РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

дисциплина: Архитектура компьютера

Студент: Юсупова Амина Руслановна

Группа: НКАбд-06-25

МОСКВА

2025 г.

Содержание

1 Цель работы 2
2 Задание 2
3 Теоретическое введение 2
4 Выполнение лабораторной работы 3
4.1 Реализация подпрограмм в NASM 3
4.1.1 Отладка программ с помощью GDB 7
4.1.2 Добавление точек останова 11
4.1.3 Работа с данными программы в GDB 12
4.1.4 Обработка аргументов командной строки в GDB 16
4.2 Задание для самостоятельной работы 17
5 Выводы 26
6 Список литературы 26

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 программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

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 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.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю каталог lab09 для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm: (рис. 1).

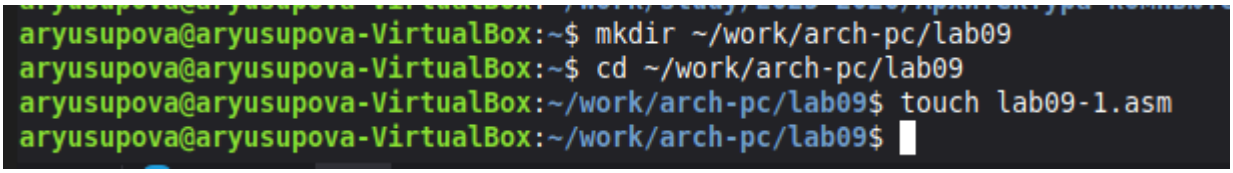

Рисунок 1: Создание рабочего каталога и переход в него

Копирую в файл код из листинга 9.1 (рис. 2) , компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 3).

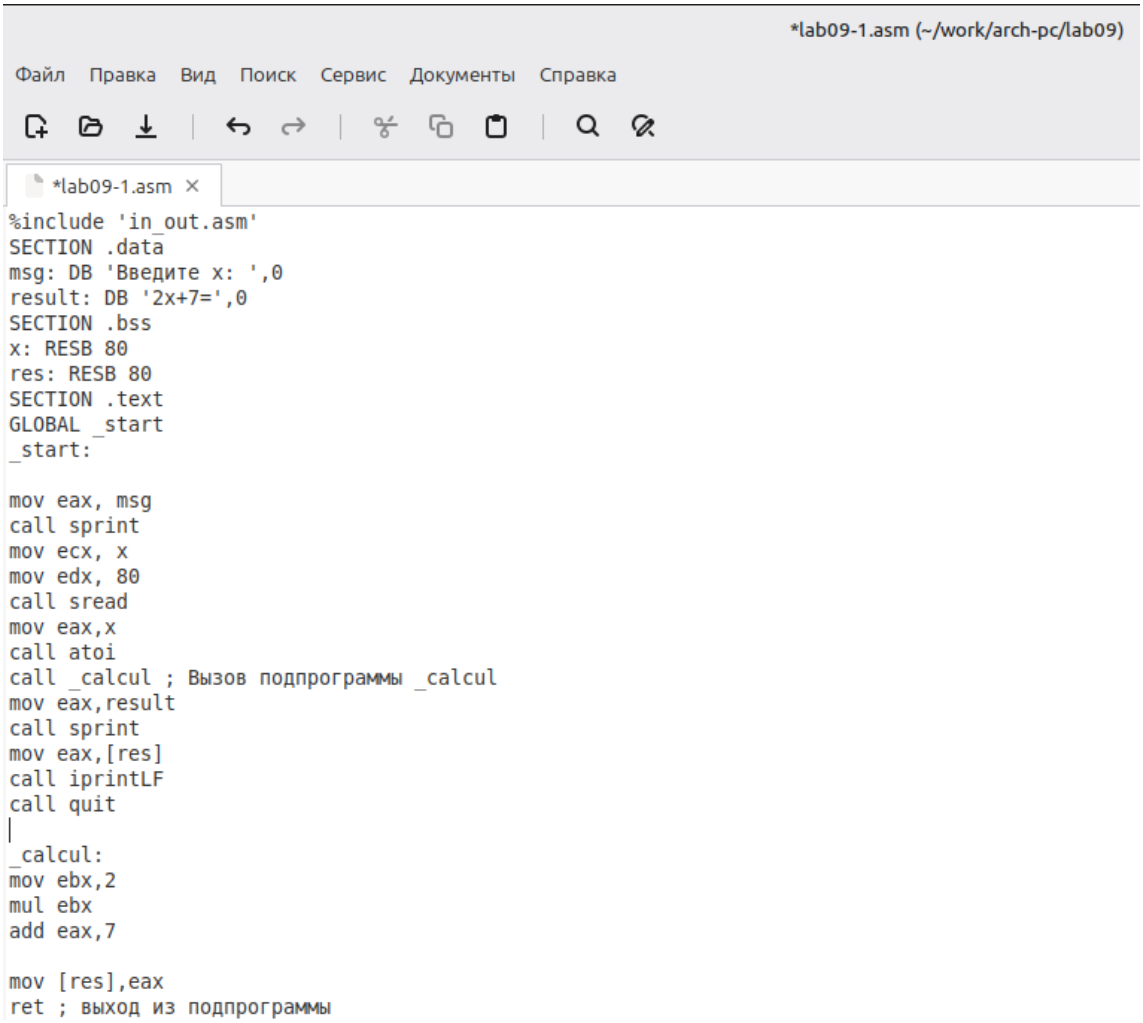

Рисунок 2: Копирование кода из листинга 9.1

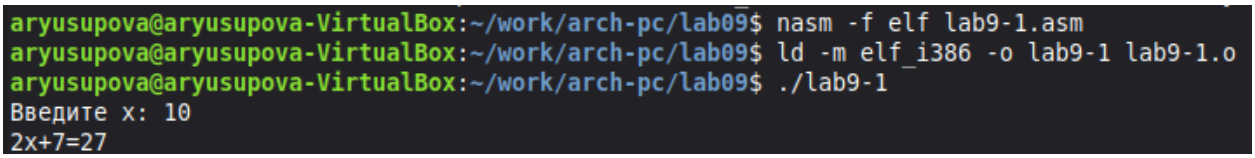

Рисунок 3: Запуск программы lab9-1.asm

Изменяю текст программы, добавив в нее подпрограмму (рис. 4). Компилирую и запускаю программу lab9-1.asm, теперь она вычисляет значение функции для выражения f(g(x)) (рис. 5).

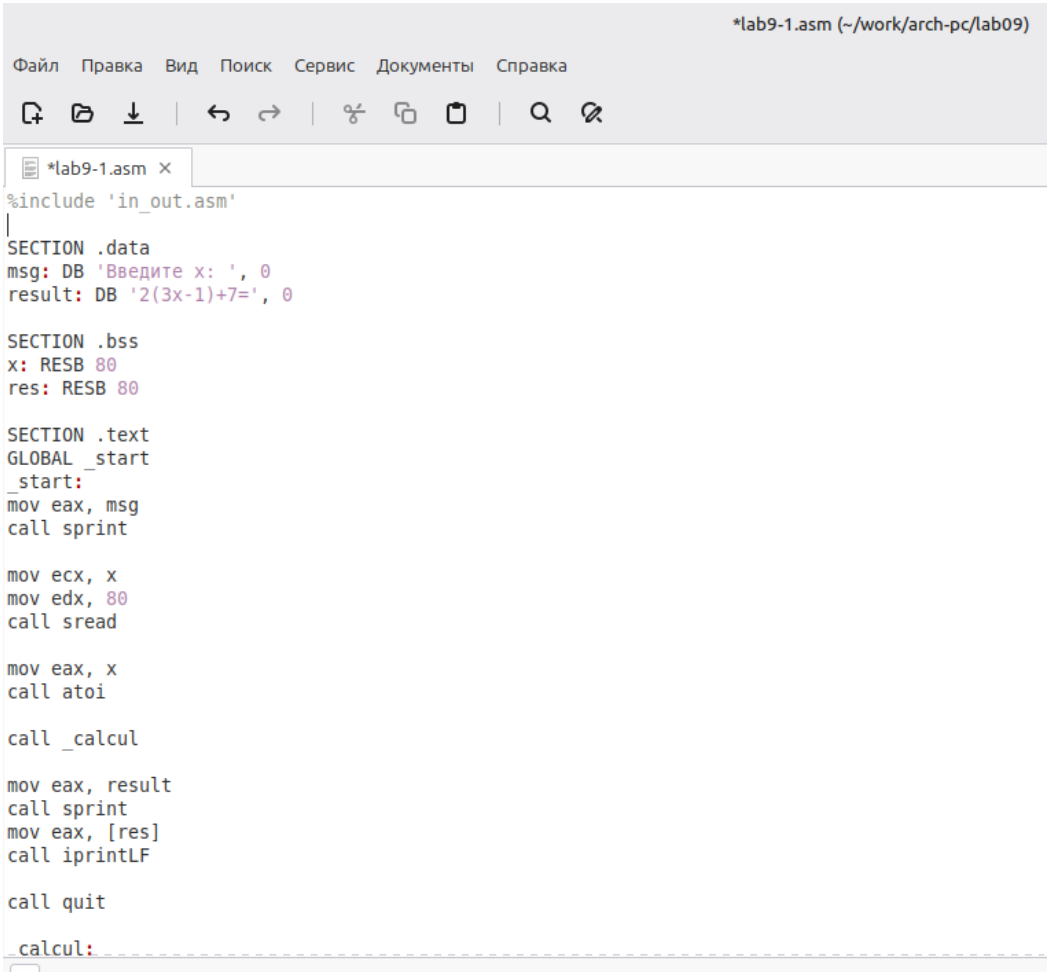

Рисунок 4: Изменение кода программы lab9-1.asm

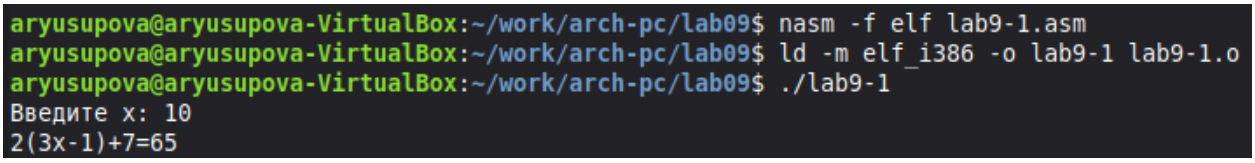

Рисунок 5: Запуск изменённой программы lab9-1.asm

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ', 0
result: DB '2(3x-1)+7=', 0

SECTION .bss
x: RESB 80
res: RESB 80
```

```
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
push eax
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7

mov [res], eax
```

```
pop eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

4.1.1 Отладка программ с помощью GDB

Создаю файл lab9-2.asm (рис.6), копирую программу листинга 9.2 (рис. 7), транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. 8).

```
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ touch lab9-2.asm
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$
```

Рисунок 6: Создание файла lab9-2.asm

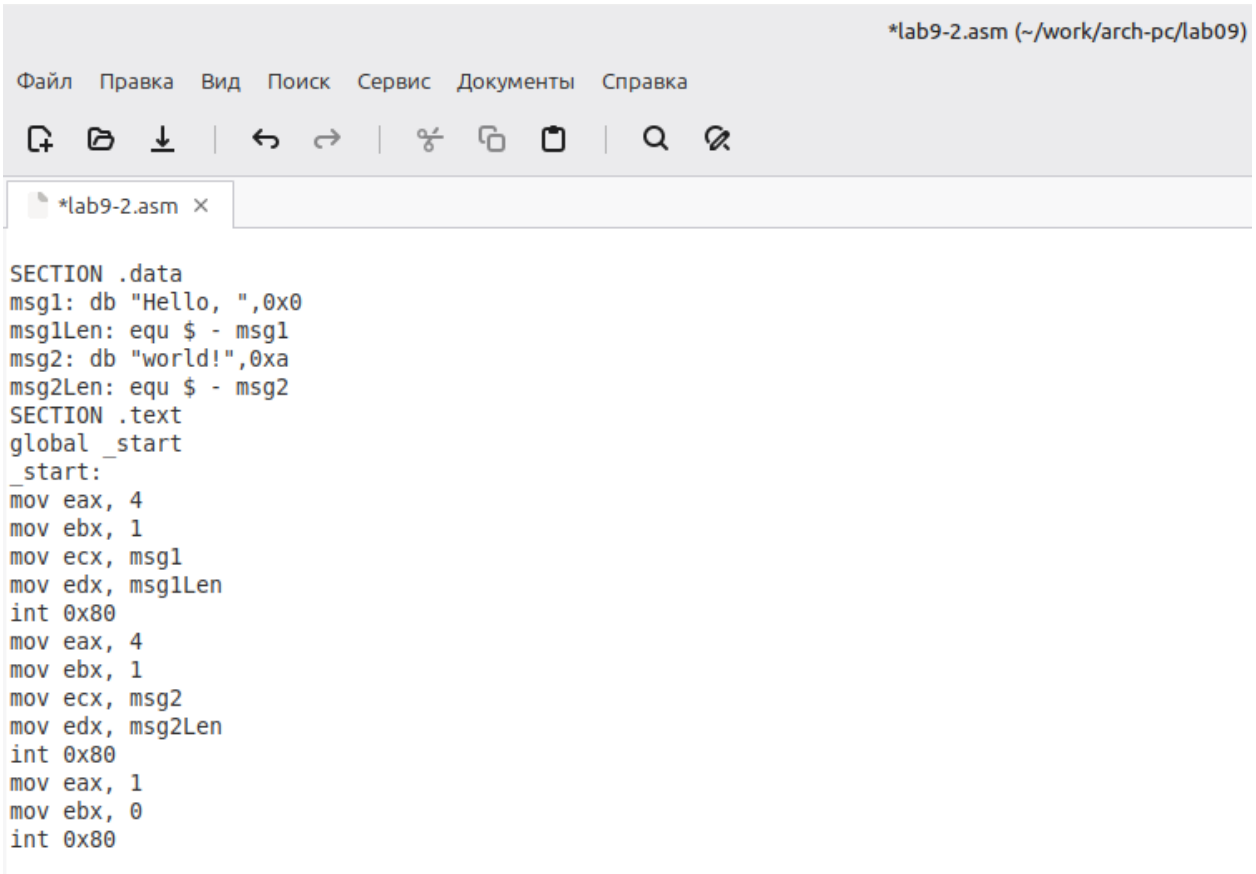

Рисунок 7: Копирование кода из листинга 9.2 в файл lab9-2.asm

```
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb)
```

Рисунок 8: Запуск программы lab9-2.asm

Запустив программу командой run, я убедилась в том, что она работает исправно (рис. 9).

```
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
aryusupova@aryusupova-VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/aryusupova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 8997) exited normally]
(gdb)
```

Рисунок 9: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку _start и снова запускаю отладку (рис. 10).

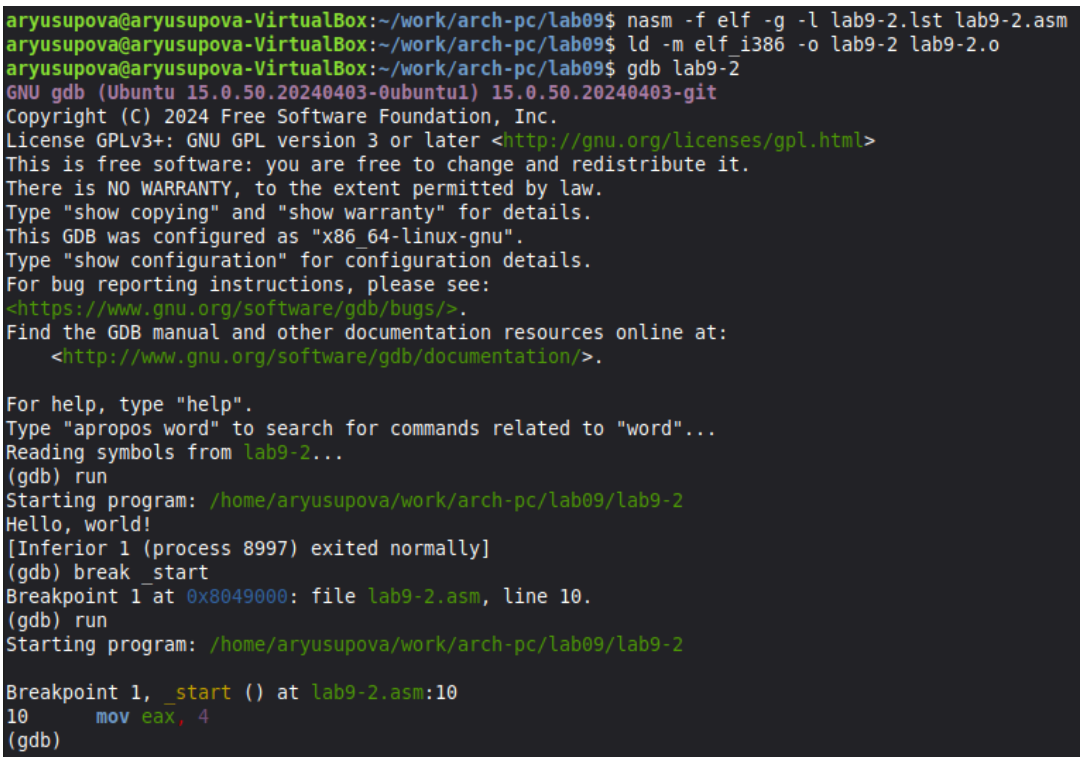

Рисунок 10: Запуск отладчика с брйкпоинт

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *амд топчик* (рис. 11).

Различия между синтаксисом ATT и Intel заключаются в порядке операндов (ATT - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (ATT - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом $; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров(ATT - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

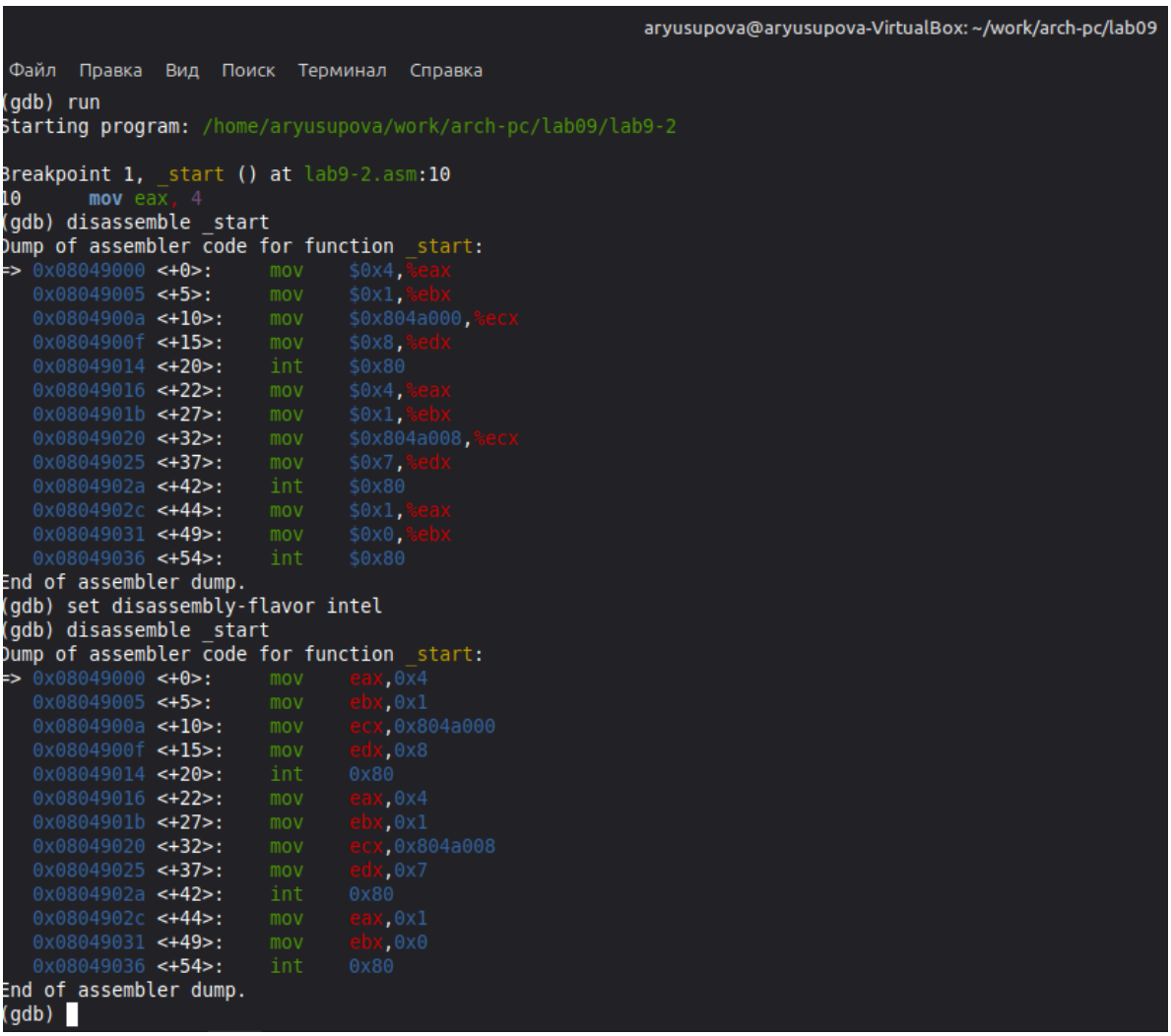

Рисунок 11: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. 12, рис. 13).

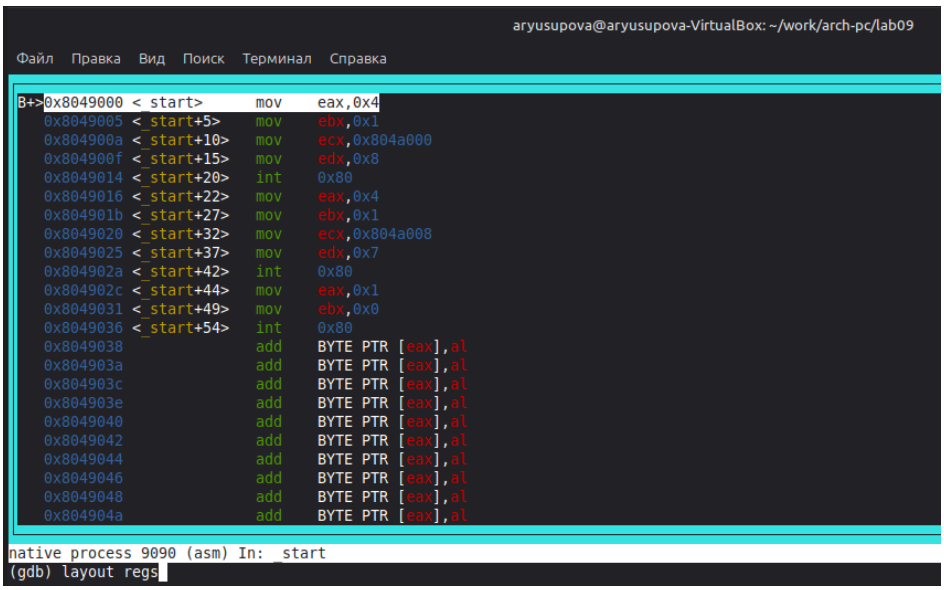

Рисунок 12: Включение режима псевдографики

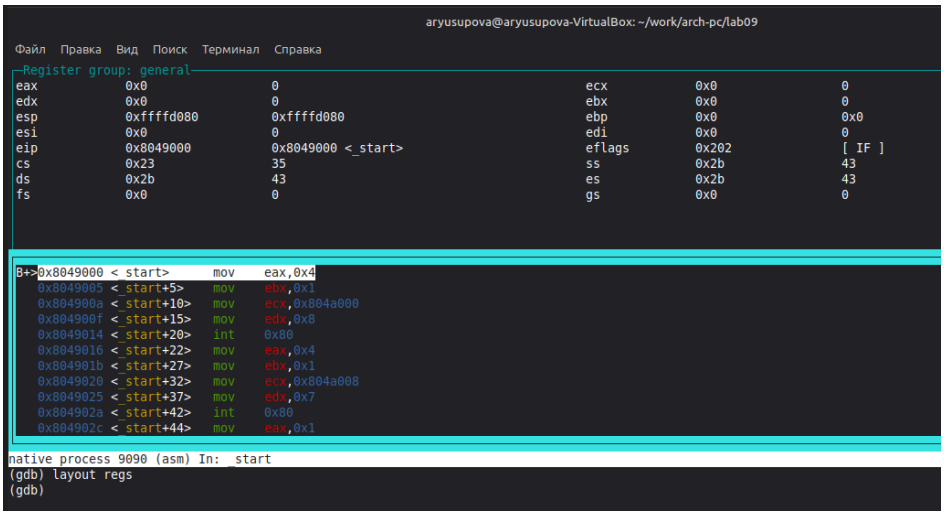

Рисунок 13: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. 14).

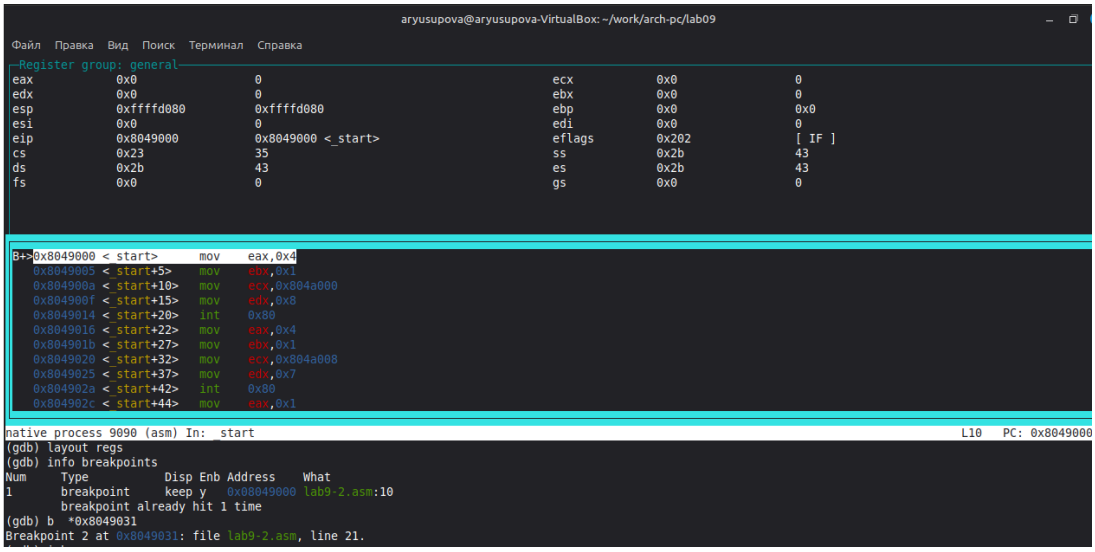

Рисунок 14: Список брейкпоинтов

Установим еще одну точку останова по адресу инструкции (с помощью команды (gdb) break *) и посмотрим информацию о всех установленных точках останова (с помощью команды i b) (рис. 15).

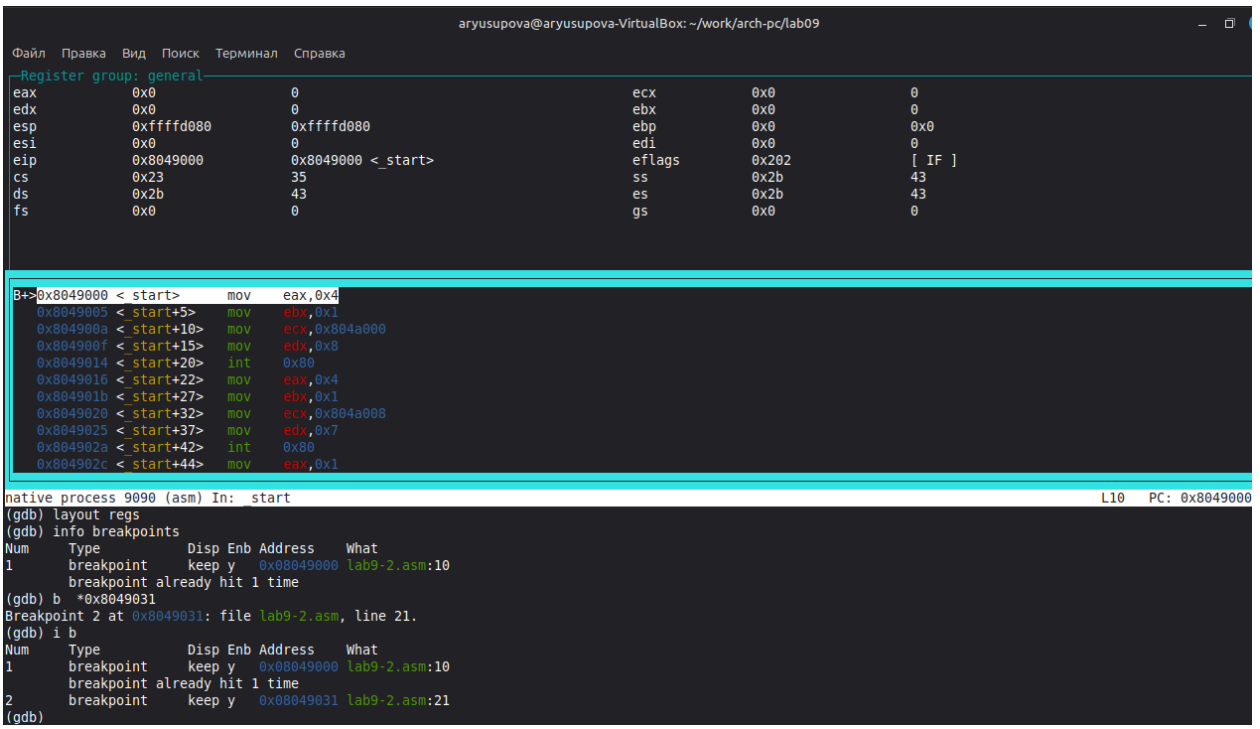

Рисунок 15: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. 16).

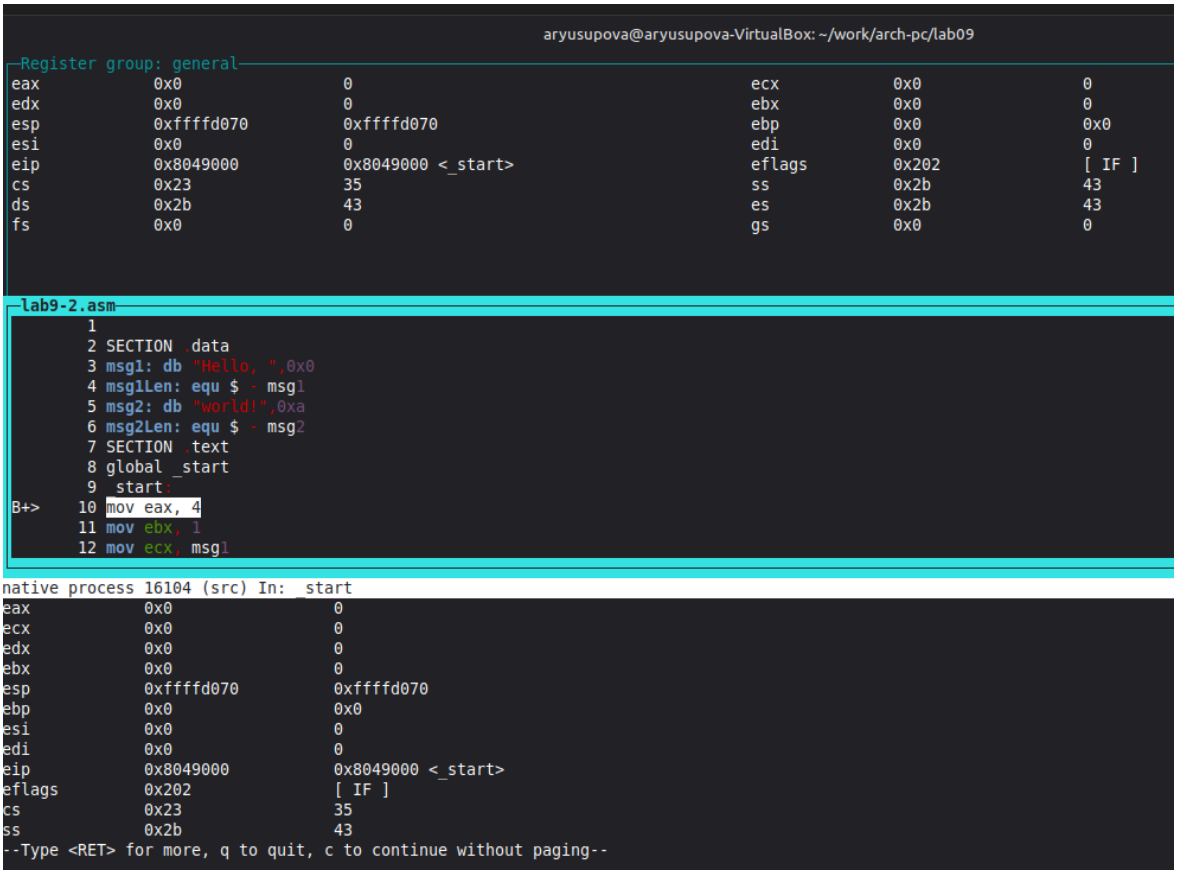

Рисунок 16: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 17).

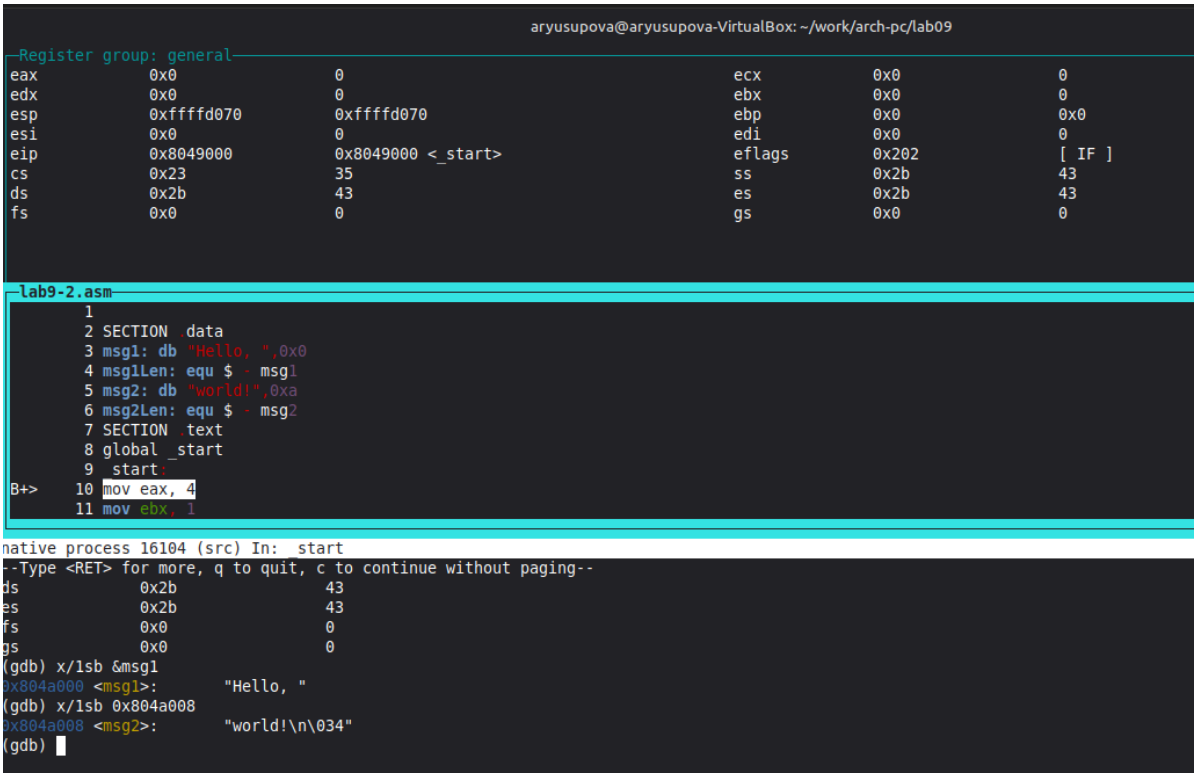

Рисунок 17: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. 18).

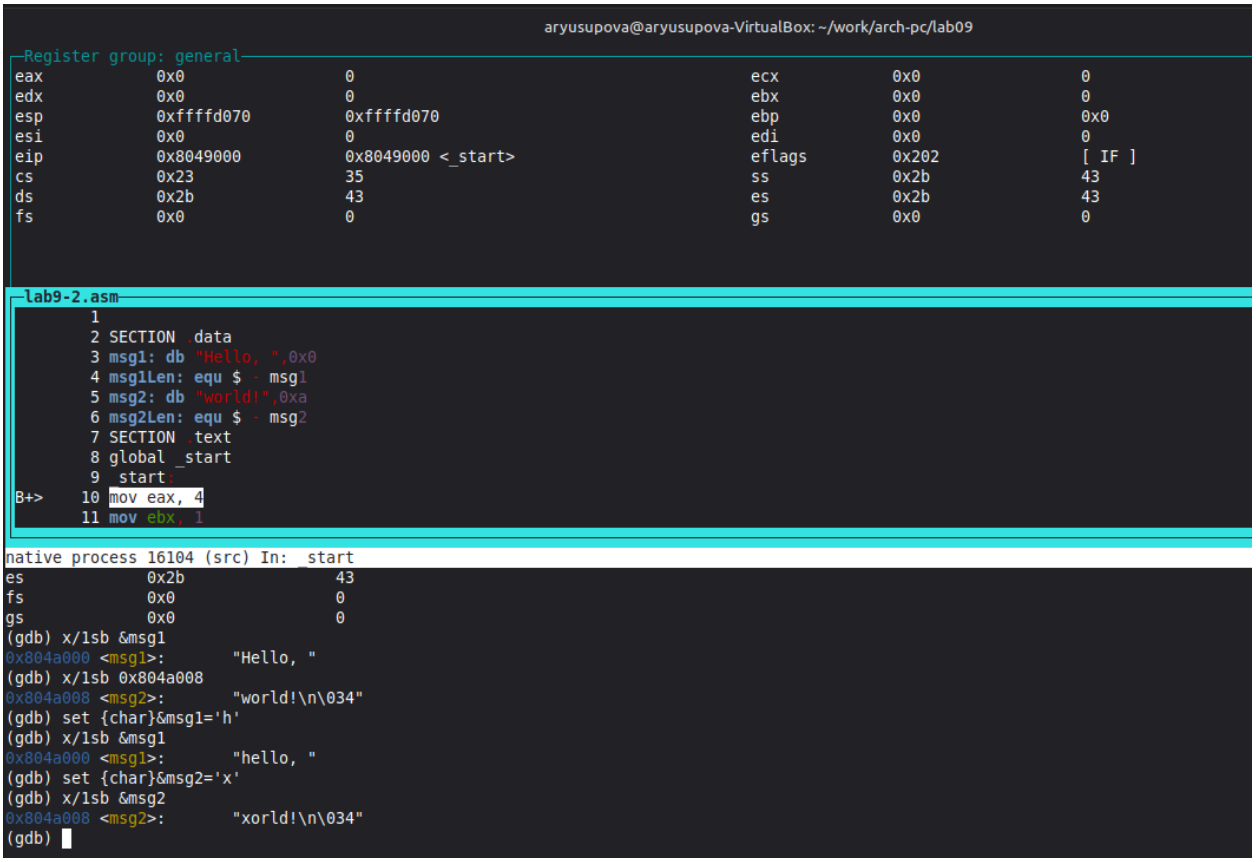

Рисунок 18: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. 19). Выводит нули, не совсем понимаю по какой причине.

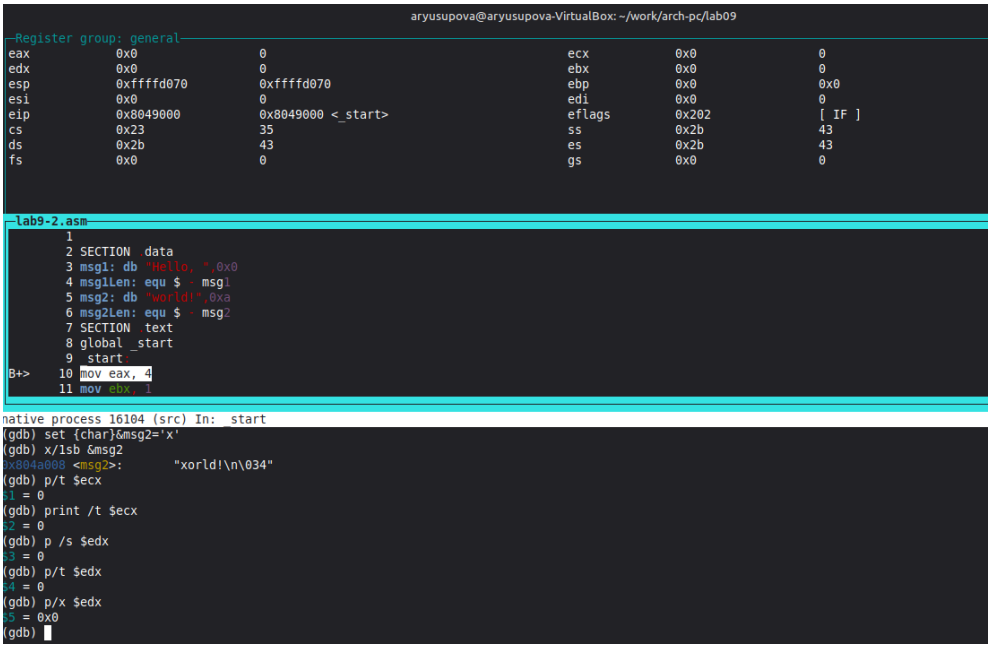

Рисунок 19: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. 20).

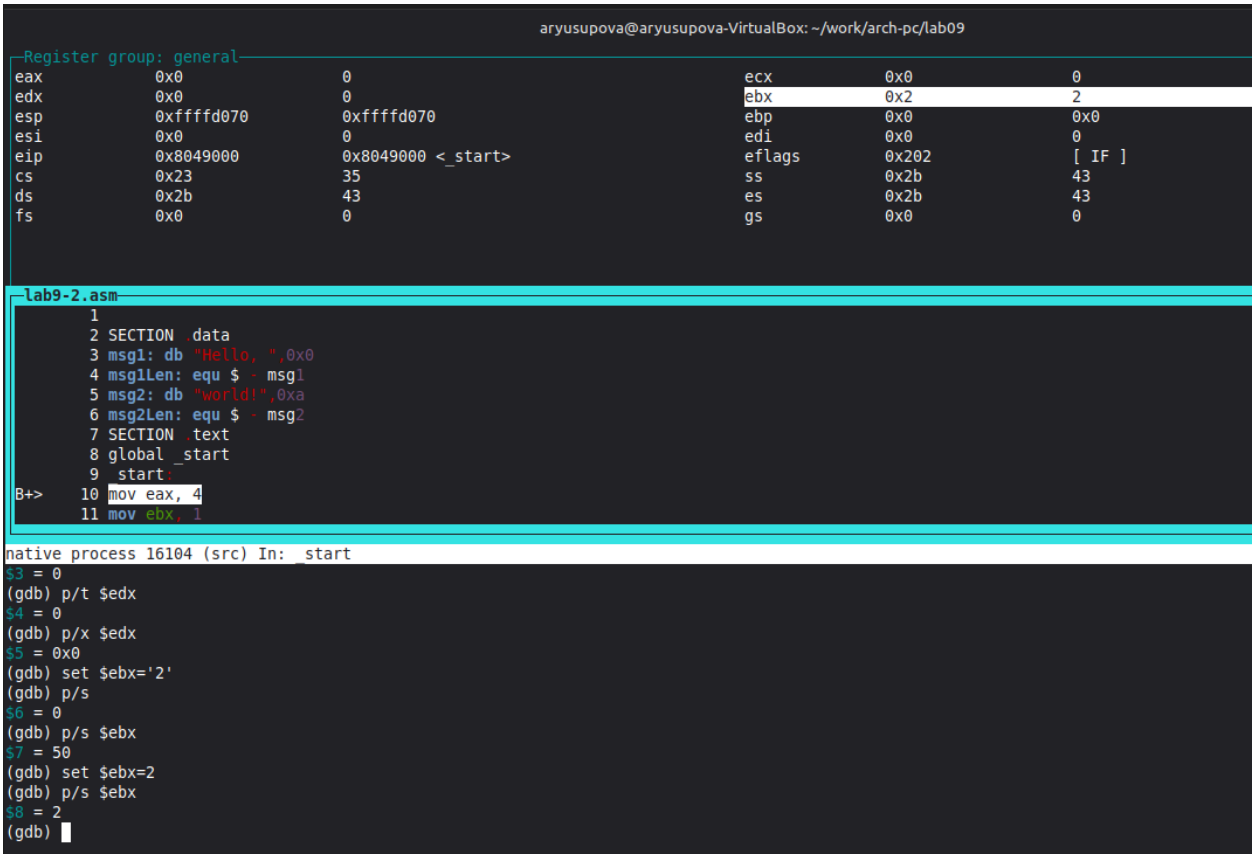

Рисунок 20: Пример использования команды set

p/s $ebx: Интерпретирует значение регистра как указатель на строку (адрес в памяти) и выводит строку, начиная с этого адреса, до нулевого терминатора.

Разница в выводе (50 vs 2) обусловлена разницей между ASCII-кодом символа '2' и числом 2.

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q) (рис. 21).

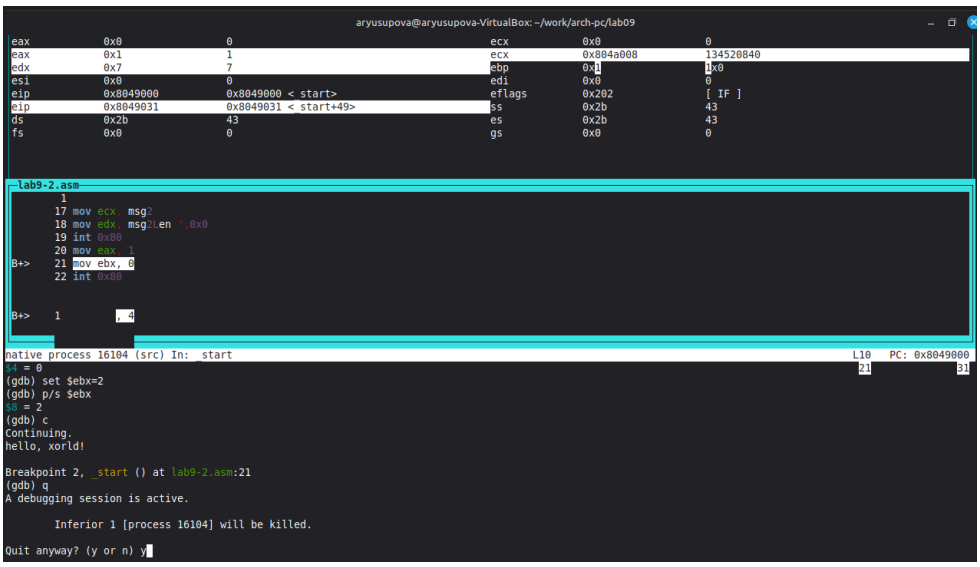

Рисунок 21: Завершение программы

4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог (рис. 22) и создаю исполняемый файл с файлом листинга и отладки (рис. 23).

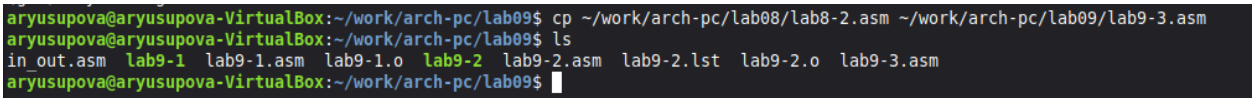

Рисунок 22: Копирование файла из лабораторной №8

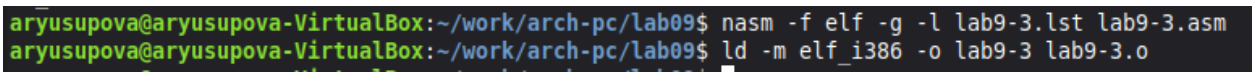

Рисунок 23: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args. Загружаю исполняемый файл в отладчик, указав аргументы (рис. 24).

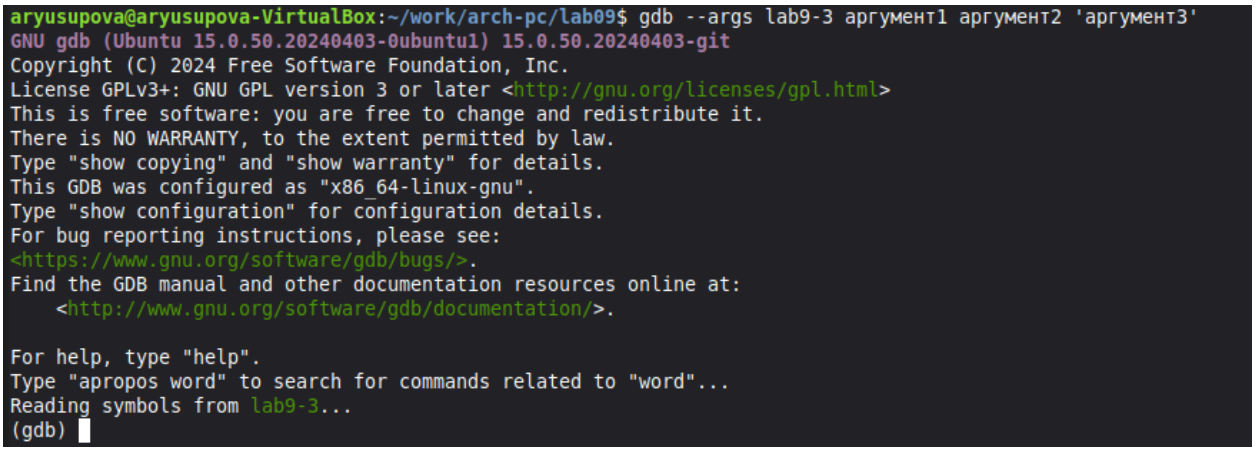

Рисунок 24: Загрузка исполняемых файлов в отладчике

Исследую расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установлю точку останова перед первой инструкцией в программе и запущу ее.

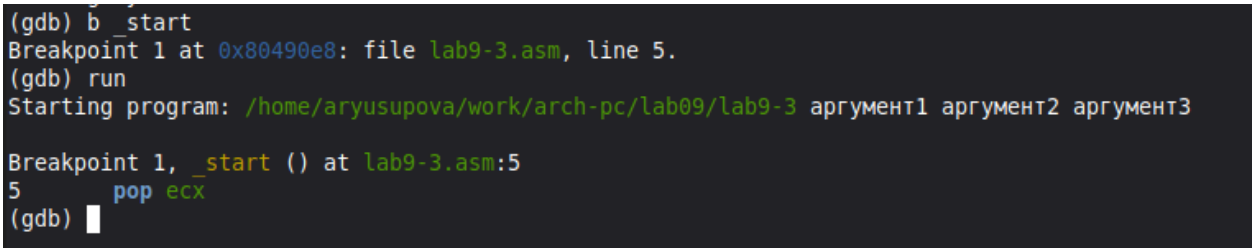

Рисунок 25: Установка точки останова и её запуск

В 32-битных системах размер указателя (адреса) составляет 4 байта. Каждый элемент массива argv[] - это указатель на строку. Поэтому каждый следующий аргумент смещен на 4 байта относительно предыдущего

$esp + 0: argc (количество аргументов)

$esp + 4: argv[0] → указатель на имя программы

$esp + 8: argv[1] → указатель на первый аргумент

$esp + 12: argv[2] → указатель на второй аргумент

$esp + 16: argv[3] → указатель на третий аргумент

$esp + 20: argv[4] → NULL (0x0) - конец массива

$esp + 24: envp[0] → первая переменная окружения

4.2 Задание для самостоятельной работы

1. Задание

Для выполнения этого задания для начала копирую файл lab8-1.asm в каталог ~/work/arch-pc/lab09 (рис. 26).

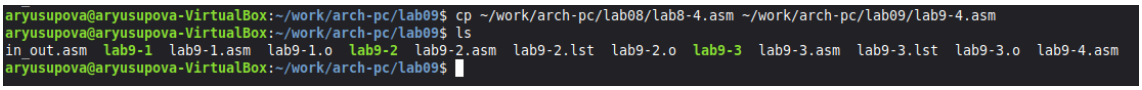

Рисунок 26: Копирование файла в рабочий каталог

Меняю программу lab9-4.asm из самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 27).

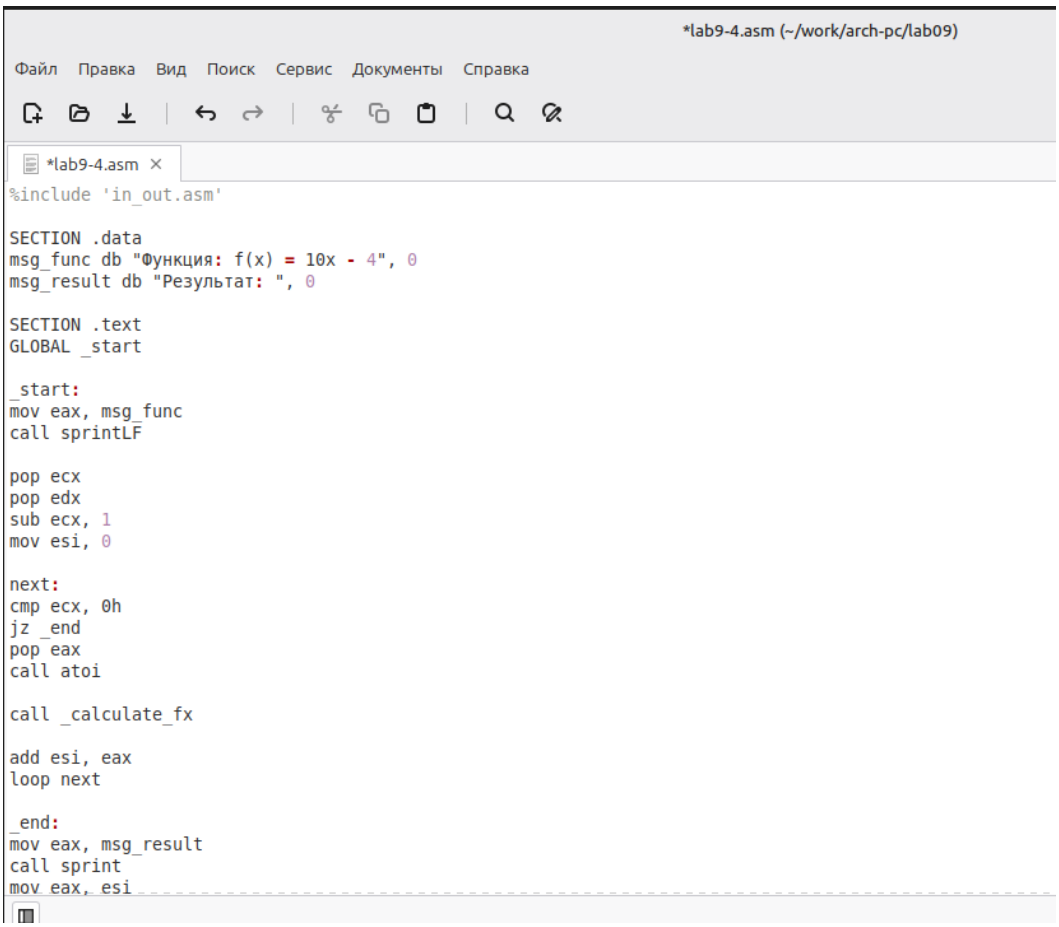

Рисунок 27: Изменение программы lab9-4.asm

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintLF

pop ecx
```

```
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
```

2. Задание

Для начала создаю файл lab9-5.asm для работы в этом файле (рис.28).

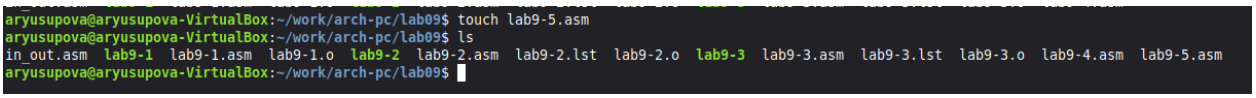

Рисунок 28: Создание файла lab9-5.asm

Копируем код из листинга 9.3 и вставляем в файл lab9-5.asm (рис. 29). Далее делаем отладку в gdb файла lab9-5.asm (рис. 30).

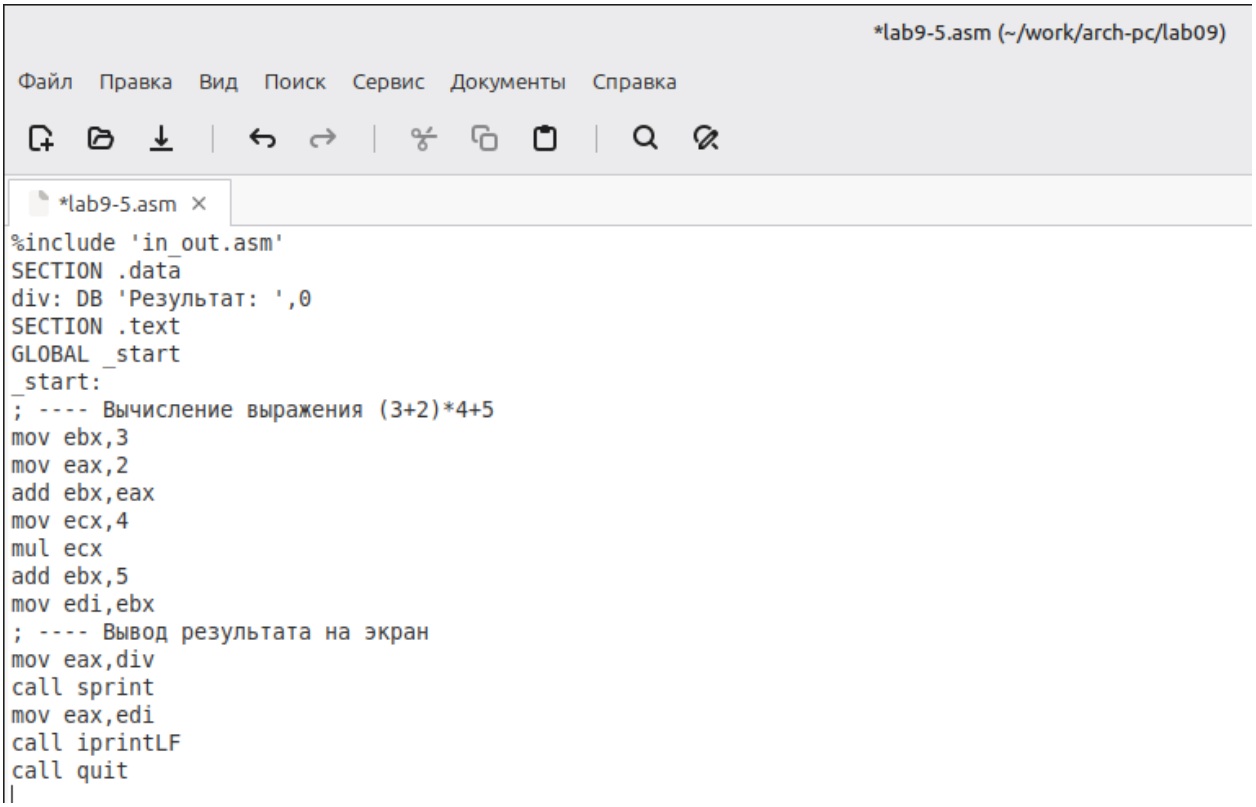

Рисунок 29: Код из листинга

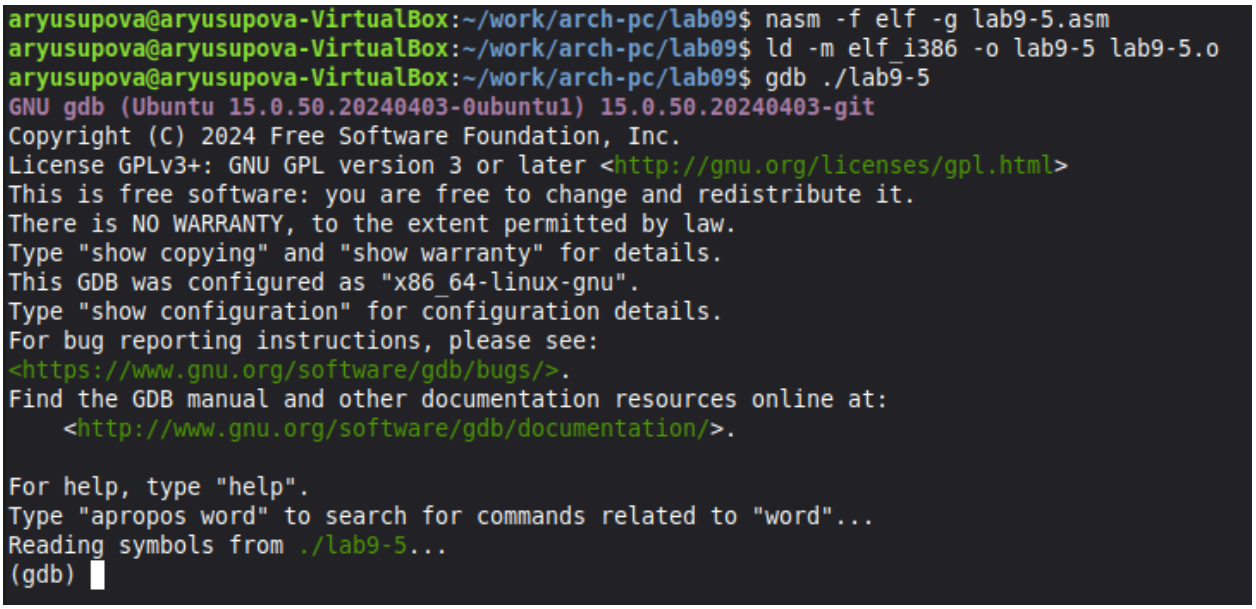

Рисунок 30: Отладка в gdb

Включаю режим псевдографики для более удобного анализа программы (рис. 31).

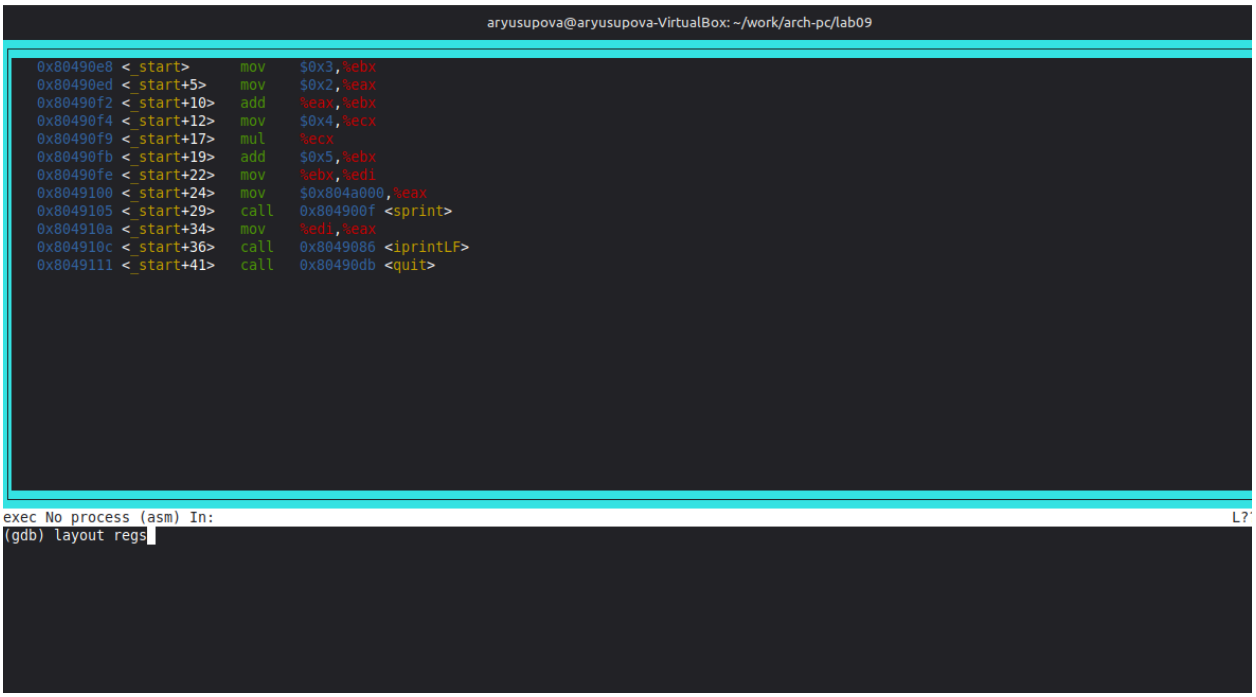

Рисунок 31: Режим псевдографики

Для более подробного анализа программы установила брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустила её (рис. 32).

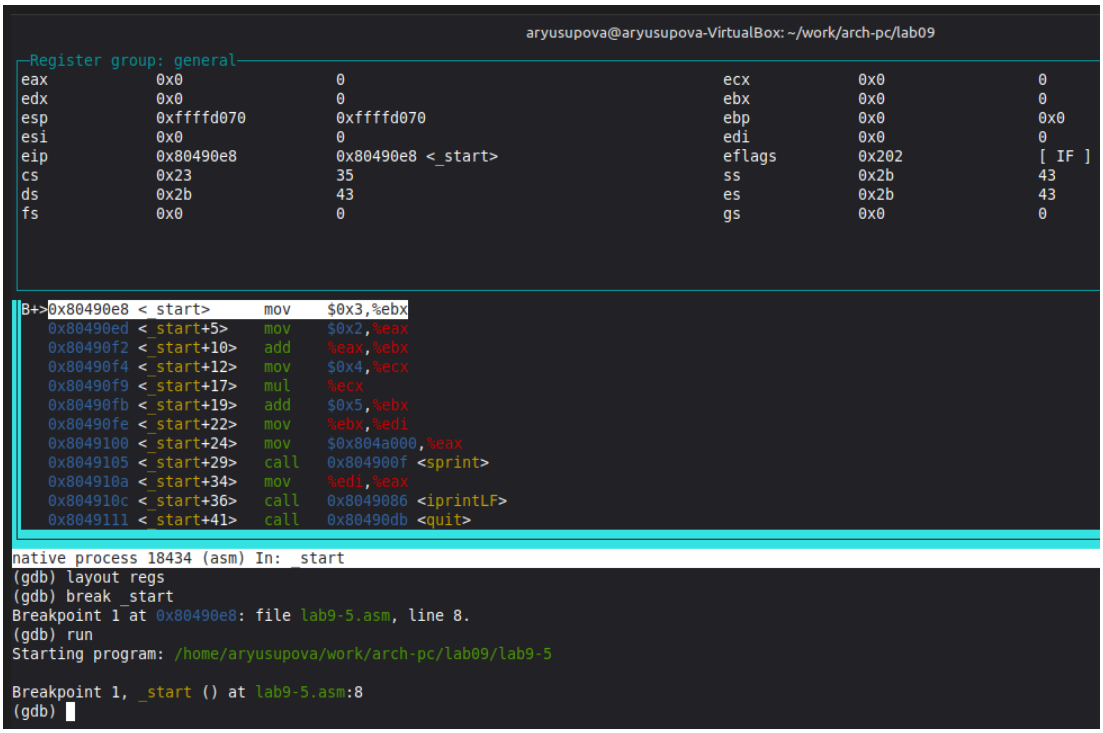

Рисунок 32: Запуск программы и установка метки

Устанавливаем отображение регистров по шагам:

Шаг 1: Устанавливаем отображение регистров (mov ebx,3) (рис. 33)

Шаг 2: Проверка ошибки. Обошлось без них (mov eax,2) (рис. 34)

Шаг 3: Снова проверка ошибки (add ebx,eax) (рис. 35)

Шаг 4: Ещё одна проверка ошибки (mov ecx,4)(рис. 36).

Шаг 5: Ошибка найдена (рис. 37). Команда mul ecx умножает регистр eax на ecx, но в eax осталось значение 2 из предыдущей операции, а не результат сложения (5).

Программа вычисляет:

(2 * 4) + 5 = 13 вместо (3 + 2) * 4 + 5 = 25

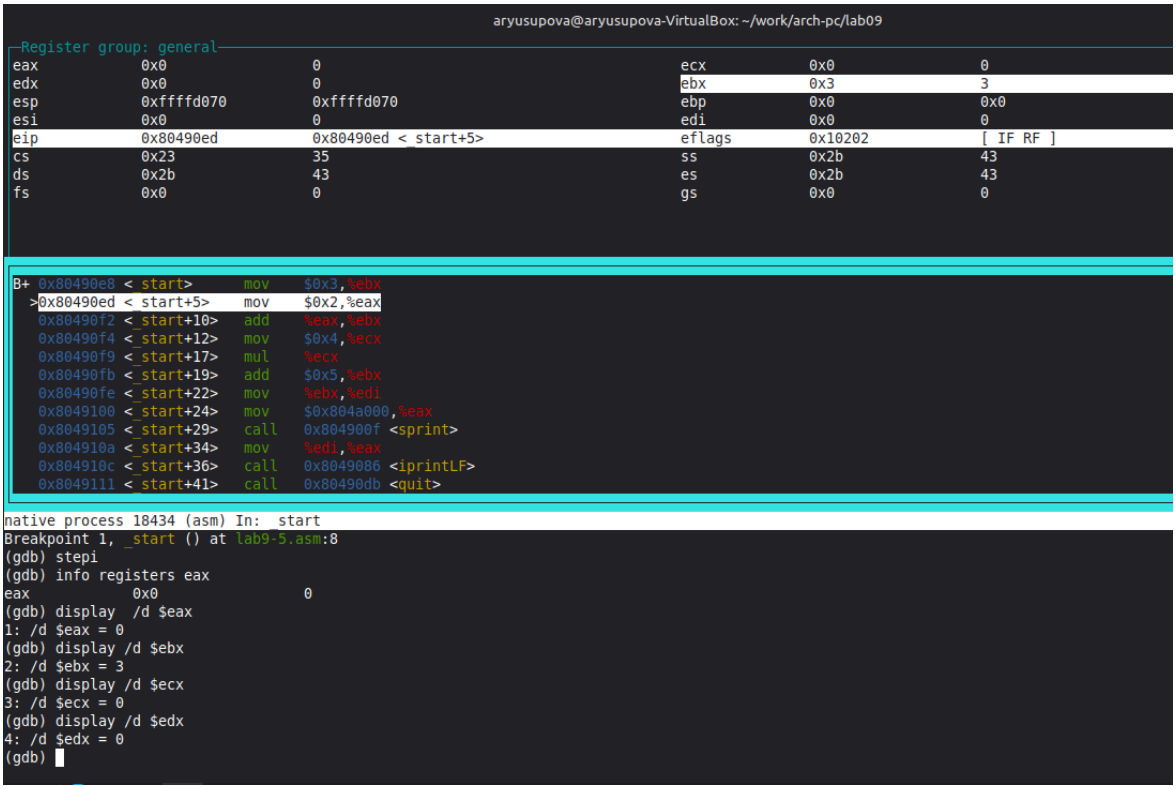

Рисунок 33: Шаг 1 (Установление отображения регистров)

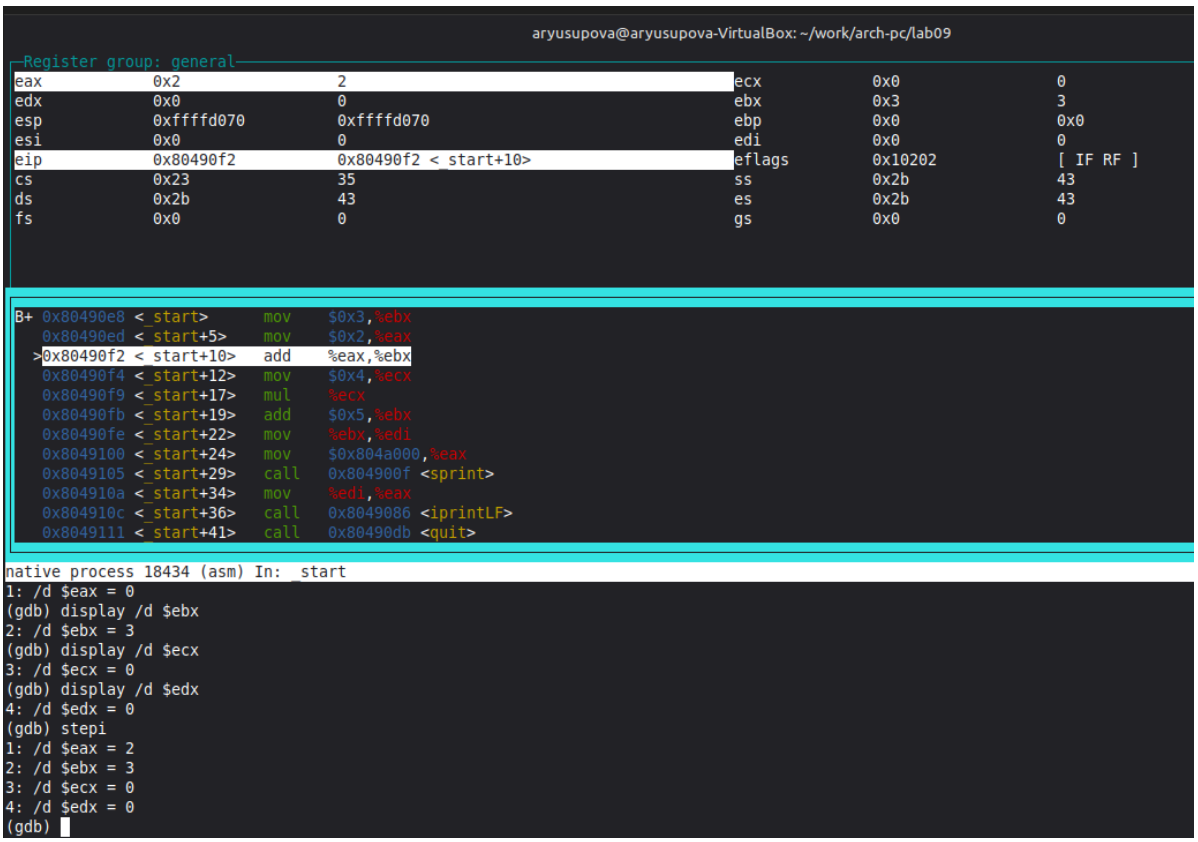

Рисунок 34: Шаг 2 проверки

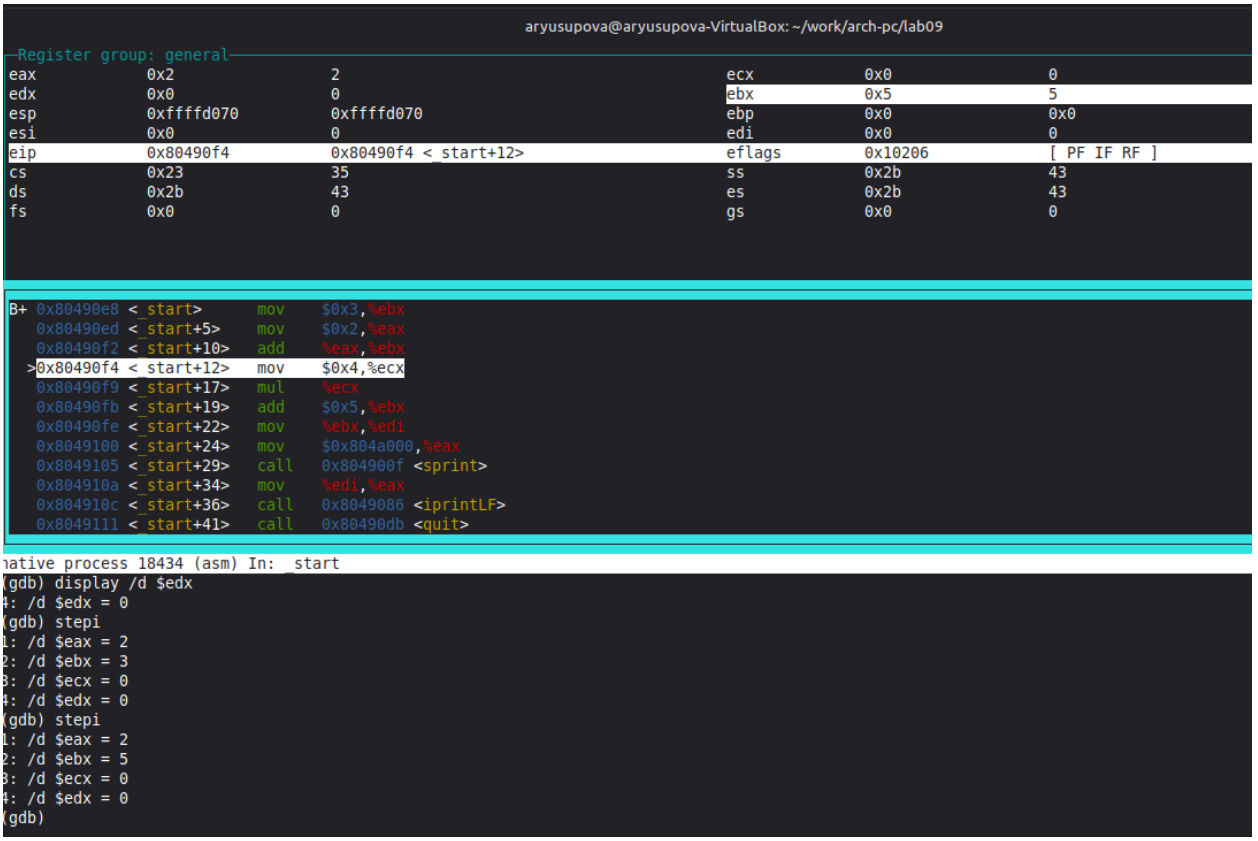

Рисунок 35: Шаг 3 проверки

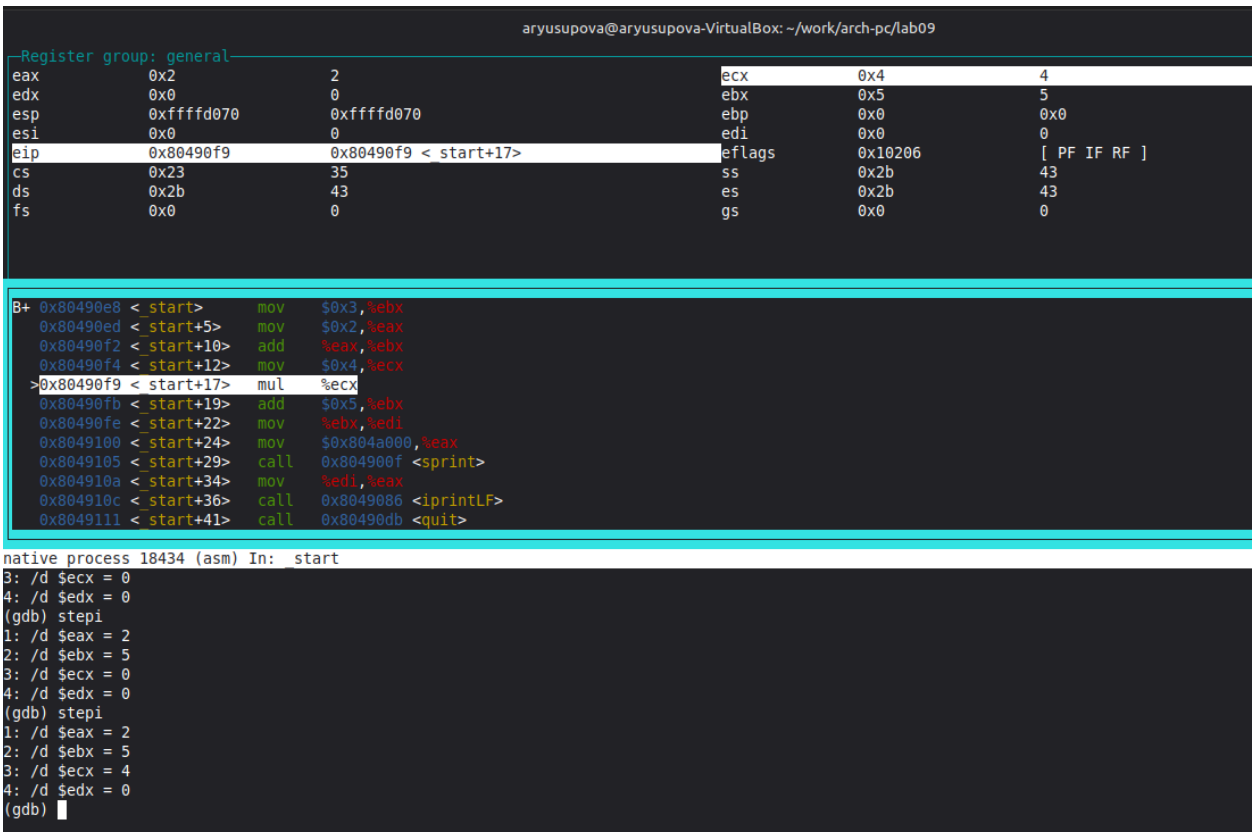

Рисунок 36: Шаг 4 проверки

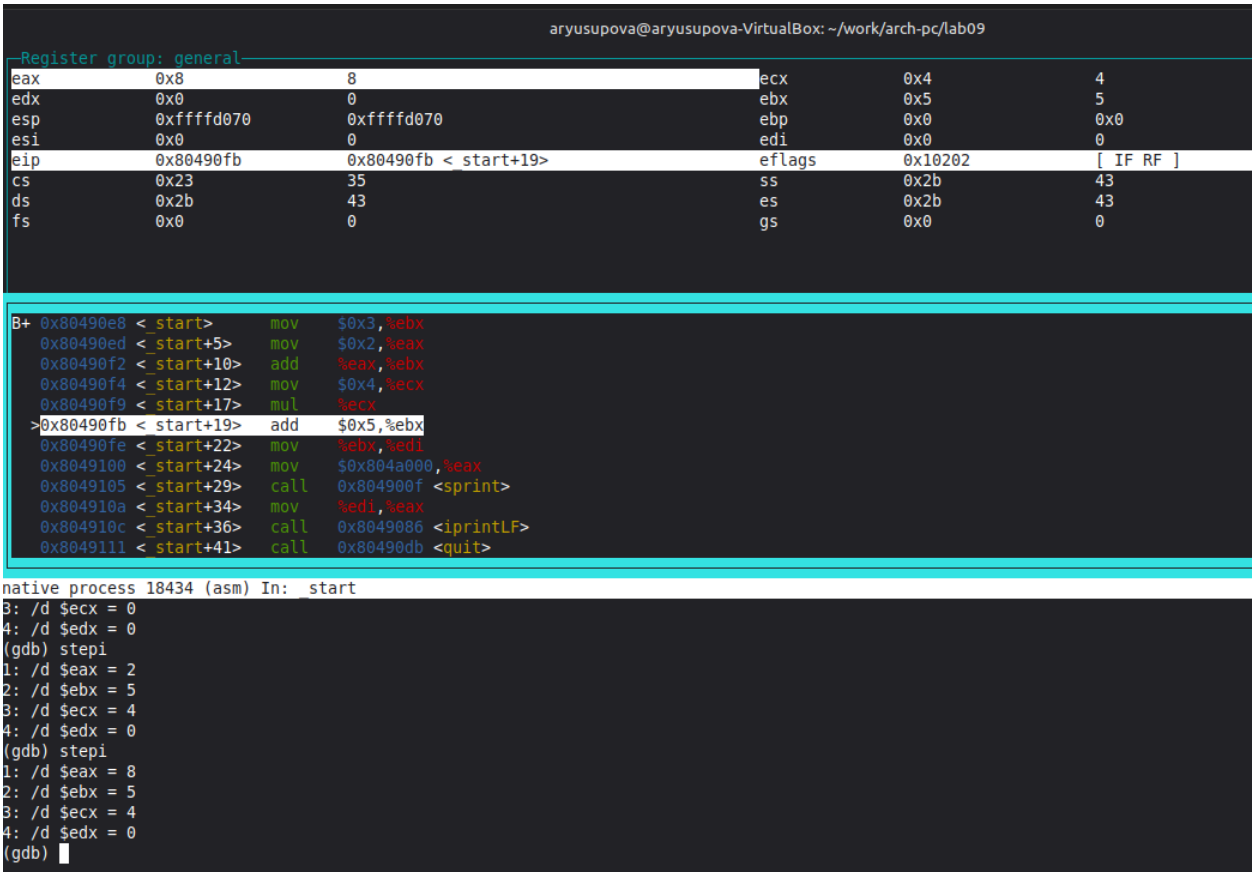

Рисунок 37: Шаг 5 (Найденная ошибка)

Исправим ошибки кода в файле lab9-5.asm (рис. 38).

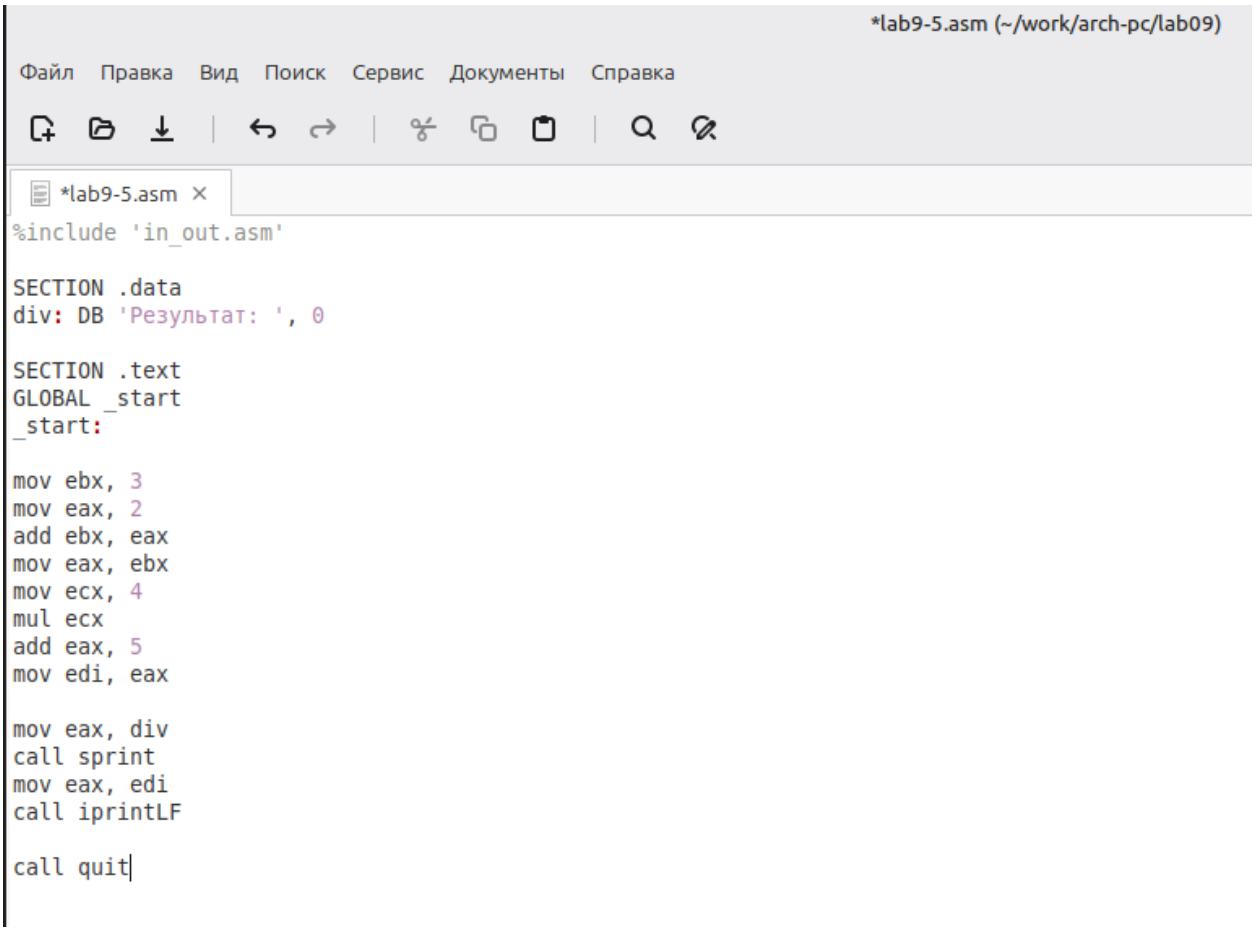

Рисунок 38: Изменение программы lab9-5.asm

Запускаю изменённую программу и вижу, что она работает корректно (рис. 39).

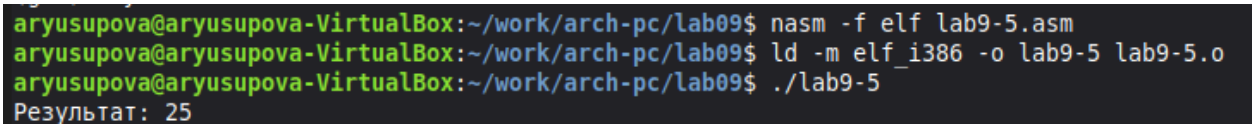

Рисунок 39: Запуск изменённой программы lab9-5.asm

Код измененной программы:

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ', 0

SECTION .text
GLOBAL _start
```

```
_start:

mov ebx, 3
mov eax, 2
add ebx, eax
mov eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit
```

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. https://esystem.rudn.ru/mod/resource/view.php?id=1030457

2. https://esystem.rudn.ru/mod/resource/view.php?id=1030557

3. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).