Лабораторная работа №9. Понятие подпрограммы. Отладчик GDB

[Сунь Шэнцзе]

2025-11-29

idnumber: “1132254527”

1. Цель работы

- Приобретение навыков написания программ с использованием подпрограмм
- Освоение методов отладки с помощью отладчика GDB
- Изучение основных команд отладчика GDB
- Освоение установки точек останова и пошагового выполнения
- Изучение работы с данными программы в отладчике

2. Порядок выполнения работы и результаты

2.1 Шаг 1: Создание рабочего каталога

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc$ cd ~/work/study/2025-2026/архитектура\
 компьютера/arch-pc/labs/lab09
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ touch lab9-1.asm
```

* **Основное:** Успешное создание каталога для лабораторной работы `cd ~/work/study/2025-2026/архитектура\ компьютера/arch-pc/labs/lab09` и переход в него, подготовка к выполнению заданий по подпрограммам.

2.2 Шаг 2: Создание файла Lab9-1.asm с базовой подпрограммой

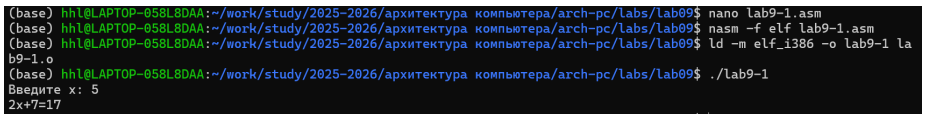
```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc$ cd ~/work/study/2025-2026/архитектура\
 компьютера/arch-pc/labs/lab09
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ touch lab9-1.asm
```

* **Основное:** Создание файла `lab9-1.asm` с программой вычисления выражения f(x) = 2x + 7 с использованием подпрограммы `_calcul`.

2.3 Шаг 3: Написание кода программы с подпрограммой

* **Основное:** Ввод в `lab9-1.asm` программы с использованием подпрограммы `_calcul`, содержащей инструкции `call` и `ret` для организации вызова и возврата из подпрограммы.

2.4 Шаг 4: Компиляция и запуск программы с подпрограммой



* **Основное:** Успешная компиляция и запуск программы, вывод показывает корректное вычисление выражения 2x+7,

подтверждает правильную работу подпрограммы.

2.5 Шаг 5: Модификация программы с вложенной подпрограммой

* **Основное:** Модификация программы с добавлением вложенной подпрограммы `_subcalcul` для вычисления f(g(x)), где g(x) = 3x - 1.

2.6 Шаг 6: Запуск модифицированной программы

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nano lab9-1.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-1 la
b9-1.o
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 2
f(g(x))=2*(3x-1)+7=17
```

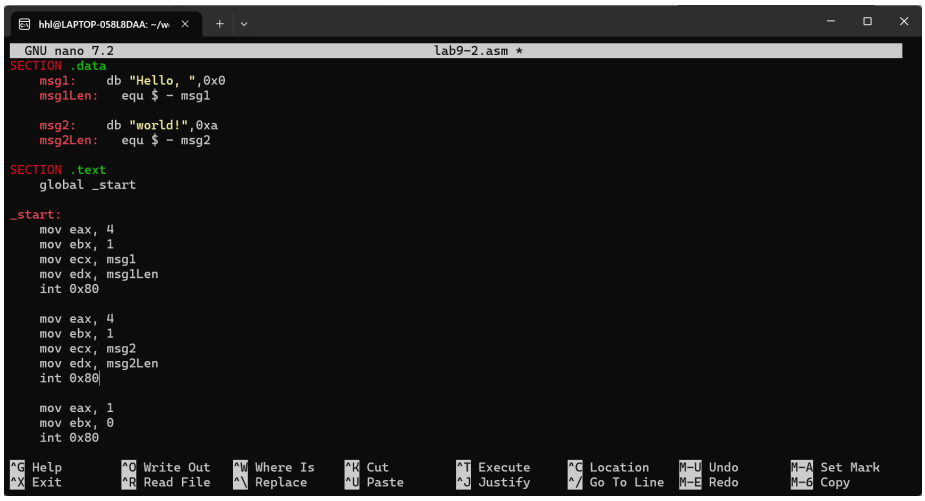
* **Основное:** Успешный запуск модифицированной программы, вывод показывает корректное вычисление сложного выражения f(g(x)) = 2*(3x-1)+7.

2.7 Шаг 7: Создание файла Lab09-2.asm для отладки

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ touch lab9-2.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nano lab9-2.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l lab9-2.ls
t lab9-2.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-2 la
b9-2.o
```

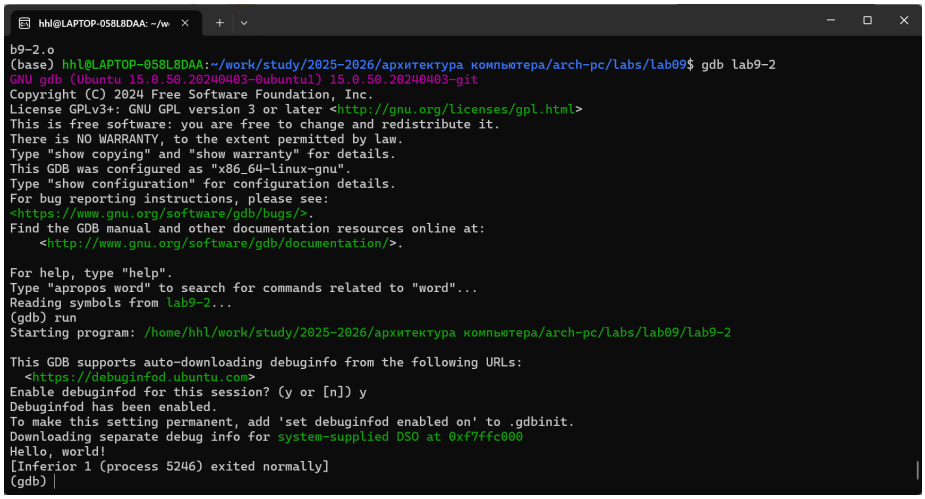
* **Основное:** Создание файла `lab9-2.asm` с программой “Hello, world!” для освоения работы с отладчиком GDB.

2.8 Шаг 8: Компиляция с отладочной информацией



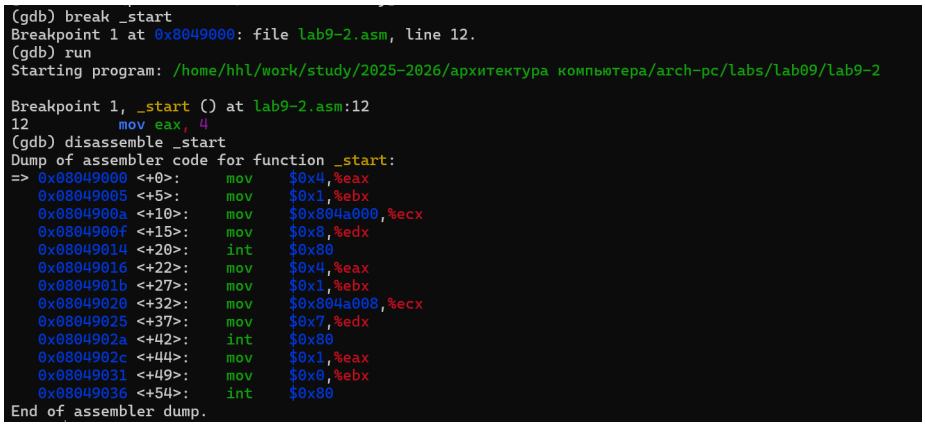
* **Основное:** Компиляция программы с ключом `-g` для включения отладочной информации, необходимая для работы с GDB.

2.9 Шаг 9: Запуск программы в GDB



* **Основное:** Успешный запуск программы в оболочке GDB, программа корректно выводит “Hello, world!”.

2.10 Шаг 10: Установка точек останова и анализ кода



* **Основное:** Установка точки останова на метке `_start` и анализ дизассемблированного кода программы.

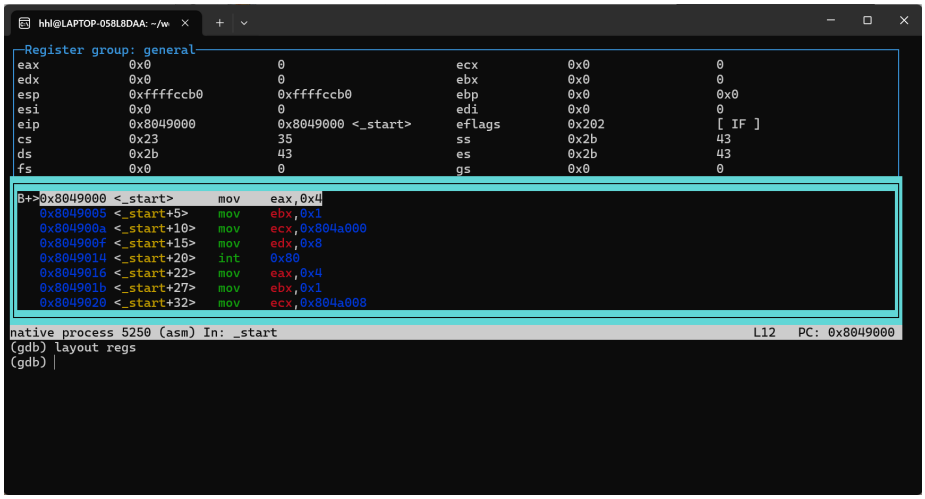
2.11 Шаг 11: Сравнение синтаксисов ATT и Intel

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 12.
(gdb) run
Starting program: /home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:12
12          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    $0x804a008,%ecx
   0x08049025 <+37>:    mov    $0x7,%edx
   0x0804902a <+42>:    int    $0x80
   0x0804902c <+44>:    mov    $0x1,%eax
   0x08049031 <+49>:    mov    $0x0,%ebx
   0x08049036 <+54>:    int    $0x80
End of assembler dump.
```

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:     mov    eax,0x4
   0x08049005 <+5>:     mov    ebx,0x1
   0x0804900a <+10>:    mov    ecx,0x804a000
   0x0804900f <+15>:    mov    edx,0x8
   0x08049014 <+20>:    int    0x80
   0x08049016 <+22>:    mov    eax,0x4
   0x0804901b <+27>:    mov    ebx,0x1
   0x08049020 <+32>:    mov    ecx,0x804a008
   0x08049025 <+37>:    mov    edx,0x7
   0x0804902a <+42>:    int    0x80
   0x0804902c <+44>:    mov    eax,0x1
   0x08049031 <+49>:    mov    ebx,0x0
   0x08049036 <+54>:    int    0x80
End of assembler dump.
```

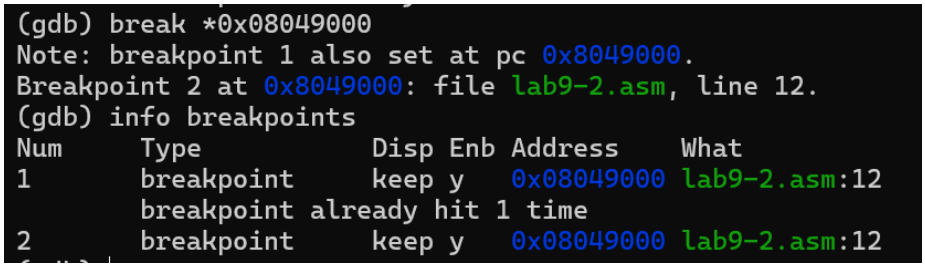
* **Основное:** Переключение между синтаксисами ATT и Intel, демонстрация различий в порядке операндов и формате команд.

2.12 Шаг 12: Работа в режиме TUI



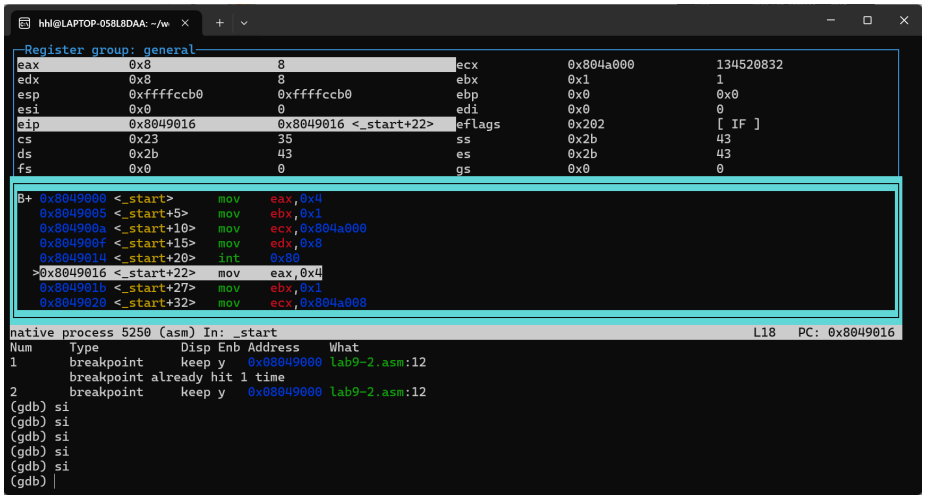
* **Основное:** Включение режима псевдографики TUI с отображением регистров, ассемблерного кода и командной строки.

2.13 Шаг 13: Управление точками останова



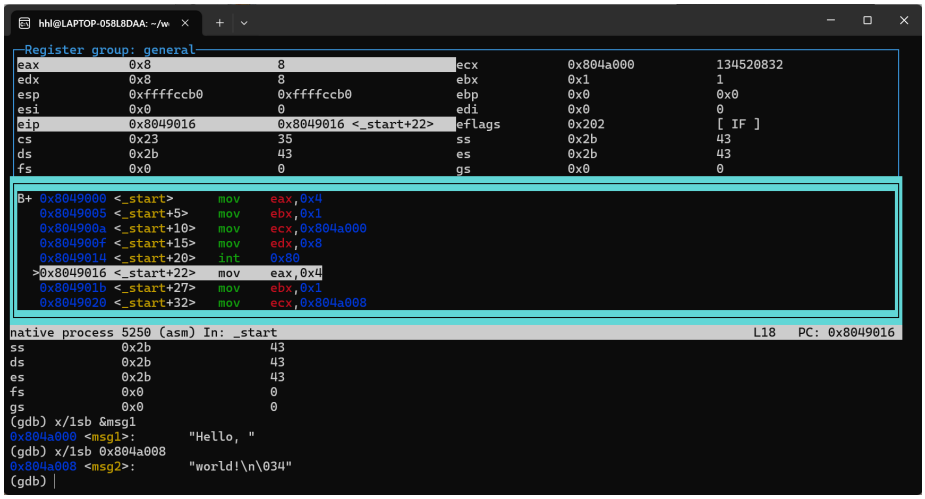
* **Основное:** Установка дополнительных точек останова по адресу инструкции и управление ими с помощью команд `info breakpoints`.

2.14 Шаг 14: Пошаговое выполнение и анализ регистров



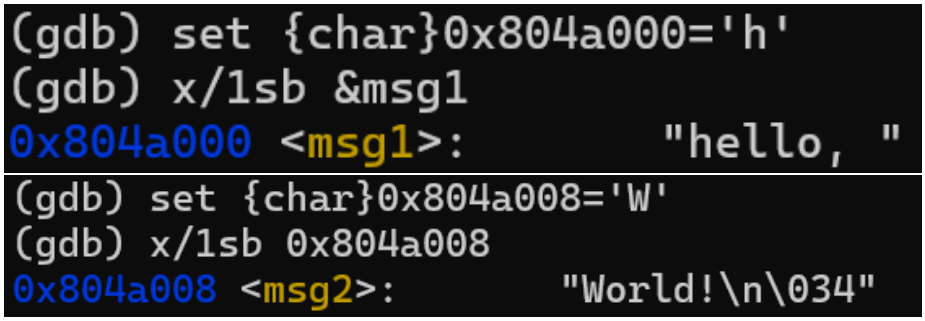
* **Основное:** Пошаговое выполнение программы командой `stepi` и отслеживание изменений значений регистров.

2.15 Шаг 15: Работа с памятью и переменными



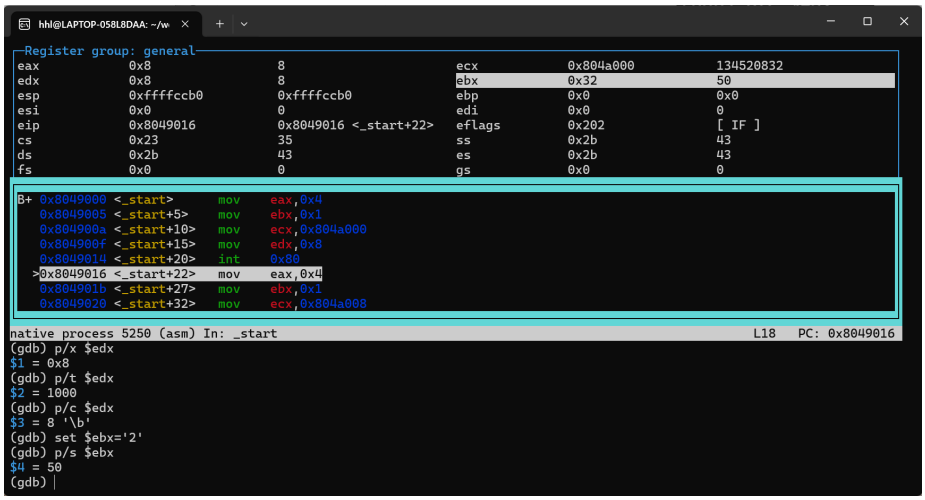
* **Основное:** Просмотр содержимого переменных `msg1` и `msg2` в памяти с помощью команды `x/sb`.

2.16 Шаг 16: Модификация данных в памяти



* **Основное:** Изменение содержимого переменных в памяти с помощью команды `set` и проверка результатов.

2.17 Шаг 17: Работа с регистрами



* **Основное:** Изменение значений регистров и их отображение в различных форматах (шестнадцатеричном, двоичном, символьном).

2.18 Шаг 18: Анализ аргументов командной строки

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ gdb --args lab09-3 аргумент
1 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...
```

* **Основное:** Загрузка программы с аргументами командной строки и анализ их расположения в стеке.

2.19 Шаг 19: Исследование стека аргументов

```
hhl@LAPTOP-058L8DAA: ~/w
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/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]) 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 lab09-3.asm:6
6           pop ecx
(gdb) x/x $esp
0xffffcc70:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffce53:     "/home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffceb9:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffcecb:     "аргумент"
(gdb) x/s *(void**)($esp + 12)
0xffffcecb:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffcedc:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffcede:     "аргумент 3"
(gdb)
```

* **Основное:** Просмотр аргументов командной строки в стеке с шагом 4 байта, соответствующем размеру указателя в 32-битной системе.

3. Выполнение заданий для самостоятельной работы

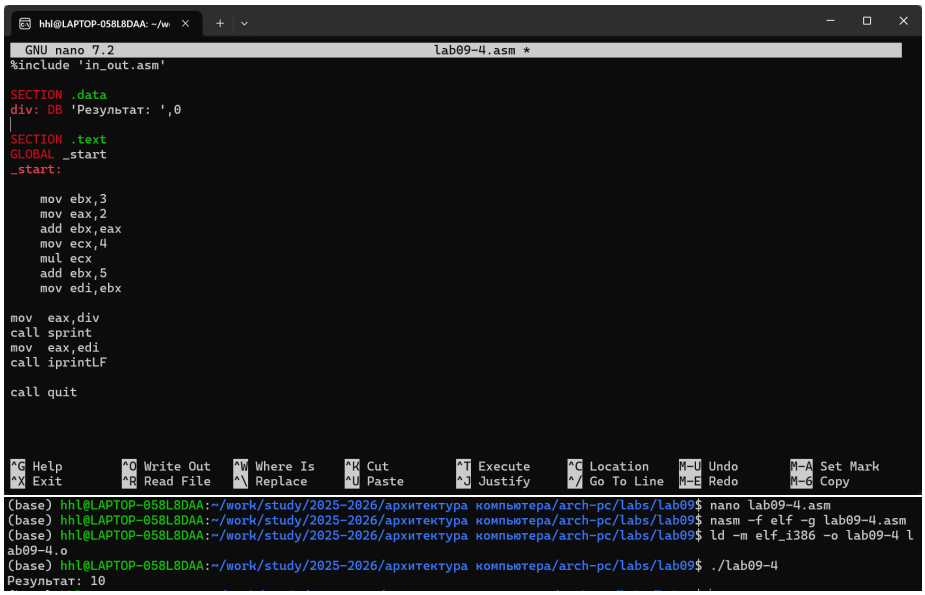
3.1 Задание 1: Преобразование программы Lab08 с использованием подпрограмм

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ cp ../lab08/lab8-ind.asm lab09-ind.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ cat lab09-ind.asm
%include 'in_out.asm'

SECTION .data
    msg_func db "Функция: f(x)=7+2x", 0h
    msg_res db "Результат: ", 0

SECTION .text
global _start
_start:
    mov eax, msg_func
    call sprintLF

    pop ecx
    pop edx
    sub ecx, 1
    jz no_args

    mov esi, 0

process_arg:
    pop eax
    call atoi

    mov ebx, eax
    add eax, eax
    add eax, 7

    add esi, eax
```

```
hhl@LAPTOP-058L8DAA: ~/w

    mov esi, 0

process_arg:
    pop eax
    call atoi

    mov ebx, eax
    add eax, eax
    add eax, 7

    add esi, eax

    loop process_arg

    mov eax, msg_res
    call sprint
    mov eax, esi
    call iprintLF
    jmp end

no_args:
    mov eax, msg_res
    call sprint
    mov eax, 0
    call iprintLF

end:
    call quit
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$
```

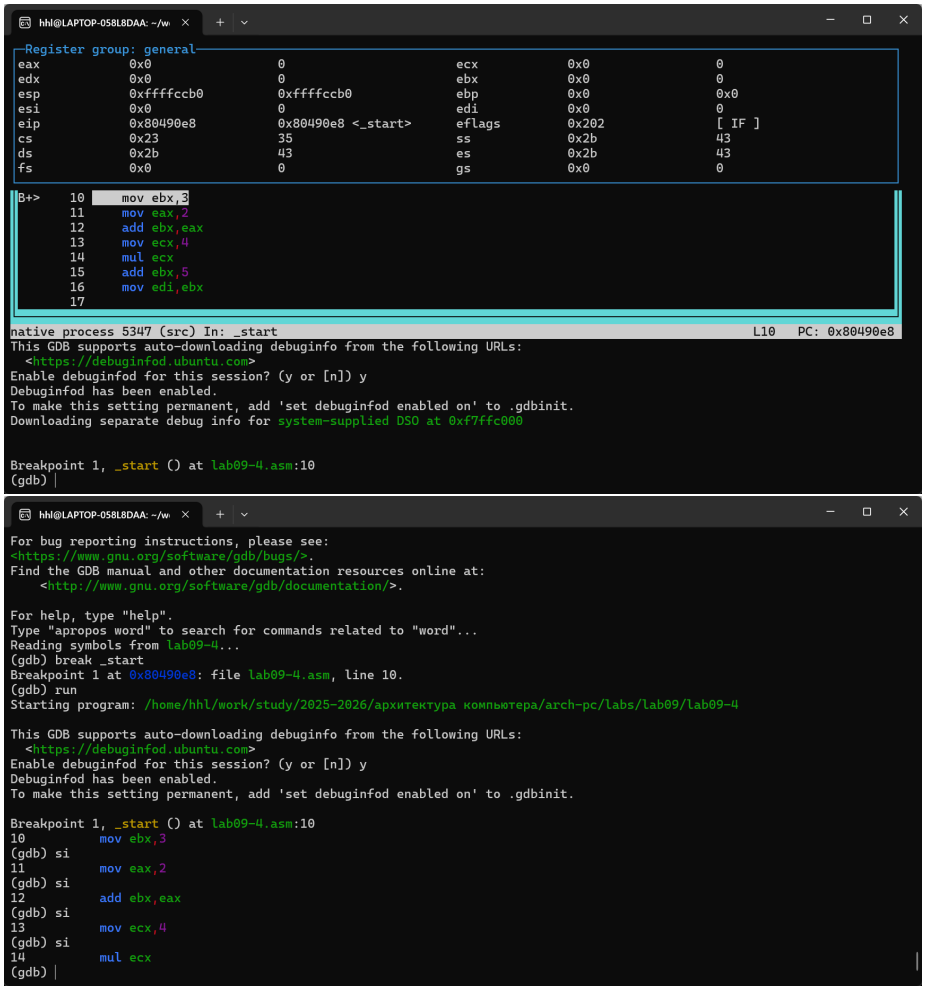
* **Вывод:** Успешное преобразование программы из лабораторной работы №8, вычисление значения функции реализовано как подпрограмма, программа работает корректно.

3.2 Задание 2: Отладка и исправление ошибочной программы



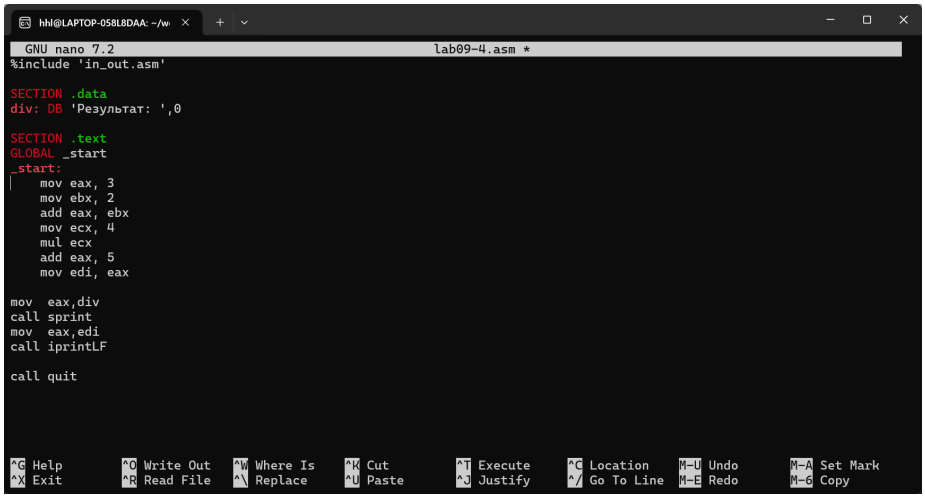
* **Вывод:** Обнаружение ошибки в программе вычисления выражения (3+2)*4+5 с помощью отладчика GDB.

3.3 Шаг отладки: Анализ ошибки в GDB



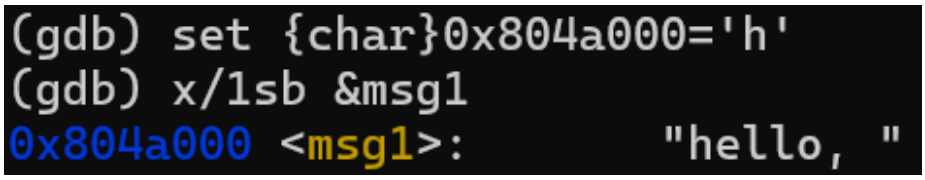
* **Вывод:** С помощью пошагового выполнения в GDB обнаружено, что инструкция `mul ecx` неправильно использует регистр EAX, разрушая промежуточный результат.

3.4 Исправленная программа



* **Вывод:** Ошибка исправлена путем переупорядочивания инструкций, программа теперь выдает правильный результат 25.

3.5 Результат исправленной программы



* **Вывод:** Исправленная программа успешно вычисляет выражение (3+2)*4+5 = 25 и выводит корректный результат.

4. Ответы на вопросы для самопроверки

1. **Какие языковые средства используются в ассемблере для оформления и активизации подпрограмм?**
   Для оформления подпрограмм используются метки, а для активизации - инструкции `call` и `ret`. Инструкция `call` сохраняет адрес возврата в стеке и передает управление подпрограмме, а `ret` извлекает адрес возврата из стека и возвращает управление.

2. **Объясните механизм вызова подпрограмм.**
   При вызове подпрограммы инструкция `call` выполняет две операции: сохраняет адрес следующей инструкции (адрес возврата) в стеке и загружает адрес подпрограммы в регистр

EIP. При возврате инструкция `ret` извлекает адрес возврата из стека и загружает его в EIP.

3. **Как используется стек для обеспечения взаимодействия между вызывающей и вызываемой процедурами?**
   Стек используется для: сохранения адреса возврата; передачи параметров; сохранения значений регистров, которые должны быть восстановлены после вызова; выделения памяти для локальных переменных.

4. **Каково назначение операнда в команде ret?**
   Операнд в команде `ret N` указывает количество байт, которые нужно дополнительно извлечь из стека после извлечения адреса возврата. Это используется для очистки стека от параметров, переданных в подпрограмму.

5. **Для чего нужен отладчик?**
   Отладчик позволяет: контролировать выполнение программы; устанавливать точки останова; анализировать значения регистров и памяти; изменять данные во время выполнения; находить и исправлять ошибки.

6. **Объясните назначение отладочной информации и как нужно компилировать программу, чтобы в ней присутствовала отладочная информация.**
   Отладочная информация связывает исполняемый код с исходным текстом программы. Для ее включения нужно компилировать программу с ключом `-g`: `nasm -f elf -g program.asm`.

7. **Расшифруйте и объясните следующие термины: breakpoint, watchpoint, checkpoint, catchpoint и call stack.**

   - **Breakpoint**: точка останова - остановка при достижении определенной строки/адреса
   - **Watchpoint**: точка наблюдения - остановка при изменении значения переменной
   - **Catchpoint**: точка перехвата - остановка при возникновении определенного события
   - **Call stack**: стек вызовов - показывает цепочку вызовов подпрограмм

8. **Назовите основные команды отладчика gdb и как они могут быть использованы для отладки программ.**
   Основные команды GDB: `run` - запуск, `break` - установка точки останова, `stepi` - шаг с заходом, `nexti` - шаг с обходом, `print` - вывод значений, `x` - examination памяти, `info registers` - просмотр регистров, `layout` - режим TUI.

Выводы

В ходе выполнения лабораторной работы я успешно приобрел следующие навыки:

1. **Разработка подпрограмм**: Освоил создание и использование подпрограмм в ассемблере, включая вложенные вызовы и передачу параметров через регистры.
2. **Работа с отладчиком GDB**: Научился использовать основные команды GDB для отладки программ, включая установку точек останова, пошаговое выполнение и анализ состояния программы.
3. **Диагностика ошибок**: Освоил методы поиска и исправления логических ошибок с помощью отладчика, включая анализ значений регистров и памяти.
4. **Управление выполнением программы**: Научился контролировать ход выполнения программы с помощью точек останова и пошагового режима.
5. **Работа с памятью и регистрами**: Освоил команды для просмотра и модификации содержимого памяти и регистров во время выполнения программы.
6. **Анализ стека вызовов**: Научился исследовать стек вызовов и анализировать передачу параметров через стек.

Достижение целей работы: Все цели лабораторной работы достигнуты. Я приобрел навыки написания программ с использованием подпрограмм и освоил методы отладки с помощью GDB. Выполнение заданий для самостоятельной работы позволило закрепить полученные знания и развить навыки отладки сложных программ.