

# **Лабораторная работа №9**

**Понятие подпрограммы. Отладчик GDB.**

Дмитрий Сергеевич Хохлов

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Самостоятельное задание . . . . .	21
<b>3</b>	<b>Выводы</b>	<b>27</b>
	<b>Список литературы</b>	<b>28</b>

## Список иллюстраций

2.1	Программа в файле lab9-1.asm . . . . .	7
2.2	Запуск программы lab9-1.asm . . . . .	7
2.3	Программа в файле lab9-1.asm . . . . .	8
2.4	Запуск программы lab9-1.asm . . . . .	9
2.5	Программа в файле 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 в файле lab9-3.asm . . . . .	21
2.17	Запуск программы lab9-3.asm . . . . .	22
2.18	Код с ошибкой . . . . .	23
2.19	Отладка . . . . .	24
2.20	Код исправлен . . . . .	25
2.21	Проверка работы . . . . .	26

## Список таблиц

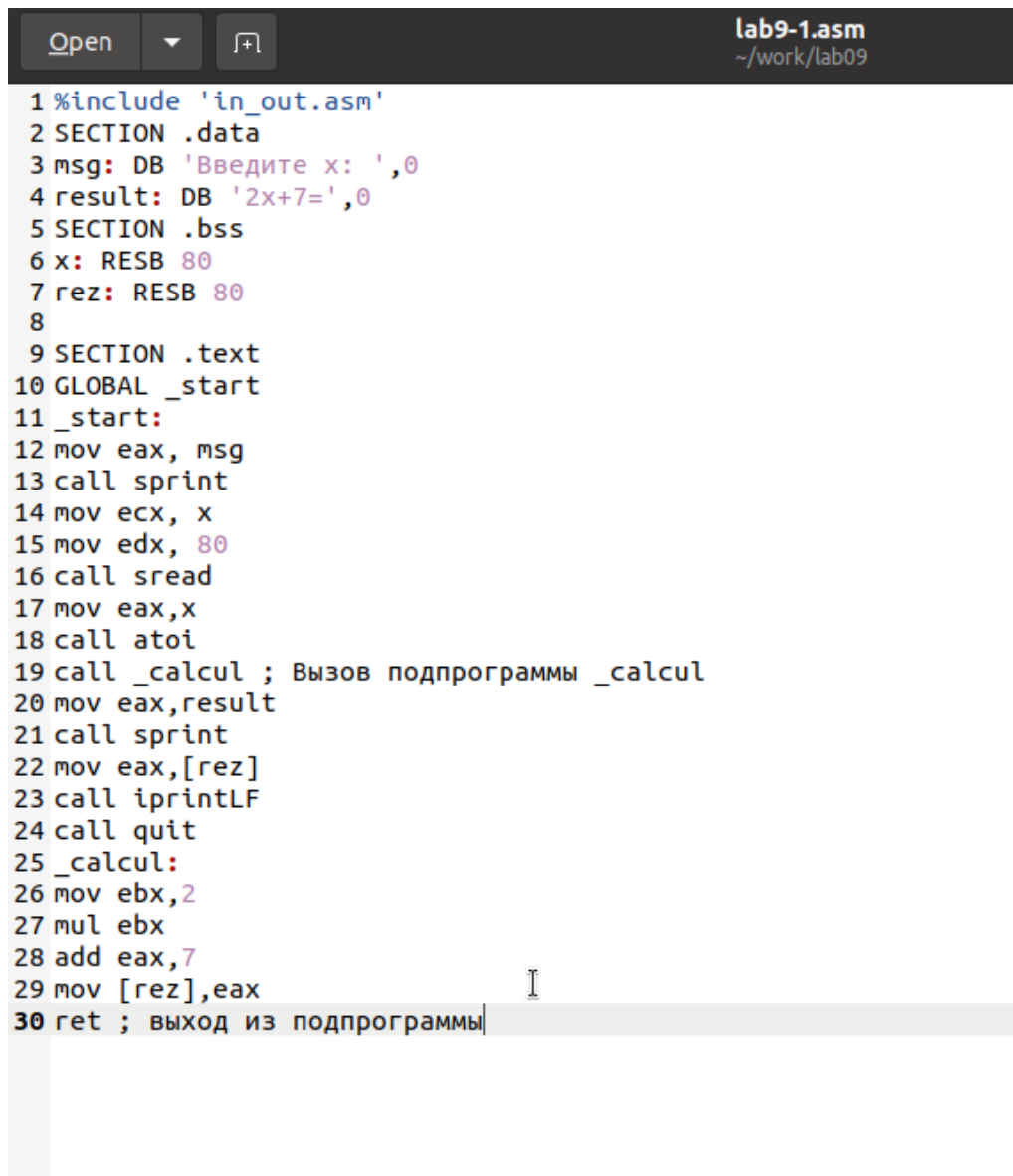
# 1 Цель работы

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

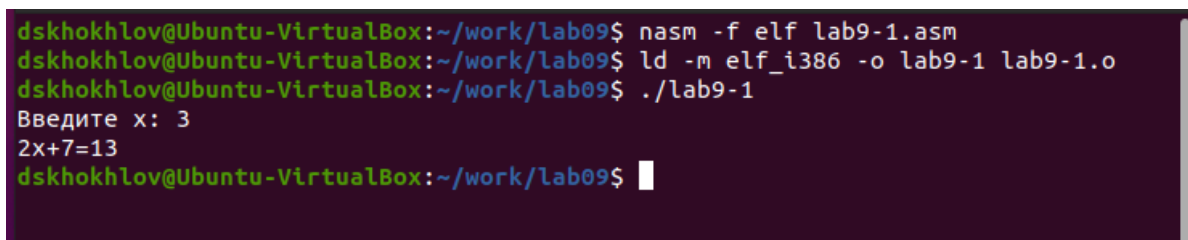
Я организовал рабочую директорию для лабораторного задания № 9 и переместился в нее. После этого произвел создание файла lab9-1.asm.

Давайте рассмотрим пример программы, которая вычисляет арифметическое выражение  $f(x) = 2x + 7$  с использованием вспомогательной функции `calcul`. В этом случае  $x$  получаем из пользовательского ввода, а расчет выражения осуществляется внутри вспомогательной функции. (см. рис. [2.1]) (см. 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
```

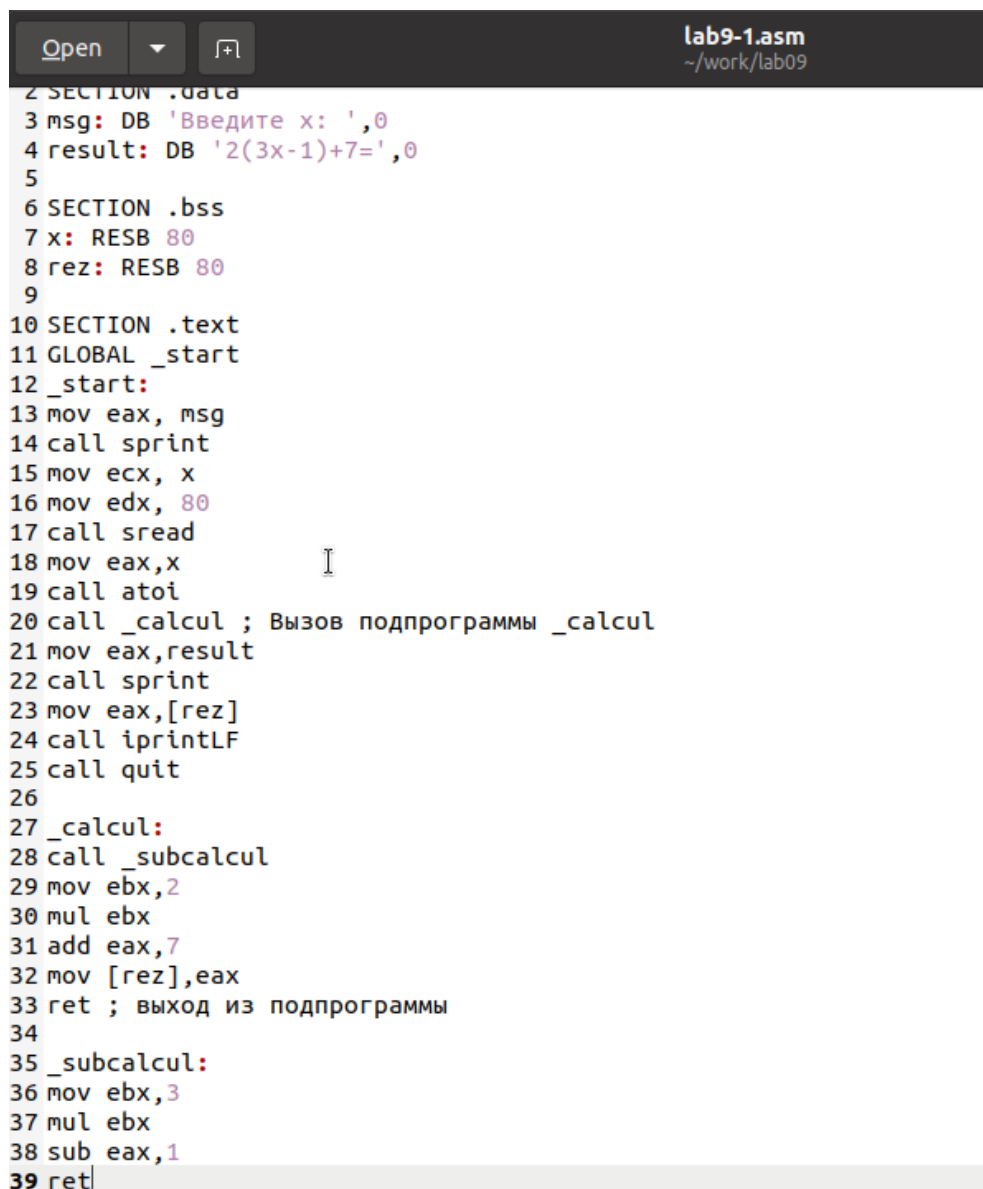
Рис. 2.1: Программа в файле lab9-1.asm



```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ nasm -f elf lab9-1.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ./lab9-1
Введите x: 3
2x+7=13
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$
```

Рис. 2.2: Запуск программы lab9-1.asm

Произвел модификацию кода программы, включив дополнительную подпрограмму `subcalcul` в уже существующую подпрограмму `calcul` для реализации расчета составного выражения  $f(g(x))$ , где  $x$  также получаем через ввод пользователя и  $f(x) = 2x + 7, g(x) = 3x - 1$ . (см. рис. [2.3]) (см. рис. [2.4])



```
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
```

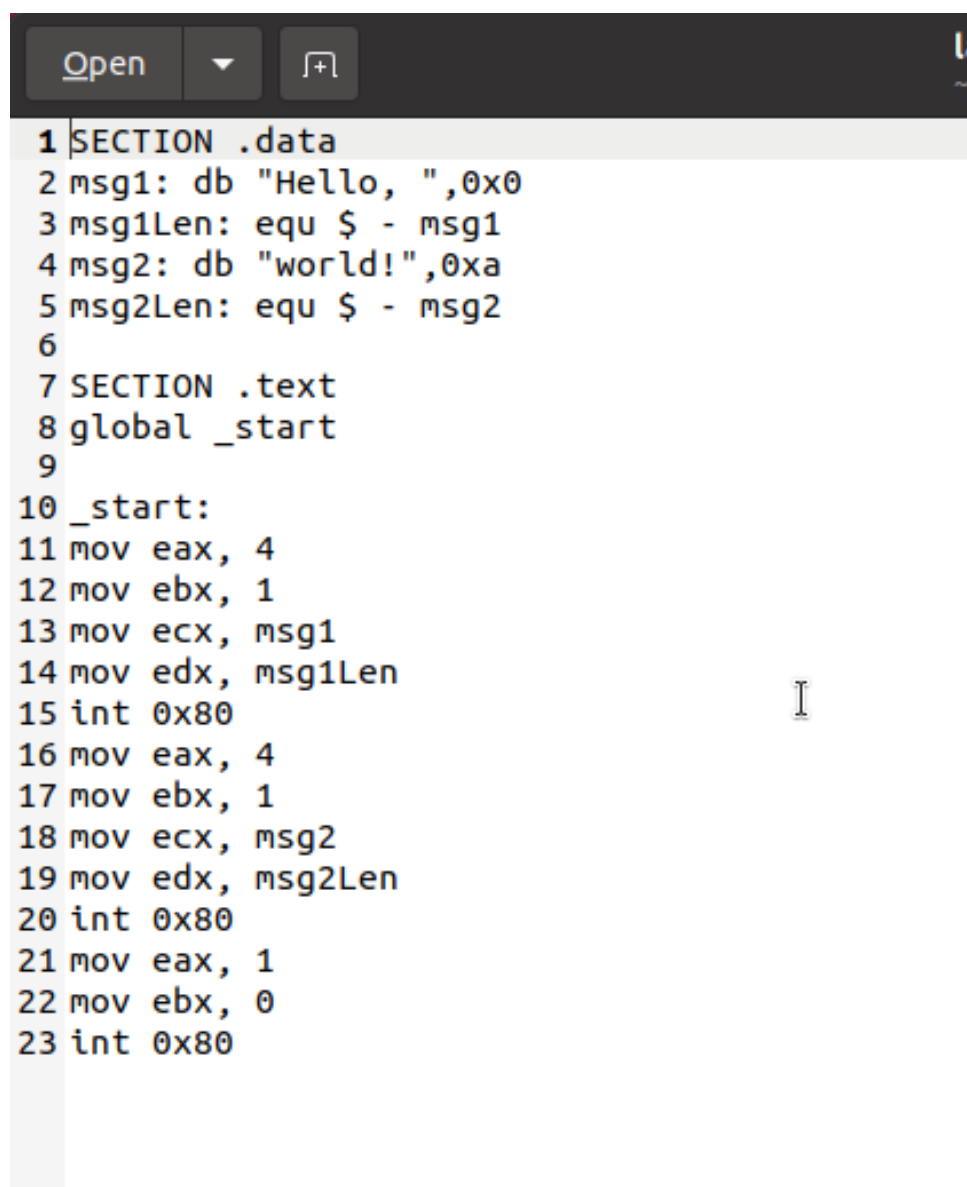
Рис. 2.3: Программа в файле lab9-1.asm



```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$  
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ nasm -f elf lab9-1.asm  
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o  
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ./lab9-1  
Введите x: 3  
2(3x-1)+7=23  
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$
```

Рис. 2.4: Запуск программы lab9-1.asm

Сформировал файл lab9-2.asm, содержащий исходный код программы из Приложения 9.2. (Эта программа осуществляет вывод на экран фразы Hello world!). (см. 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 в файле lab9-2.asm

Скомпилировал исполняемый файл, добавив отладочную информацию с использованием ключа ‘-g’ для последующей работы в отладчике GDB.

Загрузил полученный исполняемый файл в отладчик GDB и осуществил проверку функционирования программы, инициировав ее выполнение командой ‘run’ (или ‘r’ в сокращенной форме). (см. рис. [2.6])

```

dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$
dskhokhlov@Ubuntu-VirtualBox:~/work/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/dskhokhlov/work/lab09/lab9-2
Hello, world!
[Inferior 1 (process 4654) exited normally]
(gdb)

```

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

Для детального изучения программы установил точку останова на метке 'start', которая является начальной точкой выполнения любой программы на ассемблере, и выполнил запуск. После этого изучил дизассемблированный код программы. (см. рис. [2.7]) (см. рис. [2.8])

```
dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09

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/dskhokhlov/work/lab09/lab9-2
Hello, world!
[Inferior 1 (process 4654) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) run
Starting program: /home/dskhokhlov/work/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>:     I 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

```
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 установленные точки остановки, в частности для метки '\_start', я использовал команду 'info breakpoints' или в сокращении 'i b'. После этого задал новую точку остановки, нацелив ее на адрес, где располагается инструкция перед последней, а именно 'mov ebx, 0x0'. (см. рисунок [2.9])

The screenshot shows a GDB terminal window with the title bar 'dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09'. The window is divided into two main sections. The top section, titled 'Register group: general', displays the current values of various CPU registers: 

Register	Value	Comment
eax	0x0	
ecx	0x0	
edx	0x0	
ebx	0x0	
esp	0xfffffd1e0	0xfffffd1e0
ebp	0x0	0x0
esi	0x0	
edi	0x0	
eip	0x8049000	0x8049000 <_start>
eflags	0x202	[ IF ]

. The bottom section displays assembly code starting from address 0x8049000, with instructions like 'mov eax, 0x4', 'mov ebx, 0x1', 'mov ecx, 0x804a000', 'mov edx, 0x8', 'int 0x80', etc. The bottom status bar shows 'native process 4664 In: \_start', 'L??', and 'PC: 0x8049000'. The GDB prompt '(gdb)' is visible at the bottom left.

Рис. 2.9: Точка остановки

В среде отладки GDB предусмотрена возможность просмотра и редактирования содержимого памяти и регистров. Я выполнил пять шагов инструкций с использованием команды 'stepi', сокращенно 'si', и наблюдал за изменениями в регистрах. (см. рисунок [2.10]) (см. рисунок [2.11])

```
dskhokhlov@Ubuntu-VirtualBox: ~/work/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

native process 4664 In: start L?? PC: 0x8049005
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
--Type <RET> for more, q to quit, c to continue without paging--siss 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

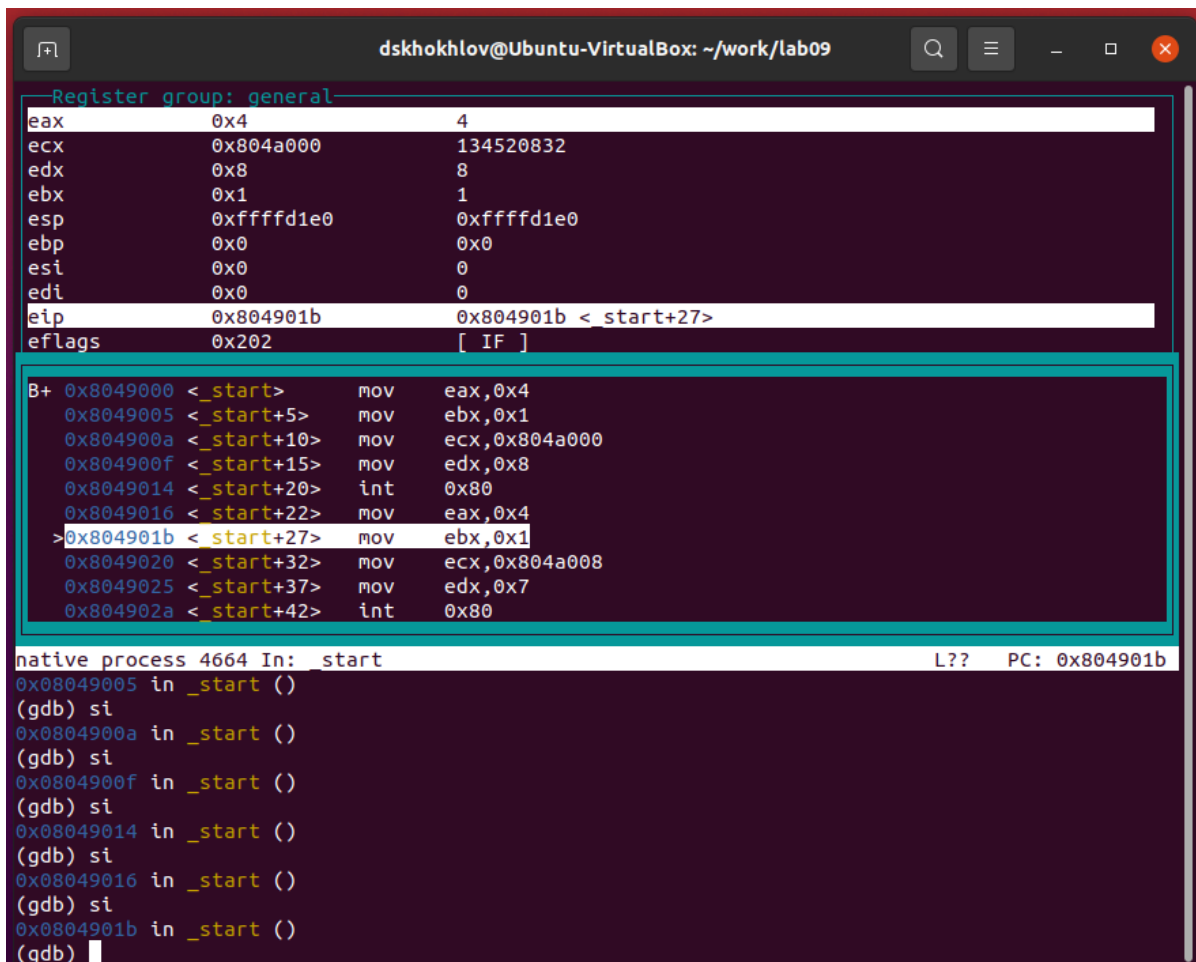


Рис. 2.11: Изменение регистров

Для модификации значений в регистрах или ячейках памяти применял команду 'set', указывая в аргументах название регистра или адрес. Таким образом, я изменил первый символ в переменной msg1. (см. рисунок [2.12])



```
dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
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

native process 4664 In: _start L?? PC: 0x804901b
0x804901b 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

Используя ту же команду 'set', я внес изменения в первый символ переменной msg1. (см. рисунок [2.13])

```
dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09

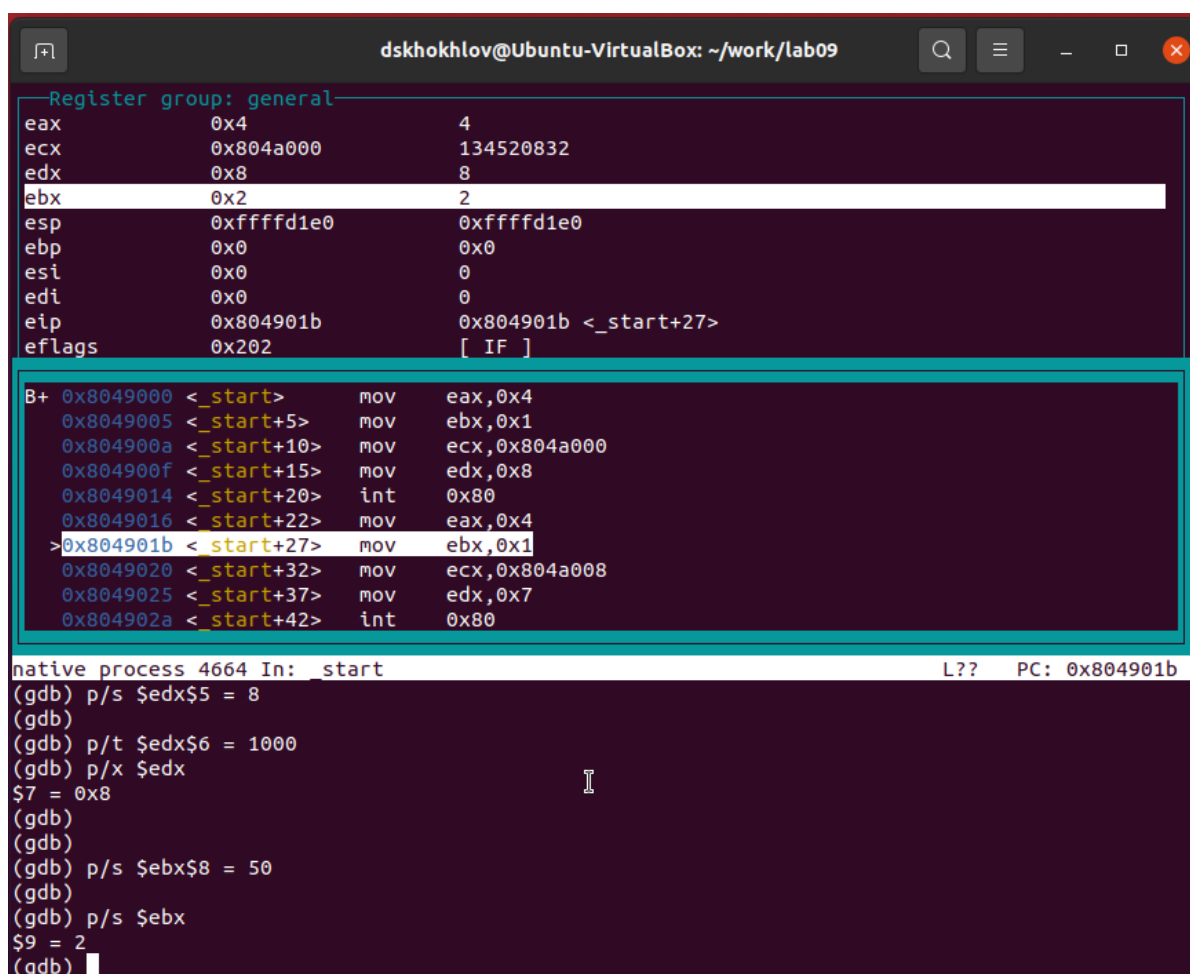
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b < start+27>
eflags   0x202    [ IF ]

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

native process 4664 In: _start L?? PC: 0x804901b
(gdb) p/t $eax$2 = 100
(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

Чтобы установить новое значение для регистра `ebx`, воспользовался командой `'set'`. (см. рисунок [2.14])



```
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
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

native process 4664 In: _start L?? PC: 0x804901b
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb) p/s $ebx
$9 = 2
(gdb)
```

Рис. 2.14: Вывод значения регистра

Скопировал исходный файл lab8-2.asm, который был создан в ходе лабораторной работы №8 и содержит код для вывода параметров командной строки. Из этого файла сформировал исполняемый файл.

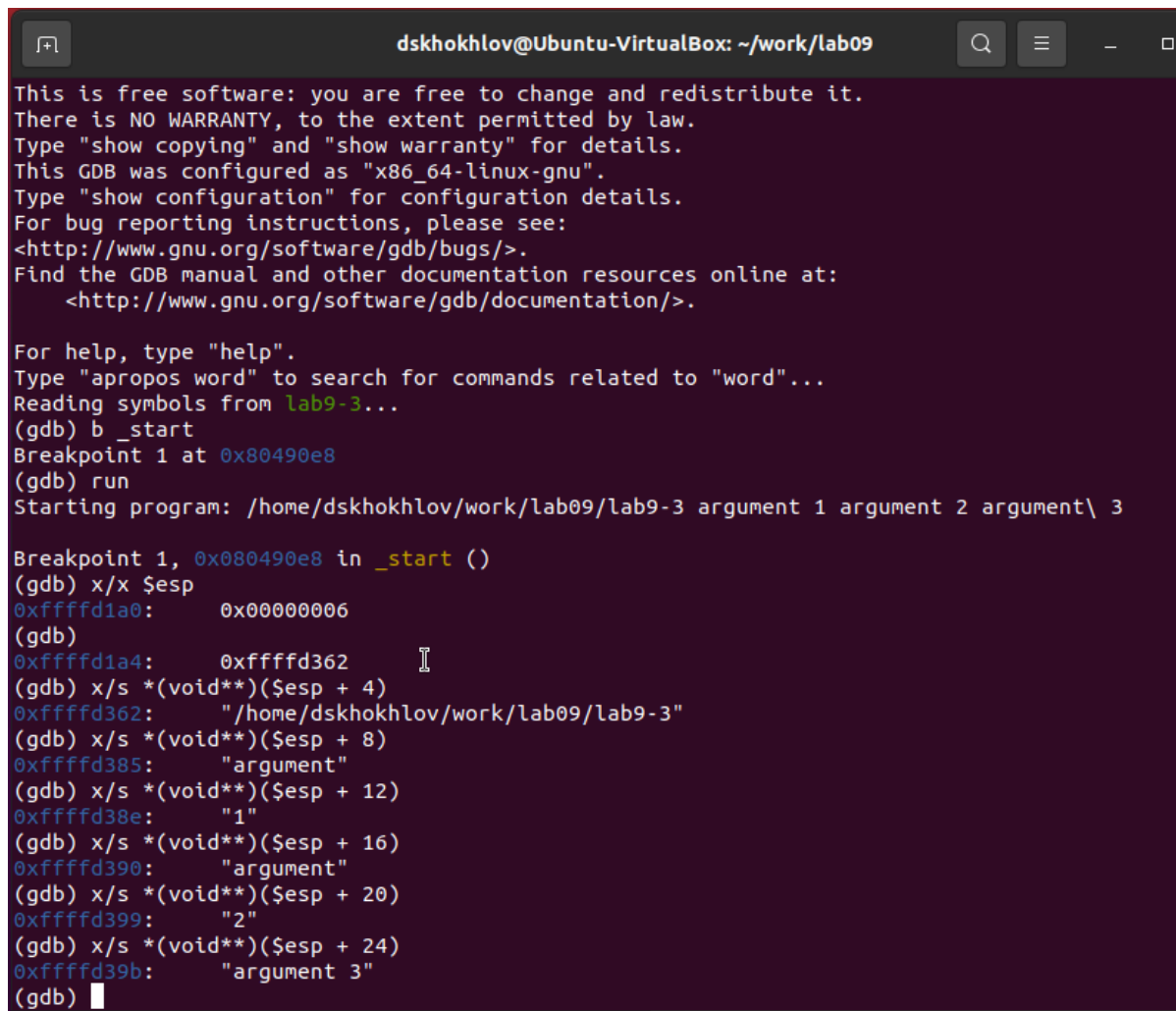
Для инициализации программы с параметрами в отладчике gdb применил опцию `-args`, после чего загрузил в gdb исполняемый файл вместе с заданными параметрами.

Установил брейкпойнт на начало выполнения программы и запустил ее.

В регистре `esp` хранится адрес вершины стека, где находится количество переданных аргументов командной строки (с учетом названия самой программы). Из содержимого по этому адресу видно, что число аргументов

составляет 5, включая наименование программы lab9-3 и параметры: аргумент1, аргумент2, 'аргумент 3'.

Осмотрел другие ячейки стека. Под адресом [esp+4] расположен указатель на имя программы в памяти. Адреса последующих аргументов расположены в ячейках [esp+8], [esp+12] и далее. (см. рис. [2.15])

A screenshot of a GDB terminal window. The window title is 'dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09'. The terminal shows the GDB startup screen with instructions on how to use it. The user enters 'b \_start' to set a breakpoint at the start of the program. Then they enter 'run' to start the program. The terminal shows the program arguments: '/home/dskhokhlov/work/lab09/lab9-3 argument 1 argument 2 argument\ 3'. The user then enters 'x/x \$esp' to display the stack memory. The output shows several memory addresses and their corresponding values: 0xffffd1a0: 0x00000006, 0xffffd1a4: 0xffffd362, 0xffffd362: "/home/dskhokhlov/work/lab09/lab9-3", 0xffffd385: "argument", 0xffffd38e: "1", 0xffffd390: "argument", 0xffffd399: "2", 0xffffd39b: "argument 3". The cursor is at the end of the last line.

```
dskhokhlov@Ubuntu-VirtualBox: ~/work/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 0x080490e8
(gdb) run
Starting program: /home/dskhokhlov/work/lab09/lab9-3 argument 1 argument 2 argument\ 3

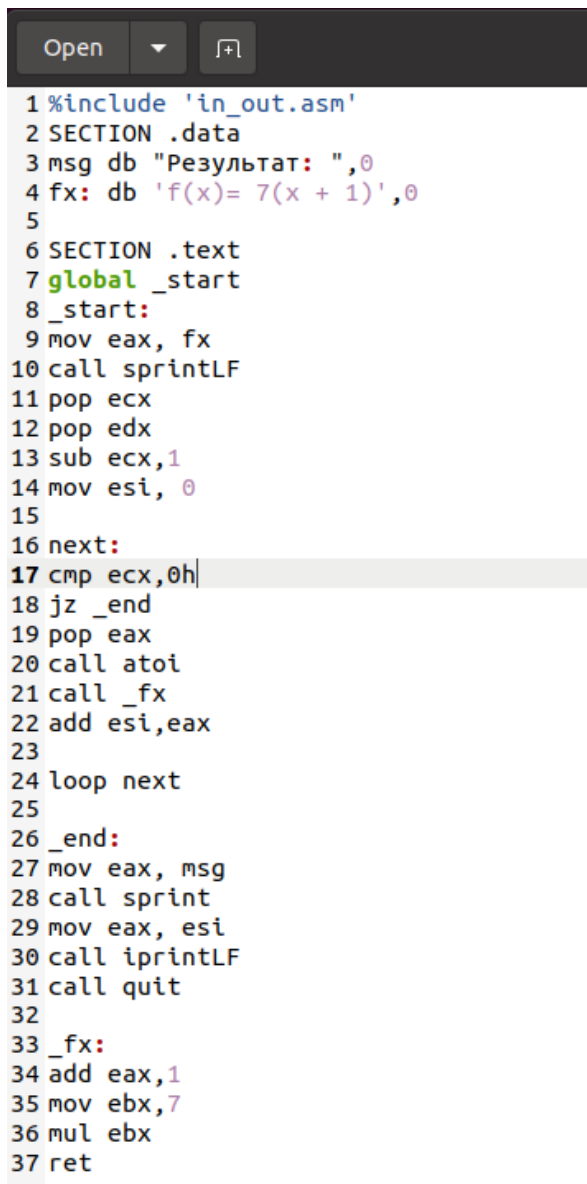
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd1a0:      0x00000006
(gdb)
0xffffd1a4:      0xffffd362
(gdb) x/s *(void**)(esp + 4)
0xffffd362:      "/home/dskhokhlov/work/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd385:      "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd38e:      "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd390:      "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd399:      "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd39b:      "argument 3"
(gdb)
```

Рис. 2.15: Вывод значения регистра

Размер шага между адресами составляет 4 байта, что обусловлено расположением адресов в стеке через каждые 4 байта от предыдущего ([esp+4], [esp+8], [esp+12]).

## 2.1 Самостоятельное задание

Изменил код программы из лабораторной работы №8 (Задание №1 для индивидуального выполнения), создав подпрограмму для расчета функции  $f(x)$ . (см. рис. [2.16]) (см. рис. [2.17])



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 7(x + 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
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 _fx
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprint
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _fx:
34 add eax,1
35 mov ebx,7
36 mul ebx
37 ret
```

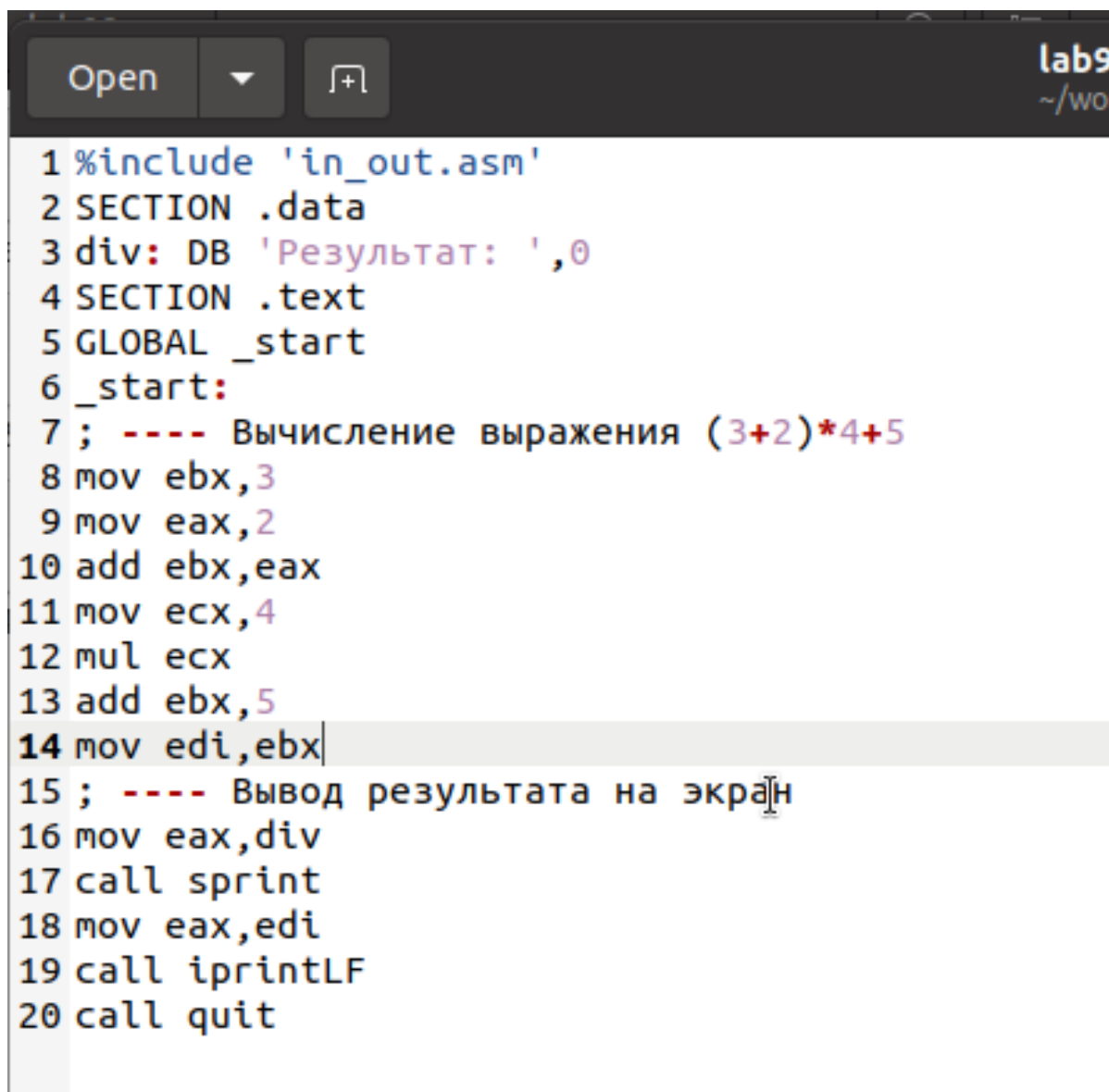
Рис. 2.16: Программа в файле lab9-3.asm

```
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ nasm -f elf lab9-4.asm
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ./lab9-4 1
f(x)= 7(x + 1)
Результат: 14
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$ ./lab9-4 1 6 5 7 8
f(x)= 7(x + 1)
Результат: 224
dskhokhlov@Ubuntu-VirtualBox:~/work/lab09$
```

Рис. 2.17: Запуск программы lab9-3.asm

Программа, представленная в листинге, предназначена для вычисления выражения  $(3 + 2) * 4 + 5$ . Однако при ее выполнении получается некорректный результат, что было выявлено с помощью отладки и анализа регистров в GDB.

Выяснил, что ошибка заключается в неправильной последовательности аргументов в инструкции `add`, и обнаружил, что в конце выполнения в регистр `edi` записывается значение из `ebx` вместо `eax`. (см. рис. [2.18])

A screenshot of a code editor window. The title bar shows 'lab9' and '~/' followed by a partially visible 'wo'. The editor has a dark theme. The code is written in assembly language. Line 14 is highlighted in grey. Line 15 contains a comment in Russian. Line 16 has a syntax error: 'div' is not a valid register name. The code is as follows:

```
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

```

dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09

eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10
eip      0x8049100 0x8049100 <_start+24>
eflags   0x206    [ PF IF ]

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,ebx04a000
0x8049100 <_start+24>   mov     eax,0x804a000rint>
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi

native process 4739 In: _start L?? PC: 0x8049100
(gdb) sNo 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: 10
[Inferior 1 (process 4739) exited normally]
(gdb)

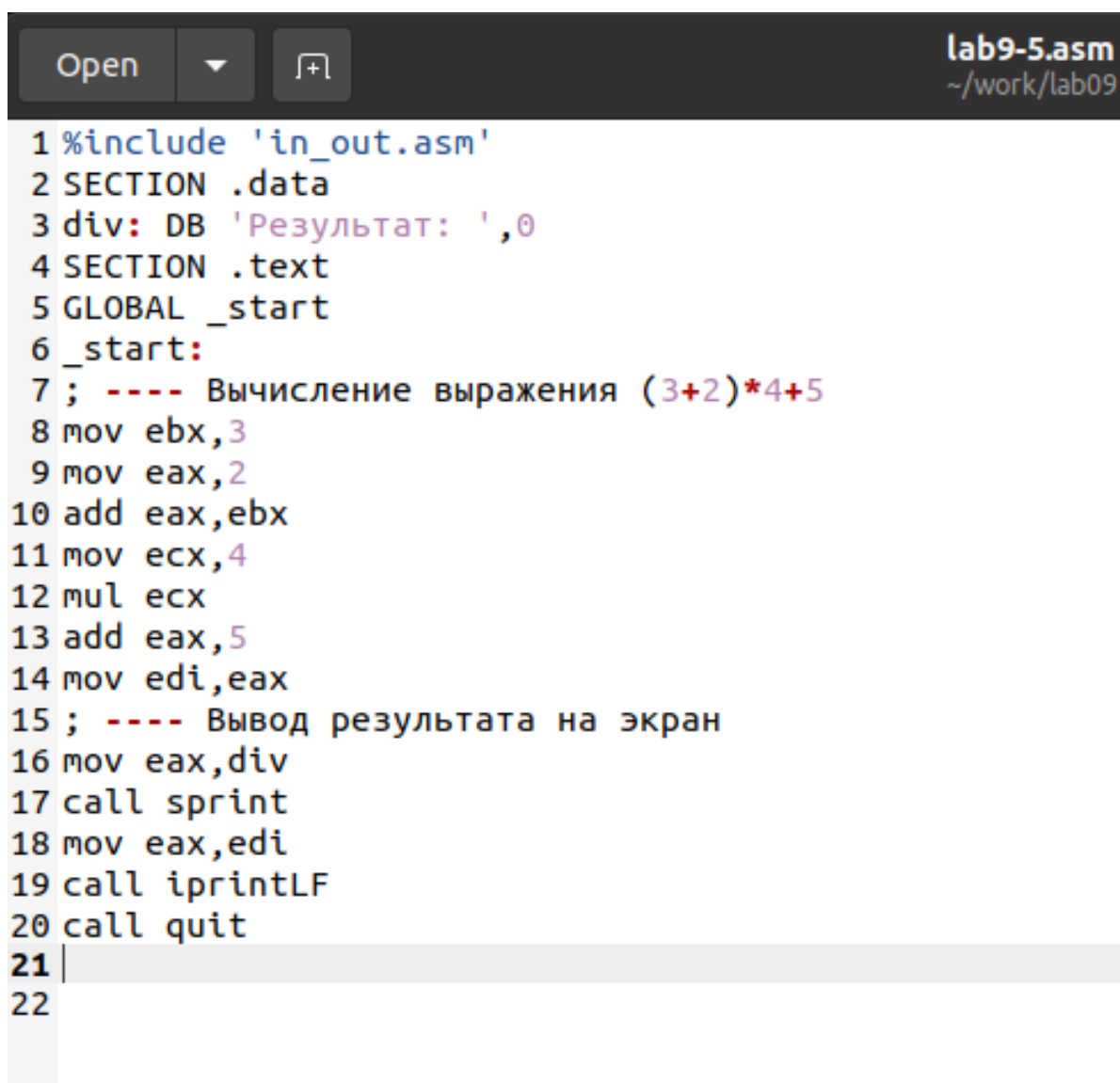
```

Рис. 2.19: Отладка

Указываю на неверное использование аргументов в инструкции add и на то, что в конце программы в регистр edi передается значение из ebx, а не из eax (см. рис. [2.19])

Корректный вариант исходного кода программы представлен далее (см. рис. [2.20]) (см. рис. [2.21]).





The image shows a screenshot of an assembly code editor. The title bar at the top right indicates the file is 'lab9-5.asm' located at '~/work/lab09'. The editor contains 22 lines of assembly code. Line 7 includes a comment in Russian: 'Вычисление выражения (3+2)\*4+5'. The code uses registers ebx, eax, ecx, and edi to perform the calculation. Line 15 includes another comment: 'Вывод результата на экран'. The program uses 'sprint' and 'iprintLF' for output and 'quit' to end execution. Line 21 is highlighted with a light gray background.

```
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
21 |
22
```

Рис. 2.20: Код исправлен

```
dskhokhlov@Ubuntu-VirtualBox: ~/work/lab09

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1e0 0xffffd1e0
ebp      0x0       0x0
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,edi

native process 4751 In: _start L?? PC:
(gdb) sNo process In: ) L?
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 4751) exited normally]
(gdb) █
```

Рис. 2.21: Проверка работы

## **3 Выводы**

Освоили работу с подпрограммами и отладчиком.

## Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).