

Отчет по лабораторной работе № 9

Дисциплины: Архитектура компьютера

Ракутуманандзара Цантамписедрана Сарубиди

Содержание

1 Цель работы	5
2 Задание	6
3 Выполнение лабораторной работы	7
4 Выводы	24
Список литературы	25

Список иллюстраций

3.1 рис 1	7
3.2 рис 2	8
3.3 рис 3	8
3.4 рис 4	9
3.5 рис 5	9
3.6 рис 6	10
3.7 рис 7	10
3.8 рис 9	11
3.9 рис 10	11
3.10 рис 11	12
3.11 рис 12	12
3.12 рис 13	13
3.13 рис 14	13
3.14 рис 15	13
3.15 рис 16	14
3.16 рис 17	14
3.17 рис 18	14
3.18 рис 19	15
3.19 рис 20	15
3.20 рис 21	15
3.21 рис 22	15
3.22 рис 23	16
3.23 рис 24	16
3.24 рис 25	16
3.25 рис 26	17
3.26 рис 27	17
3.27 рис 28	17
3.28 рис 29	18
3.29 рис 30	18
3.30 рис 31	18
3.31 рис 32	19
3.32 рис 33	19
3.33 рис 34	20
3.34 рис 35	20
3.35 рис 36	21

Список таблиц

1 Цель работы

Цель лабораторной работы – приобретение навыков написания программ с использованием подпрограмм. Введение в методы отладки с использованием GDB и его основные возможности.

Цель данного ш

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Задание для самостоятельной работы

3 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Я создам каталог для lab09, зайду в него и создам файл lab09-1.asm(рис 1)

```
tsanta@tsanta-VirtualBox:~$ mkdir ~/work/os-intro/lab9
tsanta@tsanta-VirtualBox:~$ cd
tsanta@tsanta-VirtualBox:~$ cd ~/work/os-intro/lab9
tsanta@tsanta-VirtualBox:~/work/os-intro/lab9$ touch lab09-1.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab9$
```

Рис. 3.1: рис 1

Рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы _calcul. В этом примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Я скопирую текст программы ниже и скопирую его в созданный мной файл(рис 2)

```

%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:
    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
_calcul:
    mov ebx, 2
    mul ebx

```

Рис. 3.2: рис 2

Я создам исполняемый файл и проверю его работу(рис 3)

```

tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ nasm -f elf lab09-1.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ ld -m elf_i386 -o lab09-1 lab09-1.o
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ ./lab09-1
Введите x: 4
2x+7=15
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ 

```

Рис. 3.3: рис 3

Я отредактирую программу так, чтобы она решала функцию $f(g(x))$, где $f(x)=2x+7$ и $g(x)=3x-1$ (рис 4)

The screenshot shows a code editor window with two tabs: 'lab09-1.asm' and 'tsanta.lab9.md'. The 'lab09-1.asm' tab contains the following assembly code:

```
%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:
    call _subcalcul
    mov ebx,2
```

Рис. 3.4: рис 4

Я создам исполняемый файл и проверю его работу(рис 5)

```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ nasm -f elf lab09-1.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ ld -m elf_i386 -o lab09-1 lab09-1
.o
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$ ./lab09-1
Введите x: 4
2(3x-1)+7=2
tsanta@tsanta-VirtualBox:~/work/os-intro/lab$
```

Рис. 3.5: рис 5

2. Отладка программам с помощью GDB

Я создам новый файл lab09-2asm и скопирую в него данную программу(рис 6)

```
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. 3.6: рис 6

Я создам исполняемый файл и запущу его с помощью отладчика GDB. Чтобы работать с GDB, мне нужно добавить в исполняемый файл отладочную информацию; для этого программы необходимо переводить с ключом «-g»(рис 7)

```
tsanta@tsanta-VirtualBox:~/work/os-intro/labs$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/labs$ ld -m elf_i386 -o lab09-2 lab09-2.o
tsanta@tsanta-VirtualBox:~/work/os-intro/labs$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
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-linux-gnu".
Type "show configuration" for configuration details.
```

Рис. 3.7: рис 7

Я протестирую программу, запустив ее в оболочке GDB с помощью команды запуска(рис 8)

```
(gdb) r
Starting program: /home/tsanta/work/os-intro/lab9/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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 8037) exited normally]
(gdb)
```

Для более детального анализа программы я поставлю точку останова на метку _start, с которой начинается выполнение любой ассемблерной программы, и запущу ее(рис 9)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) r
Starting program: /home/tsanta/work/os-intro/lab9/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 3.8: рис 9

Я буду просматривать дизассемблированный код программы с помощью команды дизассемблирования, начиная с метки _start(рис 10)

The screenshot shows the GDB interface with the assembly code starting at address 0x8049000. The assembly code includes instructions like mov eax, 4 and int 0x80. The register window shows values for eax, ecx, edx, ebx, esp, and ebp. The command window at the bottom shows the user entering commands like 'si' to step through the code.

```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab9
```

Register group: general		
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffce00	0xfffffce00
ebp	0x0	0x0

```
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, 0x804a000

native process 8339 (asm) In: _start                                L14   PC: 0x8049016
breakpoint already hit 1 time
z      breakpoint      keep y  0x08049000 lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 3.9: рис 10

Я переключусь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel(рис 11)

```
(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.
(gdb)
```

Рис. 3.10: рис 11

Для более удобного анализа программы включу режим псевдографики(рис 12)

```
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xfffffce00  0xfffffce00
ebp          0x0          0x0

B+>0x08049000 <_start>    mov    eax,0x4
    0x08049005 <_start+5>  mov    ebx,0x1
    0x0804900a <_start+10> mov    ecx,0x804a000
    0x0804900f <_start+15> mov    edx,0x8
    0x08049014 <_start+20> int    0x80
    0x08049016 <_start+22> mov    eax,0x4

native process 8339 (asm) In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb) █
```

Рис. 3.11: рис 12

В Intel все начинается с адреса, затем с источника, а в ATT наоборот

2.1. Добавление точек останова

На предыдущих шагах точка останова была установлена по имени метки (_start).

Я проверю это с помощью команды info Breakpoints (сокращенно i b)(рис 13)

```
native process 8339 (asm) In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type            Disp Enb Address    What
1        breakpoint      keep y  0x08049000 lab09-2.asm:9
                                breakpoint already hit 1 time
(gdb)
```

Рис. 3.12: рис 13

Я поставлю еще одну точку останова по адресу инструкции(рис 14)

```
Breakpoint 1 already set at line
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 3.13: рис 14

Теперь я посмотрю информацию обо всех установленных точках останова.

```
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 9.
(gdb) i b
Num      Type            Disp Enb Address    What
1        breakpoint      keep y  0x08049000 lab09-2.asm:9
                                breakpoint already hit 1 time
2        breakpoint      keep y  0x08049000 lab09-2.asm:9
(gdb)
```

Рис. 3.14: рис 15

2.2. Работа с данными программы в GDB

Я выполню 5 инструкций с помощью команды Stepi (или Si)(рис 16)

The screenshot shows a terminal window with GDB running. At the top, it says "tsanta@tsanta-VirtualBox: ~/work/os-intro/lab9". Below that is a table titled "Register group: general" showing the values of various registers:

	eax	ecx	edx	ebx	esp	ebp
eax	0x8	0x804a000	134520832			
ecx	0x8	0x804a000	134520832			
edx	0x8	0x804a000	134520832			
ebx	0x1	0x804a000	134520832	0x1	0xfffffce00	0x0
esp	0x8	0x804a000	134520832	0x1	0xfffffce00	0x0
ebp	0x0	0x804a000	134520832	0x1	0xfffffce00	0x0

Below the registers, the assembly code is displayed:

```

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

```

At the bottom, the command history shows several "si" commands being entered sequentially.

Рис. 3.15: рис 16

Значения eax,ecx,esp и edx изменились

Содержимое регистров также можно просмотреть с помощью команды info Registers(рис 17)

The screenshot shows the output of the "info registers" command in GDB:

```

native process 8339 (asm) In: _start          L14  PC: 0x8049016
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xfffffce00  0xfffffce00
ebp      0x0          0x0
esi      0x0          0
--Type <RET> for more, q to quit, c to continue without paging-- 

```

Рис. 3.16: рис 17

С помощью команды x & можно посмотреть содержимое переменной. Я поищу значение переменной msg1 по имени(рис 18)

The screenshot shows the output of the "x/lsb &msg1" command in GDB:

```

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

```

Рис. 3.17: рис 18

Теперь я посмотрю на значение переменной msg2 по адресу. Адрес переменной

можно определить из дизассемблированной инструкции(рис 19)

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

Рис. 3.18: рис 19

Вы можете изменить значение регистра или ячейки памяти с помощью команды set, передав ей имя или адрес регистра в качестве аргумента. Я изменю первый символ переменной msg1(рис 20)

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

Рис. 3.19: рис 20

Теперь я заменю символ во второй переменной msg2(рис 21)

```
(gdb) set {char}&msg2='n'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>:      "norld!\n\034"  
(gdb)
```

Рис. 3.20: рис 21

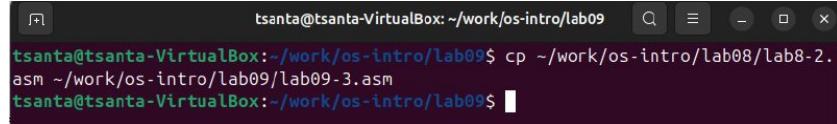
Я буду использовать команду set для изменения значения регистра ebx(рис 22)

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

Рис. 3.21: рис 22

2.3. Обработка аргументов командной строки в GDB

Я скопирую файл lab8-2.asm, созданный во время лабораторной работы 8, с помощью программы, которая печатает аргументы командной строки, в файл с именем lab09-3.asm(рис 23)



```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ cp ~/work/os-intro/lab08/lab8-2.asm ~/work/os-intro/lab09/lab09-3.asm
```

Рис. 3.22: рис 23

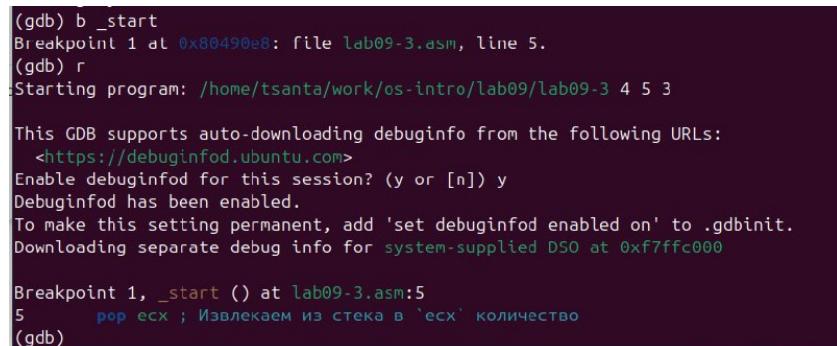
Я создам исполняемый файл и загрузим исполняемый файл в отладчик с аргументами, для загрузки программ с аргументами в gdb я буду использовать ключ -args(рис 24)



```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ nasm -f elf -g -l lab09-3.list lab09-3.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ gdb --args lab09-3 4 5 '3'
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
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.
```

Рис. 3.23: рис 24

Сначала я установлю точку останова перед первой инструкцией в программе и запущу ее(рис 25)



```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) r
Starting program: /home/tsanta/work/os-intro/lab09/lab09-3 4 5 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 3.24: рис 25

Адрес вершины стека хранится в регистре esp и по этому адресу можно увидеть число, равное количеству аргументов командной строки (включая имя программы)(рис 26)

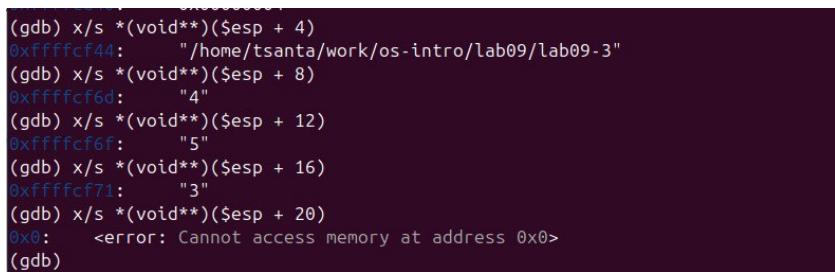


```
(gdb) x/x $esp
0xfffffc40: 0x00000004
(gdb)
```

Рис. 3.25: рис 26

Как видите, количество аргументов равно 4 – это название программы lab09-3 и сами аргументы: аргумент1, аргумент 2 и ‘аргумент 3’

Я взгляну на оставшиеся позиции стека – адрес [esp+4] хранит адрес в памяти, где находится имя программы, адрес [esp+8] хранит адрес первого аргумента, адрес [esp+12]] сохраняет второй аргумент и т. д(рис 27)



```
(gdb) x/s *(void**)( $esp + 4)
0xfffffc44: "/home/tsanta/work/os-intro/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc6d: "4"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc6f: "5"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc71: "3"
(gdb) x/s *(void**)( $esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.26: рис 27

В 32-битных компьютерах информация хранится именно так: первая память выделяется 4 бита, а вторая – 4x2

3. Задание для самостоятельной работы

Я создам новый файл с именем lab09-4.asm(рис 28)



```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ touch lab09-4.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$
```

Рис. 3.27: рис 28

Используя функцию ($f(x) = 15x - 9$) которая была у меня при выполнении lab08- задание 1, я напишу программу, которая вычисляет значение функции как подпрограмму(рис 29)

The screenshot shows a code editor window with the title bar "lab09-4.asm" and the path "~/work/os-intro/lab09". The tabs at the top include "ta.lab9.md", "lab09-1.asm", "lab09-2.asm", "lab09-3.asm", "lab09-4.asm", and "lab8-4.asm". The main code area contains the following assembly code:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(x) = 15x - 9 =',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
```

Рис. 3.28: рис 29

Я создам исполняемый файл и запущу его(рис 30)

```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ nasm -f elf lab09-4.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ ./lab09-4
Введите x: 4
f(x) = 15x - 9 =51
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$
```

Рис. 3.29: рис 30

Я создам новый файл с именем lab09-5.asm(рис 31)

```
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$ touch lab09-5.asm
tsanta@tsanta-VirtualBox:~/work/os-intro/lab09$
```

Рис. 3.30: рис 31

Я скопирую данную программу, которая вычисляет значение $(3+2)^*4+5$ (рис 32)

The screenshot shows a code editor window with the tab bar at the top. The active tab is 'lab09-5.asm' located at '~/work/os-intro/lab09'. Other tabs include '09-1.asm', 'lab09-2.asm', 'lab09-3.asm', 'lab09-4.asm', and 'lab8-4.asm'. The assembly code in the editor is as follows:

```
%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

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

call quit
```

Рис. 3.31: рис 32

Я создам исполняемый файл и запущу его с помощью GDB(рис 33)

The terminal window shows a GDB session for the program 'lab09-5'. The session starts with '(gdb) r' and 'Starting program: /home/tsanta/work/os-intro/lab09/lab09-5'. It then displays several configuration messages from GDB, including support for auto-downloading debuginfo and enabling debuginfod. Finally, it shows the output of the program: 'Результат: 10' and '[Inferior 1 (process 13378) exited normally]'. The prompt '(gdb)' is visible at the bottom.

Рис. 3.32: рис 33

Теперь я проверю, где ошибка: первый шаг нашей программы — сложить ebx, равный 3, и eax, равный 2, что делает ebx=5, затем она перемещает 4 в ecx и по умолчанию умножает ecx на eax. что дает eax 8. В-третьих, он добавит ebx к ebx, в результате чего получится 10(рис 34).

```
Register group: general
eax    0x8          8          ecx    0x4          4
edx    0x0          0          ebx    0x5          5
esp    0xfffffc50  0xfffffc50  ebp    0x0          0x0
esi    0x0          0          edi    0x0          0
ebp    0x80490fb  0x80490fb <_start+19>  eflags  0x286  [ PF IF ]
cs     0x23         35         ss     0x2b         43
ds     0x2b         43         es     0x2b         43
fs     0x0          0          gs     0x0          0

B+ 0x80490e8 <_start>    mov    ebx, 0x3
0x80490ed <_start+5>    mov    eax, 0x2
0x80490f2 <_start+10>   add    ebx, eax
0x80490f4 <_start+12>   mov    ecx, 0x2
0x80490f9 <_start+17>   mul    ecx
0x80490fb <_start+19>   add    ebx, 0x5
0x804911f <_start+22>   mov    edi, bx
0x8049180 <_start+24>   mov    eax, 0x04a000

native process 13901 (asm) in: _start
L15  PC: 0x80490fb

(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/tsanta/work/os-intro/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:10
(gdb) si 4
(gdb) si
(gdb) s
```

Рис. 3.33: рис 34

Я изменю программу так, чтобы она давала мне правильный ответ(рис 35)

Open File

lab09-5.asm
~/work/os-intro/lab09

lab09-1.asm lab09-2.asm lab09-3.asm lab09-4.asm lab08-4.asm lab09-5.asm

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

    mov ebx,3
    mov eax,2
    add 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
```

Рис. 3.34: рис 35

Теперь я создам исполняемый файл и запущу его(рис 36)

```

Reading symbols from /home/tsanta/work/os-intro/lab09/lab09-5...
(gdb) r
Starting program: /home/tsanta/work/os-intro/lab09/lab09-5

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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.
Результат: 25
[Inferior 1 (process 14295) exited normally]
(gdb) █

```

Рис. 3.35: рис 36

Текстовая программа для самостоятельной работы 1

```
%include 'in_out.asm'

SECTION .data

msg: DB 'Введите x: ',0
result: DB 'f(x) = 15x - 9 =',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:
    mov ebx, 15
    mul ebx
    sub eax, 9
    mov [res],eax
ret
```

Текстовая программа для самостоятельной работы 2

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

```
SECTION .data
div: DB 'Результат: ',0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    mov ebx,3
```

```
    mov eax,2
```

```
    add 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
```

4 Выводы

В ходе лабораторной работы я приобрел навыки написания программ с использованием подпрограмм. А также введение в методы отладки с использованием GDB и его основные возможности.

Список литературы

1. Архитектура ЭВМ