

# **Отчет по лабораторной работе №10**

**Дисциплина: Архитектура компьютера**

**Ищенко Ирина Олеговна**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выполнение заданий для самостоятельной работы</b>	<b>18</b>
<b>4</b>	<b>Выводы</b>	<b>23</b>

## Список иллюстраций

2.1	Создание каталога и файла . . . . .	6
2.2	Запуск программы вычисления выражения . . . . .	7
2.3	Запуск программы вычисления выражения с использованием под- программы . . . . .	9
2.4	Программа печати сообщения Hello world! . . . . .	10
2.5	Программа печати сообщения Hello world! . . . . .	10
2.6	Установка брейкпоинта . . . . .	11
2.7	Дисассимилированный код программы . . . . .	11
2.8	Режим псевдографики . . . . .	12
2.9	Брейкпоинты . . . . .	12
2.10	Команда si . . . . .	13
2.11	Команда si . . . . .	13
2.12	Команда si . . . . .	14
2.13	Команда si . . . . .	14
2.14	Команда si . . . . .	15
2.15	Значения переменных . . . . .	15
2.16	Изменение значений переменных . . . . .	15
2.17	Значение регистра в разных форматах . . . . .	16
2.18	Значение регистра ebx . . . . .	16
2.19	Программа вывода аргументов . . . . .	17
2.20	Позиции стека . . . . .	17
3.1	Программа вычисления значения функции . . . . .	19
3.2	Неправильный вывод программы . . . . .	20
3.3	Поиск ошибки . . . . .	21
3.4	Правильный вывод . . . . .	22

## Список таблиц

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы № 10, перейдем в него и создадим файл lab10-1.asm (рис. 2.1).

```
ioithenko@fedora ~]$ mkdir ~/work/arch-pc/lab10
ioithenko@fedora ~]$ cd ~/work/arch-pc/lab10
ioithenko@fedora lab10]$ touch lab10-1.asm
```

Рис. 2.1: Создание каталога и файла

Введем в файл lab10-1.asm текст программы из листинга 1. Создадим исполняемый файл и проверим его работу (рис. 2.2). Листинг 2:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
```

```

call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы

```



```

[ioithenko@fedora lab10]$ nasm -f elf lab10-1.asm
[ioithenko@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[ioithenko@fedora lab10]$ ./lab10-1
Введите x: 3
2x+7=13
[ioithenko@fedora lab10]$

```

Рис. 2.2: Запуск программы вычисления выражения

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, согласно листингу 2. Создадим исполняемый файл и проверим его работу (рис. 2.3). Листинг 2:

```

%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7=',0

SECTION .bss
x: RESB 80
rez: 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 ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[rez]
call iprintLF
call quit

_calcul:
call _subcalcul
mov ebx,2

```



```

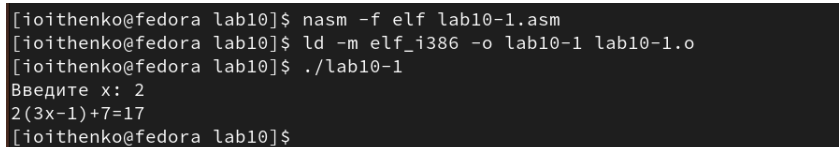
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы

```

```

_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret

```



```

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

```

Рис. 2.3: Запуск программы вычисления выражения с использованием подпрограммы

Создадим файл lab10-2.asm с текстом программы из листинга 3. Получим исполняемый файл. Загрузим исполняемый файл в отладчик gdb (рис. 2.4). Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.5). Листинг 3:

```

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

```

```

[ioithenko@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[ioithenko@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[ioithenko@fedora 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.ht
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

```

Рис. 2.4: Программа печати сообщения Hello world!

```

(gdb) run
Starting program: /home/ioithenko/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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 3243) exited normally]

```

Рис. 2.5: Программа печати сообщения Hello world!

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 2.6).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/ioithenko/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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 2.6: Установка брейкпоинта

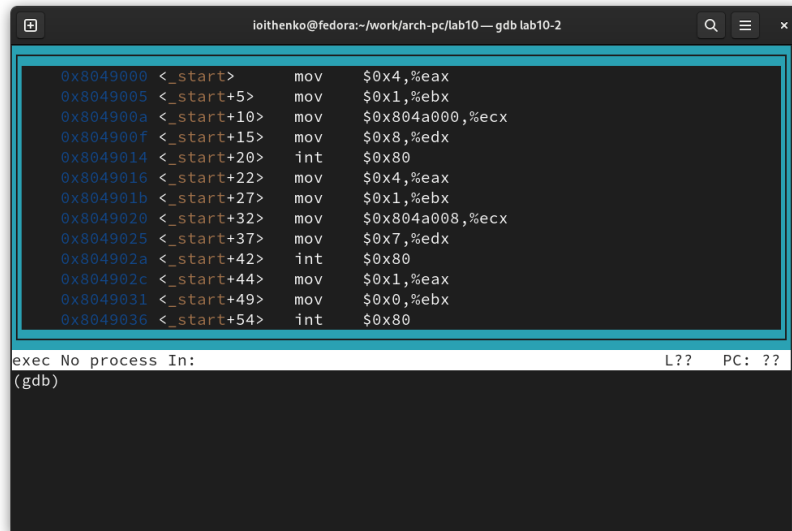
Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 2.7).

```
(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. 2.7: Дисассимилированный код программы

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Код Intel опускает символ `'%'` перед именами регистров, инструкции с несколькими операндами перечисляют их в обратном порядке.

Включим режим псевдографики для более удобного анализа программы (рис. 2.8).

A screenshot of a GDB window titled 'ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2'. The main pane displays assembly code in pseudo-graphic mode, with addresses, disassembled instructions, and comments. The code is as follows:

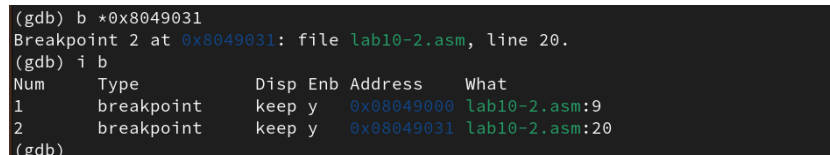
```
0x8049000 <_start>      mov     $0x4,%eax
0x8049005 <_start+5>      mov     $0x1,%ebx
0x804900a <_start+10>     mov     $0x804a000,%ecx
0x804900f <_start+15>     mov     $0x8,%edx
0x8049014 <_start+20>     int      $0x80
0x8049016 <_start+22>     mov     $0x4,%eax
0x804901b <_start+27>     mov     $0x1,%ebx
0x8049020 <_start+32>     mov     $0x804a008,%ecx
0x8049025 <_start+37>     mov     $0x7,%edx
0x804902a <_start+42>     int      $0x80
0x804902c <_start+44>     mov     $0x1,%eax
0x8049031 <_start+49>     mov     $0x0,%ebx
0x8049036 <_start+54>     int      $0x80
```

The bottom status bar shows 'exec No process in: L?? PC: ??' and '(gdb)'.

Address	Disassembly	Comment
0x8049000	mov \$0x4,%eax	<_start>
0x8049005	mov \$0x1,%ebx	<_start+5>
0x804900a	mov \$0x804a000,%ecx	<_start+10>
0x804900f	mov \$0x8,%edx	<_start+15>
0x8049014	int \$0x80	<_start+20>
0x8049016	mov \$0x4,%eax	<_start+22>
0x804901b	mov \$0x1,%ebx	<_start+27>
0x8049020	mov \$0x804a008,%ecx	<_start+32>
0x8049025	mov \$0x7,%edx	<_start+37>
0x804902a	int \$0x80	<_start+42>
0x804902c	mov \$0x1,%eax	<_start+44>
0x8049031	mov \$0x0,%ebx	<_start+49>
0x8049036	int \$0x80	<_start+54>

Рис. 2.8: Режим псевдографики

Проверим наличие брейкпоинтов с помощью команды `info breakpoints` и установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова. Посмотрим информацию о всех установленных точках останова (рис. 2.9).

A screenshot of a GDB window showing the output of the `info breakpoints` command. The output is as follows:

```
(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)
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x08049000	lab10-2.asm:9
2	breakpoint	keep	y	0x08049031	lab10-2.asm:20

Рис. 2.9: Брейкпоинты

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров (рис. 2.10), (рис. 2.11), (рис. 2.12), (рис. 2.13) и (рис. 2.14). Команда `si` делает шаг от брейкпоинта и показывает значение регистра согласно строке.

```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd200 0xffffd200

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

native process 2866 In: _start L10 PC: 0x8049005
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/ioithenko/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb) si
(gdb)
```

Рис. 2.10: Команда si

```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffd200 0xffffd200

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

native process 2866 In: _start L11 PC: 0x804900a
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/ioithenko/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.11: Команда si

```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffd200 0xffffd200

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

native process 2866 In: _start L12 PC: 0x804900f
Start it from the beginning? (y or n) yStarting program: /home/ioithenko/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.12: Команда si

```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2

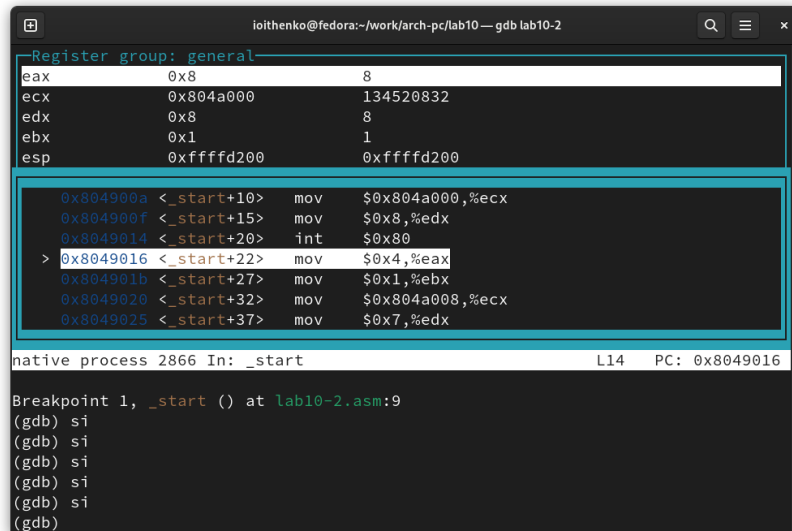
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd200 0xffffd200

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

native process 2866 In: _start L13 PC: 0x8049014
ch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.13: Команда si



```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb lab10-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd200 0xffffd200

0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
> 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx

native process 2866 In: _start L14 PC: 0x8049016
Breakpoint 1, _start () at lab10-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.14: Команда si

Посмотрим значение переменной msg1 по имени и значение переменной msg2 по адресу (рис. 2.15).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
```

Рис. 2.15: Значения переменных

Изменим первый символ переменной msg1. Заменим любой символ во второй переменной msg2 (рис. 2.16).

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

Рис. 2.16: Изменение значений переменных

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 2.17).

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

Рис. 2.17: Значение регистра в разных форматах

С помощью команды `set` изменим значение регистра `ebx` (рис. 2.18). Разница вывода команда заключается в том, что в первом случае мы вводим 2 как символ.

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

Рис. 2.18: Значение регистра `ebx`

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`).

Скопируем файл `lab9-2.asm`, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки в файл с именем `lab10-3.asm`. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 2.19).



```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb --args lab10-3 аргумент1 аргумент2 аргумент3
(gdb) layout asm
[ioithenko@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
[ioithenko@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[ioithenko@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[ioithenko@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент2 '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) b _start
```

Рис. 2.19: Программа вывода аргументов

Установим точку брейкпоинта. Посмотрим позиции стека (рис. 2.20). Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.), так как шаг равен размеру переменной - 4 байтам.

```
ioithenko@fedora:~/work/arch-pc/lab10 — gdb --args lab10-3 аргумент1 аргумент2 аргумент3
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
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.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1c0: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd374: "/home/ioithenko/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd39f: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd3b1: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd3c2: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd3c4: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.20: Позиции стека

## 3 Выполнение заданий для самостоятельной работы

Преобразуем программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции как подпрограмму, согласно листингу 4 (рис. 3.1). Листинг 4:

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0
fx: db 'f(x)=5(x+2) ',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
```

```

jz _end
pop eax
call atoi
call calc
add esi, eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

calc:
add eax, 2
mov ebx, 5
mul ebx
ret

```

```

[ioithenko@fedora lab10]$ nasm -f elf -g -l lab10-task1.lst lab10-task1.asm
[ioithenko@fedora lab10]$ ld -m elf_i386 -o lab10-task1 lab10-task1.o
[ioithenko@fedora lab10]$ ./lab10-task1 1 5 6
f(x)=5(x+2)
Результат: 90
[ioithenko@fedora lab10]$

```

Рис. 3.1: Программа вычисления значения функции

В листинге 5 приведена программа вычисления выражения  $(3+2)*4+5$ . При запуске данная программа дает неверный результат. Проверим это (рис. 3.2). С помощью отладчика GDB, анализируя изменения значений регистров (рис. 3.3), определим ошибку: перепутан порядок аргументов у инструкции `add` и

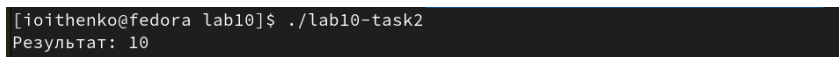
по окончании работы в `edi` отправляется регистр `ebx` вместо `eax`. Исправим ее согласно листингу 6 (рис. 3.4). Листинг 5:

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start

_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```



```
[ioithenko@fedora lab10]$ ./lab10-task2
Результат: 10
```

Рис. 3.2: Неправильный вывод программы

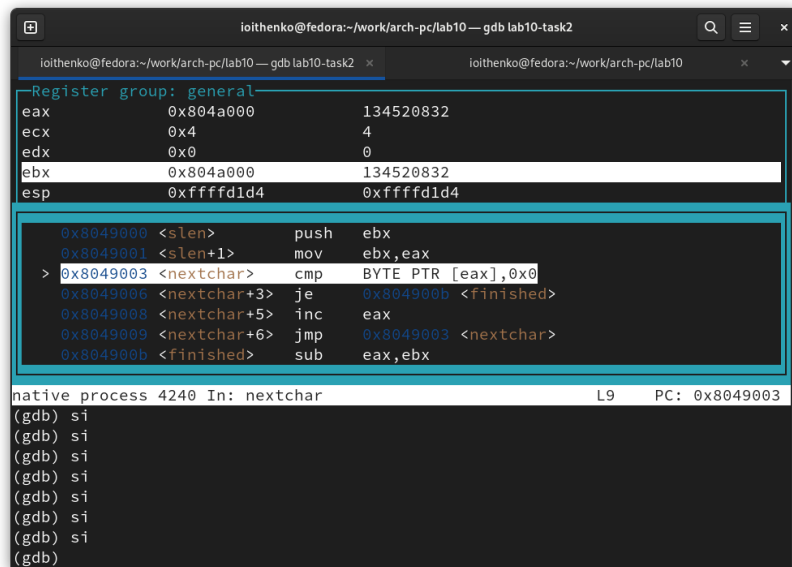


Рис. 3.3: Поиск ошибки

Листинг 6:

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
```

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

```
[ioithenko@fedora lab10]$ nasm -f elf -g -l lab10-task2.lst lab10-task2.asm
[ioithenko@fedora lab10]$ ld -m elf_i386 -o lab10-task2 lab10-task2.o
[ioithenko@fedora lab10]$ ./lab10-task2
Результат: 25
[ioithenko@fedora lab10]$
```

Рис. 3.4: Правильный вывод

## 4 Выводы

В ходе лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.