

---

# 《计算机系统》

## 原型机实验报告

班级：GitHub

学号：78268851

姓名：AnicoderAndy

# 目录

1	实验项目一.....	3
1.1	项目名称.....	3
1.2	实验目的.....	3
1.3	实验资源.....	3
2	实验任务.....	4
2.1	实验任务 A.....	4
2.1.1	执行过程 .....	4
2.1.2	程序分析 .....	5
2.2	实验任务 B.....	6
2.2.1	调试并解释 b-inst.txt.....	6
2.2.2	调试并解释 c-inst.txt.....	7
2.3	实验任务 C.....	8
2.3.1	实现乘除法 .....	8
2.3.2	完备性证明 .....	9
2.3.3	泰勒展开求函数值 .....	9
2.3.4	对比 RISC 和 CISC.....	9
3	总结.....	11
3.1	实验中出现的問題.....	11
3.2	心得体会.....	11
4	附件.....	12
4.1	除法程序代码.....	12

# 1 实验项目一

## 1.1 项目名称

原型机 vspm1.0

## 1.2 实验目的

- 1) 了解冯诺依曼体系结构；
- 2) 理解指令集结构及其作用；
- 3) 理解计算机的运行过程，即指令执行过程，并初步掌握调试方法。

## 1.3 实验资源

- 1) 课程《最小系统与原型机 I》
- 2) miniCC 工具链
- 3) vspm 原型机模拟器
- 4) 教材中冯诺依曼体系的相关内容

## 2 实验任务

### 2.1 实验任务 A

任务名称：按照说明文件中的实验步骤调试 a-inst.txt

#### 2.1.1 执行过程

1) 执行 ./vspm a-inst.txt，开始逐步调试

```

      共10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 a到 R1
VM> i r
      R0      0      0x00
      R1      0      0x00
      R2      0      0x00
      R3      0      0x00
      G       0      0x00
      PC      6      0x06
VM> si 3
3
      0000 1001          sub R1,R0    #R1的值即 a减去1,此时
会设置 G 值
VM> i r
      R0      1      0x01
      R1      3      0x03
      R2      3      0x03
      R3      0      0x00
      G       0      0x00
      PC      9      0x09
VM>

```

2) 执行 movd 指令并且将 R3 减去 3，运行到 jg 前观察寄存器的值。

```

VM> si 3
      0000 1100          add R3,R0    #R3减去3, 注意此时不
能用 SUB指令, 会影响 G值
VM> si
      0000 1101          jg          #如果 R1的值还大于1,
则跳到第2行去执行
VM> i r
      R0      -3      0xfd
      R1      2      0x02
      R2      3      0x03
      R3      7      0x07
      G       1      0x01
      PC     13      0x0d
VM> si
      0000 0111          movi 1    #设置 R0为1

```

可以发现寄存器 R3 被正常赋值为 7，之后执行 `lg` 指令正常跳转到第 2 行。

3) 再执行两次循环，观察寄存器值的变化。

```

VM> si 6
      0000 1101          lg          #如果 R1 的值还大于 1,
则跳到第 2 行去执行
VM> i r
      R0      -3      0xfd
      R1       1      0x01
      R2       5      0x05
      R3       7      0x07
      G        1      0x01
      PC      13      0x0d

VM> si
      0000 0111          movi 1    #设置 R0 为 1
VM> si 6
      0000 1101          lg          #如果 R1 的值还大于 1,
则跳到第 2 行去执行
VM> i r
      R0      -3      0xfd
      R1       0      0x00
      R2       6      0x06
      R3       7      0x07
      G        0      0x00
      PC      13      0x0d

VM> si
      0000 1110          out R2      #如果 R1 的值此时小于
等于 1, 则准备输出

```

寄存器的值正常变化。因为 R1 在进行 `sub` 操作后值小于 1，故未执行 `lg` 指令，准备输出。

4) 继续执行指令。

```

VM> si
6
      0000 1111          halt        #停机
VM> si
      0000 1111          halt        #停机
程序执行结束，原型机停机。

```

原型机正常输出  $\sum_{i=0}^3 i = 6$ 。

### 2.1.2 程序分析

本程序利用寄存器 R2 记录累加结果，寄存器 R1 记录输入值并且充当循环变量。每次循环，R1 存储值会减少 1。当 R1 减少到 0 时，寄存器 G 记录状态为 0，此时原型机不再执行 `lg` 指令，跳转到输出语句，接着执行 `halt` 停机。

## 2.2 实验任务 B

任务名称：调试并解释 b-inst.txt 和 c-inst.txt。

### 2.2.1 调试并解释 b-inst.txt

源代码存在格式问题，修正后如下：

```

vim
1 6
2 in R1      #输入第一个数 a
3 in R2      #输入第二个数 b
4 mova R0,R1 #在 R0 保存 a
5 sub R1,R2  #a-b,此时会设置 G
6 mova R1,R0 #a 保存到 R1
7 movd      #保存当前的 PC 值到 R3
8 movi 6     #R0 的值设置为 6,即跳转到 11 行
9 add R3,R0  #R3 的值加 6
10 mova R0,R2 #b 的值保存到 R0
11 jg       #如果 a 的值比 b 大,就跳转
12          #跳转地址
13 mova R0,R1 #将 a 的值保存到 R0
14 out R0    #输出 R0
15 halt

```

执行前 5 条指令，输入 5 和 3。发现寄存器 R1 再暂存了  $a - b$  的结果后又保存 R0 中暂存的 a 的值，R2 时刻保存 b 的值。

```

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 5
5
3
0000 1011      movd      #保存当前的 PC 值到 R3
VM> i r
R0      5      0x05
R1      5      0x05
R2      3      0x03
R3      0      0x00
G        1      0x01
PC      11     0x0b
VM>

```

此时 G 为 1，会执行跳转指令跳过第 10 条指令 mova，直接执行第 11 条指令 out。

```

VM> c
3
程序执行结束，原型机停机。

```

结合调试过程以及汇编语句，可以知道该程序通过 sub R1,R2 执行后设置的 G 值执行跳转，当  $a > b$  时由于跳转直接输出 R0 中的 b，其他情况下先把 R1 中的 a 赋给 R0 后再输出。该程序的功能也就是输出  $\min(a, b)$ 。

## 2.2.2 调试并解释 c-inst.txt

1) 查看源代码如下:

```

vim
1 8
2 #两个大于1的正数相乘
3 in R1      #乘数 a
4 movb R0,R1 #乘数 a 存放到内存 0000 0000
5 in R1      #被乘数 b
6 movi 1
7 movb R0,R1 #被乘数 b 存放在内存 0000 0001
8           #结果存放在内存 0000 0010
9 #开始循环
10 movi 1    #R0中的值为1
11 movc R1,R0 #从内存中取出值 b
12 movi 1    #设置 R0 中的值为 1
13 sub R1,R0 #R1 即 b 值减 1, 此时设置 G 值
14 movi 1
15 movb R0,R1 #b 值需要保存回去
16 movi 0    #R0 中设置为 0, 即内存地址 0
17 movc R2,R0 #取出 a 值
18 movi 2    #R0 中设置为 2, 即内存地址 0000 0010
19 movc R1,R0 #取出结果
20 add R1,R2  #做加法
21 movb R0,R1 #将结果存回去
22 movd      #保存当前的 PC 值到 R3
23 movi -12  #R0 的值设置为 -12
24 add R3,R0  #R3 的值加 -12
25 jg        #如果第 12 行的减法设置 G 为 1, 就跳转
26 #循环结束
27 movi 2    #R0 中设置为 2, 即内存地址 0000 0010
28 movc R1,R0 #取出结果
29 out R1     #打印结果
30 hal

```

2) 执行前 5 行, 查看内存中的值, 发现输入 3 和 2 正常存入内存。

```

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 5
3
2
0000 1101      movi 1      #R0 中的值为 1
VM> x 3 0 1 2
无法识别的命令
0000 0000      3
0000 0001      2
0000 0010      0

```

3) 执行接下来的 12 条命令, 发现内存地址 02 存的数字减去了 1, 内存地址 03 存的数字加上了 3。

```

VM> si 12
0001 1001      movd      #保存当前的 PC 值到 R3
VM> x 3 0 1 2
无法识别的命令
0000 0000      3
0000 0001      1
0000 0010      3

```

- 4) 查看此时寄存器内存放的值。由于先前执行 `sub` 指令得到结果大于 0，故 G 为 1。

```
VM> i r
      R0      2      0x02
      R1      3      0x03
      R2      3      0x03
      R3      0      0x00
      G       1      0x01
      PC     25      0x19
```

- 5) 执行接下来的 4 条指令，程序正常跳转。

```
VM> si 3
      0001 1100      jg      #如果第12行的减法设置G为1,就跳转
VM> si
      0000 1101      movi 1      #R0中的值为1
VM> i r
      R0      -12    0xf4
      R1      3      0x03
      R2      3      0x03
      R3      13     0x0d
      G       1      0x01
      PC     13     0x0d
```

- 6) 执行接下来的 12 条指令，完成第二次循环，观察内存中的值。

```
VM> si 12
      0001 1001      movd
VM> x 3 0 1 2
无法识别的命令
      0000 0000      3
      0000 0001      0
      0000 0010      6
```

注意到此时 01 地址的值经过 `sub` 已经变为了 0，此时寄存器 G 存放 0，接下来不会执行跳转。

- 7) 执行到停机，程序不再跳转，而是输出内存 02 中保存的结果 6。

```
VM> si 4
      0001 1101      movi 2
地址 0000 0010
VM> c
6
程序执行结束，原型机停机。
```

分析：结合调试过程以及汇编代码，不难发现该程序通过累加 b 次 a 得到  $a \times b$  的结果并且输出。

## 2.3 实验任务 C

任务名称：思考问题。

### 2.3.1 实现乘除法

问：如何基于这些指令实现两个整数的乘法与除法？

答：乘法已经在 `c-inst.txt` 中实现，即通过执行 b 次累加 a 的操作得到结果，将累加值暂存在



内存中，即可得到乘法结果。累加循环控制可以通过 `sub` 和 `jb` 的组合实现，累加操作通过 `add` 实现，从内存读写数据可以通过 `movb, movc` 指令实现。除法与乘法类似，计算  $a/b$  可以通过执行若干次  $a - b$  实现：（过程一）用 `sub` 命令做  $a - b$ ，将运算结果覆盖内存中存储的  $a$ ；

（过程二）如果  $a < 0$  则执行 `jb` 命令跳过程三；（过程三）结果计数器自增 1，将计数器的值存到内存中，执行 `jmp` 命令回到过程一。完成循环后计数器的值即为除法结果。附件中给出了除法的代码<sup>4.1</sup>。

### 2.3.2 完备性证明

问：vspm1.0 的指令集是否完备？如果是，那么如何证明？如果不是，那么要增加哪些指令？

答：根据可计算性理论，一个指令集完备需要具备条件分支、无限循环、数据存储和操作、基本算术和逻辑操作、输入输出操作这些功能。vspm 的指令集可通过 `JG` 指令实现条件分支，通过 `JMP` 和 `JG` 实现循环，通过 `MOVA, MOVB, MOVC, MOVD` 指令存储和操作数据，通过 `ADD` 和 `SUB` 命令进行基本算数计算，通过 `IN` 和 `OUT` 执行输入输出操作。所以 vspm1.0 的指令集是完备的。

### 2.3.3 泰勒展开求函数值

问：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？

答：可以通过将三角函数、对数函数进行泰勒展开求解其近似值。

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n}$$

据此，可以只通过循环加减乘除法来求三角函数、对数函数。乘除法可以用加减法实现<sup>2.3.1</sup>。

### 2.3.4 对比 RISC 和 CISC

问：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？

答：RISC 精简指令集计算机通过组合简单指令来实现复杂操作；CISC 复杂指令集则通过增

加专用硬件支持复杂指令，减少程序中指令数量，虽然提高了硬件复杂度，但减少了软件的复杂性。

以下场景宜选择 **RISC**：实现可以被拆解为基本操作的复杂功能（例如复杂数学运算）；强调灵活性和扩展性（方便添加新功能）的情况；对功耗有限制的情况。

以下场景宜选择 **CISC**：实现复杂但频繁使用的功能（例如浮点运算、内存操作、字符串处理）；需要减少代码长度或指令数量的情况；对计算性能有要求的情况。

## 3 总结

### 3.1 实验中出现的問題

2.1 撰写该部分报告时，无法解决 Word 小标题缩进过长的问题，通过右键标号，调整列表缩进解决。

2.1.1 使用 vspm 工具时，发现通过 `sudo apt install java` 安装的 Java 版本过低，通过安装 OpenJDK 21 解决。

2.3.4 不了解 RISC 和 CISC 概念，通过询问 AI 工具以及查阅资料解决。

### 3.2 心得体会

学习了原型机的指令和 miniCC 工具链的使用，对麻雀虽小五脏俱全的工具集震撼不已，深切感叹 HNU 老师技术力之强悍。

思考题的问题科普性质很强，本人通过查阅资料学到了例如 RISC，CISC，完备性证明等新知识。

本次任务实操内容过于简单，形成报告显得水分过多，建议课程开始不久内容不充足的情况下不开设实验。Word 排版略显麻烦，建议改用 LaTeX 或 Markdown。

## 4 附件

### 4.1 除法程序代码

```
1    10
2    in R1
3    movb R0, R1
4    movi 1
5    in R1
6    movb R0, R1
7    movi 1
8    movc R2, R0
9    movi 0
10   movc R1, R0
11   sub R1, R2
12   movb R0, R1
13   movi 0
14   sub R0, R1
15   movd
16   movi 14
17   add R3, R0
18   jg
19   movi 2
20   movc R1, R0
21   movi 1
22   add R1, R0
23   movi 2
24   movb R0, R1
25   movd
26   movi -18
27   add R3, R0
28   jmp
29   movi 2
30   movc R1, R0
31   out R1
32   halt
```