Использование команд disassemble и disassembly-flavor intel

В режиме ATT имена регистров начинаются с символа %, а имена операндов с $, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs. (рис. [12](#fig:012))

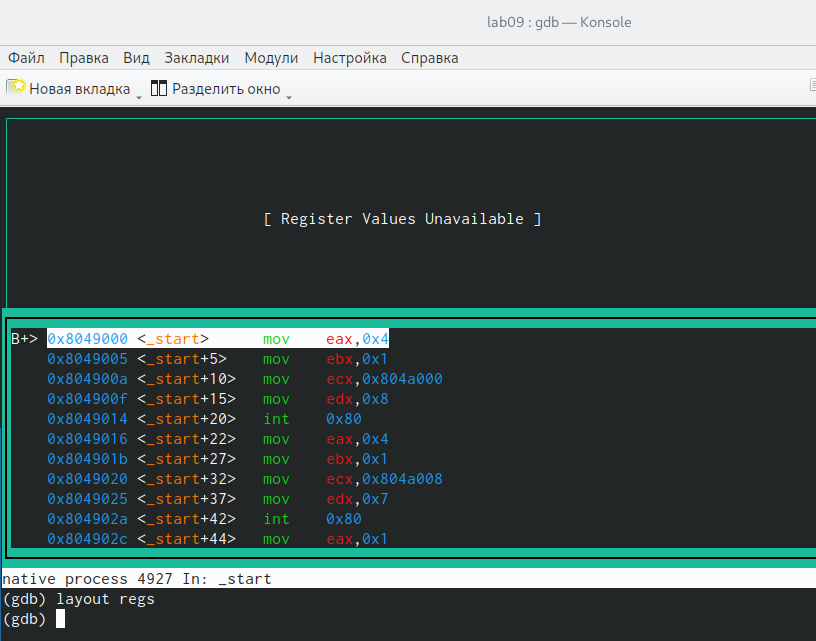


Рис. 12: Включение режима псевдографики

### 4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки \_start установлена с помощью команды info breakpoints и устанавливаю еще одну точку останова по адресу инструкции mov ebx,0x0. Просматриваю информацию о всех установленных точках останова. (рис. [13](#fig:013))

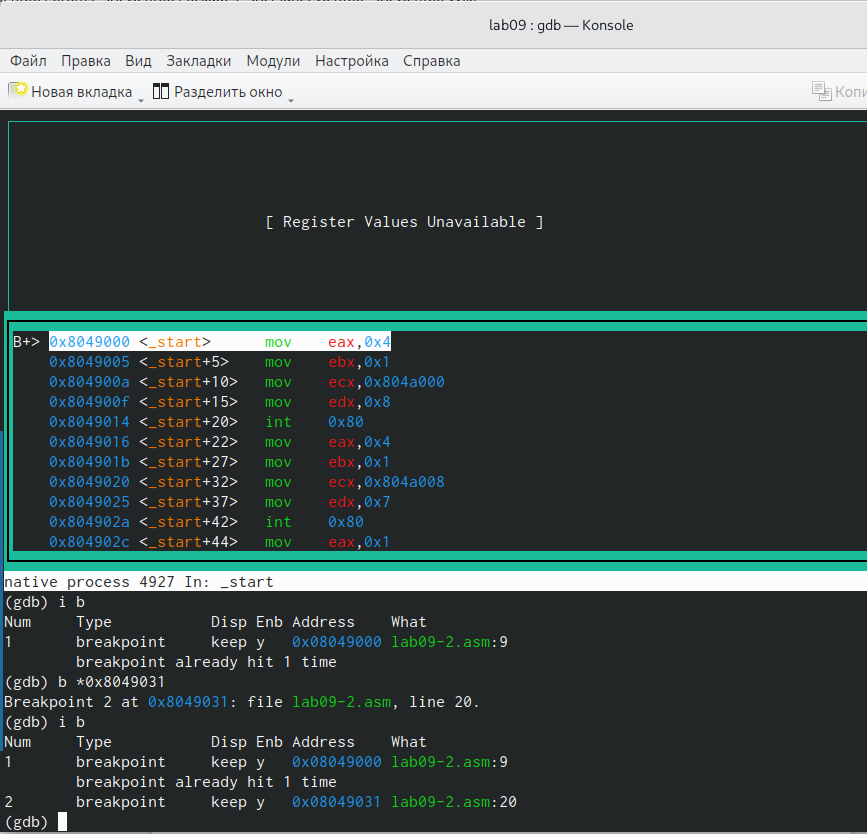


Рис. 13: Установление точек останова и просмотр информации о них

### 4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением значений регистров. (рис. [1](#fig:001))

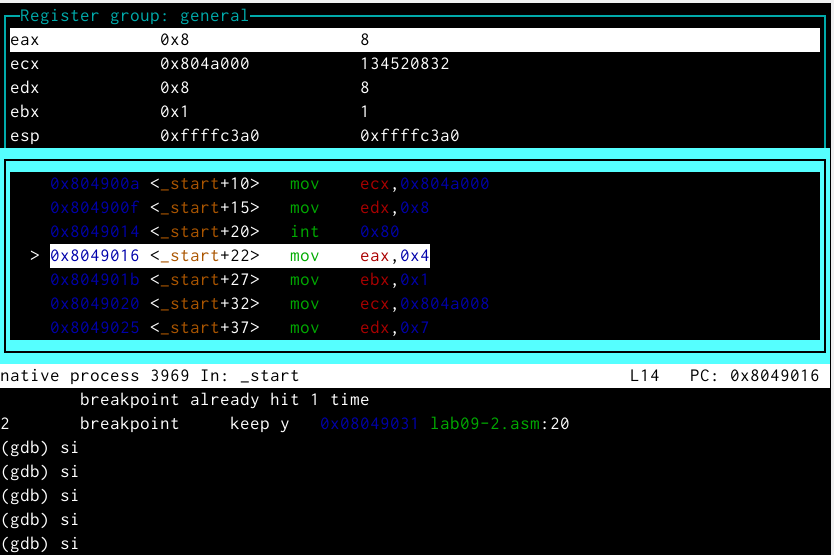


Рис. 14: Использование команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Далее просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. [15](#fig:015))

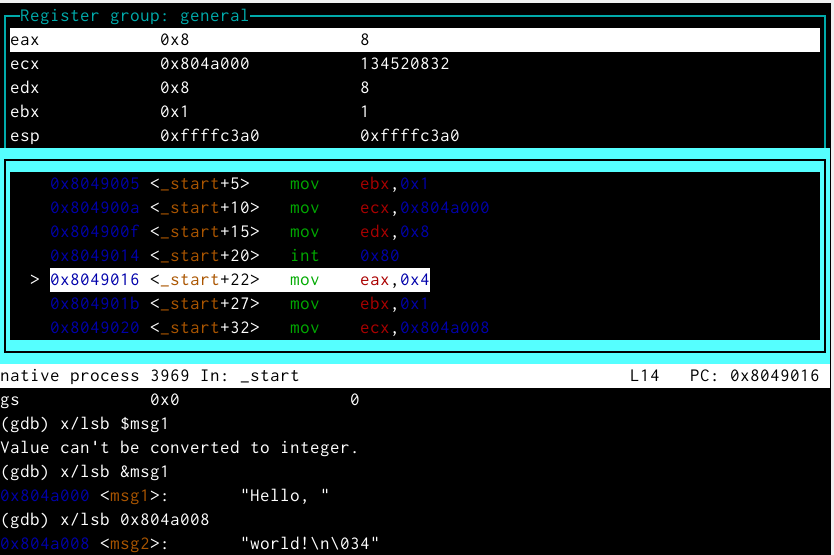


Рис. 15: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. [16](#fig:016))

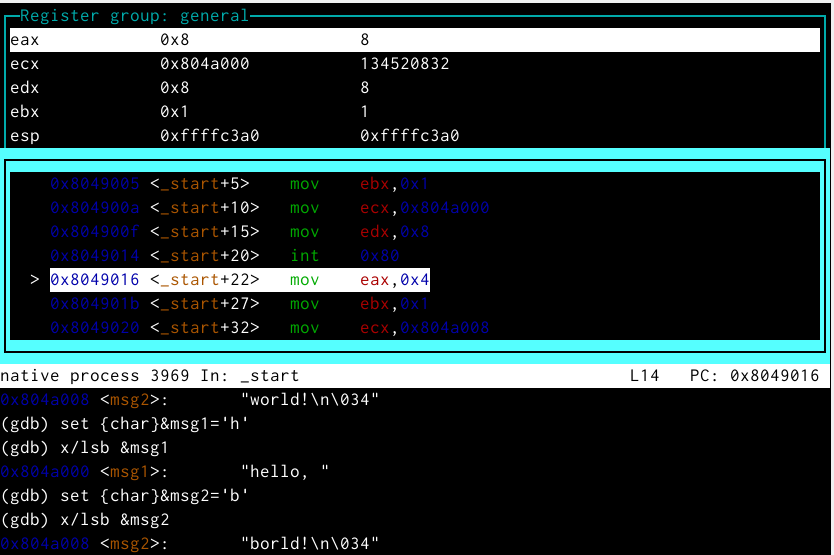


Рис. 16: Использование команды set

Затем вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F $val. И с помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. [17](#fig:017))

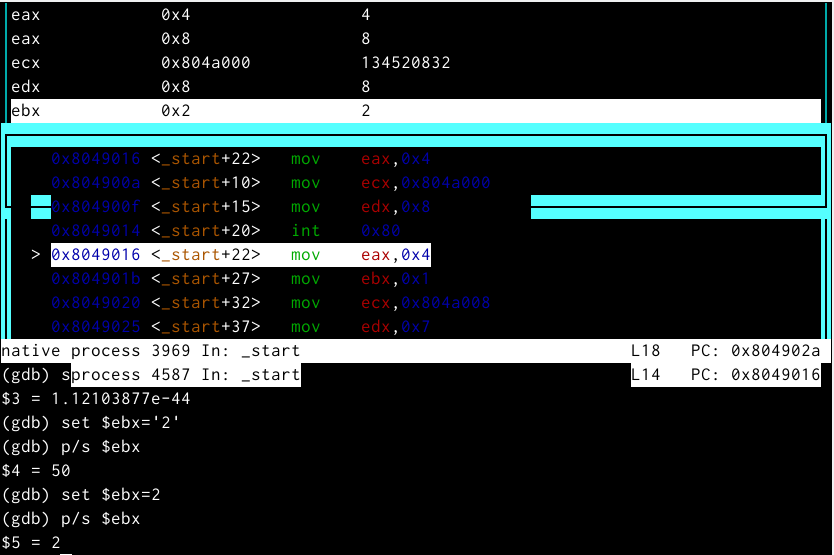


Рис. 17: Вывод значения регистра в разных представлениях и использование команды set для изменения значения регистра

Разница вывода команд p/s $ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit. (рис. [18](#fig:018))



Рис. 18: Завершение работы GDB

### 4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm (рис. [19](#fig:019))

Figure 19: Копирование файла

Рис. 19: Копирование файла

И затем создаю исполняемый файл. (рис. [20](#fig:020))

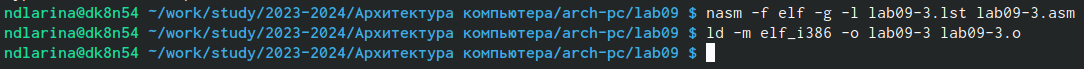


Рис. 20: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа –args. (рис. [21](#fig:021))

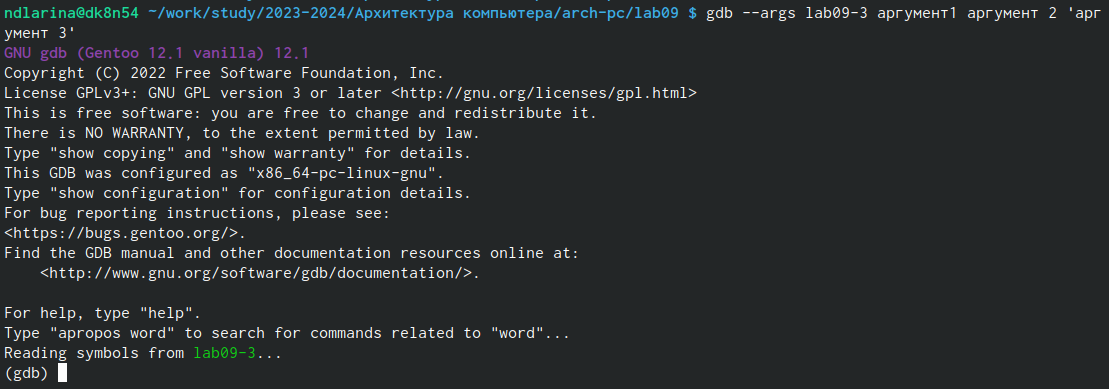


Рис. 21: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [22](#fig:022))

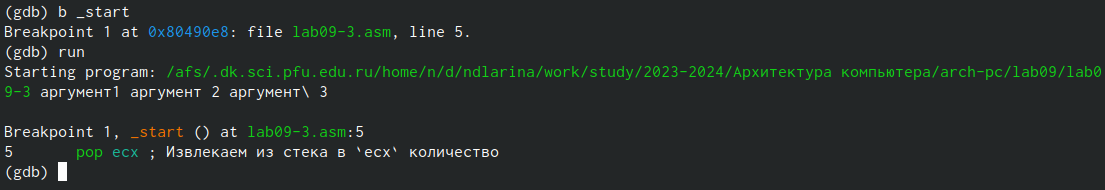


Рис. 22: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. [23](#fig:023))

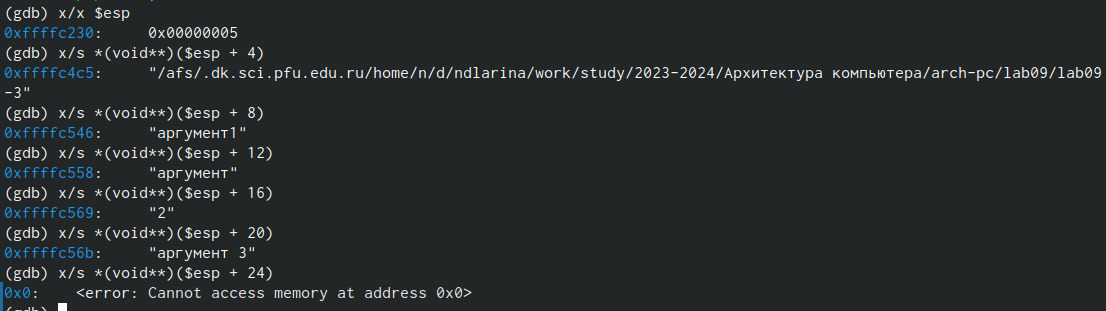


Рис. 23: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

## 4.3 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. [24](#fig:024))

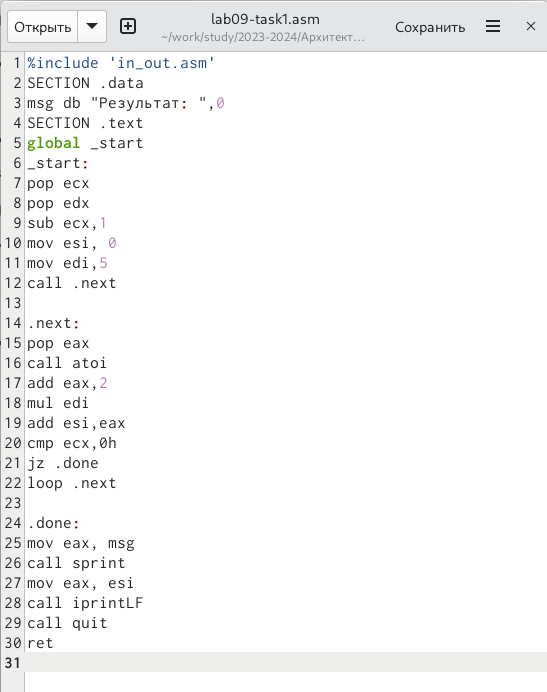


Рис. 24: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. [25](#fig:025))

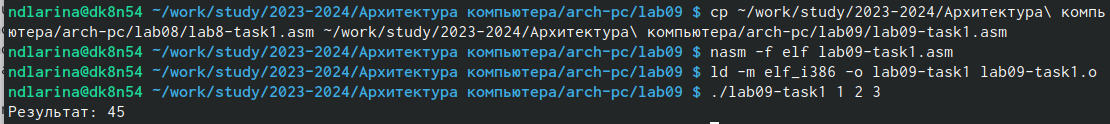


Рис. 25: Запуск программы и проверка его вывода

1. Ввожу в файл lab09-task2.asm текст программы из листинга 9.3. (рис. [26](#fig:026))

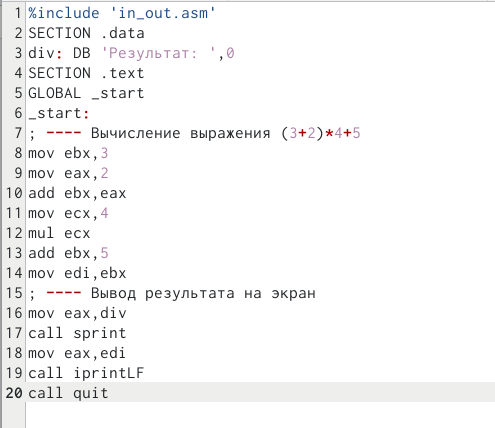


Рис. 26: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводится “25”. Создаю исполняемый файл и запускаю его. (рис. [27](#fig:027))

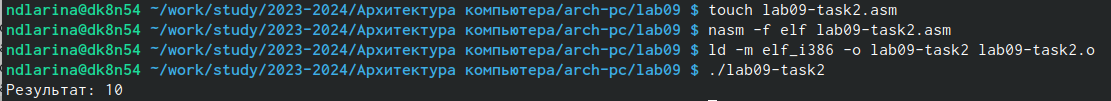


Рис. 27: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции mul ecx происходит умножение ecx на eаx, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov ecx,4 инструкция add ebx,eax не связана с mul ecx, но связана инструкция mov eax,2. (рис. [28](#fig:028))

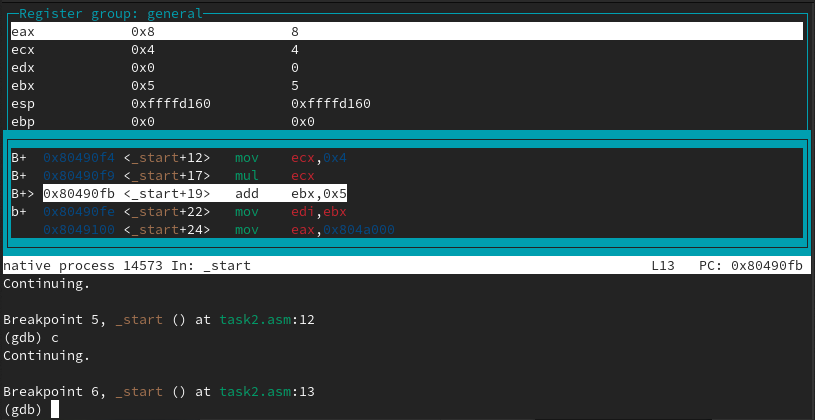


Рис. 28: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (рис. [29](#fig:029))

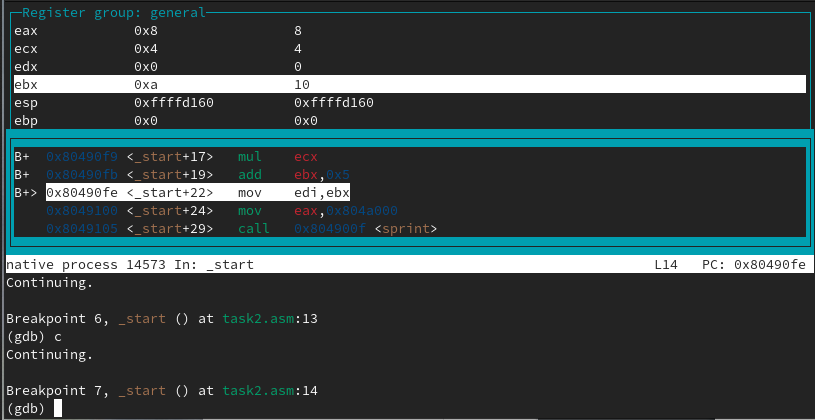


Рис. 29: Неверное изменение регистра

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx. (рис. [30](#fig:030))

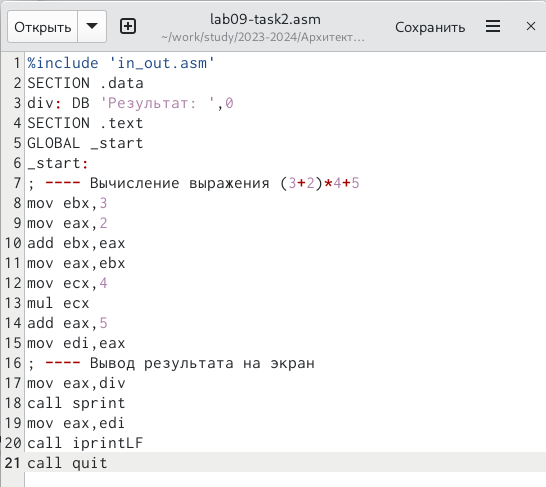


Рис. 30: Исправление ошибки

Также, вместо того, чтобы изменять значение еах, можно было изменять значение неиспользованного регистра edx.

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена. (рис. [31](#fig:031))

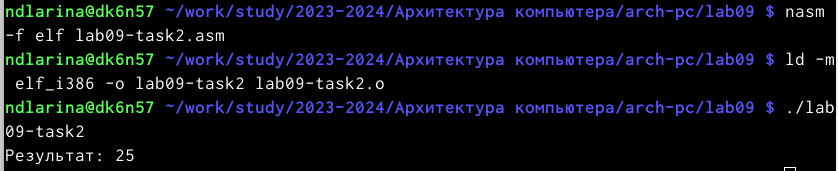


Рис. 31: Ошибка исправлена

# 5 Выводы

В ходе работы над данной лабораторной работы мне удалось приобрести навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

# 6 Список литературы

1. [Лабораторная работа №9](https://esystem.rudn.ru/mod/resource/view.php?id=1030557)