Лабораторная работа №10

Понятие подпрограммы. Отладчик GDB.

Воробчук Лилия Андреевна

Содержание

Список иллюстраций

Список таблиц

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

1. Создаю каталог lab10 и файл lab10-1.asm (рис. 2.1)

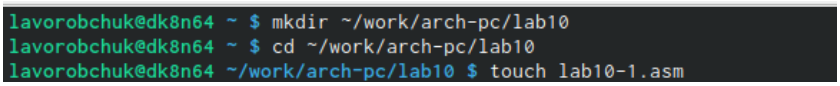

Рис. 2.1: создание файла

2. Ввожу текст листинга (рис. 2.2)

```
%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:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul
mov eax,result
call sprint
mov eax,[rez]
call iprintLF
call quit

_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret
```

Рис. 2.2: листинг

3. Запускаю программу (рис. 2.3)

```
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 4
2x+7=15
```

Рис. 2.3: создание файла

4. Меняю текст программы, чтобы она решала выражение f(g(x)). (рис. 2.4)

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x)=2x+7',0
prim2: DB 'g(x)=3x-1',0
result: DB 'f(g(x))=',0
SECTION .bss
x: RESB 80
rez: RESB 80
SECTION .text
GLOBAL _start
_start:

mov eax, prim1
call sprintLF

mov eax, prim2
call sprintLF

mov eax, msg
call sprintLF

mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _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

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

Рис. 2.4: внесение изменений

5. Запускаю измененную программу (рис. 2.5)

```
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ./lab10-1
f(x)=2x+7
g(x)=3x-1
Введите x:
4
f(g(x))=29
```

Рис. 2.5: запуск

6. Создаю файл lab10-2.asm и вписываю туда Листинг 10.2(рис. 2.6)

```
1  SECTION .data
2  msg1: db "Hello, ",0x0
3  msg1Len: equ $ - msg1
4  msg2: db "world!",0xa
5  msg2Len: equ $ - msg2
6  SECTION .text
7  global _start
8  _start:
9  mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 2.6: ввод листинга

7. Запускаю файл второй программы в отладчик gdb. (рис. 2.7)

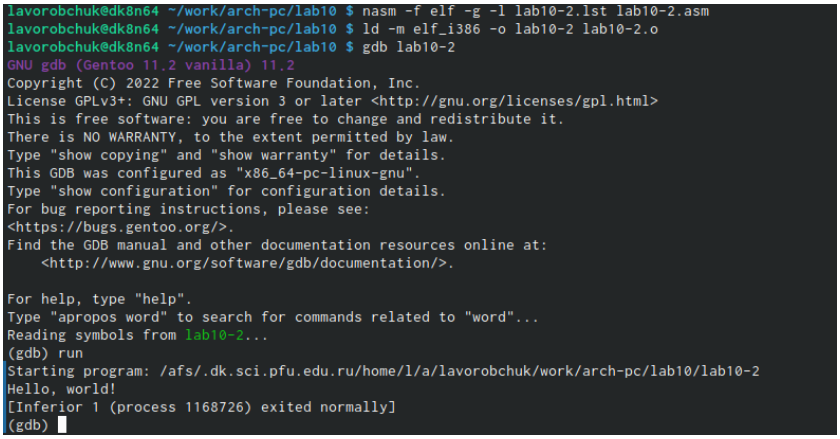

Рис. 2.7: отладчик

8. Ставлю брекпоинт на метку _start и запускаю программу(рис. 2.8)

Рис. 2.8: брекпоинт

9. Просматриваю дисассимплированный код программы, начиная с метки.(рис. 2.9)

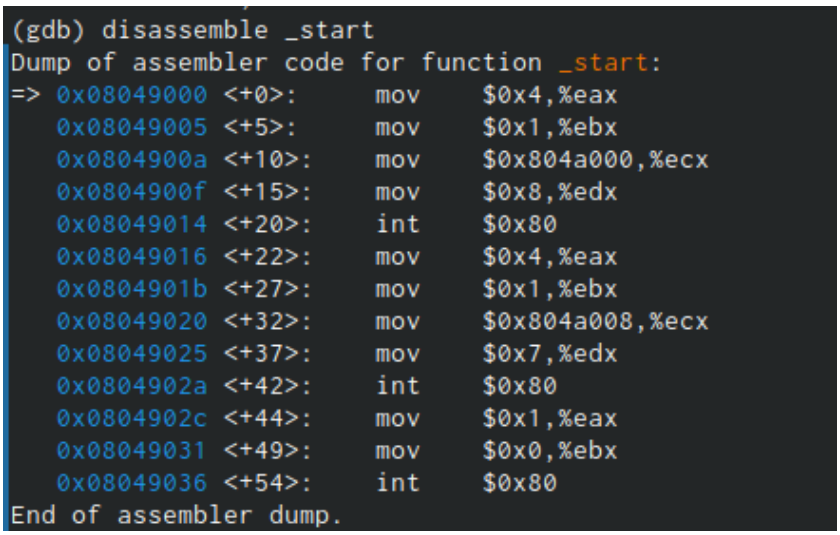

Рис. 2.9: дисассимплированный код

10. С помощью команды переключаюсь на intel'овское отображение синтаксиса. Отличие заключается в командах, в диссамилированном отображении в командах используют % и $, а в Intel отображение эти символы не используются. (рис. 2.10)

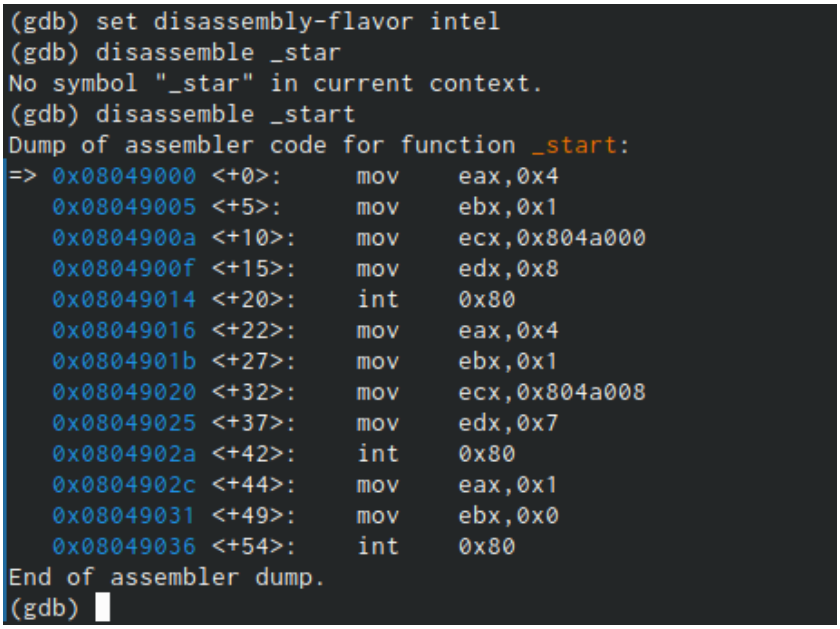

Рис. 2.10: Intel'овское отображение

11. Для удобства включаю режим псевдографики. (рис. 2.11)

Рис. 2.11: режим псевдографики

12. Посматриваю наличие меток и добавляю еще одну метку на предпоследнюю инструкцию. (рис. 2.12)

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab10-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab10-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x08049031 lab10-2.asm:20
(gdb)
```

Рис. 2.12: Наличие меток

13. С помощью команды si посматриваю регистры и изменяю их (рис. 2.13)

```
(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "Hello, "
(gdb)
```

Рис. 2.13: просмотр регистров

14. Смотрю значение второй переменной msg2. (рис. 2.14)

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "world!\n\034"
(gdb)
```

Рис. 2.14: Просмотр значения переменной

15. С помощью команды set меняю значение переменной msg1. (рис. 2.15)

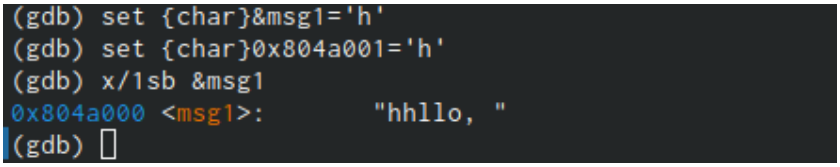

Рис. 2.15: Изменение значения переменной

16. Меняю переменную msg2.(рис. 2.16)

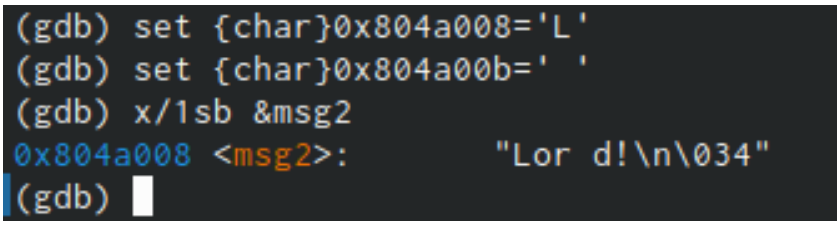

Рис. 2.16: Изменение msg2

17. Вывожу значение регистров ecx и eax (рис. 2.17)

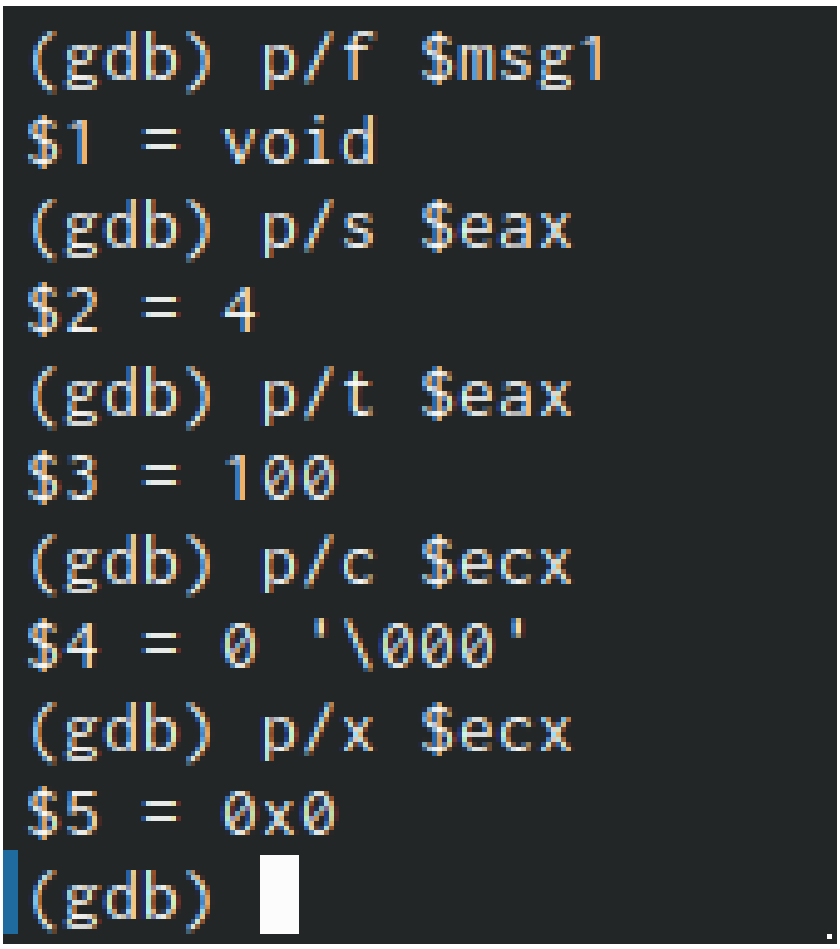

Рис. 2.17: Значение регистров ecx и eax

18. Меняю значение регистра ebx. Команда выводит два разных значения так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные. (рис. 2.18)

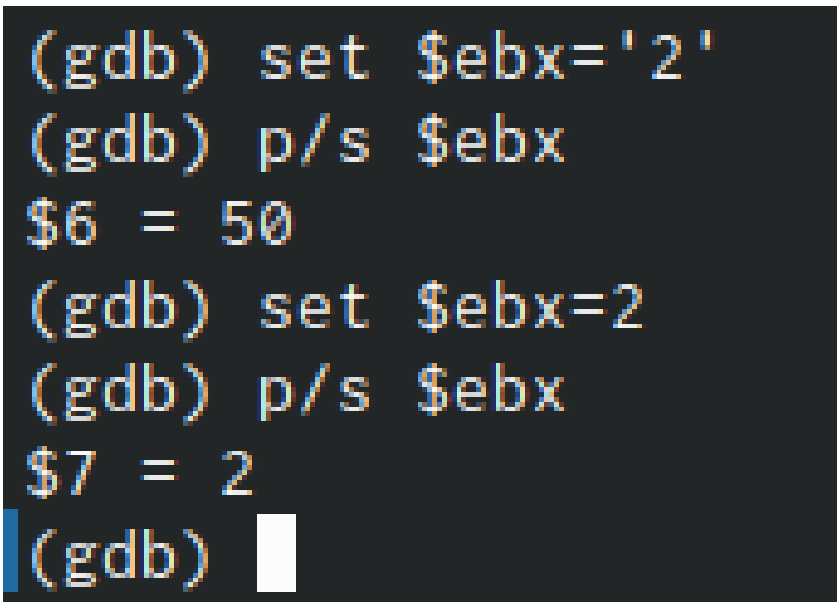

Рис. 2.18: Значение регистров ebx

19. Копирую файл lab9-2.asm и переименовываю его. Запускаю файл в отладчике и указываю аргументы.(рис. 2.19)

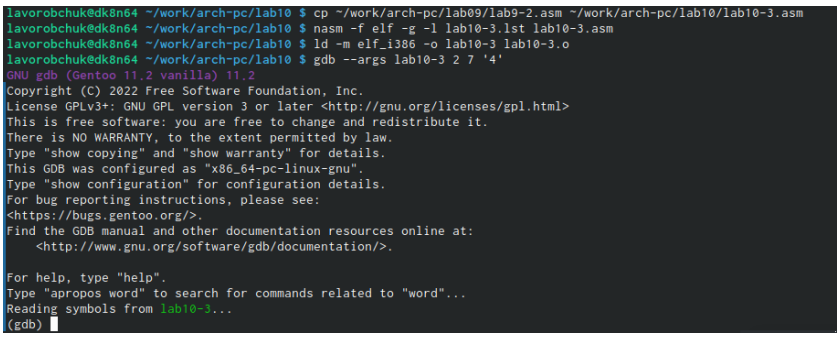

Рис. 2.19: Запуск файла в отладчике

20. Ставлю метку на _start и запускаю файл.(рис. 2.20)

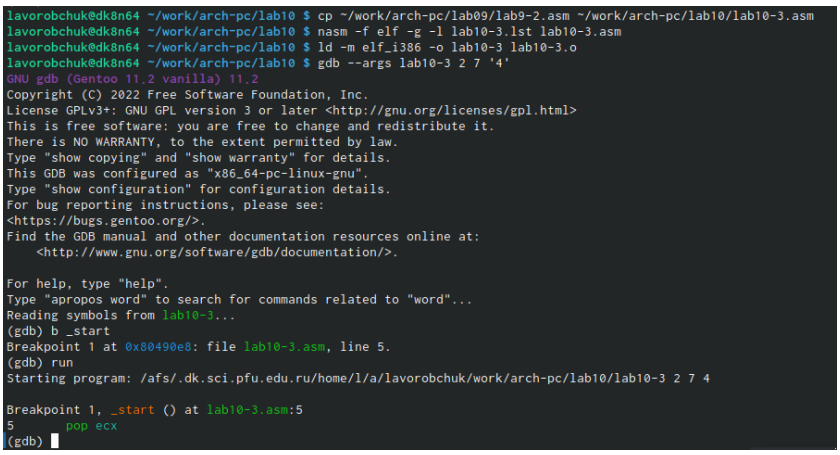

Рис. 2.20: Запуск файла lab10-3 через метку

21. Проверяю адрес вершины стека и убеждаюсь, что там хранится 5 элементов.(рис. 2.21)

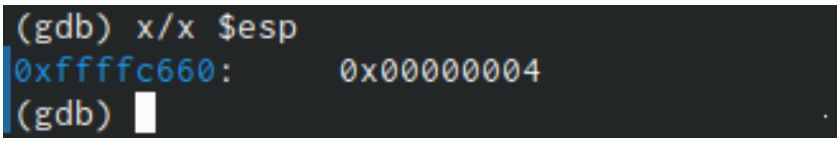

Рис. 2.21: Адрес вершины стека

22. Смотрю все позиции стека. По первому адрему хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации. (рис. 2.22)

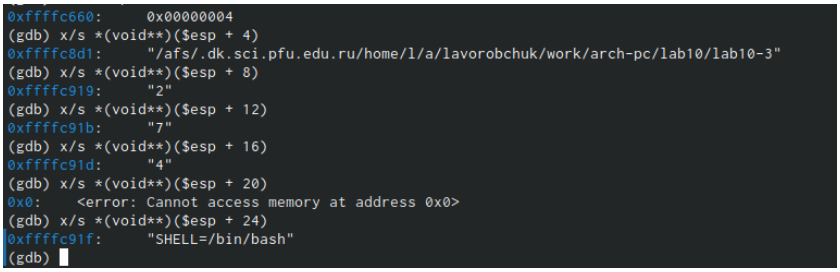

Рис. 2.22: Все позиции стека

3 Самостоятельная работа

1. Преобразовываю программу из лабораторной работы №9 и реализовываю вычисления как подпрограмму.(рис. 3.1)

```
1  %include 'in_out.asm'
2
3  SECTION data
4  primer DB 'f (x) = 4x+3',0
5  result DB 'Результат: ',0
6  SECTION .text
7  GLOBAL _start
8  _start:
9
10 pop ecx
11
12 pop edx
13
14 sub ecx,1
15
16 mov esi,0
17
18 mov eax,primer
19 call sprintLF
20 next:
21 cmp ecx, 0
22 jz _end
23
24 pop eax
25 call atoi
26 call fir
27 add esi,eax
28
29 loop next
30
31 _end:
32 mov eax,result
33 call sprint
34 mov eax,esi
35 call iprintLF
36 call quit
37
38 fir:
39 mov ebx, 4
40 mul ebx
41 add eax, 3
42 ret
43
```

Рис. 3.1: Текст программы

2. Создаю исполняемый файл и проверяю его работу. (рис. 3.2)

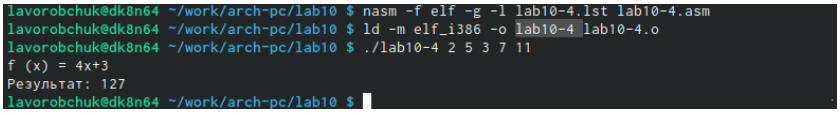

Рис. 3.2: Запуск программы

3. Ввожу в файл Листинг 10.3 (рис. 3.3)

```
%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 ecx,4
mul ecx
add ebx,5
mov edi, ebx

mov eax,div
call sprint
mov eax, edi
call iprintLF

call quit

```

Рис. 3.3: Текст програмы

4. Запускаю программу и вижу, что результат вычисления неверный(рис. 3.4)

```
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ touch backstreet.asm
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ nasm -f elf backstreet.asm
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o backstreet backstreet.o
lavorobchuk@dk8n64 ~/work/arch-pc/lab10 $ ./backstreet
Результат: 10
```

Рис. 3.4: Запуск программы

5. После появления ошибки запускаю программу в отладчике (рис. 3.5)

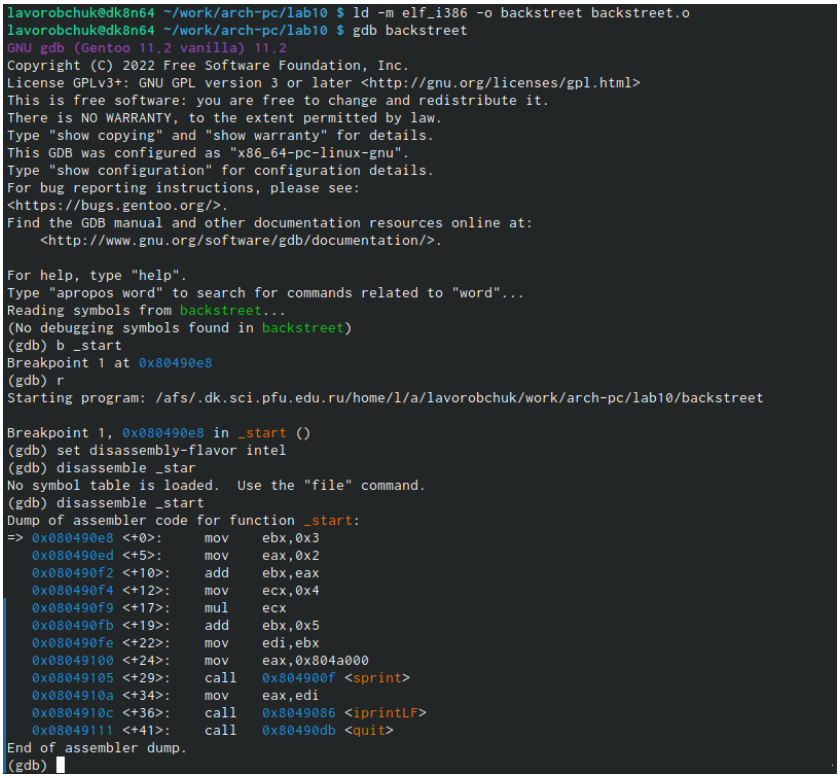

Рис. 3.5: Запуск программы в отладчике

6. Открываю регистры и, проанализировав их, понимаю, что некоторые регистры стоят не на своих местах и исправляю это (рис. 3.6)

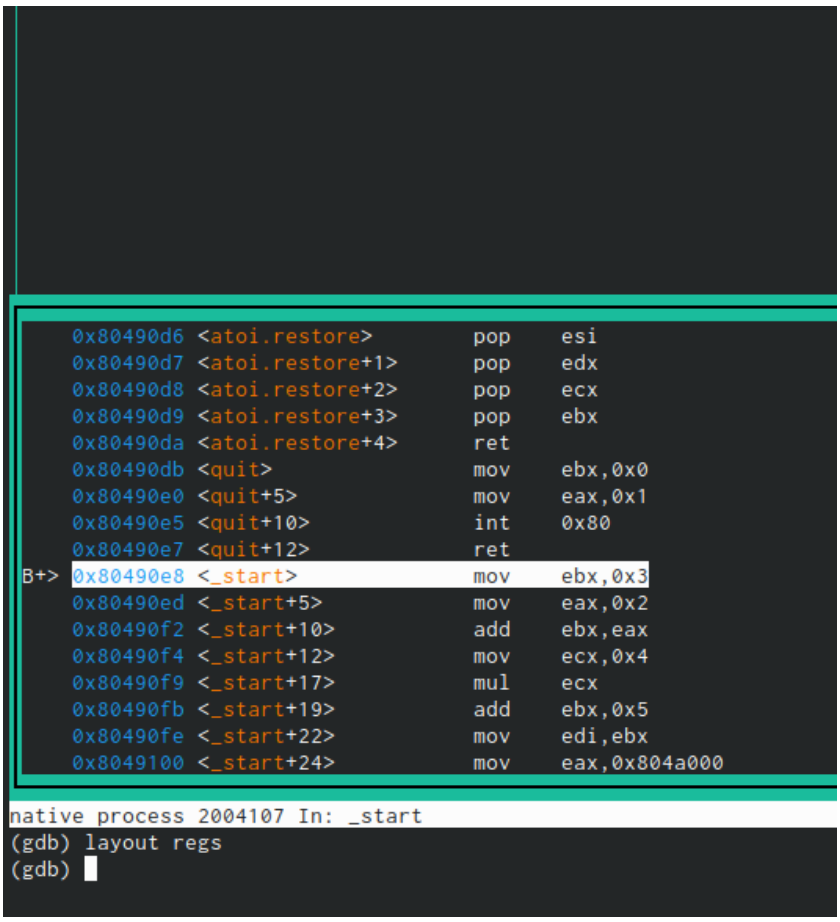

Рис. 3.6: Анализ регистров

7. Изменяю регистры и запускаю программу, программа вывела ответ 25, то есть все работает правильно. (рис. 3.7)

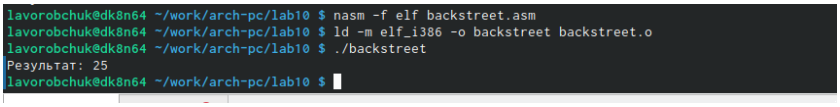

Рис. 3.7: Повторный запуск программы

4 Выводы

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