

Лабораторная работа №9

Понятие подпрограммы.Отладчик GDB.

Прозорова Елизавета Евгеньевна

Содержание

1	Цель работы	3
2	Выполнение лабораторной работы	4
3	Выполнение самостоятельной работы	18
4	Выводы	25

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

1. Сначала я создала каталог для программ лабораторной работы № 9, затем перешла в него и создала файл lab9-1.asm

```
eeprozorova@dk3n55 ~ $ mkdir ~/work/arch-pc/lab09
eeprozorova@dk3n55 ~ $ cd ~/work/arch-pc/lab09
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ touch lab09-1.asm
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создание каталога и файла lab9-1

2. Я ввела в файл lab09-1.asm текст программы из листинга 9.1.

```
lab09-1.asm      [----]  0 L
#include 'in_out.asm'

SECTION .data
msg1: DB 'Введите x: ',0
msg2: DB 'f(x)2x+7: ',0
msg3: DB 'g(x)3x-1 ',0
result: DB 'f(g(x))=',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg2
call sprintf

mov eax, msg3
call sprintf
```

Рис. 2.2: Текст программы lab09-1

Я создала исполняемый файл и запустила его. Для x я выбрала числа 2 и 9.

```

eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ touch lab09-1.asm
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ./var09-1
bash: ./var09-1: Нет такого файла или каталога
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=11
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 9
2x+7=25

```

Рис. 2.3: Создание и запуск lab09-1

Я изменила текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$.

```

30 mov eax,x
31 call atoi
32
33 call _calcul
34
35 mov eax,result
36 call sprint
37 mov eax,[res]
38 call iprintLF
39
40 call quit
41
42 _calcul:
43
44 call _subcalcul
45
46 mov ecx,2
47 mul ecx
48 add eax,7
49 mov [res],eax
50 ret
51
52 _subcalcul:
53 mov ebx,3
54 mul ebx
55 sub eax,7
56
57 ret

```

Рис. 2.4: Изменения текста

Затем я создала и проверила измененный файл.

```

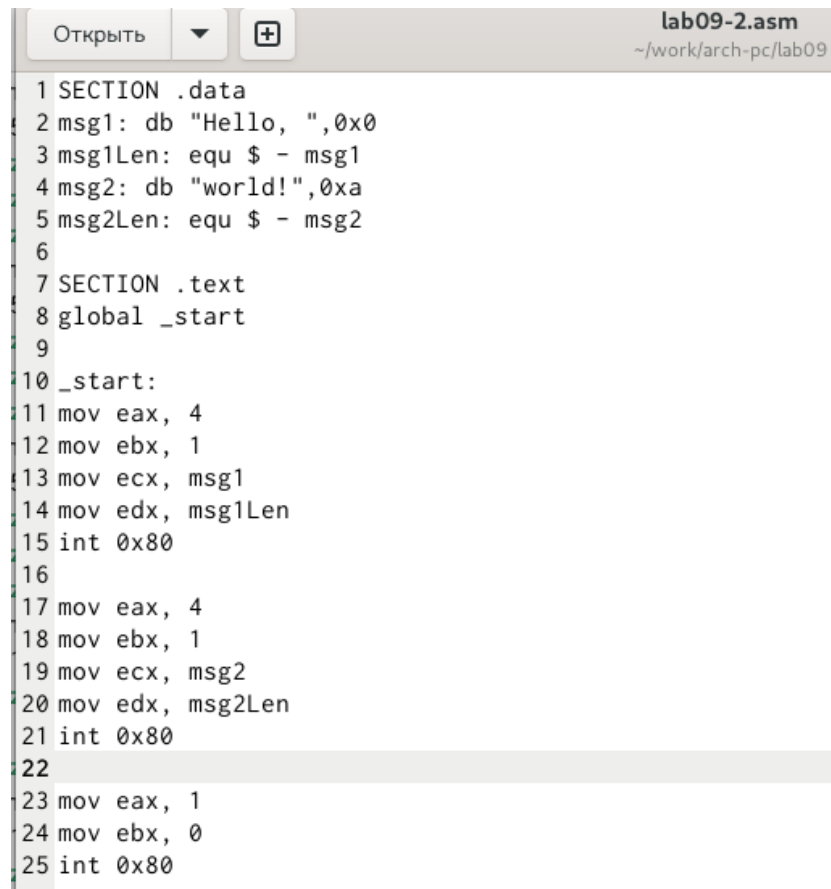
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ./lab09-1
f(x)2x+7:
g(x)3x-1
Введите x:
2
f(g(x))=17
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ./lab09-1

```

Рис. 2.5: Создание и запуск lab09-1

2. Я создала файл `lab09-2.asm` в каталоге `~/work/arch-pc/lab09` и ввела в него

текст программы из листинга 9.2.

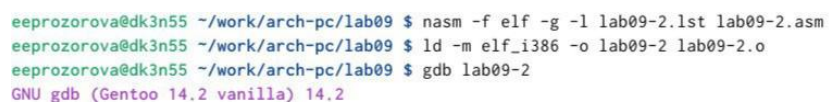


```
lab09-2.asm
~/work/arch-pc/lab09

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

Рис. 2.6: Текст программы lab09-2.asm

Затем я создала измененный файл ключом ‘-g’.



```
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
```

Рис. 2.7: Получение исполняемого файла

Я загрузила исполняемый файл в отладчик gdb:

```

eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-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 lab09-2...

```

Рис. 2.8: Отладчик gdb

Затем я проверила работу программы, запустив ее в оболочке GDB с помощью команды `run`

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/e/eeprozorova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5833) exited normally]
(gdb) 

```

Рис. 2.9: Проверка работы программы

Для более подробного анализа программы установила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её.

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/e/eeprozorova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb) 

```

Рис. 2.10: Утановка брейкпоинта

Затем я посмотрела дисассимилированный код программы


```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ec
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
      0x08049016 <+22>:     mov     $0x4,%eax
      0x0804901b <+27>:     mov     $0x1,%ebx
      0x08049020 <+32>:     mov     $0x804a008,%ec
      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.

```

Рис. 2.11: Команда disassemble

Затем я переключилась на отображение команд с Intel'овским синтаксисом

```

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.

```

Рис. 2.12: Intel'овский синтаксис

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

- 1) Порядок операндов АТТ: источник - назначение (напр. `movl %ebx, %eax`).
Intel: назначение - источник (напр. `MOV EAX, EBX`).
- 2) Регистры АТТ: С префиксом % (напр. `%eax`). Intel: Без префикса (напр. `EAX`).
- 3) Размеры операндов АТТ: Указывается суффикс (b, w, l, q для 1, 2, 4, 8 байт).
Intel: Используются указатели (BYTE PTR, DWORD PTR и т.д.).
- 4) Память АТТ: Круглые скобки для указания адреса (напр. `4(%ebx)`). Intel: Квадратные скобки (напр. `[EBX+4]`).

Я включила 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
0x8049038                add     BYTE PTR [eax],al
0x804903a                add     BYTE PTR [eax],al
0x804903c                add     BYTE PTR [eax],al
0x804903e                add     BYTE PTR [eax],al
0x8049040                add     BYTE PTR [eax],al

native process 5864 In: _start
(gdb) █
```

Рис. 2.13: Команда layout asm

```
[ 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
0x8049020 <_start+32>   mov    ecx,0x804a008
0x8049025 <_start+37>   mov    edx,0x7

native process 5864 In: _start
(gdb) layout regs
(gdb) □
```

Рис. 2.14: Команда layout regs

3. Так как на предыдущих шагах была установлена точка останова по имени метки (`_start`), я проверю это с помощью команды `info breakpoints`.

```
(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. 2.15: Команда info breakpoints.

Теперь установим точку остановк на адресе предпоследней инструкции (`mov ebx,0x0`).

```

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
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al

native process 5864 In: _start L11 P
(gdb) break mov ebx,0x0
Function "mov ebx" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(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. 2.16: Установка точки

Посмотрим информацию о всех установленных точках останова.

```

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
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al

native process 5864 In: _start L11 P
(gdb) break mov ebx,0x0
Function "mov ebx" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(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. 2.17: Команда i b

4. Я посмотрела содержимое регистров также можно с помощью команды info

registers

```
native process 5864 In: _start L11 PC: 0
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc470 0xffffc470
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.18: Команда info registers

Затем я посмотрела значение переменной msg1 по имени.

```
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
0x8049038          add BYTE PTR [eax],al
0x804903a          add BYTE PTR [eax],al
0x804903c          add BYTE PTR [eax],al
0x804903e          add BYTE PTR [eax],al
0x8049040          add BYTE PTR [eax],al

native process 5864 In: _start L11 PC: 0x80490
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)
```

Рис. 2.19: Значение переменной msg1 по имени

я посмотрела значение переменной msg2 по адресу.

```

native process 5864 In: _start L11 PC: 0
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc470 0xffffc470
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.20: Значение переменной msg2 по адресу

Изменила первый символ переменной msg1. Теперь первая буква Hello стала маленькой.

```

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

```

Рис. 2.21: Команда изменения значение для регистра msg1

Теперь заменю первый и второй символы переменной msg2 на Р и а соответственно.

```

invalid character constant.
(gdb) set {char}0x804a008='P'
(gdb) set {char}0x804a009='a'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Parld!\n\034"
(gdb)

```

Рис. 2.22: Команда изменения значение для регистра msg2

С помощью команды print вывела в различный форматах значение регистра edx.

```

(gdb) p/f $edx
$1 = 0
(gdb) p/t $edx
$2 = 0
(gdb) p/x $edx
$3 = 0x0

```

Рис. 2.23: регистр edx в различный форматах

С помощью команды set изменила значение регистра ebx

```
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
```

Рис. 2.24: Команда set

Разница выводов команд p/s \$ebx: В первом случае регистр хранит ASCII-код символа и gdb пытается интерпретировать его как строку. Во втором случае \$ebx хранит число и gdb пытается интерпретировать его как адрес, что выводит ошибку

Затем я завершила выполнение программы и вышла из GDB.

```
(gdb) si
(gdb) q
```

Рис. 2.25: Завершение выполнения программы и выход из GDB.

5. Я скопировала файл lab8-2.asm в файл с именем lab09-3.asm

```
eeprozorova@dk3n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/
/lab09-3.asm
```

Рис. 2.26: Копирование файла lab8-2.asm

Я создала исполняемый файл.

```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 2.27: Создания исполняемого файла

Загрузила исполняемый файл в отладчик, указав аргументы аргумент1 аргумент 2 'аргумент 3'

```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент
2 'аргумент 3'
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-pc-linux-gnu"
```

Рис. 2.28: загрузка в gdb программы с аргументами

Установила точку останова перед первой инструкцией в программе и запустила ее.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/e/eeprozorova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3

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

Рис. 2.29: Установка точки останова

Я вывела адрес вершины стека

```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ gdb -ex run lab09-3 аргумент1 аргумент2 аргумент\ 3
```

Рис. 2.30: Вывод адрес вершины стека

Затем я посмотрела остальные позиции стека

```
(gdb) x/s *(void**)(esp+4)
0xffffc65e: "/afs/.dk.sci.pfu.edu.ru/home/e/e/eeprozorova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffc6a6: "аргумент1"
(gdb) x/s *(void**)(esp+12)
0xffffc6b8: "аргумент"
(gdb) x/s *(void**)(esp+16)
0xffffc6c9: "2"
(gdb) x/s *(void**)(esp+20)
0xffffc6cb: "аргумент 3"
(gdb) x/s *(void**)(esp+24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 2.31: Позиции стека

Шаг изменения адреса равен 4, потому что в 32-битной архитектуре размер слова — 4 байта, стек выравнивается по 4 байта, и каждый параметр или локальная переменная занимает 4 байта.

3 Выполнение самостоятельной работы

1. Я скопировала программу из лабораторной работы №8, и назвала lab09-4.asm

```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/var.asm ~/work/arch-pc/  
lab09/lab09-4.asm  
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ █
```

Рис. 3.1: Копирование программы из лабораторной работы №8

Затем я изменила эту программу, реализовав вычисление значения функции $\pi(x)$ как подпрограмму.

```

18
19 next:
20 cmp ecx,0h
21 jz _end
22
23 pop eax
24 call atoi
25 call _fxx
26 add esi,eax
27 loop next
28
29 _end:
30
31 mov eax, msg2
32 call sprint
33 mov eax, esi
34 call iprintLF
35 call quit
36
37 _fxx:
38 mov ebx,2
39 mul ebx
40 add eax,7
41 ret

```

Рис. 3.2: Измененный текст программы

Затем я создала и проверила измененный файл.

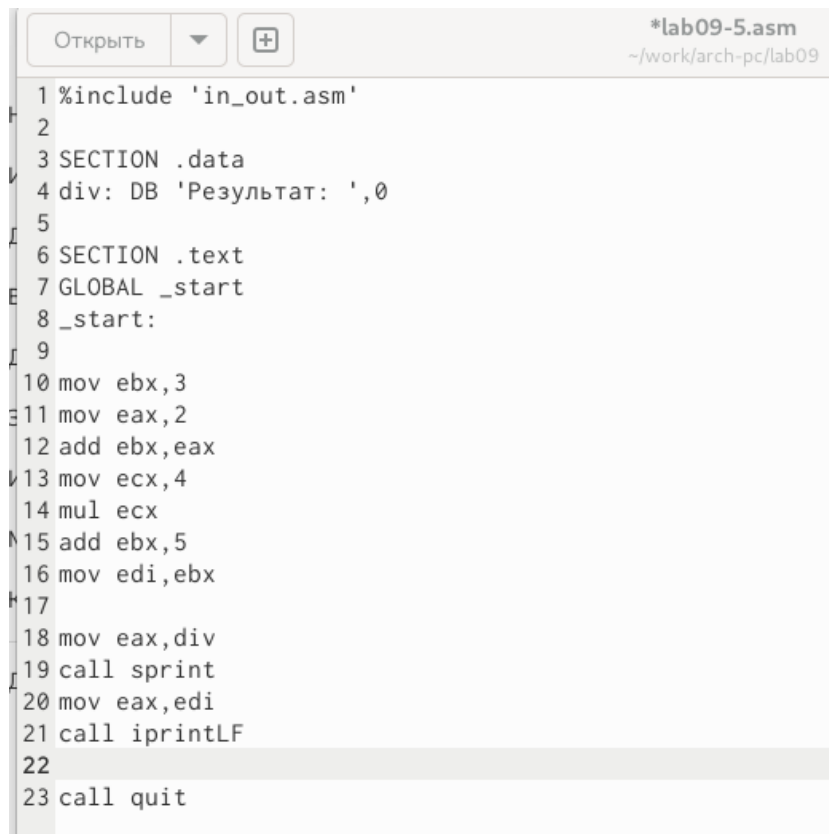
```

eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3
Функция: f(x)=7+2x
Результат: 33

```

Рис. 3.3: Проверка программы

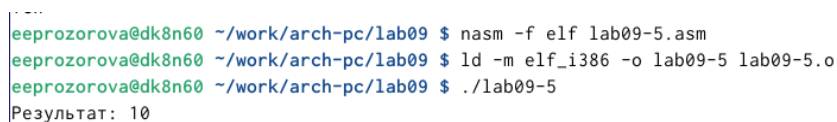
2. Я создала файл lab09-5.asm в который написала программу из листинга 9.3. с программа вычисления выражения $(3+2)*4+5$



```
Открыть ▼ + *lab09-5.asm
~/work/arch-pc/lab09
1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov ebx,3
11 mov eax,2
12 add ebx,eax
13 mov ecx,4
14 mul ecx
15 add ebx,5
16 mov edi,ebx
17
18 mov eax,div
19 call sprint
20 mov eax,edi
21 call iprintLF
22
23 call quit
```

Рис. 3.4: Текст программы из листинга 9.3.

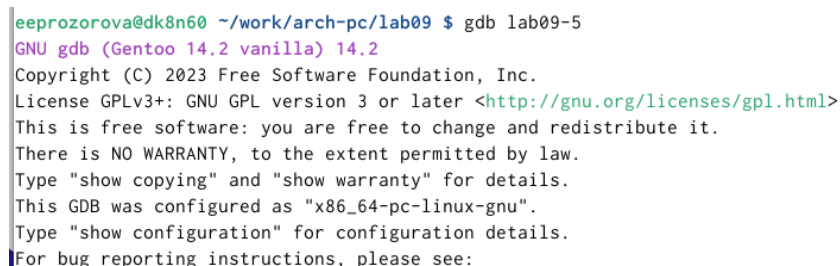
Я создала и проверила файл. Ответ неверный.



```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10
```

Рис. 3.5: Проверка программы

Затем я открыла этот файл в GDB.



```
eeprozorova@dk8n60 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

Рис. 3.6: Файл lab09-5 в GDB

Для более подробного анализа программы я установила брейкпоинт на метку `_start`.

```
Reading symbols from lab09-5...
(No debugging symbols found in lab09-5)
(gdb) b _start
Breakpoint 1 at 0x080490e8
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/e/eeprozorova/work/arch-pc/lab09/lab09-5

Breakpoint 1, 0x080490e8 in _start ()
(gdb) █
```

Рис. 3.7: Установка брейкпоинта

Затем я посмотрела дисассимилированный код программы, и переключилась на отображение команд с Intel'овским синтаксисом

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     $0x3,%ebx
    0x080490ed <+5>:      mov     $0x2,%eax
    0x080490f2 <+10>:     add     %eax,%ebx
    0x080490f4 <+12>:     mov     $0x4,%ecx
    0x080490f9 <+17>:     mul     %ecx
    0x080490fb <+19>:     add     $0x5,%ebx
    0x080490fe <+22>:     mov     %ebx,%edi
    0x08049100 <+24>:     mov     $0x804a000,%eax
    0x08049105 <+29>:     call    0x804900f <sprint>
    0x0804910a <+34>:     mov     %edi,%eax
    0x0804910c <+36>:     call    0x8049086 <iprintf>
    0x08049111 <+41>:     call    0x80490db <quit>
End of assembler dump.
```

Рис. 3.8: Дисассимилированный код программы

```

(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 <iprintLF>
    0x08049111 <+41>:     call    0x80490db <quit>
End of assembler dump.
(gdb) █

```

Рис. 3.9: Отображение команд с Intel'овским синтаксисом

Включила режим псевдографики для более удобного анализа программы

```

[ Register Values Unavailable ]

0x8049000 <slen>      push    ebx
0x8049001 <slen+1>    mov     ebx, eax
0x8049003 <nextchar>  cmp     BYTE PTR [eax], 0x0
0x8049006 <nextchar+3> je      0x804900b <finished>
0x8049008 <nextchar+5> inc     eax
0x8049009 <nextchar+6> jmp     0x8049003 <nextchar>
0x804900b <finished> sub     eax, ebx
0x804900d <finished+2> pop     ebx
0x804900e <finished+3> ret
0x804900f <sprint>    push    edx
0x8049010 <sprint+1>  push    ecx
0x8049011 <sprint+2>  push    ebx

native process 6720 In: _start
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc450 0xffffc450
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.10: Режим псевдографики

Затем я подробно изучила все строки программы, и нашла ошибку, в определенном месте значения не соответствовали своим регистрам. Я исправила ошибку, поменяв их местами.

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov ebx,3
11 mov eax,2
12 add eax,ebx
13 mov ecx,4
14 mul ecx
15 add eax,5
16 mov edi,eax
17
18 mov eax,div
19 call sprint
20 mov eax,edi
21 call iprintLF
22
23 call quit

```

Рис. 3.11: Измененный текст программы

Я создала и проверила измененный файл. Ответ теперь верный.

```

ee Prozorova@dk8n60 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
ee Prozorova@dk8n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
ee Prozorova@dk8n60 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 25
ee Prozorova@dk8n60 ~/work/arch-pc/lab09 $ █

```

Рис. 3.12: Проверка программы

4 Выводы

В ходе лабораторной работы мной были приобретены навыки написания программ с использованием подпрограмм. Я также изучила метод отладки при помощи GDB и его основными возможностями.