

# **Отчёта по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Семенов Сергей Алексеевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
2.1	Реализация подпрограмм в NASM . . . . .	5
2.2	Отладка программ с помощью GDB . . . . .	8
<b>3</b>	<b>Выводы</b>	<b>20</b>

# Список иллюстраций

2.1	Создаем каталог с помощью команды mkdir и файл с помощью команды touch . . . . .	5
2.2	Заполняем файл . . . . .	6
2.3	Запускаем файл и проверяем его работу . . . . .	6
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 . . . . .	9
2.9	Запускаем программу командой run . . . . .	9
2.10	Запускаем программу с брейкпоинтом . . . . .	10
2.11	Смотрим дисассимилированный код программы . . . . .	10
2.12	Переключаемся на синтаксис Intel . . . . .	11
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы . . . . .	12
2.14	Используем команду info breakpoints и создаем новую точку останова	12
2.15	Смотрим информацию . . . . .	13
2.16	Отслеживаем регистры . . . . .	13
2.17	Смотрим значение переменной . . . . .	13
2.18	Смотрим значение переменной . . . . .	14
2.19	Меняем символ . . . . .	14
2.20	Меняем символ . . . . .	14
2.21	Смотрим значение регистра . . . . .	14
2.22	Прописываем команды с и quit . . . . .	15
2.23	Копируем файл . . . . .	15
2.24	Создаем и запускаем в отладчике файл . . . . .	15
2.25	Устанавливаем точку останова . . . . .	16
2.26	Изучаем полученные данные . . . . .	16
2.27	Изменяем файл . . . . .	17
2.28	Проверяем работу программы . . . . .	17
2.29	Создаем файл . . . . .	17
2.30	Изменяем файл . . . . .	18
2.31	Создаем и смотрим на работу программы(работает неправильно)	18
2.32	Ищем ошибку регистров в отладчике . . . . .	18
2.33	Меняем файл . . . . .	19
2.34	Создаем и запускаем файл(работает корректно) . . . . .	19

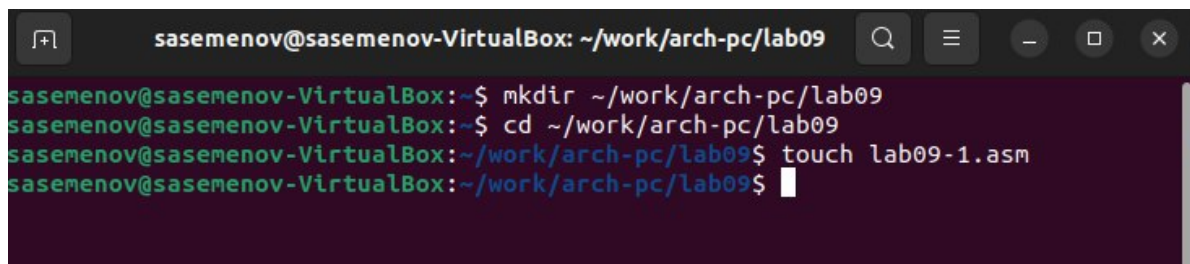
# 1 Цель работы

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

A screenshot of a terminal window with a dark background. The window title is "sasemenov@sasemenov-VirtualBox: ~/work/arch-pc/lab09". The terminal shows four lines of commands and their outputs: 1. "sasemenov@sasemenov-VirtualBox:~\$ mkdir ~/work/arch-pc/lab09" 2. "sasemenov@sasemenov-VirtualBox:~\$ cd ~/work/arch-pc/lab09" 3. "sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09\$ touch lab09-1.asm" 4. "sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09\$" followed by a cursor. The window has standard Linux window controls (minimize, maximize, close) and a search icon in the top right.

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

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

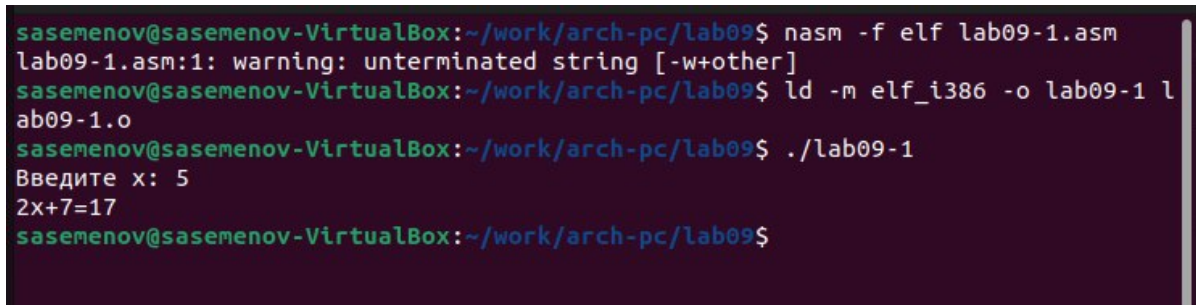
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1



```
1 %include 'in_out.asm
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы
```

Рис. 2.2: Заполняем файл

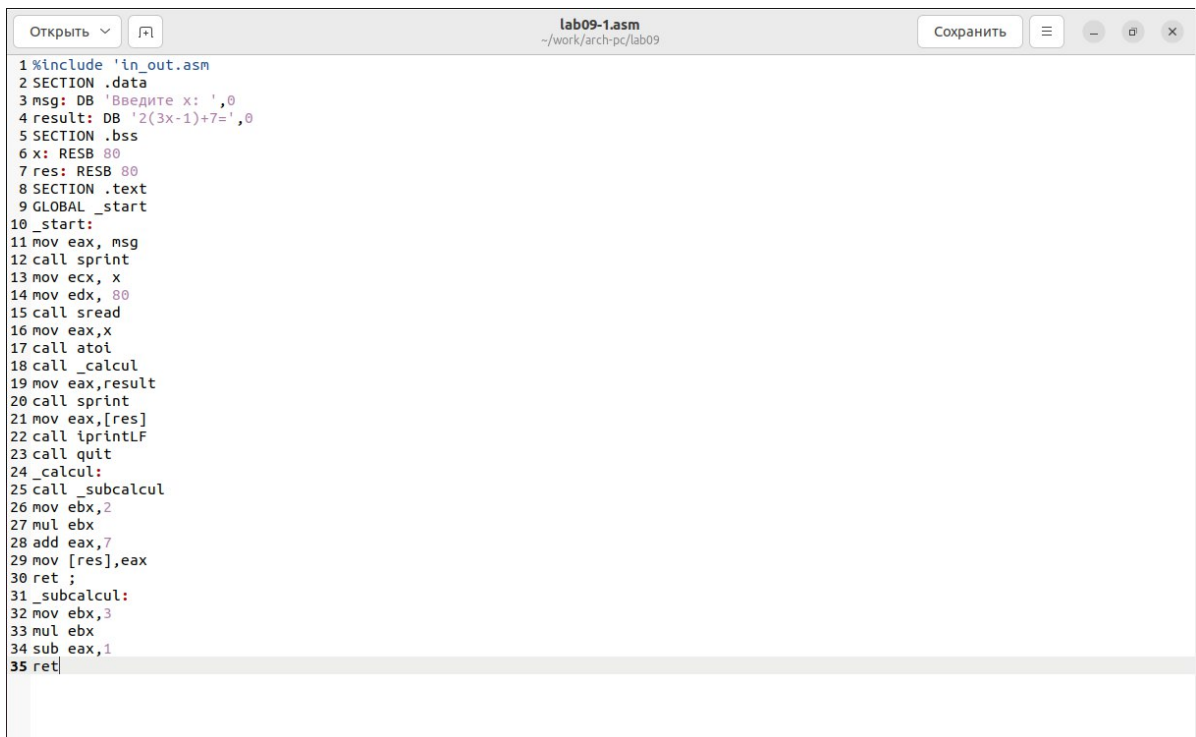
Создаем исполняемый файл и запускаем его



```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lab09-1.asm:1: warning: unterminated string [-w+other]
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.3: Запускаем файл и проверяем его работу

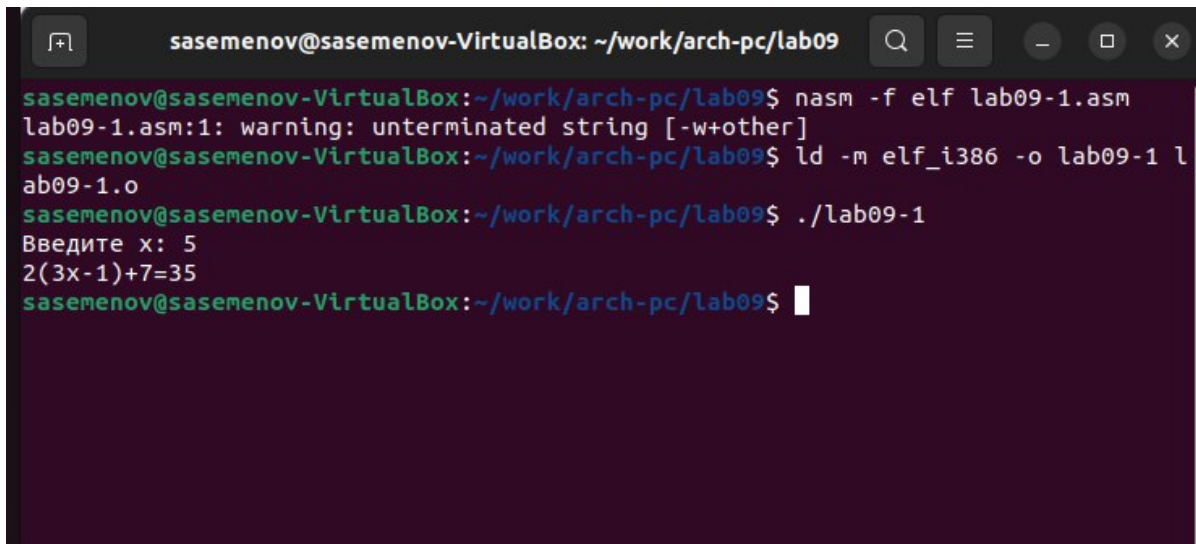
Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call tprintfLF
23 call quit
24 _calcul:
25 call _subcalcul
26 mov ebx, 2
27 mul ebx
28 add eax, 7
29 mov [res], eax
30 ret
31 _subcalcul:
32 mov ebx, 3
33 mul ebx
34 sub eax, 1
35 ret
```

Рис. 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его



```
sasemenov@sasemenov-VirtualBox: ~/work/arch-pc/lab09
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lab09-1.asm:1: warning: unterminated string [-w+other]
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
sasemenov@sasemenov-VirtualBox:~/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

```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ touch lab09-2.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2

```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
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
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb



```
sasemenov@sasemenov-VirtualBox: ~/work/arch-pc/lab09
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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.
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

```
(gdb) run
Starting program: /home/sasemenov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3081) exited normally]
(gdb) 
```

Рис. 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/sasemenov/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 брейкпоинтом

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`

```

(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. 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом

```

(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)

```

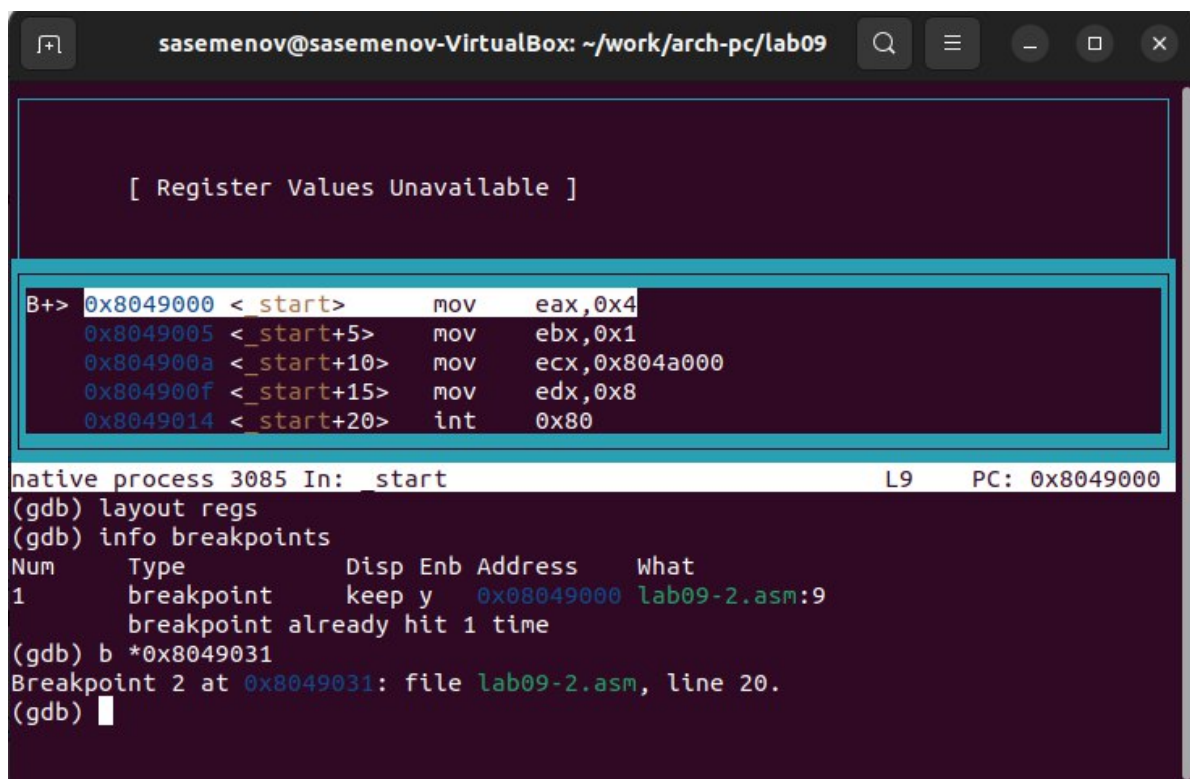
Рис. 2.12: Переключаемся на синтаксис Intel

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" (byte), "w" (word), "l" (long) и "q" (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как "b", "w", "d" 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



The screenshot shows a GDB terminal window with the title bar "sasemenov@sasemenov-VirtualBox: ~/work/arch-pc/lab09". The main display area shows "[ Register Values Unavailable ]". Below this, a list of assembly instructions is displayed, each with its address and disassembly:

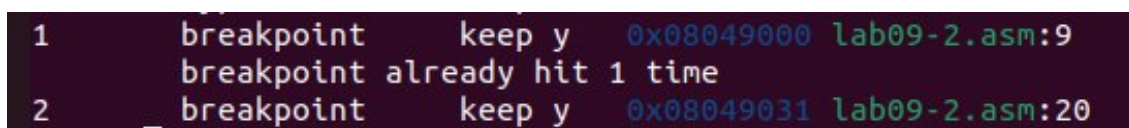
Address	Disassembly
0x8049000 <_start>	mov eax,0x4
0x8049005 <_start+5>	mov ebx,0x1
0x804900a <_start+10>	mov ecx,0x804a000
0x804900f <_start+15>	mov edx,0x8
0x8049014 <_start+20>	int 0x80

Below the assembly list, the GDB prompt shows the current state of the process:

```
native process 3085 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) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

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

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции



The screenshot shows the output of the 'info breakpoints' command in GDB:

```
1      breakpoint      keep y 0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
2      breakpoint      keep y 0x08049031 lab09-2.asm:20
```

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

Посмотрим информацию о всех установленных точках останова

```
sasemenov@sasemenov-VirtualBox: ~/work/arch-pc/lab09

[ 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

native process 3085 In: _start          L9      PC: 0x80
(gdb) layout regs
```

Рис. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si.

```
(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

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

Рис. 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу



```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\
(gdb)
```

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1

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

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2

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

Рис. 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах

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

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
```

Выводятся разные значения, так как команда

без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB

```
(gdb) c
Continuing.
World!

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

Рис. 2.22: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm

```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ touch lab09-3.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.23: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB

```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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.
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)
```

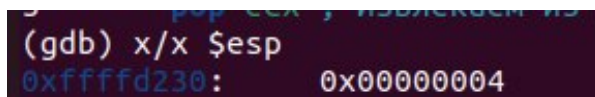
Рис. 2.24: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее

Устанавливаем точку останова

Рис. 2.25: Устанавливаем точку останова

Смотрим позиции стека по разным адресам



```
(gdb) x/x $esp
0xffffd230: 0x00000004
```

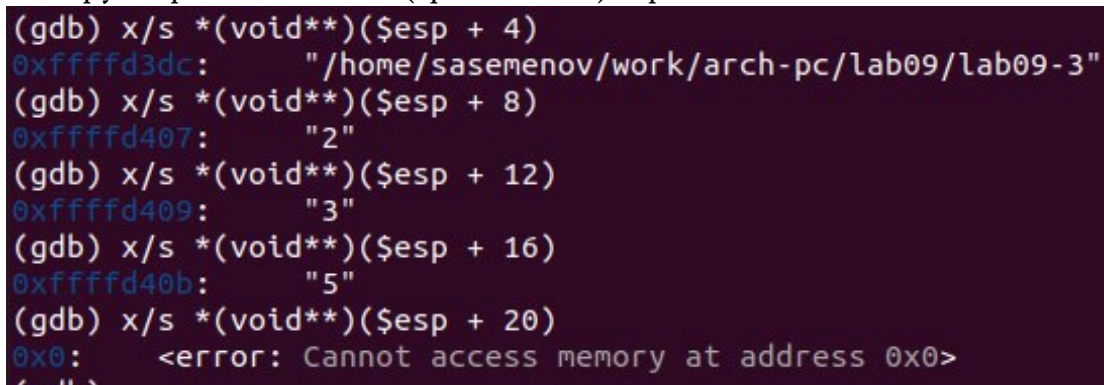
Рис. 2.26: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

###Задание 1

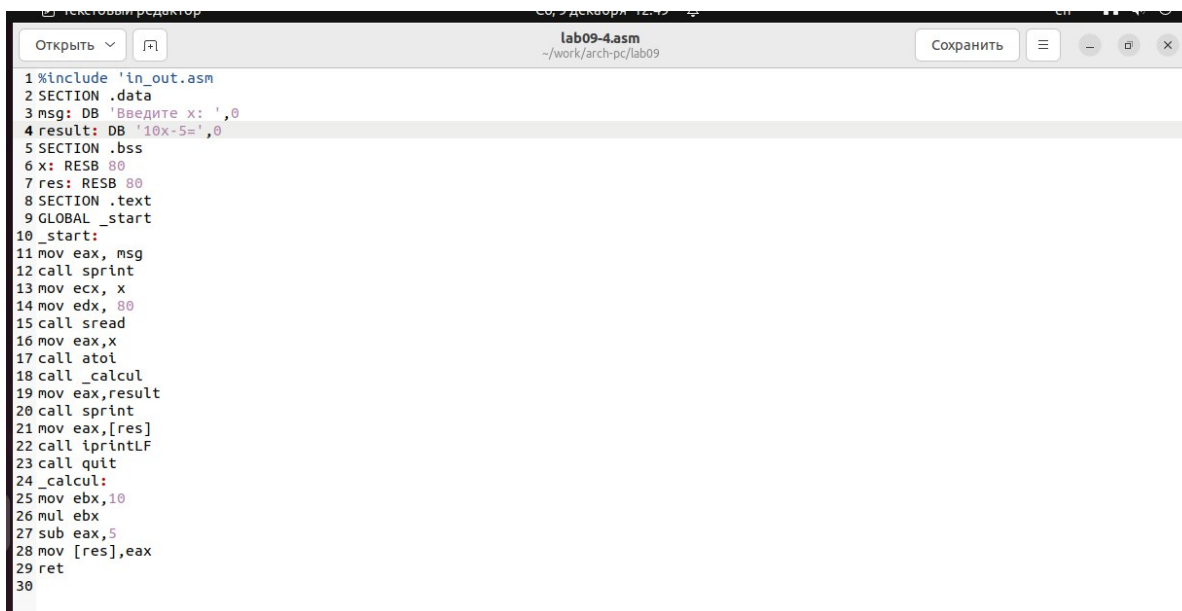
Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm



```
(gdb) x/s *(void**)($esp + 4)
0xffffd3dc: "/home/sasemenov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd407: "2"
(gdb) x/s *(void**)($esp + 12)
0xffffd409: "3"
(gdb) x/s *(void**)($esp + 16)
0xffffd40b: "5"
(gdb) x/s *(void**)($esp + 20)
0x0: <error: Cannot access memory at address 0x0>
```

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму





```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '10x-5=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 mov ebx, 10
26 mul ebx
27 sub eax, 5
28 mov [res], eax
29 ret
30
```

Рис. 2.27: Изменяем файл

Создаем исполняемый файл и запускаем его

```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
lab09-4.asm:1: warning: unterminated string [-w+other]
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 10
10x-5=95
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.28: Проверяем работу программы

### ###Задание 2

Создаем новый файл в дирректории

```
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$ touch lab09-5.asm
sasemenov@sasemenov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.29: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3

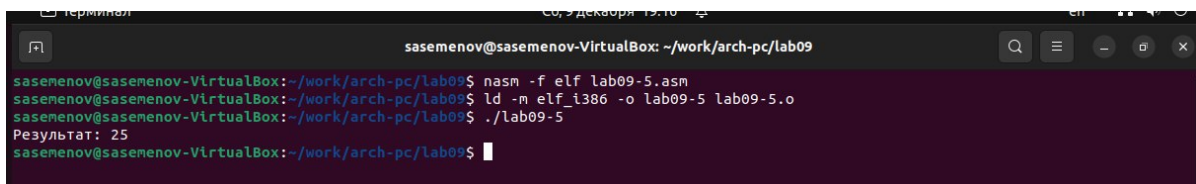
```

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

```

Рис. 2.30: Изменяем файл

Создаем исполняемый файл и запускаем его



```

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

```

Рис. 2.31: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si

Ищем ошибку регистров в отладчике

Рис. 2.32: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы

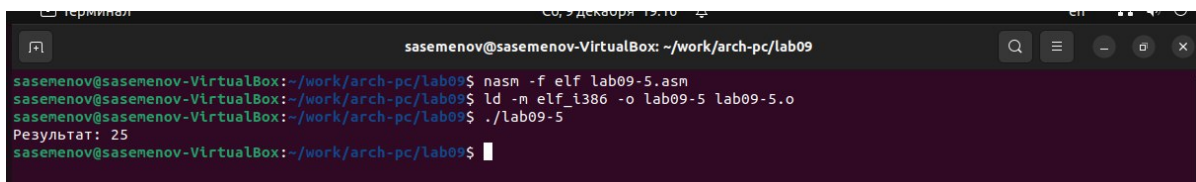
```

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

```

Рис. 2.33: Меняем файл

Создаем исполняемый файл и запускаем его



```

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

```

Рис. 2.34: Создаем и запускаем файл(работает корректно)

## **3 Выводы**

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