Отчет по лабораторной работе №10

Дисциплина: Архитектура компьютера

Ищенко Ирина Олеговна

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы № 10, перейдем в него и создадим файл lab10-1.asm (рис. 1).

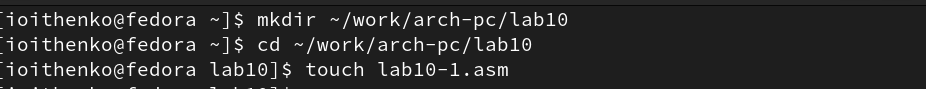


Рис. 1: Создание каталога и файла

Введем в файл lab10-1.asm текст программы из листинга 1. Создадим исполняемый файл и проверим его работу (рис. 2). Листинг 2:

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2x+7=',0  
SECTION .bss  
x: RESB 80  
rezs: 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 [rez],eax  
ret ; выход из подпрограммы

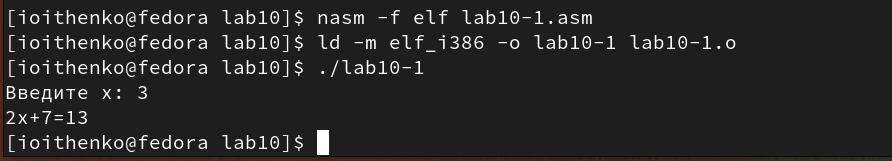


Рис. 2: Запуск программы вычисления выражения

Изменим текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, согласно листингу 2. Создадим исполняемый файл и проверим его работу (рис. 3). Листинг 2:

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2(3x-1)+7=',0  
  
SECTION .bss  
x: RESB 80  
rez: 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,[rez]  
call iprintLF  
call quit  
  
\_calcul:  
call \_subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [rez],eax  
ret ; выход из подпрограммы  
  
\_subcalcul:  
mov ebx,3  
mul ebx  
sub eax,1  
ret

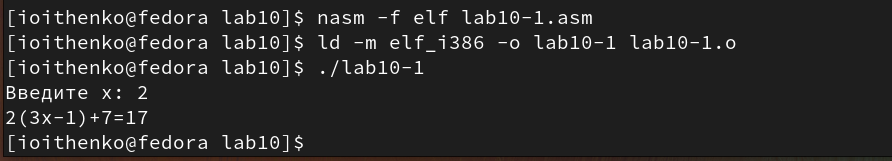


Рис. 3: Запуск программы вычисления выражения с использованием подпрограммы

Создадим файл lab10-2.asm с текстом программы из листинга 3.Получим исполняемый файл. Загрузим исполняемый файл в отладчик gdb (рис. 4). Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 5). Листинг 3:

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

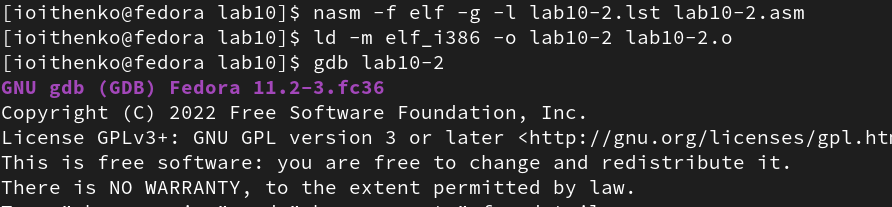


Рис. 4: Программа печати сообщения Hello world!

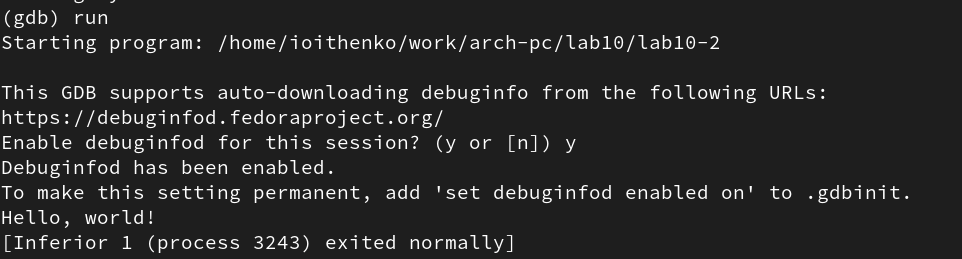


Рис. 5: Программа печати сообщения Hello world!

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 6).

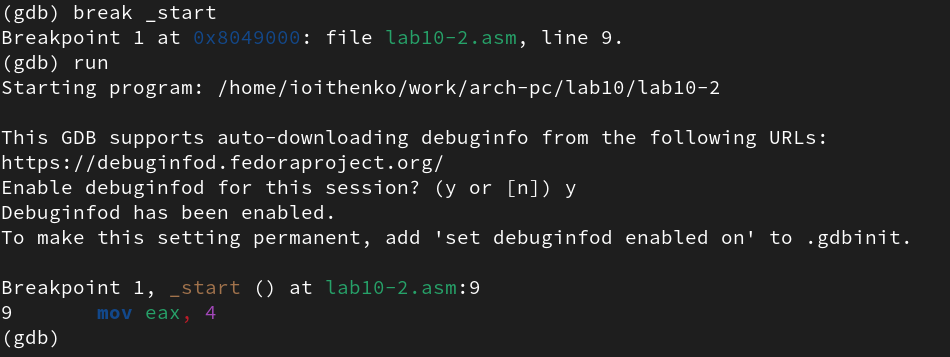


Рис. 6: Установка брейкпоинта

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 7).

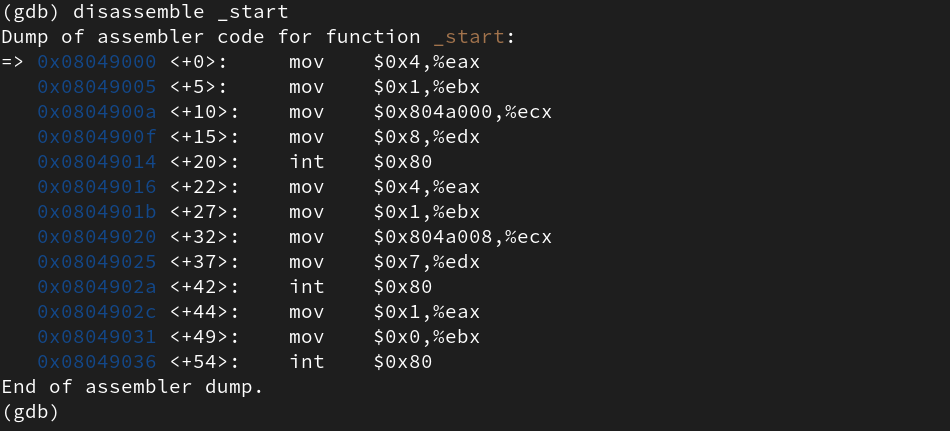


Рис. 7: Дисассимилированный код программы

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. Код Intel опускает символ ‘%’ перед именами регистров, инструкции с несколькими операндами перечисляют их в обратном порядке.

Включим режим псевдографики для более удобного анализа программы (рис. 8).

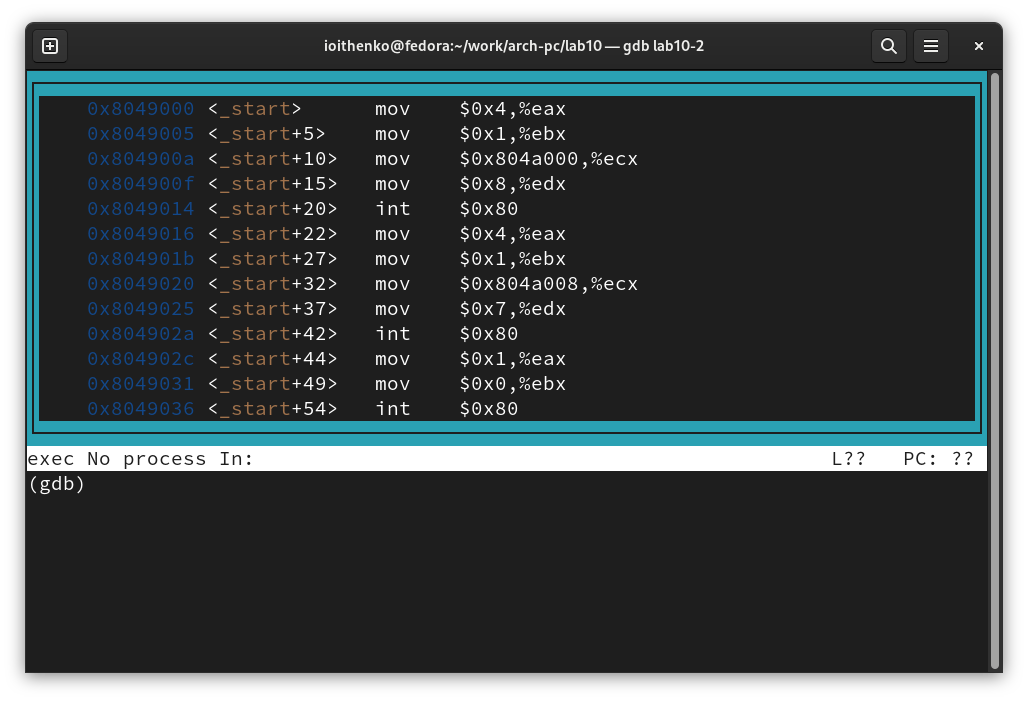


Рис. 8: Режим псевдографики

Проверим наличие брейкпоинтов с помощью команды info breakpoints и установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова. Посмотрим информацию о всех установленных точках останова (рис. 9).

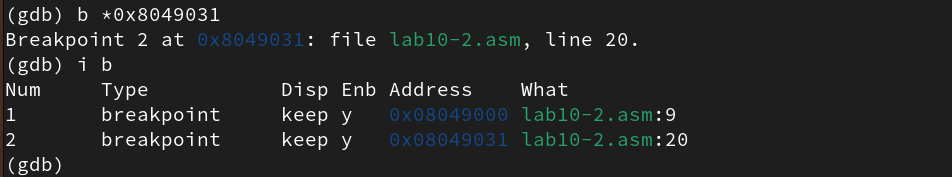


Рис. 9: Брейкпоинты

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды stepi (или si) и проследите за изменением значений регистров (рис. 10), (рис. 11), (рис. 12), (рис. 13) и (рис. 14). Команда si делает шаг от брейкпоинта и показывает значение регистра согласно строке.

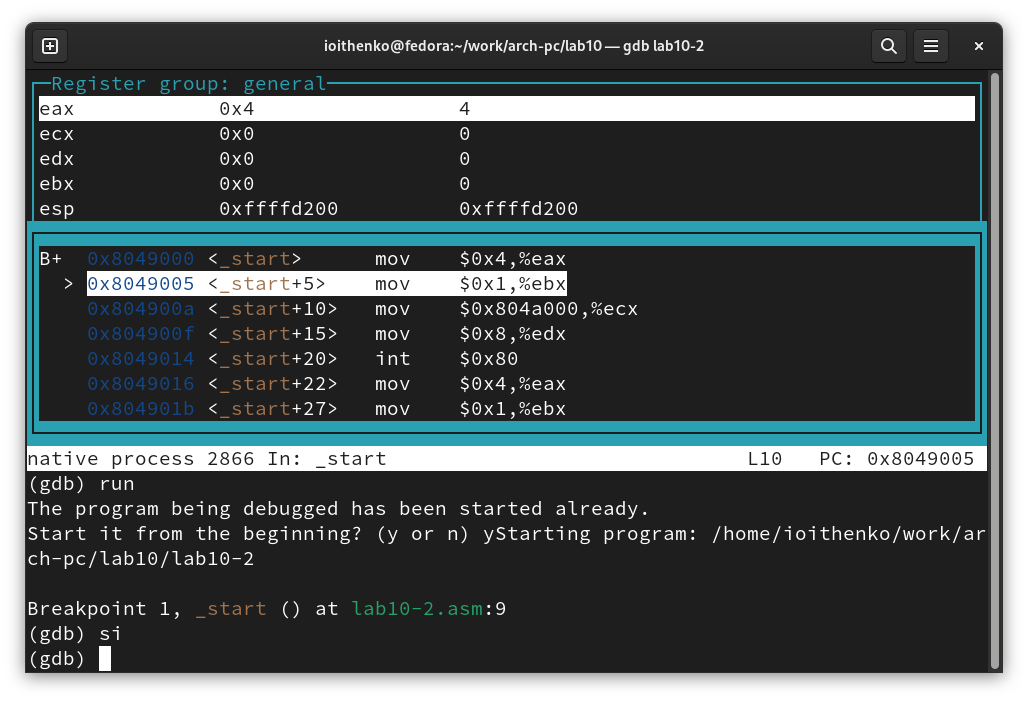


Рис. 10: Команда si

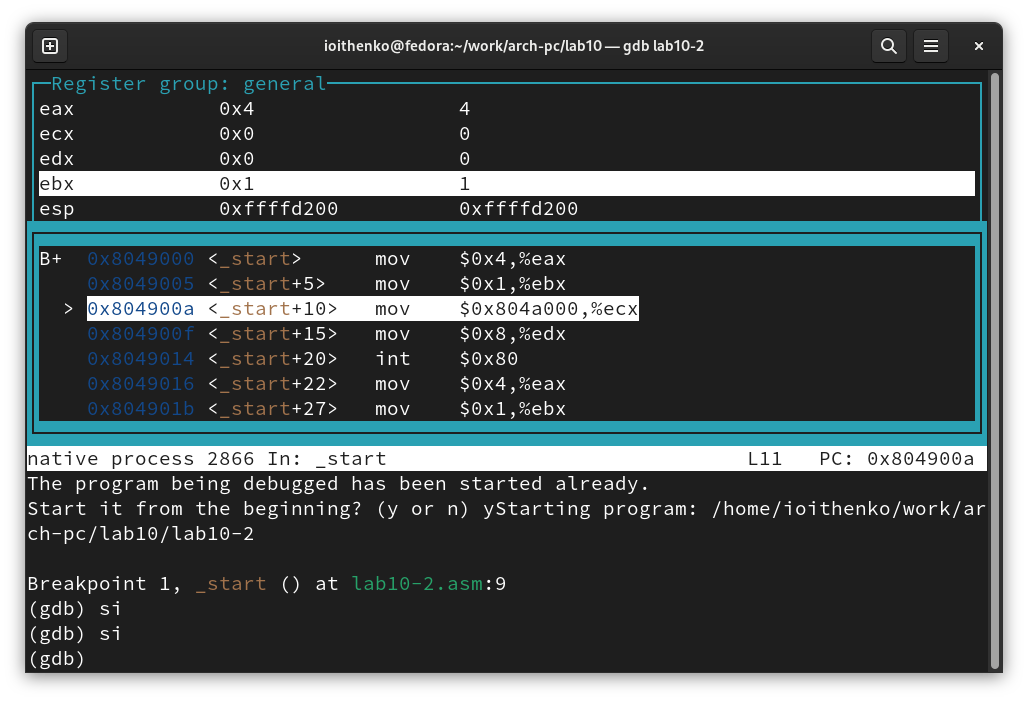


Рис. 11: Команда si

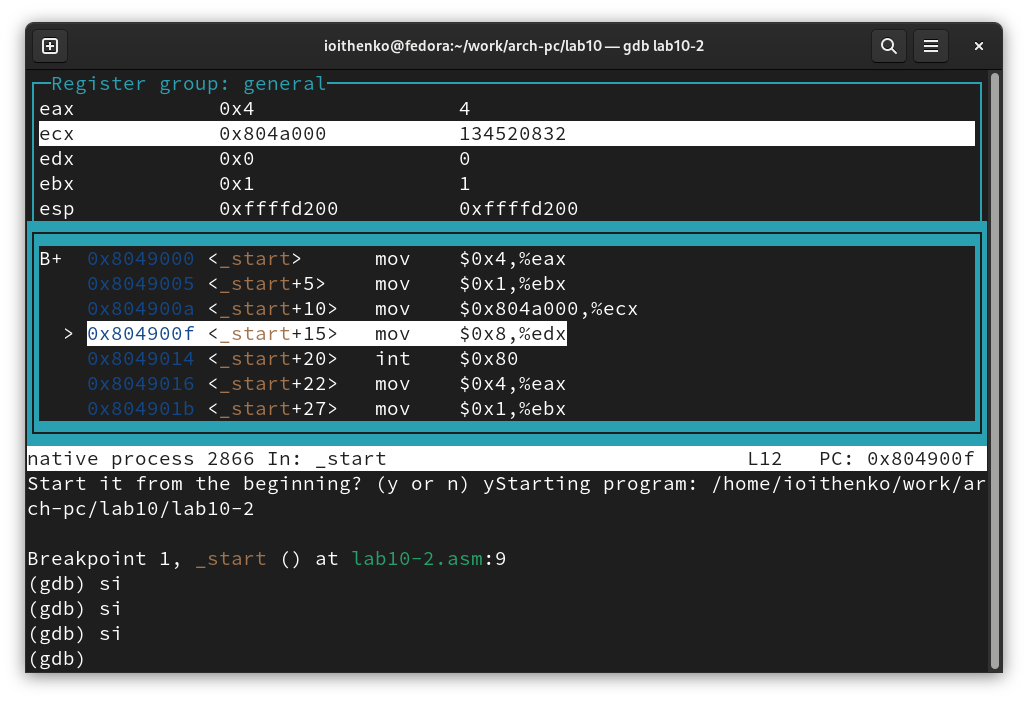


Рис. 12: Команда si

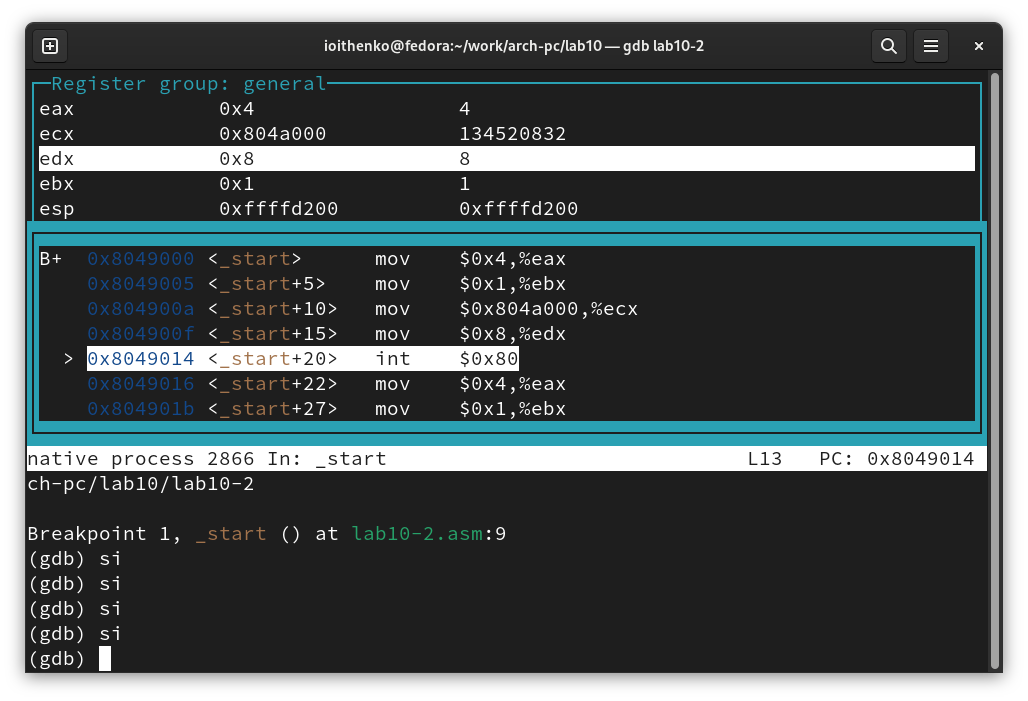


Рис. 13: Команда si

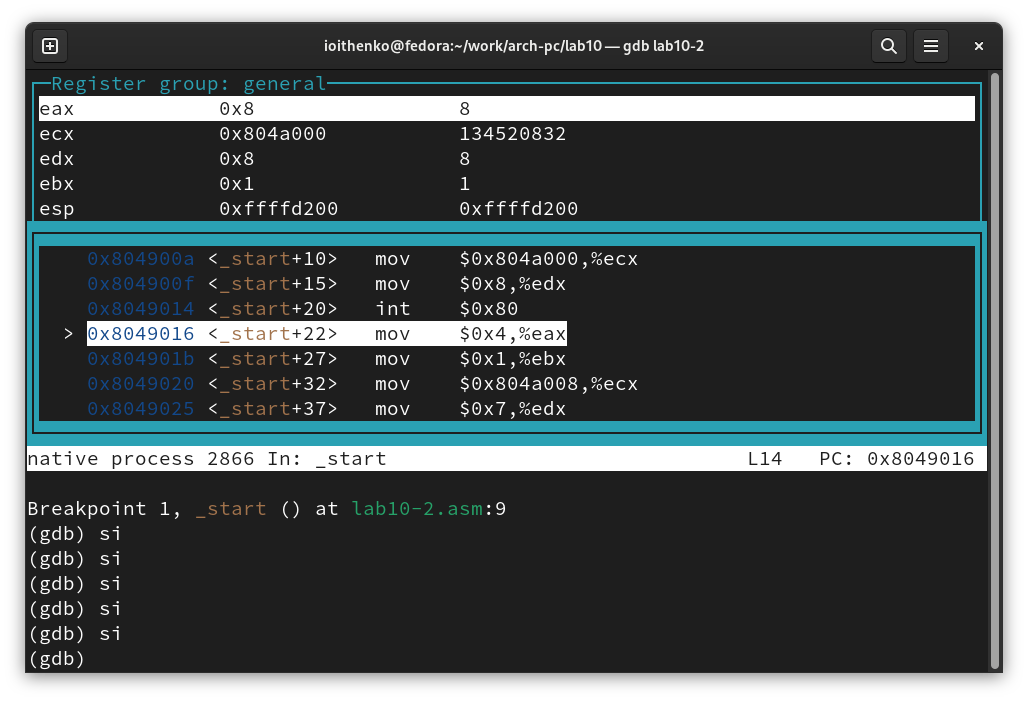


Рис. 14: Команда si

Посмотрим значение переменной msg1 по имени и значение переменной msg2 по адресу (рис. 15).

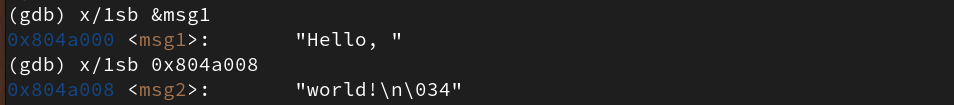


Рис. 15: Значения переменных

Изменим первый символ переменной msg1. Заменим любой символ во второй переменной msg2 (рис. 16).

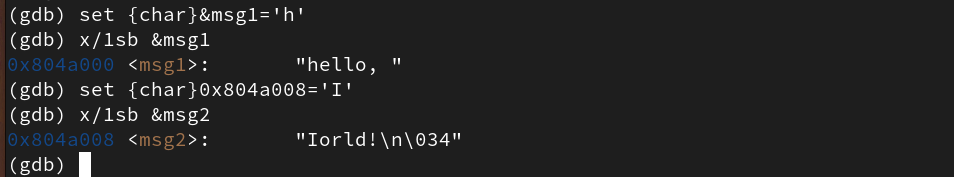


Рис. 16: Изменение значений переменных

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 17).



Рис. 17: Значение регистра в разных форматах

С помощью команды set изменим значение регистра ebx (рис. 18). Разница вывода команда заключается в том, что в первом случае мы вводим 2 как символ.



Рис. 18: Значение регистра ebx

Завершим выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдем из GDB с помощью команды quit (сокращенно q).

Скопируем файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки в файл с именем lab10-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 19).

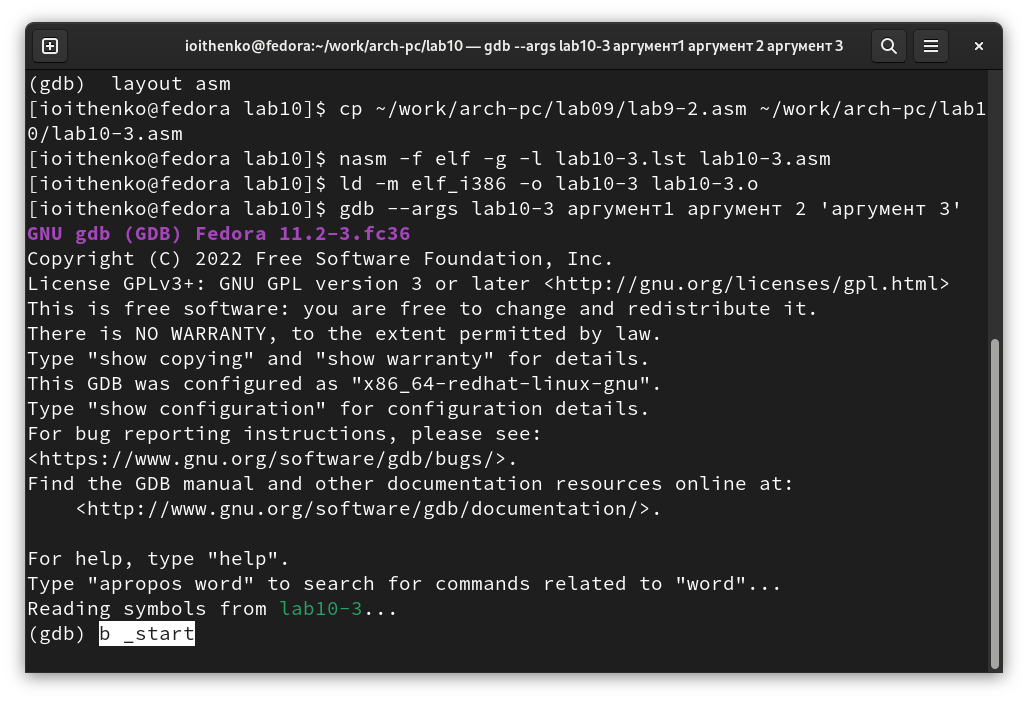


Рис. 19: Программа вывода аргументов

Установим точку брейкпоинта. Посмотрим позиции стека (рис. 20). Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.), так как шаг равен размеру переменной - 4 байтам.

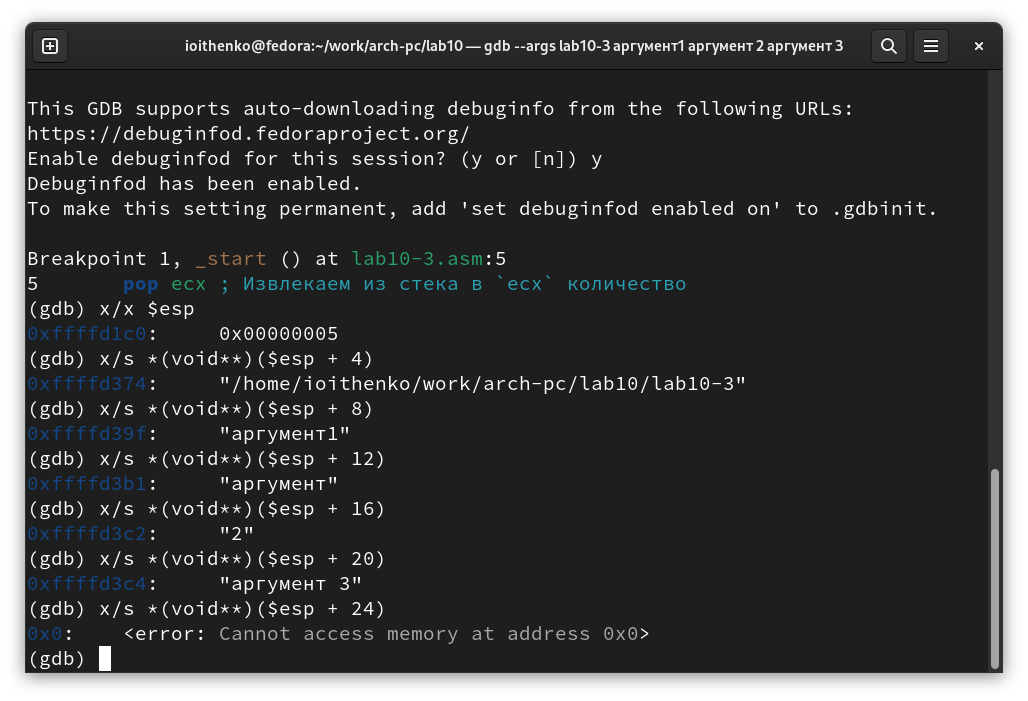


Рис. 20: Позиции стека

# 3 Выполнение заданий для самостоятельной работы

Преобразуем программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции как подпрограмму, согласно листингу 4 (рис. 21). Листинг 4:

%include 'in\_out.asm'  
SECTION .data  
msg db "Результат: ",0  
fx: db 'f(x)=5(x+2) ',0  
  
SECTION .text  
global \_start  
\_start:  
mov eax, fx  
call sprintLF  
pop ecx   
pop edx  
sub ecx,1  
mov esi, 0  
  
next:  
cmp ecx,0h  
jz \_end   
pop eax  
call atoi  
call calc  
add esi,eax  
  
loop next  
  
\_end:  
mov eax, msg  
call sprint  
mov eax, esi  
call iprintLF  
call quit  
  
calc:  
add eax,2  
mov ebx,5   
mul ebx  
ret

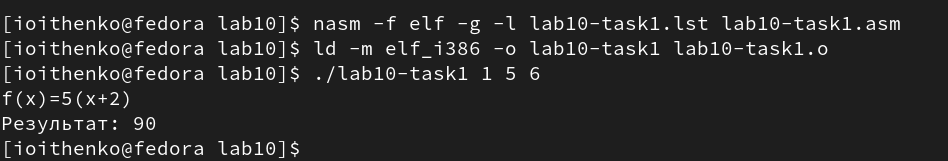


Рис. 21: Программа вычисления значения функции

В листинге 5 приведена программа вычисления выражения (3+2)\*4+5. При запуске данная программа дает неверный результат. Проверим это (рис. 22). С помощью отладчика GDB, анализируя изменения значений регистров (рис. 23), определим ошибку: перепутан порядок аргументов у инструкции add и по окончании работы в edi отправляется регистр ebx вместо eax. Исправим ее согласно листингу 6 (рис. 24). Листинг 5:

%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 ecx,4  
mul ecx  
add ebx,5  
mov edi,ebx  
; ---- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

Рис. 22: Неправильный вывод программы

Рис. 22: Неправильный вывод программы

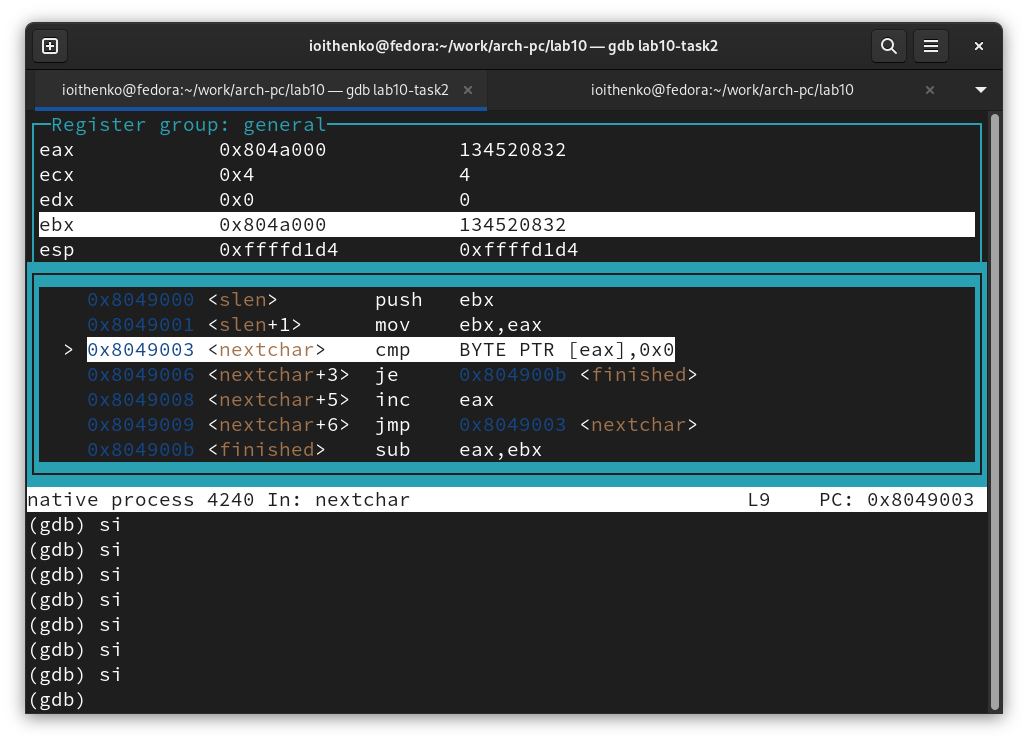


Рис. 23: Поиск ошибки

Листинг 6:

%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 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

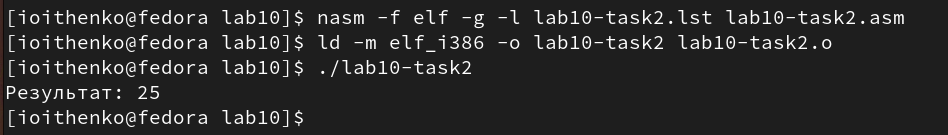


Рис. 24: Правильный вывод

# 4 Выводы

В ходе лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.