

# **Отчёт по лабораторной работе 9**

**Понятие подпрограммы. Отладчик GDB.**

Татьяна Соколова НММбд-03-24

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Реализация подпрограмм в NASM . . . . .	6
2.2	Отладка программ с помощью GDB . . . . .	9
2.3	Задание для самостоятельной работы . . . . .	21
<b>3</b>	<b>Выводы</b>	<b>27</b>

## Список иллюстраций

2.1	Программа lab9-1.asm . . . . .	7
2.2	Запуск программы lab9-1.asm . . . . .	7
2.3	Программа lab9-1.asm . . . . .	8
2.4	Запуск программы lab9-1.asm . . . . .	9
2.5	Программа lab9-2.asm . . . . .	10
2.6	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	Программа prog-1.asm . . . . .	21
2.17	Запуск программы prog-1.asm . . . . .	22
2.18	Код с ошибкой . . . . .	23
2.19	Отладка . . . . .	24
2.20	Код исправлен . . . . .	25
2.21	Проверка работы . . . . .	26

## Список таблиц

# 1 Цель работы

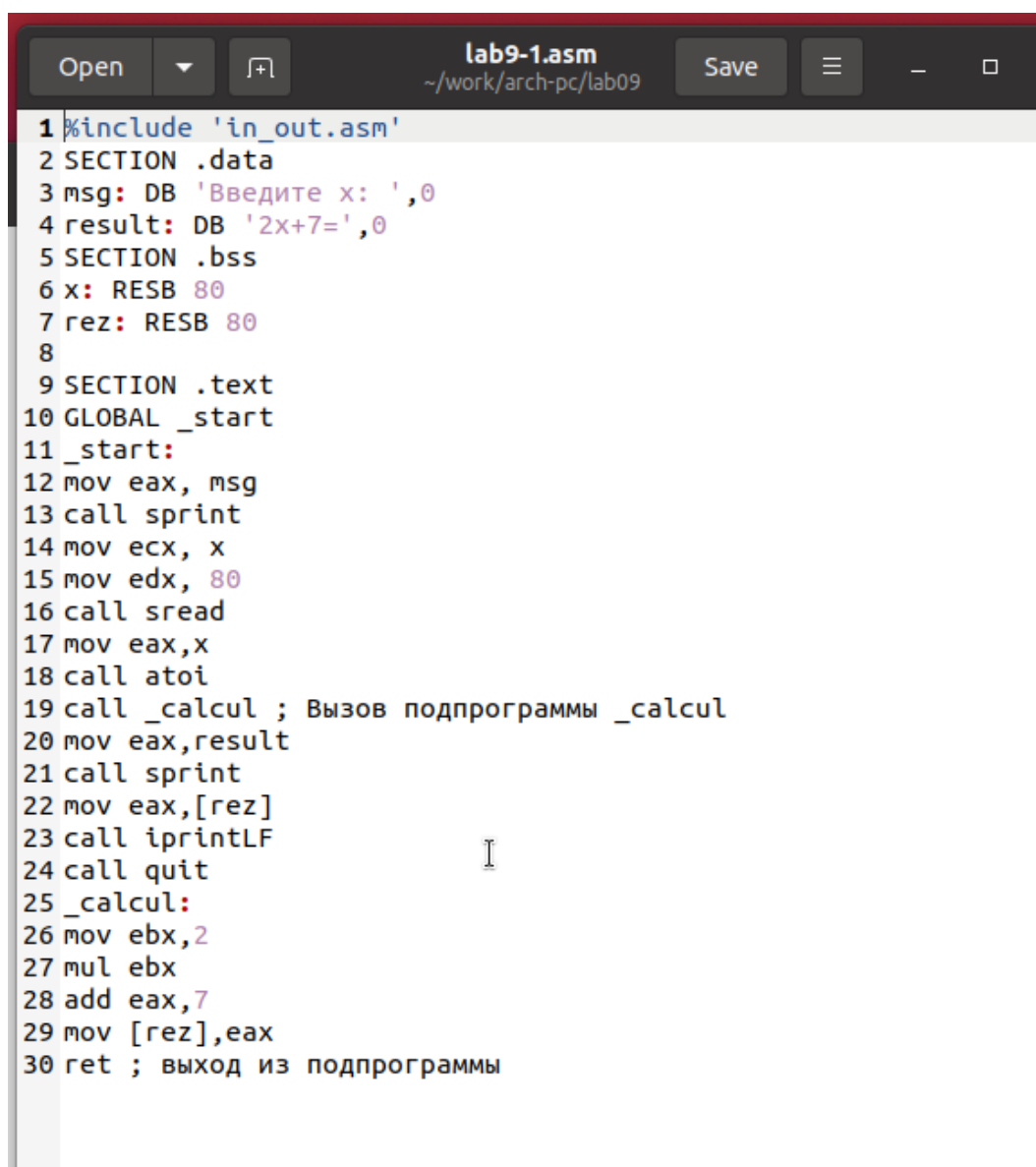
Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

## 2 Выполнение лабораторной работы

### 2.1 Реализация подпрограмм в NASM

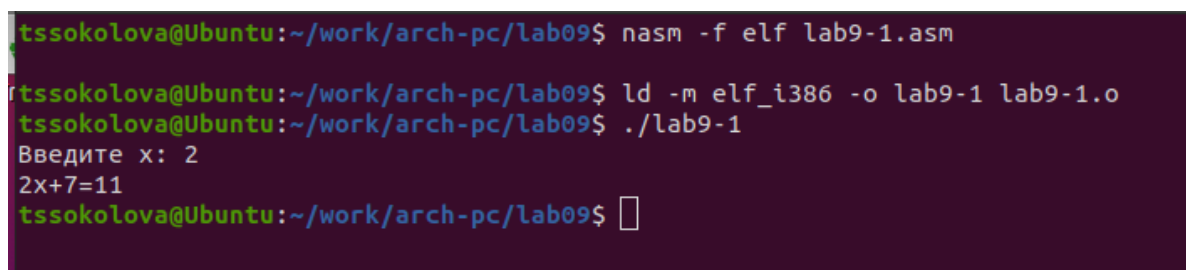
Сначала я создала новую папку, чтобы выполнять лабораторную работу номер 9, и перешла в нее. Затем я создала файл с именем lab9-1.asm.

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



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax, x
18 call atoi
19 call _calcul ; Вызов подпрограммы _calcul
20 mov eax, result
21 call sprint
22 mov eax, [rez]
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29 mov [rez], eax
30 ret ; выход из подпрограммы
```

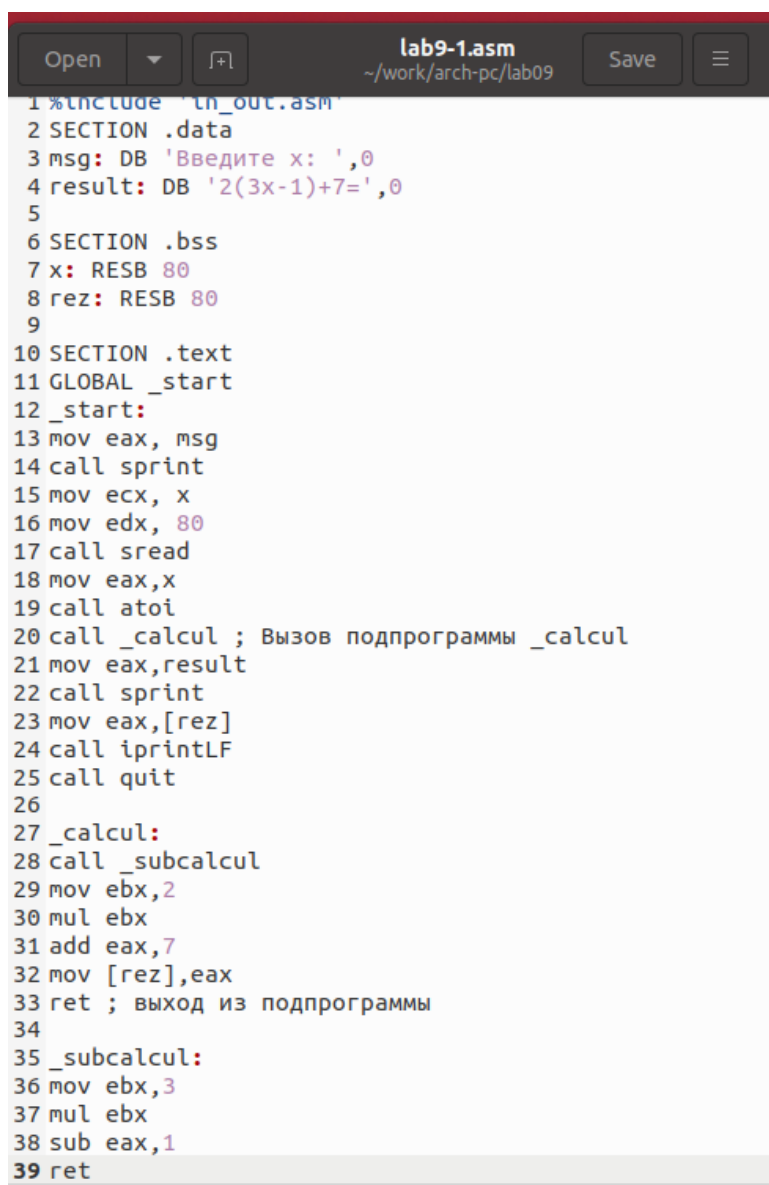
Рис. 2.1: Программа lab9-1.asm



```
tssokolova@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
tssokolova@Ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.2: Запуск программы lab9-1.asm

После этого я внесла изменения в текст программы, добавив подпрограмму `subcalcul` внутрь подпрограммы `calcul`. Это позволило 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)



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

Рис. 2.3: Программа lab9-1.asm



```

tssokolova@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
tssokolova@Ubuntu:~/work/arch-pc/lab09$
tssokolova@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2(3x-1)+7=17
tssokolova@Ubuntu:~/work/arch-pc/lab09$ █

```

Рис. 2.4: Запуск программы lab9-1.asm

## 2.2 Отладка программ с помощью GDB

Я создала файл с названием lab9-2.asm, в котором содержится программа из Листинга 9.2. Эта программа отвечает за вывод сообщения “Hello world!” на экран.(рис. 2.5)

```
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. 2.5: Программа lab9-2.asm

После этого я скомпилировала файл и получила исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, я использовала ключ “-g”. Затем я загрузила полученный исполняемый файл в отладчик GDB и проверила его работу, запустив программу с помощью команды “run” или “r”. (рис. 2.6)

```

tssokolova@Ubuntu:~/work/arch-pc/lab09$
tssokolova@Ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
tssokolova@Ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
tssokolova@Ubuntu:~/work/arch-pc/lab09$
tssokolova@Ubuntu:~/work/arch-pc/lab09$ gdb lab9-2

GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 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/tssokolova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 2647) exited normally]
(gdb) █

```

Рис. 2.6: Запуск программы lab9-2.asm в отладчике

Для более детального анализа программы я установила точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустила ее. Затем я просмотрела дизассемблированный код программы.(рис. 2.7) (рис. 2.8)

```
tssokolova@Ubuntu: ~/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:
<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/tssokolova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 2647) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) run
Starting program: /home/tssokolova/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

```
tssokolova@Ubuntu: ~/work/arch-pc/lab09
Starting program: /home/tssokolova/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

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

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

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]

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

native process 2661 In: _start L?? PC: 0x8049000
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 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

В отладчике GDB у меня была возможность просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Я выполнила 5 инструкций с помощью команды 'stepi' (сокращенно 'si') и отслеживала изменение значений регистров. (рис. 2.10) (рис. 2.11)

```
tsokolova
tssokolova@Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 < start+5>
eflags   0x202    [ IF ]

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

native process 2661 In: _start L?? PC: 0x8049005
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23      35
--Type <RET> for more, q to quit, c to continue without paging--
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0       0
gs       0x0       0
(gdb) si
0x08049005 in _start ()
(gdb) 
```

Рис. 2.10: Изменение регистров

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

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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

native process 2661 In: _start L?? PC: 0x8049016
gs      0x0      0
(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

Я также просмотрела значение переменной msg1 по имени и получила нужные данные. Чтобы изменить значение регистра или ячейки памяти, я использовала команду 'set', указав имя регистра или адрес в качестве аргумента. Я изменила первый символ переменной msg1. (рис. 2.12) (рис. 2.13)



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

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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

native process 2682 In: _start L?? PC: 0x8049016
0x08049016 in _start ()
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n"
(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"
(gdb) 
```

Рис. 2.12: Изменение значения переменной

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

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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

native process 2682 In: _start L?? PC: 0x8049016
$2 = 1000
(gdb) p/s $ecx
$3 = 134520832
(gdb) p/x $ecx
$4 = 0x804a000
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) █
```

Рис. 2.13: Вывод значения регистра

Также, с помощью команды 'set', я изменила значение регистра ebx на нужное значение.(рис. 2.14)

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

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

8+ 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

native process 2682 In: _start L?? PC: 0x8049016
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb) 
```

Рис. 2.14: Вывод значения регистра

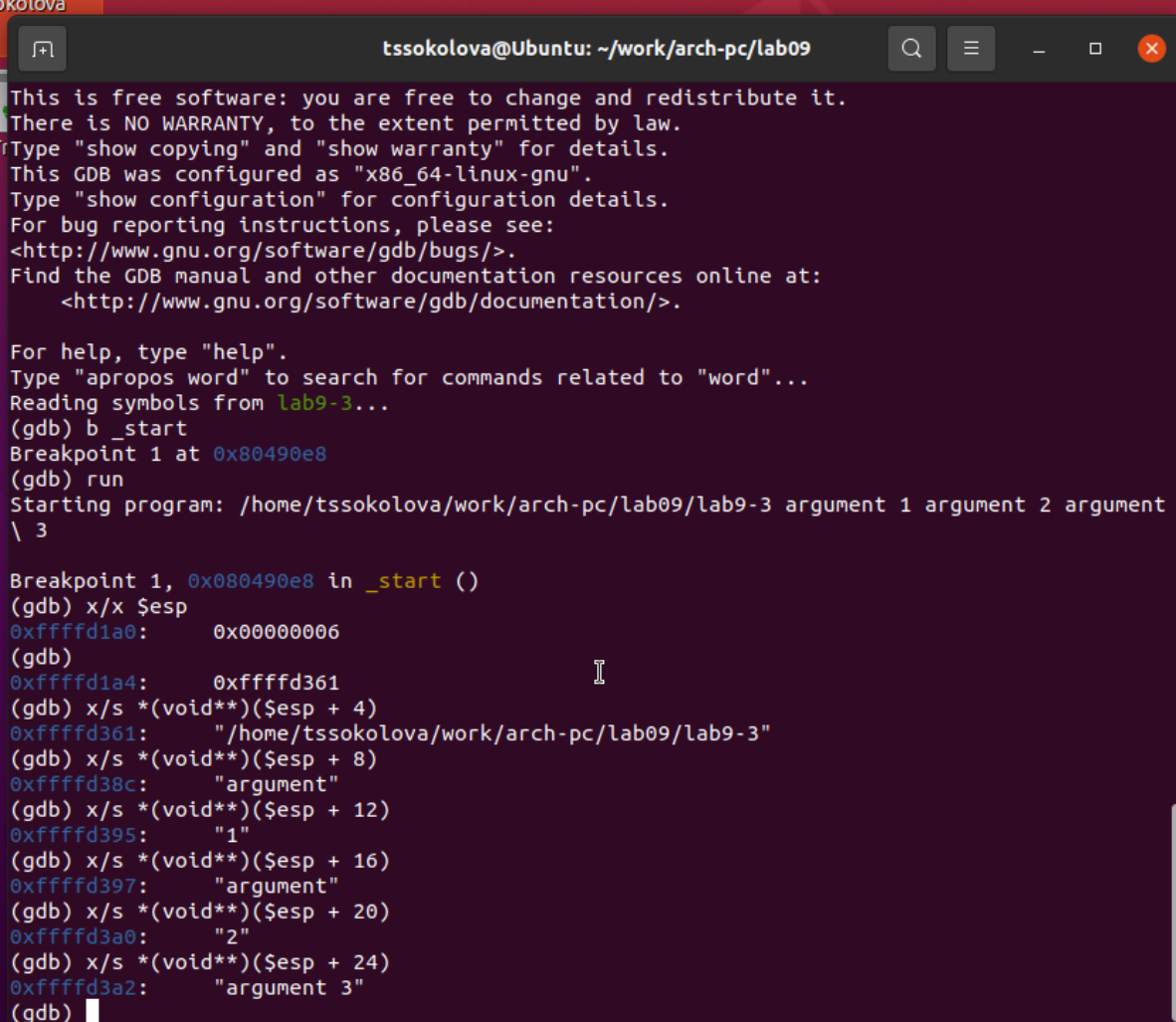
Я скопировала файл lab8-2.asm, который был создан во время выполнения лабораторной работы №8. Этот файл содержит программу для вывода аргументов командной строки. Затем я создала исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в отладчик GDB, я использовала ключ `-args` и загрузила исполняемый файл в отладчик с указанными аргументами. Я установила точку останова перед первой инструкцией программы и запустила ее.

Адрес вершины стека, где хранится количество аргументов командной

строки (включая имя программы), находится в регистре esp. По этому адресу находится число, указывающее количество аргументов. В данном случае я увидела, что количество аргументов равно 5, включая имя программы lab9-3 и сами аргументы: аргумент1, аргумент2 и 'аргумент 3'.

Я также 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. 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]). (рис. 2.15)



```
tssokolova@Ubuntu: ~/work/arch-pc/lab09
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/tssokolova/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument
\ 3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd1a0: 0x00000006
(gdb)
0xffffd1a4: 0xffffd361
(gdb) x/s *(void**)(esp + 4)
0xffffd361: "/home/tssokolova/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd38c: "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd395: "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd397: "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd3a0: "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd3a2: "argument 3"
(gdb)
```

Рис. 2.15: Вывод значения регистра

## 2.3 Задание для самостоятельной работы

Я решила преобразовать программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), добавив вычисление значения функции  $f(x)$  в виде подпрограммы. (рис. 2.16) (рис. 2.17)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 4x + 3',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 proc
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 proc:
34 mov ebx,4
35 mul ebx
36 add eax,3
37 ret
```

Рис. 2.16: Программа prog-1.asm

```

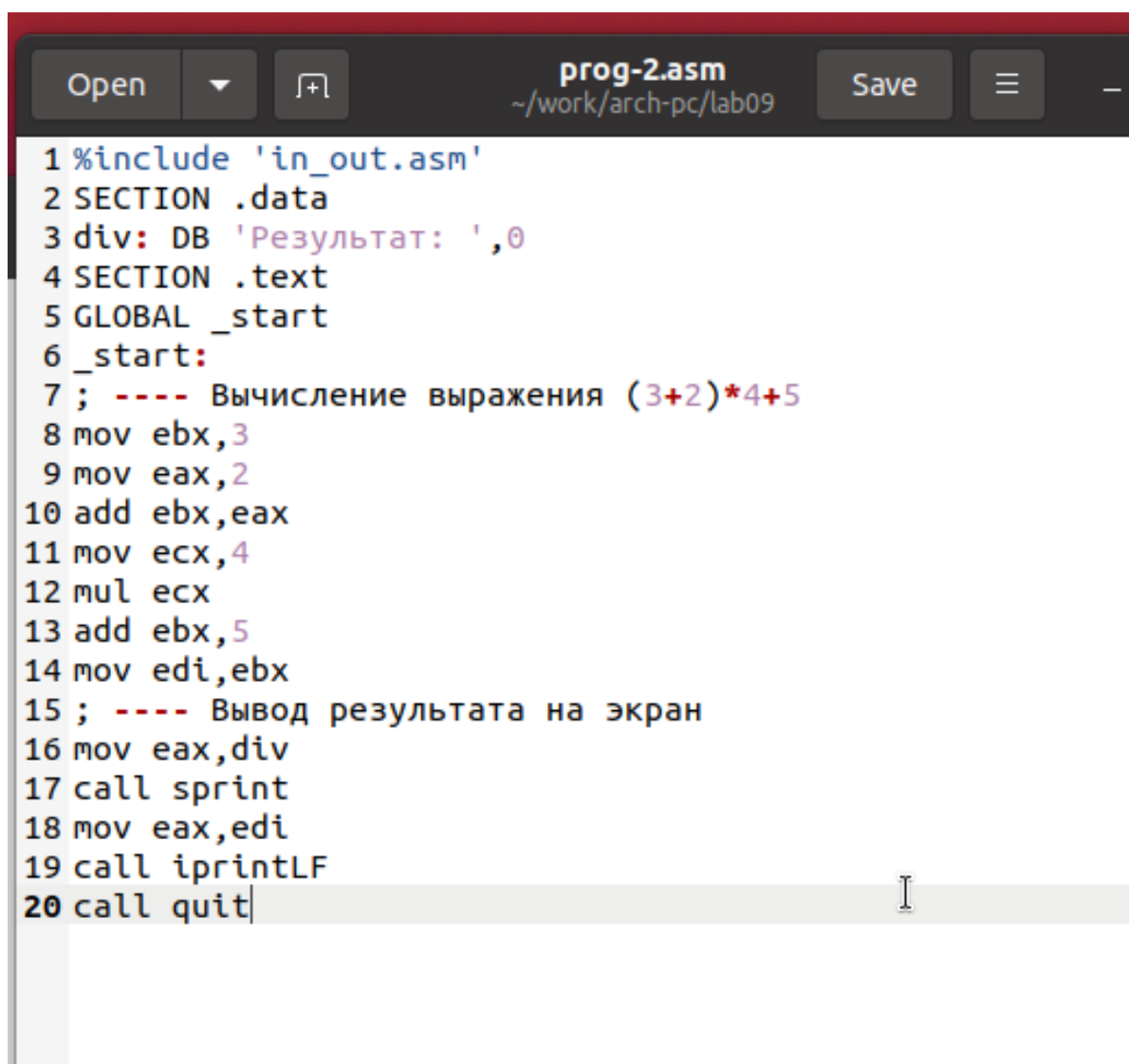
tssokolova@ubuntu:~/work/arch-pc/lab09$
tssokolova@ubuntu:~/work/arch-pc/lab09$
tssokolova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf prog-1.asm
tssokolova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 prog-1.o -o prog-1
tssokolova@ubuntu:~/work/arch-pc/lab09$ ./prog-1 3
f(x)= 4x + 3
Результат: 15
tssokolova@ubuntu:~/work/arch-pc/lab09$ ./prog-1 1 2 5 2
f(x)= 4x + 3
Результат: 52
tssokolova@ubuntu:~/work/arch-pc/lab09$ █

```

Рис. 2.17: Запуск программы prog-1.asm

В листинге представлена программа для вычисления выражения  $(3 + 2) * 4 + 5$ . Однако, при запуске программы, я обнаружила, что она дает неверный результат. Чтобы разобраться в причинах, я провела анализ изменений значений регистров с помощью отладчика GDB.

В результате анализа, я обнаружила, что порядок аргументов у инструкции `add` был перепутан. Кроме того, я заметила, что по окончании работы программы, значение `ebx` было отправлено в `edi` вместо `eax`. (рис. 2.18)



The image shows a screenshot of an assembly code editor window. The title bar indicates the file is 'prog-2.asm' located at '~/work/arch-pc/lab09'. The editor contains 20 lines of assembly code. Line 7 contains a comment in Russian: 'Вычисление выражения (3+2)\*4+5'. Line 15 contains another comment: 'Вывод результата на экран'. The code uses various x86 assembly instructions like 'mov', 'add', 'mul', and 'call'. Line 20 ends with 'call quit'. A vertical cursor is positioned at the end of line 20.

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

Рис. 2.18: Код с ошибкой

```
kolova
tssokolova@Ubuntu: ~/work/arch-pc/lab09

Register group: general
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0xa           10
esp      0xffffd1d8    0xffffd1d8
ebp      0x0           0x0
esi      0x0           0
edi      0xa           10
eip      0x8049010     0x8049010 <sprint+1>
eflags   0x206        [ PF IF ]

0x804900f <sprint>    push    edx
>0x8049010 <sprint+1> push    ecx
0x8049011 <sprint+2>  push    ebx
0x8049012 <sprint+3>  push    eax
0x8049013 <sprint+4>  call    0x8049000 <slen>
0x8049018 <sprint+9>  mov     edx,eax
0x804901a <sprint+11> pop     eax
0x804901b <sprint+12> mov     ecx,eax
0x804901d <sprint+14> mov     ebx,0x1
0x8049022 <sprint+19> mov     eax,0x4
0x8049027 <sprint+24> int     0x80

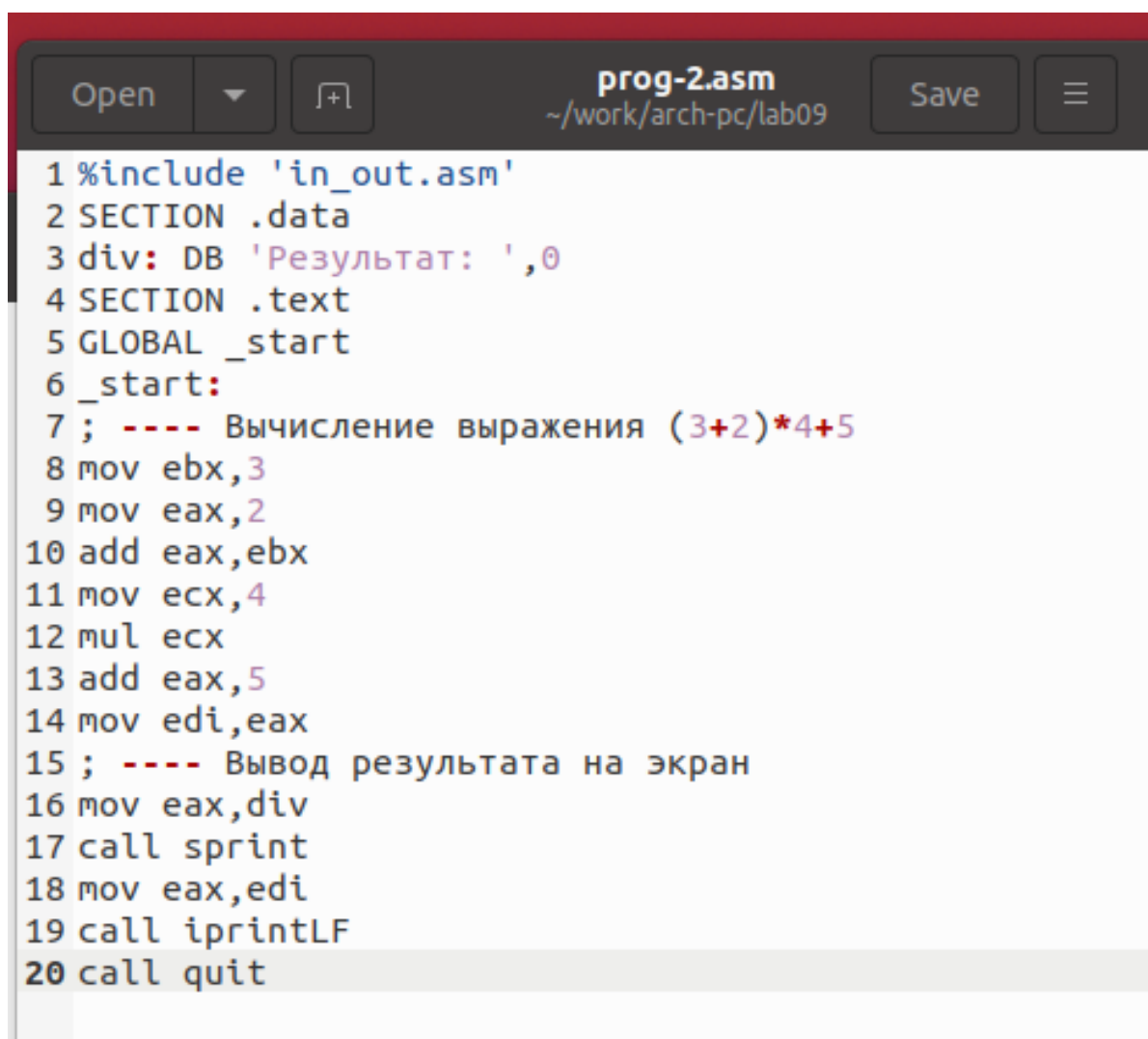
native process 2738 In: sprint L?? PC: 0x8049010
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) si
0x08049105 in _start ()
(gdb) si
0x0804900f in sprint ()
(gdb) si
0x08049010 in sprint ()
(gdb) s
```

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)





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

Рис. 2.20: Код исправлен

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

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1e0 0xffffd1e0
ebp      0x0       0
esi      0x0       0
edi      0x19      25
eip      0x8049100 0x8049100 <_start+24>
eflags   0x202     [ IF ]

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,eax04a000
0x8049100 <_start+24>     mov     eax,0x804a000rint>
0x8049105 <_start+29>     call    0x804900f <sprint>
0x804910a <_start+34>     mov     eax,edi86 <iprintf>
0x804910c <_start+36>     call    0x8049086 <iprintf>

native process 2756 In: _start L?? PC: 0x8049100
0x08049No process In: L?? PC: ??
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 2756) exited normally]
(gdb) █
```

Рис. 2.21: Проверка работы

## **3 Выводы**

Освоили работу с подпрограммами и отладчиком.