Отчёт по лабораторной работе №9

Лобанова Екатерина Евгеньевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. Рисунок 2.1).

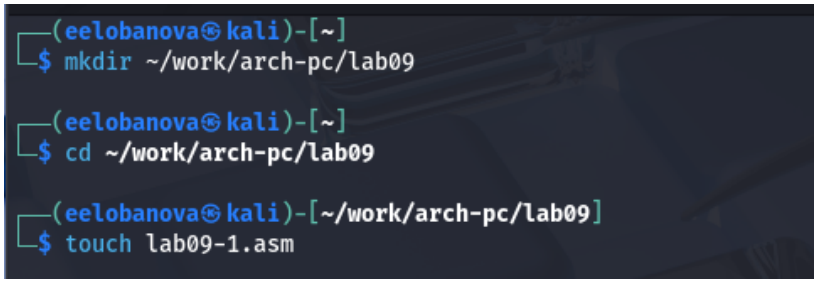

Рисунок 2.1: Создаем каталог с помощью команды mkdir и файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. Рисунок 2.2).

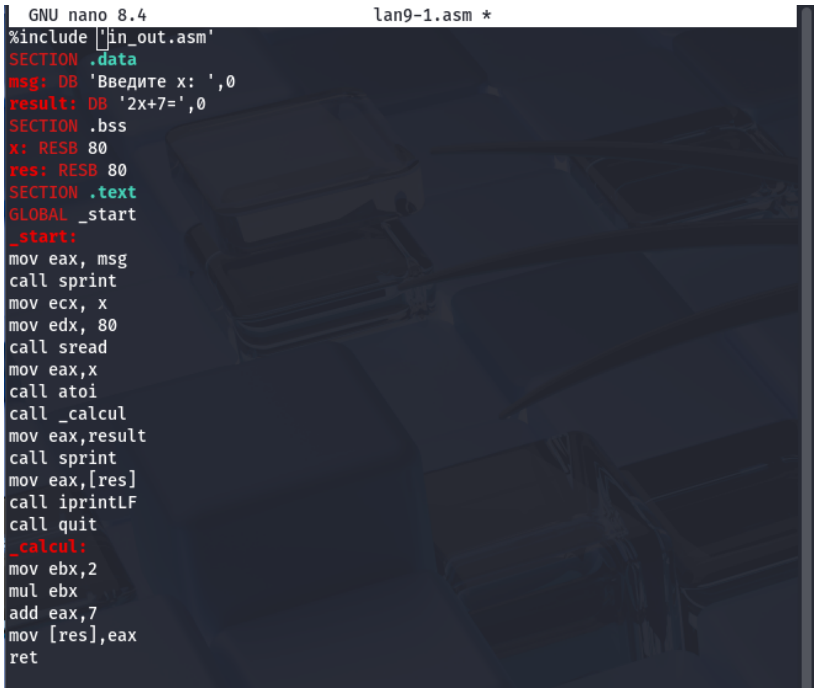

Рисунок 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.3).

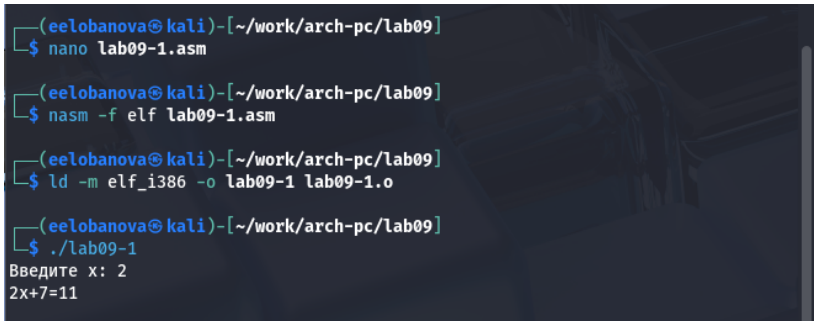

Рисунок 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. Рисунок 2.4).

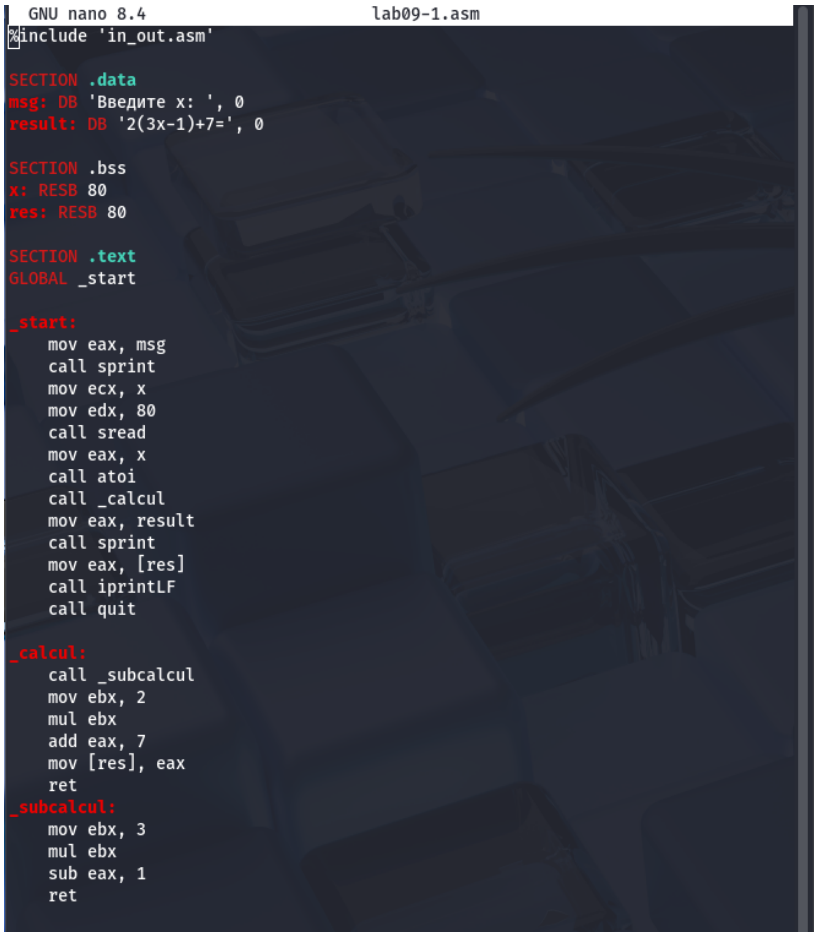

Рисунок 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.5).

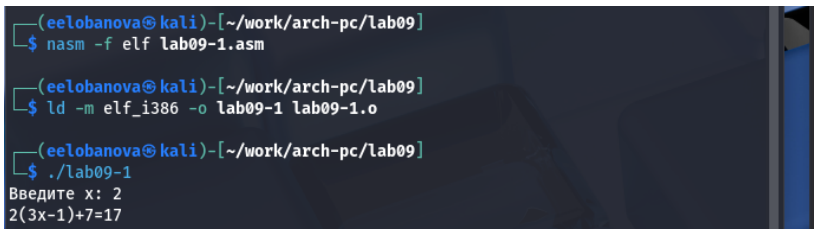

Рисунок 2.5: Запускаем файл и смотрим на его работу

2.2 Отладка программам с помощью GDB

Создаем новый файл в каталоге(рис. Рисунок 2.6).

```
┌──(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ touch lab09-2.asm
```

Рисунок 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. Рисунок 2.7).

```
 GNU nano 8.4                    lab09-2.asm
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рисунок 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. Рисунок 2.8).

```
┌──(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ nasm -f elf -g -l lab09-2.lst lab09-2.asm

┌──(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ ld -m elf_i386 -o lab09-2 lab09-2.o

┌──(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ gdb lab09-2
GNU gdb (Debian 16.3-5) 16.3
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 lab09-2...
```

Рисунок 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. Рисунок 2.9).

Рисунок 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку _start и запускаем программу (рис. Рисунок 2.10).

Рисунок 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки _start(рис. Рисунок 2.11).

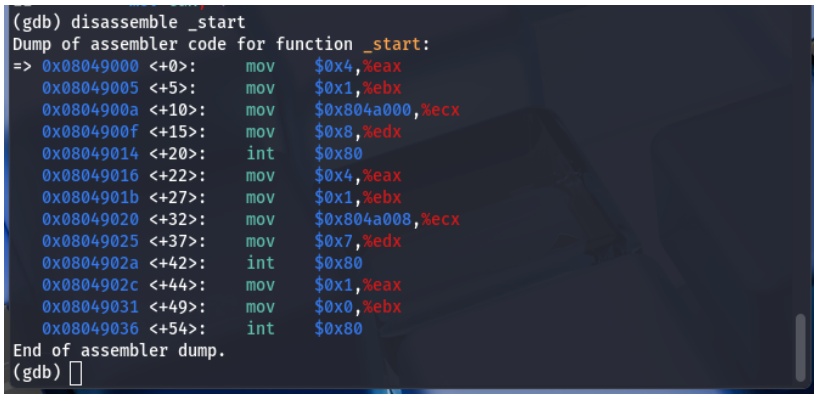

Рисунок 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. Рисунок 2.12).

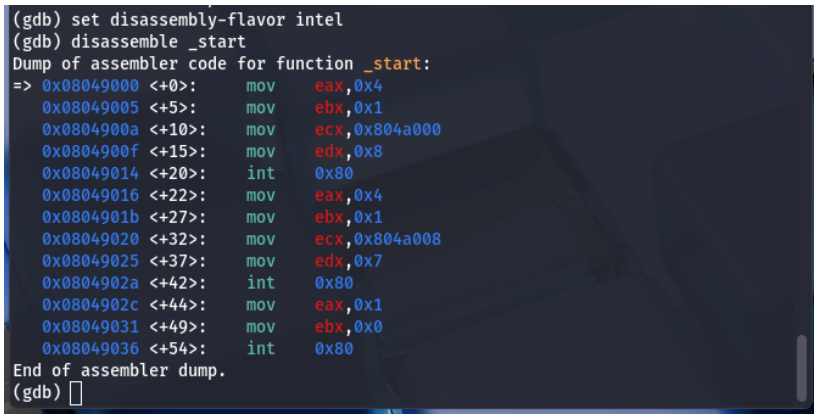

Рисунок 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1. Порядок операндов: В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2. Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3. Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как «b» (byte), «w» (word), «l» (long) и «q» (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как «b», «w», «d» и «q».

4. Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом «".$Intel$"».

5. Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6. Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа «%». В Intel синтаксисе обозначение регистра может начинаться с символа «R» или «E» (например, «%eax» или «RAX»).

Включаем режим псевдографики (рис. Рисунок 2.13).

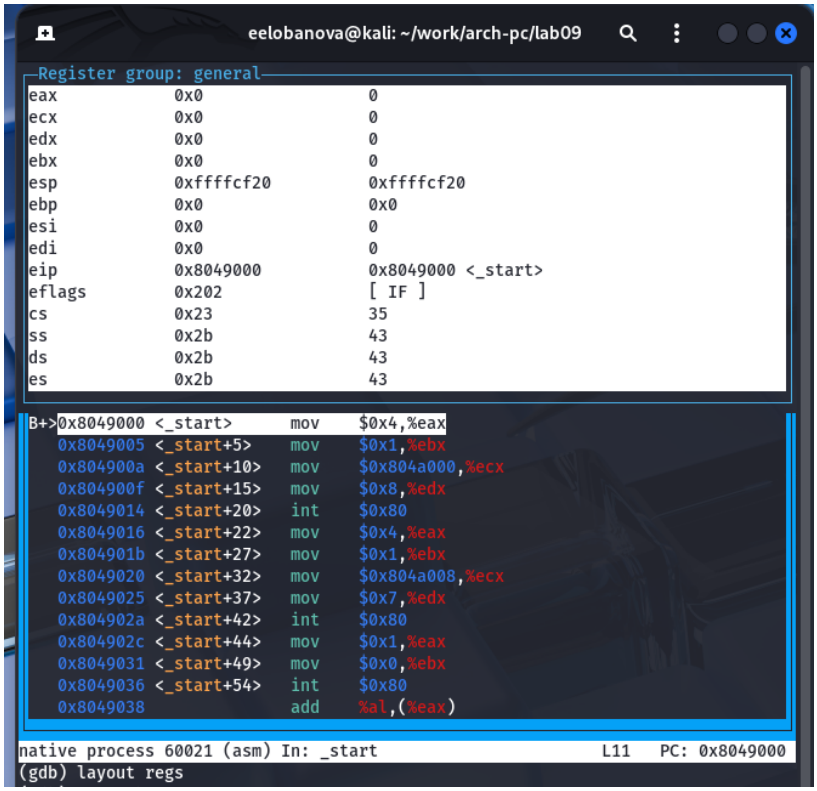

Рисунок 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. Рисунок 2.14).

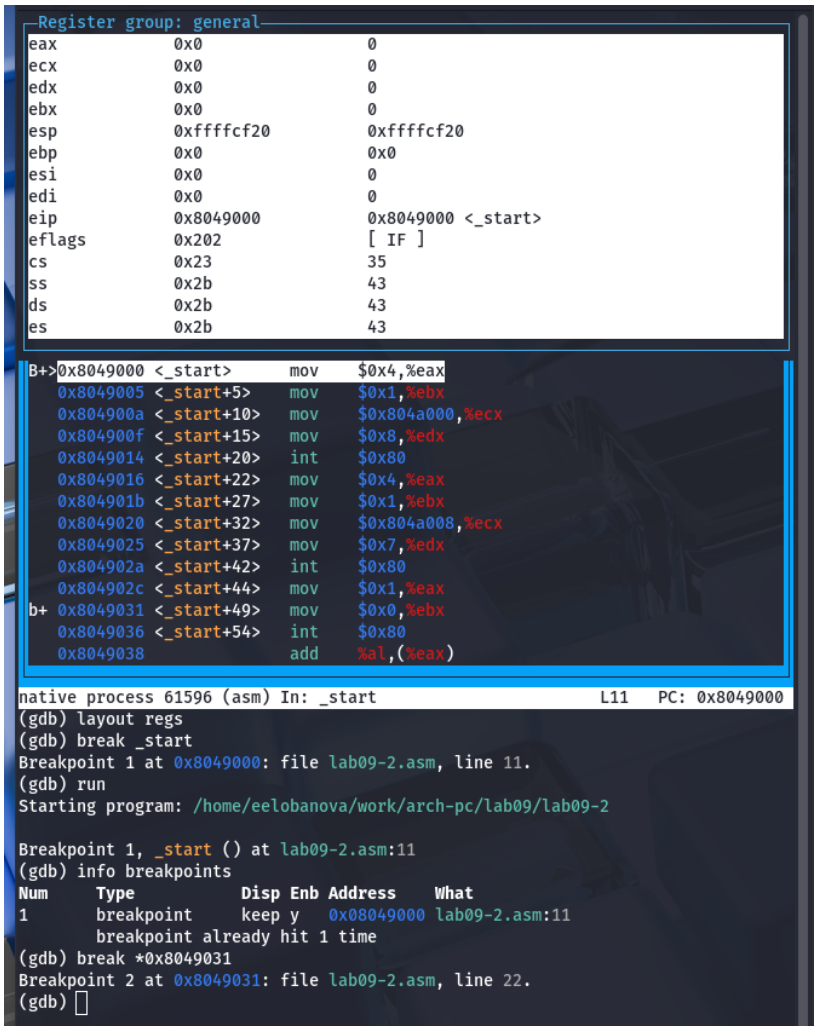

Рисунок 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. Рисунок 2.15).

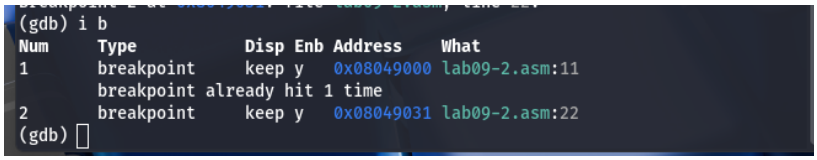

Рисунок 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. Рисунок 2.16).

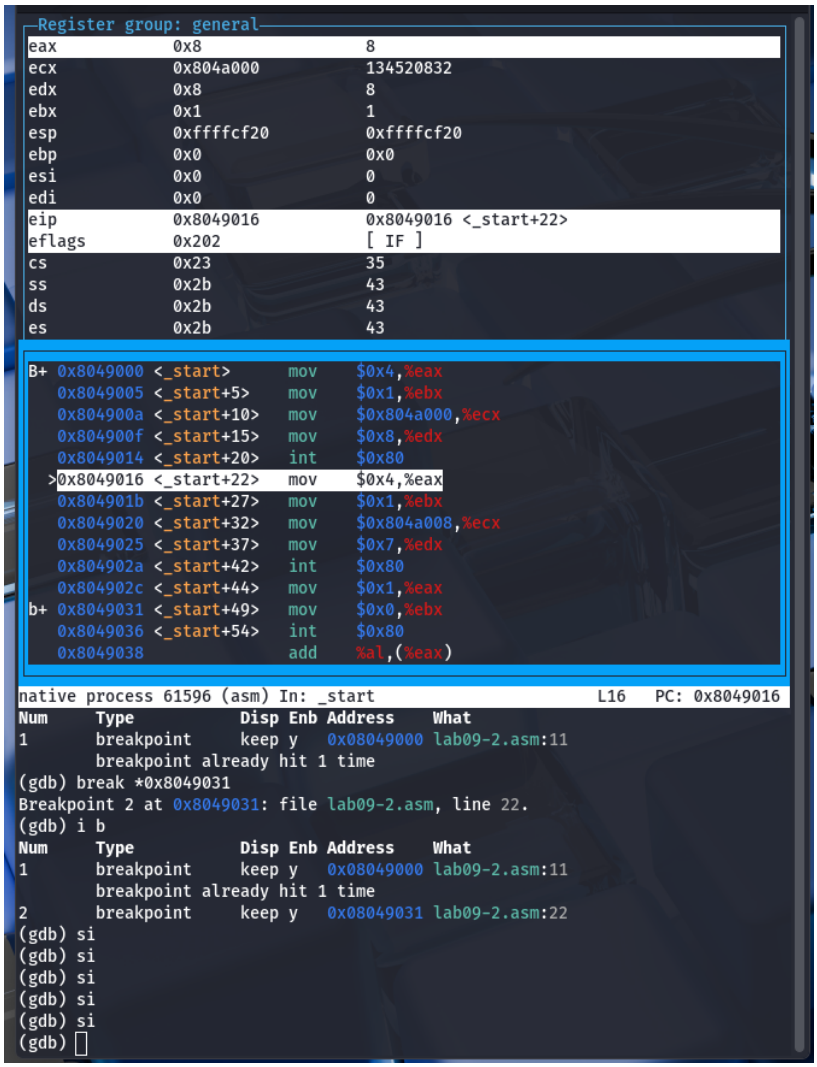

Рисунок 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx,eax, eip.

Смотрим значение переменной msg1 по имени (рис. Рисунок 2.17).

Рисунок 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. Рисунок 2.18).

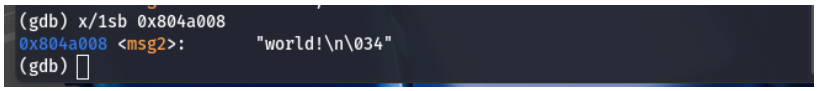

Рисунок 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. Рисунок 2.19).

Рисунок 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. Рисунок 2.20).

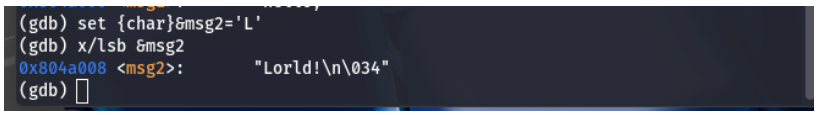

Рисунок 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. Рисунок 2.21).

Рисунок 2.21: Смотрим значение регистра

Изменяем регистор ebx (рис. Рисунок 2.22).

Рисунок 2.22: Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. Рисунок 2.23).

Рисунок 2.23: Прописываем команды с и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. Рисунок 2.24).

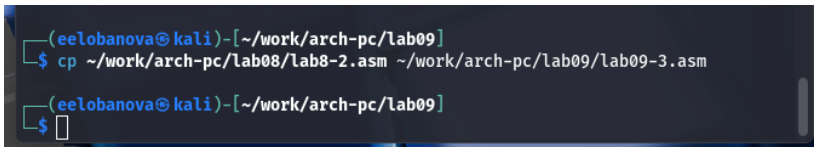

Рисунок 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. Рисунок 2.25).

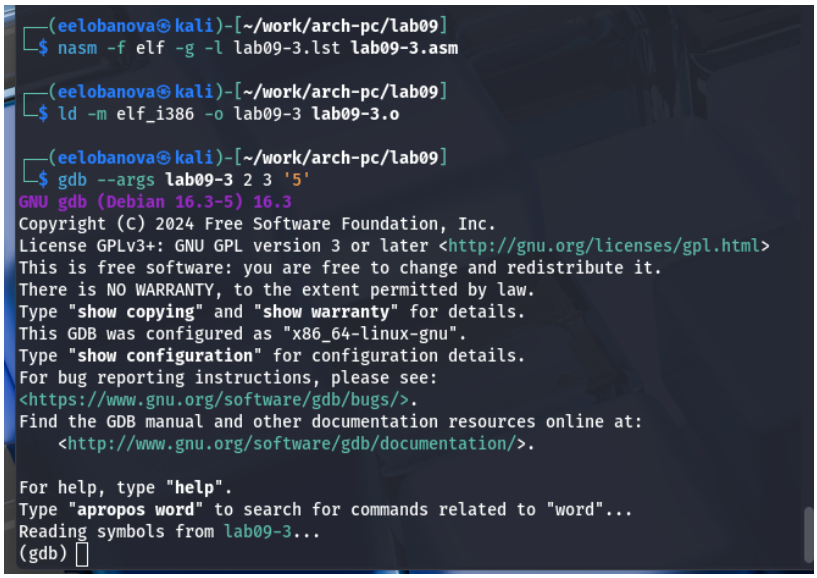

Рисунок 2.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. Рисунок 2.26).

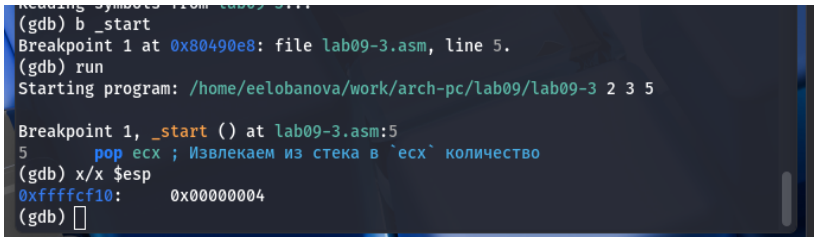

Рисунок 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. Рисунок 2.27).

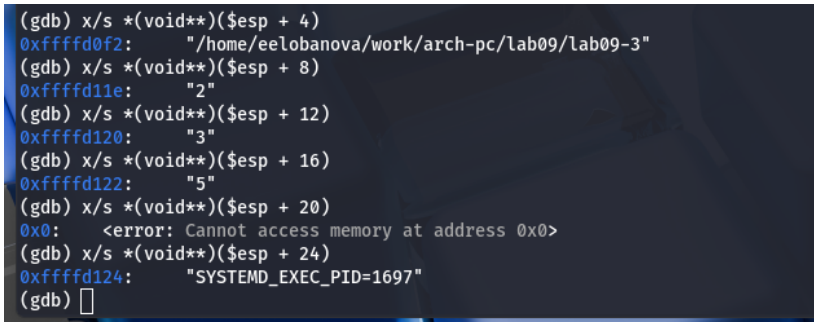

Рисунок 2.27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

2.3 Задание для самостоятельной работы

2.3.1 Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. Рисунок 2.28).

Рисунок 2.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. Рисунок 2.29).

Рисунок 2.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.30).

Рисунок 2.30: Проверяем работу программы

2.3.2 Задание 2

Создаем новый файл в дирректории (рис. Рисунок 2.31).

Рисунок 2.31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. Рисунок 2.32).

Рисунок 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.33).

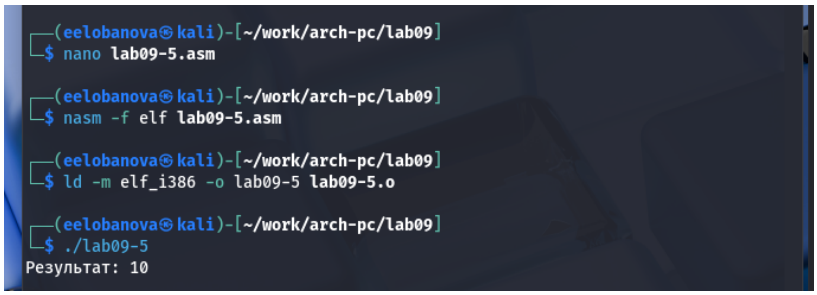

Рисунок 2.33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение решистров командой si (рис. Рисунок 2.34).

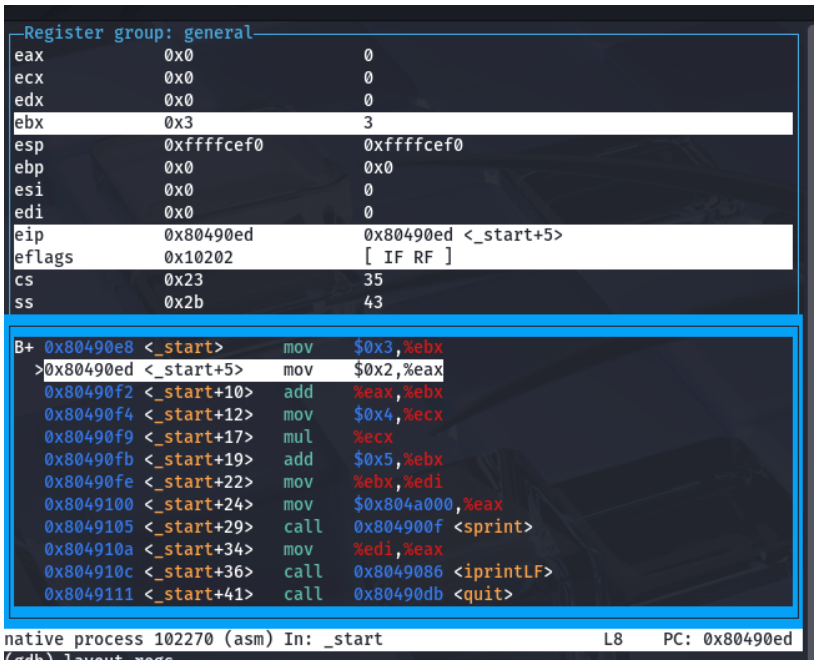

Рисунок 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. Рисунок 2.35).

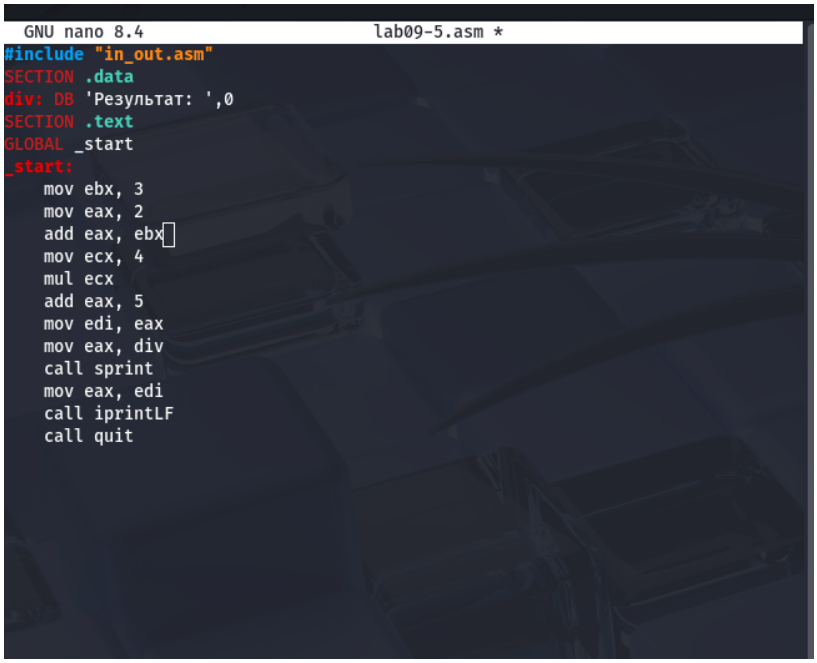

Рисунок 2.35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.36).

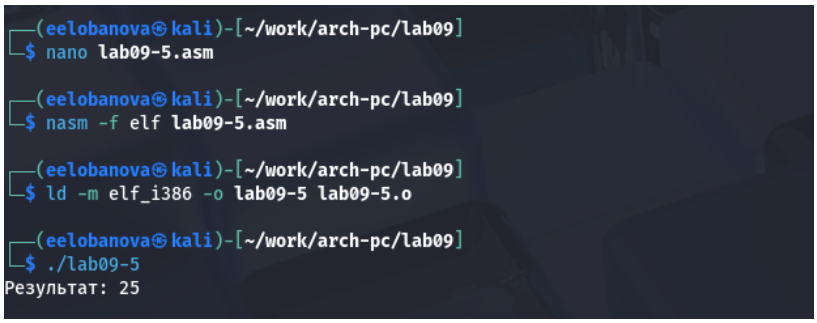

Рисунок 2.36: Создаем и запускаем файл(работает корректно)

3 Выводы

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

::: {#refs} :::