

# **Отчёт по лабораторной работе №10**

**Дисциплина: 'архитектура компьютеров'**

Бабенко Роман Игоревич

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	24

# Список иллюстраций

2.1	Создание каталога и файла . . . . .	6
2.2	Вводим программу в файл . . . . .	7
2.3	Проверка работы файла . . . . .	8
2.4	Добавляем подпрограмму . . . . .	9
2.5	Проверка программы . . . . .	10
2.6	Записываем программу из листинга . . . . .	11
2.7	Используем отладчик . . . . .	12
2.8	Запуск программы в отладчике . . . . .	12
2.9	Установка брейкпоинта . . . . .	12
2.10	Просмотр дисассимилированного кода . . . . .	13
2.11	Переключение синтаксиса . . . . .	14
2.12	Включение режима псевдографики . . . . .	14
2.13	Проверка точки останова . . . . .	15
2.14	Установка второй точки останова . . . . .	15
2.15	Просмотр содержимого регистров . . . . .	15
2.16	Просмотр значения msg1 . . . . .	16
2.17	Просмотр значения msg2 . . . . .	16
2.18	Изменяем символ в переменной msg1 . . . . .	16
2.19	Изменим символ в msg2 . . . . .	16
2.20	Вывод значения регистра edx . . . . .	17
2.21	Изменяем значение ebx . . . . .	18
2.22	Завершаем выполнение программы . . . . .	18
2.23	Создание исполняемого файла для программы из девятой лабора- торной работы . . . . .	18
2.24	Загружаем в gdb программу с аргументами . . . . .	19
2.25	Делаем точку останова и запускаем инструкцию . . . . .	19
2.26	Смотрим позиции стека . . . . .	19
2.27	Написанная программа . . . . .	20
2.28	Проверка программы . . . . .	21
2.29	Исправленная программа . . . . .	22
2.30	Полученный результат . . . . .	23

## **Список таблиц**

# 1 Цель работы

Приобрести навыки программирования с использованием подпрограмм и познакомиться с методом отладки 'GDB'

## 2 Выполнение лабораторной работы

Создаём каталог для выполнения лабораторной работы, переходим в него и создаём файл lab10-1.asm (рис. 2.1)

```
ribabenko@dk6n57 ~ $ mkdir ~/work/arch-pc/lab10  
ribabenko@dk6n57 ~ $ cd ~/work/arch-pc/lab10  
ribabenko@dk6n57 ~/work/arch-pc/lab10 $ touch lab10-1.asm
```

Рис. 2.1: Создание каталога и файла

Вводим в файл программу из листинга, создаем исполняемый файл и проверяем его работу (рис. 2.2) и (рис. 2.3)

```

%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

```

Рис. 2.2: Вводим программу в файл

```
nasm -f elf lab10-1.asm
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 2
2x+7=11
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 0
2x+7=7
```

Рис. 2.3: Проверка работы файла

Добавляем подпрограмму '\_subcalcul'(рис. 2.4)



```
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: Добавляем подпрограмму

Проверяем изменённую программу (рис. 2.5)

```
nasm -f elf lab10-1.asm
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 2
f(g(x)) = 2(3x-1)+7=17
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 1
f(g(x)) = 2(3x-1)+7=11
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 10
f(g(x)) = 2(3x-1)+7=65
ribabenko@dk8n63 ~/work/arch-pc/lab10 $
```

Рис. 2.5: Проверка программы

Создаём файл lab10-2.asm и записываем в него программу из листинга 10.2 (рис. 2.6)

```

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. 2.6: Записываем программу из листинга

Загружаем исполняемый файл в отладчик gdb (рис. 2.7)

```
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
ribabenko@dk8n63 ~/work/arch-pc/lab10 $ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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 lab10-2...
(gdb) █
```

Рис. 2.7: Используем отладчик

Запускаем программу с помощью команды run (рис. 2.8)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/r/i/ribabenko/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 4100) exited normally]
(gdb) █
```

Рис. 2.8: Запуск программы в отладчике

Устанавливаем брейкпоинт на метку \_start и далее запускаем её (рис. 2.9)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) █
```

Рис. 2.9: Установка брейкпоинта

Смотрим дисассимилированный код с помощью программы disassemble начиная с метки \_start (рис. 2.10)

```

(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.
(gdb) █

```

Рис. 2.10: Просмотр дисассимилированного кода

Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду 'set disassembly-flavor intel'. С интеловском синтаксе, в отличие от стандартного, в третьем столбце сначала пишется название регистра(рис. 2.11)

```

(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.
(gdb) █

```

Рис. 2.11: Переключение синтаксиса

Включаем режим псевдографики для более удобного анализа программы (рис. 2.12)

```

Register group: general
eax 0x0 0 ecx 0x0 0
edx 0x0 0 ebx 0x0 0
esp 0xffffc4b0 0xffffc4b0 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x8049000 0x8049000 <_start> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43

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

native process 3211 In: _start L12 PC:
(gdb) layout regs
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab10-2.asm:12
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/r/i/ribabenko/work/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:12
(gdb) █

```

Рис. 2.12: Включение режима псевдографики

Проверяем наличие точки останова с помощью команды info breakpoints (рис. 2.13)

```
(gdb) info b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab10-2.asm:12
(gdb)
```

Рис. 2.13: Проверка точки останова

Устанавливаем ещё одну точку останова по адресу инструкции и смотрим информацию о всех точках останова (рис. 2.14)

```
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:
Undefined command: "into". Try "help".
(gdb) info b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab10-2.asm:12
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 25.
(gdb) info b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab10-2.asm:12
2        breakpoint    keep y  0x08049031 lab10-2.asm:25
(gdb)
```

Рис. 2.14: Установка второй точки останова

Просматриваем содержимое регистров с помощью команды info registers (рис. 2.15)

```
(gdb) info r
```

Рис. 2.15: Просмотр содержимого регистров

Просматриваем значение переменной msg1 по имени (рис. 2.16)

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

Рис. 2.16: Просмотр значения msg1

Просматриваем значение переменной msg2 по адресу(рис. 2.17)

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

Рис. 2.17: Просмотр значения msg2

Изменим первый символ переменной msg1 с помощью команды set(рис. 2.18)

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

Рис. 2.18: Изменяем символ в переменной msg1

Также заменим символ в переменной msg2(рис. 2.19)

```
(gdb) set {char}0x804a00b=' '
(gdb) x &msg2
0x804a008 <msg2>:      "wor d!\n\034"
(gdb) █
```

Рис. 2.19: Изменим символ в msg2

Выведем значения регистра edx в разных форматах(рис. 2.20)



```
(gdb) p/x $edx
$19 = 0x32

(gdb) p/t $edx
$20 = 110010

(gdb) p/s $edx
$21 = 50
```

Рис. 2.20: Вывод значения регистра edx

С помощью команды set изменим значение регистра ebx. С p/s выводит значение в символьном виде(рис. 2.21)

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$22 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$23 = 2
(gdb)

```

Рис. 2.21: Изменяем значение ebx

Завершаем выполнение программы с помощью 'continue'(рис. 2.22)

```

(gdb) continue
Continuing.
Hello, world!
[Inferior 1 (process 3447) exited normally]
(gdb) quit

```

Рис. 2.22: Завершаем выполнение программы

Копируем файл созданный при выполнении лабораторной работы номер девять и создадим исполняемый файл(рис. 2.23)

```

ribabenko@dk6n60 ~/work/arch-pc/lab10 $ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
ribabenko@dk6n60 ~/work/arch-pc/lab10 $ masm -f elf -g -l lab10-3.lst lab10-3.asm
bash: masm: команда не найдена
ribabenko@dk6n60 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-3.lst lab10-3.asm
ribabenko@dk6n60 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-3 lab10-3.o
ribabenko@dk6n60 ~/work/arch-pc/lab10 $

```

Рис. 2.23: Создание исполняемого файла для программы из девятой лабораторной работы

Загружаем в gdb программу с аргументами используя ключ -args(рис. 2.24)

```

ribabenko@dk6n60 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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:
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.24: Загружаем в gdb программу с аргументами

Устанавливаем точку останова перед первой инструкцией и запускаем её(рис. 2.25)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/r/i/ribabenko/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab10-3.asm:7
7       pop ecx
(gdb)

```

Рис. 2.25: Делаем точку останова и запускаем инструкцию

Посмотрим различные позиции стека(рис. 2.26)

```

(gdb) x/x $esp
0xfffffc470:      0x00000005
(gdb) x/s *(void**)($est + 4)
Argument to arithmetic operation not a number or boolean.
(gdb) x/s *(void**)($esp + 8)
0xfffffc74d:      "аргумент1"
(gdb) x/s *(voit**)($esp + 12)
No symbol "voit" in current context.
(gdb) x/s *(vois**)($esp + 12)
No symbol "vois" in current context.
(gdb) x/s *(void**)($esp + 12)
0xfffffc75f:      "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffc770:      "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffc772:      "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.26: Смотрим позиции стека

### #Задания для самостоятельной работы

Напишем программу из лабораторной работы номер девять и реализуем в ней вычисление функции через подпрограмму(рис. 2.27)

```

#include 'in_out.asm'

SECTION .data
msg: db 'Сумма значений функции 3(x+2) при аргументах: '

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx,1
    mov esi,0

next:
    cmp ecx,0h
    jz _end

    pop eax
    call atoi
    call _function
    add esi,eax
    mov eax,0

    loop next

_end:
    mov eax,msg
    call sprint
    mov eax,esi
    call iprintLF
    call quit

_function:
    add eax,2
    mov edx,3
    mul edx

    ret

```

Рис. 2.27: Написанная программа

Проверяем нашу программу(рис. 2.28)

```

ribabenko@dk8n68 ~/work/arch-pc/lab10 $ nasm -f elf zadanie10.asm
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o zadanie10 zadanie10.o
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ ./zadanie10 2 7
Сумма значений функции 3(x+2) при аргументах: 39
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ ./zadanie10 1 2
Сумма значений функции 3(x+2) при аргументах: 21
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ ./zadanie10 1 2 3
Сумма значений функции 3(x+2) при аргументах: 36
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ █

```

Рис. 2.28: Проверка программы

Исправляем программу для вычисления умножения, поменяв регистры местами (там умножался на 4 не регистр ebx, f htbcsnh eax)(рис. 2.29)

```
%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. 2.29: Исправленная программа

Проверяем исправленную программу(рис. 2.30)

```
ribabenko@dk8n68 ~/work/arch-pc/lab10 $ ./file0  
Результат: 25
```

Рис. 2.30: Полученный результат

## 3 Выводы

В ходе выполнения данной лабораторной работы я приобрёл навыки программирования с использованием подпрограмм и познакомился с методом отладки 'GDB'