

实验 3

李子龙 518070910095

2021 年 3 月 19 日

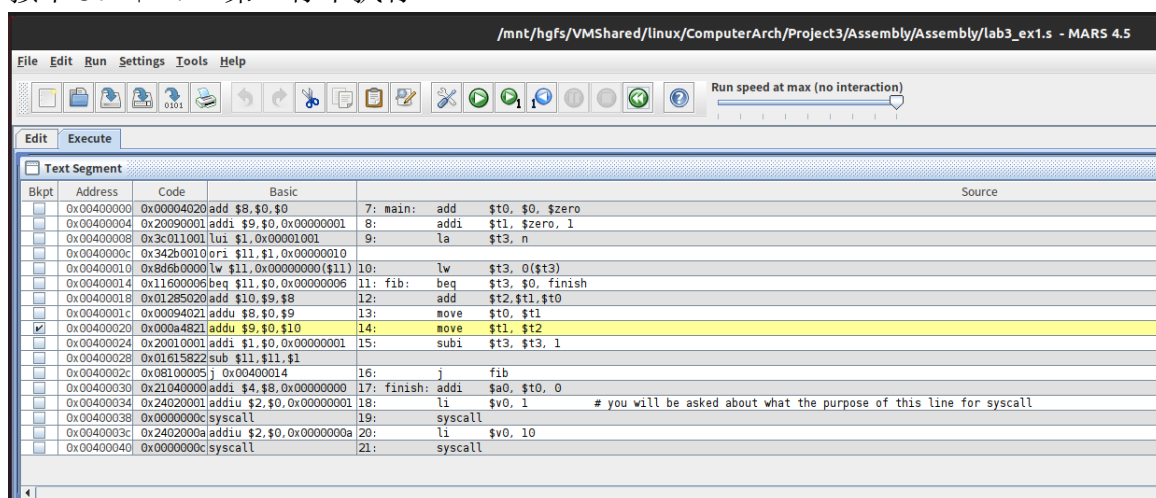
一. 熟悉 MARS

- (1) `.data` `.word` `.text` 指令的含义是什么？（即：它们的用途是什么？）

答: `.data`: 标识下一个连续区块的命令都将被存储在数据段; `.word`: 列出的操作数将会被存储在数据字段为32位字; `.text`: 标识下一个连续区块的命令都将被存储在文本段。

- (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？

答: 在“Execute”菜单中，在14行对应的“Bkpt”打勾（如果被禁用了断点需要先按下Ctrl+T）。第14行不执行。

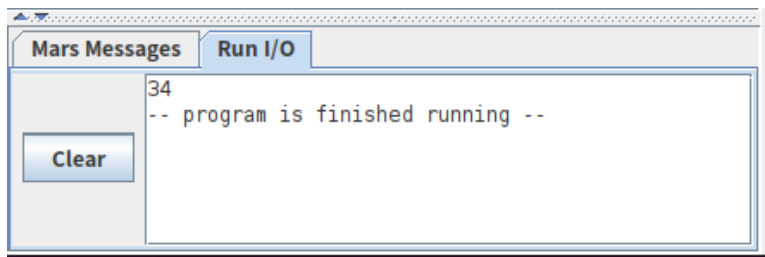


- (3) 如果在断点处，如何继续运行你的代码？如何单步调试你的代码？将代码运行至结束。

答: 继续运行按下工具栏的绿色箭头，单步调试按下含有1的绿色箭头。

- (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？

答: 34。第 9 个斐波那契数。



- (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`。

答: `n` 存储在 `0x10010010` 地址。

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000002	0x00000004	0x00000006	0x00000008	0x00000009	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

所谓的

```
la $t3, n
```

伪指令被翻译为 MIPS 指令

```
lui $l, 0x00001001
ori $l1, $l, 0x00000010
```

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00004020	add \$8,\$0,\$0	7: main: add \$t0,\$0,\$zero
	0x00400004	0x20090001	addi \$9,\$0,0x00000001	8: addi \$t1,\$zero,1
	0x00400008	0x3c011001	lui \$l,0x00001001	9: la \$t3,n
	0x0040000c	0x342b0010	ori \$l1,\$l,0x00000010	
	0x00400010	0x8d6b0000	lw \$l1,0x00000000(\$l1)	10: lw \$t3,0(\$t3)
	0x00400014	0x11600006	beq \$l1,\$0,0x00000006	11: fib: beq \$t3,\$0,finish
	0x00400018	0x01285020	add \$l0,\$9,\$8	12: add \$t2,\$t1,\$t0
	0x0040001c	0x00094021	addu \$8,\$0,\$9	13: move \$t0,\$t1
	0x00400020	0x000a4821	addu \$9,\$0,\$10	14: move \$t1,\$t2
	0x00400024	0x20010001	addi \$l,\$0,0x00000001	15: subi \$t3,\$t3,1
	0x00400028	0x01615822	sub \$l1,\$l1,\$l	
	0x0040002c	0x08100005	j 0x00400014	16: j fib
	0x00400030	0x21040000	addi \$4,\$8,0x00000000	17: finish: addi \$a0,\$t0,0
	0x00400034	0x24020001	addiu \$2,\$0,0x00000001	18: li \$v0,1 # you will be asked about w...
	0x00400038	0x0000000c	syscall	19: syscall
	0x0040003c	0x2402000a	addiu \$2,\$0,0x0000000a	20: li \$v0,10
	0x00400040	0x0000000c	syscall	21: syscall

第一条指令将 `0x00001001` 地址送至寄存器高位 `$at` 的前16位中；第二条指令将上面的地址与 `0x00000010` 做或运算存放到临时变量寄存器 `$t3` 中，这样就将 `n` 的地址取出放在了 `$t3`。

Registers		
Coprocc 1 Coprocc 0		
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x10010010
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400010
hi		0x00000000
lo		0x00000000

- (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。

答：直接将代码区的对应的内存数值修改即可，得到233：

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	2	4	6	8	13	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0
0x10010180	0	0	0	0	0	0	0	0
0x100101a0	0	0	0	0	0	0	0	0
0x100101c0	0	0	0	0	0	0	0	0

Mars Messages

Run I/O

Reset: reset completed.

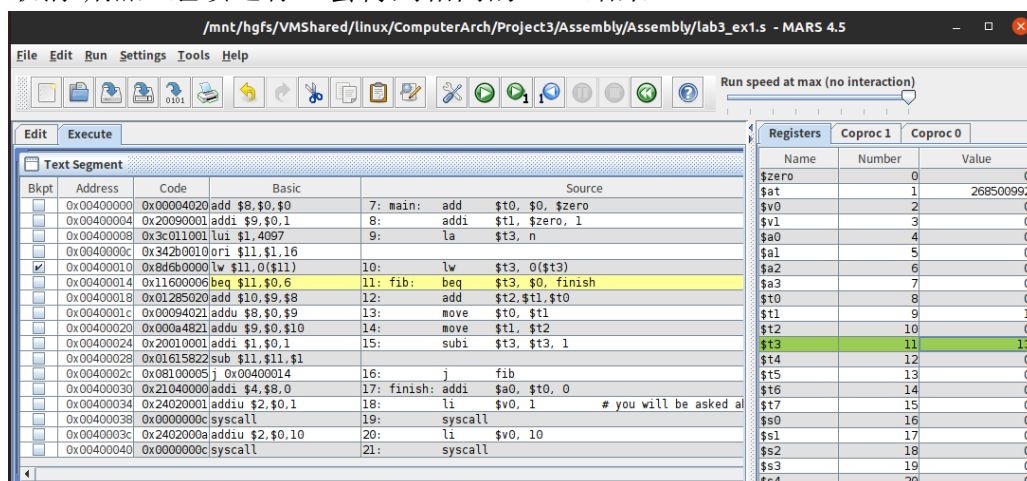
233

-- program is finished running --

Clear

- (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。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）

答: 在第 10 行处设置断点, 当该行单步运行结束后, 修改寄存器 `$t3` 的值为 13, 取消断点, 继续运行。会得到相同的 233 结果。



第19行时, 寄存器`$v0`的值被设置为1, 对应的`syscall`的模式为打印寄存器`$a0`的整数: 34. 第21行时, 寄存器`$v0`的值已经被设置为10, 对应的`syscall`的模式为退出。

二. 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。

循环部分对应的指令:

```

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
7      nop

```

- (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。

```

lui $3,%hi(dest)
lui $2,%hi(source+4)
addiu $3,$3,%lo(dest)
addiu $2,$2,%lo(source+4)

```

`source` 指针被存放在 `$2` 中 (+4是为了对齐), `dest` 指针被存放在 `$3` 中。

首先会将 `dest` 指针的地址从 `$3` 取出存放在 `$4` 中, 之后会将 `$4` 地址所对应的值设置为 `source` 对应的 `$2` 地址指向的值。之后两个指针地址寄存器+4, 准备读取

下一个数。现在 \$4 已经是之前的 dest 和 source 对应的值，如果它不等于 0 就会循环，否则会到下面的 \$L2 标签区域。

三. 函数调用的过程

需要保存 \$s0 \$s1 \$s2 \$ra 寄存器变量到栈帧中。

```
1 nchoosek:
2     # prologue
3     ### YOUR CODE HERE ##
4     addi      $sp, $sp, -16
5     sw        $ra, 12($sp)
6     sw        $s2, 8($sp)
7     sw        $s1, 4($sp)
8     sw        $s0, 0($sp)
```

```
1 return:
2     # epilogue
3     ### YOUR CODE HERE ###
4     lw        $s0, 0($sp)
5     lw        $s1, 4($sp)
6     lw        $s2, 8($sp)
7     lw        $ra, 12($sp)
8     addi      $sp, $sp, 16
9     jr        $ra
```

