

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Газизянов Владислав Альбертович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	10
2.3	Добавление точек останова	15
2.4	Работа с данными программы в GDB	16
2.5	Обработка аргументов командной строки в GDB	18
2.6	Задание для самостоятельной работы	21
2.6.1	Задание 1	21
2.6.2	Задание 2	23
3	Выводы	27

Список иллюстраций

2.1	Создаём каталог	6
2.2	Заполняем файл	7
2.3	Проверяем работу	7
2.4	Изменим текст программы	9
2.5	Результат	10
2.6	Создание файла	10
2.7	Заполнение	11
2.8	ключ -g	12
2.9	Загрузка в gdb	12
2.10	Проверка работы	12
2.11	брейкпоинт	12
2.12	Запуск	13
2.13	Просмотрим код	13
2.14	переход на Intel	13
2.15	проверка работы	14
2.16	псевдографика	15
2.17	info breakpoints	15
2.18	точка останова	15
2.19	информация о точках	16
2.20	инструкции	16
2.21	содержание регистров	17
2.22	msg1	17
2.23	msg2	17
2.24	меняем символ	17
2.25	меняем символ	17
2.26	значения	18
2.27	значения	18
2.28	завершение	18
2.29	создание файла	19
2.30	заполнение файла	19
2.31	исполняемый файл	19
2.32	загрузка в отладчик	20
2.33	точка останова	20
2.34	позиции стека	20
2.35	позиции стека	21
2.36	Копируем	22
2.37	Редатируем	23

2.38	Запуск	23
2.39	Новый файл	24
2.40	Запуск	24
2.41	Смотрим изменения	25
2.42	корректируем	25
2.43	проверка работы	26

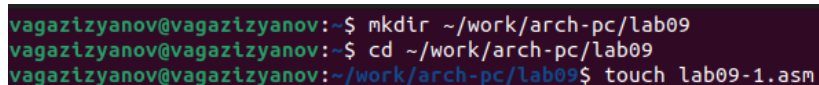
1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm:

A screenshot of a terminal window with a dark background and light-colored text. It shows three lines of commands being executed in a shell. The first line creates a directory named 'lab09' in the path '~/.work/arch-pc/'. The second line changes the current directory to '~/.work/arch-pc/lab09'. The third line creates an empty file named 'lab09-1.asm' in the current directory.

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

Рис. 2.1: Создаём каталог

В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
0 _start:
1 ;-----
2 ; Основная программа
3 ;-----
4 mov eax, msg
5 call sprint
6 mov ecx, x
7 mov edx, 80
8 call sread
9 mov eax, x
0 call atoi
1 call _calcul ; Вызов подпрограммы _calcul
2 mov eax, result
3 call sprint
4 mov eax, [res]
5 call iprintLF
6 call quit
7 ;-----
8 ; Подпрограмма вычисления
9 ; выражения "2x+7"
0 _calcul:
1 mov ebx, 2
2 mul ebx
3 add eax, 7
4 mov [res], eax
5 ret ; выход из подпрограммы
6 mov eax, msg ; вызов подпрограммы печати сообщения
7 call sprint ; 'Введите x: '
8 mov ecx, x
9 mov edx, 80

```

Рис. 2.2: Заполняем файл

```

vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2x+7=15

```

Рис. 2.3: Проверяем работу

Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x +$

$$7, g(x) = 3x - 1.$$

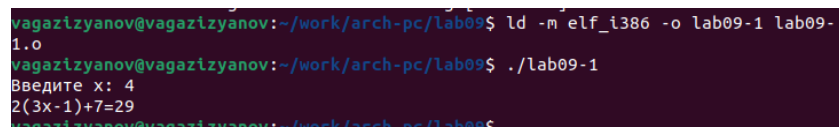

```

1 %include 'in_out.asm
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
0 _start:
1 mov eax, msg
2 call sprint
3 mov ecx, x
4 mov edx, 80
5 call sread
6 mov eax,x
7 call atoi
8 call _calcul
9 mov eax,result
0 call sprint
1 mov eax,[res]
2 call iprintLF
3 call quit
4 _calcul:
5 call _subcalcul
6 mov ebx,2
7 mul ebx
8 add eax,7
9 mov [res],eax
0 ret ;
1 _subcalcul:
2 mov ebx,3
3 mul ebx
4 sub eax,1
5 ret

```

Рис. 2.4: Изменим текст программы

Выводим результат.



```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2(3x-1)+7=29
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$
```

Рис. 2.5: Результат

2.2 Отладка программ с помощью GDB

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2.



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

Рис. 2.6: Создание файла

```
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:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 2.7: Заполнение

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

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

Рис. 2.8: ключ -g

Загрузите исполняемый файл в отладчик gdb

```
vagazizyanov@vagazizyanov:~/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...
(gdb) =
```

Рис. 2.9: Загрузка в gdb

Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run

```
(gdb) run
Starting program: /home/vagazizyanov/work/arch-pc/lab09/lab09-2
Hello, world!
```

Рис. 2.10: Проверка работы

Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её.

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
```

Рис. 2.11: брейкпоинт

```
(gdb) run
Starting program: /home/vagazizyanov/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
```

Рис. 2.12: Запуск

Посмотрите 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. 2.13: Просмотрим код

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
```

Рис. 2.14: переход на Intel

```

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

```

Рис. 2.15: проверка работы

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”).

Включите режим псевдографики для более удобного анализа программы

```
[ Register Values Unavailable ]

0x8049132      add     BYTE PTR [eax],al
0x8049134      add     BYTE PTR [eax],al
0x8049136      add     BYTE PTR [eax],al
0x8049138      add     BYTE PTR [eax],al
0x804913a      add     BYTE PTR [eax],al
0x804913c      add     BYTE PTR [eax],al
0x804913e      add     BYTE PTR [eax],al

native process 16424 In:  start                               L9      PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 2.16: псевдографика

2.3 Добавление точек останова

Проверяем точки останова с помощью команды `info breakpoints`

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
```

Рис. 2.17: `info breakpoints`

Установим еще одну точку останова по адресу инструкции

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
```

Рис. 2.18: точка останова

Посмотрите 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. 2.19: информация о точках

2.4 Работа с данными программы в GDB

Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров.

```
register group: general
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xffffd120   0xffffd120

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

active process 17936 In: start L14
breakpoint already hit 1 time
breakpoint keep y 0x08049031 lab09-2.asm:20
gdb) si
gdb) si
gdb) si
gdb) si
gdb) si
gdb)
```

Рис. 2.20: инструкции

Во время выполнения команд менялись регистры: `ebx`, `ecx`, `edx`, `eax`, `esp`.

Посмотреть содержимое регистров также можно с помощью команды `info registers`


```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd120 0xffffd120
ebp      0x0      0x0
esi      0x0      0

```

Рис. 2.21: содержание регистров

Посмотрите значение переменной msg1 по имени

```

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

```

Рис. 2.22: msg1

Посмотрите значение переменной msg2 по адресу

```

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

```

Рис. 2.23: msg2

Измените первый символ переменной msg1

```

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

```

Рис. 2.24: меняем символ

Замените любой символ во второй переменной msg2

```

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

```

Рис. 2.25: меняем символ

Выведете в различных форматах значение регистра ebx.

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

Рис. 2.26: значения

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

Рис. 2.27: значения

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

Завершите выполнение программы с помощью команды continue

```
(gdb) q  
A debugging session is active.  
  
Inferior 1 [process 17936] will be killed.  
Quit anyway? (y or n) y
```

Рис. 2.28: завершение

2.5 Обработка аргументов командной строки в GDB

Создаём файл.

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ touch lab09-3.asm
```

Рис. 2.29: создание файла

Скопируйте файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm

```
%include 'in_out.asm'
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в `ecx` количество
             ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в `edx` имя программы
             ; (второе значение в стеке)
    sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
             ; аргументов без названия программы)
    next:
    cmp ecx, 0 ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
             ; (переход на метку `_end`)
    pop eax ; иначе извлекаем аргумент из стека
    call sprintf ; вызываем функцию печати
    loop next ; переход к обработке следующего
             ; аргумента (переход на метку `next`)
    _end:
    call quit
```

Рис. 2.30: заполнение файла

Создайте исполняемый файл

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

Рис. 2.31: исполняемый файл

Загрузите исполняемый файл в отладчик, указав аргументы:

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '4'
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 lab09-3...
```

Рис. 2.32: загрузка в отладчик

установим точку останова перед первой инструкцией в программе и запустим ее.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/vagazizyanov/work/arch-pc/lab09/lab09-3 2 3 4

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
```

Рис. 2.33: точка останова

Посмотрите остальные позиции стека

```
(gdb) x/x $esp
0xffffd100: 0x00000004
```

Рис. 2.34: позиции стека

```
(gdb) x/s *(void**)(esp + 4)
0xffffd2db: "/home/vagazizyanov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd309: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd30b: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd30d: "4"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 2.35: позиции стека

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

2.6 Задание для самостоятельной работы

2.6.1 Задание 1

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

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в ecx количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в edx имя программы
0 ; (второе значение в стеке)
1 sub ecx,1 ; Уменьшаем ecx на 1 (количество
2 ; аргументов без названия программы)
3 mov esi, 0 ; Используем esi для хранения
4 ; промежуточных сумм
5 next:
6 cmp ecx,0h ; проверяем, есть ли еще аргументы
7 jz _end ; если аргументов нет выходим из цикла
8 ; (переход на метку `_end`)
9 pop eax ; иначе извлекаем следующий аргумент из стека
0 call atoi ; преобразуем символ в число
1 mov ebx,3
2 mul ebx
3 sub eax,1
4 add esi,eax ; добавляем к промежуточной сумме
5 ; след. аргумент esi=esi+eax
6 loop next ; переход к обработке следующего аргумента
7 _end:
8 mov eax, msg ; вывод сообщения "Результат: "
9 call sprint
0 mov eax, esi ; записываем сумму в регистр eax
1 call iprintLF ; печать результата
2 call quit ; завершение программы

```

Рис. 2.36: Копируем

Открываем файл и меняем его, создавая подпрограмму

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB "Введите x: ",0
5 result: DB '3x-1=',0
6 SECTION .bss
7 x: RESB 80
8 res: RESB 90
9 SECTION .text
10 global _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax, x
18 call atoi
19 call _calcul
20 mov eax, result
21 call sprint
22 mov eax, [res]
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx, 3
27 mul ebx
28 sub eax, 1
29 mov [res], eax
30 ret

```

Рис. 2.37: Редатируем

Создаем исполняемый файл и запускаем его

```

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

```

Рис. 2.38: Запуск

2.6.2 Задание 2

Создаём файл и вставляем текст из листинг 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. 2.39: Новый файл

Создаем исполняемый файл и запускаем его

```

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

```

Рис. 2.40: Запуск

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si


```

--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd100 0xffffd100

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

native process 38631 In: _start L9 PC: 0x80490e
eflags 0x202 [ 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--qQuit
(qdb) si

```

Рис. 2.41: Смотрим изменения

Изменяем программу для корректной работы

```

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov eax,3
9 mov ebx,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit

```

Рис. 2.42: корректируем

Создаем исполняемый файл и запускаем его

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

Рис. 2.43: проверка работы

3 Выводы

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