

# **ОТЧЕТ по лабораторной работе №9**

**дисциплина: Архитектура компьютера**

**Студент: Идрисов Д.А.**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Реализация подпрограмм в NASM . . . . .	8
4.2	Отладка программ с помощью GDB . . . . .	12
4.3	Задание для самостоятельной работы . . . . .	23
<b>5</b>	<b>Выводы</b>	<b>30</b>

## Список иллюстраций

4.1	Изменение кода . . . . .	9
4.2	Запуск программы . . . . .	10
4.3	Изменение кода . . . . .	11
4.4	Запуск программы . . . . .	12
4.5	Изменение кода . . . . .	13
4.6	Запуск программы в отладчике . . . . .	14
4.7	Дизассимилированный код . . . . .	15
4.8	Дизассимилированный код в режиме интел . . . . .	15
4.9	Точка остановки . . . . .	16
4.10	Изменение регистров . . . . .	17
4.11	Изменение регистров . . . . .	18
4.12	Изменение значения переменной . . . . .	19
4.13	Вывод значения регистра . . . . .	20
4.14	Вывод значения регистра . . . . .	21
4.15	Изменение кода . . . . .	22
4.16	Вывод значения регистра . . . . .	23
4.17	Изменение кода . . . . .	24
4.18	Запуск программы . . . . .	25
4.19	Код с ошибкой . . . . .	26
4.20	Отладка . . . . .	27
4.21	Код исправлен . . . . .	28
4.22	Проверка работы . . . . .	29

## **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 Задание

1. Изучение подпрограмм в ассемблере
2. Изучение отладчика GDB
3. Изучение примеров программ и процесса отладки
4. Выполнение заданий для самостоятельной работы

### 3 Теоретическое введение

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом

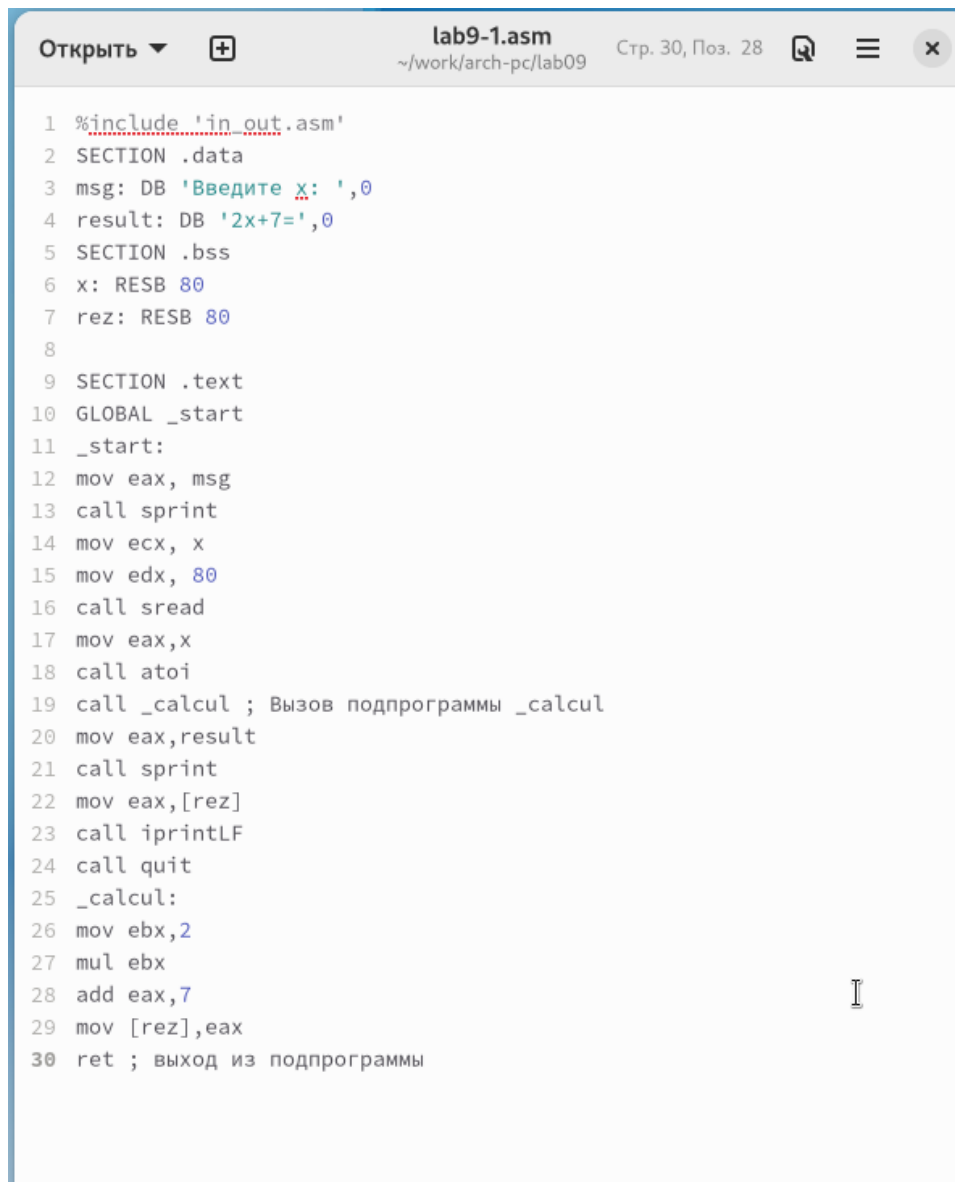
## 4 Выполнение лабораторной работы

### 4.1 Реализация подпрограмм в NASM

Я создал каталог, предназначенный для выполнения лабораторной работы №9, и перешел в него.

В рамках 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.





```
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. 4.1: Изменение кода

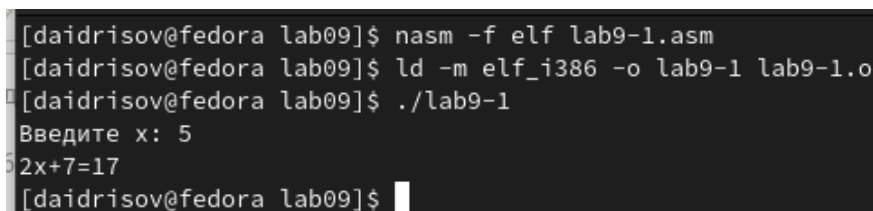
Первые строки программы отвечают за вывод сообщения на экран с использованием функции `sprint`, чтение данных, введенных с клавиатуры с помощью функции `sread`, и преобразование введенных данных из символьного в числовой формат с помощью функции `atoi`.

После инструкции `call _calcul`, которая передает управление подпрограмме `_calcul`, выполнение программы переходит к инструкциям, содержащимся внутри

подпрограммы.

Инструкция `ret` является последней в подпрограмме и ее выполнение приводит к возврату в основную программу к инструкции, следующей за инструкцией `call`, которая вызвала данную подпрограмму.

Внесены изменения в текст программы, добавлена подпрограмма `subcalcul` внутри подпрограммы `calcul` для вычисления выражения  $f(g(x))$ , где значение  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ .



```
[daidrisov@fedora lab09]$ nasm -f elf lab9-1.asm
[daidrisov@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[daidrisov@fedora lab09]$ ./lab9-1
Введите x: 5
5
2x+7=17
[daidrisov@fedora lab09]$
```

Рис. 4.2: Запуск программы

Изменил текст программы, добавив подпрограмму `subcalcul` в подпрограмму `calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ .

```
Открыть ▾ + lab9-1.asm Стр. 39, Поз. 4
~/work/arch-pc/lab09

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
```

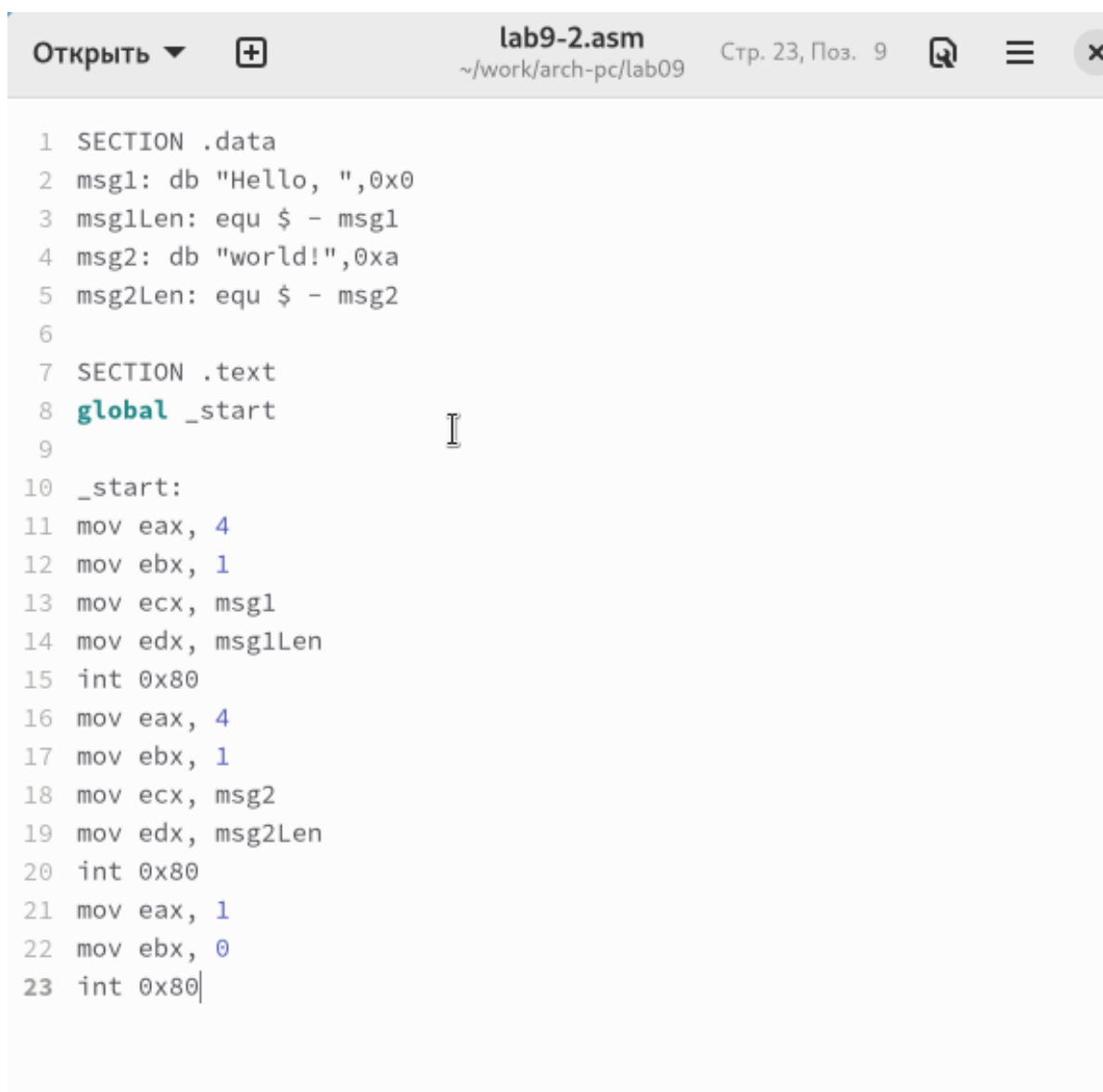
Рис. 4.3: Изменение кода

```
[daidrisov@fedora lab09]$  
[daidrisov@fedora lab09]$ nasm -f elf lab9-1.asm  
[daidrisov@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o  
[daidrisov@fedora lab09]$ ./lab9-1  
Введите x: 5  
2(3x-1)+7=35  
[daidrisov@fedora lab09]$
```

Рис. 4.4: Запуск программы

## 4.2 Отладка программ с помощью GDB

Я создал файл с именем lab9-2.asm, в котором содержится текст программы из Листинга 9.2, реализующей функцию печати сообщения “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. 4.5: Изменение кода

После компиляции получил исполняемый файл. Чтобы использовать отладчик GDB, я добавил отладочную информацию к исполняемому файлу, указав ключ “-g” при компиляции.

Затем я загрузил исполняемый файл в отладчик GDB и проверил его работу, запустив программу с помощью команды “run” (или “r” в сокращенной форме).

```

[daidrisov@fedora lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[daidrisov@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[daidrisov@fedora lab09]$ gdb lab9-2
GNU gdb (GDB) Fedora 12.1-2.fc36
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

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

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 6528) exited normally]
(gdb)

```

Рис. 4.6: Запуск программы 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.

```

[Interrupt 1 (process 6628) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) r
Starting program: /home/daidrisov/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ecx
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
      0x08049016 <+22>:     mov     $0x4,%eax
      0x0804901b <+27>:     mov     $0x1,%ebx
      0x08049020 <+32>:     mov     $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. 4.7: Дизассимилированный код

```

(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)

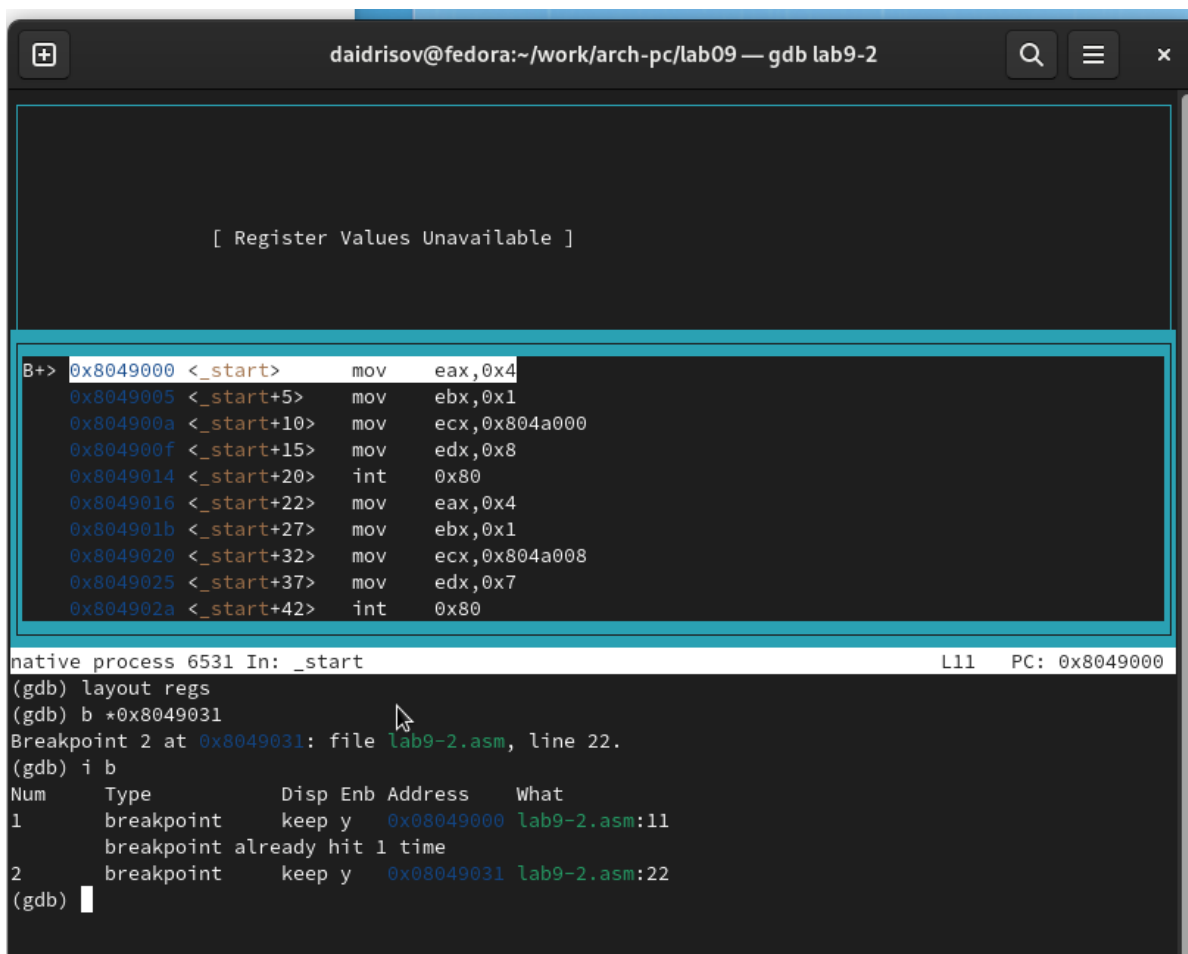
```

Рис. 4.8: Дизассимилированный код в режиме интел

Чтобы установить точку остановки, я использовал команду “break” (или “b”

в сокращенной форме). Типичным аргументом для этой команды может быть номер строки программы, имя метки или адрес. Чтобы избежать путаницы с номерами, перед адресом ставится знак “\*“.

На предыдущих шагах я уже установил точку остановки по имени метки “\_start” и проверил это с помощью команды “info breakpoints” (или “i b” в сокращенной форме). Затем я установил еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0”.



The screenshot shows a GDB terminal window with the title bar "daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2". The main display area shows assembly code for a program starting at address 0x8049000. The code is as follows:

Address	Disassembly
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

Below the assembly code, the GDB prompt shows the following commands and output:

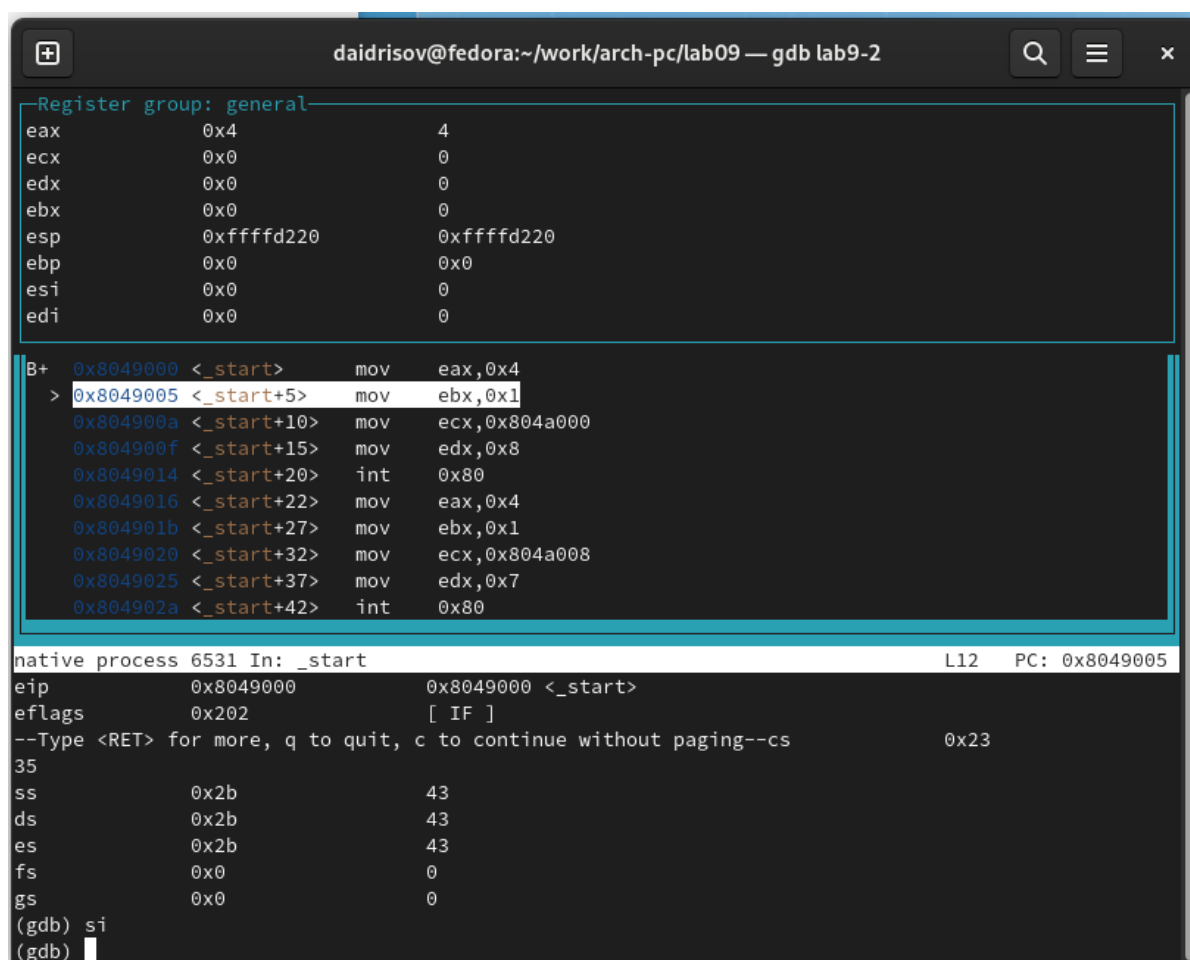
```
native process 6531 In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 22.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08049000 lab9-2.asm:11
       breakpoint already hit 1 time
2      breakpoint     keep y   0x08049031 lab9-2.asm:22
(gdb)
```

Рис. 4.9: Точка остановки

Я использовал отладчик, который позволяет просматривать содержимое ячеек памяти и регистров, а также вносить в них изменения при необходимости. Я выполнил 5 инструкций с помощью команды stepi (или si) и следил за изменениями



значений регистров.



The screenshot shows a GDB terminal window with the title bar "daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into two main sections. The top section, titled "Register group: general", displays the current values of the general-purpose registers: 

Register	Value	Comment
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd220	0xffffd220
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The bottom section displays assembly code with addresses and instructions: 

```
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
```

. Below the assembly code, the status of the native process is shown: "native process 6531 In: \_start" with "L12" and "PC: 0x8049005". Further down, the instruction pointer (eip) is "0x8049000" pointing to "<\_start>", and the eflags register is "0x202" with "[ IF ]". A prompt "--Type <RET> for more, q to quit, c to continue without paging--cs" is followed by "0x23". At the bottom, the stack segment registers (ss, ds, es, fs, gs) are listed with values "0x2b" and "43", and the instruction pointer (eip) is "0x2b". The GDB prompt "(gdb) si" is visible at the bottom left.

Рис. 4.10: Изменение регистров

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd220 0xffffd220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 6531 In: _start L16 PC: 0x8049016
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Рис. 4.11: Изменение регистров

Далее я просмотрел значение переменной `msg1`, обратившись к ней по имени. Также я просмотрел значение переменной `msg2`, обратившись к ней по адресу.

Для изменения значения регистра или ячейки памяти я использовал команду `set`, указав в качестве аргумента имя регистра или адрес. Я изменил первый символ переменной `msg1`.

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd220 0xffffd220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 6531 In: _start L16 PC: 0x8049016
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(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. 4.12: Изменение значения переменной

Для вывода значения регистра `edx` в различных форматах (шестнадцатеричном, двоичном и символьном) я использовал соответствующие команды.

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd220 0xffffd220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 6531 In: _start L16 PC: 0x8049016
(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. 4.13: Вывод значения регистра

С помощью команды set также изменил значение регистра ebx.

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb lab9-2

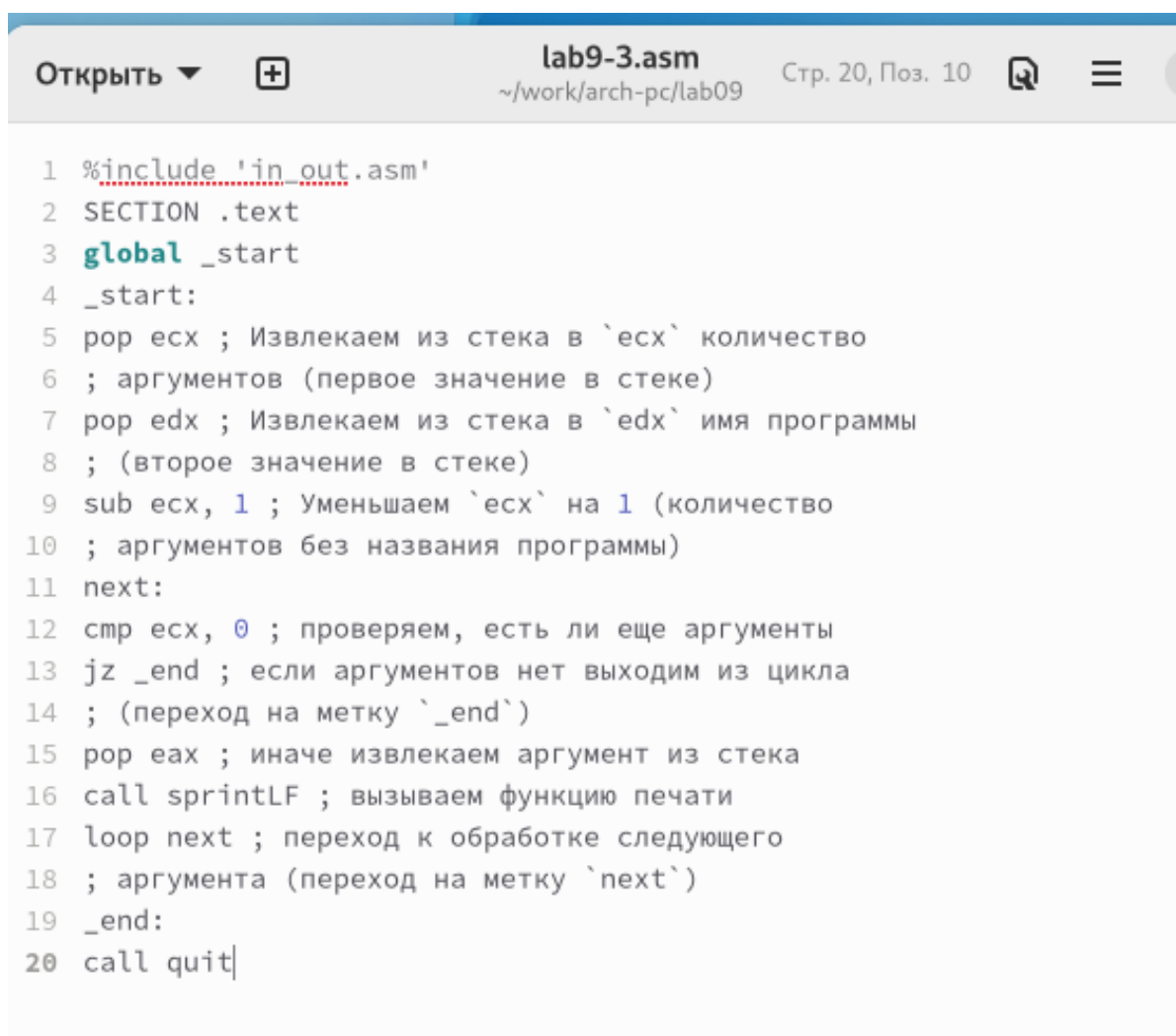
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd220 0xffffd220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 6531 In: _start L16 PC: 0x8049016
(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. 4.14: Вывод значения регистра

Я скопировал файл lab8-2.asm, созданный в ходе выполнения лабораторной работы №8, который выводит аргументы командной строки, и создал исполняемый файл. Для загрузки программы с аргументами в отладчик GDB использовал ключ `-args`, указав соответствующие аргументы. Затем установил точку остановки перед первой инструкцией в программе и запустил ее.



```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 4.15: Изменение кода

Адрес вершины стека хранится в регистре `esp`, и по этому адресу располагается число, равное количеству аргументов командной строки, включая имя программы. В данном случае число аргументов равно 5: имя программы `lab9-3` и аргументы: `аргумент1`, `аргумент2` и `‘аргумент 3’`.

Я также просмотрел остальные позиции стека. По адресу `[esp+4]` находится адрес в памяти, где хранится имя программы, по адресу `[esp+8]` хранится адрес первого аргумента, по адресу `[esp+12]` - второго и так далее.

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb --args lab9-3 argument 1 argument 2 ...
<http://www.gnu.org/software/gdb/documentation/>.

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

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

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1f0:      0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd3a3:      "/home/daidrisov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3cd:      "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd3d6:      "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd3d8:      "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd3e1:      "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd3e3:      "argument 3"
(gdb)
```

Рис. 4.16: Вывод значения регистра

Объясню, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12]). Это связано с тем, что шаг равен размеру переменной, который составляет 4 байта.

## 4.3 Задание для самостоятельной работы

Я внес изменения в программу из лабораторной работы №8, чтобы вычислить значение функции  $f(x)$  в виде подпрограммы.

```
Открыть ▾ + task-1.asm Стр. 37, Поз. 4
~/work/arch-pc/lab09
4  fx: db 'f(x)= 10(x - 1)',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 call _calc
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 _calc:
34 sub eax,1
35 mov ebx,10
36 mul ebx
37 ret
```




Рис. 4.17: Изменение кода



```
[daidrisov@fedora lab09]$  
[daidrisov@fedora lab09]$ nasm -f elf task-1.asm  
[daidrisov@fedora lab09]$ ld -m elf_i386 -o task-1 task-1.o  
[daidrisov@fedora lab09]$ ./task-1 1  
f(x)= 10(x - 1)  
Результат: 0  
[daidrisov@fedora lab09]$ ./task-1 2 3 4 5 6  
f(x)= 10(x - 1)  
Результат: 150  
[daidrisov@fedora lab09]$
```

Рис. 4.18: Запуск программы

Ниже приведен исправленный листинг программы, который вычисляет выражение  $(3 + 2) * 4 + 5$ . Однако, при запуске, программа дает неверный результат. Я решил использовать отладчик GDB для анализа изменений значений регистров и определения ошибки.

Открыть ▾  task-2.asm  
~/work/arch-pc/lab09 Стр. 20, Поз. 10  

```
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. 4.19: Код с ошибкой

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb task-2
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd220 0xffffd220
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10

B+ 0x80490e8 <_start>    mov     ebx,0x3
0x8049100 <_start+24>    mov     eax,0x804a000
0x8049105 <_start+29>    call    0x804900f <sprint>
0x804910a <_start+34>    mov     eax,edi
0x804910c <_start+36>    call    0x8049086 <iprintf>
0x8049111 <_start+41>    call    0x80490db <quit>

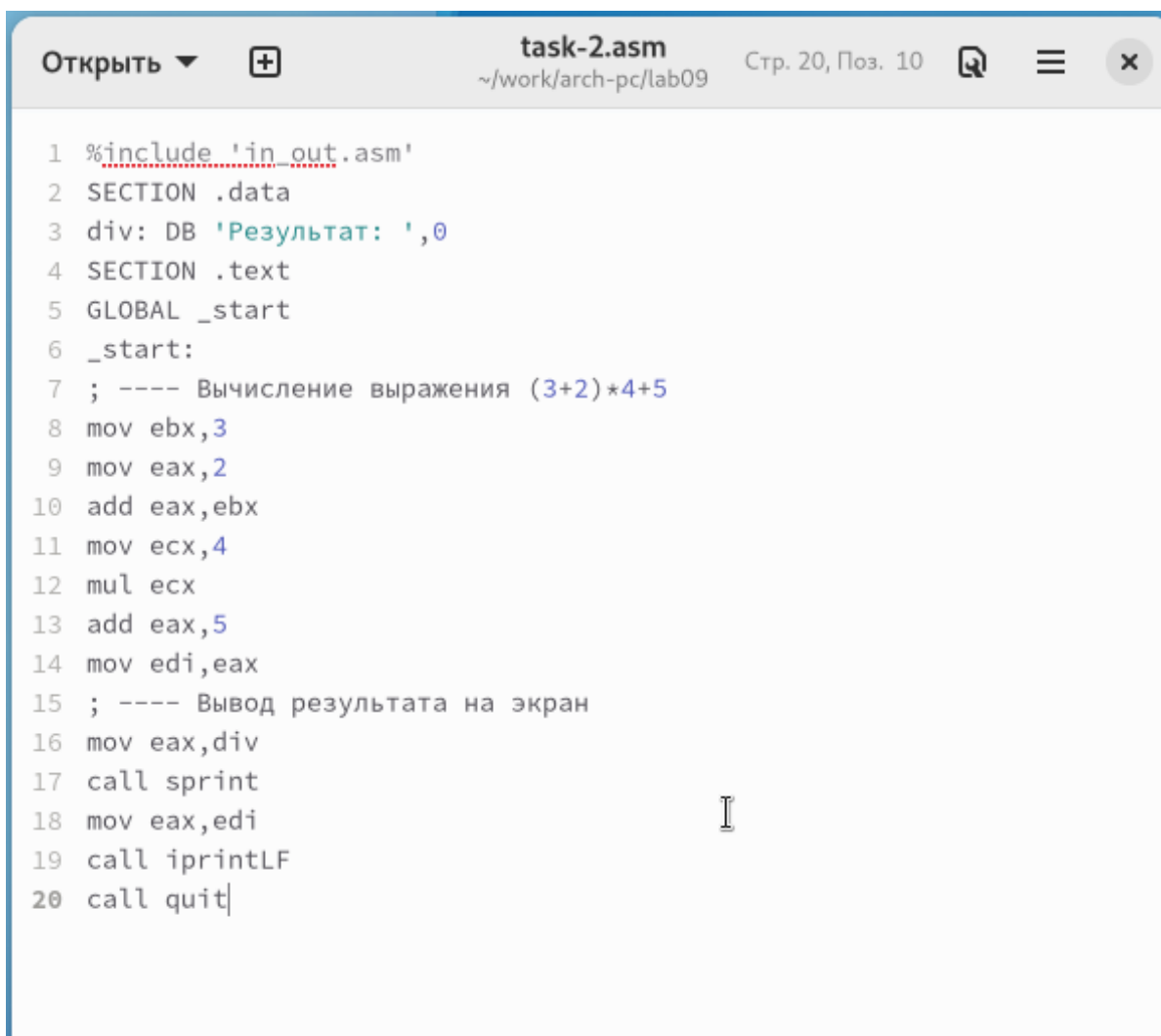
>                                04a000
                                rint>

native process 6668 In: _start L16 PC: 0x8049100
Breakpoint 1: No process In: k-2.asm:8 L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) cont
Continuing.
Результат: 10
[Inferior 1 (process 6668) exited normally]
(gdb) 
```

Рис. 4.20: Отладка

В процессе отладки я заметил, что порядок аргументов в инструкции add был перепутан, и что при завершении работы, вместо eax, значение отправлялось в edi.

Вот исправленный код программы:



```
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. 4.21: Код исправлен

```
daidrisov@fedora:~/work/arch-pc/lab09 — gdb task-2

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd220 0xffffd220
ebp      0x0       0x0
esi      0x0       0
edi      0x0       0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490fe <_start+22> mov edi,eax
0x8049100 <_start+24> add eax,eb804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mul eax,edi
0x804910c <_start+36> call 0x8049086 <iprintfLF>
> 0x8049111 <_start+41> call 0x80490db <quit>
                                04a000
                                rint>

native process 6712 In: _start L14 PC: 0x80490fe
No process In: L?? PC: ??

(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) cont
Continuing.
Результат: 25
[Inferior 1 (process 6712) exited normally]
(gdb)
```

Рис. 4.22: Проверка работы

## **5 Выводы**

Освоили работу с подпрограммами и отладчиком.