Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Абрамова Ульяна Михайловна

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. Выполнение заданиий для самостоятельной работы

# 3 Теоретическое введение

## 3.1 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 пре- рывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить доволь- но 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.

## 3.2 Методы отладки

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 сообще- ния); • использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и из- менять 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 програм- мы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

## 3.3 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. От- ладчик 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.

### 3.3.1 Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид: gdb [опции] [имя\_файла | ID процесса] После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд. Далее приведён список некоторых команд GDB. Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Если точки останова не были установлены, то программа выполняется и выводятся сооб- щения: (gdb) run Starting program: test Program exited normally. (gdb) Если точки останова были заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др. Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки: Kill the program being debugged? (y or n) y 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): (gdb) q

### 3.3.2 Дизассемблирование программы

Если есть файл с исходным текстом программы, а в исполняемый файл включена информа- ция 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. Посмотреть дизассемблированный код программы можно с помощью команды disassemble : (gdb) disassemble \_start Существует два режима отображения синтаксиса машинных команд: режим Intel, ис- пользуемый 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

### 3.3.3 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»: (gdb) break \*<адрес> (gdb) b <метка> Информацию о всех установленных точках останова можно вывести командой info (крат- ко i): (gdb) info breakpoints (gdb) i b Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно вос- пользоваться командой disable: disable breakpoint <номер точки останова> Обратно точка останова активируется командой enable: enable breakpoint <номер точки останова> Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete: (gdb) delete breakpoint <номер точки останова> Ввод этой команды без аргумента удалит все точки останова. Информацию о командах этого раздела можно получить, введя help breakpoints

### 3.3.4 Пошаговая отладка

Для продолжения остановленной программы используется команда continue (c) (gdb) с [аргумент]. 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: (gdb) si [аргумент] 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: (gdb) ni [аргумент] Информацию о командах этого раздела можно получить, введя (gdb) help running

### 3.3.5 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): (gdb) info registers Для отображения содержимого памяти можно использовать команду x/NFU , выда- ёт содержимое ячейки памяти по указанному адресу. NFU задает формат, в котором выводятся данные. Например, x/4uh 0x63450 — это запрос на вывод четырёх полуслов (h) из памяти в формате беззнаковых десятичных целых (u), начиная с адреса 0x63450. Чтобы посмотреть значения регистров используется команда print /F (сокращен- но p). Перед именем регистра обязательно ставится префикс $. Например, команда p/x $ecx выводит значение регистра в шестнадцатеричном формате. 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). Справку о любой команде gdb можно получить, введя (gdb) help [имя\_команды]

### 3.3.6 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 бу- дет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

#### 3.3.6.1 Инструкция call и инструкция ret

Для вызова подпрограммы из основной программы используется инструкция call, кото- рая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответству- ющей 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, занесён- ный 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. Важно помнить, что если в подпрограмме занести что-то в стек и не извлечь, то на вершине стека окажется не адрес возврата и это приведёт к ошибке выхода из подпрограммы. Кроме того, надо помнить, что подпрограмма без команды возврата не вернётся в точку вызова, а будет выполнять следующий за подпрограммой код, как будто он является её продолжением.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9, перейдя в него, создаю файл lab9-1.asm (рис. 1).

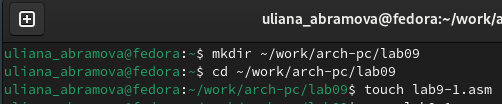


Рис. 1: Создание каталога и файла

В созданный файл ввожу программу с вызовом подпрограммы (рис. 2).

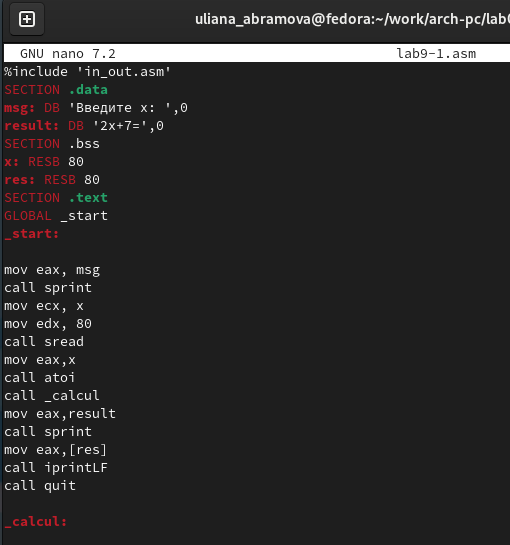


Рис. 2: Написание программы

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

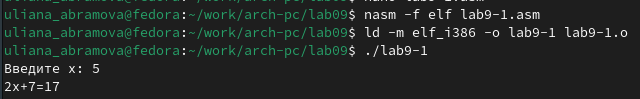


Рис. 3: Запуск программы

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul для вычисления выражения f(g(x)) (рис. 4).

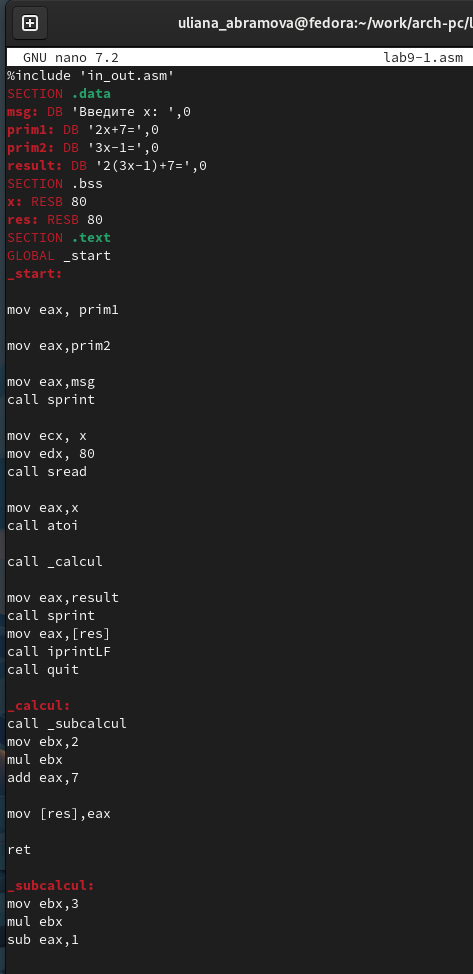


Рис. 4: Изменения программы

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

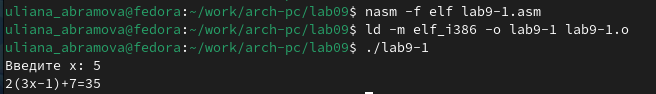


Рис. 5: Запуск программы

## 4.2 Отладка программам с помощью GDB

Создаю файл lab9-2.asm с текстом программы вывода сообщения Hello world! (рис. 6)

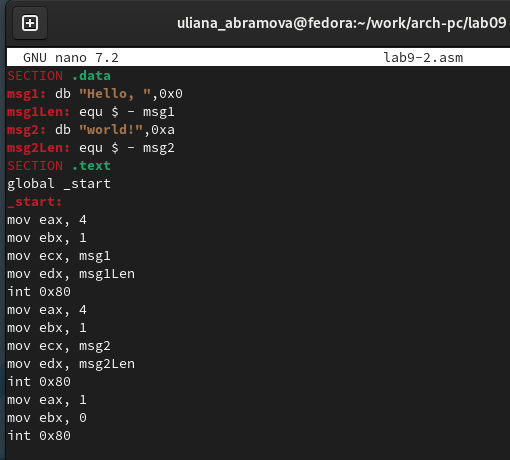


Рис. 6: Написание программы

Получаю исполняемый файл и загружаю его в отладчик gdb, а также проверяю работу программы, запустив ее в оболочке gdb с помощью команды run (сокращенно r) (рис. 7)

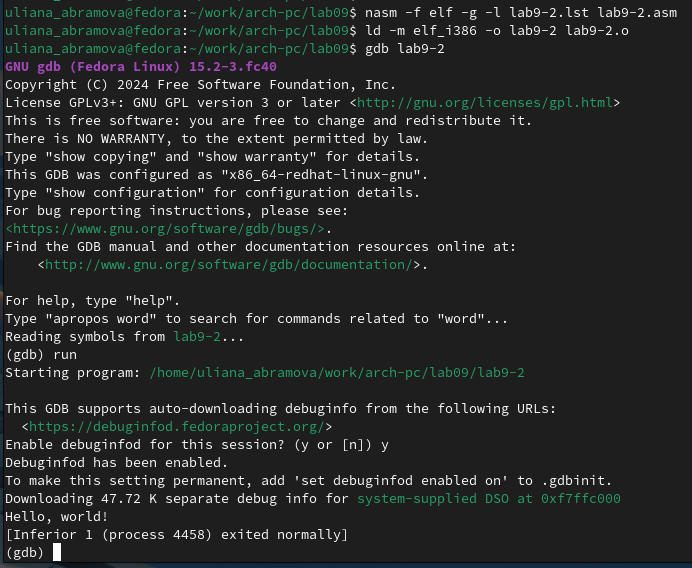


Рис. 7: Запуск программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 8)

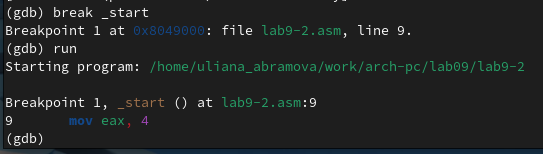


Рис. 8: Установка брейкпоинта

C помощью команды disassemble, начиная с метки \_start, смотрю дисассимилированный код программы (рис. 9)

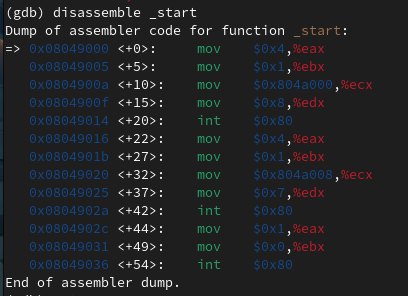


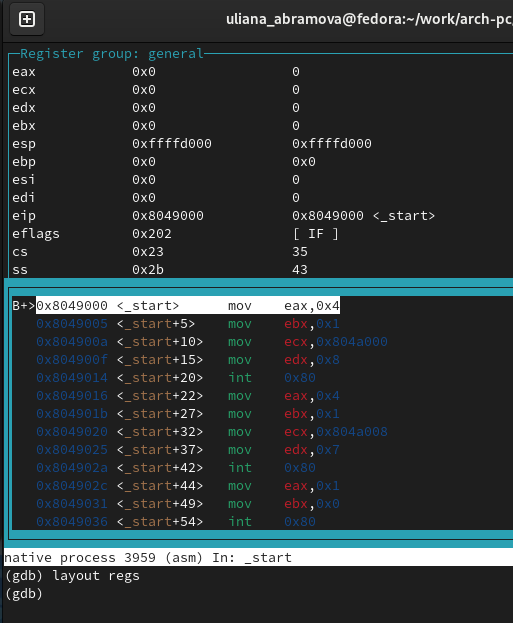
Рис. 9: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 10)



Рис. 10: Отображение команд с Intel’овским синтаксисом

Отличие заключается в командах: в диссамилированном отображении в командах используют % и $, а в Intel отображение эти символы не используются.

Включаю режим псевдографики для более удобного анализа программы (рис. **¿fig:fig11?**) 

### 4.2.1 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (кратко i b) (рис. 11)

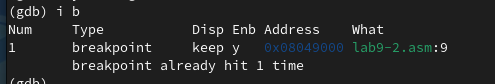


Рис. 11: Проверка точки останова

Устанавливаю еще одну точку останова по адресу инструкции и смотрю информацию о всех установленных точек останова (рис. 12, 13)

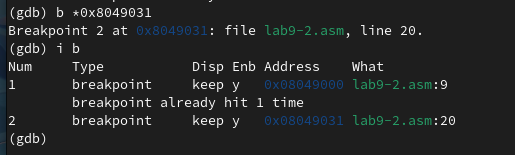


Рис. 12: Установка точки останова

|  |
| --- |
| Рис. 13: Информация о всех точках останова |

Рис. 13: Информация о всех точках останова

### 4.2.2 Работа с данными программы в GDB

С помощью команды info registers (i r) смотрю содрежимое регистров (рис. 14)

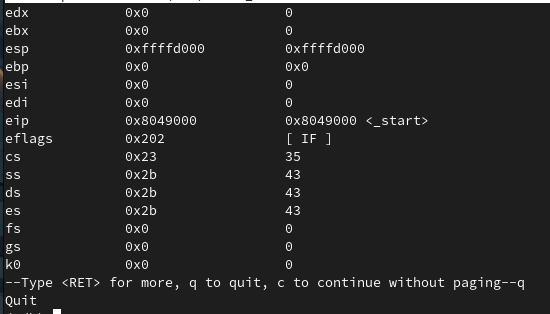


Рис. 14: Содержание регистров

С помощью команды x/NFU смотрю содержимое переменной msg1 (рис. 15)

Рис. 15: Содержимое перемнной msg1

Рис. 15: Содержимое перемнной msg1

Также смотрю значение переменной msg2 по адресу (рис. 16)

Рис. 16: Содержимое перемноой msg2

Рис. 16: Содержимое перемноой msg2

Использовав комнаду set, изменяю первый символ перемноой msg1 (рис. 17)

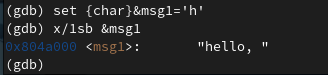


Рис. 17: Изменение первого символа

Теперь изменяю символ переменной msg2 (рис. 18)

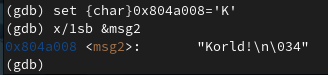


Рис. 18: Изменение символа

С помощью той же команды изменяю значение регистра ebx (рис. 19)

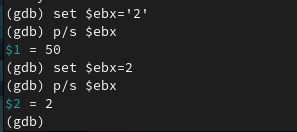


Рис. 19: Изменение регистра

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

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из gdb с помощью команды quit (сокращенно q) (рис. 20)



Рис. 20: Завершение и выход

### 4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm в файл с именем lab9-3.asm. Создаю исполняемый файл и для загрузки в gdb использую ключ –args, так как программа содержит аргументы (рис. 21)

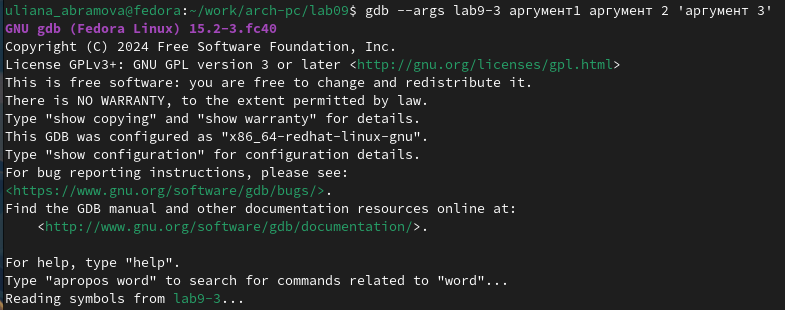


Рис. 21: Создание исполняемого файла

Для начала устанавливаю точку останова и запускаю программу (рис. 22)

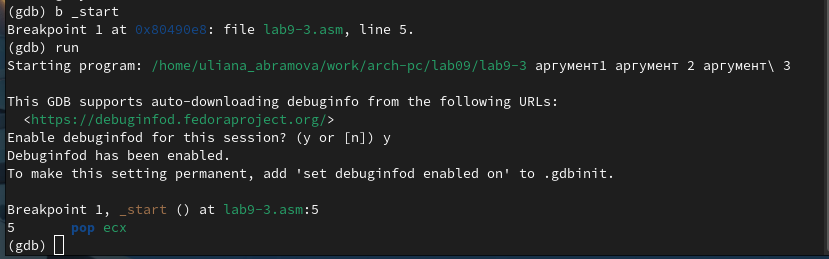


Рис. 22: Установка точки останова и запуск

Далее я проверила адрес вершины стека и убедилась, что там хранится 5 элементов. (рис. 23)

Рис. 23: Проверка адреса вершины стека

Рис. 23: Проверка адреса вершины стека

Затем просмотрела остальные позиции стека (рис. 24)

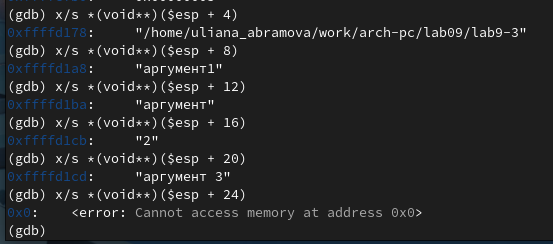


Рис. 24: Проверка остальных позиций стека

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.

## 4.3 Выполнение заданий для самостоятельной работы

### 4.3.1 1

Я преобразовала программу из лабораторной работы №8 и реализовала вычисления как подпрограмму (рис. 25)

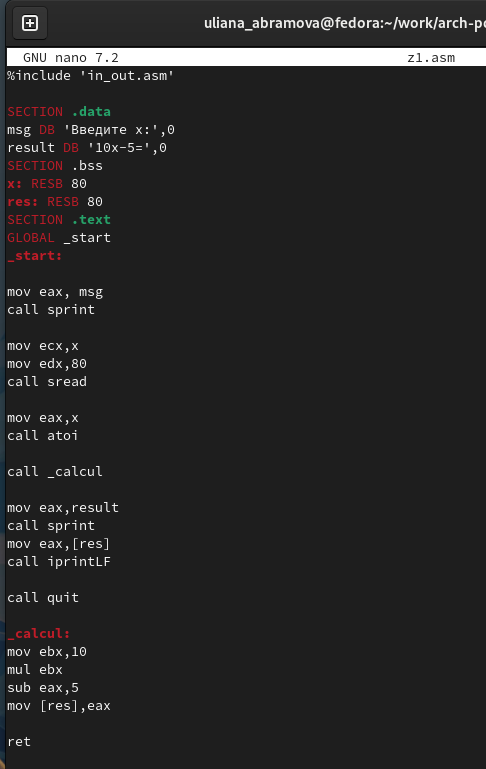


Рис. 25: Преобразование программы

Создаю исполняемый файл и проверяю корректность работы программы (рис. 26)

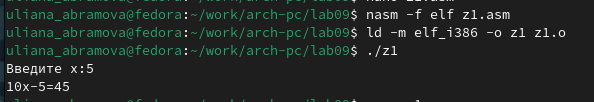


Рис. 26: Запуск программы

### 4.3.2 2

Переписав программу из Листинга 9-3, я запустила ее, чтобы увидеть арифметическую ошибку: вместо 25 программа выводила 10 (рис. 27)

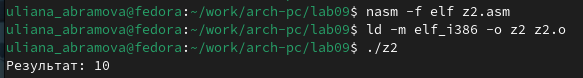


Рис. 27: Запуск программы

Я открыла регистры и проанализировала их: некоторые регистры стоят не на своих местах (рис. 28)

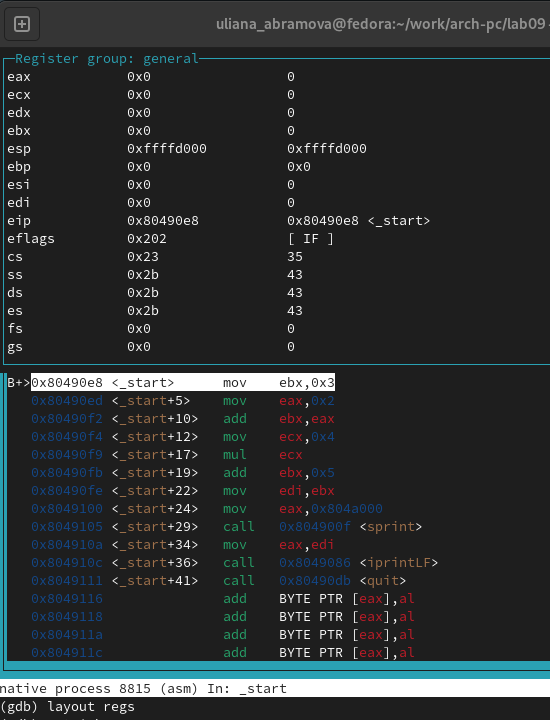


Рис. 28: Анализ изменеия значений регистров

Определив ошибку, я исправила, и теперь команда отработана верно (рис. 29)

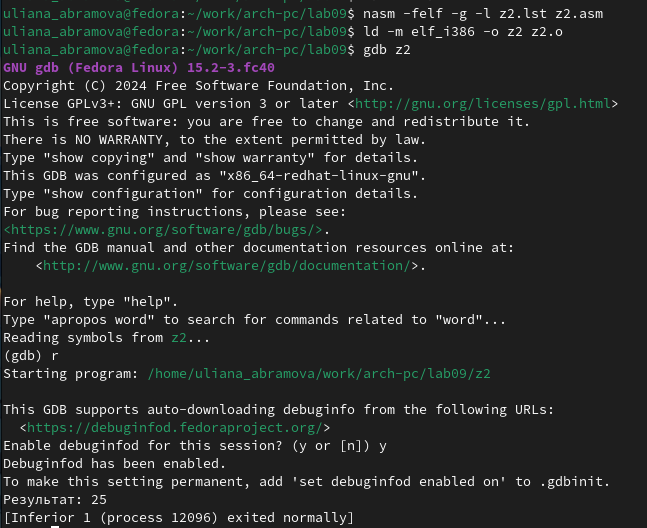


Рис. 29: Запуск исправленной команды

# 5 Вывод

Я приобрела навыки написания программ использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.

# 6 Список литературы

1. [Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089551/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)