Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров

Симонова Виктория Игоревна

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 программам с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

# 3 Теоретическое введение

## 3.1 Понятие об отладке

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.

## 3.2 Методы отладки

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 сообще- ния); • использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и из- менять 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 программа- отладчик 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 програм- мы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

## 3.3 Понятие подпрограммы

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма бу- дет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

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

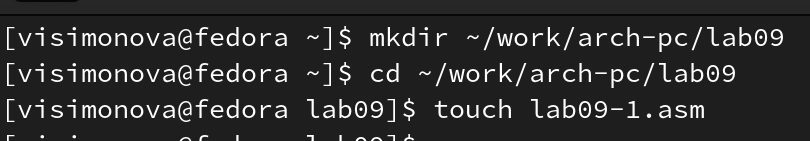


Figure 1: Создание каталога и файла

Ввожу в файл текст программы из листинга с использованием подпрограмм (рис. [[2](#fig:002)]).

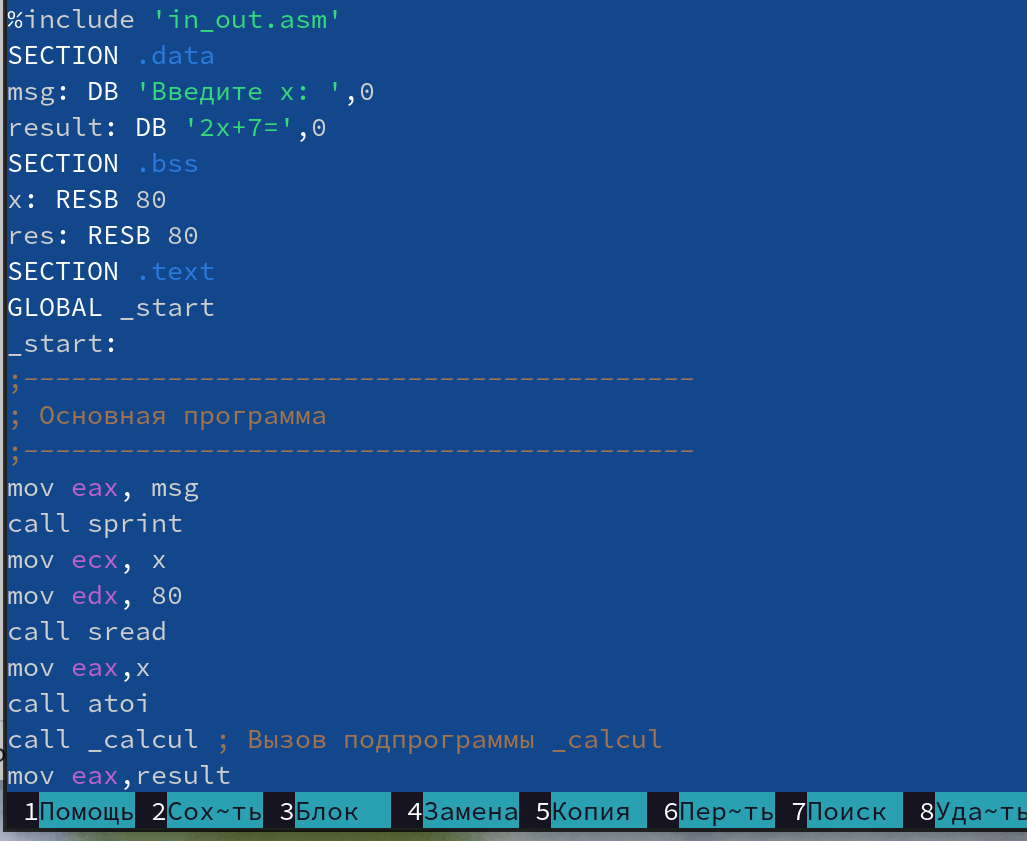


Figure 2: Ввожу код программы

Создаю исполняемый файл и проверяю его работу (рис. [[3](#fig:003)]).

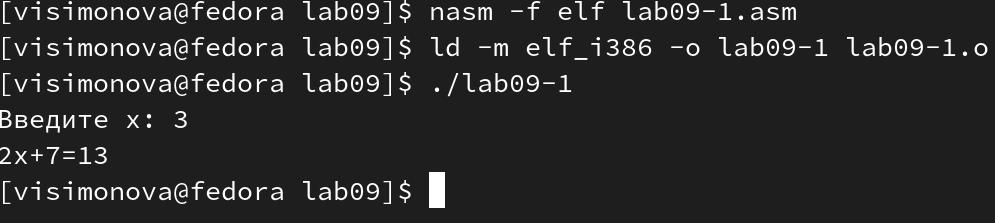


Figure 3: Запускаю исполняемый файл

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x +7, g(x) = 3x - 1. (рис. [[4](#fig:004)]).

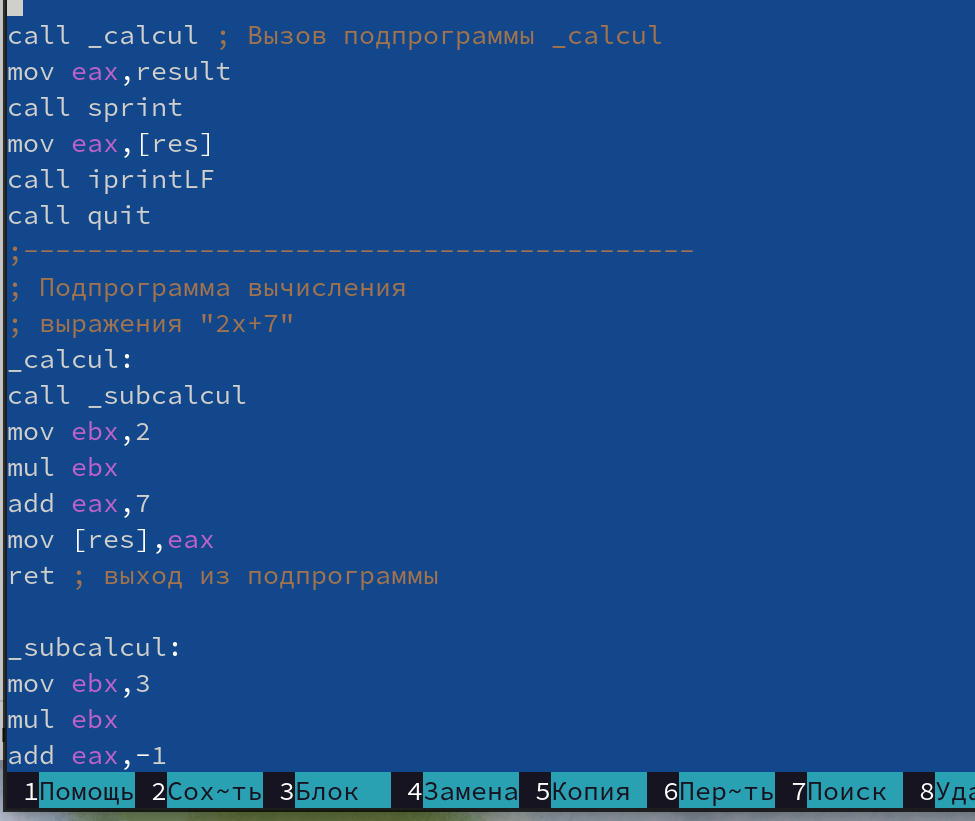


Figure 4: Добавляю подпрограмму

Создаю исполняемый файл и проверяю его работу (рис. [[5](#fig:005)]).

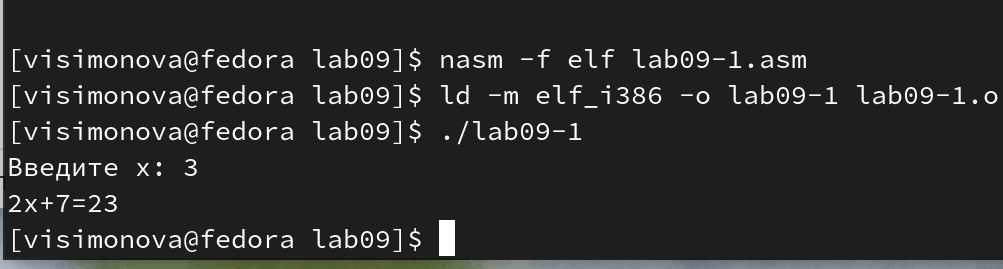


Figure 5: Запускаю исполняемый файл

## 4.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга (рис. [[6](#fig:006)]).

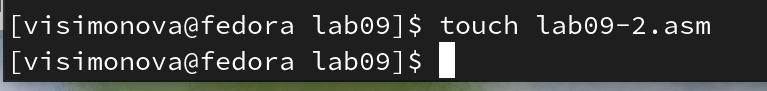


Figure 6: Создаю файл

Ввожу код программы(рис. [[7](#fig:007)]).

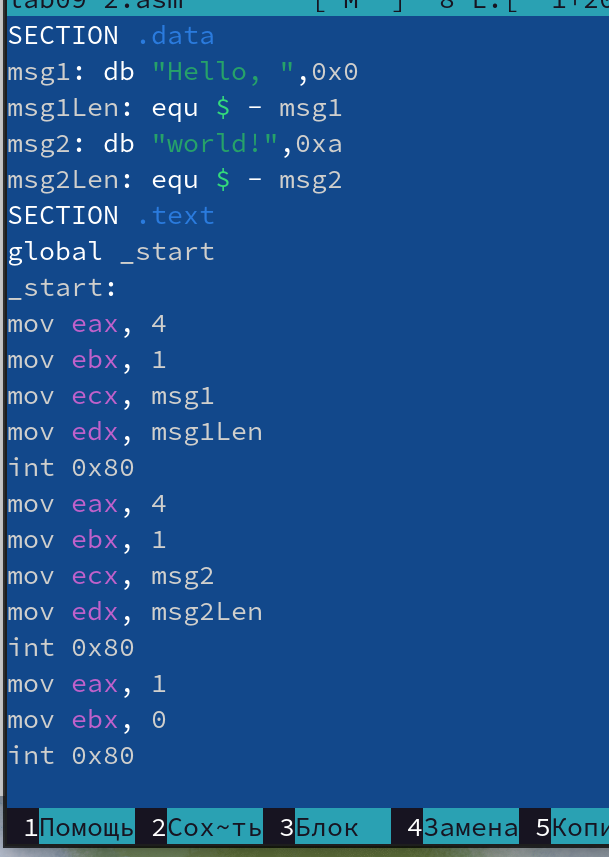


Figure 7: Ввожу код программы вывода

Создаю исполняемый файл и проверяю его работу (рис. [[8](#fig:008)]).

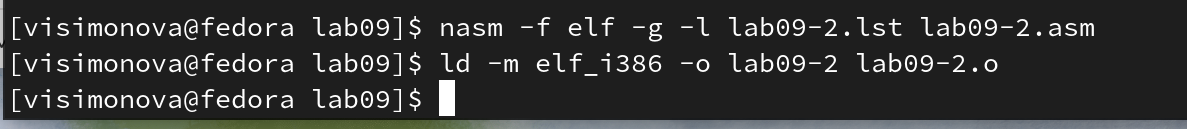


Figure 8: Запускаю исполняемый файл

Загружаю исполняемый файл в отладчик (рис. [[9](#fig:009)]).

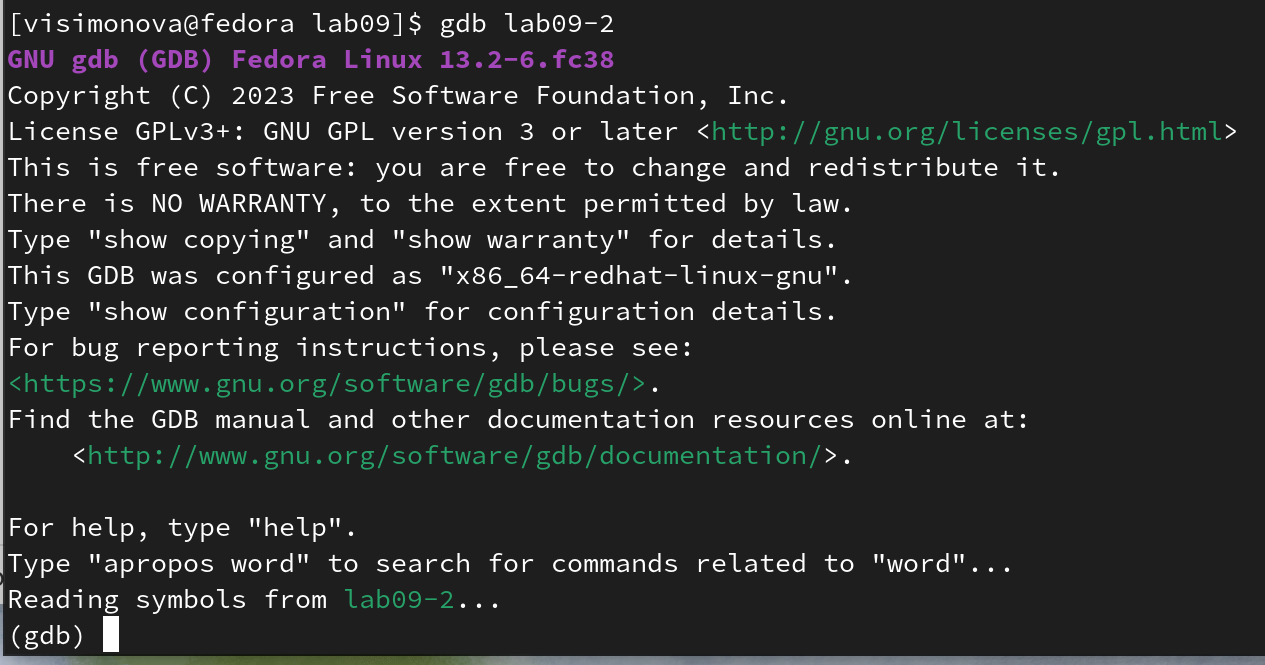


Figure 9: Загрузка

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [[10](#fig:010)]).

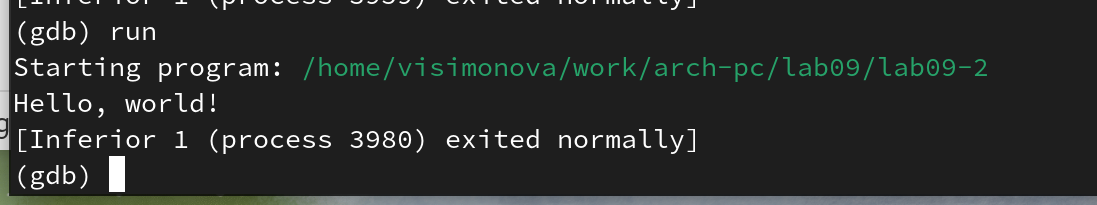


Figure 10: Проверка

Для более подробного анализа программы установливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её.(рис. [[11](#fig:011)]).

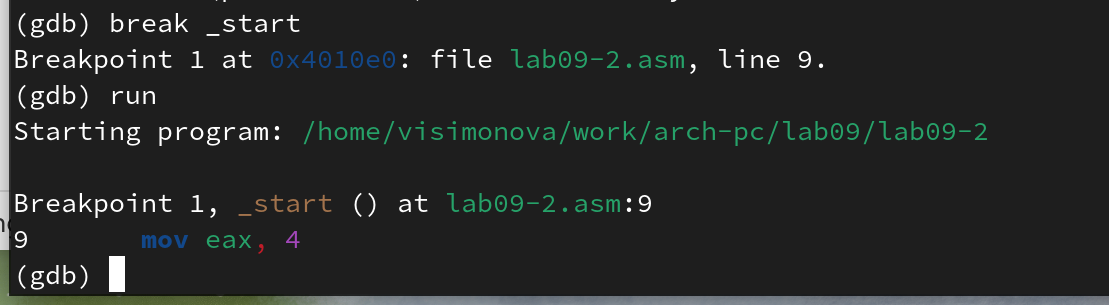


Figure 11: Установка и запуск

Посмотриваю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start и переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel(рис. [[12](#fig:012)]).

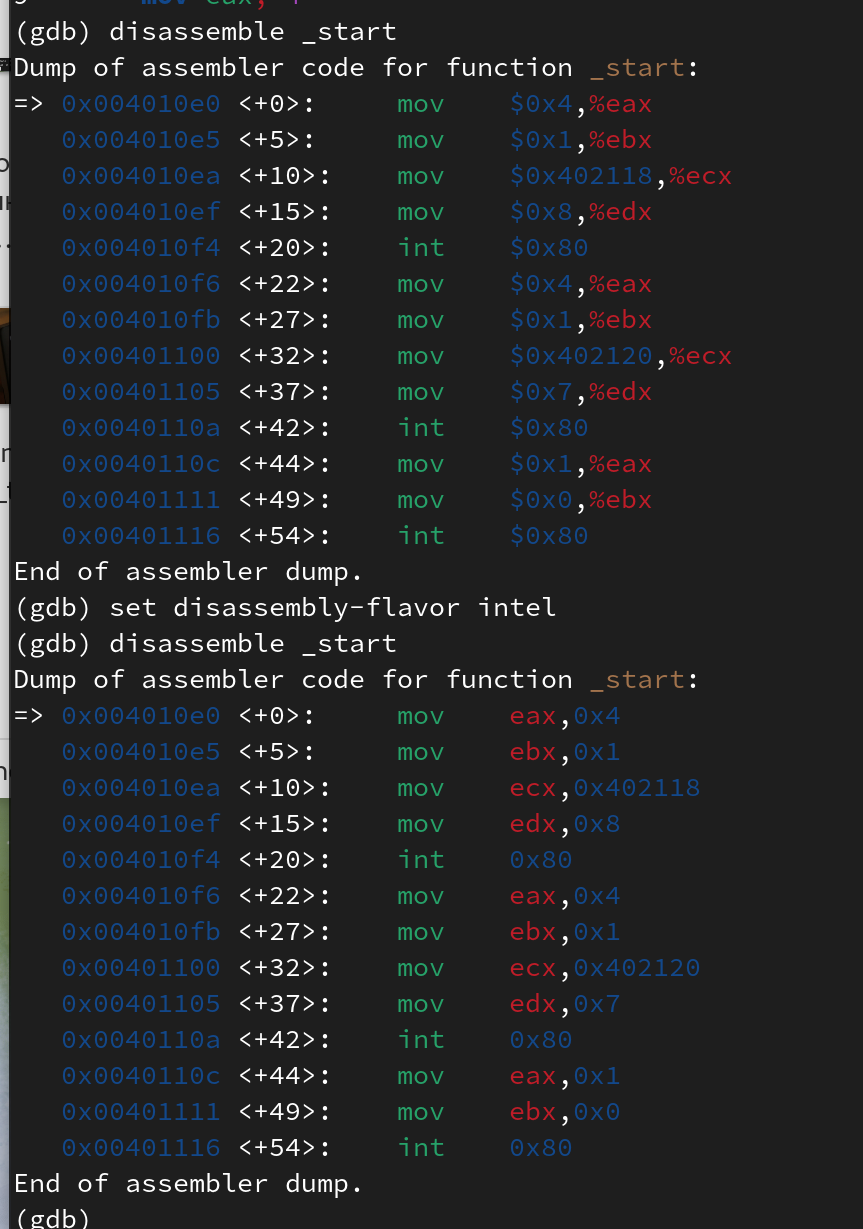


Figure 12: Использование команд

В режиме АТТ имена операндов начинаются со знака “$”,а имена регистров со знака “%” . Включите режим псевдографики для более удобного анализа программы (рис. [[13](#fig:013)]).

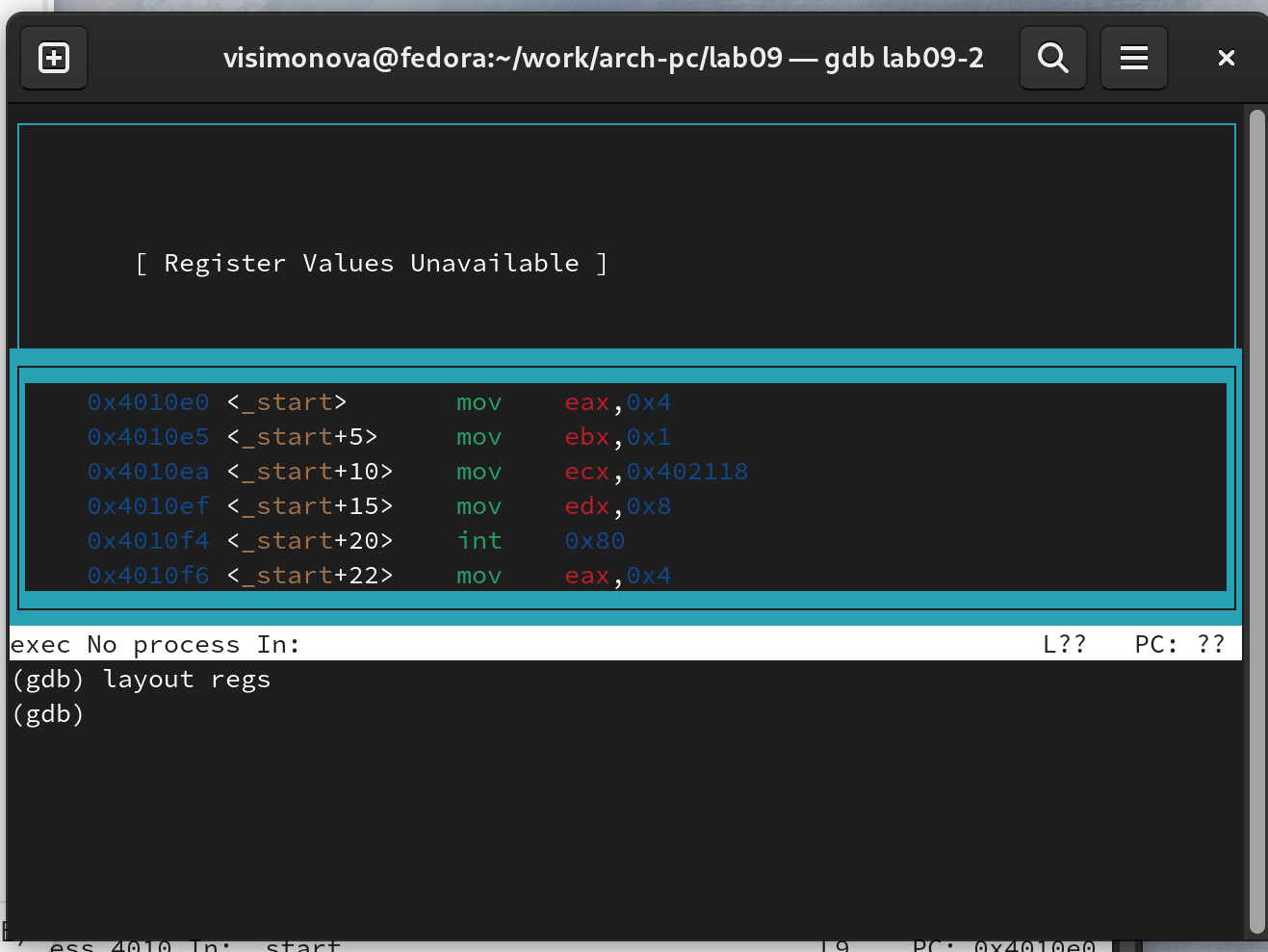


Figure 13: Включение режима псеадографики

## 4.3 Добавление точек останова

Проверяю точку останова по метке \_start(с помощью команды i b) и устанавливаю ещё одну точку останова по адресу инструкции mov ebx,0x0, и просматриваю информацию о всех точках останова. (рис. [[14](#fig:014)]).

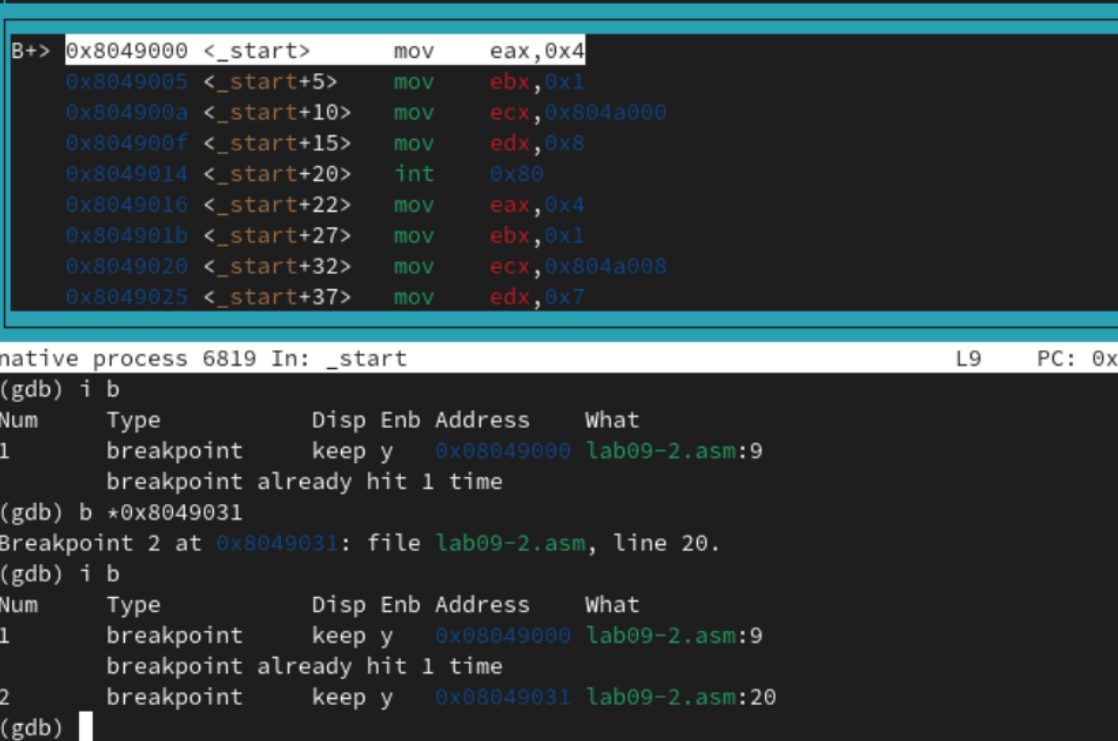


Figure 14: Установление точек останова и просмотр информации о них

## 4.4 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменинием значения регистров (рис. [[15](#fig:016)]).

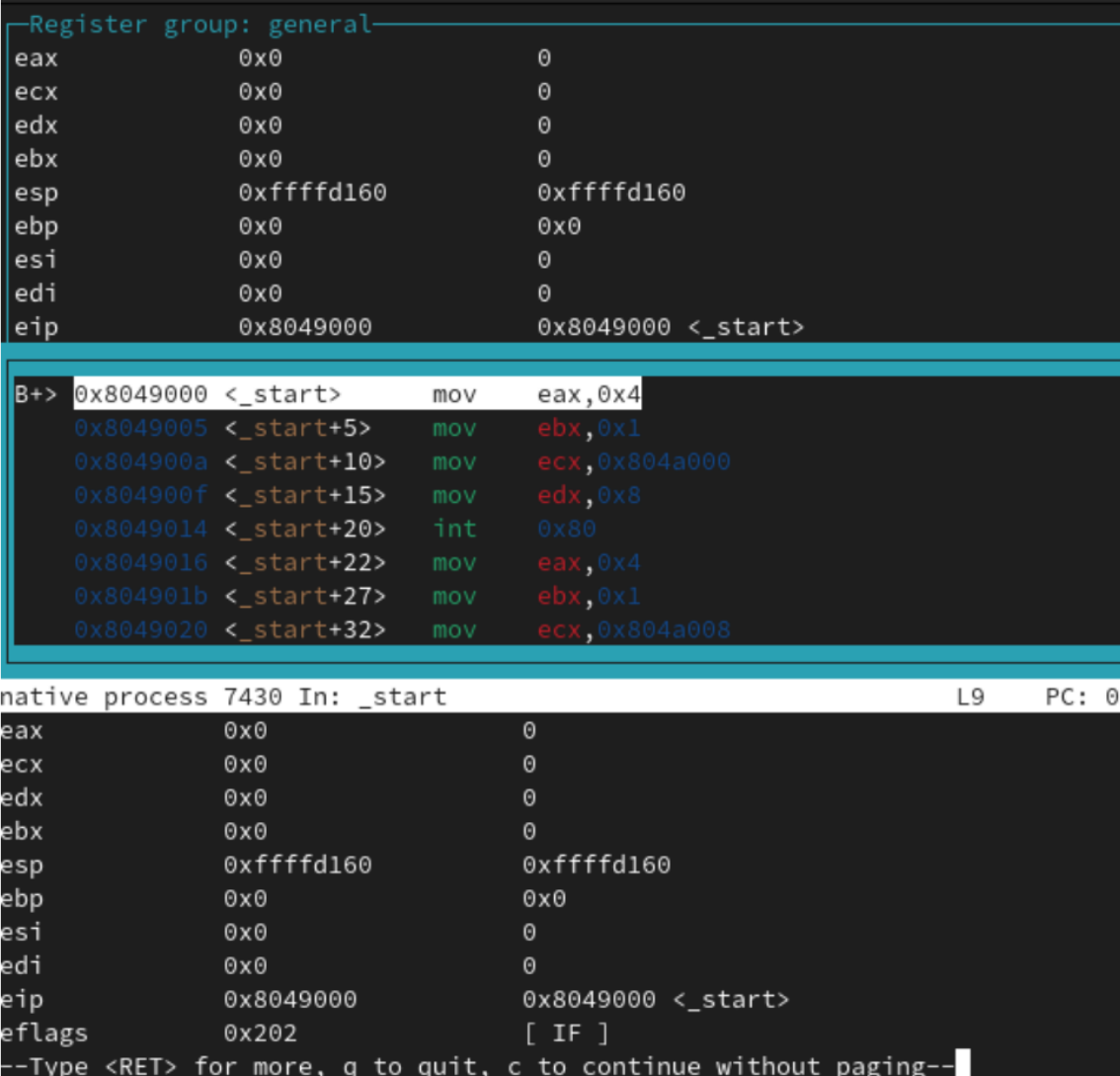


Figure 15: До использование команды stepi

После использования команды (рис. [[16](#fig:017)]).

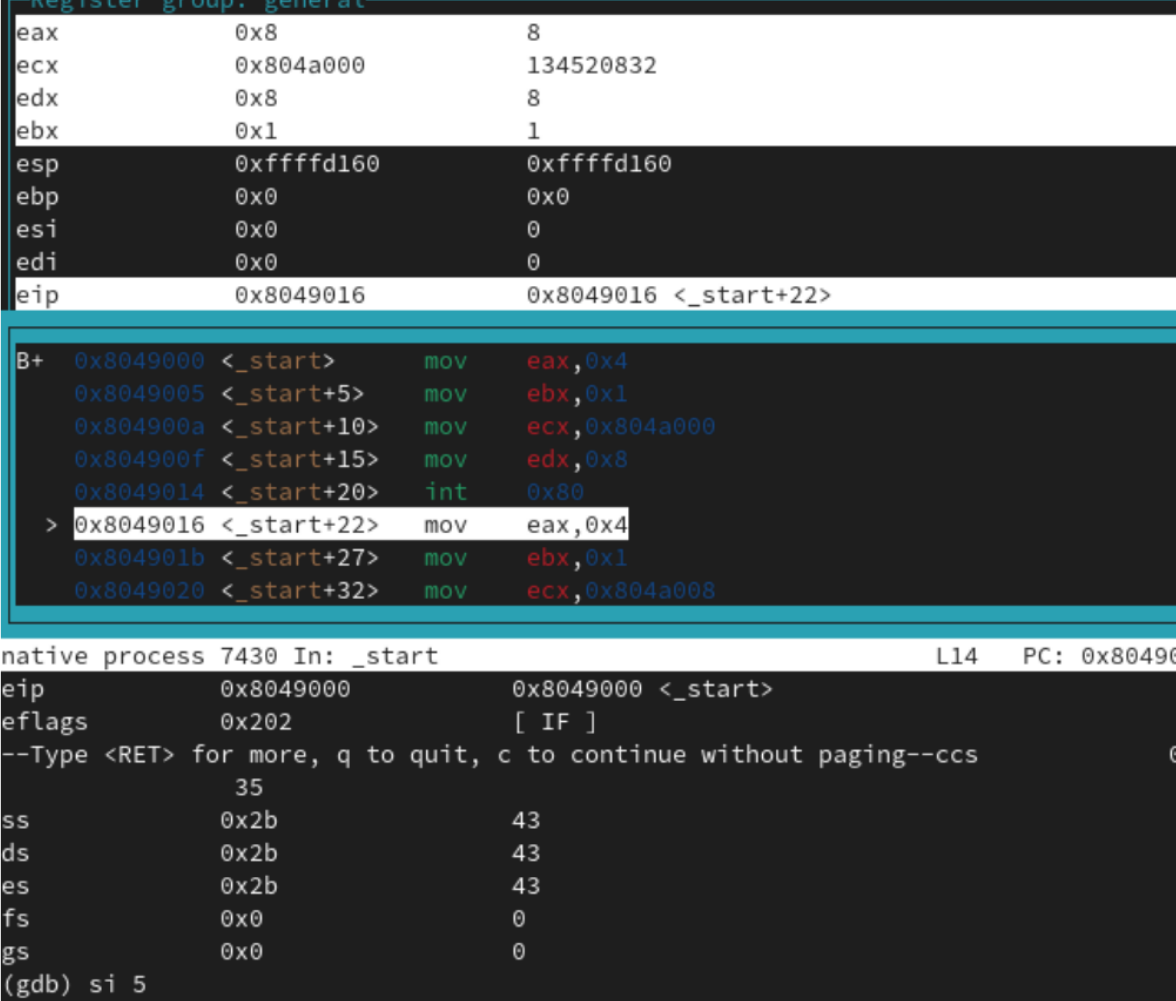


Figure 16: После использование команды stepi

Изменились значения решистров eax,ebx,ecx,edx Просматриваю значение msg1 c помощью команды x/1sb &msg1 (рис. [[17](#fig:018)]).

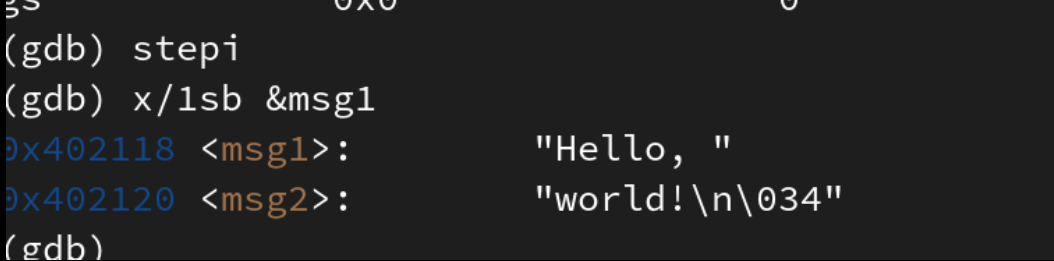


Figure 17: Просмотр

Изменяю первый символ переменной msg1, заменяю любой символ во второй переменной msg2 (рис. [[18](#fig:019)]).

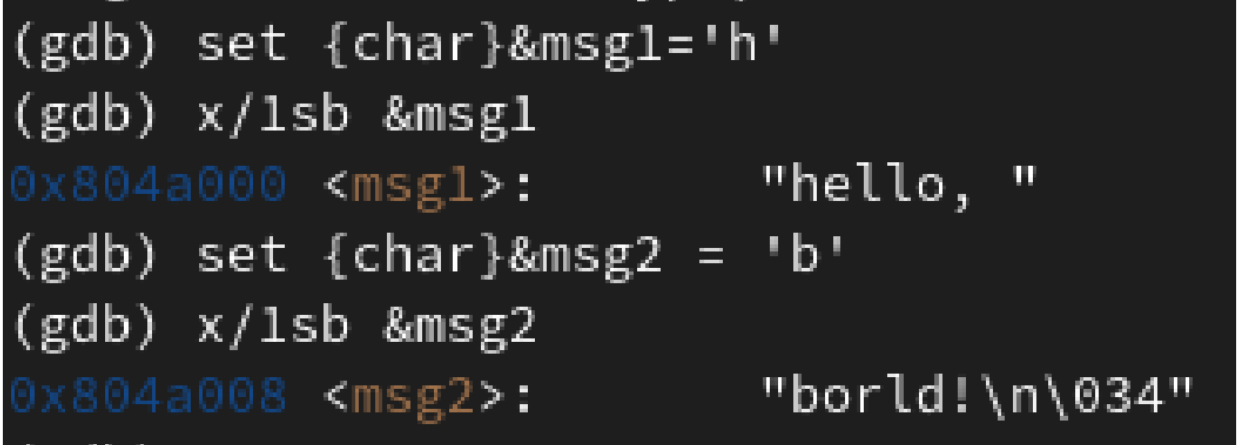


Figure 18: Изменение переменных

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [[19](#fig:020)]).

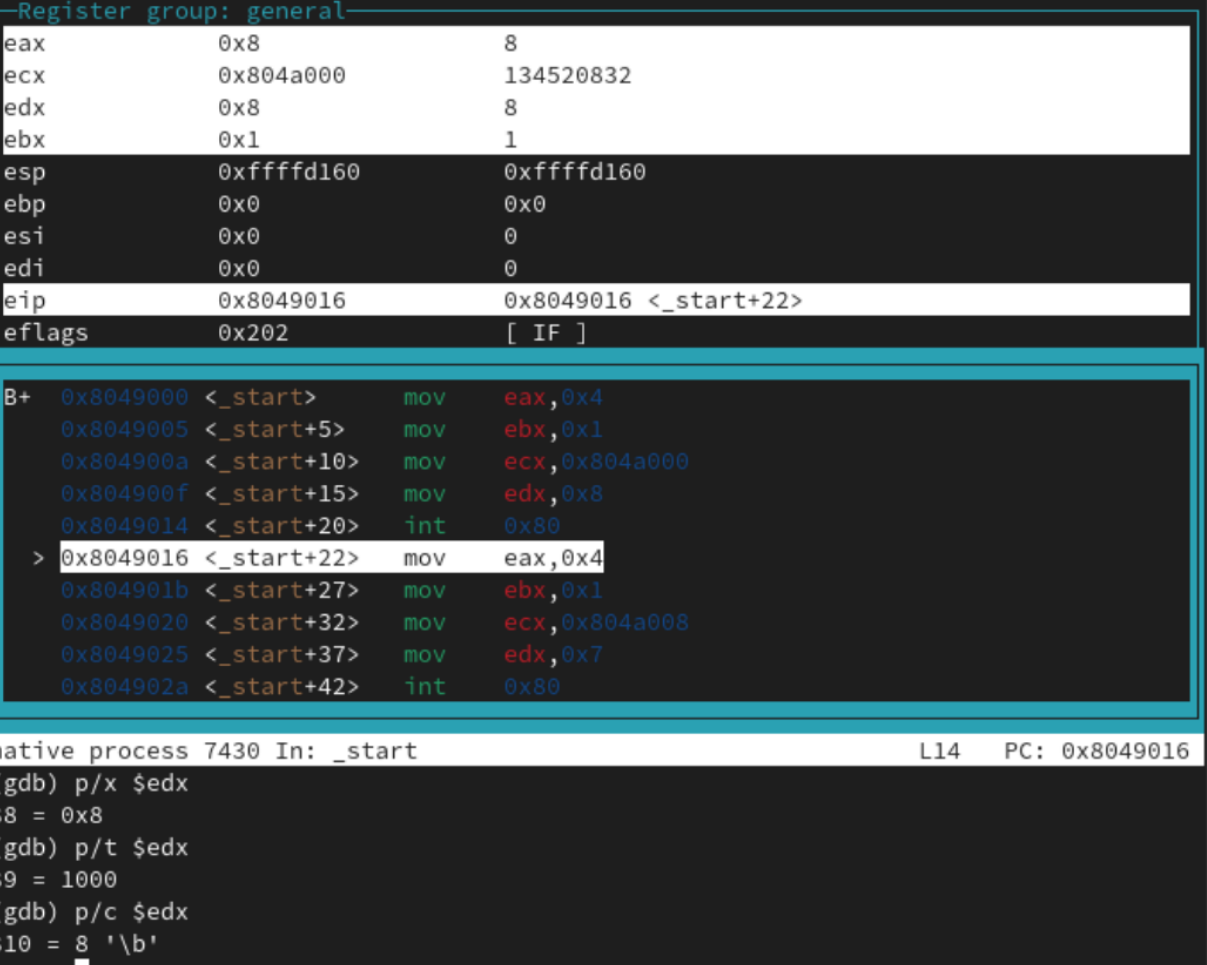


Figure 19: Ввывод регистра в разных форматах

С помощью команды set измените значение регистра ebx:(рис. [[20](#fig:021)]).

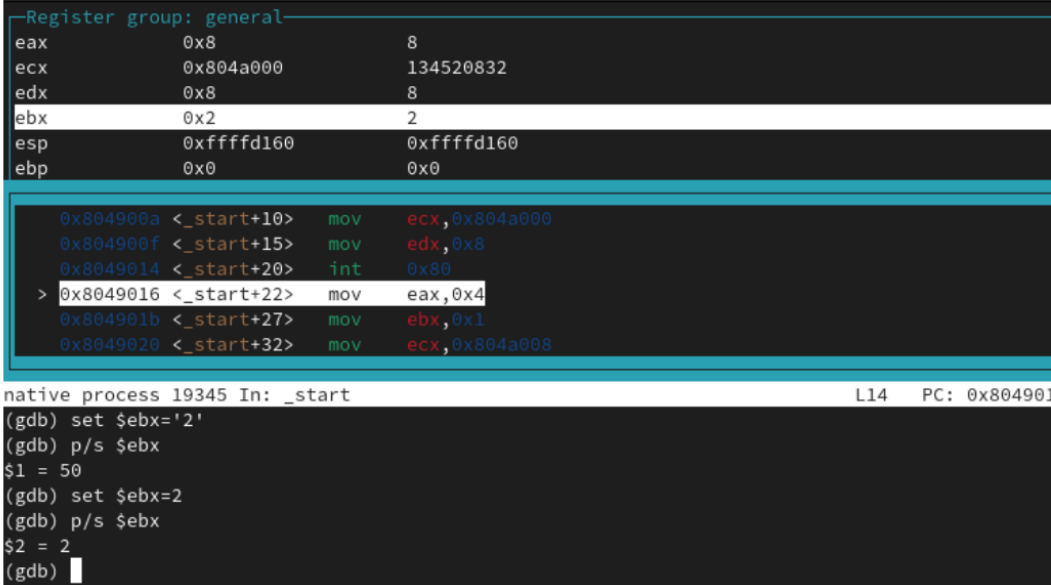


Figure 20: Изменение значения регистра

Разница: в первом случае символ переведён в строковый вид, во втором случае число в строковом виде не изменяется

Завершаю выполнение программы с помощью команды continue, выхожу командой quit (рис. [[21](#fig:022)]).

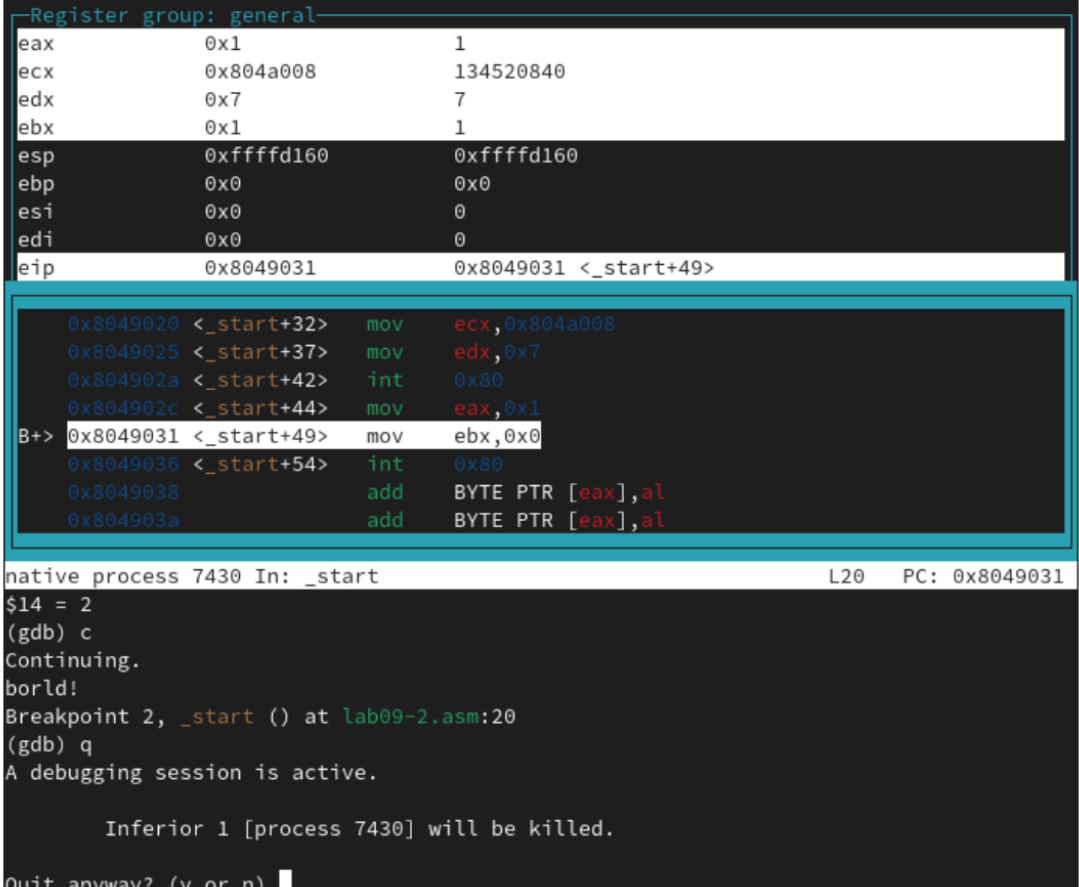


Figure 21: Завершение работы

## 4.5 Обработка аргументов командной строки в GDB

Скопирую 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. Создаю исполняемый файл.Загружаю исполняемый файл в отладчик,используя ключ –args, указав аргументы (рис. [[22](#fig:023)]).

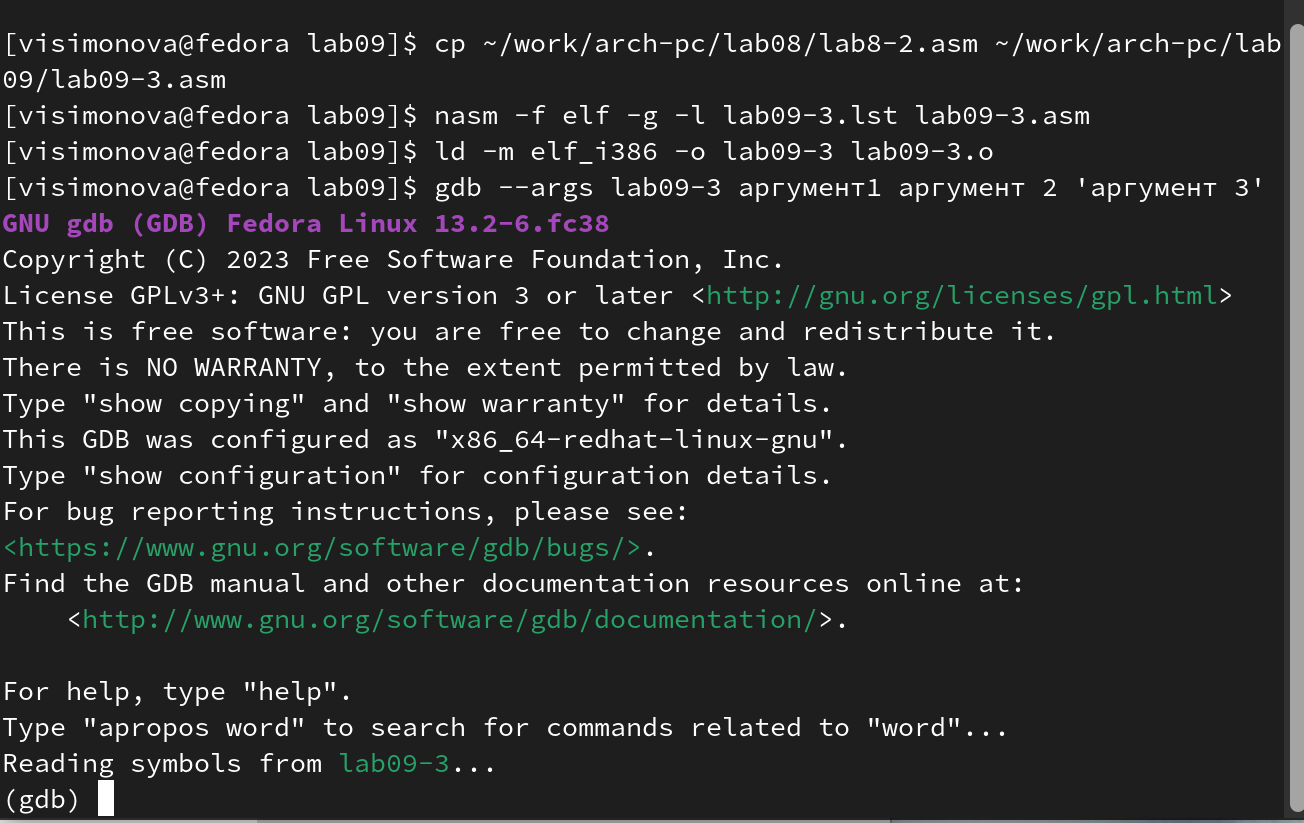


Figure 22: Копирую и загружаю в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю программу (рис. [[23](#fig:024)]).

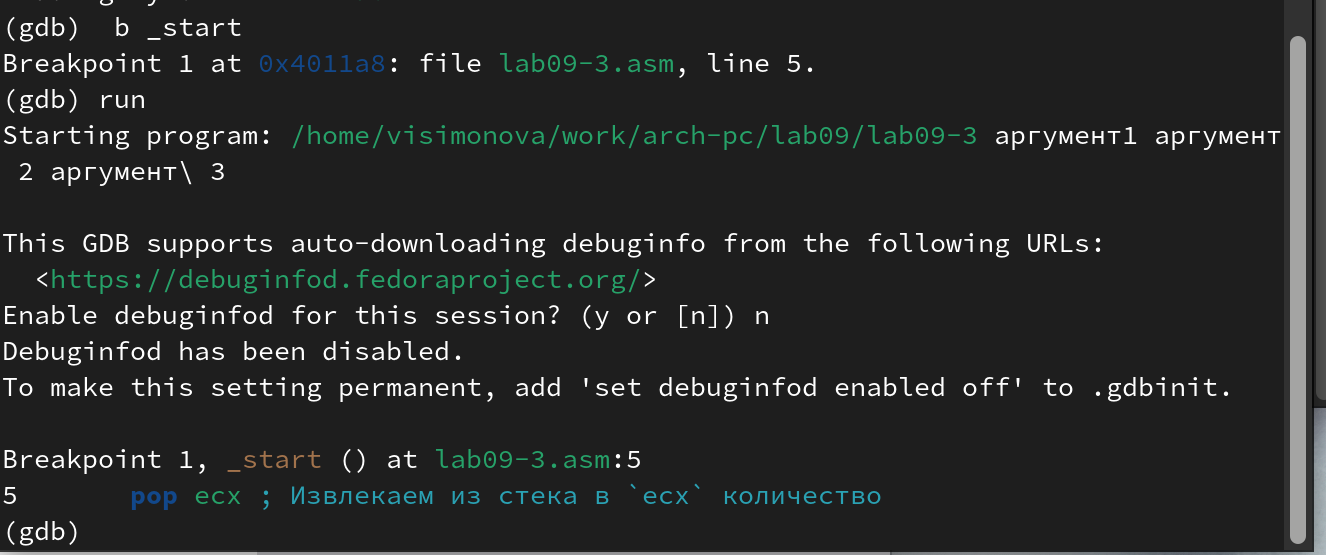


Figure 23: Запуск программы с точкой

Просматриваю позиции стека по адресам (рис. [[24](#fig:025)]).

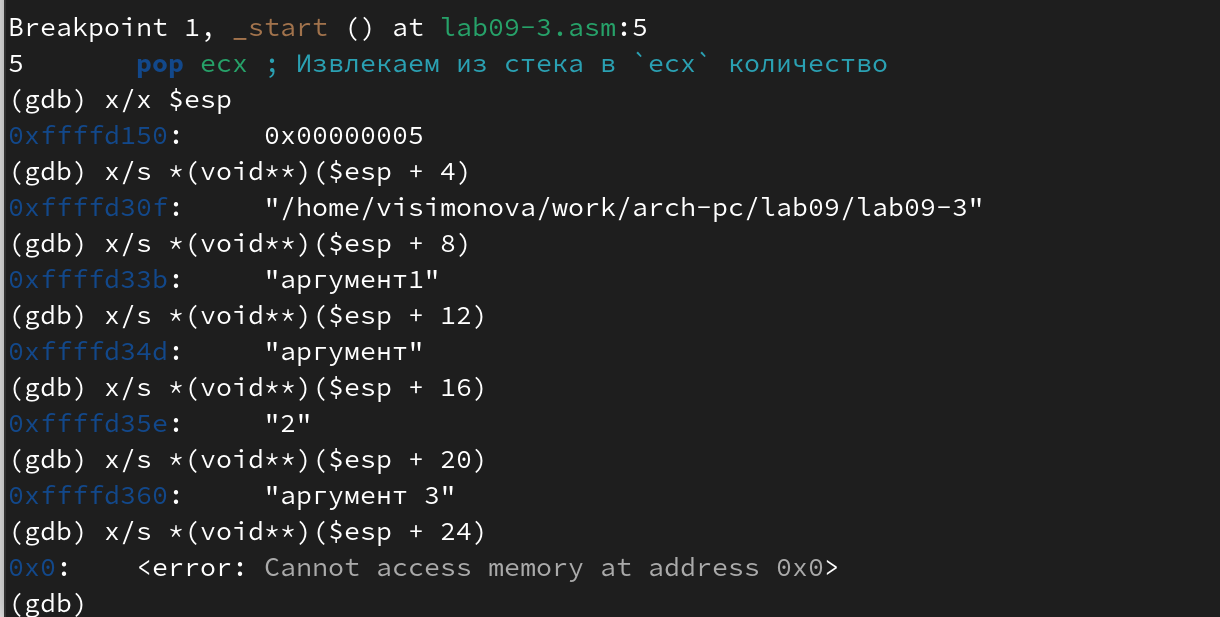


Figure 24: Просмотр значений

Аргументов командной строки 4, поэтому и шаг изменения адреса равен четырём.

## 4.6 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. [[25](#fig:026)]).

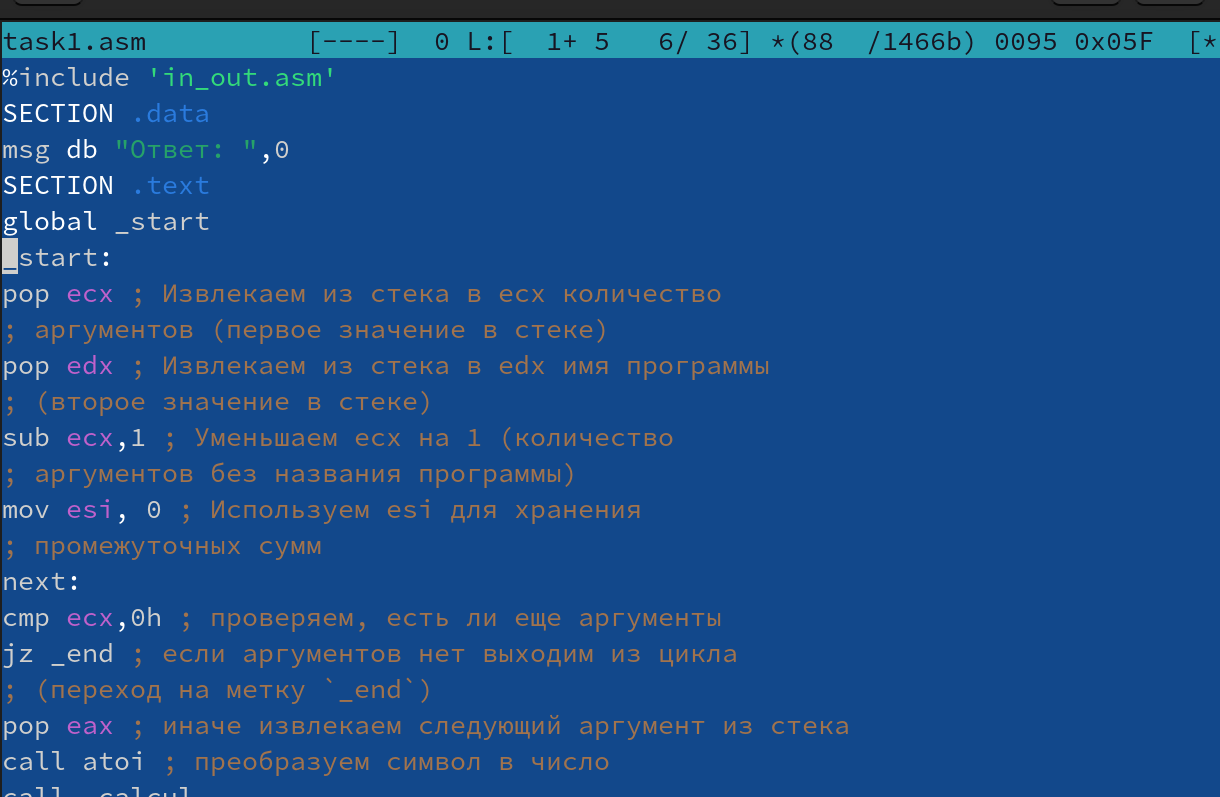


Figure 25: Ввожу код в файл

Создаю исполняемый файл и проверяю его работу (рис. [[26](#fig:027)]).

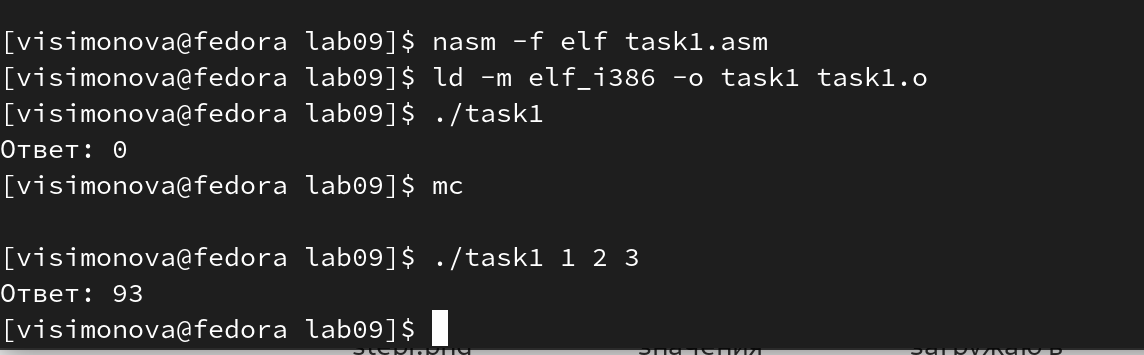


Figure 26: Запускаю исполняемый файл

Код первого задания:

%include 'in\_out.asm'  
SECTION .data  
msg db "Ответ: ",0  
SECTION .text  
global \_start  
\_start:  
pop ecx ; Извлекаем из стека в ecx количество  
; аргументов (первое значение в стеке)  
pop edx ; Извлекаем из стека в edx имя программы  
; (второе значение в стеке)  
sub ecx,1 ; Уменьшаем ecx на 1 (количество  
; аргументов без названия программы)  
mov esi, 0 ; Используем esi для хранения  
; промежуточных сумм  
next:  
cmp ecx,0h ; проверяем, есть ли еще аргументы  
jz \_end ; если аргументов нет выходим из цикла  
; (переход на метку `\_end`)  
pop eax ; иначе извлекаем следующий аргумент из стека  
call atoi ; преобразуем символ в число  
call \_calcul  
add esi,eax ; добавляем к промежуточной сумме  
; след. аргумент esi=esi+eax  
loop next ; переход к обработке следующего аргумента  
\_end:  
mov eax, msg ; вывод сообщения "Результат: "  
call sprint  
mov eax, esi ; записываем сумму в регистр eax  
call iprintLF ; печать результата  
call quit ; завершение программы  
\_calcul:  
mov ebx,12  
mul ebx  
sub eax,-7  
ret

1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) \* 4 + 5. (рис. [[27](#fig:028)]).

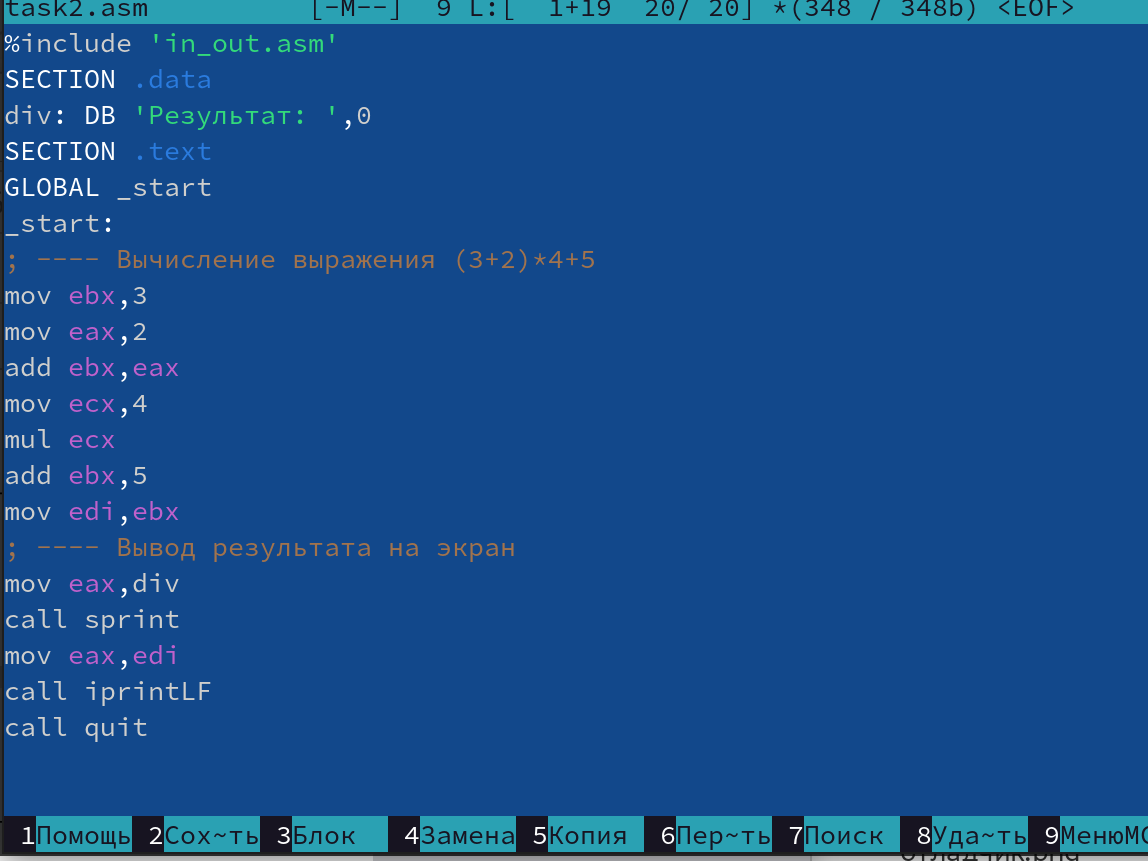


Figure 27: Исходный код

При запуске данная программа дает неверный результат. Проверяю это.(рис. [[28](#fig:029)]).

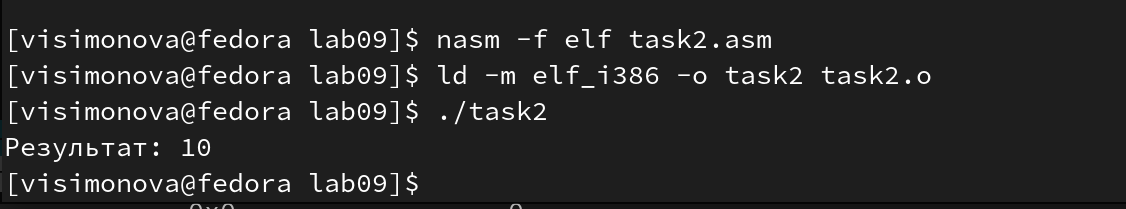


Figure 28: Запускаю исполняемый файл

С помощью отладчика GDB, анализируя изменения значений регистров, определяю ошибку и исправляю ее.

Передаю в отладчик (рис. [[29](#fig:030)]).

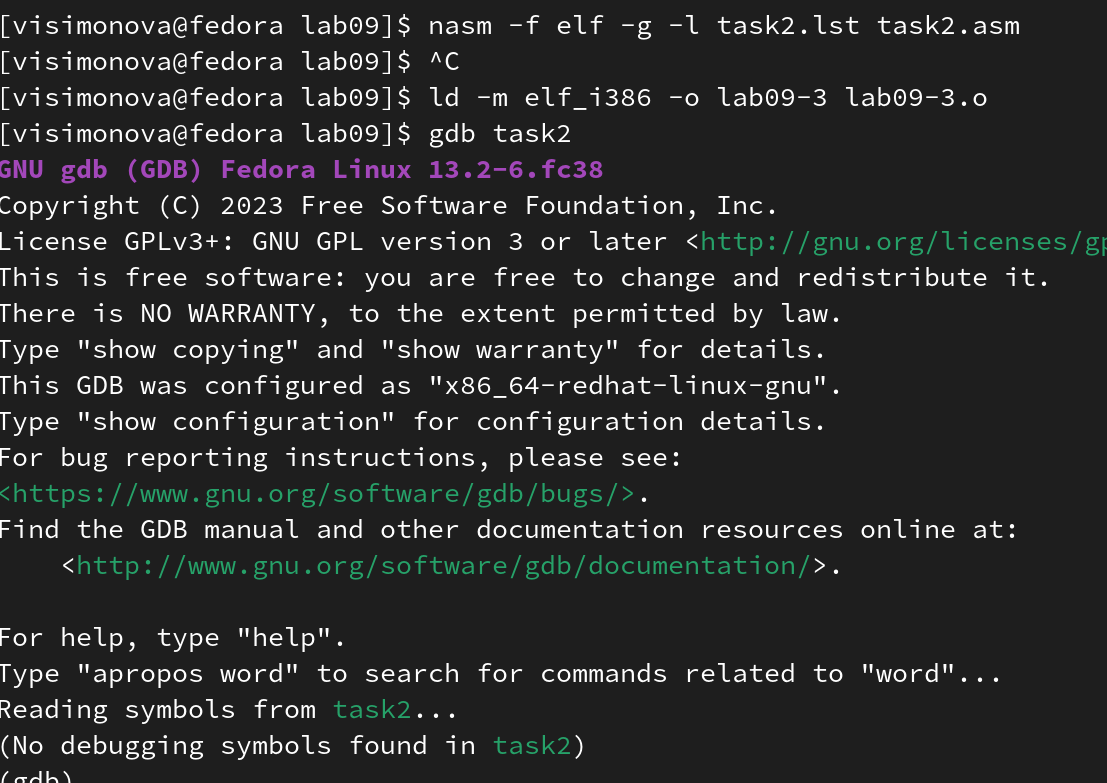


Figure 29: Передача

Ставлю брейкпоинты для всех вычислительных инструкций, прохожусь по ним. Ошибка: в момент выполнения инструкции mul ecx(4) на eax(2), а умножать надо на ebx(5).Нужно инструкцию add ebx,eax связать с mul ecx

Причина (рис. [[30](#fig:031)]).

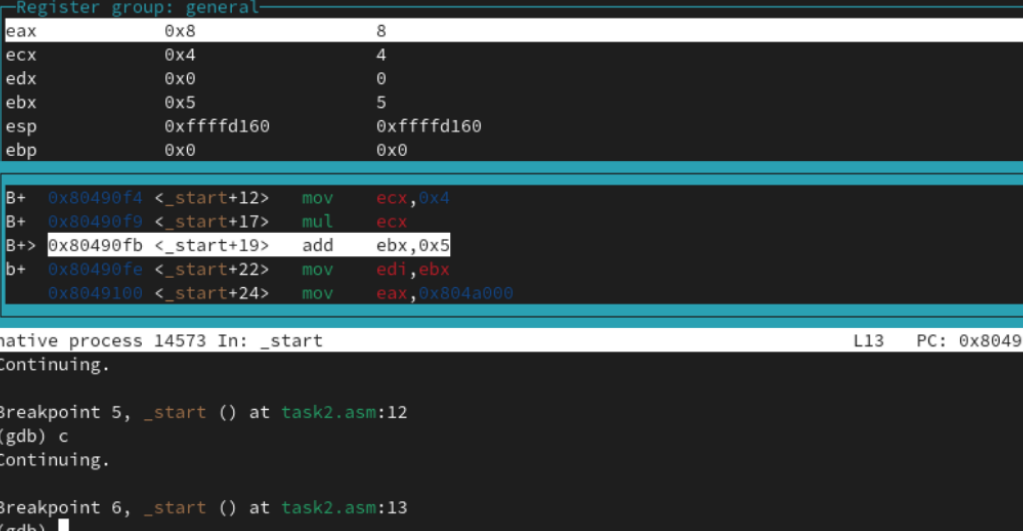


Figure 30: Передача

Значение регистра (рис. [[31](#fig:032)]).



Figure 31: Неверное изменение

Ввожу исправленную программу в файл (рис. [[32](#fig:033)]).

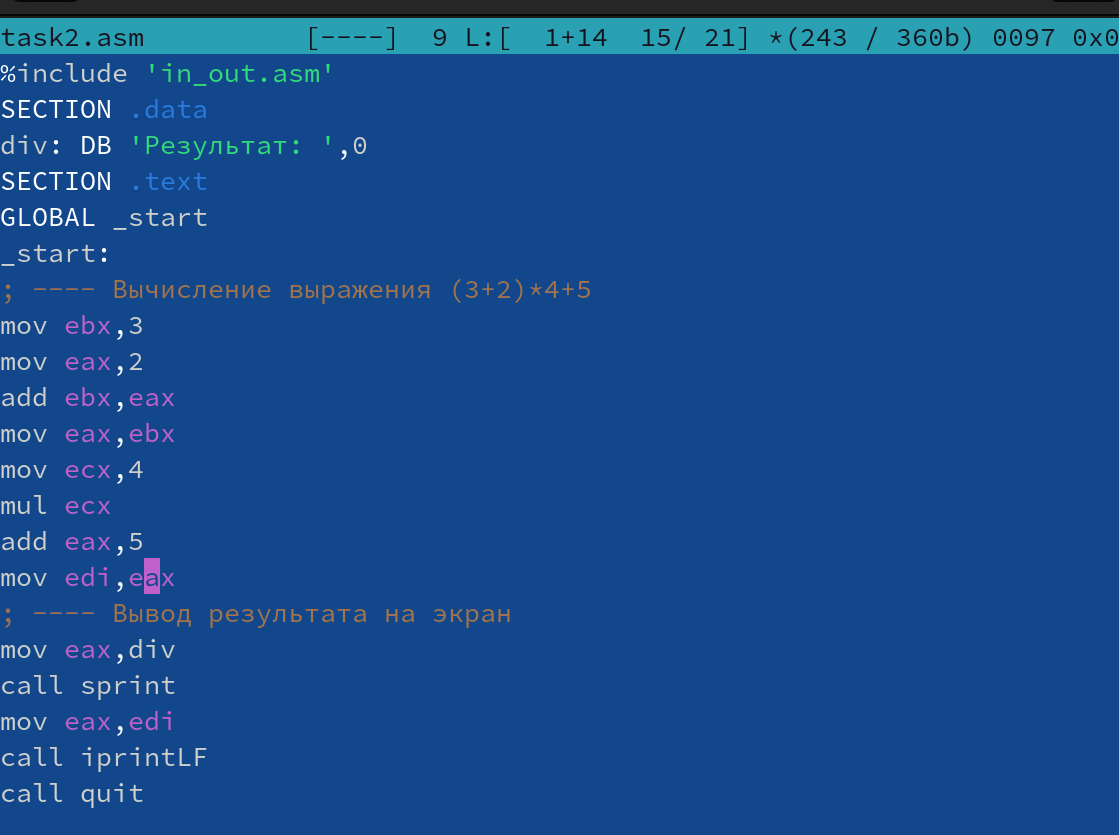


Figure 32: Исправленная программа

Создаю исполняемый файл и проверяю его работу , всё работает корректно(рис. [[33](#fig:034)]).

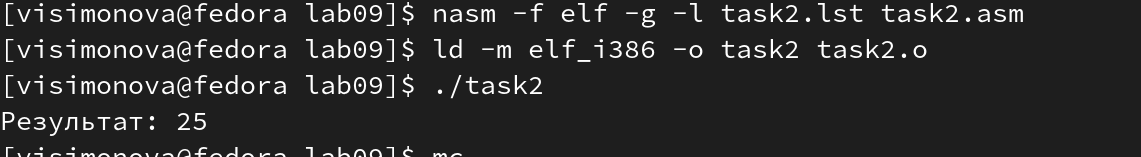


Figure 33: Запускаю исполняемый файл

Код программы задания 2:

%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 eax,5  
mov edi,eax  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

# 5 Выводы

Приобрела навыки написания программ с использованием подрограмм, познакомилась с методами отладки припомощи GBD и его основными возможностями.

# Список литературы

::: {#refs} ::