《计算机系统》

原型机 vspm1.0 实验报告

目录

1 实验项目一

1.1 项目名称

实验 1，原型机 vspm1.0。

1.2 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。

1.3 实验资源

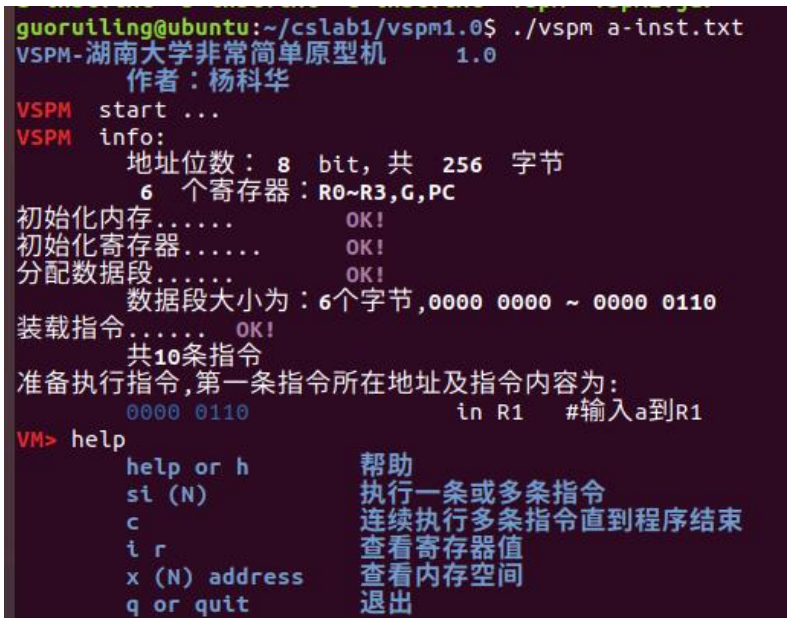
（1） 冯诺伊曼体系的相关内容；

（2） 课程《最小系统与原型机 I》。

2 实验任务

2.1 原型机 a

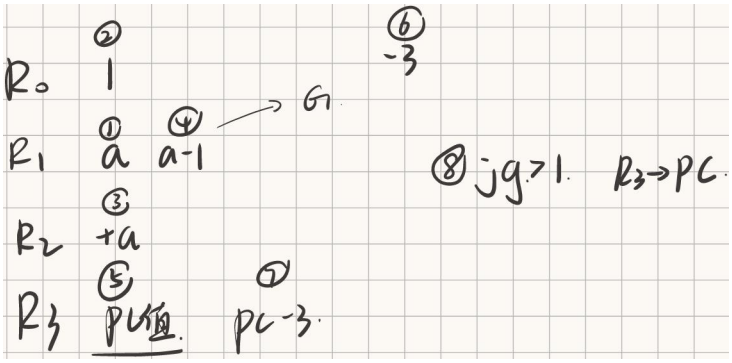
（1） 进入终端，使用 cd vspm1.0 进入目录；并使用 help 查看此模拟器支持的命令；



（2） 不断执行循环直到满足跳转条件。

①将 a 放入寄存器 R1

循环过程：②将 R0 的值置为 1，以对 a 的值减 1；③将 R2 加上当前的 R1 值，实现累加过程：④将 R2 的值减 1，并判断相减前 R2 的值是否大于，记录 G 的值；⑤记录此时的 PC 值，方便后面跳转；⑥将此时 R0 的值置为-3，以实现跳转；⑦R0R3 相加，得到第二步的 PC 值；⑧实现跳转，若 G 值为 1，即实现跳转。若非，继续执行下一指令。




```
VM> i r
        R0      0       0x00
        R1      0       0x00
        R2      0       0x00
        R3      0       0x00
        G       0       0x00
        PC      6       0x06
VM> x 7 0000
        0000 0000               0
        0000 0001               0
        0000 0010               0
        0000 0011               0
        0000 0100               0
        0000 0101               0
        0000 0110               in R1   #输入a到R1
VM> si
5
        0000 0111               movi 1  #设置R0为1
VM> i r
        R0      0       0x00
        R1      5       0x05
        R2      0       0x00
        R3      0       0x00
        G       0       0x00
        PC      7       0x07
VM> si
        0000 1000               add R2,R1       #R2存放累加值
VM> i r
        R0      1       0x01
        R1      5       0x05
        R2      0       0x00
        R3      0       0x00
        G       0       0x00
        PC      8       0x08
VM> si
        0000 1001               sub R1,R0       #R1的值即a减去
```

```
Terminal
        R0      1       0x01
        R1      5       0x05
        R2      5       0x05
        R3      0       0x00
        G       0       0x00
        PC      9       0x09
VM> si
        0000 1010               movd            #将
VM> i r
        R0      1       0x01
        R1      4       0x04
        R2      5       0x05
        R3      0       0x00
        G       1       0x01
        PC      10      0x0a
VM> si
        0000 1011               movi -3 #存放-3到R
VM> i r
        R0      1       0x01
        R1      4       0x04
        R2      5       0x05
        R3      10      0x0a
        G       1       0x01
        PC      11      0x0b
VM> si
        0000 1100               add R3,R0       #R
, 会影响G值
VM> i r
        R0      -3      0xfd
        R1      4       0x04
        R2      5       0x05
        R3      10      0x0a
        G       1       0x01
        PC      12      0x0c
VM> si
        0000 1101               jg              #如
去执行
```

```
VM> i r
        R0      -3      0xfd
        R1      4       0x04
        R2      5       0x05
        R3      7       0x07
        G       1       0x01
        PC      13      0x0d
VM> si
        0000 0111               movi 1  #设置R0为1
VM> i r
        R0      -3      0xfd
        R1      4       0x04
        R2      5       0x05
        R3      7       0x07
        G       1       0x01
        PC      7       0x07
VM> si
        0000 1000               add R2,R1       #R2存放累加值
VM>  i r
        R0      1       0x01
        R1      4       0x04
        R2      5       0x05
        R3      7       0x07
        G       1       0x01
        PC      8       0x08
VM>
```

（3） 跳出循环

直到 R1 中值为 1 的该次循环，执行完 R1-1 后 G 值为 0，在 jg 指令执行时跳出循环，输出 R2 中存储的值，停机。

2.2 原型机 b、c

2.2.1 原型机 b

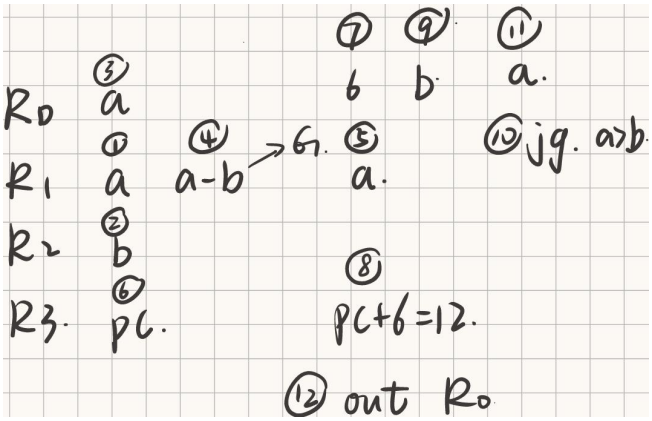
本段指令进行了对两个数字的比较，并输出较小值。

输入：①输入 a，保存在 R1 中；②输入 b，保存在 R2 中；

判断大小：③在 R0 中暂时保存 a 的值；④a-b，在此时设置 G 值，判断 ab 的大小；⑤将 a 重新保存至 R1；

是否跳转：⑥将 PC 值保存至 R3,；⑦将 R0 设置为 6，以备跳转；⑧将 R3 中的 PC 值加 R0，设置跳转后的地址为 12；⑨将 b 的值保存至 R0，此时 R0 中的值为 b，若跳转，将输出 b；⑩跳转指令，根据 G 的值判断是否跳转；11.将 R0 的值设置为 a，若不跳转，将输出 a；

输出并停机：12.输出 R0 中的值；13.停机；



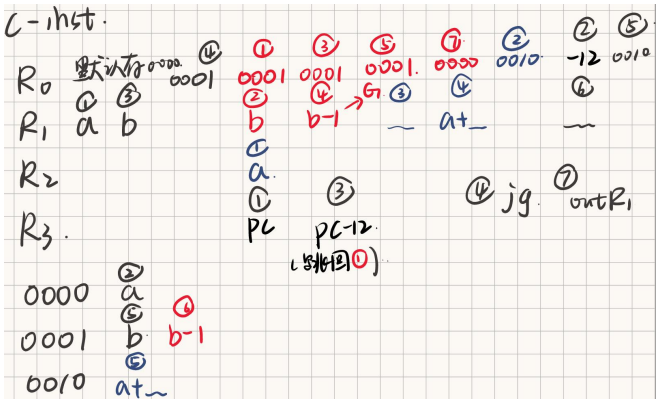

```
guorulling@ubuntu:~/cslab1/vspm1.0$ ./vspm b-inst.txt
VSPM-湖南大学非常简单原型机      1.0
        作者：杨科华
VSPM  start ...
VSPM  info:
        地址位数： 8  bit, 共  256  字节
         6  个寄存器：R0~R3,G,PC
初始化内存.....          OK!
初始化寄存器.....        OK!
分配数据段.....          OK!
        数据段大小为：6个字节,0000 0000 ~ 0000 0110
装载指令.....  OK!
        共13条指令
准备执行指令,第一条指令所在地址及指令内容为:
        0000 0110               in R1           #输入第一个数a
VM> si
2
        0000 0111               in R2           #输入第二个数b
VM> si
4
        0000 1000               mova R0,R1      #在R0保存a
VM> i r
        R0      0       0x00
        R1      2       0x02
        R2      4       0x04
        R3      0       0x00
        G       0       0x00
        PC      8       0x08
VM> si
        0000 1001               sub R1,R2       #a-b,此时会设置G
VM> i r
        R0      2       0x02
        R1      2       0x02
        R2      4       0x04
        R3      0       0x00
        G       0       0x00
        PC      9       0x09
VM> si
        0000 1010               mova R1,R0      #a保存到R1
VM> i r
        R0      2       0x02
        R1      -2      0xfe
        R2      4       0x04
        G       0       0x00
        PC      12      0x0c
VM> si
        0000 1101               add R3,R0       #R3的值加6
VM> i r
        R0      6       0x06
        R1      2       0x02
        R2      4       0x04
        R3      11      0x0b
        G       0       0x00
        PC      13      0x0d
VM> si
        0000 1110               mova R0,R2      #b的值保存到R0
VM> i r
        R0      6       0x06
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
        PC      14      0x0e
VM> si
        0000 1111               jg              #如果a的值比b大,就跳转
VM> i r
        R0      4       0x04
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
        PC      15      0x0f
VM> si
        0001 0000               mova R0,R1 #将a的值保存到R0
VM> i r
        R0      4       0x04
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
        PC      16      0x10
VM> si
        0001 0001               out R0      #输出R0
VM> i r
        R0      2       0x02
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
```

```
VM> si
        0001 0001               out R0      #输出R0
VM> i r
        R0      2       0x02
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
        PC      17      0x11
VM> si
2
        0001 0010               halt
VM> i r
        R0      2       0x02
        R1      2       0x02
        R2      4       0x04
        R3      17      0x11
        G       0       0x00
        PC      18      0x12
VM> si
        0001 0010               halt
程序执行结束，原型机停机。
```

2.2.2 原型机 c

本段指令实现了两个数字 ab 相乘，并输出结果。（使用了内存）

黑色代表了输入输出跳转停机指令，红色是处理 b 的部分，蓝色是处理 a 的部分。



输入：①输入 a 到 R1；②输入 R1 到内存 0000；③输入 b 到 R1；④将 R0 指向的地址设置为 0001；⑤将 b 由 R1 输入到内存 0001；

```
guoruiling@ubuntu:~/cslab1/vspm1.0$ ./vspm c-inst.txt
VSPM-湖南大学非常简单原型机      1.0
        作者：杨科华
VSPM  start ...
VSPM  info:
        地址位数： 8  bit, 共  256  字节
        6  个寄存器：R0~R3,G,PC
初始化内存.....          OK!
初始化寄存器.....        OK!
分配数据段.....          OK!
        数据段大小为：8个字节,0000 0000 ~ 0000 1000
装载指令.....  OK!
        共25条指令
准备执行指令,第一条指令所在地址及指令内容为:
        0000 1000               in R1           #乘数a
VM> si
3
        0000 1001               movb R0,R1      #乘数a存放到内存0000 0000
VM> si
        0000 1010               in R1           #被乘数b
VM> si
2
        0000 1011               movi 1
VM> si
        0000 1100               movb R0,R1      #被乘数b存放在内存0000 0001
VM> si
        0000 1101               movi 1          #R0中的值为1
VM> i r
        R0      1       0x01
        R1      2       0x02
        R2      0       0x00
        R3      0       0x00
        G       0       0x00
        PC      13      0x0d
VM> x 4 0000
        0000 0000               3
        0000 0001               2
        0000 0010               0
        0000 0011               0
VM>
```

对 b 进行操作：①将 R0 的地址设置为 0001；②将 b 从内存取出，存在 R1 中；③将 R0 的地址设置为 0001；④将 R1 中的 b 值减 1，并在此时设置 G 值；⑤将 R0 的地址设置为 0001；⑥将 b-1 的值保存回内存；⑦将 R0 的地址重新设置为 0000；

```
VM> si
        0000 1110               movc R1,R0      #从内存中取出值b
VM> si
        0000 1111               movi 1          #设置R0中的值为1
VM> si
        0001 0000               sub R1,R0       #R1即b值减1，此时设置G值
VM> si
        0001 0001               movi 1
VM> si
        0001 0010               movb R0,R1      #b值需要保存回去
VM> si
        0001 0011               movi 0          #R0中设置为0，即内存地址0
VM> si
        0001 0100               movc R2,R0      #取出a值
VM> i r
        R0      0       0x00
        R1      1       0x01
        R2      0       0x00
        R3      0       0x00
        G       1       0x01
        PC      20      0x14
VM> x 4 0000
        0000 0000               3
        0000 0001               1
        0000 0010               0
        0000 0011               0
VM>
```

对 a 进行操作（对 a 进行累加以实现相乘的效果）：①将 a 的值从内存中取出；②将 R0 的地址设置为 0010；③将内存中存放的结果取出放到 R1 中；④将 R1 中存储的结果加 a；⑤将结果存回内存 0010；

```
VM> si
        0001 0101               movi 2          #R0中设置为2，即内存地址0000 0010
VM> si
        0001 0110               movc R1,R0      #取出结果
VM> si
        0001 0111               add R1,R2       #做加法
VM> si
        0001 1000               movb R0,R1      #将结果存回去
VM> si
        0001 1001               movd            #保存当前的PC值到R3
VM> i r
        R0      2       0x02
        R1      3       0x03
        R2      3       0x03
        R3      0       0x00
        G       1       0x01
        PC      25      0x19
VM> x 4 0000
        0000 0000               3
        0000 0001               1
        0000 0010               3
        0000 0011               0
```

判断跳转：①保存此时 pc 的值在 R3 中；②将 R0 的值置为-12，以备后续改变 PC；③R3 的值-12；④跳转指令，如果此轮循环开始时 b 的值仍然大于 1，跳转回操作 b 前；

输出与停机：⑤设置 R0 的地址为 0010；⑥将内存中的结果取出，放在 R1 中；⑦输出 R1 的内容；⑧停机；

2.3 思考内容

（1）如何基于这些指令实现两个整数的乘法与除法？

乘法：如 c-inst.txt 所示，两个数相乘转化为累加，将一个数作为被累加的数字，另一个数用于计数，每累加一次就将计数值减一，直到全部累加完成。

除法：同样，对于除法，可以进行累减，多次计算 a-b 并计数，计数结果即为答案。

（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？

可计算性理论（Computability theory）是计算机科学的一个基础理论，它研究在不同的计算模型下哪些算法问题能够被解决。这个领域致力于建立计算的数学模型，并精确地区分哪些问题是可计算的，哪些是不可计算的。

在可计算性理论中，一个核心概念是图灵完备性（Turing completeness）。如果一个系统的指令集能够模拟单带图灵机（一种抽象的计算模型），那么这个系统就是图灵完备的。这意味着这个系统能够执行任何可计算的任务，因为单带图灵机被认为具有与任何实际计算机相同的计算能力。

而一个指令集完备的系统通常包含以下几类指令：数据传送（move）、算术运算（add）、逻辑运算、程序控制（jmp）、输入输出（in）、系统控制（halt）。可以看出我们缺少了一

些逻辑指令，与、或、非操作无法实现。另外，用于处理字符串数据的指令也可以增加，如字符串比较、字符串移动等。

（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）

泰勒级数展开是一种将函数表示为无限级数的方法，其中每一项都是基于函数的导数在某个点的值。对于许多常见的数学函数，包括三角函数（如正弦、余弦、正切）和对数函数，都存在泰勒级数展开。理论上可以通过反复应用加法和减法来近似计算正弦函数的值。

大一学习程序设计语言时，也曾编写过用加减操作进行 sinx 的值模拟的代码，但是精度和效率往往不高。所以我们需要注意的是，这种方法的计算效率通常很低，特别是对于需要高精度结果的应用。

（4）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）。

RISC（精简指令集计算机）：RISC 架构强调简单、固定的指令集，通常通过加载/存储架构来执行复杂操作，这意味着大多数操作都通过加载数据到寄存器，执行简单指令，然后存储结果来完成。RISC 架构通常提供更好的性能和功耗效率。

CISC（复杂指令集计算机）：CISC 架构拥有更复杂的指令集，其中包含许多专门的指令来执行特定任务。这可能会提高软件开发的效率，因为某些复杂操作可以通过单条指令完成。然而，CISC 架构的处理器可能面临更高的功耗和更复杂的硬件设计。

在实际操作中，我们需要将性能、成本、功耗和灵活性等因素综合考虑，并根据需求做出选择。对于需要高性能和低功耗的嵌入式系统，硬件实现可能是一个更好的选择。而对于需要高度灵活性和快速原型开发的系统，通过指令组合的软件实现可能更加合适。此外，还需要考虑团队对硬件和软件开发的熟悉程度以及维护和支持的长期成本。

3 总结

3.1 实验中出现的问题

本次实验过程中主要出现了两个问题，一是 ubuntu 上安装的 java 版本过老，实际运行 vspm 时无法执行，参考了 21 级学长分享的解决办法，仍无法解决，后上网查询了资料，用 wget 指令下载并重新指定了默认版本得以解决。二是对实验任务和所需知识点都不够熟悉，对于 vspm 的运行，学习了老师提供的教学视频；对于汇编代码的阅读，我发现将过程在草稿上记录下来会让我的思维更加清晰，在写本实验报告时我也将思考过程截图放在了上面。

3.2 心得体会

第一个实验感觉还是比较好上手的，主要是对汇编代码的阅读与调试。在使用 i r 查看寄存器和使用 x 查看内存的过程中，我感觉我对程序运行的方式的理解更加深入了，而通过阅读最小系统与原型机的文档，我也更加理解了系统的运作过程。