Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Ермакова Анастасия Алексеевна

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

Цель данной лабораторной 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. Выполнение заданий для самостоятельной 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 Реализация подпрограмм в NASM

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

Рис. 1: Создание каталога и файла

Рис. 1: Создание каталога и файла

Ввожу в файл текст программы из листинга 9.1 (рис. 2).

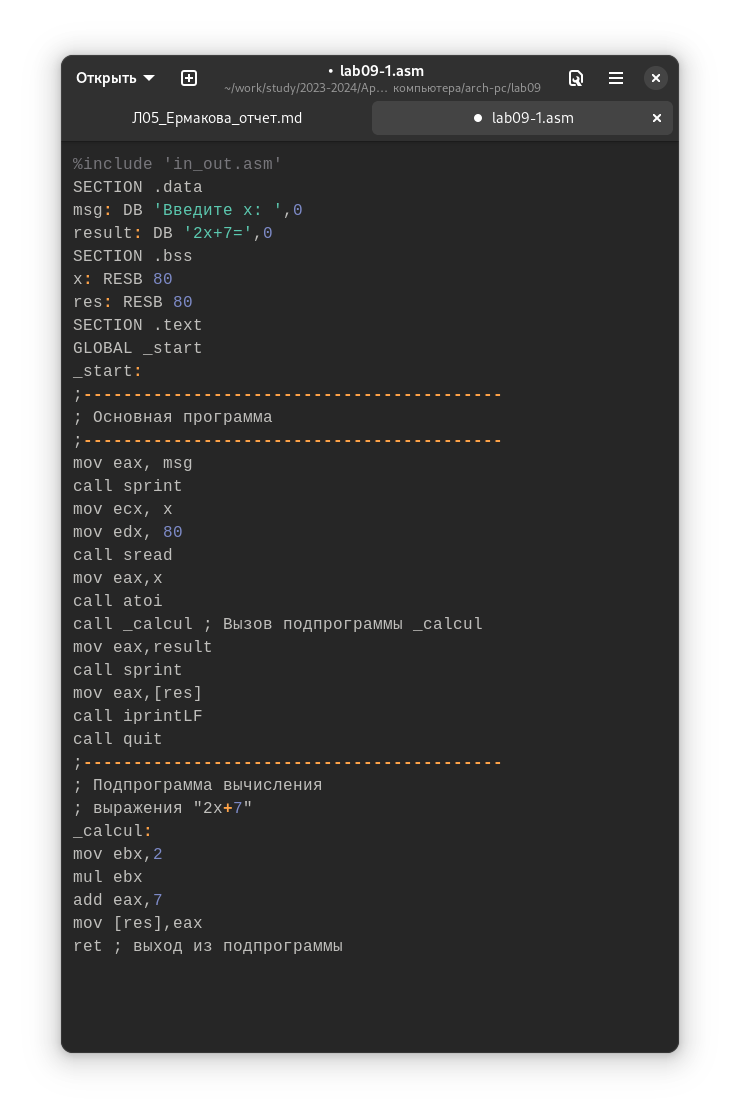


Рис. 2: Текст программы файла lab9-1.asm

Создаю исполняемый файл и запускаю его (рис. 3). Программа выполняет вычисление заданной функции.

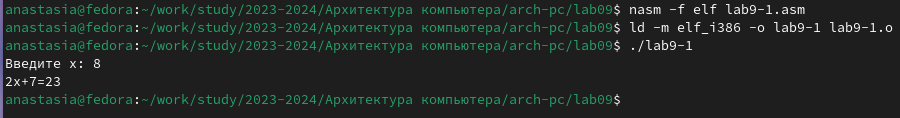


Рис. 3: Результат работы программы

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)) (рис. 4).

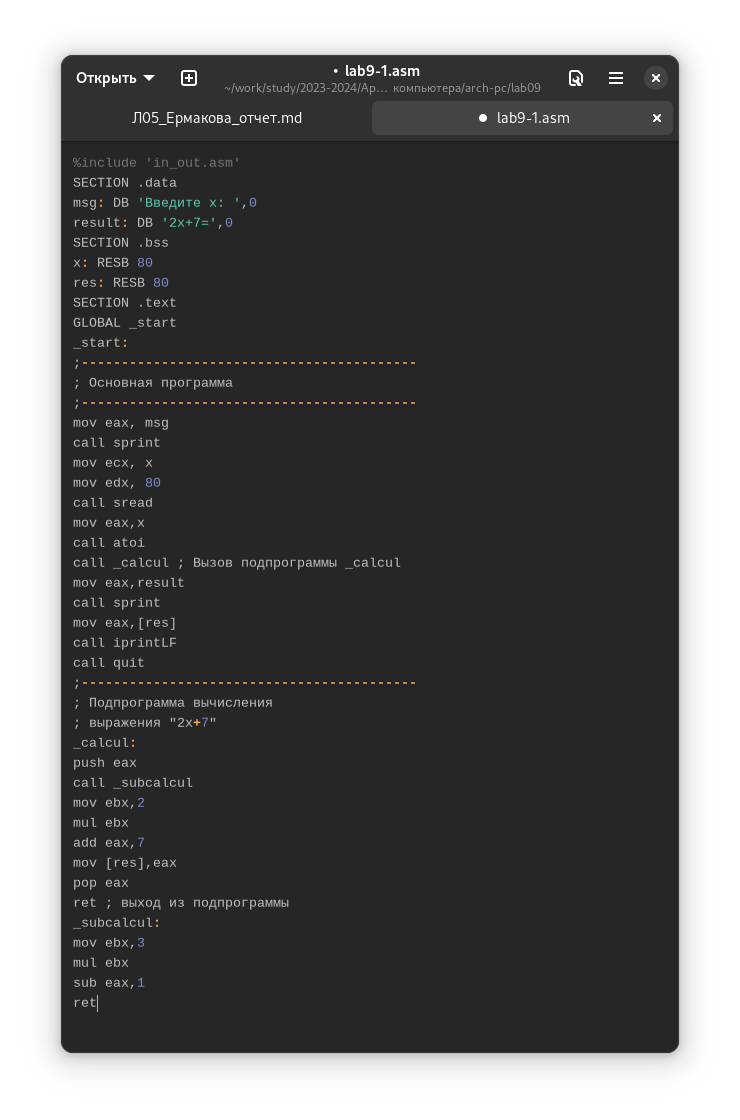


Рис. 4: Измененный текст программы

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:  
;------------------------------------------  
; 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  
\_subcalcul:  
mov ebx,3  
mul ebx  
sub eax,1  
ret

Создаю исполняемый файл и запускаю его (рис. 5). Программа работает исправно.

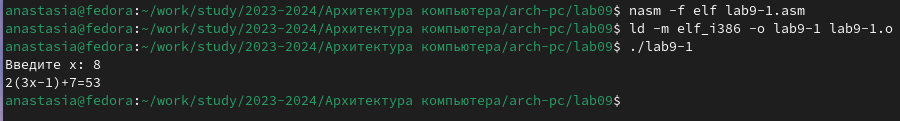


Рис. 5: Результат работы программы

## 4.2 Отладка программ с помощью GDB

Создаю файл lab9-2.asm (рис. 6).

Рис. 6: Создание файла lab9-2.asm

Рис. 6: Создание файла lab9-2.asm

В файл копирую текст программы из листинга 9.2 (рис. 7).

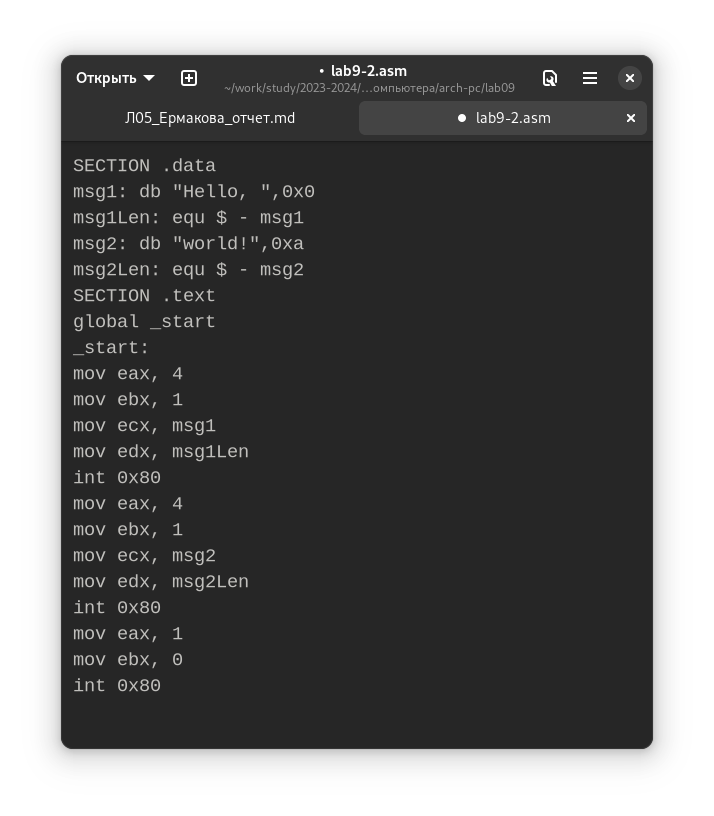


Рис. 7: Текст программы файла lab9-2.asm

Получаю исполняемый файл. Для работы с GDB в исполняемый файл добавляю отладочную информацию. Загружаю файл в отладчик gdb (рис. 8).

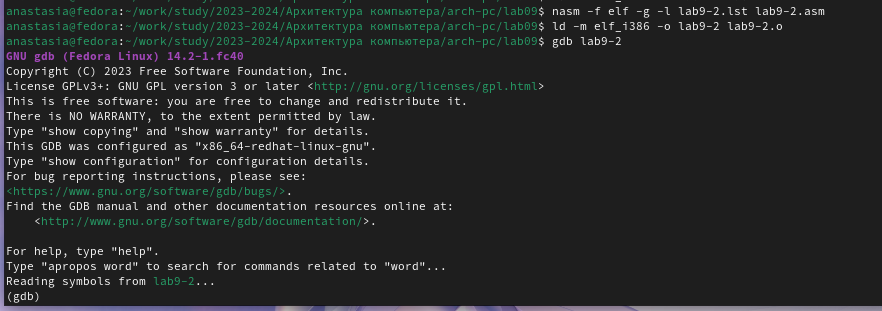


Рис. 8: Запуск программы в отладчике

Проверяю работу программы, запустив ее в отладчике с помощью команды run (рис. 9).

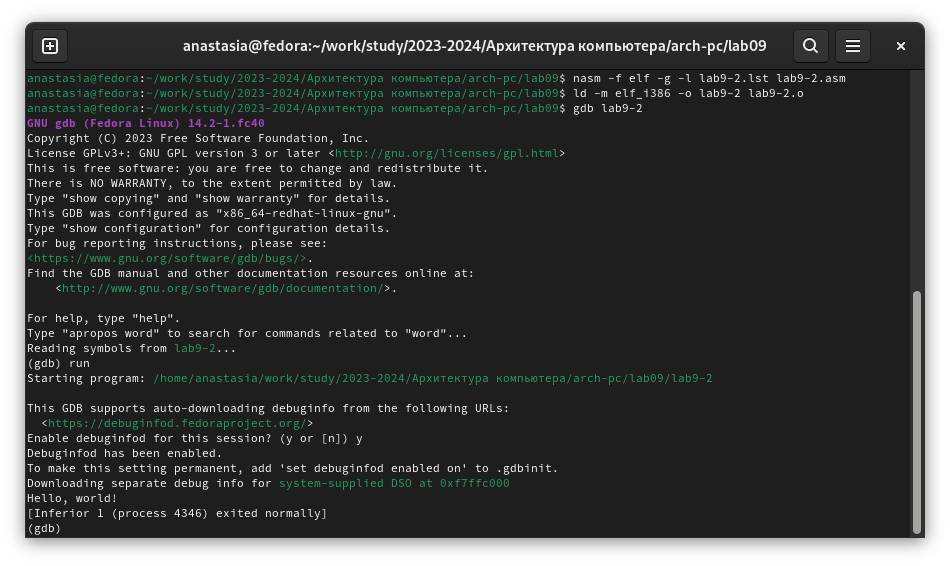


Рис. 9: Проверка работы программы

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

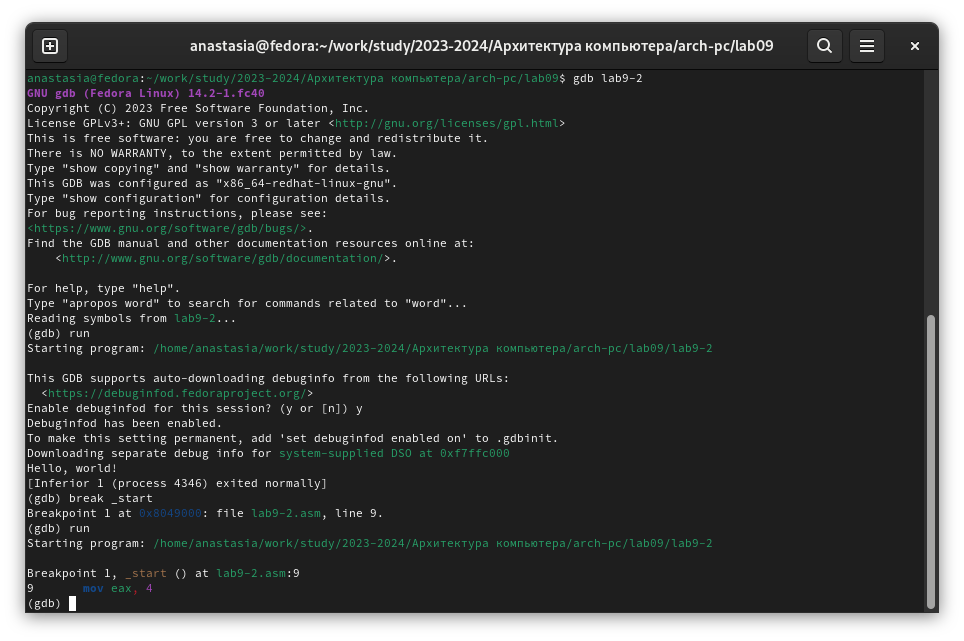


Рис. 10: Установка брейкпоинта

Смотрю дисассемблированный код программы с помощью команды disassemble начиная с метки \_start (рис. 11).

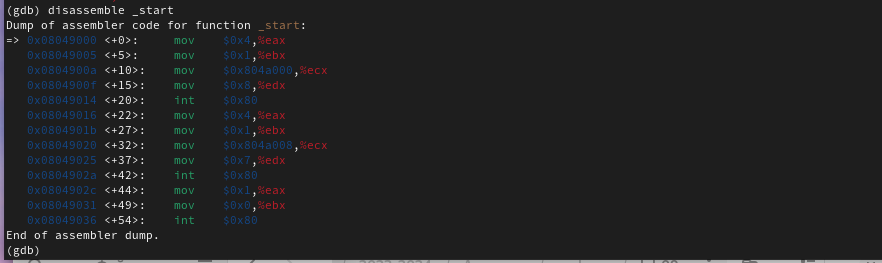


Рис. 11: Дисассемблирование программы

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

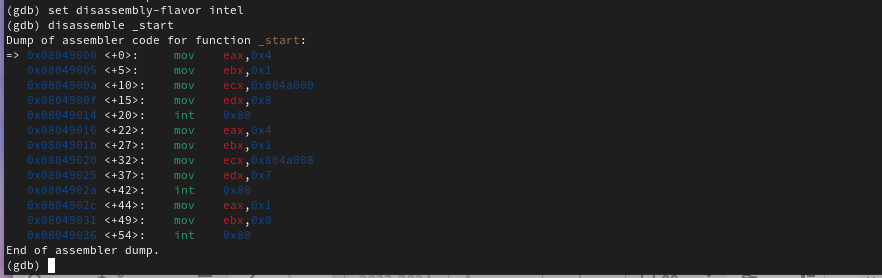


Рис. 12: Переключение отображения команд

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).

### 4.2.1 Добавление точек останова

Включаю режим псевдографики для более удобного анализа программы. С помощью команды i b (info breakpoints) проверяю, что брейкпоинт сохранился. Далее устанавливаю еще одну точку останова по адресу инструкции и просматриваю все установленные точки останова с помощью команды i b (рис. 13).



Рис. 13: Установка брейкпоинтов и их список

### 4.2.2 Работа с данными программы в GDB

Просматриваю содержимое регистров командой i r (info registers) (рис. 14).

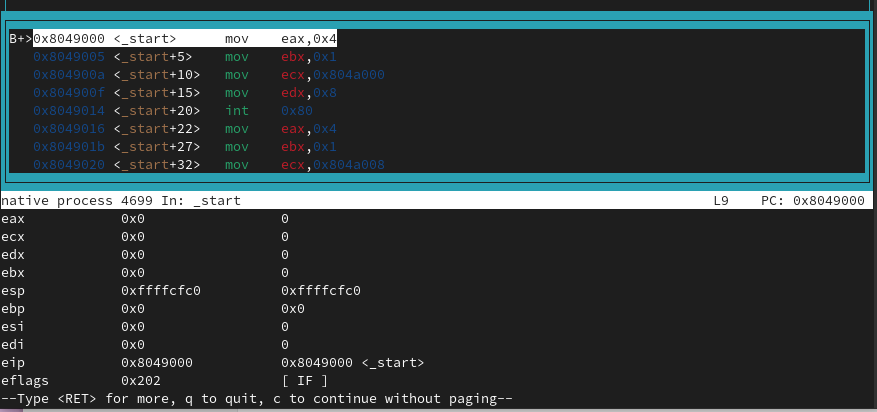


Рис. 14: Просмотр содержимого регистров

Просматриваю содержимое переменных по имени и по адресу (рис. 15).

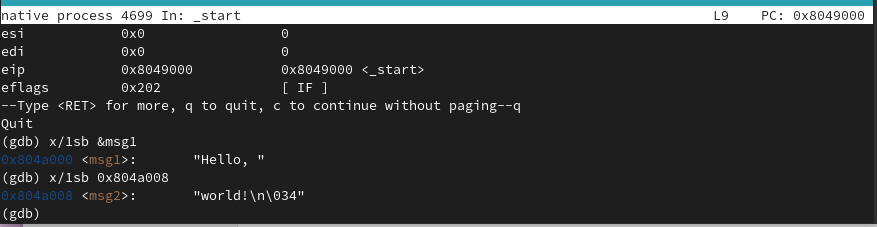


Рис. 15: Просмотр содержимого регистров разными способами

Меняю значения переменных по имени и по адресу (рис. 16).

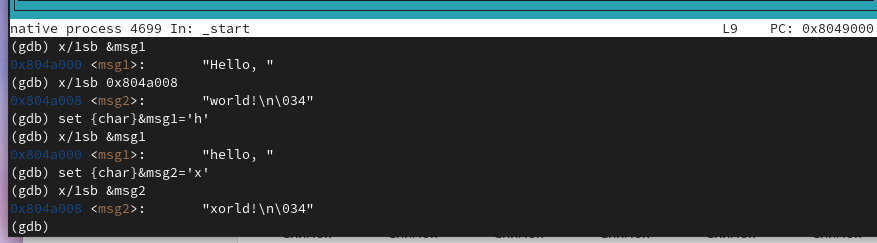


Рис. 16: Изменение значений переменных разными способами

Вывожу в различных форматах значение регистра edx (рис. 17).

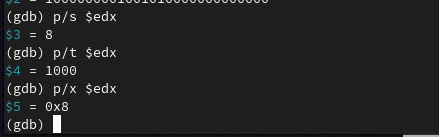


Рис. 17: Вывод значения регистра в разных форматах

С помощью команды set изменяю значение регистра ebx (рис. 18 - 19).

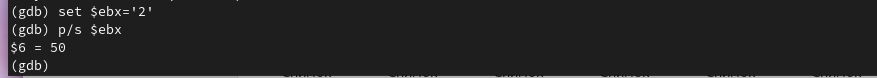


Рис. 18: Использование команды set

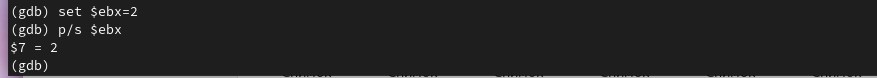


Рис. 19: Использование команды set

Разница вывода команд p/s $ebx заключается в том, что в первом случае переменной присваивается строковое значение ‘2’, поэтому выводится код ASCII этого символа (это 50 в десятичной системе). Во втором случае переменной присваивается числовое значение 2, поэтому команда выводит указатель на строку.

Завершаю выполнение программы с помощью команды c (continue) и выхожу из GDB с помощью команды q (quit) (рис. 20).

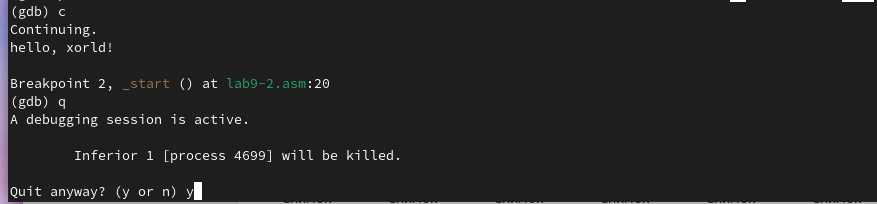


Рис. 20: Завершение программы и выход

### 4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный в ходе выполнения предыдцщей лабораторной работы, в файл с именем lab9-3.asm. Создаю исполняемый файл (рис. 21).

Рис. 21: Копирование файла и создание исполняемого

Рис. 21: Копирование файла и создание исполняемого

Загружаю исполняемый файл в отладчик с аргументами, используя ключ –args (рис. 22). При запуске аргументы загрузились в стек.

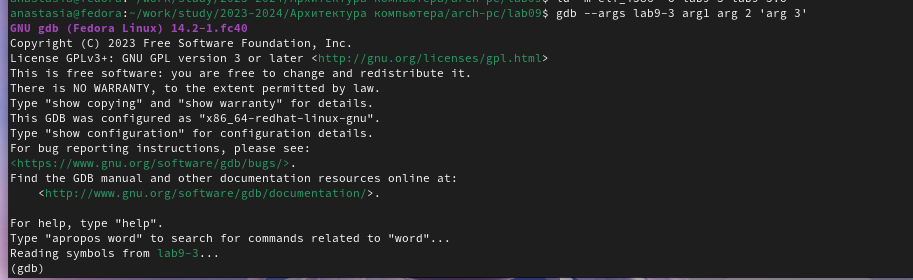


Рис. 22: Загрузка файла в отладчик с аргументами

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 23).

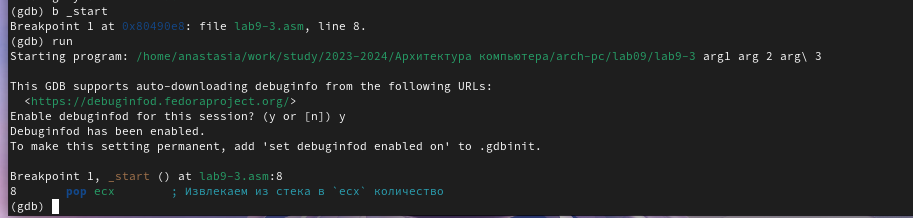


Рис. 23: Установка брейкпоинта и запуск программы

Просматриваю остальные позиции стека (рис. 24).

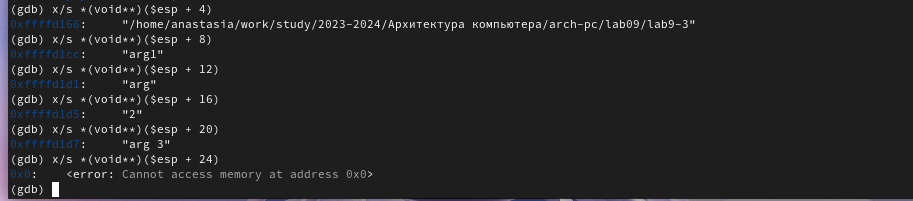


Рис. 24: Просмотр позиций стека

Заметим, что шаг изменения адрема равен 4. Число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились.

## 4.3 Выполнение заданий для самостоятельной работы

1. Преобразую программу из предыдущей лабораторной работы, реализовав вычисление значения функции f(x) как подпрограмму (рис. 25).

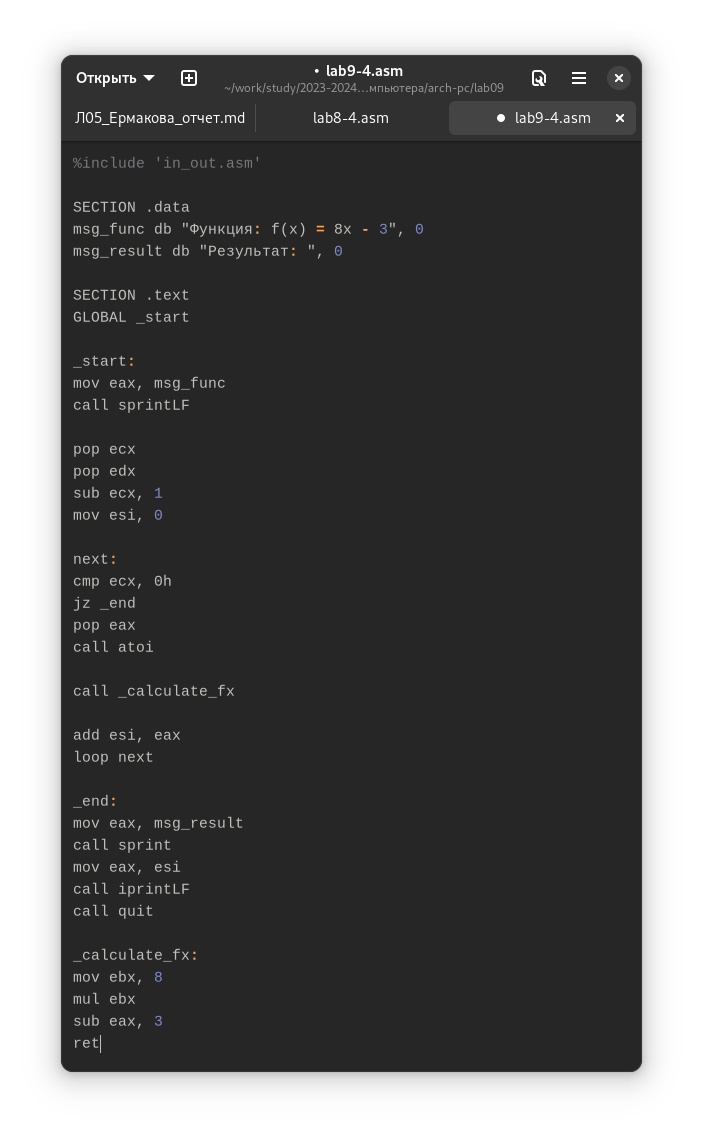


Рис. 25: Измененный текст программы

Код программы:

%include 'in\_out.asm'  
  
SECTION .data  
msg\_func db "Функция: f(x) = 8x - 3", 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, 8  
mul ebx  
sub eax, 3  
ret

1. Создаю файл lab9-5.asm (рис. 26), ввожу текст программы из листинга 9.3 (рис. 27).

Рис. 26: Создание файла lab9-5.asm

Рис. 26: Создание файла lab9-5.asm

Рис. 27: Текст программы файла

Рис. 27: Текст программы файла

Создаю исполняемый файл и открываю с помощью отладчика (рис. 28).

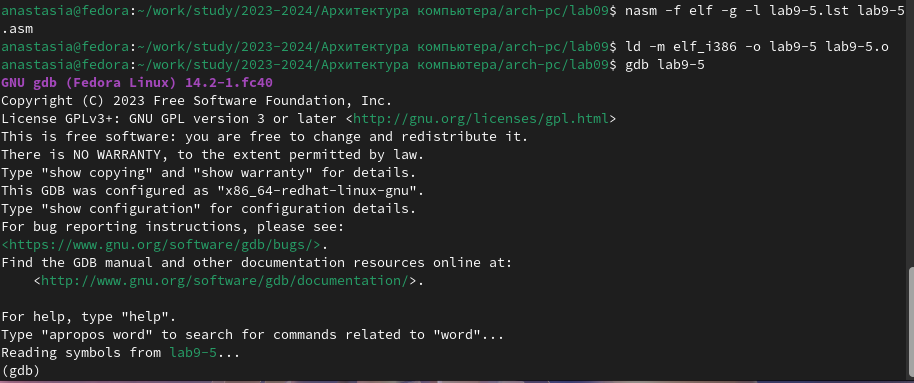


Рис. 28: Работа в отладчике

Запускаю программу с помощью команды r (run), выводится результат 10, он неверный (рис. 29).

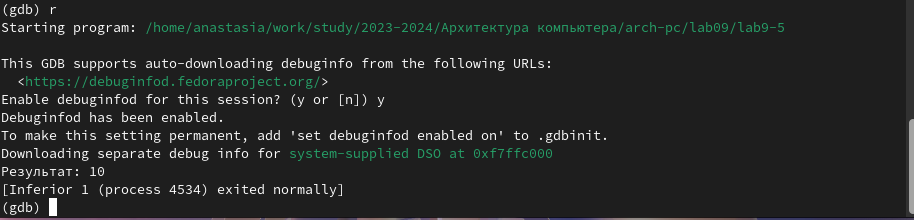


Рис. 29: Результат работы программы

Затем пошагово просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но так же меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому программма неверно подсчитывает функцию (рис. 30).

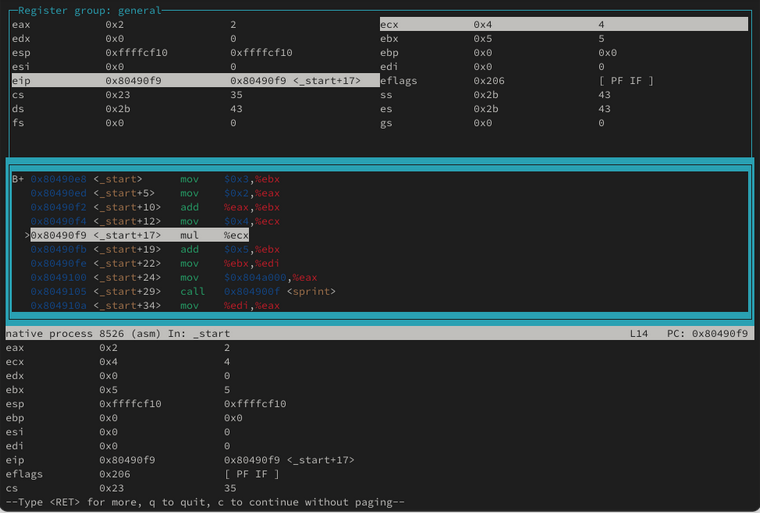


Рис. 30: Поиск ошибки через отладчик

Исправляю найденную ошибку, теперь программа верно считает функцию (рис. 31).

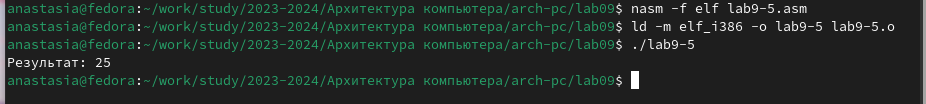


Рис. 31: Результат работы программы

Код исправленной программы:

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

# 5 Выводы

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

# 6 Список литературы

1. [Архитектура ЭВМ](https://esystem.rudn.ru/mod/resource/view.php?id=1030557)