

Отчёт по лабораторной работе №8

**Дисциплина: Компьютерные технологии и технологии
программирования**

ДЫМОВОЙ Д.Д.

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Задание для самостоятельной работы	16
5	Выводы	21
	Список литературы	22

Список иллюстраций

3.1	Создание каталога	7
3.2	Запуск исполняемого файла	7
3.3	Программа	8
3.4	Программа $f(g(x))$	9
3.5	Запуск файла	9
3.6	Оболочка GDB	9
3.7	Установка брейкпоинта	9
3.8	Просмотр кода программы	10
3.9	Переключение	10
3.10	Переключение режима	11
3.11	Точки брейкпоинта	11
3.12	Брейкпоинт	12
3.13	Точки останова	12
3.14	Информация о содержимом регистров	12
3.15	Значение переменной	13
3.16	Изменение переменной <code>msg1</code>	13
3.17	Значение аргумента <code>msg2</code>	13
3.18	Значение регистра <code>ebx</code>	14
3.19	Завершение программы	14
3.20	Выход из отладки	14
3.21	Запуск программы с аргументами 2 3, установка брейкпоинта	15
3.22	Позиции стека	15
4.1	Мною преобразованная программа	17
4.2	Запуск файла	17
4.3	Запуск файла	18
4.4	Регистр <code>ebx=5</code> , а должно быть <code>eax=5</code>	18
4.5	Ошибка в сложении	19
4.6	Проверка программы	19
4.7	Программа	20

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); • использование специальных программ-отладчиков.

3 Выполнение лабораторной работы

Создаю каталог для выполнения лабораторной работы №9, перехожу в него, создаю ассемблеровский файл для работы (рис. 3.1).

```
dddihmova@dk5n55 ~ $ mkdir ~/work/arch-pc/lab09
dddihmova@dk5n55 ~ $ cd ~/work/arch-pc/lab09
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 3.1: Создание каталога

Копирую текст листинга 9.1, создаю исполняемый файл и запускаю его (рис. 3.2).

```
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 3.2: Запуск исполняемого файла

Изменяю текст программы листинга 9.1, чтобы программа работала так, как указано в условии лабораторной работы (рис. 3.3).

```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',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 _subcalcul
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
_subcalcul:
mov ebx, 3
mul ebx
add eax, -1
mov [res], eax
ret ; выход из подпрограммы

```

Рис. 3.3: Программа

Создаю исполняемый файл и запускаю его (рис. 3.4).


```

dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
f(g(x))=11

```

Рис. 3.4: Программа $f(g(x))$

Создаю файл lab09-2.asm, вставляю туда текст программы листинга 9.2, сохраняю, создаю исполняемый файл и запускаю его (рис. 3.5).

```

dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ gdb lab09-2
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.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-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:
--Type <RET> for more, q to quit, c to continue without paging--(gdb) run

```

Рис. 3.5: Запуск файла

Запускаю файл в оболочке GDB (рис. 3.6).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/dddihmova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4419) exited normally]
(gdb)

```

Рис. 3.6: Оболочка GDB

Устанавливаю брейкпоинт на метку _start (рис. 3.7).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/dddihmova/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4

```

Рис. 3.7: Установка брейкпоинта

Просматриваю дисассимилированный код программы начиная с метки (рис. 3.8).

```

(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. 3.8: Просмотр кода программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.9).

```

(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. 3.9: Переключение

ПЕРЕЧИСЛИТЬ РАЗЛИЧИЯ ОТОБРАЖЕНИЯ СИНТАКСИСА

Включаю режим псевдографики для более удобного анализа программы (рис. 3.10).

The screenshot shows a debugger window with a dark background. At the top, there is a message: "[Register Values Unavailable]". Below this, a list of assembly instructions is displayed, each with its address and disassembly. The instructions are:

- 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
- 0x8049036 <_start+54> int 0x80

 Below the assembly list, there is a status bar showing "native process 4489 In: _start" and "L9 PC: 0x8049000". At the bottom, there are two lines of text: "(gdb) layout regs" and "(gdb) █".

Рис. 3.10: Переключение режима

Просмотр точек брейкпоинта (рис. 3.11).

The screenshot shows a debugger window with a dark background. It displays the output of the "(gdb) i b" command, which lists the breakpoints. The output is:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab09-2.asm:9

 Below the table, it says "breakpoint already hit 1 time".

Рис. 3.11: Точки брейкпоинта

Устанавливаю брейкпоинт на инструкцию mov ebx, 0x0 (рис. 3.12).

```

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
b+ 0x8049031 <_start+49>   mov     ebx,0x0
0x8049036 <_start+54>     int     0x80

exec No process in:
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint        keep y  0x08049000 lab09-2.asm:9
2     breakpoint        keep y  0x08049031 lab09-2.asm:20
(gdb) i c

```

Рис. 3.12: Брейкпоинт

Информация о всех точках останова (рис. 3.13).

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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. 3.13: Точки останова

Просматриваю содержимое регистров с помощью команды i r (рис. 3.14).

```

native process 4789 In: _start
Start process 4886 In: _start or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/...
eax      0x4          4
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffc320   0xffffc320
ebp      0x0          0
esi      0x0          0
edi      0x0          0
eip      0x8049005     0x8049005 <_start+5>
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--

```

Рис. 3.14: Информация о содержимом регистров

Смотрю значение переменной msg1 (рис. 3.15).

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

Рис. 3.15: Значение переменной

Заменяю значение переменной msg1 (рис. 3.16).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&0x804a008='L'
(gdb) █
```

Рис. 3.16: Изменение переменной msg1

Просматриваю значение аргумента msg2. 0x804a008 адрес msg2 (рис. 3.17).

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

Рис. 3.17: Значение аргумента msg2

Просматриваю значение регистра ebx в разных форматах (рис. 3.18).

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

Рис. 3.18: Значение регистра ebx

Завершаю выполнение программы с помощью команды q (рис. 3.19).

```
Active process 4267 in: _start  
(gdb) c  
Continuing.  
hello, world!  
Breakpoint 2, _start () at lab09-2.asm:20
```

Рис. 3.19: Завершение программы

Выхожу из gdb с помощью команды q (рис. 3.20).

```
(gdb) q  
A debugging session is active.  
  
Inferior 1 [process 4267] will be killed.  
Quit anyway? (y or n) █
```

Рис. 3.20: Выход из отладки

Я копирую файл lab8-2.asm в папку с лабораторной номер 9 и называю его lab09-2.asm, создаю исполняемый файл и запускаю его через gdb. Эта программы

должна находить произведение аргументов. Также я ставлю точку останова на месте start (рис. 3.21).

```
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ gdb --args lab09-3 2 3
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.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-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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/dddihmova/work/arch-pc/lab09/lab09-3 2 3

Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 3.21: Запуск программы с аргументами 2 3, установка брейкпоинта

Рассматриваю позиции стека, так как у меня всего два аргумента на шаге (+16) выдаёт ошибку (рис. 3.22).

```
(gdb) x/x $esp
0xfffffc310: 0x00000003
(gdb) x/s *(void**)(esp + 4)
0xfffffc59f: "/afs/.dk.sci.pfu.edu.ru/home/d/dddihmova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5e5: "2"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5e7: "3"
(gdb) x/s *(void**)(esp + 16)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 3.22: Позиции стека

4 Задание для самостоятельной работы

Создаю файл `zadanie1.asm` для выполнения первого задания из самостоятельной работы. Ввожу текст программы листинга 9.1 для удобства. Пишу программу согласно 13 варианту лабораторной работы номер 8 (рис. 4.1).


```

#include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '12x-7=',0

SECTION .bss
    x: RESB 80
    res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg; вывод сообщение введите x
call sprint

mov ecx, x; занесение x в регистр
mov edx, 80
call sread

mov eax,x;преобразование из символа в число
call atoi

call _zadanie

mov eax,result; вывод сообщения 12x-7=
call sprint
mov eax,[res]; вывод результат
call iprintLF

call quit

_zadanie:
mov ebx,12
mul ebx
add eax, -7
mov [res],eax
ret

```

Рис. 4.1: Мною преобразованная программа

Создаю исполняемый файл и запускаю его (рис. 4.2).

```

dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf zadanie1.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o zadanie1 zadanie1.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ./zadanie1 1
Введите x: 1
12x-7=5
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ./zadanie1
Введите x: 2
12x-7=17

```

Рис. 4.2: Запуск файла

Создаю файл `zadanie2.asm` для выполнения второго пункта лабораторной работы, ввожу туда текст листинга 9.3, сохраняю, создаю исполняемый файл и запускаю его. Убеждаюсь, что результат неверный. Ответ должен быть 25 (рис. 4.4).

Register group: general		
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x5	5
esp	0xffffc320	0xffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f4	0x80490f4 <_start+12>
eflags	0x206	[PF IF]
cs	0x23	35
ss	0x2b	43

Рис. 4.3: Запуск файла

Запускаю отладчик, смотрю как изменяются регистры `eax`, `ebx`, `ecx` пошагово с помощью команды `si`. Замечаю, что на третьем шаге регистр `ebx` имеет значение 5, а на четвертом шаге `ecx` и `eax` перемножаются, что и дает неверный результат. Следовательно изменяю программу так, чтобы результат сложения записывался в `eax` (рис. 4.4).

Register group: general		
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x5	5
esp	0xffffc320	0xffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f4	0x80490f4 <_start+12>
eflags	0x206	[PF IF]
cs	0x23	35
ss	0x2b	43

Рис. 4.4: Регистр `ebx=5`, а должно быть `eax=5`

У меня снова вывелся неправильный результат, пошагово проверяю как изменяются регистры, замечаю что в следующем сложении 5 прибавляется к регистру `ebx`, а должно к регистру `eax`, и в конце результат `edi` записывается в `ebx`, а должен в `eax`, изменяю и эту строчку (рис. 4.5).

Register group: general		
eax	0x14	20
ecx	0x4	4
edx	0x0	0
ebx	0x8	8
esp	0xffffc320	0xffffc320
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490fe	0x80490fe <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43

Рис. 4.5: Ошибка в сложении

Создаю исполняемый файл и запускаю его (рис. 4.6).

```
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ nasm -f elf zadanie2.asm
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o zadanie2 zadanie2.o
dddihmova@dk5n55 ~/work/arch-pc/lab09 $ ./zadanie2
Результат: 25
```

Рис. 4.6: Проверка программы

Текст моей программы (рис. 4.7).

```
%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 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. 4.7: Программа

5 Выводы

Я приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы