Лабораторная работа №9

Понятие подпрограммы.Отладчик GDB

Баранов Никита Дмитриевич

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm. В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы.Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.Измените 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. fig. 1)(рис. fig. 2)(рис. fig. 3)(рис. fig. 4)(рис. fig. 5).

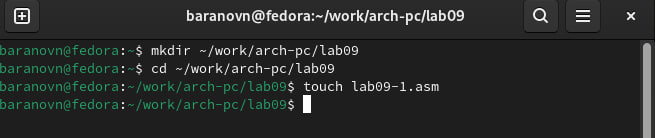


Рис. 1: Создаем директории и файл

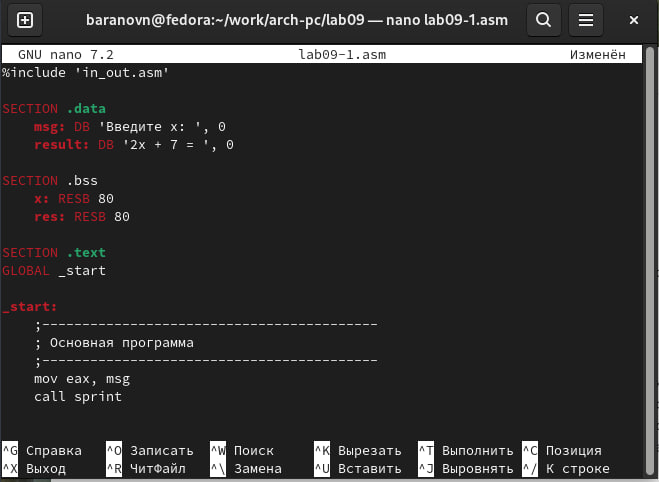


Рис. 2: Вставляем код в файл

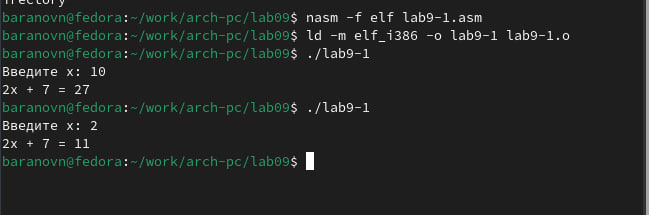


Рис. 3: Создаем исполняемый файл и проверяем его работу

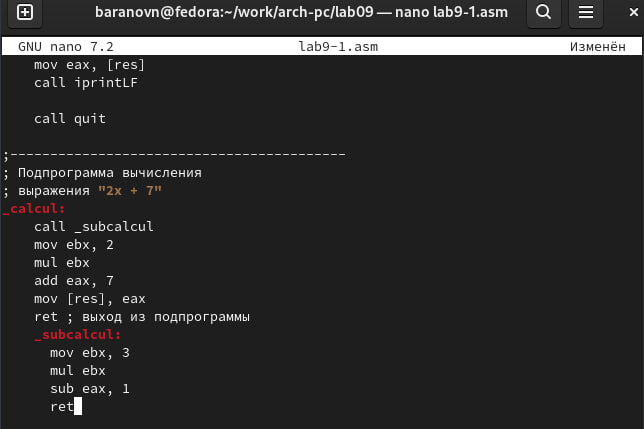


Рис. 4: Редактируем программу, добавляя subcalcul

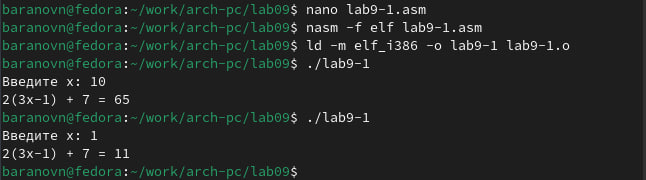


Рис. 5: Создаем объектный файл и проверяем работу программы

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!Получите исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.Загрузите исполняемый файл в отладчик gdb.Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r). Для более подробного анализа программы установите брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. Переключитесь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel.Перечислите различия отображения синтаксиса машинных команд в режимах ATT и Intel. Включите режим псевдографики для более удобного анализа программы(рис. fig. 6)(рис. fig. 7)(рис. fig. 8)(рис. fig. 9)(рис. fig. 10)(рис. fig. 11)(рис. fig. 12)

Создаем новый файл

Рис. 6: Создаем новый файл

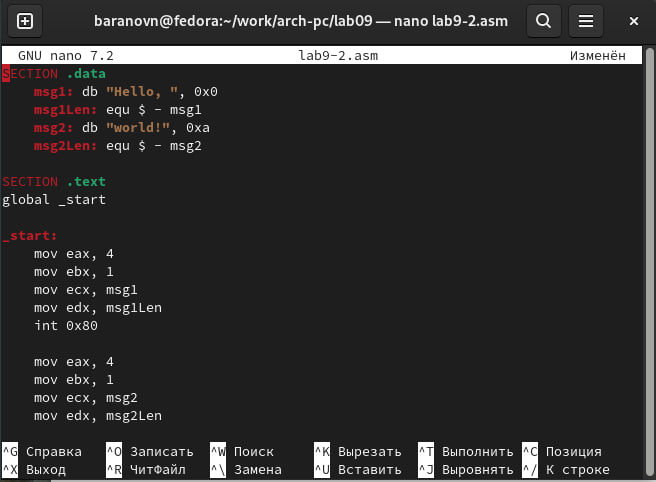


Рис. 7: Вводим программу

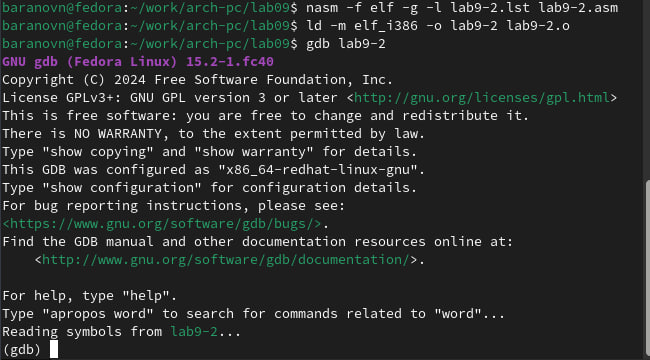


Рис. 8: Загружаем файл в отладчик

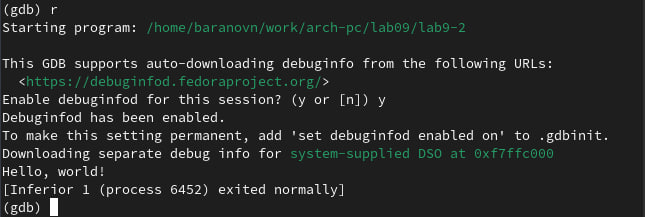


Рис. 9: Проверяем работу программы

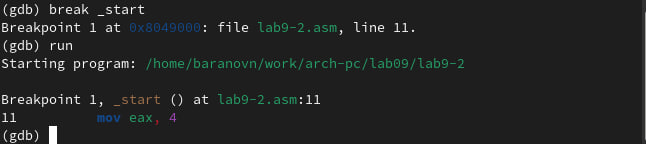


Рис. 10: Запускаем с брейкпоинтом

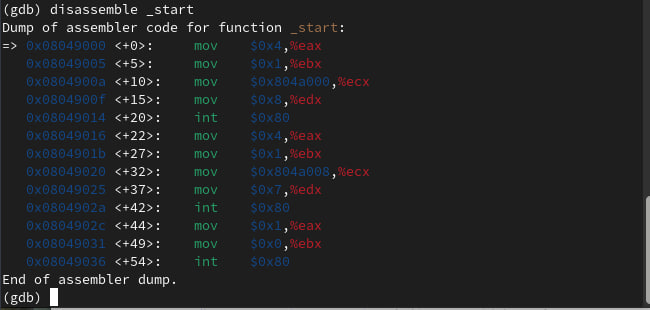


Рис. 11: Смотрим диссимилированный код программы



Рис. 12: Вводим команду set, переключаясь на синтаксис Intel

Различия в отображении синтаксиса: Порядок операндов: Intel: Операнды записываются в порядке destination, source. ATT: Операнды записываются в порядке source, destination. Суффиксы: Intel: Суффиксы для указания размера данных не используются. ATT: Используются суффиксы для указания размера данных. Регистры: Intel: Регистры указываются без каких-либо префиксов.ATT: Регистры указываются с префиксом. Константы:Intel: Константы записываются без специальных символов.ATT: Константы обозначаются с помощью $.

Включаем режим псевдографики.На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверьте это с помощью команды info breakpoints(кратко i b).Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3).Определите адрес предпоследней инструкции (mov ebx,0x0) и установите точку останова.Посмотрите информацию о всех установленных точках останова(рис. fig. 13)(рис. fig. 14)

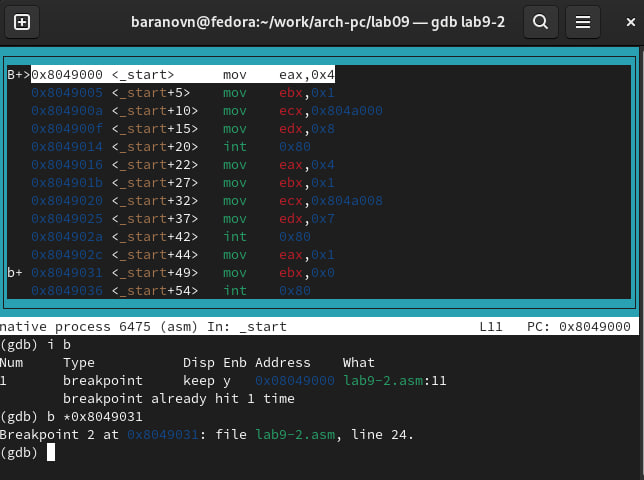


Рис. 13: Проверяем точку останова с помощью команды i b и ставим новую точку

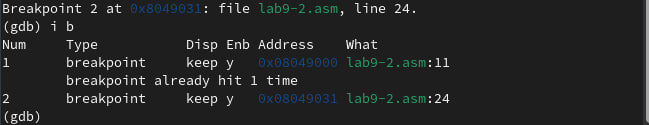


Рис. 14: Смотрим инфо о всех точках остановы

Выполните 5 инструкций с помощью команды stepi (или si) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды info registers (или i r).Посмотрите значение переменной msg1 по имени.Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2. Измените первый символ переменной msg1.Замените любой символ во второй переменной msg2.Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.С помощью команды set измените значение регистра ebx.Объясните разницу вывода команд p/s $ebx. Завершите выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q)(рис. fig. 15)(рис. fig. 16)(рис. fig. 17)(рис. fig. 18)(рис. fig. 19)(рис. fig. 20)(рис. fig. 21)(рис. fig. 22)

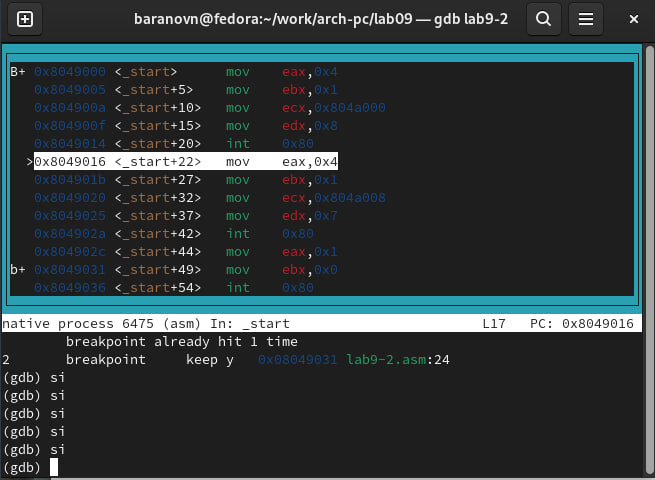


Рис. 15: Выполняем 5 команд командой si

Изменяются значения регистров - eax,ebx,ecx,edx,eip

Просматриваем содержимое переменной по имени

Рис. 16: Просматриваем содержимое переменной по имени

Просматриваем значение по адресу

Рис. 17: Просматриваем значение по адресу



Рис. 18: Изменяем первый символ переменной

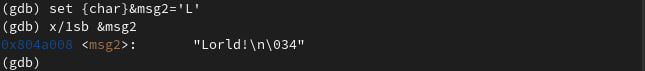


Рис. 19: Изменяем первый символ второй переменной

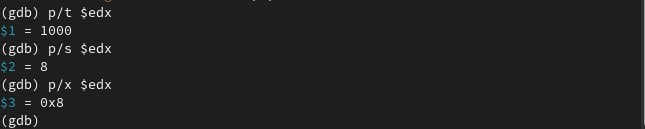


Рис. 20: Смотрим значения регистра в различных форматах

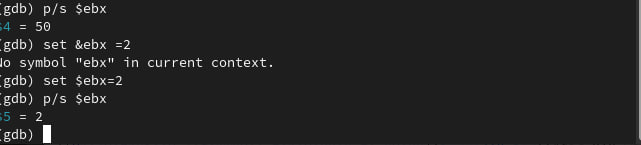


Рис. 21: Изменяем значения ebx и смотрим выводы

Выводятся разные значения т.к. команда без кавычек присваивает регистру вводимое значение.

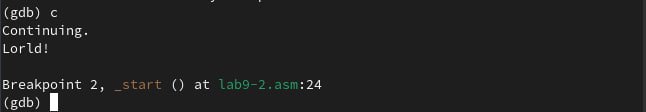


Рис. 22: Завершаем выполнение программы и выходим из gdb

Скопируйте файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.Создайте исполняемый файл.Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузите исполняемый файл в отладчик, указав аргументы.Для начала установим точку останова перед первой инструкцией в программе и запустим ее.Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы).Посмотрите остальные позиции стека – по адесу esp+4 располагается адрес в памяти где находиться имя программы, по адесу esp+8 храниться адрес первого аргумента, по аресу esp+12 – второго и т.д.Объясните, почему шаг изменения адреса равен 4 (esp+4, esp+8, esp+12 и т.д.)(рис. fig. 23)(рис. fig. 24)(рис. fig. 25)

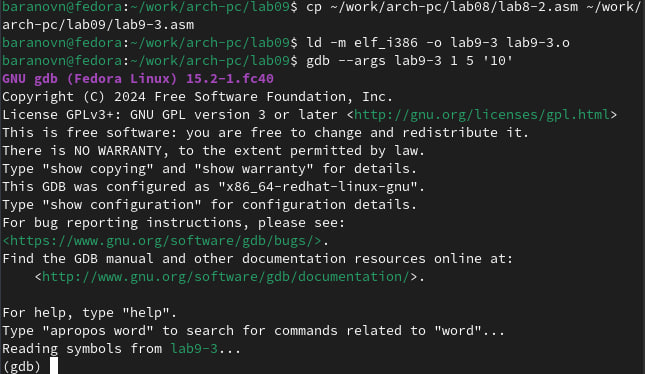


Рис. 23: Копируем программу в новый файл и создаем объектный файл и запускаем его в отладчике GDB

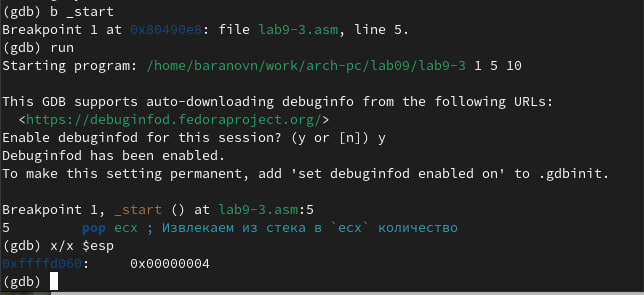


Рис. 24: Устанавливаем точку остановы и запускаем ее

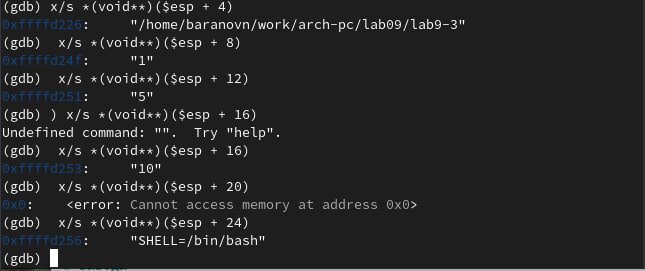


Рис. 25: Смотрим позиции стека по разным адресам

Шаг изменения равен 4 потому что регистры имеют размерность в 4 байта.

# 3 Самостоятельная работа

Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. fig. 26)(рис. fig. 27)(рис. fig. 28)(рис. fig. 29)(рис. fig. 30)(рис. fig. 31)(рис. fig. 32)(рис. fig. 33)

Копируем программу

Рис. 26: Копируем программу

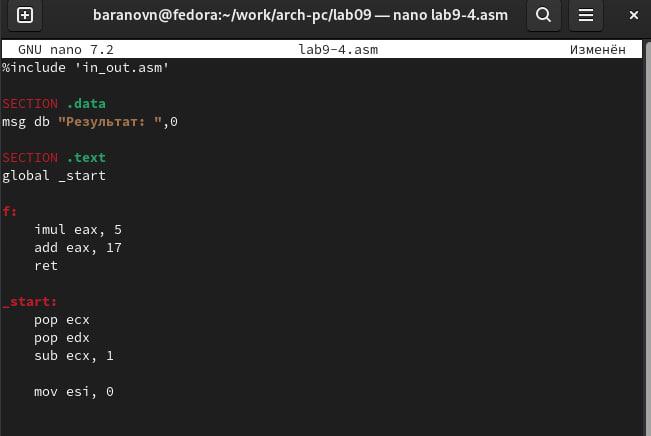


Рис. 27: Изменяем программу под условия

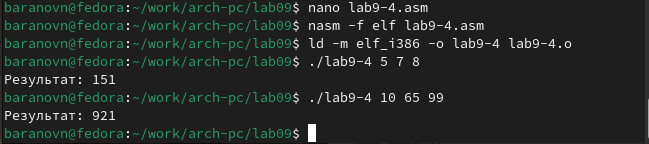


Рис. 28: Создаем объектный файл и проверяем работу программы

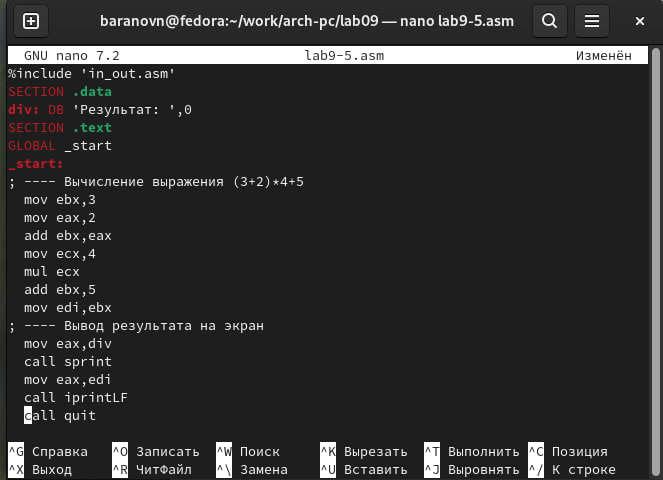


Рис. 29: Вводим программу из листинга в новый файл

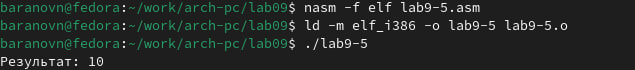


Рис. 30: Создаем объектный файл и запускаем его(есть ошибка)

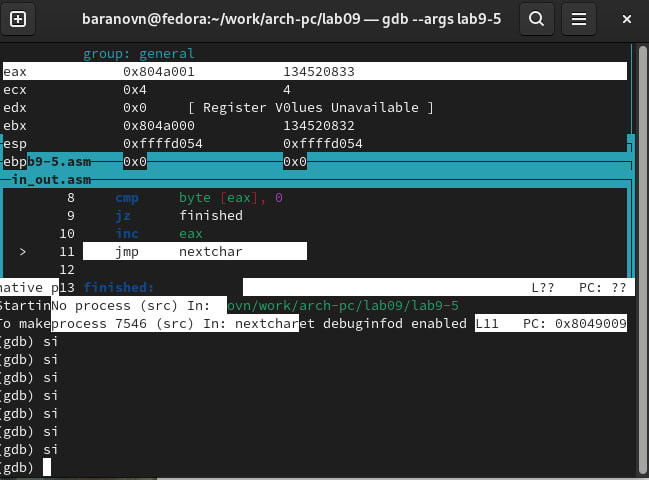


Рис. 31: Запускаем файл в отладчике и с помощью команды si находим ошибку, просматривая регистры

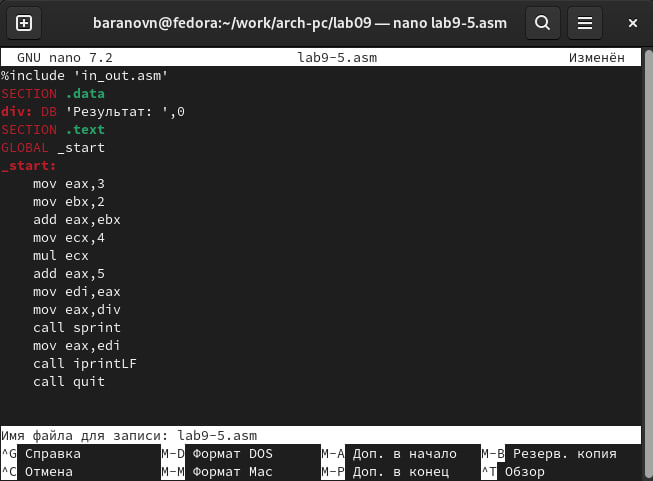


Рис. 32: Редактируем файл, исправляя ошибки

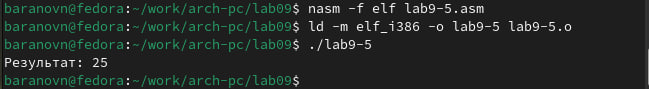


Рис. 33: Создаем объектный файл и запускаем его(нет ошибки)

# 4 Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.