Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Гаязов Рузаль Ильшатович

Содержание

# Цель работы

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

# 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:

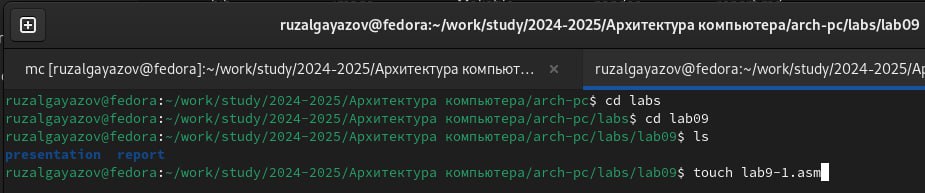
• синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — 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.

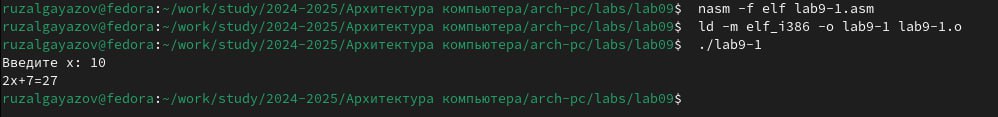
# Выполнение лабораторной работы

Создаю файл lab9-1.asm (рис. -@fig:001).



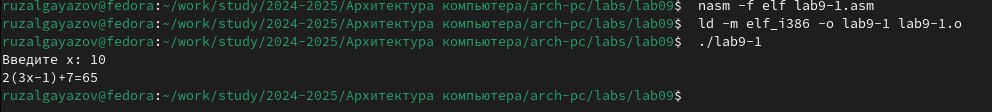
Создание файлa

Копирую в файл код из листинга, компилирую и запускаю его (рис. -@fig:002).



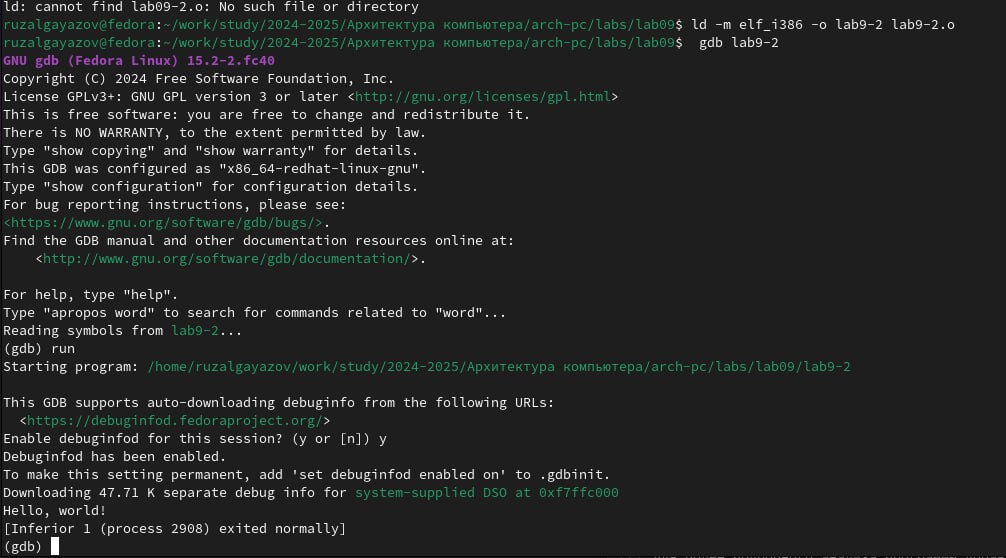
Запуск программы

Изменяю текст программы, добавив в нее подпрограмму (рис. -@fig:003).



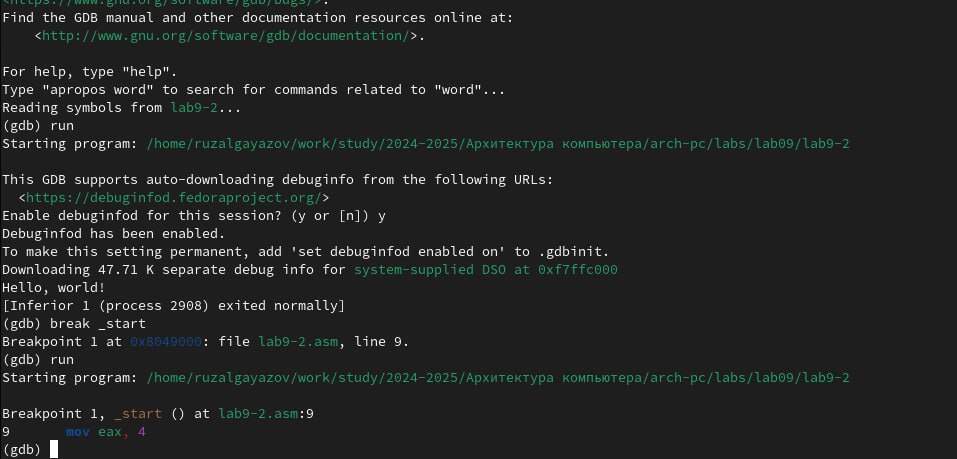
Изменение программы

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике. Запустил программу командой run (рис. -@fig:004).



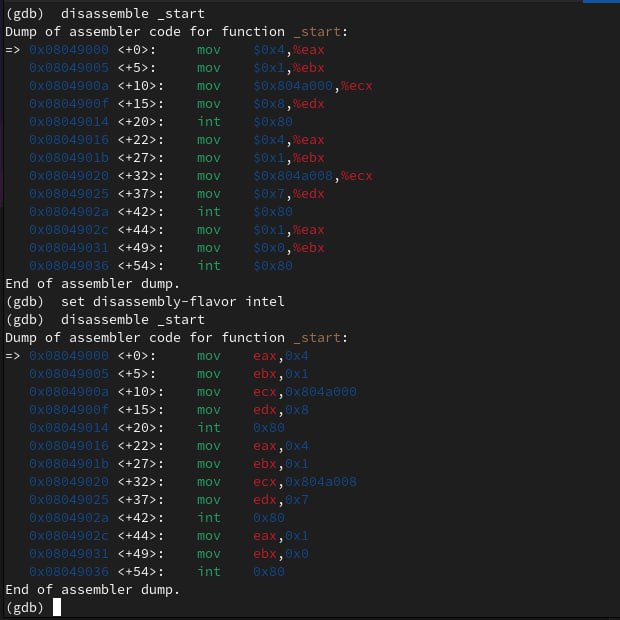
Запуск программы

Для более подробного анализа программы добавляю брейкпоинт на метку \_start и снова запускаю отладку (рис. -@fig:005).



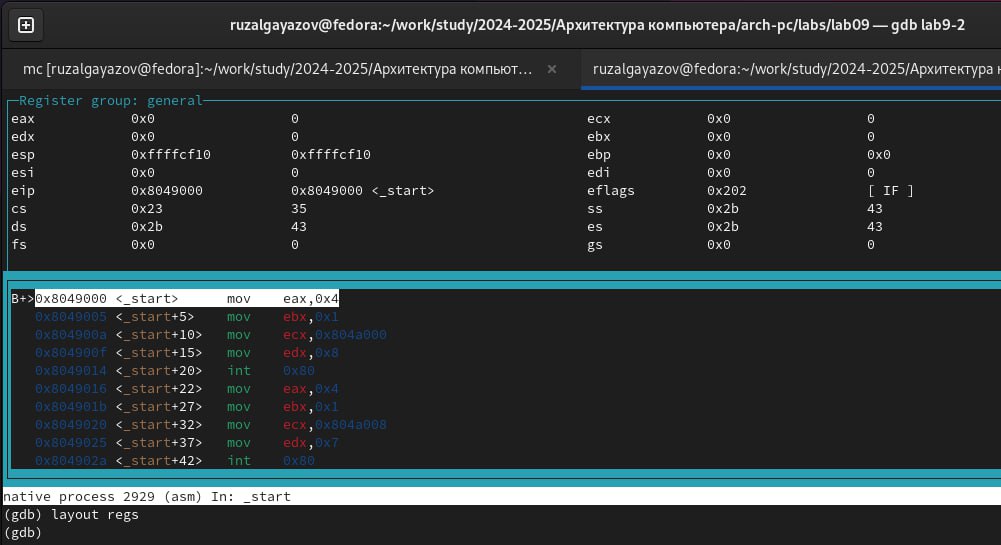
Запуск отладичка

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel (рис. -@fig:006). 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).



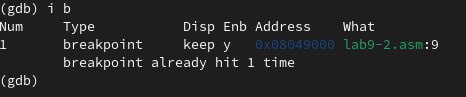
Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -@fig:007).



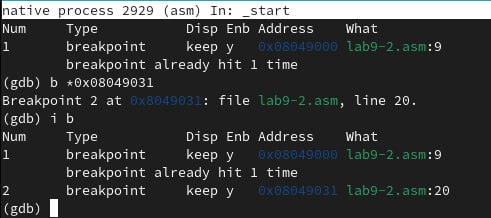
Режим псевдографики

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. -@fig:008).



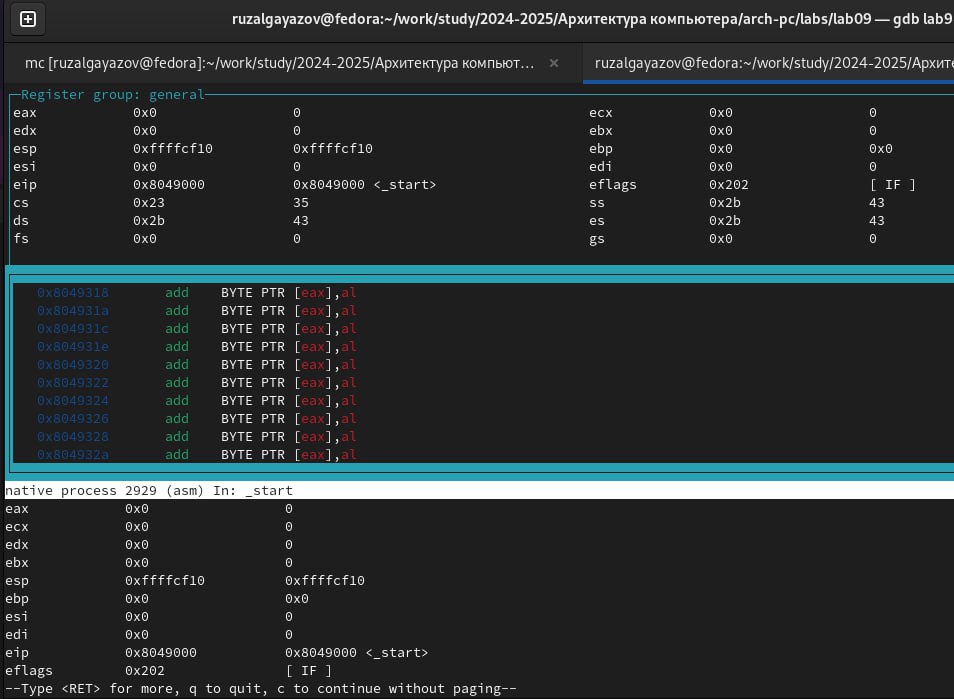
Список брейкпоинтов

Устаналиваю еще одну точку останова по адресу инструкции (рис. -@fig:009).



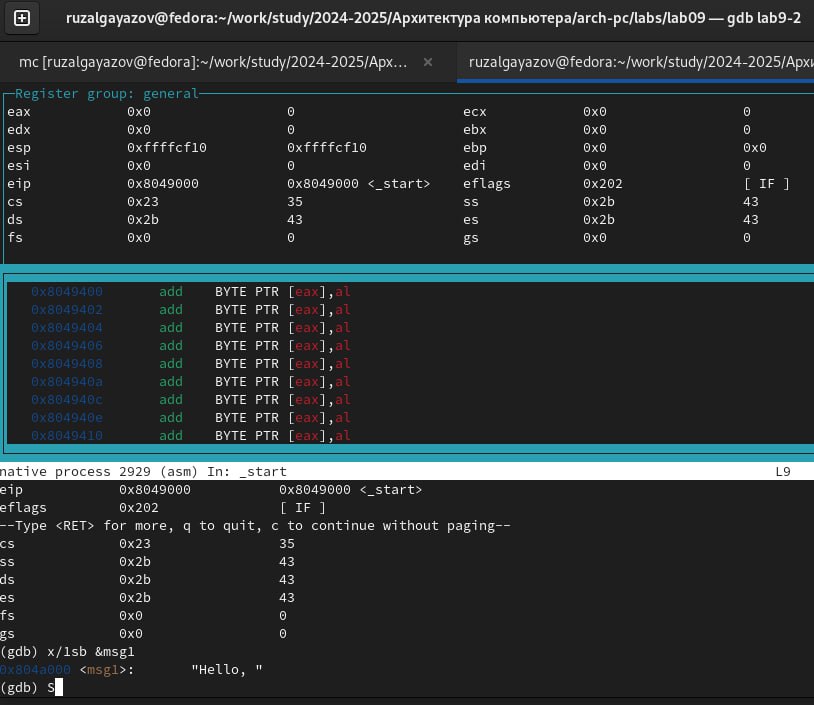
Добавление точки останова

Просматриваю содержимое регистров командой info registers (рис. -@fig:010).



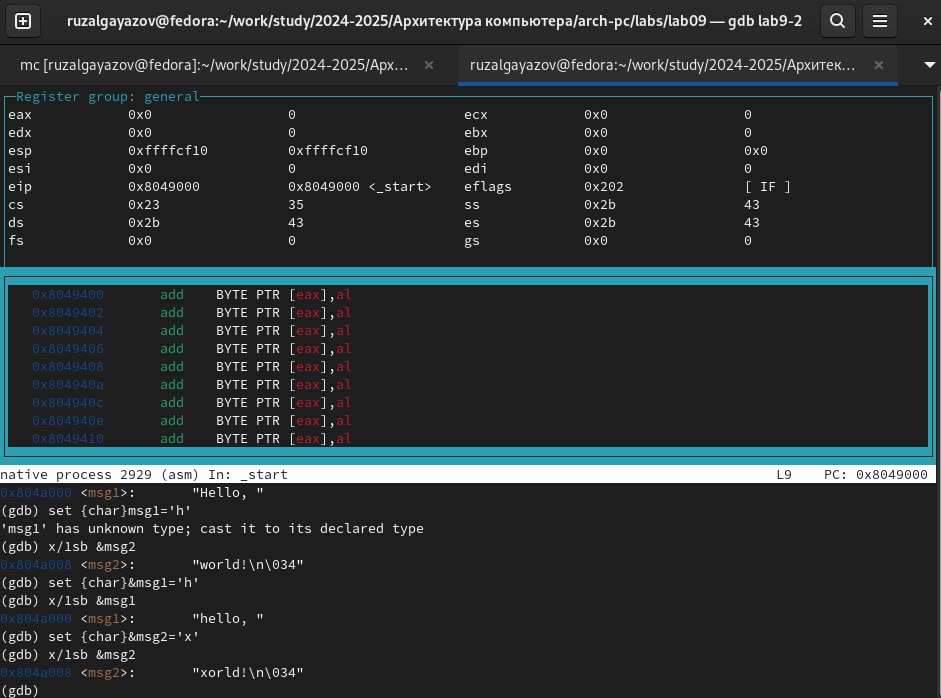
Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -@fig:011).



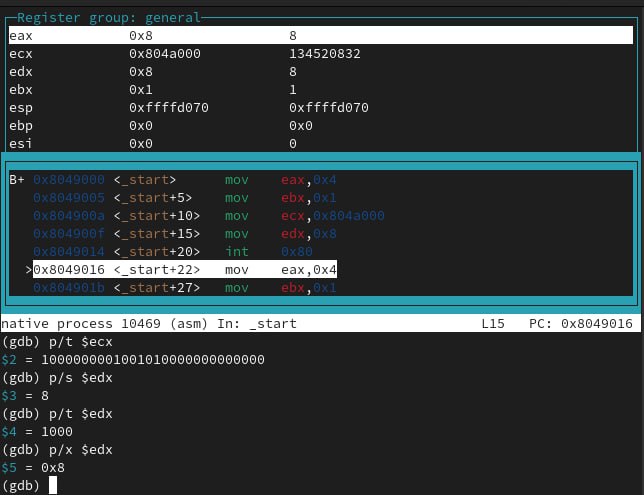
Просмотр содержимого

Меняю содержимое переменных по имени и по адресу (рис. -@fig:012).



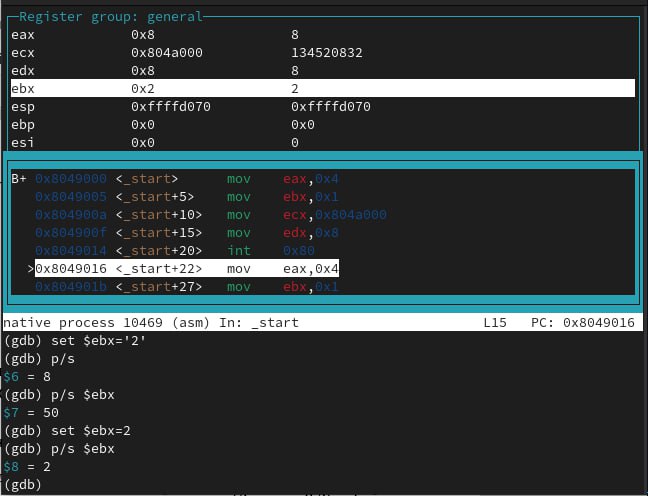
Изменение содержимого

Вывожу в различных форматах значение регистра edx (рис. -@fig:013).



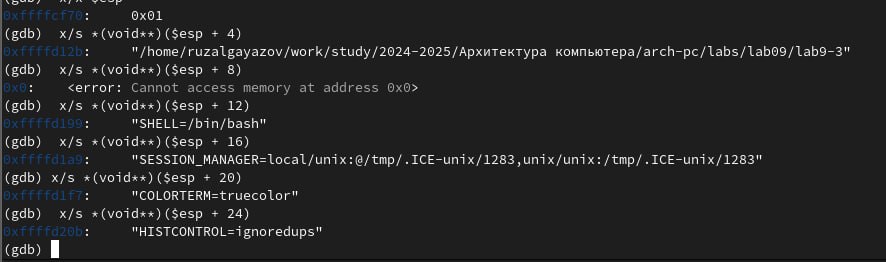
Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -@fig:014).



Примеры использования set

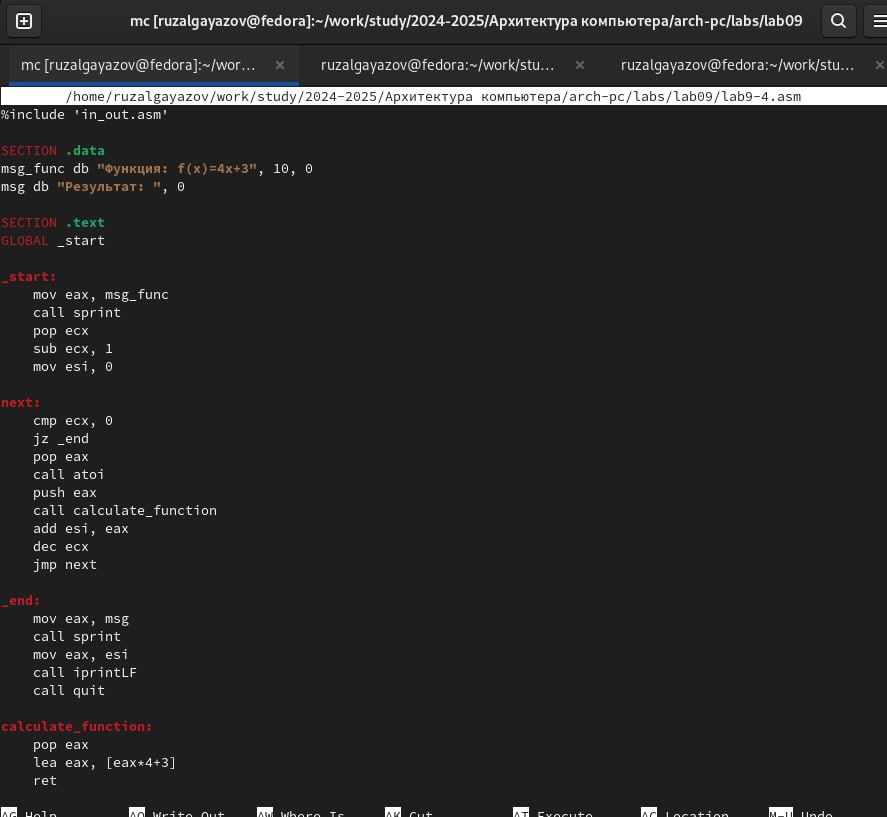
Копирую программу из предыдущей лабораторной работы в текущий каталог. Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпопнт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились. (рис. -@fig:015).



Проверка работы стека

## Задание для самостоятельной работы

Меняю программу самостоятельной части с использованием подпрограммы (рис. -@fig:016).

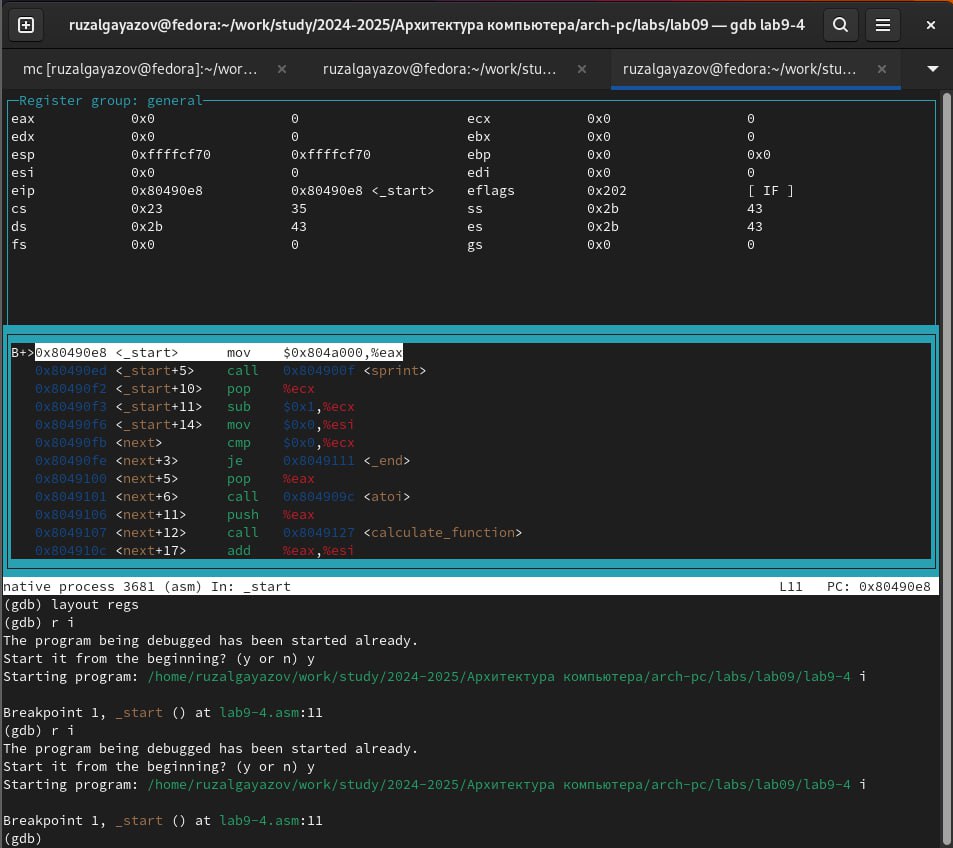


Измененная программа

Код программы:

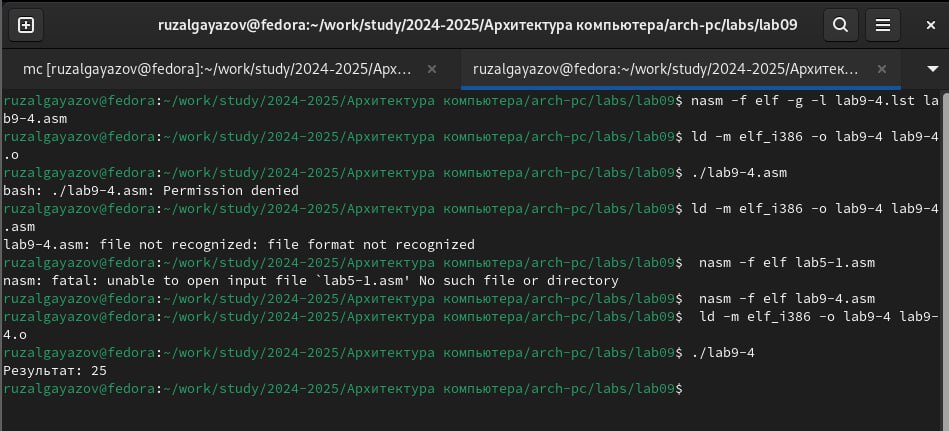
%include 'in\_out.asm'  
  
SECTION .data  
msg\_func db "Функция: f(x)=4x+3", 10, 0  
msg db "Результат: ", 0  
  
SECTION .text  
GLOBAL \_start  
  
\_start:  
 mov eax, msg\_func  
 call sprint  
 pop ecx  
 sub ecx, 1  
 mov esi, 0  
  
next:  
 cmp ecx, 0  
 jz \_end  
 pop eax  
 call atoi  
 push eax  
 call calculate\_function  
 add esi, eax  
 dec ecx  
 jmp next  
  
\_end:  
 mov eax, msg  
 call sprint  
 mov eax, esi  
 call iprintLF  
 call quit  
  
calculate\_function:  
 pop eax  
 lea eax, [eax\*4+3]  
 ret

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:017).



Поиск ошибки

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -@fig:018).



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

# 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. 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,
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).