answer to labrotary work 9

Discipline: Computer Architecture

ali hosseinabadi

Content

# 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).

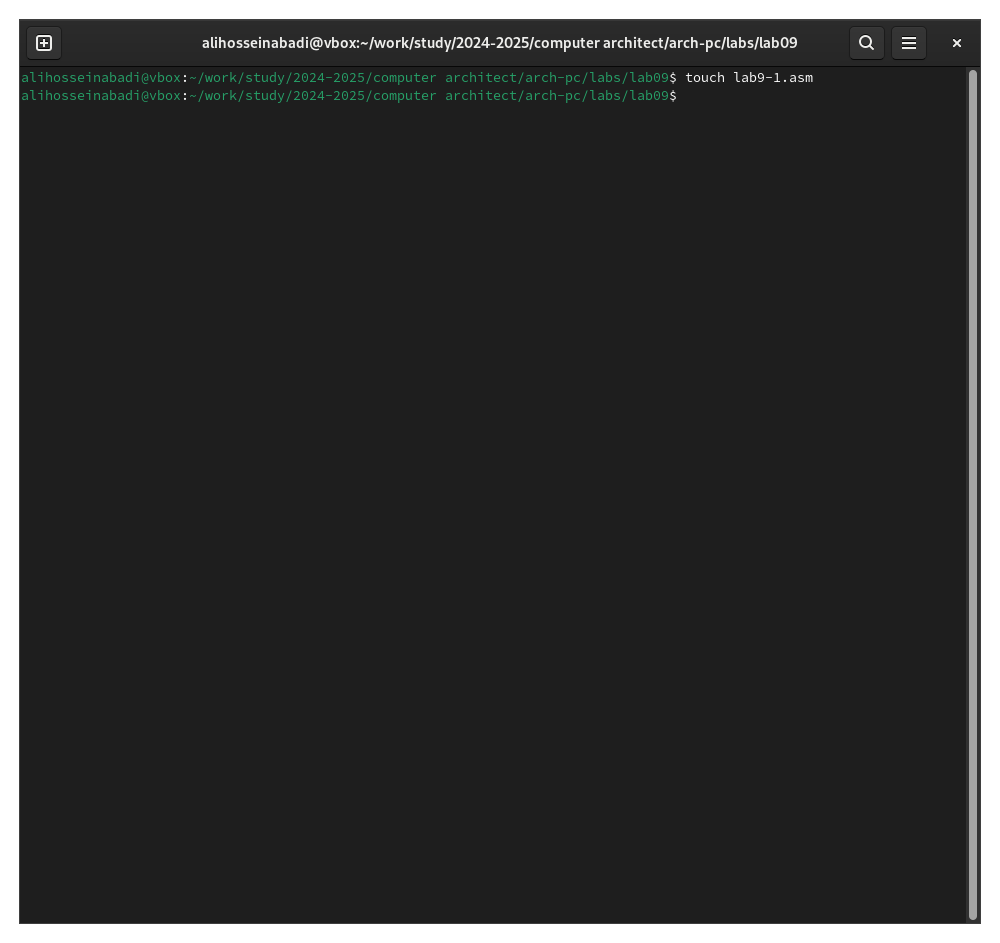


Fig. 1: Creating a working directory

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

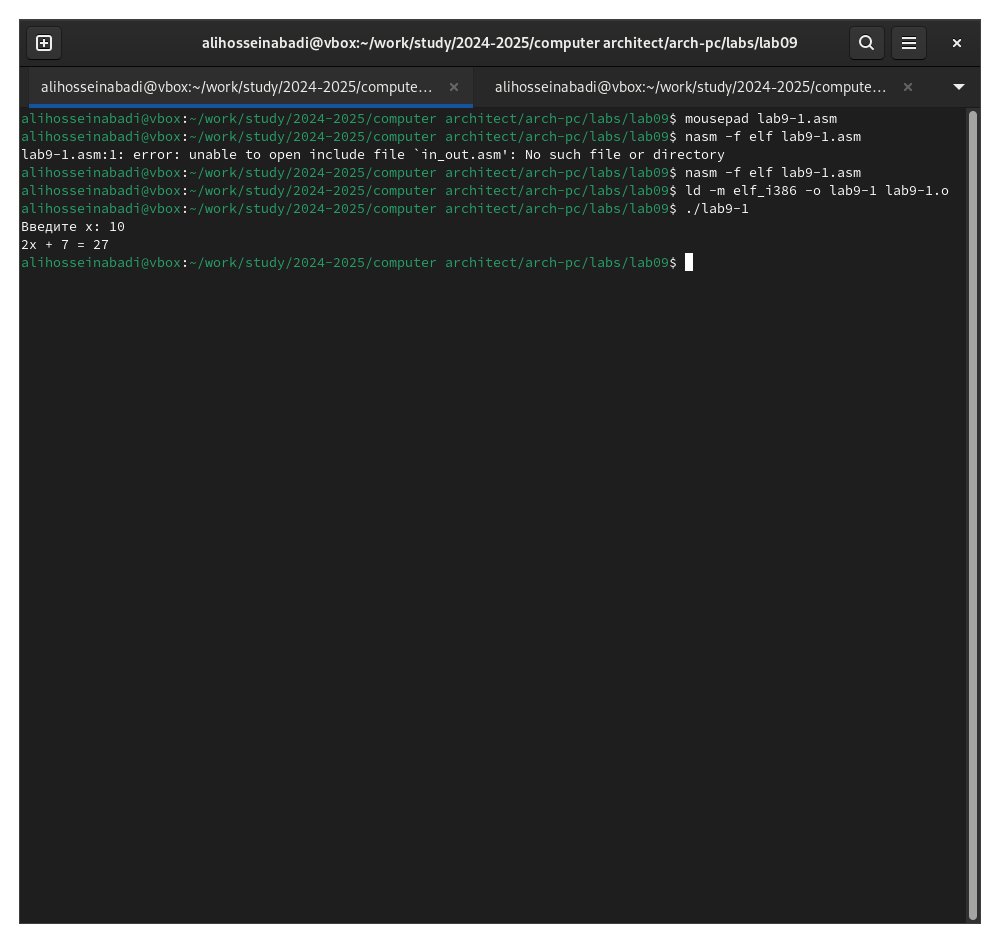


Fig. 2: Running the program from the listing

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).

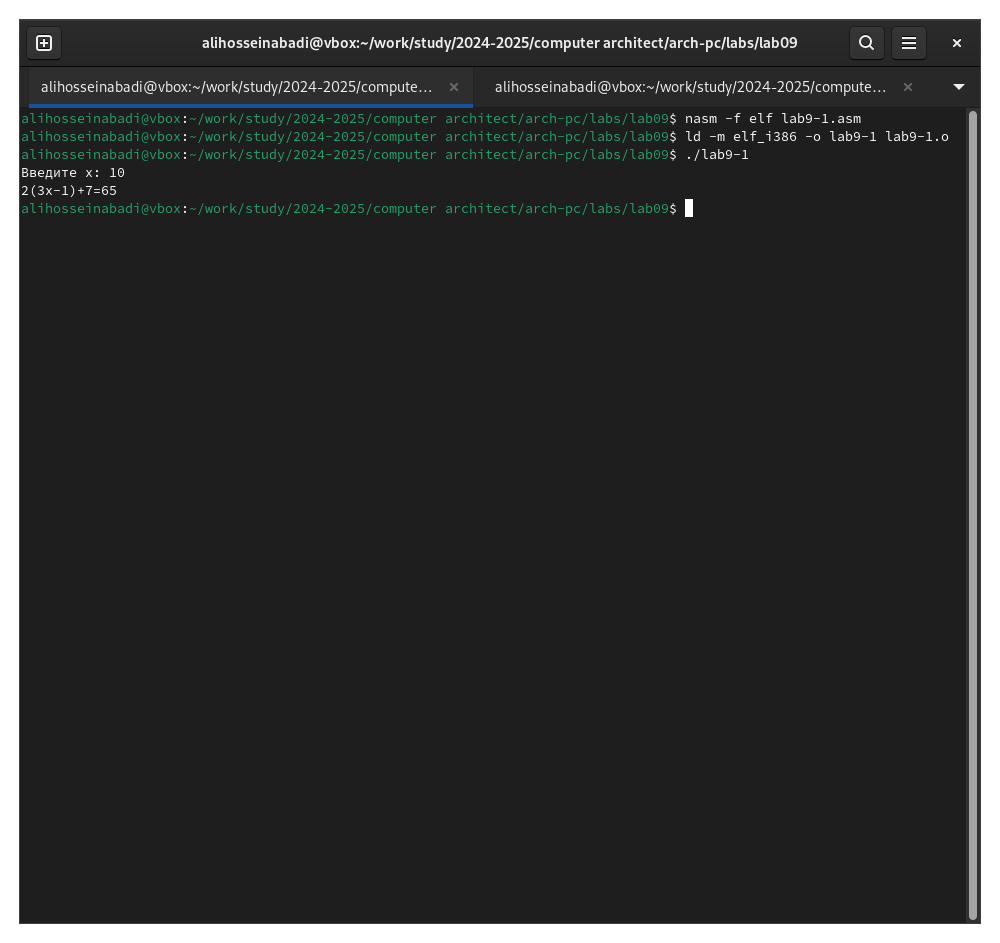


Fig. 3: Changing the program of the first listing

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).

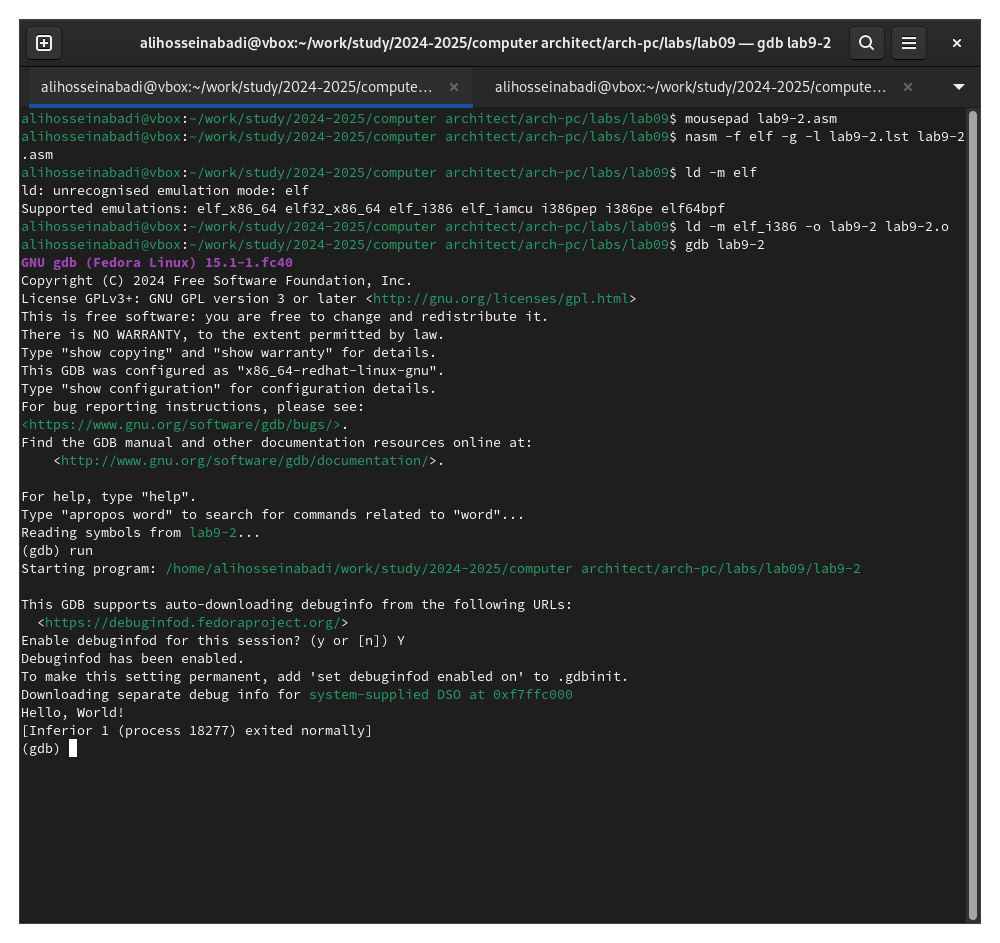


Fig. 4: Running the program in the debugger

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

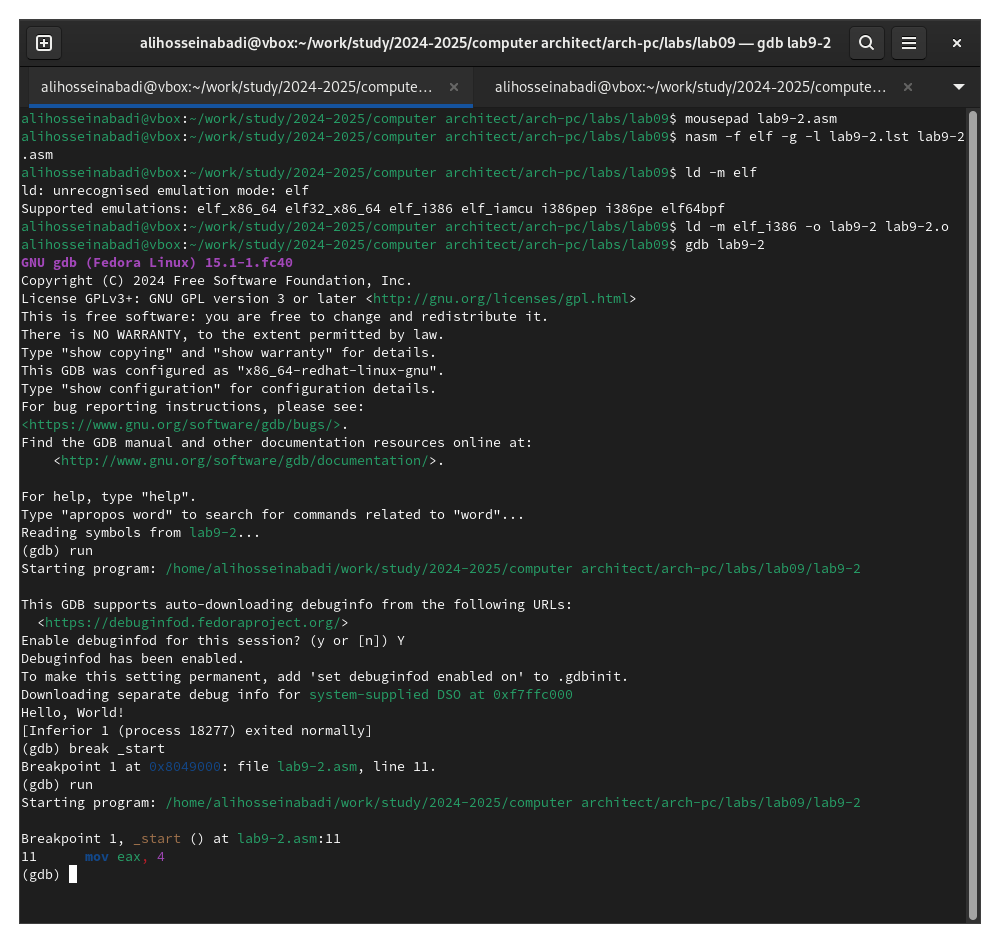


Fig. 5: Checking the program with the debugger

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

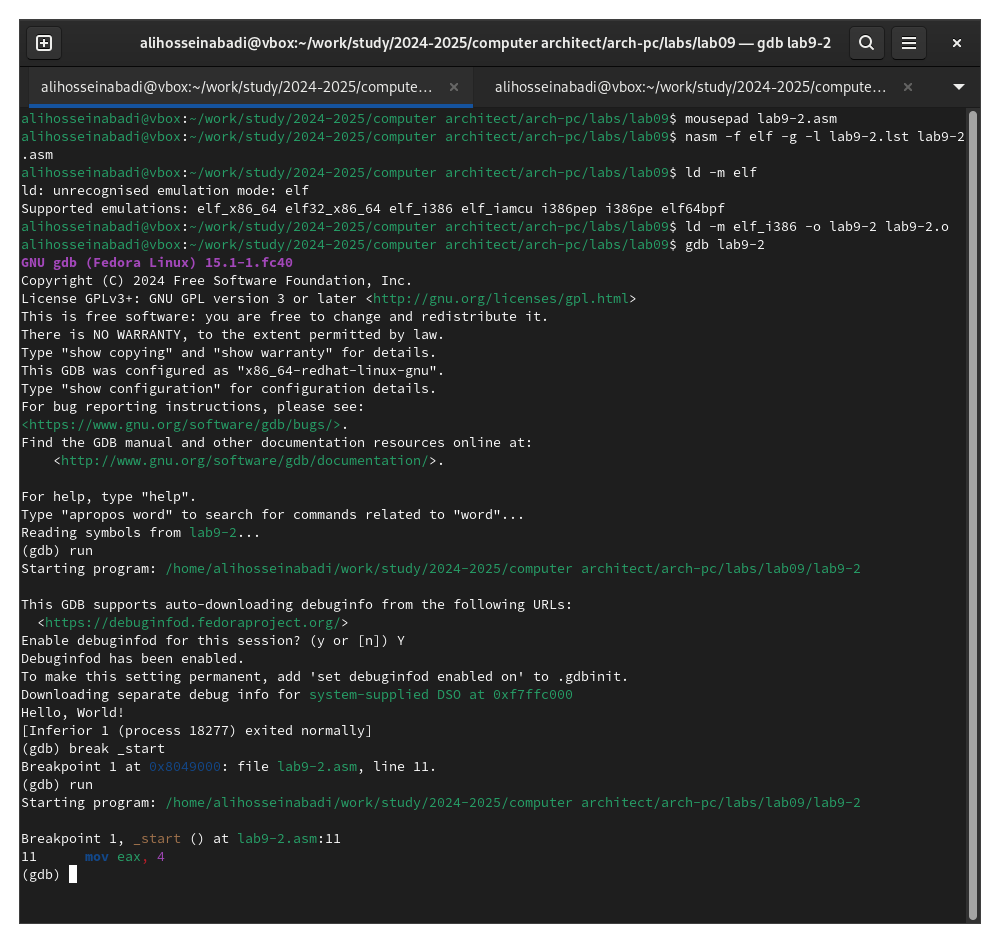


Fig. 6: Running the debugger with a breakpoint

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).

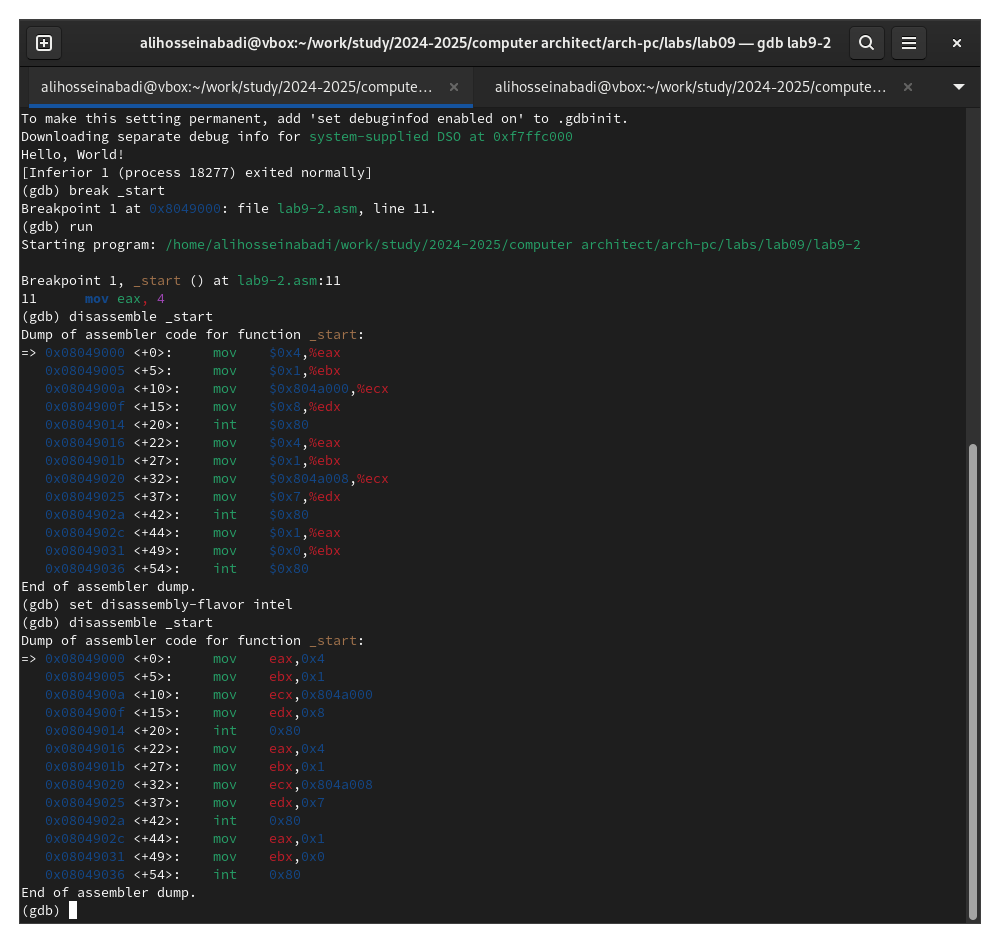


Fig. 7: Disassembling the program

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

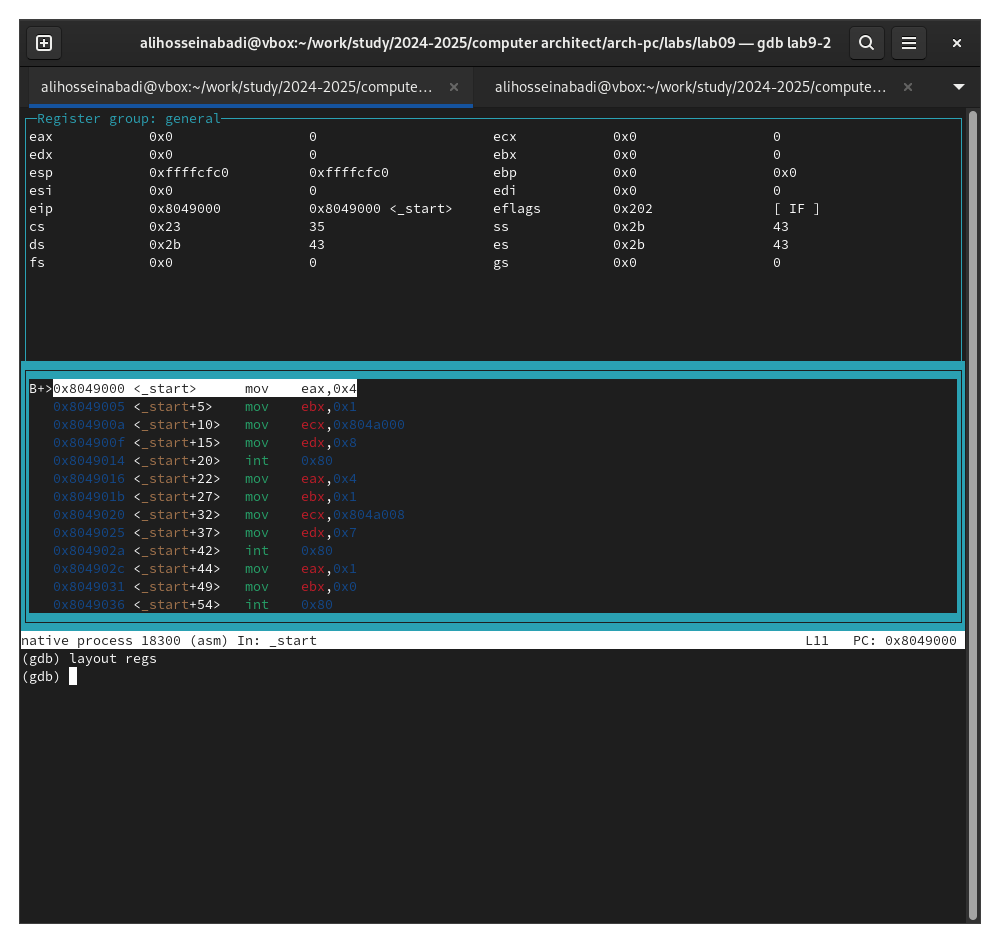


Fig. 8: Pseudo-graphics mode

### 4.1.2 Adding Breakpoints

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

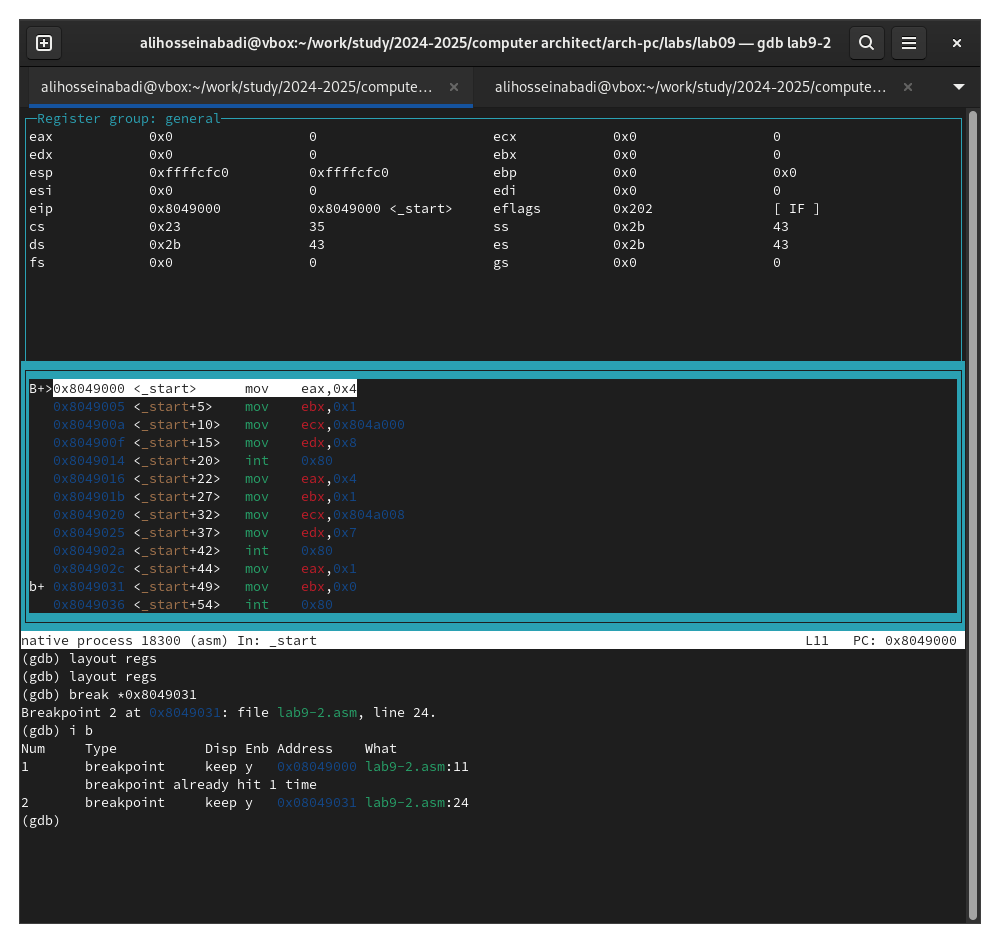


Fig. 9: Breakpoint list

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

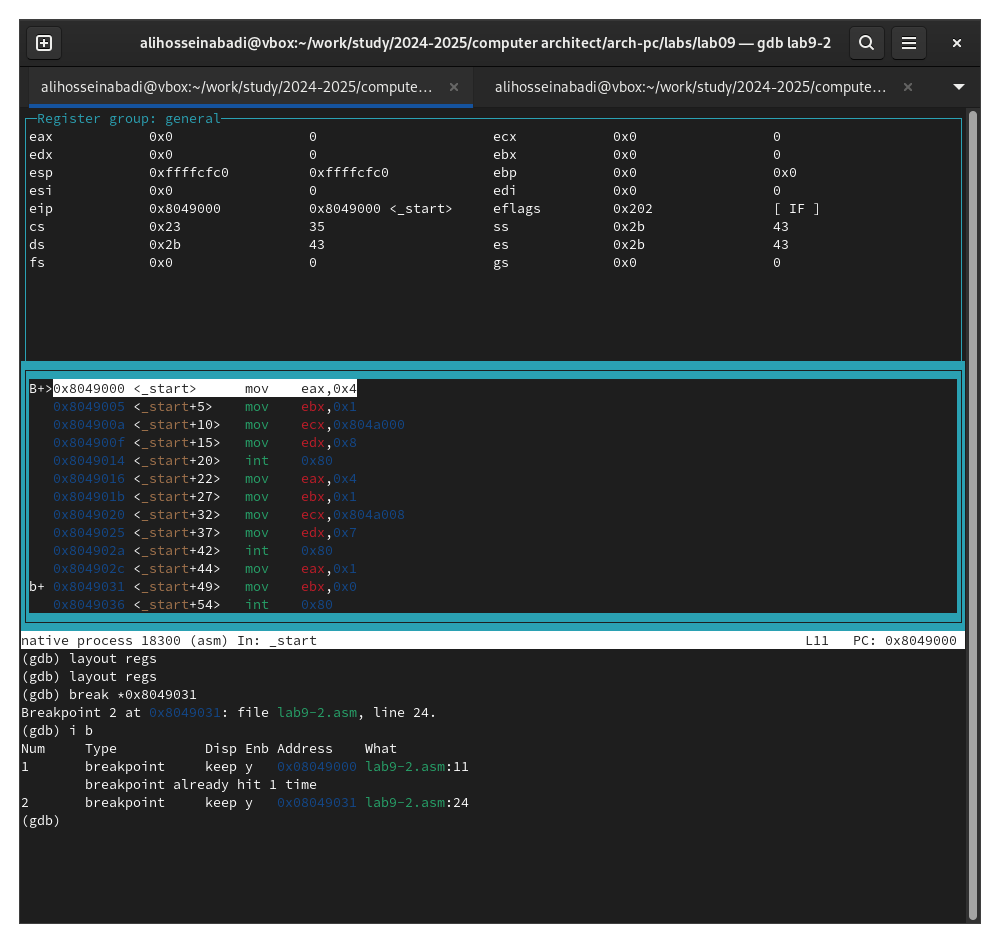


Fig. 10: Adding a second breakpoint

### 4.1.3 Working with Program Data in GDB

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

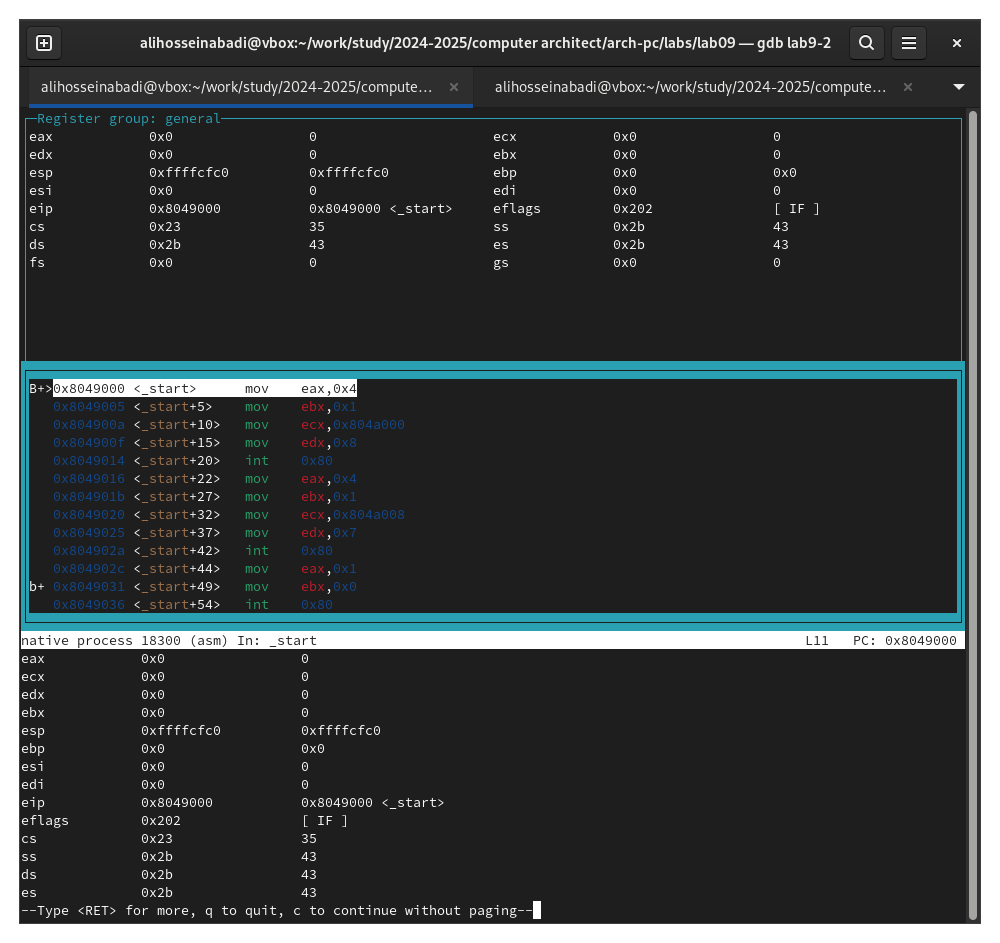


Fig. 11: Viewing the contents of registers

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

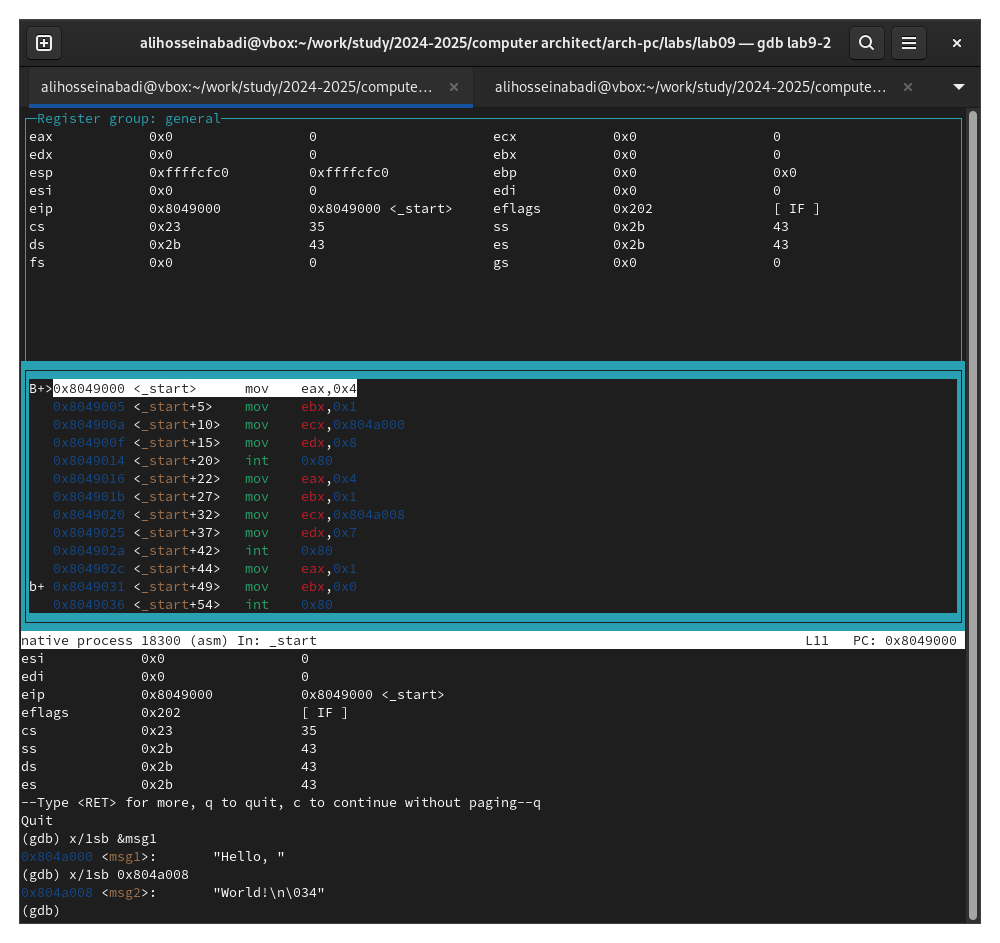


Fig. 12: Viewing the contents of variables in two ways

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

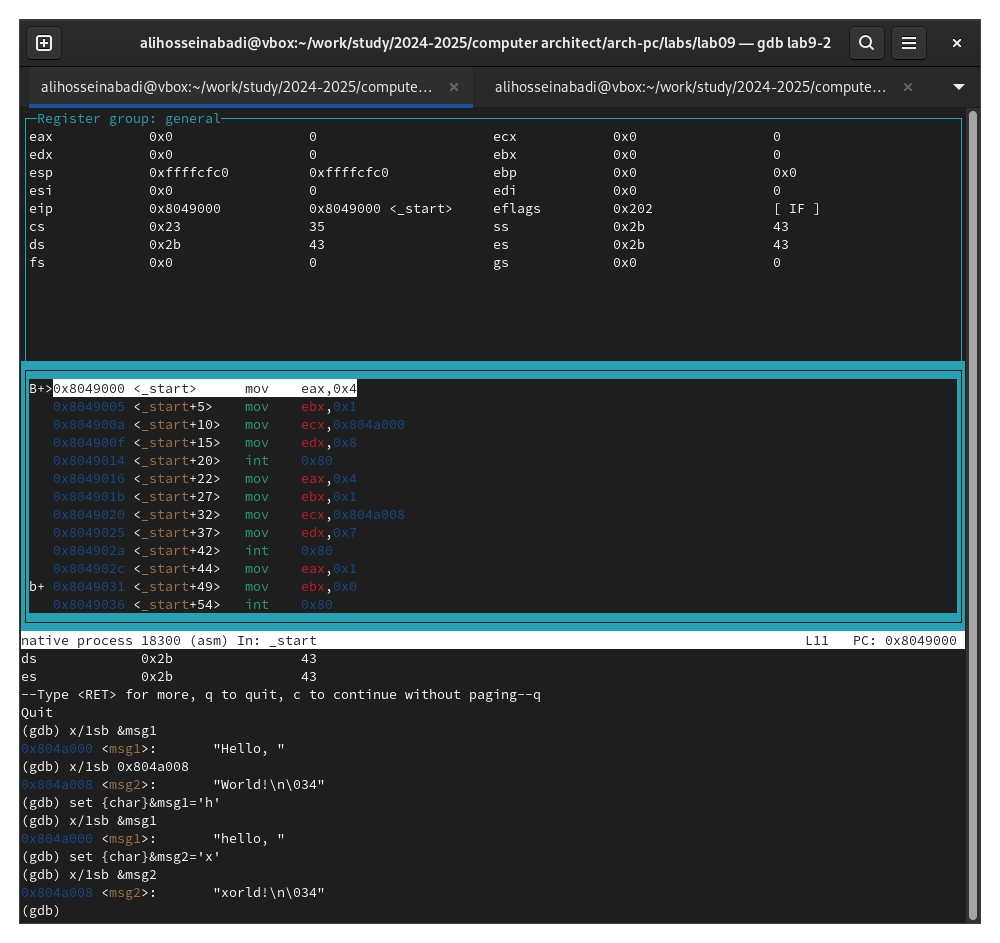


Fig. 13: Changing the contents of variables in two ways

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

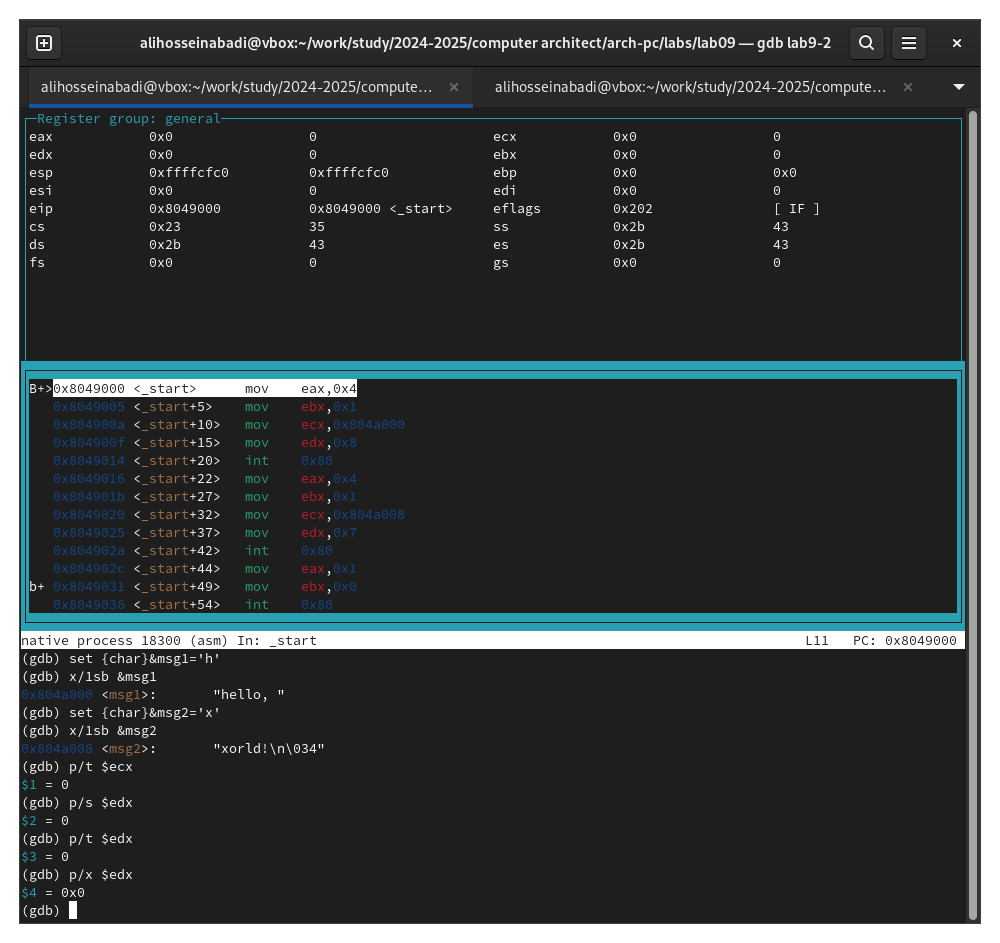


Fig. 14: Viewing the register value in different representations

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

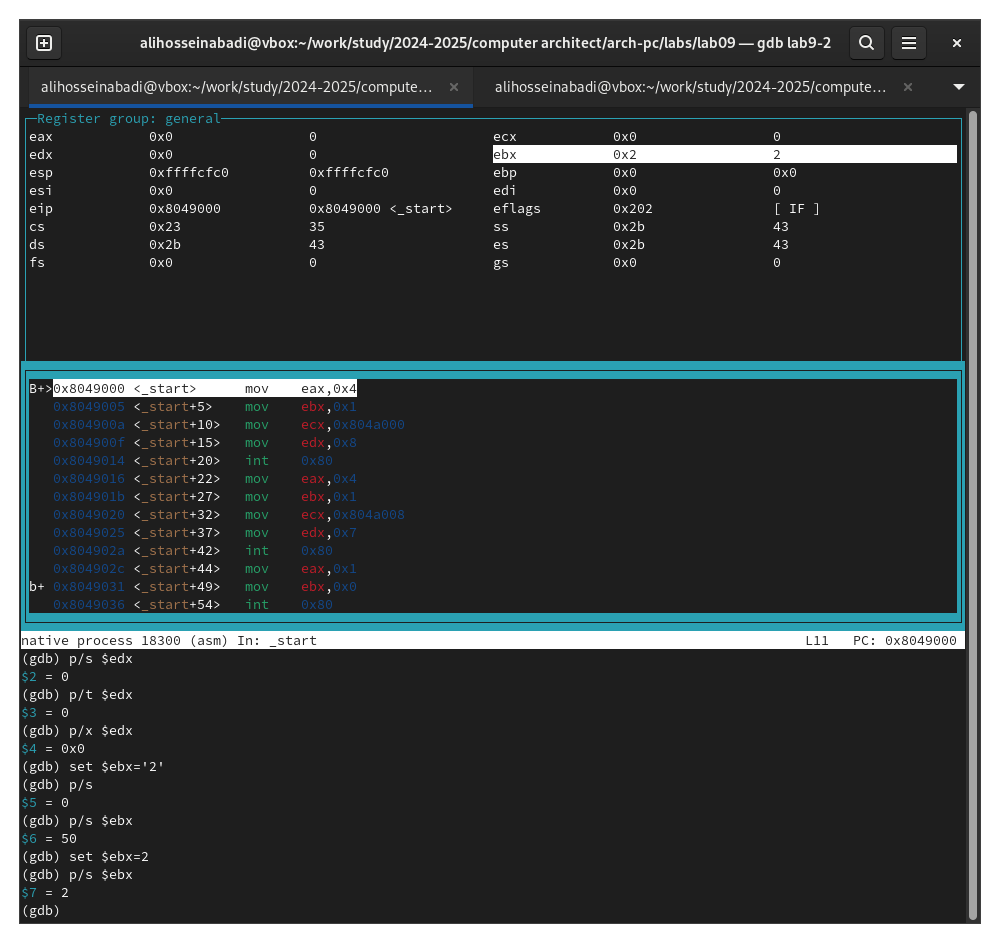


Fig. 15: Examples of using the set command

### 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).

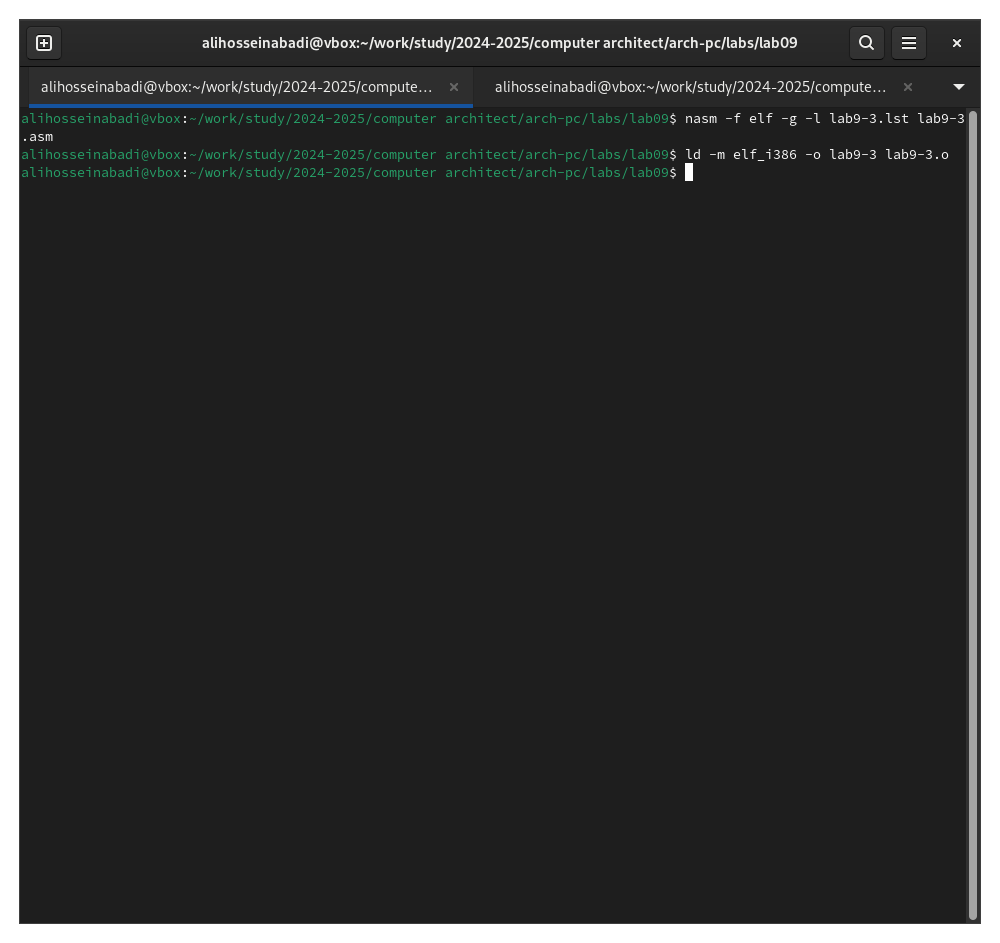


Fig. 16: Preparing a new program

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).

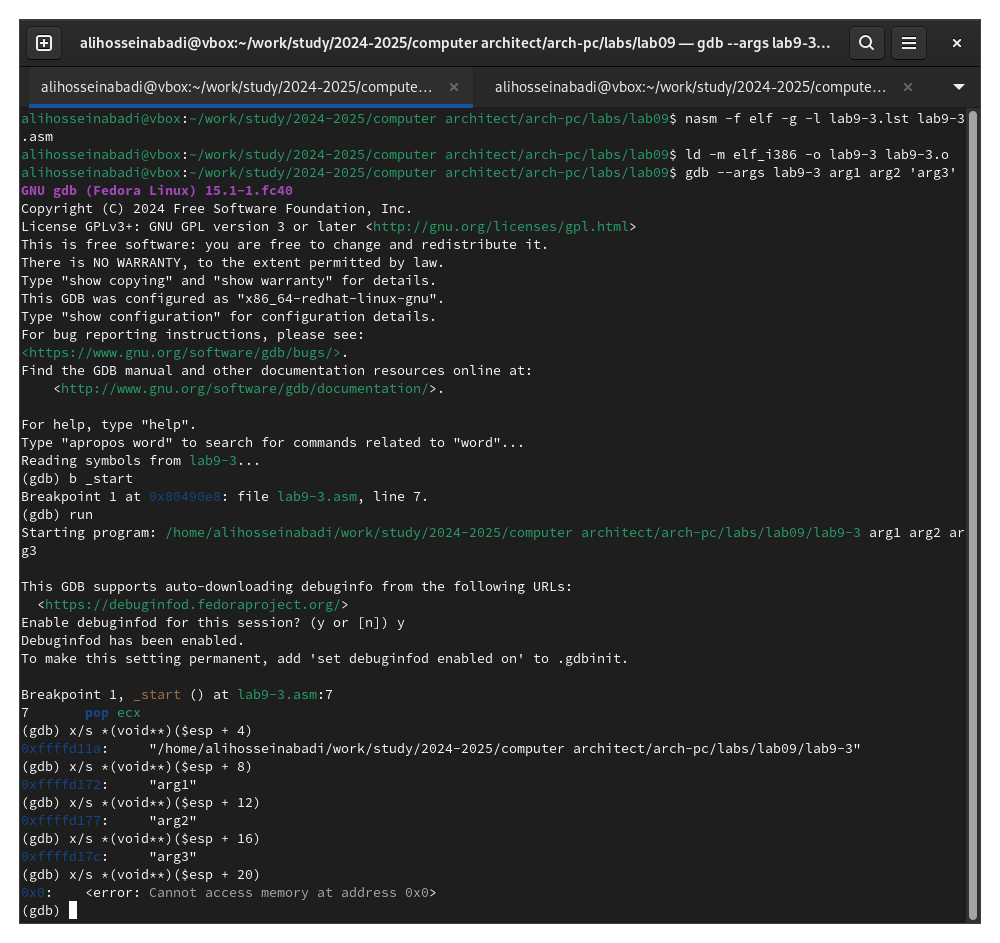


Fig. 17: Checking the stack operation

## 4.2 Independent Work Assignment

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

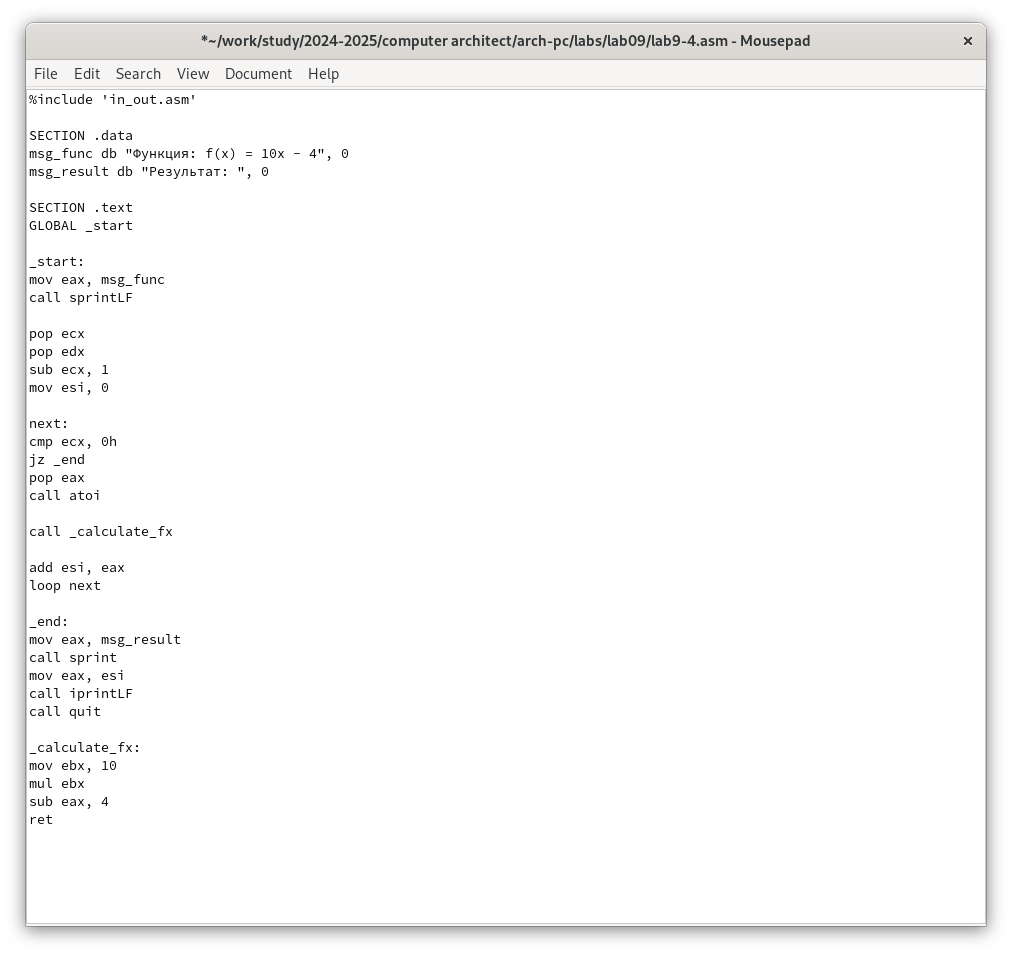


Fig. 18: Modified program of the previous laboratory work

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 sprintLF  
  
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).

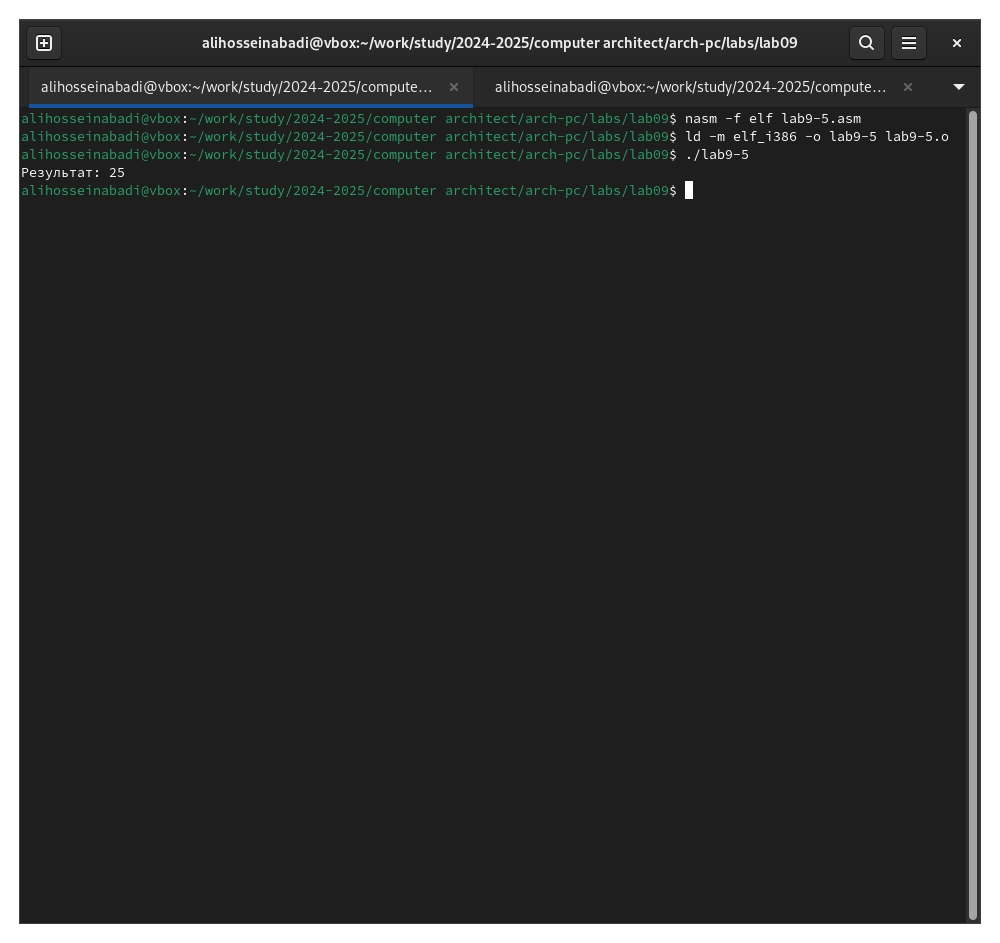


Fig. 19: Verification of corrections in the program

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](https://esystem.rudn.ru/course/view.php?id=112)
2. [Laboratory work No. 9](https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20.pdf)
3. [Programming in NASM Assembly Language Stolyarov A. V.](https://esystem.rudn.ru/pluginfile.php/2088953/mod_resource/content/2/%D0%A1%D1%82%D0%BE%D0%BB%D1%8F%D1%80%D0%BE%D0%B2%20%D0%90.%20%D0%92.%20-%20%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BD%D0%B0%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5%20%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0%20NASM%20%D0%B4%D0%BB%D1%8F%20%D0%9E%D0%A1%20Unix.pdf)