Лабораторная работа №9

. Понятие подпрограммы. Отладчик GDB.

Богданюк Анна Васильевна

Содержание

# Цель работы

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

# Задание

1. Выполнение лабораторной работы
2. Задания для самостоятельной работы

# 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.

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); • использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программаотладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); • Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его). Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

# Выполнение лабораторной работы

1. Выполнение лабораторной работы

Создаю каталог для выполнения лабораторной работы № 9, перейдаю в него и создаю файл lab09-1.asm (рис. [1](#fig:001)).

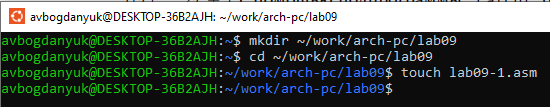


Figure 1: Создание каталога и файла

Ввожу текст программы из листинга в файл lab09-1.asm (рис. [2](#fig:002)).

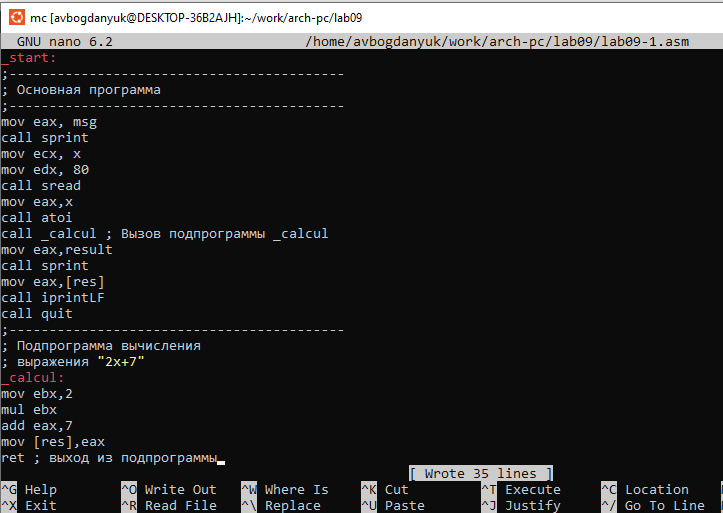


Figure 2: Файл lab09-1.asm

Листинг:

%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  
; выражения "2x+7"  
\_calcul:  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы

Создаю исполнительный файл и запускаю его. Программа работает корректно (рис. [3](#fig:003)).

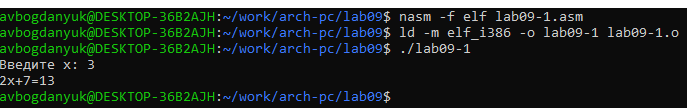


Figure 3: Создание и вывод

Изменяю текст программы так, чтобы сначала считалась 3х-1, затем результ этого уравнение был х в уравнении 2х+7. В первом уравнении х=3, выражение равно 8, значит f(x)=16+7=23. Создаю исполнительный файл и запускаю его. Программа работает корректно (рис. [4](#fig:004)).

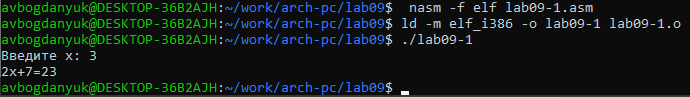


Figure 4: Создание и вывод

Листинг:

%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  
; выражения "2x+7"  
\_calcul:  
call \_subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret  
  
\_subcalcul:  
mov ebx, 3  
mul ebx  
add eax, -1  
mov [res], eax  
ret

Создаю файл lab09-2 и ввожу текст программы из листинга (рис. [5](#fig:005)).

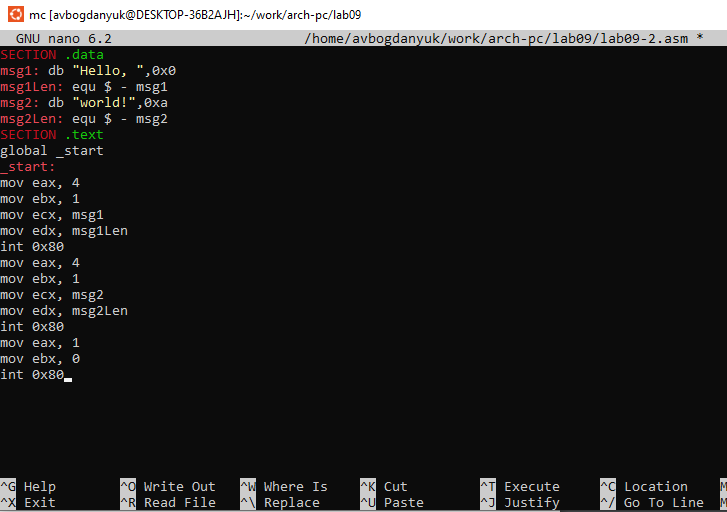


Figure 5: Файл lab09-2

Листинг:

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

Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. [6](#fig:006)).

Figure 6: Создание исполняемого файла

Figure 6: Создание исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. [7](#fig:007)).

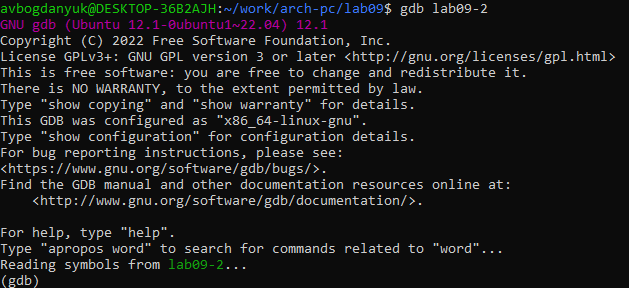


Figure 7: Загрузка в gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [8](#fig:008)).

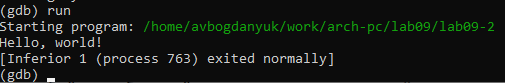


Figure 8: Проверка

Для более подробного анализа программы установливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. [9](#fig:009)).

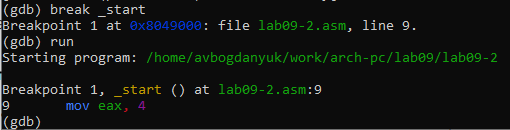


Figure 9: Устанавлию breakpoint и запуск

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. [10](#fig:010)).



Figure 10: Дисассимилированный код

Переключаю на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel’овским синтаксисом можно, введя команду set disassembly-flavor intel (рис. [11](#fig:011)).

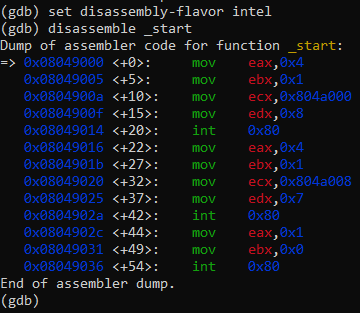


Figure 11: Переключение на отображение команд с Intel’овским синтаксисом

Включаю режим псевдографики для более удобного анализа программы (рис. [12](#fig:012)).

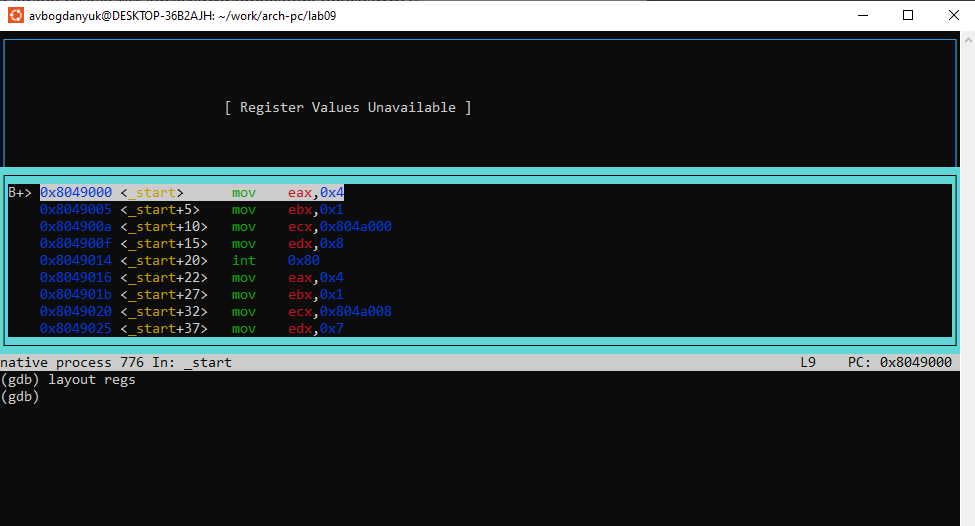


Figure 12: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (рис. [13](#fig:013)).

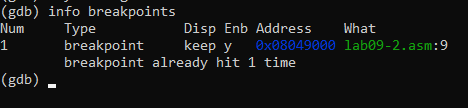


Figure 13: info breakpoints

Установим еще одну точку останова по адресу инструкции. Определяю адрес предпоследней инструкции (mov ebx,0x0) и установливаю точку останова (рис. [14](#fig:014)).

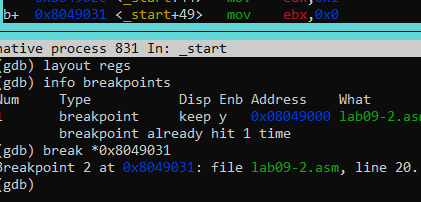


Figure 14: Устанавливаю точку останова

Смотрю информацию о всех установаленных точках останова (рис. [15](#fig:015)).

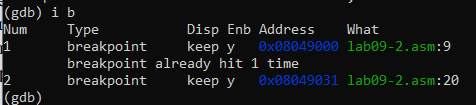


Figure 15: Точки останова

Смотрю содержимое регистров с помощью команды info registers (рис. [16](#fig:016)).

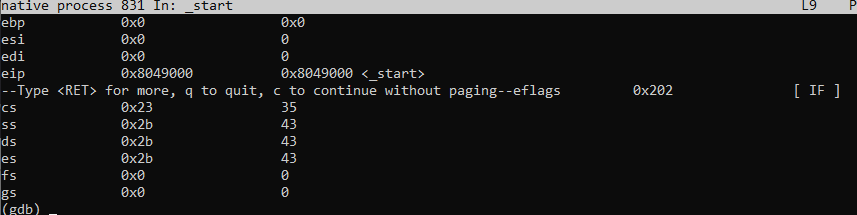


Figure 16: Содержимое регистров

Смотрю значение переменной msg1 по имени. Значение переменной msg2 на рис.19 (рис. [17](#fig:017)).

Figure 17: Значение msg1

Figure 17: Значение msg1

Изменяю первый символ переменной msg1 (рис. [18](#fig:018)).

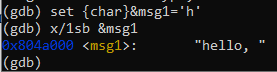


Figure 18: Изменение символа и вывод

Вывожу содержимое переменной msg2. Изменяю первый символ на ‘v’ (рис. [19](#fig:019)).

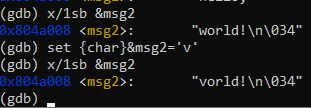


Figure 19: msg2

С помощью команды set изменяю значение регистра ebx (рис. [20](#fig:020)).

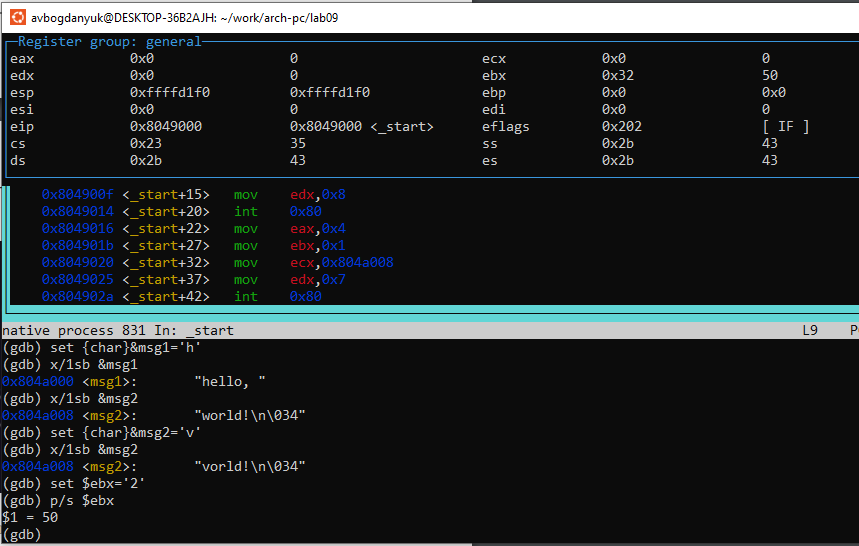


Figure 20: Изменение значения регистра

С помощью команды set изменяю значение регистра ebx (рис. [21](#fig:021)).

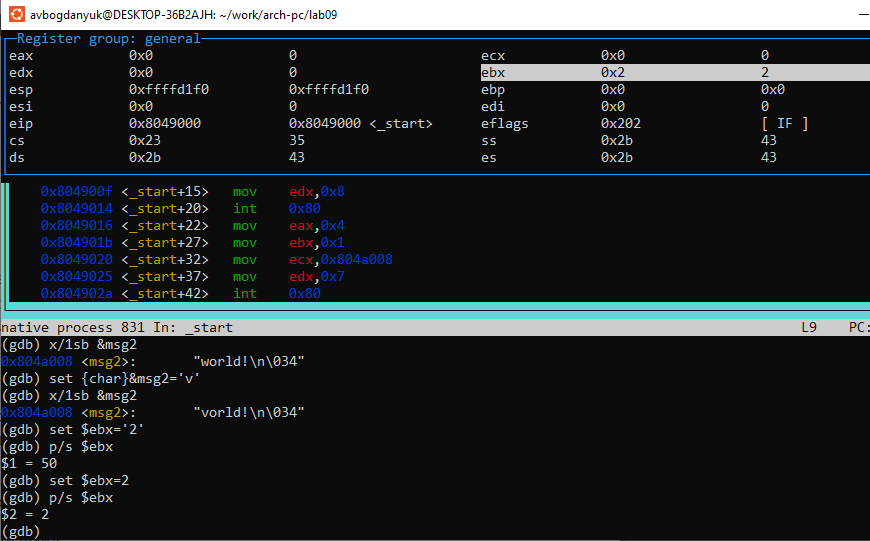


Figure 21: Изменение значения регистра

Завершаю выполнение программы с помощью команды continue (рис. [22](#fig:022)).

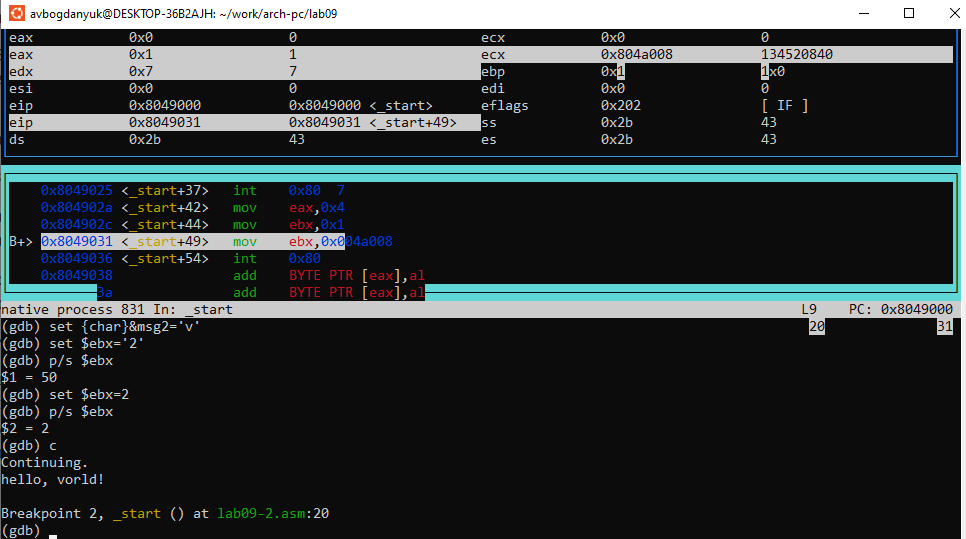


Figure 22: Завершаю выполнение

Выхожу из GDB с помощью команды quit (рис. [23](#fig:023)).

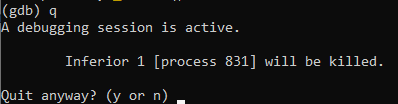


Figure 23: Выхожу из GDB

Копируйте 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. [24](#fig:024)).

Figure 24: Копирую файл

Figure 24: Копирую файл

Создаю исполняемый файл (рис. [25](#fig:025)).

Figure 25: Создание файла

Figure 25: Создание файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загружаю исполняемый файл в отладчик, указав аргументы (рис. [26](#fig:026)).

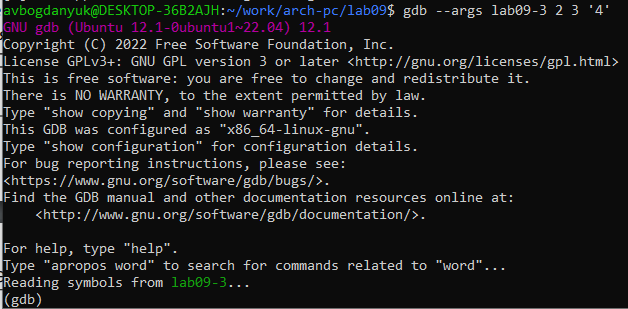


Figure 26: Загружаю файл

Для начала установливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [27](#fig:027)).

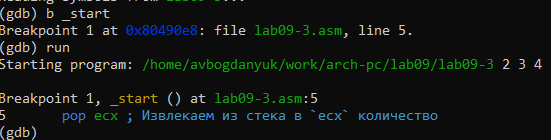


Figure 27: Установка точки останова и запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. [28](#fig:028)).

Figure 28: Аргументов 3 + название = 4

Figure 28: Аргументов 3 + название = 4

Смотрю остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д. (рис. [29](#fig:029)).

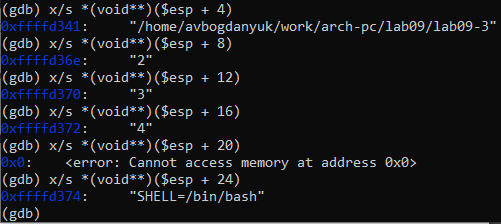


Figure 29: Позиции стека

1. Задания для самостоятельной работы

Создаю файл lab09-4.asm и ввожу в него текст программы листинга (рис. [30](#fig:030)).

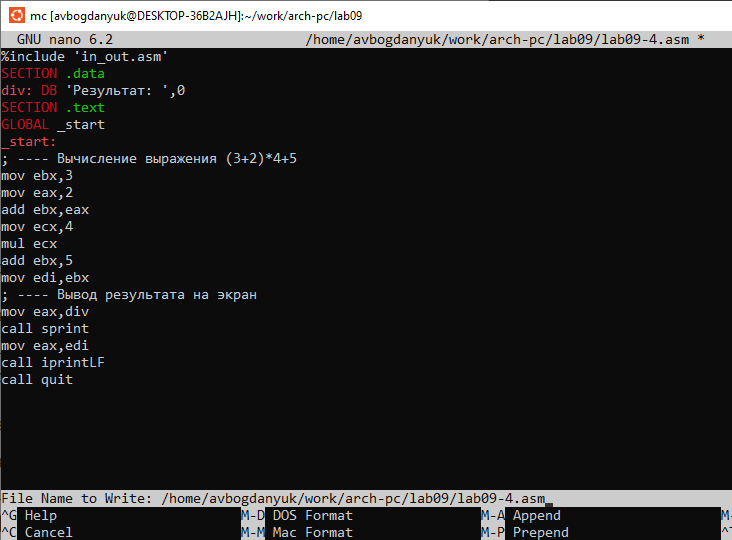


Figure 30: Файл lab09-4.asm

Создаю исполнительный файл и запускаю его. Действительно, программа работает неверно (рис. [31](#fig:039)).

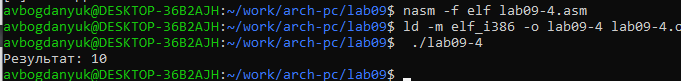


Figure 31: Создание и запуск

Создаю исполнительный файл для работы с GDB (рис. [32](#fig:031)).

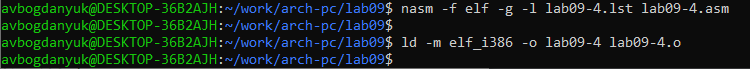


Figure 32: Создание файла

Загружаю исполнительный файл в отладчик gdb (рис. [33](#fig:032)).

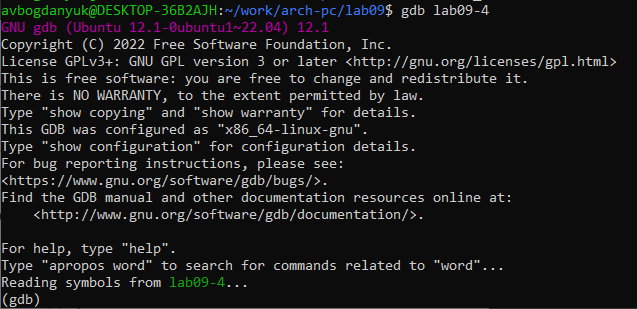


Figure 33: Загружаю файл в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [34](#fig:033)).

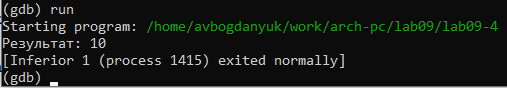


Figure 34: Проверка работы программы

Для более подробного анализа программы установливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. [35](#fig:034)).

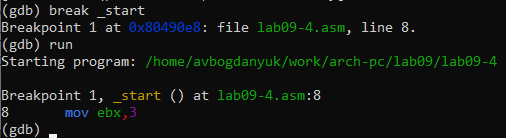


Figure 35: Устанавливаю breakpoint и запускаю

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. [36](#fig:035)).

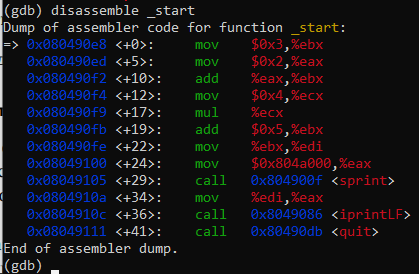


Figure 36: Дисассимилированный код

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel(рис. [37](#fig:036)).

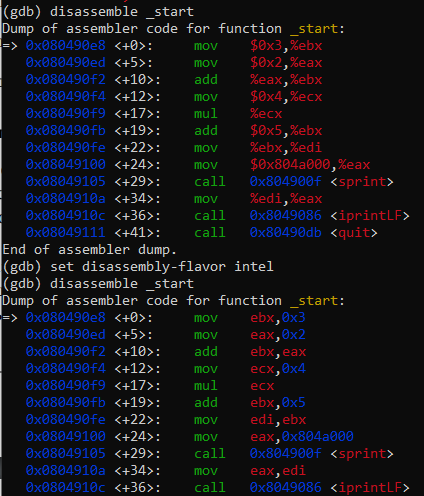


Figure 37: Отображение команд с Intel’овским синтаксисом

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (рис. [38](#fig:037)).

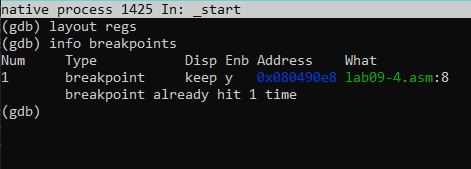


Figure 38: Загружаю файл в отладчик

С помощью stepi буду отслеживать, как изменяются значения регистров (рис. [39](#fig:038)).

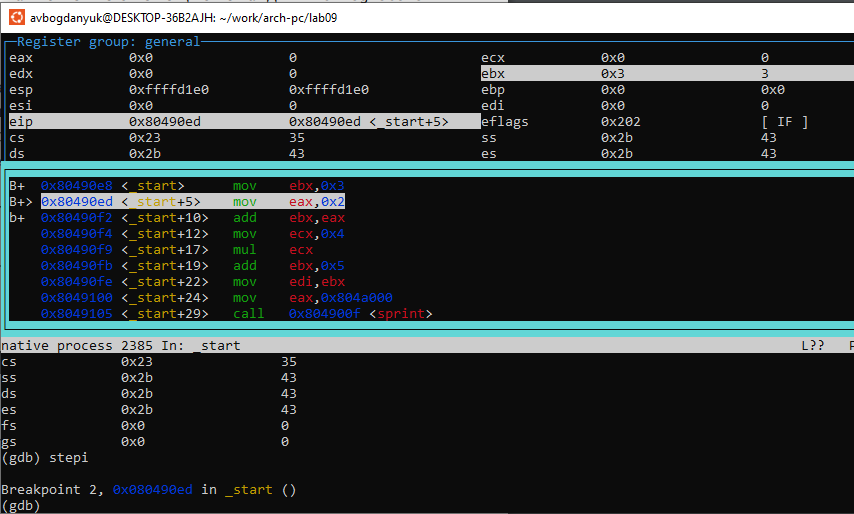


Figure 39: Изменение регистров

На 3 шаге значение ebx = 5 (рис. [40](#fig:040)).

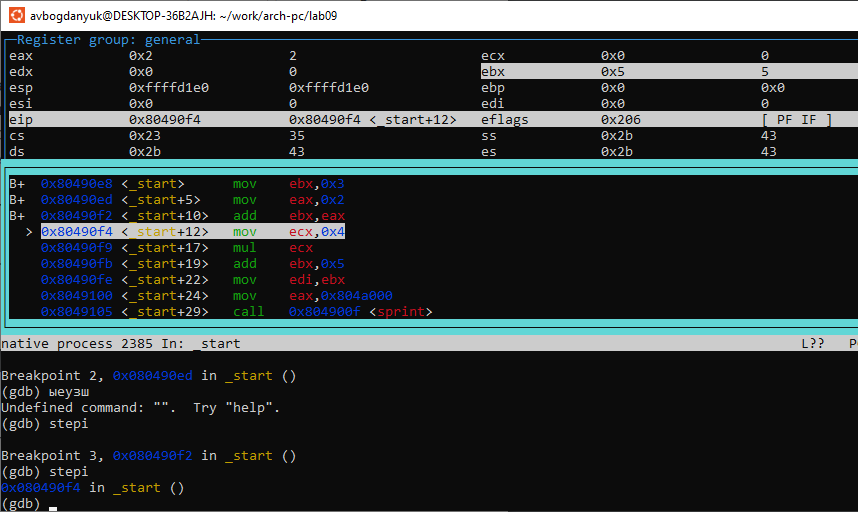


Figure 40: Значение регистра ebx

На шаге 5 произошла ошибка. ecx было умножено не на ebx, а на eax. Значение ebx а этом шаге должно было быть равно 20, но оно равно 2\*4=8 (рис. [41](#fig:041)).

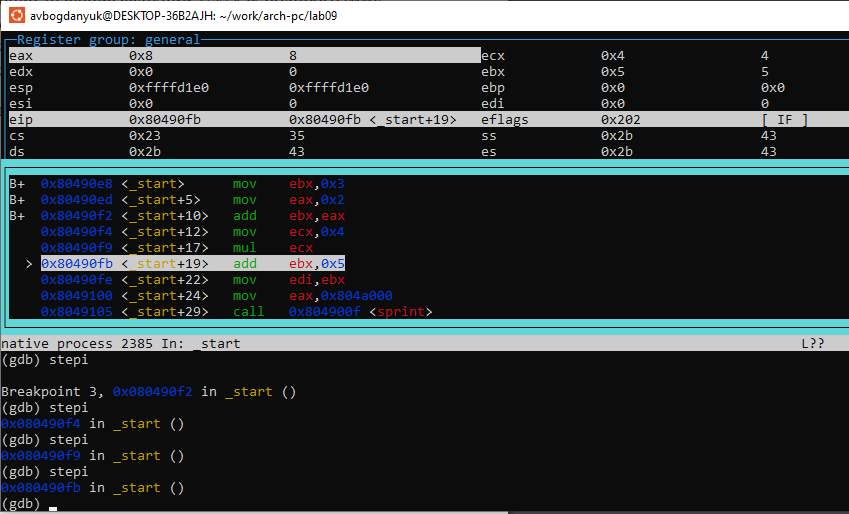


Figure 41: Ошибка

На этом шаге виден результат вычисления программы. ebx = ebx + 5, но, так как ebx = 5 = 3+2, а не ebx=20+5, конечный результат равен не 25, а 10 (рис. [42](#fig:042)).

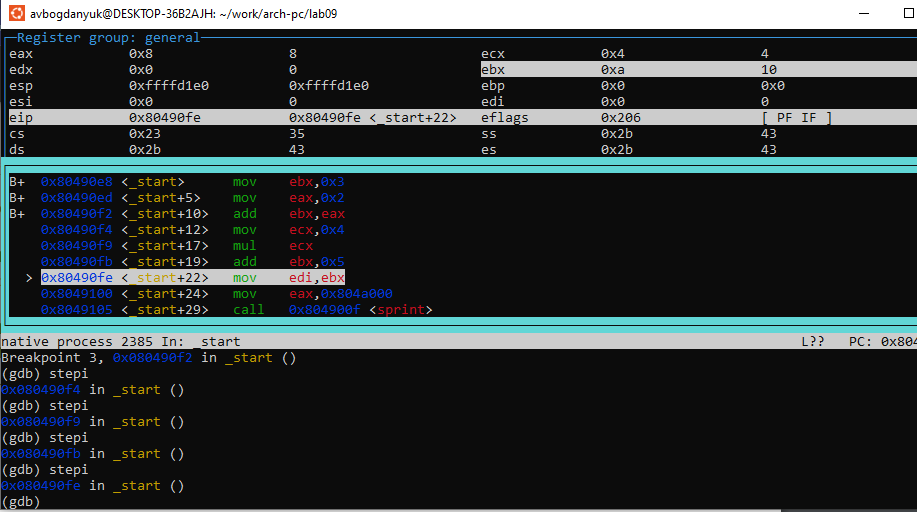


Figure 42: Результат

Изменяю текст программы так, чтобы вычисления были верными (рис. [43](#fig:043)).

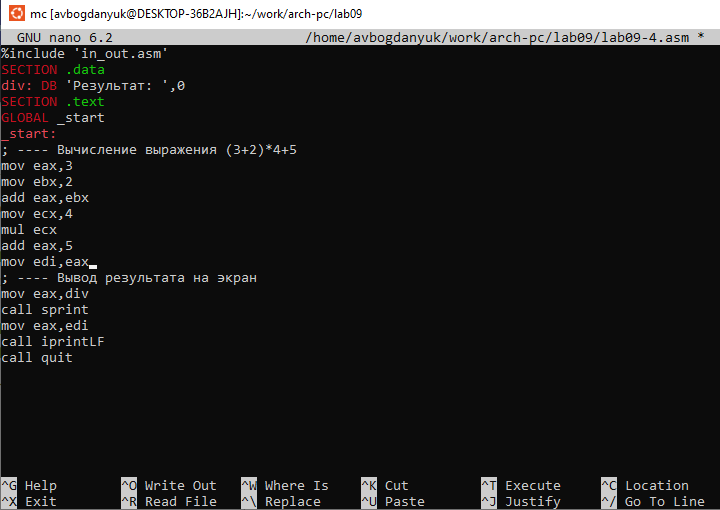


Figure 43: Файл lab09-4.asm

Создаю исполнительный файл и запускаю его. Программа работает корректно (рис. [44](#fig:044)).

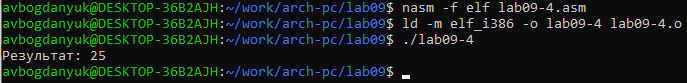


Figure 44: Создание и запуск

Листинг:

%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov eax,3  
mov ebx,2  
add eax,ebx  
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

# Выводы

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

# Список литературы