

# **Лабораторная работа №9**

**Отчет**

Устинова Виктория Вадимовна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Выводы</b>	<b>21</b>

# Список иллюстраций

3.1	Переходим в каталог и создаем там файл lab9-1.asm . . . . .	7
3.2	Заполняем данный файл . . . . .	8
3.3	Смотрим как работает файл . . . . .	8
3.4	Редактируем файл . . . . .	9
3.5	Смотрим как работает файл . . . . .	9
3.6	Используем команду touch . . . . .	9
3.7	Заполняем наш файл . . . . .	10
3.8	Выгружаем файл в отладчик . . . . .	10
3.9	Запускаем программу командой run в отладчике . . . . .	11
3.10	Запускаем файл . . . . .	11
3.11	Смотрим дисассимилированный код . . . . .	11
3.12	Переключаемся на синтаксис Intel . . . . .	12
3.13	Открываются три окна . . . . .	13
3.14	Используем команду info breakpoints, создаем новую точку . . . . .	13
3.15	Просматриваем информацию . . . . .	14
3.16	Смотрим значения регистров и переменной msg1 . . . . .	14
3.17	Смотрим значение переменной msg2 . . . . .	14
3.18	Меняем символ . . . . .	14
3.19	Меняем символ . . . . .	15
3.20	Смотрим значение регистра . . . . .	15
3.21	Изменяем регистр ebx командой set . . . . .	15
3.22	Копируем файл и запускаем его . . . . .	16
3.23	Устанавливаем точку останова . . . . .	16
3.24	Смотрим позиции стека по разным адресам . . . . .	16
3.25	Копируем файл . . . . .	17
3.26	Изменяем файл . . . . .	17
3.27	Запускаем файл . . . . .	18
3.28	Изменяем файл . . . . .	18
3.29	Запускаем файл . . . . .	19
3.30	Ищем ошибку . . . . .	19
3.31	Меняем файл . . . . .	19
3.32	Запускаем файл . . . . .	20

## **Список таблиц**

# 1 Цель работы

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

## **2 Задание**

Выполнить отчет по лабораторной работе №9, исправить ошибки программы и написать свою.

## 3 Выполнение лабораторной работы

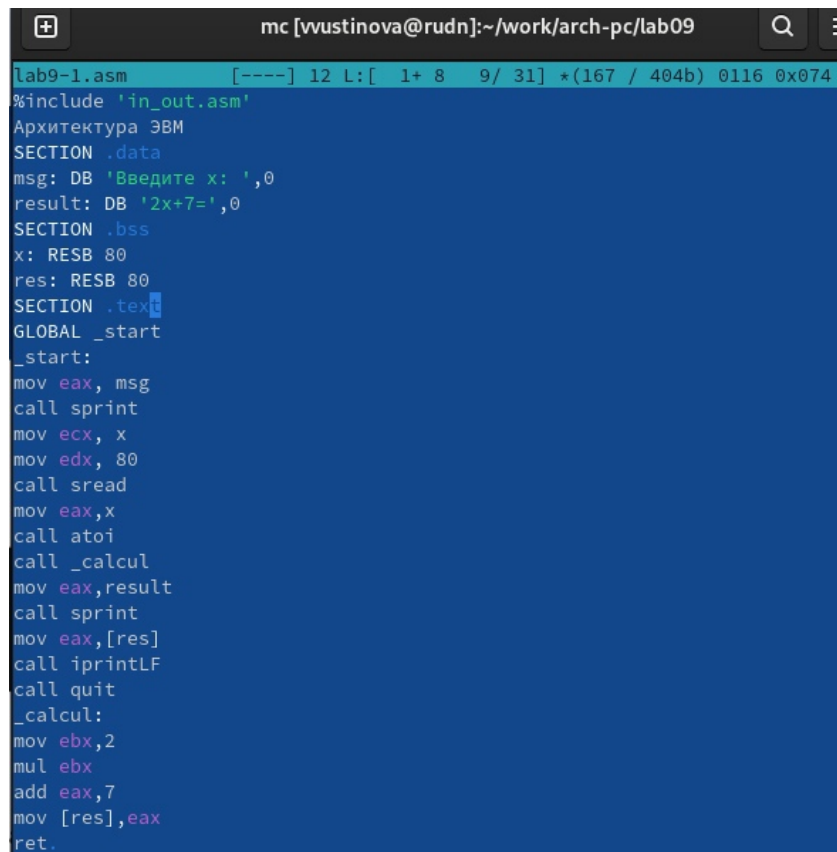
### Реализация подпрограмм в NASM

Создаем каталог для 9 лабораторной работы(рис. 3.1).

```
vvustanova@rudn:~$ mkdir ~/work/arch-pc/lab09  
vvustanova@rudn:~$ cd ~/work/arch-pc/lab09  
vvustanova@rudn:~/work/arch-pc/lab09$ touch lab9-1.asm  
vvustanova@rudn:~/work/arch-pc/lab09$
```

Рис. 3.1: Переходим в каталог и создаем там файл lab9-1.asm

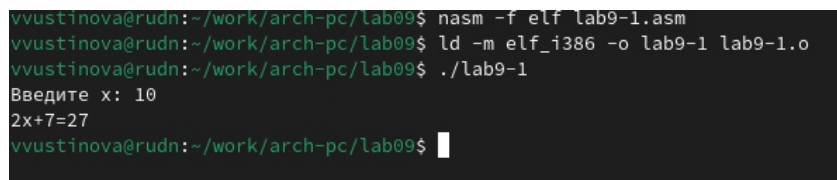
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1(рис. 3.2).



```
lab9-1.asm [----] 12 L: [ 1+ 8 9/ 31] *(167 / 404b) 0116 0x074
#include 'in_out.asm'
Архитектура ЭВМ
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. 3.2: Заполняем данный файл

Запускаем файл(рис. 3.3).



```
vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x+7=27
vvustinova@rudn:~/work/arch-pc/lab09$
```

Рис. 3.3: Смотрим как работает файл

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму(рис. 3.4).



```

lab9-1.asm      [----]  9 L:[  5+27  32/ 36]  *(416 / 439b) 0010 0x00A
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. 3.4: Редактируем файл

Запускаем файл(рис. 3.5).

```

vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2(3x-1)+7=65
vvustinova@rudn:~/work/arch-pc/lab09$

```

Рис. 3.5: Смотрим как работает файл

## Отладка программ с помощью GDB

Создаем файл Lab9-2.asm(рис. 3.6).

```

vvustinova@rudn:~/work/arch-pc/lab09$ touch lab9-2.asm
vvustinova@rudn:~/work/arch-pc/lab09$

```

Рис. 3.6: Используем команду touch

Открываем файл в Midnight Commander и заполняем его в соответствии с ли-

стингом 9.2(рис. 3.7).

```
lab9-2.asm      [----]  0 L:[ 1+ 0  1/ 22] *(0  / 294b) 0083 0x05
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. 3.7: Заполняем наш файл

Получаем исходный файл с использованием отладчика gdb(рис. 3.8).

```
vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
vvustinova@rudn:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-1.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...
(gdb)
```

Рис. 3.8: Выгружаем файл в отладчик

```
(gdb) run
Starting program: /home/vvustinova/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 separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6116) exited normally]
(gdb)
```

Рис. 3.9: Запускаем программу командой run в отладчике

Устанавливаем брейкпоинт на метку `_start` и запускаем программу(рис. 3.10).

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

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

Рис. 3.10: Запускаем файл

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`(рис. 3.11).

```
(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. 3.11: Смотрим дисассимилированный код

Переключаемся на отображение команд с Intel'овским синтаксисом(рис. 3.12).

```

(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. 3.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATТ и Intel: 1. В ATТ операнды указываются в формате сперва источник потом назначение, а в intel наоборот. 2. В ATТ перед регистрами ставится %, а перед значениями \$. В Intel префиксы отсутствуют. 3. В ATТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок. 4. В ATТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты /.

Переходим в режим псевдографики(рис. 3.13).

```

vustinova@rudn:~/work/arch-pc/lab09 — gdb lab9-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd060 0xffffd060
ebp      0x0      0x0

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 6188 (asm) In: _start      L9      PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 3.13: Открываются три окна

### Добавление точек останова

Проверяем была ли установлена точка останова и устанавливаем точку останова(рис. 3.14).

```

0x8049336 add     BYTE PTR [eax],al
0x8049338 add     BYTE PTR [eax],al
0x804933a add     BYTE PTR [eax],al
0x804933c add     BYTE PTR [eax],al
0x804933e add     BYTE PTR [eax],al
0x8049340 add     BYTE PTR [eax],al
0x8049342 add     BYTE PTR [eax],al
0x8049344 add     BYTE PTR [eax],al
0x8049346 add     BYTE PTR [eax],al

native process 6188 (asm) In: _start      L9      PC: 0x8049000
(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) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb)

```

Рис. 3.14: Используем команду info breakpoints, создаем новую точку

Посмотрим информацию о всех установленных точках останова(рис. 3.15).

```

native process 6188 (asm) In: _start L9 PC: 0x8049031
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 3.15: Просматриваем информацию

## Работа с данными программы в GDB

Посматриваем содержимое регистров с помощью команды `info registers`(рис. 3.16).

```

native process 6188 (asm) In: _start L9 PC: 0x8049000
esi 0x0 0
edi 0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x0 0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 3.16: Смотрим значения регистров и переменной `msg1`

Поменялись регистры `ebx`, `ecx`, `edx`, `eax`, `eip`.

```

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

```

Рис. 3.17: Смотрим значение переменной `msg2`

Изменим первый символ переменной `msg1`(рис. 3.18).

```

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

```

Рис. 3.18: Меняем символ

Изменим первый символ переменной msg2(рис. 3.19).

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

Рис. 3.19: Меняем символ

Смотрим значение регистра edx в разных форматах(рис. 3.20).

```
(gdb) p/t $edx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/x $edx
$3 = 0x0
```

Рис. 3.20: Смотрим значение регистра

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

Рис. 3.21: Изменяем регистр ebx командой set

Выводятся разные значения, потому что команда без кавычек присваивает регистру это значение.

## Обработка аргументов командной строки в GDB

Копируем файл lab8-2.asm в файл с именем lab09-3.asm и запускаем его в отладчике(рис. 3.22).

```
vvustinova@rudn:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
vvustinova@rudn:~/work/arch-pc/lab09$ gdb --args lab9-3 1 2 '3'
```

Рис. 3.22: Копируем файл и запускаем его

Установим точку останова перед первой инструкцией в программе и запустим ее(рис. 3.23).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/vvustinova/work/arch-pc/lab09/lab9-3 1 2 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 ecx
(gdb)
```

Рис. 3.23: Устанавливаем точку останова

```
(gdb) x/x $esp
0xffffd050: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd211: "/home/vvustinova/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd23c: "1"
(gdb) x/s *(void**)(esp + 12)
0xffffd23e: "2"
(gdb) x/s *(void**)(esp + 16)
0xffffd240: "3"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.24: Смотрим позиции стека по разным адресам

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита

## Задание для самостоятельной работы

1 задание

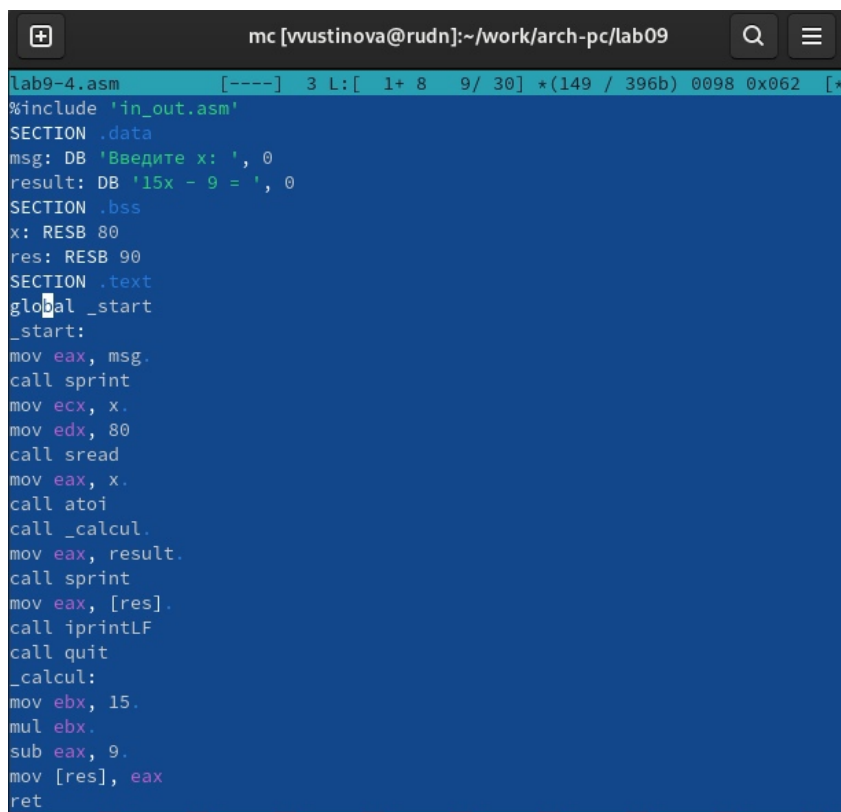


Переходим в нужный каталог и копируем оттуда файл lab8-4.asm в lab9-4.asm(рис. 3.25).

```
vvustinova@rudn:~$ cd ~/work/arch-pc/lab08
vvustinova@rudn:~/work/arch-pc/lab08$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/
arch-pc/lab09/lab9-4.asm
vvustinova@rudn:~/work/arch-pc/lab08$
```

Рис. 3.25: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму(рис. 3.26).



```
lab9-4.asm [----] 3 L: [ 1+ 8 9/ 30] *(149 / 396b) 0098 0x062 [+
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ', 0
result: DB '15x - 9 = ', 0
SECTION .bss
x: RESB 80
res: RESB 90
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
```

Рис. 3.26: Изменяем файл

```

vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-4
Введите x: 2
15x - 9 = 21
vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-4
Введите x: 10
15x - 9 = 141
vvustinova@rudn:~/work/arch-pc/lab09$

```

Рис. 3.27: Запускаем файл

## 2 задание

Создаем новый файл lab9-5.asm и открываем его в Midnight Commander и выполняем его в соответствии с листингом 9.3(рис. 3.28).

```

lab9-5.asm [----] 0 L:[ 1+ 0 1/ 20] *
%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. 3.28: Изменяем файл

```

vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-5
Результат: 10

```

Рис. 3.29: Запускаем файл

Создаем файл и запускаем его в отладчике, смотрим на изменение регистров(рис. 3.30).

The screenshot shows a debugger window with the following assembly code and register values:

Address	Disassembly	Comment
0x80490e7	<quit+12>	ret
0x80490e8	<_start>	mov ebx,0x3
0x80490ed	<_start+5>	mov eax,0x2
0x80490f2	<_start+10>	add ebx,eax
0x80490f4	<_start+12>	mov ecx,0x4
0x80490f9	<_start+17>	mul ecx
0x80490fb	<_start+19>	add ebx,0x5
0x80490fe	<_start+22>	mov edi,ebx
0x8049100	<_start+24>	mov eax,0x804a000

Below the assembly code, the debugger status bar shows: native process 8003 (asm) In: \_start L8 PC: 0x80490e8.

Рис. 3.30: Ищем ошибку

Изменяем программу(рис. 3.31).

```

lab9-5.asm [-M--] 6 L: [ 1+17 18/ 20] *(276 / 294b) 0112 0x070
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
mov ebx,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
mov eax,div
call sprintf
mov eax,edi
call iPrintLF
call quit

```

Рис. 3.31: Меняем файл

```
vvustinova@rudn:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
vvustinova@rudn:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
vvustinova@rudn:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
vvustinova@rudn:~/work/arch-pc/lab09$
```

Рис. 3.32: Запускаем файл

## 4 Выводы

Мы приобрели навыки написания программ с использованием подпрограмм. Ознакомились с методами отладки при помощи GDB и его основными возможностями.