Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Бизев Никита Владимирович

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. Отладчики позволяют управлять ходом выполнения программы, контролировать 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.

GDB (GNU Debugger — отладчик проекта GNU) 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 сторонних графических 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 происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB.

Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда quit (или сокращённо q).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой info (кратко i).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable.

Обратно точка останова активируется командой enable.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.

Для продолжения остановленной программы используется команда continue (c). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N − 1 точку останова (выполнение остановится на N-й точке).

Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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 переход. 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.

# 4 Выполнение лабораторной работы

## 4.1 **Реализация подпрограмм в NASM**

Создаю файл lab09-1.asm. (рис. 1)

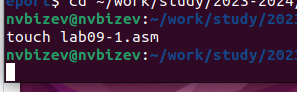


Рис 1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 2)

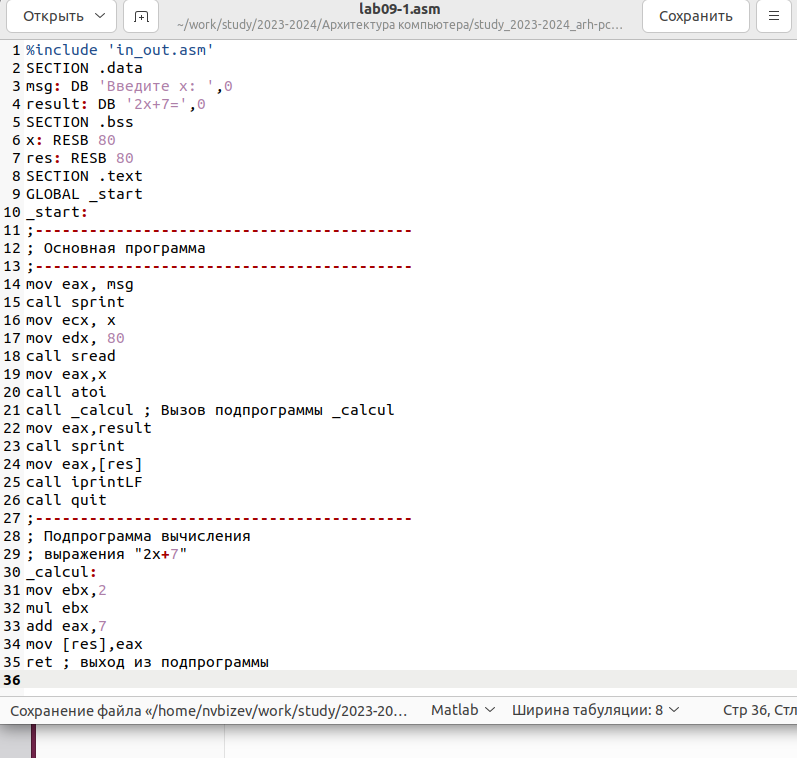


Рис 2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 3)

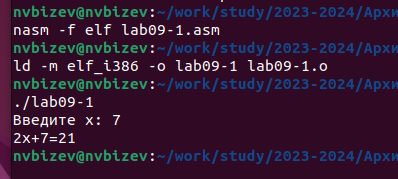


Рис 3: Запуск исполняемого файла

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)

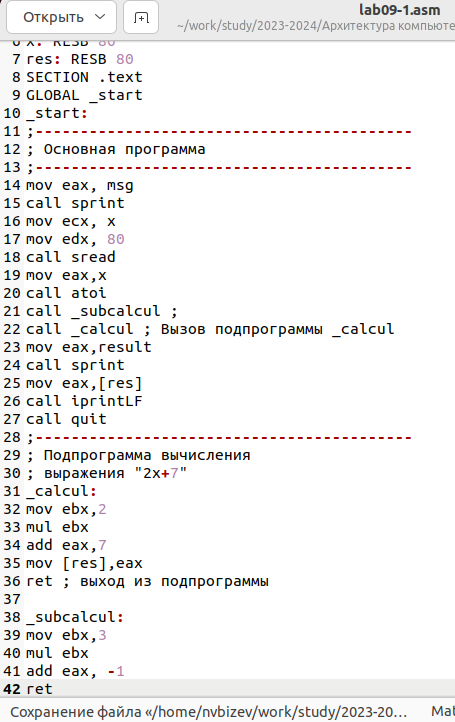


Рис 4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. 5)

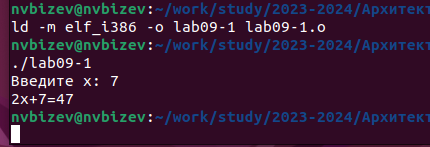


Рис 5: Запуск исполняемого файла

## 4.2 **Отладка программам с помощью GDB**

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 6)

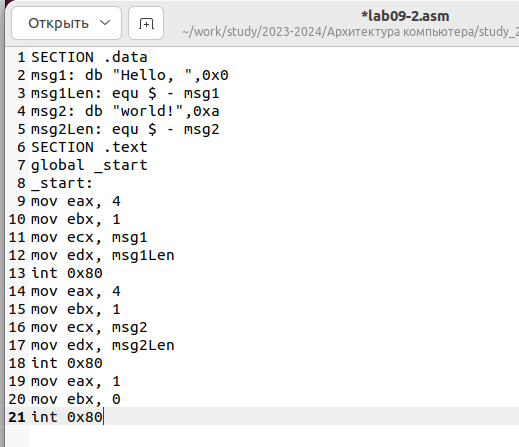


Рис 6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом ‘-g’. (рис. 7)

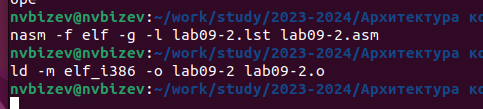


Рис 7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 8)

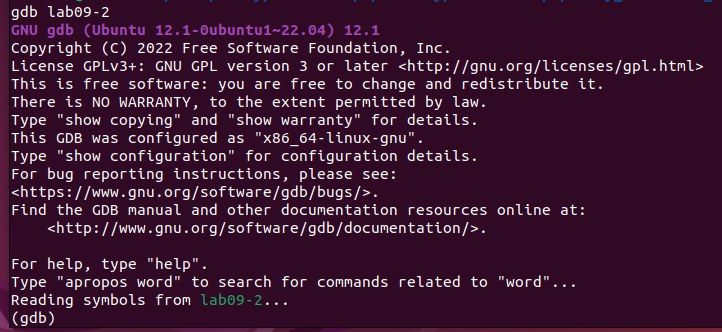


Рис 8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. 9)

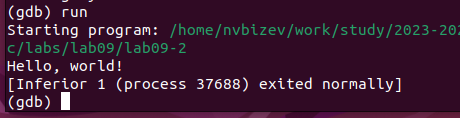


Рис 9: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её. (рис. 10)

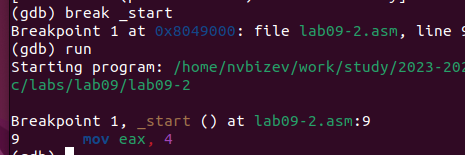


Рис 10: Установка брейкпоинта и запуск программы

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)

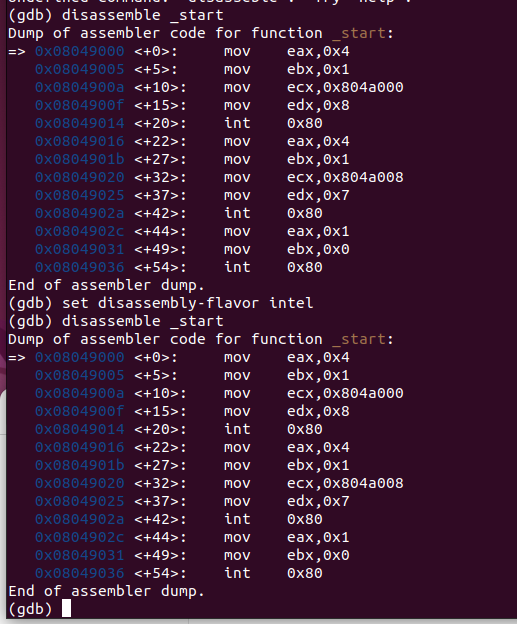


Рис 11: Использование команд disassemble и disassembly-flavor intel

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)

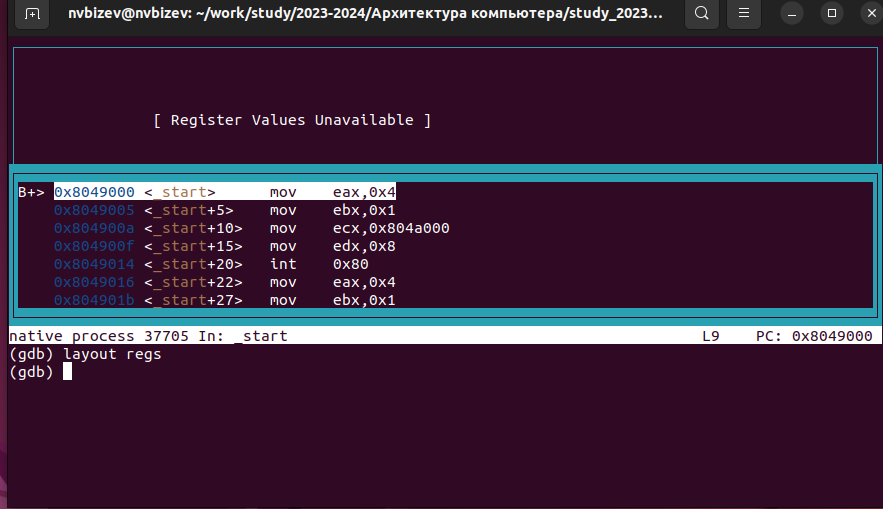


Рис 12: Включение режима псевдографики

### 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)

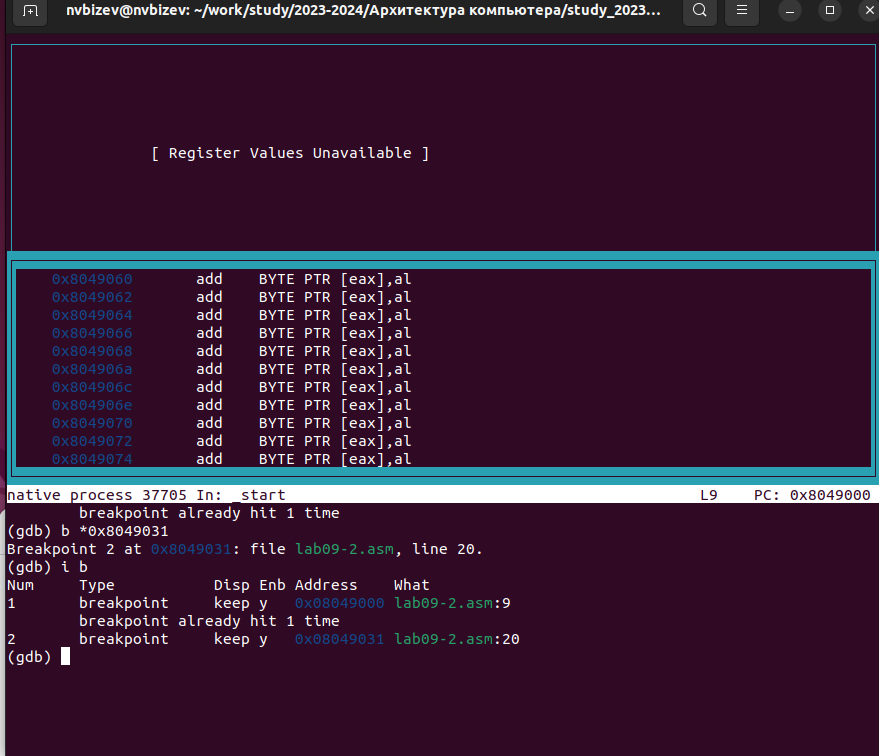


Рис 13: Установление точек останова и просмотр информации о них

### 4.2.2 **Работа с данными программы в GDB**

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением значений регистров. (рис. 14)

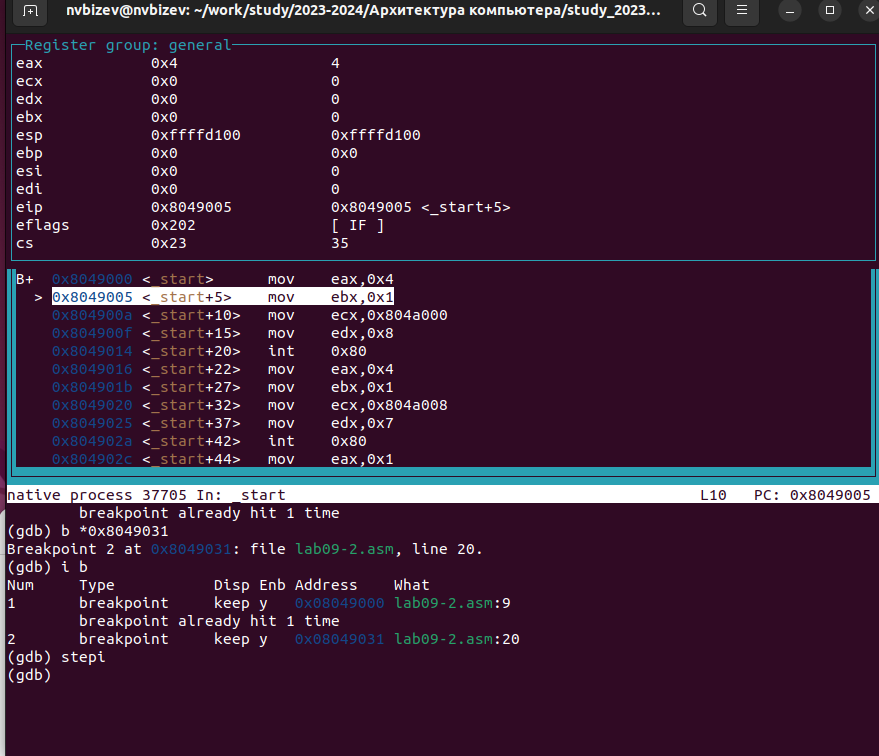


Рис 14: До использования команды stepi

(рис. 15)

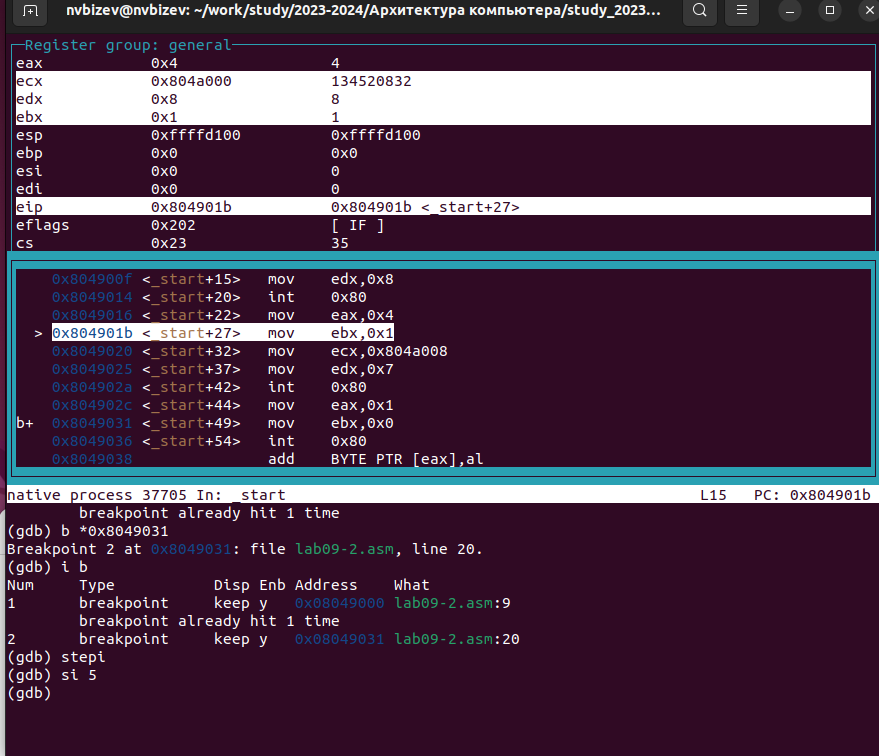


Рис 15: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. 16)

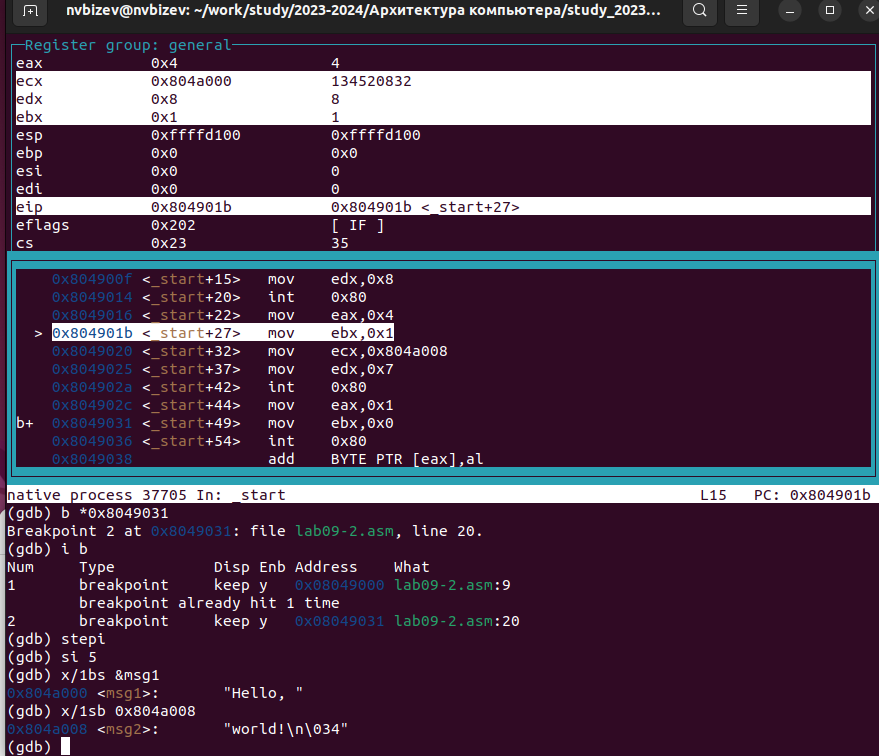


Рис 16: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. 17)

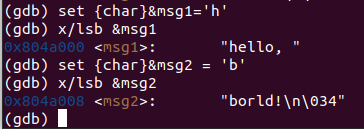


Рис 17: Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F $val. (рис. 18)

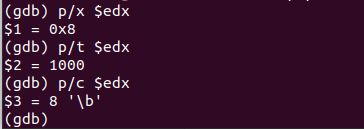


Рис 18: Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. 19)

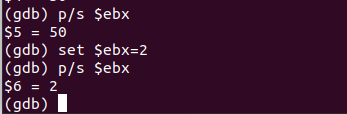


Рис 19: Использование команды set для изменения значения регистра

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. 20)

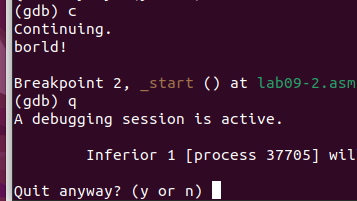


Рис 20: Завершение работы GDB

### 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 и создаю исполняемый файл. (рис. 21)

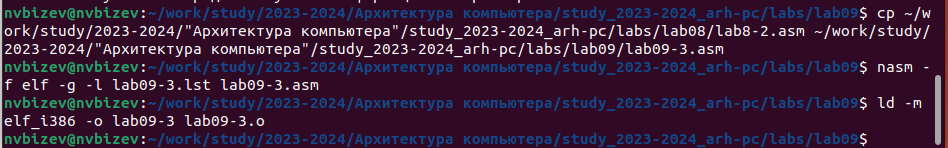


Рис 21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа –args. (рис. 22)

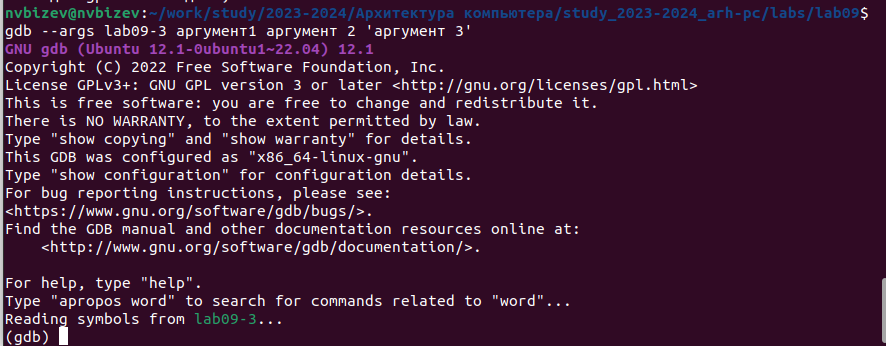


Рис 22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 23)

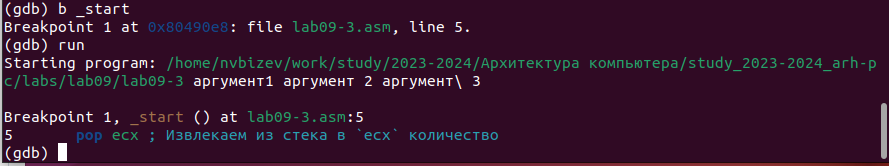


Рис 23: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. 24)

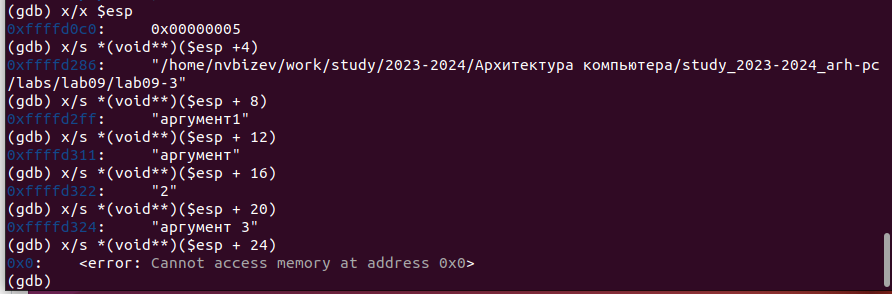


Рис 24: Просмотр значений, введенных в стек

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

# 5 Выводы

Во время выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи GDB и его основными возможностями.

# 6 Список литературы

1. https://esystem.rudn.ru/pluginfile.php/2089096/mod\_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20..pdf