

# **Отчет по лабораторной работе №9**

**Дисциплина: Архитектура компьютера**

Абронина Алиса Кирилловна

# Содержание

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Цель работы</b>                                   | <b>5</b>  |
| <b>2</b> | <b>Задание</b>                                       | <b>6</b>  |
| <b>3</b> | <b>Теоретическое введение</b>                        | <b>7</b>  |
| <b>4</b> | <b>Выполнение лабораторной работы</b>                | <b>8</b>  |
| <b>5</b> | <b>Выполнение заданий для самостоятельной работы</b> | <b>14</b> |
| <b>6</b> | <b>Выводы</b>  | <b>16</b> |

## Список иллюстраций

|      |   |    |
|------|---|----|
| 4.1  | Создание рабочего каталога . . . . .                            | 8  |
| 4.2  | Запуск программы из листинга . . . . .                          | 8  |
| 4.3  | Изменение программы первого листинга . . . . .                  | 8  |
| 4.4  | Запуск программы в отладчике . . . . .                          | 9  |
| 4.5  | Проверка программы отладчиком . . . . .                         | 9  |
| 4.6  | Запуск отладчика с брейкпойнтом . . . . .                       | 10 |
| 4.7  | Режим псевдографики . . . . .                                   | 10 |
| 4.8  | Список брейкпойнтов; Добавление второй точки останова . . . . . | 11 |
| 4.9  | Просмотр содержимого регистров . . . . .                        | 11 |
| 4.10 | Просмотр содержимого переменных . . . . .                       | 11 |
| 4.11 | Просмотр значения регистра разными представлениями . . . . .    | 12 |
| 4.12 | Примеры использования команды set . . . . .                     | 12 |
| 4.13 | Подготовка новой программы . . . . .                            | 13 |
| 4.14 | Проверка работы стека . . . . .                                 | 13 |
| 5.1  | Измененная программа предыдущей лабораторной работы . . . . .   | 14 |
| 5.2  | Проверка корректировок в программе . . . . .                    | 15 |

## **Список таблиц**

# 1 Цель работы

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

## **2 Задание**

1. Выполнение лабораторной работы
2. Выполнение заданий для самостоятельной работы

## 3 Теоретическое введение

Отладка — это когда программист ищет и устраняет ошибки в коде. Этот процесс состоит из четырёх шагов:

1. Обнаружение ошибки — вы видите, что что-то пошло не так.
2. Поиск места ошибки — нужно понять, где именно в коде проблема.
3. Определение причины ошибки — почему произошла эта ошибка?
4. Исправление ошибки — устранение проблемы.

Есть несколько типов ошибок:

- Синтаксические ошибки — они возникают, когда код написан неправильно для языка программирования. Эти ошибки легко заметить, потому что программа просто не запустится.
- Семантические ошибки — эти ошибки сложнее. Программа работает, но делает не то, что вы хотели.
- Ошибки времени выполнения — такие ошибки появляются уже после запуска программы. Например, деление на ноль или переполнение памяти могут остановить работу программы.

Чтобы найти ошибку, лучше всего разбивать программу на маленькие части и проверять каждую часть отдельно. Когда находишь проблему, обычно становится понятно, почему она возникла. Исправляешь ошибку, снова запускаешь программу, и если всё хорошо, продолжаешь дальше. Если нет — ищем следующую ошибку!

## 4 Выполнение лабораторной работы

Создаю каталог для выполнения лабораторной работы №9 (рис. -fig. 4.1).

```
akabronina@vbox:~/work/arch-pc$ mkdir lab09
akabronina@vbox:~/work/arch-pc$ ls
lab08 lab09
akabronina@vbox:~/work/arch-pc$ cd lab09
akabronina@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
akabronina@vbox:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. -fig. 4.2).

```
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab09-1.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
akabronina@vbox:~/work/arch-pc/lab09$ ./lab09
bash: ./lab09: Нет такого файла или каталога
akabronina@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 9
2x+7=25
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab09-1.asm
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения  $f(g(x))$  (рис. -fig. 4.3).

```
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab09-1.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
akabronina@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 9
2(3x-1)+7=59
akabronina@vbox:~/work/arch-pc/lab09$
```

Рис. 4.3: Изменение программы первого листинга

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компилирую и запускаю в отладчике (рис.



-fig. 4.4).

```
akabronina@vbox:~/work/arch-pc/lab09$ touch lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
akabronina@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `run`, я убедилась в том, что она работает исправно (рис. -fig. 4.5).

```
akabronina@vbox:~/work/arch-pc/lab09$ touch lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
akabronina@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/akabronina/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 3329) exited normally]
(gdb) break _start
```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку. Далее смотрю дисассимилированный код программы, перевожу на команд (рис. -fig. 4.6). Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым.

Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/akabronina/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov     eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:   mov     $0x4,%eax
0x08049005 <+5>:   mov     $0x1,%ebx
0x0804900a <+10>:  mov     $0x804a000,%ecx
0x0804900f <+15>:  mov     $0x8,%edx
0x08049014 <+20>:  int     $0x80
0x08049016 <+22>:  mov     $0x4,%eax
0x0804901b <+27>:  mov     $0x1,%ebx
0x08049020 <+32>:  mov     $0x804a000,%ecx
0x08049025 <+37>:  mov     $0x7,%edx
0x0804902a <+42>:  int     $0x80
0x0804902c <+44>:  mov     $0x1,%eax
0x08049031 <+49>:  mov     $0x0,%ebx
0x08049036 <+54>:  int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:   mov     eax,0x4
0x08049005 <+5>:   mov     ebx,0x1
0x0804900a <+10>:  mov     ecx,0x804a000
0x0804900f <+15>:  mov     edx,0x8
0x08049014 <+20>:  int     0x80
0x08049016 <+22>:  mov     eax,0x4
0x0804901b <+27>:  mov     ebx,0x1
0x08049020 <+32>:  mov     ecx,0x804a000
0x08049025 <+37>:  mov     edx,0x7
0x0804902a <+42>:  int     0x80
0x0804902c <+44>:  mov     eax,0x1
0x08049031 <+49>:  mov     ebx,0x0
0x08049036 <+54>:  int     0x80
End of assembler dump.
(gdb) █
```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Включаю режим псевдографики для более удобного анализа программы (рис. 4.7).

```
0x0804907c      add     BYTE PTR [eax],al
0x0804907e      add     BYTE PTR [eax],al
0x08049080      add     BYTE PTR [eax],al
0x08049082      add     BYTE PTR [eax],al
0x08049084      add     BYTE PTR [eax],al

native process 3333 In: _start      L11  PC: 0x80
(gdb) layout regs
(gdb) █
```

Рис. 4.7: Режим псевдографики

Проверяю в режиме псевдографики, что брейкпоинт сохранился. Устанавливаю еще одну точку останова по адресу инструкции (рис. -fig. 4.8).

```

0x804a041  add     BYTE PTR [eax],al
0x804a045  add     BYTE PTR [eax],al
0x804a047  ins     BYTE PTR es:[edi],dx
0x804a048  popa
0x804a049  bound   edi,QWORD PTR [ecx]

native process 3333 In: _start          L11  PC: 0x80
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
(gdb) break *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab9-2.asm, line 11.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
2     breakpoint       keep y  0x8049000 lab9-2.asm:11
(gdb)

```

Рис. 4.8: Список брейкпоинтов; Добавление второй точки останова

Просматриваю содержимое регистров командой info registers (рис. -fig. 4.9).

```

native process 3333 In: _start          L11  PC: 0x80
esi            0x0                      0
edi            0x0                      0
eip            0x8049000                0x8049000 <_start>
eflags         0x202                    [ IF ]
cs             0x23                     35
ss             0x2b                     43
ds             0x2b                     43
--Type <RET> for more, q to quit, c to continue without paging--
Quit

```

Рис. 4.9: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.10).

```

native process 3333 In: _start          L11  PC: 0x80
esi            0x0                      0
edi            0x0                      0
eip            0x8049000                0x8049000 <_start>
eflags         0x202                    [ IF ]
cs             0x23                     35
ss             0x2b                     43
ds             0x2b                     43
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>:  "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:  "world!\n\034"
(gdb)

```

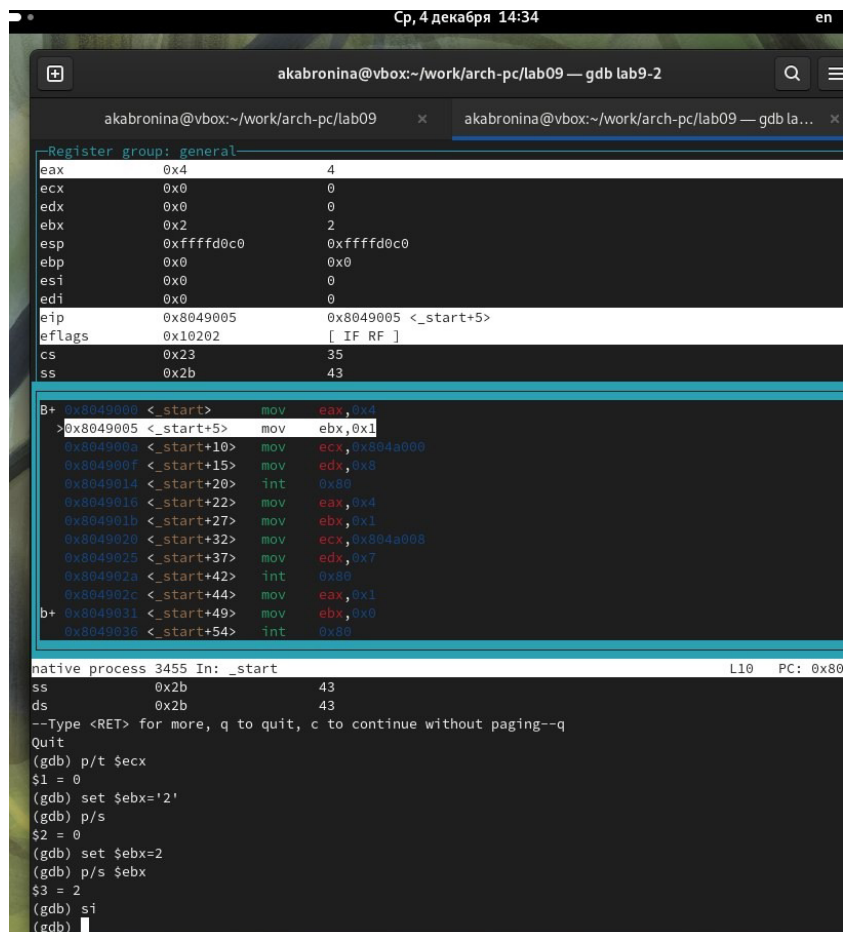
Рис. 4.10: Просмотр содержимого переменных

Вывожу в различных форматах значение регистра edx (рис. -fig. 4.11).

```
(gdb) p/t $ecx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/t $edx
$3 = 0
(gdb)
```

Рис. 4.11: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -fig. 4.12).



The screenshot shows a GDB terminal window with the following content:

```

Ср, 4 декабря 14:34
akabronina@vbox:~/work/arch-pc/lab09 — gdb lab9-2
akabronina@vbox:~/work/arch-pc/lab09 x akabronina@vbox:~/work/arch-pc/lab09 — gdb la...
Register group: general
eax 0x4 4
ecx 0x0 0
edx 0x0 0
ebx 0x2 2
esp 0xffffd0c0 0xffffd0c0
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x8049005 0x8049005 <_start+5>
eflags 0x10202 [ IF RF ]
cs 0x23 35
ss 0x2b 43

B+ 0x8049000 <_start> mov eax, 4
>0x8049005 <_start+5> mov ebx, 0x1
0x804900a <_start+10> mov ecx, 0x804a000
0x804900f <_start+15> mov edx, 0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax, 0x4
0x804901b <_start+27> mov ebx, 0x1
0x8049020 <_start+32> mov ecx, 0x804a008
0x8049025 <_start+37> mov edx, 0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax, 0x1
b+ 0x8049031 <_start+49> mov ebx, 0x0
0x8049036 <_start+54> int 0x80

native process 3455 In: _start L10 PC: 0x80
ss 0x2b 43
ds 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) p/t $ecx
$1 = 0
(gdb) set $ebx='2'
(gdb) p/s
$2 = 0
(gdb) set $ebx=2
(gdb) p/s $ebx
$3 = 2
(gdb) si
(gdb)

```

Рис. 4.12: Примеры использования команды set

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки. Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает

как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились. (рис. -fig. 4.13).(рис. -fig. 4.14)

```
akabronina@vbox:~/work/arch-pc/lab09$ cp lab8-2.asm lab9-3.asm
akabronina@vbox:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1.asm lab8-2.asm lab9-2.asm lab9-2.o
lab09-1     lab09-1.o lab9-2     lab9-2.lst lab9-3.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
akabronina@vbox:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/akabronina/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 4.13: Подготовка новой программы

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/akabronina/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd050: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd21b: "/home/akabronina/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd246: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd258: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd269: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd26b: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.14: Проверка работы стека

## 5 Выполнение заданий для самостоятельной работы

- 1) Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 5.1).

```
_start:
mov eax, msg_func
call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _calculate_fx
add esi, eax
loop next

_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintf
call quit
_calculate_fx:
mov ebx, 5
mul ebx
add eax, 17
ret
```

Рис. 5.1: Измененная программа предыдущей лабораторной работы

- 2)

Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul ebx можно заметить, что результат умножения записывается в регистр eax, но также

меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию. Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 5.2).

```
(gdb) disassemble _start
Undefined command: "disassembly". Try "help".
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>: mov     ebx,0x3
    0x080490ed <+5>: mov     eax,0x2
    0x080490f2 <+10>: add     ebx,eax
    0x080490f4 <+12>: mov     ecx,0x4
    0x080490f9 <+17>: mul     ecx
    0x080490fb <+19>: add     ebx,0x5
    0x080490fe <+22>: mov     edi,ebx
    0x08049100 <+24>: mov     eax,0x0804a000
    0x08049105 <+29>: call    0x0804908f <sprint>
    0x0804910a <+34>: mov     eax,edi
    0x0804910c <+36>: call    0x08049086 <iprintLF>
    0x08049111 <+41>: call    0x080490db <quit>
End of assembler dump.
(gdb) layout asm
akabronina@vbox:~/work/arch-pc/lab09$ mousepad lab9-4.asm
akabronina@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
akabronina@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
akabronina@vbox:~/work/arch-pc/lab09$ ./lab9-4
Результат: 25
```

Рис. 5.2: Проверка корректировок в программе

## **6 Выводы**

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.