Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Пономарева Татьяна Александровна

Содержание

# 1 Цель работы

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

# 2 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 довольно 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 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

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

Терминал. Создание каталога lab09. Создание файла lab09-1.asm

Рис. 1: Терминал. Создание каталога lab09. Создание файла lab09-1.asm

Ввожу в файл lab09-1.asm текст программы из листинга 9.1 (рис. 2).

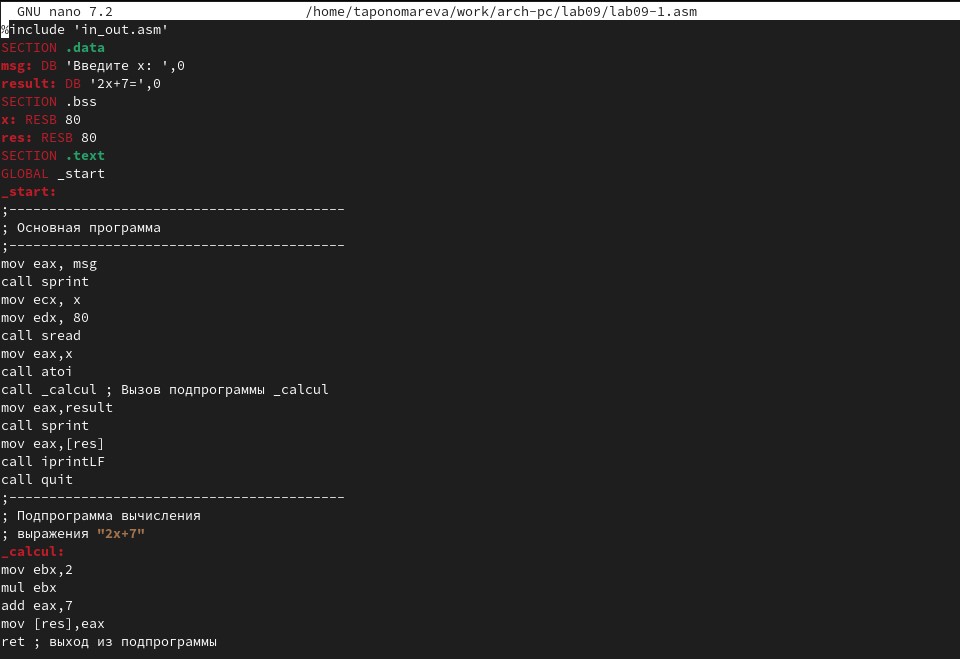


Рис. 2: Окно Midnight Commander. Содержание файла lab09-1.asm

Создаю исполняемый файл и проверяю его работу (рис. 3). Программа работает корректно и при вводе x=10 дает результат 27.

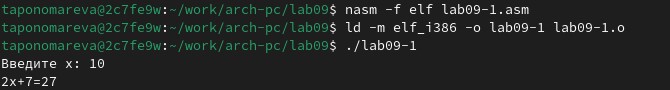


Рис. 3: Терминал. Терминал. Создание исполняемого файла lab09-1. Проверка работы lab09-1

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры.

Текст программы в lab09-1.asm

%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:  
;------------------------------------------  
; Основная программа  
;------------------------------------------  
mov eax, msg  
call sprint  
  
mov ecx, x  
mov edx, 80  
call sread  
  
mov eax,x  
call atoi  
  
call \_calcul ; Вызов подпрограммы \_calcul  
  
mov eax,result  
call sprint  
mov eax,[res]  
call iprintLF  
  
call quit  
;------------------------------------------  
; Подпрограмма вычисления  
; выражения "2x+7"  
\_calcul:  
push eax  
call \_subcalcul  
  
mov ebx, 2  
mul ebx  
add eax, 7  
  
mov [res],eax  
pop eax  
ret ; выход из подпрограммы  
;------------------------------------------  
; Подпрограмма вычисления  
; выражения "3x-1"  
\_subcalcul:  
mov ebx, 3  
mul ebx  
sub eax, 1  
ret

Компилирую исполняемый файл и проверяю его работу (рис. 4). Заметим, что программа работает корректно, выводя 65 при x=10.

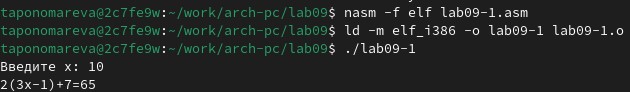


Рис. 4: Терминал. Компиляция исполняемого файла lab09-1. Проверка работы lab09-1

## 3.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm (рис. 5).

Терминал. Создание файла lab09-2.asm

Рис. 5: Терминал. Создание файла lab09-2.asm

Ввожу в файл lab09-2.asm текст программы из листинга 9.2 (рис. 6).

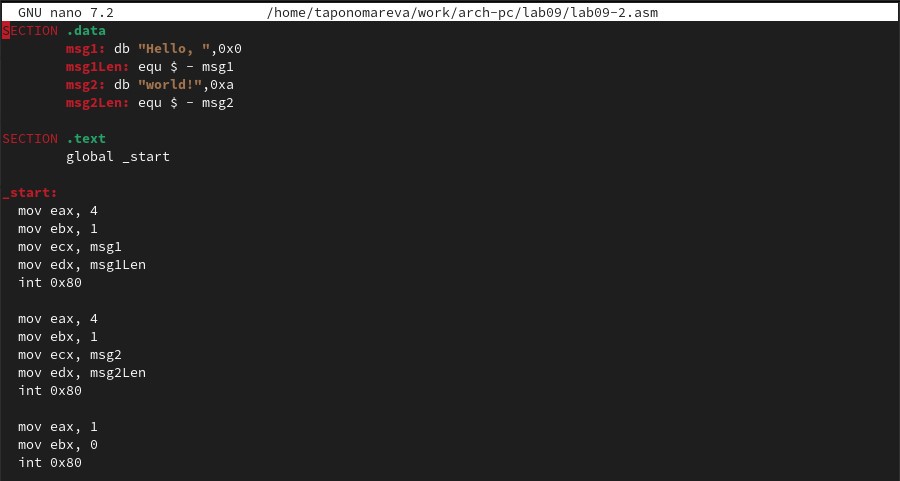


Рис. 6: Окно Midnight Commander. Содержание файла lab09-2.asm

Получаю исполняемый файл. Так как для работы с GDB в исполняемый файл необходимо добавить отладочную информацию,то для этого провожу трансляцию программ с ключом -g (рис. 7).

Терминал. Компиляция исполняемого файла lab09-2

Рис. 7: Терминал. Компиляция исполняемого файла lab09-2

Загружаю исполняемый файл в отладчик gdb (рис. 8).

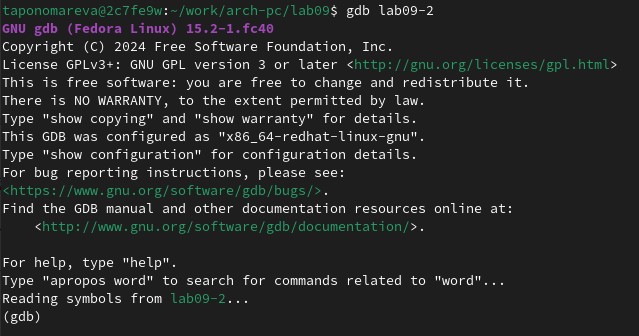


Рис. 8: Терминал. Загрузка исполняемого файла lab09-2 в gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 9). Программа выводит на экран надпись “Hello, world!”, что указывает на правильность работы программы.

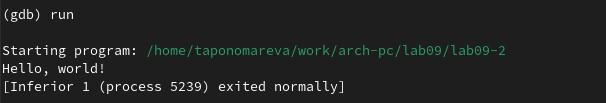


Рис. 9: Терминал. Проверка корректности работы исполняемого файла lab09-2 в gdb

Для более подробного анализа программы установливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 10).



Рис. 10: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Установка брейкпоинта

Затем рассматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. 11).



Рис. 11: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Команда disassemble

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 12).

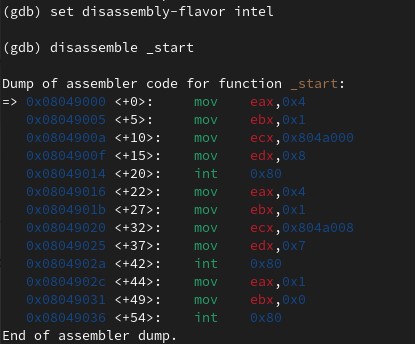


Рис. 12: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Переключение на отображение команд с синтаксисом Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel.

Синтаксис машинных команд в режимах ATT и Intel различается порядком операндов, обозначениями регистров, префиксами констант и обращением к памяти. В синтаксисе Intel первый операнд — назначение, второй — источник (например, mov eax, ebx), тогда как в ATT — наоборот (movl %ebx, %eax). Регистры в ATT начинаются с %, а в Intel указываются без префиксов. Константы в ATT начинаются с $, тогда как в Intel они пишутся без символов.

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

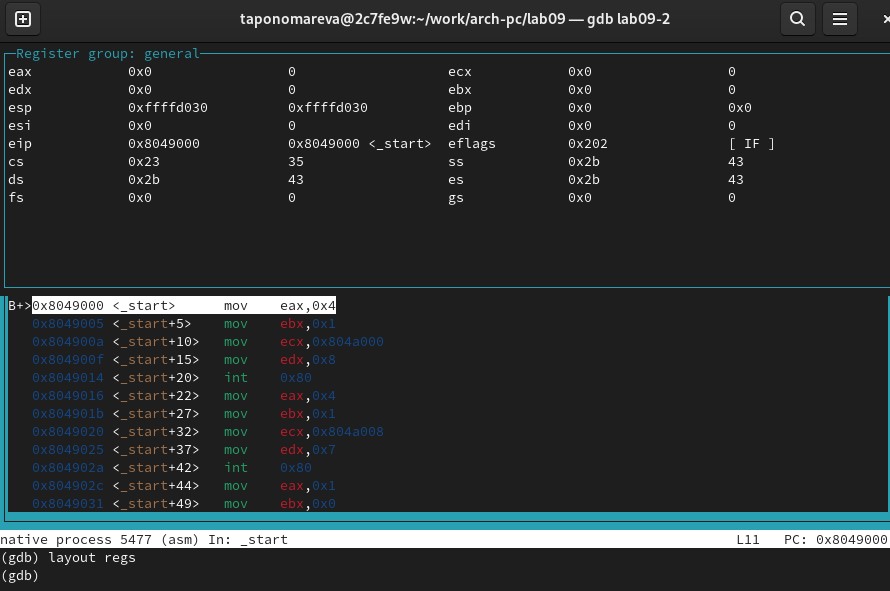


Рис. 13: Терминал. Работа в gdb. Режим псевдографики gdb

## 3.3 Добавление точек останова

Поскольку на предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (кратко i b) (рис. 14).

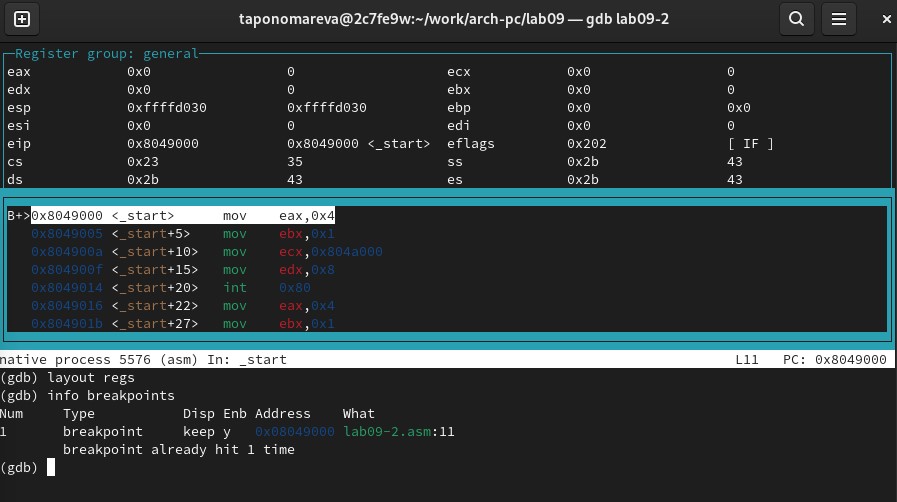


Рис. 14: Терминал. Работа в gdb. Проверка на точку останова

Устанавливаю еще одну точку останова по адресу инструкции. Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова при помощи команды break \* (рис. 15).

Терминал. Работа в gdb. Установка точки останова по адресу инструкции

Рис. 15: Терминал. Работа в gdb. Установка точки останова по адресу инструкции

Рассматриваю информацию о всех установленных точках останова (рис. 16).

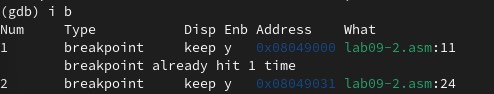


Рис. 16: Терминал. Работа в gdb. Информация о точках останова

## 3.4 Работа с данными программы в GDB

Рассматриваю содержимое регистров с помощью команды info registers (или i r) (рис. 17).

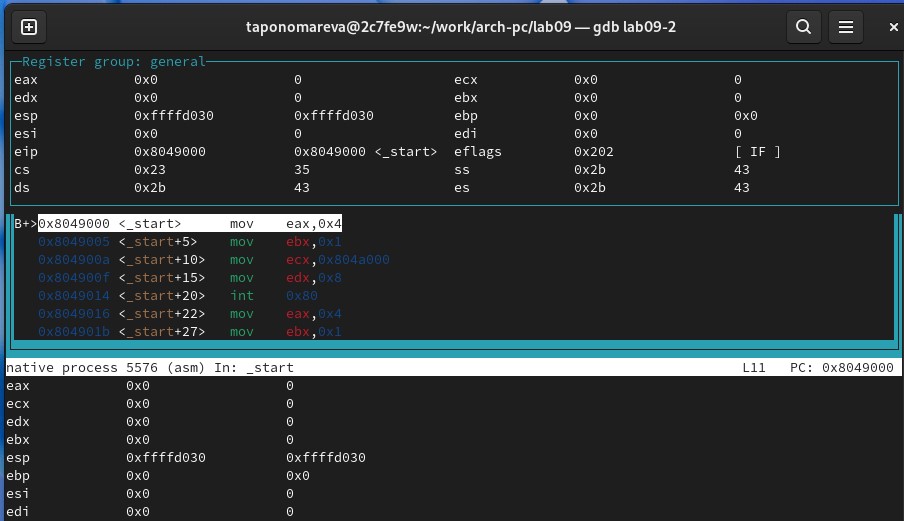


Рис. 17: Терминал. Работа в gdb. Содержимое регистров

Смотрю содержимое переменных msg1 и msg2 по имени и по адресу (рис. 18).

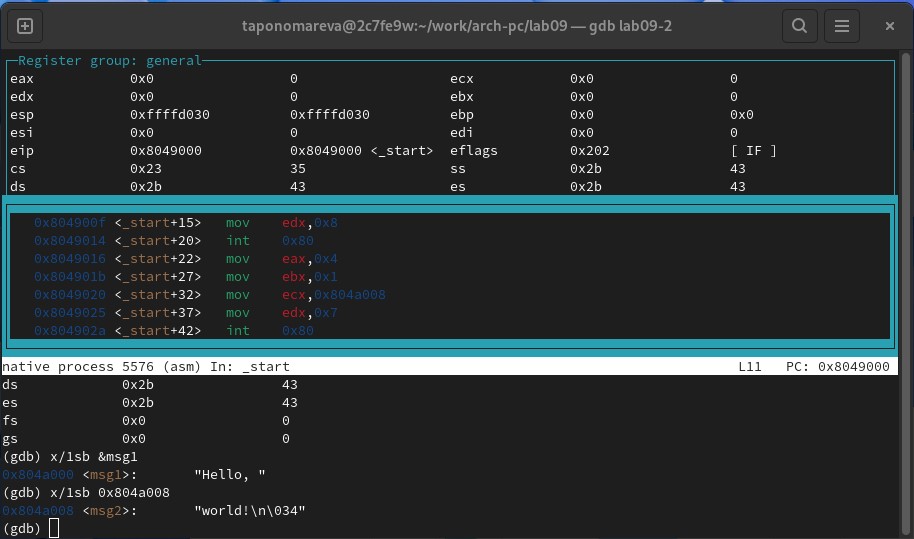


Рис. 18: Терминал. Работа в gdb. Отображение содержимого памяти

Меняю содержимое переменных msg1 и msg2 по имени и по адресу (рис. 19).

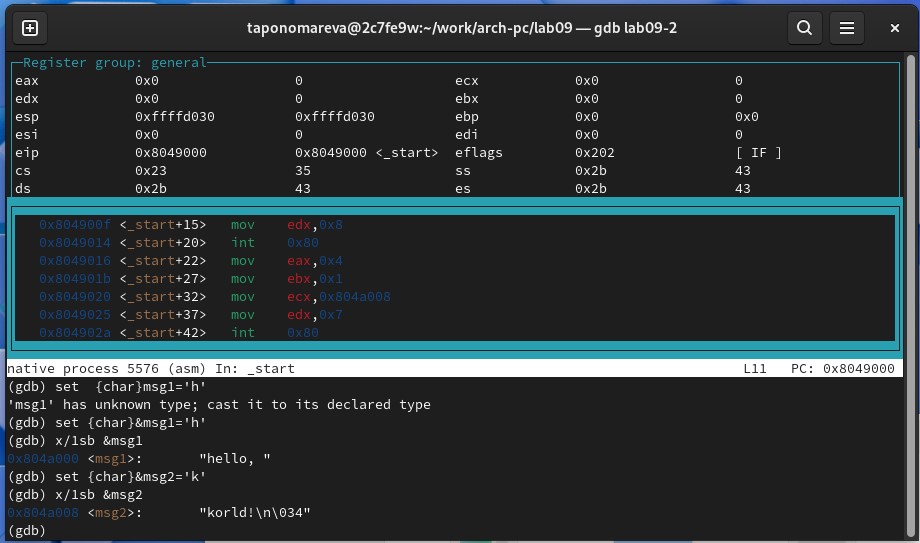


Рис. 19: Терминал. Работа в gdb. Отображение измененного содержимого памяти

Вывожу в различных форматах значение регистра edx (рис. 20).

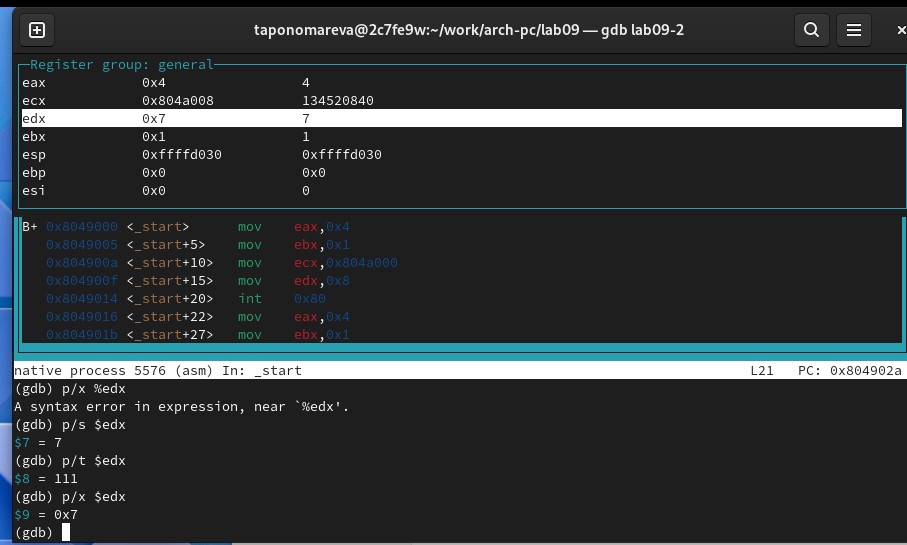


Рис. 20: Терминал. Работа в gdb. Форматы регистра edx

С помощью команды set изменяю значение регистра ebx (рис. 21). В первом случае ‘2’ - это символ с ASCII-кодом 50. Команда p/s $ebx интерпретирует содержимое регистра $ebx как строку символов. Результат равен 50, потому что ASCII-код символа ‘2’ соответствует 50. Во втором случае 2 - это целое число, интерпретируемое как указатель в памяти, который не ведет к осмысленной строке.

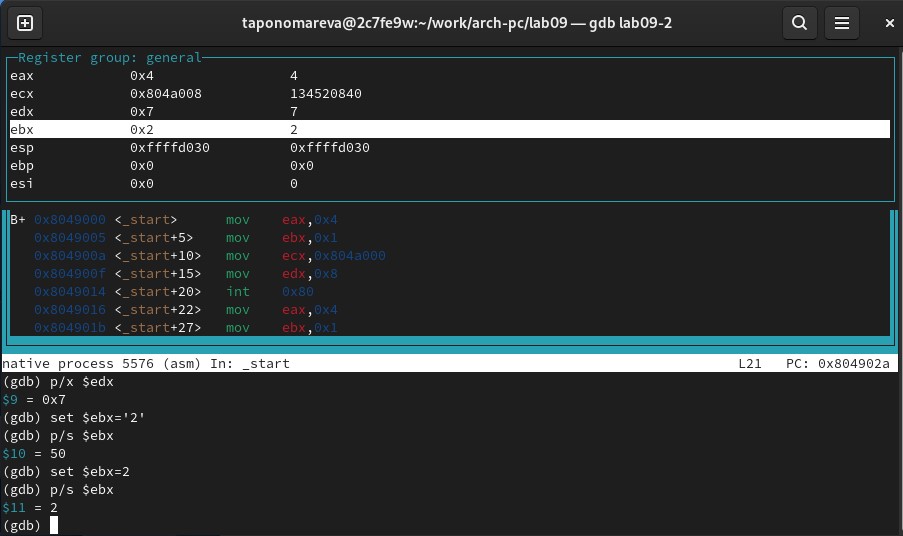


Рис. 21: Терминал. Работа в gdb. Команда set

Завершаю выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выхожу из GDB с помощью команды quit (сокращенно q).

## 3.5 Обработка аргументов командной строки в GDB

Скопирую 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 (рис. 22).

Терминал. Копирование файла lab8-2.asm в файл lab09-3.asm

Рис. 22: Терминал. Копирование файла lab8-2.asm в файл lab09-3.asm

Создаю исполняемый файл (рис. 23).

Терминал. Создание исполняемого файла lab09-3

Рис. 23: Терминал. Создание исполняемого файла lab09-3

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

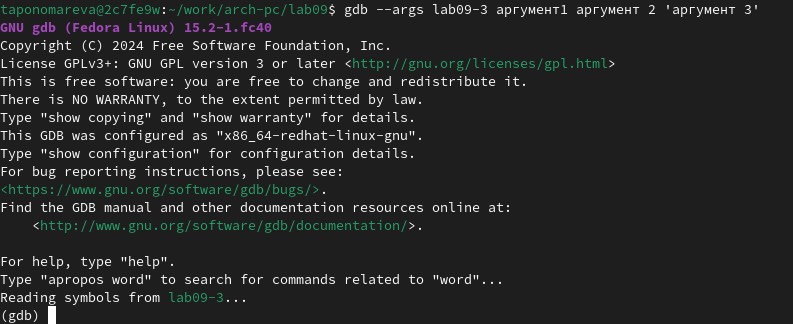


Рис. 24: Терминал. Загрузка исполняемого файла lab09-3 в gdb

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 25). Затем рассматриваю остальные позиции стека. Шаг изменения адреса равен 4, поскольку это соответствует размеру одного элемента типа void\* на 32-битных системах с размером указателя в 4 байта. Это означает, что при обращении к следующему элементу в стеке необходимо сдвинуть указатель на 4 байта.

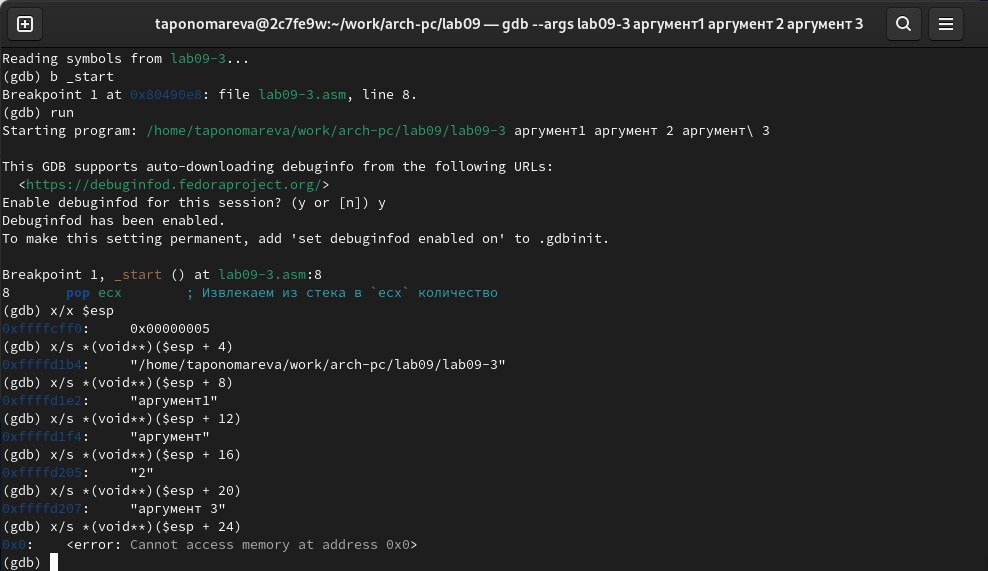


Рис. 25: Терминал. Установка точки останова. Позиции стека

# 4 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8(Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x)=10x-5 как подпрограмму.

Код программы файла lab09-4.asm

%include 'in\_out.asm'  
  
SECTION .data  
msg\_function db "Функция: f(x) = 10x - 5", 0  
msg\_res db "Результат: ", 0  
  
SECTION .text  
GLOBAL \_start  
  
\_start:  
 mov eax, msg\_function  
 call sprintLF  
  
 pop ecx  
 pop edx  
 sub ecx, 1  
 mov esi, 0  
  
next:  
 cmp ecx, 0h  
 jz \_end  
 pop eax  
 call atoi  
  
 call \_solve\_f  
  
 add esi, eax  
 loop next  
  
\_end:   
 mov eax, msg\_res  
 call sprint  
 mov eax, esi  
 call iprintLF  
 call quit  
  
\_solve\_f:  
 mov ebx, 10 ; Загружаем множитель 10 в ebx  
 mul ebx ; Умножаем eax на 10, результат в eax (eax = eax \* 10)  
 sub eax, 5 ; Вычитаем 5 из eax (eax = eax - 5)  
 ret ; Возвращаемся из подпрограммы

Компилирую исполняемый файл и проверяю его работу (рис. 26). Программа работает корректно, т.к. 10*1-5 + 10*2-5+10\*3-5 = 5+15+25 = 45.

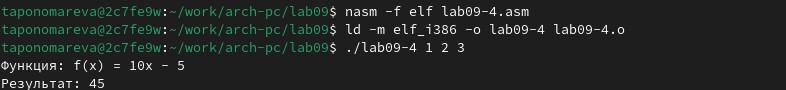


Рис. 26: Терминал. Компиляция исполняемого файла lab09-4. Проверка работы lab09-4

1. Проанализировав программу при помощи отладчика GDB, можно заметить, что ошибка заключается в неправильном использовании регистров при выполнении арифметических операций. В инструкции mul ecx результат умножения сохраняется в регистрах edx и eax, но программа ожидает, что результат будет в ebx, что приводит к потере данных и неправильному результату (рис. 27).

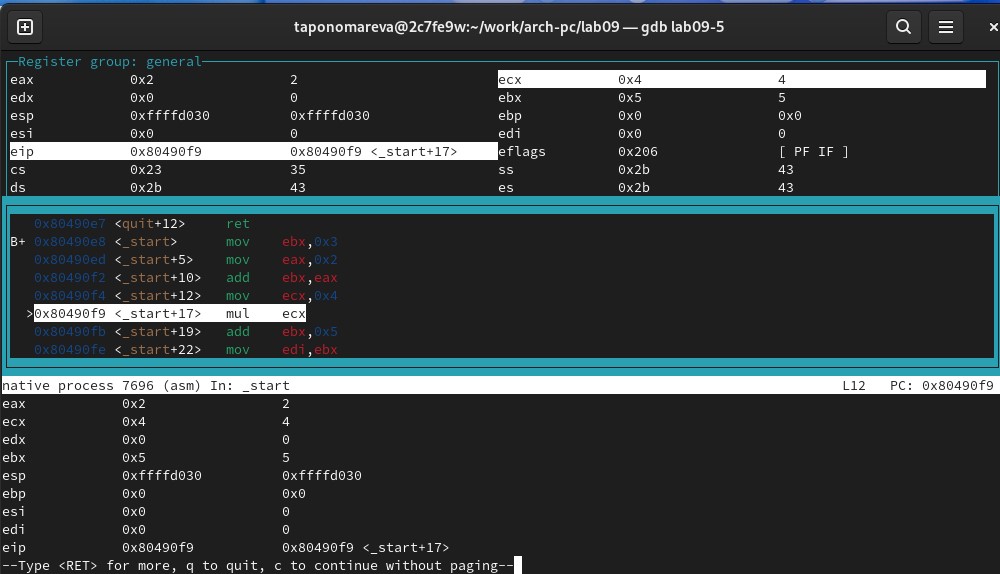


Рис. 27: Терминал. Работа в gdb. Просмотр lab09-5

После исправления кода получаем верный ответ (рис. 28).

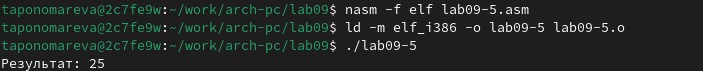


Рис. 28: Терминал. Компиляция исполняемого файла lab09-5. Проверка работы lab09-5

Исправленный код

%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov eax,3   
add eax,2   
mov ecx,4   
mul ecx   
add eax,5   
mov edi,eax   
  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

Загружаю отчет на GitHub.

# 5 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Было произведено знакомство с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. [Курс на ТУИС](https://esystem.rudn.ru/course/view.php?id=112)
2. [Лабораторная работа №9](https://esystem.rudn.ru/pluginfile.php/2089096/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)