Лабораторная работа 9

Понятие подпрограммы. Отладчик

Головина Мария Игоревна

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. Создать каталог для выполнения лабораторной работы № 9, перейти в него и создать файл lab09-1.asm.
2. Ввести в файл lab09-1.asm текст программы из листинга 1 методического указания. Создать исполняемый файл и проверить его работу.
3. Изменить текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2х + 7, g(x) = 3х + 1. Т.е. х 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.
4. Создать файл lab09-2.asm с текстом программы из Листинга 2 методического указания.
5. Получить исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию. Загрузить исполняемый файл в отладчик.
6. Проверить работу программы, запустив ее в оболочке GDB.
7. Установить брейкпоинт на метку \_start и запустить её.
8. Посмотреть дисассимилированный код программы.
9. Переключить на отображение команд с Intel’овским синтаксисом.
10. Включить режим псевдографики.
11. Установить несколько точек останова.
12. Выполнить 5 инструкций с помощью команды si.
13. Посмотреть значение переменных msg1 и msg2.
14. Изменить значение переменных msg1 и msg2.
15. Вывести в различных форматах значение регистра edx.
16. Изменить значение регистра ebx.
17. Скопировать 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. Создать исполняемый файл.
18. Загрузить исполняемый файл в отладчик, указав аргументы.
19. Исследовать расположение аргументов командной строки в стеке после запуска программы.
20. Проверить адрес вершины стека и посмотреть все позиции стека.

Задание для самостоятельной работы

1. Преобразовать программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.
2. В листинге из самостоятельного задания приведена программа вычисления выражения (3 + 2) \* 4 + 5. При запуске данная программа дает неверный результат. Проверить это. С помощью отладчика GDB, анализируя изменения значений регистров, определить ошибку и исправить ее.

# 3 Теоретическое введение

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 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 прерывание 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.

Наиболее часто применяют следующие методы отладки:

создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

использование специальных программ-отладчиков.

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.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить 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 программистом);

Watchpoint — точка просмотра (выполнение программы приостанавливается, 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.

Основные возможности отладчика GDB

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 сторон них 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. GDB может выполнять следующие действия:

начать выполнение программы, задав всё, что может повлиять на её поведение;

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.

Дизассемблирование программы

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.

Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel’овским синтаксисом можно, введя команду set disassembly-flavor intel.

Точки останова

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.

Пошаговая отладка

Для продолжения остановленной программы используется команда continue. 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.

При указании в качестве аргумента целого числа N отладчик выполнит команду step N раз при условии, что не будет точек останова или выполнение программы не прервётся по другим причинам.

Команда nexti (или ni) аналогична stepi, но вызов процедуры (функции) трактуется отладчиком как одна инструкция.

Работа с данными программы в GDB

Как уже упоминалось, отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды info registers (или i r).

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 (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си).

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.

Инструкция call и инструкция ret

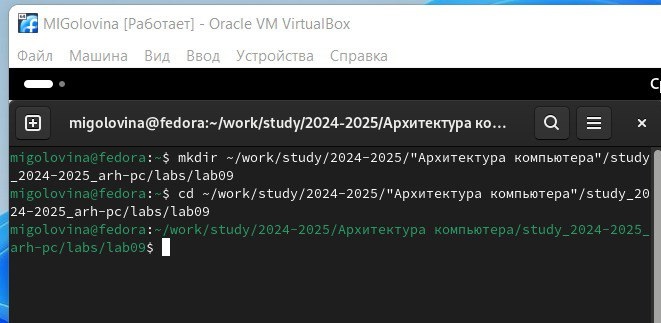
Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может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

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

# 4 Выполнение лабораторной работы

1. Создали каталог для выполнения лабораторной работы № 9, перешли в него и создали файл lab09-1.asm (рис. 4.1 Создание каталога для выполнения лабораторной работы).



Создание каталога для выполнения лабораторной работы

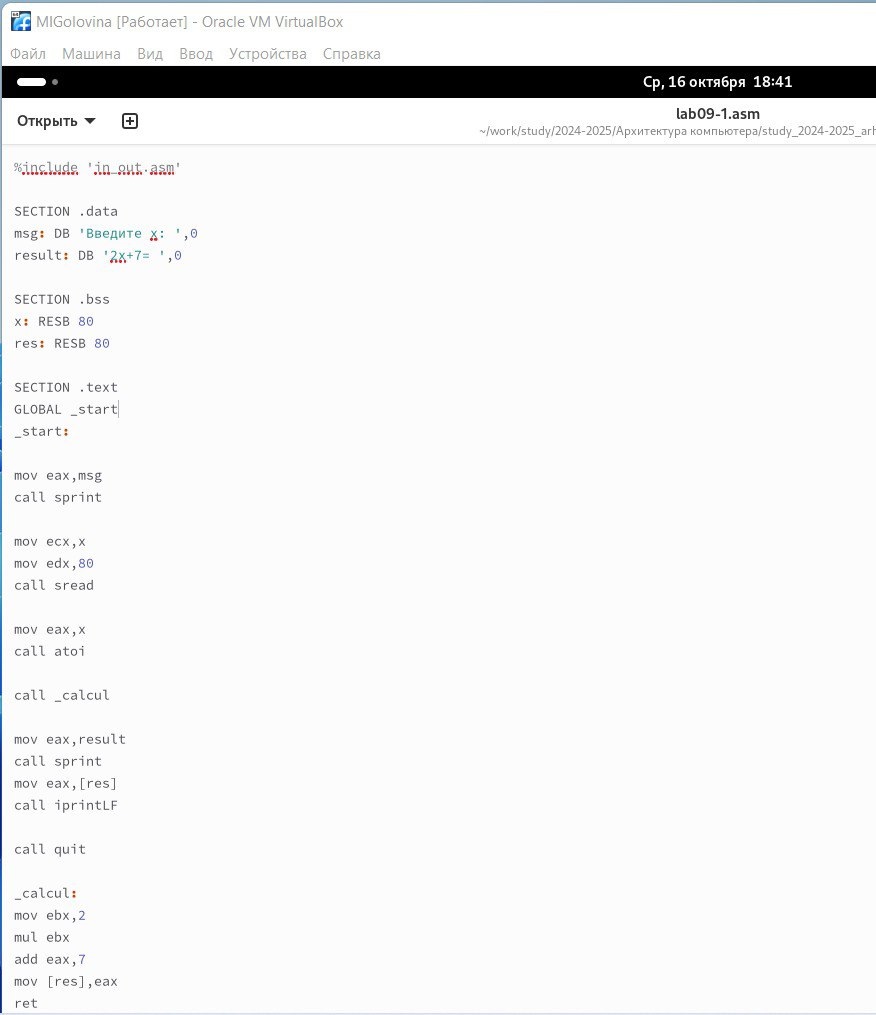
1. В качестве примера рассмотрели программу вычисления арифметического выражения f(x) = 2х + 7 с помощью подпрограммы \_calcul. В данном примере х вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучили текст программы из Листинга 1 методического указания (рис 4.2 Листинг1).

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).

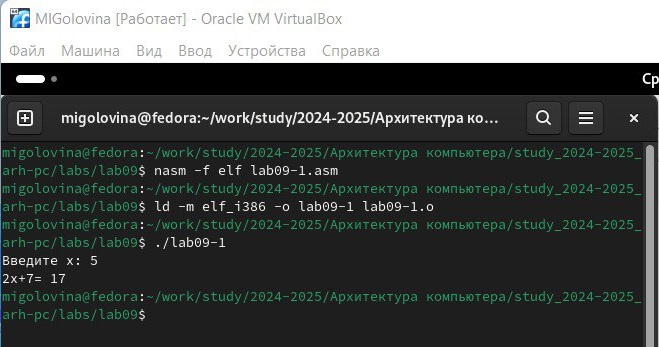
Инструкция ret является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму.

Последние строки программы реализуют вывод сообщения (call sprint), результата вычисления (call iprintLF) и завершение программы (call quit).

Ввели в файл lab09-1.asm текст программы из листинга 1 методического указания. Создали исполняемый файл и проверили его работу (рис. 4.3 Результаты работы программы из листинга 1 методического указания).

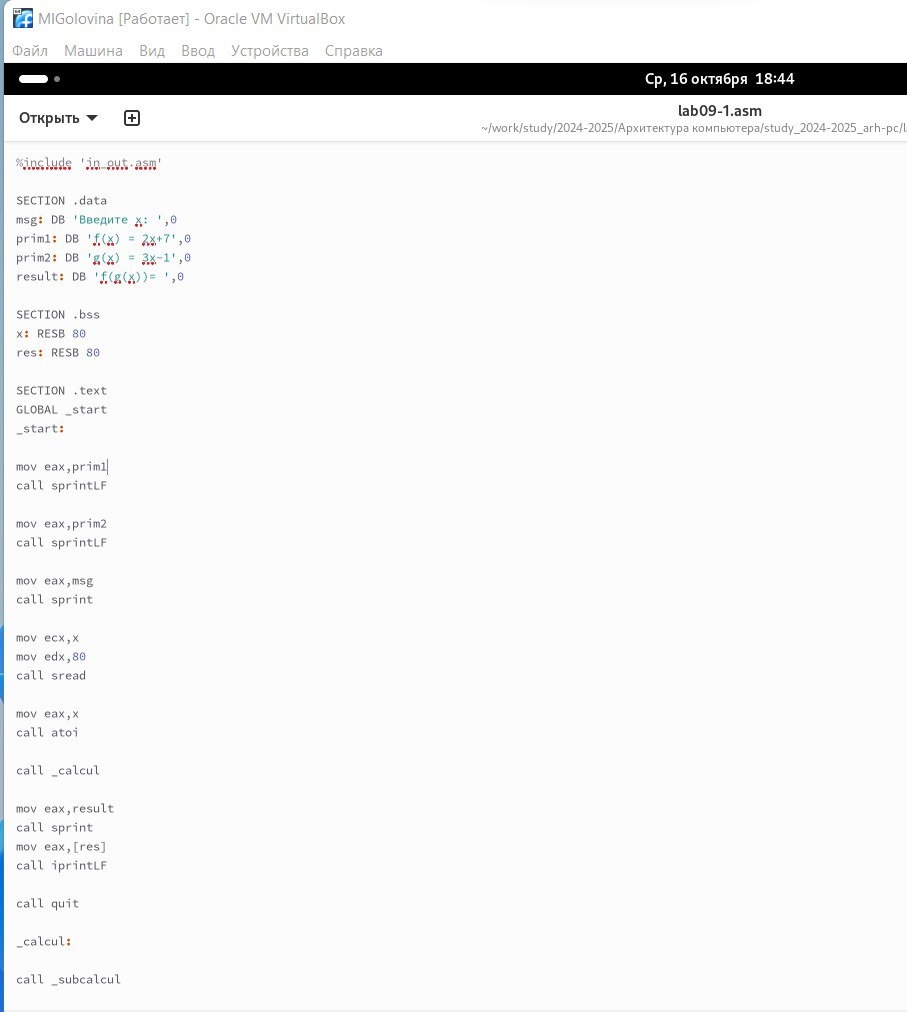


Листинг 1

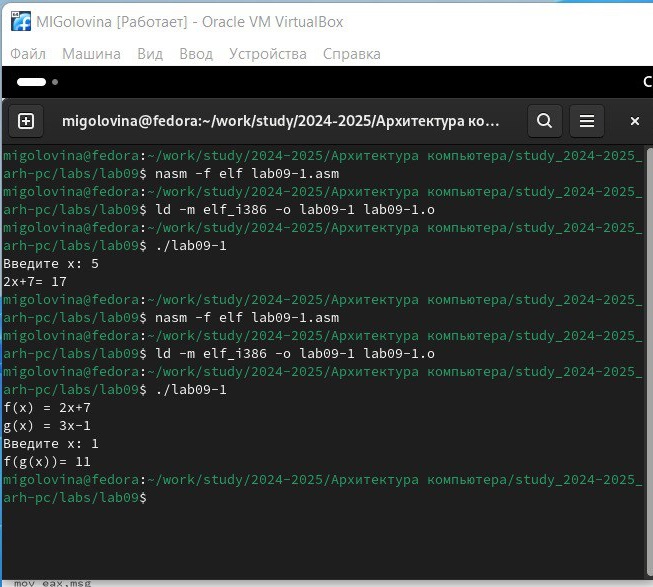


Результаты работы программы из листинга 1 методического указания

Изменили текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2х + 7, g(x) = 3х + 1. Т.е. х 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.

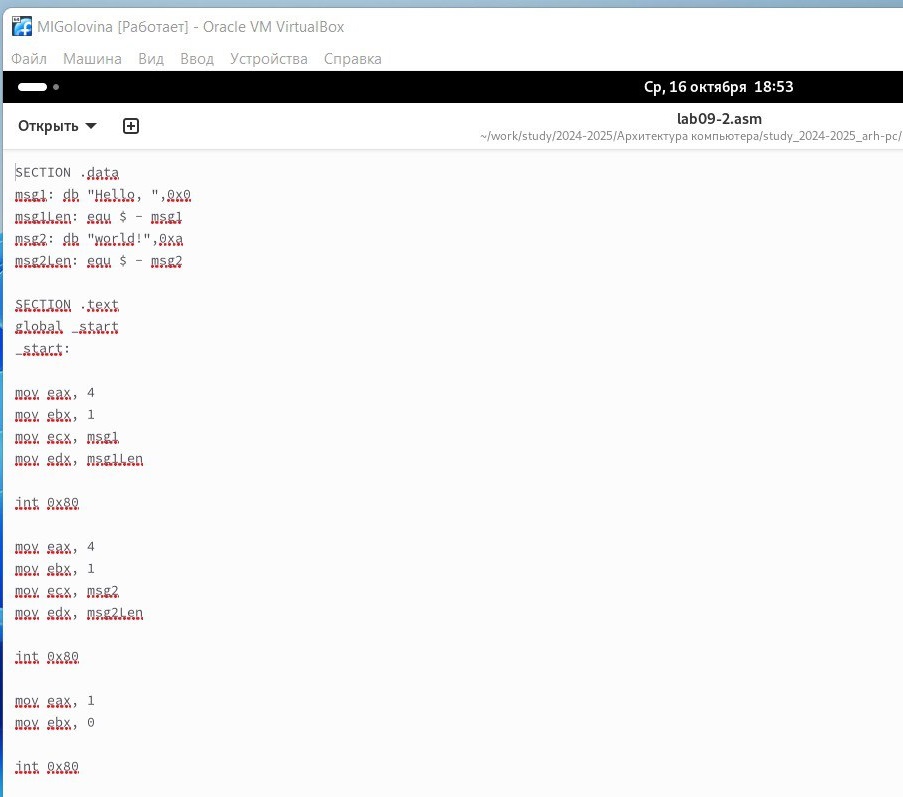


Листинг1 с внесенными изменениями



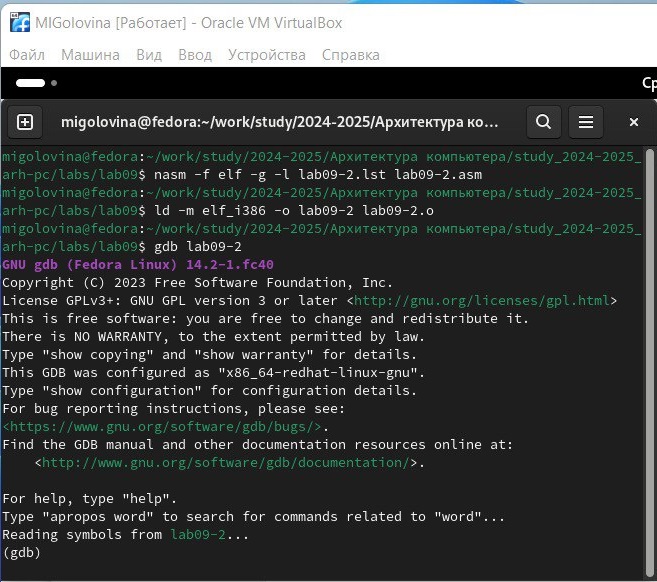
Результаты работы программы из листинга 1 с внесенными изменениями

1. Создали файл lab09-2.asm с текстом программы из Листинга 2 методического указания (рис. 4.6 Листинг 2).



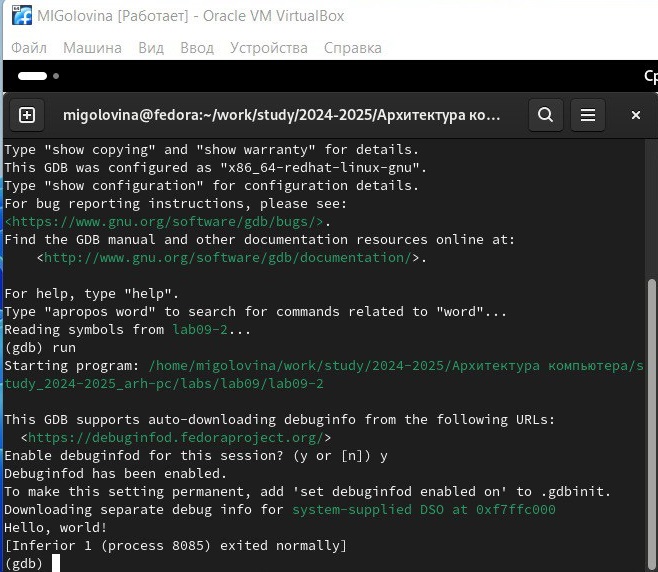
Листинг 2

Получили исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Загрузили исполняемый файл в отладчик gdb (рис. 4.7 Отладка программы из Листинга 2).



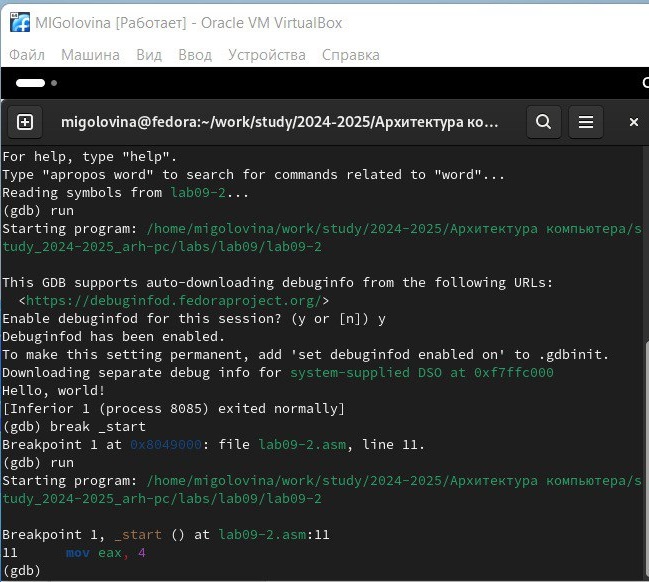
Отладка программы из Листинга 2

Проверили работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 4.8 Проверка работы программы с помощью команды run).



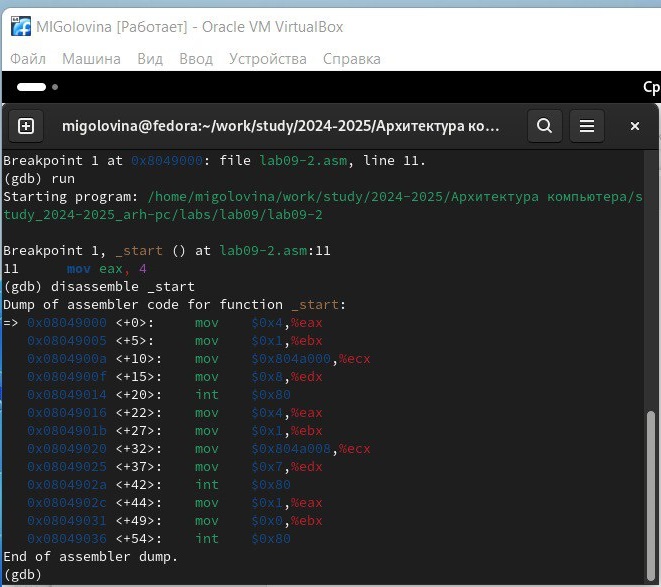
Проверка работы программы с помощью команды run

Для более подробного анализа программы установили брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустили её (рис. 4.9 Брейкпоинт на метку \_start).



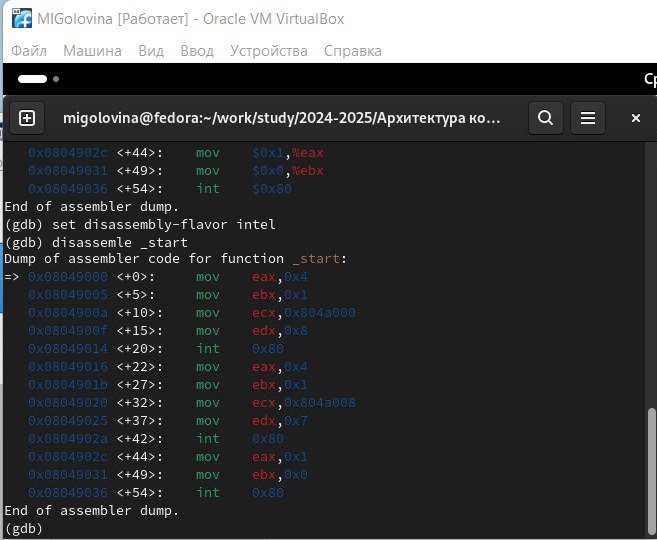
Брейкпоинт на метку \_start

Посмотрели дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 4.10 Дисассимилированный код программы).



Дисассимилированный код программы

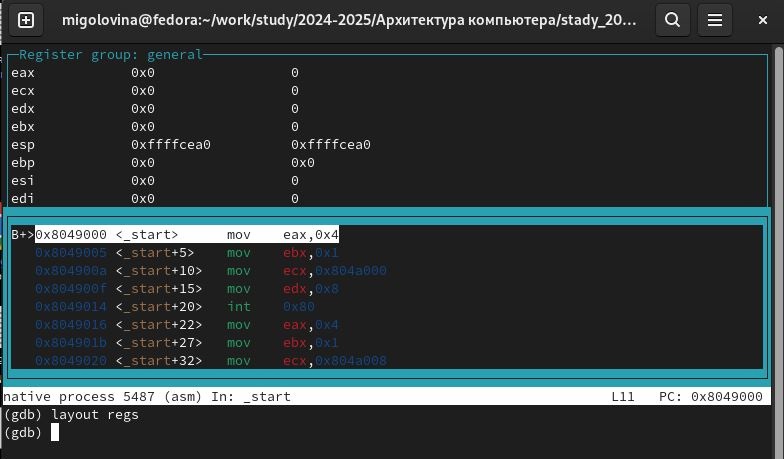
Переключились на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 4.11 Отображение команд с Intel’овским синтаксисом).



Отображение команд с Intel’овским синтаксисом

Отличие заключается в командах, в диссасимилированном отображении в командах используют % и $, а в Intel отображение эти символы не используются. На такое отображение удобнее смотреть.

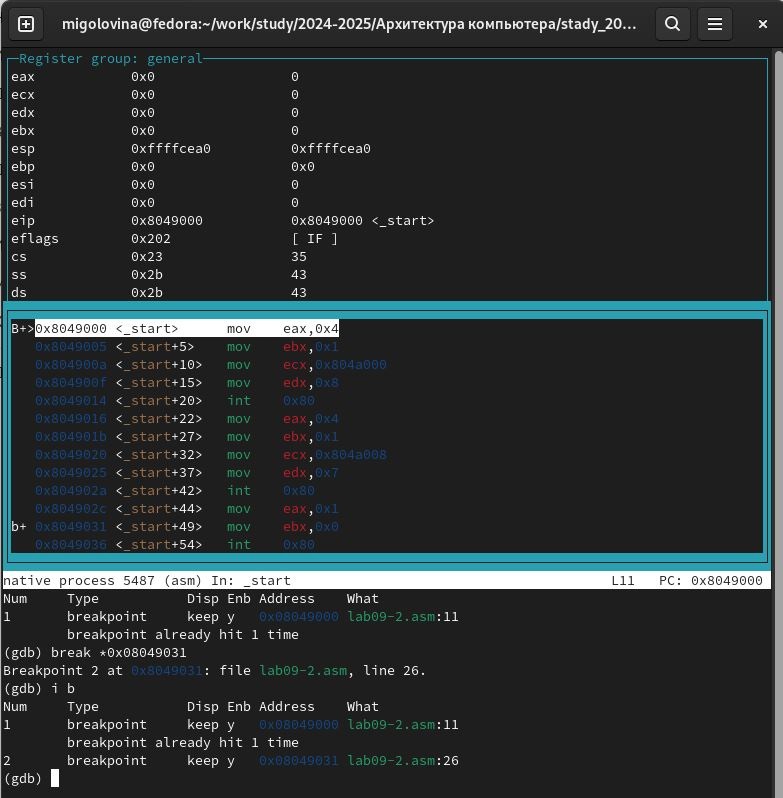
Включили режим псевдографики для более удобного анализа программы (рис. 4.12 Режим псевдографики). В этом режиме есть три окна: в верхней части видны названия регистров и их текущие значения; в средней части виден результат дисассимилирования программы; нижняя часть доступна для ввода команд



Режим псевдографики

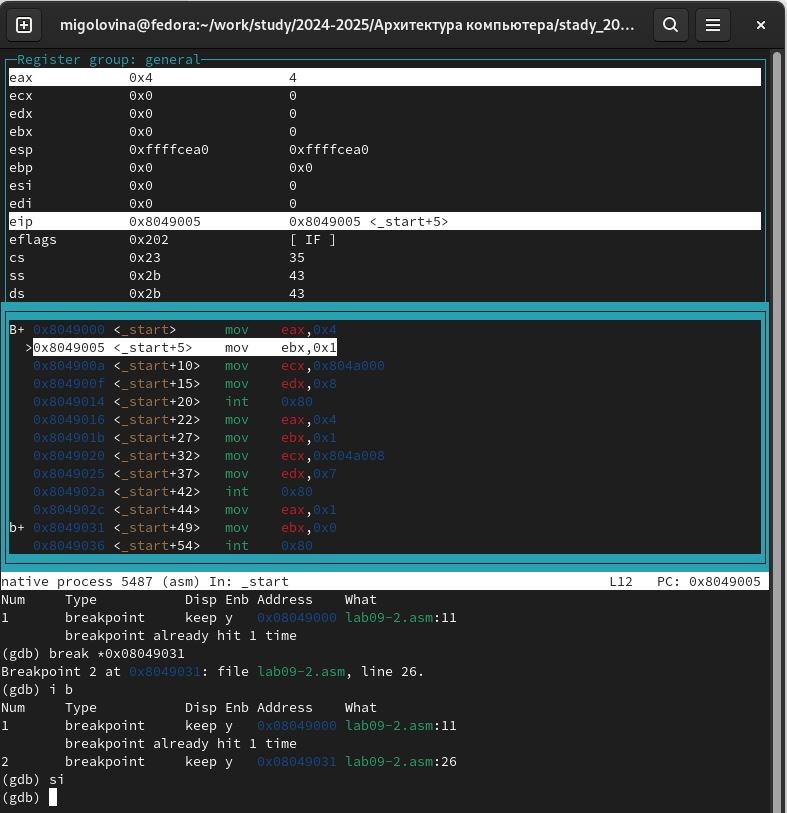
4 Установить точку останова можно командой break. Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с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

Проверили это с помощью команды info breakpoints. 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. 4.13 Установка точки останова по адресу инструкции).

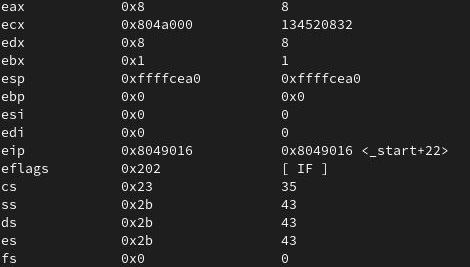


Установка точки останова по адресу инструкции

5 Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнили 5 инструкций с помощью команды si и проследили за изменением значений регистров. Изменяются регистры eax и eip. Посмотреть содержимое регистров также можно с помощью команды info registers (рис. 4.14 Просмотр регистров).



Просмотр регистров



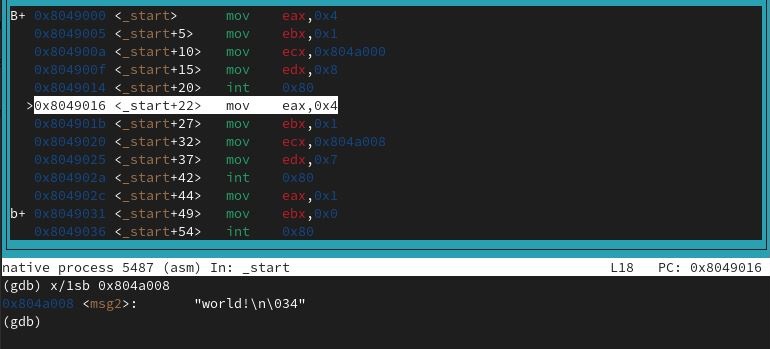
Измененные регистры

6 С помощью команды посмотрели значение переменной msg1 (рис. 4.16 Значение переменной msg1).

Значение переменной msg1

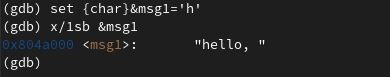
Значение переменной msg1

7 Посмотрели значение переменной msg2 по адресу. Адрес переменной можно определить по дисассемблированной инструкции. Посмотрели инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2 (рис. 4.17 Значение переменной msg2).

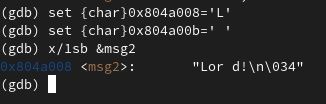


Значение переменной msg2

8 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 (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Изменили первый символ переменной msg1 (рис. 4.18 Измененное значение переменной msg1) и заменили символ во второй переменной msg2 (рис. 4.19 Измененное значение переменной msg2).

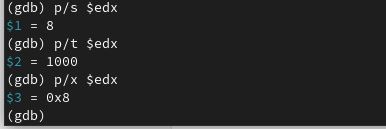


Измененное значение переменной msg1



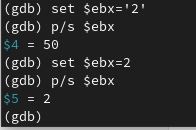
Измененное значение переменной msg2

9 Вывели в различных форматах значение регистра edx (рис. 4.20 Значение регистра edx в различных форматах).



Значение регистра edx в различных форматах

10 С помощью команды set изменили значение регистра ebx (рис. 4.21 Значение регистра ebx).



Значение регистра ebx

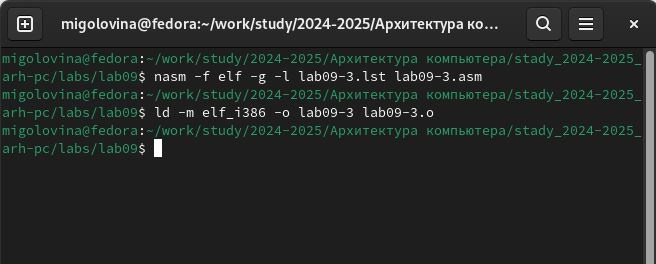
Команда выводит разные значения, так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные.

11 Завершили выполнение программы и вышли из GDB. (рис. 4.22 Завершение работы с файлов)

Завершение работы с файлов

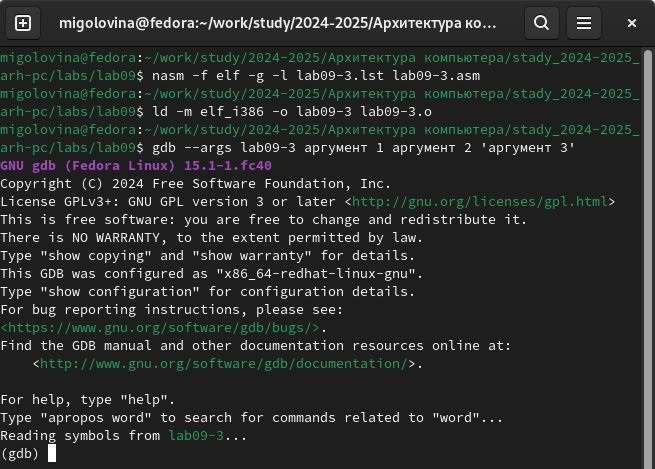
Завершение работы с файлов

12 Скопировали 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. Создали исполняемый файл (рис. 4.23 Создание исполняемого файла lab09-3.asm).



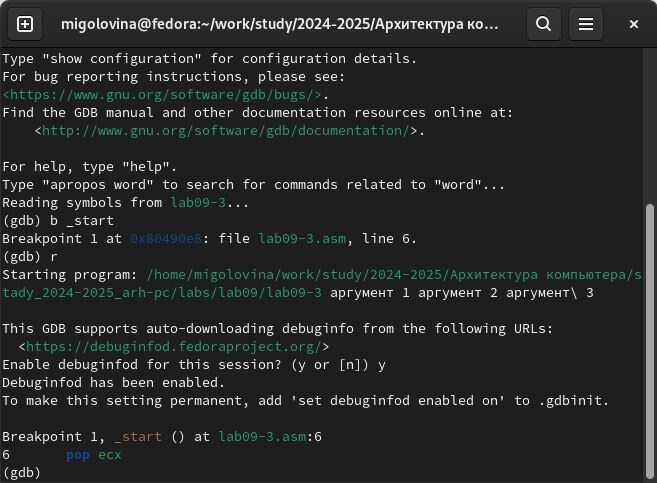
Создание исполняемого файла lab09-3.asm

13 Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузили исполняемый файл в отладчик, указав аргументы (рис. 4.24 Запуск файла в отладчике).



Запуск файла в отладчике

14 Исследовали расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установили точку останова перед первой инструкцией в программе и запустим ее (рис. 4.25 Точка останова перед первой инструкцией в программе).



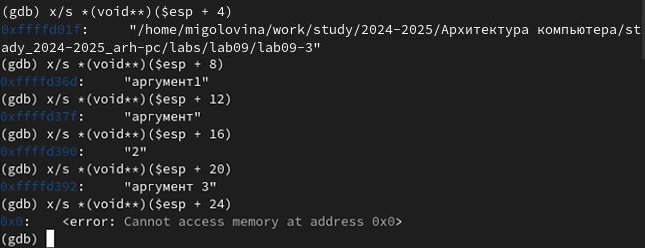
Точка останова перед первой инструкцией в программе

Проверили адрес вершины стека и убедились, что там хранится 5 элементов (рис. 4.26 Адрес вершины стека).

Адрес вершины стека

Адрес вершины стека

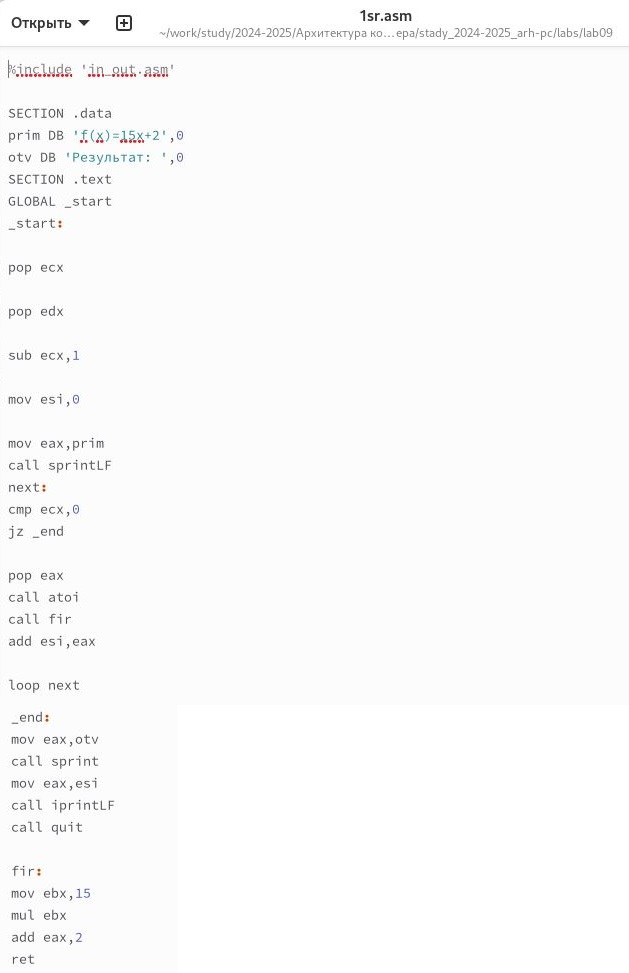
Посмотрели все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации (рис.4.27 Все позиции стека).



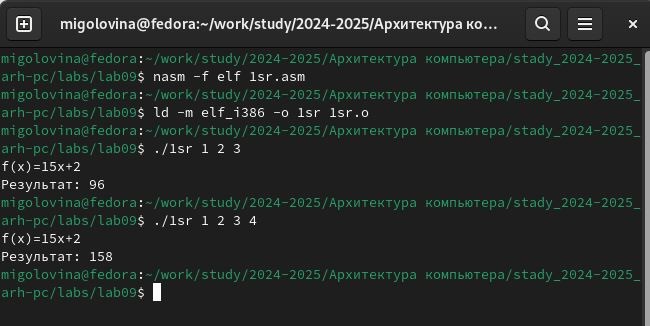
Все позиции стека

Задание для самостоятельной работы

1. Преобразовать программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.



Листинг 1 самостоятельного задания №1



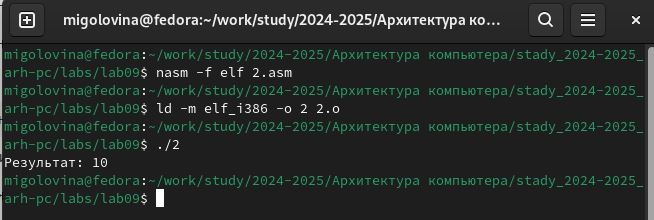
Результат работы программы

1. В листинге из самостоятельного задания приведена программа вычисления выражения (3 + 2) \* 4 + 5. При запуске данная программа дает неверный результат. Проверить это. С помощью отладчика GDB, анализируя изменения значений регистров, определить ошибку и исправить ее.

2.1 Переписали программу и попробовали запустить ее чтобы увидеть ошибку. Ошибка была арифметическая, так как вместо 25, программа выводит 10

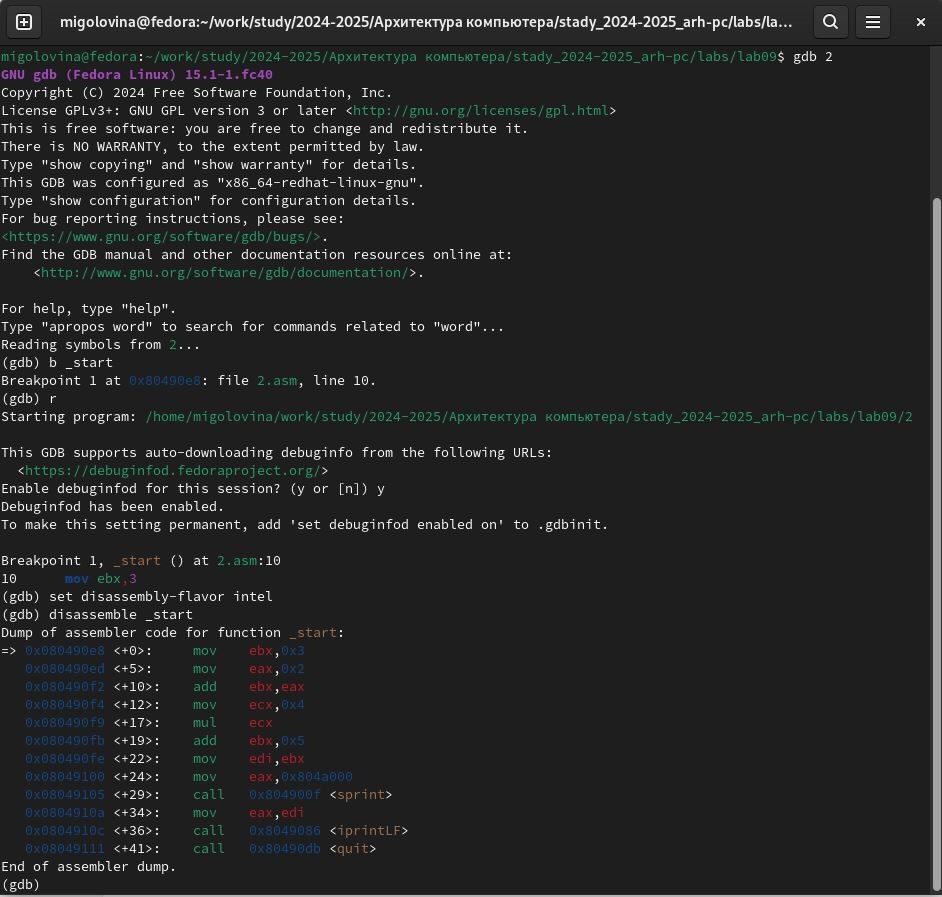


Листинг из самостоятельного задания №2



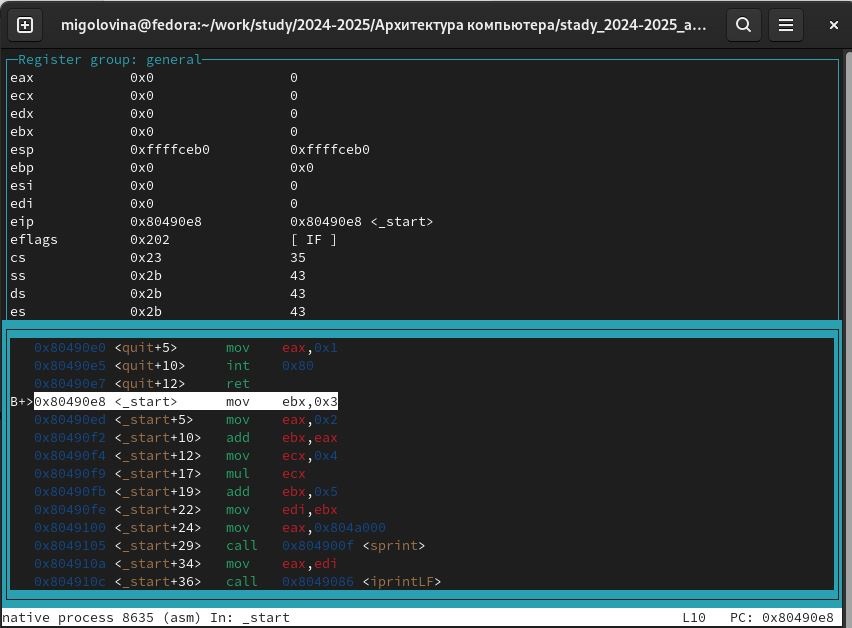
Проверка результата работы программы из Листинга самостоятельного задания №2

2.2 После обнаружения ошибки, запустили программу в отладчике (рис. 4.32 Запуск программы в отладчике).



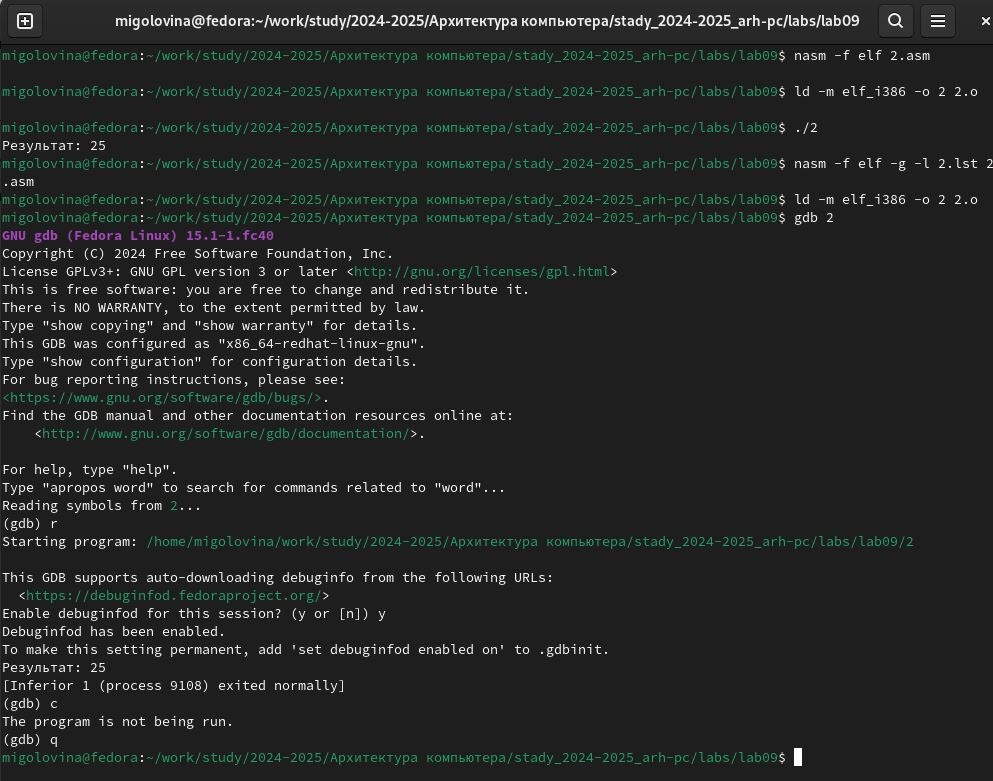
Запуск программы в отладчике

2.3 Открыли регистры и проанализировали их, поняли что некоторые регистры стоят не на своих местах, исправили это.

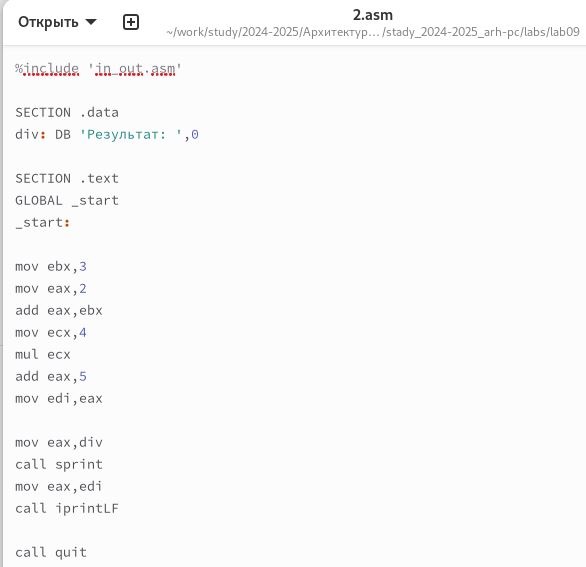


Анализ регистров

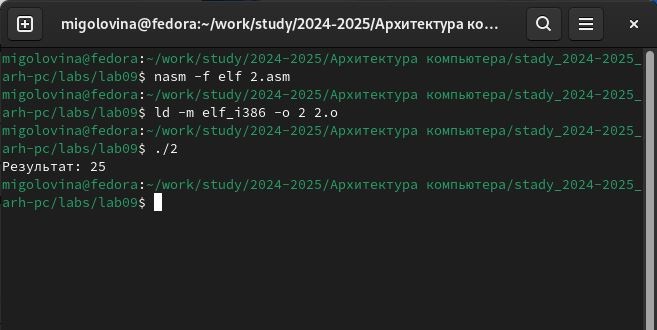
2.4 Изменили регистры и запустили программу, программа вывела ответ 25, то есть все работает правильно (рис. 4.34 Повторный запуск программы).



Повторный запуск программы



Листинг программы



Проверка результата работы программы

# 5 Выводы

Приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. GDB: The GNU Project Debugger. — URL:https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm\_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).