

Отчёт по лабораторной работе №9

НПМбв-02-21

Гугульян Ксения Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	21
	Список литературы	22

Список иллюстраций

3.1	Создание каталога	8
3.2	Ввод текста в файл	9
3.3	Создание исп. файла	9
3.4	Создание файла с текстом	10
3.5	Загрузка исп. файла в отладчик	11
3.6	Проверка	11
3.7	Установка брейкпоинт	11
3.8	Просмотр программы	12
3.9	Переключение	12
3.10	Включение режима	13
3.11	Команда info breakpoints	14
3.12	Определение адреса и просмотр информации	15
3.13	Инструкции с помощью stepi	16
3.14	Просмотр значения переменной msg1 по имени	17
3.15	Просмотр значения переменной msg2 по адресу	17
3.16	Изменение символов переменной msg1	17
3.17	Замена символов переменной msg2	18
3.18	Ввод в различных форматах регистра edx	18
3.19	Изменение регистра ebx	19
3.20	Копирование файла, создание исп. файла и загрузка	20
3.21	Исследование расположения аргументов к.с. в стеке	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 Задание

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm.
2. Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\mathbb{X}(\mathbb{X}(\mathbb{X}))$, где \mathbb{X} вводится с клавиатуры, $\mathbb{X}(\mathbb{X}) = 2\mathbb{X} + 7$, $\mathbb{X}(\mathbb{X}) = 3\mathbb{X} - 1$.
3. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. Получите исполняемый файл. Загрузите исполняемый файл в отладчик gdb. Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run`.
4. Установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её. Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Включаем режим псевдографики для более удобного анализа программы.
5. На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints`. Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова. Посмотрите информацию о всех установленных точках останова.

6. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотрите значение переменной `msg1` по имени. Посмотрите значение переменной `msg2` по адресу. Измените первый символ переменной `msg1`. Замените любой символ во второй переменной `msg2`. Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. С помощью команды `set` измените значение регистра `ebx`.
7. Скопируйте файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm`. Создайте исполняемый файл. Загрузите исполняемый файл в отладчик, указав аргументы. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью `gdb`.

3 Выполнение лабораторной работы

1. Создаём каталог для выполнения лабораторной работы № 9, перейдём в него и создаём файл lab09-1.asm (рис. 3.1).

```
kaguguljyan@dk6n62 ~ $ mkdir ~/work/arch-pc/lab09
kaguguljyan@dk6n62 ~ $ cd ~/work/arch-pc/lab09
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ touch lab09-1.asm
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание каталога

2. Введём в файл lab09-1.asm текст программы из листинга 9.1 (рис. 3.2).


```

lab09-1.asm      [-M--]  0 L:[ 1+29 30/ 30]
%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. 3.2: Ввод текста в файл

Создаём исполняемый файл и проверяем его работу (рис. 3.3).

```

kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 10
2x+7=27
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ █

```

Рис. 3.3: Создание исп. файла

3. Создаём файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. 3.4).



```
lab09-2.asm [-M--] 0 L:[ 1+21 22,
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. 3.4: Создание файла с текстом

Получаем исполняемый файл и загружаем исполняемый файл в отладчик gdb (рис. 3.5).

```

kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
kaguguljyan@dk6n62 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
kaguguljyan@dk6n62 ~/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:
<http://www.gnu.org/software/gdb/documentation/>.

```

Рис. 3.5: Загрузка исп. файла в отладчик

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. 3.6).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/a/kaguguljyan/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 9748) exited normally]
(gdb)

```

Рис. 3.6: Проверка

4. Установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её (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/k/a/kaguguljyan/work/arch-pc/lab09/lab09-2

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

```

Рис. 3.7: Установка брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (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.
(gdb) □
```

Рис. 3.8: Просмотр программы

Переключимся 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.
(gdb) □
```

Рис. 3.9: Переключение

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

```
[ 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
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

native process 9800 In: _start L9
(gdb) layout regs
(gdb) □
```

Рис. 3.10: Включение режима

5. Проверим это с помощью команды `info breakpoints` (рис. 3.11).

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

native process 3626 In: _start
(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. 3.11: Команда info breakpoints

Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова. Посмотрим информацию о всех установленных точках останова (рис. 3.12).

```

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
0x8049038          add     BYTE PTR [eax],al
0x804903a          add     BYTE PTR [eax],al
0x804903c          add     BYTE PTR [eax],al

```

```

native process 3626 In: _start
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y
(gdb) break *0x8049031
Breakpoint 3 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  <PENDING>  0x8049031
3        breakpoint    keep y  0x08049031 lab09-2.asm:20
(gdb) 

```

Рис. 3.12: Определение адреса и просмотр информации

6. Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров (рис. 3.13).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc300 0xffffc300
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900a 0x804900a <_start+10>
eflags   0x202    [ IF ]
cs       0x23     35
ss       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
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

native process 3626 In: _start
breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y
(gdb) break *0x8049031
Breakpoint 3 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   <PENDING> 0x8049031
3      breakpoint      keep y   0x08049031 lab09-2.asm:20
(gdb) stepi
(gdb) si
(gdb) 

```

Рис. 3.13: Инструкции с помощью stepi

Меняются значения регистров esx.

Посмотрим значение переменной msg1 по имени (рис. 3.14).


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

Рис. 3.14: Просмотр значения переменной msg1 по имени

Посмотрим значение переменной msg2 по адресу (рис. 3.15).

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

Рис. 3.15: Просмотр значения переменной msg2 по адресу

Изменим первый символ переменной msg1 (рис. 3.16).

```
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hh1lo, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hh1lo, "
(gdb) □
```

Рис. 3.16: Изменение символов переменной msg1

Заменяем любой символ во второй переменной msg2 (рис. 3.17).

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

Рис. 3.17: Замена символов переменной msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 3.18).

```
(gdb) p/s $edx  
$1 = 0  
(gdb) p/t $edx  
$2 = 0  
(gdb) p/s $edx  
$3 = 0  
(gdb) p/x $edx  
$4 = 0x0  
(gdb) □
```

Рис. 3.18: Ввод в различных форматах регистра edx

С помощью команды set изменим значение регистра ebx (рис. 3.19).

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$5 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$6 = 2  
(gdb) 
```

Рис. 3.19: Изменение регистра ebx

7. Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создаём исполняемый файл. Загружаем исполняемый файл в отладчик, указав аргументы (рис. 3.20).

```

kaguguljyan@dk6n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
kaguguljyan@dk6n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kaguguljyan@dk6n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
kaguguljyan@dk6n55 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 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) 

```

Рис. 3.20: Копирование файла, создание исп. файла и загрузка

Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb (рис. 3.21).

```

(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/a/kaguguljyan/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xfffffc2b0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc54f: "/afs/.dk.sci.pfu.edu.ru/home/k/a/kaguguljyan/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc597: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5a9: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc5ba: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc5bc: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) 

```

Рис. 3.21: Исследование расположения аргументов к.с. в стеке

4 Выводы

В ходе лабораторной работы я приобрела навыки написания программ с использованием подпрограмм. Ознакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы