

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Рахматова Жылдыз Талантбековна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Самостоятельная работа	17
4	Вывод	22

Список иллюстраций

2.1	Создание каталога и файла	6
2.2	Текст программы	7
2.3	Работа программы	7
2.4	Измененный текст программы	8
2.5	Проверка работы программы	8
2.6	Текст второй программы	9
2.7	Отладка второго файла	10
2.8	Брежпоинт на метку _start	10
2.9	Дисассимплированный код	11
2.10	Intel'овское отображение	11
2.11	Псевдографика	12
2.12	Наличие меток	12
2.13	Просмотр регистров	13
2.14	Измененные регистры	13
2.15	Просмотри значения переменной	13
2.16	Значение переменной msg2	14
2.17	Изменение значения переменной	14
2.18	Изменение msg2	14
2.19	Значение регистров ехх и еах	14
2.20	Значение регистров еbх	15
2.21	Завершение работы с файлов	15
2.22	Запуск файла в отладчике	15
2.23	Запуск файла lab9-3 через метку	15
2.24	Адрес вершины стека	15
2.25	Все позиции стека	16
3.1	Текст программы	18
3.2	Запуск программы	19
3.3	Текст программы	20
3.4	Запуск программы	21
3.5	Запуск программы в отладчике	21
3.6	Анализ регистров	21
3.7	Повторный запуск программы	21


Список таблиц

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

1) Я создала каталог lab09 и создал файл lab9-1.asm



```
ztrakhmatova@dk3n55 ~ $ mkdir ~/work/arch-pc/lab09
ztrakhmatova@dk3n55 ~ $ cd ~/work/arch-pc/lab09
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ touch lab9-1.asm
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создание каталога и файла

2)Я ввела текст листинга в файл и запустила программу.

```

#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

```

Рис. 2.2: Текст программы

```

ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 5
2x+7=17

```

Рис. 2.3: Работа программы

3) Я изменила текст программы, чтобы она решала выражение $f(g(x))$.

```

lab9-1.asm      [----] 15 L:[ 1+ 2
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x) = 3x-1',0
result: DB 'f(g(x))=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
-----
    Основная программа
-----

mov eax,prim1
call printf

mov eax,prim2
call printf

```

Рис. 2.4: Измененный текст программы

```

ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ ./lab9-1
f(x) = 2x+7
g(x) = 3x-1

```

Рис. 2.5: Проверка работы программы

4)Я создала файл lab9-2.asm и вписала туда программу.


```
lab9-2.asm      [-M--]  8 L:[  1+  
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: Текст второй программы

5)Я загрузила и запустила файл второй программы в отладчик gdb.

```

ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-2.lst lab9-2.asm
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
ztrakhmatova@dk3n55 ~/work/arch-pc/lab09 $ gdb lab9-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 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 lab9-2...

```

Рис. 2.7: Отладка второго файла

6) Я поставила брекпоинт на метку `_start` и запустила программу.

```

Reading symbols from lab9-2...
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/z/t/ztrakhmatova/work/arch/9/lab9-2
Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4

```

Рис. 2.8: Брекпоинт на метку `_start`

7) Я просмотрела дисассимплированный код программы начиная с метки.

```
(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. 2.9: Дисассимплированный код

- 8) С помощью команды я переключилась на intel'овское отображение синтаксиса. Отличие заключается в командах, в дисассимилированном отображении в командах используют % и \$, а в Intel отображение эти символы не используются. На такое отображение удобнее смотреть.

```
(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. 2.10: Intel'овское отображение

9) Для удобства я включила режим псевдографики.

```
[ 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

native process 5701 In: _start L9 PC: 0
(gdb) layout regs
(gdb) █
```

Рис. 2.11: Псевдографика

10) Я посмотрела наличие меток и добавила еще одну метку на предпоследнюю инструкцию.

```
native process 5701 In: _start L9 PC: 0
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y   0x08049031 lab9-2.asm:20
(gdb) █
```

Рис. 2.12: Наличие меток

11) С помощью команды si я посмотрела регистры и изменила их.

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc490 0xffffc490
ebp      0x0      0x0

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 5701 In: _start
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 lab9-2.asm:20
(gdb) si
(gdb)

```

Рис. 2.13: Просмотр регистров

```

eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
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

```

Рис. 2.14: Измененные регистры

12) С помощью команды я посмотрела значение переменной msg1.

```

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

```

Рис. 2.15: Просмотр значения переменной

13) Следом я посмотрела значение второй переменной msg2.

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

Рис. 2.16: Значение переменной msg2

14) С помощью команды set я изменила значение переменной msg1.

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhllo, "
(gdb)
```

Рис. 2.17: Изменение значения переменной

15) Я изменила переменную msg2.

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor d!\n\034"
(gdb)
```

Рис. 2.18: Изменение msg2

16) Я вывела значение регистров ecx и eax.

```
(gdb) p/f $msg1
$2 = void
(gdb) p/s $eax
$3 = 4
(gdb) p/t $eax
$4 = 100
(gdb) p/c $ecx
$5 = 0 '\000'
(gdb) p/x $ecx
$6 = 0x0
(gdb)
```

Рис. 2.19: Значение регистров ecx и eax

- 17) Я изменила значение регистра ebx. Команда выводит два разных значения так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные.

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$7 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$8 = 2
```

Рис. 2.20: Значение регистров ebx

- 18) Я завершила работу с файлов вышел.

```
[Inferior 1 (process 3985) exited normally]
```

Рис. 2.21: Завершение работы с файлов

- 19) Я скопировала файл lab8-2.asm и переименовала его. Запустила файл в отладчике и указала аргументы.

Запуск файла в отладчике

Рис. 2.22: Запуск файла в отладчике

- 20) Поставила метку на _start и запустила файл.

Запуск файла lab9-3 через метку

Рис. 2.23: Запуск файла lab9-3 через метку

- 21) Я проверила адрес вершины стека и убедилась что там хранится 5 элементов.

```
(gdb) x/x $esp  
0xffffd180: 0x00000005  
(gdb)
```

Рис. 2.24: Адрес вершины стека

22) Я посмотрела все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

Все позиции стека

Рис. 2.25: Все позиции стека

3 Самостоятельная работа

- 1) Я преобразовала программу из лабораторной работы №8 и реализовала вычисления как подпрограмму.

```

#include 'in_out.asm'

SECTION .data
prim DB 'f(x)=2x+15',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end

pop eax
call atoi
call fir
add esi,eax

loop next

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

fir:
mov ebx,2
mul ebx
add eax,15
ret

```

Рис. 3.1: Текст программы

Запуск программы

Рис. 3.2: Запуск программы

- 2) Я переписала программу и попробовала запустить ее чтобы увидеть ошибку. Ошибка была арифметическая, так как вместо 25, программа выводит 10.

```
5.asm [-M--
#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. 3.3: Текст программы

Запуск программы

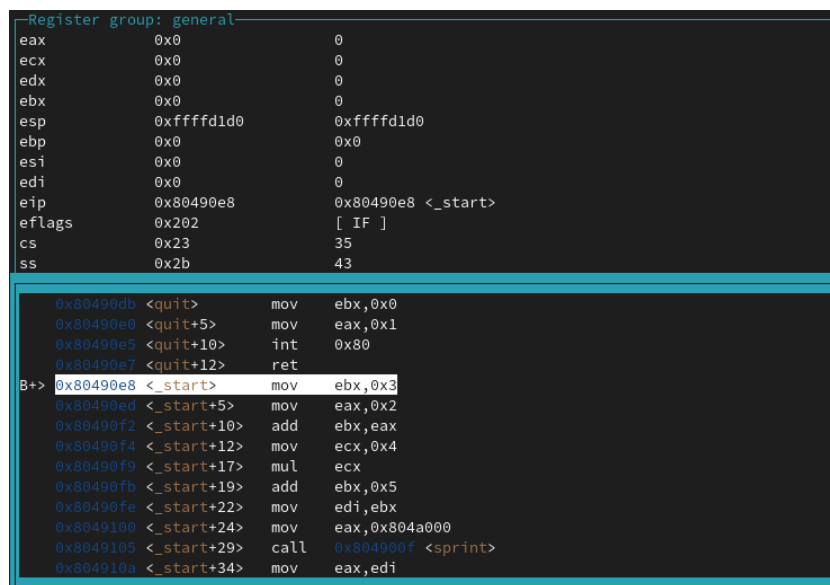
Рис. 3.4: Запуск программы

После появления ошибки, я запустила программу в отладчике.

Запуск программы в отладчике

Рис. 3.5: Запуск программы в отладчике

Я открыла регистры и проанализировала их, поняла что некоторые регистры стоят не на своих местах и исправила это.



```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

0x80490db <quit>      mov     ebx,0x0
0x80490e0 <quit+5>    mov     eax,0x1
0x80490e5 <quit+10>   int     0x80
0x80490e7 <quit+12>   ret
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
0x8049100 <_start+24> mov     eax,0x804a000
0x8049105 <_start+29> call    0x804900f <sprint>
0x804910a <_start+34> mov     eax,edi
```

Рис. 3.6: Анализ регистров

Я изменила регистры и запустила программу, программа вывела ответ 25, то есть все работает правильно.

Повторный запуск программы

Рис. 3.7: Повторный запуск программы

4 Вывод

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