

answer to labrotary work 9

Discipline: Computer Architecture

ali hosseinabadi

Content

| | | |
|----------|--|-----------|
| 1 | Work Goal | 5 |
| 2 | Assignment | 6 |
| 3 | Theoretical Introduction | 7 |
| 4 | Performing Laboratory Work | 8 |
| 4.1 | Implementation of Subroutines in NASM | 8 |
| 4.1.1 | Debugging Programs Using GDB | 12 |
| 4.1.2 | Adding Breakpoints | 16 |
| 4.1.3 | Working with Program Data in GDB | 18 |
| 4.1.4 | Processing Command-Line Arguments in GDB | 23 |
| 4.2 | Independent Work Assignment | 25 |
| 5 | Conclusions | 30 |
| 6 | Bibliography | 31 |

List of illustrations

| | | |
|------|---|----|
| 4.1 | Creating a working directory | 8 |
| 4.2 | Running the program from the listing | 9 |
| 4.3 | Changing the program of the first listing | 10 |
| 4.4 | Running the program in the debugger | 12 |
| 4.5 | Checking the program with the debugger | 13 |
| 4.6 | Running the debugger with a breakpoint | 14 |
| 4.7 | Disassembling the program | 15 |
| 4.8 | Pseudo-graphics mode | 16 |
| 4.9 | Breakpoint list | 17 |
| 4.10 | Adding a second breakpoint | 18 |
| 4.11 | Viewing the contents of registers | 19 |
| 4.12 | Viewing the contents of variables in two ways | 20 |
| 4.13 | Changing the contents of variables in two ways | 21 |
| 4.14 | Viewing the register value in different representations | 22 |
| 4.15 | Examples of using the set command | 23 |
| 4.16 | Preparing a new program | 24 |
| 4.17 | Checking the stack operation | 25 |
| 4.18 | Modified program of the previous laboratory work | 26 |
| 4.19 | Verification of corrections in the program | 28 |

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).

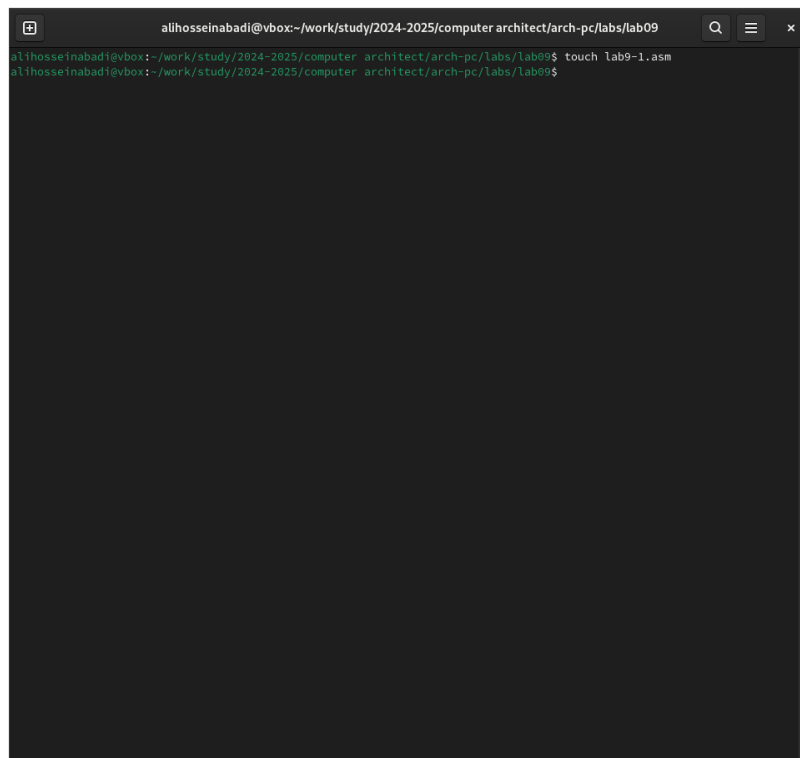
The second stage is finding the location of the error. Some errors are quite difficult to detect. The best way to find the place in the program where the error is located is to break the program into parts and debug them separately from each other.

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

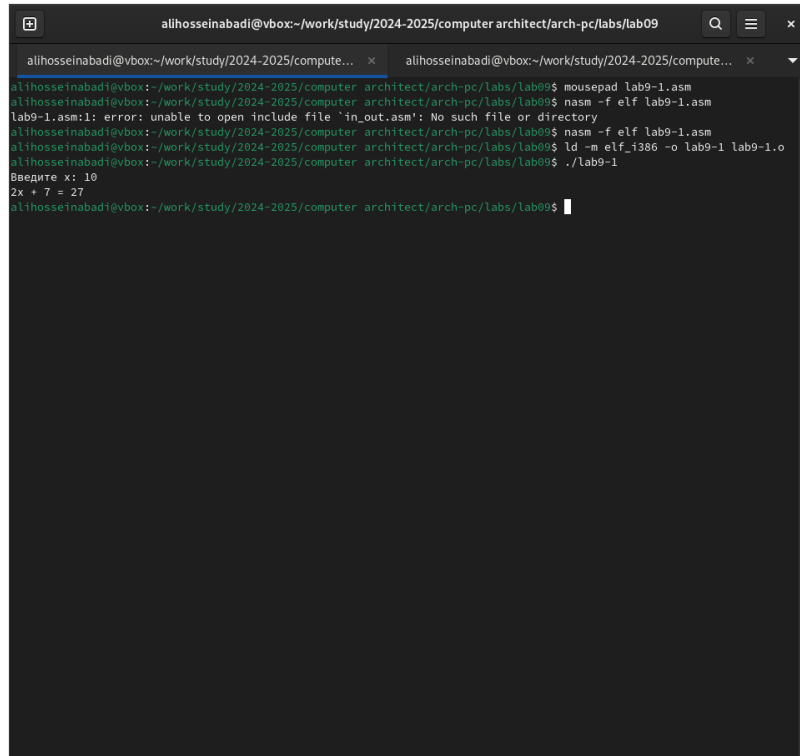
I create a directory for performing laboratory work No. 9 (Figure 1).

A terminal window with a dark background. The title bar at the top reads 'alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09'. The terminal shows two lines of text: the first line is 'alihosseiniabadi@vbox: ~/work/study/2024-2025/computer architect/arch-pc/labs/lab09\$ touch lab9-1.asm' and the second line is 'alihosseiniabadi@vbox: ~/work/study/2024-2025/computer architect/arch-pc/labs/lab09\$'.

```
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09
alihosseiniabadi@vbox: ~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ touch lab9-1.asm
alihosseiniabadi@vbox: ~/work/study/2024-2025/computer architect/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).



```
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ mousepad lab9-1.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
lab9-1.asm:1: error: unable to open include file 'in_out.asm': No such file or directory
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ./lab9-1
Введите x: 10
2x + 7 = 27
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/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).

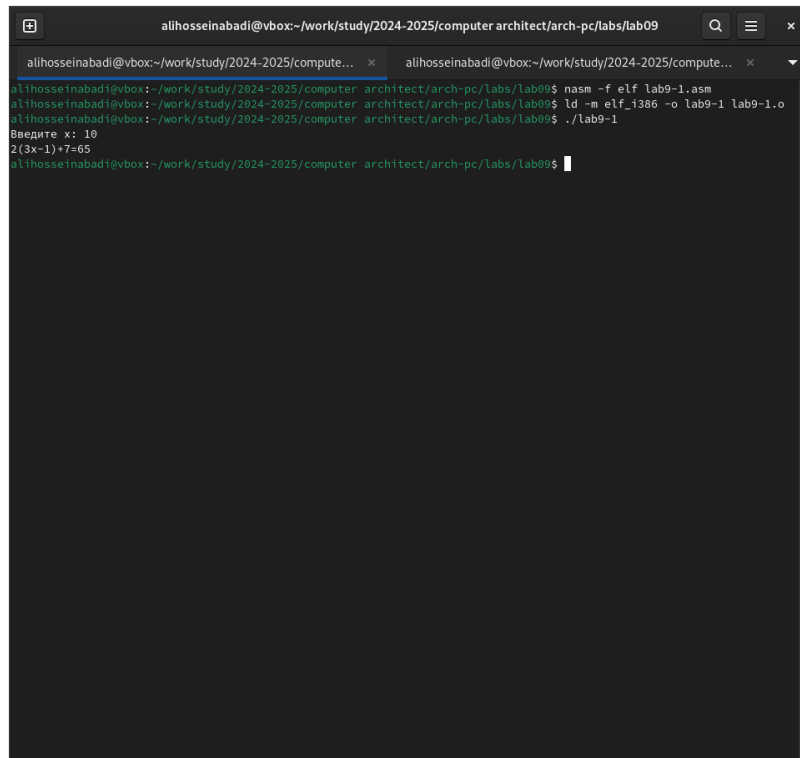
A screenshot of a terminal window with a dark background. The window title is 'alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09'. The terminal shows the following commands and output:
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09\$ nasm -f elf lab9-1.asm
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09\$ ld -m elf_i386 -o lab9-1 lab9-1.o
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09\$./lab9-1
Введите x: 10
2(3x-1)+7=65
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/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).

```

alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ mousepad lab9-2.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf
ld: unrecognized emulation mode: elf
Supported emulations: elf_x86_64 elf32_x86_64 elf_i386 elf_i386pe elf_i386pep i386pep i386pep elf64bp
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.1-1.fc40
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-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) run
Starting program: /home/alhosseinabadi/work/study/2024-2025/computer architect/arch-pc/labs/lab09/lab9-2

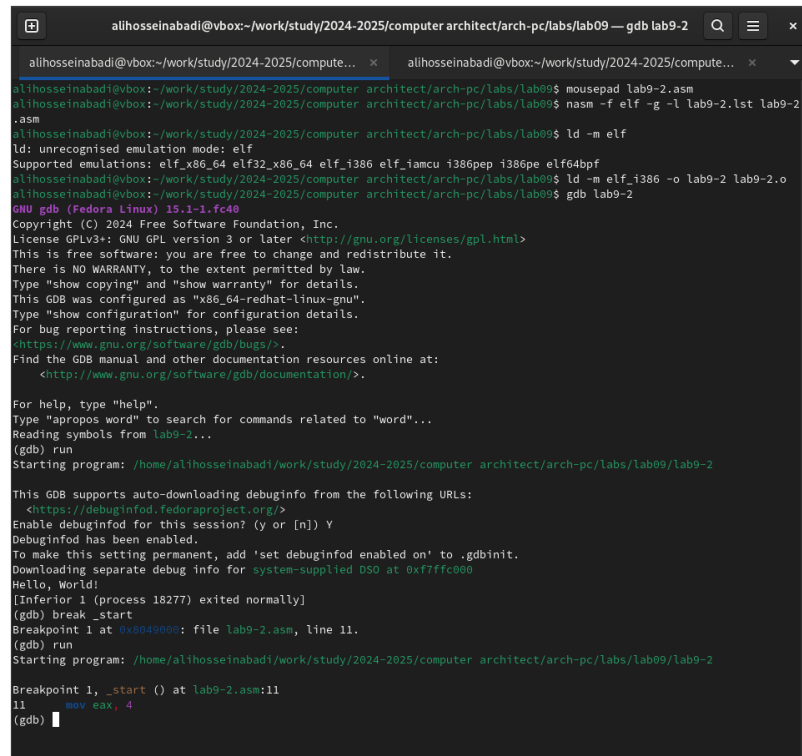
This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) Y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, World!
[Inferior 1 (process 18277) exited normally]
(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).



```
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ gdb lab9-2
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ mousepad lab9-2.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ ld -m elf
ld: unrecognized emulation mode: elf
Supported emulations: elf_x86_64 elf32_x86_64 elf_i386 elf_i386elf_iamcu i386pep i386pe elf64bpf
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
alhosseinabadi@vbox:~/work/study/2024-2025/computer_architect/arch-pc/labs/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.1-1.fc40
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-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) run
Starting program: /home/alhosseinabadi/work/study/2024-2025/computer_architect/arch-pc/labs/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]) 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 18277) exited normally]
(gdb) break _start
Breakpoint 1 at 0x00400000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/alhosseinabadi/work/study/2024-2025/computer_architect/arch-pc/labs/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(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).


```

alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb lab9-2
alhosseinabadi@vbox:~/work/study/2024-2025/compute... x alhosseinabadi@vbox:~/work/study/2024-2025/compute... x
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc900
Hello, World!
[Inferior 1 (process 18277) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/alhosseinabadi/work/study/2024-2025/computer architect/arch-pc/labs/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 $0x1,%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
(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. 4.7: Disassembling the program

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

The screenshot shows the GDB interface in pseudo-graphics mode. The top panel displays the 'Register group: general' with the following values:

| Register | Value |
|----------|------------|
| eax | 0x0 |
| edx | 0x0 |
| esp | 0xffffcfc0 |
| esi | 0x0 |
| eip | 0x8049000 |
| cs | 0x23 |
| ds | 0x2b |
| fs | 0x0 |
| ecx | 0x0 |
| ebx | 0x0 |
| ebp | 0x0 |
| edi | 0x0 |
| eflags | 0x202 |
| ss | 0x2b |
| es | 0x2b |
| gs | 0x0 |

The middle panel shows the assembly code for the function `_start` at address `0x8049000`:

```
0->0x8049000 <_start> mov    eax,0x4
0x8049005 <_start+5> mov    edi,edi
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
0x8049036 <_start+54> int    0x80
```

The bottom panel shows the native process `18300` (asm) in the `_start` function, with the current instruction pointer (PC) at `0x8049000`. The GDB layout is set to `regs`.

Fig. 4.8: Pseudo-graphics mode

4.1.2 Adding Breakpoints

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


```
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb lab9-2
alhosseinabadi@vbox:~/work/study/2024-2025/compute... x alhosseinabadi@vbox:~/work/study/2024-2025/compute... x
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0  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      es       0x2b     43
fs       0x0      0      gs       0x0      0

b+>0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov esi,0x1
0x8049008 <_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
0x8049030 <_start+54> int 0x80

native process 18300 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(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:24
(gdb)
```

Fig. 4.9: Breakpoint list

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

```

alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0  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      es       0x2b      43
fs       0x0      0      gs       0x0      0

b+0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov esi,0x1
0x8049008 <_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
0x8049030 <_start+54> int 0x80

native process 18300 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint      keep y  0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
2     breakpoint      keep y  0x8049031 lab9-2.asm:24
(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).

```

alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab9-2 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0  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      es       0x2b     43
fs       0x0      0      gs       0x0      0

b> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov esi,0x1
0x8049008 <_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
0x8049036 <_start+54> int 0x80

native process 18380 (asm) In: _start L11 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0
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
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Fig. 4.11: Viewing the contents of registers

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

```

alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab9-2 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0  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      es       0x2b     43
fs       0x0      0      gs       0x0      0

b* 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov esi,0x1
0x8049008 <_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
0x8049036 <_start+54> int 0x80

native process 18380 (asm) In: _start L11 PC: 0x8049000
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a000 <msg2>: "World!\n034"
(gdb)

```

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

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

```

alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xfffffc0 0xfffffc0  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      es       0x2b      43
fs       0x0      0      gs       0x0      0

b+0x8049000 <_start> mov    eax,0x4
0x8049005 <_start+5> mov    ebx,0x1
0x8049008 <_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
0x8049036 <_start+54> int    0x80

native process 18380 (asm) In: _start L11 PC: 0x8049000
ds      0x2b      43
es      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
0x804a000 <msg2>: "xor!\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).

```
alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcfc0 0xffffcfc0  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      es       0x2b      43
fs       0x0      0      gs       0x0      0

b> 0x8049000 <_start> mov     eax,0x4
0x8049005 <_start+5> mov     esi,0x1
0x8049009 <_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,0x804a000
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 18300 (asm) In: _start L11 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>: "xorId!\n\034"
(gdb) p/t $ecx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/t $edx
$3 = 0
(gdb) p/x $edx
$4 = 0x0
(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).

```

alihosseinaadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab9-2 — gdb lab9-2
alihosseinaadi@vbox:~/work/study/2024-2025/compute... x alihosseinaadi@vbox:~/work/study/2024-2025/compute... x
Register group: general
eax 0x0 0 ecx 0x0 0
edx 0x0 0 ebx 0x2 2
esp 0xffffcfc0 0xffffcfc0 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 es 0x2b 43
fs 0x0 0 gs 0x0 0

b+0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov esi,0x1
0x8049008 <_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
0x8049036 <_start+54> int 0x80

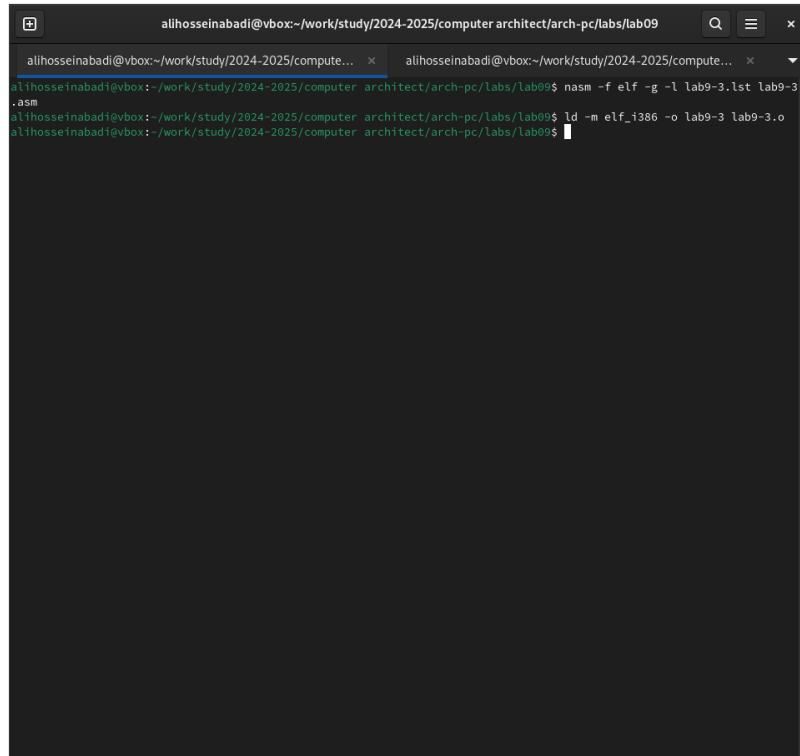
native process 18300 (asm) In: _start L11 PC: 0x8049000
(gdb) p/s $edx
$2 = 0
(gdb) p/t $edx
$3 = 0
(gdb) p/x $edx
$4 = 0x0
(gdb) set $ebx='2'
(gdb) p/s
$5 = 0
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb)

```

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).

A terminal window with a dark background and light text. The window title is 'alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09'. The terminal shows the following commands and their outputs:

```
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
alihosseiniabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
```

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).


```
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09 — gdb --args lab9-3...
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ gdb --args lab9-3 arg1 arg2 'arg3'
GNU gdb (Fedora Linux) 15.1-1.fc49
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-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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e0: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/alhosseinabadi/work/study/2024-2025/computer architect/arch-pc/labs/lab09/lab9-3 arg1 arg2 ar
g3

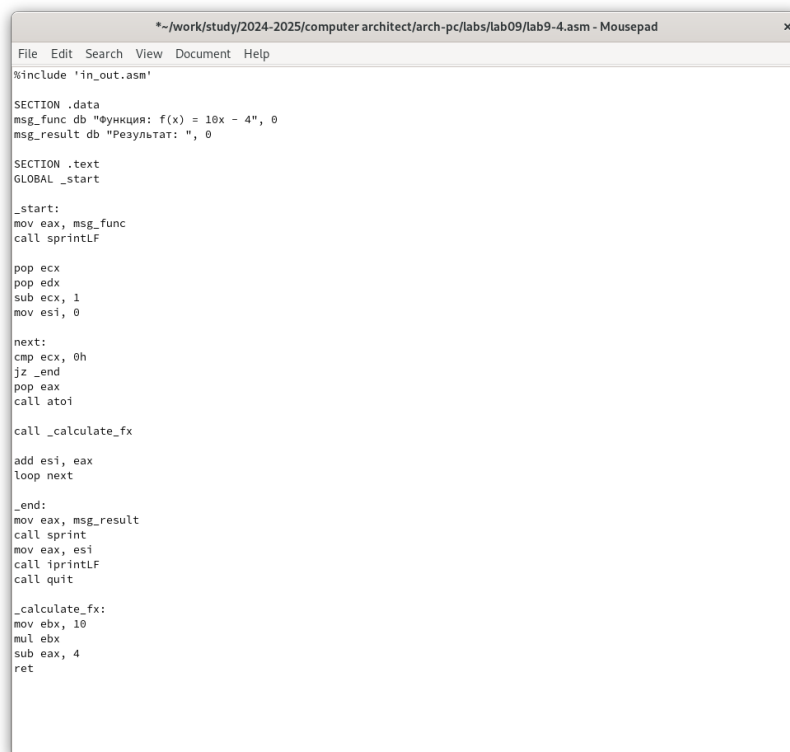
This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:7
7      pop ecx
(gdb) x/s *(void**)(esp + 4)
0xffffd11a: "/home/alhosseinabadi/work/study/2024-2025/computer architect/arch-pc/labs/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd172: "arg1"
(gdb) x/s *(void**)(esp + 12)
0xffffd177: "arg2"
(gdb) x/s *(void**)(esp + 16)
0xffffd17c: "arg3"
(gdb) x/s *(void**)(esp + 20)
***: <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 architect/arch-pc/labs/lab09/lab9-4.asm - Mousepad
File Edit Search View Document Help
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_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
call atoi

call _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintf
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
ret
```

Fig. 4.18: Modified program of the previous laboratory work

Program code:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_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
call atoi

call _calculate_fx

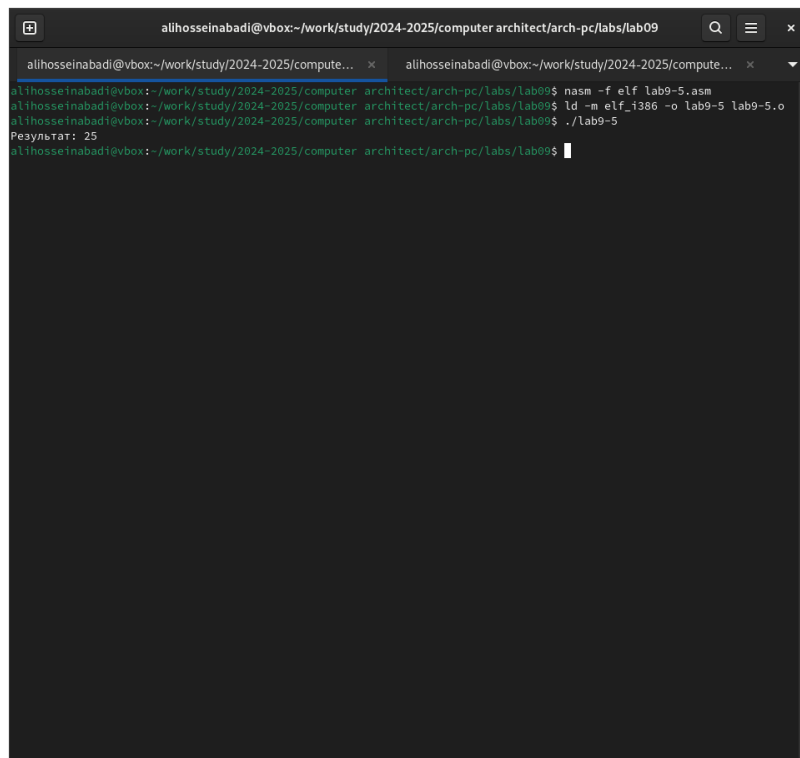
add esi, eax
loop next

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

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
ret

```

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



```
alhosseinabadi@vbox: ~/work/study/2024-2025/computer architect/arch-pc/labs/lab09
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ nasm -f elf lab9-5.asm
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/arch-pc/labs/lab09$ ./lab9-5
Результат: 25
alhosseinabadi@vbox:~/work/study/2024-2025/computer architect/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.