

Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Власов Артем Сергеевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	8
3	Выводы	21

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	8
2.7	Заполняем файл	9
2.8	Загружаем исходный файл в отладчик	9
2.9	Запускаем программу командой <code>run</code>	10
2.10	Запускаем программу с брейкпоинтом	10
2.11	Смотрим дисассимилированный код программы	10
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	14
2.19	Меняем символ	14
2.20	Меняем символ	15
2.21	Смотрим значение регистра	15
2.22	Изменяем регистр командой <code>set</code>	15
2.23	Прописываем команды <code>c</code> и <code>quit</code>	15
2.24	Копируем файл	16
2.25	Создаем и запускаем в отладчике файл	16
2.26	Устанавливаем точку останова	16
2.27	Изучаем полученные данные	16
2.28	Копируем файл	17
2.29	Изменяем файл	17
2.30	Проверяем работу программы	18
2.31	Создаем файл	18
2.32	Изменяем файл	18
2.33	Создаем и смотрим на работу программы(работает неправильно)	18
2.34	Ищем ошибку регистров в отладчике	19
2.35	Меняем файл	20

2.36 Создаем и запускаем файл(работает корректно)	20
---	----

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. fig. 2.1).

```
vlasovas@vbox:~$ mkdir ~/work/arch-pc/lab09
vlasovas@vbox:~$ cd ~/work/arch-pc/lab09
vlasovas@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
vlasovas@vbox:~/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. fig. 2.2).

```

lab09-1.asm  [----] 12 L: [ 1+11 12/ 30] *(191 / 395b) 0010 0x00A [*][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. fig. 2.3).

```

vlasovas@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vlasovas@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vlasovas@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13
vlasovas@vbox:~/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. fig. 2.4).

```

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. fig. 2.5).

```

vlasovas@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vlasovas@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vlasovas@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2(3x-1)+7=23
vlasovas@vbox:~/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. fig. 2.6).

```

vlasovas@vbox:~/work/arch-pc/lab09$ touch lab09-2.asm
vlasovas@vbox:~/work/arch-pc/lab09$

```

Рис. 2.6: Создаем файл

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


```

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. fig. 2.8).

```

vlasovas@vbox:~/work/arch-pc/lab09$ touch lab09-2.asm
vlasovas@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
vlasovas@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
vlasovas@vbox:~/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)

```

Рис. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. fig. 2.9).

```
(gdb) run
Starting program: /home/vlasovas/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
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 3871) exited normally]
(gdb)
```

Рис. 2.9: Запускаем программу командой run

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

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/vlasovas/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. fig. 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. fig. 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. В Intel синтаксисе порядок обычно прямой.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители 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 “\$”. В Intel синтаксисе операнды с позитивными значениями могут быть указаны без него.

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе 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. fig. 2.13).

```
[ Register Values Unavailable ]

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

native process 3905 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 (рис. fig. 2.14).

```

[ Register Values Unavailable ]

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

native process 3905 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. fig. 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).

```

--Register group: general--
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      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

native process 3905 In: _start L14 PC: 0x8049016
(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, ebp. Смотрим значение переменной msg1 по имени (рис. fig. 2.17).

```

(gdb) x/lsb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 2.17: Смотрим значение переменной

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

```

(gdb) x/lsb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 2.18: Смотрим значение переменной

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

```

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

```

Рис. 2.19: Меняем символ

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

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

Рис. 2.20: Меняем символ

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

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

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. fig. 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 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. fig. 2.23).

```
(gdb) c
Continuing.
Lorld!

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. fig. 2.24).

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

Рис. 2.24: Копируем файл

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

```
vlasovas@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
vlasovas@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
vlasovas@vbox:~/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. fig. 2.26).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/vlasovas/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
0xffffd090: 0x00000004
```

Рис. 2.26: Устанавливаем точку останова

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

```
(gdb) x/s *(void**)(esp + 4)
0xffffd25d: "/home/vlasovas/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd287: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd289: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd28b: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd28d: "SHELL=/bin/bash"
(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

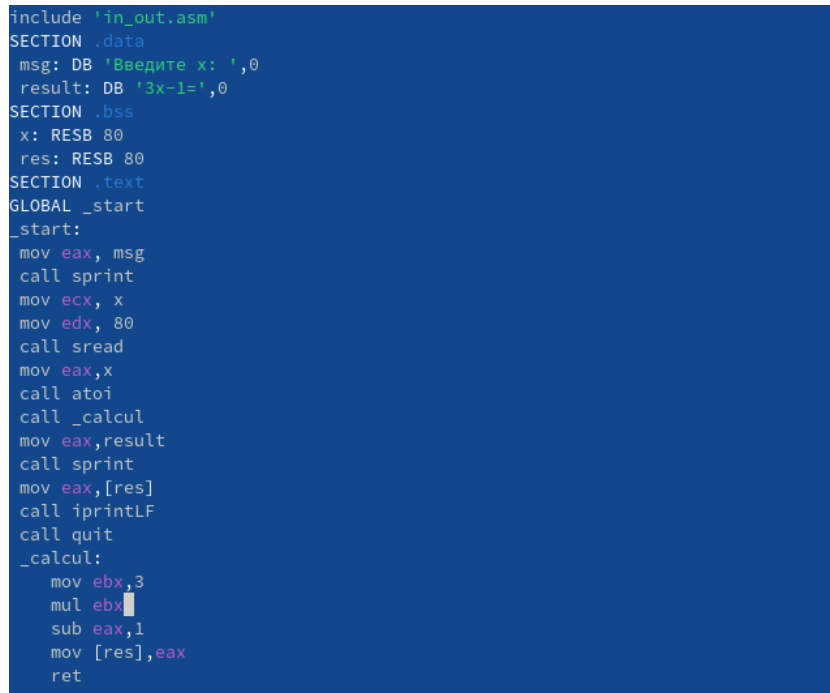
Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. fig. 2.28).



```
vlasovas@vbox:~/work/arch-pc/lab09$ 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 (рис. fig. 2.29).



```
include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '3x-1=',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, 3
mul ebx
sub eax, 1
mov [res], eax
ret
```

Рис. 2.29: Изменяем файл

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

```

vlasovas@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
vlasovas@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
vlasovas@vbox:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 5
3x-1=14
vlasovas@vbox:~/work/arch-pc/lab09$

```

Рис. 2.30: Проверяем работу программы

###Задание 2

Создаем новый файл в директории (рис. fig. 2.31).

```

vlasovas@vbox:~/work/arch-pc/lab09$ touch lab09-5.asm
vlasovas@vbox:~/work/arch-pc/lab09$

```

Рис. 2.31: Создаем файл

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

```

#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. fig. 2.33).

```

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

```
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x10206 [ PF IF RF ]
cs       0x23     35

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
    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 <iprintLF>
    0x8049111 <_start+41> call    0x80490db <quit>

native process 5172 In: _start L11 PC: 0x80490f9
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 7.
(gdb) run
Starting program: /home/vlasovas/work/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:7
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.34: Ищем ошибку регистров в отладчике

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

```
lab09-5.asm      [-M--] 12 L:[ 1+13 14/ 18] *(198 / 250b) 0010 0x00A  [*][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
```

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC10Выход

Рис. 2.35: Меняем файл

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

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