Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

# 1 Цель работы

Целью данной лабораторной работы является 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 программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Выполнение заданий 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 каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. 1).

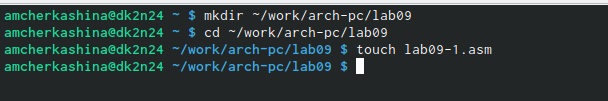


Рис. 1: Создание каталога и файла

С помощью команды cp копирую в текущий каталог файл in\_out.asm, т.к. он будет использоваться в программах данной лабораторной работы (рис. 2).

Создание копии файла

Рис. 2: Создание копии файла

Ввожу в файл lab09-1.asm текст программы с использованием вызова подпрограммы из листинга 9.1 (рис. 3).

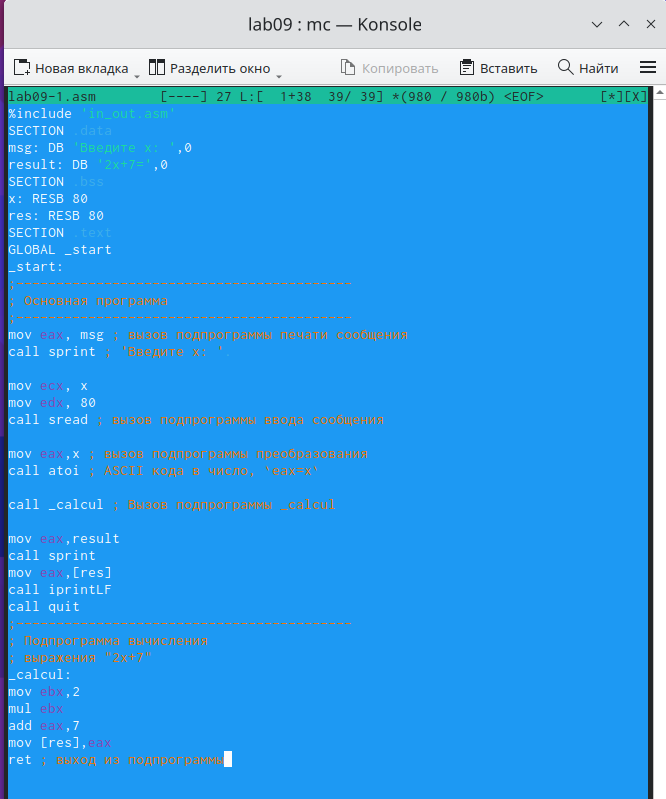


Рис. 3: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. 4).

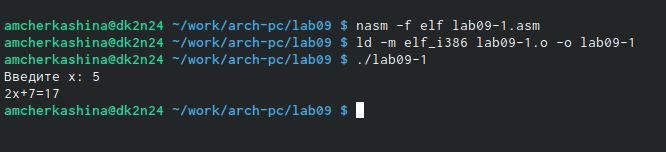


Рис. 4: Запуск исполняемого файла

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. 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. 5).



Рис. 5: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 6).

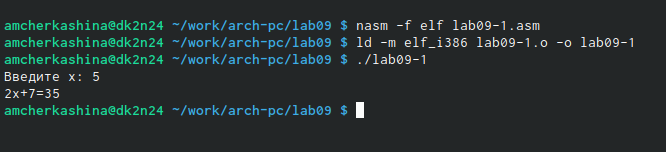


Рис. 6: Запуск нового исполняемого файла

## 4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm и ввожу в него текст программы вывода сообщения Hello world! из листинга 9.2 (рис. 7).

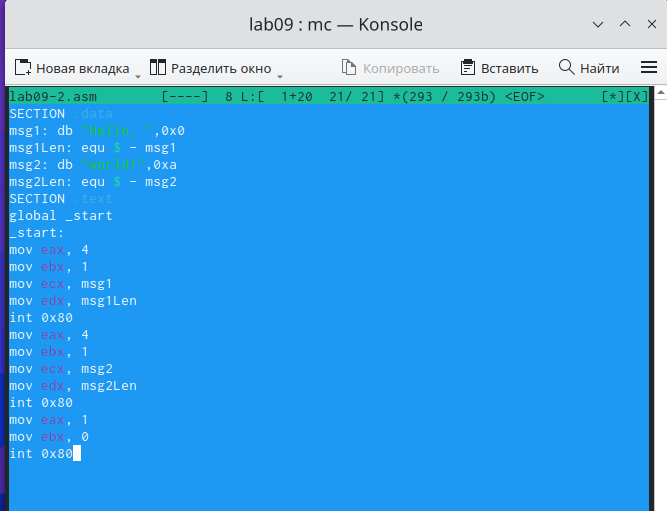


Рис. 7: Редактирование файла

Получаю исполняемый файл для работы с GDB добавив в него отладочную информацию. Для этого трансляцию программ провожу с ключом -g (рис. 8).

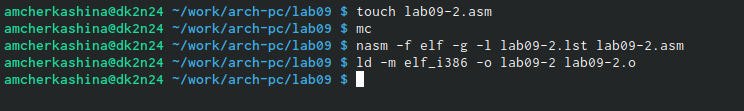


Рис. 8: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. 9).

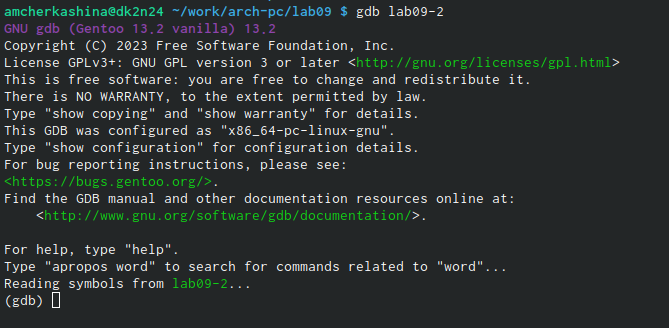


Рис. 9: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 10).

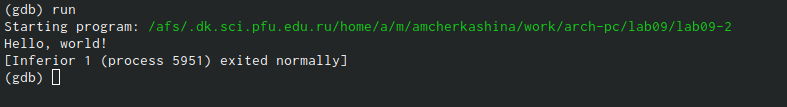


Рис. 10: Запуск программы в GDB

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение ассемблерной программы, и запускаю её (рис. 11).

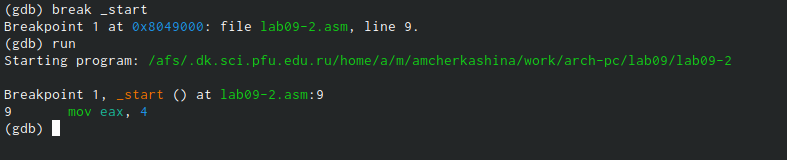


Рис. 11: Установка брейкпоинта и запуск программы

Просматриваю дизассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. 12).

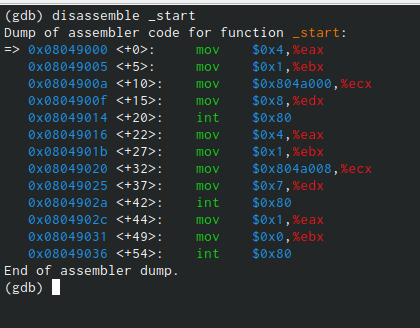


Рис. 12: Дизассемблированный код программы

Переключаюсь на отображение команд с синтаксисом Intel и снова просматриваю дизассимилированный код программы (рис. 13).



Рис. 13: Дизассемблированный код программы с синтаксисом 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. 14).

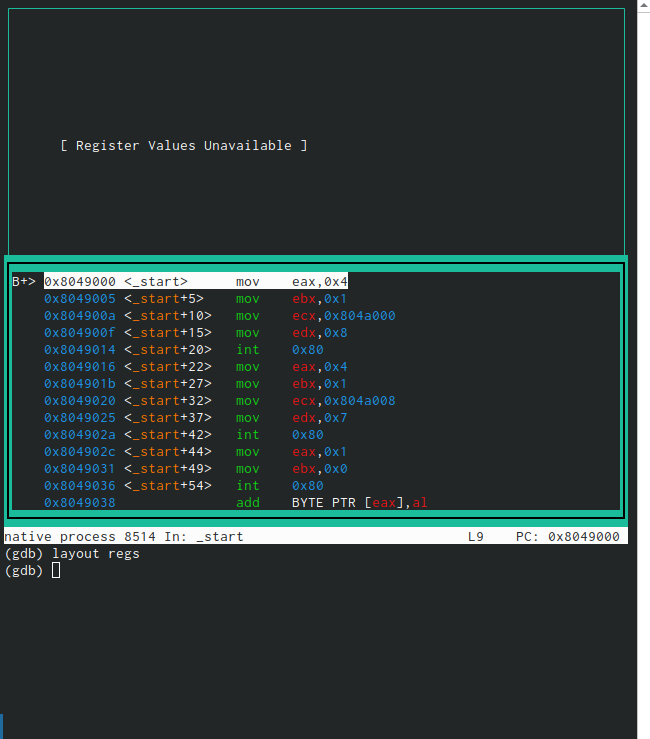


Рис. 14: Включение режима псевдографики

В этом режиме есть три окна: • В верхней части видны названия регистров и их текущие значения; • В средней части виден результат дисассимилирования программы; • Нижняя часть доступна для ввода команд

## 4.3 Добавление точек останова

Ранее мной была установлена точка останова по имени метки \_start. Проверяю это с помощью команды info breakpoints (кратко i b) (рис. 15).

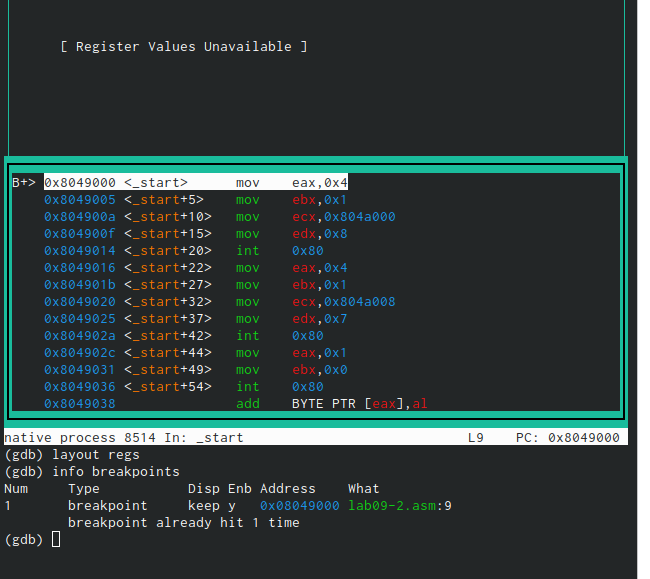


Рис. 15: Просмотр информации о точках останова

Устанавливаю еще одну точку останова по адресу предпоследней инструкции (mov ebx,0x0). Адрес инструкции нахожу в средней части экрана в левом столбце соответствующей инструкции. Просматриваю информацию о всех установленных точках останова с помощью i b - сокращения команды info breakpoints (рис. 16).

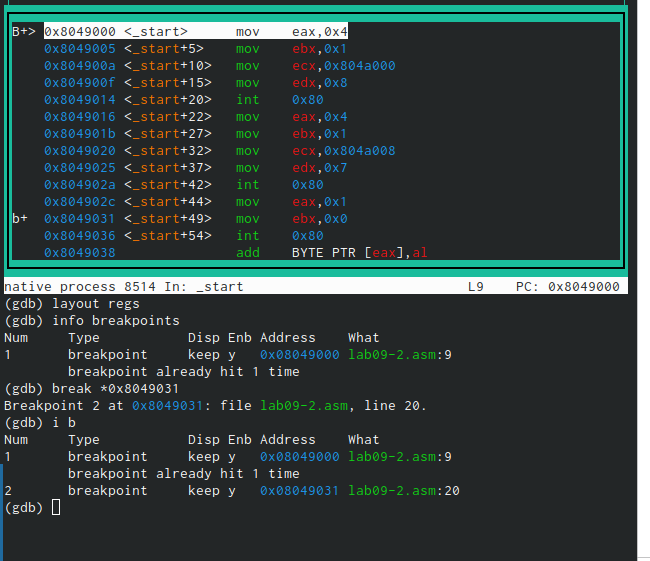


Рис. 16: Установление брейкпоинта

## 4.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info registers (i r) (рис. 17).

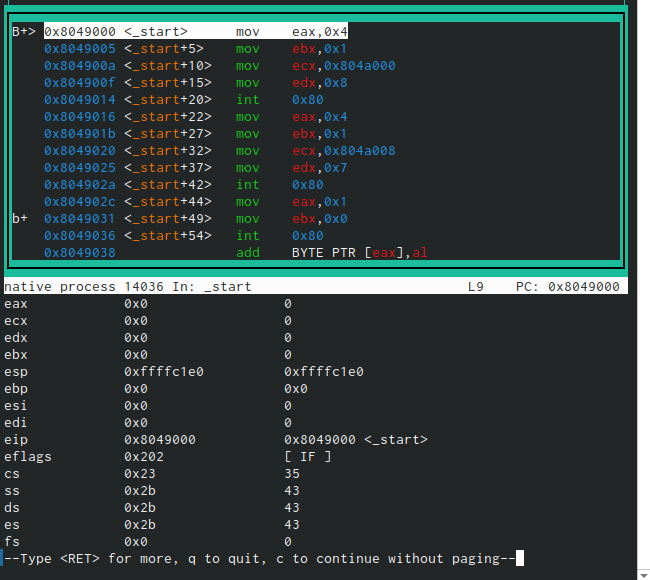


Рис. 17: Просмотр содержимого регистров

Выполняю 5 инструкций с помощью команды stepi (si) (рис. 18).

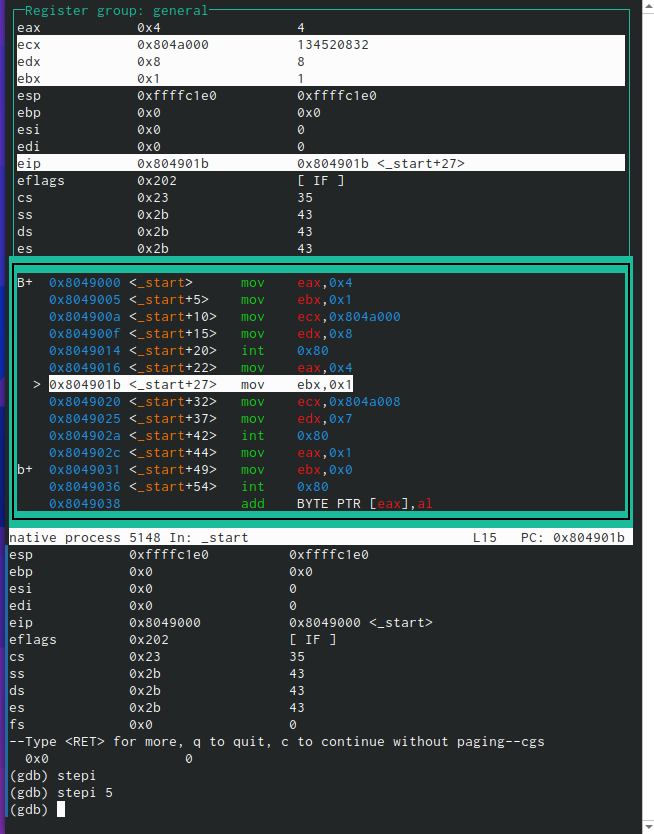


Рис. 18: Выполнение инструкций с помощью stepi

Изменились значения регистров ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1, а затем просматриваю значение переменной msg2 по ее адресу(x/1sb ). Адрес переменной определяю по дизассемблированной инструкциии (рис. 19).

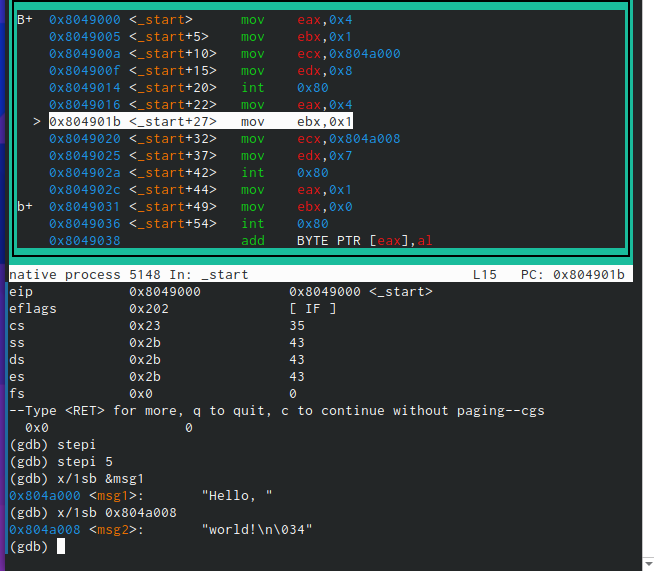


Рис. 19: Просмотр значений переменных

С помощью команды set заменяю первый символ переменной msg1 на ‘h’, а далее изменяю один символ второй переменной msg2 (рис. 20).

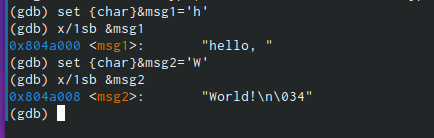


Рис. 20: Замена символов переменных

С помощью команды print print/F $val вывожу значение регистра edx в трех различных форматах: в шестнадцатеричном формате, в двоичном формате и в символьном виде (рис. 21).

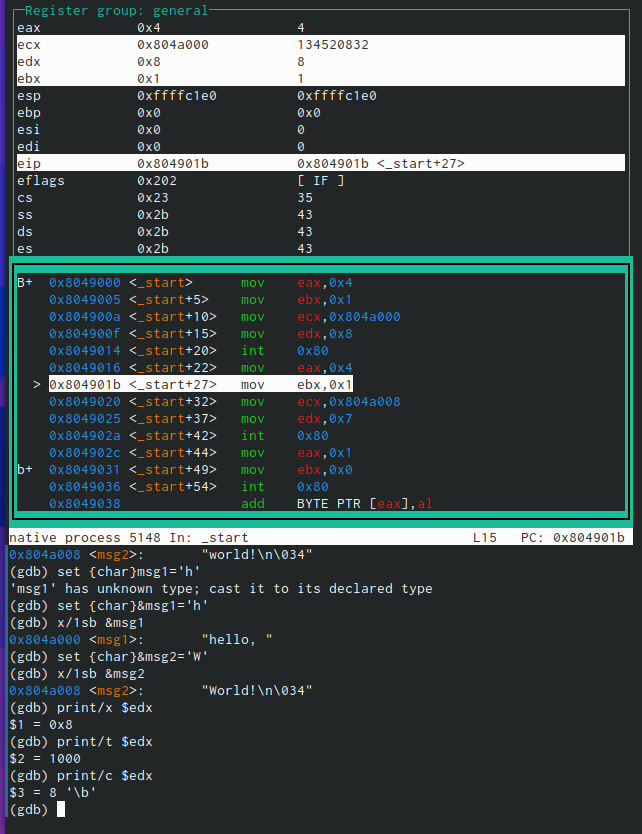


Рис. 21: Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием и вывожу его значения (рис. 22).

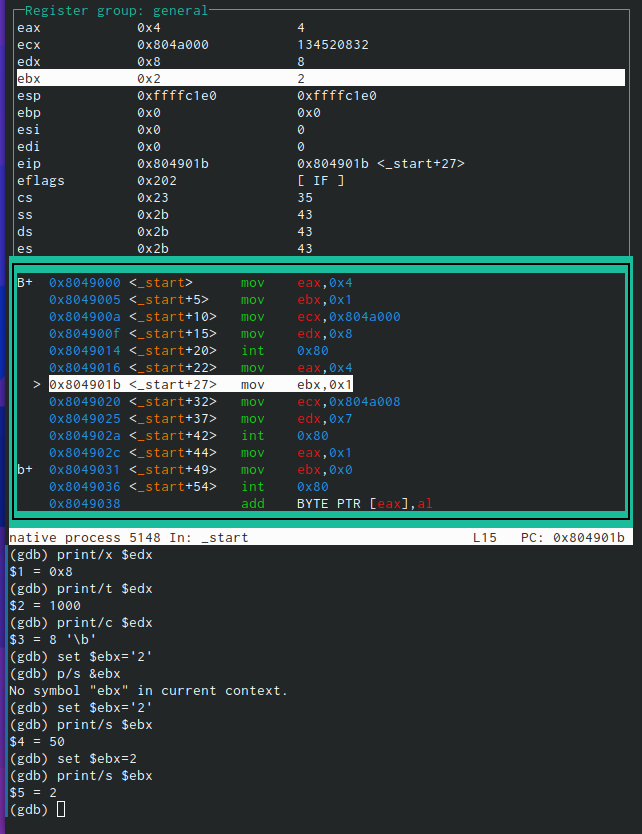


Рис. 22: Изменение значения регистра

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. 23).

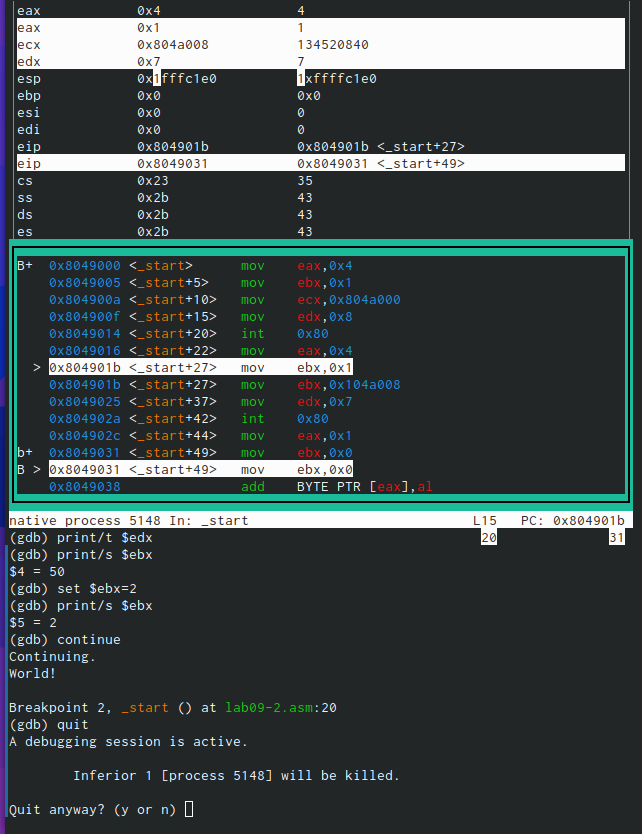


Рис. 23: Завершение работы GDB

## 4.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm (рис. 24).

Создание копии файла

Рис. 24: Создание копии файла

Создаю исполняемый файл (рис. 25).

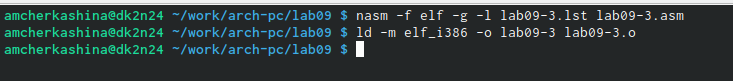


Рис. 25: Создание исполняемого файла

Загружаю исполняемый файл программы с аргументами в отладчик, используя для этого ключ–args (рис. 26).

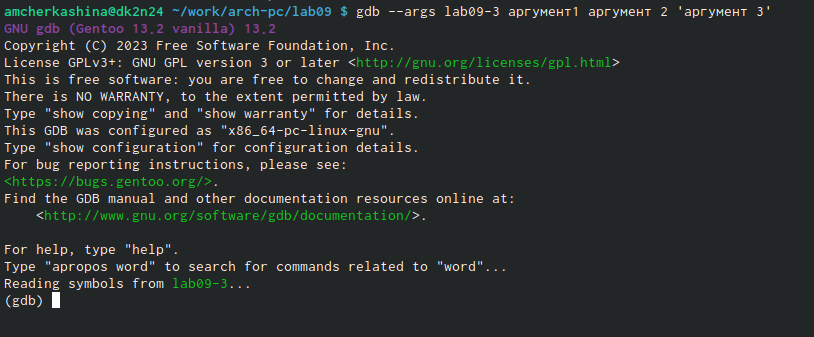


Рис. 26: Загрузка исполняемого файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 27).

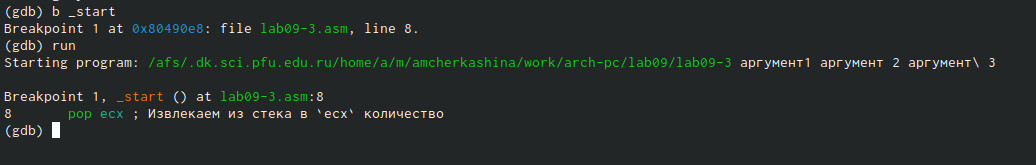


Рис. 27: Установка брейкпоинта и запуск программы

Посматриваю вершину стека и остальные позиции стека по их адресам (по адресу [esp+4] находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д) (рис. 28).

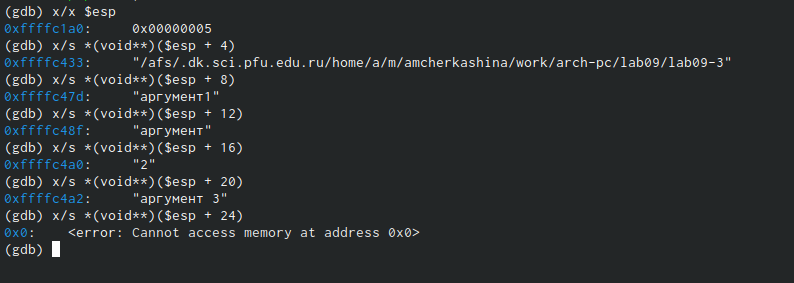


Рис. 28: Просмотр значений, введенных в стек

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

## 4.6 Выполнение заданий для самостоятельной работы

1. Копирую файл lab8-4.asm, созданный при выполнении самостоятельного задания лабораторной работы №8, в файл с именем lab09-4.asm Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. У меня был 11 вариант: f(x) = 15\*x+2 (рис. 29).

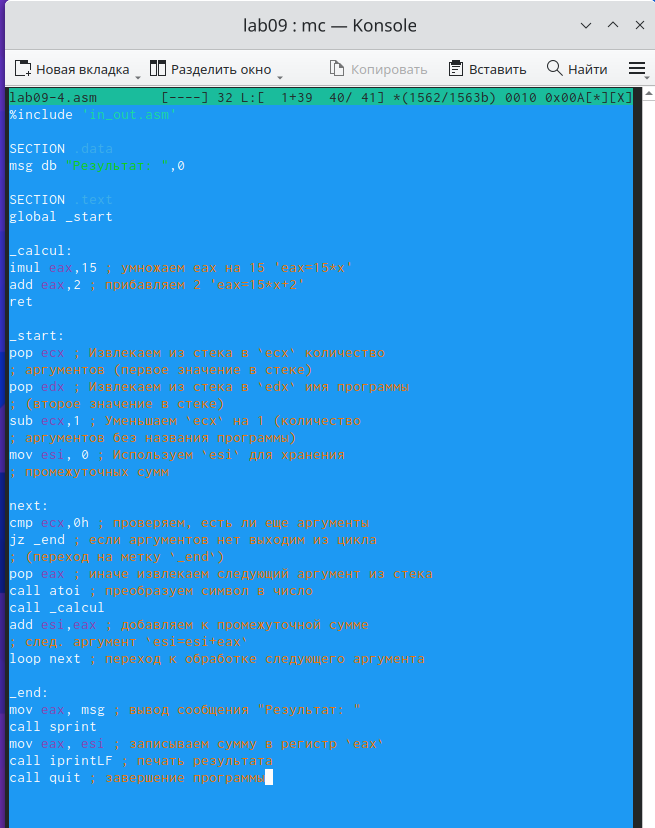


Рис. 29: Написание кода программы

Листинг 4.1. Программа нахождения суммы значений функции f(x)=15\*x+2

%include 'in\_out.asm'  
  
SECTION .data  
 msg db "Результат: ", 0  
  
SECTION .text  
 global \_start  
  
\_calcul:  
 imul eax, 15 ; умножаем eax на 15 'eax=15\*x'  
 add eax, 2 ; прибавляем 2 'eax=15\*x+2'  
 ret  
  
\_start:  
 pop ecx ; Извлекаем из стека в ecx количество  
 ; аргументов (первое значение в стеке)  
 pop edx ; Извлекаем из стека в edx имя программы  
 ; (второе значение в стеке)  
 sub ecx, 1 ; Уменьшаем ecx на 1 (количество  
 ; аргументов без названия программы)  
 mov esi, 0 ; Используем esi для хранения  
 ; промежуточных сумм  
  
next:  
 cmp ecx, 0h ; проверяем, есть ли еще аргументы  
 jz \_end ; если аргументов нет, выходим из цикла  
 ; (переход на метку \_end)  
 pop eax ; иначе извлекаем следующий аргумент из стека  
 call atoi ; преобразуем символ в число  
 call \_calcul ; вызываем подпрограмму для вычисления f(x)  
 add esi, eax ; добавляем значение функции для   
 ; конкретного аргумента к промежуточной сумме  
 loop next ; переход к обработке следующего аргумента  
  
\_end:  
 mov eax, msg ; вывод сообщения "Результат: "  
 call sprint  
 mov eax, esi ; записываем сумму в регистр eax  
 call iprintLF ; печать результата  
 call quit ; завершение программы

# 5 Выводы

В ходе выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

# 6 Список литературы

1. Архитектура ЭВМ