

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Спелов Андрей Николаевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	9
3	Выводы	23

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

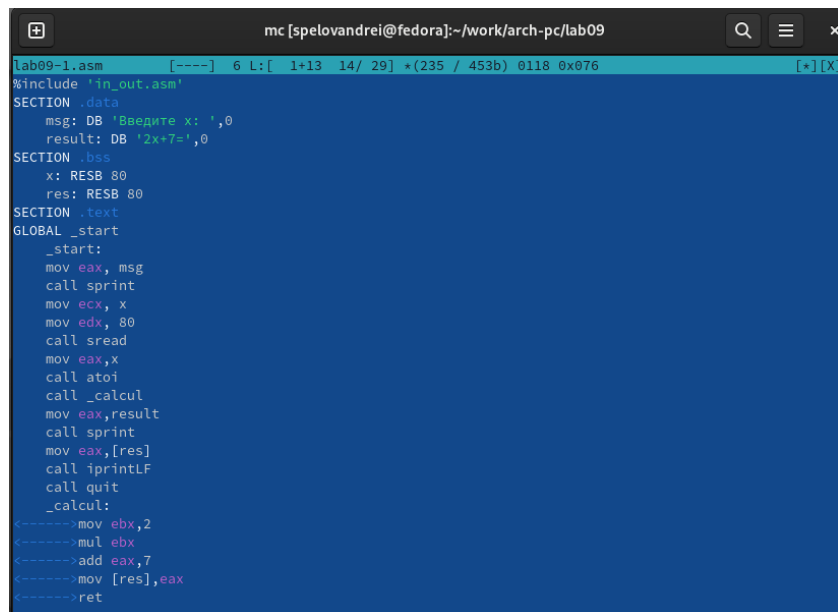
2.1 Реализация подпрограмм в NASM

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

```
[spelovandrei@fedora ~]$ mkdir ~/work/arch-pc/lab09  
[spelovandrei@fedora ~]$ cd ~/work/arch-pc/lab09  
[spelovandrei@fedora lab09]$ touch lab09-1.asm  
[spelovandrei@fedora lab09]$
```

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

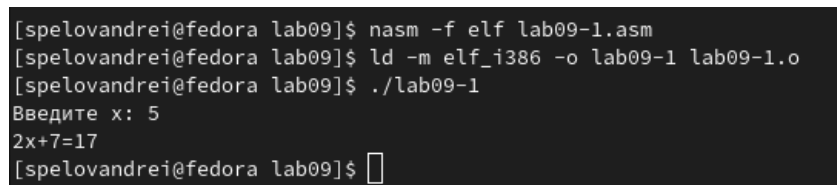
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. 2.2).



```
lab09-1.asm [----] 6 L: [ 1+13 14/ 29] *(235 / 453b) 0118 0x076 [*][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).



```
[spelovandrei@fedora lab09]$ nasm -f elf lab09-1.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[spelovandrei@fedora lab09]$ ./lab09-1
Введите x: 5
2x+7=17
[spelovandrei@fedora 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      [----] 23 L:[ 1+24 25/ 35] *(424 / 540b) 0
#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 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).

```

[spelovandrei@fedora lab09]$ nasm -f elf lab09-1.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[spelovandrei@fedora lab09]$ ./lab09-1
Введите x: 5
2(3x-1)+7=35

```

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).

```
[spelovandrei@fedora lab09]$ touch lab09-2.asm  
[spelovandrei@fedora 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  [-M--] 12 L:[ 1+20 21/ 21] *(364 / 364b)  
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).

```
[spelovandrei@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[spelovandrei@fedora lab09]$ gdb lab09-2
GNU gdb (Fedora Linux) 13.2-10.fc39
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. 2.9).

```
[Inferior 1 (process 5604) exited normally]
(gdb) run
Starting program: /home/spelovandrei/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5604) exited normally]
(gdb)
```

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/spelovandrei/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. 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 указывает-

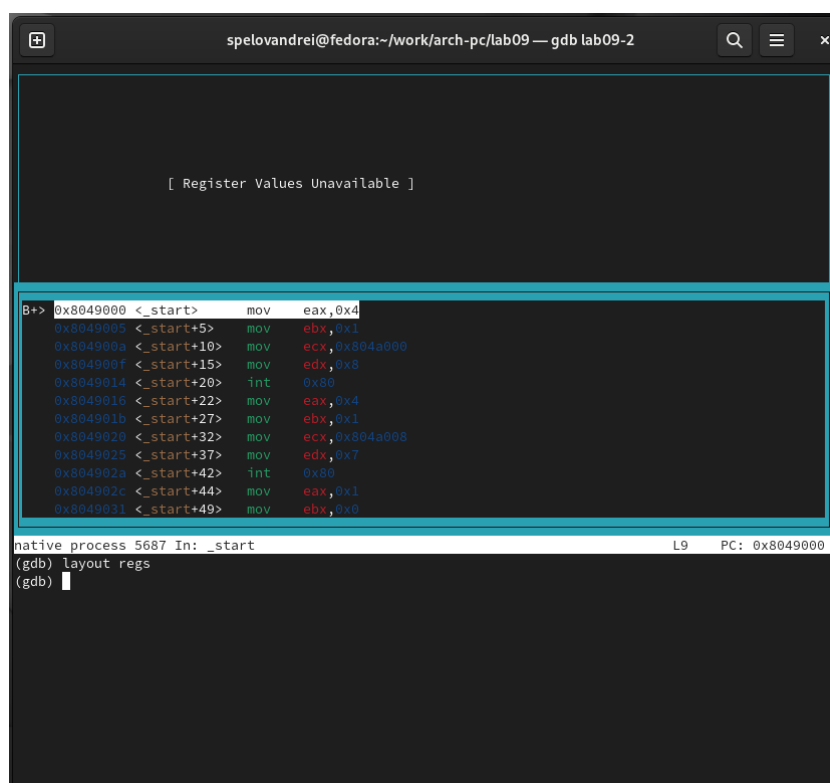
ся перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4. Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “*.*”. В Intel 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 GDB terminal window with the title bar "spelovandrei@fedora:~/work/arch-pc/lab09 — gdb lab09-2". The main display area shows the message "[Register Values Unavailable]". Below this, a list of assembly instructions is displayed, each with its address, disassembly, and comment. The instructions are as follows:

Address	Disassembly	Comment
0x8049000	mov	eax,0x4
0x8049005	mov	ebx,0x1
0x804900a	mov	ecx,0x804a000
0x804900f	mov	edx,0x8
0x8049014	int	0x80
0x8049016	mov	eax,0x4
0x804901b	mov	ebx,0x1
0x8049020	mov	ecx,0x804a008
0x8049025	mov	edx,0x7
0x804902a	int	0x80
0x804902c	mov	eax,0x1
0x8049031	mov	ebx,0x0

At the bottom of the window, the status bar shows "native process 5687 In: _start" and "L9 PC: 0x8049000". Below the status bar, the command "(gdb) layout regs" is entered, and the prompt "(gdb) " is visible.

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

Проверяем была ли установлена точка останова и устанавливаем точку остано-

ва предпоследней инструкции (рис. 2.14).

```
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

native process 6117 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).

```
(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).

```
spelovandrei@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0c0 0xffffd0c0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 6117 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
       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
(gdb) 
```

Рис. 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip. Смотрим значение переменной msg1 по имени (рис. 2.17).

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

Рис. 2.17: Смотрим значение переменной

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

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

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. 2.19).

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

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

Выводятся 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)
```

Рис. 2.23: Прописываем команды c и quit

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

```
[spelovandrei@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[spelovandrei@fedora lab09]$
```

Рис. 2.24: Копируем файл

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

```
[spelovandrei@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[spelovandrei@fedora 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
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/spelovandrei/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xffffd0b0: 0x00000004
(gdb)

```

Рис. 2.26: Устанавливаем точку останова

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

```

(gdb) x/s *(void**)(esp + 4)
0xffffd272: "/home/spelovandrei/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2a8: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd2a2: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd2a4: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>

```

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).

```

[spelovandrei@fedora lab08]$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
[spelovandrei@fedora lab08]$

```

Рис. 2.28: Копируем файл

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

```

lab09-4.asm      [----] 11 L: [ 1+28 2
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '3(10+x)=',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:
<----->add eax,10
<----->mov ebx,3
<----->mul ebx
<----->mov [res],eax
<----->ret

```

Рис. 2.29: Изменяем файл

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

```
[spelovandrei@fedora lab09]$ nasm -f elf lab09-4.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-4 lab09-4.o
[spelovandrei@fedora lab09]$ ./lab09-4
Введите x: 5
3(10+x)=45
[spelovandrei@fedora lab09]$
```

Рис. 2.30: Проверяем работу программы

###Задание 2

Создаем новый файл в директории (рис. 2.31).

```
[spelovandrei@fedora lab09]$ touch lab09-5.asm
[spelovandrei@fedora 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      [----] 13 L: [ 1+17 18
%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,dib
    call sprint
    mov eax,edi
    call iprintLF
    call quit

```

Рис. 2.32: Изменяем файл

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

```

[spelovandrei@fedora lab09]$ nasm -f elf lab09-5.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[spelovandrei@fedora lab09]$ ./lab09-5
Результат: 10
[spelovandrei@fedora 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).

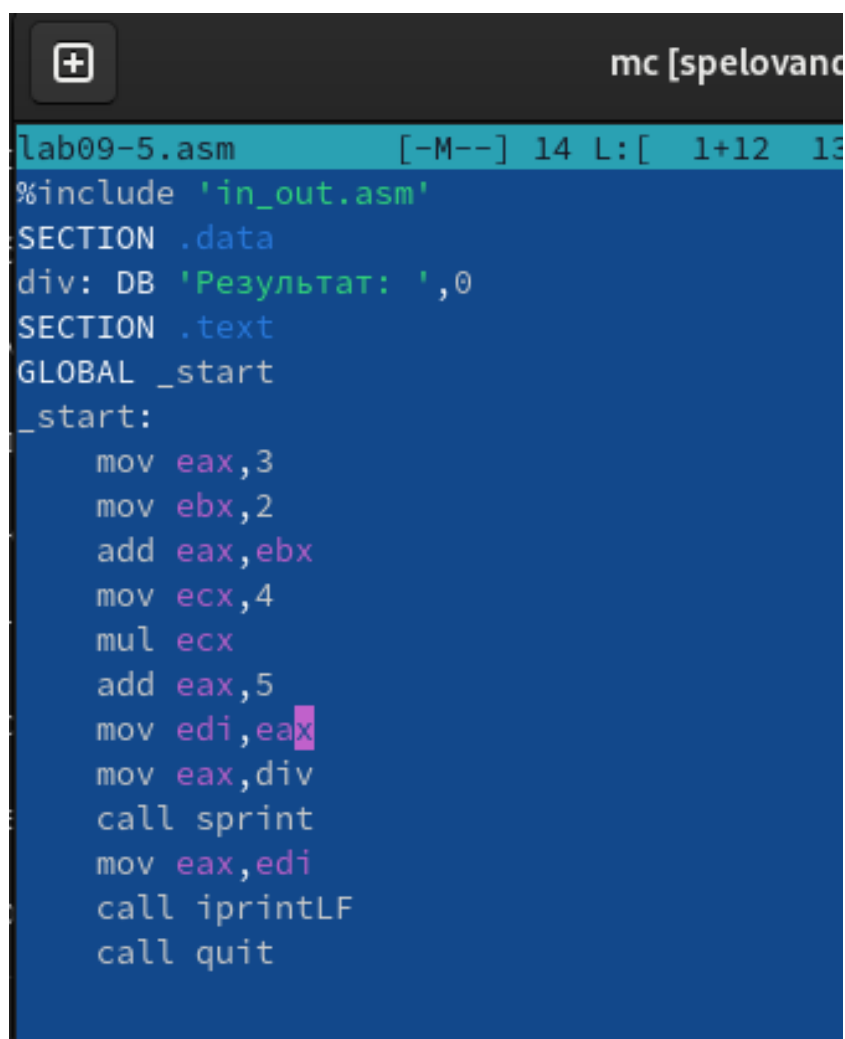
```
100% + Снимок экрана от 2023-11-20 20:57-25.png
spelovandrei@fedora:~/work/arch-pc/lab09 — gdb lab09-5
Register group: general
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffd0b0 0xffffd0b0
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35
ss 0x2b 43

B+ 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
0x80490fd <_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 <iprintfLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al

native process 8879 In: _start L11 PC: 0x80490f9
esp 0xffffd0b0 0xffffd0b0
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490f4 0x80490f4 <_start+12>
eflags 0x206 [ PF IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--fs 0x0 0
```

Рис. 2.34: Ищем ошибку регистров в отладчике

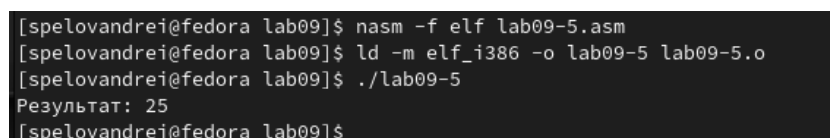
Изменяем программу для корректной работы (рис. 2.35).



```
lab09-5.asm [-M--] 14 L: [ 1+12 13
#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).



```
[spelovandrei@fedora lab09]$ nasm -f elf lab09-5.asm
[spelovandrei@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[spelovandrei@fedora lab09]$ ./lab09-5
Результат: 25
[spelovandrei@fedora lab09]$
```

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

3 Выводы

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