

# **РУДН. Архитектура компьютеров**

**Отчёт по лабораторной работе №9**

Косинов Никита Андреевич, НПИМбв-02-20

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Ход работы</b>	<b>7</b>
<b>3</b>	<b>Написание подпрограммы</b>	<b>8</b>
<b>4</b>	<b>Отладка программ: точки останова</b>	<b>14</b>
<b>5</b>	<b>Работа с данными через GDB</b>	<b>20</b>
<b>6</b>	<b>Работа с аргументами через GDB</b>	<b>24</b>
<b>7</b>	<b>Самостоятельная работа</b>	<b>27</b>
<b>8</b>	<b>Выводы</b>	<b>35</b>

# Список иллюстраций

3.1	Создание файла . . . . .	8
3.2	Код программы lab09-1 . . . . .	9
3.3	Исполнение lab09-1 . . . . .	11
3.4	Фрагмент кода программы lab09-1-1 . . . . .	11
3.5	Исполнение lab09-1-1 . . . . .	13
4.1	Код программы lab09-2 . . . . .	14
4.2	Открытие программы lab90-2 с gdb . . . . .	16
4.3	Точка останова по метке . . . . .	16
4.4	Дисасемблированный код в АТТ . . . . .	16
4.5	Дисасемблированный код в Intel . . . . .	17
4.6	Псевдографика gdb . . . . .	17
4.7	Информация о точках останова . . . . .	17
4.8	Запуск программы lab09-2 в gdb . . . . .	18
4.9	Точка останова по адресу . . . . .	19
5.1	Пошаговое исполнение программы . . . . .	20
5.2	Информация о состоянии регистров . . . . .	21
5.3	Значение переменной msg1 . . . . .	21
5.4	Значение переменной msg2 . . . . .	21
5.5	Изменение значения переменной msg1 . . . . .	22
5.6	Завершение исполнения lab09-2 . . . . .	22
5.7	Изменение значения регистра . . . . .	23
6.1	Копирование и компиляция lab09-3 . . . . .	24
6.2	Открытие lab09-3 в gdb . . . . .	25
6.3	Вершина стека аргументов . . . . .	26
6.4	Адреса и значения элементов стека . . . . .	26
7.1	Код программы sr-1 . . . . .	27
7.2	Исполнение sr-1 . . . . .	29
7.3	Код программы sr-2 . . . . .	30
7.4	Исполнение sr-2 . . . . .	30
7.5	Отладка sr-2 . . . . .	31
7.6	Третий шаг исполнения sr-2 . . . . .	31
7.7	Код программы sr-1 . . . . .	32
7.8	Исправленный код программы sr-2 . . . . .	33

7.9	Исполнение исправленной sr-2 . . . . .	34
-----	--	----

## Список таблиц

# 1 Цель работы

При достаточно больших проектах часто требуется повторять одну и ту же операцию. Если их повторять подряд, то можно использовать циклы, но они не помогут для каких-то базовых часто используемых инструкций, таких как, например, ввод и вывод, возведение числа в степень и т.п. Для этого можно писать подпрограммы, внутри исходного файла, либо внутри подключаемого.

Также бывает довольно сложно в написанной неверно программе “глазами” найти в коде ошибку. Для решения этой проблемы созданы отладчики.

Цель данной работы заключается в освоении принципа написания подпрограмм и отладка программ с помощью отладчика **GDB**.

## 2 Ход работы

Лабораторная работа выполнена с использованием консоли **ОС Linux** и языка программирования ассемблера **NASM**.

1. Написание подпрограммы;
2. Отладка программ: точки останова;
3. Работа с данными через **GDB**;
4. Работа с аргументами через **GDB**.

В конце выполнена самостоятельная работа.

## 3 Написание подпрограммы

Для удобства написания программ и чтобы разобраться с работой подключаемых файлов, разберёмся с написанием подпрограммы.

1. Создаём рабочий файл.

```
nakosinov@dk6n55 ~ $ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/  
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git pull  
Уже актуально.  
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ cd labs/lab09/  
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ touch lab09-1.asm  
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit lab09-1.asm
```

Рис. 3.1: Создание файла

2. Напишем программу, считающую  $f(x) = 2x + 7$  для  $x$ , введённого с клавиатуры.



```

Открыть  lab09-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09  Сохранить  ▢  ▾  ▴  ✕

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg: DB 'Введите x: ',0
5     result: DB '2x+7=',0
6
7 SECTION .bss
8     x: RESB 80
9     res: RESB 80
10 SECTION .text
11     GLOBAL _start
12
13 _start:
14 ;-----
15 ; Основная программа
16 ;-----
17     mov eax, msg
18     call sprint
19
20     mov ecx, x
21     mov edx, 80
22     call sread
23
24     mov eax, x
25     call atoi
26
27     call _calcul ; Вызов подпрограммы _calcul
28
29     mov eax, result
30     call sprint
31     mov eax, [res]
32     call iprintLF
33     call quit
34 ;-----
35 ; Подпрограмма вычисления
36 ; выражения "2x+7"
37 _calcul:
38     mov ebx, 2
39     mul ebx
40     add eax, 7
41     mov [res], eax
42     ret ; выход из подпрограммы

```

Рис. 3.2: Код программы lab09-1

```
%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

3. В этой программе этап вычисления функции  $f(x)$  вынесен в конец программы с меткой *calcul*.
4. Компилируем и запускаем.

```
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf lab09-1.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 3.3: Исполнение lab09-1

5. Изменим программу так, чтобы она считала значение  $f(g(x))$ , где  $g(x) = 3x - 1$ . Вынесем этап вычисления  $g(x)$  в подпрограмму *subcalcul*, а в функции *calcul* вызовем её для подстановки.

```
34 ; -----
35 ; Подпрограмма вычисления
36 ; выражения "2x+7"
37 _calcul:
38     call _subcalcul
39     mov ebx, 2
40     mul ebx
41     add eax, 7
42     mov [res], eax
43     ret ; выход из подпрограммы
44 ; -----
45 ; Подпрограмма вычисления "3x-1"
46 _subcalcul:
47     mov ebx, 3
48     mul ebx
49     add eax, -1
50     ret
```

Рис. 3.4: Фрагмент кода программы lab09-1-1

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg: DB 'Введите x: ', 0
```

```
result: DB '2(3x-1)+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:
    call _subcalcul
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret ; выход из подпрограммы
;-----
;Подпрограмма вычисления "3x-1"
_subcalcul:
    mov ebx,3
    mul ebx
    add eax,-1
    ret

```

6. Компилируем, запускаем и проверяем корректность работы.

```

nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit lab09-1-1.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf lab09-1-1.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-1-1 lab09-1-1.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-1-1
Введите x: 5
2(3x-1)+7=35

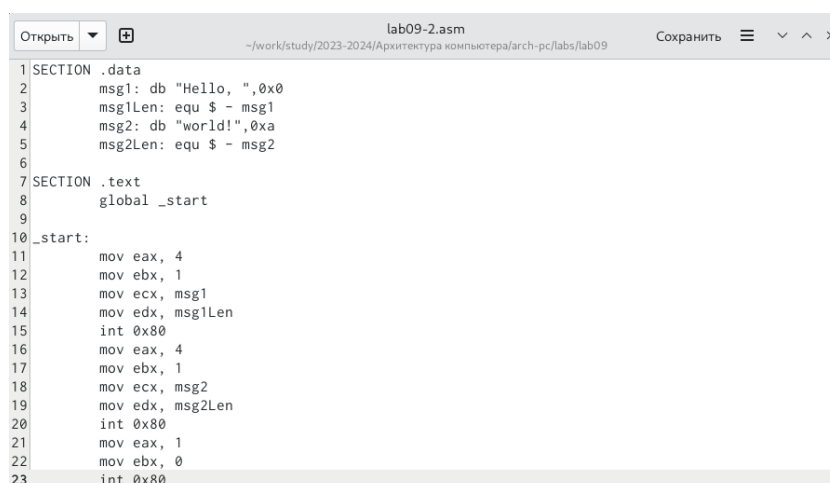
```

Рис. 3.5: Исполнение lab09-1-1

## 4 Отладка программ: точки останова

Данный раздел посвящён, в основном, точкам останова: местам в программе, на которых должно приостановиться её исполнение. Это необходимо, чтобы тщательнее изучить работу программы.

1. Для экспериментов с отладчиком **GDB** создаём файл *“Hello, world!”*



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

Рис. 4.1: Код программы lab09-2

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
```

2. Компилируем её с ключом **-g**, чтобы работать с ней в отладчике. Запустим для проверки корректности работы и открываем исполняемый файл с командой **gdb**.

```

nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ touch lab09-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit lab09-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf -g -l lab0
9-2.lst lab09-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab0
9-2 lab09-2.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./lab09-2
Hello, world!
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/a
rch-pc/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3810) exited normally]
(gdb)

```

Рис. 4.2: Открытие программы lab90-2 с gdb

3. В первую очередь, поставим точку останова на самое начало программы - метку *start*.

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/a
rch-pc/labs/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb)

```

Рис. 4.3: Точка останова по метке

4. Посмотрим преобразованный код с помощью команды *disassemble* с метки *start*.

```

(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. 4.4: Дисасемблированный код в АТТ



5. Посмотрим на этот код в синтаксисе **Intel**. Видим, что его особенность в наличии подсветки и замене местами регистров и адресов значений в них.

```
(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. 4.5: Дисасемблированный код в Intel

6. Запустим псевдографику командами *layout* с аргументами *asm*, для просмотра кода, и *regs*, для просмотра значений регистров.

```
[ Register Values Unavailable ]

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

native process 3846 In: _start                               L11  PC: 0x8049000
```

Рис. 4.6: Псевдографика gdb

7. Вспомним, где мы поставили метку, с помощью сокращённой версии команды *info breakpoints*.

```
native process 3846 In: _start                               L11  PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num   Type      Disp Enb Address      What
1     breakpoint keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
(gdb)
```

Рис. 4.7: Информация о точках останова

8. Запустим нашу программу командой *run*. Увидим, что она остановила своё выполнение на 1-й и пока единственной точке останова на метке *start*.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc210 0xffffc210
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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

native process 3952 In: _start L11 PC: 0x8049000
1 breakpoint keep y 0x8049000 lab09-2.asm:11
  breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/s
tudy/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:11
(gdb) |
```

Рис. 4.8: Запуск программы lab09-2 в gdb

9. Поставим ещё одну точку останова на предпоследней инструкции по её адресу. Проверим командой *i b*.

```
--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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

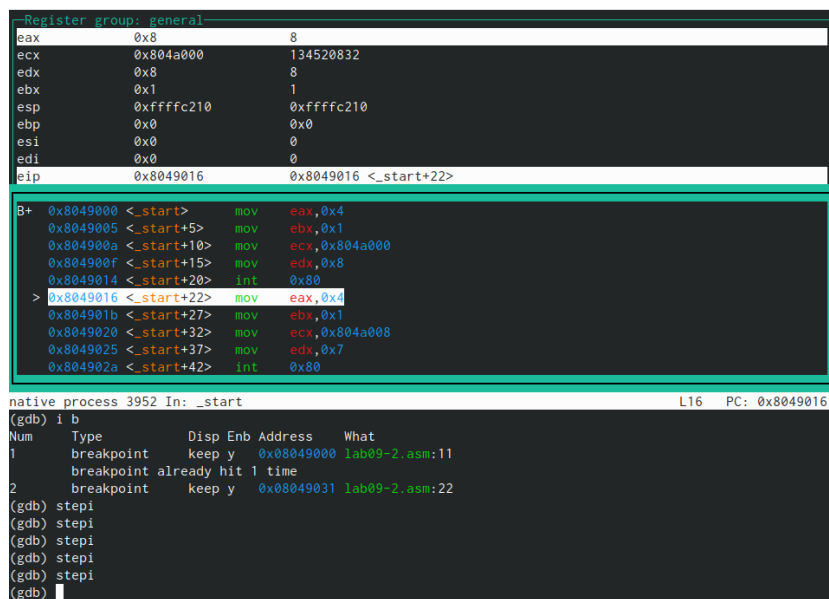
native process 3952 In: _start L11 PC: 0x8049000
udy/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:11
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 22.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 lab09-2.asm:22
(gdb) 
```

Рис. 4.9: Точка останова по адресу

## 5 Работа с данными через GDB

Программу гораздо легче исправлять, если видны изменения её объектов “*step\_by\_step*”.

1. Выполним несколько инструкций *stepi*. Так мы реализуем её пошаговое исполнение от инструкции к инструкции.



The screenshot shows the GDB interface with the following content:

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc210 0xfffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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

native process 3952 In: _start L16 PC: 0x8049016
(gdb) i b
Num  Type      Disp Enb Address  What
1    breakpoint keep y 0x08049000 lab09-2.asm:11
    breakpoint already hit 1 time
2    breakpoint keep y 0x08049031 lab09-2.asm:22
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) 
```

Рис. 5.1: Пошаговое исполнение программы

2. Посмотрим на текущие значения регистров командой *info registers*.

```

--Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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

native process 3952 In: _start L16 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 5.2: Информация о состоянии регистров

3. Выведем на экран текущее значений переменной *msg1* по её названию.

```

eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 5.3: Значение переменной *msg1*

4. Также можно выводить значение переменной по её адресу. Выведем так значение *msg2*.

```

--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 5.4: Значение переменной *msg2*

5. Изменим значение переменной *msg1* командой *set*. Для этого укажем тип данных, название переменной (обязательно с символом “&”!), и значение,

которое хотим записать. При этом у нас поменяется только первый символ, т.к. *msg1* занимает несколько бит, а мы изменяем только первый. Таким образом имя переменной здесь работает непосредственно как указатель на первую ячейку выделенной под неё памяти.

```
(gdb) si
(gdb) si
(gdb) si
(gdb) set (char)&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
```

Рис. 5.5: Изменение значения переменной *msg1*

- Изменим значения символов *msg2*, указывая адреса ячеек, чтобы поменять не только первый символ. Сокращённой версией команды *continue* завершим исполнение программы. Видим успешную замену и вывод *Hello, Lord!*

```
eip 0x8049000 0x8049000 <_start>
B+> 0x8049000 <_start> mov eax,0x4
B+> 0x8049000 <_start> mov eax,0x4
B+> 0x8049000 <_start> mov eax,0x4a000
B+> 0x8049000 <_start> mov eax,0x4
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
native No process in: L?? PC: ??
Value 0process 4257 In: _startinteger. L16 PC: 0x8049016
(gdb) sprocess 4293 In: _start L11 PC: 0x8049000
Continueprocess 4294 In: _start L11 PC: 0x8049000
(gdb) rprocess 4346 In: _start L11 PC: 0x8049000
(gdb) sNo process in: L?? PC: ??
(gdb) set (char)&msg1='h'
(gdb) set (char)0x804a008='L'
(gdb) set (char)0x804a00a='r'
(gdb) set (char)0x804a00b='d'
(gdb) set (char)0x804a00c='!'
(gdb) set (char)0x804a00d=' '
(gdb) c
Continuing.
hello, Lord!
[Inferior 1 (process 4346) exited normally]
(gdb)
```

Рис. 5.6: Завершение исполнения lab09-2

- Также можно изменять значения регистра. Изменим значение *ebx* на символную '2' и выведем в разных форматах. Видим, что формат */s* выводит нам номер этого символа в десятичной форме, а форматы */t* и */x* - в двоичной и шестнадцатиричной формах.

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$10 = 50  
(gdb) p/t $ebx  
$11 = 110010  
(gdb) p/x $ebx  
$12 = 0x32  
(gdb) █
```

Рис. 5.7: Изменение значения регистра

## 6 Работа с аргументами через GDB

Данный раздел посвящён отладке программы, запуск которой предполагает ввод аргументов.

1. Скопируем программу вывода введённых аргументов на экран из 8-й лабораторной работы. Дадим ему новое имя и скомпилируем, не забыв ключ **-g**.

```
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ cp ../lab08/lab8-2.asm lab09-3.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 6.1: Копирование и компиляция lab09-3

```
%include 'in_out.asm'
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
    pop ecx      ; Извлекаем из стека в `ecx` количество
                  ; аргументов (первое значение в стеке)
```

```
    pop edx      ; Извлекаем из стека в `edx` имя программы
                  ; (второе значение в стеке)
```

```
    sub ecx, 1   ; Уменьшаем `ecx` на 1 (количество
```



; аргументов без названия программы)

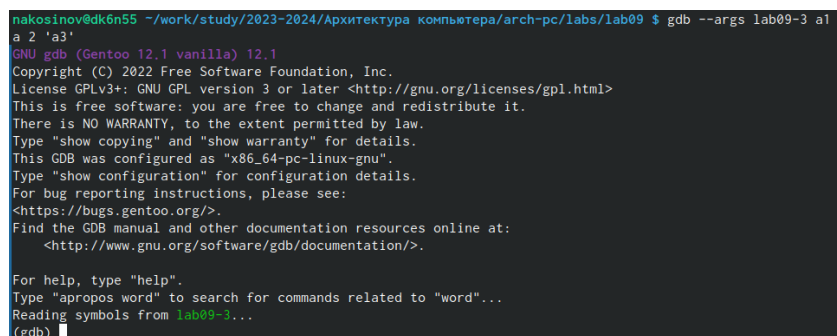
next:

```
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end    ; если аргументов нет выходим из цикла
          ; (переход на метку `_end`)
pop eax    ; иначе извлекаем аргумент из стека
call printf ; вызываем функцию печати
loop next  ; переход к обработке следующего
          ; аргумента (переход на метку `next`)
```

\_end:

```
call quit
```

2. Откроем программу в **GDB**, указав ключ **-args** и 4 аргумента разных форматов.



```
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gdb --args lab09-3 a1
a 2 'a3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-3...
(gdb)
```

Рис. 6.2: Открытие lab09-3 в gdb

3. Установим точку останова на метку *start* и запустим программу. Выведем на экран значение регистра *esp*. Увидим адрес вершины стека и значение, там расположенное - 5.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3 a1 a 2 a3

Breakpoint 1, _start () at lab09-3.asm:7
7       pop ecx          ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffc1f0: 0x00000005
(gdb)

```

Рис. 6.3: Вершина стека аргументов

4. 5 на 1 больше числа введенных нами аргументов. Это потому, что в стеке также хранится название самой программы. Убедимся в этом, выведя на экран все элементы, хранящиеся в стеке. Указываем шаг изменения адреса “+4”, т.к. на хранение одного адреса выделены 4б.

```

(gdb) x/s *(void**)(esp+4)
0xffffc48d: "/afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffc514: "a1"
(gdb) x/s *(void**)(esp+12)
0xffffc517: "a"
(gdb) x/s *(void**)(esp+16)
0xffffc519: "2"
(gdb) x/s *(void**)(esp+20)
0xffffc51b: "a3"
(gdb) x/s *(void**)(esp+24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 6.4: Адреса и значения элементов стека

## 7 Самостоятельная работа

В первой задаче самостоятельной работы предлагается взять программу из восьмой лабораторной и оформить вычисление значения  $f(x) = 2x + 15$  как подпрограмму. Во второй задаче дан код, выдающий неверный результат и требуется найти и исправить ошибку.

1. Скопируем самостоятельную работу из предыдущей лабораторной и изменим код, вынеся вычисление функции в подпрограмму.

```
28      ; ----- Считаем в цикле f(x) и прибавляем к буферу esi
29 next:
30      pop eax          ; извлекаем следующий аргумент из стека
31      call atoi        ; преобразуем символ в число
32      call _func       ; считаем выражение f(x)
33      add esi, eax
34      loop next       ; переход к обработке следующего аргумента
35
36      ; ----- Вывод результата работы программы
37 _end:
38      mov eax, msg2    ; вывод сообщения "Результат: "
39      call sprintf
40      mov eax, esi     ; записываем произведение в регистр 'eax'
41      call iprintfLF   ; печать результата
42      call quit        ; завершение программы
43
44      ; ----- Считаем значение f(x)
45 _func:
46      mov ebx, 2
47      mul ebx
48      add eax, 15
49      ret
```

Рис. 7.1: Код программы sr-1

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg0 db "Вариант I",0
```

```
msg1 db "Функция: f(x) = 2x+15 ",0
```

```
msg2 db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
; ----- Вывод приглашающих сообщений
```

```
mov eax, msg0
```

```
call sprintf
```

```
mov eax, msg1
```

```
call sprintf
```

```
; ----- Подготовка стека аргументов и буфера для хранения промежуточных сум
```

```
pop ecx
```

```
pop edx
```

```
sub ecx,1
```

```
mov esi, 0
```

```
; ----- Проверяем, есть ли аргументы
```

```
cmp ecx,0h
```

```
jz _end
```

```
; ----- Считаем в цикле f(x) и прибавляем к буферу esi
```

```
next:
```

```
pop eax ; извлекаем следующий аргумент из стека
```

```
call atoi ; преобразуем символ в число
```

```
call _func ; считаем выражение f(x)
```

```
add esi, eax
```

```
loop next ; переход к обработке следующего аргумента
```

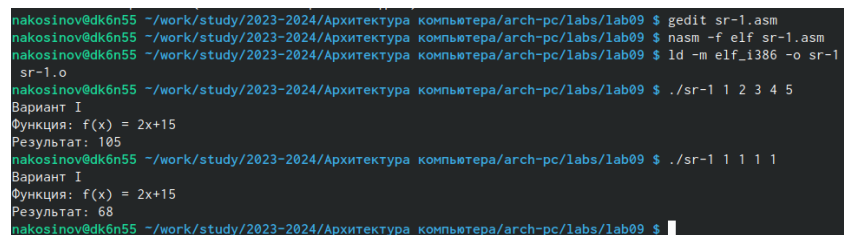
```

; ----- Вывод результата работы программы
_end:
    mov eax, msg2    ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi      ; записываем произведение в регистр `eax`
    call iprintLF     ; печать результата
    call quit         ; завершение программы

; ----- Считаем значение f(x)
_func:
    mov ebx, 2
    mul ebx
    add eax, 15
    ret

```

## 2. Запустим и убедимся в корректности работы программы



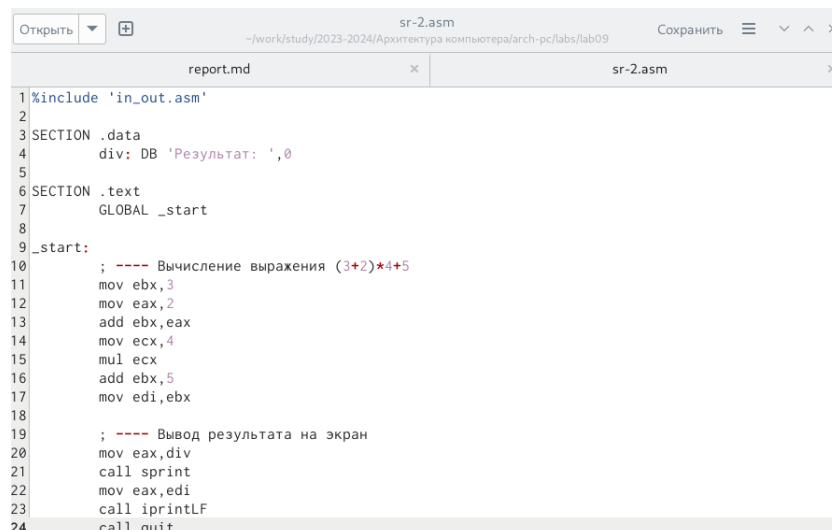
```

nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit sr-1.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -f elf sr-1.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o sr-1 sr-1.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./sr-1 1 2 3 4 5
Вариант I
Функция: f(x) = 2x+15
Результат: 105
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./sr-1 1 1 1 1 1
Вариант I
Функция: f(x) = 2x+15
Результат: 68
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $

```

Рис. 7.2: Исполнение sr-1

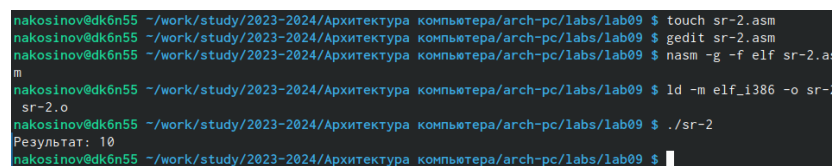
## 3. Для решения второй задачи скопируем код в sr-2.



```
1 %include 'in_out.asm'
2
3 SECTION .data
4     div: DB 'Результат: ',0
5
6 SECTION .text
7     GLOBAL _start
8
9 _start:
10    ; ---- Вычисление выражения (3+2)*4+5
11    mov ebx,3
12    mov eax,2
13    add ebx,eax
14    mov ecx,4
15    mul ecx
16    add ebx,5
17    mov edi,ebx
18
19    ; ---- Вывод результата на экран
20    mov eax,div
21    call sprint
22    mov eax,edi
23    call iprintLF
24    call quit
```

Рис. 7.3: Код программы sr-2

4. Скомпилируем и запустим. Ответ, который мы ожидаем увидеть  $(2 + 3) \times 4 + 5 = 25$ , но результат программы не тот.



```
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ touch sr-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit sr-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -g -f elf sr-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o sr-2
sr-2.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./sr-2
Результат: 10
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 7.4: Исполнение sr-2

5. Откроем *sr-2* в *gdb*. Поставим точку останова на метке *start* и запустим программу командой *run*.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>

B+> 11      mov ebx,3
    12      mov eax,2
    13      add ebx,eax
    14      mov ecx,4
    15      mul ecx
    16      add ebx,5
    17      mov edi,ebx
    18
    19      ; ---- Вывод результата на экран
    20      mov eax,div

native process 5222 In: _start L11 PC: 0x80490e8
(gdb) break _start
Breakpoint 1 at 0x80490e8: file sr-2.asm, line 11.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/sr-2

Breakpoint 1, _start () at sr-2.asm:11
(gdb) si
```

Рис. 7.5: Отладка sr-2

6. Прогоняем программу командами *si*, по шагам. Первое, что бросается в глаза - результат сложения в строке 13 увеличил регистр *ebx*. Теперь там лежит значение 5.

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>

sr-2.asm
B+> 11      mov ebx,3
    12      mov eax,2
    13      add ebx,eax
    14      mov ecx,4
    15      mul ecx
    16      add ebx,5
    17      mov edi,ebx
    18
    19      ; ---- Вывод результата на экран
    20      mov eax,div

native process 5222 In: _start L16 PC: 0x80490fb
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/sr-2

Breakpoint 1, _start () at sr-2.asm:11
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 7.6: Третий шаг исполнения sr-2

7. Ошибка: команда *mul* в строке 15 увеличивает значения регистра *eax*, но промежуточные значения мы храним в *ebx*. Следовательно ошибка логическая.

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>

--sr-2.asm
B+ 11      mov ebx,3
    12      mov eax,2
    13      add ebx,eax
> 14      mov ecx,4
    15      mul ecx
    16      add ebx,5
    17      mov edi,ebx
    18
    19      ; ---- Вывод результата на экран
    20      mov eax,div

native process 5222 In: _start L14 PC: 0x80490f4
(gdb) break _start
Breakpoint 1 at 0x80490e8: file sr-2.asm, line 11.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/a/nakosinov/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09/sr-2

Breakpoint 1, _start () at sr-2.asm:11
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Рис. 7.7: Код программы sr-1

8. Нам надо поменять либо место хранения промежуточных значений, либо перед использованием *mul* поместить в *eax* значение из *ebx*, а затем выгрузить его обратно. Применим первый способ, как наименее трудоёмкий.



report.md	sr-2.asm
1	%include 'in_out.asm'
2	
3	SECTION .data
4	div: DB 'Результат: ',0
5	
6	SECTION .text
7	GLOBAL _start
8	
9	_start:
10	; ---- Вычисление выражения (3+2)*4+5
11	mov ebx,3
12	mov eax,2
13	add eax,ebx ; Меняем местами регистры eax и ebx
14	mov ecx,4
15	mul ecx
16	add eax,5 ; Хранение промежуточного результата сейчас в eax
17	mov edi,eax
18	
19	; ---- Вывод результата на экран
20	mov eax,div
21	call sprint
22	mov eax,edi
23	call iprintLF
24	call quit

Рис. 7.8: Исправленный код программы sr-2

```
%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 ; Меняем местами регистры eax и ebx
```

```
mov ecx,4
```

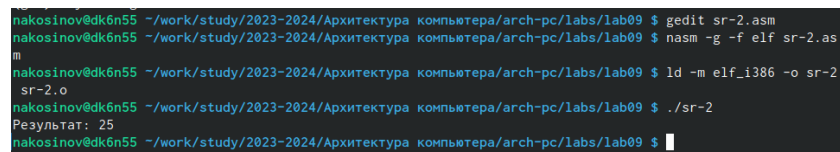
```
mul ecx
```

```
add eax,5 ; Хранение промежуточного результата сейчас в eax
```

```
mov edi,eax
```

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

9. Компилируем и запускаем. Ответ верный, следовательно, ошибка исправлена успешно!



```
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ gedit sr-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ nasm -g -f elf sr-2.asm
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ld -m elf_i386 -o sr-2 sr-2.o
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $ ./sr-2
Результат: 25
nakosinov@dk6n55 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab09 $
```

Рис. 7.9: Исполнение исправленной sr-2

## 8 Выводы

В ходе данной лабораторной работы мы научились двум основным процессам программирования: выделение логических участков кода - подпрограмм, а также работе с отладчиком **GDB**, упрощающим поиск ошибок в коде или работе программы.