

计算机体系结构 Lab03

熟悉 MARS

- 1. **.data, .word, .text 指令的含义是什么？（即：它们的用途是什么？）**
.data表示后续内容从data段的下一个可用地址开始存储。
.text表示后续内容从text段（指令段）的下一个可用地址开始存储。
.word表示后续内容占用一个字（32bit）的空间。
- 2. **如何在 MARS 中设置断点？在第 14 行设置断点并运行至此。指令的地址是什么？第 14 行是否执行？**

在运行窗口中，找到指定的行，勾选第一栏Bkpt即可添加断点。

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400008	0x3c011001	lui \$1, 4097	9: la \$t3, n
<input type="checkbox"/>	0x0040000c	0x342b0010	ori \$11, \$1, 16	
<input type="checkbox"/>	0x00400010	0x8d6b0000	lw \$11, 0(\$11)	10: lw \$t3, 0(\$t3)
<input type="checkbox"/>	0x00400014	0x11600006	beq \$11, \$0, 6	11: fib: beq \$t3, \$0, finish
<input type="checkbox"/>	0x00400018	0x01285020	add \$10, \$9, \$8	12: add \$t2, \$t1, \$t0
<input type="checkbox"/>	0x0040001c	0x00094021	addu \$8, \$0, \$9	13: move \$t0, \$t1
<input checked="" type="checkbox"/>	0x00400020	0x000a4821	addu \$9, \$0, \$10	14: move \$t1, \$t2
<input type="checkbox"/>	0x00400024	0x20010001	addi \$1, \$0, 1	15: subi \$t3, \$t3, 1
<input type="checkbox"/>	0x00400028	0x01615822	sub \$11, \$11, \$1	
<input type="checkbox"/>	0x0040002c	0x08100005	j 0x00400014	16: j fib
<input type="checkbox"/>	0x00400030	0x21040000	addi \$4, \$8, 0	17: finish: addi \$a0, \$t0, 0
<input type="checkbox"/>	0x00400034	0x24020001	addiu \$2, \$0, 1	18: li \$v0, 1 # you will be asked about what the
<input type="checkbox"/>	0x00400038	0x00000000	syscall	19: syscall

指令地址为0x00400020，该指令未被执行。

- 3. **如果在断点处，如何继续运行你的代码？如何单步调试你的代码？将代码运行至结束。**
Run→Go（或按键 F5）可以继续执行代码。使用Run→Step（或按键 F7）可以单步执行代码。
- 4. **找到“Run I/O”窗口.程序输出的数字是什么？如果 0 是第 0 个斐波那契数，那么这是第几个斐波那契数？**

程序输出如下图所示：

Mars Messages		Run I/O
		34
		-- program is finished running --
Clear		

这是第9个斐波拉契数。

5. 在内存中，`n` 存储在哪个地址？尝试通过（1）查看 **Data Segment**，以及（2）查看机器代码（**Text Segment** 中的 **Code** 列）理解，如何从存储器中读取 `n`。

`n` 存储的地址为 `0x10010010`。先通过 `lui` 在 `$at` 中写入地址的高16位，再通过 `ori` 将低16位写入 `$t3`，然后通过 `lw` 将 `$t3` 中存储的内存地址所对应的值写入 `$t3`。

6. 如何在不改变“Edit”栏下的代码的条件下，通过在执行前手动修改存储位置的值，让程序计算第 13 个斐波那契数（索引从 0 开始）？你可以取消勾选 **Data Segment** 底部的“Hexadecimal Values”框方便观察。

在 **Data Segment** 窗口中，找到地址 `0x10010010`，双击将其值从 9 修改为 13。

Data Segment						
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+12)
0x10010000	2	4	6	8	13	10
0x10010020	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0

7. 如何观察和修改一个寄存器中的值？重置模拟（**Run**→**Reset**）并通过（1）在一个设置好的断点停下，（2）只修改一个寄存器，（3）解除断点，来计算第 13 个斐波那契数。

在第 11 行设置断点，程序运行到断点后，将寄存器 `$t3` 的值从 9 修改为 13。

\$a0	7	0
\$t0	8	0
\$t1	9	1
\$t2	10	0
\$t3	11	13
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0

计算结果为 233。

8. 第 19 行和第 21 行用到了 `syscall` 指令。它是什么？如何使用它？（提示：可以查看 **MARS** 的 **Help** 菜单）

系统调用指令，用于调用系统服务。第 19 行调用 1 号系统调用输出一个整型变量，第 21 行调用 10 号系统调用退出程序。

首先将系统调用编号写入 `$v0` 寄存器，然后按照系统调用函数的需求将参数写入 `$a0`，`$a1`，`$a2`，或 `$f12` 寄存器，执行 `syscall` 指令，可以在指定的寄存器读取返回值。

将 C 编译为 MIPS

1. 在生成的 MIPS 汇编代码 `lab3_ex2.s` 中找到将 `source` 复制到 `dest` 的循环部分所对应的指令。

循环部分指令如下：

```

1  $L3:
2      sw      $4,0($3)
3      lw      $4,0($2)
4      addiu   $3,$3,4
5      addiu   $2,$2,4
6      bne     $4,$0,$L3

```

2. 找到 lab_ex2.c 中的 source 和 dest 指针最初在汇编文件中存储的位置。最后，解释这些指针是如何通过循环进行操作的。

dest在汇编文件的第49行，source在第55行。

每个循环使用addiu指令对两个指针加4，使其指向的地址每次向后移动一个字。

函数调用的过程

prologue处代码：

```

1  subi    $sp,$sp,16
2  sw      $ra,0($sp)
3  sw      $s0,4($sp)
4  sw      $s1,8($sp)
5  sw      $s2,12($sp)

```

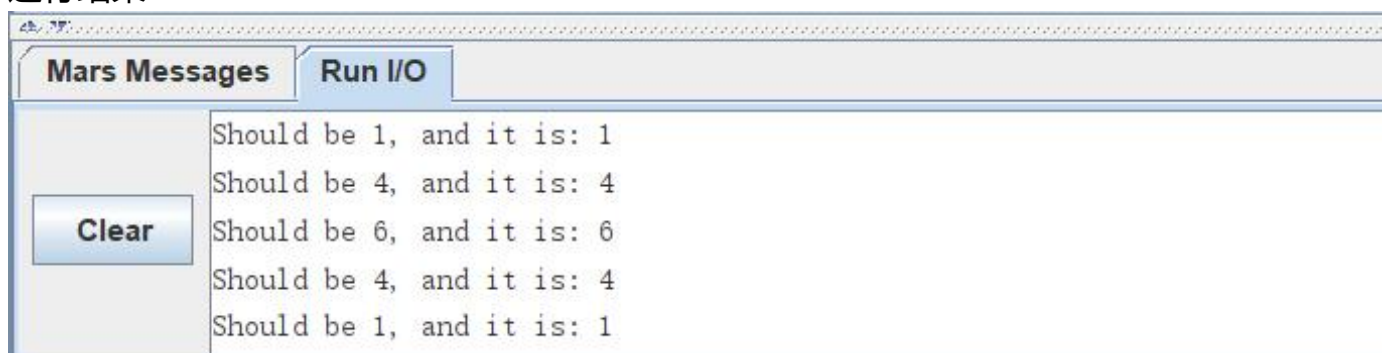
epilogue处代码：

```

1  lw      $ra,0($sp)
2  lw      $s0,4($sp)
3  lw      $s1,8($sp)
4  lw      $s2,12($sp)
5  addi    $sp,$sp,16
6  jr      $ra

```

运行结果：



The screenshot shows the Mars Messages window with the following output:

```

Should be 1, and it is: 1
Should be 4, and it is: 4
Should be 6, and it is: 6
Should be 4, and it is: 4
Should be 1, and it is: 1

```