Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Гашимова Эсма Эльшан кызы

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 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 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.

# 4 Выполнение лабораторной работы

## 4.1 Релазиация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 (рис. -fig. 1).



Рис. 1: Создание рабочего каталога

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

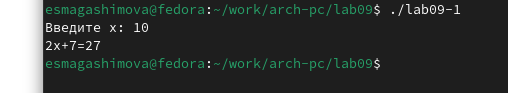


Рис. 2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения f(g(x)) (рис. -fig. 3).

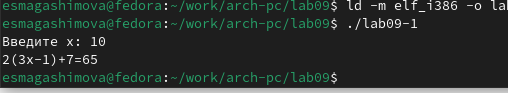


Рис. 3: Изменение программы первого листинга

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

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. -fig. 4).

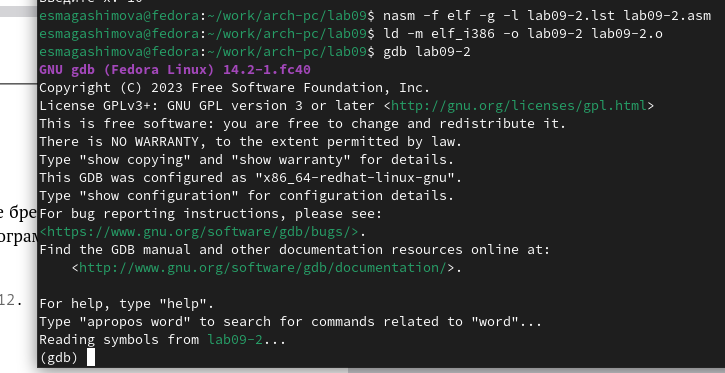


Рис. 4: Запуск программы в отладчике

Запустив программу командой run, я убедился в том, что она работает исправно (рис. -fig. 5).

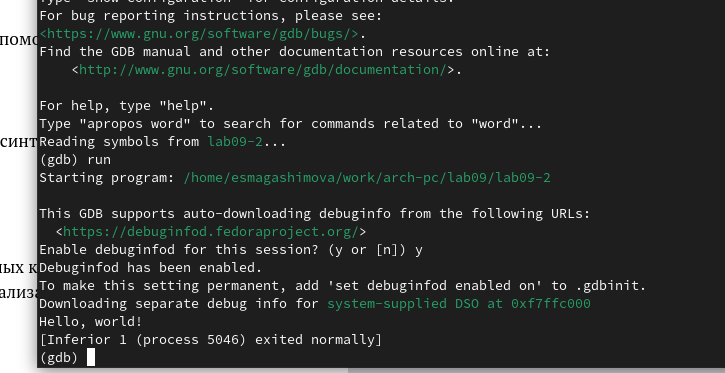


Рис. 5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку \_start и снова запускаю отладку (рис. -fig. 6).

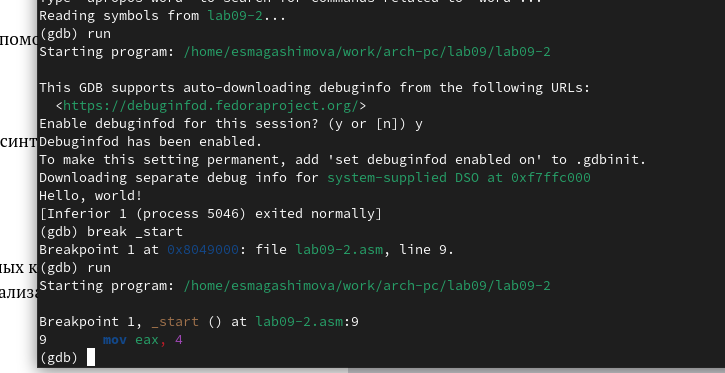


Рис. 6: Запуск отладичка с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *амд топчик* (рис. -fig. 7).

Различия между синтаксисом ATT и Intel заключаются в порядке операндов (ATT - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (ATT - pазмер 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).

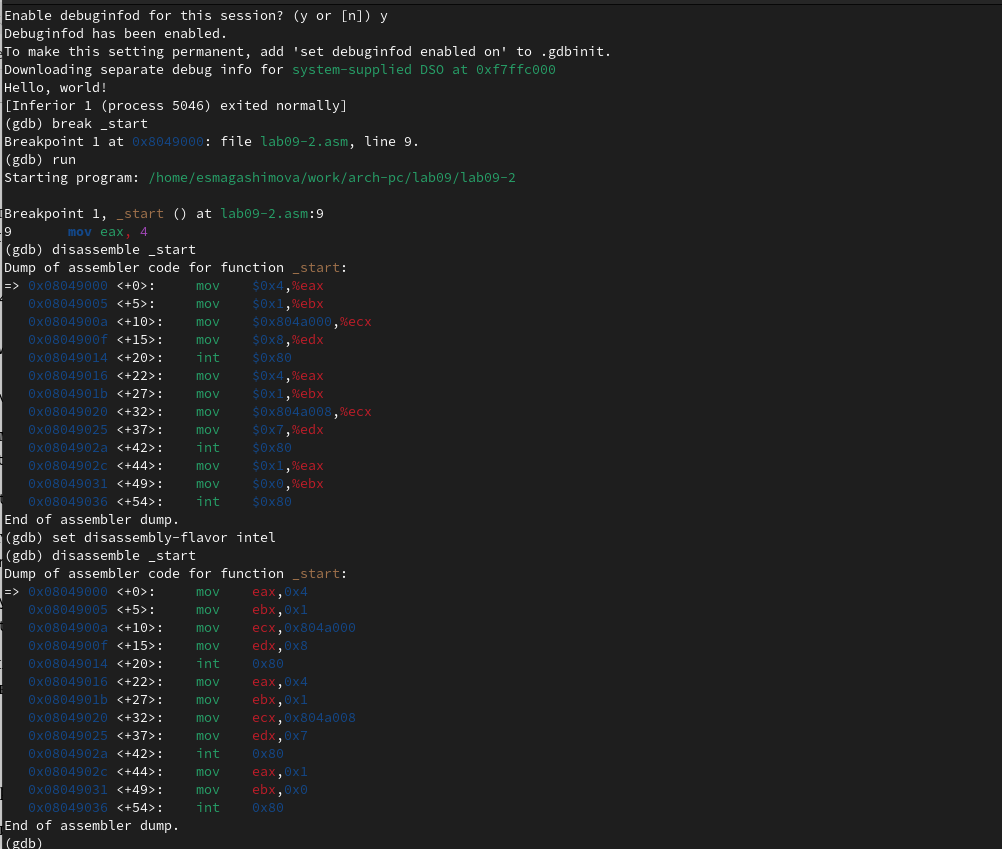


Рис. 7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 8).

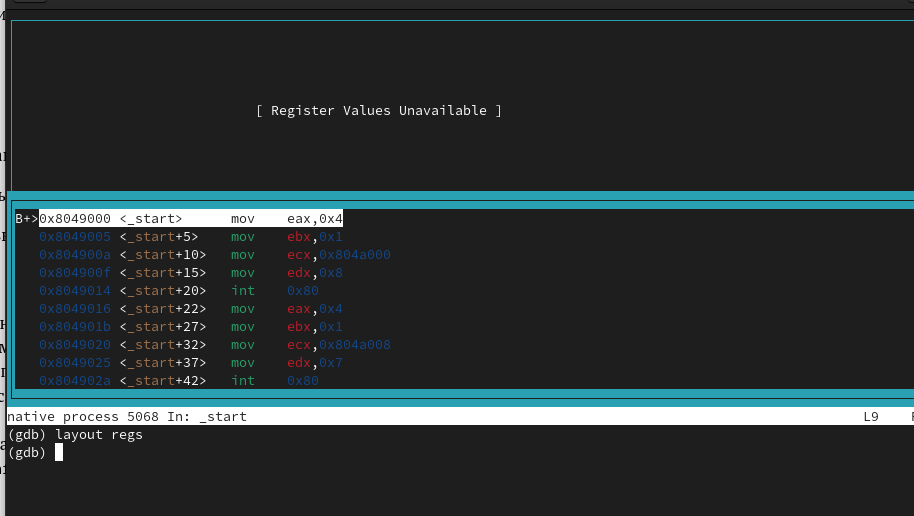


Рис. 8: Режим псевдографики

### 4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. -fig. 9).

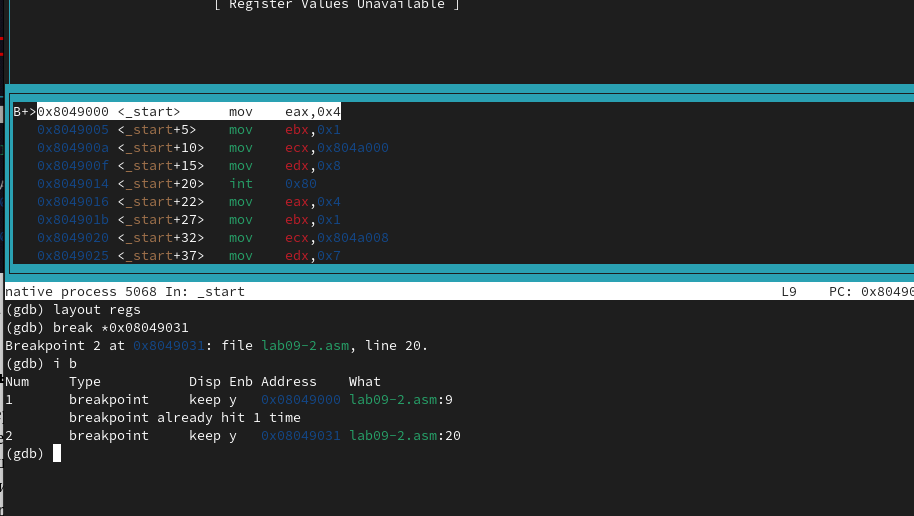


Рис. 9: Список брейкпоинтов

Устаналиваю еще одну точку останова по адресу инструкции (рис. -fig. 10).

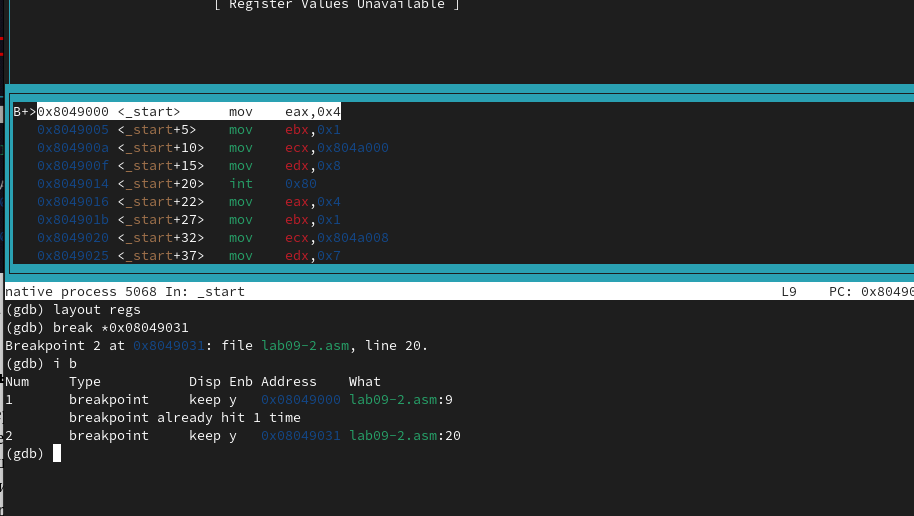


Рис. 10: Добавление второй точки останова

### 4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. -fig. 11).

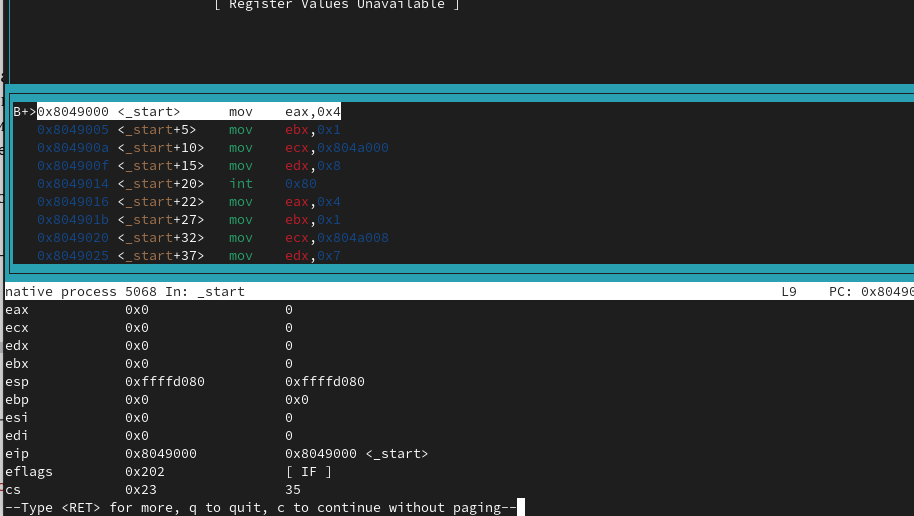


Рис. 11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 12).

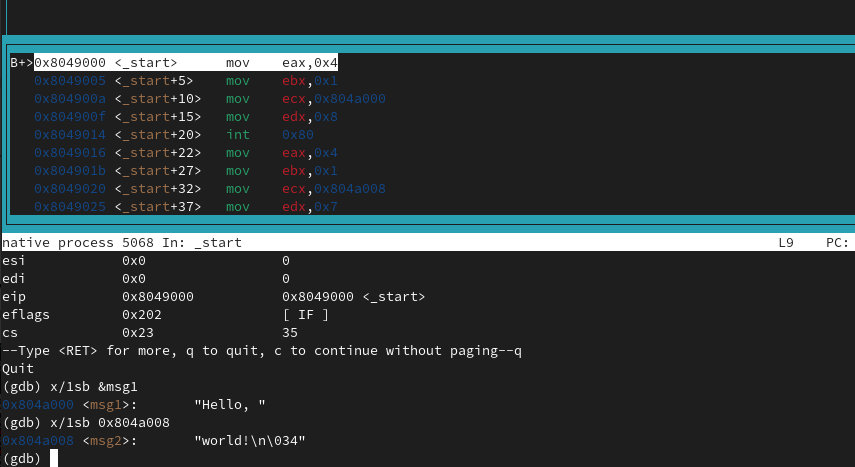


Рис. 12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -fig. 13).

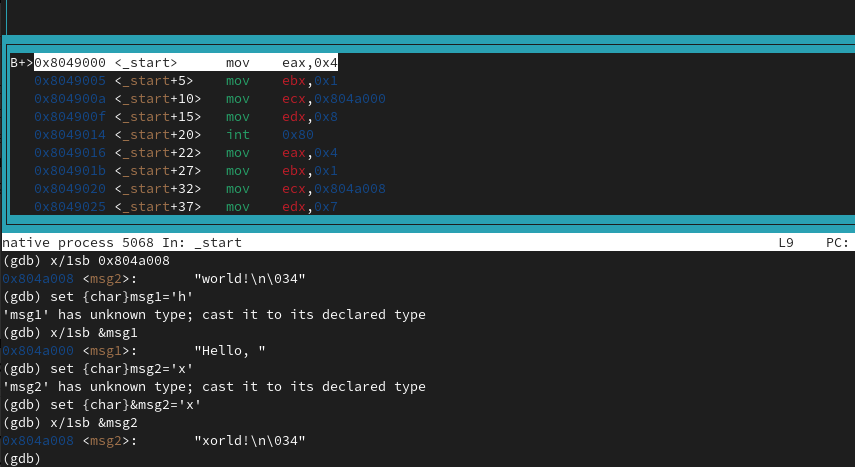


Рис. 13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -fig. 14).

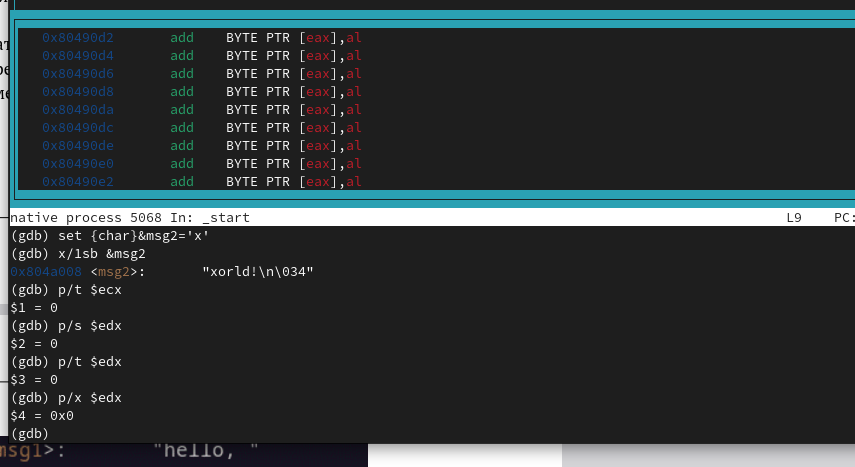


Рис. 14: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -fig. 15).

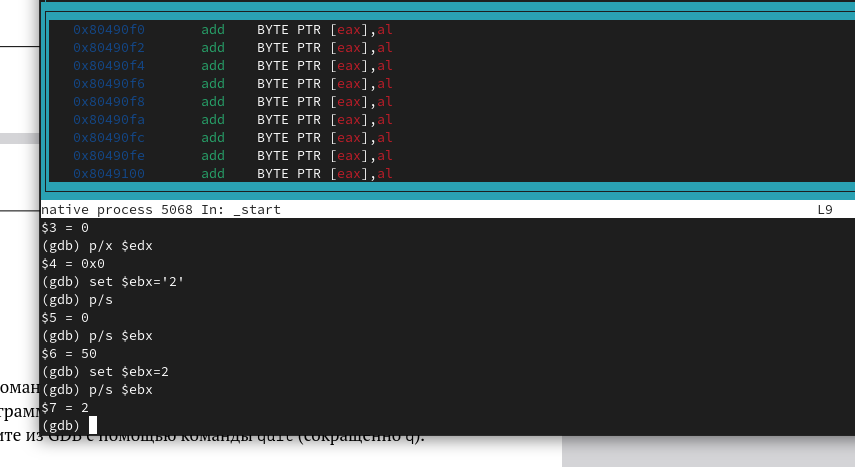


Рис. 15: Примеры использования команды set

### 4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и и создаю исполняемый файл с файлом листинга и отладки (рис. -fig. 16).

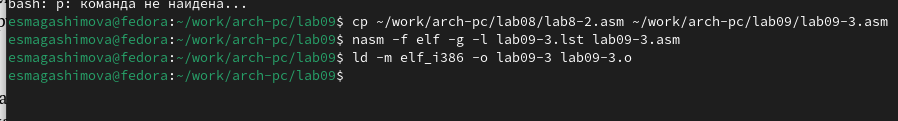
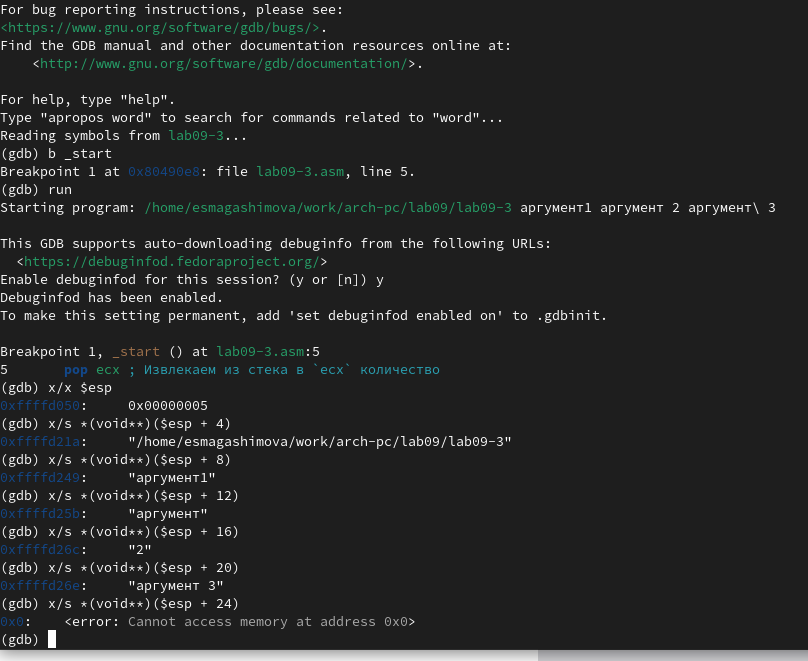


Рис. 16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпопнт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились. (рис. -fig. **¿fig:017?**).

 ## Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 17).



Рис. 17: Измененная программа предыдущей лабораторной работы

Код программы:

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2x+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 f(g(x)) = 2 \* g(x) + 7  
;------------------------------------------  
\_calcul:  
; Вызов подпрограммы \_subcalcul для вычисления g(x)  
mov eax, [x] ; загружаем x  
call \_subcalcul ; eax = g(x)  
  
; Теперь вычисляем f(g(x)) = 2 \* g(x) + 7  
mov ebx, 2  
mul ebx ; eax = 2 \* g(x)  
add eax, 7 ; eax = 2 \* g(x) + 7  
mov [res], eax ; сохраняем результат в переменной res  
ret  
  
;------------------------------------------  
; Подпрограмма вычисления g(x) = 3 \* x - 1  
;------------------------------------------  
\_subcalcul:  
mov eax, [x] ; загружаем x  
mov ebx, 3  
mul ebx ; eax = 3 \* x  
sub eax, 1 ; eax = 3 \* x - 1  
ret  
  
mi, [07.12.2024 20:50]  
%include 'in\_out.asm'  
  
SECTION .data  
 msg\_func db "Функция: f(x) = 2x + 15", 0  
 msg\_result db "Результат: ", 0  
  
SECTION .text  
GLOBAL \_start  
\_start:  
 ; Выводим описание функции  
 mov eax, msg\_func  
 call sprintLF  
  
 ; Подготовка для чтения аргументов  
 pop ecx ; В ecx количество аргументов  
 pop edx ; В edx адрес списка аргументов  
 sub ecx, 1 ; Уменьшаем на 1, так как первый элемент - это количество  
  
 ; Инициализируем сумму в esi  
 mov esi, 0 ; Сумма, которая будет накапливать результат  
  
next:  
 cmp ecx, 0h ; Проверяем, есть ли еще аргументы  
 jz \_end ; Если нет, переходим к выводу результата  
 pop eax ; Получаем следующий аргумент (x\_i)  
 call atoi ; Преобразуем строку в число  
  
 ; Вызов подпрограммы для вычисления f(x)  
 push eax ; Сохраняем аргумент x в стеке  
 call \_calculate\_fx ; Вычисляем f(x)  
 add esi, eax ; Добавляем результат в сумму  
 pop eax ; Восстанавливаем x (необходимо для следующего вызова)  
  
 loop next ; Переходим к следующему аргументу  
  
\_end:  
 ; Выводим результат  
 mov eax, msg\_result  
 call sprint  
  
 mov eax, esi ; Результат суммы  
 call iprintLF ; Выводим результат  
  
 ; Завершаем программу  
 call quit  
  
; Подпрограмма для вычисления f(x) = 2x + 15  
\_calculate\_fx:  
 ; На входе в eax - аргумент x  
 mov ebx, 2 ; Загружаем 2 в ebx  
 mul ebx ; eax = eax \* ebx (2 \* x)  
 add eax, 15 ; eax = eax + 15 (2x + 15)  
 ret ; Возвращаем результат в eax

1. Запускаю программу в режике отладичка и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. -fig. 18).

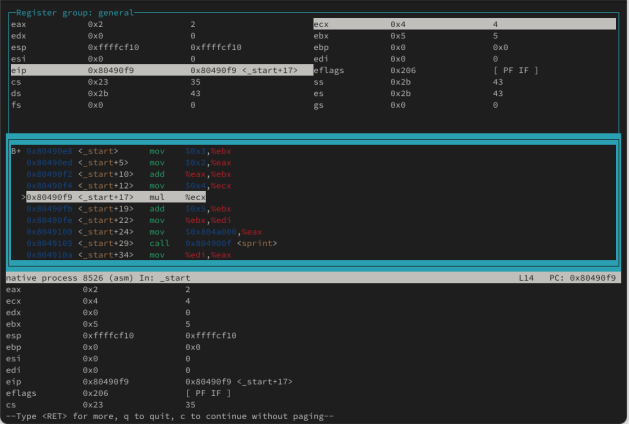


Рис. 18: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 19).

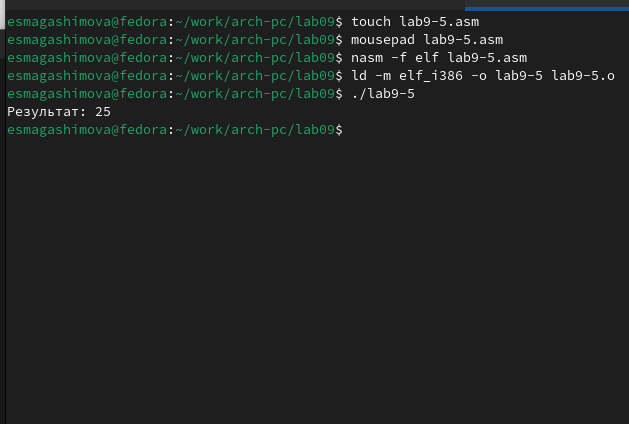


Рис. 19: Проверка корректировок в программме

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 приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при поомщи 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)
3. [Программирование на языке ассемблера NASM Столяров А. В.](https://esystem.rudn.ru/pluginfile.php/2088953/mod_resource/content/2/%D0%A1%D1%82%D0%BE%D0%BB%D1%8F%D1%80%D0%BE%D0%B2%20%D0%90.%20%D0%92.%20-%20%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BD%D0%B0%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5%20%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0%20NASM%20%D0%B4%D0%BB%D1%8F%20%D0%9E%D0%A1%20Unix.pdf)