

Лабораторная работа №9

Архитектура компьютера

Овчинников Антон

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
4	Задания для самостоятельной работы	14
5	Выводы	16

Список иллюстраций

3.1	Создание файла	7
3.2	Текст файла	8
3.3	Вывод программы	8
3.4	Работа программы	9
3.5	Запускаем файл в gdb	9
3.6	breakpoint	9
3.7	Содержимое регистров	10
3.8	breakpoint	10
3.9	Установка точки останова	10
3.10	Информация о точках останова	11
3.11	Вывод по имени	11
3.12	Вывод по имени	11
3.13	Код программы	11
3.14	Код программы	12
3.15	Значение регистра	12
3.16	Запуск программы	12
3.17	Адреса аргументов	13
4.1	Текст листинга	14
4.2	Результат программы	15

Список таблиц

1 Цель работы

Приобрести навыки 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, отрабатывает, 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). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

3 Выполнение лабораторной работы

Создаю рабочую директорию и файл. (рис. 3.1).

```
agovchinnikov@dk8n69 ~ $ mkdir ~/work/arch-pc/lab09
agovchinnikov@dk8n69 ~ $ cd ~/work/arch-pc/lab09
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-1.asm
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание файла

Записываю туда программу из листинга, при этом исправив опечатки. (рис. 3.2).

```

lab09-1.asm [----] 3 L:[ 26+26 52/ 52] *
mov eax, result
call sprint
mov eax,[res]
call iprintLF

call quit

_calcul:
push ebx
mov ebx,2
mul ebx

add eax, 7

pop ebx
ret

_subcalcul:
push ebx
mov ebx, 3
mul ebx
dec ebx
mov [res],eax

pop ebx
ret

```

Рис. 3.2: Текст файла

Проверим работу файла (рис. 3.3).

```

agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
f(g(x))=3
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 3.3: Вывод программы

Создаю файл lab9-2.asm и проассемблируем его с другими ключами, чтобы была возможность открыть этот файл через gdb. (рис. 3.4).


```

agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-2.lst lab9-2.asm
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ ./lab9-2
Hello, world!
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ █

```

Рис. 3.4: Работа программы

Открываю файл lab9-2 с помощью gdb (рис. 3.5).

```

agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ gdb lab9-2
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/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) █

```

Рис. 3.5: Запускаем файл в gdb

Поставим точку остановки на метке _start (рис. 3.6).

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/g/agovchinnikov/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6089) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/g/agovchinnikov/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb) █

```

Рис. 3.6: breakpoint

В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так записываются адреса вторых аргументов. Включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров (рис. 3.7).

```

[ 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

native process 6166 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 3.7: Содержимое регистров

Поставим точку остановки на метке `_start` (рис. 3.8).

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/g/agovchinnikov/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6089) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/g/agovchinnikov/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 3.8: breakpoint

Установим еще одну точку останова (рис. 3.9).

```

(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab9-2.asm:9
          breakpoint already hit 1 time
(gdb)

```

Рис. 3.9: Установка точки останова

Посмотрим информацию о наших точках останова. Сделать это можно коротко

командой `i b` (рис. 3.10).

```
(gdb) b *0x8049014
Breakpoint 2 at 0x8049014: file lab9-2.asm, line 13.
(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   0x08049014 lab9-2.asm:13
(gdb) █
```

Рис. 3.10: Информация о точках останова

В отладчике можно вывести текущее значение переменных(рис. 3.12).

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

Рис. 3.11: Вывод по имени

В отладчике можно вывести текущее значение переменных(рис. 3.12).

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

Рис. 3.12: Вывод по имени

Также можно обращаться по адресам переменных. Здесь был заменен первый символ переменной `msg2` на символ отступа(рис. 3.14).

```
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "\torld!\n\034"
(gdb) █
```

Рис. 3.13: Код программы

Также можно обращаться по адресам переменных. Здесь был заменен первый символ переменной `msg2` на символ отступа(рис. 3.14).

```
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "\torld!\n\034"
(gdb) █
```

Рис. 3.14: Код программы

Зададим регистру значения (рис. 3.15).

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

Рис. 3.15: Значение регистра

Скопируем файл из предыдущей лабораторной, переименуем и создадим исполняемый файл. Создадим точку останова на метке `_start` и запустим программу. (рис. 3.16).

```
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3
.asm
agovchinnikov@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ld -m elf_i386 -o lab09-3 lab09-3.o
agovchinnikov@dk8n69 ~/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. 3.16: Запуск программы

Посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга. (рис. 3.17)

```

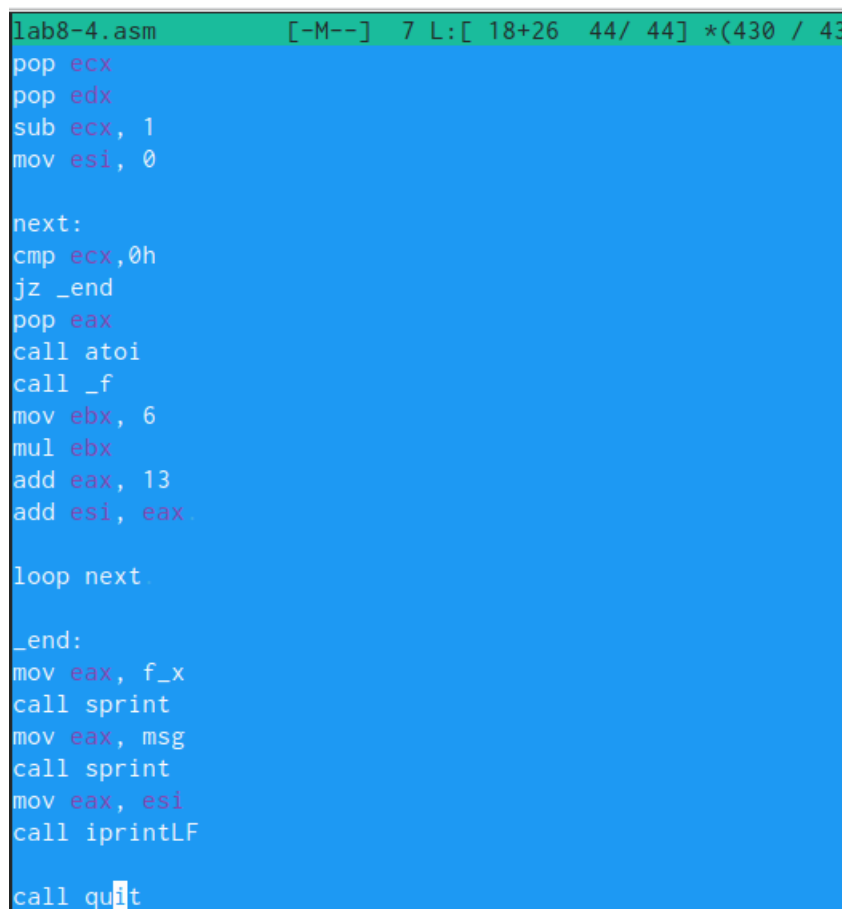
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc2a0: 0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xfffffc533: "/afs/.dk.sci.pfu.edu.ru/home/a/g/agovchinnikov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc57d: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc58f: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc5a0: "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffc5a2: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.17: Адреса аргументов

4 Задания для самостоятельной работы

Программа из лабораторной 9, но с использованием подпрограмм. (рис. 4.1)



```
lab8-4.asm      [-M--]  7 L:[ 18+26  44/ 44] *(430 / 43
pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _f
mov ebx, 6
mul ebx
add eax, 13
add esi, eax

loop next

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

Рис. 4.1: Текст листинга

Запускаю исполняемый файл. (рис. 4.2)

```
agovchinnikov@dk8n60 ~/work/arch-pc/lab09 $ ./lab8-4
f(x)=6x+13
результат: 0
agovchinnikov@dk8n60 ~/work/arch-pc/lab09 $ ./lab8-4 1 2 3
f(x)=6x+13
результат: 219
agovchinnikov@dk8n60 ~/work/arch-pc/lab09 $
```

Рис. 4.2: Результат программы

5 Выводы

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