《计算机系统》

原型机实验报告

班级：计科 210X

学号：202108010XXX

姓名：甘晴void

目录

1 实验项目一

1.1 项目名称

原型机 I

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。

（4） 理解计算机的运行过程，对指令集进行修改

1.3 实验资源

（1）教材中冯诺伊曼体系的相关内容；

（2）课程《最小系统与原型机 I》。

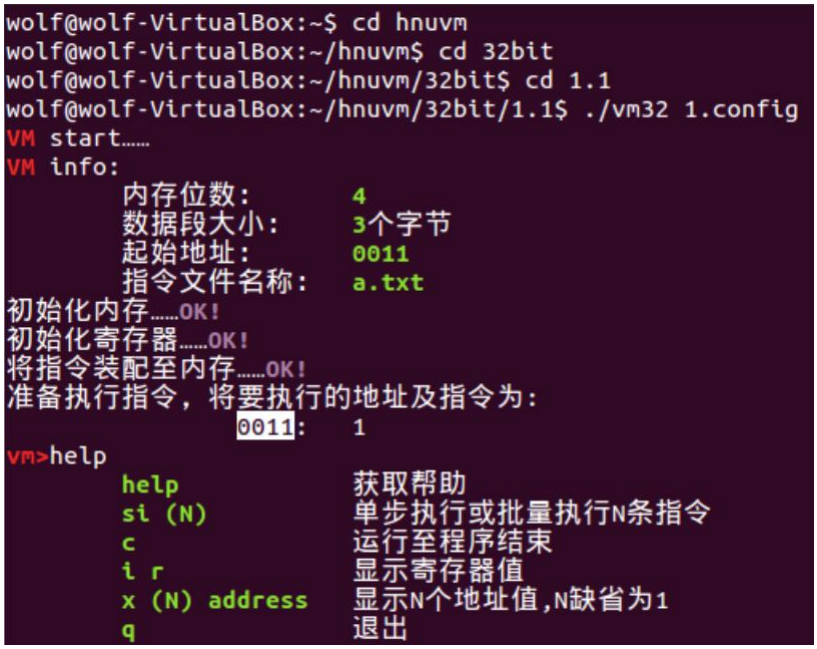
2 实验任务

2.1 原型机 I

2.1.1 练习内容

（1） 按照上述的实验步骤，完成相关操作；

①打开与初始化，认识帮助界面



②逐步运行至判断跳转指令，

指令 4 1 R2 将累加步长 1 存放在 R2 内；

此后指令 2 R0 R1，这步每次将 R0 累加至 R1，其中 R1 存放累加结果，也就是最终结果；

指令 3 R2 R0 是计算下一个需要累加的数并存放在 R0 内。

```
vm>i r
        R0:     0       0x0
        R1:     0       0x0
        R2:     0       0x0
        R3:     0       0x0
        PC:     0011
vm>x 6 0000
        0000:   0
        0001:   0
        0010:   0
        0011:   1
        0100:   5 R0 0000
        0101:   4 1 R2
vm>si
5
                0100:   5 R0 0000
vm>i r
        R0:     5       0x5
        R1:     0       0x0
        R2:     0       0x0
        R3:     0       0x0
        PC:     0100
vm>si
        R3:     0       0x0
        PC:     0100
vm>si
                0101:   4 1 R2
vm>i r
        R0:     5       0x5
        R1:     0       0x0
        R2:     0       0x0
        R3:     0       0x0
        PC:     0101
vm>x 8 0000
        0000:   00000101
        0001:   0
        0010:   0
        0011:   1
        0100:   5 R0 0000
        0101:   4 1 R2
        0110:   2 R0 R1
        0111:   3 R2 R0
vm>si
                0110:   2 R0 R1
vm>i r
        R0:     5       0x5
        R1:     0       0x0
        R2:     1       0x1
        R3:     0       0x0
        PC:     0110
vm>si
                0111:   3 R2 R0
vm>i r
        R0:     5       0x5
        R1:     5       0x5
        R2:     1       0x1
        R3:     0       0x0
        PC:     0111
vm>si
                1000:   6 -2
vm>i r
        R0:     4       0x4
        R1:     5       0x5
        R2:     1       0x1
        R3:     1       0x1
        PC:     1000
```

③判断跳转

由于上一步的减法运算结果不为 0，故 R3=1，满足跳转条件，程序回到之前重复累加计算。

```
                1000:   6 -2
vm>i r
        R0:     4       0x4
        R1:     5       0x5
        R2:     1       0x1
        R3:     1       0x1
        PC:     1000
vm>si
                0110:   2 R0 R1
vm>i r
        R0:     4       0x4
        R1:     5       0x5
        R2:     1       0x1
        R3:     1       0x1
        PC:     0110
```

④多次运行 si 3 跳过循环

```
vm>i r
        R0:     4       0x4
        R1:     5       0x5
        R2:     1       0x1
        R3:     1       0x1
        PC:     0110
vm>si 3
                0111:   3 R2 R0
                1000:   6 -2
                0110:   2 R0 R1
vm>i r
        R0:     3       0x3
        R1:     9       0x9
        R2:     1       0x1
        R3:     1       0x1
        PC:     0110
vm>si 3
                0111:   3 R2 R0
                1000:   6 -2
                0110:   2 R0 R1
vm>i r
        R0:     2       0x2
        R1:     12      0xc
        R2:     1       0x1
        R3:     1       0x1
        PC:     0110
vm>si 3
                0111:   3 R2 R0
                1000:   6 -2
                0110:   2 R0 R1
vm>i r
        R0:     1       0x1
        R1:     14      0xe
        R2:     1       0x1
        R3:     1       0x1
        PC:     0110
vm>si 3
                0111:   3 R2 R0
                1000:   6 -2
                1001:   5 R1 0001
vm>i r
        R0:     0       0x0
        R1:     15      0xf
        R2:     1       0x1
        R3:     0       0x0
        PC:     1001
```

⑤判断跳转跳出循环，输出结果

此时由于 R3=0，不符合跳转回循环的条件，故继续执行下一步，即跳出循环。

然后将结果传输回内存中，并输出结果。

```
                1001:   5 R1 0001
vm>i r
        R0:     0       0x0
        R1:     15      0xf
        R2:     1       0x1
        R3:     0       0x0
        PC:     1001
vm>si
                1010:   8 R1
vm>i r
        R0:     0       0x0
        R1:     15      0xf
        R2:     1       0x1
        R3:     0       0x0
        PC:     1010
vm>si
15
                1011:   0
vm>si
指令执行完成,程序退出！
```

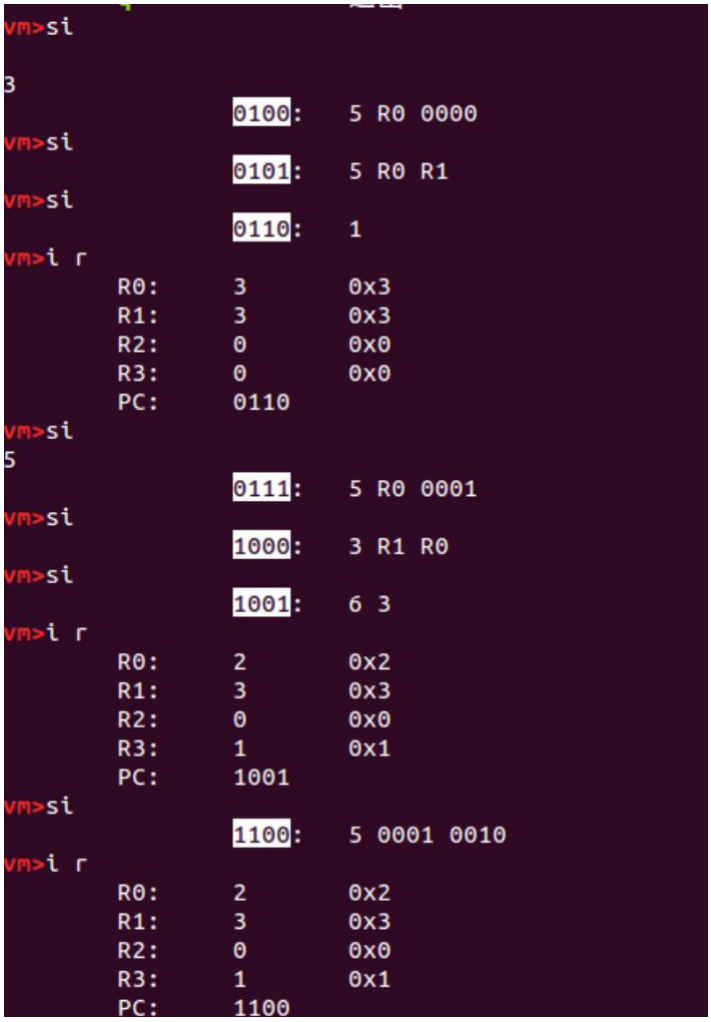
（2） 在目录下还有 **2.config，3.config**，其对应的指令代码文件分别为 **b.txt** 和 **c.txt**，请运行并调试，并对这些代码所做的工作进行解释；

Ⅰ对于 b.txt

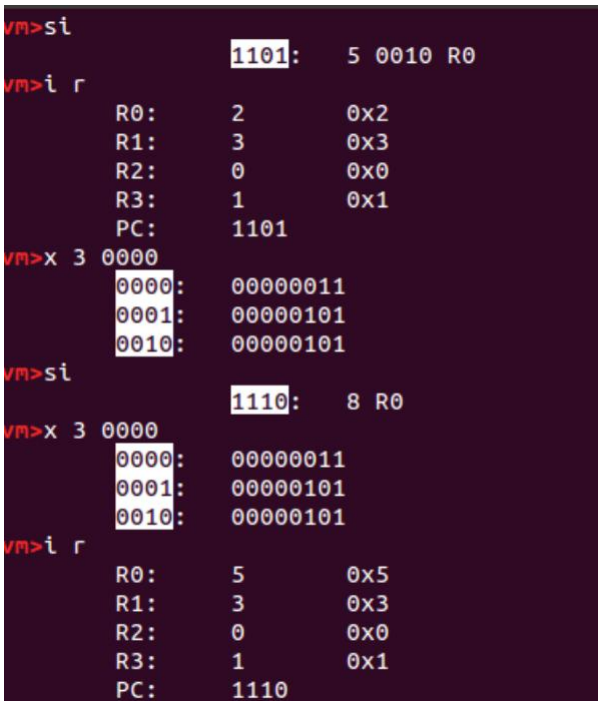
①运行调试

```
wolf@wolf-VirtualBox:~$ cd hnuvm
wolf@wolf-VirtualBox:~/hnuvm$ cd 32bit
wolf@wolf-VirtualBox:~/hnuvm/32bit$ cd 1.1
wolf@wolf-VirtualBox:~/hnuvm/32bit/1.1$ ./vm32 2.config
VM start……
VM info:
        内存位数:       4
        数据段大小:     3个字节
        起始地址:       0011
        指令文件名称:   b.txt
初始化内存……OK!
初始化寄存器……OK!
将指令装配至内存……OK!
准备执行指令，将要执行的地址及指令为:
                0011:   1
vm>help
        help            获取帮助
        si (N)          单步执行或批量执行N条指令
        c               运行至程序结束
        i r             显示寄存器值
        x (N) address   显示N个地址值,N缺省为1
        q               退出
vm>
```

范例 1：A=3，B=5



到上一步为止，进行的是 A 和 B 的比较并且已经比较出结果



这里是将较大结果输出的过程

范例 2：A=7，B=4

```
准备执行指令，将要执行的地址及指令为：
                0011:   1
vm>si
7
                0100:   5 R0 0000
vm>si
                0101:   5 R0 R1
vm>si
                0110:   1
vm>si
4
                0111:   5 R0 0001
vm>si
                1000:   3 R1 R0
vm>si
                1001:   6 3
vm>si
4
                0111:   5 R0 0001
vm>si
                1000:   3 R1 R0
vm>si
                1001:   6 3
vm>i r
        R0:     -3      0xfffffffd
        R1:     7       0x7
        R2:     0       0x0
        R3:     -1      0xffffffff
        PC:     1001
vm>si
                1010:   5 0000 0010
vm>x 3 0000
        0000:   00000111
        0001:   00000100
        0010:   0
vm>si
                1011:   7 2
vm>x 3 0000
        0000:   00000111
        0001:   00000100
        0010:   00000111
vm>si
                1101:   5 0010 R0
vm>si
                1110:   8 R0
vm>i r
        R0:     7       0x7
        R1:     7       0x7
        R2:     0       0x0
        R3:     -1      0xffffffff
        PC:     1110
vm>si
7
                1111:   0
vm>si
指令执行完成,程序退出！
vm>
```

②工作解释

先后输入两个数 A 和 B，比较 A 与 B 的大小并输出较大的数

下面给出逐步解释：

	指令	解释
1	1	输入 A，存储在 R0
2	5 R0 0000	将 A 存储在地址 0000
3	5 R0 R1	将 A 转移至 R1
4	1	输入 B
5	5 R0 0001	将 B 存储在 0001
6	3 R1 R0	A 与 B 作差，生成 R3
7	6 3	若 B>A,跳转至 10
8	5 0000 0010	A>B，将 A 传输给 0010
9	7 2	A>B，跳转至 11
10	5 0001 0010	A<B，将 B 传输给 0010
11	5 0010 R0	将 0010 传输给 R0，也就是较大的数
12	8 R0	输出结果
13	0	停机

Ⅱ对于 b.txt

①运行调试

```
wolf@wolf-VirtualBox:~/hnuvm/32bit/1.1$ ./vm32 3.config
VM start……
VM info:
        内存位数:       5
        数据段大小:     3个字节
        起始地址:       00011
        指令文件名称:   c.txt
初始化内存……OK!
初始化寄存器……OK!
将指令装配至内存……OK!
准备执行指令，将要执行的地址及指令为:
                00011:  4 0 R1
vm>
```

```
                00011:  1
vm>si
4
                00100:  5 R0 00000
vm>si
                00101:  1
vm>si
2
                00110:  5 00000 00001
vm>si
                00111:  5 00001 R1
vm>si
                01000:  3 R0 R1
vm>si
                01001:  6 2
vm>si
                01011:  5 00010 R2
vm>si
                01100:  4 1 R3
vm>si
                01101:  2 R3 R2
vm>si
                01110:  5 R2 00010
vm>si
                01111:  5 R1 00001
vm>si
                10000:  7 -9
vm>si
                00111:  5 00001 R1
vm>si
                01000:  3 R0 R1
vm>si
                01001:  6 2
vm>si
                01010:  7 7
vm>si
                10001:  5 00001 R1
vm>si
                10010:  5 R0 R2
vm>si
                10011:  3 R1 R2
vm>si
                10100:  6 5
vm>si
                10101:  5 00010 R2
vm>si
                10110:  4 1 R3
vm>si
                10111:  2 R3 R2
vm>si
                11000:  5 R2 00010
vm>si
                11001:  5 00010 R1
vm>si
                11010:  8 R1
vm>si
2
                11011:  0vm>si
指令执行完成,程序退出！
```

②工作解释

这段代码做的是一个**整除**的工作，通过不断模拟减法与计数来统计最终的结果

2.1.2 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?

①实现乘法：

用到 R0、R1、R2。对于 A*B 的计算，选定 A 与 B 的较小值作为计数量，较大值作为基础量。假定 A<B，那么原理就是将 B 自加 A 次。具体操作如下：

（由于只有三个寄存器可用，所以需要涉及到多次从内存中存取，较为繁琐，这里就不写代码了）

R0 作为基础量，R1 作为结果，R2 作为计数量。

R0 加到 R1 上；

R2 自减 1；

检验 R3（R2-1 是不是为 0），在 R3=0 时跳出循环

输出结果

②实现整除：

用到 R0、R1、R2。对于 A/B 的计算，原理就是多次将 A-B，但保证 A>0，计数结果即为答案。具体操作如下：

（由于只有三个寄存器可用，所以需要涉及到多次从内存中存取，较为繁琐，这里就不写代码了）

循环内

R0 为 A，R1 为 B，R2 作为计数量。

3 R1 R0；

若 R3 跳转至：R2 自加 1 并继续循环；

否则跳转结束并输出 R2

(2) 原型机 I 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?

原型机 I 的指令集是不完备的，

比较浅薄地观察可以发现，缺少了一些基本的逻辑指令，比如按位运算指令的按位与、按位或、按位非、求补、求反这些都无法完成。

2.2 原型机 II-扩充指令集

2.2.1 练习内容

（1） 按照上述的实验步骤，完成相关操作；

①增添代码段

```
void ExecuteMul(char source[],char dest[],int *result)
{
        char op;
        if(0==strcmp(source,"R0"))
                op=R0;
        else if( 0==strcmp(source,"R1"))
                op=R1;
        else if (0==strcmp(source,"R2"))
                op=R2;
        else if(0==strcmp(source,"R3"))
                op=R3;
        else
                *result=-1;
        if (0==strcmp(dest,"R0\n"))
                R0*=op ;
        else if( 0==strcmp(dest,"R1\n"))
                R1*=op;
        else if( 0==strcmp(dest,"R2\n"))
                R2*=op ;
        else if(0==strcmp(dest,"R3\n"))
                R3*=op;
        else
                *result=-1;

}
```

在 ExecuteInstruction 增加一个判断分支如下

```
case '9':               //
        split(instruction_buffer," ",revbuf,&num);
        if(3>num)
                *result=-1;     //出错
        else
                ExecuteMul(revbuf[1],revbuf[2],result);
        if(*result!=-1) *result=9;
        PC++;
        break;
```

②增加一个 d.txt 文件，其中包括有乘法指令并修改 1.config 为

4

3

0011

d.txt

③使用 make 生成可执行文件，

④执行代码，操作过程如下

```
VM info:
        内存位数:       4
        数据段大小:     3个字节
        起始地址:       0011
        指令文件名称:   d.txt
初始化内存……OK!
初始化寄存器……OK!
将指令装配至内存……OK!
准备执行指令，将要执行的地址及指令为:
                0011:   1
VM>si
4
                0100:   5 R0 R1
VM>si
                0101:   1
VM>si
5
                0110:   9 R0 R1
VM>si
                0111:   8 R1
VM>si
20
                1000:   0
VM>
```

⑤增加一个 e.txt 文件，基于原型机 I 的指令完成两个数的乘法操作，

这里为了节省空间，只演示 3*2 的运算过程

```
HNUVM start……
VM info:
        内存位数:       4
        数据段大小:     3个字节
        起始地址:       0011
        指令文件名称:   e.txt
初始化内存……OK!
初始化寄存器……OK!
将指令装配至内存……OK!
准备执行指令，将要执行的地址及指令为:
                0011:   1
VM>si
3
                0100:   5 R0 0000
VM>si
                0101:   1
VM>si
2
                0110:   4 1 R2
VM>si
                0111:   5 0000 R3
VM>si
                1000:   2 R3 R1
VM>si
                1001:   3 R2 R0
VM>si
                1010:   6 -3
VM>si
                0111:   5 0000 R3
VM>si
                1000:   2 R3 R1
VM>si
                1001:   3 R2 R0
VM>si
                1010:   6 -3
VM>si
                1011:   8 R1
VM>si
6
                1100:   0
VM>
```

(2) 为原型机 II 增加整除指令，并基于原型机 I 的指令写出两数整除的代码，进行对比；

Ⅰ为原型机 II 增加整除指令

①增添代码段

```
void ExecuteDiv(char source[],char dest[],int *result)
{
        char op;
        if(0==strcmp(source,"R0"))
                op=R0;
        else if(0==strcmp(source,"R1"))
                op=R1;
        else if(0==strcmp(source,"R2"))
                op=R2;
        else if(0==strcmp(source,"R3"))
                op=R3;
        else
                *result=-1;
        if(0==strcmp(dest,"R0\n"))
                R0/=op;
        else if(0==strcmp(dest,"R1\n"))
                R1/=op;
        else if(0==strcmp(dest,"R2\n"))
                R2/=op;
        else if(0==strcmp(dest,"R3\n"))
                R3/=op;
        else
                *result=-1;

}
```

在 ExecuteInstruction 增加一个判断分支如下

```
case 'A':               //
        split(instruction_buffer," ",revbuf,&num);
        if(3>num)
                *result=-1;     //出错
        else
                ExecuteDiv(revbuf[1],revbuf[2],result);
        if(*result!=-1) *result=9;
        PC++;
        break;
```

②增加一个 f.txt 文件，其中包括有除法指令并创建 4.config

f.txt

```
1
5 R0 R1
1
A R0 R1
8 R1
0
```

4.config

```
4
3
0011
f.txt
```

③使用 make 生成可执行文件，

④执行代码，操作过程如下

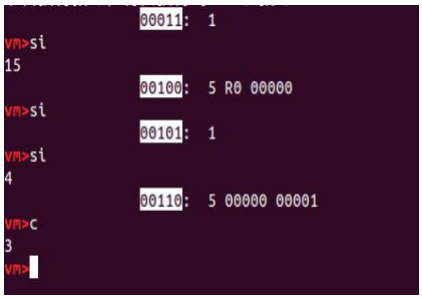
Ⅱ基于原型机 I 的指令写出两数整除的代码

代码及解读如下

1	1	读入被除数 A
2	5 R0 00000	将 A 存入 00000
3	1	读入除数 B（一直保存在 R0 中）
4	5 00000 00001	将 A 存入 00001 （此后 00001 用来保存 A 被减去 n 次后剩下的数 M）
5	5 00001 R1	将 M 存入 R1
6	3 R0 R1	M 减去 B，结果存入 R1
7	6 2	若 M>B（即还能继续减），跳转至 9（继续循环）
8	7 7	若 M<=B（即不够减了），跳转至 15（跳出循环）
9	5 00010 R2	00010 中的值存入 R2（取出 ans）
10	4 1 R3	
11	2 R3 R2	ans 自加 1（即 M-B 成功了）
12	5 R2 00010	ans 存入 00010

13	5 R1 00001	(M-B)存入 00001，成为新的 M
14	7 -9	跳转至 5
15	5 00001 R1	将 M 存入 R1
16	5 R0 R2	将 B 存入 R2
17	3 R1 R2	B 减去 M，结果保存在 R0 中
18	6 5	若 M<B（即不能继续减了，直接输出 ans 即可）跳转至 23
19	5 00010 R2	若 M>=B，即还可以再减一次，00010 存入 R2
20	4 1 R3	
21	2 R3 R2	ans 自加 1
22	5 R2 00010	将 R2 存入 00010
23	5 00010 R1	将 00010 存入 R1
24	8 R1	输出 R1
25	0	结束

运行效果：



对比：显然使用模块化的处理会比自己编写程序更为方便，但是使用逐步编写程序的方法也是可以将这个功能实现的。

2.2.2 思考问题

（1）原型机 I 与原型机 II 完成乘法和除法操作的方式有何不同?

原型机 I 需要使用现有指令集去编写实现乘法与除法的功能；

而原型机 II 将乘法与除法操作写入了指令集中，可以直接调用指令完成功能。

（2）在指令集中增加乘法、除法等指令时，原型机中需要增加代码，那么硬件实现上需要增加什么样的部件?

需要增加一个移位寄存器，可以实现移位操作，进而实现乘法与除法的功能。

（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）

可以计算三角函数，

对三角函数与对数函数可以进行麦克劳林展开，可以在舍弃无穷小的情况下近似为只含四则运算的运算式，而由模型机 I 可知乘法与除法可由加法与减法来实现。

综上可知，该计算机可以计算三角函数与对数函数。

（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）。

CPU 从指令集的特点上可以分为两类：CISC 和 RISC。

RISC 是“精简指令运算集”，CISC 就是“复杂指令运算集”。RISC 的指令系统相对简单，它只要求硬件执行很有限且最常用的那部分指令，大部分复杂的操作则使用成熟的编译技术，由简单指令合成。关于 RISC 与 CISC 的比较如下

1)RISC 更能充分利用 VLSI 芯片的面积。CISC 的控制器大多采用微程序控制，其控制存储器在 CPU 芯片内所占的面积为 50%以上，而 RISC 控制器采用组合逻辑控制，其硬布线逻辑只占 CPU 芯片面积的 10%左右。

2)RISC 更能提高运算速度。RISC 的指令数、寻址方式和指令格式种类少，又设有多个通用寄存器，采用流水线技术，所以运算速度更快，大多数指令在一个时钟周期内完成。

3)RISC 便于设计，可降低成本，提高可靠性。RISC 指令系统简单，故机器设计周期

短；其逻辑简单，故可靠性高。

4)RISC 有利于编译程序代码优化。RISC 指令类型少，寻址方式少，使编译程序容易选择更有效的指令和寻址方式，并适当地调整指令顺序，使得代码执行更高效化。

3 总结

3.1 实验中出现的问题

在 1.1 任务中，出现的问题有

（1）对于指令集的掌握不够牢固，在阅读学习汇编代码的时候需要经常性地查阅指令集来猜测汇编代码的意思，但是通过阅读多个汇编代码并多次练习，已经有所掌握与改善。

（2）对于汇编代码的含义理解不够到位，由于是第一次阅读汇编代码，对于较为简短的程序还好理解，比方说 a.txt 和 b.txt，但是对于较为长的代码就不是很好理解，比如说 c.txt，我还是结合程序运行的情况再加上一些自己的猜测才能大致猜测出写的是一个整除的程序。

（3）对于 ubuntu 的操作还不是很熟练。

在 1.2 任务中，出现的问题有

（1）直接从老师给的说明文档上复制下来的汇编代码中间有“，”，导致运行失败，这个花费了许多时间才纠正。

（2）老师给的说明文档上的 ExecuteInstruction 与模型机上的不配套，需要自行更改为 case 语句以配套，这一点需要小心。

3.2 心得体会

由于上学期数电的学习，对于模型机我有一定的基础，因此第一个实验还是比较好上手的。但是 ubuntu 这个崭新的环境还是给我的模型机实验带来了一些挑战。所幸按照老师给的说明文档所铺设的梯度，在一次一次的使用 i r 查看寄存器值与使用 x 3 0000 查看内存地址的值的情况下，我对于汇编程序的思想与结构还是有了更为深入的了解。

总的感受就是，这是一门有趣的课程，但是需要花费时间与精力才能学习好。这是一门好玩的学科，也值得花费更多尝试。