

# **Лабораторная работа №9.**

**Понятие подпрограммы.Отладчик GDB.**

Швед Карина Дмитриевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Ход работы</b>	<b>6</b>
<b>3</b>	<b>Самостоятельная работа</b>	<b>14</b>
<b>4</b>	<b>Выводы</b>	<b>16</b>

# Список иллюстраций

2.1	код программы lab09-1.asm . . . . .	7
2.2	работа программы lab09-1.asm . . . . .	7
2.3	код программы lab09-1.asm . . . . .	8
2.4	работа программы lab09-1.asm . . . . .	8
2.5	запуск программы lab09-2.asm в отладчике . . . . .	8
2.6	Дизассемблированный код . . . . .	9
2.7	Дизассемблированный код в режиме intel . . . . .	9
2.8	точка останова . . . . .	10
2.9	точка останова . . . . .	10
2.10	все точки останова . . . . .	11
2.11	изменение регистров . . . . .	11
2.12	изменение значения переменной . . . . .	12
2.13	вывод значения регистра . . . . .	12
2.14	вывод значения регистра . . . . .	12
2.15	вывод значения регистра . . . . .	13
3.1	программа в файле task8-1.asm . . . . .	14
3.2	программа в файле task8-1.asm . . . . .	14
3.3	код с ошибкой . . . . .	15
3.4	код исправлен . . . . .	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 Ход работы

Я создала каталог для выполнения лабораторной работы № 9, перешла в него и создала файл lab09-1.asm

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. Я внимательно изучила текст программы (Листинг 9.1). Ввела в файл lab09-1.asm текст программы из листинга 9.1. Создала исполняемый файл (рис. 2.1) и проверила его работу (рис. 2.2).

```

%include "in_out.asm"

SECTION .data
msg: DB "Введите x: ",0
result: DB "2x+7=",0
SECTION .bss
x: RESB 80
res: 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 [res], eax
ret ; выход из подпрограммы

mov eax, msg ; вызов подпрограммы печати сообщения

```

Рис. 2.1: код программы lab09-1.asm

```

vboxuser@hiy:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2x+7=15
vboxuser@hiy:~/work/arch-pc/lab09$

```

Рис. 2.2: работа программы lab09-1.asm

Я изменила текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $\varphi(\varphi(\varphi))$ , где  $\varphi$  вводится с клавиатуры,  $\varphi(\varphi) = 2\varphi + 7$ ,  $\varphi(\varphi) = 3\varphi - 1$ . Т.е.  $\varphi$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $\varphi(\varphi)$ , результат возвращается в `_calcul` и вычисляется выражение  $\varphi(\varphi(\varphi))$ . Результат возвращается в основную программу для вывода результата на экран. Я создала исполняемый файл (рис. 2.3) и проверила его работу (рис. 2.4)

код программы lab09-1.asm

Рис. 2.3: код программы lab09-1.asm

```
vboxuser@hiy:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vboxuser@hiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vboxuser@hiy:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2(3x-1)+7=29
vboxuser@hiy:~/work/arch-pc/lab09$
```

Рис. 2.4: работа программы lab09-1.asm

Я создала файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!)

Получила исполняемый файл и добавила отладочную информацию с помощью ключа '-g' для работы с GDB.(рис. 2.5)

```
vboxuser@hiy: ~/work/arch-pc/lab09
This GDB was configured as "x86_64-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 lab09-2...
(gdb) run
Starting program: /home/vboxuser/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
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 17230) exited normally]
(gdb) break _start
Breakpoint 1 at 0x00490000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 2.5: запуск программы lab09-2.asm в отладчике

Для более подробного анализа программы, установила точку остановки на метке 'start', с которой начинается выполнение любой ассемблерной программы, и запустила ее. Затем просмотрела дизассемблированный код программы (рис. 2.6) (рис. 2.7)



```
vboxuser@hiy: ~/work/arch-pc/lab09
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/vboxuser/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      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     $0x804a008,%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)
```

Рис. 2.6: Дизассемблированный код

```
vboxuser@hiy: ~/work/arch-pc/lab09
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,0x804a008
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. 2.7: Дизассемблированный код в режиме intel

Включила режим псевдографики для более удобного анализа программы. Для проверки точки останова по имени метки '\_start', использовала команду 'info breakpoints' (сокращенно 'i b'). Затем установила еще одну точку останова по адресу инструкции, определив адрес предпоследней инструкции 'mov ebx, 0x0'.

(рис. 2.7)

```
vboxuser@hiy: ~/work/arch-pc/lab09
b+ 0x8049000 <_start> mov eax,0x4
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
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80

native process 18813 (asm) In: _start L9 PC: 0x8049000
source language asm.
Arglist at unknown address.
Locals at unknown address, Previous frame's sp in esp
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 2.8: точка останова

Установила еще одну точку останова по адресу инструкции `mov ebx,0x0` (рис. 2.9)

```
vboxuser@hiy: ~/work/arch-pc/lab09
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
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al

native process 18813 (asm) In: _start L9 PC: 0x8049000
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис. 2.9: точка останова

Посмотрите информацию о всех установленных точках останова (рис. 2.10)

```

vboxuser@hiy: ~/work/arch-pc/lab09
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
0x8049038          add     BYTE PTR [eax],al
0x804903a          add     BYTE PTR [eax],al
0x804903c          add     BYTE PTR [eax],al

native process 18813 (asm) In: _start          L9      PC: 0x8049000
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y   0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint        keep y   0x08049031 lab09-2.asm:20
(gdb)

```

Рис. 2.10: все точки останова

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнила 5 инструкций с помощью команды stepi (или si) и проследила за изменением значений регистров (рис. 2.11)

```

vboxuser@hiy: ~/work/arch-pc/lab09
B+ 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>       mov     ebx,0x1
0x804900a <_start+10>      mov     ecx,0x804a008
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 18813 (asm) In: _start          L14     PC: 0x8049016
       breakpoint already hit 1 time
2      breakpoint        keep y   0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.11: изменение регистров

Просмотрела значение переменной msg1 по имени и получил нужные данные. Просмотрела значение переменной msg1 по имени и получил нужные данные. Для изменения значения регистра или ячейки памяти использовала команду set, указав имя регистра или адрес в качестве аргумента. Изменила первый символ переменной msg1 и первый символ переменной msg2 (рис. 2.12)

```
native process 18813 (asm) In: _start L14 PC: 0x8049016
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "Lorld!\n\034"
(gdb)
```

Рис. 2.12: изменение значения переменной

Вывела в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 2.13) С помощью команды set изменила значение регистра ebx (рис. 2.14)

```
native process 18813 (asm) In: _start L14 PC: 0x8049016
(gdb) set $ebx="2"
evaluation of this expression requires the program to have a function "malloc".
(gdb) p/s $eax
$2 = 8
(gdb) p/t $eax
$3 = 1000
(gdb) p/x $ecx
$4 = 0x804a000
(gdb)
```

Рис. 2.13: вывод значения регистра

```
native process 18813 (asm) In: _start L14 PC: 0x8049016
$2 = 8
(gdb) p/t $eax
$3 = 1000
(gdb) p/x $ecx
$4 = 0x804a000
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.14: вывод значения регистра

Скопировала файл lab8-2.asm, созданный во время выполнения лабораторной работы №8, который содержит программу для вывода аргументов командной

строки. Создала исполняемый файл из скопированного файла. Для загрузки программы с аргументами в gdb использовала ключ `-args` и загрузил исполняемый файл в отладчик с указанными аргументами. Установила точку останова перед первой инструкцией программы и запустила ее. Адрес вершины стека, содержащий количество аргументов командной строки (включая имя программы), хранится в регистре `esp`. По этому адресу находится число, указывающее количество аргументов. В данном случае видно, что количество аргументов равно 5, включая имя программы `lab9-3` и сами аргументы: `аргумент1`, `аргумент2` и 'аргумент 3'. Просмотрела остальные позиции стека. По адресу `[esp+4]` находится адрес в памяти, где располагается имя программы. По адресу `[esp+8]` хранится адрес первого аргумента, по адресу `[esp+12]` - второго и так далее. (рис. 2.15)

```
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/vboxuser/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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.

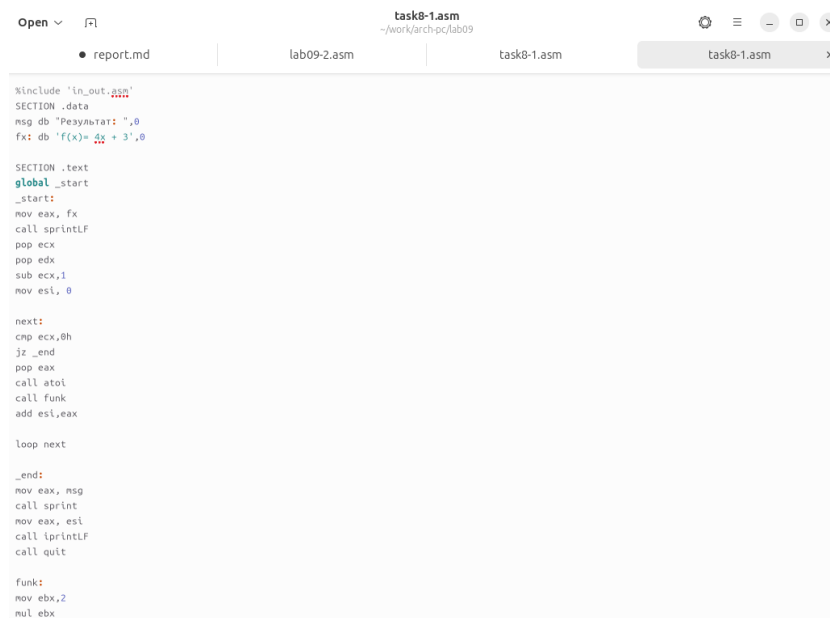
Breakpoint 1, _start () at lab09-3.asm:6
6      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffcff0: 0x00000005
(gdb) x/s *(void*)($esp + 4)
0xfffffd1bd: "/home/vboxuser/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void*)($esp + 8)
0xfffffd1e7: "аргумент1"
(gdb) x/s *(void*)($esp + 12)
0xfffffd1f9: "аргумент"
(gdb) x/s *(void*)($esp + 16)
0xfffffd20a: "2"
(gdb) x/s *(void*)($esp + 20)
0xfffffd20c: "аргумент 3"
(gdb) x/s *(void*)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.15: вывод значения регистра

Шаг изменения адреса равен 4, так как каждый следующий адрес на стеке находится на расстоянии 4 байт от предыдущего (`[esp+4]`, `[esp+8]`, `[esp+12]`).

### 3 Самостоятельная работа

1) Преобразовала программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x) = 4x + 3$  как подпрограмму (рис. 3.1)



```
task8-1.asm
~/work/arch-pc/lab09

report.md | lab09-2.asm | task8-1.asm | task8-1.asm x

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
fx: db "f(x)= 4x + 3",0

SECTION .text
global _start
_start:
mov eax, fx
call sprintf
pop ecx
pop edx
sub ecx,1
mov esi, 0

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

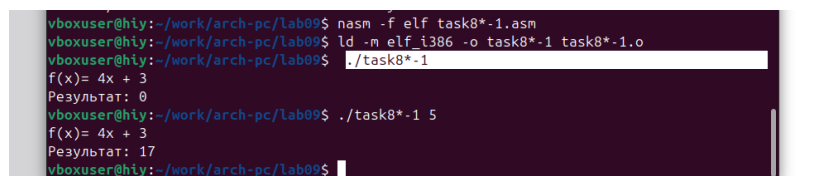
loop next

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

funk:
mov ebx, 2
mul ebx
```

Рис. 3.1: программа в файле task8-1.asm

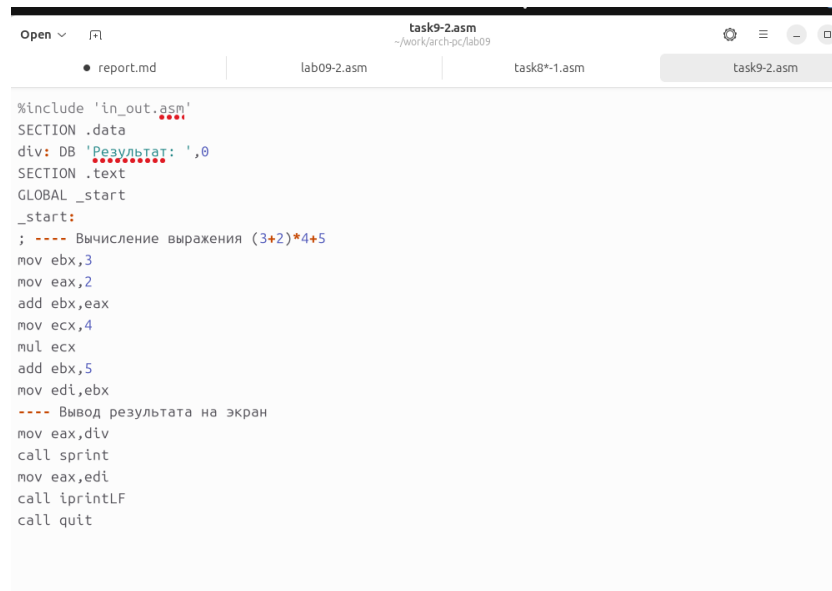
Создала исполняемый файл и проверила его работу (рис. 3.2)



```
vboxuser@hiy:~/work/arch-pc/lab09$ nasm -f elf task8-1.asm
vboxuser@hiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o task8-1 task8-1.o
vboxuser@hiy:~/work/arch-pc/lab09$ ./task8-1
f(x)= 4x + 3
Результат: 0
vboxuser@hiy:~/work/arch-pc/lab09$ ./task8-1 5
f(x)= 4x + 3
Результат: 17
vboxuser@hiy:~/work/arch-pc/lab09$
```

Рис. 3.2: программа в файле task8-1.asm

2) В листинге приведена программа вычисления выражения  $(3+2)*4+5$ . При запуске данная программа дает неверный результат. Проверила это, анализируя изменения значений регистров с помощью отладчика GDB. Определила ошибку - перепутан порядок аргументов у инструкции add. Также обнаружила, что по окончании работы в edi отправляется ebx вместо eax (рис. 3.3)

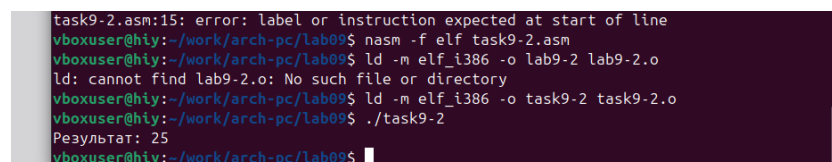


```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ----- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
----- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintfLF
call quit
```

Рис. 3.3: код с ошибкой

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

(рис. 3.4)



```
task9-2.asm:15: error: label or instruction expected at start of line
vboxuser@hiy:~/work/arch-pc/lab09$ nasm -f elf task9-2.asm
vboxuser@hiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
ld: cannot find lab9-2.o: No such file or directory
vboxuser@hiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o task9-2 task9-2.o
vboxuser@hiy:~/work/arch-pc/lab09$ ./task9-2
Результат: 25
vboxuser@hiy:~/work/arch-pc/lab09$
```

Рис. 3.4: код исправлен

## 4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм, а также познакомилась с методами отладки при помощи GDB и его основными возможностями.