

# **Отчёт по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB**

Лань Цяньин

# **1. Цель работы**

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями

## **2. Порядок выполнения лабораторной работы**

### **2.1. Порядок выполнения лабораторной работы**

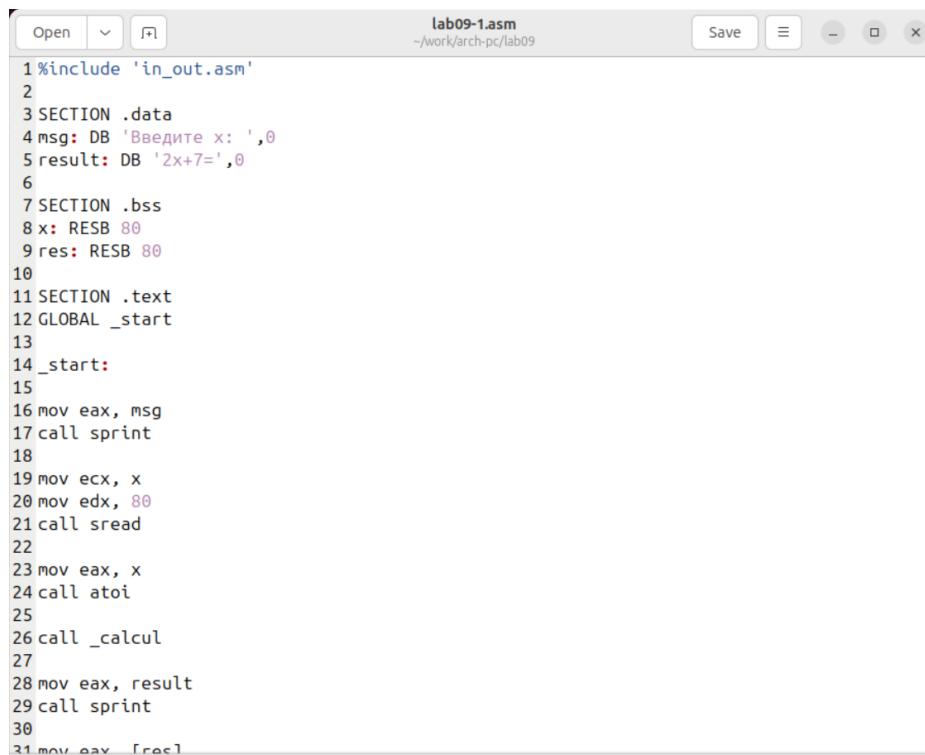
Для выполнения лабораторной работы №9 создан каталог lab09, внутри него создан файл lab09-1.asm в соответствии с инструкциями.(рис. Рисунок 1)

```
clanj1@clanj1:~/work/arch-pc/lab08$ mkdir ~/work/arch-pc/lab09
clanj1@clanj1:~/work/arch-pc/lab08$ cd ~/work/arch-pc/lab09
clanj1@clanj1:~/work/arch-pc/lab09$ touch lab09-1.asm
clanj1@clanj1:~/work/arch-pc/lab09$ gedit lab09-1.asm
```

Рисунок 1: Создание каталога и файла для лабораторной работы №9

Выполнены начальные шаги по подготовке окружения для реализации программы с использованием подпрограмм (процедур).

Программа lab09-1.asm для вычисления выражения  $2x+7$  с использованием подпрограммы была написана в соответствии с Листингом 9.1.(рис. Рисунок 2)



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: DB '2x+7=',0
6
7 SECTION .bss
8 x: RESB 80
9 res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13
14 _start:
15
16 mov eax, msg
17 call sprint
18
19 mov ecx, x
20 mov edx, 80
21 call sread
22
23 mov eax, x
24 call atoi
25
26 call _calcul
27
28 mov eax, result
29 call sprint
30
31 mov eax, [res]
```

Рисунок 2: Исходный код программы lab09-1.asm (часть 1)

Программа lab09-1.asm была успешно откомпилирована и запущена, демонстрируя работу с подпрограммой для вычисления выражения  $2x+7$ .(рис. Рисунок 3)

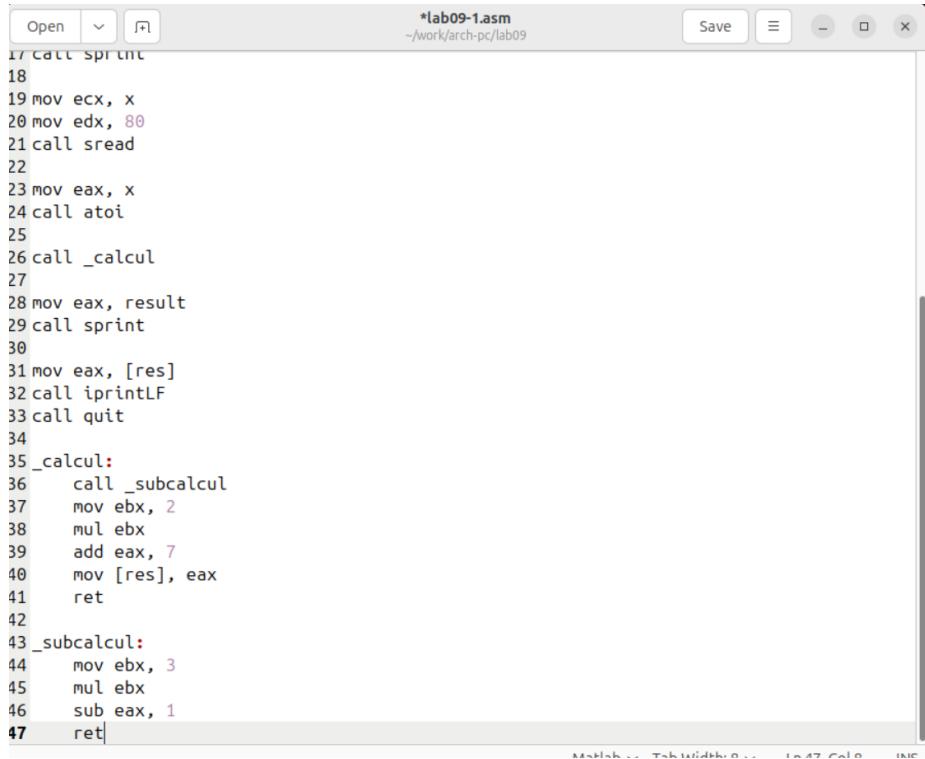
```
clanj1@clanj1:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
clanj1@clanj1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
clanj1@clanj1:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13
```

Рисунок 3: Компиляция, сборка и выполнение программы lab09-1

Программа корректно запросила ввод  $x$ , вызвала подпрограмму `_calcul`, вычислила результат и вывела его. При вводе  $x=3$  результат 13 соответствует выражению  $2 \cdot 3 + 7$ , что подтверждает правильность реализации передачи параметров через регистр `eax`, работы подпрограммы и возврата результата.

Программа lab09-1.asm была модифицирована для вычисления составной функции  $f(g(x))$  путём добавления вложенной подпрограммы `_subcalcul`(рис.

Рисунок 4)



```
*lab09-1.asm
~/work/arch-pc/lab09
Save  -  x
17 call sprint
18
19 mov ecx, x
20 mov edx, 80
21 call sread
22
23 mov eax, x
24 call atoi
25
26 call _calcul
27
28 mov eax, result
29 call sprint
30
31 mov eax, [res]
32 call iprintfLF
33 call quit
34
35 _calcul:
36     call _subcalcul
37     mov ebx, 2
38     mul ebx
39     add eax, 7
40     mov [res], eax
41     ret
42
43 _subcalcul:
44     mov ebx, 3
45     mul ebx
46     sub eax, 1
47     ret|
```

Рисунок 4: Модифицированный код программы lab09-1.asm с вложенной подпрограммой

Модифицированная программа lab09-1.asm с вложенной подпрограммой была успешно откомпилирована и протестирована(рис. Рисунок 5)

```
clanj1@clanj1:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
clanj1@clanj1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
clanj1@clanj1:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=23
```

Рисунок 5: Выполнение модифицированной программы lab09-1

При вводе  $x=3$  программа вывела результат 23. Это соответствует ожидаемому значению составной функции  $f(g(x))$ , где  $g(x)=3x-1$ ,  $f(x)=2x+7$ :  $g(3)=8$ ,  $f(8)=23$ . Результат подтверждает корректную работу механизма вложенных подпрограмм и передачи параметров через регистры

## 2.2. Отладка программам с помощью GDB

Программа lab09-2.asm была успешно откомпилирована с ключом -g для генерации отладочной информации и сгенерирован исполняемый файл, пригодный для отладки в GDB(рис. Рисунок 6)

```
clanj1@clanj1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
clanj1@clanj1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
clanj1@clanj1:~/work/arch-pc/lab09$ gdb lab09-2
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 lab09-2...
(gdb) run
```

Рисунок 6: Компиляция с отладочной информацией и запуск GDB

Программа была запущена в отладчике GDB с использованием точек останова для демонстрации пошагового контроля выполнения(рис. Рисунок 7)

```
(gdb) run
Starting program: /home/clanj1/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 37993) exited normally]
```

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/clanj1/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
```

Рисунок 7: Установка точки останова и выполнение до неё в GDB

В ходе отладки программы lab09-2 была выполнена команда `disassemble _start` для просмотра дизассемблированного кода функции `_start` в синтаксисе AT&T (по умолчанию в GDB)(рис. Рисунок 8)

```
(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.
```

Рисунок 8: Дизассемблированный код `_start` в синтаксисе AT&T

После переключения синтаксиса дизассемблирования на Intel с помощью команды `set disassembly-flavor intel`, код функции `_start` был выведен в соответствующем формате.(рис. Рисунок 9)

```
(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.
```

Рисунок 9: Дизассемблированный код `_start` в синтаксисе Intel

В режиме псевдографики GDB, активированном командой `layout regs`, интерфейс отладчика разделён на три функциональные области (окна)(рис. Рисунок 10)

```

Register group: general
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xfffffcf80    0xfffffcf80
ebp      0x0          0x0

0x804927c    add    BYTE PTR [eax],al
0x804927e    add    BYTE PTR [eax],al
0x8049280    add    BYTE PTR [eax],al
0x8049282    add    BYTE PTR [eax],al
0x8049284    add    BYTE PTR [eax],al
0x8049286    add    BYTE PTR [eax],al

native process 38134 (asm) In: _start
(gdb) layout regs
(gdb)

```

Рисунок 10: Интерфейс GDB в режиме layout regs

### 2.2.1. Добавление точек останова

В графическом режиме GDB (layout regs) была установлена вторая точка останова по адресу инструкции 0x8049031 (строка 24 в lab09-2.asm) — соответствующей команде `mov ebx, 0x0`(рис. Рисунок 11)

```

Register group: general
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xfffffcf80    0xfffffcf80
ebp      0x0          0x0

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 38291 (asm) In: _start
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 24.
(gdb) i b
Num  Type            Disp Enb Address    What
1   breakpoint        keep y  0x08049000 lab09-2.asm:11
                                already hit 1 time
2   breakpoint        keep y  0x08049031 lab09-2.asm:24

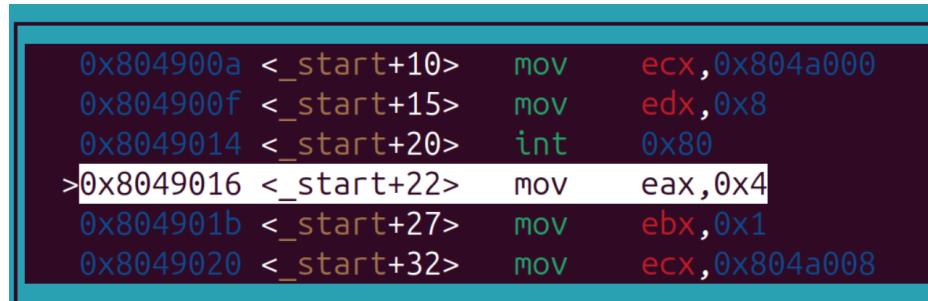
```

Рисунок 11: Установка точки останова по адресу

Команда `info breakpoints` подтвердила наличие двух точек: первая на `_start` (строка 11) — вторая — по адресу 0x8049031 (строка 24). Это позволяет детально анализировать выполнение программы от начала до предпоследней инструкции

## 2.2.2. Работа с данными программы в GDB

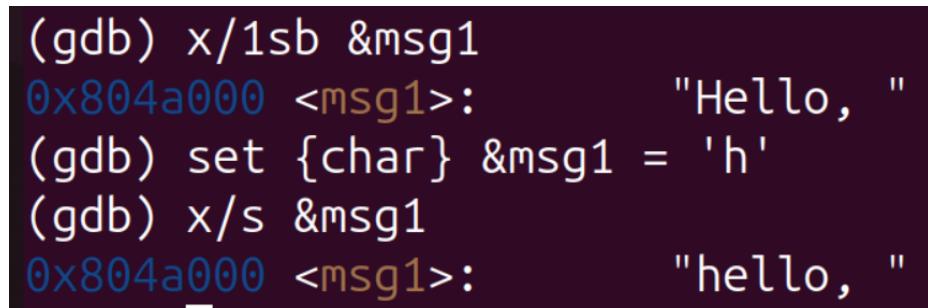
В ходе пошагового выполнения программы с помощью команды `si` 5 были выполнены инструкции(рис. Рисунок 12)



```
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
```

Рисунок 12: Регистры eax, ebx, ecx, edx изменялись для передачи параметров системному вызову

С помощью команды `set {char}msg1='h'` первый символ строки msg1 был изменён с 'H' на 'h'. Повторная команда `x/1sb &msg1` подтвердила изменение: вывелаась строка "hello, "(рис. Рисунок 13)



```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) set {char} &msg1 = 'h'
(gdb) x/s &msg1
0x804a000 <msg1>:      "hello, "
```

Рисунок 13: Просмотр и изменение строк в памяти

На рисунке показано использование команд `set` и `print` в GDB для присвоения регистру ebx значения символа и целого числа с последующей проверкой.(рис. Рисунок 14)

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$1 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$2 = 2
```

Рисунок 14: Присвоение регистру значения символа и числа

Результат наглядно демонстрирует разницу: присваивание `$ebx='2'` сохраняет ASCII-код символа (50), а `$ebx=2` – непосредственно числовое значение.

На рисунке показано использование команды `print` в GDB с различными форматами вывода для отображения одного и того же значения регистра `edx`.(рис. Рисунок 15)

```
(gdb) p/x $edx  
$3 = 0x8  
(gdb) p/t $edx  
$4 = 1000  
(gdb) p/s $edx  
$5 = 8
```

Рисунок 15: Вывод значения регистра edx в разных форматах

Команды p/x, p/t и p/s демонстрируют одно и то же значение (8) в шестнадцатеричном (0x8), двоичном (1000) и десятичном (8) представлении соответственно. Это наглядно показывает, как один и тот же числовой тип данных может быть отображён в разных системах счисления.

### 2.2.3. Обработка аргументов командной строки в GDB

На рисунке показан процесс подготовки и загрузки программы lab09-3 с аргументами командной строки в отладчик GDB.(рис. Рисунок 16)

```

clanj1@clanj1:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
clanj1@clanj1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
clanj1@clanj1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
clanj1@clanj1:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
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 lab09-3...

```

Рисунок 16: Компиляция и запуск программы с аргументами в GDB

На скриншоте последовательно выполняются команды копирования исходного файла, его компиляции (nasm), компоновки (ld) и, наконец, загрузки в GDB с использованием ключа --args и тремя тестовыми аргументами, включая аргумент в кавычках ('аргумент 3'). Успешный вывод информации о версии GDB и сообщения "Reading symbols from lab09-3..."

На рисунке показан процесс установки точки останова и запуска программы в GDB.(рис. Рисунок 17)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/clanj1/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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

Breakpoint 1, _start () at lab09-3.asm:7
7      pop    ecx

```

Рисунок 17: Установка точки останова на \_start и запуск программы

После выполнения команд b \_start и run программа останавливается на первой инструкции pop ecx (строка 7), что позволяет исследовать начальное состояние стека и регистров.

На рисунке показана практическая проверка расположения аргументов командной строки в стеке с помощью команд GDB x/x и x/s. (рис. Рисунок 18)

```
(gdb) x/x $esp
0xfffffcf40:      0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xfffffd112:      "/home/clanj1/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd13a:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd14c:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd15d:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffd15f:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:   <еггог: Cannot access memory at address 0x0>
```

Рисунок 18: Практическая проверка аргументов в стеке

Вершина стека (\$esp) содержит количество аргументов (5). Команды x/s показывают сами аргументы, хранящиеся по адресам в стеке. Шаг в 4 байта между элементами обусловлен размером указателя в 32-битной архитектуре.

## 3. Задание для самостоятельной работы

В данном задании требуется откомпилировать, запустить и проанализировать программу lab09-4.asm, предназначенную для вычисления выражения  $(3+2)^*4+5$ , с целью обнаружения возможных ошибок с помощью отладчика GDB. (рис. Рисунок 1)

The screenshot shows a code editor window titled "lab09-4.asm". The code is written in assembly language using the AT&T syntax. It consists of two sections: B+ and >. The B+ section contains instructions 11 through 15. The > section contains instruction 16. The assembly code is as follows:

```
B+    11 mov ebx, 3
      12 mov eax, 2
      13 add ebx, eax
      14 mov ecx, 4
      15 mul ecx
>     16 add ebx, 5
```

Рисунок 1: Фрагмент кода программы lab09-4.asm в редакторе

```
clanj1@clanj1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-4.lst lab09-4.asm
clanj1@clanj1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
clanj1@clanj1:~/work/arch-pc/lab09$ ./lab09-4
Результат: 25
```

Рисунок 2: Компиляция, сборка и запуск программы lab09-4

Программа была успешно скомпилирована и собрана с помощью `nasm` и `ld`. При запуске она выводит строку «Результат: 25». Хотя результат совпадает с ожидаемым значением выражения  $(3+2)*4+5 = 25$ , согласно условию задания программа содержит логическую ошибку, которую необходимо выявить путём пошагового анализа в GDB. Это требует проверки последовательности инструкций и состояния регистров на каждом этапе вычисления. (рис. Рисунок 2)

## **4 ВЫВОД**

Освоены основы создания подпрограмм на ассемблере и их отладки в GDB, что развивает ключевые практические навыки для программирования низкого уровня.