

# **Отчёта по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Жозе Рамос Домингуш

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Реализация подпрограмм в NASM . . . . .	6
2.2	Отладка программ с помощью GDB . . . . .	8
<b>3</b>	<b>Выводы</b>	<b>19</b>

# Список иллюстраций

2.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code> . . . . .	6
2.2	Заполняем файл . . . . .	6
2.3	Запускаем файл и проверяем его работу . . . . .	7
2.4	Изменяем файл, добавляя еще одну подпрограмму . . . . .	7
2.5	Запускаем файл и смотрим на его работу . . . . .	7
2.6	Создаем файл . . . . .	8
2.7	Заполняем файл . . . . .	8
2.8	Загружаем исходный файл в отладчик . . . . .	8
2.9	Запускаем программу командой <code>run</code> . . . . .	9
2.10	Запускаем программу с брейкпоинтом . . . . .	9
2.11	Смотрим дисассимилированный код программы . . . . .	9
2.12	Переключаемся на синтаксис Intel . . . . .	10
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы . . . . .	11
2.14	Используем команду <code>info breakpoints</code> и создаем новую точку останова	11
2.15	Смотрим информацию . . . . .	12
2.16	Отслеживаем регистры . . . . .	12
2.17	Смотрим значение переменной . . . . .	12
2.18	Смотрим значение переменной . . . . .	13
2.19	Меняем символ . . . . .	13
2.20	Меняем символ . . . . .	13
2.21	Смотрим значение регистра . . . . .	13
2.22	Изменяем регистр командой <code>set</code> . . . . .	13
2.23	Прописываем команды <code>c</code> и <code>quit</code> . . . . .	14
2.24	Копируем файл . . . . .	14
2.25	Создаем и запускаем в отладчике файл . . . . .	14
2.26	Устанавливаем точку останова . . . . .	14
2.27	Изучаем полученные данные . . . . .	15
2.28	Копируем файл . . . . .	15
2.29	Изменяем файл . . . . .	15
2.30	Проверяем работу программы . . . . .	16
2.31	Создаем файл . . . . .	16
2.32	Изменяем файл . . . . .	16
2.33	Создаем и смотрим на работу программы(работает неправильно)	16
2.34	Ищем ошибку регистров в отладчике . . . . .	17
2.35	Меняем файл . . . . .	17

2.36 Создаем и запускаем файл(работает корректно) . . . . .	18
---	----

# 1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

## 2 Выполнение лабораторной работы

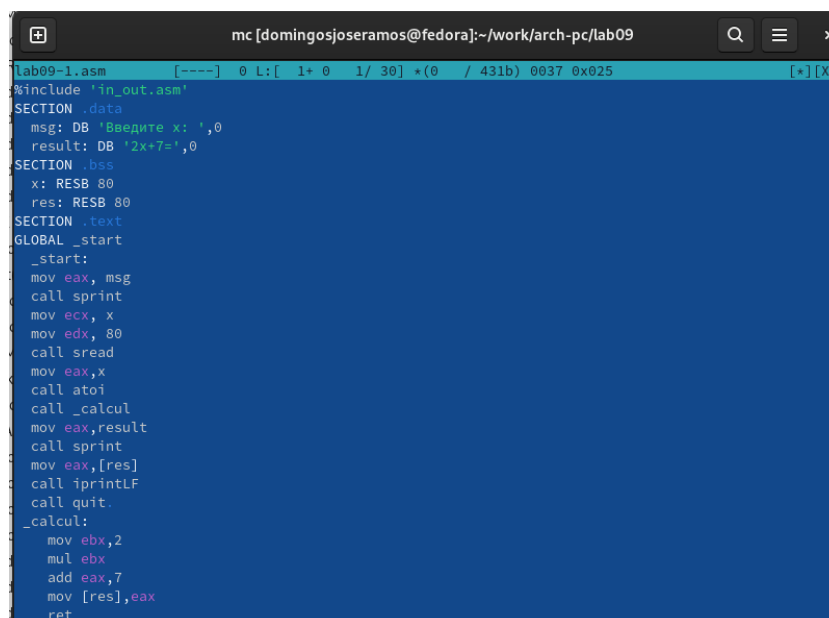
### 2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. fig. 2.1).

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

Рис. 2.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. fig. 2.2).



```
mc [domingosjoseramos@fedora]:~/work/arch-pc/lab09
lab09-1.asm  [----]  0 L: [ 1+ 0 1/ 30] *(0 / 431b) 0037 0x025  [*] [X]
%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
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit.
_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret
```

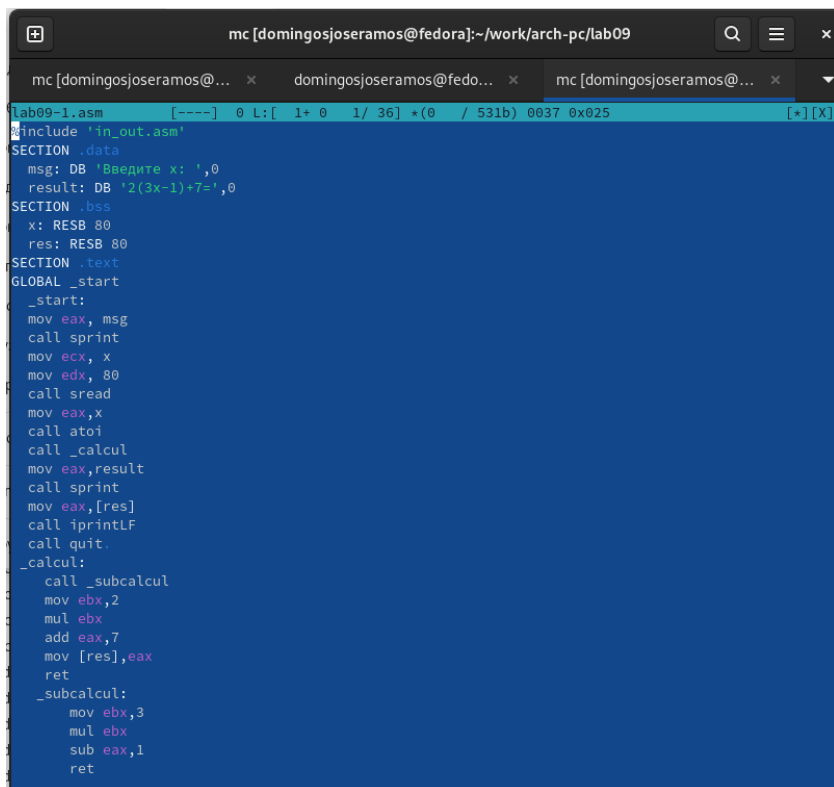
Рис. 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. fig. 2.3).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. fig. 2.4).



```
lab09-1.asm  [-----]  0 L: [ 1+ 0 1/ 36] *(0 / 531b) 0037 0x025 [*] [X]
#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 iprintfLF
call quit.
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

Рис. 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. fig. 2.5).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.5: Запускаем файл и смотрим на его работу

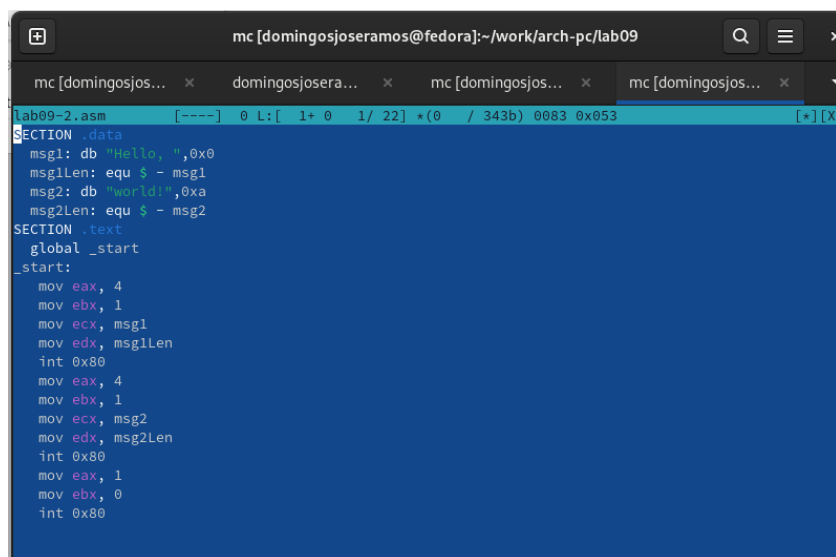
## 2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. fig. 2.6).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. fig. 2.7).



```
mc [domingosjoseramos@fedora]:~/work/arch-pc/lab09
lab09-2.asm  [-----]  0 L: [ 1+ 0 1/ 22] *(0 / 343b) 0083 0x053  [*] [X]
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. 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. fig. 2.8).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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 lab09-2...
(gdb)
```

Рис. 2.8: Загружаем исходный файл в отладчик



Запускаем команду в отладчике (рис. fig. 2.9).

```
(gdb) run
Starting program: /home/domingosjoseramos/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4572) exited normally]
(gdb) █
```

Рис. 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. fig. 2.10).

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

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

Рис. 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`(рис. fig. 2.11).

```
(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) set disassembly-flavor intel █
```

Рис. 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel 'овским синтаксисом (рис. fig. 2.12).

```

(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. 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel :

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (bytes), “w” (word), “L” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “l” и “q”.

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “*.Intel*”.

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. fig. 2.13).

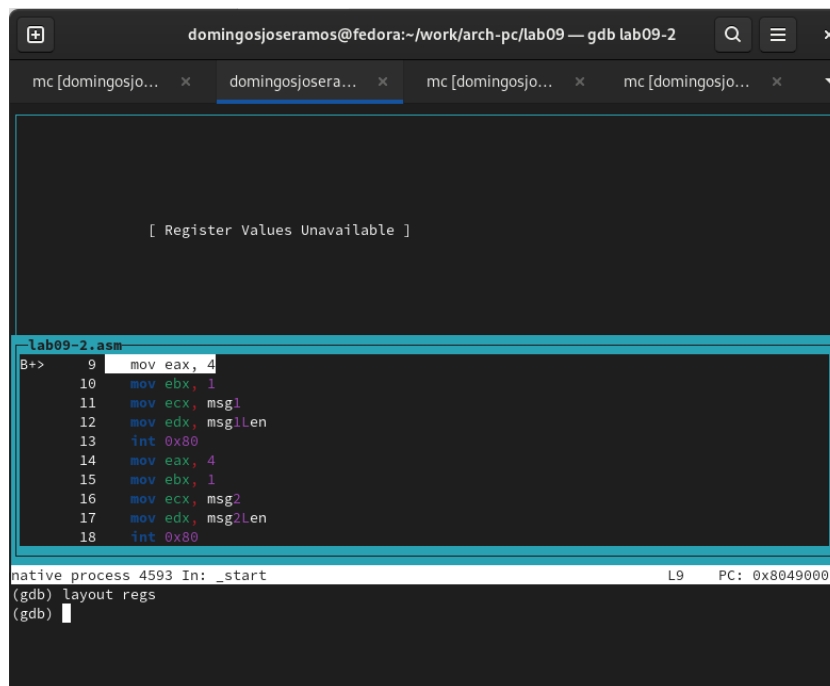


Рис. 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. 2.14).

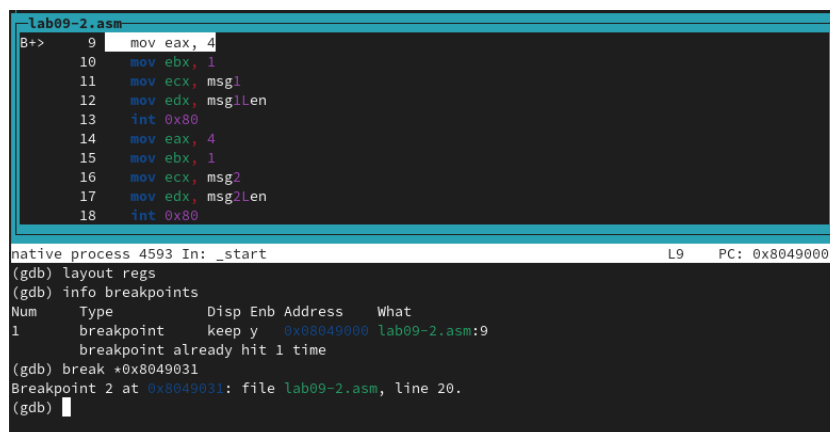


Рис. 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. fig. 2.15).

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
2        breakpoint already hit 1 time
3        breakpoint keep y  0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. fig. 2.16).

The screenshot shows the GDB interface with the following content:

Register group: general

Register	Value	Comment
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffcf70	0xffffcf70
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[ IF ]
cs	0x23	35

lab09-2.asm

```
B+ 9  mov eax, 4
    10 mov ebx, 1
    11 mov ecx, msg1
    12 mov edx, msg1Len
    13 int 0x80
    > 14 mov eax, 4
    15 mov ebx, 1
    16 mov ecx, msg2
    17 mov edx, msg2Len
    18 int 0x80
```

native process 4593 In: \_start L14 PC: 0x8049016

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
2        breakpoint already hit 1 time
3        breakpoint keep y  0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. fig. 2.17).

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

Рис. 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. fig. 2.18).

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

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. fig. 2.19).

```
(gdb) set {char}&msg1='h'
(gdb) x/lb &msg1
0x804a008 <msg1>: "hello, "
(gdb)
```

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. fig. 2.20).

```
(gdb) x/lb &msg2
0x804a008 <msg2>: "Lor!d!\n\034"
(gdb)
```

Рис. 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. fig. 2.21).

```
(gdb) p/t $edx
$1 = 1000
(gdb) p/s $edx
$2 = 8
(gdb) p/s $edx
$3 = 8
(gdb) p/x $edx
$4 = 0x8
(gdb)
```

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. fig. 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рис. 2.22: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. fig. 2.23).

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)
```

Рис. 2.23: Прописываем команды с и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. fig. 2.24).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. fig. 2.25).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
GNU gdb (Fedora Linux) 14.2-1.fc40
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 lab09-3...
(gdb)
```

Рис. 2.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. fig. 2.26).

```
(gdb) b _start
Note: breakpoint 1 also set at pc 0x80490e8.
Breakpoint 2 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/domingosjoseramos/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xffff700: 0x00000004
(gdb)
```

Рис. 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. fig. 2.27).



Создаем исполняемый файл и запускаем его (рис. fig. 2.30).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 5
3(10*x)=45
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.30: Проверяем работу программы

### ###Задание 2

Создаем новый файл в дирректории (рис. fig. 2.31).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ touch lab09-5.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. fig. 2.32).

```
mc [domingosjoseramos@fedora:~/work/arch-pc/lab09]
... x d... x ... x d... x ... x d... x ... x d... x ... x
lab09-5.asm [---] 0 L: 1+ 0 1/ 19) *(0 / 285b) 0037 0x025 [+]X
%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 ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx
    mov eax,div
    call sprint
    mov eax,edi
    call iprintf
    call quit
```

Рис. 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. fig. 2.33).

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

Рис. 2.33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. fig. 2.34).



The screenshot shows a GDB window with the title 'domingosjoseamos@fedora:~/work/arch-pc/lab09 — gdb --args lab09-3...'. The 'Register group: general' is expanded, showing registers: eax (0xffffd157, -11945), ecx (0x3, 3), edx (0xffffd124, -11996), ebx (0x0, 0), esp (0xffffcf50, 0xffffcf50), ebp (0x0, 0x0), esi (0x0, 0), edi (0x0, 0), eip (0x8049000, 0x8049000 <slen>), eflags (0x206, [ PF IF ]), and cs (0x23, 35). Below the registers, the assembly code for 'in\_out.asm' is displayed, starting with a comment in Russian: '2 ; Функция вычисления длины сообщения'. The code includes instructions like 'push ebx', 'mov ebx, eax', and a loop 'nextchar:' with 'cmp byte [eax], 0', 'jz finished', 'inc eax', and 'jmp nextchar'. At the bottom, the status bar shows 'native process 6004 In: slen', 'L4', and 'PC: 0x804'.

```
Register group: general
eax      0xffffd157      -11945
ecx      0x3            3
edx      0xffffd124     -11996
ebx      0x0            0
esp      0xffffcf50     0xffffcf50
ebp      0x0            0x0
esi      0x0            0
edi      0x0            0
eip      0x8049000      0x8049000 <slen>
eflags   0x206          [ PF IF ]
cs       0x23           35

in_out.asm
2 ; Функция вычисления длины сообщения
3 slen:
> 4 push ebx
5 mov ebx, eax
6
7 nextchar:
8 cmp byte [eax], 0
9 jz finished
10 inc eax
11 jmp nextchar

native process 6004 In: slen L4 PC: 0x804
(gdb) si
(gdb) si
```

Рис. 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. fig. 2.35).

The screenshot shows a text editor window with the title 'mc [domingosjoseamos@fedora]:~/work/arch-pc/lab09'. The file 'lab09-5.asm' is open, showing assembly code. It includes a header line with flags: 'lab09-5.asm [----] 0 L:[ 1+ 0 1/ 19] +(0 / 285b) 0037 0x025'. The code starts with '%include "in\_out.asm"', followed by 'SECTION .data' and 'div: DB "Результат: ",0'. Then 'SECTION .text' and 'GLOBAL \_start' are shown. The main code block starts with '\_start:' and contains instructions: 'mov eax,3', 'mov ebx,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', and 'call quit'.

```
lab09-5.asm [----] 0 L:[ 1+ 0 1/ 19] +(0 / 285b) 0037 0x025
%include "in_out.asm"
SECTION .data
div: DB "Результат: ",0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,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. 2.35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. fig. 2.36).

```
domingosjoseramos@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
domingosjoseramos@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
domingosjoseramos@fedora:~/work/arch-pc/lab09$
```

Рис. 2.36: Создаем и запускаем файл(работает корректно)

## **3 Выводы**

Мы познакомились с методами отладки при помощи GDB и его возможностями.