Отчёт по лабораторной работе №9

Иванова Анастасия Сергеевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm рис. 2.1:

Рисунок 2.1: Создание каталога и файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы _calcul.

Введем в файл lab09-1.asm текст программы рис. 2.2:

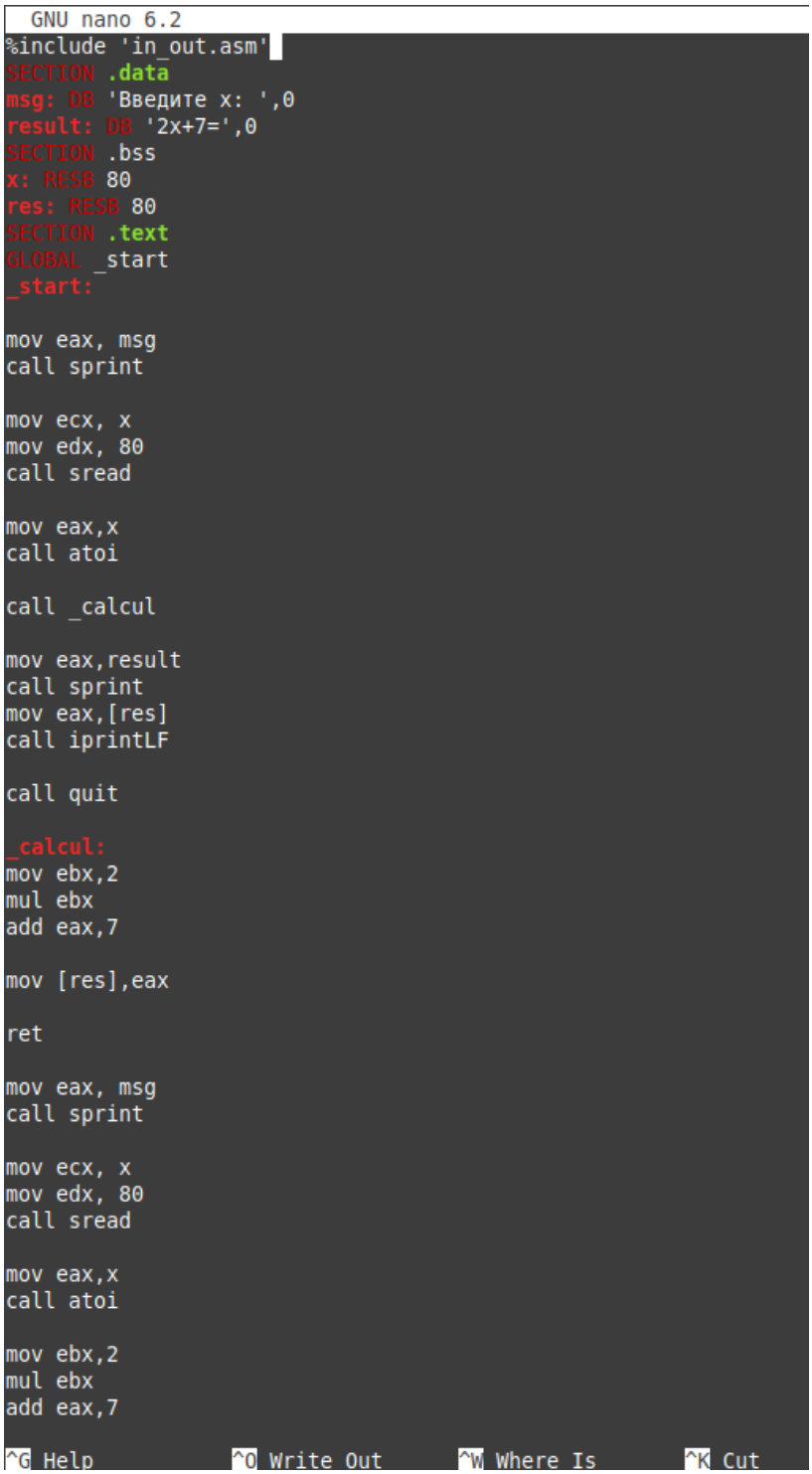

Рисунок 2.2: Ввод текста программы

Создадим исполняемый файл и проверим его работу рис. 2.3:

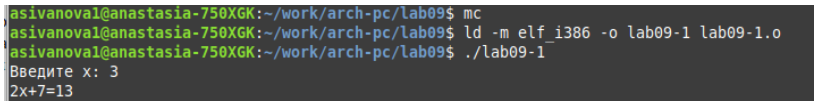

Рисунок 2.3: Создание исполняемого файла и запуск

Изменим 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)) рис. 2.4:

```
 GNU nano 6.2
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x)) = 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 eax, [esp+4]
mov ebx, 3
mul ebx
sub eax, 1

ret
```

Рисунок 2.4: Изменение текста программы

Запустим измененный файл рис. 2.5:

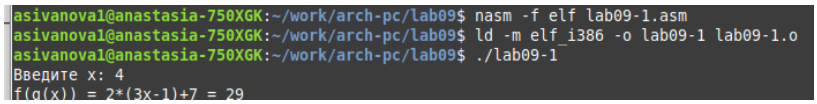

Рисунок 2.5: Проверка работы

3. Создадим файл lab09-2.asm с текстом программы печати сообщения Hello world! рис. 2.6:

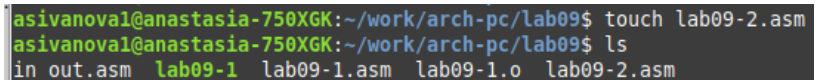

Рисунок 2.6: Создание файла

рис. 2.7:

Рисунок 2.7: Ввод программы

Получим исполняемый файл и трансляцию программ, а затем загрузим исполняемый файл в отладчик gdb рис. 2.8:

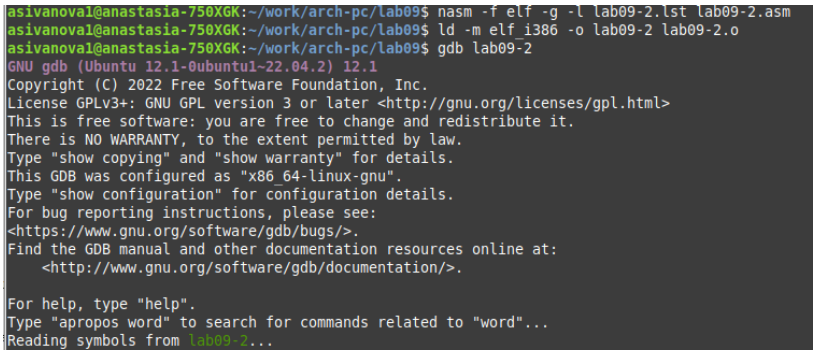

Рисунок 2.8: Получение исполняемого файла и загрузка файла в отладчик gdb

Проверим работу программы, запустим ее в оболочке GDB с помощью команды run рис. 2.9:

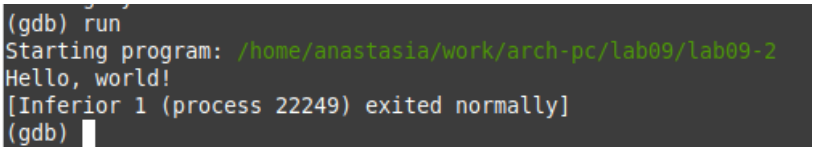

Рисунок 2.9: Запуск программы

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её рис. 2.10:

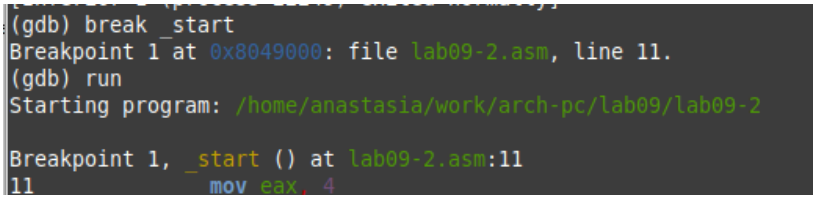

Рисунок 2.10: Установка метки

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start рис. 2.11:

Рисунок 2.11: Просмотр кода

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel рис. 2.12:

Рисунок 2.12: Переключение на отображение команд

Различия отображения синтаксиса: AT&T синтаксис более последователен (всегда источник⟶цель, явные суффиксы размера), а Intel синтаксис более читаем для человека (естественный порядок «куда⟵откуда»).

Включим режим псевдографики для более удобного анализа программы

рис. 2.13:

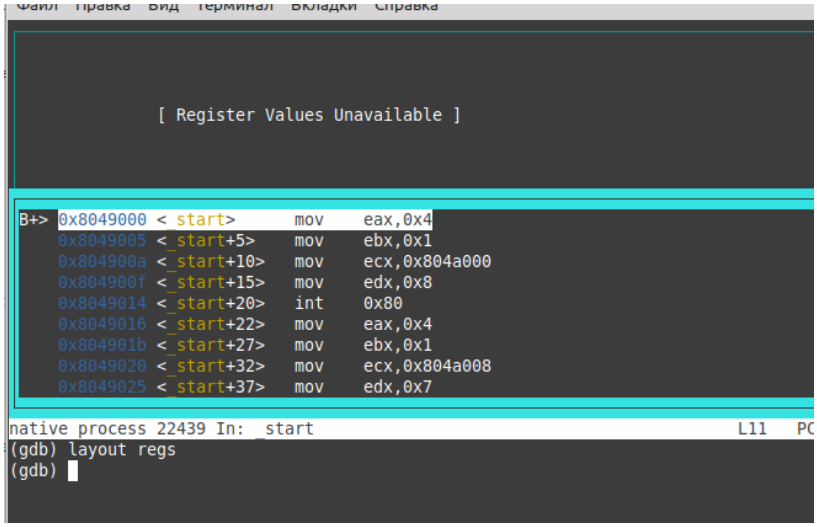

Рисунок 2.13: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверим это с помощью команды info breakpoints рис. 2.14:

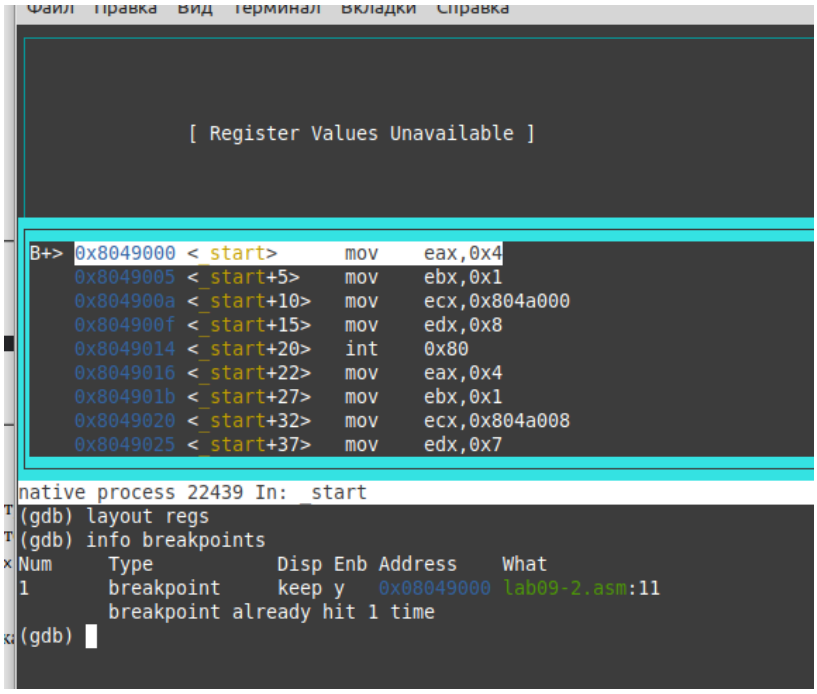

Рисунок 2.14: Проверка установленой точки

Установим еще одну точку остановки по адресу инструкции рис. 2.15:

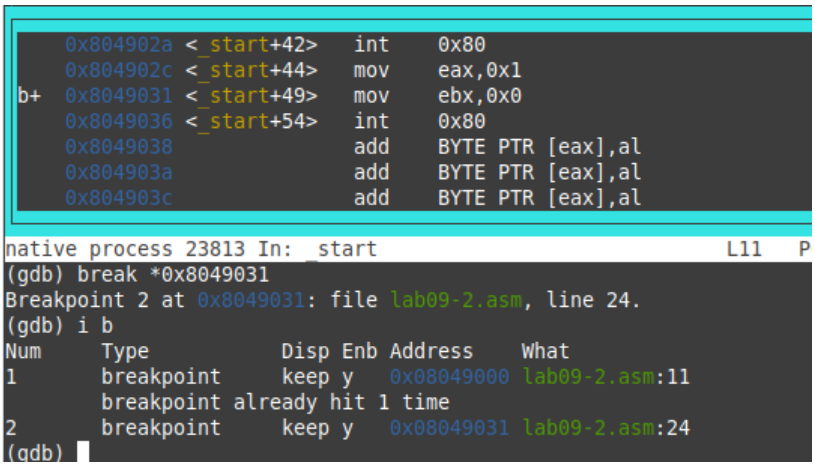

Рисунок 2.15: Установка еще одной точки

Посмотрим информацию о всех установленных точках останова рис. 2.16:

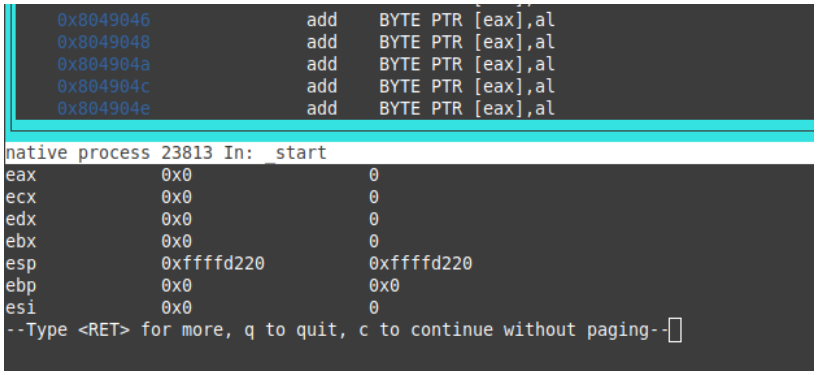

Рисунок 2.16: Просмотр информации о всех точках

Посмотрим значение переменной msg1 по имени рис. 2.17:

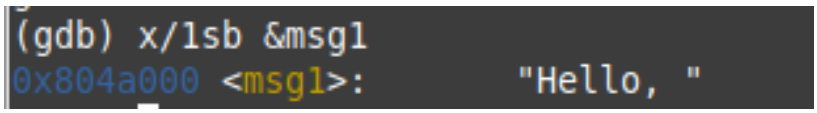

Рисунок 2.17: Просмотр значения

Посмотрим значение переменной msg2 по адресу рис. 2.18:

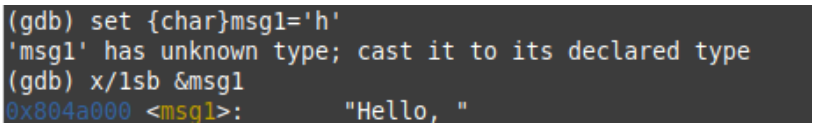

Рисунок 2.18: Просмотр значения

С помощью команды set изменим значение регистра ebx рис. 2.19:

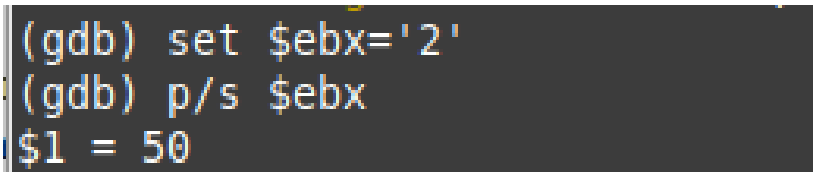

Рисунок 2.19: Изменение значения

рис. 2.20:

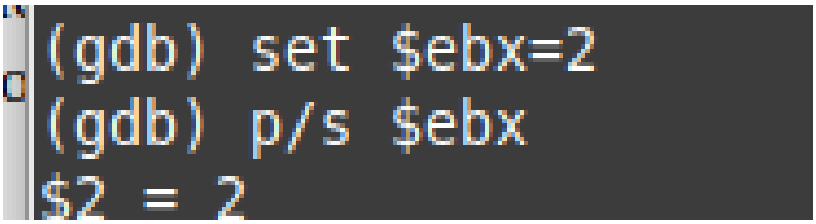

Рисунок 2.20: Изменение значения

Разницу вывода команд p/s $ebx: p/s — показывает число со знаком (как положительное или отрицательное), p/d или p — показывает как абсолютное значение (только положительное)

4. Скопируем 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, создадим исполняемый файл и загрузим исполняемый файл в отладчик, указав аргументы рис. 2.21:

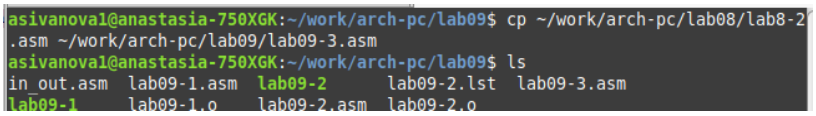

Рисунок 2.21: Копирование файла

Для начала установим точку останова перед первой инструкцией в программе и запустим ее рис. 2.22:

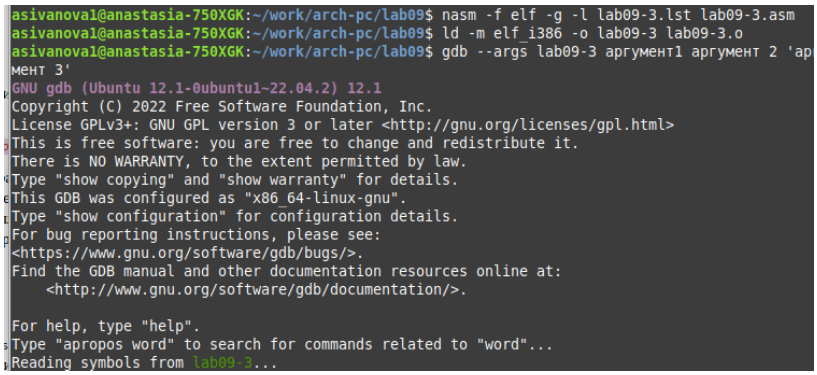

Рисунок 2.22: Установка точки

рис. 2.23:

Рисунок 2.23: Запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки. Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и „аргумент 3“.

Посмотрим остальные позиции стека рис. 2.24:

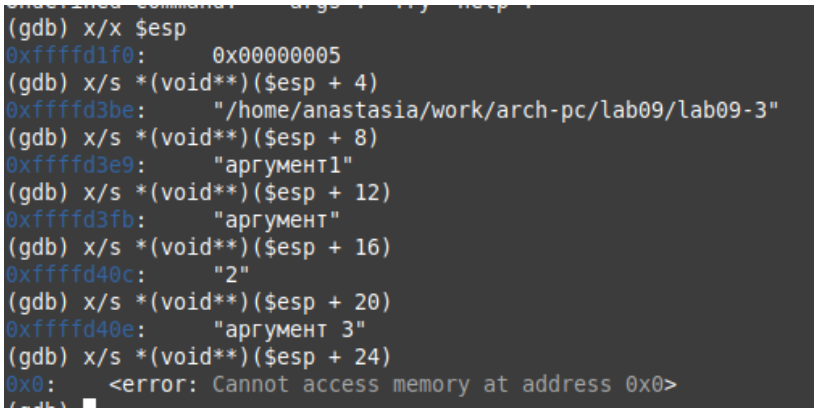

Рисунок 2.24: Позиции стека

Объясним, почему шаг изменения адреса равен 4: шаг равен 4 байтам потому что в 32-битной архитектуре каждый указатель на строку аргумента занимает 4 байта в памяти.

3 Задание для самостоятельной работы

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

Введем программу рис. 3.1:

```
 GNU nano 6.2
%include 'in_out.asm'
SECTION .data
func_msg db "Функция: f(x)=30x-11",0
result_msg db "Результат: ",0
SECTION .text
global _start
f:
mov ebx,30
mul ebx
sub eax,11
ret
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
next:
cmp ecx,0
jz _end
pop eax
call atoi
call f
add esi,eax
dec ecx
jmp next
_end:
mov eax,func_msg
call sprintLF
mov eax,result_msg
call sprint
mov eax,esi
call iprintLF
call quit
```

Рисунок 3.1: Ввод программы

Создадим исполняемый файл и проверим его работу рис. 3.2:

```
asivanova1@anastasia-750XGK:~/work/arch-pc/lab08$ nasm -f elf zadanie.asm
asivanova1@anastasia-750XGK:~/work/arch-pc/lab08$ ld -m elf_i386 -o zadanie zadanie.o
asivanova1@anastasia-750XGK:~/work/arch-pc/lab08$ ./zadanie
Функция: f(x)=30x-11
Результат: 0
```

Рисунок 3.2: Проверка работы

2. Нам приведена в пример программа вычисления выражения $(3 + 2) * 4 + 5$.

При запуске данная программа дает неверный результат. Проверим это, а затем помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее.

Создадим файл и запишем туда текст программы рис. 3.3:

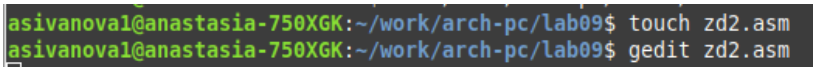

Рисунок 3.3: Создание файла

рис. 3.4:

```
 GNU nano 6.2                  /home/ana
%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 ecx,4
mul ecx
add ebx,5
mov edi,ebx
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Рисунок 3.4: Ввод программы

Создадим исполняемый файл и запустим программу рис. 3.5:

```
asivanova1@anastasia-750XGK:~/work/arch-pc/lab09$ nasm -f elf zd2.asm
asivanova1@anastasia-750XGK:~/work/arch-pc/lab09$ ld -m elf_i386 -o zd2 zd2.o
asivanova1@anastasia-750XGK:~/work/arch-pc/lab09$ ./zd2
Результат: 10
```

Рисунок 3.5: Запуск программы

С помощью GBD посмотрим что происходит в программе, а затем исправим код в файле рис. 3.6:

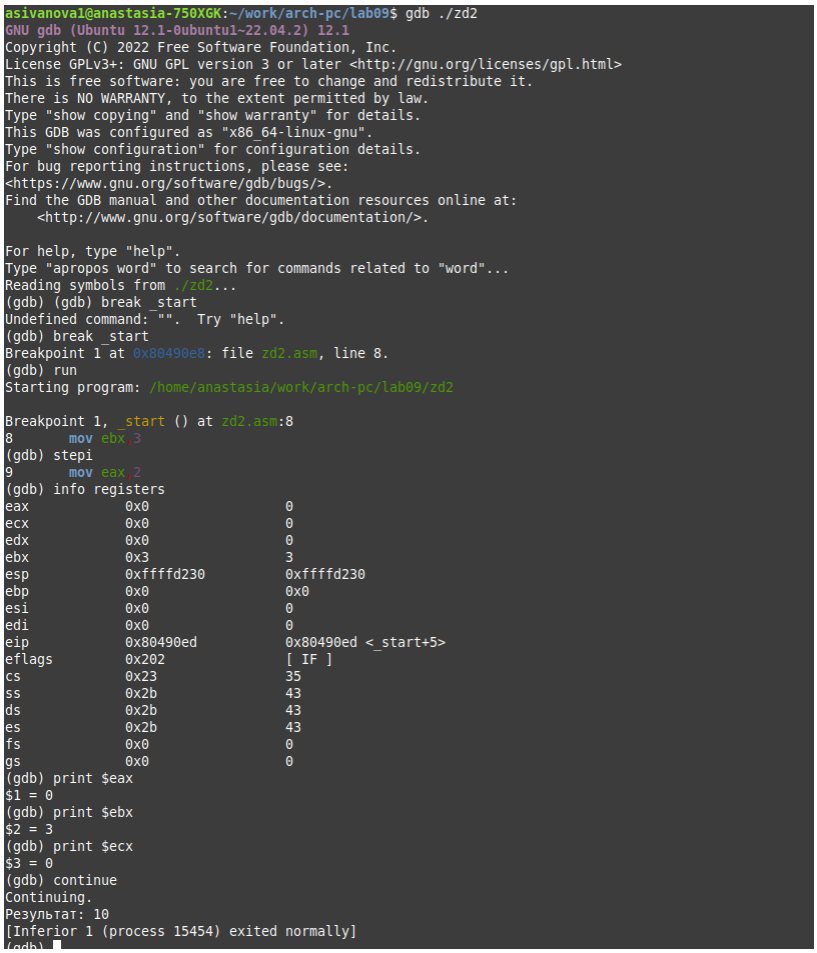

Рисунок 3.6: Просмотр процесс исполнения программы

рис. 3.7:

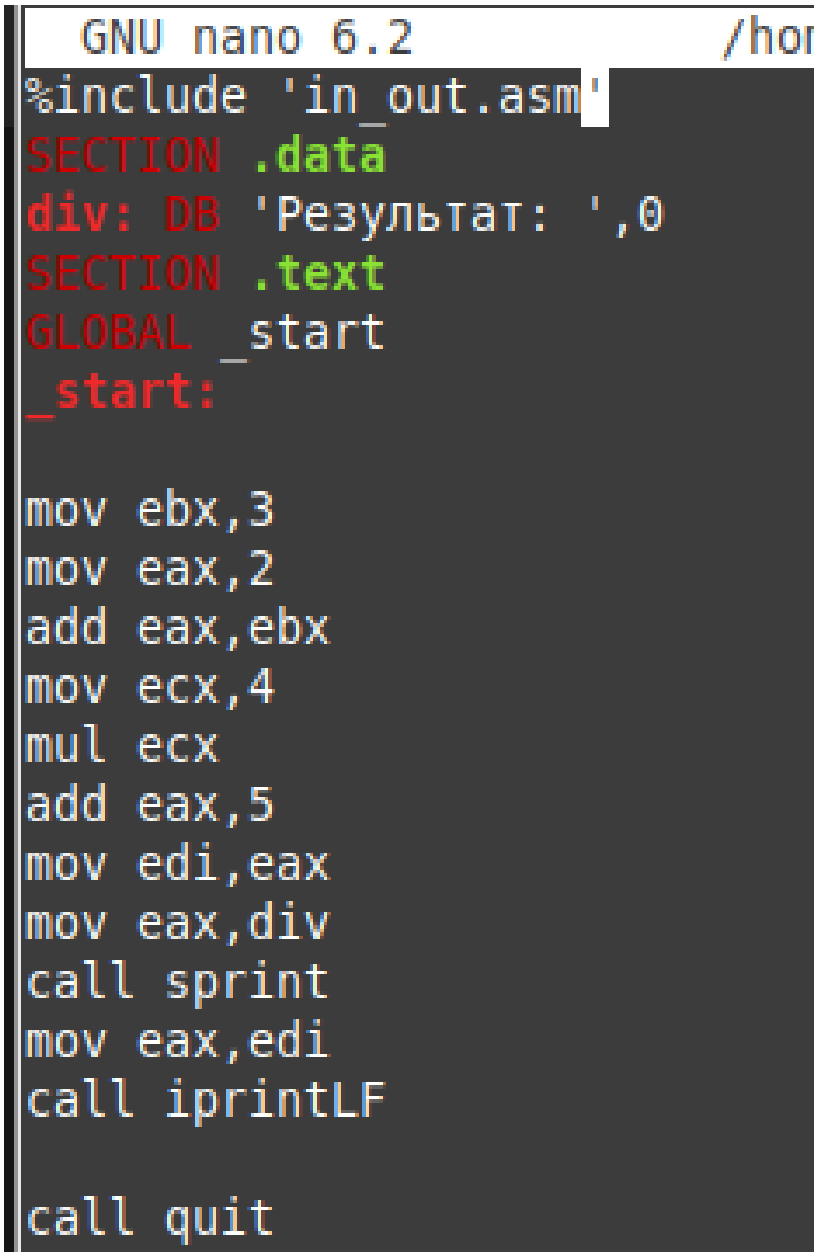

Рисунок 3.7: Изменение кода программы

Создадим исполняемый файл и запустим программу рис. 3.8:

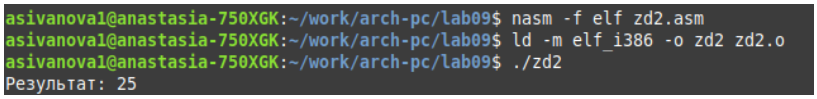

Рисунок 3.8: Проверка работы

Теперь программы вывлдит верное значение.

4 Вывод

Мы приобрели навыки написания программ с использованием подпрограмм, а также ознакомились с методами отладки при помощи GDB и его основными возможностями.