

Отчёт по лабораторной работе №10

Дисциплина: Архитектура компьютера

Барсегян Вардан Левонович НПИбд-01-22

Содержание

Цель работы.....	3
Теоретическое введение.....	4
Выполнение лабораторной работы.....	5
Реализация подпрограмм в NASM.....	5
Отладка программ с помощью GDB.....	9
Добавление точек останова.....	17
Работа с данными программы в GDB.....	18
Обработка аргументов командной строки в GDB.....	24
Задания для самостоятельной работы.....	29
Выводы.....	36

Цель работы

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

Теоретическое введение

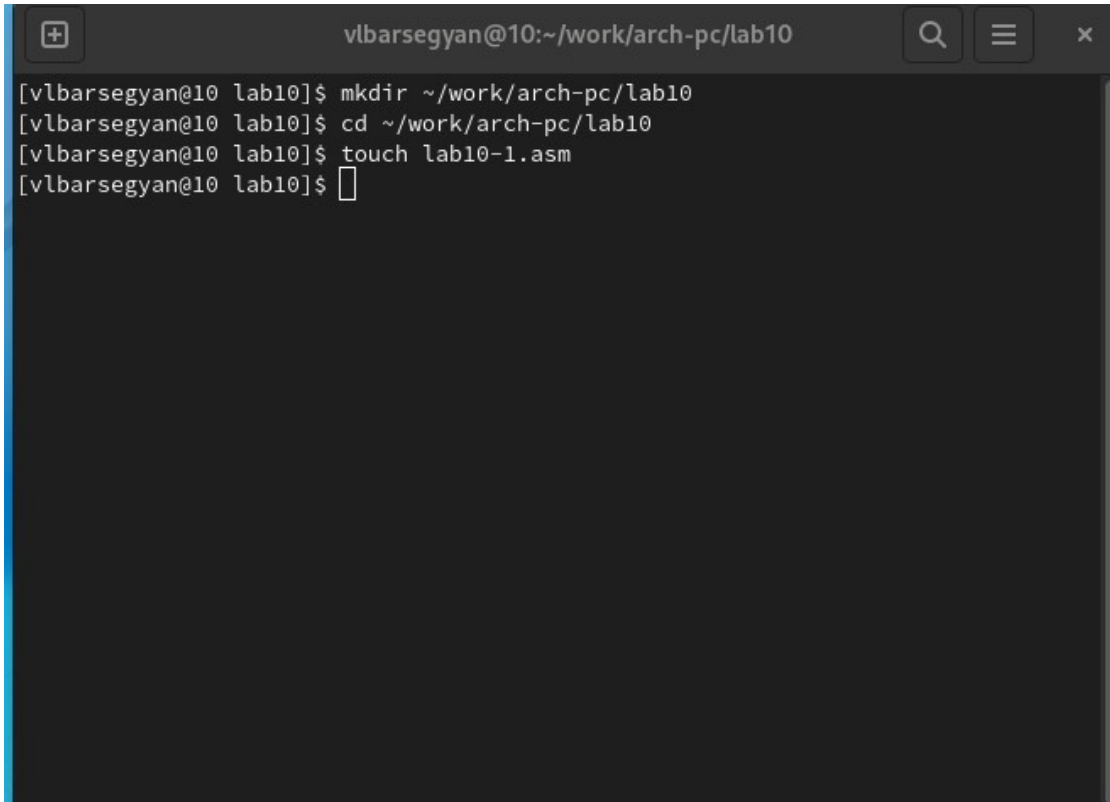
Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Выполнение лабораторной работы

Реализация подпрограмм в NASM

1. Создаю каталог для выполнения лабораторной работы № 10, перехожу в него и создаю файл *lab10-1.asm* (рис. 1)

A screenshot of a terminal window with a dark background. The window title is 'vlbarsegyan@10:~/work/arch-pc/lab10'. The terminal shows four lines of commands and their outputs: 1. '[vlbarsegyan@10 lab10]\$ mkdir ~/work/arch-pc/lab10' followed by a new line. 2. '[vlbarsegyan@10 lab10]\$ cd ~/work/arch-pc/lab10' followed by a new line. 3. '[vlbarsegyan@10 lab10]\$ touch lab10-1.asm' followed by a new line. 4. '[vlbarsegyan@10 lab10]\$' followed by a cursor. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

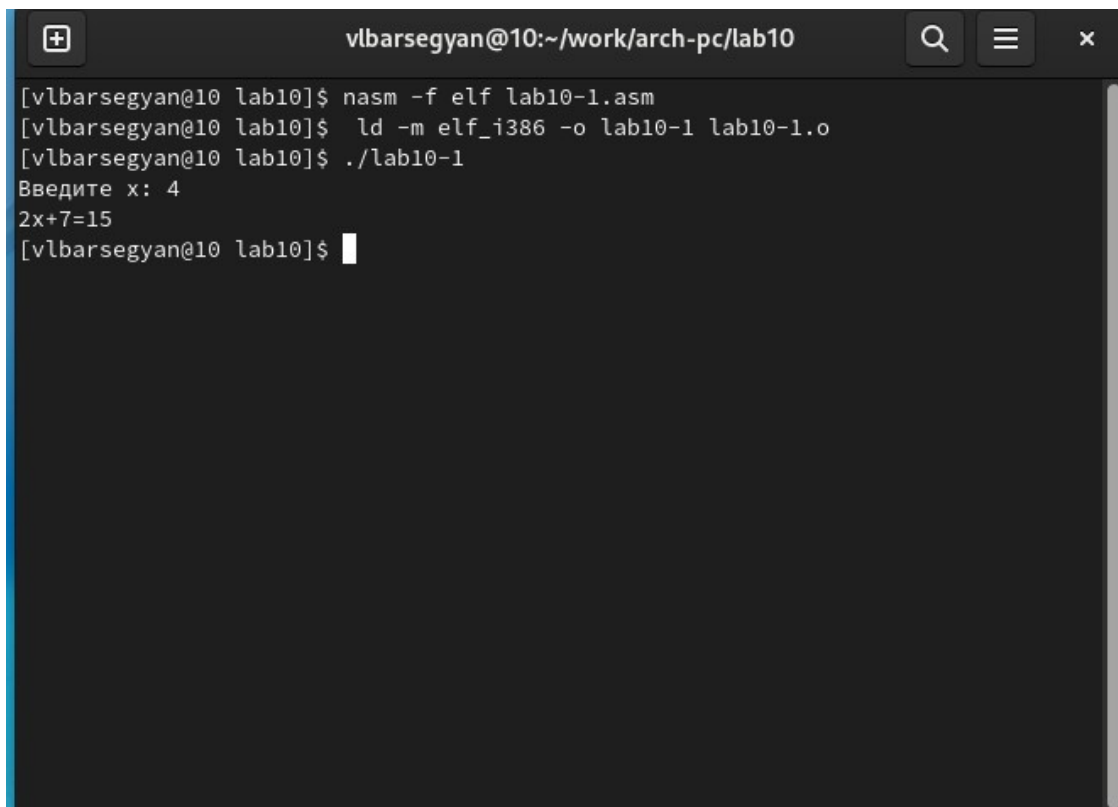
```
[vlbarsegyan@10 lab10]$ mkdir ~/work/arch-pc/lab10
[vlbarsegyan@10 lab10]$ cd ~/work/arch-pc/lab10
[vlbarsegyan@10 lab10]$ touch lab10-1.asm
[vlbarsegyan@10 lab10]$
```

Рис. 1: Создание каталога и файла

2. Ввожу в файл текст программы листинга 1 - программа вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы *_calcul*. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме (рис. 2), компилирую и запускаю исполняемый файл (рис. 3)

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rezs: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [rezs]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [rezs], eax
35 ret ; выход из подпрограммы
```

Рис. 2: Текст программы листинга 1

A terminal window with a dark background and light gray text. The window title is 'vlbarsegyan@10:~/work/arch-pc/lab10'. The terminal shows the following commands and output:

```
[vlbarsegyan@10 lab10]$ nasm -f elf lab10-1.asm
[vlbarsegyan@10 lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[vlbarsegyan@10 lab10]$ ./lab10-1
Введите x: 4
2x+7=15
[vlbarsegyan@10 lab10]$
```

Рис. 3: Компиляция и запуск исполняемого файла

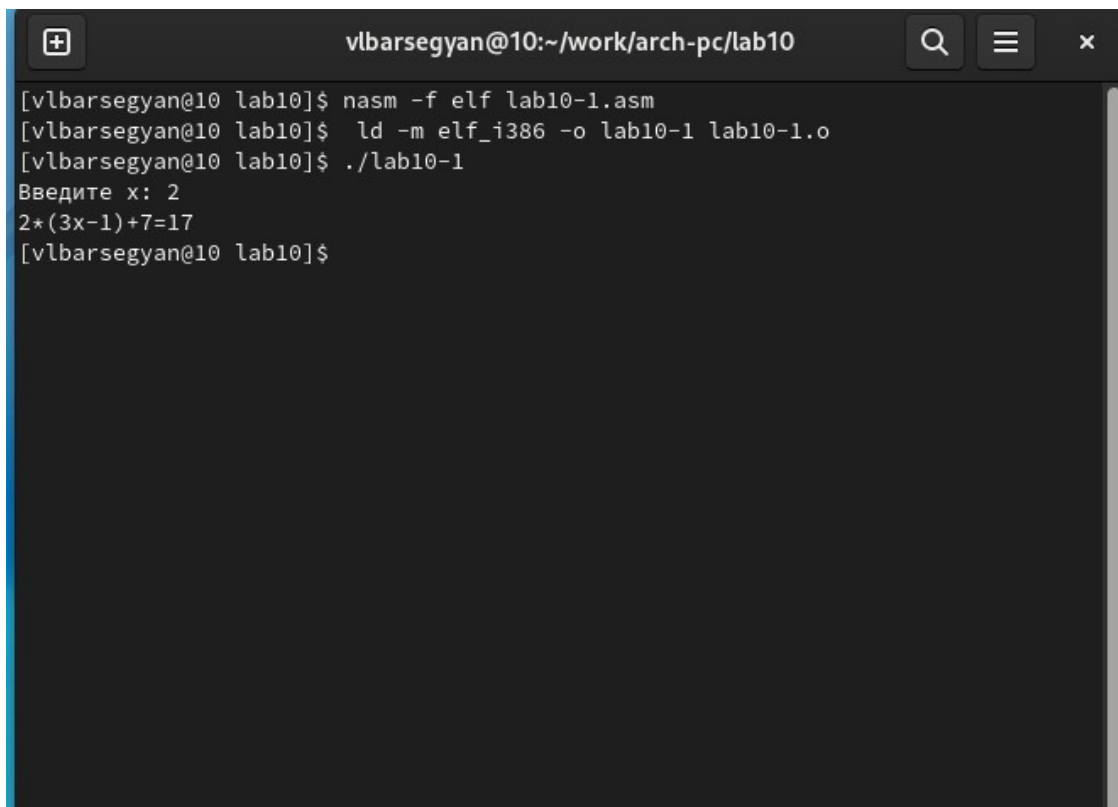
3. Изменяю текст программы, добавив подпрограмму `_subcalcul` (рис. 4).
Компилирую исполняемый файл и проверяю его работу (рис. 5)

```

15 ; основная программа
16 ;-----
17 mov eax, msg
18 call sprint
19 mov ecx, x
20 mov edx, 80
21 call sread
22 mov eax, x
23 call atoi
24 call _calcul ; Вызов подпрограммы _calcul
25 mov eax, result
26 call sprint
27 mov eax, [rezs]
28 call iprintLF
29 call quit
30
31 ;-----
32 ; Подпрограмма вычисления
33 ; выражения "2x+7"
34 _calcul:
35 call _subcalcul
36 mov ebx, 2
37 mul ebx
38 add eax, 7
39 mov [rezs], eax
40 ret ; выход из подпрограммы
41
42 _subcalcul:
43 mov ebx, 3
44 mul ebx
45 sub eax, 1
46 ret ; выход из подпрограммы

```

Рис. 4: Обновленный экст программы

A terminal window with a dark background and light text. The window title is 'vlbarsegyan@10:~/work/arch-pc/lab10'. It contains the following text:

```
[vlbarsegyan@10 lab10]$ nasm -f elf lab10-1.asm
[vlbarsegyan@10 lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[vlbarsegyan@10 lab10]$ ./lab10-1
Введите x: 2
2*(3x-1)+7=17
[vlbarsegyan@10 lab10]$
```

Рис. 5: Создание и запуск исполняемого файла

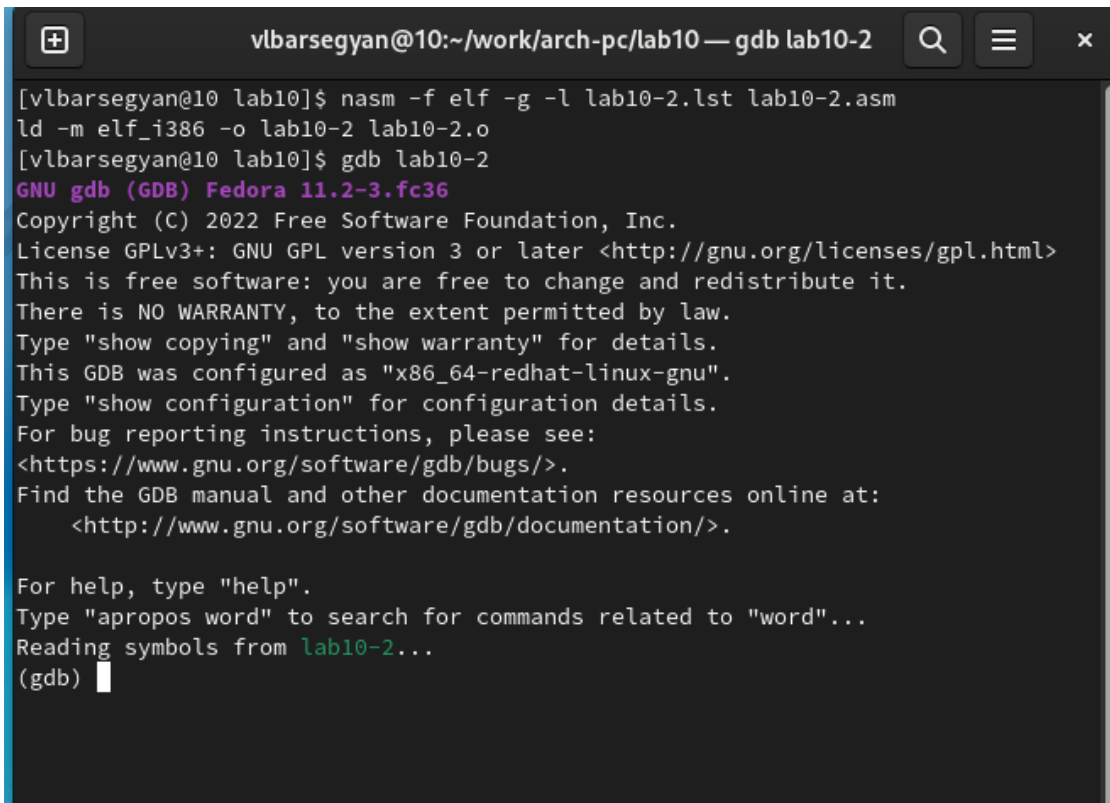
Отладка программ с помощью GDB

4. Создаю файл *lab10-2.asm* с помощью команды *touch lab10-2.asm* с текстом программы из Листинга 10.2 (Программа печати сообщения Hello world!) (рис. 6)

report.md	lab10-1.asm	lab10-2.asm	x
<pre>1 SECTION .data 2 msg1: db "Hello, ",0x0 3 msg1Len: equ \$ - msg1 4 msg2: db "world!",0xa 5 msg2Len: equ \$ - msg2 6 SECTION .text 7 global _start 8 _start: 9 mov eax, 4 10 mov ebx, 1 11 mov ecx, msg1 12 mov edx, msg1Len 13 int 0x80 14 mov eax, 4 15 mov ebx, 1 16 mov ecx, msg2 17 mov edx, msg2Len 18 int 0x80 19 mov eax, 1 20 mov ebx, 0 21 int 0x80</pre>			

Рис. 6: Текст программы файла lab10-2.asm

5. Создаю исполняемый файл, добавив отладочную информацию с помощью ключа `-g` и загружаю его в отладчик GDB (рис. 7)



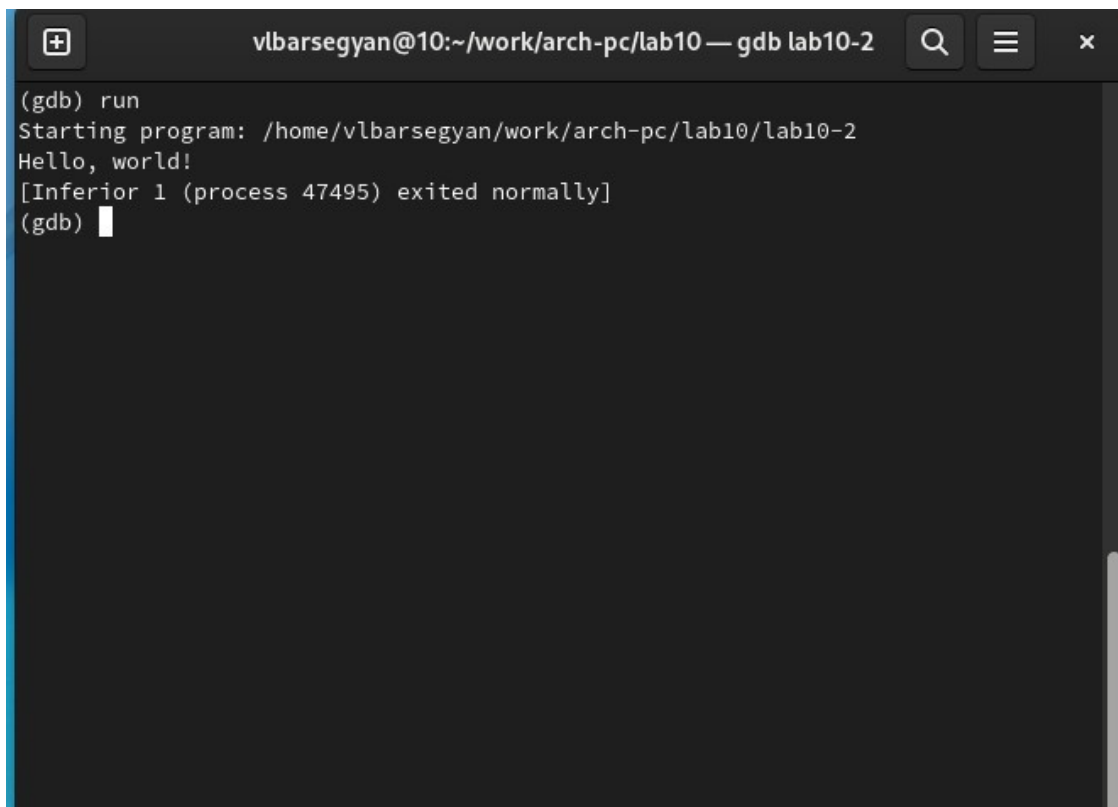
```

[vlbarsegyan@10 lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
ld -m elf_i386 -o lab10-2 lab10-2.o
[vlbarsegyan@10 lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 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 lab10-2...
(gdb) 
```

Рис. 7: Компиляция исполняемого файла и его загрузка в отладчик GDB

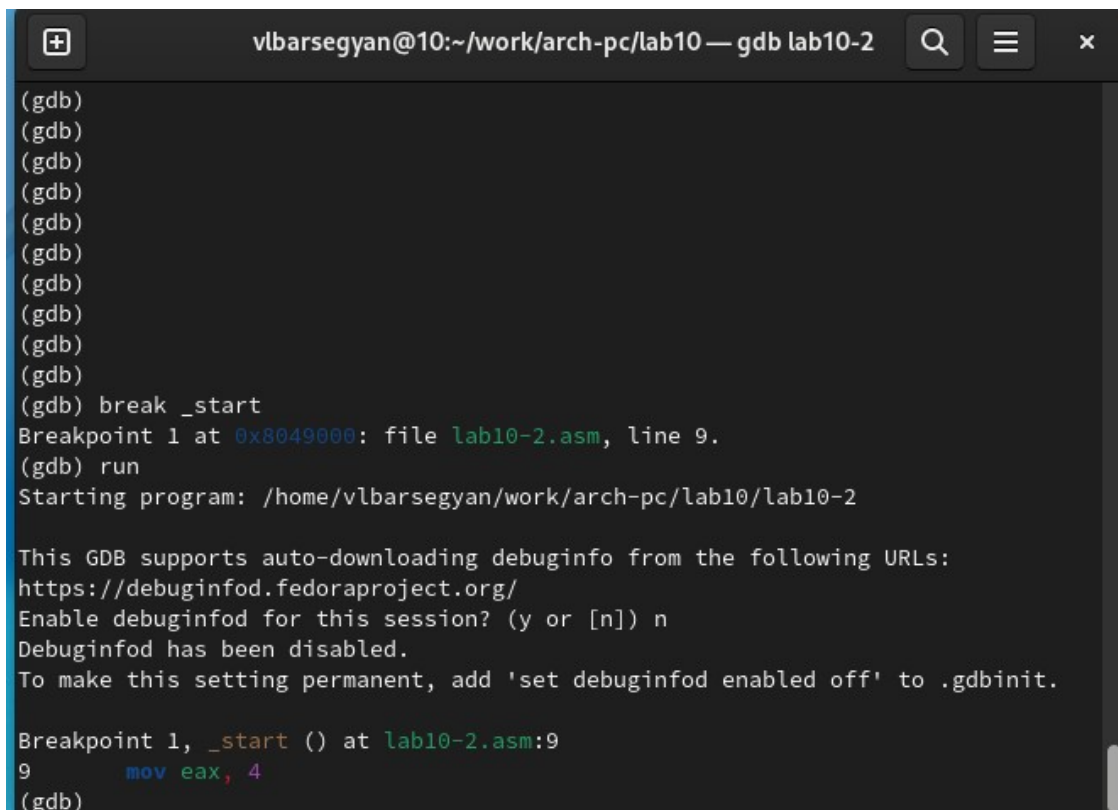
6. Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. 8)

A screenshot of a GDB terminal window. The window title is "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2". The terminal shows the following text: (gdb) run, Starting program: /home/vlbarsegyan/work/arch-pc/lab10/lab10-2, Hello, world!, [Inferior 1 (process 47495) exited normally], and (gdb) followed by a cursor. The window has a dark background and standard window controls (search, menu, close) in the title bar.

```
(gdb) run
Starting program: /home/vlbarsegyan/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 47495) exited normally]
(gdb) 
```

Рис. 8: Запуск программы в оболочке GDB

7. Устанавливаю брейкпоинт на метку `_start` и запускаю её (рис. 9)



```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/vlbarsegyan/work/arch-pc/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
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 lab10-2.asm:9
9      mov eax, 4
(gdb)
```

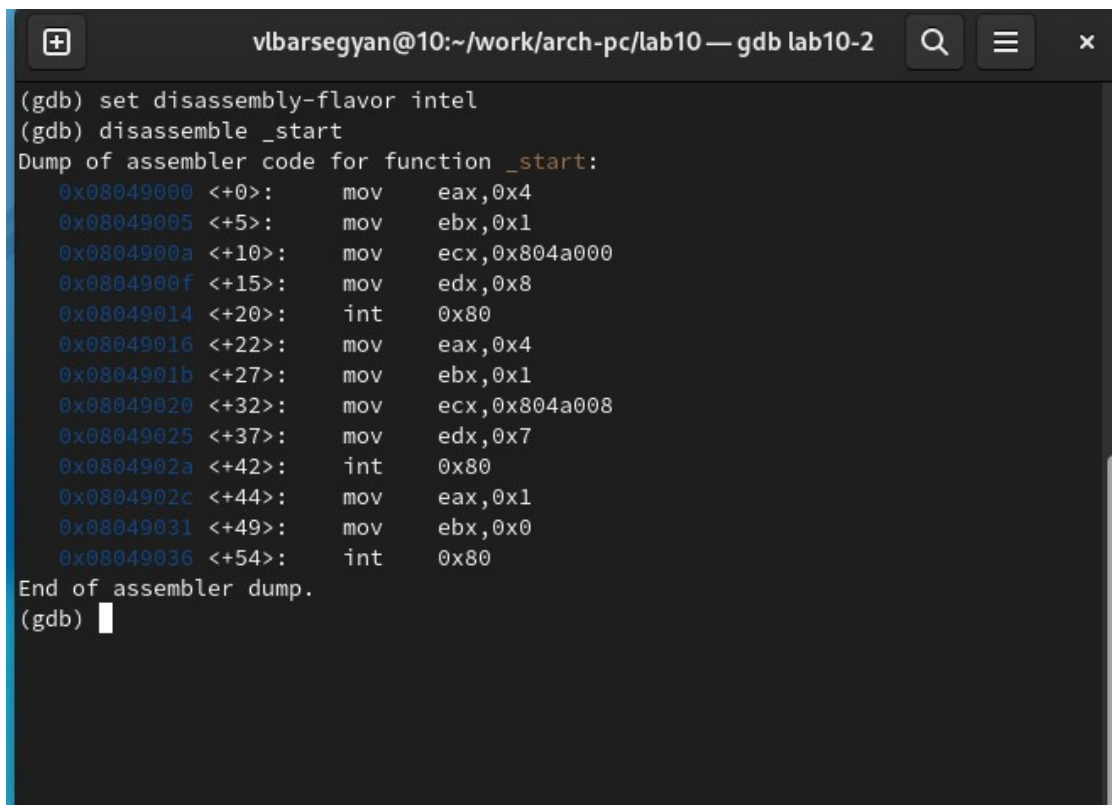
Рис. 9: Установка брейкпоинта и запуск

8. Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 10)

```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(gdb)
(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)
```

Рис. 10: Дисассимилированный код программы

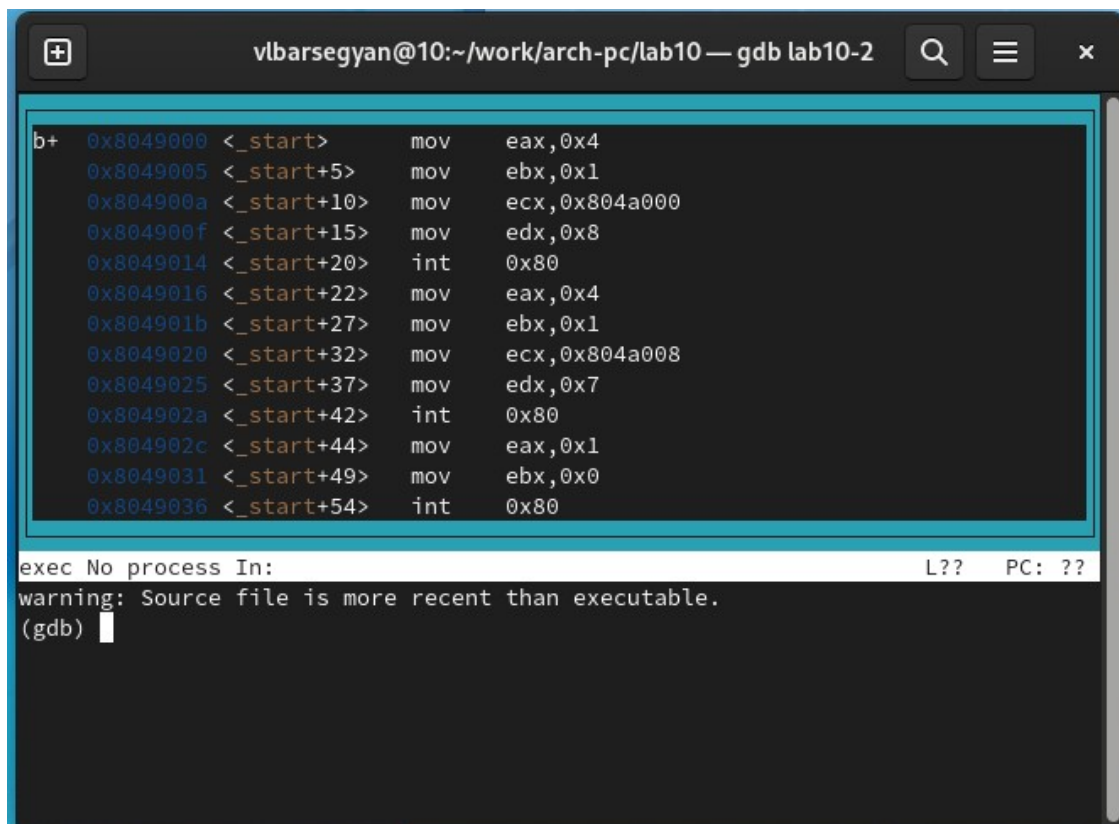
9. Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду *set disassembly-flavor intel* (рис. 11). После смены интерфейса команды отображаются с привычным Intel'овским синтаксисом



```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2
(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.
(gdb) 
```

Рис. 11: Дисассимилированный код программы в режиме Intel

10. Включаю режим псевдографики для более удобного анализа программы (рис. 12, 13)

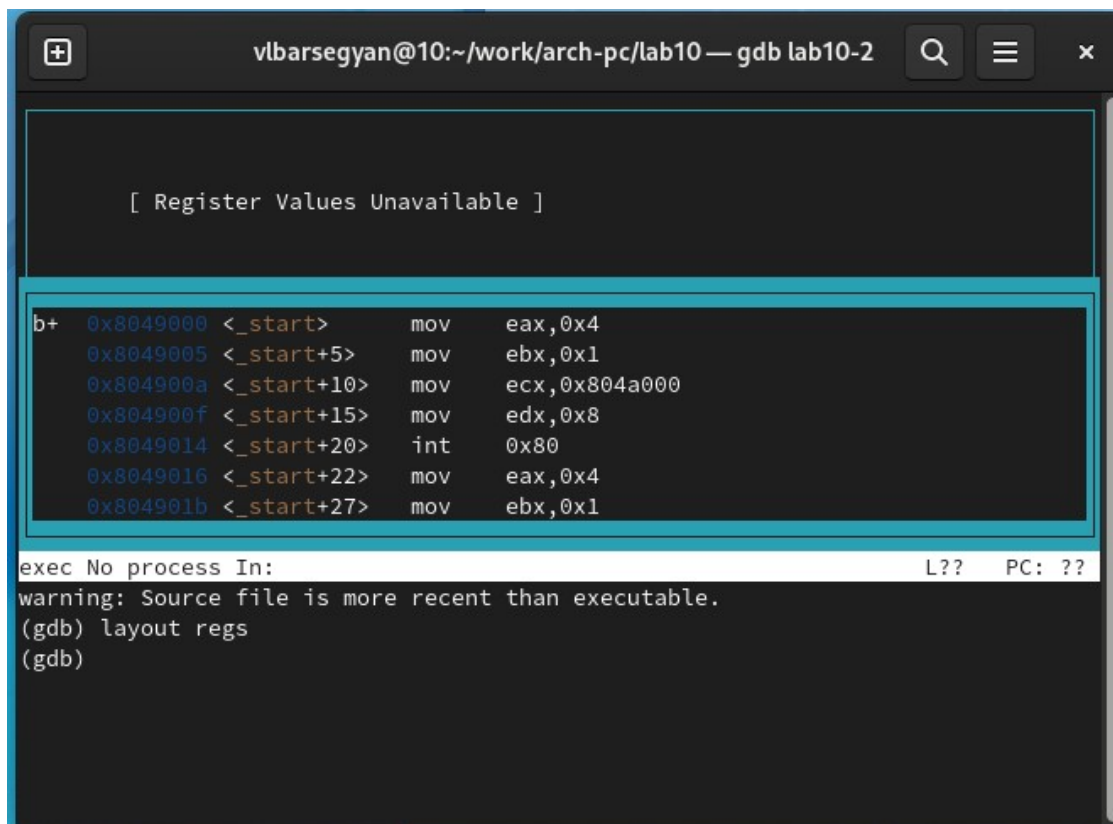


The screenshot shows a GDB terminal window with the title bar "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2". The main area displays assembly code for a function starting at address 0x8049000. The code includes several instructions: `mov eax,0x4`, `mov ebx,0x1`, `mov ecx,0x804a000`, `mov edx,0x8`, `int 0x80`, `mov eax,0x4`, `mov ebx,0x1`, `mov ecx,0x804a008`, `mov edx,0x7`, `int 0x80`, `mov eax,0x1`, `mov ebx,0x0`, and `int 0x80`. Below the assembly code, the status bar shows "exec No process in:" and "L?? PC: ??". A warning message "warning: Source file is more recent than executable." is displayed, followed by the prompt "(gdb) ".

```
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
    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

exec No process in:                                     L??  PC: ??
warning: Source file is more recent than executable.
(gdb) 
```

Рис. 12: Включение режима псевдографики



The screenshot shows a GDB window titled "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2". The main pane displays assembly code with addresses and instructions. A cyan box highlights the instructions from 0x8049000 to 0x804901b. Below the assembly pane, the status bar shows "exec No process in:" and "L?? PC: ??". The console pane shows the command "(gdb) layout regs" and a warning: "warning: Source file is more recent than executable."

```
[ Register Values Unavailable ]
```

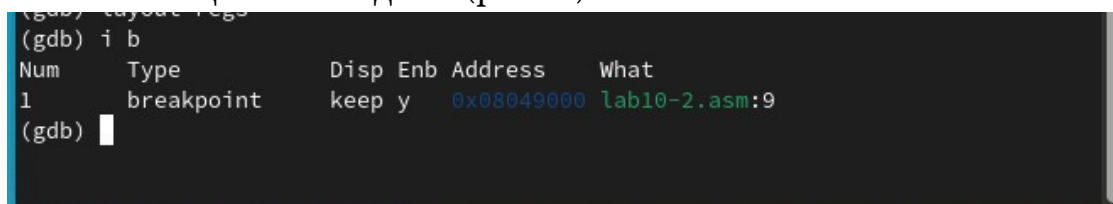
Address	Instruction
0x8049000	<_start> mov eax,0x4
0x8049005	<_start+5> mov ebx,0x1
0x804900a	<_start+10> mov ecx,0x804a000
0x804900f	<_start+15> mov edx,0x8
0x8049014	<_start+20> int 0x80
0x8049016	<_start+22> mov eax,0x4
0x804901b	<_start+27> mov ebx,0x1

```
exec No process in: L?? PC: ??
warning: Source file is more recent than executable.
(gdb) layout regs
(gdb)
```

Рис. 13: Включение режима псевдографики

Добавление точек останова

11. Проверяю установку точки останова по имени метки (`_start`) с помощью команды `i b` (рис. 14)



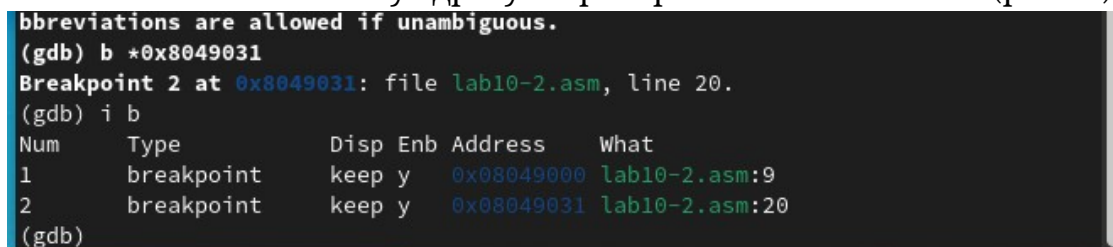
The screenshot shows the GDB console with the command "(gdb) i b". The output is a table showing the breakpoint at address 0x08049000.

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x08049000	lab10-2.asm:9

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab10-2.asm:9
(gdb)
```

Рис. 14: Проверка установки точки

12. Адрес предпоследней инструкции - 0x8049031. Устанавливаю точку останова по этому адресу и проверяю все такие точки (рис. 15)



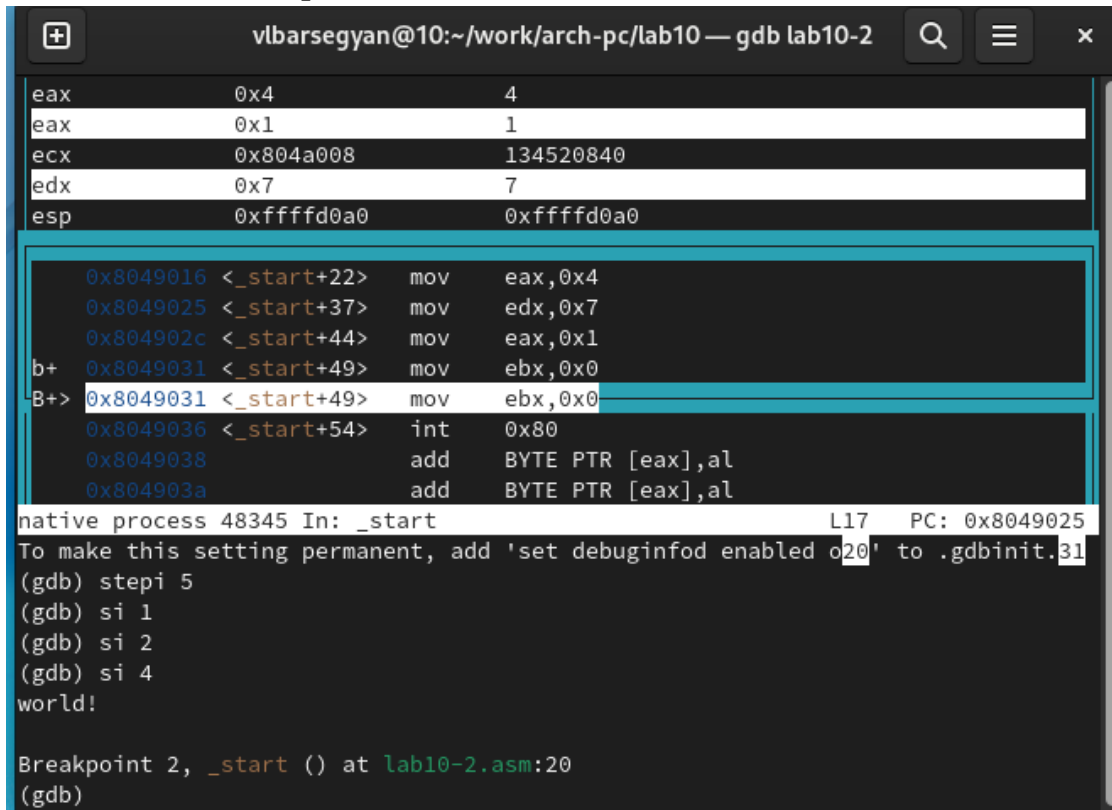
The screenshot shows the GDB console with the command "(gdb) b *0x8049031". The output shows a new breakpoint at address 0x8049031. Then, the command "(gdb) i b" is executed, showing both breakpoints.

```
bbreviations are allowed if unambiguous.
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab10-2.asm:9
2        breakpoint keep y  0x08049031 lab10-2.asm:20
(gdb)
```

Рис. 15: Установка точки останова и проверка все таких точек

Работа с данными программы в GDB

13. Выполняю 5 инструкций `stepi` (`si`). Каждый раз выполняется соответствующая инструкция, и подсвечивается соответствующее изменение (рис. 16)



```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2
eax      0x4      4
ecx      0x804a008 134520840
edx      0x7      7
esp      0xffffd0a0 0xffffd0a0

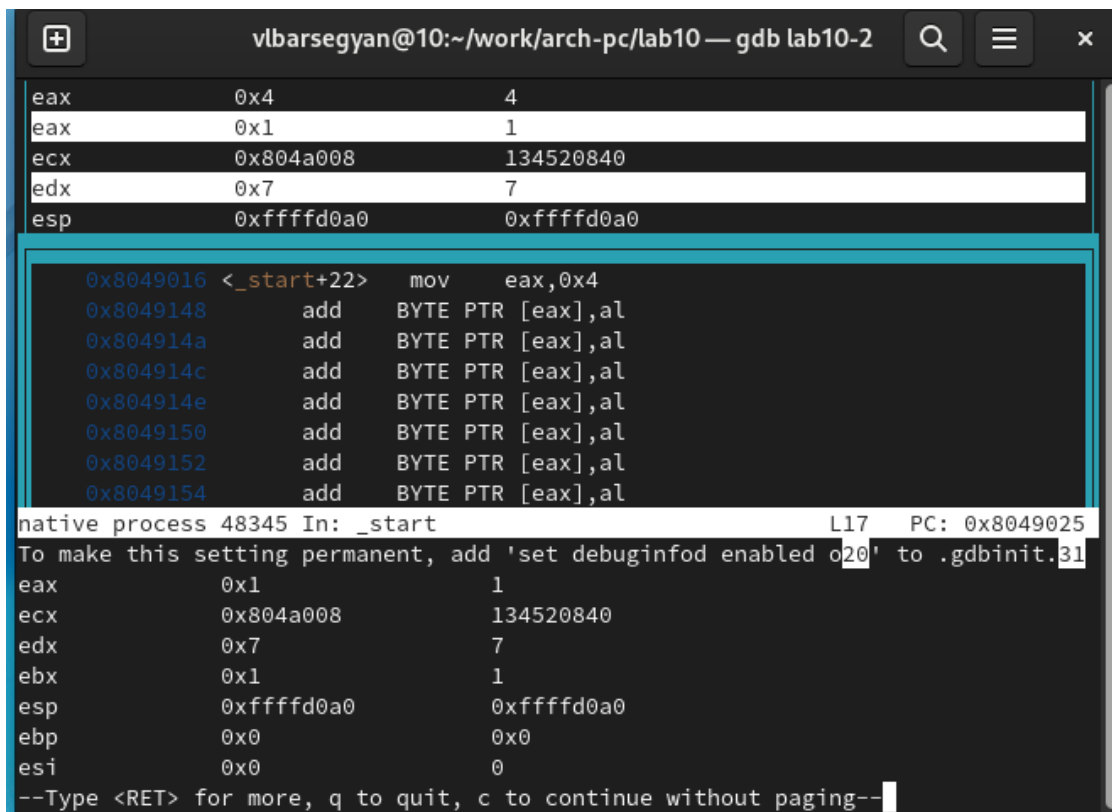
0x8049016 <_start+22> mov    eax,0x4
0x8049025 <_start+37> mov    edx,0x7
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
B+> 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038      add    BYTE PTR [eax],al
0x804903a      add    BYTE PTR [eax],al

native process 48345 In: _start L17 PC: 0x8049025
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.31
(gdb) stepi 5
(gdb) si 1
(gdb) si 2
(gdb) si 4
world!

Breakpoint 2, _start () at lab10-2.asm:20
(gdb)
```

Рис. 16: Выполнение инструкции `stepi`

14. Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). (рис. 17)



The screenshot shows a GDB terminal window with the title bar 'vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2'. The window is divided into two main sections. The top section displays the current values of several CPU registers:

eax	0x4	4
ecx	0x804a008	134520840
edx	0x7	7
esp	0xffffd0a0	0xffffd0a0

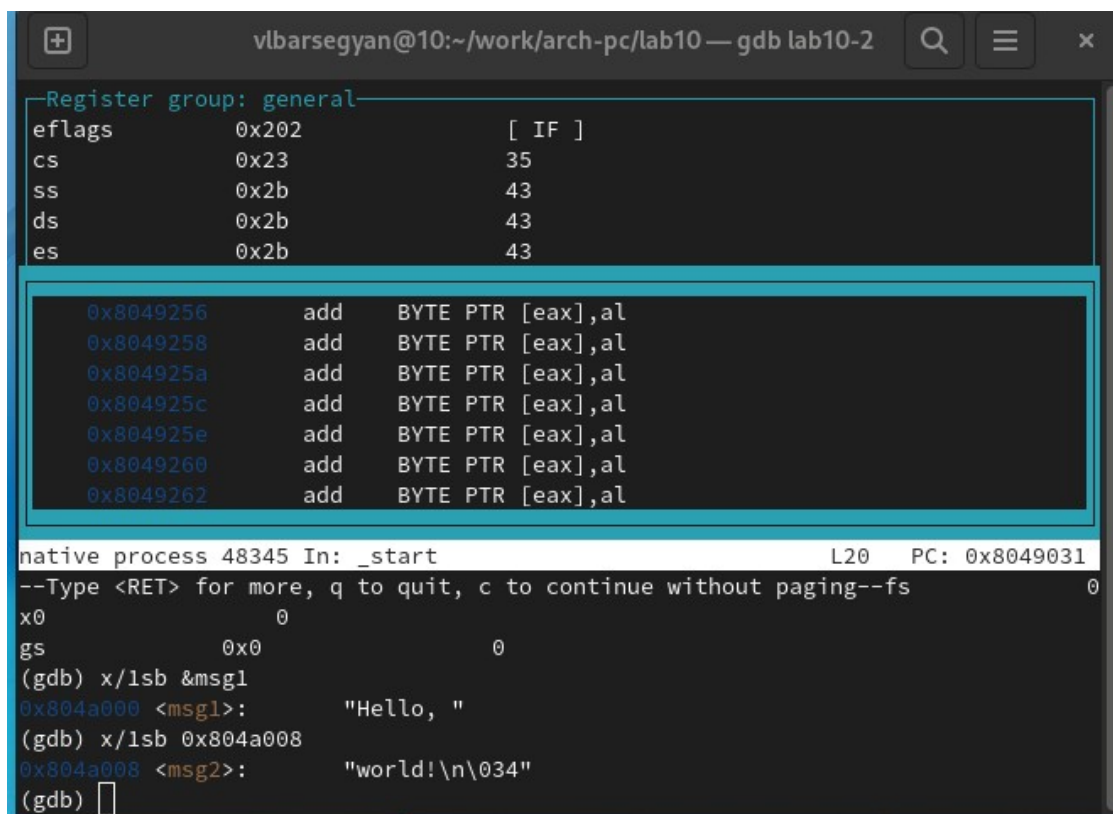
. The bottom section shows a list of assembly instructions with their addresses and the current instruction pointer (PC) at 0x8049025. The instructions are:

0x8049016	<_start+22>	mov	eax,0x4
0x8049148		add	BYTE PTR [eax],al
0x804914a		add	BYTE PTR [eax],al
0x804914c		add	BYTE PTR [eax],al
0x804914e		add	BYTE PTR [eax],al
0x8049150		add	BYTE PTR [eax],al
0x8049152		add	BYTE PTR [eax],al
0x8049154		add	BYTE PTR [eax],al

. Below the assembly list, the GDB status bar shows 'native process 48345 In: _start' and 'L17 PC: 0x8049025'. A message from GDB suggests adding a setting to .gdbinit. At the bottom, a prompt asks to type <RET> for more, q to quit, or c to continue without paging.

Рис. 17: Просмотр содержимого регистров

15. Смотрю значение переменной msg1 по имени, а значение переменной msg2 - по адресу (рис. 18)



The screenshot shows a GDB terminal window with the title bar "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2". The window is divided into several sections. The top section, titled "Register group: general", lists the values of several registers: eflags (0x202, [IF]), cs (0x23, 35), ss (0x2b, 43), ds (0x2b, 43), and es (0x2b, 43). Below this, a list of assembly instructions is displayed, each starting with an address and followed by the instruction: "add BYTE PTR [eax], al". The addresses range from 0x8049256 to 0x8049262. The bottom section of the window shows the state of the native process 48345, including the instruction pointer (PC) at 0x8049031. It also displays the contents of memory locations 0x804a000 and 0x804a008, which contain the strings "Hello, " and "world!\n\034" respectively. The GDB prompt "(gdb)" is visible at the bottom.

```
Register group: general
eflags      0x202      [ IF ]
cs          0x23      35
ss          0x2b      43
ds          0x2b      43
es          0x2b      43

0x8049256    add     BYTE PTR [eax], al
0x8049258    add     BYTE PTR [eax], al
0x804925a    add     BYTE PTR [eax], al
0x804925c    add     BYTE PTR [eax], al
0x804925e    add     BYTE PTR [eax], al
0x8049260    add     BYTE PTR [eax], al
0x8049262    add     BYTE PTR [eax], al

native process 48345 In: _start          L20    PC: 0x8049031
--Type <RET> for more, q to quit, c to continue without paging--fs      0
x0          0
gs          0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) 
```

Рис. 18: Просмотр содержимого переменных

16. Изменяю первый символ переменной `msg1` и два символа в переменной `msg2`, используя ее адрес (рис. 19)

```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eflags    0x202      [ IF ]
cs         0x23      35
ss         0x2b      43
ds         0x2b      43
es         0x2b      43

0x8049256   add     BYTE PTR [eax],al
0x8049258   add     BYTE PTR [eax],al
0x804925a   add     BYTE PTR [eax],al
0x804925c   add     BYTE PTR [eax],al
0x804925e   add     BYTE PTR [eax],al
0x8049260   add     BYTE PTR [eax],al
0x8049262   add     BYTE PTR [eax],al

native process 48345 In: _start          L20    PC: 0x8049031
(gdb) set {char}&msg1='r'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "rello, "
(gdb) set {char}0x804a008='P'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Por d!\n\034"
(gdb)
```

Рис. 19: Изменение символов переменных

17. Вывожу значение регистра `edx` в различных форматах (`p/x` - шестнадцатеричный, `p/t` - двоичный, `p/s` - символьный) (рис. 20)

The screenshot shows a GDB terminal window with the title bar 'vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2'. The window is divided into three main sections. The top section, titled 'Register group: general', lists the values of several registers: eflags (0x202, [IF]), cs (0x23, 35), ss (0x2b, 43), ds (0x2b, 43), and es (0x2b, 43). The middle section, which is highlighted with a red rectangle, displays a list of assembly instructions at memory addresses 0x8049256 through 0x8049262, all of which are 'add BYTE PTR [eax], al'. The bottom section shows the current state of the process: 'native process 48345 In: _start' with 'L20' and 'PC: 0x8049031'. Below this, several GDB commands and their outputs are shown: '\$4 = 0x7', '(gdb) p/x \$edx' resulting in '\$5 = 0x7', '(gdb) p/t \$edx' resulting in '\$6 = 111', and '(gdb) p/s \$edx' resulting in '\$7 = 7'. The prompt '(gdb) ' is visible at the bottom.

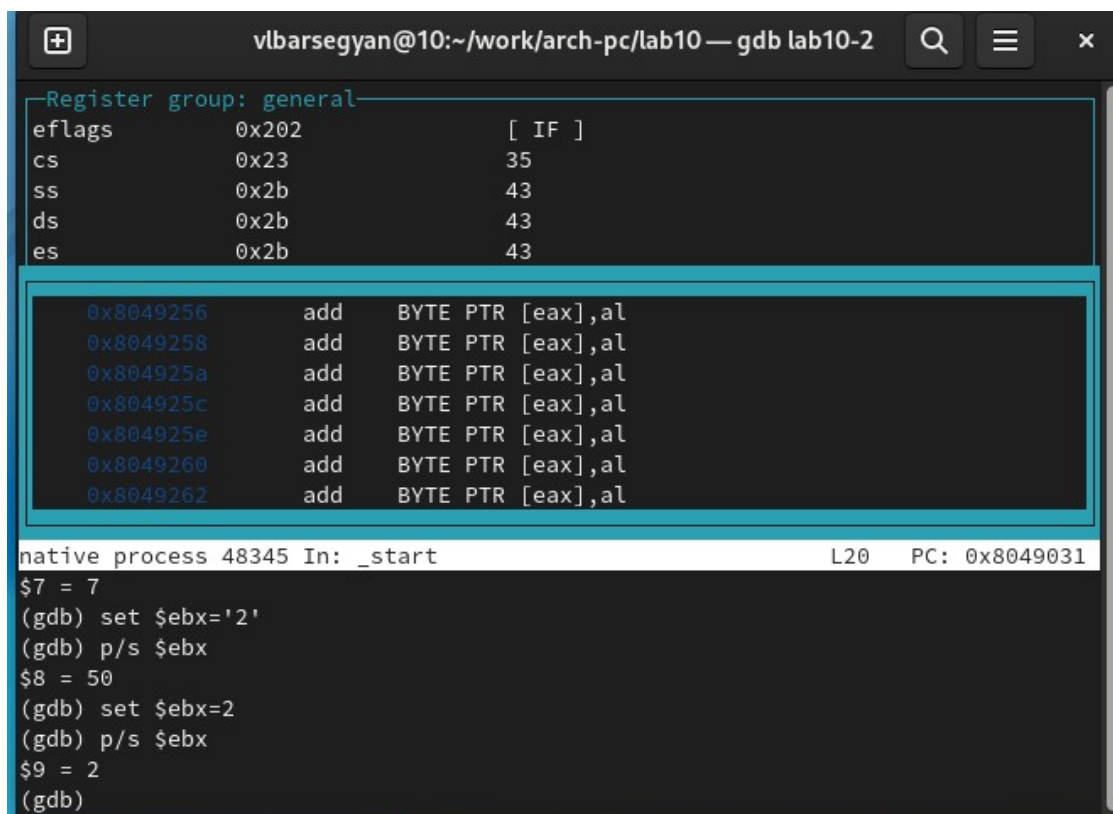
```
Register group: general
eflags      0x202      [ IF ]
cs           0x23      35
ss           0x2b      43
ds           0x2b      43
es           0x2b      43

0x8049256     add     BYTE PTR [eax], al
0x8049258     add     BYTE PTR [eax], al
0x804925a     add     BYTE PTR [eax], al
0x804925c     add     BYTE PTR [eax], al
0x804925e     add     BYTE PTR [eax], al
0x8049260     add     BYTE PTR [eax], al
0x8049262     add     BYTE PTR [eax], al

native process 48345 In: _start                L20    PC: 0x8049031
$4 = 0x7
(gdb) p/x $edx
$5 = 0x7
(gdb) p/t $edx
$6 = 111
(gdb) p/s $edx
$7 = 7
(gdb) 
```

Рис. 20: Вывод значения регистра в различных форматах

18. С помощью команды `set` изменяю значение регистра `ebx` (рис. 21). Разница в выводе в том, что в первом случае 2 вводится как символ, и в символьном виде выводится `ascii`-номер символа, во втором же случае 2 вводится как число и в символьном виде показывается как число



The screenshot shows a GDB terminal window with the title bar "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2". The window is divided into three main sections. The top section, titled "Register group: general", lists the values of several registers: eflags (0x202, [IF]), cs (0x23, 35), ss (0x2b, 43), ds (0x2b, 43), and es (0x2b, 43). The middle section, which is highlighted with a red rectangle, displays a list of assembly instructions at memory addresses 0x8049256 through 0x8049262, all of which are "add BYTE PTR [eax], al". The bottom section shows the GDB prompt and a series of commands and their outputs: "native process 48345 In: _start" (L20, PC: 0x8049031), "\$7 = 7", "(gdb) set \$ebx='2'", "(gdb) p/s \$ebx", "\$8 = 50", "(gdb) set \$ebx=2", "(gdb) p/s \$ebx", "\$9 = 2", and "(gdb)".

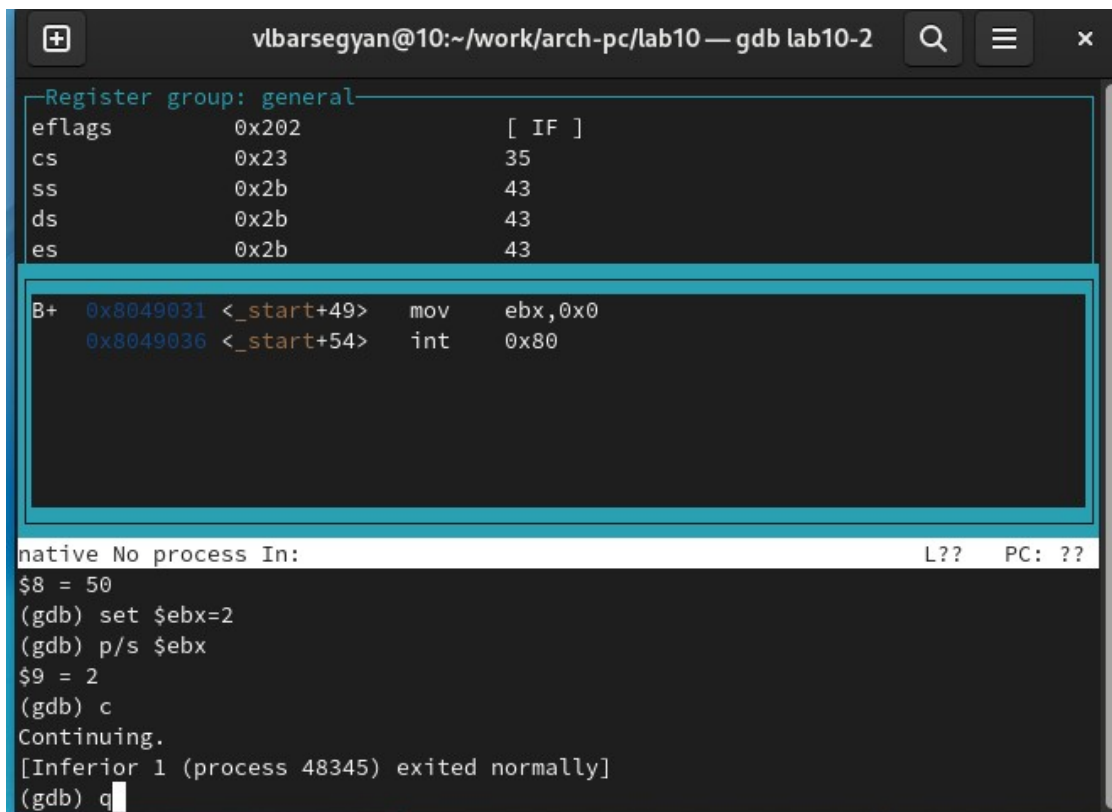
```
Register group: general
eflags      0x202      [ IF ]
cs          0x23      35
ss          0x2b      43
ds          0x2b      43
es          0x2b      43

0x8049256    add     BYTE PTR [eax], al
0x8049258    add     BYTE PTR [eax], al
0x804925a    add     BYTE PTR [eax], al
0x804925c    add     BYTE PTR [eax], al
0x804925e    add     BYTE PTR [eax], al
0x8049260    add     BYTE PTR [eax], al
0x8049262    add     BYTE PTR [eax], al

native process 48345 In: _start      L20    PC: 0x8049031
$7 = 7
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb)
```

Рис. 21: Изменение значения регистра

19. Продолжаю выполнение программы с помощью команды `c` и выхожу из GDB с помощью команды `q` (рис. 22)



The screenshot shows a GDB terminal window with the title bar 'vlbarsegyan@10:~/work/arch-pc/lab10 — gdb lab10-2'. The window is divided into several sections. The top section, titled 'Register group: general', displays the values of several registers: eflags (0x202, [IF]), cs (0x23, 35), ss (0x2b, 43), ds (0x2b, 43), and es (0x2b, 43). Below this, a section of assembly code is highlighted with a red border, showing two instructions: 'B+ 0x8049031 <_start+49> mov ebx,0x0' and '0x8049036 <_start+54> int 0x80'. The bottom section of the window shows the GDB prompt and a series of commands and responses: 'native No process in: L?? PC: ??', '\$8 = 50', '(gdb) set \$ebx=2', '(gdb) p/s \$ebx', '\$9 = 2', '(gdb) c', 'Continuing.', '[Inferior 1 (process 48345) exited normally]', and '(gdb) q'.

```
Register group: general
eflags      0x202      [ IF ]
cs          0x23      35
ss          0x2b      43
ds          0x2b      43
es          0x2b      43

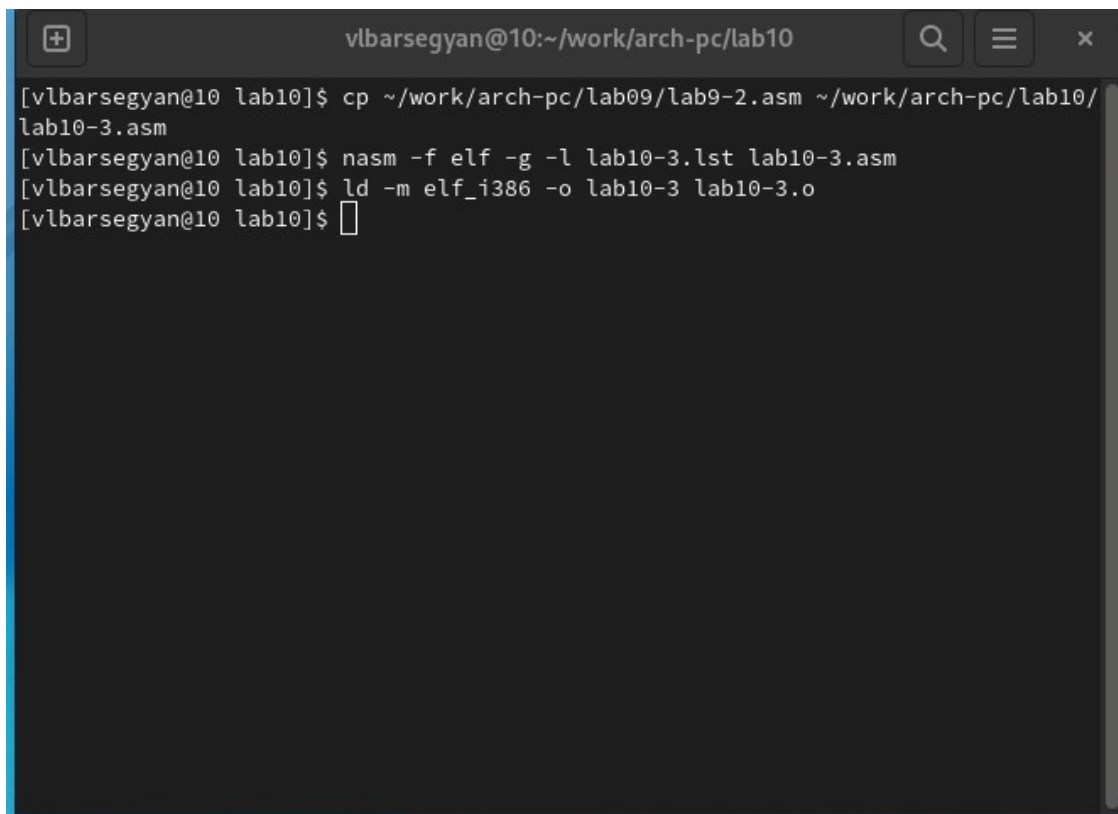
B+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80

native No process in: L?? PC: ??
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb) c
Continuing.
[Inferior 1 (process 48345) exited normally]
(gdb) q
```

Рис. 22: Продолжение выполнения программы и выход из GDB

Обработка аргументов командной строки в GDB

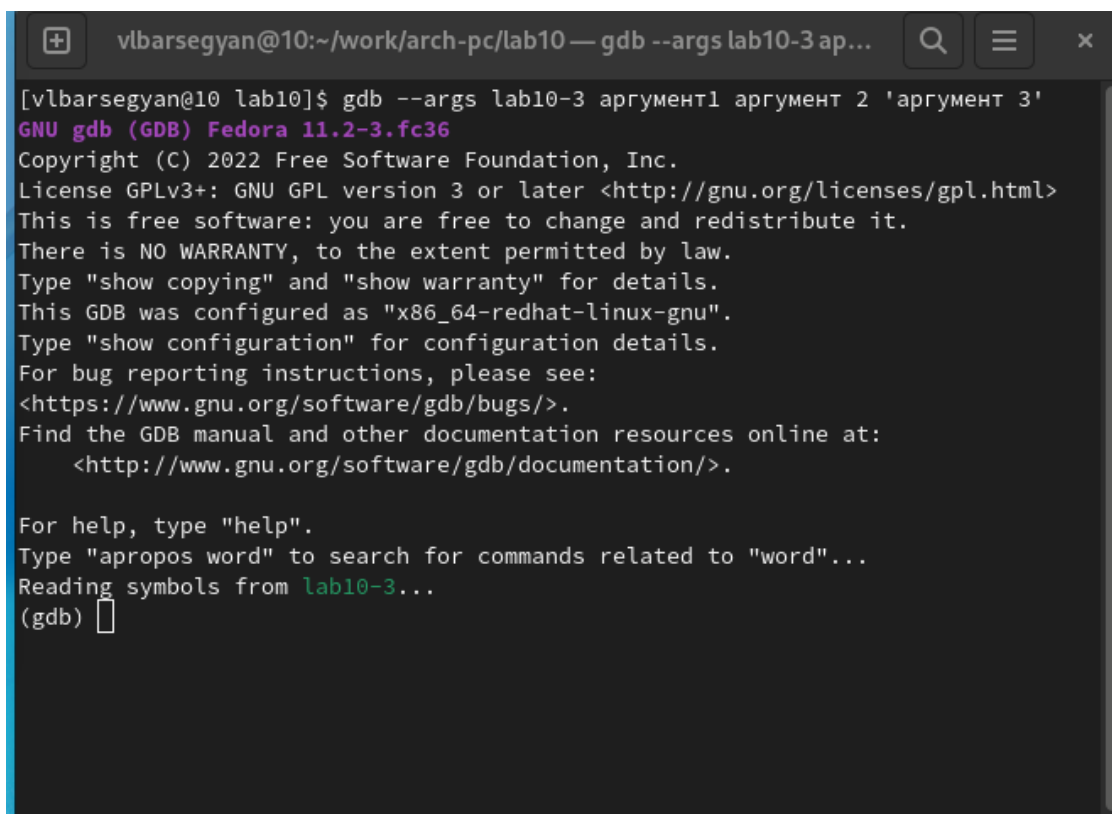
20. Копирую файл *lab9-2.asm*, созданный при выполнении лабораторной работы №9 в файл с именем *lab10-3.asm* и создаю исполняемого файла (рис. 23)

A terminal window with a dark background and light gray text. The window title is 'vlbarsegyan@10:~/work/arch-pc/lab10'. It contains four lines of commands and their outputs: 1. 'cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm' 2. 'nasm -f elf -g -l lab10-3.lst lab10-3.asm' 3. 'ld -m elf_i386 -o lab10-3 lab10-3.o' 4. A blank line with the prompt '[vlbarsegyan@10 lab10]\$' and a cursor.

```
[vlbarsegyan@10 lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/
lab10-3.asm
[vlbarsegyan@10 lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[vlbarsegyan@10 lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[vlbarsegyan@10 lab10]$
```

Рис. 23: Копирование файла, создание исполняемого

21. Загружаю исполняемый файл в отладчик, используя следующие аргументы (рис. 24)

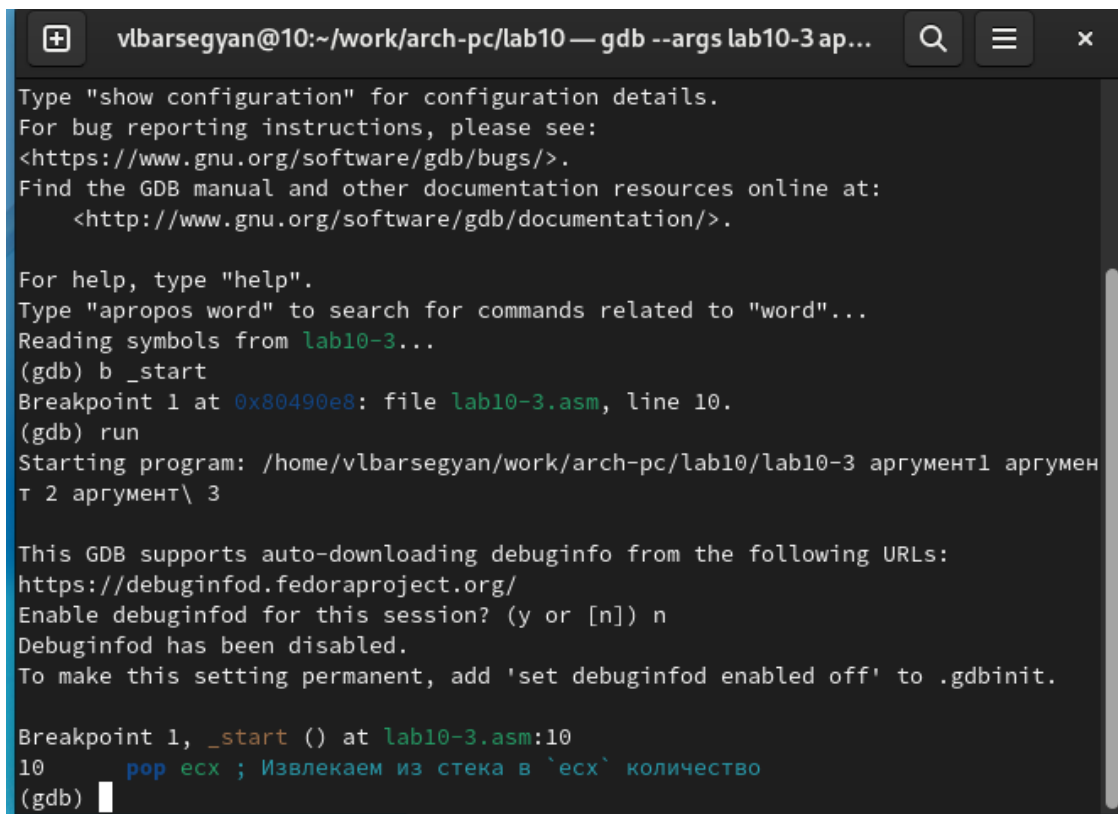


```
[vlbarsegyan@10 lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute 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 lab10-3...
(gdb) 
```

Рис. 24: Загрузка в отладчик

22. Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 25)



```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb --args lab10-3 ap...
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 lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 10.
(gdb) run
Starting program: /home/vlbarsegyan/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
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 lab10-3.asm:10
10      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) 
```

Рис. 25: Установка точки останова и запуск

23. Смотрю позиции стека по их адресам (рис. 26). Шаг изменения адреса равен 4, т.к. размер переменной - 4 байта

```
vlbarsegyan@10:~/work/arch-pc/lab10 — gdb --args lab10-3 ap...
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
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 lab10-3.asm:10
10      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd050:      0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd20f:      "/home/vlbarsegyan/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd23c:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd24e:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd25f:      "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd261:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 26: Позиции стека

Задания для самостоятельной работы

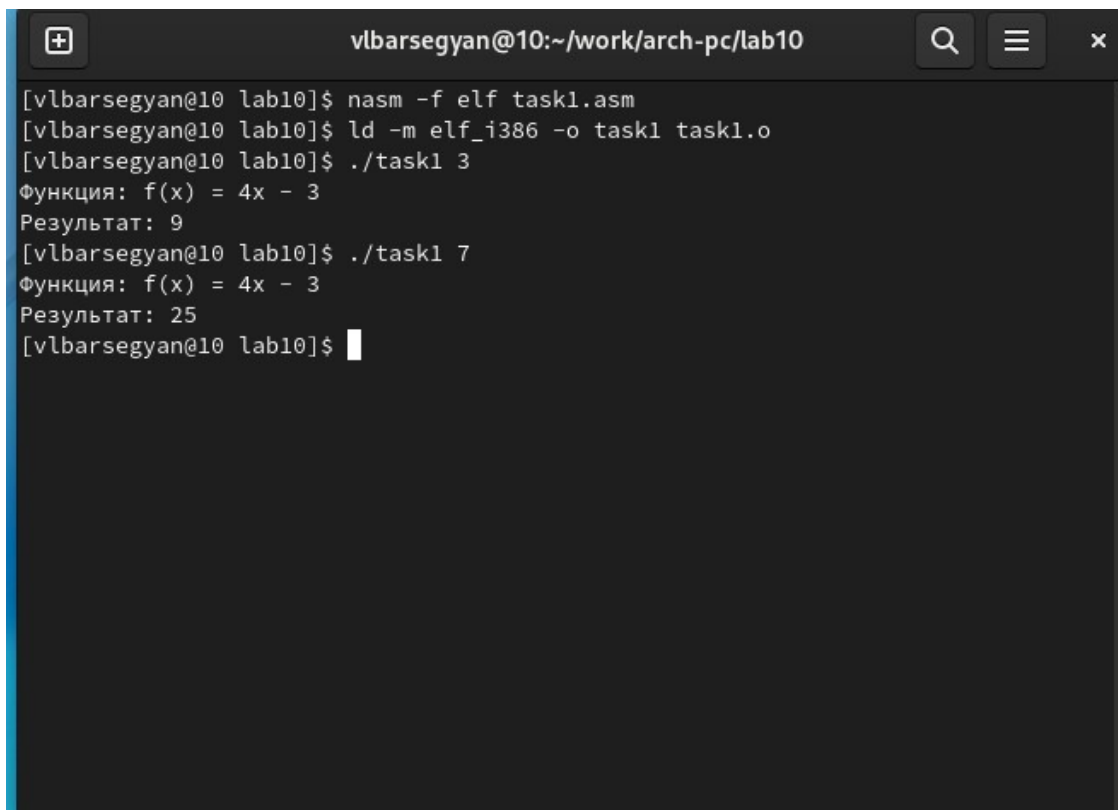
1. Создаю файл *task1.asm*, пишу текст программы (Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму) (рис. 27). Компилирую исполняемый файл и проверяю его работу (рис. 28)

```

6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, func
12 call sprintLF
13 pop ecx
14 pop edx
15 sub ecx, 1
16 mov esi, 4 ; Используем `esi` как множитель для x
17 mov edi, 0 ; в edi храним общую сумму
18
19 next:
20 cmp ecx, 0h
21 jz _end
22 pop eax
23 call atoi
24 call _calc
25 add edi, eax
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprint
31 mov eax, edi
32 call iprintLF
33 call quit
34
35 _calc:
36 mul esi
37 sub eax, 3
38 ret

```

Рис. 27: Текст программы



A terminal window with a dark background and light text. The title bar at the top shows the user 'vlbarsegyan@10' and the directory '~/work/arch-pc/lab10'. The terminal contains the following text:

```
[vlbarsegyan@10 lab10]$ nasm -f elf task1.asm
[vlbarsegyan@10 lab10]$ ld -m elf_i386 -o task1 task1.o
[vlbarsegyan@10 lab10]$ ./task1 3
Функция:  $f(x) = 4x - 3$ 
Результат: 9
[vlbarsegyan@10 lab10]$ ./task1 7
Функция:  $f(x) = 4x - 3$ 
Результат: 25
[vlbarsegyan@10 lab10]$
```

Рис. 28: Проверка программы

2. Создаю файл *task2.asm*, ввожу текст программы из листинга 10.4 (рис. 29)

report.md	task1.asm	task2.asm	x
-----------	-----------	-----------	---

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8
9 _start:
10 ; ---- Вычисление выражения (3+2)*4+5
11 mov ebx,3
12 mov eax,2
13 add ebx,eax
14 mov ecx,4
15 mul ecx
16 add ebx,5
17 mov edi,ebx
18
19 ; ---- Вывод результата на экран
20 mov eax,div
21 call sprint
22 mov eax,edi
23 call iprintLF
24
25 call quit
```

Рис. 29: Текст неправильной программы

3. Открываю отладчик GDB, отлаживаю код (рис. 30). В коде перепутаны местами регистры ebx и eax в команде add, а также в регистр edi копируются данные из регистра ebx, а не eax. Также 5 должно прибавлять к значению регистра eax, а не ebx

The screenshot shows a GDB terminal window with the title bar "vlbarsegyan@10:~/work/arch-pc/lab10 — gdb task2". The window is divided into several sections. At the top, the "Register group: general" is displayed, showing the values of registers eax (0x2), ecx (0x0), edx (0x0), ebx (0x3), and esp (0xffffd0c0). Below this, a list of assembly instructions is shown, with the instruction at address 0x80490f2, "add %eax,%ebx", highlighted. The instruction list includes: "B+ 0x80490e8 <_start> mov \$0x3,%ebx", "0x80490ed <_start+5> mov \$0x2,%eax", "> 0x80490f2 <_start+10> add %eax,%ebx", "0x80490f4 <_start+12> mov \$0x4,%ecx", "0x80490f9 <_start+17> mul %ecx", "0x80490fb <_start+19> add \$0x5,%ebx", and "0x80490fe <_start+22> mov %ebx,%edi". Below the assembly list, the status bar shows "native process 3056 In: _start L?? PC: 0x80490f2". A message states: "To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit." Below this, a breakpoint is listed: "Breakpoint 1, 0x080490e8 in _start ()". The GDB prompt "(gdb) si" is shown three times, indicating step-through execution, with the current instruction address updating from 0x080490ed to 0x080490f2. The prompt "(gdb) " is shown at the bottom.

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd0c0 0xffffd0c0

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
> 0x80490f2 <_start+10>  add    %eax,%ebx
0x80490f4 <_start+12>    mov    $0x4,%ecx
0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi

native process 3056 In: _start L?? PC: 0x80490f2
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, 0x080490e8 in _start ()
(gdb) si
0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb) 
```

Рис. 30: Отладка программы

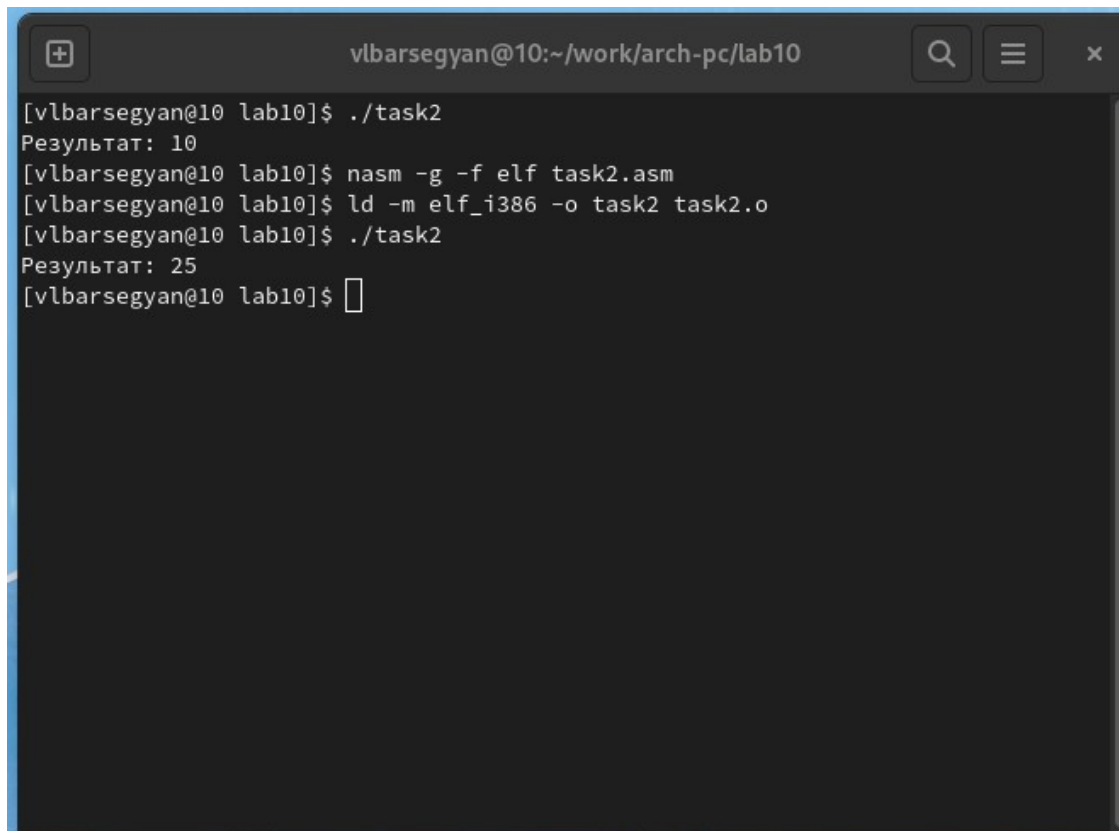
4. Исправленный код программы (рис. 31)

report.md	task1.asm	task2.asm	×
-----------	-----------	-----------	---

```
1  %include 'in_out.asm'
2
3  SECTION .data
4  div: DB 'Результат: ',0
5
6  SECTION .text
7  GLOBAL _start
8
9  _start:
10 ; ---- Вычисление выражения (3+2)*4+5
11 mov ebx,3
12 mov eax,2
13 add eax,ebx
14 mov ecx,4
15 mul ecx
16 add eax,5
17 mov edi,eax
18
19 ; ---- Вывод результата на экран
20 mov eax,div
21 call sprint
22 mov eax,edi
23 call iprintLF
24
25 call quit
```

Рис. 31: Исправленный код программы

5. Проверяю правильность работы программы (рис. 32)

A terminal window with a dark background and light text. The window title is 'vlbarsegyan@10:~/work/arch-pc/lab10'. It shows a sequence of commands: running './task2' which outputs 'Результат: 10', compiling 'task2.asm' to 'task2.o' using 'nasm', and linking 'task2.o' using 'ld'. A second run of './task2' outputs 'Результат: 25'.

```
[vlbarsegyan@10 lab10]$ ./task2
Результат: 10
[vlbarsegyan@10 lab10]$ nasm -g -f elf task2.asm
[vlbarsegyan@10 lab10]$ ld -m elf_i386 -o task2 task2.o
[vlbarsegyan@10 lab10]$ ./task2
Результат: 25
[vlbarsegyan@10 lab10]$
```

Рис. 32: Проверка работы программы

Выводы

Я научился писать программы с использованием подпрограмм, переписал старую программу с использованием подпрограммы. Узнал, что такое отладка и как пользоваться отладчиком GDB и познакомился с его основными возможностями