

# **Лабораторная работа №9**

**Понятие подпрограммы. Отладчик GDB.**

Славинский Владислав Вадимович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>3</b>	<b>Выполнение задания для самостоятельной работы</b>	<b>22</b>
<b>4</b>	<b>Вывод</b>	<b>32</b>

# Список иллюстраций

2.1	Создание lab9-1 . . . . .	7
2.2	Код программы . . . . .	8
2.3	Запуск . . . . .	9
2.4	Замена . . . . .	10
2.5	Запуск измененной программы . . . . .	11
2.6	Код программы . . . . .	12
2.7	Программа . . . . .	13
2.8	Запуск . . . . .	13
2.9	Брейкпоинт . . . . .	14
2.10	Дизассемблирование . . . . .	14
2.11	Переключение на другой синтаксис . . . . .	15
2.12	Включение режима . . . . .	16
2.13	Вывод информации . . . . .	16
2.14	Создание и информация о брейкпоинтах . . . . .	17
2.15	si . . . . .	17
2.16	Информация о регистрах . . . . .	18
2.17	Вывод значения msg1 . . . . .	18
2.18	Вывод значения msg2 . . . . .	18
2.19	Смена первого символа . . . . .	19
2.20	Смена первого символа . . . . .	19
2.21	Смена первого символа . . . . .	19
2.22	Смена первого символа . . . . .	20
2.23	Копирование . . . . .	20
2.24	Запуск . . . . .	21
2.25	Вывод . . . . .	21
3.1	Создание lab9-4.asm . . . . .	23
3.2	Запуск . . . . .	24
3.3	Код . . . . .	25
3.4	Запуск . . . . .	26
3.5	Установка брейкпоинта . . . . .	26
3.6	Значение регистров . . . . .	26
3.7	Значение1 . . . . .	27
3.8	Значение2 . . . . .	27
3.9	Значение3 . . . . .	28
3.10	Значение4 . . . . .	28
3.11	Значение5 . . . . .	29

3.12 Значениеб . . . . .	29
3.13 Правильный код . . . . .	30
3.14 Правильный код . . . . .	31

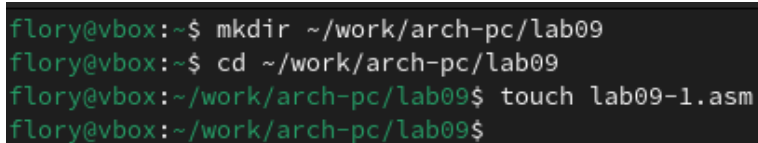
## **Список таблиц**

# 1 Цель работы

Приобрести навыки написания программ с использованием подпрограмм и ознакомиться с методами отладки при помощи GDB и его основными возможностями.

## 2 Выполнение лабораторной работы

1) Создал каталог для лабораторной работы №9 и создал файл lab9-1.asm: (Рис. 2.1)



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

Рис. 2.1: Создание lab9-1

2) Ввел код программы с использованием вызова подпрограммы: (Рис. 2.2)

```

lab9-1.asm      [----]  0 L:[ 1+35  36/ 36] *(708 / 708b) <EOF>
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: 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 [res], eax
ret ; выход из подпрограммы

```

Рис. 2.2: Код программы

3) Запуск программы: (Рис. 2.3)



```
flory@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
flory@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
flory@vbox:~/work/arch-pc/lab09$
```

Рис. 2.3: Запуск

4) Теперь изменю файл так, чтобы внутри подпрограммы была подпрограмма, которая вычисляет значение  $g(x)$ : (Рис. 2.4)

```

lab9-1.asm      [----]  9 L: [ 1+36  37/ 43] *(701 / 772b) 0010 0x00A
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',0
SECTION .bss
x: RESB 80
res: 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:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
ret ; выход из подпрограммы

```

Рис. 2.4: Замена

5) Запустил измененный файл: (Рис. 2.5)

```
flory@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
f(g(x))=17
flory@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 1
f(g(x))=11
flory@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 3
f(g(x))=23
flory@vbox:~/work/arch-pc/lab09$
```

Рис. 2.5: Запуск измененной программы

6) Теперь создам файл lab09-2.asm с текстом программы из Листинга 9.2 и заполню его: (Рис. 2.6)

```

lab9-2.asm      [----]  8 L:[  1+20  21/ 21] *(293 / 293b) <EOF>
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

```

Рис. 2.6: Код программы

7)Запустим программу и использованием -g:(Рис. 2.7)

```

flory@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
flory@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-3.fc40
Copyright (C) 2024 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...

```

Рис. 2.7: Программа

8)Теперь запустим в отладчике при помощи команды run:(Рис. 2.8)

```

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

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.
Downloading 47.72 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6127) exited normally]
(gdb)

```

Рис. 2.8: Запуск

9)Создам брейкпоинт на метке \_start с помощью break:(Рис. 2.9)

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/flory/work/arch-pc/lab09/lab9-2

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

```

Рис. 2.9: Брейкпоинт

10)Дизассемблируем её:(Рис. 2.10)

```

(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. 2.10: Дизассемблирование

11)Теперь переключусь на отображение команд с интеловским синтаксисом с помощью команды set disassembly-flavor intel и снова дизассемблировал:(Рис. 2.11)

```

(gdb) set disassembly-flavor intel
(gdb) dis
disable      disassemble  disconnect  display
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb) 

```

Рис. 2.11: Переключение на другой синтаксис

12) Включу режим псевдографики: (Рис. 2.12)

```

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
0x8049042                add     BYTE PTR [eax],al
0x8049044                add     BYTE PTR [eax],al
0x8049046                add     BYTE PTR [eax],al

native process 6389 (asm) In: _start
(gdb) layout regs
(gdb)

```

Рис. 2.12: Включение режима

13)Информация о брейкпоинтах:(Рис. 2.13)

```

native process 6389 (asm) In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y  0x08049000  lab9-2.asm:9
          breakpoint already hit 1 time
(gdb)

```

Рис. 2.13: Вывод информации

14)Создам брейпоинт по адресу выведу информацию о брейкпоинтах:(Рис. 2.14)





```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 2.16: Информация о регистрах

17)Теперь посмотрю значение переменной msg1 по имени:(Рис. 2.17)

```

(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 2.17: Вывод значения msg1

18)Теперь посмотрю значение переменной msg2 по адресу:(Рис. 2.18)

```

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

```

Рис. 2.18: Вывод значения msg2

19)Попробую изменить первый символ переменной:(Рис. 2.19)

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

Рис. 2.19: Смена первого символа

20)Изменю второй символ переменной, обратясь по адресу:(Рис. 2.20)

```
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhllo, "
(gdb) █
```

Рис. 2.20: Смена первого символа

21)Изменю несколько символов второй переменной:(Рис. 2.19)

```
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) █
```

Рис. 2.21: Смена первого символа

22)Выведу значения регистра в строковом, двоичном и шестнадцатеричном виде и изменю значения регистра:(Рис. 2.22)

```

(gdb) print /s $edx
$1 = 8
(gdb) print /t $edx
$2 = 1000
(gdb) print /x $edx
$3 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

Рис. 2.22: Смена первого символа

В регистр записались разные значения. Всё из-за того, что мы пишем в одном случае число, а в другом строку.

23)Скопирую файл lab8-2.asm в файл lab9-3.asm и создам исполняемый файл: (Рис. 2.23)

```

flory@vbox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
flory@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
flory@vbox:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'

```

Рис. 2.23: Копирование

24)Создам брейкпоинт и запущу программу: (Рис. 2.23)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/flory/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 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 есх ; Извлекаем из стека в `есх` количество
(gdb)

```

Рис. 2.24: Запуск

25)Теперь выведу значение регистра есп и выведу значение всех элементов:  
(Рис. 2.23)

```

(gdb) x/x $esp
0xffffd030: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd1f2: "/home/flory/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd218: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd22a: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd23b: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd23d: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.25: Вывод

Из-за того, что с шагом 4 располагаются данные в стеке, для каждого элемента нужно менять значение адреса с шагом 4.

### **3 Выполнение задания для самостоятельной работы**

1)Скопировал первый файл самостоятельной работы из лабораторной работы №8 и отредактировал код :(Рис. 3.1)

```

lab9-4.asm      [----]  3 L: [ 1+ 0  1/ 46] *(3  / 518b) 0099 0x063
#include 'in_out.asm'

SECTION .data
msg db "Результат: ", 0
msg2 db "Функция: f(x) = 5(2 + x)", 0

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx, 1
    ....
    mov esi, 0

next:
    cmp ecx, 0
    jz _end

    pop eax
    call atoi

    add eax, 2
    mov ebx, 5
    mul ebx
    add esi, eax

    loop next

_end:
    mov eax, msg2
    call sprintLF

    mov eax, msg
    call sprint

    mov eax, esi
    call iprintLF

    call quit

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

```

Рис. 3.1: Создание lab9-4.asm

2)Запуск программы:(Рис. 3.2)

```
flory@vbox:~$ mc

flory@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
flory@vbox:~/work/arch-pc/lab09$ ld-m elf_i386 -o lab9-4 lab9-4.o
bash: ld-m: команда не найдена...
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
ld: невозможно найти lab9-4.o: Нет такого файла или каталога
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
flory@vbox:~/work/arch-pc/lab09$ ./lab9-4
Функция:  $f(x) = 5(2 + x)$ 
Результат: 0
flory@vbox:~/work/arch-pc/lab09$ ./lab9-4 1 2 3
Функция:  $f(x) = 5(2 + x)$ 
Результат: 60
flory@vbox:~/work/arch-pc/lab09$
```

Рис. 3.2: Запуск

3)Создам теперь файл для второй самостоятельной работы и вставлю код из листинга 9.3:(Рис. 3.3)



```

lab9-5.asm      [----] 11 L:[ 1+18 19/ 20] *(336 / 348b) 0076 0x0
#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

```

Рис. 3.3: Код

4)Соберем программу и запустим её. Видим, что ответ не правильный:(Рис. 3.4)



7) Значение регистров на 1 шаге:(Рис. 3.7)

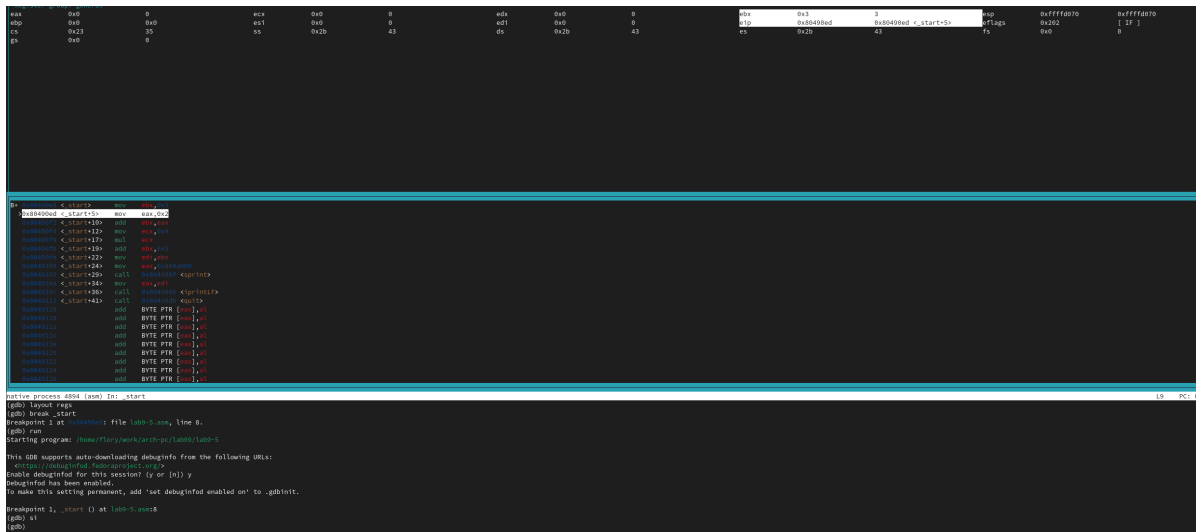


Рис. 3.7: Значение1

8) Значение регистров на 2 шаге:(Рис. 3.8)

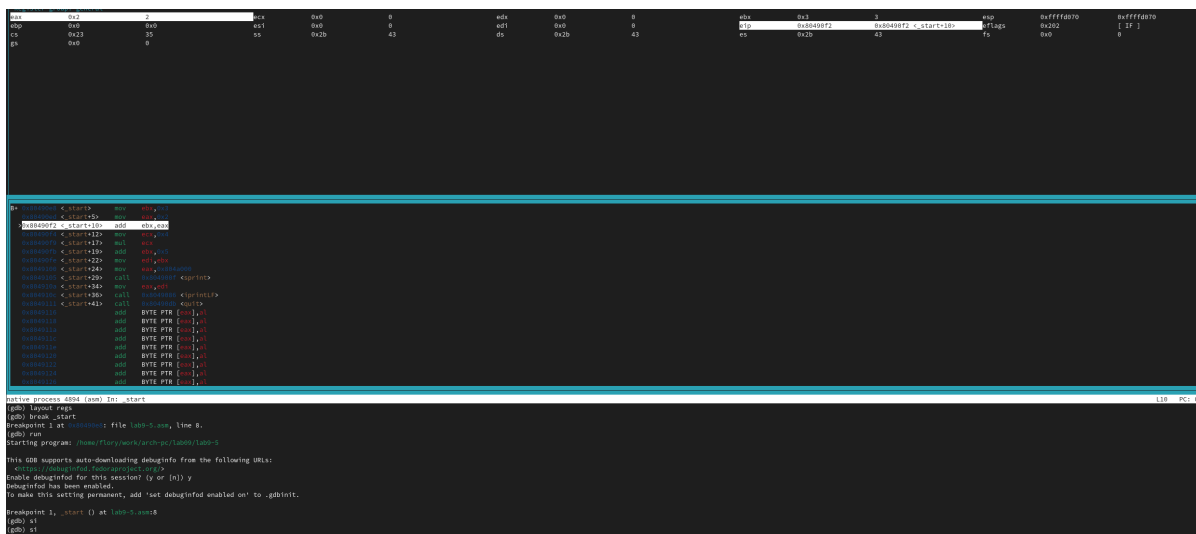


Рис. 3.8: Значение2

9) Значение регистров на 3 шаге:(Рис. 3.9)





Рис. 3.11: Значение5

## 12) Значение регистров на 6 шаге:(Рис. 3.12)



Рис. 3.12: Значение6

Ошибка заключается в том, что мы умножали значение регистра ebx, а нужно было умножать значение регистра ecx. А результаты хранить в регистре eax.

## 13) Теперь изменю код:(Рис. 3.13)

```

%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

```

Рис. 3.13: Правильный код

14) Запущу его и вижу, что ответ правильный:(Рис. 3.14)

```
flory@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
flory@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
flory@vbox:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
flory@vbox:~/work/arch-pc/lab09$
```

Рис. 3.14: Правильный код

## 4 Вывод

В ходе выполнения лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи gdb и его основными возможностями.