

# **answer to labrotary work 9**

**Discipline: Computer Architecture**

Ерфан Хосейнабади

# Content

<b>1</b>	<b>Work Goal</b>	<b>5</b>
<b>2</b>	<b>Assignment</b>	<b>6</b>
<b>3</b>	<b>Theoretical Introduction</b>	<b>7</b>
<b>4</b>	<b>Performing Laboratory Work</b>	<b>8</b>
4.1	Implementation of Subroutines in NASM . . . . .	8
4.1.1	Debugging Programs Using GDB . . . . .	13
4.1.2	Adding Breakpoints . . . . .	18
4.1.3	Working with Program Data in GDB . . . . .	20
4.1.4	Processing Command-Line Arguments in GDB . . . . .	25
4.2	Independent Work Assignment . . . . .	27
<b>5</b>	<b>Conclusions</b>	<b>33</b>
<b>6</b>	<b>Bibliography</b>	<b>34</b>

# List of illustrations

4.1	Creating a working directory . . . . .	9
4.2	Running the program from the listing . . . . .	10
4.3	Changing the program of the first listing . . . . .	11
4.4	Running the program in the debugger . . . . .	14
4.5	Checking the program with the debugger . . . . .	15
4.6	Running the debugger with a breakpoint . . . . .	16
4.7	Disassembling the program . . . . .	17
4.8	Pseudo-graphics mode . . . . .	18
4.9	Breakpoint list . . . . .	19
4.10	Adding a second breakpoint . . . . .	20
4.11	Viewing the contents of registers . . . . .	21
4.12	Viewing the contents of variables in two ways . . . . .	22
4.13	Changing the contents of variables in two ways . . . . .	23
4.14	Viewing the register value in different representations . . . . .	24
4.15	Examples of using the set command . . . . .	25
4.16	Preparing a new program . . . . .	26
4.17	Checking the stack operation . . . . .	27
4.18	Modified program of the previous laboratory work . . . . .	28
4.19	Verification of corrections in the program . . . . .	31

## List of Tables

# **1 Work Goal**

Acquiring skills in writing programs using subroutines. Familiarization with debugging methods using GDB and its main capabilities.

## **2 Assignment**

1. Implementation of subroutines in NASM
2. Debugging programs using GDB
3. Independent completion of tasks based on the materials of the laboratory work

### 3 Theoretical Introduction

Debugging is the process of finding and fixing errors in a program. In general, it can be divided into four stages:

- Error detection;
- Locating the error;
- Determining the cause of the error;
- Fixing the error.

The following types of errors can be distinguished:

- Syntax errors — detected during the compilation of the source code and are caused by a violation of the expected form or structure of the language;
- Semantic errors — are logical and lead to the fact that the program starts, runs, but does not give the desired result;
- Runtime errors — are not detected during compilation and cause the program execution to be interrupted (for example, these are errors related to overflow or division by zero).

The second stage is finding the location of the error. Some errors are quite difficult to detect. The best way to find the place in the program where the error is located is to break the program into parts and debug them separately from each other.

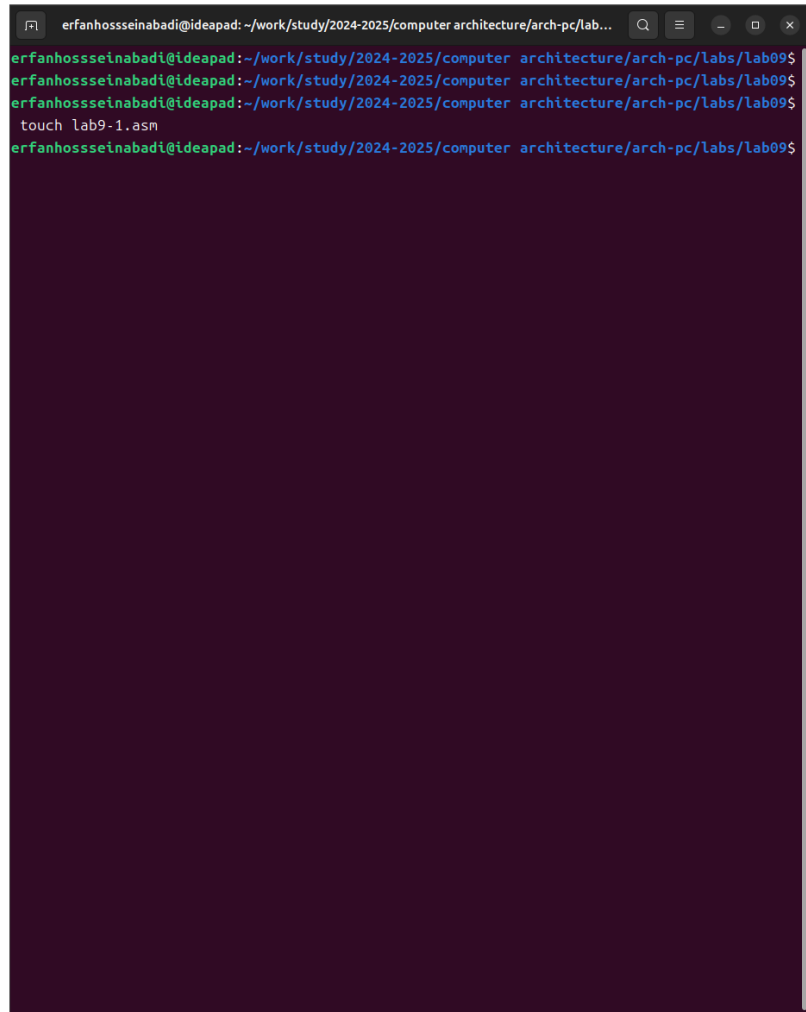
The third stage is determining the cause of the error. After determining the location of the error, it is usually easier to determine the cause of the incorrect operation of the program. The last stage is fixing the error. After that, when the program is restarted, the next error may be found, and the debugging process will start again.

## **4 Performing Laboratory Work**

### **4.1 Implementation of Subroutines in NASM**

I create a directory for performing laboratory work No. 9 (Figure 1).

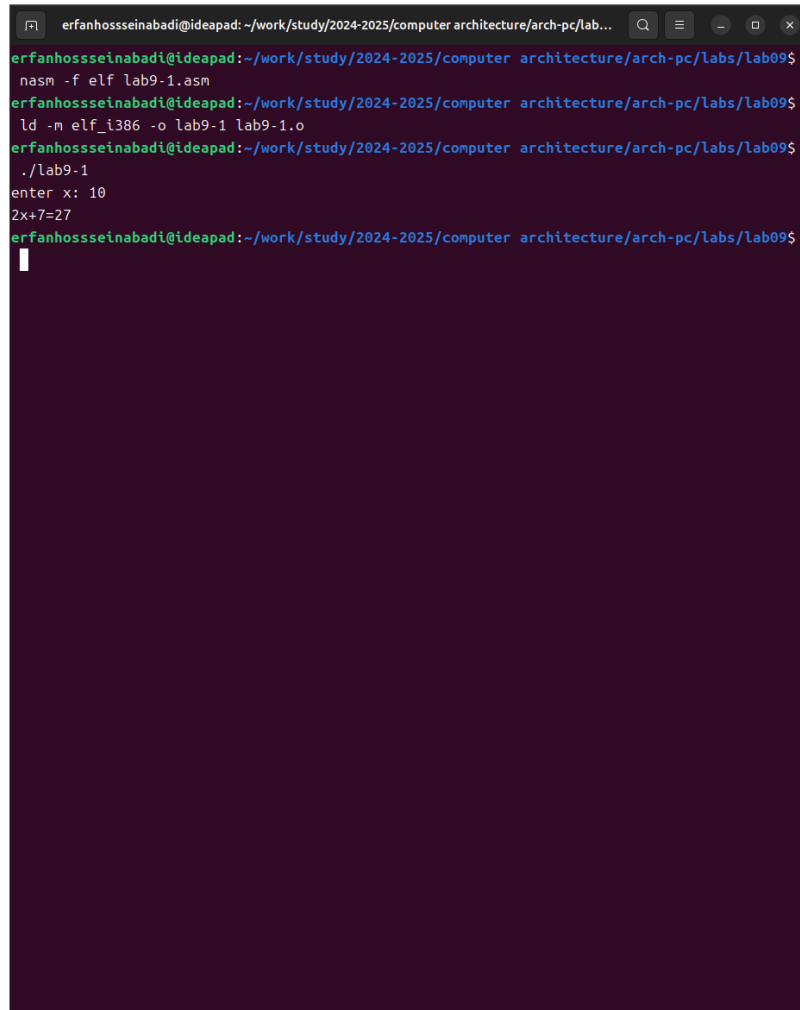


A terminal window with a dark purple background. The window title bar shows the user 'erfanhosseinabadi@ideapad' and the current directory path. The terminal shows three lines of commands and their outputs. The first two lines are identical, showing the current directory. The third line shows the command 'touch lab9-1.asm' and its successful execution.

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
touch lab9-1.asm  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
```

Fig. 4.1: Creating a working directory

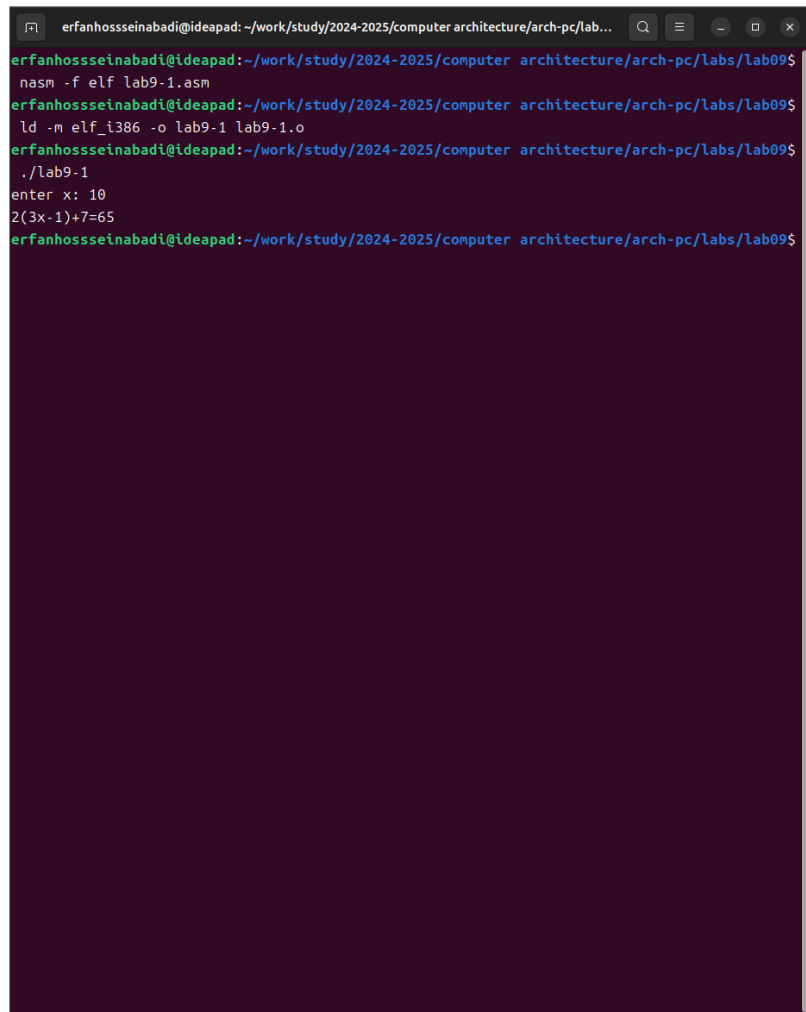
I copy the code from the listing into the file, compile and run it. This program performs the calculation of the function (Figure 2).



```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/labs...  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
nasm -f elf lab9-1.asm  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
ld -m elf_i386 -o lab9-1 lab9-1.o  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
./lab9-1  
enter x: 10  
2x+7=27  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
```

Fig. 4.2: Running the program from the listing

I change the program text by adding a subroutine to it. Now it calculates the value of the function for the expression  $f(g(x))$  (Figure 3).

A terminal window with a dark purple background. The window title is 'erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09'. The terminal shows the following commands and output:

```
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
nasm -f elf lab9-1.asm  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
ld -m elf_i386 -o lab9-1 lab9-1.o  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
./lab9-1  
enter x: 10  
2(3x-1)+7=65  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
```

Fig. 4.3: Changing the program of the first listing

Program code:

```
%include 'in_out.asm'  
  
SECTION .data  
msg: DB 'enter x: ', 0  
result: DB '2(3x-1)+7=', 0  
  
SECTION .bss  
x: RESB 80
```

```

res: 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

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
push eax
call _subcalcul

mov ebx, 2

```

```
mul ebx
add eax, 7

mov [res], eax
pop eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

#### 4.1.1 Debugging Programs Using GDB

I copy the program from the second listing into the created file, translate it with the creation of a listing and debugging file, link and run it in the debugger (Figure 4).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
nasm -f elf -g -l lab9-2.lst lab9-2.asm  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
ld -m elf_i386 -o lab9-2 lab9-2.o  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
gdb lab9-2  
Command 'gdp' not found, but there are 16 similar ones.  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
sudo dnf install gdb  
[sudo] password for erfanhosseinabadi:  
sudo: dnf: command not found  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
gdb lab9-2  
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git  
Copyright (C) 2024 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:  
<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)
```

Fig. 4.4: Running the program in the debugger

Having run the program with the run command, I made sure that it works correctly (Figure 5).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
nasm -f elf -g -l lab9-2.lst lab9-2.asm
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
ld -m elf_i386 -o lab9-2 lab9-2.o
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
gdb lab9-2
Command 'gdb' not found, but there are 16 similar ones.
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
sudo dnf install gdb
[sudo] password for erfanhosseinabadi:
sudo: dnf: command not found
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, World!
[Inferior 1 (process 12941) exited normally]
(gdb) █
```

Fig. 4.5: Checking the program with the debugger

For a more detailed analysis of the program, I add a breakpoint to the `_start` label and run the debugging again (Figure 6).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
nasm -f elf -g -l lab9-2.lst lab9-2.asm
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
ld -m elf_i386 -o lab9-2 lab9-2.o
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 16188) exited normally]
(gdb) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2
Hello, world!
[Inferior 1 (process 16191) exited normally]
(gdb)
```

Fig. 4.6: Running the debugger with a breakpoint

Next, I look at the disassembled code of the program, translated into a command with Intel syntax (Figure 7).

The differences between ATT and Intel syntax are in the order of operands (ATT: source operand first; Intel: destination operand first), their size (ATT: explicitly specified with suffixes; Intel: implicitly determined by context), and register names (ATT: preceded by '%'; Intel: without prefixes).



```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
Copyright (C) 2024 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:
<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) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 16188) exited normally]
(gdb) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2
Hello, world!
[Inferior 1 (process 16191) exited normally]
(gdb) break
No default breakpoint address now.
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)
```

Fig. 4.7: Disassembling the program

I enable pseudo-graphics mode for easier analysis of the program (Figure 8).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
(gdb) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-2
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab9-2.asm:9
9      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     $0x804a000,%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
No symbol "disassembly_flavor" in current context.
(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,0x804a000
      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)
```

Fig. 4.8: Pseudo-graphics mode

## 4.1.2 Adding Breakpoints

I check in pseudo-graphics mode that the breakpoint is saved (Figure 9).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-202... x erfanhosseinabadi@ideapad: ~/work/study/2024-202...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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
0x8049031 <_start+49> mov ebx,0x0

native process 24943 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type      Disp Enb Address      What
1     breakpoint keep y 0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(gdb)
```

Fig. 4.9: Breakpoint list

I set another breakpoint at the instruction address (Figure 10).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-2025... x erfanhosseinabadi@ideapad: ~/work/study/2024-2025...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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
0x8049031 <_start+49> mov ebx,0x0

native process 24943 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type      Disp Enb Address      What
1     breakpoint keep y 0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(gdb)
```

Fig. 4.10: Adding a second breakpoint

### 4.1.3 Working with Program Data in GDB

I view the contents of the registers using the `info registers` command (Figure 11).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-202... x erfanhosseinabadi@ideapad: ~/work/study/2024-202...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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 26536 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type      Disp Enb Address  What
1     breakpoint keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type      Disp Enb Address  What
1     breakpoint keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint keep y  0x08049031 lab9-2.asm:20
(gdb) |
```

Fig. 4.11: Viewing the contents of registers

I look at the contents of the variables by name and by address (Figure 12).

```

erfanhossseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhossseinabadi@ideapad: ~/work/study/2024-2025... x erfanhossseinabadi@ideapad: ~/work/study/2024-2025...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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 26536 (asm) In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Fig. 4.12: Viewing the contents of variables in two ways

I change the contents of variables by name and by address (Figure 13).

```

erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-2025... x erfanhosseinabadi@ideapad: ~/work/study/2024-2025...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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 26536 (asm) In: _start L9 PC: 0x8049000
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Fig. 4.13: Changing the contents of variables in two ways

I output the value of the edx register in various formats (Figure 14).

```

erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-2025... x erfanhosseinabadi@ideapad: ~/work/study/2024-2025...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcec0 0xffffcec0
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 26536 (asm) In: _start L9 PC: 0x8049000
ds      0x2b      43
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(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}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "xorld!\n\034"
(gdb)

```

Fig. 4.14: Viewing the register value in different representations

Using the set command, I change the contents of the ebx register (Figure 15).



```

erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad: ~/work/study/2024-2025... x erfanhosseinabadi@ideapad: ~/work/study/2024-2025...

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x32     50
esp      0xffffcec0 0xffffcec0
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 26536 (asm) In: _start L9 PC: 0x8049000
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb) p/t $ecx
$1 = 0
(gdb) set $ebx='2'
(gdb) p/t $ecx
$2 = 0
(gdb) p/s $ebx
$3 = 50
(gdb)

```

Fig. 4.15: Examples of using the set command

#### 4.1.4 Processing Command-Line Arguments in GDB

I copy the program from the previous laboratory work to the current directory and create an executable file with a listing and debugging file (Figure 16).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
nasm -f elf -g -l lab9-3.lst lab9-3.asm
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
ld -m elf_i386 -o lab9-3 lab9-3.o
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
gdb -args lab9-3 "arg1" "arg2" "arg3"
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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"...
lab09-3: No such file or directory.
(gdb) Quit
(gdb) quit
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
gdb -args lab9-3 "arg1" "arg2" "arg3"
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab9-3...
(gdb) break _start
```

Fig. 4.16: Preparing a new program

I run the program in debug mode specifying arguments, specify a breakpoint and start debugging. I check the operation of the stack, changing the argument of the command to view the esp register to +4 (the number is determined by the system's bit depth, and a void pointer occupies 4 bytes); an error with the argument +24 means that the input program arguments have ended. (Figure 17).

```
erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/lab...
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<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 lab9-3...
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-
pc/labs/lab09/lab9-3 arg1 arg2 arg3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab9-3.asm:7
7      pop ecx
(gdb) x/s *(void**)($esp + 4)
0xffffd07b:  "/home/erfanhosseinabadi/work/study/2024-2025/computer architecture/arch-p
c/labs/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd0d9:  "arg1"
(gdb) x/s *(void**)($esp + 12)
0xffffd0de:  "arg2"
(gdb) x/s *(void**)($esp + 16)
0xffffd0e3:  "arg3"
(gdb) x/s *(void**)($esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Fig. 4.17: Checking the stack operation

## 4.2 Independent Work Assignment

1. I change the program of the independent part of the previous laboratory work using a subroutine (Figure 18).

```
~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09/lab9-4.asm - Mousepad
File Edit Search View Document Help
%include 'in_out.asm'

SECTION .data
msg_func db "Function: f(x) = 5 * (2 + x)", 0
msg_result db "Result: ", 0

SECTION .bss
x_input resd 1

SECTION .text
GLOBAL _start

calculate_f:
    push ebp
    mov ebp, esp
    mov eax, [ebp+8]
    add eax, 2
    mov ebx, 5
    mul ebx
    mov esp, ebp
    pop ebp
    ret

_start:
    mov eax, msg_func
    call sprintf
    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    push eax
    call calculate_f
    add esi, eax
    loop next

_end:
    mov eax, msg_result
    call sprintf
    mov eax, esi
    call iprintLF
    call quit
```

Fig. 4.18: Modified program of the previous laboratory work

Program code:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Function: f(x) = 5 * (2 + x)", 0
msg_result db "Result: ", 0

SECTION .bss
x_input resd 1
```

```
SECTION .text
```

```
GLOBAL _start
```

```
calculate_f:
```

```
    push ebp
```

```
    mov ebp, esp
```

```
    mov eax, [ebp+8]
```

```
    add eax, 2
```

```
    mov ebx, 5
```

```
    mul ebx
```

```
    mov esp, ebp
```

```
    pop ebp
```

```
    ret
```

```
_start:
```

```
    mov eax, msg_func
```

```
    call sprintf
```

```
    pop ecx
```

```
    pop edx
```

```
    sub ecx, 1
```

```
    mov esi, 0
```

```
next:
```

```
    cmp ecx, 0h
```

```
    jz _end
```

```
    pop eax
```

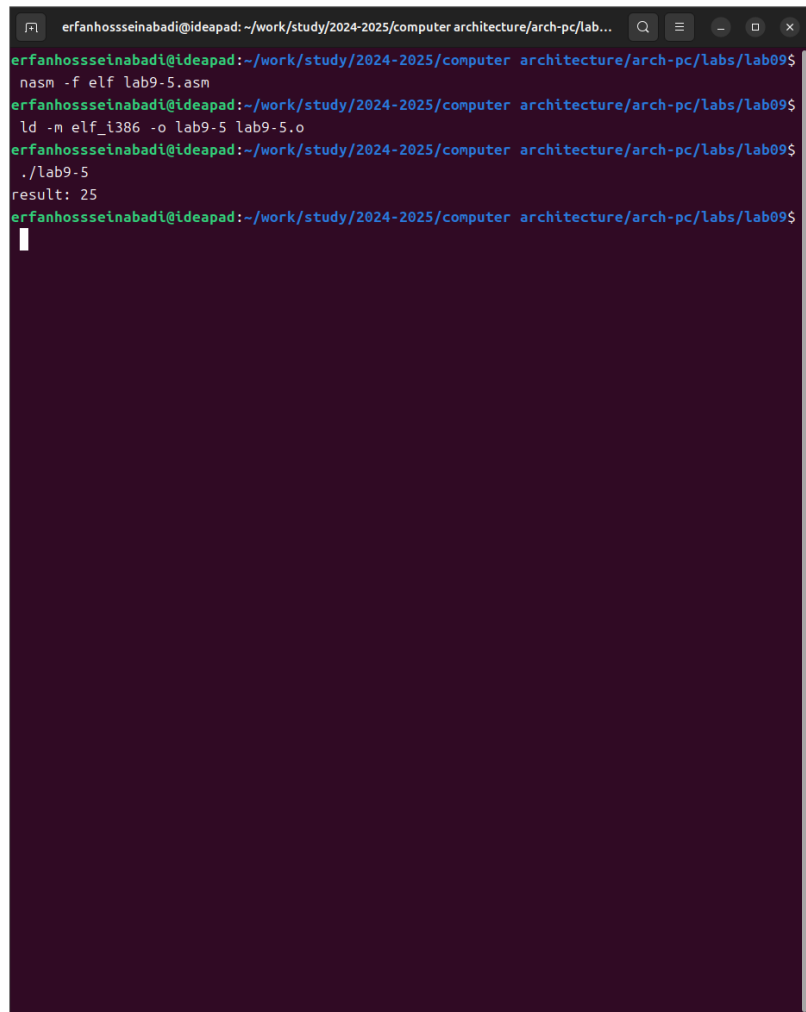
```
    push eax
```

```
    call calculate_f
```

```
    add esi, eax
    loop next

_end:
    mov eax, msg_result
    call sprint
    mov eax, esi
    call iprintLF
    call quit
```

I correct the found error; now the program correctly calculates the value of the function (Figure 20).

A terminal window with a dark purple background. The window title is 'erfanhosseinabadi@ideapad: ~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09'. The terminal shows the following commands and output:

```
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
nasm -f elf lab9-5.asm  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
ld -m elf_i386 -o lab9-5 lab9-5.o  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$  
./lab9-5  
result: 25  
erfanhosseinabadi@ideapad:~/work/study/2024-2025/computer architecture/arch-pc/labs/lab09$
```

Fig. 4.19: Verification of corrections in the program

Modified program code:

```
%include 'in_out.asm'  
  
SECTION .data  
div: DB 'Result: ', 0  
  
SECTION .text  
GLOBAL _start  
_start:
```

```
mov ebx, 3
mov eax, 2
add ebx, eax
mov eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax
```

```
mov eax, div
call sprint
mov eax, edi
call iprintLF
```

```
call quit
```



## 5 Conclusions

As a result of completing this laboratory work, I acquired skills in writing programs using subroutines, and also became acquainted with debugging methods using GDB and its main capabilities.

## 6 Bibliography

1. Course on TUIS
2. Laboratory work No. 9
3. Programming in NASM Assembly Language Stolyarov A. V.