

# **Отчёта по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Виме Давид Тененте

# 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>22</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 .....   | 7  |
| 2.3 Запускаем файл и проверяем его работу .....  | 7  |
| 2.4 Изменяем файл, добавляя еще одну подпрограмму .....  | 8  |
| 2.5 Запускаем файл и смотрим на его работу .....   | 8  |
| 2.6 Создаем файл .....   | 9  |
| 2.7 Заполняем файл .....   | 9  |
| 2.8 Загружаем исходный файл в отладчик .....   | 10 |
| 2.9 Запускаем программу командой <code>run</code> .....  | 10 |
| 2.10 Запускаем программу с брейкпоинтом .....  | 10 |
| 2.11 Смотрим дисассимилированный код программы .....   | 11 |
| 2.12 Переключаемся на синтаксис Intel .....  | 11 |
| 2.13 Включаем отображение регистров, их значений и результат дисассимилирования программы .....            | 12 |
| 2.14 Используем команду <code>info breakpoints</code> и создаем новую точку останова                       | 13 |
| 2.15 Смотрим информацию .....  | 13 |
| 2.16 Отслеживаем регистры .....  | 14 |
| 2.17 Смотрим значение переменной .....   | 14 |
| 2.18 Смотрим значение переменной .....   | 15 |
| 2.19 Меняем символ .....   | 15 |
| 2.20 Меняем символ .....   | 15 |
| 2.21 Смотрим значение регистра .....   | 15 |
| 2.22 Изменяем регистр командой <code>set</code> .....  | 16 |
| 2.23 Прописываем команды <code>c</code> и <code>quit</code> .....  | 16 |
| 2.24 Копируем файл .....   | 16 |
| 2.25 Создаем и запускаем в отладчике файл .....  | 16 |
| 2.26 Устанавливаем точку останова .....  | 17 |
| 2.27 Изучаем полученные данные .....   | 17 |
| 2.28 Копируем файл .....   | 17 |
| 2.29 Изменяем файл .....   | 18 |
| 2.30 Проверяем работу программы .....  | 19 |
| 2.31 Создаем файл .....  | 19 |
| 2.32 Изменяем файл .....   | 20 |
| 2.33 Создаем и смотрим на работу программы (работает неправильно)  | 20 |

|   |    |
|---|----|
| 2.34 Ищем ошибку регистров в отладчике .....            | 21 |
| 2.35 Меняем файл .....                                  | 22 |
| 2.36 Создаем и запускаем файл(работает корректно) ..... | 22 |

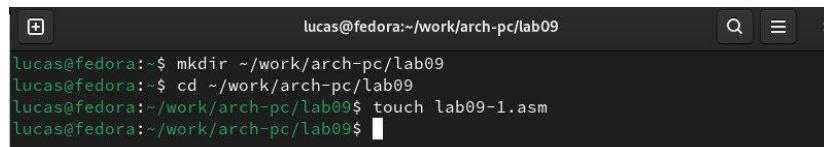
# 1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

## 2 Выполнение лабораторной работы

### 2.1 Реализация подпрограмм в NASM

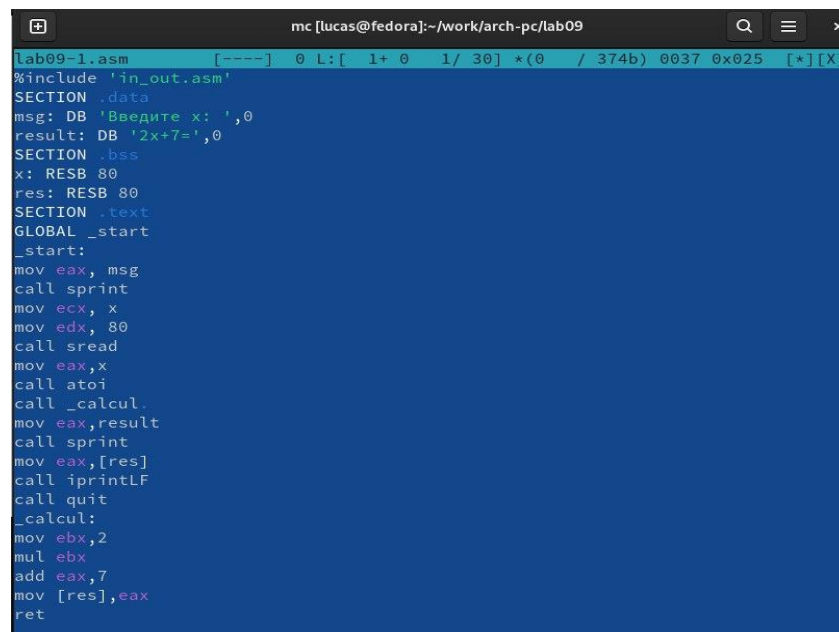
Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. 2.1).



```
lucas@fedora:~/work/arch-pc/lab09
lucas@fedora:~$ mkdir ~/work/arch-pc/lab09
lucas@fedora:~$ cd ~/work/arch-pc/lab09
lucas@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
lucas@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. 2.2).



```
lab09-1.asm  [----]  0 L: [ 1+ 0 1/ 30] *(0 / 374b) 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
```

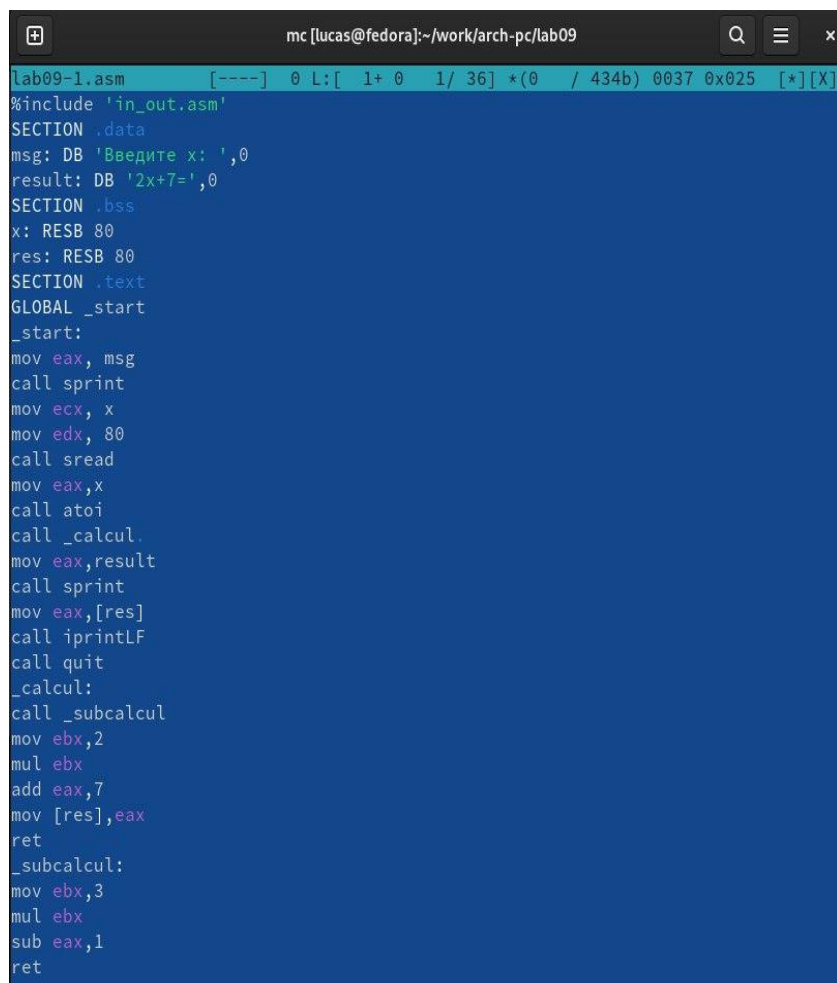
Рис. 2.2: Заполняем файл

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

```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
lucas@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
lucas@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. 2.4).



```
lab09-1.asm  [----]  0  L: [ 1+ 0  1/ 36]  *(0  / 434b)  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:
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. 2.5).

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

Рис. 2.5: Запускаем файл и смотрим на его работу

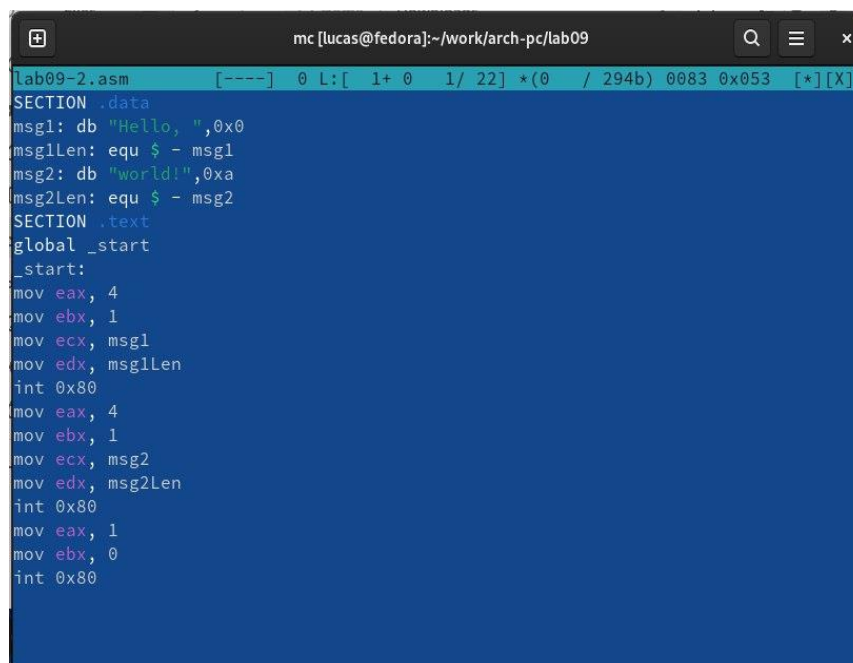
## 2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. 2.6).

```
lucas@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
lucas@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. 2.7).



```
lab09-2.asm  [----]  0 L:[ 1+ 0 1/ 22] *(0 / 294b) 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. 2.8).

```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
lucas@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) jrun
```

Рис. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. 2.9).

```
(gdb) run
Starting program: /home/lucas/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
```

Рис. 2.9: Запускаем программу командой run

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

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

Рис. 2.11: Смотрим дисассимилированный код программы

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

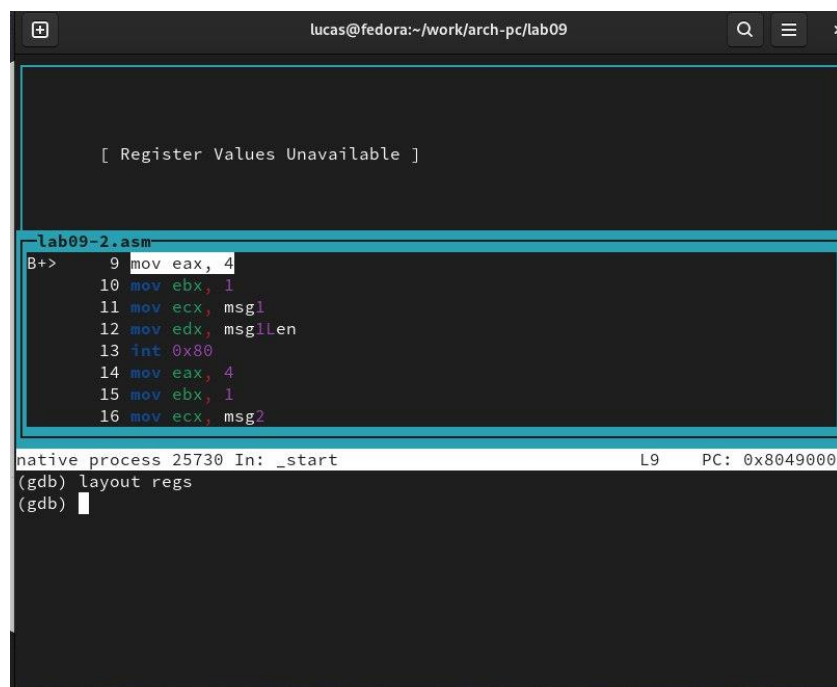
3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов,таких как “b”(byte),“w”(word), “l”(long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов,таких как “b”,“w”,“d” и “q”.

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

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. 2.13).



The screenshot shows a terminal window with the title bar "lucas@fedora:~/work/arch-pc/lab09". The main area displays assembly code from a file named "lab09-2.asm". The code is as follows:

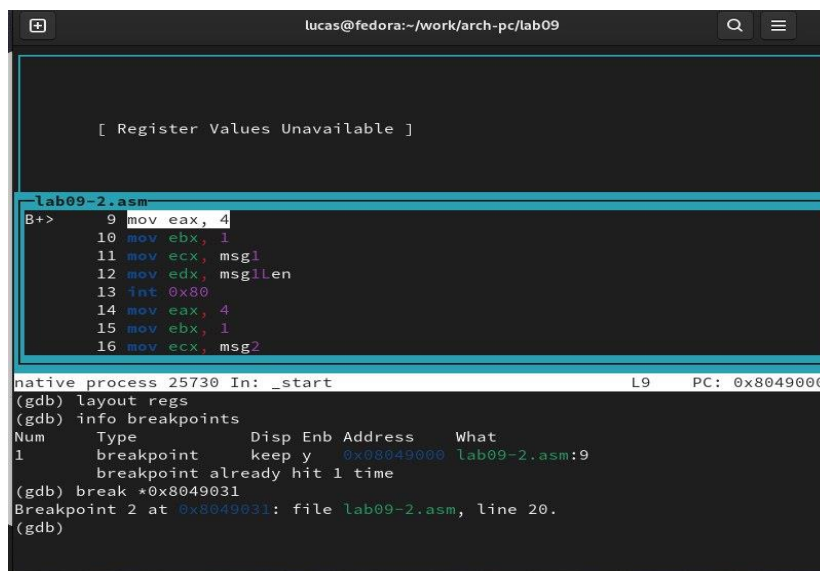
```
B+> 9 mov eax, 4
    10 mov ebx, 1
    11 mov ecx, msg1
    12 mov edx, msg1Len
    13 int 0x80
    14 mov eax, 4
    15 mov ebx, 1
    16 mov ecx, msg2
```

Below the assembly code, the terminal shows GDB output:

```
native process 25730 In: _start L9 PC: 0x8049000
(gdb) layout regs
(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 (рис. 2.14).



The screenshot shows a GDB terminal window with the following content:

```
lucas@fedora:~/work/arch-pc/lab09
[ Register Values Unavailable ]

~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

native process 25730 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) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

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

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



The screenshot shows a GDB terminal window with the following content:

```
(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   0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. 2.16).

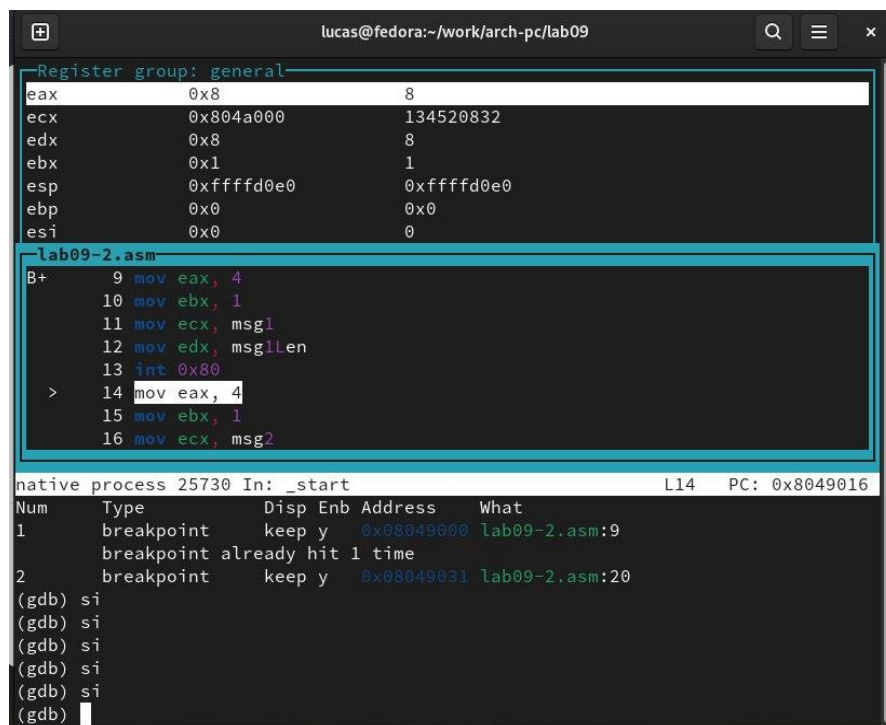


Рис. 2.16: Отслеживаем регистры

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

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

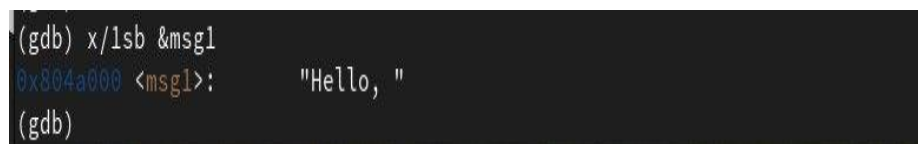


Рис. 2.17: Смотрим значение переменной

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

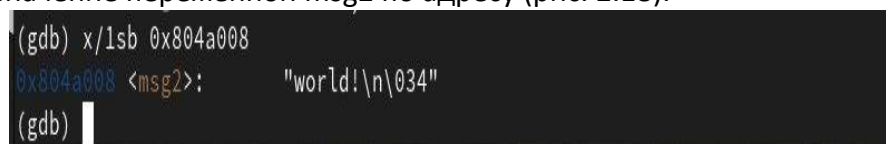


Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. 2.19).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb)
```

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. 2.20).

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

Рис. 2.20: Меняем символ

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

```
(gdb) p/t $edx
$3 = 1000
(gdb) p/s $edx
$4 = 8
(gdb) p/x $edx
$5 = 0x8
(gdb)
```

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. 2.22).

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

Рис. 2.22: Изменяем регистр командой set

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

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

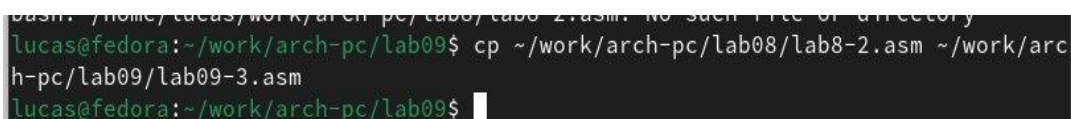


```
(gdb) c
Continuing.
World!

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

Рис. 2.23: Прописываем команды c и quit

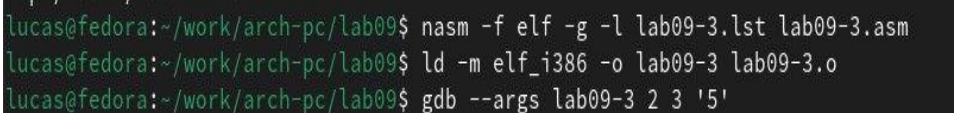
Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. 2.24).



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

Рис. 2.24: Копируем файл

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



```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
lucas@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
```

Рис. 2.25: Создаем и запускаем в отладчике файл

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

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/lucas/work/arch-pc/lab09/lab09-3 2 3 5

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 lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xfffffd0f0: 0x00000004
(gdb)

```

Рис. 2.26: Устанавливаем точку останова

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

```

(gdb) x/s *(void**)(esp + 4)
0xfffffd2a2: "/home/lucas/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffd2c9: "2"
(gdb) x/s *(void**)(esp + 12)
0xfffffd2cb: "3"
(gdb) x/s *(void**)(esp + 16)
Argument to arithmetic operation not a number or boolean.
(gdb) x/s *(void**)(esp + 16)
0xfffffd2cd: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.27: 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 ( рис. 2.28).

```

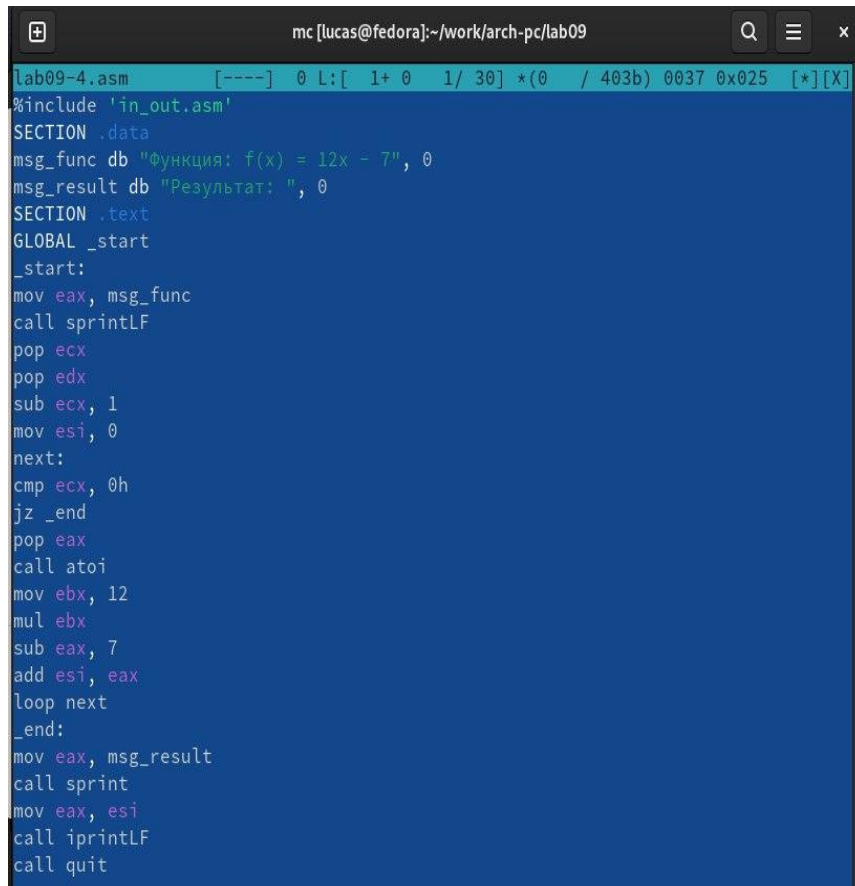
lucas@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
lucas@fedora:~/work/arch-pc/lab09$

```



Рис. 2.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. 2.29).



```
lab09-4.asm  [----]  0 L: [ 1+ 0 1/ 30] *(0 / 403b) 0037 0x025 [*][X]
#include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 12x - 7", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start
_start:
mov eax, msg_func
call sprintf
pop ecx
pop edx
sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
mov ebx, 12
mul ebx
sub eax, 7
add esi, eax
loop next
_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintLF
call quit
```

Рис. 2.29: Изменяем файл Создаем

исполняемый файл и запускаем его (рис. 2.30).

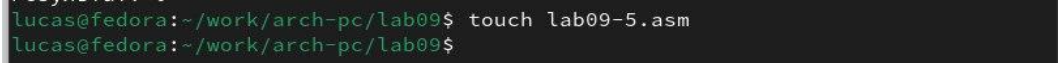


```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
lucas@fedora:~/work/arch-pc/lab09$ ./lab09-4
Функция: f(x) = 12x - 7
Результат: 0
lucas@fedora:~/work/arch-pc/lab09$
```

Рис. 2.30: Проверяем работу программы

### ###Задание 2

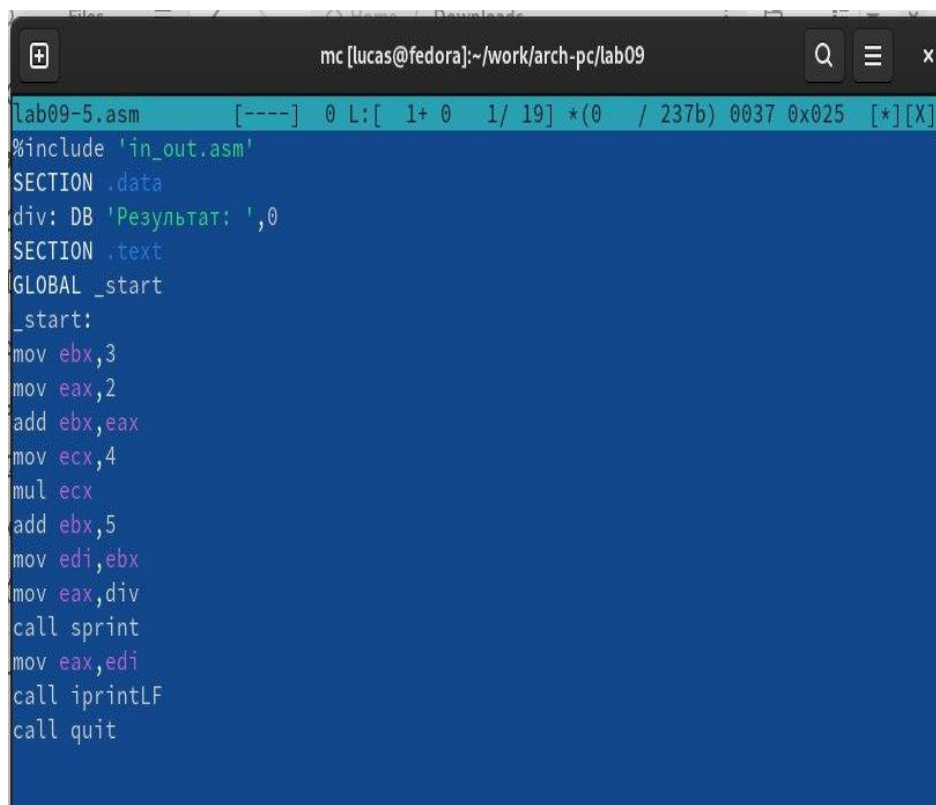
Создаем новый файл в дирректории (рис. 2.31).



```
lucas@fedora:~/work/arch-pc/lab09$ touch lab09-5.asm
lucas@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. 2.32).



```
lab09-5.asm  [----]  0 L:[ 1+ 0 1/ 19] *(0 / 237b) 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 iprintLF
call quit
```

Рис. 2.32: Изменяем файл

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



```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
lucas@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
lucas@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. 2.34).

The screenshot shows a debugger window with the title bar 'lucas@fedora:~/work/arch-pc/lab09'. It has two tabs, both with the same title. The top pane displays the state of CPU registers:

| Register | Value | Comment |
|----------|-------|---------|
| eax      | 0x8   | 8       |
| eflags   | 0x202 | [ IF ]  |
| cs       | 0x23  | 35      |
| ss       | 0x2b  | 43      |
| ds       | 0x2b  | 43      |
| es       | 0x2b  | 43      |
| fs       | 0x0   | 0       |
| gs       | 0x0   | 0       |

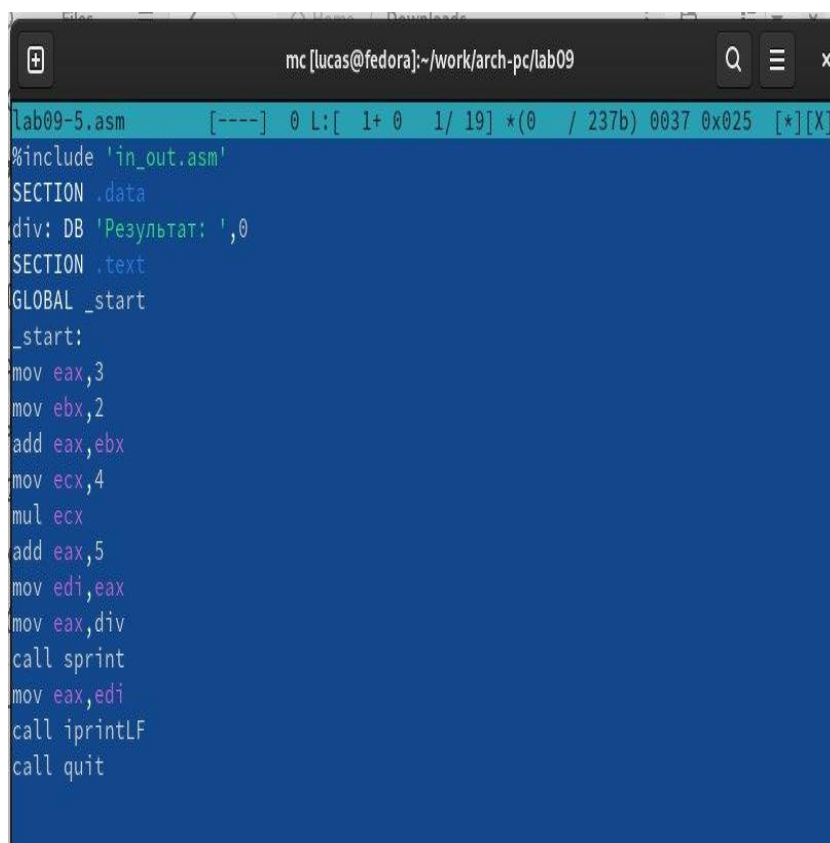
The bottom pane shows assembly code with line numbers 2 through 9. Line 9 is highlighted in blue. The code is as follows:

```
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:
B+ 9 mov eax, 4
```

At the bottom of the window, the status bar shows 'native process 25730 In: \_start', 'L14', 'PC: 0x8049016', and '\$5 = 0x8'.

Рис. 2.34: Ищем ошибку регистров в отладчике

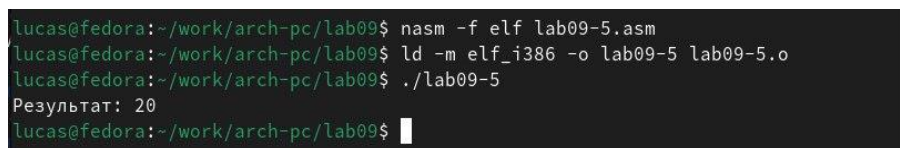
Изменяем программу для корректной работы (рис. 2.35).



```
lab09-5.asm  [----]  0 L: [ 1+ 0 1/ 19] *(0 / 237b) 0037 0x025 [*][X]
#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. 2.36).



```
lucas@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
lucas@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
lucas@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 20
lucas@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.