

Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Чипурной Михаил Евгеньевич

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Реализация подпрограмм в NASM	7
2.2	Отладка программ с помощью GDB	9
2.3	Задание для самостоятельной работы	20
3	Выводы	25

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

2.35	Меняем файл	24
2.36	Создаем и запускаем файл(работает корректно)	24

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 (рис. Рисунок 2.1).

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

Рисунок 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. Рисунок 2.2).

```

lab09-1.asm      [-M--]  7 L:[ 1+33 34/ 34] *(455 / 455b) <EOF>
#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

```

Рисунок 2.2: Заполняем файл

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

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 la
b09-1.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. Рисунок 2.4).


```

lab09-1.asm [-M--] 22 L:[ 1+ 4 5/ 40] *(90 / 544b) 0039 0x027
#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

```

Рисунок 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.5).

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 la
b09-1.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 2.5: Запускаем файл и смотрим на его работу

2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. Рисунок 2.6).

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

Рисунок 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. Рисунок 2.7).

```
lab09-2.asm [-M--] 4 L:[ 1+20 21/ 22] *(357 / 366b) 0105 0x069
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
```

Рисунок 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. Рисунок 2.8).

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab09-2...
(gdb) █

```

Рисунок 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. Рисунок 2.9).

```

(gdb) run
Starting program: /home/mechipurnoyj/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 9651) exited normally]
(gdb) █

```

Рисунок 2.9: Запускаем программу командой run

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

```

(gdb) break _start
Breakpoint 1 at 0x8048080: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/mechipurnoyj/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) █

```

Рисунок 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. Рисунок 2.11).

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     $0x4,%eax
    0x08048085 <+5>:      mov     $0x1,%ebx
    0x0804808a <+10>:     mov     $0x8049000,%ecx
    0x0804808f <+15>:     mov     $0x8,%edx
    0x08048094 <+20>:     int     $0x80
    0x08048096 <+22>:     mov     $0x4,%eax
    0x0804809b <+27>:     mov     $0x1,%ebx
    0x080480a0 <+32>:     mov     $0x8049008,%ecx
    0x080480a5 <+37>:     mov     $0x7,%edx
    0x080480aa <+42>:     int     $0x80
    0x080480ac <+44>:     mov     $0x1,%eax
    0x080480b1 <+49>:     mov     $0x0,%ebx
    0x080480b6 <+54>:     int     $0x80
End of assembler dump.
(gdb) █

```

Рисунок 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. Рисунок 2.12).

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     eax,0x4
    0x08048085 <+5>:      mov     ebx,0x1
    0x0804808a <+10>:     mov     ecx,0x8049000
    0x0804808f <+15>:     mov     edx,0x8
    0x08048094 <+20>:     int     0x80
    0x08048096 <+22>:     mov     eax,0x4
    0x0804809b <+27>:     mov     ebx,0x1
    0x080480a0 <+32>:     mov     ecx,0x8049008
    0x080480a5 <+37>:     mov     edx,0x7
    0x080480aa <+42>:     int     0x80
    0x080480ac <+44>:     mov     eax,0x1
    0x080480b1 <+49>:     mov     ebx,0x0
    0x080480b6 <+54>:     int     0x80
End of assembler dump.
(gdb) █

```

Рисунок 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATТ 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 (рис. Рисунок 2.13).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf60 0xffffcf60
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8048080 0x8048080 <_start>
eflags   0x202    [ IF ]

lab09-2.asm
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:
B+> 9   mov eax, 4
10  mov ebx, 1

native process 9713 (src) In: _start L9 PC: 0x8048080
(gdb) layout regs
(gdb) █
```

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

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. Рисунок 2.14).

```
Registers
[ Register Values Unavailable ]

0x804809b <_start+27>  mov     ebx,0x1
0x80480a0 <_start+32>  mov     ecx,0x8049008
0x80480a5 <_start+37>  mov     edx,0x7
0x80480aa <_start+42>  int     0x80
0x80480ac <_start+44>  mov     eax,0x1
b+ 0x80480b1 <_start+49>  mov     ebx,0x0
0x80480b6 <_start+54>  int     0x80

exec No process (asm) In: L?? PC: ??
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08048080 lab09-2.asm:9
(gdb) break *0x80480b1
Breakpoint 2 at 0x80480b1: file lab09-2.asm, line 20.
(gdb) █
```

Рисунок 2.14: Используем команду info breakpoints и создаем новую точку останова

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

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08048080 lab09-2.asm:9
2        breakpoint keep y  0x080480b1 lab09-2.asm:20
(gdb) █
```

Рисунок 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. Рисунок 2.16).

```

Register group: general
eax      0x8      8
ecx      0x8049000 134516736
edx      0x8      8
ebx      0x1      1
esp      0xffffcf60 0xffffcf60
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8048096 0x8048096 <_start+22>
eflags   0x202    [ IF ]

B+ 0x8048080 <_start>      mov     eax,0x4
0x8048085 <_start+5>      mov     ebx,0x1
0x804808a <_start+10>     mov     ecx,0x8049000
0x804808f <_start+15>     mov     edx,0x8
0x8048094 <_start+20>     int     0x80
>0x8048096 <_start+22>     mov     eax,0x4
0x804809b <_start+27>     mov     ebx,0x1
0x80480a0 <_start+32>     mov     ecx,0x8049008
0x80480a5 <_start+37>     mov     edx,0x7
0x80480aa <_start+42>     int     0x80

native process 10677 (asm) In: _start L14 PC: 0x8048096
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/mechipurnoyj/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) siHello,
(gdb) █

```

Рисунок 2.16: Отслеживаем регистры

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

```

(gdb) x/lsb &msg1
0x8049000 <msg1>:      "Hello, "
(gdb) █

```

Рисунок 2.17: Смотрим значение переменной

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


```
(gdb) x/lsb 0x8049008
0x8049008 <msg2>:      "world!\n\034"
(gdb) █
```

Рисунок 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. Рисунок 2.19).

```
(gdb) set {char}&msg1='h'
(gdb) x/lsb &msg1
0x8049000 <msg1>:      "hello, "
(gdb) █
```

Рисунок 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. Рисунок 2.20).

```
(gdb) set {char}&msg2='L'
(gdb) x/lsb &msg2
0x8049008 <msg2>:      "Lorld!\n\034"
(gdb) █
```

Рисунок 2.20: Меняем символ

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

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

Рисунок 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. Рисунок 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb) █
```

Рисунок 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. Рисунок 2.23).

```
(gdb) cLorld!

Continuing.

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

Рисунок 2.23: Прописываем команды с и quit

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

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

Рисунок 2.24: Копируем файл

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

```
mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.ls
t lab09-3.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 la
b09-3.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab09-3...
(gdb) █
```

Рисунок 2.25: Создаем и запускаем в отладчике файл

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

```

(gdb) b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/mechipurnoyj/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:7
7      pop ecx
(gdb) x/x $esp
0xffffcf70: 0x00000004
(gdb)

```

Рисунок 2.26: Устанавливаем точку останова

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

```

(gdb) x/s *(void**)(esp + 4)
0xffffd14c: "/home/mechipurnoyj/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd17a: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd17c: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd17e: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рисунок 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 lab09-3.asm (рис. Рисунок 2.28).

```

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

```

Рисунок 2.28: Копируем файл

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

```

lab09-4.asm [-M--] 16 L:[ 1+22 23/ 38] *(265 / 446b) 0010 0x00A
#include "in_out.asm"

SECTION .data
msg db "Результат: ",0

SECTION .bss
prm: resb 80

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi
    call _calcul
    add esi, eax
    loop next

_end:
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintLF
    call quit

_calcul:
    sub eax, 1
    mov ebx, 10
    mul ebx
    ret

```

Рисунок 2.29: Изменяем файл

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

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 la
b09-4.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ./lab09-4 10 9 3
Результат: 190
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 2.30: Проверяем работу программы

2.3.2 Задание 2

Создаем новый файл в дирректории (рис. Рисунок 2.31).

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ touch lab09-5.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 2.31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. Рисунок 2.32).

```

lab09-5.asm  [-M--] 13 L:[ 1+20 21/ 21] *(295 / 295b) <EOF>
#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

```

Рисунок 2.32: Изменяем файл

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

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 la
b09-5.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 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. Рисунок 2.34).

```

mechipurnoyj@fedora:~/work/arch-pc/lab09 — gdb lab09-5
mechipurnoyj@fedora:~/work/arch-pc/lab09 — /usr/
mechipurnoyj@fedora:~/work/arch-pc/lab09 — flamesho

Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffcf90 0xffffcf90
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8048179 0x8048179 <_start+17>
eflags   0x10206   [ PF IF RF ]

B+ 0x8048168 <_start>      mov     ebx,0x3
0x804816d <_start+5>      mov     eax,0x2
0x8048172 <_start+10>     add     ebx,eax
0x8048174 <_start+12>     mov     ecx,0x4
>0x8048179 <_start+17>    mul     ecx
0x804817b <_start+19>     add     ebx,0x5
0x804817e <_start+22>     mov     edi,ebx
0x8048180 <_start+24>     mov     eax,0x8049000
0x8048185 <_start+29>     call    0x804808f <sprint>
0x804818a <_start+34>     mov     eax,edi

native process 14669 (asm) In: _start      L14  PC: 0x8048179
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffcf90 0xffffcf90
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8048179 0x8048179 <_start+17>
eflags   0x10206   [ PF IF RF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рисунок 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. Рисунок 2.35).

```

lab09-5.asm      [-M--] 16 L:[ 1+18 19/ 21] *(263 / 295b) 0010 0x00
%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

```

Рисунок 2.35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.36).

```

mechipurnoyj@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 la
b09-5.o
mechipurnoyj@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
mechipurnoyj@fedora:~/work/arch-pc/lab09$

```

Рисунок 2.36: Создаем и запускаем файл(работает корректно)

3 Выводы

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