

Отчёт по лабораторной работе

Лабораторная №9

Полина Витальевна Барабаш

Содержание

<i>1</i>	<i>Цель работы</i>	<i>4</i>
<i>2</i>	<i>Выполнение работы</i>	<i>5</i>
<i>3</i>	<i>Выполнение самостоятельной работы</i>	<i>21</i>
<i>4</i>	<i>Выводы</i>	<i>28</i>

Список иллюстраций

2.1	Создание каталога и файла	5
2.2	Запуск программы с подпрограммой _calcul	6
2.3	Запуск измененной программы с использованием подпрограммы _subcalcul	6
2.4	Создание файла и запуск программы в оболочке GDB	7
2.5	Установка брейкпоинта, запуск программы, просмотр дисассимилированного кода программы	8
2.6	Вид дисассимилированного кода программы с Intel'овским синтаксисом	9
2.7	Режим псевдографики	10
2.8	Проверка установки точки останова	11
2.9	Установка новой точки останова по адресу инструкции и просмотр точек останова	12
2.10	Выполнение команды si	13
2.11	Просмотр значения переменной по имени и адресу	14
2.12	Замена символов с помощью set	15
2.13	Выведение значения регистра	16
2.14	Вывод замены значения регистра ebx	17
2.15	Завершение выполнения программы	18
2.16	Запуск программы с аргументами в GDB	19
2.17	Установка точки останова перед первой инструкцией и её запуск	19
2.18	Просмотр позиций стека	20
3.1	Работа программы с подпрограммой вычисления значения функции	22
3.2	Работа программы вычисления выражения	22
3.3	Открытие программы в GDB	23
3.4	Постановка точки останова, вывод дисассимилированного кода с Intel'овским синтаксисом	24
3.5	Включение режима псевдографики	25
3.6	Обнаруженная ошибка в программе	26
3.7	Неверные вычисления	27
3.8	Верная работа программы	27

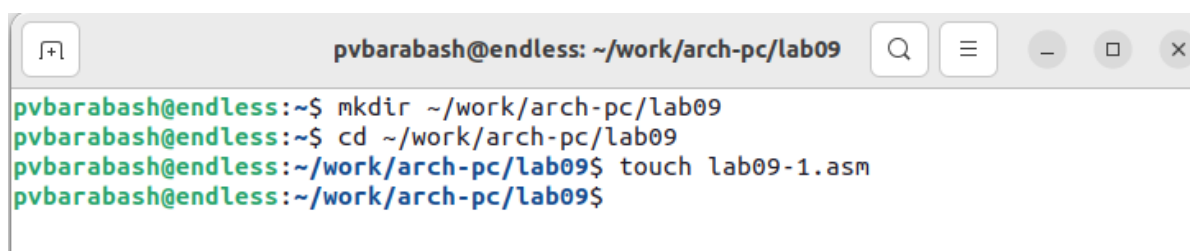
1 Цель работы

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

2 Выполнение работы

Задание №1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm.

Я создала каталог для программ лабораторной работы № 9 с помощью команды `mkdir ~/work/arch-pc/lab09`. Затем я перешла в него с помощью команды `cd` и создала файл lab09-1.asm с помощью `touch` (рис. 2.1).

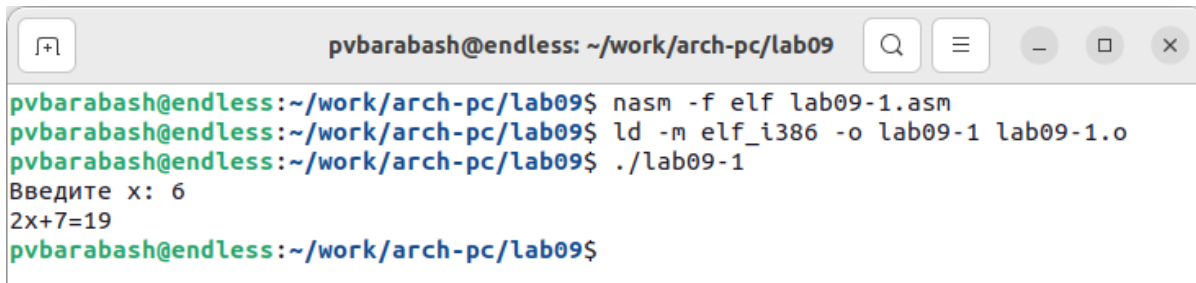


```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~$ mkdir ~/work/arch-pc/lab09
pvbarabash@endless:~$ cd ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ touch lab09-1.asm
pvbarabash@endless:~/work/arch-pc/lab09$
```

Рис. 2.1: Создание каталога и файла

Задание №2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.

Я внимательно изучила текст программы и ввела его в созданный файл. После этого создала исполняемый файл и проверила его работу (рис. 2.2).



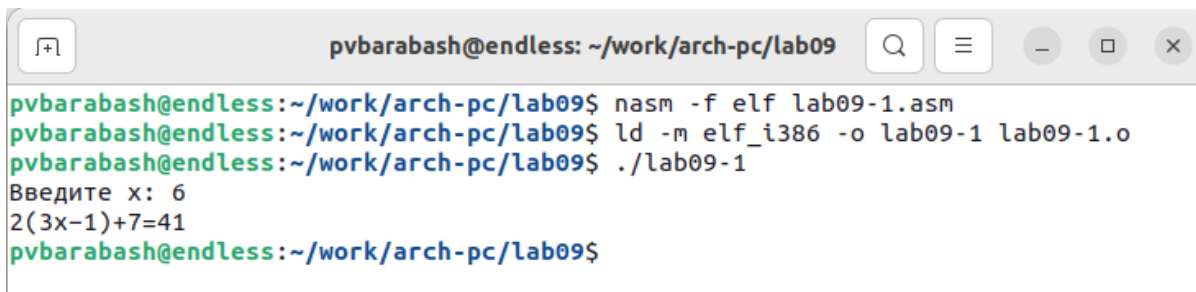
```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 6
2x+7=19
pvbarabash@endless:~/work/arch-pc/lab09$
```

Рис. 2.2: Запуск программы с подпрограммой _calcul

Программа работает корректно.

Задание №3. Измените текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $g(x)$, результат возвращается в _calcul и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран.

Я изменила текст программы, добавив подпрограмму _subcalcul. Я создала исполняемый файл и проверила его работу (рис. 2.3).



```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 6
2(3x-1)+7=41
pvbarabash@endless:~/work/arch-pc/lab09$
```

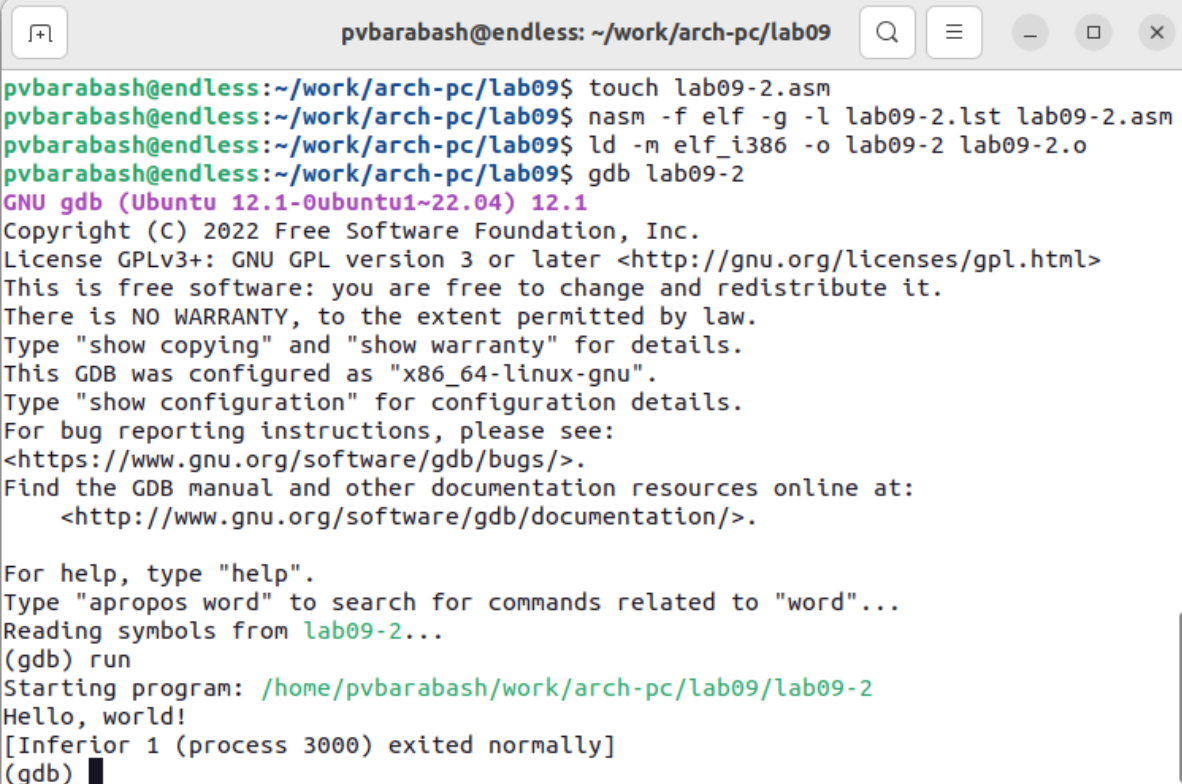
Рис. 2.3: Запуск измененной программы с использованием подпрограммы _subcalcul

Проверим вручную, какой результат должен был получиться при $x = 6$. $3 \cdot 6 - 1 = 17$, $2 \cdot 17 + 7 = 41$. Такой результат выдала и программа.

Задание №4. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. Получите исполняемый файл. Для работы с GDB в исполняемый файл необ-

ходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузите исполняемый файл в отладчик gdb. Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r).

Я создала файл lab09-2.asm и ввела в него текст программы листинга 9.2. Затем создала исполняемый файл, добавив ключом '-g' при трансляции и запустила его в оболочке GDB с помощью команды run (рис. 2.4).



```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ touch lab09-2.asm
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
pvbarabash@endless:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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
Starting program: /home/pvbarabash/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3000) exited normally]
(gdb)
```

Рис. 2.4: Создание файла и запуск программы в оболочке GDB

Программа работает и выводит Hello, world!.

Задание №5. Для более подробного анализа программы установите брейк-поинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor`

intel. Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включите режим псевдографики для более удобного анализа программы.

Я установила брейкпоинт на метку `_start` с помощью `break _start` и запустила программу с помощью `run`. После этого я посмотрела дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. Данные действия отображены на (рис. 2.5).



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

Breakpoint 1, _start () at lab09-2.asm:11
11      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) █
```

Рис. 2.5: Установка брейкпоинта, запуск программы, просмотр дисассимилированного кода программы

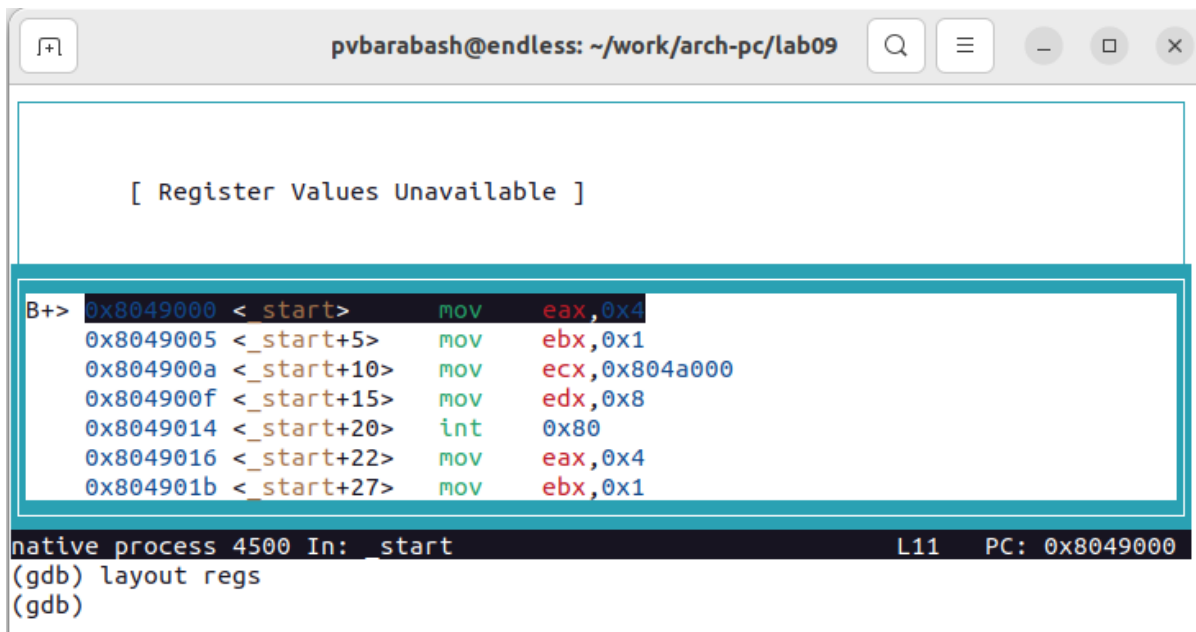
Я переключилась на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Посмотрела дисассимилированный код программы с помощью `disassemble` начиная с метки `_start` (рис. 2.6).


```
pvbarabash@endless: ~/work/arch-pc/lab09
(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. 2.6: Вид дисассимилированного кода программы с Intel'овским синтаксисом

Сравнивая два дисассимилированных кода, мы видим, что левые части одинаковы, а вот правые отличаются. Intel'овский вид нам привычнее, при команде `mov` мы видим сначала знакомые нам регистры `eax`, `ebx`, `ecx` и т.д., а затем идут адреса перемещаемых переменных. Вид АТТ сначала даёт адреса, а потом регистры.

Я включила режим псевдографики для более удобного анализа программы с помощью последовательного ввода команд `layout asm` и `layout regs`, действительно содержится три окна, как написано в руководстве (рис. 2.7).



The screenshot shows a GDB terminal window with the title bar "pvbarabash@endless: ~/work/arch-pc/lab09". The main display area shows "[Register Values Unavailable]". Below this, a list of assembly instructions is displayed, each with its address, a comment, and the instruction itself. The instructions are:

Address	Comment	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

At the bottom of the terminal, the status bar shows "native process 4500 In: _start L11 PC: 0x8049000". Below the status bar, the commands "(gdb) layout regs" and "(gdb)" are entered.

Рис. 2.7: Режим псевдографики

Задание №6. На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints`. Определите адрес предпоследней инструкции (`mov ebx, 0x0`) и установите точку останова. Посмотрите информацию о всех установленных точках останова.

С помощью команды `info breakpoints` я проверила, что точка останова поставлена на `_start` (рис. 2.8).

```
pvbarabash@endless: ~/work/arch-pc/lab09

[ Register Values Unavailable ]

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

native process 4500 In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num    Type             Disp Enb Address      What
1      breakpoint       keep y   0x08049000 lab09-2.asm:11
breakpoint already hit 1 time
(gdb) █
```

Рис. 2.8: Проверка установки точки останова

Затем я установила точку останова на предпоследнюю инструкцию по её адресу и проверила с помощью `i b`, что она установлена (рис. 2.9).

The screenshot shows a GDB terminal window with the following content:

```

pvbarabash@endless: ~/work/arch-pc/lab09
[ Register Values Unavailable ]

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
0x8049036 <_start+54>  int     0x80

native process 4500 In: start L11 PC: 0x8049000
(gdb) break *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
       breakpoint already hit 1 time
2      breakpoint      keep y   0x08049031 lab09-2.asm:24
(gdb)

```

Рис. 2.9: Установка новой точки останова по адресу инструкции и просмотр точек останова

Задание №7. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Посмотрите значение переменной `msg1` по имени. Посмотрите значение переменной `msg2` по адресу. Измените первый символ переменной `msg1`. Замените любой символ во второй переменной `msg2`. Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. С помощью команды `set` измените значение регистра `ebx` сначала на '2', а затем на 2. Объясните разницу вывода команд `p/s $ebx`. Завершите выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдите из GDB с помощью команды `quit` (сокращенно `q`).

Я выполнила 5 инструкций с помощью команды `si`, добавив к `si` аргумент 5. Были изменены регистры `eax`, `ebx`, `ecx` и `edx` (рис. 2.10).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd130 0xffffd130

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

native process 8736 In: _start L17 PC: 0x8049016
2 breakpoint keep y 0x08049031 lab09-2.asm:24
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:24
(gdb) si 5
(gdb)
```

Рис. 2.10: Выполнение команды si

Я посмотрела значение переменной msg1 по имени и значение переменной msg2 по адресу с помощью x/1sb (рис. 2.11).

```
pvbarabash@endless: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd130 0xffffd130

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

native process 2450 In: _start L17 PC: 0x8049016
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) █
```

Рис. 2.11: Просмотр значения переменной по имени и адресу

Я изменила первый символ переменной msg1 с 'H' на 'h'. Затем я заменила 4-ый символ во второй переменной msg2 с 'r' на 'R' (рис. 2.12).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd130 0xffffd130

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

native process 2450 In: _start L17 PC: 0x8049016
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a00a='R'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "woRld!\n\034"
(gdb) █
```

Рис. 2.12: Замена символов с помощью set

Я вывела в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 2.13).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd130 0xffffd130

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

native process 2450 In: start L17 PC: 0x8049016
0x804a008 <msg2>: "world!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/s $edx
$3 = 8
(gdb) █
```

Рис. 2.13: Выведение значения регистра

С помощью команды `set` я изменила значение регистра `ebx` сначала на '2', а затем на 2. Разница вывода возникает, потому что для '2' выводится код символа, а для 2 просто 2 (рис. 2.14).


```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd130 0xffffd130

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

native process 2450 In: _start L17 PC: 0x8049016
$7 = 8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb)
```

Рис. 2.14: Вывод замены значения регистра ebx

Я завершила выполнение программы с помощью команды continue и вышла из GDB с помощью команды quit (рис. 2.15).

```
pvbarabash@endless: ~/work/arch-pc/lab09
0x0804902a <+42>:    int     $0x80
0x0804902c <+44>:    mov     $0x1,%eax
0x08049031 <+49>:    mov     $0x0,%ebx
0x08049036 <+54>:    int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
    0x08049005 <+5>:    mov     ebx,0x1
    0x0804900a <+10>:   mov     ecx,0x804a000
    0x0804900f <+15>:   mov     edx,0x8
    0x08049014 <+20>:   int     0x80
    0x08049016 <+22>:   mov     eax,0x4
    0x0804901b <+27>:   mov     ebx,0x1
    0x08049020 <+32>:   mov     ecx,0x804a008
    0x08049025 <+37>:   mov     edx,0x7
    0x0804902a <+42>:   int     0x80
    0x0804902c <+44>:   mov     eax,0x1
    0x08049031 <+49>:   mov     ebx,0x0
    0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb) layout asm
pvbarabash@endless: ~/work/arch-pc/lab09$
```

Рис. 2.15: Завершение выполнения программы

Задание №8. Скопируйте файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создайте исполняемый файл. Загрузите исполняемый файл в отладчик, указав аргументы. Установите точку останова перед первой инструкцией в программе и запустите ее. Адрес вершины стека хранится в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы), число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрите остальные позиции стека. Объясните, почему шаг изменения адреса равен 4.

Я копировала файл lab8-2.asm в lab09 с именем lab09-3.asm. Затем создала исполняемый файл и загрузила исполняемый файл в отладчик, указав аргументы, добавив –args (рис. 2.16).

```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
pvbarabash@endless:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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...
(gdb)
```

Рис. 2.16: Запуск программы с аргументами в GDB

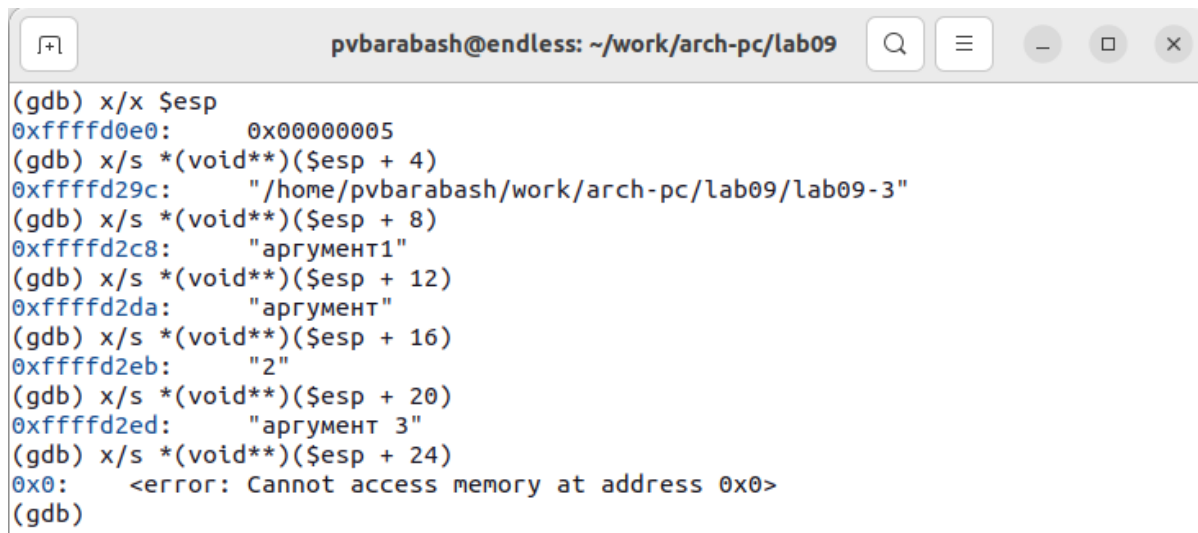
Я установила точку останова перед первой инструкцией в программе и запустила ее (рис. 2.17).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/pvbarabash/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3

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

Рис. 2.17: Установка точки останова перед первой инструкцией и её запуск

Я повторила команду `x/x $esp`, чтобы посмотреть количество аргументов. Затем я посмотрела остальные позиции стека (рис. 2.18).

A screenshot of a terminal window with a title bar. The title bar contains the text 'pvbarabash@endless: ~/work/arch-pc/lab09' and standard window control icons (search, menu, zoom, close). The terminal displays a series of GDB commands and their outputs. The commands are: (gdb) x/x \$esp, (gdb) x/s *(void**)(\$esp + 4), (gdb) x/s *(void**)(\$esp + 8), (gdb) x/s *(void**)(\$esp + 12), (gdb) x/s *(void**)(\$esp + 16), (gdb) x/s *(void**)(\$esp + 20), (gdb) x/s *(void**)(\$esp + 24), and (gdb) x/s *(void**)(\$esp + 28). The outputs show memory addresses and their corresponding values, which are pointers to string literals. The last command results in an error: 'error: Cannot access memory at address 0x0'.

```
(gdb) x/x $esp
0xffffd0e0: 0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd29c: "/home/pvbarabash/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd2c8: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd2da: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd2eb: "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd2ed: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.18: Просмотр позиций стека

Шаг равен 4, так как размер регистра равен 4 байтам.

3 Выполнение самостоятельной работы

Задание №1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

Нужно написать программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$, где в подпрограмму вынести вычисление $f(x)$ в подпрограмму. Значения x_i передаются как аргументы. Мой вариант второй. Следовательно мне необходимо написать подпрограмму вычисления функции $3 \cdot x - 1$.

Я скопировала файл lab8-1-iw.asm с названием lab09-1-iw.asm и преобразовала в нём программу. Я создала исполняемый файл и проверила работу программы на трёх наборах из предыдущей лабораторной, где проверила результаты вручную. Ответы совпадают, программа работает верно (рис. 3.1).



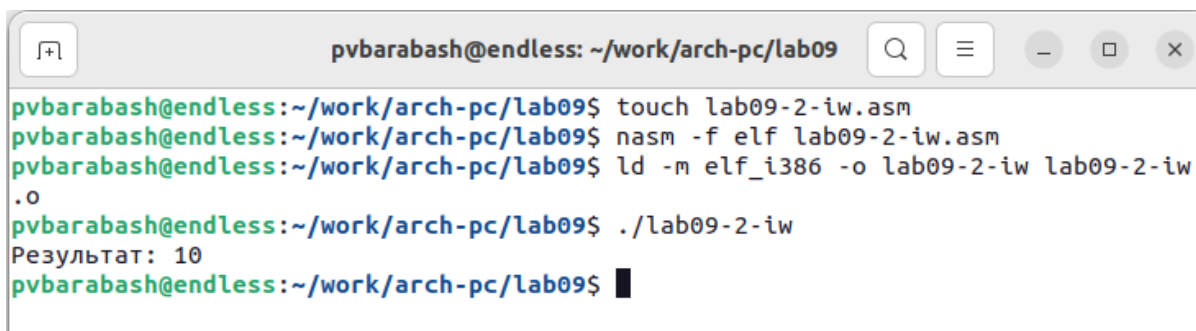
```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-1-iw.asm ~/work/arch-pc/lab09/lab09-1-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf lab09-1-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1-iw lab09-1-iw.o
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-1-iw 1 2 3 4
Функция: f(x) = 3*x-1
Результат: 26
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-1-iw 100 50 20 15 35
Функция: f(x) = 3*x-1
Результат: 655
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-1-iw 77 17 7
Функция: f(x) = 3*x-1
Результат: 300
pvbarabash@endless:~/work/arch-pc/lab09$
```

Рис. 3.1: Работа программы с подпрограммой вычисления значения функции

Задание №2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Значение выражения равно 25 ($5 * 4 = 20$, $20 + 5 = 25$).

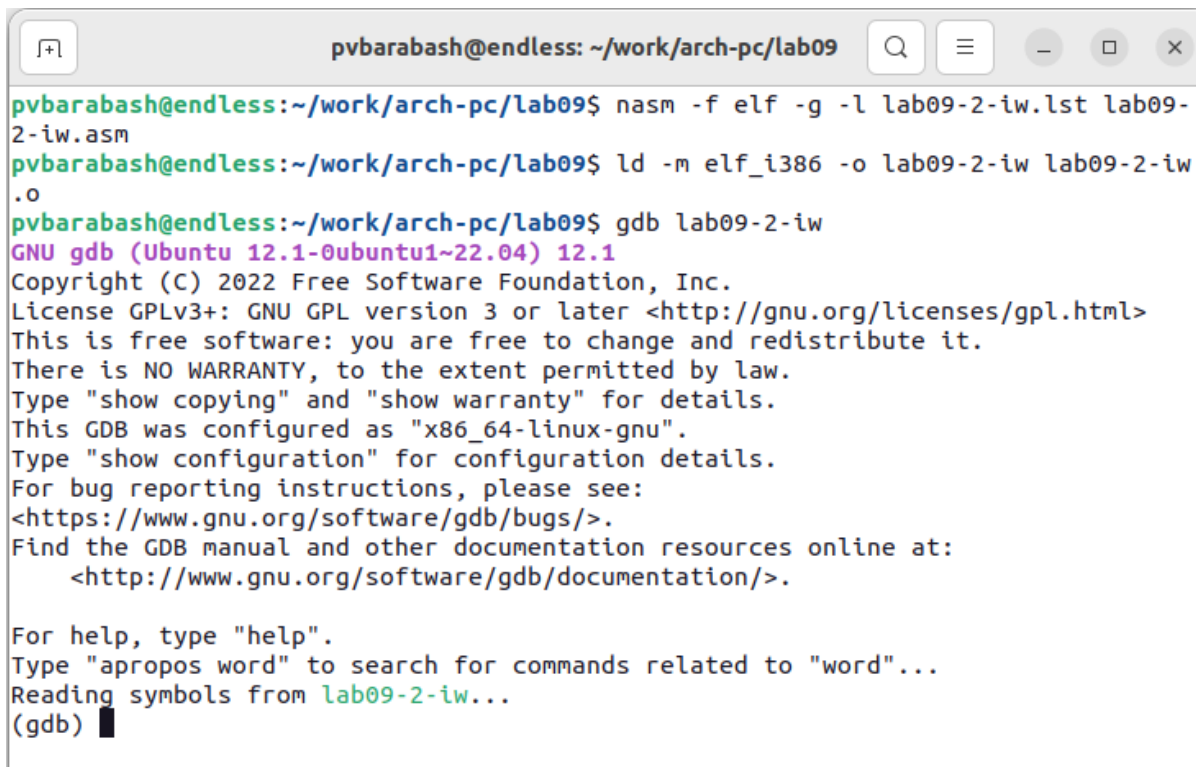
Я создала файл lab09-2-iw.asm и скопировала в него текст программы. Затем создала исполняемый файл и запустила его. Результат, выдаваемый программой, равен 10, он ошибочен (рис. 3.2).



```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ touch lab09-2-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf lab09-2-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2-iw lab09-2-iw.o
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-2-iw
Результат: 10
pvbarabash@endless:~/work/arch-pc/lab09$
```

Рис. 3.2: Работа программы вычисления выражения

Я открыла программу в GDB (рис. 3.3).



```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2-iw.lst lab09-2-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2-iw lab09-2-iw.o
pvbarabash@endless:~/work/arch-pc/lab09$ gdb lab09-2-iw
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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-iw...
(gdb) █
```


Рис. 3.3: Открытие программы в GDB

Запустила её, поставила точку останова на `_start`, затем переключилась на отображение команд с Intel'овским синтаксисом и посмотрела дисассимилированный код программы (рис. 3.4).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Reading symbols from lab09-2-iw...
(gdb) run
Starting program: /home/pvbarabash/work/arch-pc/lab09/lab09-2-iw
Результат: 10
[Inferior 1 (process 2713) exited normally]
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab09-2-iw.asm, line 11.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
   0x080490e8 <+0>:    mov     ebx,0x3
   0x080490ed <+5>:    mov     eax,0x2
   0x080490f2 <+10>:   add     ebx,eax
   0x080490f4 <+12>:   mov     ecx,0x4
   0x080490f9 <+17>:   mul     ecx
   0x080490fb <+19>:   add     ebx,0x5
   0x080490fe <+22>:   mov     edi,ebx
   0x08049100 <+24>:   mov     eax,0x804a000
   0x08049105 <+29>:   call    0x804900f <sprint>
   0x0804910a <+34>:   mov     eax,edi
   0x0804910c <+36>:   call    0x8049086 <iprintfLF>
   0x08049111 <+41>:   call    0x80490db <quit>
End of assembler dump.
(gdb) █
```

Рис. 3.4: Постановка точки останова, вывод дисассимилированного кода с Intel'овским синтаксисом

Затем я включила режим псевдографики (рис. 3.5).



```
pvbarabash@endless: ~/work/arch-pc/lab09

[ Register Values Unavailable ]

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

native process 2319 In: _start L11 PC: 0x80490e8
(gdb) layout regs
(gdb)
```

Рис. 3.5: Включение режима псевдографики

И начала последовательно идти по программе с помощью команды `si` и отслеживать изменение значений регистров. Я увидела, что инструкция `mul ecx` умножает на 4 регистр `eax`, а необходимый множитель находится не в `eax`, а в `ebx` (рис. 3.6).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8      Умножение eax
ecx      0x4      4
edx      0x0      0
ebx      0x5      5      В это время ebx остаётся
esp      0xffffd120 0xffffd120      результат сложения

0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
> 0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>

native process 2319 In: _start      L16      PC: 0x80490fb
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) □
```

Рис. 3.6: Обнаруженная ошибка в программе

Затем к `ebx` прибавляется 5. Так как в `ebx` результат сложения $3+2=5$, то и получается неверный ответ 10 (рис. 3.7).

```
pvbarabash@endless: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd120 0xffffd120

0x80490f2 <_start+10> add    ebx,eax
0x80490f4 <_start+12> mov    ecx,0x4
0x80490f9 <_start+17> mul    ecx
0x80490fb <_start+19> add    ebx,0x5
> 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>

native process 2319 In: _start L17 PC: 0x80490fe
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 3.7: Неверные вычисления

Автором программы не было учтено, что `mul` умножает регистр `eax`. Я исправила код программы, создала исполняемый файл и проверила её работу (рис. 3.8).

```
pvbarabash@endless: ~/work/arch-pc/lab09
pvbarabash@endless:~/work/arch-pc/lab09$ nasm -f elf lab09-2-iw.asm
pvbarabash@endless:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2-iw lab09-2-iw.o
pvbarabash@endless:~/work/arch-pc/lab09$ ./lab09-2-iw
Результат: 25
pvbarabash@endless:~/work/arch-pc/lab09$
```

Рис. 3.8: Верная работа программы

4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм. Также я познакомилась с методами отладки при помощи GDB и его основными возможностями, применив новые знания на практике для нахождения ошибки в программе.