

Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Югай Александр Витальевич

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Реализация подпрограмм в NASM	7
2.2	Отладка программ с помощью GDB	11
2.3	Задание для самостоятельной работы	22
2.3.1	Задание 1	22
2.3.2	Задание 2	25
3	Выводы	29

Список иллюстраций

2.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code>	7
2.2	Заполняем файл	8
2.3	Запускаем файл и проверяем его работу	9
2.4	Изменяем файл, добавляя еще одну подпрограмму	10
2.5	Запускаем файл и смотрим на его работу	11
2.6	Создаем файл	11
2.7	Заполняем файл	12
2.8	Загружаем исходный файл в отладчик	13
2.9	Запускаем программу командой <code>run</code>	13
2.10	Запускаем программу с брейкпоинтом	14
2.11	Смотрим дисассимилированный код программы	14
2.12	Переключаемся на синтаксис Intel	15
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы	16
2.14	Используем команду <code>info breakpoints</code> и создаем новую точку останова	17
2.15	Смотрим информацию	18
2.16	Отслеживаем регистры	18
2.17	Смотрим значение переменной	19
2.18	Смотрим значение переменной	19
2.19	Меняем символ	19
2.20	Меняем символ	19
2.21	Смотрим значение регистра	20
2.22	Изменяем регистр командой <code>set</code>	20
2.23	Прописываем команды <code>c</code> и <code>quit</code>	21
2.24	Копируем файл	21
2.25	Создаем и запускаем в отладчике файл	21
2.26	Устанавливаем точку останова	22
2.27	Изучаем полученные данные	22
2.28	Копируем файл	23
2.29	Изменяем файл	24
2.30	Проверяем работу программы	25
2.31	Создаем файл	25
2.32	Изменяем файл	26
2.33	Создаем и смотрим на работу программы	26
2.34	Ищем ошибку регистров в отладчике	27
2.35	Меняем файл	28

2.36 Создаем и запускаем файл	28
---	----

Список таблиц

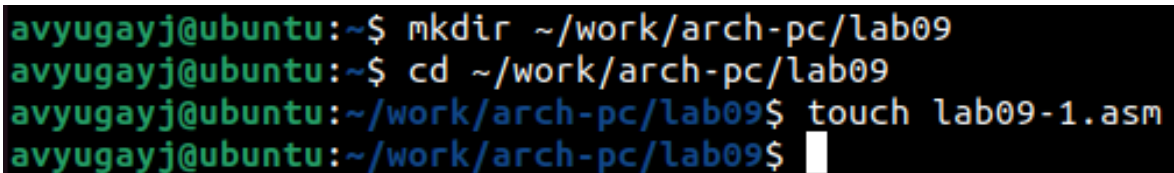
1 Цель работы

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



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

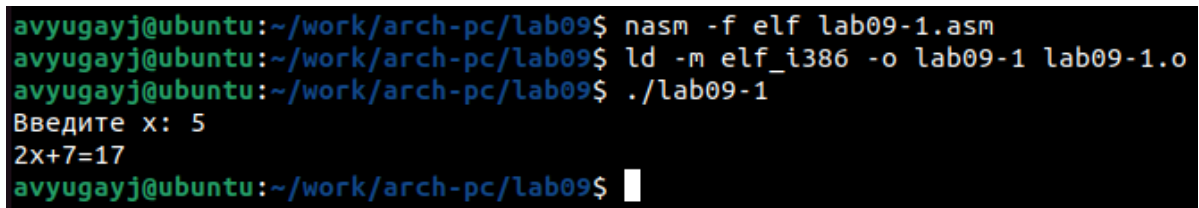
```

GNU nano 6.2
#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:
;-----
; Основная программа
;-----
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
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

```

Рис. 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его



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

```

%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:
;-----
; Основная программа
;-----
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
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_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

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

```
avyugayj@ubuntu:~/work/arch-pc/lab09$ touch lab09-2.asm
avyugayj@ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.6: Создаем файл

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

```
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

```

avyugayj@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
avyugayj@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
avyugayj@ubuntu:~/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...

```

Рис. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике

```

(gdb) run
Starting program: /home/avyugayj/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5332) 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/avyugayj/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

```

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 (/).

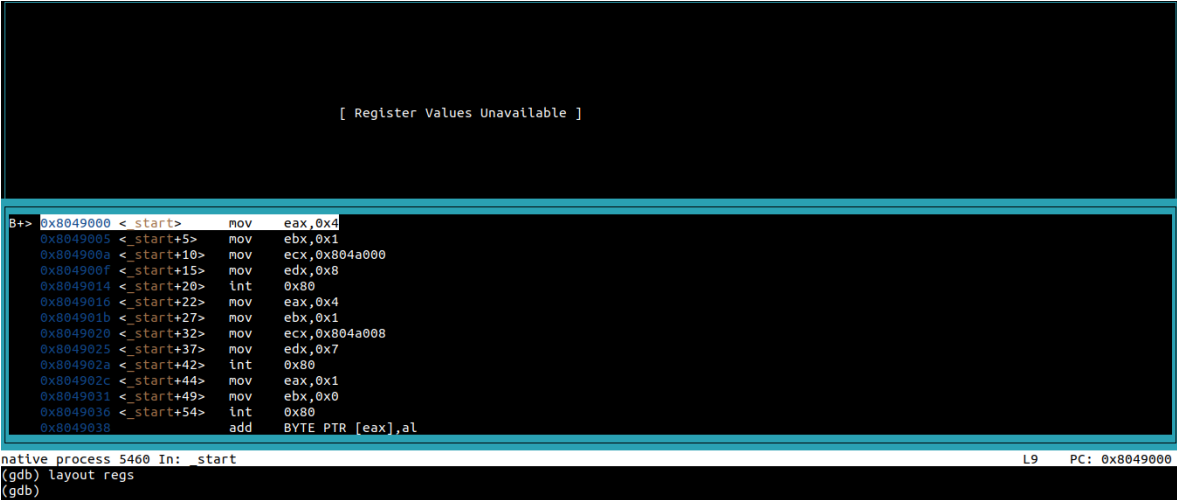
3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как 'b' (byte), 'w' (word), 'l' (long) и 'q' (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как 'b', 'w', 'd' и 'q'.

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 debugger window with a dark background. At the top, a message box says "[Register Values Unavailable]". Below it, a list of assembly instructions is displayed, each with its address and a comment. The instructions are as follows:

Address	Instruction
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
0x8049020	< start+32> mov ecx,0x804a008
0x8049025	< start+37> mov edx,0x7
0x804902a	< start+42> int 0x80
0x804902c	< start+44> mov eax,0x1
0x8049031	< start+49> mov ebx,0x0
0x8049036	< start+54> int 0x80
0x8049038	add BYTE PTR [eax],al

At the bottom of the window, the following text is visible:

```
native process 5460 In: start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

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

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


```
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
      0x8049020 <_start+32>   mov     ecx,0x804a008
      0x8049025 <_start+37>   mov     edx,0x7
      0x804902a <_start+42>   int     0x80
      0x804902c <_start+44>   mov     eax,0x1
b+  0x8049031 <_start+49>   mov     ebx,0x0
      0x8049036 <_start+54>   int     0x80
      0x8049038              add     BYTE PTR [eax],al

native process 5948 In: _start
(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

```
(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

```
Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd150 0xffffd150  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>  eflags   0x202      [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

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
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
b+ 0x8049031 <_start+49>     mov     ebx,0x0
0x8049036 <_start+54>     int     0x80
0x8049038      add     BYTE PTR [eax],al

native process 5948 In: start
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) 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) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

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\034"
(gdb) █
```

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1

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

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2

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

Рис. 2.20: Меняем символ

Смотрим значение регистра `edx` в разных форматах

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

Рис. 2.21: Смотрим значение регистра

Изменяем регистр `ebx`

```
(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`

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

Прописываем команды для завершения программы и выхода из GDB

```
(gdb) c
Continuing.
horld!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

    Inferior 1 [process 5948] will be killed.

Quit anyway? (y or n) █
```

Рис. 2.23: Прописываем команды c и quit

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

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

Рис. 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB

```
avyugayj@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
avyugayj@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
avyugayj@ubuntu:~/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

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/avyugayj/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd130:      0x00000004
(gdb) █

```

Рис. 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам

```

0xffffd130:      0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd30c:      "/home/avyugayj/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd336:      "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd338:      "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd33a:      "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).

2.3 Задание для самостоятельной работы

2.3.1 Задание 1

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

```
avyugayj@ubuntu:~$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
```

Рис. 2.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму

```

%include 'in_out.asm'
SECTION .data
msg: DB "Введите x: ",0
result: DB '10x-5= ',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, 10
mul ebx
sub eax, 5
mov [res], eax
ret

```

Рис. 2.29: Изменяем файл

Создаем исполняемый файл и запускаем его


```
avyugayj@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
avyugayj@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09
avyugayj@ubuntu:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 2
10x-5= 15
avyugayj@ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.30: Проверяем работу программы

2.3.2 Задание 2

Создаем новый файл в директории

```
avyugayj@ubuntu:~/work/arch-pc/lab09$ touch lab09-5.asm
```

Рис. 2.31: Создаем файл

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

```

%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
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его

```

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

```

Рис. 2.33: Создаем и смотрим на работу программы

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на

изменение реестров командой si

```
Терминал                               Пт, 8 декабря 01:29
avyugayj@ubuntu: ~/work/arch-pc/lab09

~Register group: general~
eax      0x2      2      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffd150 0xffffd150  ebp      0x0      0
esi      0x0      0      edi      0x0      0
eip      0x80490f9 0x80490f9 < start+17>  eflags   0x206     [ PF IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

0+ 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
0x8049105 < _start+29>  call    0x804900f <sprint>
0x804910a < _start+34>  mov     eax,edi
0x804910c < _start+36>  call    0x8049086 <printfLF>
0x8049111 < _start+41>  call    0x80490db <quit>
0x8049116             add     BYTE PTR [eax],al
0x8049118             add     BYTE PTR [eax],al

native process 21850 In: start
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы

```

#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
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.35: Меняем файл

Создаем исполняемый файл и запускаем его

```

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

```

Рис. 2.36: Создаем и запускаем файл

3 Выводы

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