《计算机系统》

原型机

实验报告

目录

[1 实验项目一 3](#_Toc115379316)

[1.1 项目名称 3](#_Toc115379317)

[1.2 实验目的 3](#_Toc115379318)

[1.3 实验资源 3](#_Toc115379319)

[2 实验任务 4](#_Toc115379320)

[2.1 实验任务A 4](#_Toc115379321)

[2.2 实验任务B 6](#_Toc115379322)

[2.3 实验任务C 12](#_Toc115379323)

[3 总结 14](#_Toc115379324)

[3.1 实验中出现的问题 14](#_Toc115379325)

[3.2 心得体会 14](#_Toc115379326)

# 实验项目一

## 项目名称

最小系统与原型机

## 实验目的

1. 了解冯诺伊曼体系结构；
2. 理解指令集结构及其作用；
3. 理解计算机的运行过程，就是指令的执行过程，并初步掌握调试方法。

## 实验资源

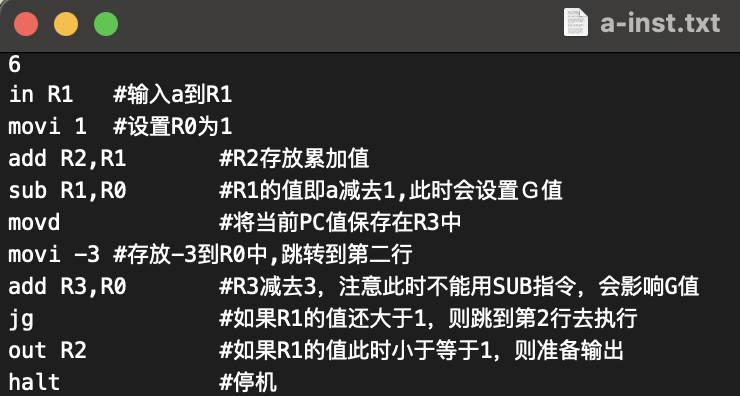
1. vspm1.0
2. a-inst.txt等代码文件

# **实验任务**

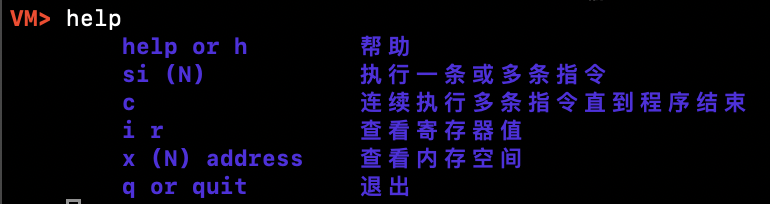
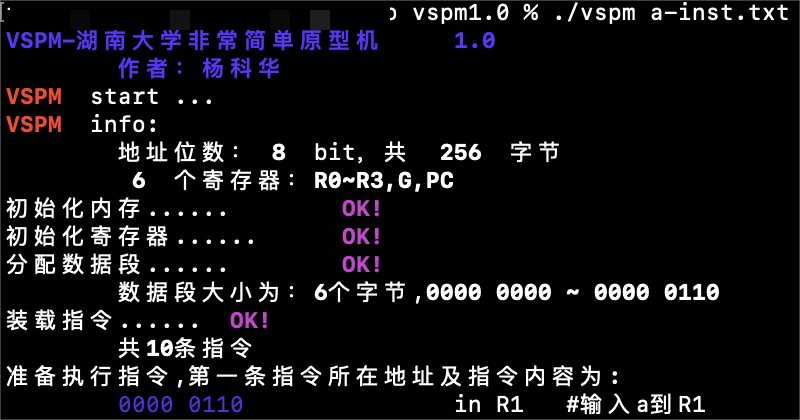
## **实验任务A**

**任务名称：用提供的vspm调试a-inst.txt代码文件，并对代码所做的工作进行解释**

1. 进入终端，使用cd vspm1.0进入目录； 使用./vspm a-inst.txt来运行原型机1.0的模拟器，其中a-inst.txt为代码文件:



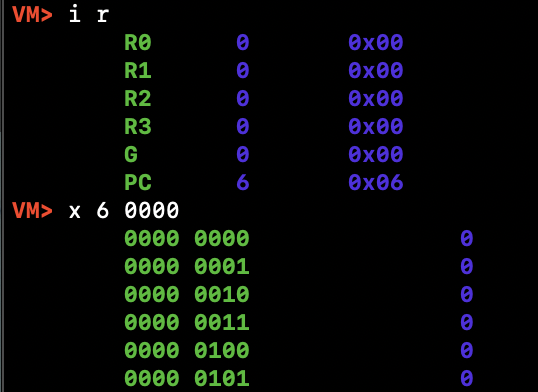
1. 运行后界面如下图，并输入help获取帮助了解vspm指令使用方法



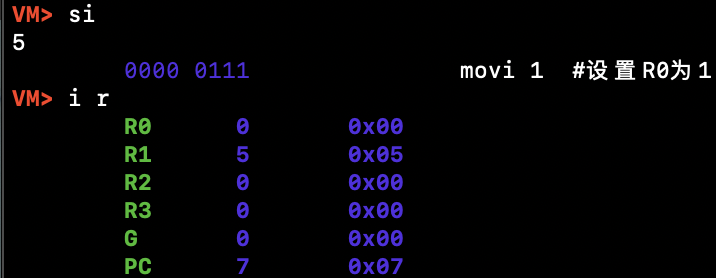
1. 在运行后，提示将要执行的指令地址及内容，在本例中，提示要执行位于内存00000110处的指令“in R1”，即等待输入一个整数值。此时输入si则表示执行此指令，同时也可以输入其他的指令：

输入ir查看寄存器的值，每个寄存器的值都用十进制和十六进制表示

输入x 6 0000查看从0000开始的连续6个内存地址的值：（初始值均为0）



1. 此时输入si指令，表示运行当前指令，例如此时运行的指令是“in R1”，表示等待输入一个整数值，输入整数值5，然后输入ir查看寄存器内容，可观察到输入值5确实已存在R1寄存器中



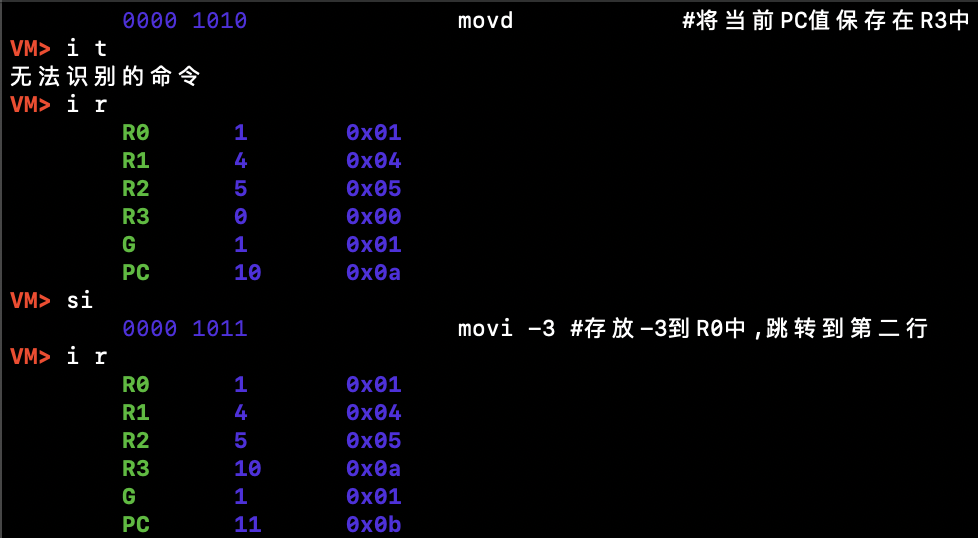
1. 运行指令“movi 1”和“add R2, R1”后输入i r查看寄存器，可观察到R0寄存器的值已置为1，且R2存放当前累加值5



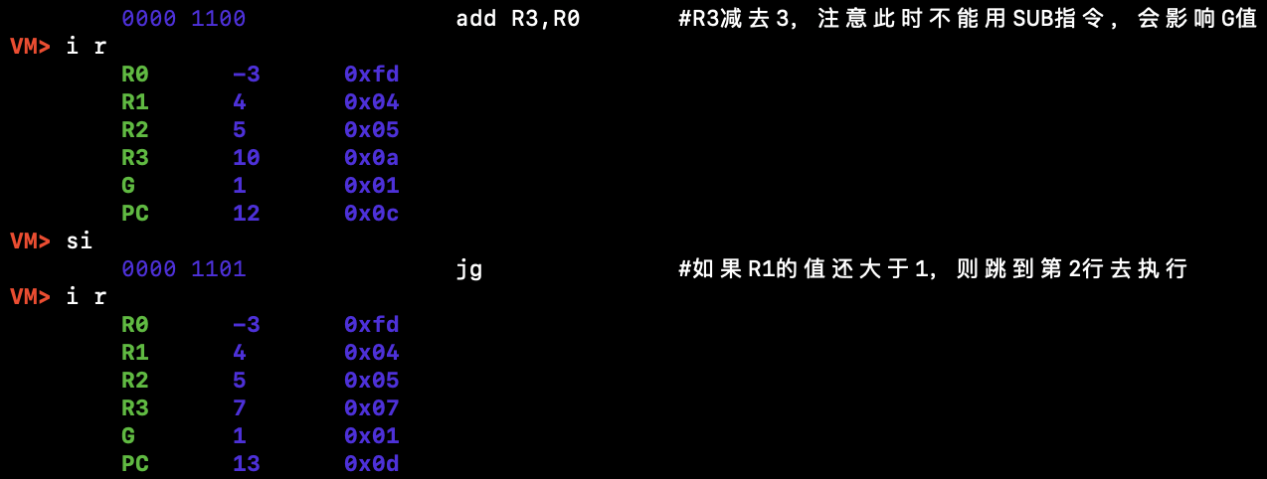
1. 运行“sub R1, R0”指令后查看寄存器，可观察到R1值已减1，因为R1 > R0 所以G值设置为1



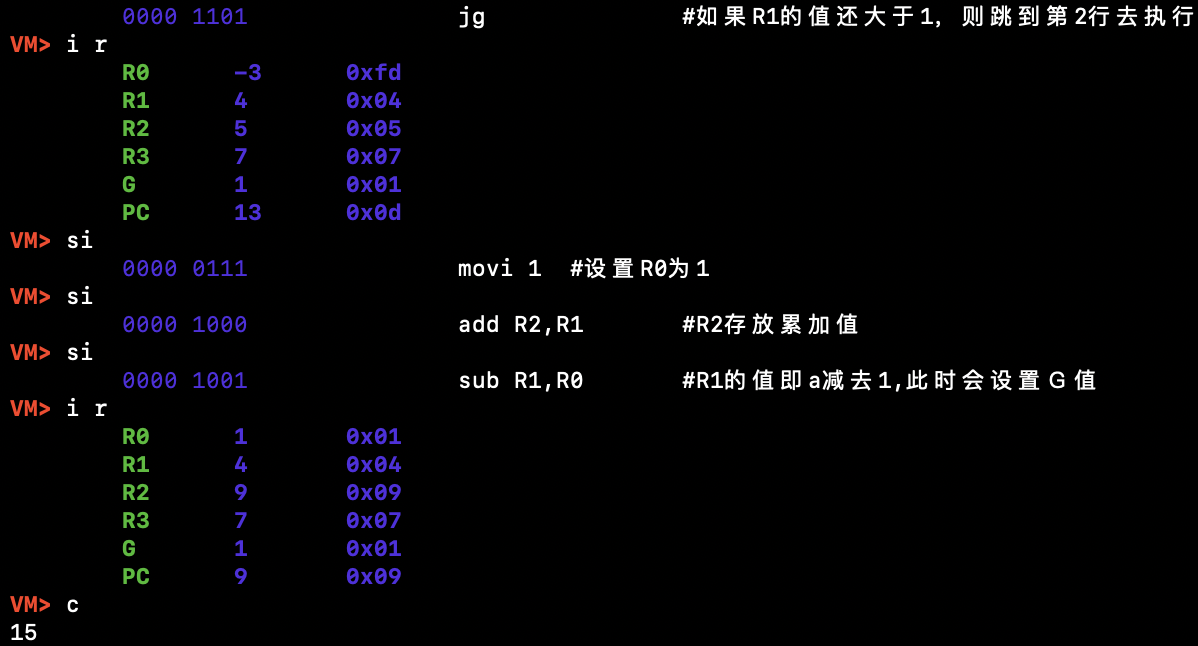
1. 运行“movd”指令后查看寄存器的值，可观察到当前PC值10(0000 1010)已保存在R3中



1. 运行“movi -3”和“add R3, R0”后输入i r查看寄存器的值，可观察到R0中保存-3，R3与R0相加后为7，此步是为了在在运行jg指令后判断跳转到代码第二行继续累加，且此时不能用sub指令，因为会影响G值



1. 运行“jg”后可观察到程序跳转到第2行代码重复累加，最后输入c指令让模型机连续执行指令直到程序结束，可观察到最后输出结果为15（5+4+3+2+1）。



## **实验任务B**

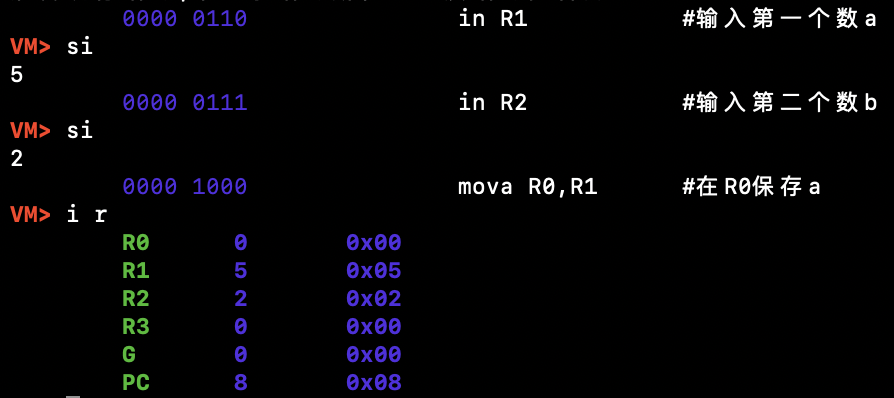
**任务名称：用提供的vspm调试b-inst.txt和c-inst.txt代码文件，并对代码所做的工作进行解释**

### **运行调试b-inst.txt**

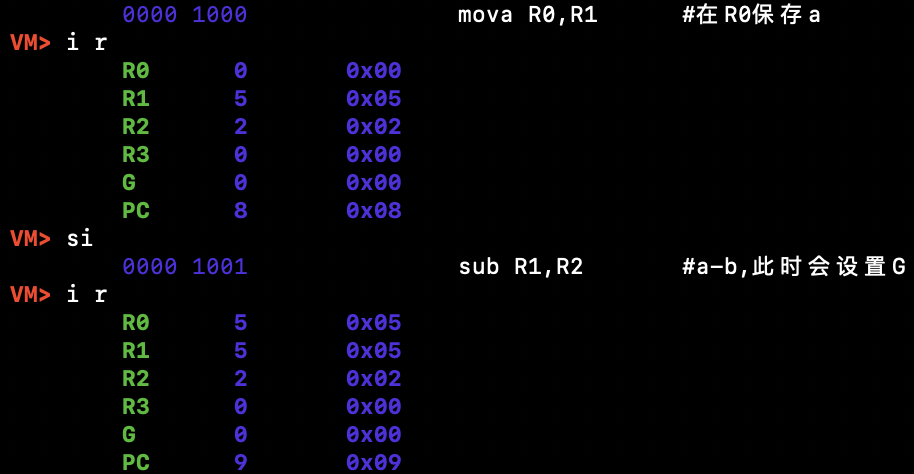
1. 使用./vspm b-inst.txt来运行原型机1.0的模拟器，其中b-inst.txt为代码文件:

********

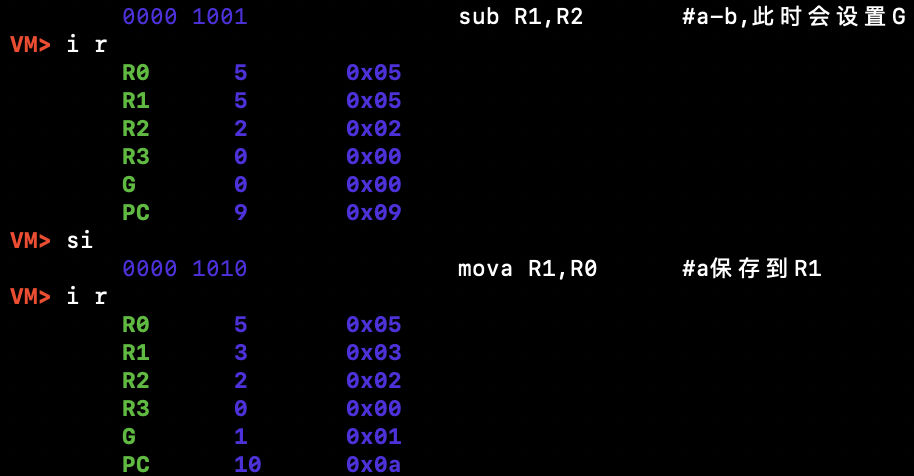
1. **运行“in R1”和“in R2”后程序都会**等待输入一个整数值，分别输入整数值5，2，然后**输入i r查看寄存器的值，可观察到5、2都已分别存入R1和R2寄存器**

****

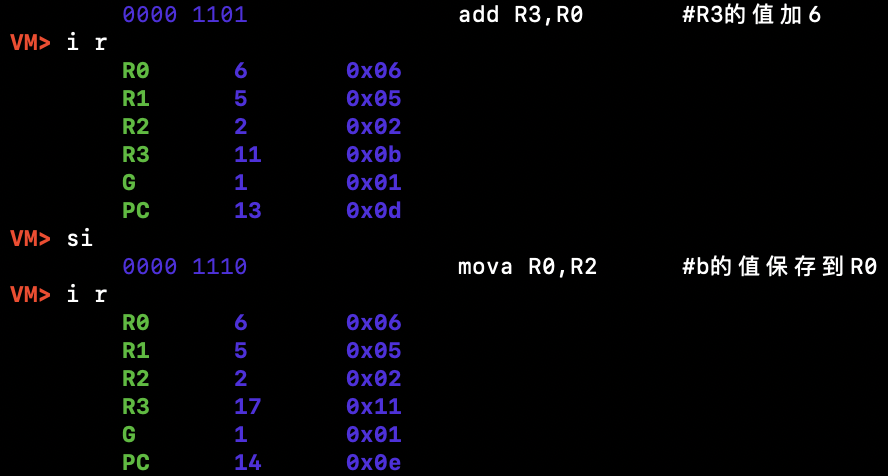
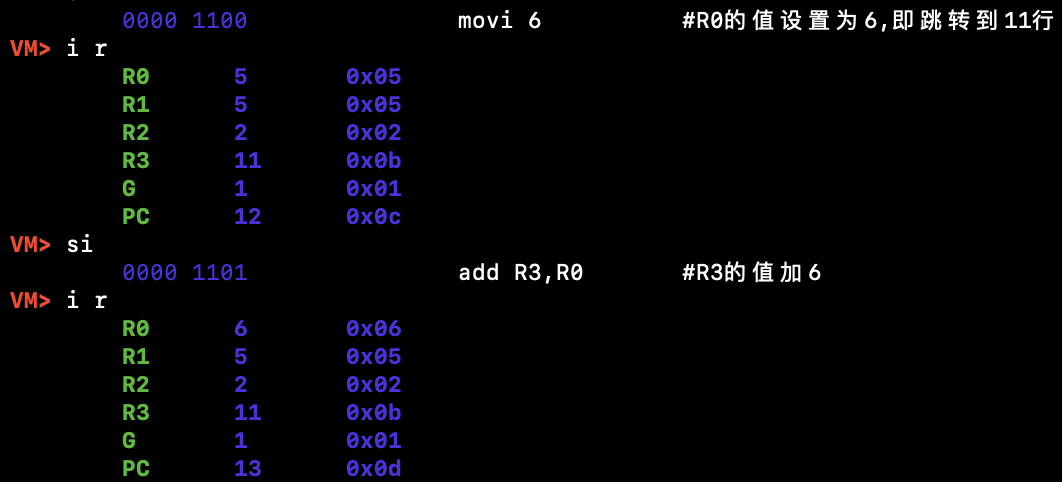
1. **运行“mova R0，R1”后输入i r查看寄存器的值，可观察到R1已把整数5保存在R0中**

****

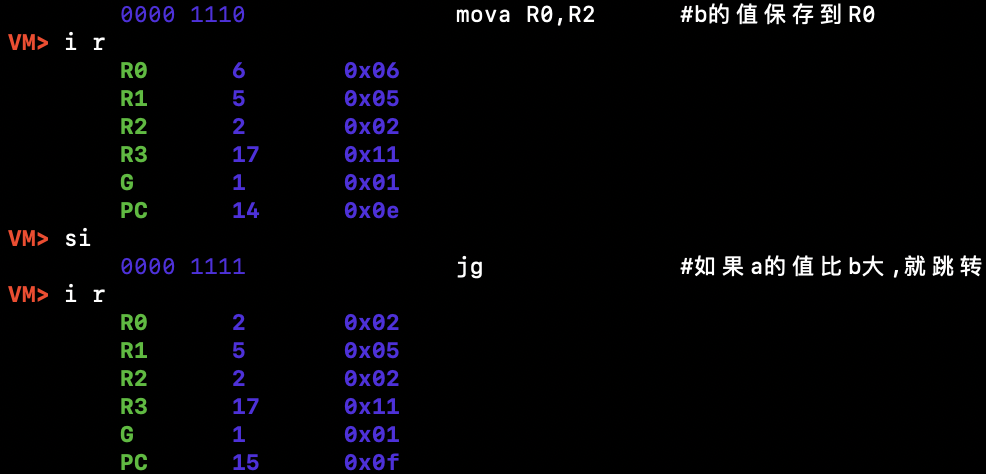
1. **运行“sub R1,R2”指令后查看寄存器的值，可观察到R1-R2的结果3已保存在R1中，且因为R1>R2，G的值设置为1**

****

1. **运行“movd”、“movi 6”和“add R3，R0”指令后观察积存的值，可观察到“movd”指令PC值11（00001011）保存到R3中，后面两个指令将R3的值加6目的是后续让程序跳转到代码第11行运行**

****

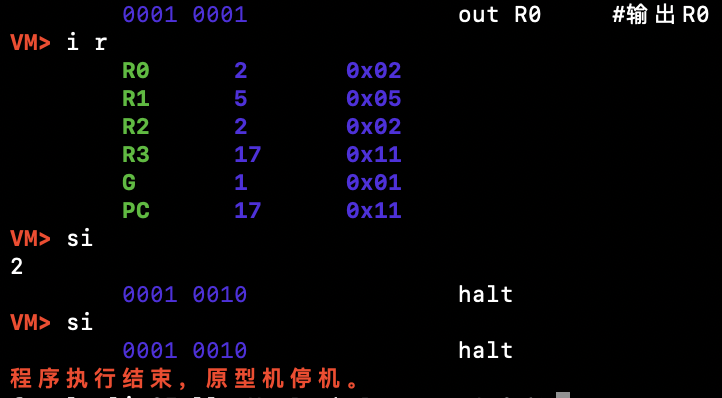
1. **运行“mova R0,R2”指令后输入i r观察寄存器的值，可观察到R2的值被保存到R0中**

****

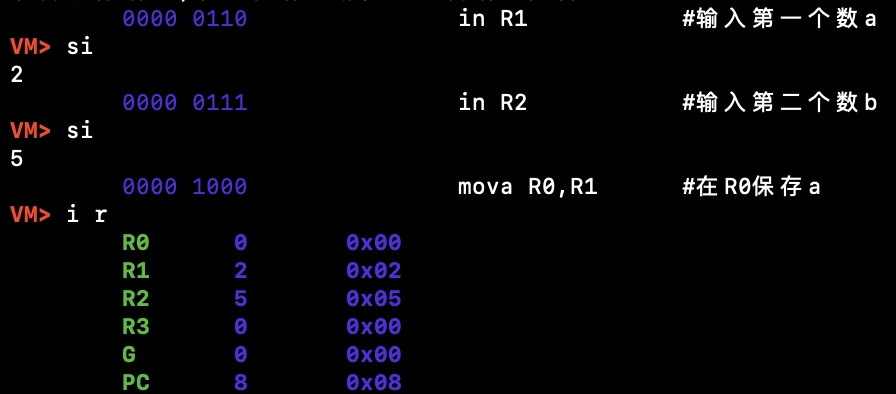
1. **因为G的值为1，满足跳转条件，运行“jg”指令后，可观察到程序直接跳转到代码第11行（即输出指令）运行**

****

1. **运行“out R0”指令后可观察到R0的值2（b的值）被输出，后续执行“halt”停机指令，模型机停机**

****

1. **如果输入的a的值比b小（R1<R2），则在执行“sub R1, R2”指令时G值不会设置为1，在运行“jg”指令时不满足跳转条件，程序继续按顺序运行**

****

****

1. **程序继续执行下一条指令“mova R0，R1”后输入i r观察寄存器的值，可观察到R1的值2保存到R0中，后续执行“out R0”指令输出的结果为2（a的值），最后执行停机指令原型机停机**

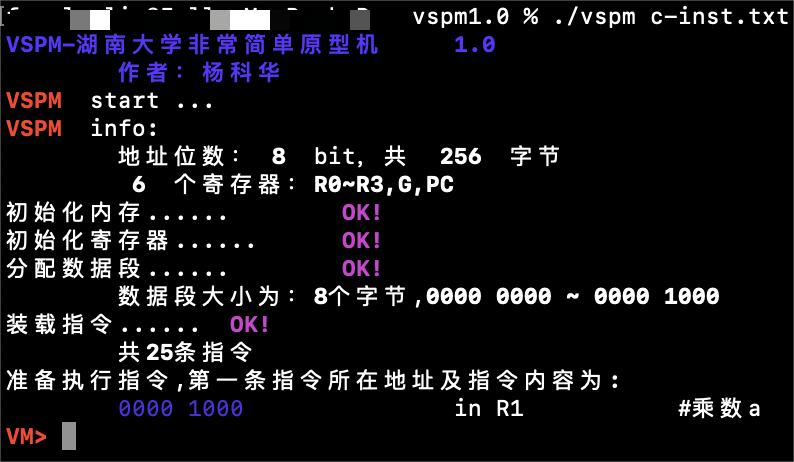
****

****代码所做的工作解释：****

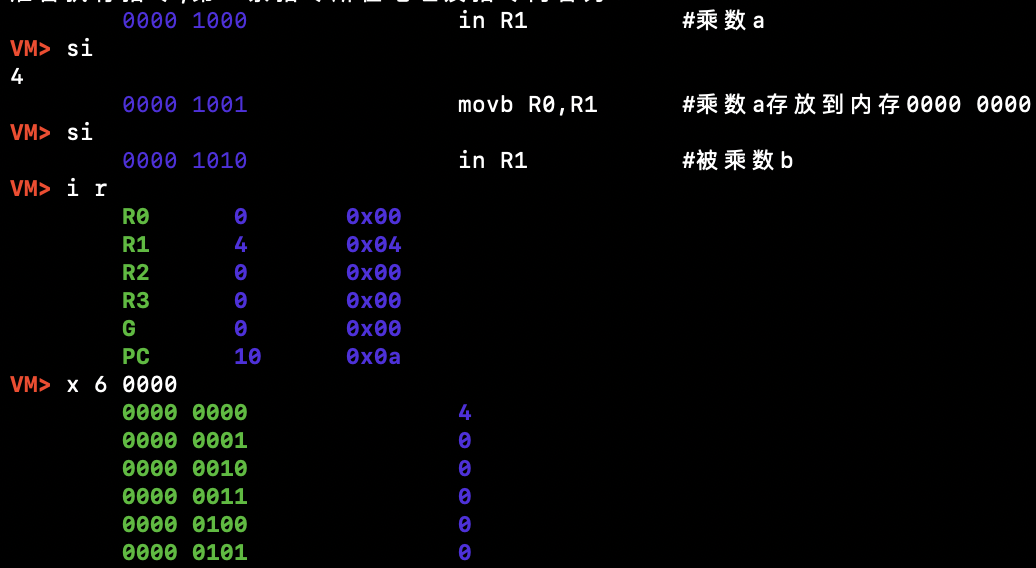
**综上，b-inst.txt中汇编代码的工作是输入两个整数，最后输出较小的那一个数。**

### **运行调试c-inst.txt**

1. 使用./vspm c-inst.txt来运行原型机1.0的模拟器，其中c-inst.txt为代码文件:

****

1. **运行“in R1”和“movb R0，R1”指令且输入整数4后，分别输入i r和x 6 0000指令查看寄存的值和从0000 0000开始6个连续内存地址的值，可观察到值4已存在R1中且被保存在内存地址0000 0000中**



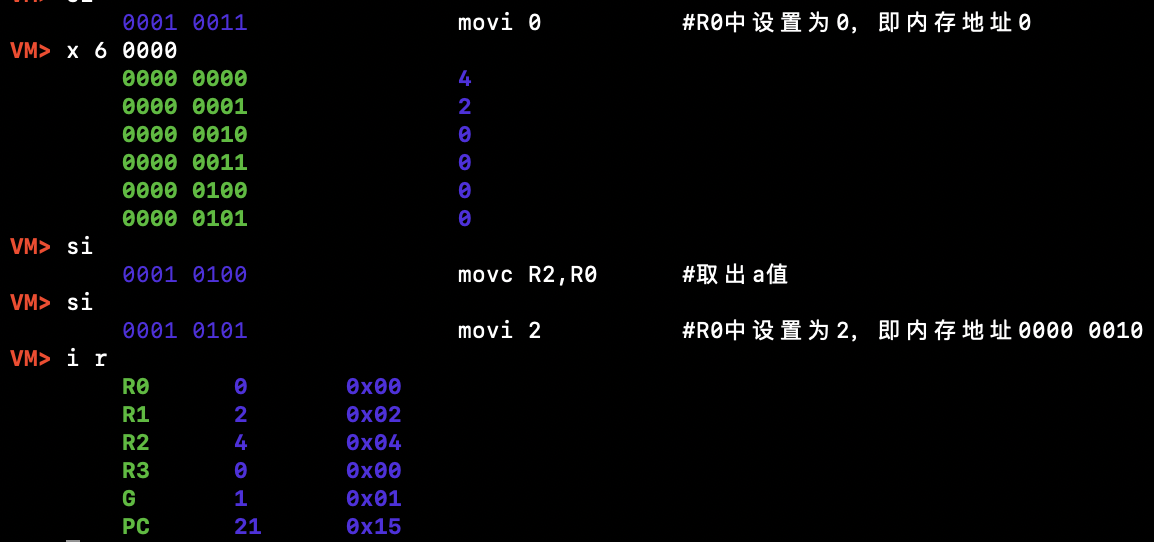
1. 同样执行“in R1”并输入整数3，并将值3保存到内存中，但因为地址0000 0000已保存值4，这里需要执行“movi 1”将把值1保存到R0中，后续执行“movb R0, R1”才能根据R0的值把值4保存到内存地址0000 0001中



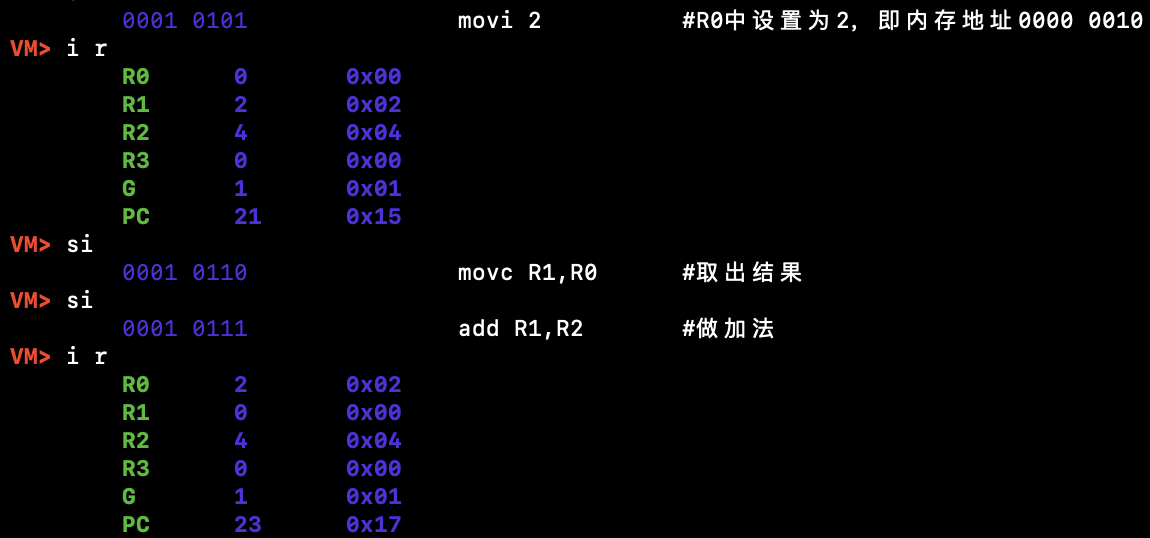
1. 运行“movi 1”和“movc R1, R0”，从内存地址0000 0001中取出值3保存到R1中，输入i r查看寄存器的值可观察到值3确实从内存中取出并保存到R1中，接着运行“movi 1”和“sub R1, R0”后输入i r查看寄存器的值，可观察到R1中的值3已减去1变为2，最后“movi 1”和“movb R1, R0”指令后输入x 6 0000查看内存，可观察到R1中的值2保存到0000 0001地址中



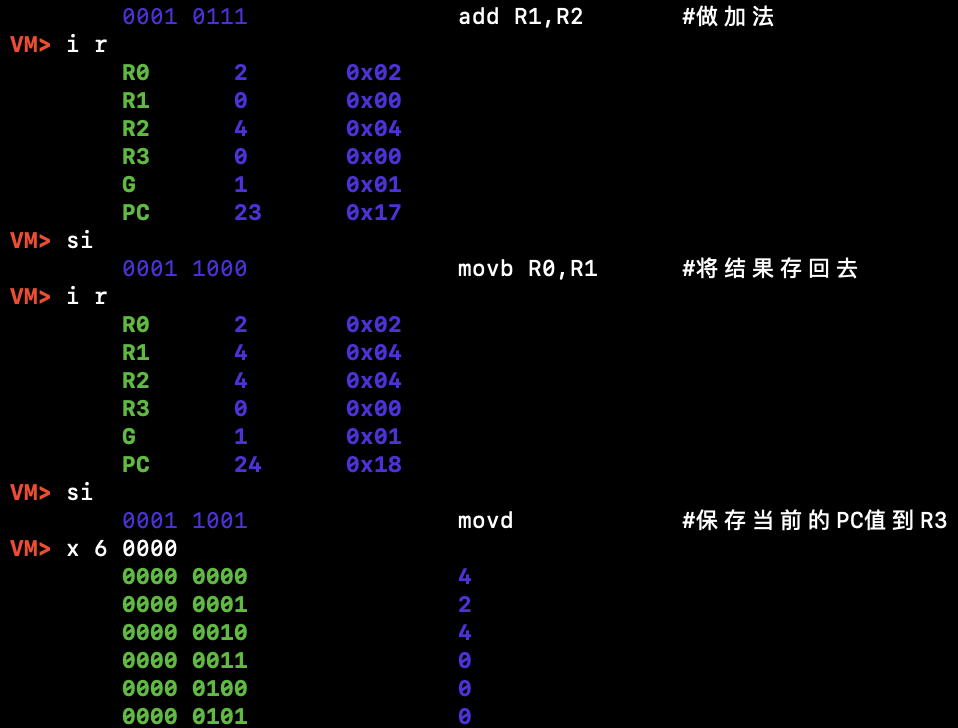
1. 运行“movi 0”和“movc R2，R0”后输入i r查看寄存器的值，可观察到内存地址0000 0000中的值4已取出存放到了R2中



1. 运行“movi 0”、“movc R1，R0”“add R1，R2”指令并查看寄存器的值，可观察到内存地址0000 0010中的值（默认为0）被取出并存放到R1中



1. 运行“add R1，R2”指令后查看寄存器的值，可观察到值4已累加存放到R1中，再运行“movd R0，R1”指令，查看内存地址的值，可观察到R1中的累加结果已存放到0000 0010地址中

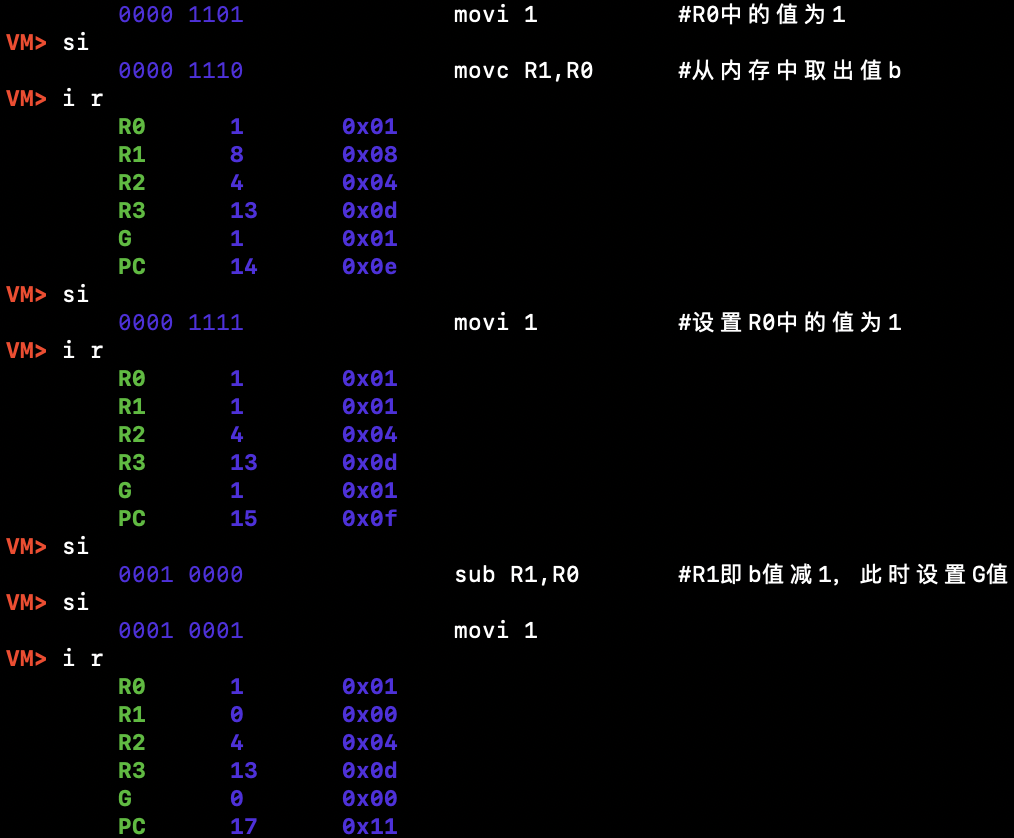


1. 接下来运行这一段汇编指令用于跳转，首先运行“movd”指令将当前PC值26（0001 1001）保存到R3，然后将R3加-12，后续如果执行“jg”指令能跳转到代码第6行，注意这里不能用sub指令直接减去12，因为会影响到G的值。

因为此时G的值为1，执行“jg”指令后可观察到程序已经跳转回到代码第6行



1. 多次执行循环跳转后，此时内存地址0000 0001中到值b已减wei1，再次执行“sub R1, R0”指令后查看寄存器到值，可观察到G到值设置为0

********

1. **后续再执行“jg”指令，因为G的值为0，所以不会再跳转，继续执行下一行指令**

****

1. **最后执行“movi 2”和“movc R1，R0”指令，从内存地址0000 0010中取出累加结果12保存到R1中，最后执行“out R1”指令，打印结果12。**

**执行“halt”停机指令，程序执行结束，原型机停机**

****

****代码所做的工作解释：综上，c-inst.txt中汇编代码的工作是用加法循环模拟实现两个整数的乘法运算,且被乘数b不能是负数****

# **思考问题**

1. 如果基于这些指令实现两个整数的乘法与除法？

答：**因为原型机没有专门的乘法和除法指令，但有加法和减法指令，而乘法运算可以用循环加法运算模拟；除法运算可以用循环减法模拟，其中，除法的结果是循环的次数，余数是循环减法时小于除数的被减数。**

(2)vspm1.0的指令集是否完备？如果是，那么如何证明（提示：搜索并阅读“可计算性理论”）？如果不是，那么要增加哪些指令？

答：**完备；判断一个原型机的指令集是否完备，通常需要检查它是否满足图灵完备性。在[可计算性理论](https://baike.baidu.com/item/%E5%8F%AF%E8%AE%A1%E7%AE%97%E6%80%A7%E7%90%86%E8%AE%BA/0?fromModule=lemma_inlink" \t "/Users/fanglunlin/Documents\\x/_blank)里，如果一系列操作数据的规则（如[指令集](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E9%9B%86/0?fromModule=lemma_inlink" \t "/Users/fanglunlin/Documents\\x/_blank)、[编程语言](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80/0?fromModule=lemma_inlink" \t "/Users/fanglunlin/Documents\\x/_blank)、[细胞自动机](https://baike.baidu.com/item/%E7%BB%86%E8%83%9E%E8%87%AA%E5%8A%A8%E6%9C%BA/0?fromModule=lemma_inlink" \t "/Users/fanglunlin/Documents\\x/_blank)）可以用来模拟单带图灵机，那么它是图灵完备的。如果一个系统是图灵完备的，那么它能够执行任何计算，只要给定足够的时间和资源。**

**vspm1.0指令集可以实现基本算数操作，如加法、减法、乘法、除法等；**

**vspm1.0的指令集能实现数据在内存和寄存器之间移动，以及在内存中的不同位置间移动；**

**vspm1.0的指令集包含控制流指令，如条件跳转；**

**vspm1.0的指令集支持数据外部输入和向外部输出；**

**综上vspm1.0指令集能够模拟图灵机的基本操作，所以它就是图灵完备的。**

(3)如果一台计算机只支持加法、减法操作，那么能否计算三角函数，对数函数？（提示：搜索并阅读“泰勒级数展开”等内容）

答：**可以计算，三角函数和对数函数都可以通过麦克劳林展开来近似计算，其中还涉及乘法和除法运算，但乘法运算可以通过加法循环计算模拟实现，除法运算可以通过减法循环计算模拟实现。**

1. 对于某个需要完成的功能，如果既可以通过硬件上增加电路来实现，也可以通过其他已有指令的组合来实现，那么如何判断哪一种比较合适？（提示：搜索并阅读RISC与CISC）。

答：**不同机器有不同的指令系统，从计算机指令系统设计角度，可将计算机分为复杂指令系统计算机CISC（Complex Instruction Set Computer）和精简指令系统计算机RISC（Reduced Instruction Set Computer），CISC是为了弥补高级语言程序和机器语言程序之间的差异，RISC指令系统是CISC指令系统的改进。**

**如果要完成的功能用CISC架构所包含已有指令的组合来实现，并且这种实现方式不会显著增加复杂度和功耗，则选择这种方法会更合适，但指令会复杂优化编译较难，且效率会受限；**

如果要完成的功能需要在一个时钟周期内执行多条指令，且寄存器和内存之间需要频繁访问存取数据，则选择RISC架构通过硬件上增加电路来实现较合适，但源代码长度较长且工艺成本较高；

综上如果要完成某个功能，如果该功能较为复杂，用已有指令的组合来实现的时间成本较大，则通过硬件上增加电路来实现合适；如果完成该功能通过硬件上增加电路来实现会徒劳增加工艺成本，则用其他已有指令的组合来实现较合适。

# **总结**

## 实验中出现的问题

1. 对汇编代码中部分指令的功能不够熟悉，需要频繁查表了解指令的功能

## 心得体会

1. 使用原型机自己运行调试程序，能更加清晰理解程序运行过程中数据在寄存器和内存的流动，能直观看到原型机是执行汇编指令后的结果，且亲自调试的过程有趣且深刻。
2. 通过调试三个汇编代码程序，我对汇编指令相对应的功能更加熟悉，且能自己编写一些简单汇编代码并自己调试运行验证功能。
3. 学习计算机系统需要自己多动手学习才能学好，其中涉及许多命令行操作，很多指令必须多使用才能记牢，所以一定要多上手实操。