

Отчёт по лабораторной работе 9

Архитектура компьютера

Диденко Дмитрий Владимирович НПИбд-03-23

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	9
2.2.1	Точки остановки	13
2.2.2	Работа с данными программы в GDB	14
2.2.3	Обработка аргументов командной строки в GDB	19
2.3	Задание для самостоятельной работы	21
3	Выводы	28

Список иллюстраций

2.1	Код программы lab9-1.asm	7
2.2	Компиляция и запуск программы lab9-1.asm	7
2.3	Код программы lab9-1.asm	8
2.4	Компиляция и запуск программы lab9-1.asm	9
2.5	Код программы lab9-2.asm	10
2.6	Компиляция и запуск программы lab9-2.asm в отладчике	11
2.7	Дизассемблированный код	12
2.8	Дизассемблированный код в режиме интел	13
2.9	Точка остановки	14
2.10	Изменение регистров	15
2.11	Изменение регистров	16
2.12	Изменение значения переменной	17
2.13	Вывод значения регистра	18
2.14	Вывод значения регистра	19
2.15	Вывод значения регистра	20
2.16	Код программы prog-1.asm	22
2.17	Компиляция и запуск программы prog-1.asm	23
2.18	Код с ошибкой	24
2.19	Отладка	25
2.20	Код исправлен	26
2.21	Проверка работы	27

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 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Для начала я создал новую директорию, в которой планировал выполнять лабораторную работу номер 9, и перешел в нее. Затем я создал файл с именем lab9-1.asm.

В качестве примера рассмотрим программу, которая вычисляет арифметическое выражение $f(x) = 2x + 7$ с использованием подпрограммы calcul. В данном примере значение переменной x вводится с клавиатуры, а само выражение вычисляется внутри подпрограммы. (рис. [2.1]) (рис. [2.2])

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09
GNU nano 4.8 /home/dvdidenko/work/arch-pc/lab09/lab9-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rez: RESB 80

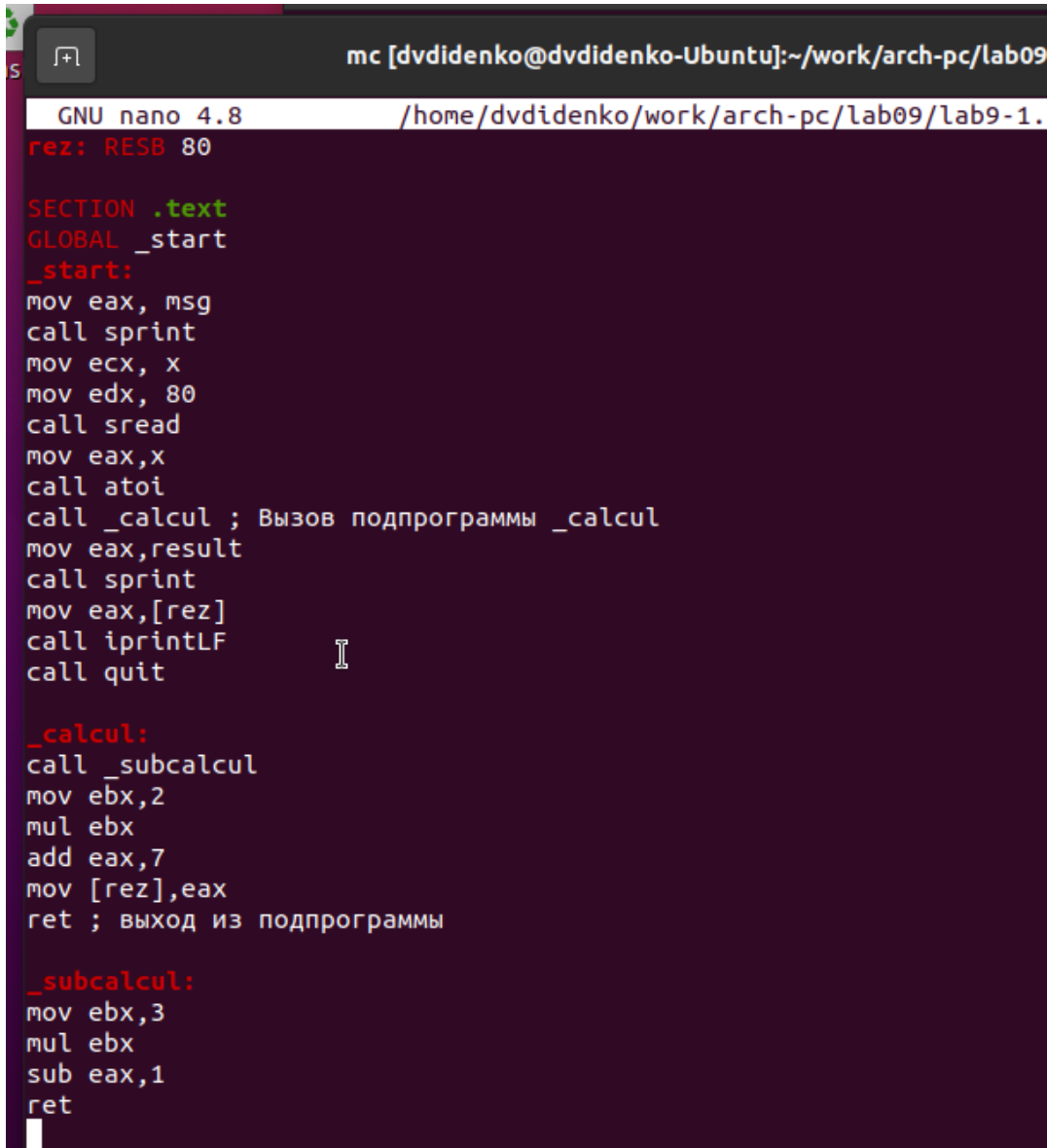
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[rez]
call iprintLF
call quit
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы
```

Рис. 2.1: Код программы lab9-1.asm

```
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2x+7=15
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.2: Компиляция и запуск программы lab9-1.asm

После этого я внес изменения в текст программы, добавив подпрограмму `subcalcul` внутрь подпрограммы `calcul`. Это позволяет вычислить составное выражение $f(g(x))$, где значение x также вводится с клавиатуры. Функции определены следующим образом: $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. [2.3]) (рис. [2.4])



```
mc [dvdidenko@dvdidenko-Ubuntu]:~/work/arch-pc/lab09
GNU nano 4.8 /home/dvdidenko/work/arch-pc/lab09/lab9-1.
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. 2.3: Код программы lab9-1.asm


```
dvddenko@dvddenko-Ubuntu: ~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dvddenko@dvddenko-Ubuntu: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dvddenko@dvddenko-Ubuntu: ~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2(3x-1)+7=29
dvddenko@dvddenko-Ubuntu: ~/work/arch-pc/lab09$
```

Рис. 2.4: Компиляция и запуск программы lab9-1.asm

2.2 Отладка программ с помощью GDB

Я создал файл с именем lab9-2.asm, в котором содержится текст программы из Листинга 9.2. Эта программа отвечает за печать сообщения “Hello world!”. (рис. [2.5])

A screenshot of a terminal window with a dark background. At the top, a status bar shows 'mc [dvdidenko@dvdidenko-Ubun'. Below it, the terminal header reads 'GNU nano 4.8' and the file path '/home/dvdidenko/work/ar'. The main content is assembly code. It starts with 'SECTION .data' in green, followed by 'msg1: db "Hello, ",0x0', 'msg1Len: equ \$ - msg1', 'msg2: db "world!",0xa', and 'msg2Len: equ \$ - msg2'. Then, 'SECTION .text' is shown in green, followed by 'global _start'. A blank line is present, then '_start:' in red. The code continues with two identical blocks of instructions: 'mov eax, 4', 'mov ebx, 1', 'mov ecx, msg1', 'mov edx, msg1Len', 'int 0x80', followed by 'mov eax, 4', 'mov ebx, 1', 'mov ecx, msg2', 'mov edx, msg2Len', 'int 0x80', 'mov eax, 1', 'mov ebx, 0', and 'int 0x80'.

Рис. 2.5: Код программы lab9-2.asm

После этого я скомпилировал файл и получил исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, использовал ключ “-g”.

Затем я загрузил полученный исполняемый файл в отладчик GDB и проверил его работу, запустив программу с помощью команды “run” или “r”. (рис. [2.6])

```

dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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/dvdidenko/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6080) exited normally]
(gdb)

```

Рис. 2.6: Компиляция и запуск программы lab9-2.asm в отладчике

Для более детального анализа программы, я установил точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустил ее. Затем я просмотрел дизассемблированный код программы.(рис. [2.7]) (рис. [2.8])

```

[Inferior 1 (process 6080) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/dvddenko/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x08049000 in _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. 2.7: Дизассемблированный код

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/dvdidenko/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x08049000 in _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) 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. 2.8: Дизассемблированный код в режиме интел

2.2.1 Точки остановки

Чтобы проверить точку остановки по имени метки “_start”, я использовал команду “info breakpoints” или “i b”. Затем установил еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0” (рис. [2.9])

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35

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
b+ 0x8049031 <_start+49> mov    ebx,0x0

native process 6084 In: _start L?? PC: 0x8049000
(gdb)
(gdb)
(gdb) b *0x8049031Breakpoint 2 at 0x8049031
(gdb) i b
Num    Type      Disp Enb Address    What
1      breakpoint keep y  0x08049000 <_start>
       breakpoint already hit 1 time
2      breakpoint keep y  0x08049031 <_start+49>
(gdb) 
```

Рис. 2.9: Точка остановки

2.2.2 Работа с данными программы в GDB

В отладчике GDB можно просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Выполнил 5 инструкций с помощью команды 'stepi' (сокращенно 'si') и отследил изменение значений регистров. (рис. [2.10]) (рис. [2.11])

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35

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
b+ 0x8049031 <_start+49> mov ebx,0x0

native process 6084 In: _start L?? PC: 0x8049005
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
0x08049005 in _start ()
(gdb)
```

Рис. 2.10: Изменение регистров

The screenshot shows a GDB debugger window with the title bar "dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09". The window is divided into several sections:

- Register group: general**: A table showing the current values of general-purpose registers.

Register	Value	Comment
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd1d0	0xffffd1d0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 < _start+22>
eflags	0x202	[IF]
cs	0x23	35
- Assembly code**: A list of instructions with their addresses and disassembled form.

```
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
b+ 0x8049031 <_start+49> mov     ebx,0x0
```
- Process information**: Shows the current process and instruction pointer.

native process 6084 In: _start L?? PC: 0x8049016
- GDB commands and output**: A list of commands entered in the GDB prompt and their outputs.

```
(gdb) si
0x08049005 in _start ()
(gdb) si
0x0804900a in _start ()
(gdb) si
0x0804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb) si
0x08049016 in _start ()
(gdb) 
```

Рис. 2.11: Изменение регистров

Просмотрел значение переменной `msg1` по имени и получил нужные данные.
Просмотрел значение переменной `msg1` по имени и получил нужные данные.
Для изменения значения регистра или ячейки памяти использовал команду `set`, указав имя регистра или адрес в качестве аргумента. Изменил первый символ переменной `msg1`. (рис. [2.12])


```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 < _start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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
b+ 0x8049031 < _start+49>     mov     ebx,0x0

native process 6084 In: _start L?? PC: 0x8049016
(gdb) si
0x08049016 in _start ()
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>: "Hello, "
(gdb)
(gdb) x/1sb 0x804a0080x804a008 <msg2>: "world!\n"
(gdb)
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>: "hello, "
(gdb)
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "Lorld!\n"
(gdb)
```

Рис. 2.12: Изменение значения переменной

Для изменения значения регистра или ячейки памяти использовал команду `set`, указав имя регистра или адрес в качестве аргумента. Изменил первый символ переменной `msg1`. (рис. [2.13])

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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
b+ 0x8049031 <_start+49> mov    ebx,0x0

native process 6084 In: _start L?? PC: 0x8049016
(gdb)
(gdb) p/t $eax$2 = 1000
(gdb)
(gdb) p/s $ecx$3 = 134520832
(gdb)
(gdb) p/x $ecx$4 = 0x804a000
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```

Рис. 2.13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx на нужное значение.
(рис. [2.14])

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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
b+ 0x8049031 <_start+49> mov    ebx,0x0

native process 6084 In: _start L?? PC: 0x8049016
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb) p/s $ebx
$9 = 2
(gdb) 
```

Рис. 2.14: Вывод значения регистра

2.2.3 Обработка аргументов командной строки в GDB

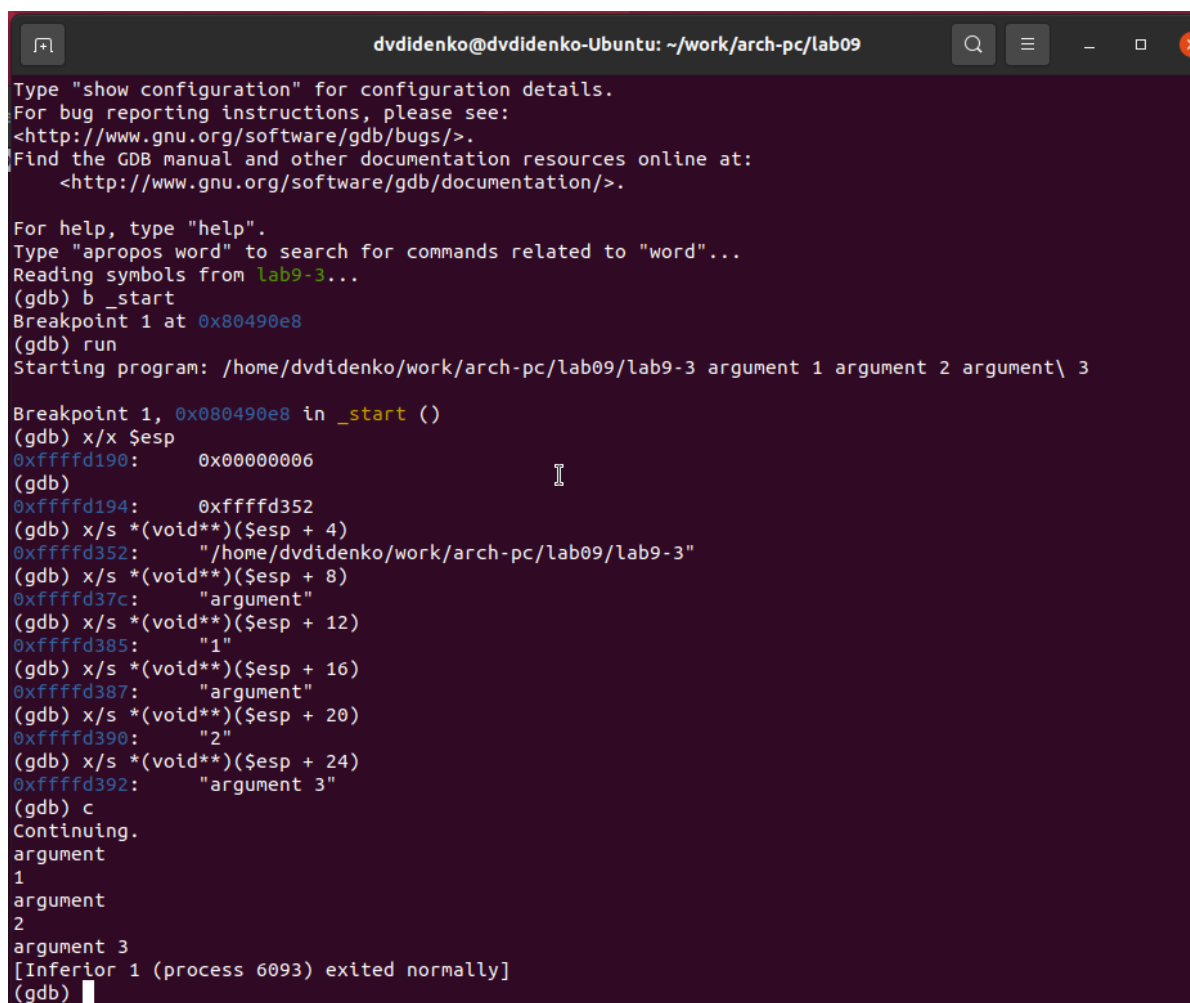
Скопировал файл lab8-2.asm, созданный во время выполнения лабораторной работы №8, который содержит программу для вывода аргументов командной строки. Создал исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в gdb использовал ключ `-args` и загрузил исполняемый файл в отладчик с указанными аргументами.

Установил точку останова перед первой инструкцией программы и запустил 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'.

Просмотрел 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])



```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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
(gdb) run
Starting program: /home/dvdidenko/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd190: 0x00000006
(gdb)
0xffffd194: 0xffffd352
(gdb) x/s *(void**)($esp + 4)
0xffffd352: "/home/dvdidenko/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd37c: "argument"
(gdb) x/s *(void**)($esp + 12)
0xffffd385: "1"
(gdb) x/s *(void**)($esp + 16)
0xffffd387: "argument"
(gdb) x/s *(void**)($esp + 20)
0xffffd390: "2"
(gdb) x/s *(void**)($esp + 24)
0xffffd392: "argument 3"
(gdb) c
Continuing.
argument
1
argument
2
argument 3
[Inferior 1 (process 6093) exited normally]
(gdb)
```

Рис. 2.15: Вывод значения регистра

Шаг изменения адреса равен 4, так как каждый следующий адрес на стеке на-

ходится на расстоянии 4 байт от предыдущего ([esp+4], [esp+8], [esp+12]).

2.3 Задание для самостоятельной работы

Преобразовал программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [2.16]) (рис. [2.17])

```
mc [dvdidenko@dvdidenko-Ubuntu]:~/work/arch-pc/lab05
GNU nano 4.8 /home/dvdidenko/work/arch-pc/lab05
SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

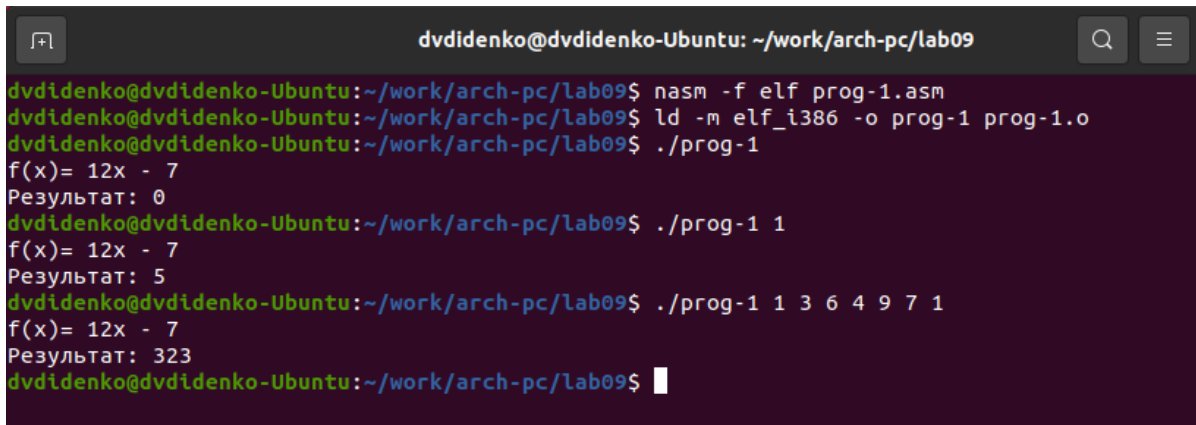
next:
cmp ecx,0h
jz _end
pop eax
call atoi
call _fx
add esi,eax

loop next

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

_fx:
mov ebx,12
mul ebx
sub eax,7
ret
```

Рис. 2.16: Код программы prog-1.asm

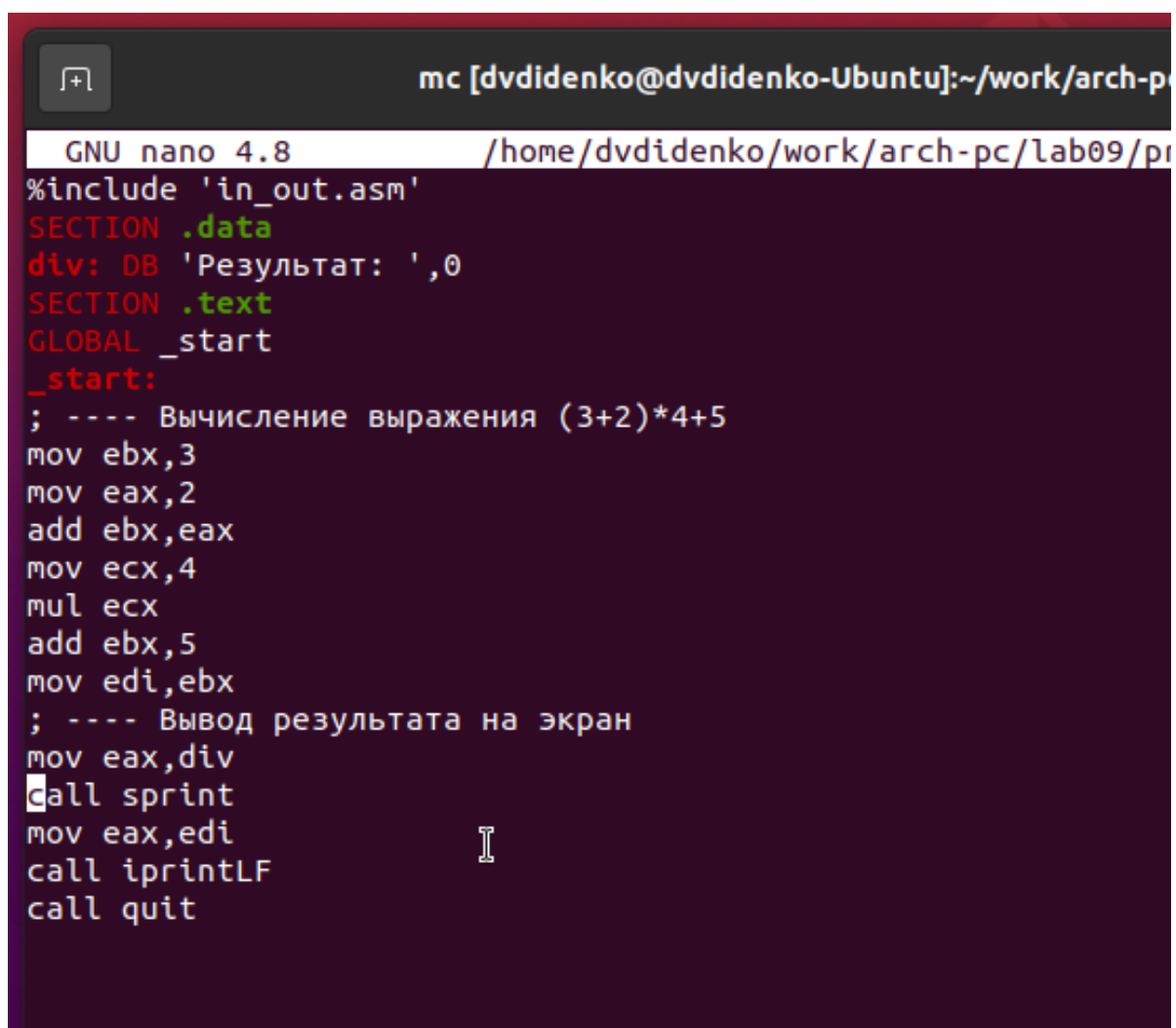
A terminal window with a dark background and light text. The title bar at the top reads 'dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09'. The terminal shows the following commands and output:

```
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ nasm -f elf prog-1.asm
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o prog-1 prog-1.o
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ./prog-1
f(x)= 12x - 7
Результат: 0
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ./prog-1 1
f(x)= 12x - 7
Результат: 5
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$ ./prog-1 1 3 6 4 9 7 1
f(x)= 12x - 7
Результат: 323
dvdidenko@dvdidenko-Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.17: Компиляция и запуск программы prog-1.asm

В листинге приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверил это, анализируя изменения значений регистров с помощью отладчика GDB.

Определил ошибку - перепутан порядок аргументов у инструкции add. Также обнаружил, что по окончании работы в edi отправляется ebx вместо eax.(рис. [2.18])



```
mc [dvdidenko@dvdidenko-Ubuntu]:~/work/arch-pc/
GNU nano 4.8 /home/dvdidenko/work/arch-pc/lab09/p1
#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 iprintLF
call quit
```

Рис. 2.18: Код с ошибкой


```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0
esi      0x0      0
edi      0xa      10
eip      0x8049100 0x8049100 <_start+24>
eflags   0x206    [ PF IF ]
cs       0x23     35

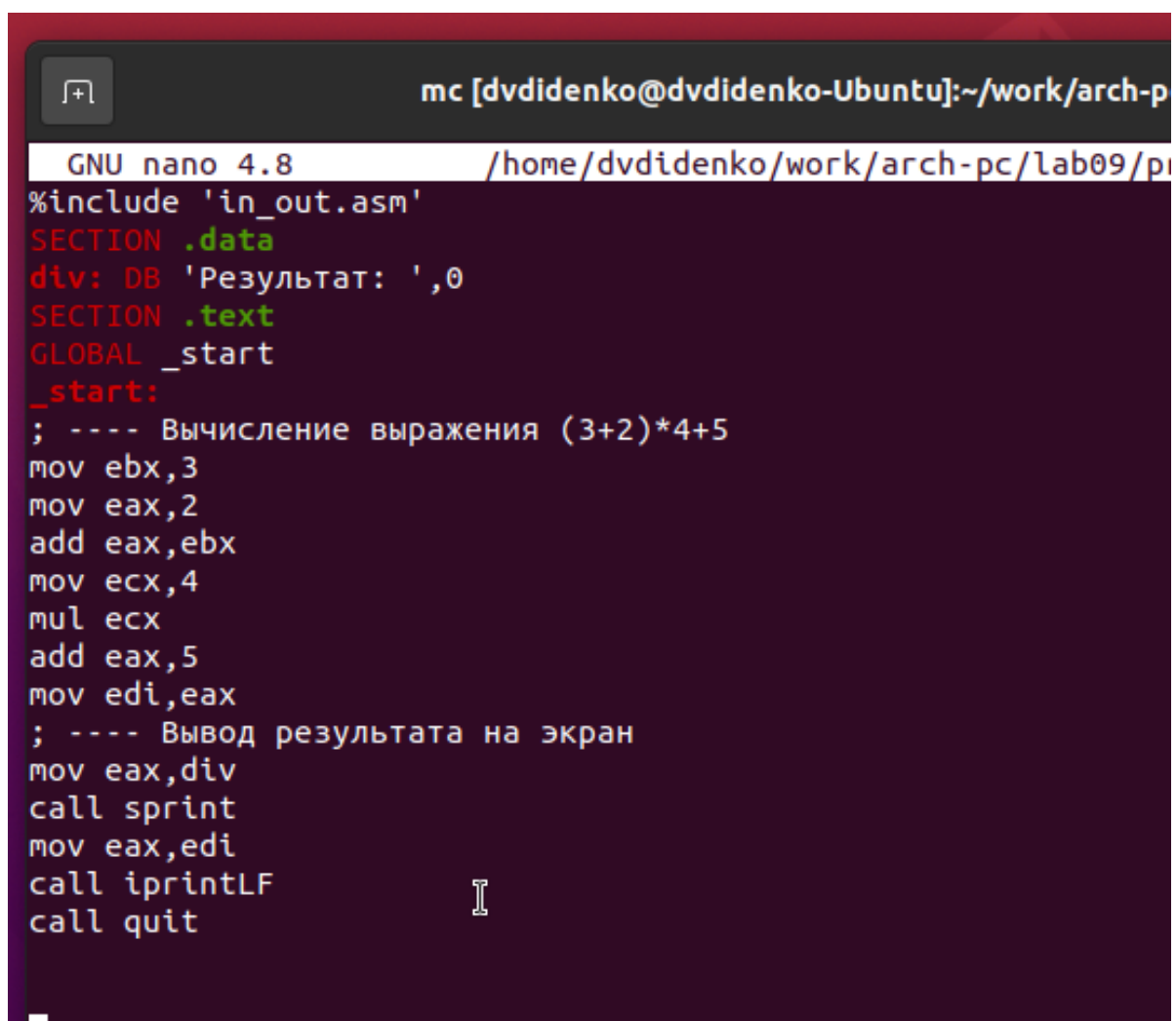
B+ 0x80490e8 <_start>      mov     ebx,0x3
B+ 0x80490e8 <_start+5>    mov     ebx,0x3
0x80490ed <_start+5>      mov     eax,0x2
0x80490f2 <_start+10>     add     ebx,eax
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx,0x5
0x80490fb <_start+19>     add     ebx,0x5
>0x80490fe <_start+22>    mov     edi,ebx
0x8049100 <_start+24>     mov     eax,0x804a000<print>
0x8049105 <_start+29>     call    0x804900f <sprint>
0x804910a <_start+34>     mov     eax,edi86 <iprintLF>
0x804910c <_start+36>     call    0x8049086 <iprintLF>
0x8049111 <_start+41>     call    0x80490db <quit>

native process 6164 In: _start
(gdb) No process In:
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 6164) exited normally]
(gdb)
```

Рис. 2.19: Отладка

Отмечу, что перепутан порядок аргументов у инструкции `add` и что по окончании работы в `edi` отправляется `ebx` вместо `eax` (рис. [2.19])

Исправленный код программы (рис. [2.20]) (рис. [2.21])



The image shows a terminal window with a dark background. At the top, a status bar indicates the user is 'mc [dvdidenko@dvdidenko-Ubuntu]:~/work/arch-p'. Below this, the terminal shows the GNU nano 4.8 editor interface. The file path is '/home/dvdidenko/work/arch-pc/lab09/pi'. The code being edited is assembly language. It starts with '%include \'in_out.asm\'', followed by a data section 'SECTION .data' containing 'div: DB \'Результат: ',0'. Then, a text section 'SECTION .text' begins with 'GLOBAL _start'. The main code starts at '_start:' with a comment '; ---- Вычисление выражения (3+2)*4+5'. The instructions are: 'mov ebx,3', 'mov eax,2', 'add eax,ebx', 'mov ecx,4', 'mul ecx', 'add eax,5', 'mov edi,eax'. Another comment '; ---- Вывод результата на экран' is followed by 'mov eax,div', 'call sprint', 'mov eax,edi', 'call iprintLF', and finally 'call quit'. A cursor is visible on the line 'call iprintLF'.

```
mc [dvdidenko@dvdidenko-Ubuntu]:~/work/arch-p
GNU nano 4.8 /home/dvdidenko/work/arch-pc/lab09/pi
%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. 2.20: Код исправлен

```
dvdidenko@dvdidenko-Ubuntu: ~/work/arch-pc/lab09

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1d0 0xffffd1d0
ebp      0x0       0
esi      0x0       0
edi      0x19      25
eip      0x8049100 0x8049100 <_start+24>
eflags   0x202     [ IF ]
cs       0x23      35

B+ 0x80490e8 <_start>      mov     ebx,0x3
B+ 0x80490e8 <_start+5>    mov     ebx,0x3
0x80490ed <_start+5>      mov     eax,0x2
0x80490f2 <_start+10>     add     eax,ebx
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx,0x5
0x80490fb <_start+19>     add     eax,0x5
>0x80490fe <_start+22>     mov     edi,eax
0x8049100 <_start+24>     mov     eax,0x804a000<rint>
0x8049105 <_start+29>     call    0x804900f<sprintf>
0x804910a <_start+34>     mov     eax,edi86 <lprintLF>
0x804910c <_start+36>     call    0x8049086<lprintLF>
0x8049111 <_start+41>     call    0x80490db<quit>

native process 6192 In: _start      L??  PC: 0x8049100
(gdb) No process In:                L??  PC: ??
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 6192) exited normally]
(gdb)
```

Рис. 2.21: Проверка работы

3 Выводы

Освоили работу с подпрограммами и отладчиком.