《计算机系统》

原型机-实验报告

班级：软件 2203 班

学号：202226010304

姓名：丁海桐

目录

1 实验项目一

1.1 项目名称

原型机 vspm1.0

1.2 实验目的

i. 了解冯诺伊曼体系结构；

ii. 理解指令集结构及其作用；

iii. 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

i. 教材《深入理解计算机系统》

ii. 视频《最小系统与原型机》

iii. 头歌教学平台

2 实验任务

2.1 使用 VSPM 分析代码

2.1.1 用 VSPM 分析、调试、运行 a-inst.txt

inst.txt 文件内容如下:

```
6
in R1          #输入 a 到 R1
movi 1         #设置 R0 为 1
add R2,R1      #R2 存放累加值
sub R1,R0      #R1 的值即 a 减去 1,此时会设置 G 值
movd           #将当前 PC 值保存在 R3 中
movi -3        #存放-3 到 R0 中,跳转到第二行
add R3,R0      #R3 减去 3，注意此时不能用 SUB 指令，会影响 G 值
jg             #如果 R1 的值还大于 1，则跳到第 2 行去执行
out R2         #如果 R1 的值此时小于等于 1，则准备输出
halt           #停机
```

该程序对于一个输入 a ，输出 a + a – 1 + a – 2 + … + 1 的值。

[in R1]

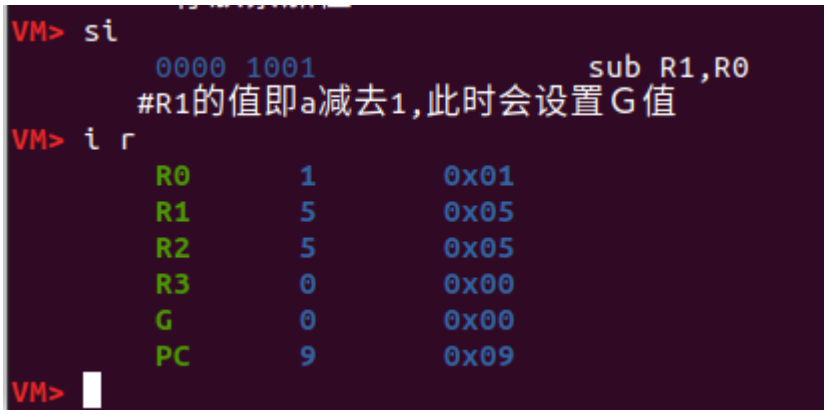
R1 = a（假设 a = 5）

[movi 1]

R0 = 1，R0 的 1 用来将 R1 的减去 1，再将结果加到 R2 的累加值中。

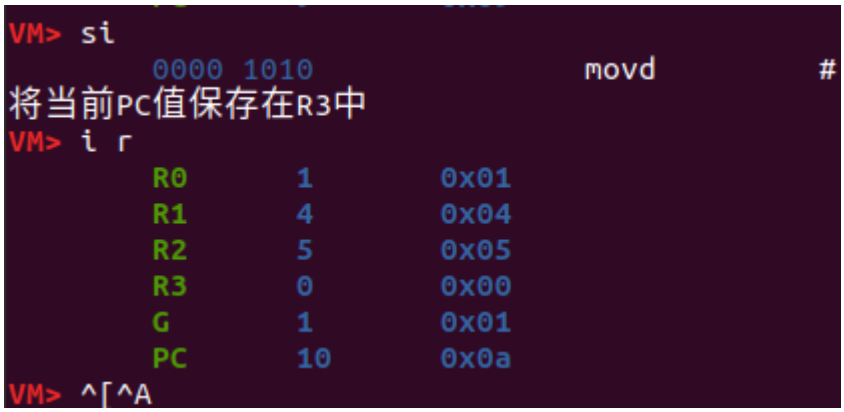
[add R2, R1]

R2 = R2 + R1，R2 存放的是累加值，也就是最后输出结果。此时 sum = a。



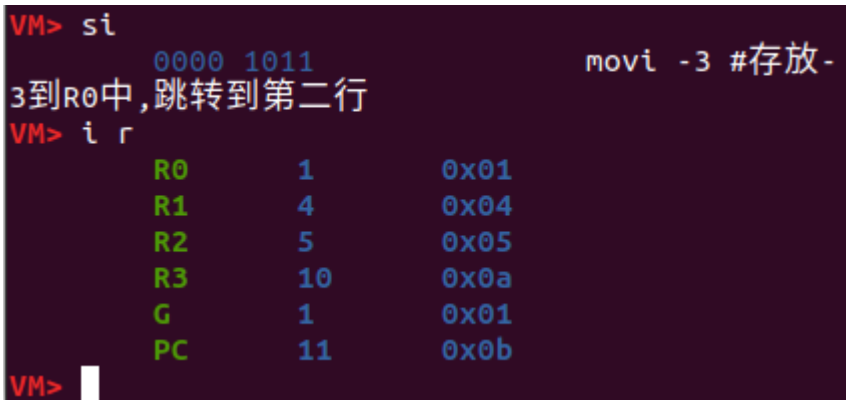
[sub R1, R0]

R1 = R1 –1= a – 1 = 4， R1 递减，等待下一次累加到 R2 的 sum 上。由于 R1 > 1，所以 G = 1，这说明 R1 还可以继续递减累加到 R2，所以遇到[jg]时，就会跳转回开头，实现循环。



[movd]

存储当前[movd] 指令的地址，存到 R3 中。当前地址为 0000 1010。



[movi -3]

[add R3 R0]

改变 R3 存储的地址，也就是改变跳转后的地址到 0000 0111，该指令为[movi

1]

```
VM> x 10 00000000
        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
        0000 0111                movi 1   #设置R0为1
        0000 1000                add R2,R1     #R2存放累加
        0000 1001                sub R1,R0     #R1的值即a减
```

[jg]

判断 G 值，为 1 则 PC = R3，进行跳转，也就是进行下一次循环。

```
VM> si
        0000 1101                jg              #如果R1的值
还大于1，则跳到第2行去执行
VM> si
        0000 0111                movi 1   #设置R0为1
VM>
```

[out R2]

[halt]

输出累加值 15， 停机。

```
VM> si 10
15
        0000 1111                halt            #停机
程序执行结束，原型机停机。
```

2.1.2 用 VSPM 分析、调试、运行 b-inst.txt

b-inst.txt 文件内容如下：

```
6
in R1          #输入第一个数 a
in R2          #输入第二个数 b
mova R0,R1     #在 R0 保存 a
sub R1,R2      #a-b,此时会设置 G
mova R1,R0     #a 保存到 R1   #R1 存 a R2 存 b
movd           #保存当前的 PC 值到 R3
movi 6         #R0 的值设置为 6,即跳转到 11 行
add R3,R0      #R3 的值加 6
mova R0,R2     #b 的值保存到 R0
jg             #如果 a 的值比 b 大,就跳转
mova R0,R1     #将 a 的值保存到 R0
out R0         #输出 R0 jg 目标地址
halt
```

该程序对于两个输入 a、b ，输出其中较小的值。

[in R1]

[in R2]

R1 = a（假设 a = 5），R2 = b（假设 b = 4）

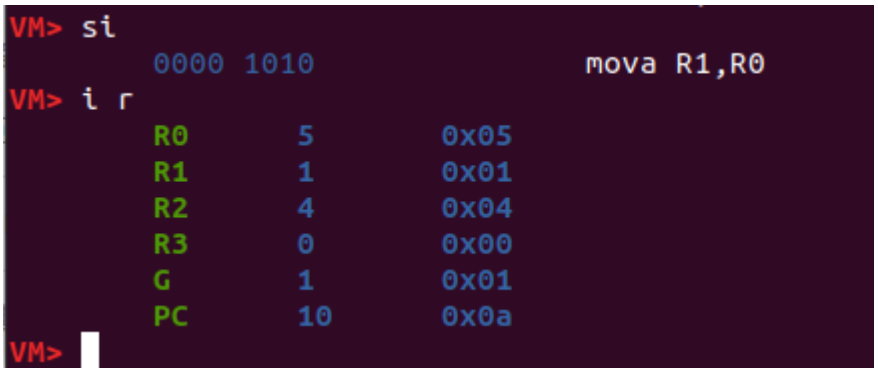
[mova R0,R1]

在 R0 中备份 R1 中的 a，这是因为后面有[sub]指令，会损坏 R1 中的 a

[sub R1,R2]

这里是在比较 a、b 大小，要是 b 小，G = 1，反之 G = 0。

这里可以看到 R0 为 a = 5 的备份，R1 = a – b = 1 > 0，所以 G = 1。



[mova R1,R0]

将 R0 备份的 a 值放回 R1。此时 R1 = a = 5。

```
VM> si
        0000 1011                   movd            #
VM> i r
        R0      5       0x05
        R1      5       0x05
        R2      4       0x04
        R3      0       0x00
        G       1       0x01
        PC      11      0x0b
VM>
```

[movd]

存储当前[movd] 指令的地址，存到 R3 中。当前地址为 0000 1011。

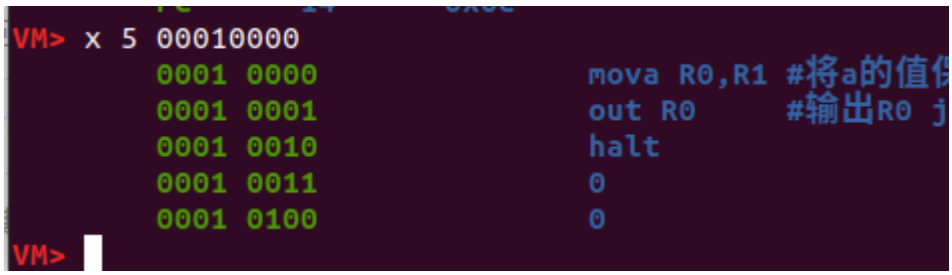
```
VM> si
        0000 1100                   movi 6          #
VM> i r
        R0      5       0x05
        R1      5       0x05
        R2      4       0x04
        R3      11      0x0b
        G       1       0x01
        PC      12      0x0c
VM>
```

[movi 6]

[add R3, R0]

改变 R3 存储的地址，也就是改变跳转后的地址到 0001 0000，该指令为 [out R0]，也就是直接输出默认的 b。

```
VM> si
        0000 1101                   add R3,R0       #R3的
VM> si
        0000 1110                   mova R0,R2      #b的
VM> i r
        R0      6       0x06
        R1      5       0x05
        R2      4       0x04
        R3      17      0x11
        G       1       0x01
        PC      14      0x0e
VM>
```



[mova R0,R2]

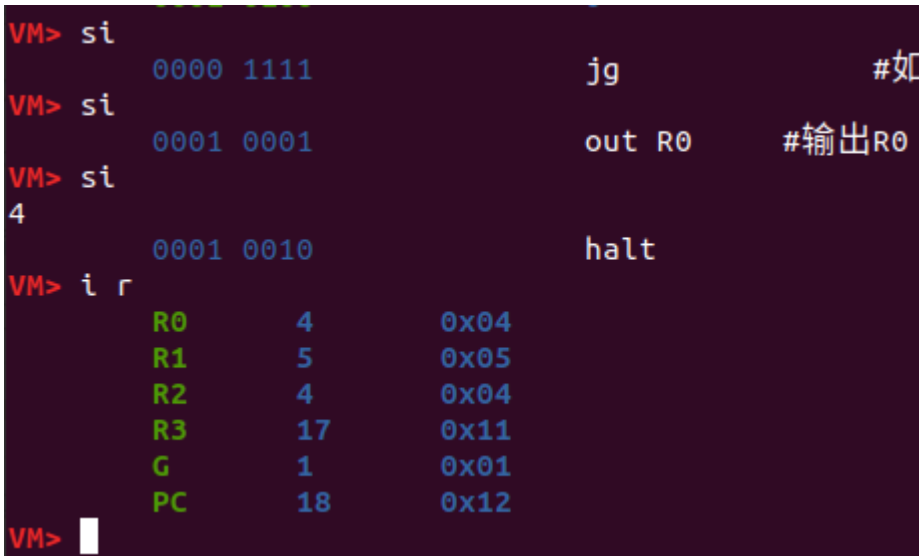
R0 中存放要输出的值，这里先默认 b 小，输出 b，再根据后面[jg]验证，要是错误再换成 a。

[jg]

判断 G 值，为 1 则跳过下一个指令，直接输出 R0 中的 b。这里 G = 1，表示已经用[sub] 指令判断了 b < a，所以默认输出 b 不再更换。

[mova R0,R1]

将 R1 中存的 a 放到 R0，准备输出。这里被跳过了。



[out R2]

[halt]

输出较小值 b = 4， 停机。

2.1.3 用 VSPM 分析、调试、运行 c-inst.txt

c-inst.txt 文件内容如下:

```
8
#两个大于 1 的正数相乘

in R1          #乘数 a
movb R0,R1  #乘数 a 存放到内存 0000 0000
in R1          #被乘数 b
movi 1
movb R0,R1  #被乘数 b 存放在内存 0000 0001
               #结果存放在内存  0000 0010
movi 1         #R0 中的值为 1
movc R1,R0  #从内存中取出值 b
movi 1         #设置 R0 中的值为 1
sub R1,R0      #R1 即 b 值减 1，此时设置 G 值
movi 1
movb R0,R1  #b 值需要保存回去
movi 0         #R0 中设置为 0，即内存地址 0
movc R2,R0  #取出 a 值
movi 2         #R0 中设置为 2，即内存地址 0000 0010
movc R1,R0  #取出结果
add R1,R2      #做加法
movb R0,R1  #将结果存回去
movd           #保存当前的 PC 值到 R3
movi -12           #R0 的值设置为-12
add R3,R0      #R3 的值加-12
jg             #如果第 12 行的减法设置 G 为 1,就跳转
movi 2         #R0 中设置为 2，即内存地址 0000 0010
movc R1,R0  #取出结果
out R1         #打印结果
halt
```

该程序对于两个输入 a、b，输出 a * b 的值（结果不能溢出）

[in R1]

[movb R0,R1]

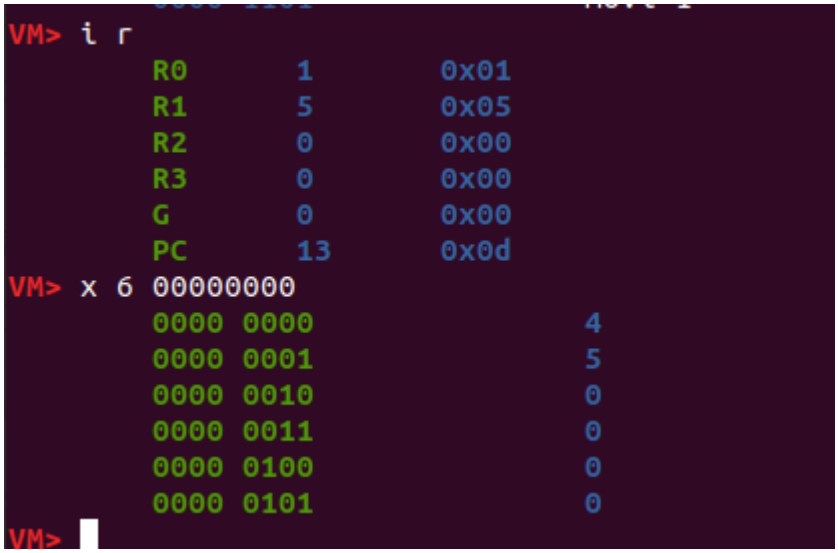
[in R1]

[movi 1]

[movb R0,R1]

乘数 a= 4 存放到内存 0000 0000，被乘数 b= 5 存放在内存 0000 0001，

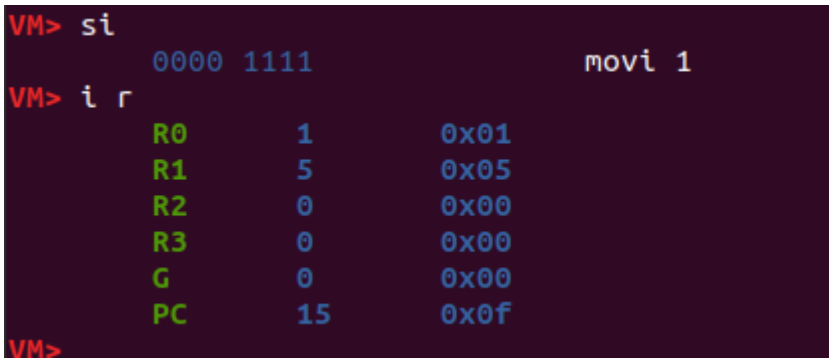
结果存放在内存 0000 0010。



[movi 1]

[movc R1,R0]

从内存 0000 0001 中取出 b = 5 放到 R1 中。



[movi 1]

[sub R1,R0]

R1 中的 b 相当于 while(b--)，用来统计循环了多少次，也就是需要加多少次 a。这里 G = 1，所以下面遇到[jg]指令就跳转到上面的取出 b 那一步，开始新的循环。

```
VM> si
        0001 0001                   movi 1
VM> i r
        R0      1       0x01
        R1      4       0x04
        R2      0       0x00
        R3      0       0x00
        G       1       0x01
        PC      17      0x11
VM>
```

[movi 1]

[movb R0,R1]

把 R1 中的 b 放回 0000 0001 的位置。0000 0000 保存的是 a，一直不变，0000 0010 保存的是结果。

```
VM> si
        0001 0011                   movi 0
VM> x 5 00000000
        0000 0000                   4
        0000 0001                   4
        0000 0010                   0
        0000 0011                   0
        0000 0100                   0
VM>
```

[movi 0]

[movc R2,R0]

[movi 2]

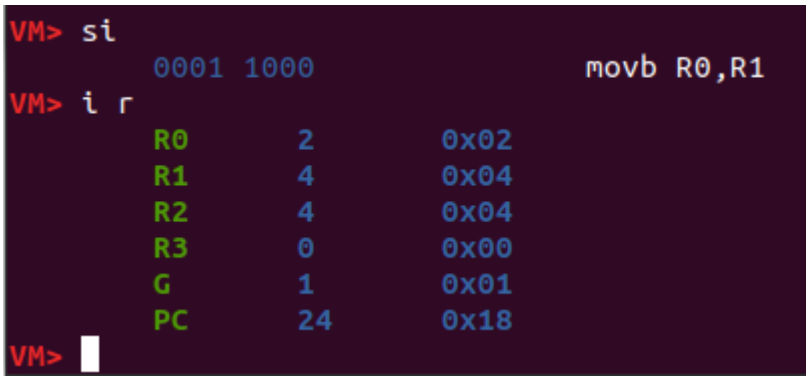
[movc R1,R0]

取出 a 放到 R2，取出结果值 ans 放到 R1。这里 a = 4,，不会改变，本次循环 ans 会没有+ a，所以当前为 0。

```
VM> si
        0001 0110                   movc R1,R0
VM> si
        0001 0111                   add R1,R2
VM> i r
        R0      2       0x02
        R1      0       0x00
        R2      4       0x04
        R3      0       0x00
        G       1       0x01
        PC      23      0x17
VM>
```

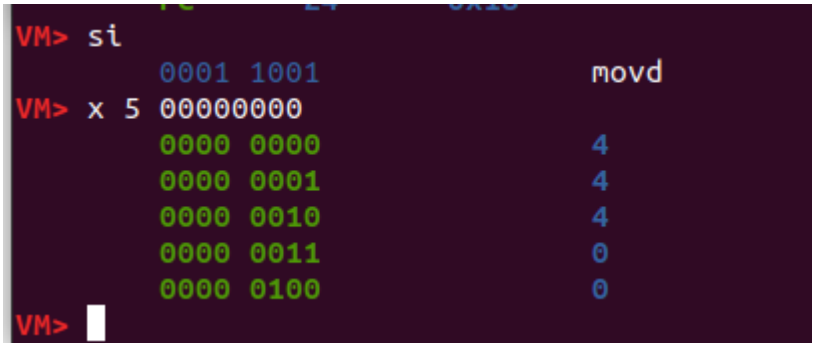
[add R1,R2]

R2 = R2 + R1，ans += a。



[movb R0,R1]

内存 0000 0010 中保存的是 ans，此时加了一次 a，结果为 4。



[movd]

[movi -12]

[add R3,R0]

[jg]

当前 R3 = 13，对应的内存储存的指令为[movi 1]，也就是准备取出 b 减 1，此时 G = 1，应该跳转。

```
VM> x 8 00001000
        0000 1000               in R1           #乘数a
        0000 1001               movb R0,R1      #乘数a存放到内存0000 0000
        0000 1010               in R1           #被乘数b
        0000 1011               movi 1
        0000 1100               movb R0,R1      #被乘数b存放在内存0000 0001
        0000 1101               movi 1          #R0中的值为1
        0000 1110               movc R1,R0      #从内存中取出值b
        0000 1111               movi 1          #设置R0中的值为1
VM>
```

[movi 2]

[movc R1,R0]

[out R1]

[halt]

从 0000 0010 中取出结果 ans 并打印。当前 a = 4，b = 5，　ans = 20。

```
VM> si 20
20
        0010 0000                       halt
程序执行结束，原型机停机。
```

2.2 思考问题

2.2.1 如何基于这些指令实现两个整数的乘法与除法？

（1） 两个数的乘法在 c-inst.txt 中已经实现，原理就是累加。把 b 当做循环次数，不断累加 a，得到结果 ans。

（2） 两个数的除法原理如下：对于被除数 a 和除数 b，比较二者的大小，若 a >= b，则商 c 加 1，a 减去 b。之后 a 与 b 再次比较，一直循环直到 a < b，此时将 a 的值赋给余数 d。

具体代码如下：

```
6
#1、输入阶段
in      R1          #被除数 a R1
movb R0, R1         #被除数 a 存放到内存 0x00
in      R2          #除数 b R2
movi    1
movb R0, R2         #除数 b 存放到内存 0x01
                    #商 c 存放到内存 0x02
                    #余数 d 存放到内存 0x03

#2、把 a、b 分别从内存中取出放到 R1、R2 中
movi    0
movc R1, R0         #R1 = a
movi    1
movc R2, R0         #R2 = b

#3、比较 a 与 b，得到 G 值
sub     R2, R1      #如果 b > a 说明不用循环，后面用[jg]跳出循环
add     R2, R1      #恢复 b 的值

#4、 若 b > a，说明商已经确定，无需进行步骤 5、6、7，跳到 8
movd                #R3 = 0001 0001 = 17
movi    17
add     R3, R0      #R3 = 0010 0010 = 34
jg                  #相当于 break

#5、处理被除数 a，a = a – b，并放回内存
sub     R1, R2      #R1 = a - b
movi    0
movb R0, R1         #0x00 被除数 a : a -> a - b

#6、处理商 c，c = c + 1，并放回内存
movi    2
movc R2, R0         #R2 = c
movi    1
add     R2, R0      #R2 = c + 1
movi    2
movb R0, R2         #0x02 商 c : c -> c + 1

#7、若 G = 0，说明 a > b，商还没有确定，继续循环，跳到步骤 2
movd                #R3 = 0001 1110 = 30
movi    -19
add     R3, R0      #R3 = 0000 1011 = 11
jmp
```

```
#8、处理余数，此时商已经确定，余数 d 就是剩下 R1 中的 a
movi   0
movc R1, R0        #R1 = a [ - b]...
movi   3
movb R0, R1        #在把余数送到 0x03

#9、取出商 c 到 R2，余数就是 R1 中剩下的 a，输出 R1 就是余数
movi 2
movc R2, R0

#10、输出商和余数
out R2
out R1
halt
```

2.2.2 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？

完备。有运算、分支、读写内存指令。00

2.2.3 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）

一台只支持加法和减法操作的计算机无法**直接**计算三角函数或对数函数这类复杂函数。然而，在数学中，许多复杂的函数可以通过它们在某一点处的泰勒级数来近似表示和计算。因此我们可以**近似**计算三角函数和对数函数。

（1）三角函数的泰勒级数展开，以正弦函数的泰勒级数展开为例：

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

（2）对数函数的泰勒级数展开，以自然对数函数（ln）为例：

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

这里有几点注意事项：

I、 每次计算需要足够多的项来获得准确的结果，这可能会导致计算复杂度较高。

II、级数展开只在某些范围内有效，超出这些范围可能会导致较大的误差。

III、 因为计算泰勒级数通常需要包括乘法和除法在内的多种算术运算。但是，我们这台计算机要用只能实现加法和减法，而泰勒级数计算需要用到乘法、除法和乘方。这两种运算需要用加法和减法模拟。所以理论上讲，这可能会极其低效且复杂。实际上，在现代计算机架构中，我们会利用硬件支持的高效算术运算单元来快速准确地计算这些函数，而不是依赖于纯加减法的模拟。

2.2.4 **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）。

通过硬件增加电路还是通过软件指令组合来实现特定功能，这两种方法各有优劣，我们在决定时可以从以下方面入手：

1. 性能：考虑哪种方法可以提供更好的性能。增加硬件电路可能会导致更快的执行速度，而通过软件实现可能会受到处理器性能和指令执行速度的限制。

2. 成本与复杂度：硬件设计和生产可能需要昂贵的资金，设计和验证也会增加项目的复杂度。而软件实现则相对灵活，只需要修改代码即可实现功能变更。

3. 可编程性和灵活性：考虑功能的需求是否可能会变化。软件实现更容易升级和修改，可以根据需求变化快速调整功能。硬件一旦设计制造完成，改变其功能的成本较高。

4. 能耗：硬件电路可能具有更低的能耗，特别是在处理大量数据或高性能计算场景下，定制化的硬件电路相比通用处理器执行相同任务可能更为节能。

5. **处理器架构**：我们以 RISC（精简指令集计算机）和 CISC（复杂指令集计算机）为例子。

- RISC 处理器通常采用简单的指令集，更倾向于通过已有指令的组合来实现功能。这是因为 RISC 架构的设计理念是精简指令集，更注重指令的简洁和执行效率，而不是增加复杂的硬件电路。通过已有指令的组合，可以灵活地实现新功能，并且不会增加处理器的复杂性和成本。
- CISC 处理器通常具有复杂的指令集，其中包含了许多功能丰富的指令。设计思想是尽量在单个指令中完成更多的功能，以减少程序员编写程序时的工作量，并且减少程序执行时的指令数目。因此增加一些额外的电路可能会更加符合其设计理念。

总结

2.3 实验中出现的问题

【0. 环境配置】

1) 安装虚拟机、配置主机、安装 ubuntu 操作系统很容易出错，哪怕跟着视频一步一步走也回车出现不清楚的报错，有时候谷歌出来的解决方案在别人电脑上能解决，在我电脑上就解决不了。所以得一遍遍卸载重装，一遍遍尝试不同的解决方案。

2) 第一次接触 Linux（Ubuntu）操作系统，面对全是命令行的操作模式很不习惯，各种指令使用也不熟练。但是用上两三天之后就好了很多。

3) vspm 的 jdk 版本要求 jdk21，jdk21 只有 64 位版本，但是刚上课的时候老师说尽量让我们安装 32 位 Ubuntu 和课本上一样，导致花了很多时间在安装 jdk 上，又下载了不同版本 64 位 Ubuntu 尝试。塞翁失马焉知非福，在这过程中我对 Ubuntu 的使用越来越熟练。

4) 下载好 jdk 尝试运行./vspm ，提醒没有权限。本来以为要用管理员权限 sudo，但是还是不行，最后摸索出是文件的权限。

【1. abc 文件的调试】

1) 对于指令集的指令不熟悉，每次遇到一个指令就要去查表。

2) 对于寄存器的定位不明确。R3 是存储地址，遇到[jg]和[jmp]指令将 R3 中的地址

赋给 PC；R1、R2 一般是存储内存数据段的数据；R0 的功能大多是被赋值内存地址，通过[movb]、[movc]从内存送出或送入数据，或者被赋值常数，通过[add][sub]对其它寄存器中的值进行更改。

3) 每个文件第一次调试的时候，对于程序所要完成的任务都是连蒙带猜，有时候调试完一遍了都不知道程序在干什么。

【2. 思考问题】

1) 在写除法的代码时，由于前三个文件，形成 “[jg]必须是跳转开始下一个循环”的思维定式。但是“[jg]必须是跳转开始下一个循环”的前提条件是，循环条件是“单边不含等号”，也即是类似于“a < b”而不能是“a <= b”，而我的算法中循环条件正是后者，所以[jg]被我当成“break”，而[jmp]才是跳转开始下一个循环。

2) 在调试完前三个文件后，只对其中包含的指令熟悉，对[mova][jmp]不熟悉，在写除法的代码时就一直遇到问题，直到把这两个不熟悉的指令加进去才解决。

2.4 心得体会

【真实感受】

1) 这门课入门的历程，让我想起来大一时配置 C++环境的艰难。面对几乎陌生的操作系统，要做好硬盘管理，跟着视频一个字符一个字符敲代码也会报错，报错看不懂不知道怎么改。甚至下载了 vscode，尝试用 cmake 工具，对这一大堆 json 文件和命令行报错不知所措。

2) 这段时间，我熟悉了 ubuntu 系统，对于各种命令都逐渐掌握。调试汇编代码，对冯诺依曼体系有了全新的认识。自己上手写了一份汇编代码，实现了正整数除法，这对我理解汇编指令有很大帮助。

3) 编写模型机真的很有意思，老师们如果可以，希望把其放到 github 上，让每一届学生不断完善。

4) 思考题中提到了“可计算理论”，以及图灵机和图灵完备的知识，我感觉这十分有趣，如果可以，希望老师们上课讲讲。