

Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Амина Аджигалиева

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	19
2.6	Задание для самостоятельной работы	20
3	Выводы	26

Список иллюстраций

2.1	Новый каталог и файл	6
2.2	Код программы	7
2.3	Запуск программы	8
2.4	Подпрограмма	9
2.5	Запуск программы	10
2.6	Новый файл	10
2.7	Код программы	11
2.8	Исходный файл	12
2.9	Загрузка	12
2.10	Проверка программы	12
2.11	Брейкпоинт	13
2.12	Дисассимилированный код	13
2.13	Intel'овский синтаксис	14
2.14	Режим псевдографики	15
2.15	Info breakpoints	15
2.16	Точки останова	16
2.17	Si	16
2.18	msg1	16
2.19	msg2	17
2.20	Замена символа	17
2.21	Замена символа	17
2.22	Значение регистра	18
2.23	Замена регистра	18
2.24	Завершение программы	19
2.25	Копия файла	19
2.26	Запуск программы	19
2.27	Точка останова	19
2.28	Адрес вершины стека	20
2.29	Позиции стека	20
2.30	Копия файла	20
2.31	Код программы	21
2.32	Запуск программы	22
2.33	Новый файл	22
2.34	Код программы	23
2.35	Запуск программы	23
2.36	Отладчик GDB	24
2.37	Замена программы	25

2.38 Запуск программы	25
---------------------------------	----

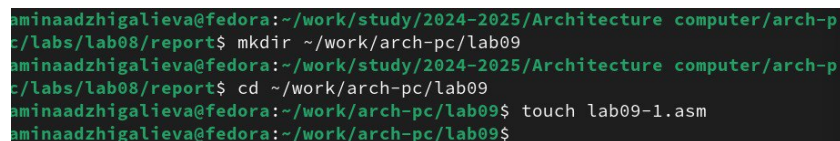
1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

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



```
aminaadzhigalieva@fedora:~/work/study/2024-2025/Architecture computer/arch-p  
c/labs/lab08/report$ mkdir ~/work/arch-pc/lab09  
aminaadzhigalieva@fedora:~/work/study/2024-2025/Architecture computer/arch-p  
c/labs/lab08/report$ cd ~/work/arch-pc/lab09  
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm  
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$
```

Рис. 2.1: Новый каталог и файл

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

```

%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:
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: Код программы

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

```
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab
09-1.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$
```

Рис. 2.3: Запуск программы

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


```
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: Подпрограмма

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

```
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab
09-1.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$
```

Рис. 2.5: Запуск программы

2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге (рис. 2.6).

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

Рис. 2.6: Новый файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. 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: Код программы

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

```

aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst
lab09-2.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab
09-2.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-2.fc41
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) run

```

Рис. 2.8: Исходный файл

Загрузим исполняемый файл в отладчик (рис. 2.9).

```

(gdb) run
Starting program: /home/aminaadzhigalieva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 24329) exited normally]
(gdb)

```

Рис. 2.9: Загрузка

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.10).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/aminaadzhigalieva/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.10: Проверка программы

Устанавливаем брейкпоинт на метку _start и запускаем программу (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: Брейкпоинт

Посмотрим дисассимилированный код программы (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.

```

Рис. 2.12: Дисассимилированный код

Переключимся на отображение команд с Intel'овским синтаксисом (рис. 2.13).

```

--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf50 0xffffcf50
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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

native process 24566 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 2.13: Intel'овский синтаксис

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 "+".
- 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”).

Включим режим псевдографики для более удобного анализа программы (рис. 2.14).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf50 0xffffcf50
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

0x8049574 add BYTE PTR [eax],al
0x8049576 add BYTE PTR [eax],al
0x8049578 add BYTE PTR [eax],al
0x804957a add BYTE PTR [eax],al
0x804957c add BYTE PTR [eax],al
0x804957e add BYTE PTR [eax],al
0x8049580 add BYTE PTR [eax],al
0x8049582 add BYTE PTR [eax],al
0x8049584 add BYTE PTR [eax],al
0x8049586 add BYTE PTR [eax],al

native process 24566 (asm) 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)
```

Рис. 2.14: Режим псевдографики

2.3 Добавление точек останова

Проверим это с помощью команды info breakpoints (рис. 2.15).

```
(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. 2.15: Info breakpoints

Посмотрим информацию о всех установленных точках останова (рис. 2.16).


```
(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
```

Рис. 2.16: Точки останова

2.4 Работа с данными программы в GDB

Выполняем 5 инструкций командой si (рис. 2.17).

```
Register group: general
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xffffcf50   0xffffcf50
ebp      0x0          0
esi      0x0          0
edi      0x0          0
eip      0x8049016    0x8049016 <_start+22>

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

native process 24566 (asm) In: _start L14 PC: 0x8049016
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
3        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.17: Si

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

Смотрим значение переменной msg1 по имени (рис. 2.18).

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

Рис. 2.18: msg1

Смотрим значение переменной msg2 по адресу (рис. 2.19).

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

Рис. 2.19: msg2

Изменим первый символ переменной msg1 (рис. 2.20).

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

Рис. 2.20: Замена символа

Изменим первый символ переменной msg2 (рис. 2.21).

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

Рис. 2.21: Замена символа

Смотрим значение регистра edx в разных форматах (рис. 2.22).

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

Рис. 2.22: Значение регистра

Изменяем регистр ebx (рис. 2.23).

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

Рис. 2.23: Замена регистра

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

Прописываем команды для завершения программы и выхода из GDB (рис. 2.24).

```
(gdb) c
Continuing.
Lor!d!

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

Рис. 2.24: Завершение программы

2.5 Обработка аргументов командной строки в GDB

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. 2.25).

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

Рис. 2.25: Копия файла

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. 2.26).

```
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
```

Рис. 2.26: Запуск программы

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 2.27).

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

Рис. 2.27: Точка останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (рис. 2.28).

```
(gdb) x/x $esp
0xffffcf40:      0x00000004
(gdb)
```

Рис. 2.28: Адрес вершины стека

Смотрим позиции стека по разным адресам (рис. 2.29).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd109:      "/home/aminaadzhigalieva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd13c:      "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd13e:      "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd140:      "5"
(gdb) x/s *(void**)(esp + 20)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.29: Позиции стека

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

2.6 Задание для самостоятельной работы

Задание 1. Копируем файл lab8-4.asm в файл с именем lab09-3.asm (рис. 2.30).

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

Рис. 2.30: Копия файла

Меняем код, создавая подпрограмму (рис. 2.31).

```

%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ', 0
    result: DB '2(x-1) = ', 0
SECTION .bss
    x: RESB 80
    res: RESD 1
SECTION .text
global _start
_start:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _func
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
    _func:
        sub eax, 1
        mov ebx, 2
        mul ebx
        mov [res], eax
        ret

```

Рис. 2.31: Код программы

Создаем исполняемый файл и запускаем его (рис. 2.32).

```
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 7
2(x-1) = 12
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$
```

Рис. 2.32: Запуск программы

Задание 2. Создаем новый файл (рис. 2.33).

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

Рис. 2.33: Новый файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. 2.34).

```

%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. 2.34: Код программы

Создаем исполняемый файл и запускаем его (рис. 2.35).

```

aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab
09-5.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10

```

Рис. 2.35: Запуск программы

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. 2.36).

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffcf50 0xffffcf50
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>

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 26741 (asm) In: _start L12 PC: 0x80490fb
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffcf50 0xffffcf50
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.36: Отладчик GDB

Изменяем программу для корректной работы (рис. 2.37).


```

#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. 2.37: Замена программы

Создаем исполняемый файл и запускаем его (рис. 2.38).

```

aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab
09-5.o
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
aminaadzhigalieva@fedora:~/work/arch-pc/lab09$

```

Рис. 2.38: Запуск программы

3 Выводы

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