Лабораторная Работа 10

Понятие подпрограммы. Отладчик GDB.

Герра Гарсия Максимиано
Антонио

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Изучите примеры реализации подпрограмм
2. Изучите работу с отладчиком GDB
3. Выполните самостоятельное задание
4. Загрузите файлы на GitHub.

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.

4 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 10, перейдите в него и создайте файл lab10-1.asm:

2. В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x+7 с помощью подпрограммы calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 10.1). (рис. 4.1, 4.2)

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rez: 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,[rez]
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. 4.1: Файл lab10-1.asm

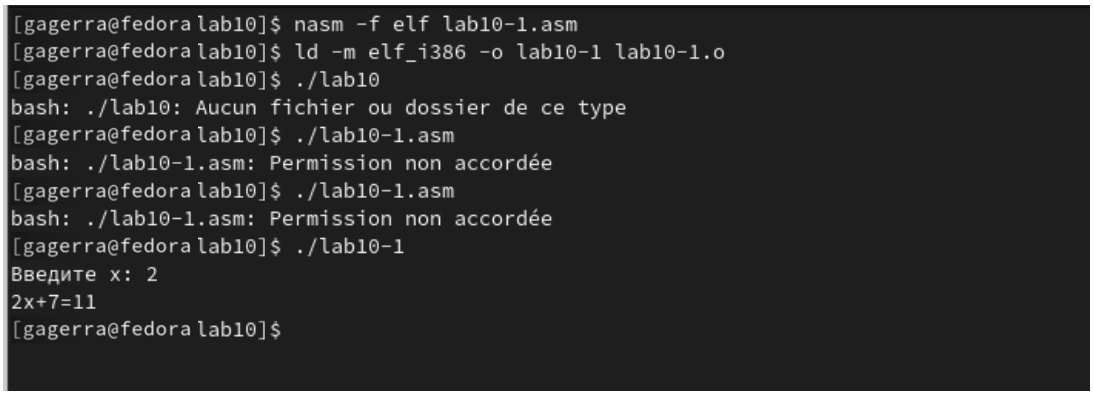


Рис. 4.2: Работа программы lab10-1.asm

3. Измените 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. 4.3, 4.4)

```
%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. 4.3: Файл lab10-1.asm

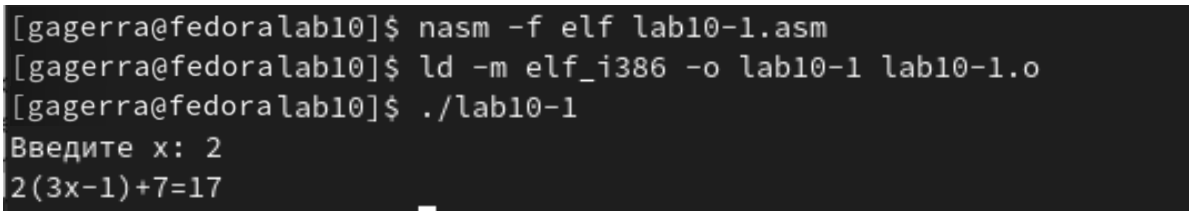


Рис. 4.4: Работа программы lab10-1.asm

4. Создайте файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 4.5)

```
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. 4.5: Файл lab10-2.asm

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Загрузите исполняемый файл в отладчик gdb: Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r):(рис. 4.6)

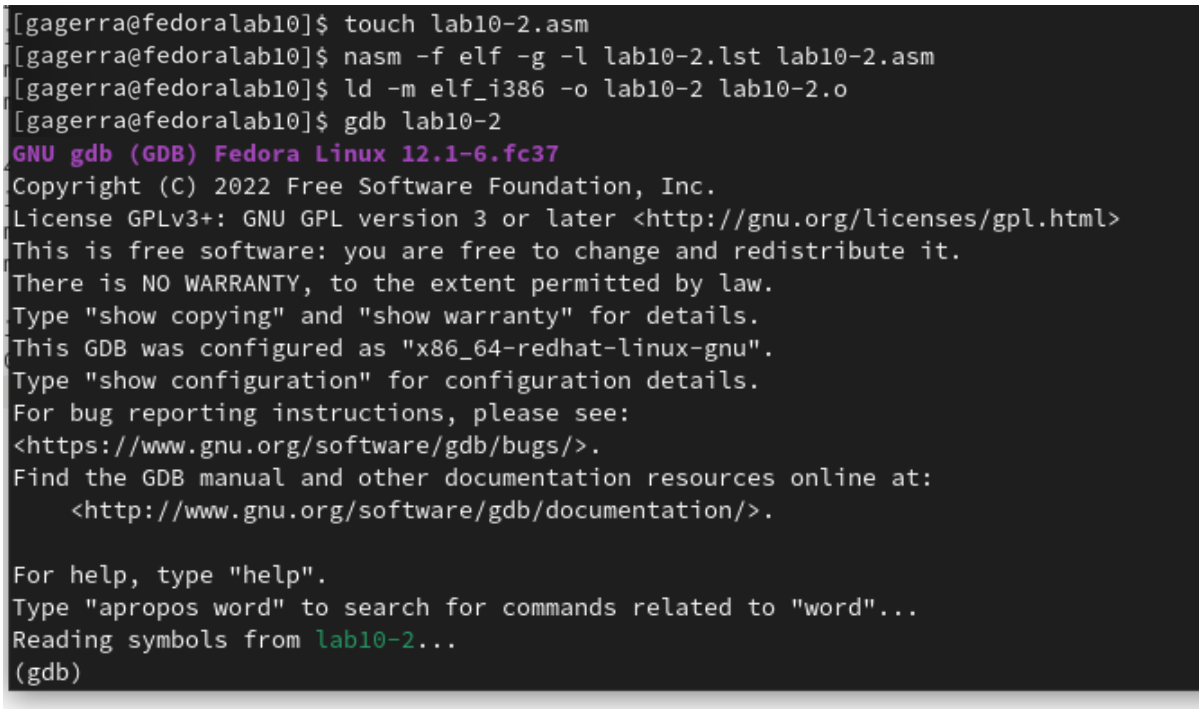


Рис. 4.6: Работа программы lab10-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы (рис. 4.7, 4.8)

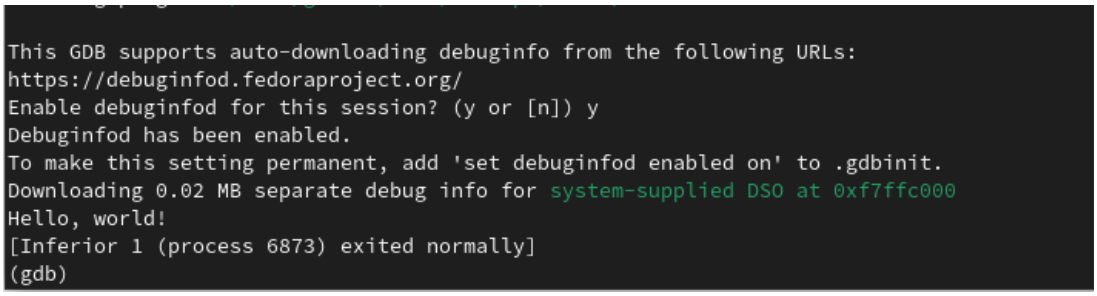


Рис. 4.7: дисассимилированный код

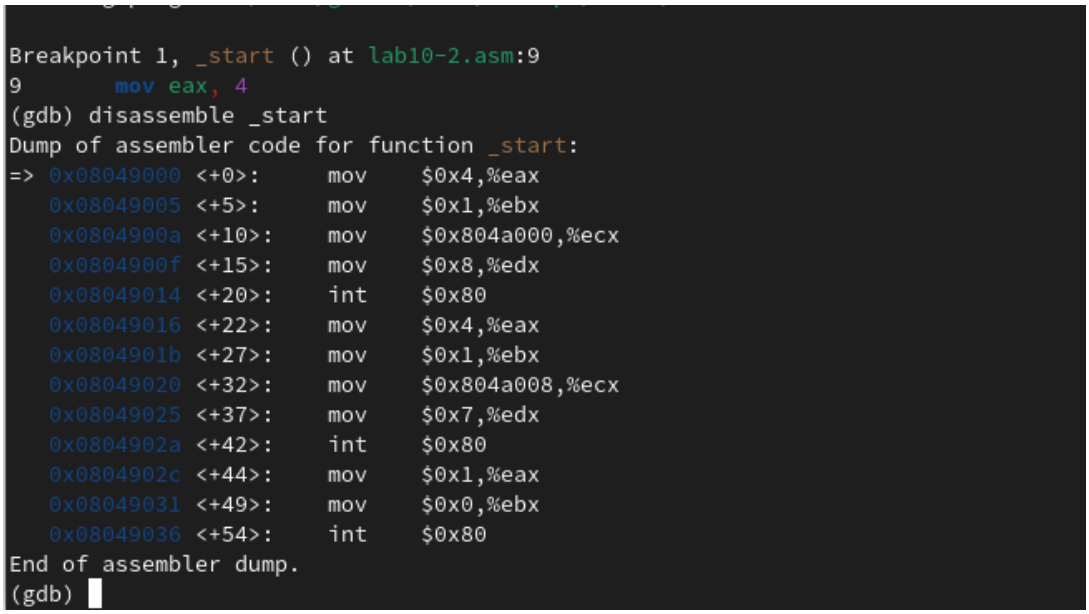


Рис. 4.8: дисассимилированный код в режиме интел

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверьте это с помощью команды info breakpoints (кратко i b) Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определите адрес предпоследней инструкции (mov ebx,0x0) и установите точку.(рис. 4.9)

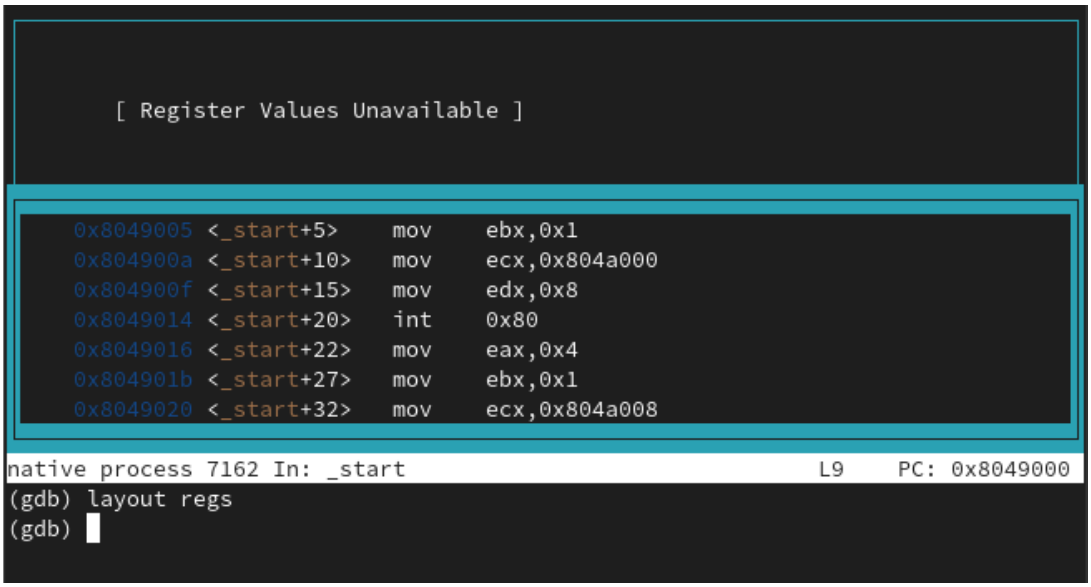


Рис. 4.9: точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды stepi (или si) и проследите за изменением значений регистров. (рис. 4.11 4.12)

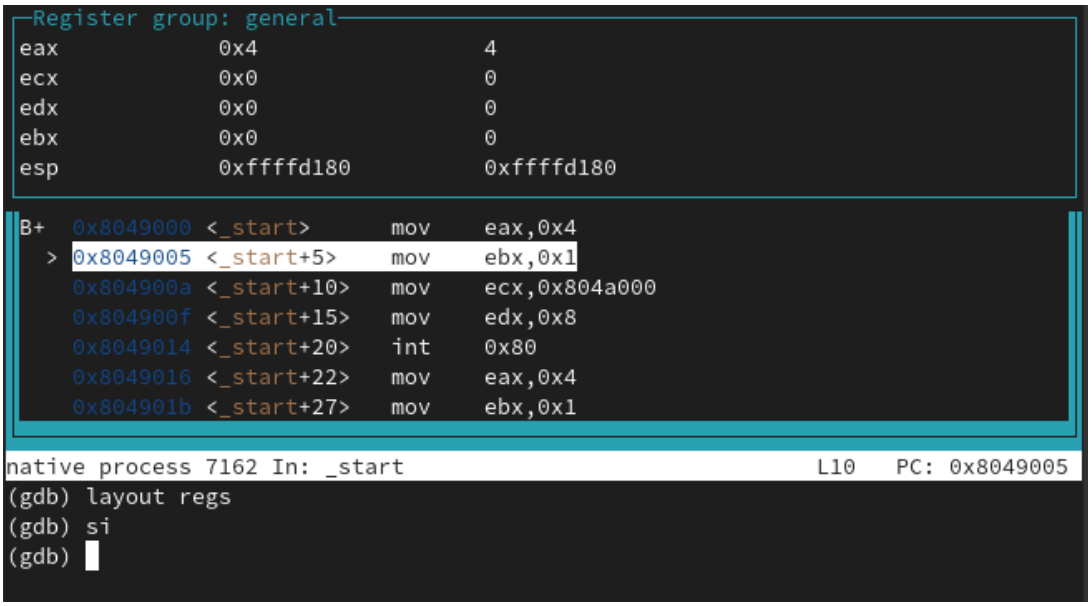


Рис. 4.10: изменение регистров

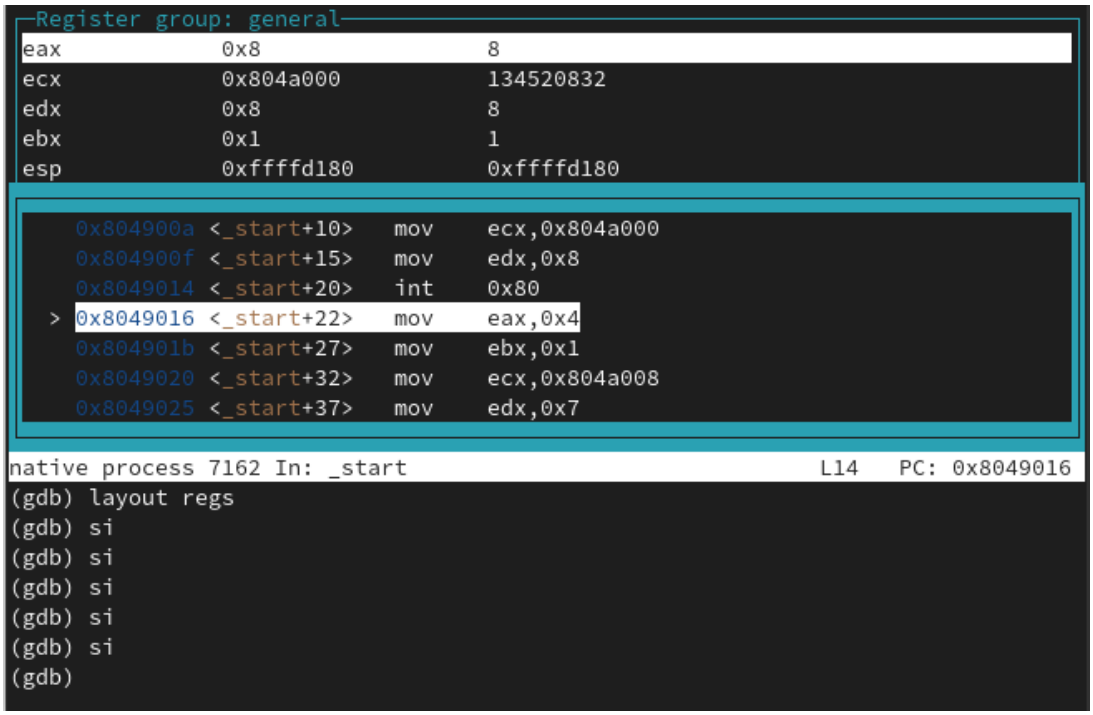


Рис. 4.11: изменение регистров

Посмотрите значение переменной msg1 по имени Посмотрите значение переменной msg2 по адресу Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Измените первый символ переменной msg1 Замените любой символ во второй переменной msg2. (рис. 4.12)

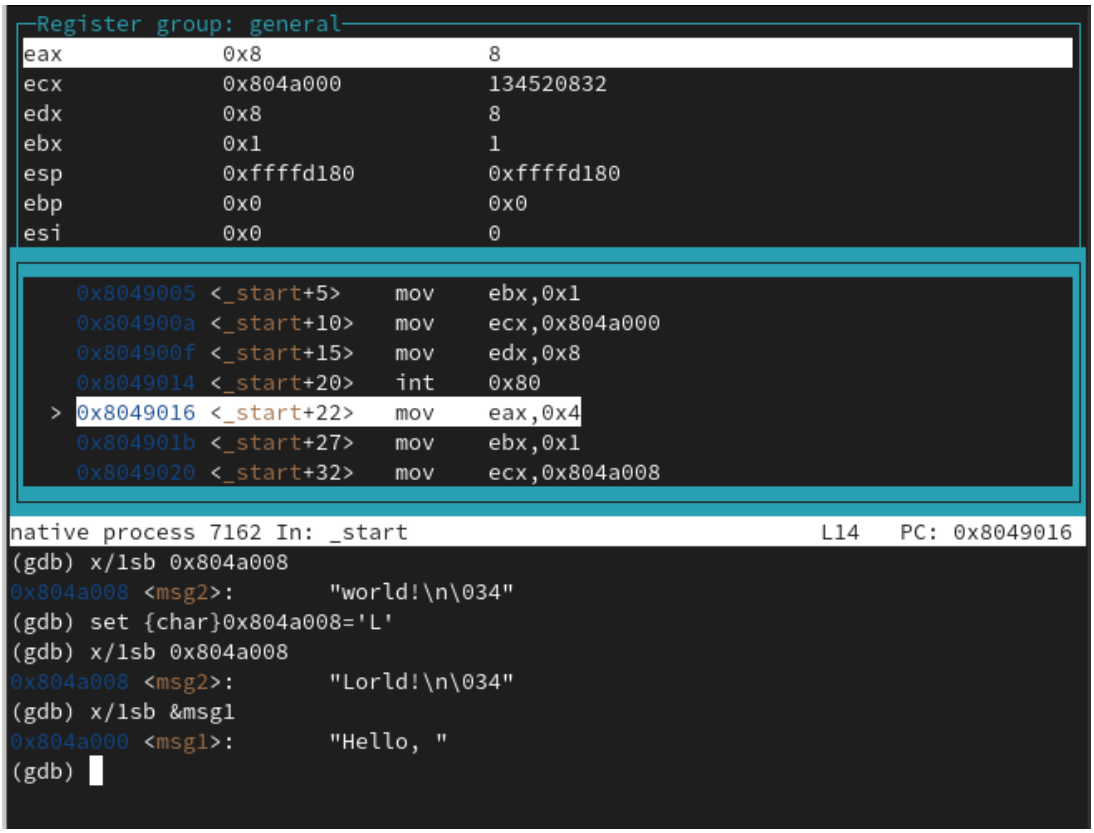


Рис. 4.12: изменение значения переменной

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set измените значение регистра ebx:(рис. 4.13)

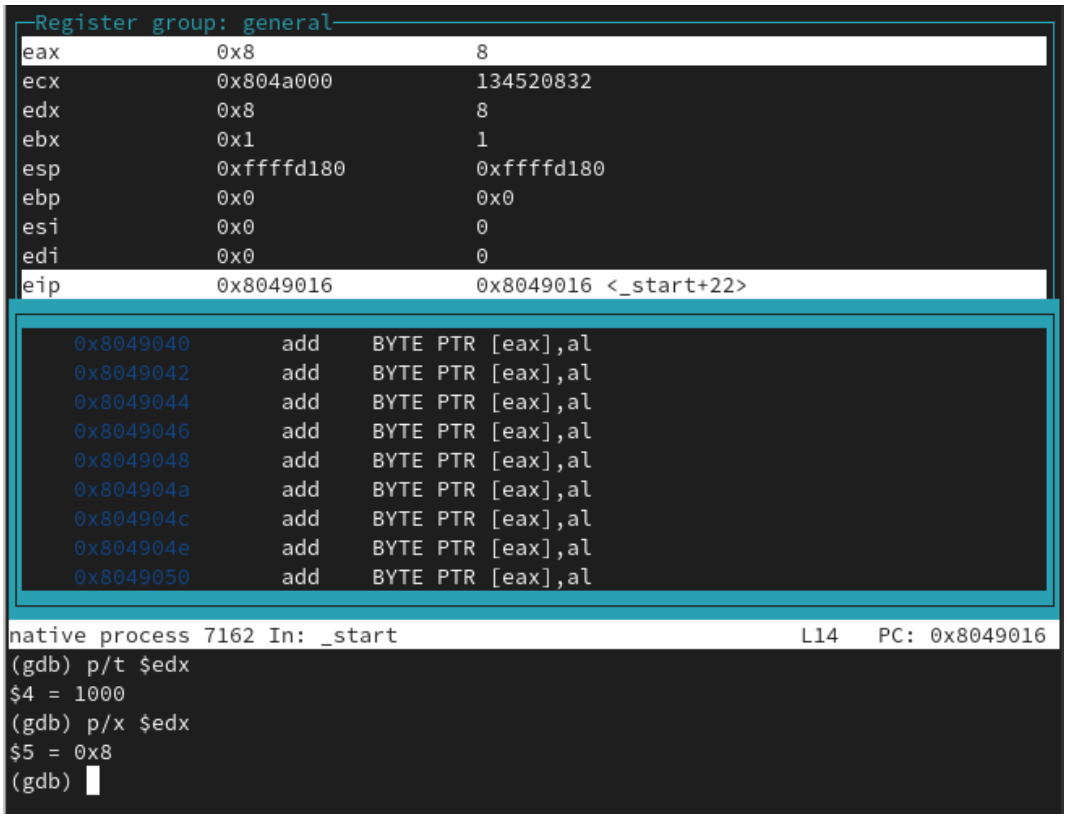


Рис. 4.13: вывод значения регистра

С помощью команды set измените значение регистра ebx:(рис. 4.14)

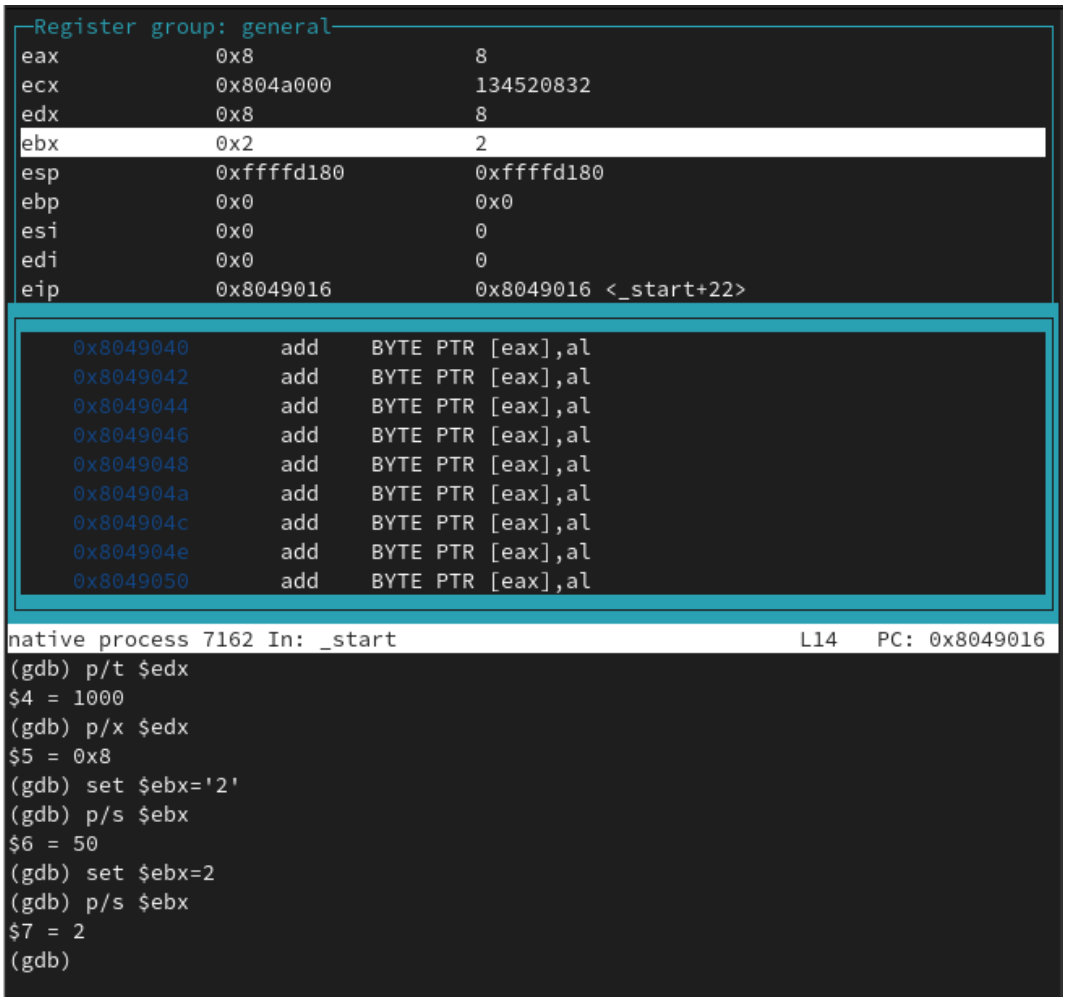


Рис. 4.14: вывод значения регистра

5. Скопируйте файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузите исполняемый файл в отладчик, указав аргументы

Для начала установим точку останова перед первой инструкцией в программе

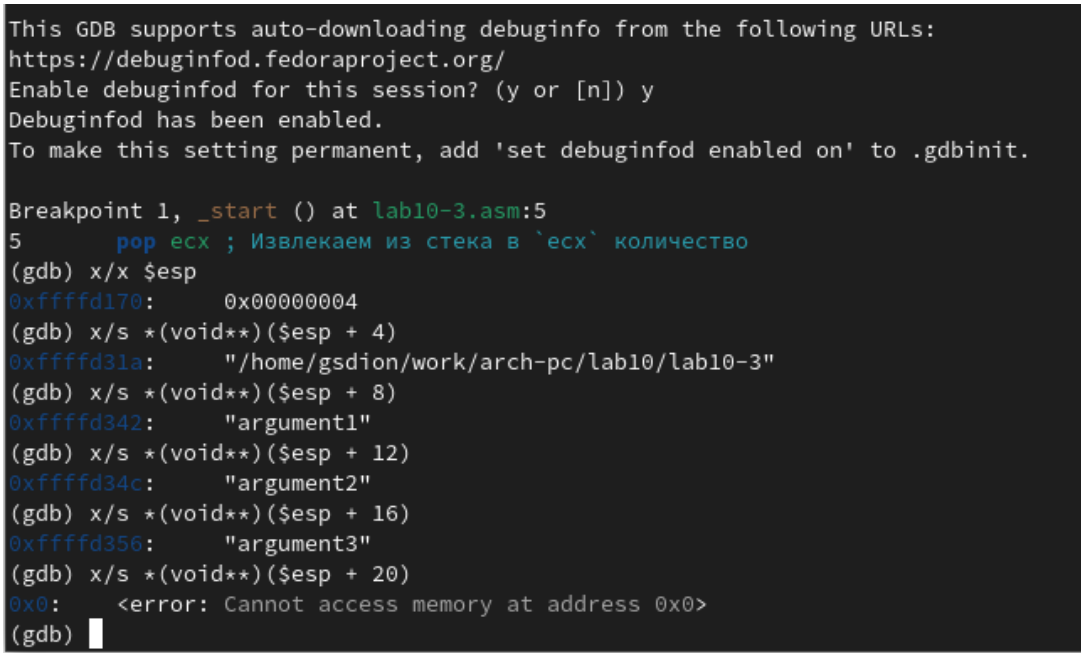
и запустим ее.

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрите остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д. (рис. 4.15)

Рис. 4.15: вывод значения регистра

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] -



шаг равен размеру переменной - 4 байтам.

6. Преобразуйте программу из лабораторной работы №9 (Задание №1 для

самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. 4.16 4.17)

Рис. 4.16: Файл lab10-4.asm

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db 'f(x)=5(2+x) ',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 _subcalc
add esi,eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

_subcalc:
add eax,2
mov ebx,5
mul ebx
ret
```

```
[gagerra@fedora lab10]$ touch lab10-4.asm
[gagerra@fedora lab10]$ nasm -f elf lab10-4.asm
[gagerra@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[gagerra@fedora lab10]$ ./lab10-4
f(x)=5(2+x)
Результат: 0
[gagerra@fedora lab10]$ ./lab10-4 2 3 4 5 6 7 8
f(x)=5(2+x)
Результат: 245
[gagerra@fedora lab10]$
```

Рис. 4.17: Работа программы lab10-4.asm

7. В листинге приведена программа вычисления выражения (3+2)* 4+5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. 4.18 4.19 4.20 4.21)

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

Рис. 4.18: код с ошибкой

```
%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. 4.19: отладка

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax

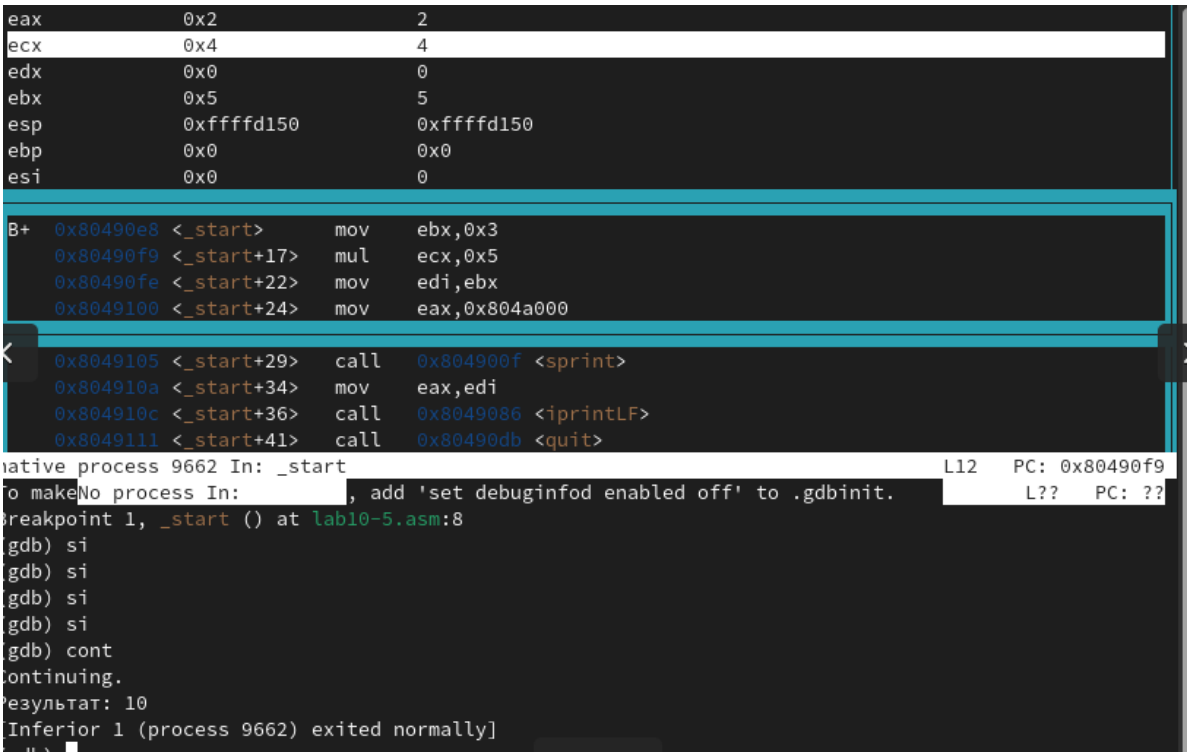


Рис. 4.20: код исправлен

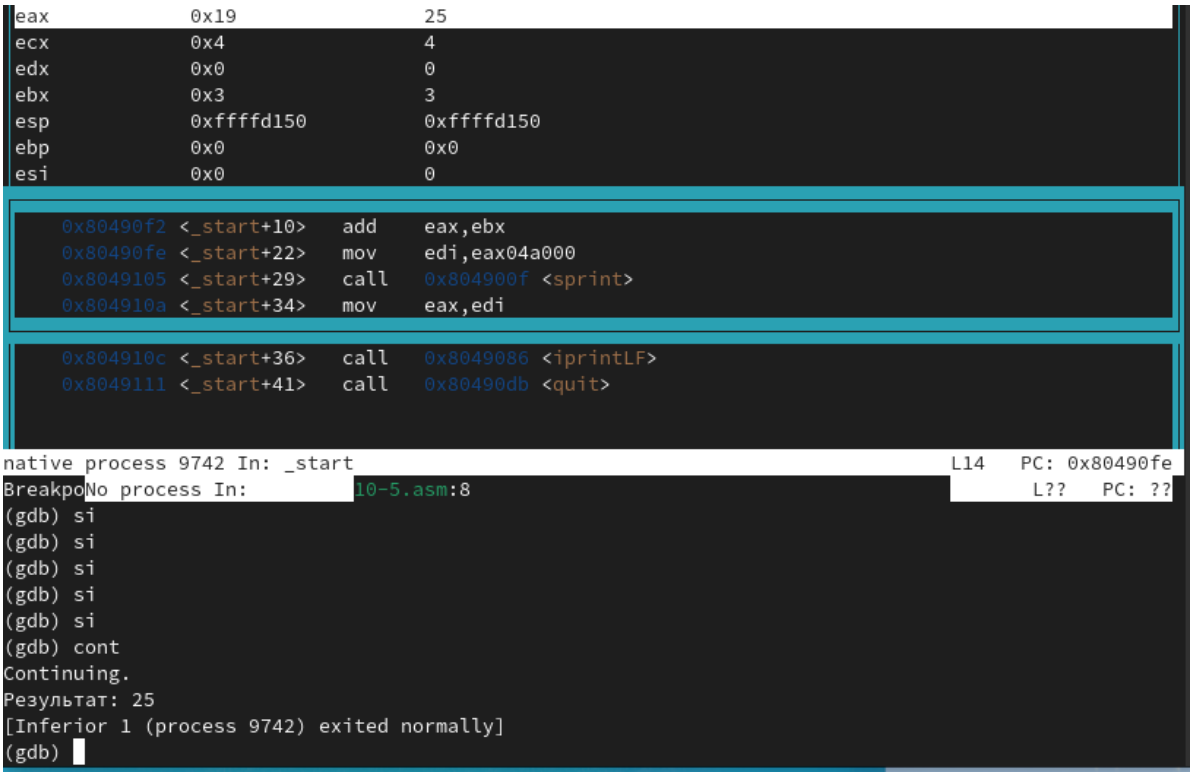


Рис. 4.21: проверка работы

5 Выводы

Освоили работу с подпрограммами и отладчиком.

Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux