

# **Отчет по лабораторной работе №9**

**Дисциплина: архитектура компьютера**

Романова Елизавета Романовна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Релаксация подпрограмм в NASM . . . . .	8
4.1.1	Отладка программ с помощью GDB . . . . .	10
4.1.2	Добавление точек останова . . . . .	13
4.1.3	Работа с данными программы в GDB . . . . .	14
4.1.4	Обработка аргументов командной строки в GDB . . . . .	17
4.2	Задание для самостоятельной работы . . . . .	17
<b>5</b>	<b>Выводы</b>	<b>22</b>
<b>6</b>	<b>Список литературы</b>	<b>23</b>

# Список иллюстраций

4.1	Создание рабочего каталога . . . . .	8
4.2	Запуск программы из листинга . . . . .	8
4.3	Изменение программы первого листинга . . . . .	8
4.4	Запуск программы в отладчике . . . . .	11
4.5	Проверка программы отладчиком . . . . .	11
4.6	Запуск отладчика с брейкпойнтом . . . . .	12
4.7	Дисассимилирование программы . . . . .	12
4.8	Режим псевдографики . . . . .	13
4.9	Список брейкпойнтов . . . . .	13
4.10	Добавление второй точки останова . . . . .	14
4.11	Просмотр содержимого регистров . . . . .	14
4.12	Просмотр содержимого переменных двумя способами . . . . .	15
4.13	Изменение содержимого переменных двумя способами . . . . .	15
4.14	Просмотр значения регистра разными представлениями . . . . .	16
4.15	Примеры использования команды set . . . . .	16
4.16	Подготовка новой программы . . . . .	17
4.17	Проверка работы стека . . . . .	17
4.18	Измененная программа предыдущей лабораторной работы . . . . .	18
4.19	Поиск ошибки в программе через пошаговую отладку . . . . .	20
4.20	Проверка корректировок в программе . . . . .	20

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

### 3 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, отработывает, 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).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

## 4 Выполнение лабораторной работы

### 4.1 Релазиация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9.

```
bash-5.2$ cd ~/work/arch-pc/lab09
bash-5.2$ touch lab9-1.asm
```

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции.

```
bash-5.2$ nasm -f elf lab9-1.asm
bash-5.2$ ld -m elf_i386 -o lab9-1 lab9-1.o
bash-5.2$ ./lab9-1
Введите x: 10
2x+7=27
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения  $f(g(x))$ .

```
bash-5.2$ nasm -f elf lab9-1.asm
bash-5.2$ ld -m elf_i386 -o lab9-1 lab9-1.o
bash-5.2$ ./lab9-1
Введите x: 10
2(3x-1)+7=65
```

Рис. 4.3: Изменение программы первого листинга

Код программы:

```
%include 'in_out.asm'
```



**SECTION** .data

msg: DB 'Введите x: ', 0

result: DB '2(3x-1)+7=', 0

**SECTION** .bss

x: RESB 80

res: RESB 80

**SECTION** .text

**GLOBAL** \_start

\_start:

mov eax, msg

call sprint

mov ecx, x

mov edx, 80

call sread

mov eax, x

call atoi

call \_calcul

mov eax, result

call sprint

mov eax, [res]

call iprintLF

call quit

```
_calcul:
push eax
call _subcalcul
```

```
mov ebx, 2
mul ebx
add eax, 7
```

```
mov [res], eax
pop eax
ret
```

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

#### 4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике.

```

bash-5.2$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
bash-5.2$ ld -m elf_i386 -o lab9-2 lab9-2.o
bash-5.2$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc48
Copyright (C) 2023 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 lab9-2...
(gdb)

```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `run`, я убедился в том, что она работает исправно.

```

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 lab9-2...
(gdb) run
Starting program: /home/erromanova/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URIs:
<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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 9428) exited normally]
(gdb)

```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку.

```

(gdb) run
Starting program: /home/erromanova/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 9891) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/erromanova/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4

```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *amd топчик* (рис. -fig. 4.7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```

0x0040020: <+37>: mov 38x7,%eax
0x0040020: <+42>: int 38x86
0x0040020: <+44>: mov 38x1,%eax
0x0040020: <+49>: mov 38x8,%eax
0x0040020: <+54>: int 38x86
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x0040000: <+8>: mov eax,0x0
0x0040000: <+5>: mov ebx,0x1
0x0040000: <+18>: mov ecx,0x0040000
0x0040007: <+15>: mov edx,0x0
0x0040010: <+28>: int 0x80
0x0040010: <+22>: mov eax,0x0
0x004001b: <+27>: mov ebx,0x1
0x0040020: <+32>: mov ecx,0x0040000
0x0040020: <+37>: mov edx,0x7
0x0040020: <+42>: int 0x80
0x0040020: <+44>: mov eax,0x1
0x0040031: <+49>: mov ebx,0x0
0x0040030: <+54>: int 0x80
End of assembler dump.
(gdb)

```

Рис. 4.7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы.

```

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

native process 9959 In: _start L9 PC: 0x8049000
(gdb)

```

Рис. 4.8: Режим псевдографики

## 4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился.

```

~Register group: general~
eax 0x0      0      ecx 0x0      0
edx 0x0      0      ebx 0x0      0
esp 0xffffcf10 0xffffcf10 ebp 0x0      0x0
esi 0x0      0      edi 0x0      0
eip 0x8049000 0x8049000 <_start> eflags 0x202    [ IF ]
cs  0x23     35     ss  0x2b     43
ds  0x2b     43     es  0x2b     43
fs  0x0      0      gs  0x0      0

0x80490f2 add    BYTE PTR [eax],al
0x80490f4 add    BYTE PTR [eax],al
0x80490f6 add    BYTE PTR [eax],al
0x80490f8 add    BYTE PTR [eax],al
0x80490fa add    BYTE PTR [eax],al
0x80490fc add    BYTE PTR [eax],al
0x80490fe add    BYTE PTR [eax],al
0x8049700 add    BYTE PTR [eax],al
0x8049702 add    BYTE PTR [eax],al
0x8049704 add    BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num Type      Disp Enb Address What
1 breakpoint keep y 0x8049005 lab9-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x8049031 lab9-2.asm:24
(gdb)

```

Рис. 4.9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции.

```

Register group: general
eax    0x0      0      ecx    0x0      0
edx    0x0      0      ebx    0x0      0
esp    0xffffcf10 0xffffcf10  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x8049000 0x8049000 <_start>  eflags  0x202    [ IF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

0x80490f2  add    BYTE PTR [eax],al
0x80490f4  add    BYTE PTR [eax],al
0x80490f6  add    BYTE PTR [eax],al
0x80490f8  add    BYTE PTR [eax],al
0x80490fa  add    BYTE PTR [eax],al
0x80490fc  add    BYTE PTR [eax],al
0x80490fe  add    BYTE PTR [eax],al
0x8049100  add    BYTE PTR [eax],al
0x8049102  add    BYTE PTR [eax],al
0x8049104  add    BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num    Type      Disp Enb Address      What
1      breakpoint keep y  0x8049000 lab9-2.asm:11
      breakpoint already hit 1 time
2      breakpoint keep y  0x8049031 lab9-2.asm:24
(gdb)

```

Рис. 4.10: Добавление второй точки останова

### 4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers.

```

eax    0x0      0      ebx    0x0      0
edx    0x0      0      ecx    0x0      0
esp    0xffffcf10 0xffffcf10  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x8049000 0x8049000 <_start>  eflags  0x202    [ IF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

0x80490f2  add    BYTE PTR [eax],al
0x80490f4  add    BYTE PTR [eax],al
0x80490f6  add    BYTE PTR [eax],al
0x80490f8  add    BYTE PTR [eax],al
0x80490fa  add    BYTE PTR [eax],al
0x80490fc  add    BYTE PTR [eax],al
0x80490fe  add    BYTE PTR [eax],al
0x8049100  add    BYTE PTR [eax],al
0x8049102  add    BYTE PTR [eax],al
0x8049104  add    BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
Type 'q' to quit, 'c' to continue without paging--

```

Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу.

```

Register group: general
eax 0x0 0 00000000 134520812
edx 0x0 0 00000000 0
esp 0xfffff10 0xfffff10 0 0
esi 0x0 0 00000000 0
ebp 0x0049016 0x0049016 <_start+22> 0x202 [ IF ]
cs 0x23 35 0x2b 43
ds 0x2b 43 0x2b 43
fs 0x0 0 0x0 0

0x0049000 <_start> mov eax,edx
0x0049005 <_start+5> mov ebx,ecx
0x004900a <_start+10> mov ecx,0x004a000
0x004900f <_start+15> mov edx,edx
0x0049014 <_start+20> int 0x0
0x0049016 <_start+22> mov eax,edx
0x004901b <_start+27> mov ebx,ecx
0x0049020 <_start+32> mov ecx,0x004a000
0x0049025 <_start+37> mov edx,edx
0x004902a <_start+42> int 0x0

native process 5886 (asm) in: _start L17 PC: 0x0049016
eax 0x0 0
edx 0x0 0
esp 0x0049016 0x0049016 <_start+22>
eflags 0x202 [ IF ]
cs 0x23 35
--Type (RET) for more, q to quit, c to continue without paging--
Quit
(gdb) x/10b &msg1
0x0049000: "Hello, "
(gdb) x/10b 0x004a000
0x004a000: "World!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу.

```

Register group: general
eax 0x0 0 00000000 134520832
edx 0x0 0 00000000 1
esp 0xfffff10 0xfffff10 0 0
esi 0x0 0 00000000 0
ebp 0x0049016 0x0049016 <_start+22> 0x202 [ IF ]
cs 0x23 35 0x2b 43
ds 0x2b 43 0x2b 43
fs 0x0 0 0x0 0

0x0049000 <_start> mov eax,edx
0x0049005 <_start+5> mov ebx,ecx
0x004900a <_start+10> mov ecx,0x004a000
0x004900f <_start+15> mov edx,edx
0x0049014 <_start+20> int 0x0
0x0049016 <_start+22> mov eax,edx
0x004901b <_start+27> mov ebx,ecx
0x0049020 <_start+32> mov ecx,0x004a000
0x0049025 <_start+37> mov edx,edx
0x004902a <_start+42> int 0x0

native process 5886 (asm) in: _start L17 PC: 0x0049016
(gdb) x/10b <msg1>
0x0049000: "Hello, "
(gdb) x/10b 0x004a000
0x004a000: "World!\n\034"
(gdb) set (char)msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set (char)&msg1='h'
(gdb) x/10b &msg1
0x0049000: "hello, "
(gdb) set (char)&msg2='x'
(gdb) x/10b &msg2
0x004a000: "xoridi!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      0x0      0

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
(gdb) p/t $ecx
$2 = 100000000100101000000000000000
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 4.14: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      0x0      0

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
(gdb) set $ebx='2'
(gdb) p/s
$6 = 8
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb)

```

Рис. 4.15: Примеры использования команды set



#### 4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки .

```
bash-5.2$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
bash-5.2$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
bash-5.2$ ld -m elf_i386 -o lab9-3 lab9-3.o
```

Рис. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта, ошибка при аргументе `+24` означает, что аргументы на вход программы закончились.

```
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/erromanova/work/arch-pc/lab09/lab9-3

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 lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/s*(void**)(esp+4)
```

Рис. 4.17: Проверка работы стека

## 4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы.

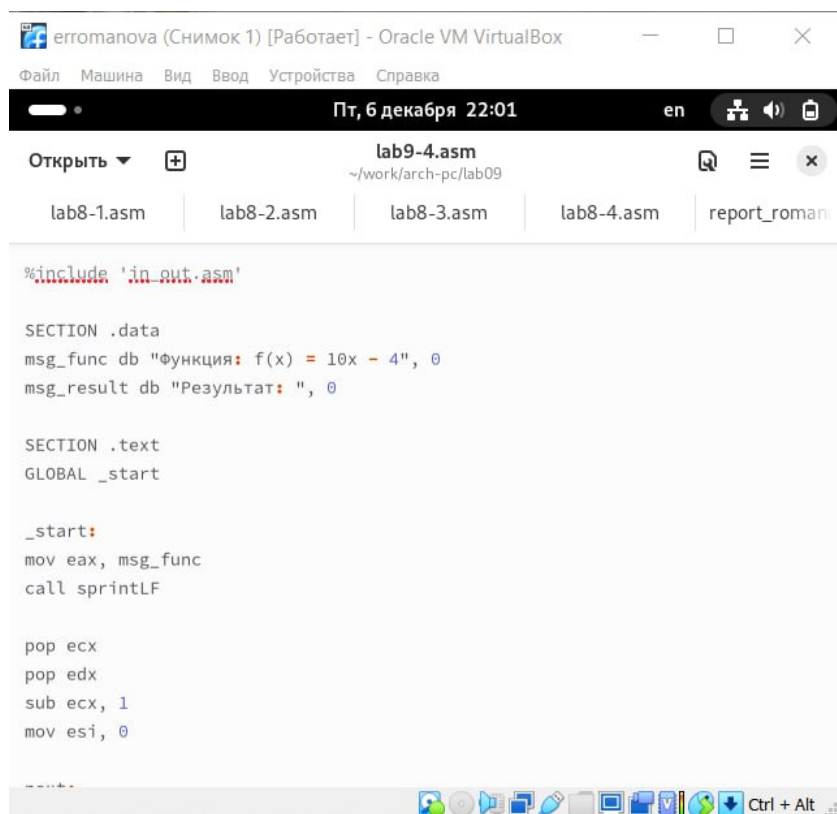


Рис. 4.18: Измененная программа предыдущей лабораторной работы

Код программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg_func db "Функция: f(x) = 10x - 4", 0
```

```
msg_result db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

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

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладчика и пошагово через si просматриваю

изменение значений регистров через `i` `r`. При выполнении инструкции `mul` `ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию .

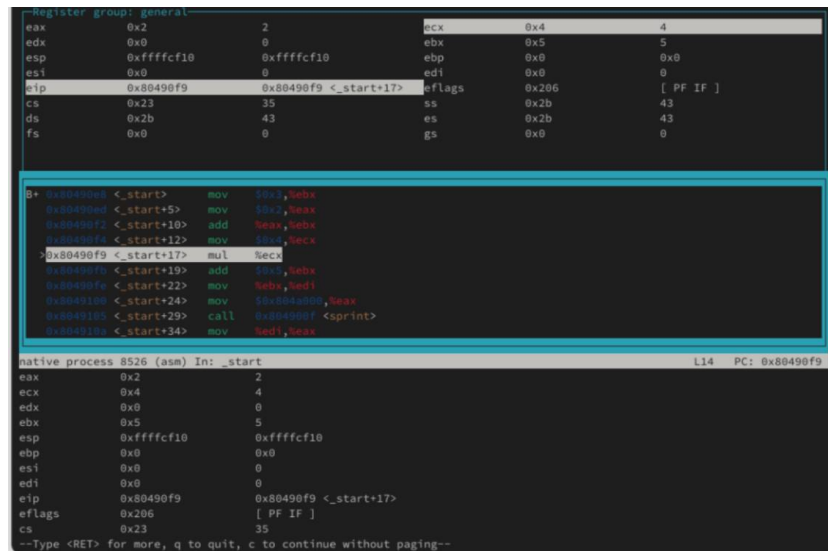


Рис. 4.19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции.

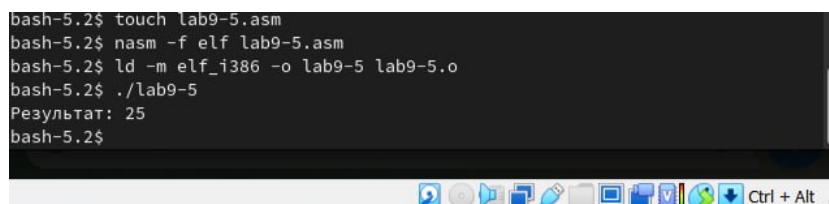


Рис. 4.20: Проверка корректировок в программе

Код измененной программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
div: DB 'Результат: ', 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov ebx, 3
```

```
mov eax, 2
```

```
add ebx, eax
```

```
mov eax, ebx
```

```
mov ecx, 4
```

```
mul ecx
```

```
add eax, 5
```

```
mov edi, eax
```

```
mov eax, div
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```

## **5 Выводы**

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.

## **6 Список литературы**

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.