Отчёт по лабораторной работе №9

Архитектура компьютера

Ирина Васильевна Панявкина

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 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.

# 4 Выполнение лабораторной работы

Реализация подпрограмм в NASM

Создаю каталог для программ лабораторной работы №9, а также файл lab9-1.asm и копирую в текущий каталог файл in\_out.asm с помощью утилиты cp, т.к. он будет использоваться во время выполнения самостоятельной работы (рис. 1).

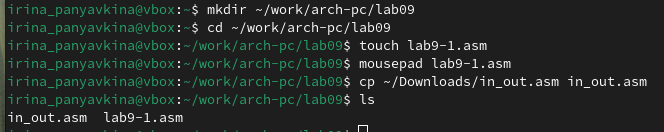


Рис. 1: Создание каталога и файла для программы, создание копии внешнего файла

Открываю созданный файл и вставляю в него скопированную программу из листинга, компилирую и запускаю его, программа выполняет вычисление функции (рис. 2).

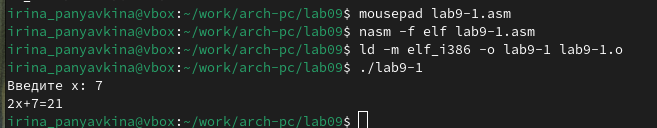


Рис. 2: Запуск программы из листинга

Изменяю программу, добавив в неё подпрограмму, таким образом, чтобы она вычисляла значение функции для выражения f(g(x)) (рис. 3).

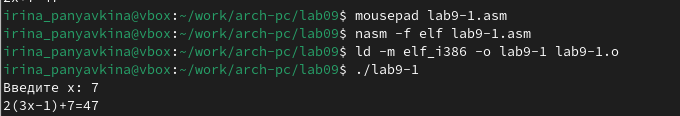


Рис. 3: Изменение программы первого листинга

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

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. 4).

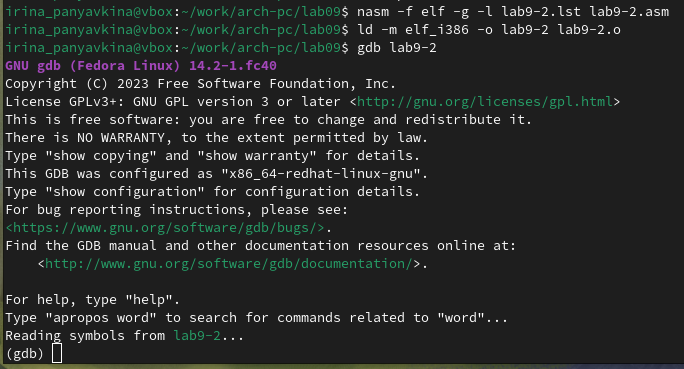


Рис. 4: Запуск программы в отладчике

Запустив программу командой run, я убедилась в том, что она работает корректно (рис. 5).

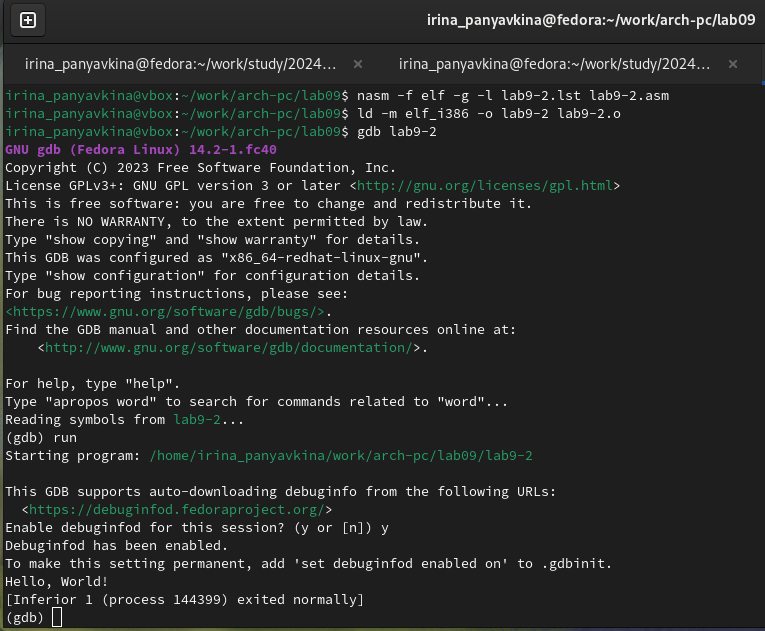


Рис. 5: Проверка программы отладчиком

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

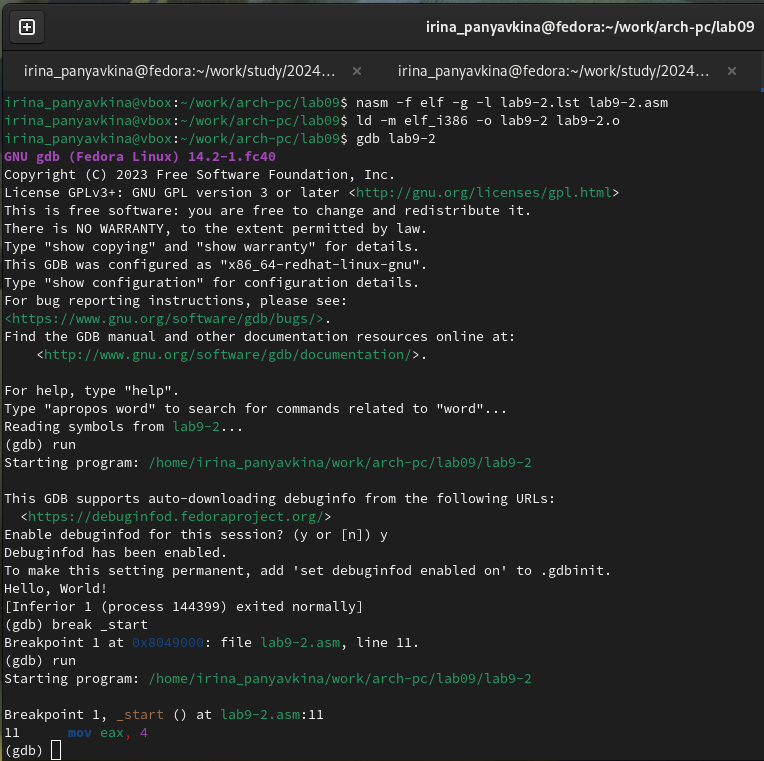


Рис. 6: Запуск отладчика с брейкпоинтом

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).

Затем смотрю дисассимилированный код программы, переключаюсь на отображение команд с синтаксисом Intel (рис. 7).

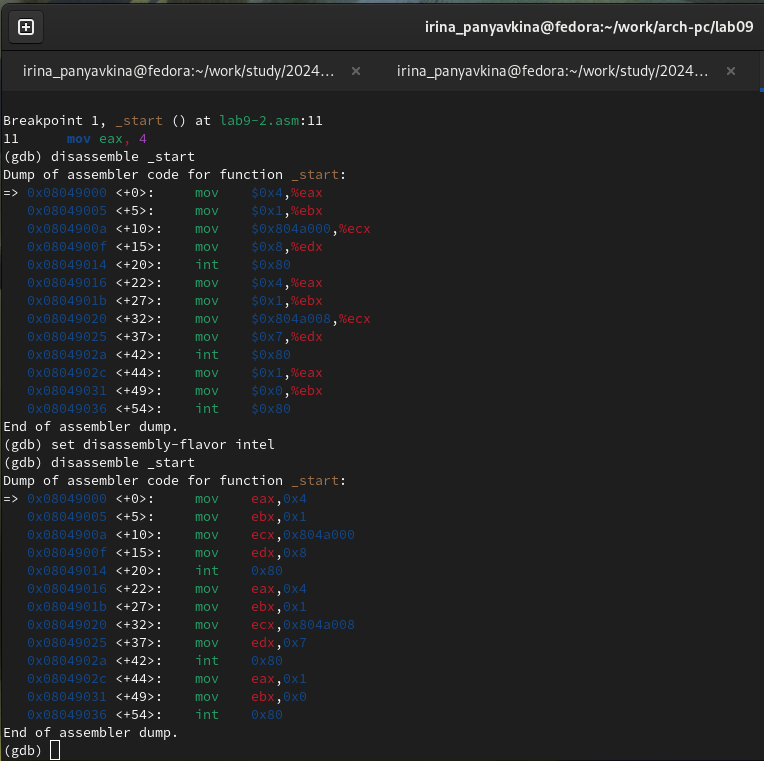


Рис. 7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы. К сожалению, во время выполнения работы, я столкнулась с проблемой («Register Values Unavailable»): интерфейс TUI не смог корректно распознать и отобразить все регистры в помощью предложенной команды «layout regs», все попытки исправить проблему оказались тщетными, причина мне так и неизвестна (делала строго опираясь на лекцию), поэтому мне пришлось воспользоваться другой командой и вывести список регистров иным способом с помощью «tui reg general». (рис. 8)

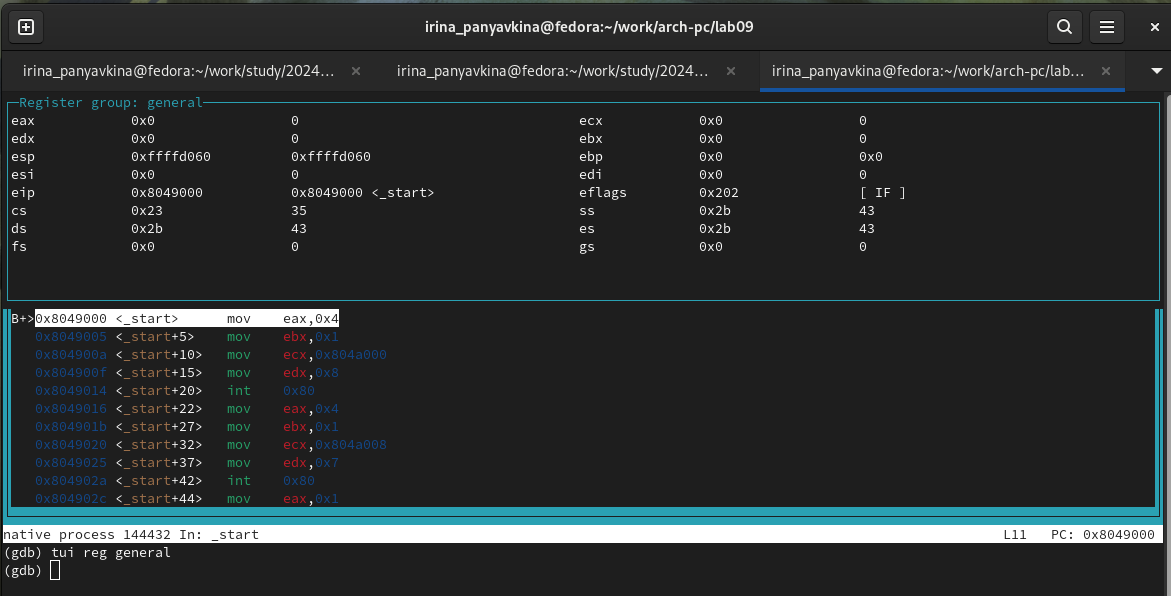


Рис. 8: Режим псевдографики

Добавление точек останова

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

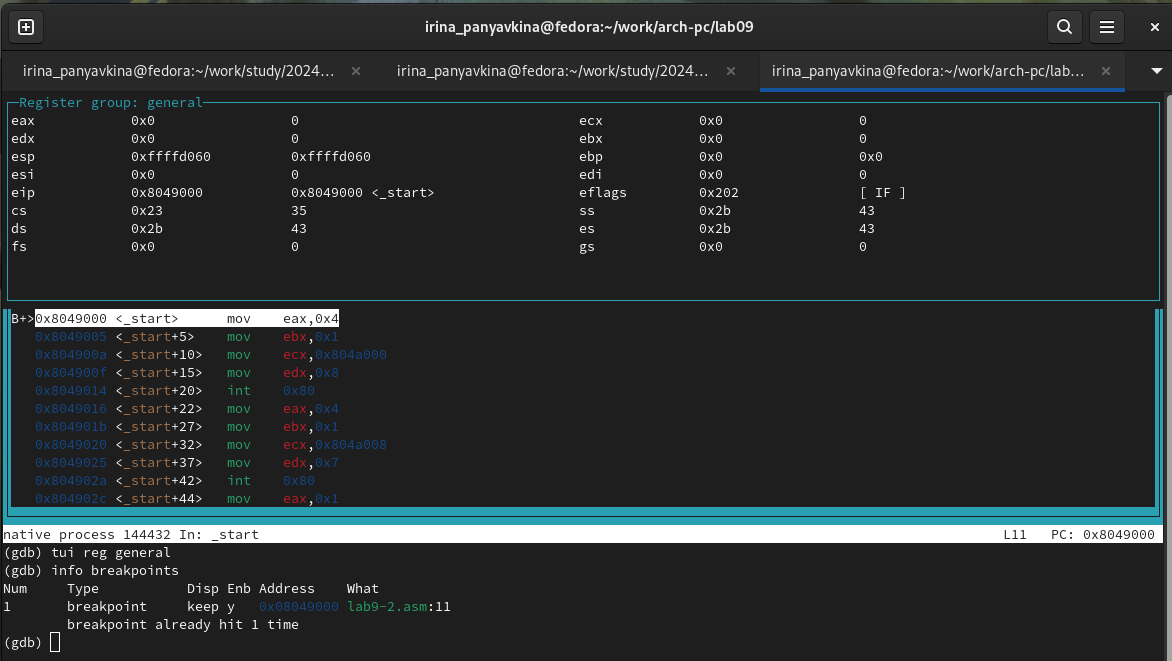


Рис. 9: Список брейкпоинтов

Устанавливаю ещё одну точку останова по адресу инструкции (рис. 10).

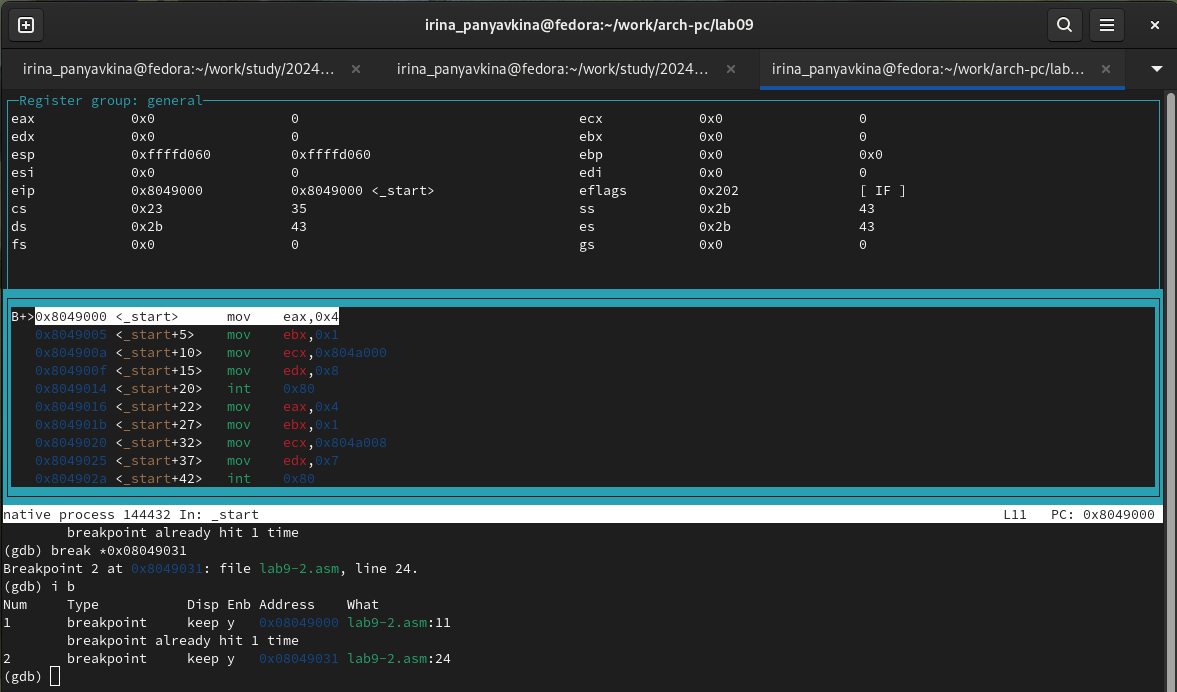


Рис. 10: Добавление второй точки останова

Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды «info registers» (рис. 11).

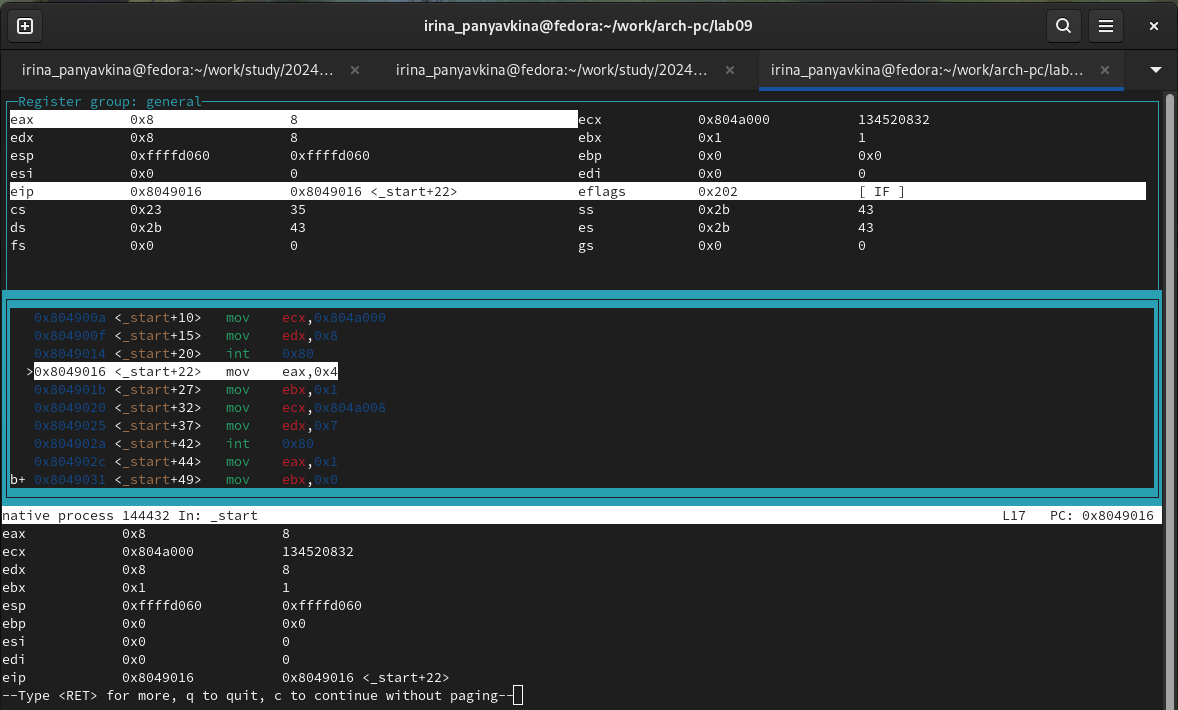


Рис. 11: Просмотр содержимого регистров

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

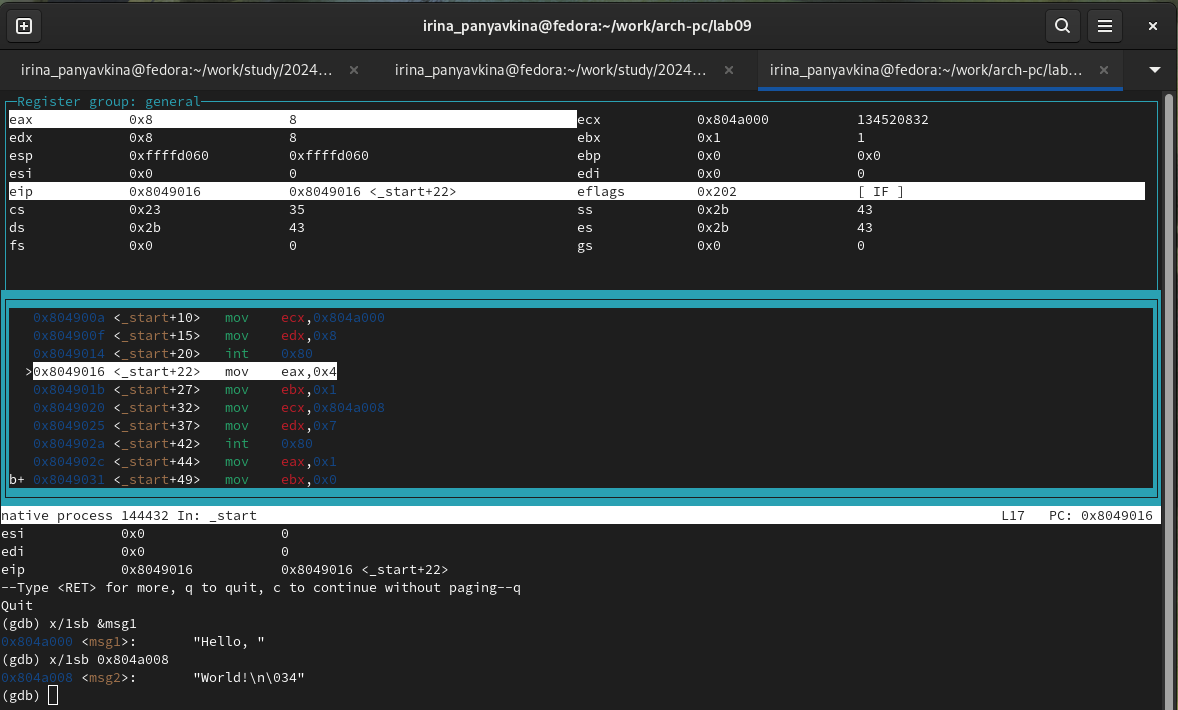


Рис. 12: Просмотр содержимого переменных двумя способами

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

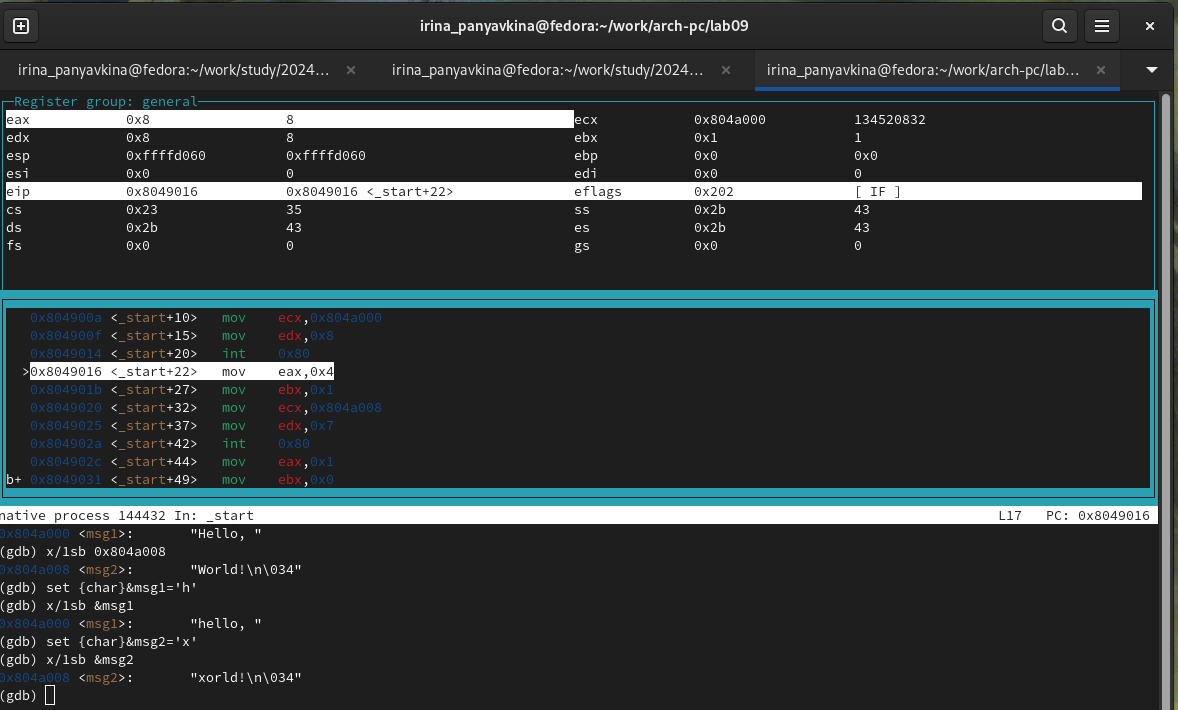


Рис. 13: Изменение содержимого переменных двумя способами

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

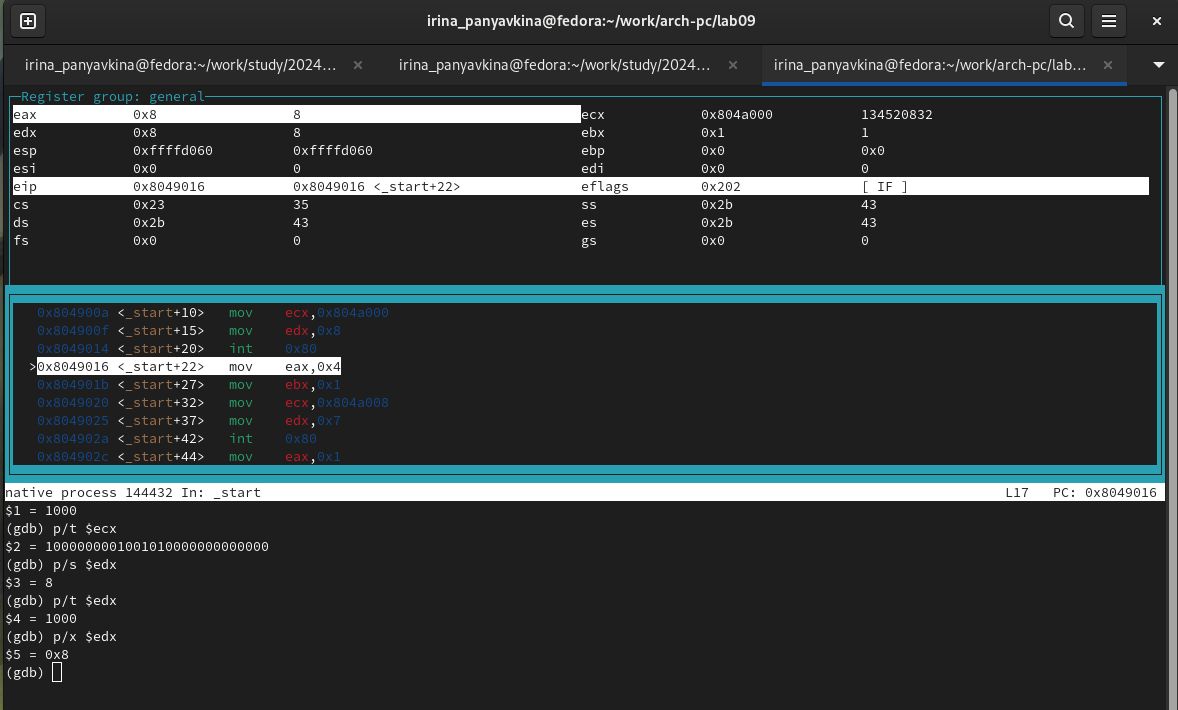


Рис. 14: Просмотр значения регистра разными представлениями

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

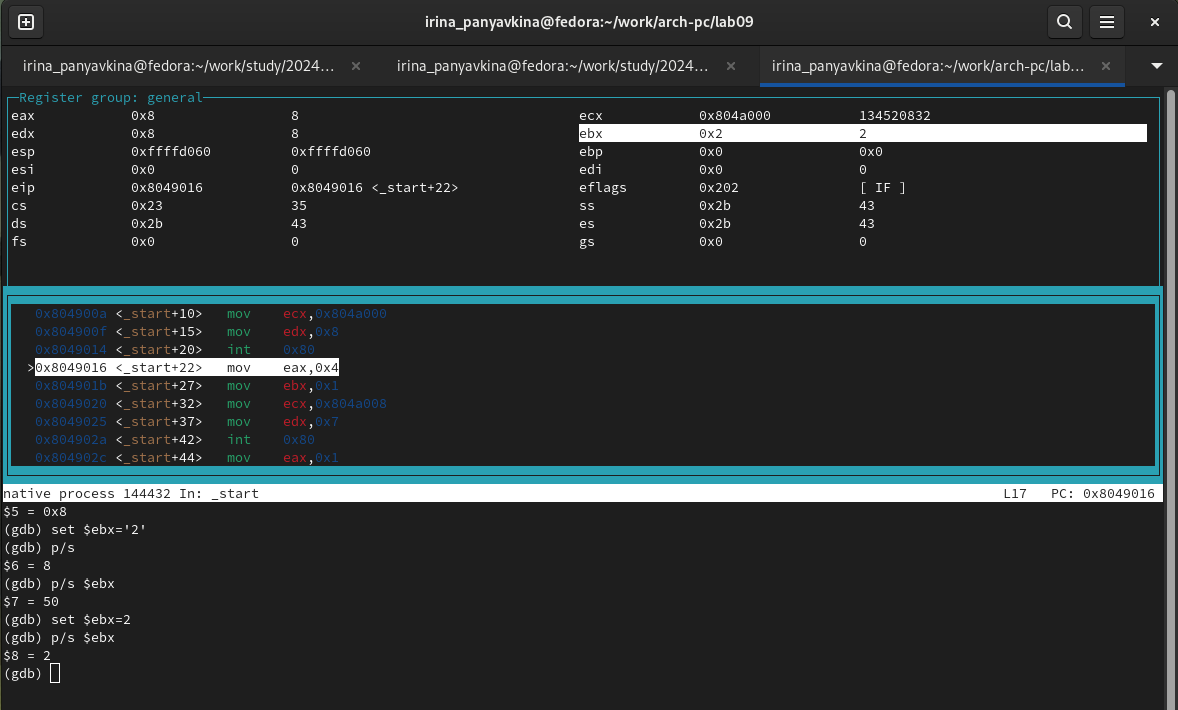


Рис. 15: Подготовка новой программы

Обработка аргументов командной строки в GDB

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

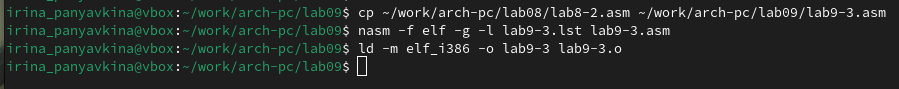


Рис. 16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт 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. 17).

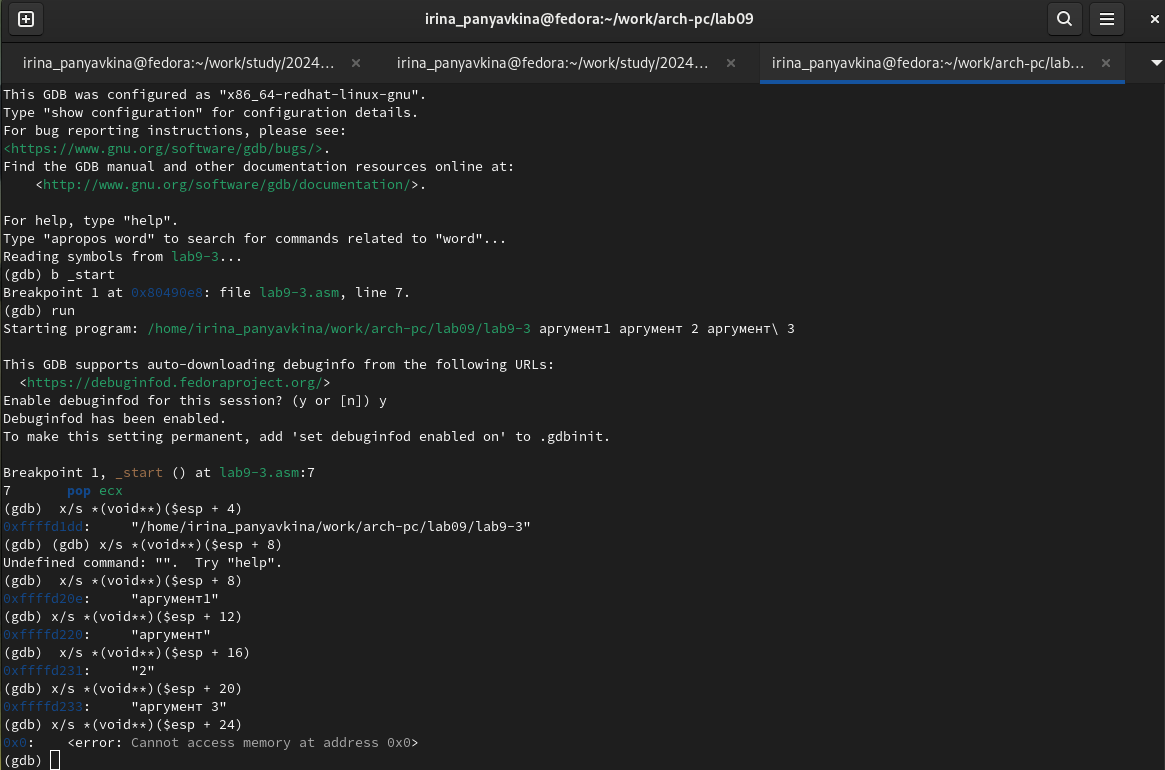


Рис. 17: Проверка работы стека

Выполнение заданий для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 18).

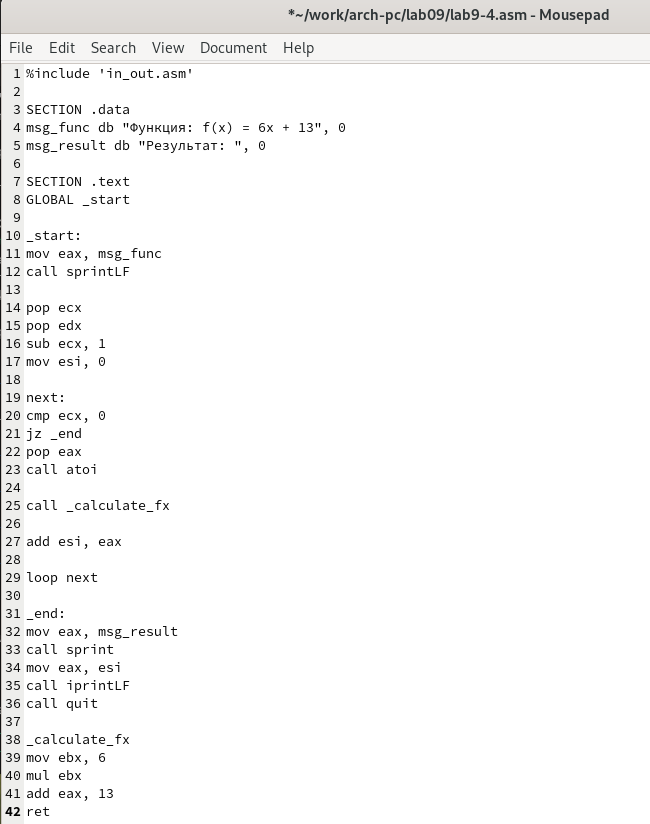


Рис. 18: Изменённая программа предыдущей лабораторной работы

Код программы:

%include 'in\_out.asm'  
  
SECTION .data  
msg\_func db "Функция: f(x) = 6x + 13", 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, 0  
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, 6  
mul ebx  
add eax, 13  
ret

1. Запускаю программу в режиме отладчика и пошагово через «stepi» просматриваю изменение значений регистров через «info registers». 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 в результате программа неверно высчитывает функцию (рис. 19).

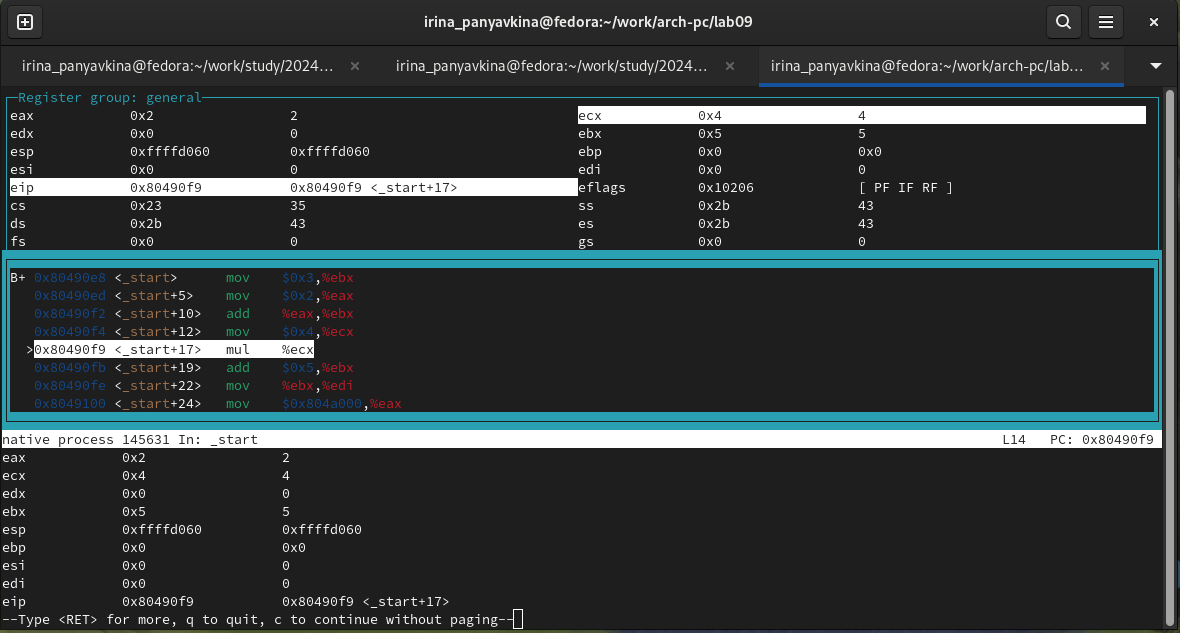


Рис. 19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа правильно считает значение функции (рис. 20).

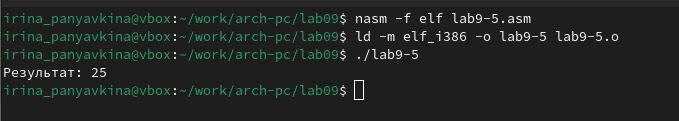


Рис. 20: Проверка корректировок в программе

Код изменённой 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 ходе выполнения лабораторной работы, я приобрела навыки написания программ с использованием подпрограмм и познанакомилась с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. Org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm\_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).