实践3 实验报告

陈文迪 519021910071

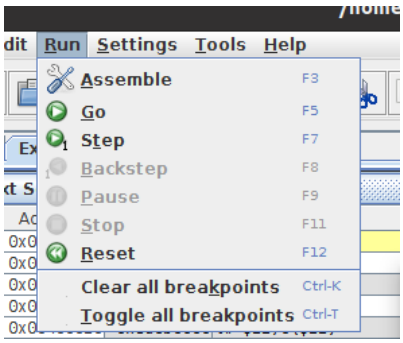
任务1 熟悉MARS

问题回答：

1. `.data` 标记了存储在Data segment中下一个可用地址处的数据项。`.word` 可以将所列值存储在一个32位字长的内存空间中。 `.text` 标记了存储在Text segment中下一个可用地址处的指令。
2. 可以在 Execute 标签页的Text Segment窗口中，在指定行的左侧Bkpt复选框中打勾来设置断点。指令的地址是 `0x00400020` 。第14行没有执行。

0x00400018	0x01285020	add $10,$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 $1,$0,1	15:	subi	$t3, $t3, 1
0x00400028	0x01615822	sub $11,$11,$1			

3. 可以通过上方工具栏中的`Run->Go` 来继续运行代码。可以通过上方工具栏中的 `Run->Step` 来单步调试代码。



4. 窗口输出的数字是34，这是第9个斐波那契数。
5. 在内存中，n被存储在 `0x10010010` 中。从存储器中读取n的步骤如下：（1）将n的内存地址读入寄存器（在具体指令中拆分成 `lui` 与 `ori` 两句语句）（2）将寄存器指向的内存地址中的数据读入寄存器（ `lw` 语句）。

0x00400008	0x3c011001	lui $1,4097	9:	la	$t3, n
0x0040000c	0x342b0010	ori $11,$1,16			
0x00400010	0x8d6b0000	lw $11,0($11)	10:	lw	$t3, 0($t3)

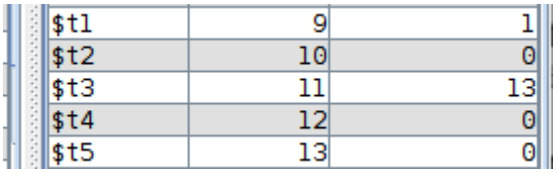
6. 可以利用断点，在n所在内存被访问之前，将 `0x10010010` 中的值改为13，即可计算第13个斐波那契数。最终结果为233。

Bkpt	Address	Code	Basic		Source
	0x00400000	0x00004020	add $8,$0,$0	7: main:	add $t0, $0, $zero
✔	0x00400004	0x20090001	addi $9,$0,1	8:	addi $t1, $zero, 1
	0x00400008	0x3c011001	lui $1,4097	9:	la $t3, n
	0x0040000c	0x342b0010	ori $11,$1,16		

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value
0x10010000	2	4	6	8	13	
0x10010020	0	0	0	0	0	

7. 类似于6的方法，我们将断点设置在n被读入寄存器之后，修改 `t3` 寄存器即可，如下图，最终结果为233。

0x0040000c	0x342b0010	ori $11,$1,16			
0x00400010	0x8d6b0000	lw $11,0($11)	10:	lw	$t3, 0($t3)
✔ 0x00400014	0x11600006	beq $11,$0,6	11: fib:	beq	$t3, $0, finish



$t1	9	1
$t2	10	0
$t3	11	13
$t4	12	0
$t5	13	0

8. `syscall` 是一种伪指令，可以利用输入输出设备来输出或改变某些结果寄存器的值。使用 `syscall` 有以下几个基本步骤：（1）将服务编号读入寄存器 `$v0` （2）将参数值读入 `$a0` 、`$a1` 、 `$a2` 或 `$f12` （3）调用 `syscall` 指令 （4）从结果寄存器中获取返回值。例如，在本题代码中，使用 `li  $v0, 1` 指定服务为打印整数，再使用 `addi  $a0, $t0, 0` 将需要打印的数读入参数寄存器 `$a0` ，最后19行调用伪指令 `syscall` 将 `$a0` 中的值打印出来。而20行的 `syscall` 则是利用 `li  $v0, 10` 调用了程序退出服务。

任务2 将C编译为MIPS

问题回答：

1. 下列代码将 `$2` 所指向内存中的内容持续复制到 `$3` 所指向的内存，并不断将 `$2` 和 `$3` 存储的内存地址增加4个字节，直到 `$2` 所指向的内存中的数据为0。这样就实现了将 source 复制到 dest 的循环。

```
$L3:
    sw  $4,0($3)
    lw  $4,0($2)
    addiu   $3,$3,4
    addiu   $2,$2,4
    bne $4,$0,$L3
    nop
```

2.

```
    .comm   dest,40,4
    .globl  source
    .data
    .align  2
    .type   source, @object
    .size   source, 28
source:
    .word   3
    .word   1
    .word   4
    .word   1
    .word   5
    .word   9
    .word   0
    .ident  "GCC: (GNU) 9.1.0"
```

从上述代码中我们可以看到，dest指针和source指针的初始化过程，它们分别指向了一块大小为40个字节和大小为28个字节的连续内存区域，也正是两数组首元素的地址。这样的话我们可以先将二者存储到两个寄存器 `$3` 和 `$2` 中，每次循环后将两个寄存器的值增加4，这样寄存器中的内存就指向了数组的下一个元素，每次借用 `$4` 寄存器读取和修改对应内存地址的数据，执行复制操作，直到 `$2` 所指向的内存中的数据为0。

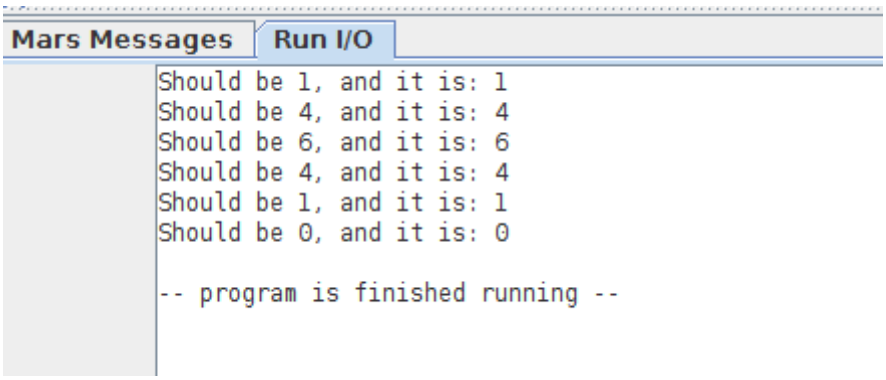
任务3 函数调用的过程

```
# calculate C($a0,$a1)
nchoosek:
    # prologue
    ### YOUR CODE HERE ###
    addi $sp, $sp, -16
    sw $ra, 12($sp)
    sw $s2, 8($sp)
    sw $s1, 4($sp)
    sw $s0, 0($sp)

    beq $a1, $0, return1
    beq $a0, $a1, return1
    beq $a0, $0, return0
    blt $a0, $a1, return0

    addi    $a0, $a0, -1        # C(n,k) = C(n-1,k) + C(n-1,k-1)
    move    $s0, $a0
    move    $s1, $a1
    jal nchoosek
    move    $s2, $v0
    move    $a0, $s0
    addi    $a1, $s1, -1
    jal nchoosek
    add $v0, $v0, $s2
    j   return
return0:
    move    $v0, $0
    j   return
return1:
    addi    $v0, $0, 1

return:
    # epilogue
    ### YOUR CODE HERE ###
    lw $ra, 12($sp)
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $s0, 0($sp)
    addi $sp, $sp, 16
    jr $ra
```



本题是一个函数递归调用的过程，需要把有可能在递归调用中被修改的寄存器通过栈的形式保存下来，并在返回时恢复。