Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Петрова Алевтина Александровна

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

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения работы №9 (рис. [1](#fig:001)).

Figure 1: Создание каталога

Figure 1: Создание каталога

Перехожу в созданную директорию (рис. [2](#fig:002)).

Figure 2: Перемещение по директории

Figure 2: Перемещение по директории

Создаю файл lab09-1.asm в новом каталоге (рис. [3](#fig:003)).

Figure 3: Создание файла

Figure 3: Создание файла

Копирую файл in\_out.asm в созданный каталог, так как он понадобится для написания программ (рис. [4](#fig:004)).

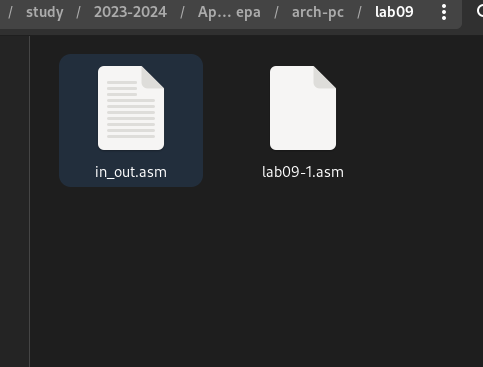


Figure 4: Копирование файла

Открываю файл в текстовом редакторе и переписываю код программы из листинга 9.1 (рис. [5](#fig:005)).

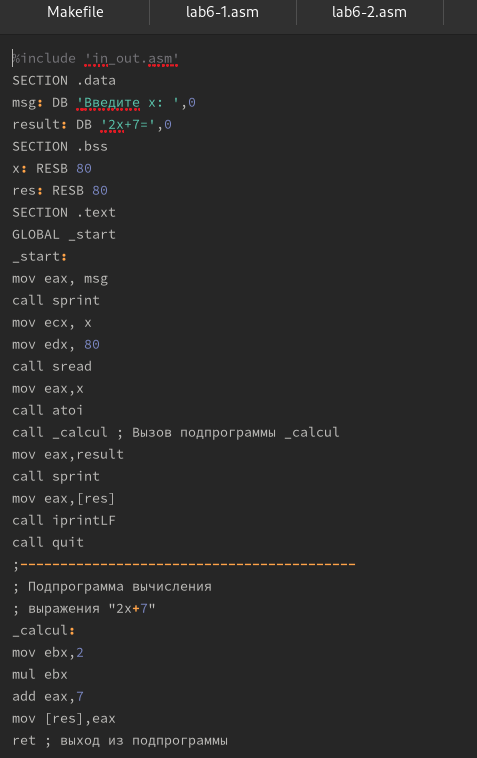


Figure 5: Редактирование файла

Создаю объектный файл программы и после компановки запускаю его (рис. [6](#fig:006)). Код с подпрограммой работает успешно.

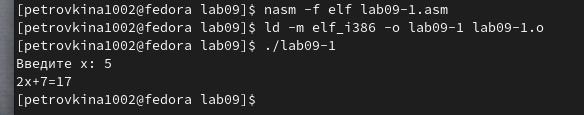


Figure 6: Запуск программы

Изменяю текст файла,добавив подпрограмму sub\_calcul в подпрограмму \_calcul (рис. [7](#fig:007)).

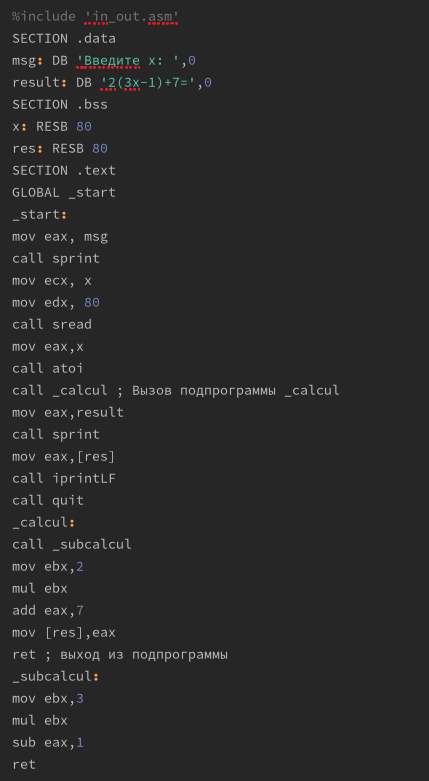


Figure 7: Редактирование файла

%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 ; Вызов подпрограммы \_calcul  
mov eax,result  
call sprint  
mov eax,[res]  
call iprintLF  
call quit  
\_calcul:  
call \_subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы  
\_subcalcul:  
mov ebx,3  
mul ebx  
sub eax,1  
ret

Запускаю исполняемый файл (рис. [8](#fig:008)).Программа работает верно.

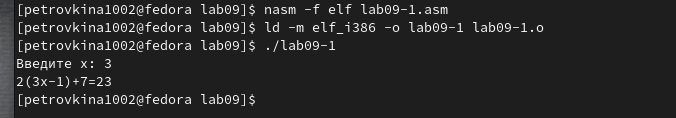


Figure 8: Запуск программы

## 3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm, используя команду touch (рис. [9](#fig:009)).

Figure 9: Создание файла

Figure 9: Создание файла

Записываю код программы из листинга 9.2,который выводит сообщение Hello world (рис. [10](#fig:010)).

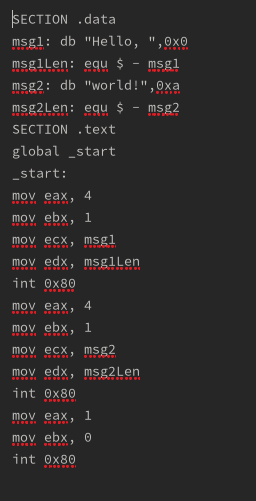


Figure 10: Редактирование файла

Получаю исполняемый файл. Для работы с GDB провожу трансляцию программ с ключом “-g” и загружаю исполняемый файл в отладчик (рис. [11](#fig:011)).

Figure 11: Запуск исполняемого файла

Figure 11: Запуск исполняемого файла

Проверяю работу программы в оболочке GDB с помощью команды run (рис. [12](#fig:012)).

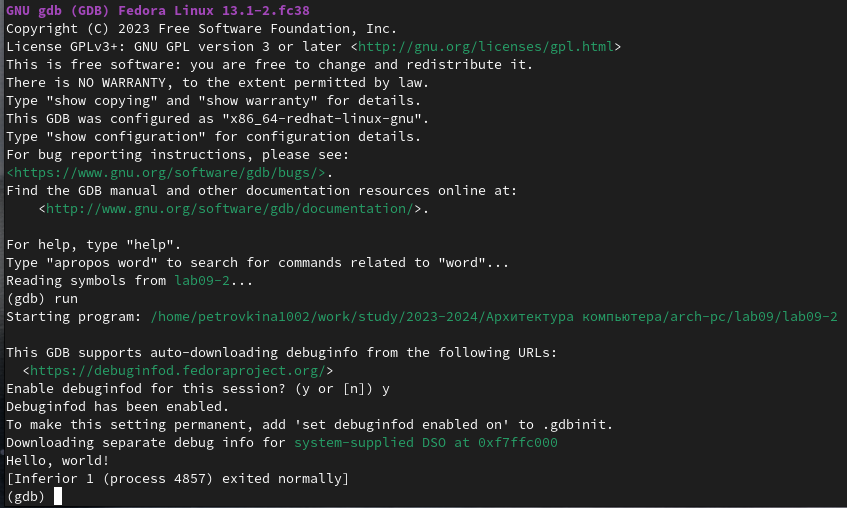


Figure 12: Запуск программы в отладчике

Для более подробного анализа устанавливаю брейкпоинт на метку \_start,с которой начинается выполнение ассемблерной программы (рис. [13](#fig:013)).

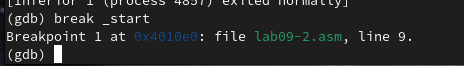


Figure 13: Установка брейкпоинта

Запускаю её (рис. [14](#fig:014)).

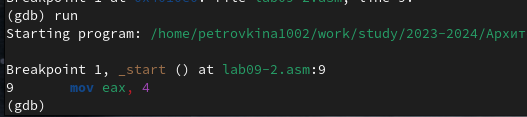


Figure 14: Запуск

С помощью команды “disassemble \_start” просматриваю дисассимилированный код программы (рис. [15](#fig:015)).

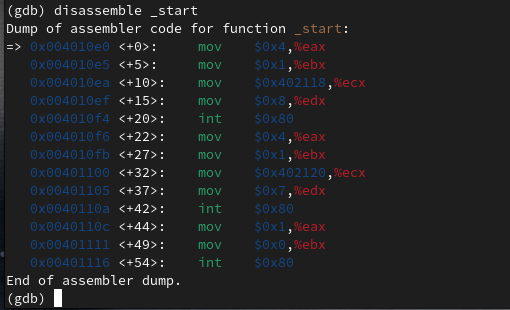


Figure 15: Диссассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду “set disassembly-flavor intel” (рис. [16](#fig:016)).

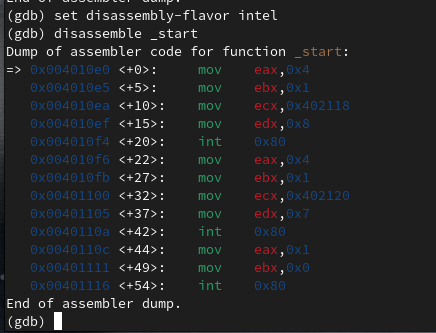


Figure 16: Отображение с Intel’овским синтаксисом

Основное различие заключается в том,что в режиме Intel пишется сначала сама команда, а потом её машинный код, в то время как в режиме ATT идет сначала машинный код,а только потом сама команда.

## 3.3 Добавление точек останова

Проверяю наличие точки останова с помощью команды info breakpoints (i b) (рис. [17](#fig:017)).

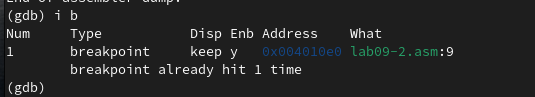


Figure 17: Точка останова

Устанавливаю ещё одну точку останова по адресу инструкции,которую можно найти в средней части в левом столбце соответствующей инструкции (рис. [18](#fig:018)).

Figure 18: Установка точки останова

Figure 18: Установка точки останова

Просматриваю информацию о точках останова (рис. [19](#fig:019)).

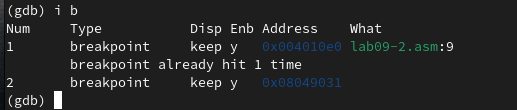


Figure 19: Точки останова

## 3.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info registers (рис. [20](#fig:020)).

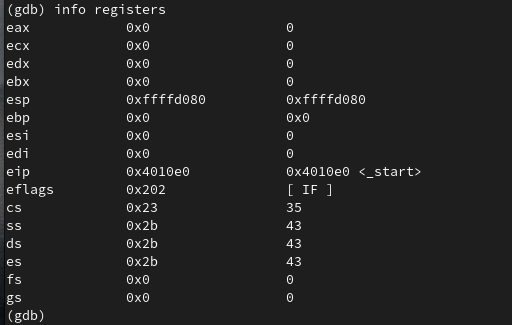


Figure 20: info register

Узнаю значение переменной msg1 по имени (рис. [21](#fig:021)).

Figure 21: Значение переменной по имени

Figure 21: Значение переменной по имени

Просматриваю значение переменной msg2 по адресу,который можно определить по дизассемблированной инструкции (рис. [22](#fig:022)).

Figure 22: Значение переменной по адресу

Figure 22: Значение переменной по адресу

Меняю первый символ переменной msg1 (рис. [23](#fig:023)).

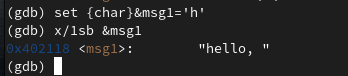


Figure 23: Изменение переменной

Также меняю первый символ переменной msg2 (рис. [24](#fig:024)).

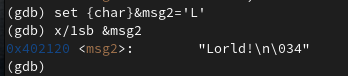


Figure 24: Изменение второй переменной

Вывожу значение регистра edx в различных форматах (в шестнадцатеричном, двоичном и символьном форматах) (рис. [25](#fig:025)).

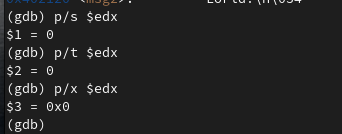


Figure 25: Изменение значений в разные форматы

С помощью команды set изменяю значение регистра ebx (рис. [26](#fig:026)).

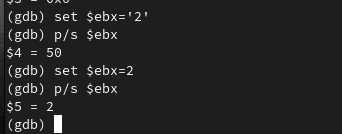


Figure 26: Изменение значений ebx

Значение регистра отличаются,так как в первом случае мы выводим код символа 2, который в десятичной системе счисления равен 50,а во втором случае выводится число 2,представленное в этой же системе.

## 3.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm,созданный при выполнении лабораторной работы №8, который выводит на экран аргументы, в файл с именем lab09-3.asm (рис. [27](#fig:027)).

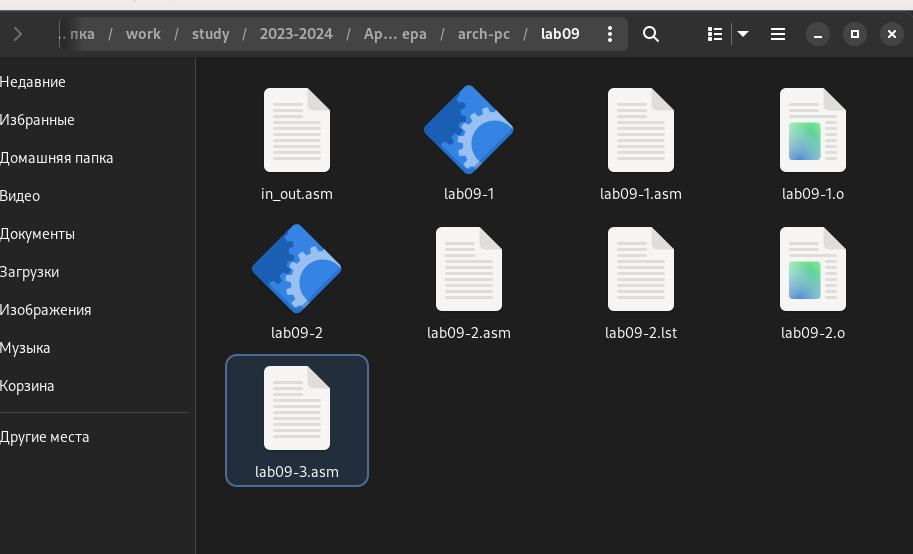


Figure 27: Копирование файла

Создаю исполняемый файл,использую ключ –args для загрузки программы в GDB. Загружаю исполняемый файл,указав аргументы (рис. [28](#fig:028)).

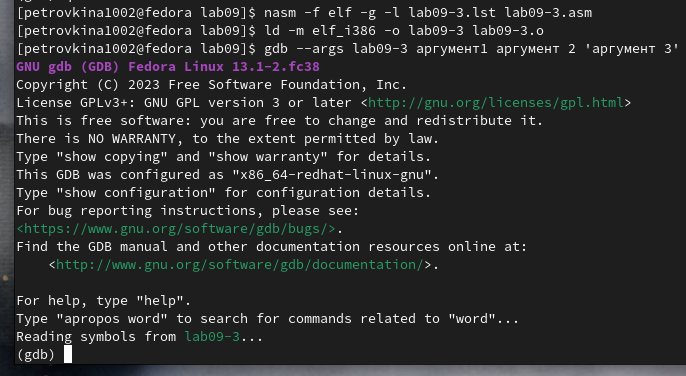


Figure 28: Создание файла

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. [29](#fig:029)).

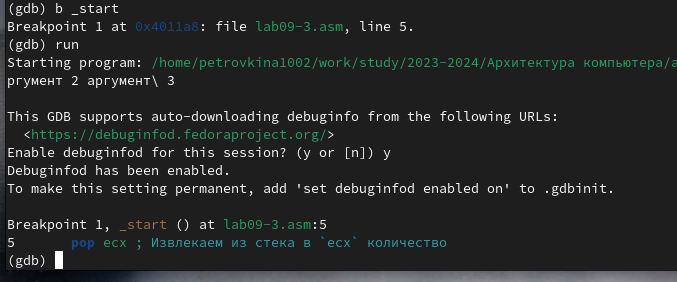


Figure 29: Запуск программы с точкой останова

Просматриваю адрес вершины стека,который хранится в регистре esp (рис. [30](#fig:030)).

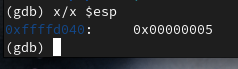


Figure 30: Регистр esp

Ввожу другие позиции стека- в отличие от адресов, располагается адрес в памяти: имя, первый аргумент, второй и т.д (рис. [31](#fig:031)).

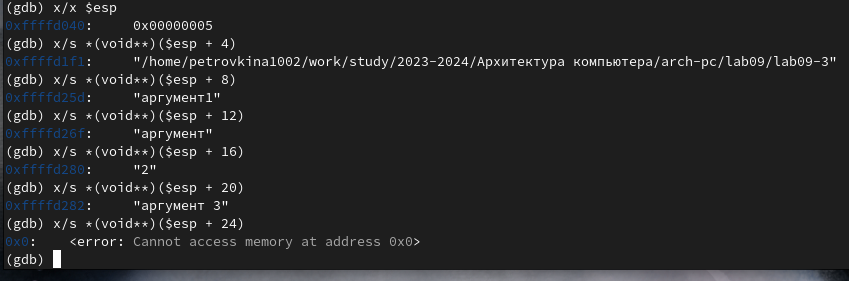


Figure 31: Позиции стека

Количество аргументов командной строки 4,следовательно и шаг равен четырем.

## 3.6 Задание для самостоятельной работы

Создаю файл для первого самостоятельного задания,который будет называться lab09-4.asm (рис. [32](#fig:032)).

Figure 32: Создание файла

Figure 32: Создание файла

Редактирую код программы lab8-4.asm,добавив подпрограмму,которая вычисляет значения функции f(x) (рис. [33](#fig:033)).

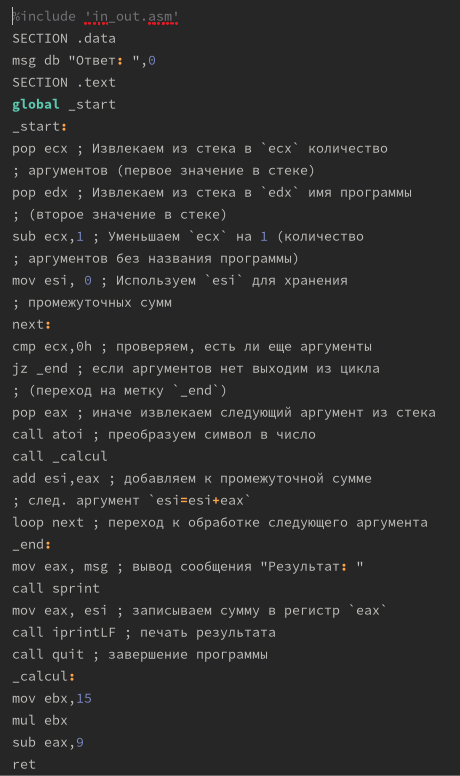


Figure 33: Редактирование файла

%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,15  
mul ebx  
sub eax,9  
ret

Создаю исполняемый файл и ввожу аргументы (рис. [34](#fig:034)). Программа работает верно.

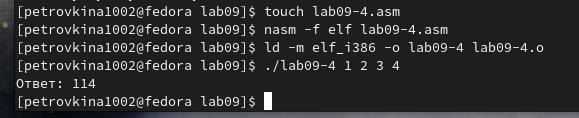


Figure 34: Запуск программы

Создаю файл и ввожу код из листинга 9.3 (рис. [35](#fig:035)).

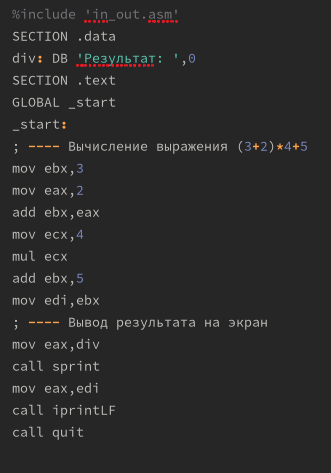


Figure 35: Редактирование файла

Открываю файл в отладчике GDB и запускаю программу (рис. [36](#fig:036)). Программа выдает ответ 10.

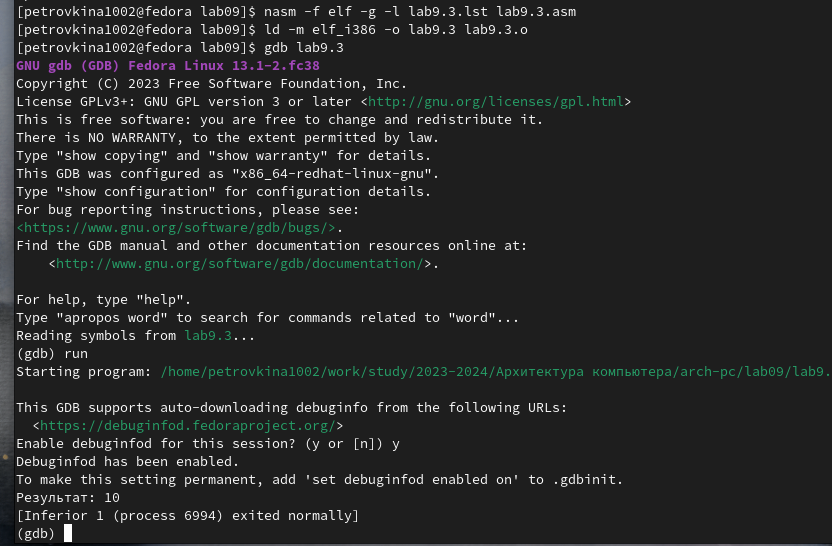


Figure 36: Запуск программы в отладчике

Просматриваю дисассимилированный код программы, ставлю точку останова перед прибавлением 5 и открываю значения регистров на данном этапе (рис. [37](#fig:037)).

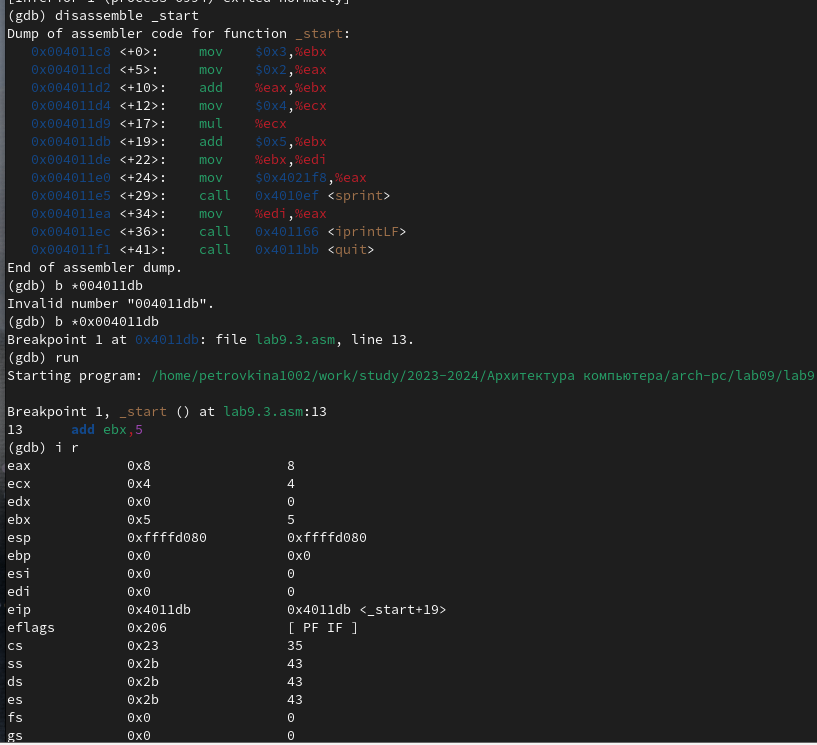


Figure 37: Действия в отладчике

Как можно увидеть, регистр ecx со значением 4 умножается не на ebx, сложенным c eax, а только с eax со значением 2. Значит нужно поменять значения регистров(например присвоить eax значение 3 и просто прибавит 2. После изменений программа будет выглядить следующим образом:

%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

Пробуем запустить программу (рис. [38](#fig:038)). Она работает верно.

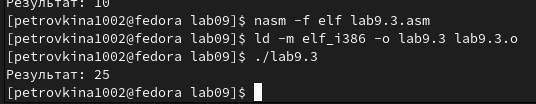


Figure 38: Запуск программы

# 4 Выводы

В данной работе я приобрел навыки написания программ с подпрограммами и познакомился с методами отладки при помощи GDB.

# Список литературы

Лабораторная работа №9