

Лабораторная работа номер 9.

Понятие подпрограммы. Отладчик GDB.

Сорокин Кирилл

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
4.1	Самостоятельная работа	18
5	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Директории и файл	7
4.2	Листинг 9.1	8
4.3	Работа файла	9
4.4	Изменение кода	9
4.5	Работа файла	9
4.6	Листинг второй программы	10
4.7	Работа GBD	11
4.8	Запуск программы	11
4.9	Брейкпоинт	11
4.10	Дисассимилированный код	12
4.11	Переключение кода	12
4.12	layout asm	13
4.13	layout regs	13
4.14	break	14
4.15	Команда i b	14
4.16	Команда info registers	15
4.17	Значение msg1	15
4.18	Замена символа	15
4.19	Изменение msg2	16
4.20	Значение регистра edx	16
4.21	Значение ebx	17
4.22	Довыполним программу	17
4.23	lab09-3.asm	17
4.24	Запуск через gdb	18
4.25	Позиции в стеке	18
4.26	Изменённый текст 8 лабораторной	19
4.27	Работа самостоятельной	19
4.28	Текст второго файла	20
4.29	Работа второй программы	20
4.30	Брейкпоинт	21
4.31	Поиск ошибки	21
4.32	Код исправленной программы	22
4.33	Работа lab09-s2c	22

1 Цель работы

Научиться писать подпрограммы и работать с отладчиком GDB.

2 Задание

Используя написать подпрограммы внутри программ, а также проверять работу программ с помощью отладчика.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

4 Выполнение лабораторной работы

Создадим директории и файл для работы (рис. 4.1).

```
kvsorokin@dk4n68 ~ $ mkdir ~/work/arch-pc/lab09  
kvsorokin@dk4n68 ~ $ cd ~/work/arch-pc/lab09  
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 4.1: Директории и файл

Запишем листинг 9.1 в файл(рис. 4.2).

```

#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:
; -----
; Основная программа
; -----
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul ; Вызов подпрограммы _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
; -----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret ; выход из подпрограммы

```

Рис. 4.2: Листинг 9.1

Проверим работу файла(рис. 4.3).


```

kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1
.o
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ ./lab09
bash: ./lab09: Нет такого файла или каталога
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=11

```

Рис. 4.3: Работа файла

Изменим код программы для вложенных функций (рис. 4.4).

```

_calcul:
    call _subcalcul
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret ; выход из подпрограммы

_subcalcul:
    mov ebx, 3
    mul ebx
    add eax, -1
    mov [res], eax
    ret ; выход из подподпрограммы

```

Рис. 4.4: Изменение кода

Проверим работу (рис. 4.5).

```

kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ gedit lab09-1.asm
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1
.o
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2(3x-1)+7=17
kvsorokin@dk4n68 ~/work/arch-pc/lab09 $

```

Рис. 4.5: Работа файла

Введём листинг второй программы в новый файл (рис. 4.6).

```
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. 4.6: Листинг второй программы

Скомпилируем файлы и запустим в оболочке GDB командой `gdb lab09-2`(рис. 4.7).

```

GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.h
tml>
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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. 4.7: Работа GDB

Запустим программу командой run (рис. 4.8).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/v/kvsorokin/work/arch-pc
/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4469) exited normally]
(gdb)

```

Рис. 4.8: Запуск программы

Поставим брейкпоинт и запустим программу снова (рис. 4.9).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/v/kvsorokin/work/arch-pc
/lab09/lab09-2

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

```

Рис. 4.9: Брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 4.10).

```

(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.

```

Рис. 4.10: Дисассимилированный код

Переключим на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.11).

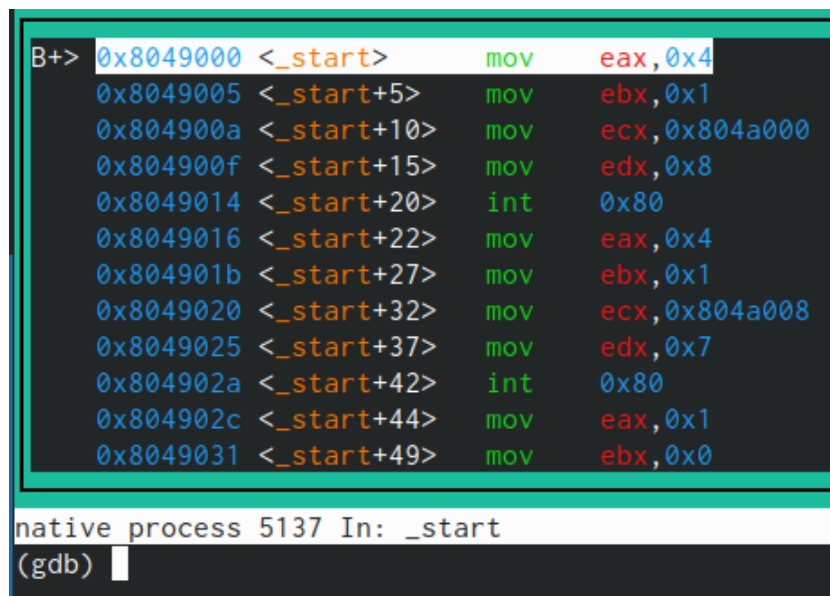
```

(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. 4.11: Переключение кода

Командой `layout asm`, откроем окно с дисассимилированием программы (рис. 4.12).

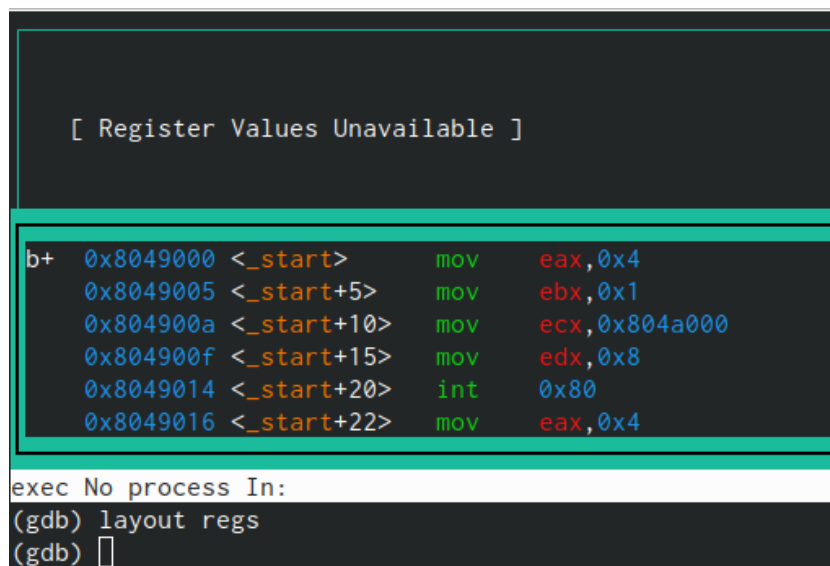


```
B+> 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x8
0x8049014 <_start+20>     int     0x80
0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
0x8049031 <_start+49>     mov     ebx,0x0

native process 5137 In: _start
(gdb)
```

Рис. 4.12: layout asm

Командой `layout regs`, откроем окно с названиями регистров и их текущие значения (рис. 4.13).



```
[ 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

exec No process In:
(gdb) layout regs
(gdb)
```

Рис. 4.13: layout regs

Командой `break` установим ещё один брейкпоинт (рис. 4.14).

```
[ Register Values Unavailable ]

0x804902a <_start+42>  int    0x80
0x804902c <_start+44>  mov    eax,0x1
b+ 0x8049031 <_start+49>  mov    ebx,0x0
0x8049036 <_start+54>  int    0x80

exec No process In:
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █
```

Рис. 4.14: `break`

Командой `i b` посмотрим позиции брейкпоинтов (рис. 4.15).

```
1      breakpoint      keep y   0x08049000  lab09-2.asm:9
2      breakpoint      keep y   0x08049031  lab09-2.asm:20
```

Рис. 4.15: Команда `i b`

Командой `info registers`, увидим информацию о регистрах (рис. 4.16).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc320 0xffffc320

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

native process 6394 In: _start L9 PC: 0x8
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc320 0xffffc320
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.16: Команда info registers

Посмотрим значение переменной msg1 по имени (рис. 4.17).

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

Рис. 4.17: Значение msg1

Изменим значение первой буквы на маленькую версию(рис. 4.18).

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

Рис. 4.18: Замена символа

Аналогично сделаем с msg2 (рис. 4.19).

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

Рис. 4.19: Изменение msg2

Выведем в различных форматах значение регистра edx (рис. 4.20).

```
(gdb) p/s $edx
$3 = 0
(gdb) p/x
$4 = 0x0
(gdb) p/t
$5 = 0
```

Рис. 4.20: Значение регистра edx

Заменяем значение ebx на '2', а затем на 2. Заметим, что выводится разное значение из-за разных форматов данных (рис. 4.21).


```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2

```

Рис. 4.21: Значение ebx

Довыполним программу командой c, затем выйдем из gdb командой quit(рис. 4.22).

```

(gdb) c
Continuing.
hello, World!

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

```

Рис. 4.22: Довыполним программу

Скопируем файл lab8-2.asm с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm и скомпилируем файлы (рис. 4.23).

```

kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm
~/work/arch-pc/lab09/lab09-3.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o

```

Рис. 4.23: lab09-3.asm

Запустим gdb передав аргументы вместе с файлом, постави брейкпоинт и увидим количество аргументов командной строки (рис. 4.24).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/v/kvsorokin/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
```

Рис. 4.24: Запуск через gdb

Посмотрим все позиции стека по адресу. Заметим, что все аргументы расположены на расстоянии в 4 байта друг от друга (рис. 4.25).

```
(gdb) x/x $esp
0xfffffc2d0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc56d: "/afs/.dk.sci.pfu.edu.ru/home/k/v/kvsorokin/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5b3: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5c5: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc5d6: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc5d8: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 4.25: Позиции в стеке

4.1 Самостоятельная работа

Изменим текст самостоятельной работы из 8 лабораторной так, чтобы в ней задействовалась подпрограмма (рис. 4.26).

```

#include 'in_out.asm'
SECTION .data
fun db "Функция: f(x)=2x+15", 0
msg db "Результат: ",0
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)
    mov esi, 0 ; Используем 'esi' для хранения
              ; промежуточных сумм

next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
    pop eax
              ; (переход на метку '_end')
              ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    call _calcul
    loop next ; переход к обработке следующего аргумента

_end:
    mov eax, fun
    call sprintf
    mov eax, msg ; вывод сообщения "Результат: "
    call sprintf
    mov eax, esi ; записываем сумму в регистр 'eax'
    call iprintLF; печать результата
    call quit ; завершение программы

_calcul:
    mov edx, 2
    mul edx
    add esi,eax ; добавляем к промежуточной сумме
              ; след. аргумент 'esi=esi+eax'
    mov edx, 15
    add esi, edx
    ret

```

Рис. 4.26: Изменённый текст 8 лабораторной

Проверим, что она всё ещё работает корректно (рис. 4.27).

```

kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ./lab09-s1 1 2
Функция: f(x)=2x+15
Результат: 36

```

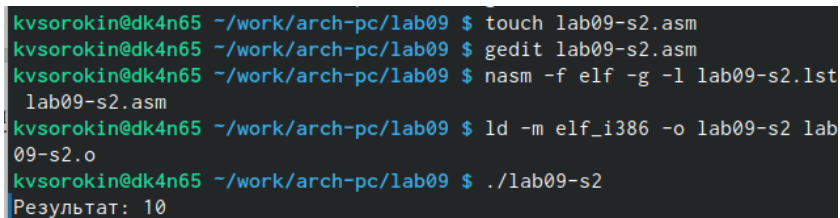
Рис. 4.27: Работа самостоятельной

Введём листинг 9.3 во второй файл самостоятельной работы (рис. 4.28).

```
%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. 4.28: Текст второго файла

Проверим работу и увидим, что результат не корректен (рис. 4.29).



```
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ touch lab09-s2.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ gedit lab09-s2.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-s2.lst
lab09-s2.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-s2 lab
09-s2.o
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ./lab09-s2
Результат: 10
```

Рис. 4.29: Работа второй программы

Запустим программу в отладчике и поставим брейкпоинт в месте, до того как прибавили 5 (рис. 4.30).

```

b+ 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>

exec No process In: L?
(gdb) b *0x80490fb
Breakpoint 1 at 0x80490fb: file lab09-s2.asm, line 13.
(gdb)

```

Рис. 4.30: Брейкпоинт

Увидим, что значение ebx не равно 20, значит проблема в умножении(рис. 4.31).

```

ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffc310 0xffffc310

    0x80490f2 <_start+10> add    ebx,eax
    0x80490f4 <_start+12> mov    ecx,0x4
    0x80490f9 <_start+17> mul    ecx
B+> 0x80490fb <_start+19> add    ebx,0x5
    0x80490fe <_start+22> mov    edi,ebx
    0x8049100 <_start+24> mov    eax,0x804a000
    0x8049105 <_start+29> call   0x804900f <sprint>

native process 17128 In: _start L13 PC
Breakpoint 1 at 0x80490fb: file lab09-s2.asm, line 13.
(gdb) layout regs
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/v/kvsorok
-pc/lab09/lab09-s2

Breakpoint 1, _start () at lab09-s2.asm:13
(gdb)

```

Рис. 4.31: Поиск ошибки

Исправим ошибку(рис. 4.32).

```

#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
    mov eax,3
    mov ebx,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
; ---- Вывод результата на экран
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
call quit

```

Рис. 4.32: Код исправленной программы

Удостоверимся, что теперь программа работает корректно (рис. 4.33).

```

kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ cp lab09-s2.asm lab09-s2c.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ gedit lab09-s2c.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-s2c.ls
t lab09-s2c.asm
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-s2c la
b09-s2c.o
kvsorokin@dk4n65 ~/work/arch-pc/lab09 $ ./lab09-s2c
Результат: 25

```

Рис. 4.33: Работа lab09-s2c

5 Выводы

Мы научились GDB, а также писать подпрограммы

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnightcommander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. -

874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. -СПб.
: Питер,

17. — 1120 с. — (Классика Computer Science)