

# **Отчёт по лабораторной работе №10**

**Понятие подпрограммы. Отладчик GDB.**

Михаил Александрович Мелкомуков

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Задание</b>	<b>7</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
3.1	Реализация подпрограмм в NASM . . . . .	8
3.2	Отладка программ с помощью GDB . . . . .	11
3.3	Добавление точек останова . . . . .	14
3.4	Работа с данными программы в GDB . . . . .	15
3.5	Обработка аргументов командной строки в GDB . . . . .	23
<b>4</b>	<b>Задание для самостоятельной работы</b>	<b>25</b>
<b>5</b>	<b>Выводы</b>	<b>32</b>

## Список иллюстраций

3.1	Создали каталог для выполнения лабораторной работы №10, перешли в него и создали файл lab10-1.asm . . . . .	8
3.2	Ввели в файл lab10-1.asm текст программы . . . . .	9
3.3	Создали исполняемый файл и проверили его работу . . . . .	9
3.4	Изменили текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul . . . . .	10
3.5	Создали исполняемый файл и проверили его работу . . . . .	10
3.6	Создали файл lab10-2.asm и ввели в него текст программы . . . .	11
3.7	Получили исполняемый файл. Чтобы работать с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом '-g' . . . . .	11
3.8	Загрузили исполняемый файл в отладчик gdb . . . . .	12
3.9	Проверили работу программы, запустив ее в оболочке GDB с помощью команды run . . . . .	12
3.10	Для более подробного анализа программы установили брейкпоинт на метку _start и запустили её . . . . .	12
3.11	Посмотрели дисассимилированный код программы с помощью команды disassemble . . . . .	13
3.12	На этот раз дисассимилированный код выглядит следующим образом	13
3.13	Включили режим псевдографики для более удобного анализа программы. Для этого ввели команды: layout asm и layout regs . . . .	14
3.14	С помощью команды info breakpoints проверили, что на одном из предыдущих шагов была установлена точка останова . . . . .	14
3.15	Установили еще одну точку останова по адресу инструкции. Для этого перед адресом добавляем '*' . . . . .	15
3.16	Посмотрели информацию о всех установленных точках останова с помощью команды i b (info breakpoints . . . . .	15
3.17	eax присваивается значение 4 . . . . .	16
3.18	ebx присваивается значение 1 . . . . .	17
3.19	ecx присваивается значение 134520832 . . . . .	18
3.20	edx присваивается значение 8 . . . . .	19
3.21	eax увеличивается на 4 . . . . .	20
3.22	Посмотрели содержимое регистров с помощью команды info registers (коротко i r) . . . . .	20
3.23	С помощью команды x/1sb &msg1 посмотрели значение переменной msg1 по имени . . . . .	21

3.24	Посмотрели значение переменной msg2 по адресу. Адрес переменной определили по дизассемблированной инструкции . . . . .	21
3.25	С помощью команды x/1sb &msg1 посмотрели значение переменной msg1 . . . . .	21
3.26	Результат изменений переменной msg2 . . . . .	21
3.27	Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. Для этого использовали команды p/x, p/t и p/s соответственно. Перед именем регистра ставим префикс '\$' . . . . .	22
3.28	С помощью команды set изменили значение регистра ebx . . . .	22
3.29	Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9 в файл с именем lab10-3.asm и создали исполняемый файл . . . . .	23
3.30	Для загрузки в gdb программы с аргументами использовали ключ -args. Загрузили исполняемый файл в отладчик и указали аргументы	23
3.31	Установили точку останова перед первой инструкцией в программе и запустили ее . . . . .	24
3.32	Посмотрели на число аргументов командной строки . . . . .	24
3.33	Посмотрели остальные позиции стека . . . . .	24
4.1	Скопировали программу из лабораторной работы №9 . . . . .	25
4.2	Реализовали вычисление значения функции как подпрограмму .	26
4.3	Создали исполняемый файл и проверили его работу . . . . .	26
4.4	Создали файл lab10-5.asm и ввели в него текст программы . . . .	27
4.5	Первая ошибка: сумма элементов eax и ebx записывается в ebx . .	28
4.6	Вторая ошибка: при умножении на 5 меняется не eax, а ebx . . . .	29
4.7	Третья ошибка: выводится значение ebx, а не eax . . . . .	30
4.8	Определили ошибки и исправили программу . . . . .	31
4.9	Создали исполняемый файл и проверили его работу. Ошибка устранена . . . . .	31

## Список таблиц

# 1 Цель работы

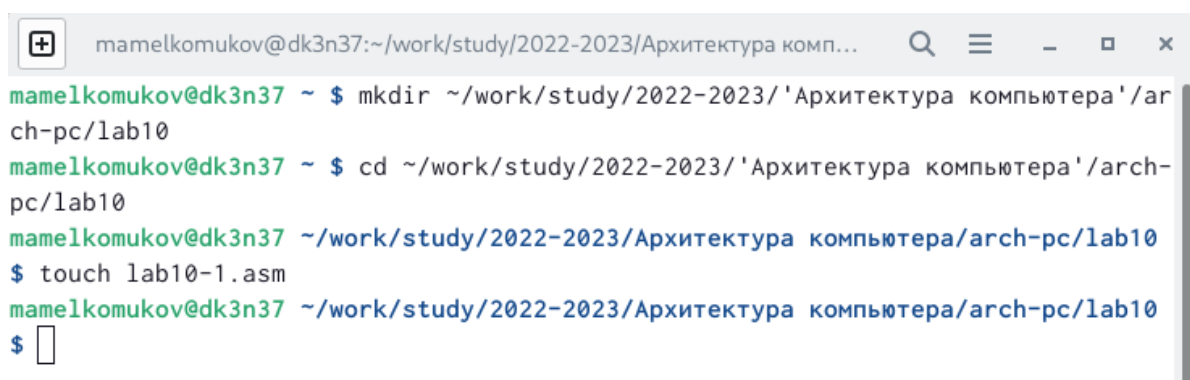
Целью работы является приобретение навыков написания программ с использованием подпрограмм а также знакомство с методами отладки при помощи GDB и его основными возможностями.

## 2 Задание

Узнать понятие подпрограммы, познакомиться с инструкциями `call` и `ret`. Ознакомиться с основными возможностями отладчика GDB: как запускать отладчик и выходить из него, как дизассемблировать программы, как устанавливать точки останова, как осуществлять пошаговую отладку, как работать с данными программы в GDB. На практике применить полученные знания.

## 3 Выполнение лабораторной работы

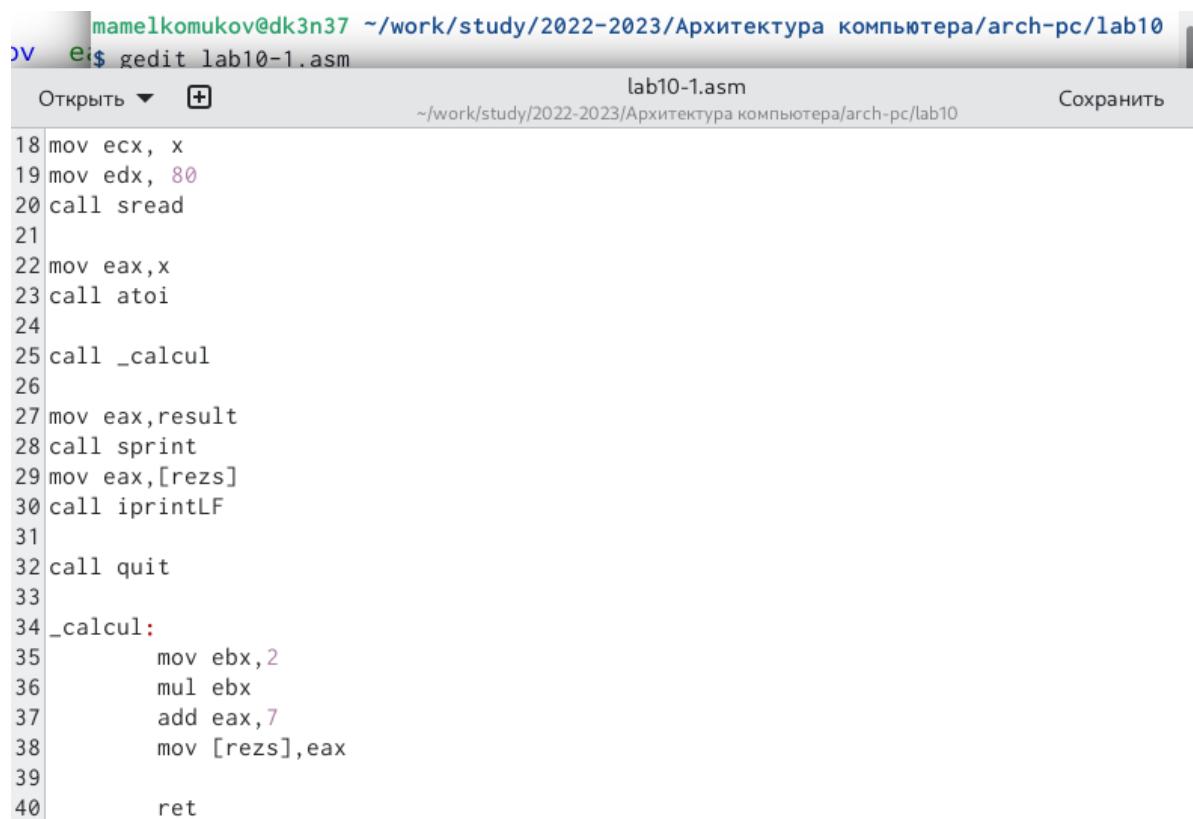
### 3.1 Реализация подпрограмм в NASM



```
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
mamelkomukov@dk3n37 ~ $ mkdir ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab10
mamelkomukov@dk3n37 ~ $ cd ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab10
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ touch lab10-1.asm
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$
```

Рис. 3.1: Создали каталог для выполнения лабораторной работы №10, перешли в него и создали файл lab10-1.asm

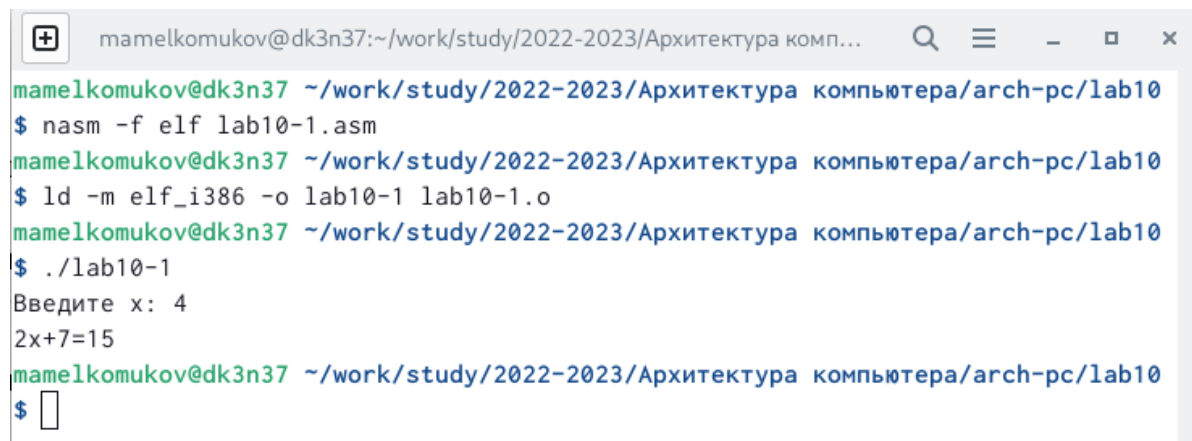




The screenshot shows a gedit editor window titled 'lab10-1.asm'. The menu bar includes 'Открыть', a '+' icon, the filename 'lab10-1.asm', the path '~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10', and 'Сохранить'. The code is as follows:

```
18 mov ecx, x
19 mov edx, 80
20 call sread
21
22 mov eax, x
23 call atoi
24
25 call _calcul
26
27 mov eax, result
28 call sprint
29 mov eax, [rezs]
30 call iprintLF
31
32 call quit
33
34 _calcul:
35     mov ebx, 2
36     mul ebx
37     add eax, 7
38     mov [rezs], eax
39
40     ret
```

Рис. 3.2: Ввели в файл lab10-1.asm текст программы



The screenshot shows a terminal window with the following commands and output:

```
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
$ nasm -f elf lab10-1.asm
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ld -m elf_i386 -o lab10-1 lab10-1.o
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ./lab10-1
Введите x: 4
2x+7=15
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$
```

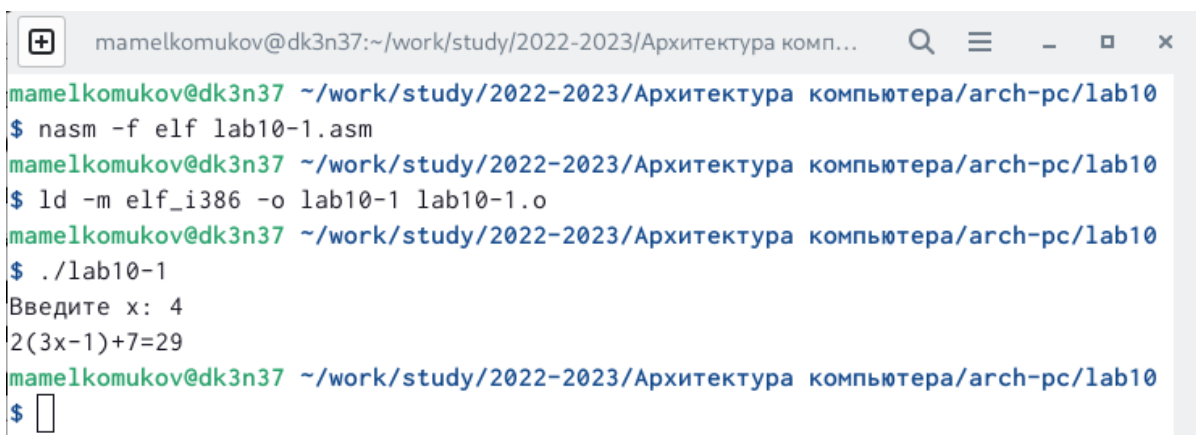
Рис. 3.3: Создали исполняемый файл и проверили его работу

```

25
26 mov eax,result
27 call sprint
28 mov eax,[rezs]
29 call iprintLF
30
31 call quit
32
33 _subcalcul:
34     mov ebx,3
35     mul ebx
36     sub eax,1
37
38     ret
39
40 _calcul:
41     call _subcalcul
42     mov ebx,2
43     mul ebx
44     add eax,7
45     mov [rezs],eax
46
47     ret

```

Рис. 3.4: Изменили текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`



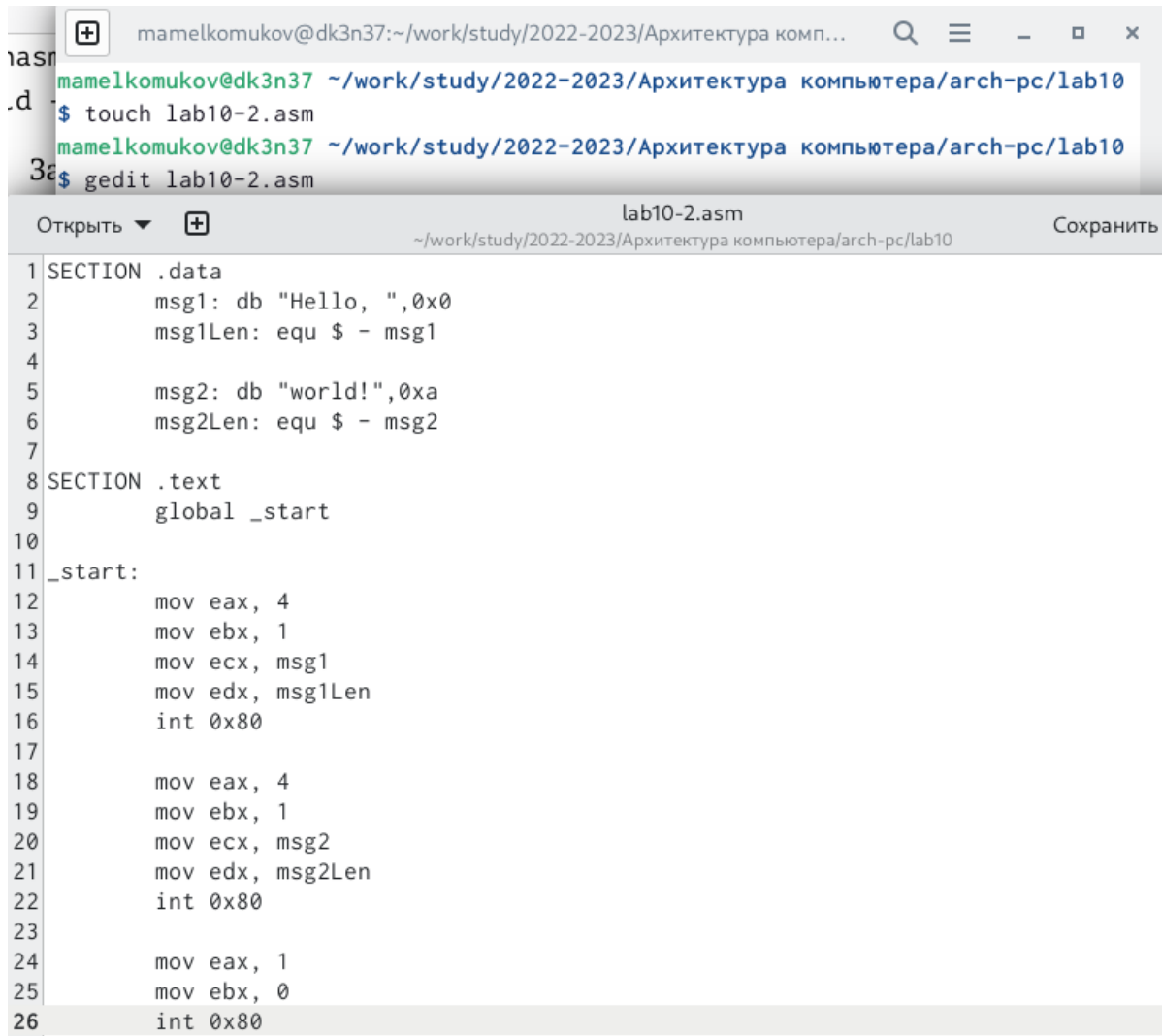
```

mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ nasm -f elf lab10-1.asm
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ld -m elf_i386 -o lab10-1 lab10-1.o
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ./lab10-1
Введите x: 4
2(3x-1)+7=29
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ 

```

Рис. 3.5: Создали исполняемый файл и проверили его работу

## 3.2 Отладка программ с помощью GDB



```
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ touch lab10-2.asm
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ gedit lab10-2.asm

lab10-2.asm
~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
Сохранить

1 SECTION .data
2     msg1: db "Hello, ",0x0
3     msg1Len: equ $ - msg1
4
5     msg2: db "world!",0xa
6     msg2Len: equ $ - msg2
7
8 SECTION .text
9     global _start
10
11 _start:
12     mov eax, 4
13     mov ebx, 1
14     mov ecx, msg1
15     mov edx, msg1Len
16     int 0x80
17
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg2
21     mov edx, msg2Len
22     int 0x80
23
24     mov eax, 1
25     mov ebx, 0
26     int 0x80
```

Рис. 3.6: Создали файл lab10-2.asm и ввели в него текст программы

```
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ld -m elf_i386 -o lab10-2 lab10-2.o
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$
```

Рис. 3.7: Получили исполняемый файл. Чтобы работать с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом '-g'

```
mamelkomukov@dk3n37 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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. 3.8: Загрузили исполняемый файл в отладчик gdb

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamelkomukov/work/study/2022-
2023/Архитектура компьютера/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3676) exited normally]
(gdb) 
```

Рис. 3.9: Проверили работу программы, запустив ее в оболочке GDB с помощью команды run

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamelkomukov/work/study/2022-
2023/Архитектура компьютера/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
```

Рис. 3.10: Для более подробного анализа программы установили брейкпоинт на метку \_start и запустили её

```
(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. 3.11: Посмотрели дисассимилированный код программы с помощью команды `disassemble`

Переключились на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`, после чего снова посмотрели дисассимилированный код программы с помощью команды `disassemble`.

```
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. 3.12: На этот раз дисассимилированный код выглядит следующим образом

В режиме АТТ непосредственные операнды пишутся после '\$', в то время как

в режиме Intel операнды не выделяются. Регистровые операнды АТТ пишутся после '%', регистровые операнды Intel не выделяются.

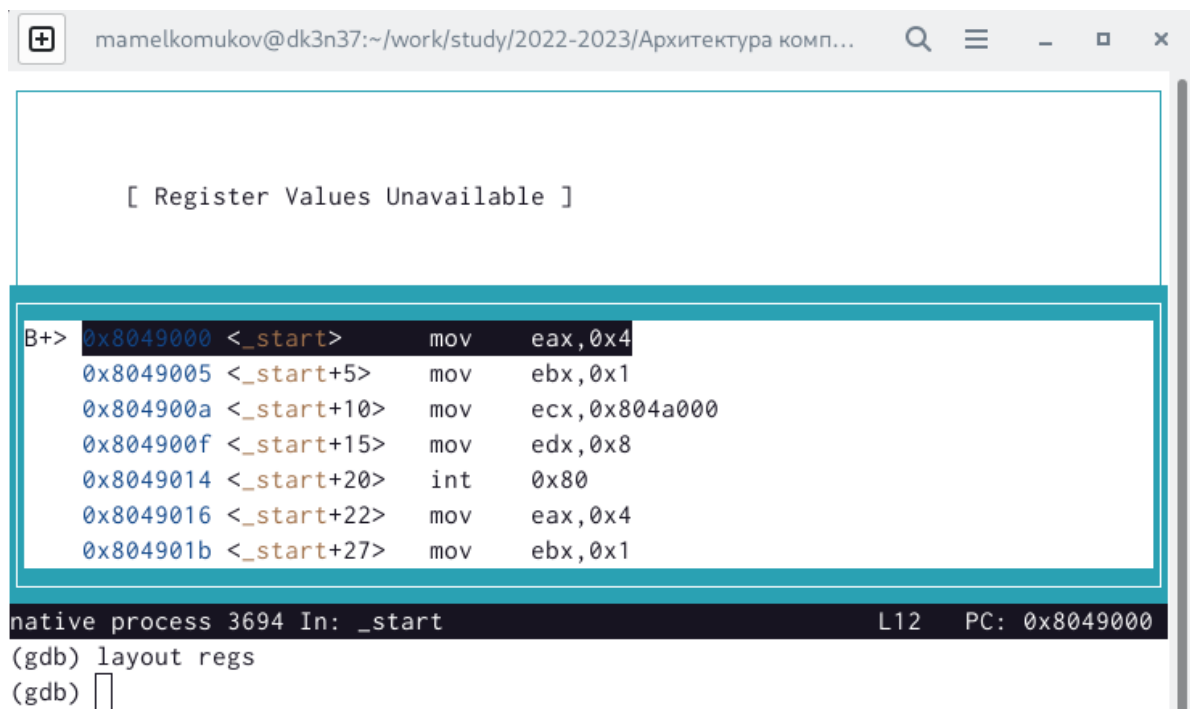


Рис. 3.13: Включили режим псевдографики для более удобного анализа программы. Для этого ввели команды: layout asm и layout regs

### 3.3 Добавление точек останова

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:12
breakpoint already hit 1 time
(gdb)
```

Рис. 3.14: С помощью команды info breakpoints проверили, что на одном из предыдущих шагов была установлена точка останова

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) █
```

Рис. 3.15: Установили еще одну точку останова по адресу инструкции. Для этого перед адресом добавляем '\*'

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab10-2.asm:12
2        breakpoint      keep y  0x08049031 lab10-2.asm:25
(gdb) █
```

Рис. 3.16: Посмотрели информацию о всех установленных точках останова с помощью команды i b (info breakpoints)

## 3.4 Работа с данными программы в GDB

Выполнили 5 инструкций с помощью команды si (stepi) и проследили за изменением значений регистров.

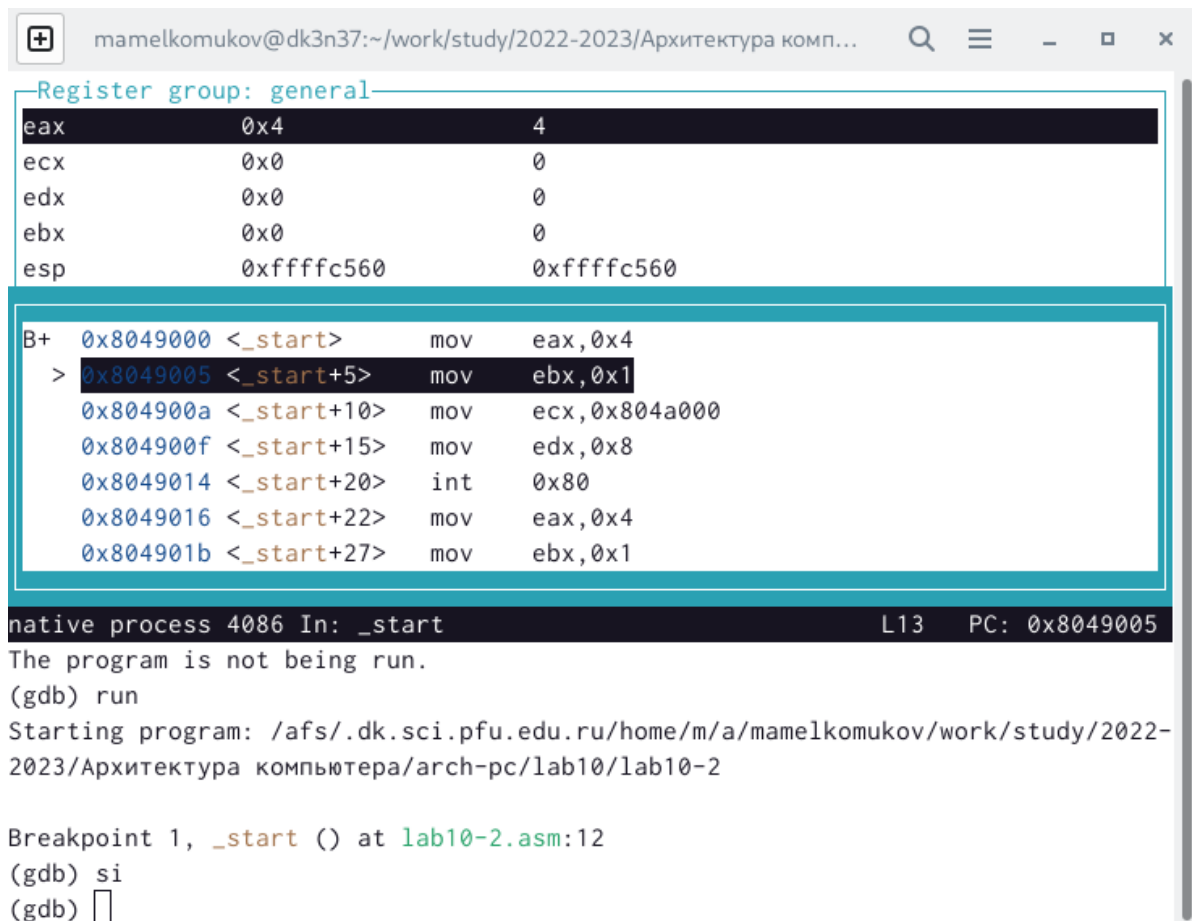


Рис. 3.17: eax присваивается значение 4



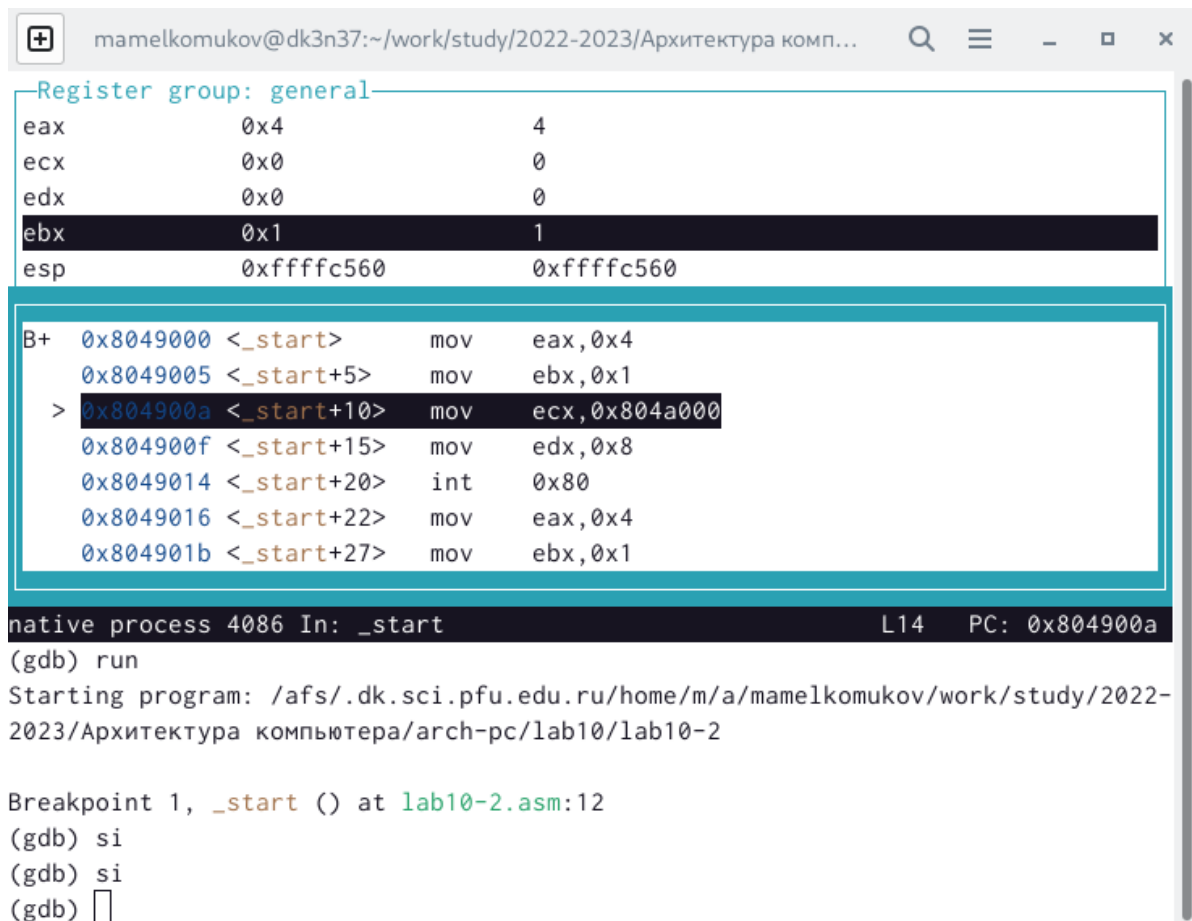


Рис. 3.18: ebx присваивается значение 1

```
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc560 0xffffc560

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 4086 In: _start L15 PC: 0x804900f
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamelkomukov/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) █
```

Рис. 3.19: ecx присваивается значение 134520832

```
mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc560 0xffffc560

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 4086 In: _start L16 PC: 0x8049014
2023/Архитектура компьютера/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) □
```

Рис. 3.20: edx присваивается значение 8

```

mamelkomukov@dk3n37:~/work/study/2022-2023/Архитектура комп...
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc560 0xffffc560

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 4086 In: _start L18 PC: 0x8049016

Breakpoint 1, _start () at lab10-2.asm:12
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 

```

Рис. 3.21: eax увеличивается на 4

```

native process 2985 In: _start L18 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc560 0xffffc560
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.22: Посмотрели содержимое регистров с помощью команды info registers (коротко i r)

```

native process 2985 In: _start                                L18   PC: 0x8049016
ebp                0x0                                0x0
esi                0x0                                0
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "

```

Рис. 3.23: С помощью команды `x/1sb &msg1` посмотрели значение переменной `msg1` по имени

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) 

```

Рис. 3.24: Посмотрели значение переменной `msg2` по адресу. Адрес переменной определили по дизассемблированной инструкции

Изменили первый символ переменной `msg1` с помощью команды `set`. Для этого задали ей в качестве аргумента имя регистра. Перед именем регистра поставили префикс `$`, а перед адресом в фигурных скобках указали тип данных.

```

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

```

Рис. 3.25: С помощью команды `x/1sb &msg1` посмотрели значение переменной `msg1`

Заменяли три символа во второй переменной `msg2`: первый на `'L'`, четвёртый на `' '` и пятый на `'%'`.

```

(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) set {char}0x804a00c=37
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor %!\n\034"
(gdb) 

```

Рис. 3.26: Результат изменений переменной `msg2`

```
(gdb) p/s $edx
$6 = 8
(gdb) p/t $edx
$7 = 1000
(gdb) p/x $edx
$8 = 0x8
(gdb) 
```

Рис. 3.27: Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. Для этого использовали команды `p/x`, `p/t` и `p/s` соответственно. Перед именем регистра ставим префикс '\$'

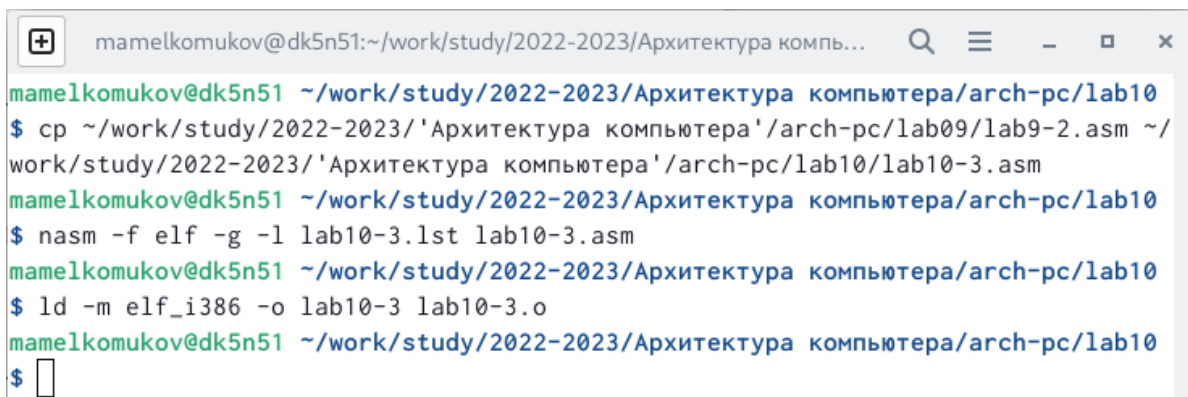
```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) 
```

Рис. 3.28: С помощью команды `set` изменили значение регистра `ebx`

Разница вывода команд `p/s $ebx` заключается в следующем:

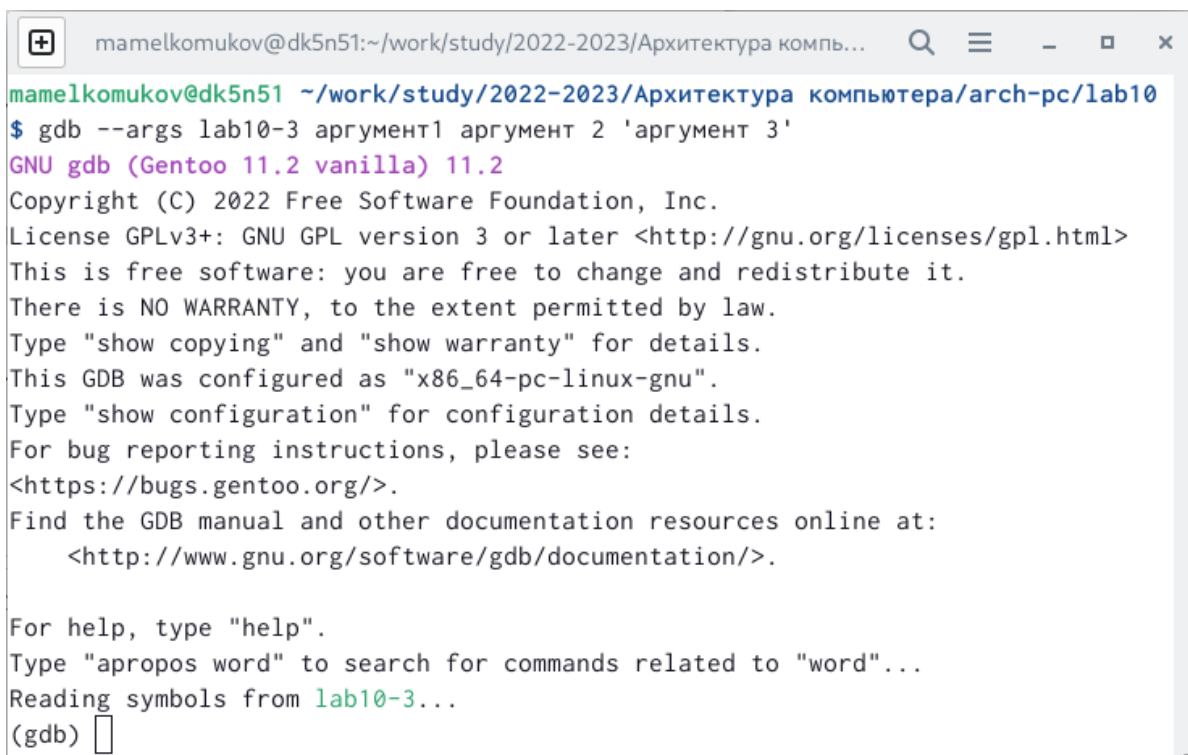
С помощью команды `continue` (сокращённо `c`) завершили выполнение программы и вышли из GDB с помощью команды `quit` (сокращённо `q`).

### 3.5 Обработка аргументов командной строки в GDB



```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ cp ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab09/lab9-2.asm ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab10/lab10-3.asm
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ld -m elf_i386 -o lab10-3 lab10-3.o
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$
```

Рис. 3.29: Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9 в файл с именем lab10-3.asm и создали исполняемый файл



```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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. 3.30: Для загрузки в gdb программы с аргументами использовали ключ `--args`. Загрузили исполняемый файл в отладчик и указали аргументы

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/a/mamelkomukov/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\
3

Breakpoint 1, _start () at lab10-3.asm:7
7               pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 3.31: Установили точку останова перед первой инструкцией в программе и запустили ее

```

(gdb) x/x $esp
0xfffffc520:      0x000000005
(gdb)

```

Рис. 3.32: Посмотрели на число аргументов командной строки

Число аргументов равно 5 – это имя программы lab10-3 и аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

```

(gdb) x/s *(void**)( $esp + 4)
0xfffffc781:      "/afs/.dk.sci.pfu.edu.ru/home/m/a/mamelkomukov/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc806:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc818:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc829:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffc82b:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)

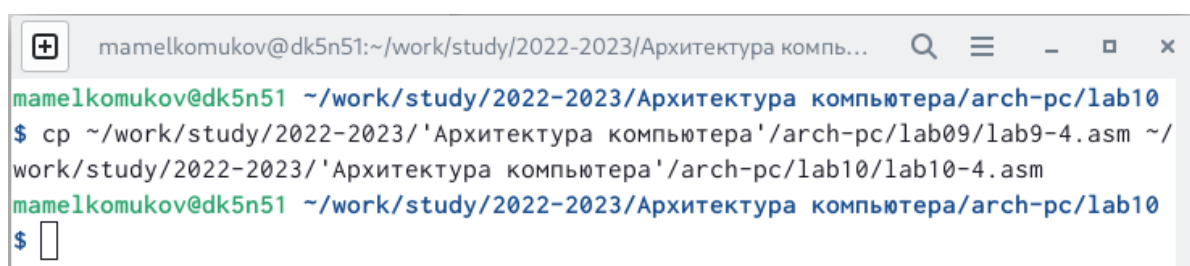
```

Рис. 3.33: Посмотрели остальные позиции стека

Их адреса располагаются в четырёх байтах друг от друга, так как именно столько занимает элемент стека.



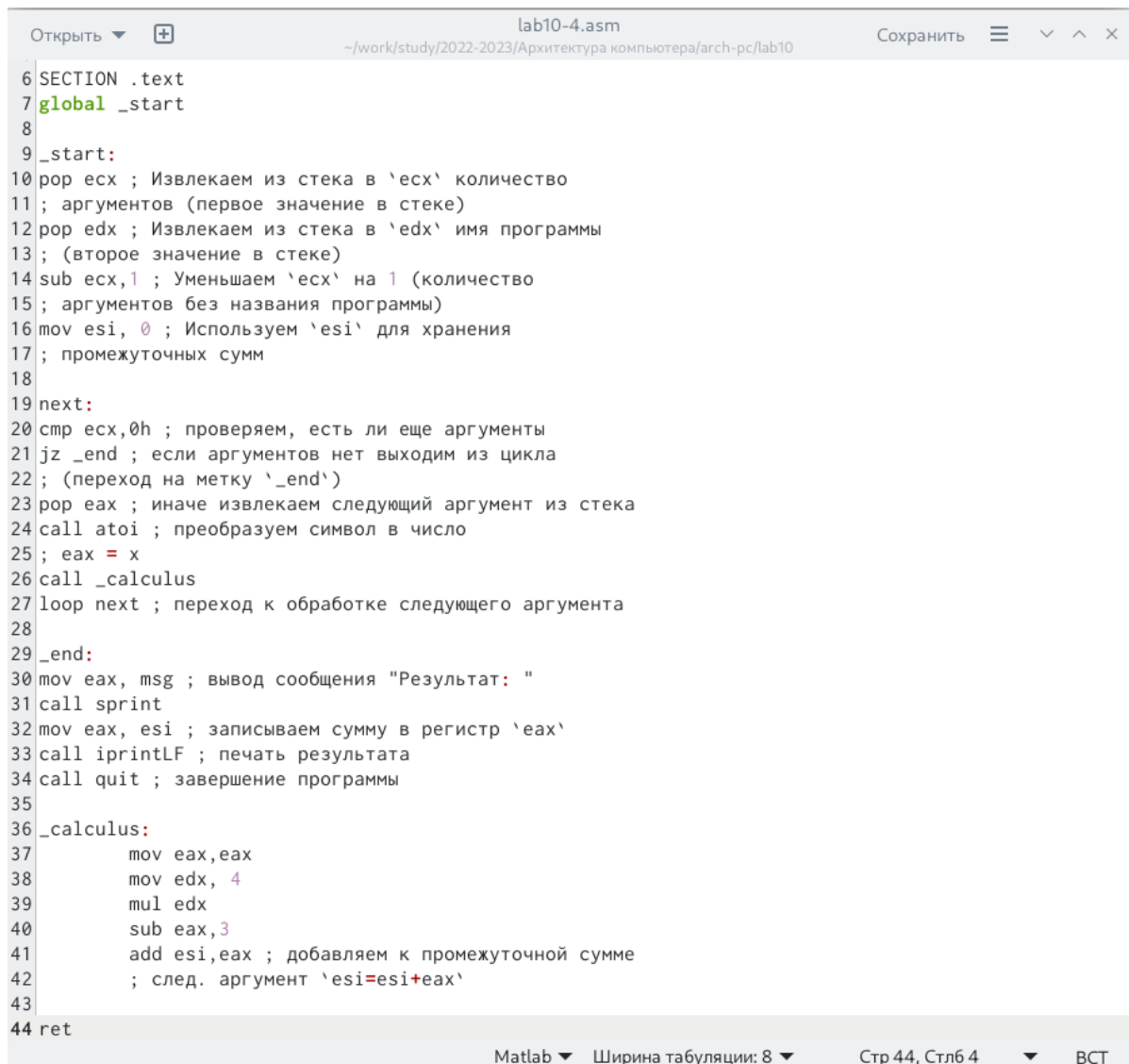
## 4 Задание для самостоятельной работы



The image shows a terminal window with a title bar containing a plus icon, the username and host 'mamelkomukov@dk5n51', the current directory path '~/work/study/2022-2023/Архитектура компь...', and standard window controls (search, menu, zoom, close). The terminal content shows the user 'mamelkomukov@dk5n51' at the directory '~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10'. They enter a command to copy a file: '\$ cp ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab09/lab9-4.asm ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab10/lab10-4.asm'. The prompt returns, and the user enters a new line '\$ '.

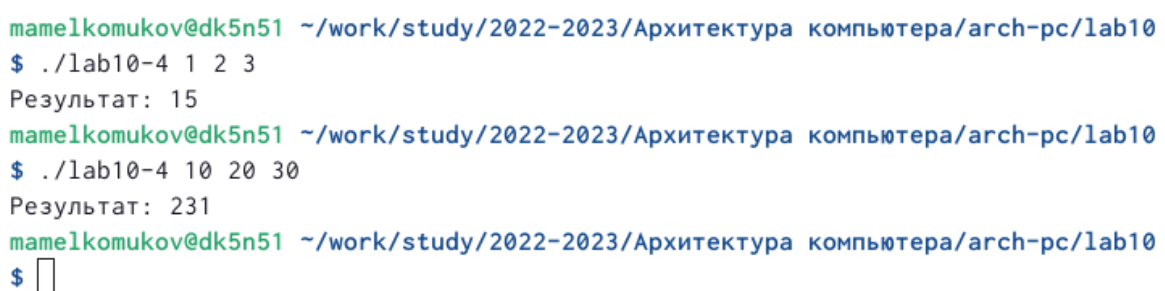
```
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компь...
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ cp ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab09/lab9-4.asm ~/work/study/2022-2023/'Архитектура компьютера'/arch-pc/lab10/lab10-4.asm
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ 
```

Рис. 4.1: Скопировали программу из лабораторной работы №9



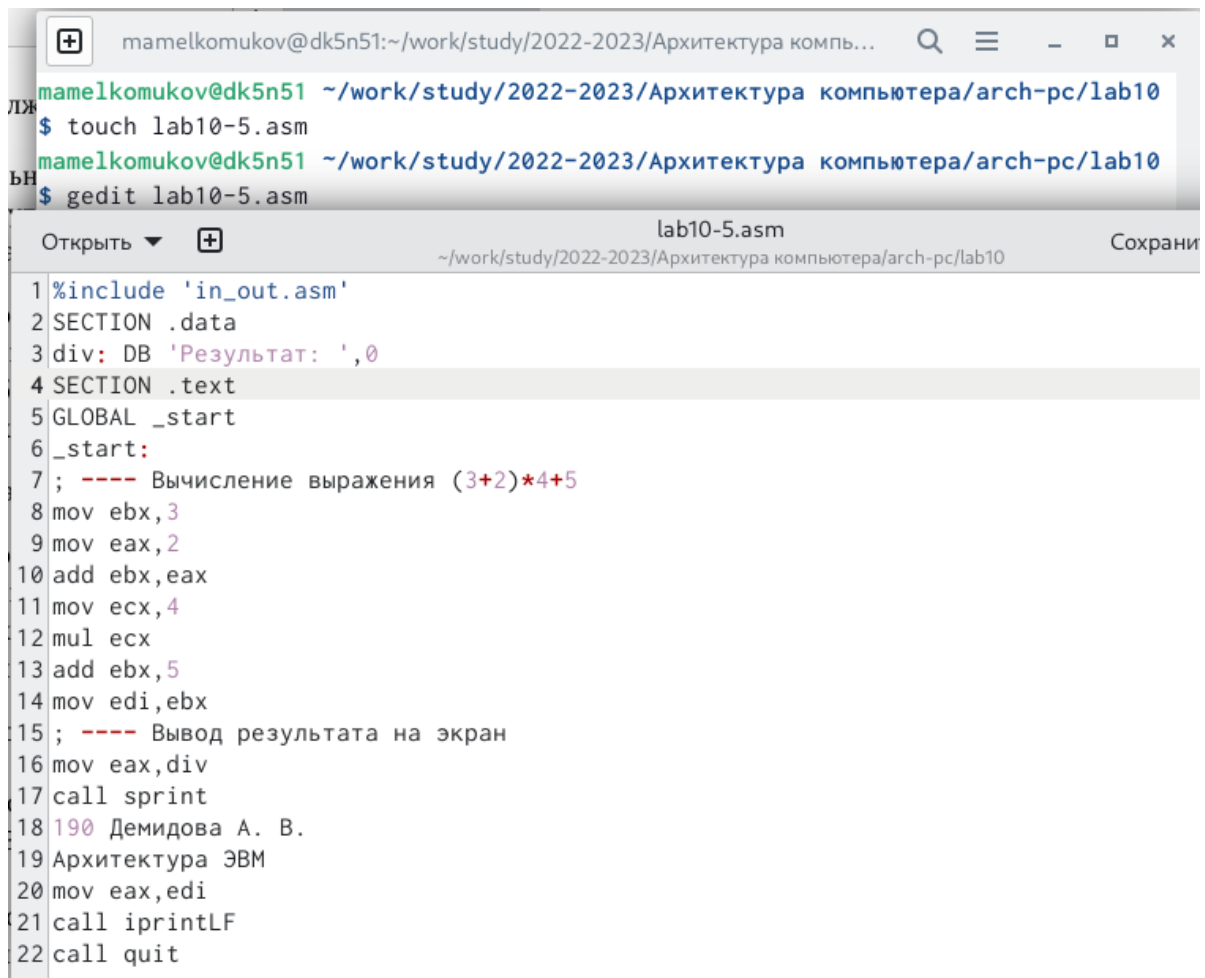
```
6 SECTION .text
7 global _start
8
9 _start:
10 pop ecx ; Извлекаем из стека в 'ecx' количество
11 ; аргументов (первое значение в стеке)
12 pop edx ; Извлекаем из стека в 'edx' имя программы
13 ; (второе значение в стеке)
14 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
15 ; аргументов без названия программы)
16 mov esi, 0 ; Используем 'esi' для хранения
17 ; промежуточных сумм
18
19 next:
20 cmp ecx,0h ; проверяем, есть ли еще аргументы
21 jz _end ; если аргументов нет выходим из цикла
22 ; (переход на метку '_end')
23 pop eax ; иначе извлекаем следующий аргумент из стека
24 call atoi ; преобразуем символ в число
25 ; eax = x
26 call _calculus
27 loop next ; переход к обработке следующего аргумента
28
29 _end:
30 mov eax, msg ; вывод сообщения "Результат: "
31 call sprint
32 mov eax, esi ; записываем сумму в регистр 'eax'
33 call iprintLF ; печать результата
34 call quit ; завершение программы
35
36 _calculus:
37     mov eax,eax
38     mov edx, 4
39     mul edx
40     sub eax,3
41     add esi,eax ; добавляем к промежуточной сумме
42     ; след. аргумент 'esi=esi+eax'
43
44 ret
```

Рис. 4.2: Реализовали вычисление значения функции как подпрограмму



```
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ./lab10-4 1 2 3
Результат: 15
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$ ./lab10-4 10 20 30
Результат: 231
mamelkomukov@dk5n51 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10
$
```

Рис. 4.3: Создали исполняемый файл и проверили его работу



The image shows a terminal window and a text editor. The terminal window at the top shows the user `mamelkomukov@dk5n51` in the directory `~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10`. The user enters the command `$ touch lab10-5.asm` and then `$ gedit lab10-5.asm`. The text editor below shows the contents of `lab10-5.asm`. The file contains assembly code for a program that calculates the expression  $(3+2)*4+5$  and prints the result. The code includes comments in Russian and a copyright notice for Demidova A. V.

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 190 Демидова А. В.
19 Архитектура ЭВМ
20 mov eax,edi
21 call iprintLF
22 call quit
```

Рис. 4.4: Создали файл `lab10-5.asm` и ввели в него текст программы

С помощью отладчика GDB проанализировали изменения значений регистров.

```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffc560 0xffffc560

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 8119 In: _start L?? PC: 0x80490f4
(gdb) layout regs
(gdb) si
0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb) si
0x080490f4 in _start ()
(gdb) 
```

Рис. 4.5: Первая ошибка: сумма элементов eax и ebx записывается в ebx

```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffc560 0xffffc560

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 8119 In: _start L?? PC: 0x80490fe
0x080490f4 in _start ()
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) □
```

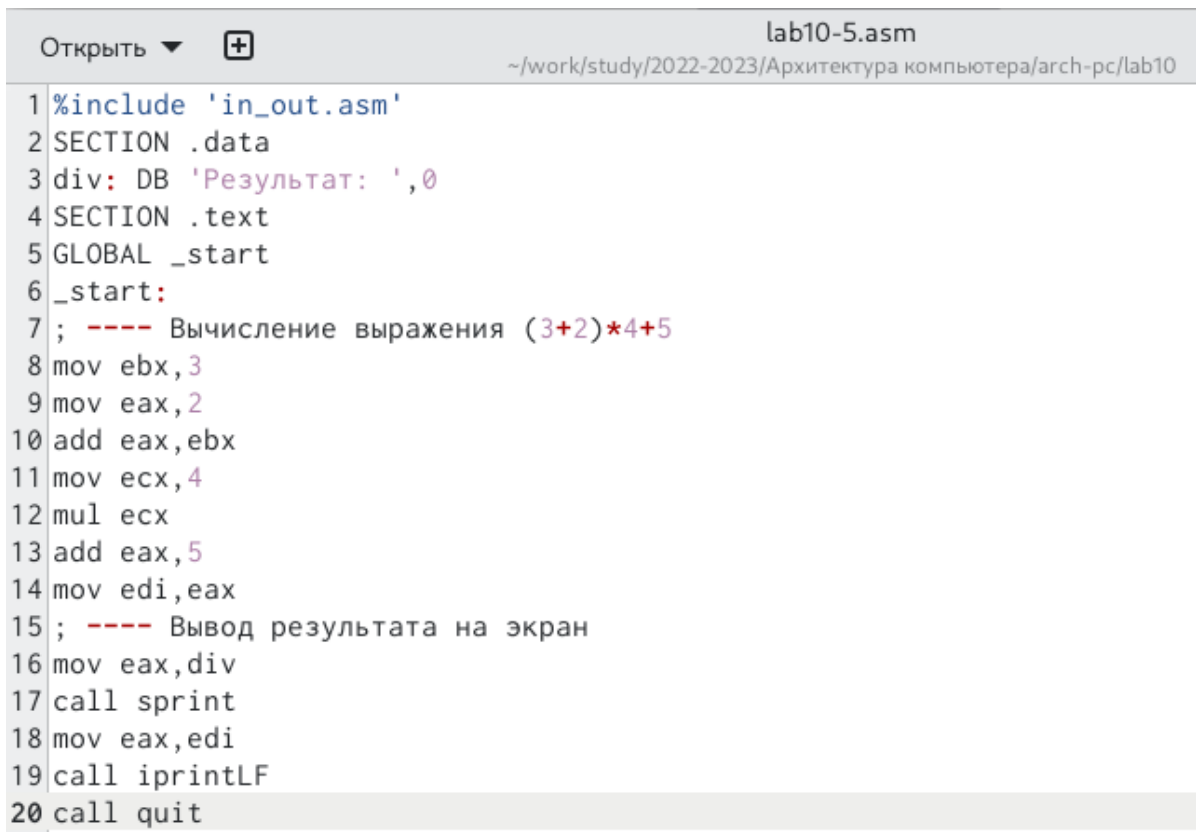
Рис. 4.6: Вторая ошибка: при умножении на 5 меняется не еах, а ебх

```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffc560 0xffffc560

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>
0x804910a <_start+34> mov    eax,edi
0x804910c <_start+36> call   0x8049086 <iprintLF>

native process 8119 In: _start L?? PC: 0x8049100
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) 
```

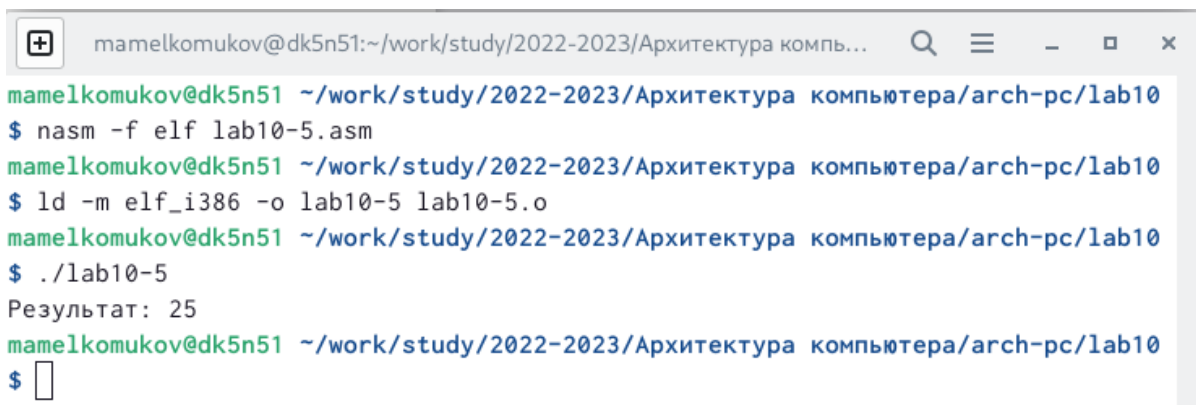
Рис. 4.7: Третья ошибка: выводится значение ebx, а не eax



```
lab10-5.asm
~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10

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

Рис. 4.8: Определили ошибки и исправили программу



```
mamelkomukov@dk5n51:~/work/study/2022-2023/Архитектура компь...
$ nasm -f elf lab10-5.asm
$ ld -m elf_i386 -o lab10-5 lab10-5.o
$ ./lab10-5
Результат: 25
$
```

Рис. 4.9: Создали исполняемый файл и проверили его работу. Ошибка устранена

## 5 Выводы

Приобрели навыки написания программ с использованием подпрограмм а также познакомились с методами отладки при помощи GDB и его основными возможностями.