《计算机系统》

原型机实验报告

班级: 信安 2101 班

学号: 202109070105

姓名: 孙照海

目录

1	实验项目一		
	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		
	2.1	实验任务 A	4
	2.2	实验任务 B	8
3	总结		
	3.1	实验中出现的问题	24
	3.2	心得体会	24
4	实验项目二		
	4.1	项目名称	25
	4.2	实验目的	25
	4.3	实验资源	25
5	实验任务		
	5.1	实验任务 A	26
	5.2	实验任务 B	29
6	总结		
	6.1	实验中出现的问题	32
	6.2	心得体会	32

1 实验项目一

1.1 项目名称

实验 1.1 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。

1.3 实验资源

- (1) 阅读教材,掌握冯诺伊曼体系的相关内容;
- (2) 学习课程《最小系统与原型机 I》。

2 实验任务

2.1 实验任务 A

任务名称:对 2.config 及其所对应的 b.txt 进行调试

(1) 运行后界面如下图所示。

```
      szh@ubuntu:~/hnuvm-code/hnuvm/64bit/1.1$ ./vm64 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.....0K!

      初始化寄存器.....0K!

      将指令装配至内存.....0K!

      准备执行指令,将要执行的地址及指令为:

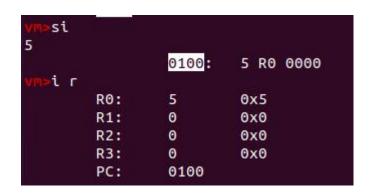
      0011: 1
```

(2) 此时输入 i r ,可以查看各个寄存器的值,而输入 x 6 0000 则表示查看从 0000 开始的连续 6 个内存地址值,结果如图所示。

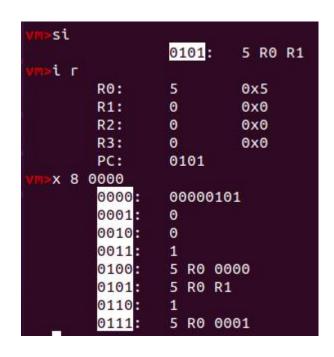
```
iг
     R0:
              0
                       0x0
     R1:
              0
                       0x0
     R2:
              0
                       0x0
     R3:
              0
                       0x0
     PC:
              0011
x 6 0000
     0000:
              0
     0001
              0
     0010
              0
     0011
              5 RO 0000
     0100:
     0101:
              5 R0 R1
```

(3) 输入 si,则表示运行一条指令,例如此时运行的指令是"1",表示等待输入,输入一个值 5,在输入完成后将此数值存至 R0 寄存器,运行完成后,再运行 i r 指令,就可以看到输入的值 5 确实是已经存在 R0 寄存器中,每个寄存器的值都用十进制和十六进制表

示,如下图所示。



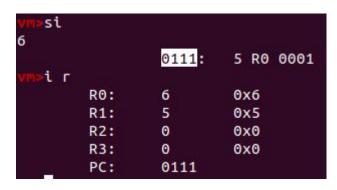
(4) 继续使用 si 指令单步执行指令 5 R0 0000, 表示将 R0 寄存器中的值 5 传送到内存地址 0000 处。



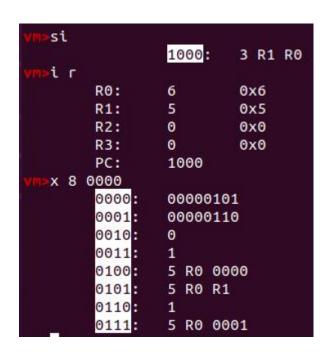
(5) **继续使用** si **指令单步执行指令** 5 R0 R1, **表示将** R0 **寄存器中的值** 5 **传送到** R1 **寄存器中。**

vm>si	(S) (S)		
NAME OF TAXABLE PARTY.		0110	1
vm>i r			
	RO:	5	0x5
	R1:	5	0x5
	R2:	0	0x0
	R3:	0	0x0
	PC:	0110	

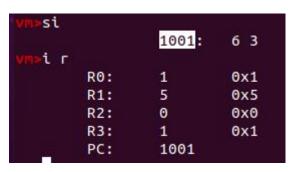
(6) 继续使用 si 指令单步执行指令 1, 表示需要输入一个整数并将其数值存在 RO 寄存器中。



(7) 继续使用 si 指令单步执行指令 5 R0 0001, 表示将 R0 寄存器中的值 6 传送到内存地址 0001 处。



(8) 继续 si 执行下一条指令 3 R1 R0,表示将 R0 的值减去 R1 的值,结果放于 R0中,注意观察寄存器 R0 值与 R3 值的变化情况。(当结果大于 0 时,R3 中赋值为 1)



(9) 继续 si 执行下一条指令 6 3,即有条件跳转,如果 R3 的值为 1,则需要向前或向后跳转,此时跳转的值为 3,则表示需要向后跳转三条指令。执行完 6 3 这一操作后, PC 值发生变化,在下图中观察到了 PC 值的改变。

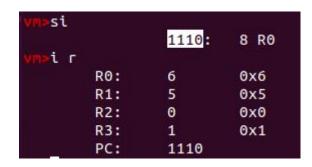
```
iг
     R0:
              1
                       0x1
     R1:
              5
                       0x5
     R2:
              0
                       0x0
     R3:
              1
                       0x1
     PC:
              1001
si
              1100:
                       5 0001 0010
ir
     R0:
              1
                       0x1
     R1:
              5
                       0x5
     R2:
              0
                       0x0
     R3:
              1
                       0x1
              1100
     PC:
```

(10) 继续使用 si 指令单步执行指令 5 0001 0010, 表示将内存地址 0001 中的值 6 传送到内存地址 0010 处。

```
x 8 0000
     0000:
              00000101
     0001:
              00000110
     0010:
              0
     0011:
              1
     0100:
              5 RO 0000
              5 R0 R1
     0101:
     0110:
              1
     0111:
              5 R0 0001
si
              1101: 5 0010 R0
x 8 0000
     0000:
              00000101
     0001:
              00000110
     0010:
              00000110
     0011:
     0100:
              5 RO 0000
     0101:
              5 R0 R1
     0110:
              1
              5 R0 0001
     0111:
```

(11) 继续使用 si 指令单步执行指令 5 0010 R0, 表示将内存地

址 0001 中的值 6 传送到 RO 寄存器中。



(12) 最后我们使用了8 RO 来输出最后结果,因此会打印出6这一结果。



(13) 程序执行完毕后,可以使用 q 退出。

```
vm>si
指令执行完成,程序退出!
vm>q
szh@ubuntu:~/hnuvm-code/hnuvm/64bit/1.1$
```

2.2 实验任务 B

任务名称:对 3.config 及其所对应的 c.txt 进行调试

(1) 运行后界面如下图所示。

```
WM start......

WM 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.....0K!

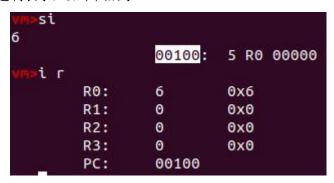
初始化寄存器.....0K!

将指令装配至内存.....0K!

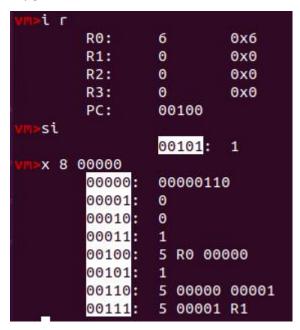
准备执行指令,将要执行的地址及指令为:

00011: 1
```

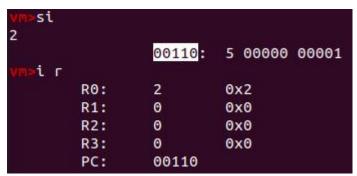
(2) 输入 si,则表示运行一条指令,例如此时运行的指令是"1",表示等待输入,输入 一个值 6,在输入完成后将此数值存至 R0 寄存器,运行完成后,再运行 ir 指令, 就可以看到输入的值 6 确实是已经存在 R0 寄存器中,每个寄存器的值都用十进制和十六进制表示,如下图所示。



(3) 继续使用 si 指令单步执行指令 5 R0 00000,表示将 R0 寄存器中的值 6 传送到内存地址 00000 处。

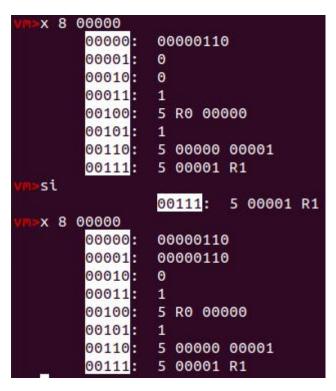


(4) 继续输入 si 运行的指令"1",表示等待输入,输入一个值 2,在输入完成后将此数值存至 R0 寄存器,运行完成后,再运行 ir指令,就可以看到输入的值 2 确实是已经存在 R0 寄存器中,每个寄存器的值都用十进制和十六进制表示,如下图所示。

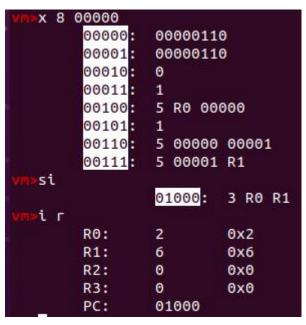


(5) 继续使用 si 指令单步执行指令 5 00000 00001, 表示将内存地址 00000 处的值

6 传送到内存地址 00001 处。



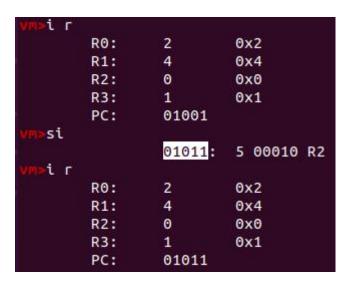
(6) 继续使用 si 指令单步执行指令 5 00001 R1, 表示将内存地址 00001 处的值 6 传送到 R1 寄存器中。



(7) 继续 si 执行下一条指令 3 R0 R1,表示将 R1 的值减去 R0 的值,结果放于 R1 中,注意观察寄存器 R1 值与 R3 值的变化情况。

```
iг
               2
      R0:
                        0x2
      R1:
               6
                        0х6
      R2:
               0
                        0x0
      R3:
               0
                        0x0
      PC:
               01000
si
               01001:
                        6 2
iг
     R0:
               2
                        0x2
               4
                        0x4
     R1:
      R2:
               0
                        0x0
      R3:
               1
                        0x1
      PC:
               01001
```

(8) 继续 si 执行下一条指令 6 2,即有条件跳转,如果 R3 的值为 1,则需要向前或向后跳转,此时跳转的值为 2,则表示需要向后跳转两条指令。执行完 6 2 这一操作后,PC 值发生变化,在下图中观察到了 PC 值的改变。



(9) 继续使用 si 指令单步执行指令 5 00010 R2, 表示将内存地址 00010 处的值 0 传送到 R2 寄存器中。

```
x 8 00000
     00000:
              00000110
     00001:
              00000110
     00010:
              0
     00011:
              1
     00100:
              5 RO 00000
     00101:
     00110:
              5 00000 00001
     00111:
              5 00001 R1
si
              01100:
                      4 1 R3
iг
     RO:
              2
                       0x2
     R1:
              4
                       0x4
     R2:
              0
                       0x0
     R3:
              1
                       0x1
     PC:
              01100
```

(10) 继续 si 执行下一条指令 4 1 R3,表示将立即数 1 传送到寄存器 R3 中,结果如下图所示。

vm>si	01101:	2 R3 R2
vm>i r	-	
R0:	2	0x2
R1:	4	0x4
R2:	0	0x0
R3:	1	0x1
_ PC:	01101	

(11) 继续 si 执行下一条指令 2 R3 R2,表示将 R3 与 R2 的值相加,结果放于 R2 中,注意观察寄存器 R2 值的变化情况。

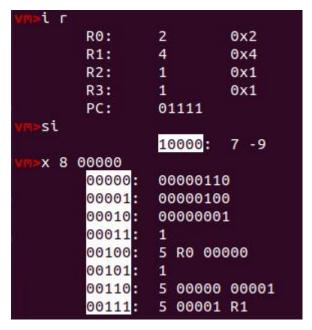
```
ir
              2
     R0:
                       0x2
     R1:
              4
                       0x4
     R2:
              0
                       0x0
     R3:
                       0x1
              1
     PC:
              01101
si
              01110
                     5 R2 00010
iг
              2
     R0:
                       0x2
     R1:
                       0x4
              4
     R2:
              1
                       0x1
                       0x1
     R3:
              1
     PC:
              01110
```

(12) 继续使用 si 指令单步执行指令 5 R2 00010,表示将 R2 寄存器中的值 1 传送到

内存地址 00010 处。

```
iг
                      0x2
     RO:
              2
     R1:
             4
                      0x4
     R2:
              1
                      0x1
     R3:
                      0x1
     PC:
             01110
si
             01111: 5 R1 00001
x 8 00000
     00000
             00000110
     00001:
             00000110
     00010
             00000001
     00011:
              1
     00100
             5 RO 00000
     00101:
             1
             5 00000 00001
     00110
     00111
            5 00001 R1
```

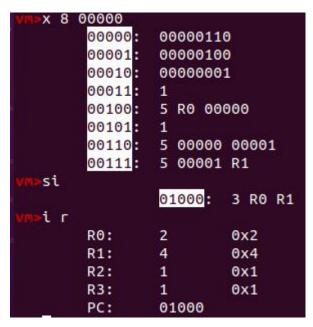
(13) 继续使用 si 指令单步执行指令 5 R1 00001,表示将 R1 寄存器中的值 4 传送到内存地址 00001 处。



(14) 继续使用 si 指令单步执行指令 7 -9, 需要向前或向后跳转, 此时跳转的值为-9, 则表示需要向前跳转九条指令。执行完 7 -9 这一操作后, PC 值发生变化, 在下图中观察到了 PC 值的改变。

```
iг
      RO:
               2
                        0x2
      R1:
               4
                         0x4
      R2:
               1
                         0x1
      R3:
               1
                         0x1
      PC:
               10000
si
                        5 00001 R1
               00111:
iг
      R0:
               2
                        0x2
      R1:
               4
                        0x4
      R2:
               1
                        0x1
      R3:
               1
                        0x1
      PC:
               00111
```

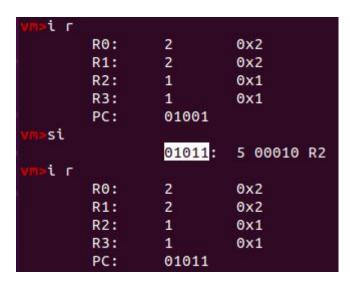
(15) 继续使用 si 指令单步执行指令 5 00001 R1, 表示将内存地址 00001 处的值 4 传送到 R1 寄存器中。



(16) 继续 si 执行下一条指令 3 R0 R1,表示将 R1 的值减去 R0 的值,结果放于 R1 中,注意观察寄存器 R1 值与 R3 值的变化情况。

```
R0:
               2
                        0x2
     R1:
               4
                        0x4
     R2:
               1
                        0x1
     R3:
                        0x1
               1
     PC:
               01000
si
               01001:
                        6 2
iг
     R0:
               2
                        0x2
               2
     R1:
                        0x2
     R2:
               1
                        0x1
     R3:
               1
                        0x1
     PC:
               01001
```

(17) 继续 si 执行下一条指令 6 2,即有条件跳转,如果 R3 的值为 1,则需要向前或向后跳转,此时跳转的值为 2,则表示需要向后跳转两条指令。执行完 6 2 这一操作后, PC 值发生变化,在下图中观察到了 PC 值的改变。



(18) 继续使用 si 指令单步执行指令 5 00010 R2, 表示将内存地址 00010 处的值 1 传送到 R2 寄存器中。

```
x 8 00000
     00000:
              00000110
              00000100
     00001
     00010:
              00000001
     00011:
     00100:
              5 R0 00000
     00101
     00110:
              5 00000 00001
     00111
              5 00001 R1
si
              01100:
                      4 1 R3
iг
     R0:
              2
                      0x2
     R1:
              2
                      0x2
     R2:
              1
                      0x1
     R3:
              1
                      0x1
     PC:
              01100
```

(19) 继续 si 执行下一条指令 4 1 R3,表示将立即数 1 传送到寄存器 R3 中,结果如下图所示。

vm>si vm>i r	01101	2 R3 R2
RO:	2	0x2
R1:	2	0x2
R2:	1	0x1
R3:	1	0x1
PC:	01101	

(20) 继续 si 执行下一条指令 2 R3 R2,表示将 R3 与 R2 的值相加,结果放于 R2 中,注意观察寄存器 R2 值的变化情况。

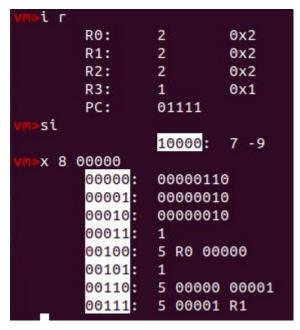
```
iг
              2
     RO:
                       0x2
     R1:
              2
                       0x2
     R2:
              1
                       0x1
     R3:
                       0x1
              1
     PC:
              01101
si
              01110:
                      5 R2 00010
iг
     R0:
              2
                       0x2
     R1:
              2
                       0x2
     R2:
              2
                       0x2
     R3:
              1
                       0x1
     PC:
              01110
```

(21) 继续使用 si 指令单步执行指令 5 R2 00010,表示将 R2 寄存器中的值 2 传送到

内存地址 00010 处。

```
R0:
              2
                       0x2
              2
                       0x2
     R1:
     R2:
              2
                       0x2
     R3:
              1
                       0x1
     PC:
              01110
si
              01111:
                       5 R1 00001
x 8 00000
     00000:
              00000110
     00001:
              00000100
     00010:
              00000010
     00011:
              1
     00100:
              5 RO 00000
     00101:
              1
              5 00000 00001
     00110:
     00111:
              5 00001 R1
```

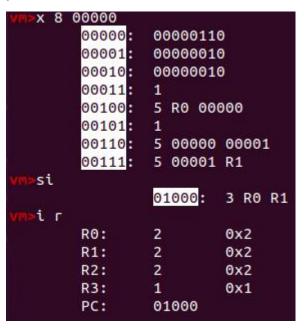
(22) 继续使用 si 指令单步执行指令 5 R1 00001,表示将 R1 寄存器中的值 2 传送到内存地址 00001 处。



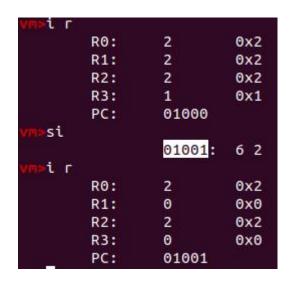
(23) 继续使用 si 指令单步执行指令 7 -9, 需要向前或向后跳转, 此时跳转的值为-9,则表示需要向前跳转九条指令。执行完 7 -9 这一操作后, PC 值发生变化, 在下图中观察到了 PC 值的改变。

```
ir
               2
      RO:
                        0x2
               2
      R1:
                        0x2
      R2:
               2
                        0x2
      R3:
               1
                        0x1
      PC:
               10000
si
               00111:
                        5 00001 R1
ir
      R0:
               2
                        0x2
               2
      R1:
                        0x2
      R2:
               2
                        0x2
      R3:
               1
                        0x1
               00111
      PC:
```

(24) 继续使用 si 指令单步执行指令 5 00001 R1, 表示将内存地址 00001 处的值 2 传送到 R1 寄存器中。



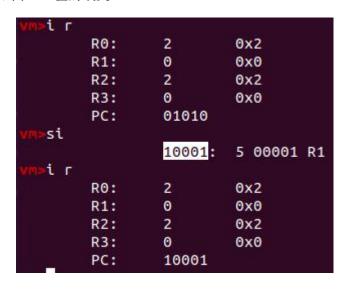
(25) 继续 si 执行下一条指令 3 R0 R1,表示将 R1 的值减去 R0 的值,结果放于 R1 中,注意观察寄存器 R1 值与 R3 值的变化情况。



(26) 继续 si 执行下一条指令 6 2,即有条件跳转,而此时 R3 的值为 0,会执行下一条指令



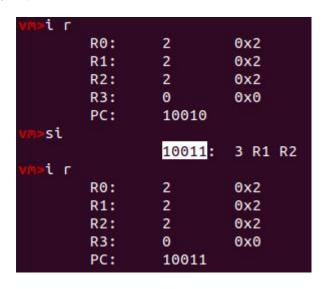
(27) 继续使用 si 指令单步执行指令 7 7, 需要向前或向后跳转,此时跳转的值为 7, 则表示需要向后跳转七条指令。执行完 7 7这一操作后, PC 值发生变化, 在下图中观察到了 PC 值的改变。



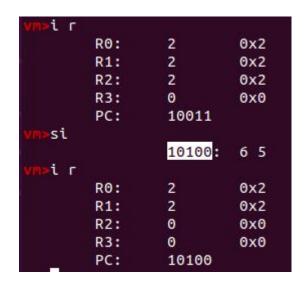
(28) 继续使用 si 指令单步执行指令 5 00001 R1, 表示将内存地址 00001 处的值 2 传送到 R1 寄存器中。

```
x 8 00000
     00000:
              00000110
     00001:
              00000010
     00010:
              00000010
     00011:
     00100:
              5 RO 00000
     00101:
     00110:
              5 00000 00001
     00111:
              5 00001 R1
si
              10010:
                       5 R0 R2
iг
     RO:
              2
                       0x2
     R1:
              2
                       0x2
              2
     R2:
                       0x2
     R3:
              0
                       0x0
     PC:
              10010
```

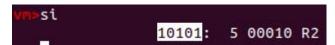
(29) 继续使用 si 指令单步执行指令 5 R0 R2, 表示将 R0 寄存器中的值 2 传送到 R2 寄存器中。



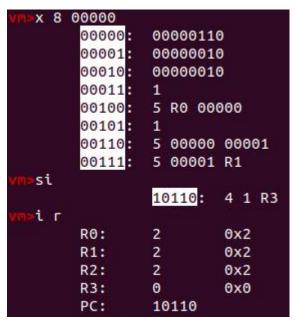
(30) 继续 si 执行下一条指令 3 R1 R2,表示将 R2 的值减去 R1 的值,结果放于 R2 中,注意观察寄存器 R2 值与 R3 值的变化情况。



(31) 继续 si 执行下一条指令 6 5,即有条件跳转,而此时 R3 的值为 0,会执行下一条指令



(32) 继续使用 si 指令单步执行指令 5 00010 R2, 表示将内存地址 00010 处的值 2 传送到 R2 寄存器中。



(33) 继续 si 执行下一条指令 4 1 R3,表示将立即数 1 传送到寄存器 R3 中,结果如下图所示,注意观察寄存器 R3 值的变化情况。

```
si
               10111:
                        2 R3 R2
     R0:
               2
                        0x2
     R1:
               2
                        0x2
     R2:
               2
                        0x2
     R3:
               1
                        0x1
     PC:
               10111
```

(34) 继续 si 执行下一条指令 2 R3 R2,表示将 R3 与 R2 的值相加,结果放于 R2 中,注意观察寄存器 R2 值的变化情况。

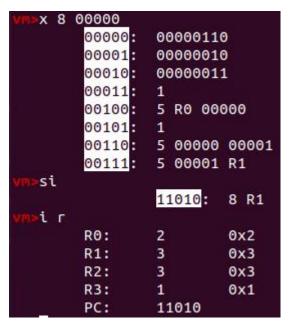
```
iг
              2
     R0:
                        0x2
              2
                        0x2
     R1:
     R2:
              2
                        0x2
     R3:
               1
                        0x1
     PC:
              10111
si
              11000:
                       5 R2 00010
iг
     R0:
              2
                        0x2
     R1:
              2
                        0x2
              3
     R2:
                        0x3
     R3:
                        0x1
     PC:
               11000
```

(35) 继续使用 si 指令单步执行指令 5 R2 00010,表示将 R2 寄存器中的值 3 传送到内存地址 00010 处。

```
ir
     RO:
              2
                      0x2
     R1:
              2
                       0x2
     R2:
              3
                       0x3
     R3:
              1
                       0x1
     PC:
              11000
si
              11001
                      5 00010 R1
x 8 00000
     00000:
              00000110
     00001
              00000010
     00010
              00000011
      00011
              1
      00100
              5 RO 00000
     00101:
     00110
              5 00000 00001
     00111:
              5 00001 R1
```

(36) 继续使用 si 指令单步执行指令 5 00010 R1,表示将内存地址 00010 处的值 3 传

送到 R1 寄存器中。



(37) 最后我们使用了8 R1来输出最后结果,因此会打印出3这一结果。



(38) 程序执行完毕后,可以使用 q 退出。

指令执行完成,程序退出! vm>q

问题思考:

(1) 乘法指令可通过加法指令实现,例如 a*b, 假设结果存在 sum 中, sum 初始化为 0, 可循环执行 sum+=a, 每执行一次 sum+=a 时 b--, 当 b 为 0 时则跳出,输出 sum

除法指令可通过减法指令实现,例如 a/b,假设结果存在 result 中,result 初始 化为 0,可循环执行 a=b,每执行一次 a=b 时 result++,当 a 为 0 时则跳出,输出 result

(2) 是完备的。在可计算理论中,当一组数据操作的规则(一组指令集,编程语言,或者元胞自动机)满足任意数据按照一定的顺序可以计算出结果,被称为图灵完备(turing complete)。一个有图灵完备指令集的设备被定义为通用计算机。

3 总结

3.1 实验中出现的问题

本实验较简单 无问题

3.2 心得体会

学习到了除法的机器执行过程, 有利于更好地了解计算机和编程

4 实验项目二

4.1 项目名称

实验 1.2 原型机 II-扩充指令集

4.2 实验目的

- 1) 理解指令集结构及其作用;
- 2) 理解计算机的运行过程,对指令集进行修改;

4.3 实验资源

- (1) 阅读教材,掌握冯诺伊曼体系的相关内容;
- (2) 学习《最小系统与原型机 I》内容,完成实验 1.1

5 实验任务

5.1 实验任务 A

任务名称:增加乘法指令

}

(1) 在原型机 I 的基础上,我们对指令集进行扩充,增加一条乘法指令,其格式为 9 Ra Rb,即将寄存器 Ra 的值与寄存器 Rb 的值相乘,结果放在 Rb 寄存器中,因此需要增加一个 ExecuteMul 函数。

```
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;
```

(2) 在 ExecuteInstruction 增加一个判断分支,从而能够识别此条指令。

(3) 输入 make 生成可执行文件

然后使用 ./vm32 1.config 来运行程序。

```
szh@ubuntu:~/hnuvm/32bit/1.2$ ./vm32 1.config
 M info:
         内存位数:
         数据段大小:
                           3个字节
         起始地址:
                           0011
         指令文件名称:
初始化内存......0K!
初始化寄存器......0K!
将指令装配至内存......0K!
准备执行指令,将<u>要执</u>行
               将要执行的地址及指令为:
                  0011:
                           1
 m>si
                  0100:
                           5 R0 R1
 m>si
                  0101:
                           1
 n>si
                  0110:
                           9 R0 R1
 m>si
                  0111:
                           8 R1
 m>si
20
                  1000:
                           0
```

(4) 增加一个 e.txt 文件,基于原型机 I 的指令完成了两个数的乘法操作,其基本思路 是将乘法分解为加法,例如对于 5*6,执行 6 次加 5 的操作: 5*6=5+5+5+5+5

```
szh@ubuntu:~/hnuvm/32bit/1.2$ ./vm32 2.config
      start.....
 M info:
         内存位数:
数据段大小:
起始地址:
                           3个字节
                           0011
         指令文件名称:
                           e.txt
初始化内存......0K!
初始化寄存器.....0K!
将指令装配至内存......0K!
准备执行指令,将要执行的地址及指令为:
                  0011
                           1
 m>si
5
                  0100:
                           5 RO 0000
 m>si
                  0101:
                           1
  1>si
                  0110
                           4 1 R2
 m>si
                  0111:
                           5 0000 R3
 m>si
                  1000:
                           2 R3 R1
 m>si
                  1001:
                           3 R2 R0
 m>si
                  1010:
                           6 -3
 m>si
                  0111:
                           5 0000 R3
 m>si
                  1000:
                           2 R3 R1
 m>si
                  1001:
                           3 R2 R0
 m>si
                  1010:
                           6 -3
 m>si
                  0111:
                           5 0000 R3
 m>si
                  1000:
                           2 R3 R1
 m>si
                  1001:
                           3 R2 R0
 m>si
                  1010:
                           6 -3
 m>si
                  0111:
                           5 0000 R3
 m>si
                  1000:
                           2 R3 R1
 m>si
                  1001:
                           3 R2 R0
 m>si
                  1010:
                           6 -3
 m>si
                  0111:
                           5 0000 R3
 m>si
                  1000:
                           2 R3 R1
 m>si
                  1001:
                           3 R2 R0
 m>si
                  1010:
                           6 -3
 m>si
                           8 R1
                  1011:
 m>si
30
                  1100:
                           0
```

5.2 实验任务 B

任务名称:增加整除指令

(1) 修改 3.config

4

3

0011

a.txt

(2) 修改 a.txt 文件, 其中包括有乘法指令

1

5 R0 R1

1

9 R0 R1

8 R1

0

(3) 在原型机 I 的基础上,我们对指令集进行扩充,增加一条除法指令,其格式为 9 Ra Rb,即将寄存器 Rb 的值与寄存器 Ra 的值相除,结果放在 Rb 寄存器中,因此需要增加一个 ExecuteDiv 函数。

```
void ExecuteDiv(char source[],char dest[],int *result)
 {
         char op;
         if(0==strcmp(source, "RO"))
                 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,从而能够识别此条指令。
(4)
case '9':
                        //除法
        split(instruction_buffer," ",revbuf,&num);
        if(3>num)
                               //出错
                *result=-1;
        else
                ExecuteDiv(revbuf[1],revbuf[2],result);
        if(*result!=-1) *result=2;
        PC++;
        break:
     输入 make 生成可执行文件
(5)
```

然后使用 ./vm32 1.config 来运行程序。

```
szh@ubuntu:~/hnuvm/32bit/1.2$ ./vm32 3.config
      start.....
/M 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:
                         a.txt
初始化内存.....ок!
初始化寄存器.....oк!
将指令装配至内存.....oк!
准备执行指令,将<u>要执</u>行的地址及指令为:
                 0011:
                         1
/m>si
                 0100:
                         5 R0 R1
vm>si
                 0101:
                         1
/m>si
                 0110:
                         9 R0 R1
/m>si
                 0111:
                         8 R1
m>si
                 1000:
                         0
```

问题思考:

- (1) 原型机 I 是通过加法实现乘法,通过减法实现除法 而原型机 II 则是通过编写的乘除法的 C 语言代码实现的乘除法操作
- (2) 算术逻辑单元 ALU
- (3) 可以。可以使用泰勒级数展开将三角函数,对数函数展开为多项式之和,即可 通过基本的加减法指令算出

6 总结

6.1 实验中出现的问题

删除 txt 文件后面的注释后, txt 文件的每一行结尾仍有空格, 如果不删去的话, 会报错如下

vm>si 指令执行出错,程序异常退出!

6.2 心得体会

了解了原型机实现原理,学会了用 C 语言编写基本乘除法指令,对原型机和计算机有了更为深入的了解