

Отчёт по лабораторной работе № 9

Дисциплина: архитектура компьютера

Мусатова Екатерина Викторовна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	17

Список иллюстраций

2.1	Создание файлов и каталога	6
2.2	Проверка программы	6
2.3	Изменение программы	7
2.4	Проверка	7
2.5	Загрузка в отладчик	8
2.6	Запуск программы	8
2.7	Установка брейкпоинта	9
2.8	Просмотр кода	9
2.9	Переключение	10
2.10	Псевдографика	10
2.11	Проверка	11
2.12	проверка	11
2.13	Инструкции	12
2.14	Просмотр	12
2.15	Другой способ посмотреть значение переменной	13
2.16	Изменение символа	13
2.17	Замена символа в другой переменной	13
2.18	Изменение значения регистра	14
2.19	Завершение	14
2.20	Загрузка в отладчик	15
2.21	Точка основа и запуск	15
2.22	Проверка	15
2.23	Просмотр	16

Список таблиц

1 Цель работы

Освоить работу с подпрограммами и отладчиком gdb.

2 Выполнение лабораторной работы

Создаю необходимый каталог и файлы (рис. 2.1).

```
[evmusatova@fedora ~]$ mkdir ~/work/arch-pc/lab09
[evmusatova@fedora ~]$ cd ~/work/arch-pc/lab09
[evmusatova@fedora lab09]$ touch lab09-1.asm
[evmusatova@fedora lab09]$
```

Рис. 2.1: Создание файлов и каталога

Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверяю его работу (рис. 2.2).

```
[evmusatova@fedora lab09]$ nasm -f elf lab09-1.asm
[evmusatova@fedora lab09]$ ld -m elf_i386 lab09-1.o -o lab09-1
[evmusatova@fedora lab09]$ ./lab09-1
Введите x: 5
2x+7=17
[evmusatova@fedora lab09]$
```

Рис. 2.2: Проверка программы

Добавляю подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\boxtimes(\boxtimes(\boxtimes))$, (рис. 2.3).

```

37 imul ebx
38
39 add eax, 7
40
41 pop eax
42 ret
43
44 _subcalcul:
45 push eax
46 mov ebx, 3
47 imul ebx
48 dec ebx
49 mov [res],eax
50
51 pop eax
52 ret

```

Рис. 2.3: Изменение программы

Создаю исполняемый файл и проверяю работу программы (рис. 2.4).

```

[ ~ - tab09 - orphan]
[evmusatova@fedora lab09]$ nasm -f elf lab09-1.asm
[evmusatova@fedora lab09]$ ld -m elf_i386 lab09-1.o -o lab09-1
[evmusatova@fedora lab09]$ ./lab09-1
Введите x: 1
f(g(x))=3
[evmusatova@fedora lab09]$

```

Рис. 2.4: Проверка

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. Затем получаю исполняемый файл и загружаю его в отладчик gdb (рис. 2.5).

```

Hello, world!
[evmusatova@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[evmusatova@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[evmusatova@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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. 2.5: Загрузка в отладчик

Запускаю программу в оболочке GDB с помощью команды run (рис. 2.6).

```

<https://www.gnu.org/software/gdb/bugs/>.
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) run
Starting program: /home/evmusatova/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
    <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 22636) exited normally]
(gdb)

```

Рис. 2.6: Запуск программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 2.7).


```

[Enter] 1 (process 22000) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/evmusatova/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.7: Установка брейкпоинта

Смотрю дисассимилированный код программы с помощью команды `disassemble` (рис. 2.8).

```

9      mov eax, 4
(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. 2.8: Просмотр кода

Переключаюсь на отображение команд с Intel'овским синтаксисом (рис. 2.9). В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так записываются адреса вторых аргументов.

```

(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. 2.9: Переключение

Включаю режим псевдографики (рис. 2.10).

```

[ Register Values Unavailable ]

B+> 0x08049000 <_start>  mov     eax,0x4
    0x08049005 <_start+5>  mov     ebx,0x1
    0x0804900a <_start+10> mov     ecx,0x804a000
    0x0804900f <_start+15> mov     edx,0x8
    0x08049014 <_start+20> int      0x80
    0x08049016 <_start+22> mov     eax,0x4

native process 22753 In: _start          L9      PC: 0x08049000
(gdb) layout regs
(gdb)

```

Рис. 2.10: Псевдографика

Проверяю что была установлена точка основа. Затем устанавливаю еще одну точку основа по адресу инструкции (рис. 2.11).

```

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

native process 23828 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 2.11: Проверка

Смотрю информацию о всех установленных точках основа (рис. 2.12).

```

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. 2.12: проверка

Выполняю 5 инструкций (рис. 2.13).

```
evmusatova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0

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 23828 In: _start L14 PC: 0x8049016
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) si
(gdb) si
(gdb) si 3
(gdb) 
```

Рис. 2.13: Инструкции

Смотрю значение переменной по имени (рис. 2.14).

```
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 23828 In: _start L14 PC: 0x8049016
ds      0x2b      43
es      0x2b      43
--Type <RET> for more, q to quit, c to continue without paging-- 0
x0      0
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) 
```

Рис. 2.14: Просмотр

Теперь смотрю значение переменной по адресу (рис. 2.15).

```

msg1 has unknown type, cast to char
(gdb) x/1sb 0x804a000
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 2.15: Другой способ посмотреть значение переменной

Изменяю первый символ переменной msg1 (рис. 2.16).

```

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

```

Рис. 2.16: Изменение символа

Теперь заменяю символ во второй переменной (рис. 2.17).

```

No symbol table loaded for this file in current context
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "\torld!\n\034"
(gdb)

```

Рис. 2.17: Замена символа в другой переменной

С помощью команды set изменяю значение регистра ebx (рис. 2.18). Однако при попытке задать строчное значение, происходит ошибка.

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$1 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$2 = 2  
(gdb) 
```

Рис. 2.18: Изменение значения регистра

Завершаю выполнение программы (рис. 2.19).

```
(gdb) c  
Continuing.  
orld!  
  
Breakpoint 2, _start () at lab09-2.asm:20  
(gdb)
```

Рис. 2.19: Завершение

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, создаю исполняемый файл и загружаю его в отладчик, указав аргументы (рис. 2.20).

```
[evmusatova@fedora lab09]$ gdb --args lab09-3 2 3 4
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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. 2.20: Загрузка в отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим её (рис. 2.21).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/evmusatova/work/arch-pc/lab09/lab09-3 2 3 4

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n])
```

Рис. 2.21: Точка основа и запуск

Смотрим количество аргументов (рис. 2.22).

```
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd190: 0x00000004
(gdb) █
```

Рис. 2.22: Проверка

Смотрю остальные позиции стека (рис. 2.23). Их адреса располагаются в 4 байтах друг от друга(именно столько занимает элемент стека).

```
(gdb) x/x $esp
0xffffd190:    0x00000004
(gdb) x/s *(void**)( $esp + 4)
0xffffd34e:    "/home/evmusatova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd37a:    "2"
(gdb) x/s *(void**)( $esp + 12)
0xffffd37c:    "3"
(gdb) x/s *(void**)( $esp + 16)
0xffffd37e:    "4"
(gdb) x/s *(void**)( $esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)( $esp + 24)
0xffffd380:    "SHELL=/bin/bash"
(gdb) 
```

Рис. 2.23: Просмотр

3 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.