

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) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc$ cd ~/work/study/2025-2026/архитектура\ компьютера/arch-pc/labs/lab09  
(base) hh1@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) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc$ cd ~/work/study/2025-2026/архитектура\ компьютера/arch-pc/labs/lab09  
(base) hh1@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

The screenshot shows two terminal windows side-by-side, both titled "GNU nano 7.2" and displaying the same assembly code for "lab9-1.asm". The code defines a program that prompts for input, calculates $2x+7$, and prints the result.

```
GNU nano 7.2                                lab9-1.asm *
%include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2x+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:
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret
|
```

The bottom window shows the final state of the assembly code after modifications:

```
GNU nano 7.2                                lab9-1.asm *
    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:
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret
|
```

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

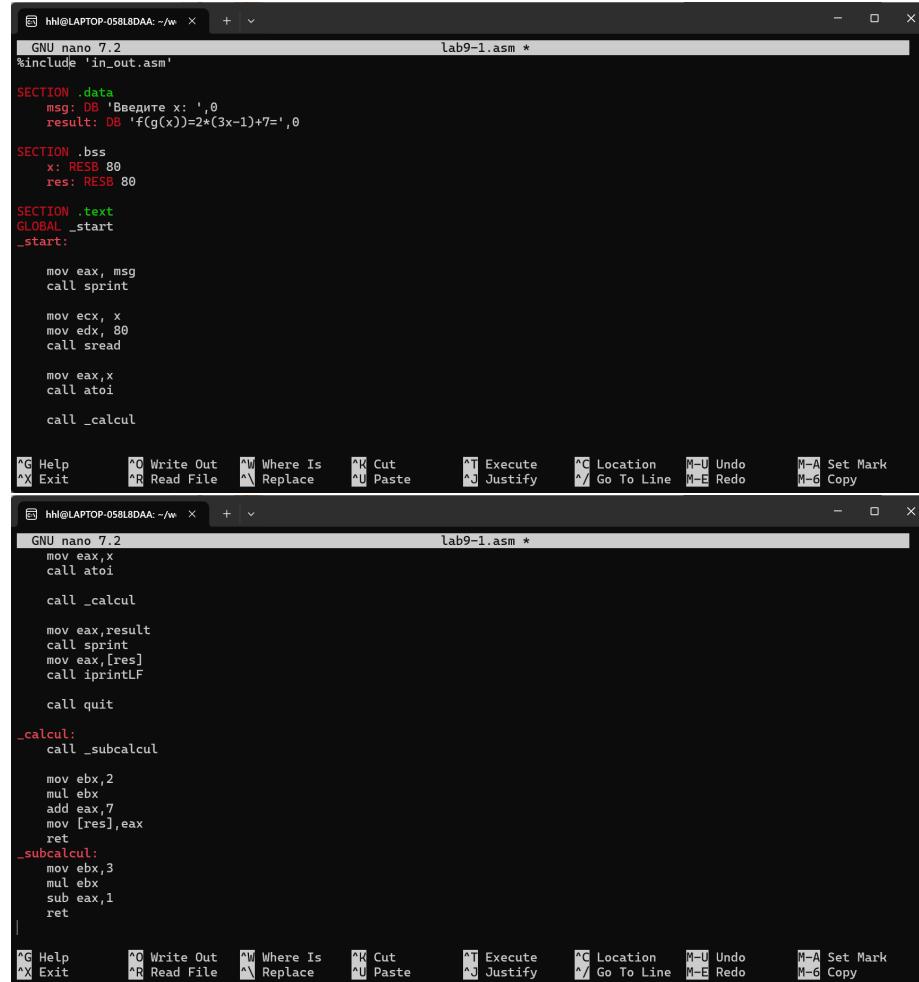
2.4 Шаг 4: Компиляция и запуск программы с подпрограммой

```
(base) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nano lab9-1.asm
(base) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab9-1.asm
(base) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
(base) hh1@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ ./lab9-1
Введите x: 5
2x+7=17
```

* **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



The image shows two terminal windows side-by-side, both displaying assembly code for a file named `lab9-1.asm`. The top window contains the initial code, and the bottom window shows the modified code after adding the `_subcalcul` subroutine.

Top Window (Initial Code):

```
GNU nano 7.2                               lab9-1.asm *
%include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB 'f(g(x))=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


```

Bottom Window (Modified Code):

```
GNU nano 7.2                               lab9-1.asm *
    mov eax,x
    call atoi

    call _calcul

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

    call quit

._calcul:
    call _subcalcul

    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret
._subcalcul:
    mov ebx,3
    mul ebx
    sub eax,1
    ret
|
```

* **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 lab9-1.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
```

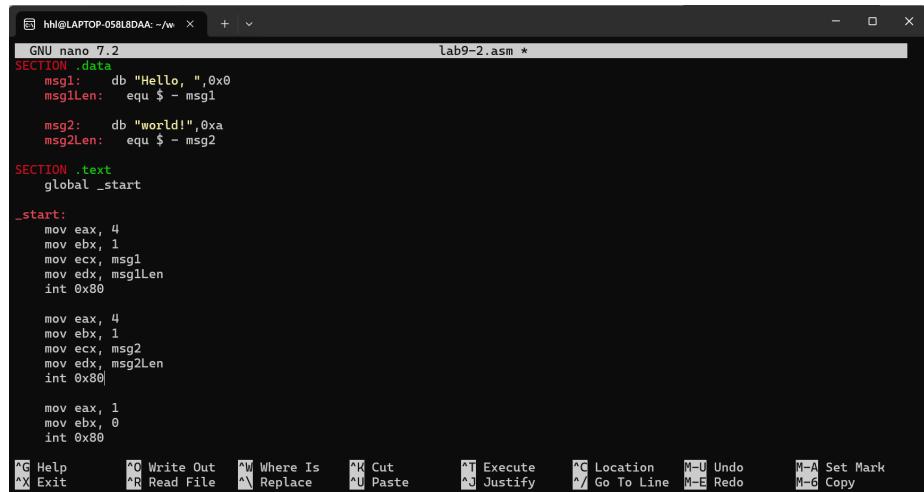
* **Основное:** Успешный запуск модифицированной программы, вывод показывает корректное вычисление сложного выражения $f(g(x)) = 2*(3x-1)+7$.

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 lab9-2.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



The screenshot shows the assembly code for a "Hello, world!" program. The code is written in AT&M assembly language. It defines two strings in the .data section: "Hello, " and "world!". The .text section contains the _start label, which initializes EAX to 4 (for sys_write), EBX to 1 (for file descriptor 1), ECX to msg1 (the address of the first string), and EDX to msg1Len (the length of the first string). It then calls the interrupt 0x80. The same sequence is repeated for the second string. Finally, it exits with a status of 0. The assembly code is as follows:

```
GNU nano 7.2                                         lab9-2.asm *
SECTION .data
    msg1:    db "Hello, ",0x0
    msg1Len:   equ $ - msg1

    msg2:    db "world!",0xa
    msg2Len:   equ $ - msg2

SECTION .text
    global _start

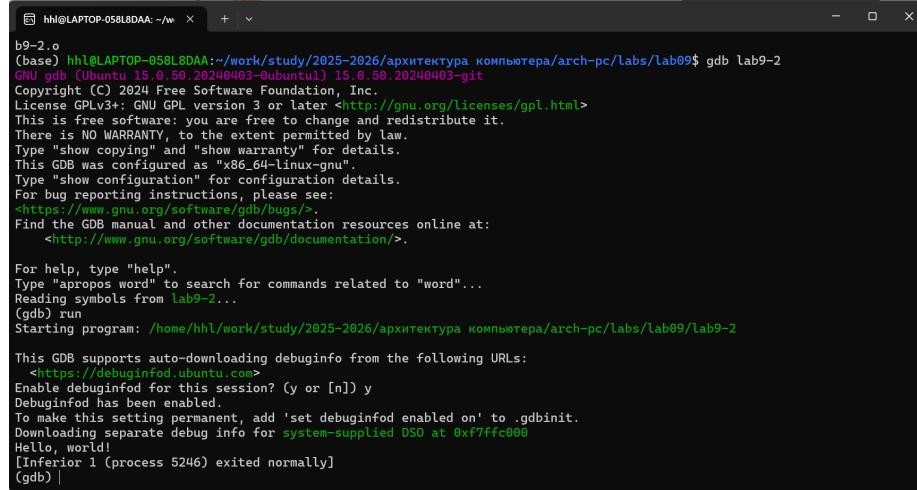
_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msg1Len
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, msg2
    mov edx, msg2Len
    int 0x80

    mov eax, 1
    mov ebx, 0
    int 0x80
```

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

2.9 Шаг 9: Запуск программы в GDB



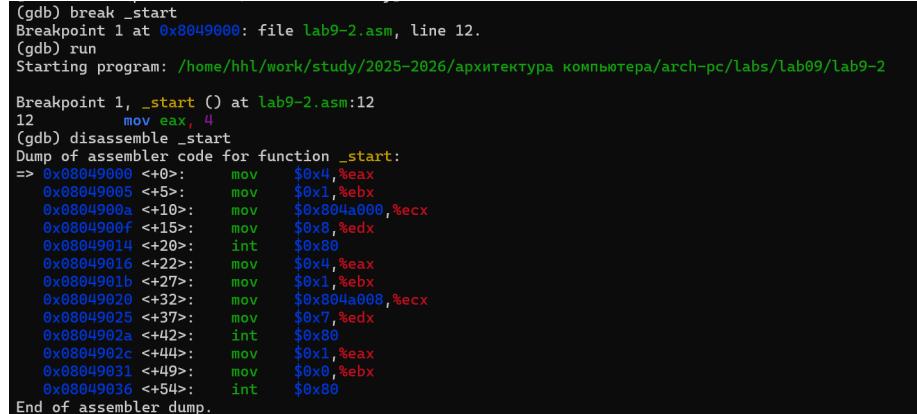
```
b9-2.o
(base) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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/hhl/work/study/2025-2026/архитектура компьютера/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 5246) exited normally]
(gdb) 
```

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

2.10 Шаг 10: Установка точек останова и анализ кода



```
(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    $0x4,%eax
(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    $0x8,%ebx
  0x08049036 <+54>:  int    $0x80
End of assembler dump.
```

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

2.11 Шаг 11: Сравнение синтаксисов ATT и Intel

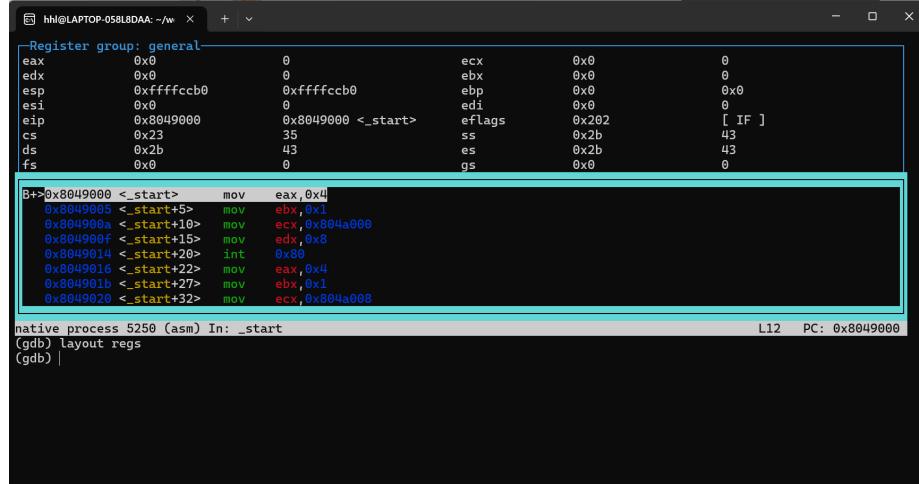
```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 12.
(gdb) run
Starting program: /home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09/lab9-2

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 operandов и формате команд.

2.12 Шаг 12: Работа в режиме TUI



The screenshot shows the GDB TUI interface. At the top, there's a register dump titled "Register group: general". Below it, the assembly code for the program starts at address 0x8049000. The assembly code includes instructions like mov eax, 0x4, mov ebx, 0x1, and int 0x80. The command line at the bottom shows the user entering commands like "break *0x08049000" and "info breakpoints". The status bar at the bottom right indicates the current line is L12 and the PC is at 0x8049000.

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

2.13 Шаг 13: Управление точками останова

```
(gdb) break *0x08049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab9-2.asm, line 12.
(gdb) info breakpoints
Num      Type            Disp Enb Address     What
1        breakpoint      keep y  0x08049000  lab9-2.asm:12
                  breakpoint already hit 1 time
2        breakpoint      keep y  0x08049000  lab9-2.asm:12
```

* **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

The screenshot shows the Immunity Debugger interface. The assembly pane displays the following code:

```
B+ 0x0049000 <_start>    mov    eax,0x4
0x0049005 <_start+5>    mov    ebx,0x1
0x004900a <_start+10>   mov    ecx,0x804a000
0x004900f <_start+15>   mov    edx,0x8
0x0049014 <_start+20>   int    0x80
>0x0049016 <_start+22>  mov    eax,0x4
0x004901b <_start+27>   mov    ebx,0x1
0x0049020 <_start+32>   mov    ecx,0x804a008
```

The registers pane shows the following state:

Register	Value	Description
eax	0x8	8
edx	0x8	8
esp	0xfffffccb0	0xfffffccb0
esi	0x0	0
eax	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ds	0x2b	43
fs	0x0	0
ss	0x2b	43
es	0x2b	43
gs	0x0	0

The status bar at the bottom indicates: native process 5250 (asm) In: _start L18 PC: 0x0049016

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

2.15 Шаг 15: Работа с памятью и переменными

The screenshot shows the Immunity Debugger interface. The assembly pane displays the same assembly code as in Step 14. The memory dump pane shows the following data:

Address	Value	Content
0x804a008	43	x1b %msg1
0x804a008 <msg1>	43	"Hello, "
0x804a008	43	x1b 0x804a008
0x804a008 <msg2>	0	"world!\n\034"

The status bar at the bottom indicates: native process 5250 (asm) In: _start L18 PC: 0x0049016

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

2.16 Шаг 16: Модификация данных в памяти

```
(gdb) set {char}0x804a000='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a008='W'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "World!\n\034"
```

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

2.17 Шаг 17: Работа с регистрами

The screenshot shows the GDB interface with two windows. The top window displays a register dump for the 'general' group. The bottom window shows assembly code for the '_start' function, with several instructions highlighted in blue. The bottom window also shows some native process commands and variable assignments.

Register	Value	Description
eax	0x8	8
edx	0x8	8
esp	0xfffffcbb0	0xfffffcbb0
esi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
cs	0x23	35
ds	0x2b	43
fs	0x0	0
ecx	0x804a000	134520832
ebx	0x32	50
ebp	0x0	0x0
edi	0x0	0
eflags	0x202	[IF]
ss	0x2b	43
es	0x2b	43
gs	0x0	0

```
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

native process 5250 (asm) In: _start
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 `b'
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) |
```

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

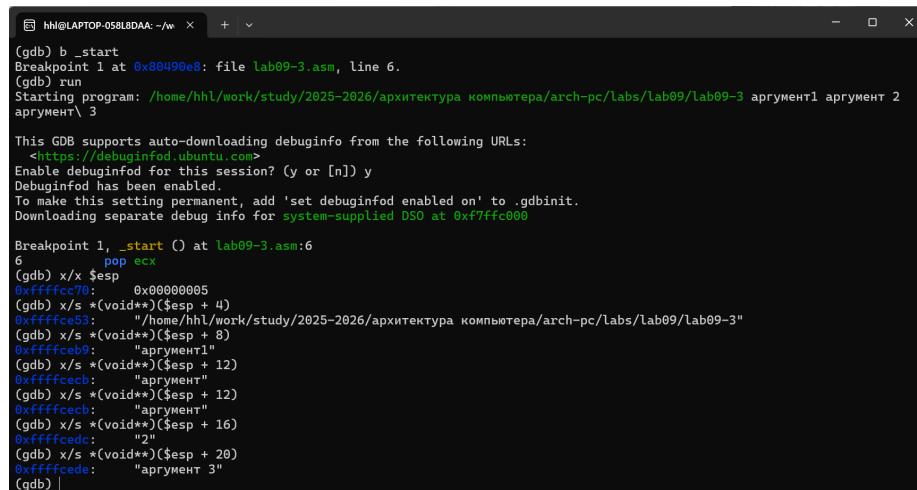
2.18 Шаг 18: Анализ аргументов командной строки

```
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
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



```
hh@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 аргумент2 'аргумент3'

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 $esp
0xfffffc70:   0x00000005
(gdb) x/s *(void**)$esp + 4)
0xfffffce5:   "/home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)$esp + 8)
0xfffffce9:   "аргумент1"
(gdb) x/s *(void**)$esp + 12)
0xffffccb:    "аргумент"
(gdb) x/s *(void**)$esp + 12)
0xffffccb:    "аргумент"
(gdb) x/s *(void**)$esp + 16)
0xffffcccd:   "2"
(gdb) x/s *(void**)$esp + 20)
0xffffccde:   "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) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ cp ..//lab08/lab08-ind.asm lab09-ind.asm
(base) hh@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

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) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09$ |
```

The image shows two terminal windows side-by-side. Both windows have a title bar 'GNU nano 7.2' and a file name 'lab09-ind.asm *'. The top window contains the assembly code for a simple program. The bottom window shows the same code with additional assembly instructions for printing the result.

```

GNU nano 7.2                                     lab09-ind.asm *
;include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB 'f(x)=',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 _calculate

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

    call quit

._calculate:
    mov ebx, 2
    mul ebx
    add eax, 5
    mov [res], eax
    ret
|

```

At the bottom of the bottom window, there is a command-line history:

```

(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ nano lab09-ind.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ nasm -f elf lab09-ind.asm
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ ld -m elf_i386 -o lab09-ind
lab09-ind.o
(base) hhl@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ ./lab09-ind
Введите x: 2
f(x)=9

```

*** 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

The screenshot shows a terminal window with the following content:

```
GNU nano 7.2                                         lab09-4.asm *
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx

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

(base) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ nano lab09-4.asm
(base) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ nasm -f elf -g lab09-4.asm
(base) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
(base) hh@LAPTOP-058L8DAA:~/work/study/2025-2026/архитектура компьютера/arch-pc/labs/Lab09$ ./lab09-4
Результат: 10
```

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

3.3 Шаг отладки: Анализ ошибки в GDB

The screenshot shows two terminal windows for GDB. The top window displays assembly code and a register dump. The assembly code is:

```
B+> 10    mov ebx,3
11    mov eax,2
12    add ebx,eax
13    mov ecx,4
14    mul ecx
15    add ebx,5
16    mov edi,ebx
17
```

The register dump shows:

Register	Value	Description
eax	0x0	0
edx	0x0	0
esp	0xfffffccb0	0xfffffccb0
esi	0x0	0
eip	0x80490e8	0x80490e8 <_start>
cs	0x23	35
ds	0x2b	43
fs	0x0	0
ecx	0x0	0
ebx	0x0	0
ebp	0x0	0x0
edi	0x0	0
eflags	0x202	[IF]
ss	0x2b	43
es	0x2b	43
gs	0x0	0

The bottom window shows the assembly code again and some help text:

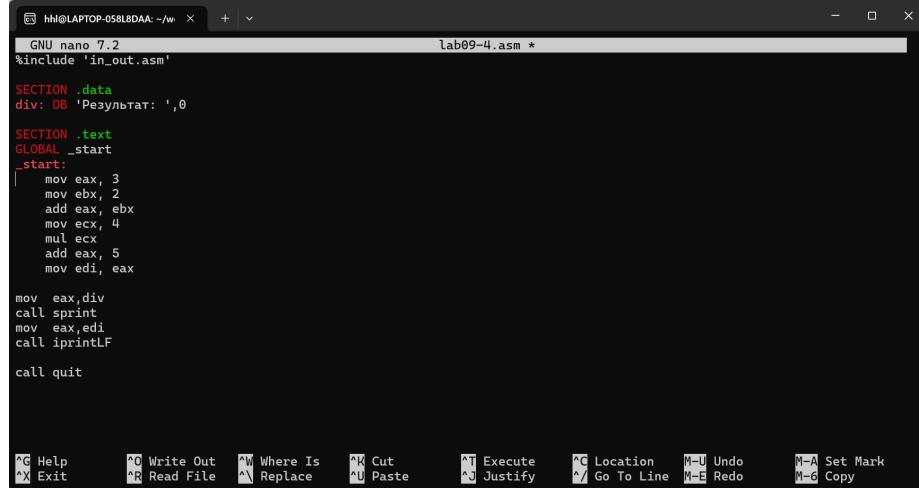
```
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
```

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from lab09-4...
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab09-4.asm, line 10.
(gdb) run
Starting program: /home/hhl/work/study/2025-2026/архитектура компьютера/arch-pc/labs/lab09-4

This GDB supports auto-downloading debuginfo from the following URLs:
<<https://debuginfod.ubuntu.com>>

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

3.4 Исправленная программа



The screenshot shows a terminal window titled "GNU nano 7.2" with the file "lab09-4.asm" open. The assembly code is as follows:

```
GNU nano 7.2                                lab09-4.asm *
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
    mov eax, 3
    mov ebx, 2
    add eax, ebx
    mov ecx, 4
    mul ecx
    add eax, 5
    mov edi, eax

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

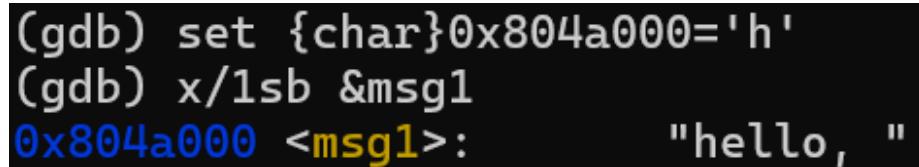
    call quit


```

At the bottom of the window, there is a menu bar with Russian labels: Help, Write Out, Where Is, Cut, Execute, Location, Undo, Set Mark, Exit, Read File, Replace, Paste, Justify, Go To Line, Redo, and Copy.

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

3.5 Результат исправленной программы



The screenshot shows the GDB debugger. The command "(gdb) x/1sb &msg1" is entered, and the output shows the memory at address 0x804a000 contains the string "hello, ".

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

* **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.