目 录

[1 CPU设计实验 2](#_Toc24744775)

[1.1 设计要求 2](#_Toc24744776)

[1.2 方案设计 3](#_Toc24744777)

[1.3 实验步骤 4](#_Toc24744778)

[1.4 故障与调试 4](#_Toc24744779)

[1.5 测试与分析 5](#_Toc24744780)

[2 总结与心得 7](#_Toc24744781)

[2.1 实验总结 7](#_Toc24744782)

[2.2 实验心得 7](#_Toc24744783)

[参考文献 8](#_Toc24744784)

# CPU设计实验

**请仔细阅读所有的批注，阅读理解后删除批注**

**模板各个标题下面的内容仅是举例，作者应依照自己思想重写该部分内容**

## 设计要求

（1）单周期MIPS CPU的设计

实验目的

掌握硬布线控制器设计的基本原理

利用相关原理在logisim平台上实现单周期MIPS CPU

设计出的CPU满足8条核心指令（如表1.1），能运行冒泡排序测试程序

主要内容

绘制单周期MIPS CPU的数据通路

实现单周期硬布线控制器（满足8条核心指令，如表1.1）

测试联调，运行冒泡排序程序

（2） 多周期MIPS CPU的设计

实验目的

理解微程序控制器设计的基本原理，并实现基于微程序控制器的多周期MIPS CPU

理解硬布线控制器设计的基本原理，并实现基于硬布线控制器的多周期MIPS CPU

设计出的CPU满足8条核心指令（如表1.1），能运行冒泡排序程序

主要内容

绘制多周期MIPS CPU的数据通路

实现多周期微程序控制器

测试基于微程序控制器的多周期MIPS CPU，运行冒泡排序程序

实现多周期硬布线控制器

测试基于硬布线控制器的多周期MIPS CPU，运行冒泡排序程序

表 1.1 MIPS实验 8条核心指令集

|  |  |  |
| --- | --- | --- |
| # | MIPS指令 | RTL功能描述 |
| 1 | add $rd,$rs,$rt | R[$rd]←R[$rs]+R[$rt] 溢出时候产生异常，不修改R[$rd] |
| 2 | slt $sd,$rs,$rt | R[$rd]←R[$rs]<R[$rt] 小于置1，有符号比较 |
| 3 | addi $rt,$rs,imm | R[$rt]←R[$rs]+SignExt16b(imm) |
| 4 | lw $rt,imm($rs) | R[$rt]←Mem4B(R[$rs]+SignExt16b(imm)) |
| 5 | sw $rt,imm($rs) | Mem4B[R[$rs]+SignExt16b­(imm)]←R[$rt] |
| 6 | beq $rs,$rt,imm | if(R[$rs]==R[$rt]) PC←PC+SignExt18b({imm,00}) |
| 7 | bne $rs,$rt,imm | if(R[$rs]!=R[$rt]) PC←PC+SignExt18b({imm,00}) |
| 8 | syscall | 系统调用，这里用于停机 |

## 数据通路的构建

### 单周期MIPS CPU的数据通路

单周期MIPS CPU包含的主要部件有：PC寄存器、RegFile寄存器组、ALU运算器、指令存储器、数据存储器、单周期硬布线控制器。

按照老师给出的结构框架，在logisim上将这些主要部件组装构成的数据通路如下图1.1所示：

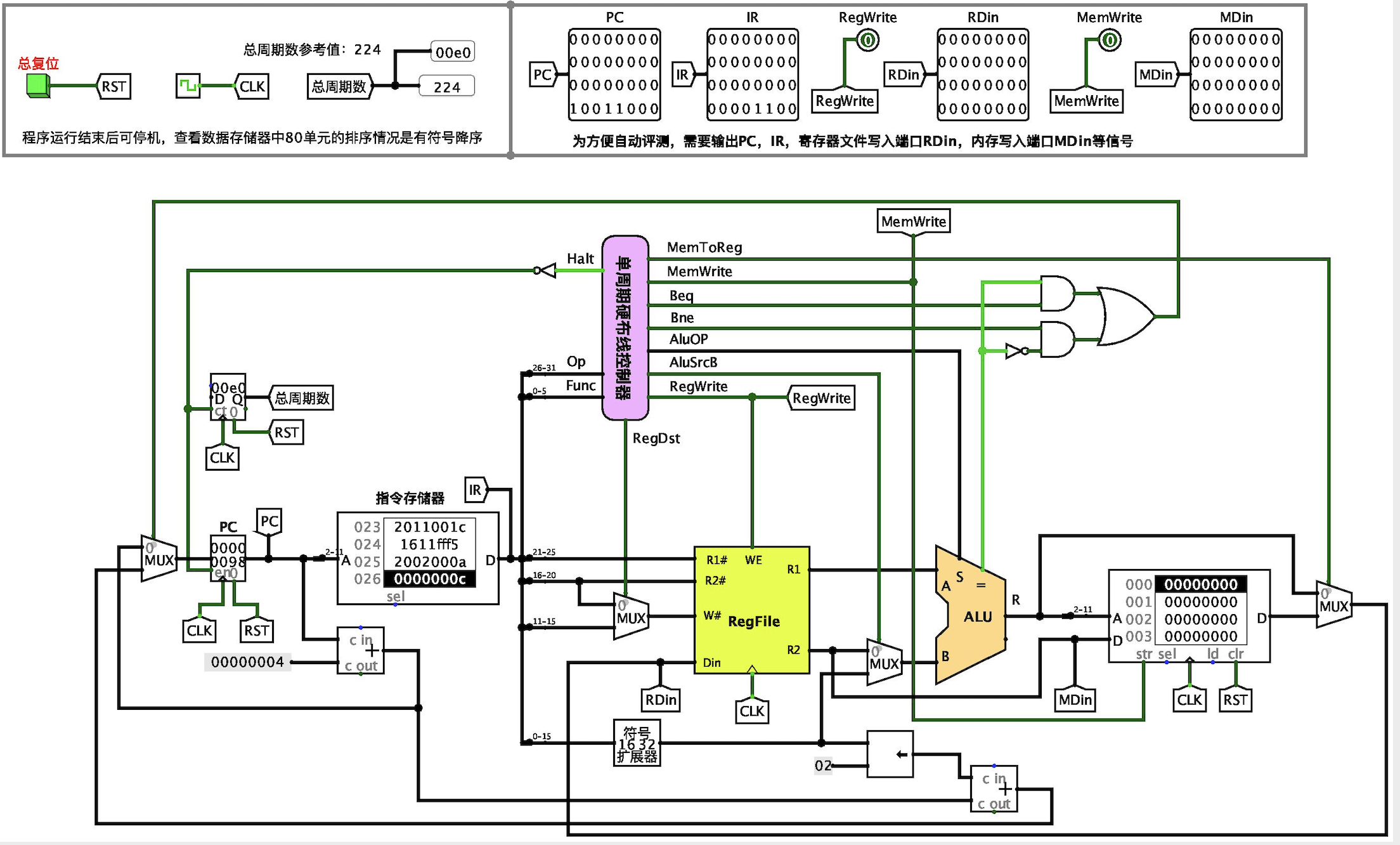


图 1.1 单周期MIPS CPU数据通路的logisim实现图

### 多周期MIPS CPU的数据通路

多周期MIPS CPU包含的主要部件有：PC寄存器、IR寄存器、DR寄存器、RegFile寄存器组、数据寄存器A和B，存储器Mem（包括指令存储器和数据存储器）、ALU运算器、多周期微程序/硬布线控制器。

按照老师给出的结构框架，在logisim平台上将这些部件连接起来数据通路如下图1.2所示：

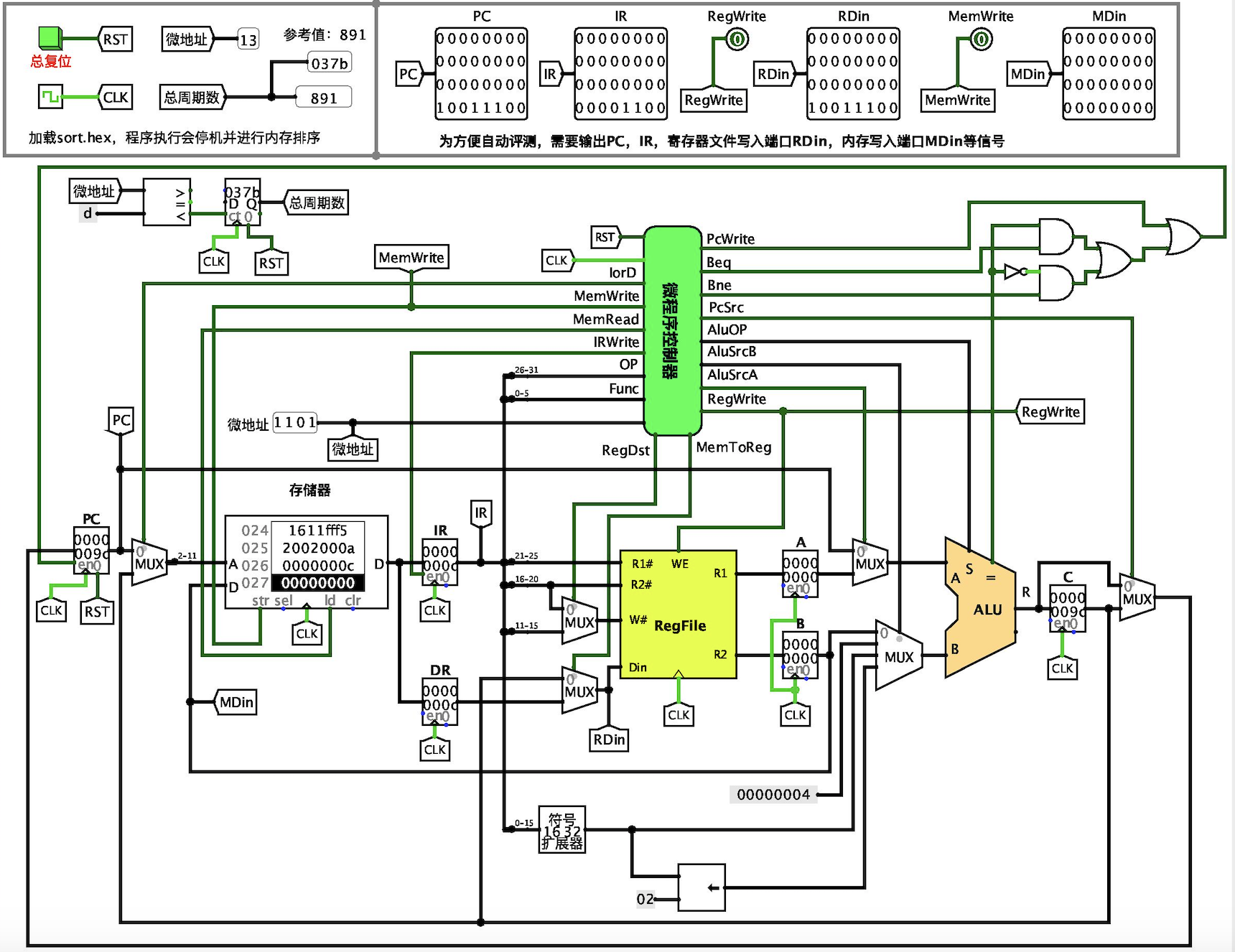


图 1.2 多周期MIPS CPU数据通路的logisim实现图

## 控制器设计

### 单周期MIPS CPU（硬布线）的控制器

1. 实现内部逻辑（指令译码和ALU控制）

完成指令译码的逻辑，需要将指令数据中分离出来的OP字段和FUNC字段和具体的指令对应起来；完成ALU控制逻辑，我们要用ALU\_Control的值决定运算器运算选择控制信号ALU\_OP的值，任务如下图1.3所示：

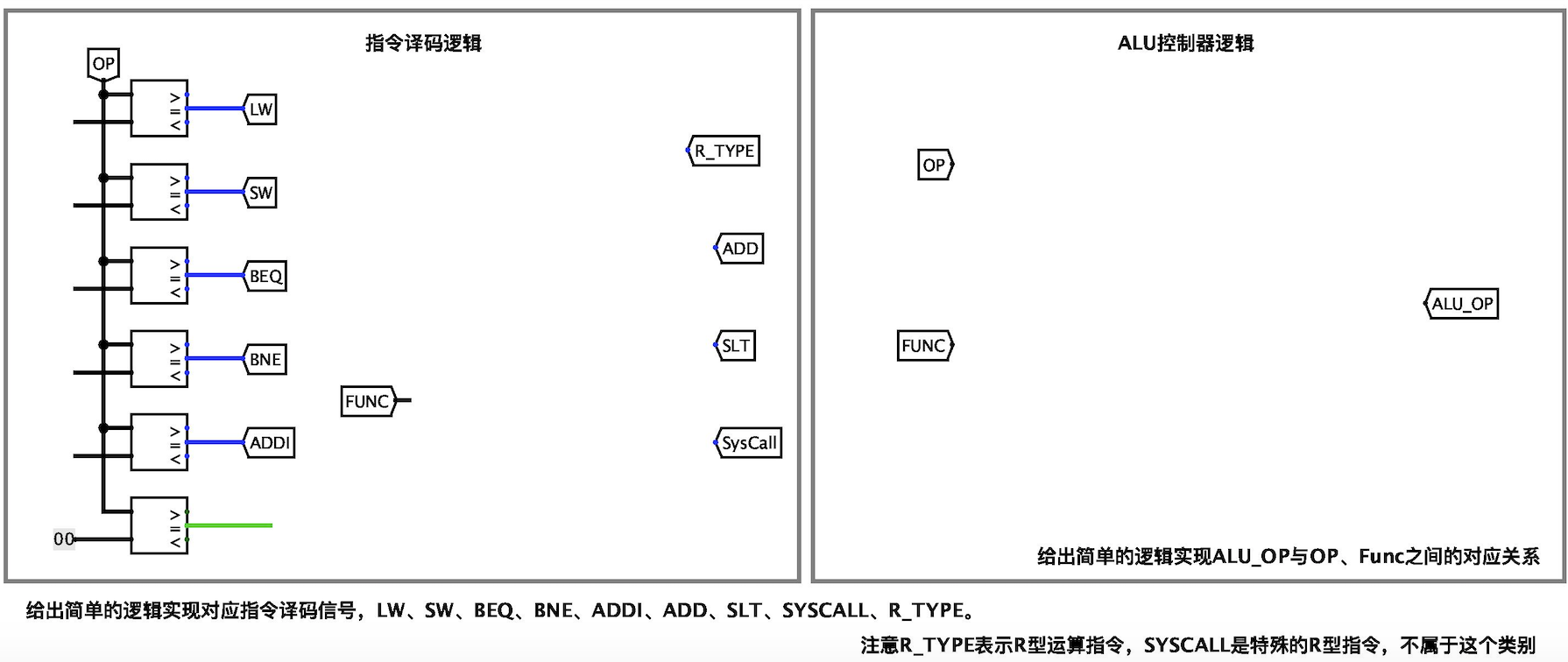


图 1.3 单周期CPU控制器内部逻辑任务图

下面分别详细介绍两个电路的完成方式：

1.完成指令译码逻辑

实验包中给出的MIPS\_Green\_Sheet告诉了我们具体指令和其OP/FUNC字段的对应关系，我们参照这个手册完成以下逻辑即可：

对于LW、SW、BEQ、BNE、ADDI。这几条指令是I型指令，我们直接让指令对应的操作码和分离出的OP字段比较，若相等，对应指令高电平即可。

对于ADD、SLT、SysCall，这些指令的op字段为0，所以我们还要比较分理处的FUNC字段。逻辑为：如果OP字段为0且FUNC字段与指定功能对应，则对应指令输出高电平。

对于ADD、SLT这两个指令，其为R型指令，我们将这两个指令进行或逻辑，连接R\_TYPE。

2.完成ALU控制逻辑

查阅资料我们知道ALU\_OP为5代表加法、为6代表减法、为11（0xb）代表

我们用一个多路选择器实现，选择0接数据0xb，表示有符号比较；选择1接0x5，表示加法。选择端规定如下：

如果OP是0x23（LW）、0x2b（SW）、0x8（ADDI）或者OP端为0且FUNC端为0x20（ADD），证明是不用比较的逻辑，选通选择端1

其他情况选择选通端0（包括BEQ 、BNE跳转）

3.内部逻辑电路图

指令译码和ALU控制的逻辑实现后的电路图如下图1.4所示：

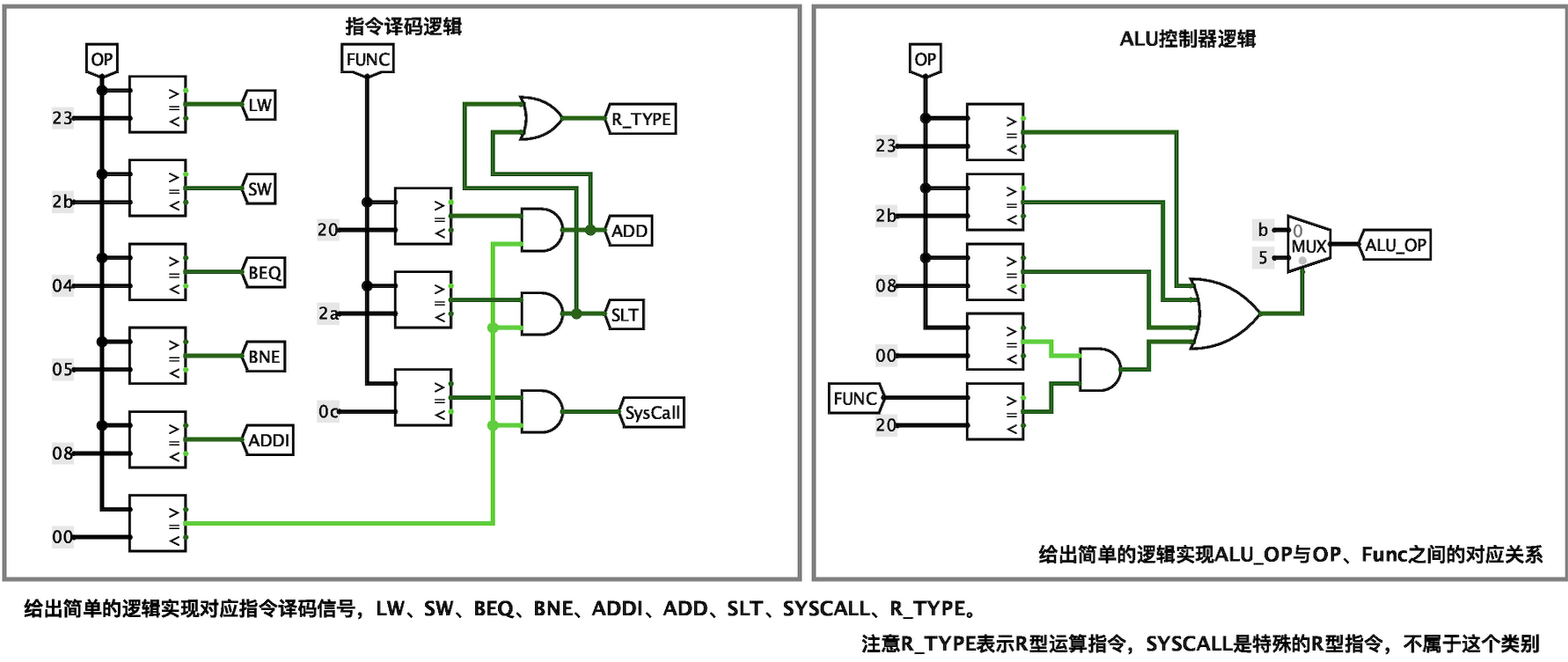


图 1.4 单周期CPU控制器内部逻辑电路图

1. 实现信号逻辑

我们需要根据指令译码信号给出所有的控制信号逻辑，任务如图：

我们用表给出单周期硬布线控制器中每个控制信号的说明和产生条件，如表1.2所示。

表 1.2 单周期硬布线控制器中控制信号的产生条件

|  |  |  |  |
| --- | --- | --- | --- |
| # | 控制信号 | 信号说明 | 产生条件 |
| 1 | RegDst | R型指令 | R\_TYPE |
| 2 | RegWrite | 寄存器写使能控制信号 | R\_TYPE、ADDI、LW |
| 3 | MemToReg | 写入寄存器的数字来自存储器 | Lw |
| 4 | MemWrite | 写内存的控制信号 | Sw |
| 5 | AluSrc | 运算器第二输入选择 | Lw、Sw、ADDI |
| 6 | Beq | Beq指令译码信号 | Beq |
| 7 | Bne | Bne指令译码信号 | Bne |
| 8 | Halt | 停机信号，取反后控制PC使能端 | SysCall |

如表1.2，我们列举出了每一个信号的产生条件，多个条件只需要用或逻辑即可，结果如下图1.5所示：

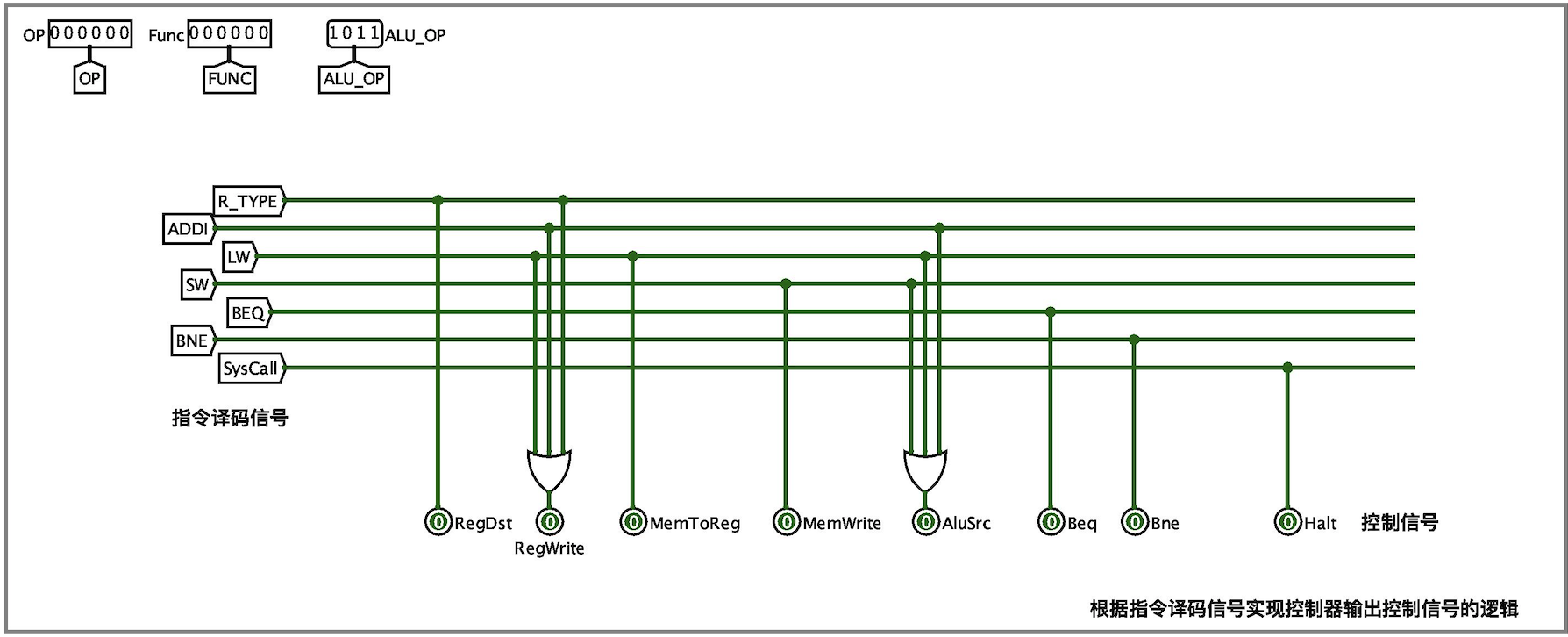


图 1.5 单周期CPU信号逻辑电路图

### 多周期MIPS CPU（微程序）的控制器

1. 实现控制器内部逻辑（指令译码和ALU控制）

多周期的指令译码逻辑与单周期的完全相同，这里不再多叙述。

多周期的ALU控制逻辑与单周期略有不同，主要体现在其多了一个ALU\_Conrol控制信号。要求：ALU\_Control=00时运算器做加法，ALU\_Control=01时运算器做减法，ALU\_Control=10时运算方式由Func决定。

显然，ALU\_Control=10时的逻辑我们已经在单周期时实现，我们只需要多加一个4输入的多路选择器即可，逻辑如下：

选择端为00，数据端连接0x5

选择端为01，数据端连接0x6

选择端为10，数据端连接单周期时候的ALU\_OP逻辑输出

选择端10悬空

多周期微程序控制器的内部逻辑（指令译码和ALU控制）如下图1.6所示：

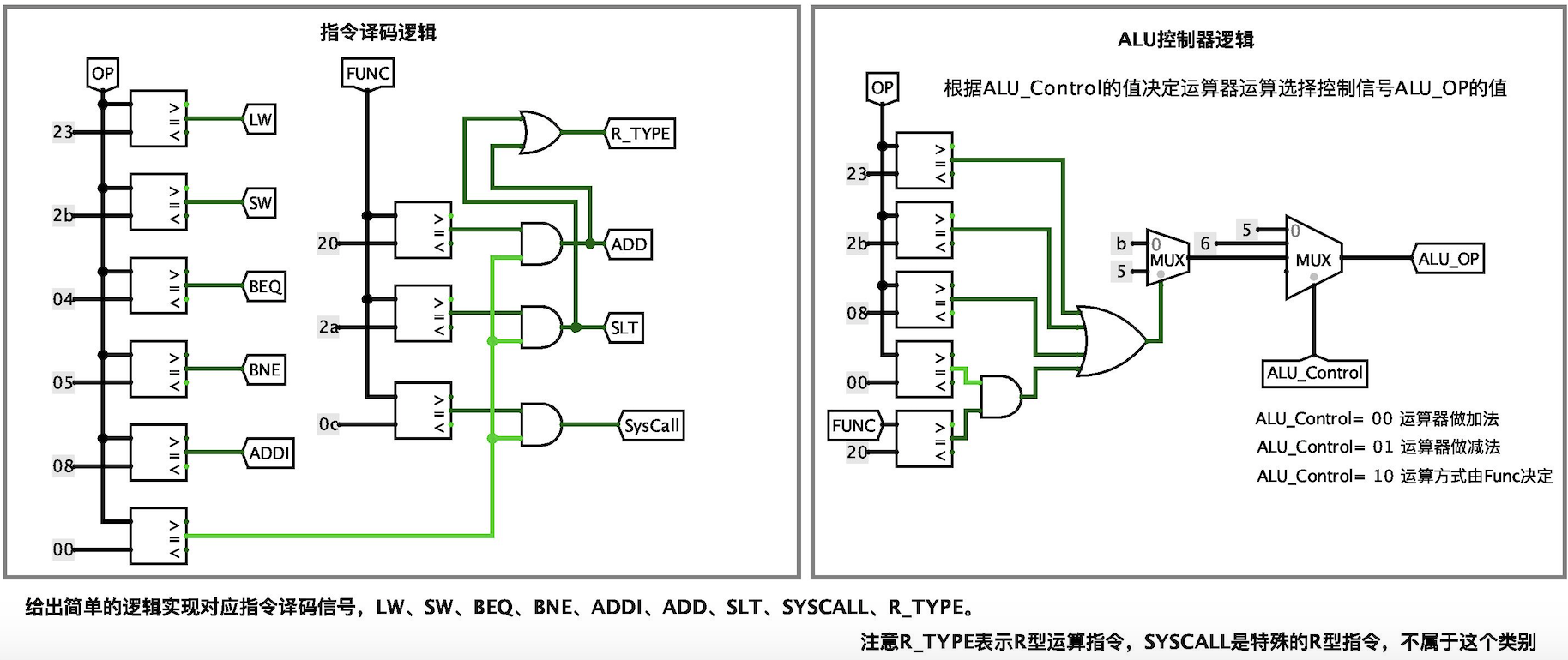


图 1.6 多周期CPU内部逻辑电路图

1. 构建指令状态变换图

多周期程序执行的指令状态变换图如下图1.7所示：

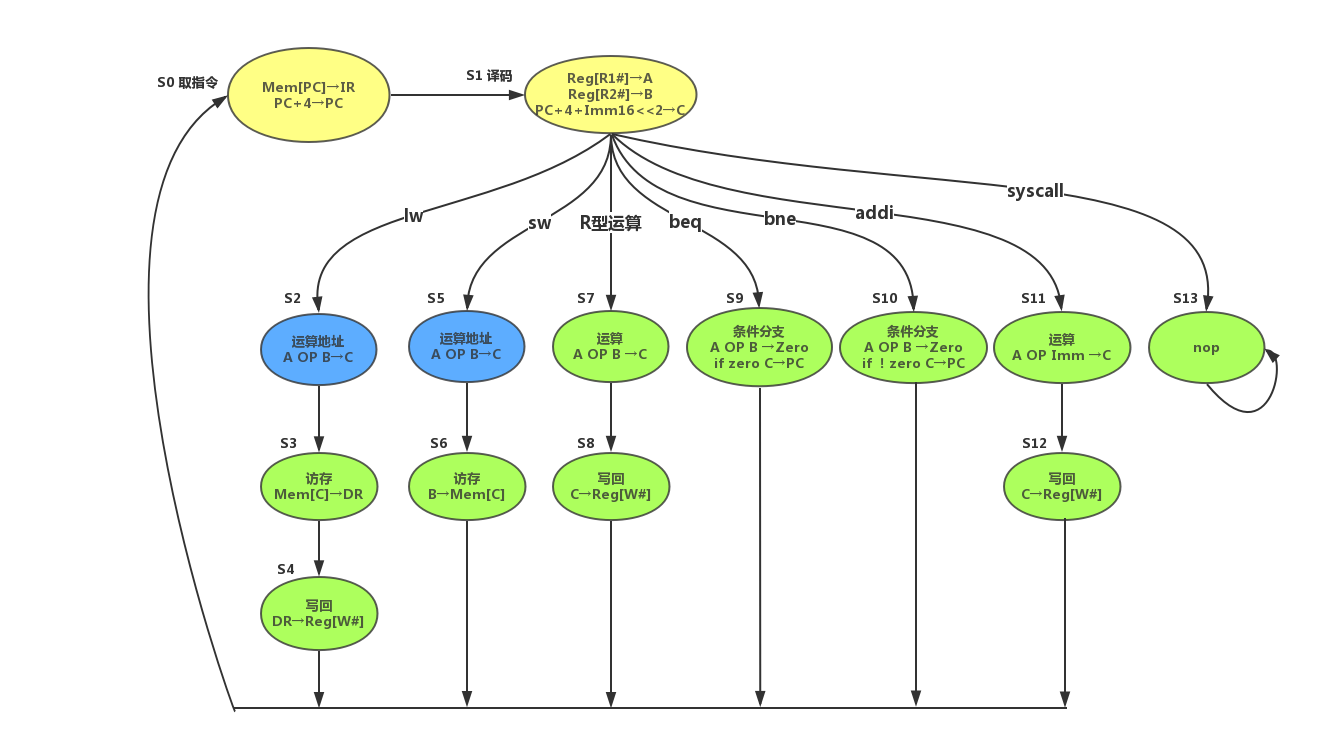


图 1.7 多周期CPU指令状态变换图

需要根据这个转换图实现多周期的地址转移逻辑和控制存储器的设计，需要完善如下图1.8的逻辑电路：

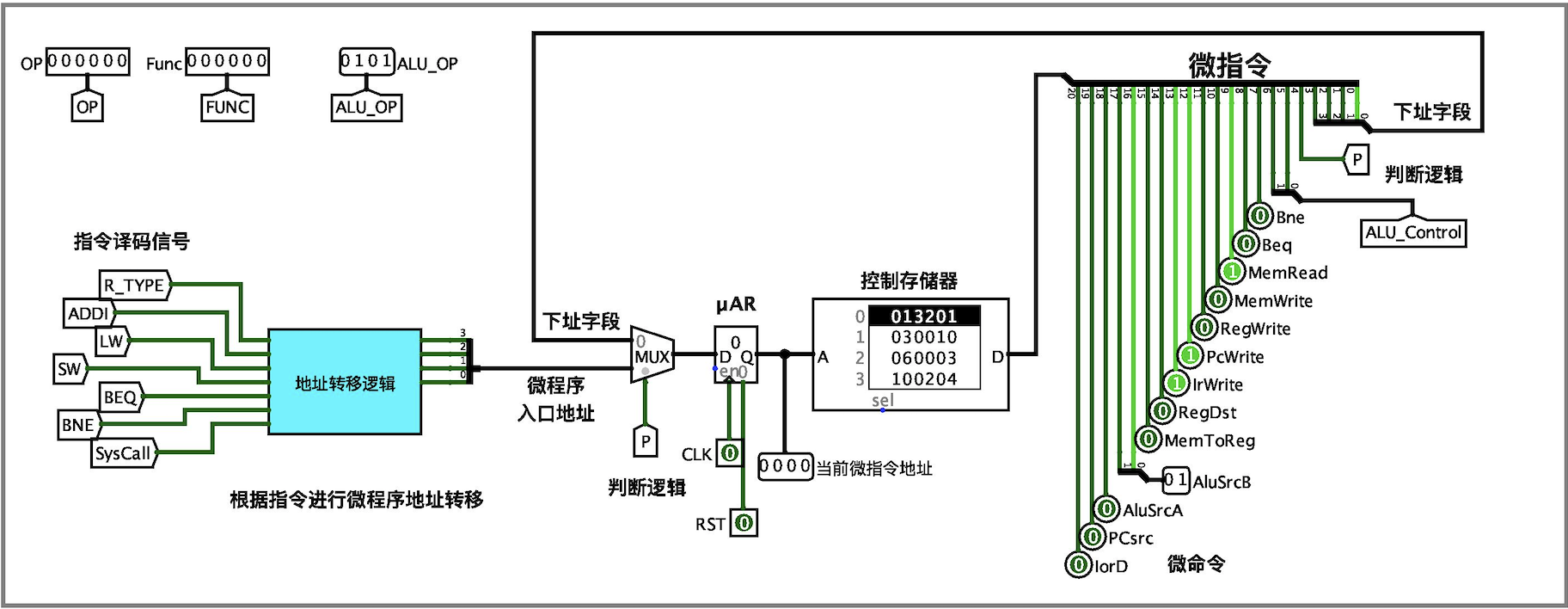


图 1.8 多周期CPU微程序控制器的顶级视图

（3）、（4）两节将分别详细介绍微程序地址转移逻辑和控制存储器的内部实现。

1. 实现微程序地址转移逻辑

按照上述图1.7所示，我们把每一条微指令的编号当作其地址，比如LW指令的入口编号为S2，我们就设其入口地址为0010，依次类推，我们可以推出每一个程序的入口地址。由于我们有14条微指令，所以我们需要地址为4位。

我们用excel表填写每一个程序的入口地址，实现地址转移逻辑，如下图1.9所示：



图 1.9 微程序地址转移逻辑excel图表

将自动生成的表达式复制到logisim的分析电路功能中，自动生成微程序地址转移逻辑子电路，生成的电路如下图1.10所示：

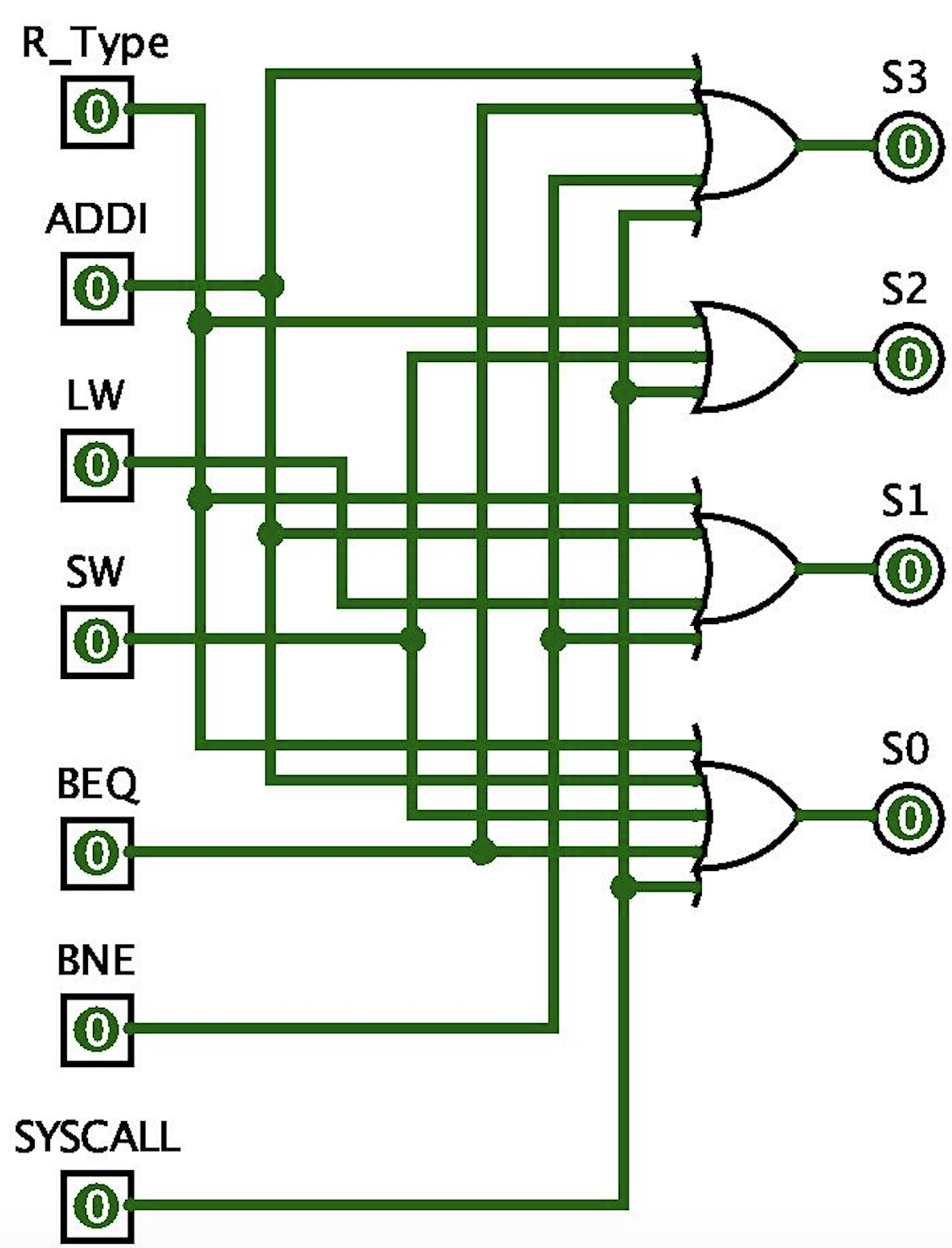


图 1.10 微程序地址转移逻辑电路图

1. 根据状态图构建微程序

根据状态图1.7我们知道了每一条微指令所对应的功能、地址编号以及转移方式，现在我们根据这些已知信息完成微程序，输入到excel表中，自动生成16进制的微程序，复制粘贴到控存中即可。下图1.11展现了我们已经完成的微程序指令excel表：

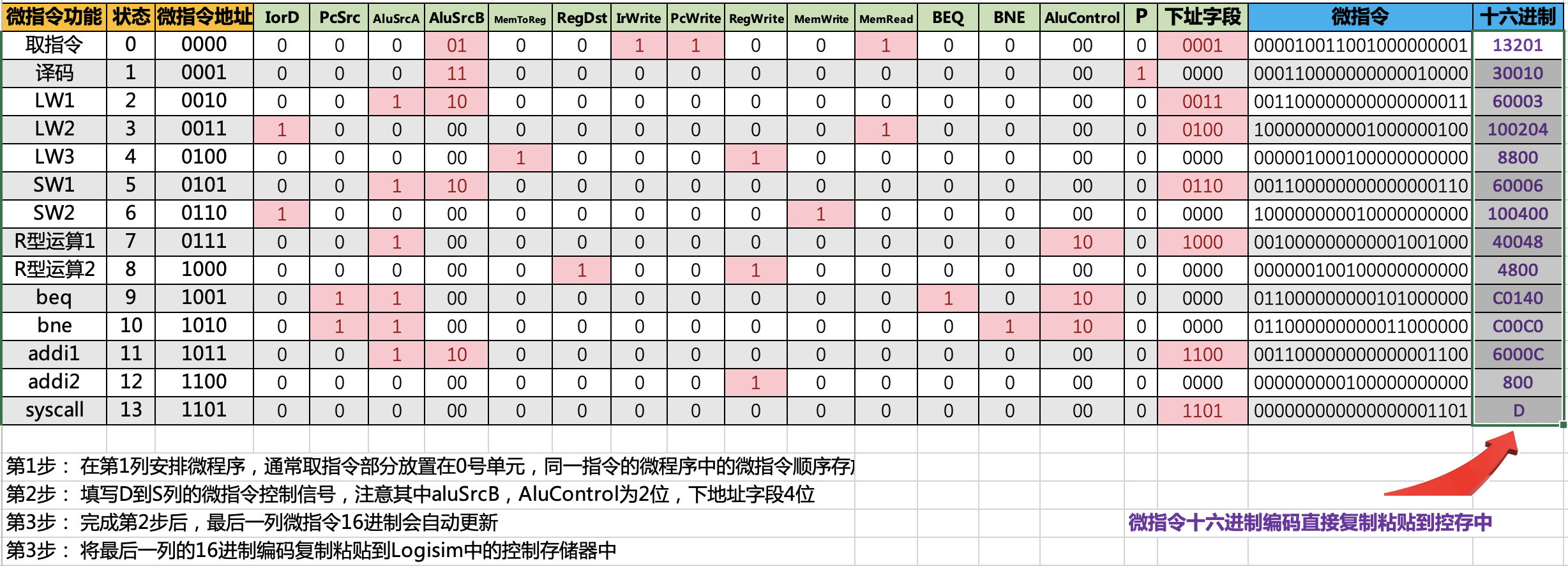


图 1.11 微程序指令生成excel图表

设计的过程有一定难度，我们在这里对每一条指令进行简要说明：

1. 取指令（0000）:要完成Mem[PC]→IR，需要读Mem、写IR，对应指令MemRead、IrWrite。要完成PC+4→PC，需要写PC，选择ALU的+4端口，对应PcWrite、AluSrcB选择01。下一个跳转的是译码功能，所以判别字段为0，下址字段为0001。
2. 译码（0001）:要完成Reg[R1#]→A和Reg[R2#]→B，需要一个时钟即可，不要指令。实现PC+4+Imm16→C需要ALU的B端选择立即数左移两位的结果，也就是AluSrcB的11。其判别字段为1，因为需要根据程序需求跳转，下址字段为0000。
3. LW1（0010）:需要实现A OP B→C，对应 R[$rs]+SignExt16b(imm)这一加法，所以AluSrcA选择1，AluSrcB选择10。下一个执行指令为LW2，故判别字段为0，下址字段为0011。
4. LW2（0011）：需要实现Mem[C]→DR，要访问存储器需要MemRead信号，需要C的数据传递到存储器，需要IorD信号为1。下一个执行的指令为LW3，故判别字段为0，下址字段为0100。
5. LW3（0100）：需要实现DR→Reg[W#]，则要MemToReg信号和RegWrite信号。下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
6. SW1（0101）:需要实现A OP B→C，对应R[$rs]+SignExt16b­(imm)这一加法，所以和LW1一样，AluSrcA选择1，AluSrcB选择10，下一个执行的指令为SW2，故判别字段为0，下址字段为0110
7. SW2（0110）:需要实现B→Mem[C]，故需要IorD信号为1，同时需要让内存开启写，即MemWrite为1。下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
8. R型运算1（0111）:需要实现A OP B→C，是两个寄存器的值的操作，故AluSrcA选择1，AluSrcB选择00。因为是R型操作，可能包含SLT操作，故此时要自选运算器的运算操作，即ALU\_Control需要为10。下一个执行的指令为R型运算2，故判别字段为0，下址字段为1000。
9. R型运算2（1000）：需要实现C→Reg[W#]，需要选通RegDst信号传递Rd寄存器的地址，还需要有RegWrite操作，允许写入寄存器。下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
10. BEQ（1001）:需要判断A、B寄存器是否相等，则AluSrcA选择1，AluSrcB 选择00，因为涉及到比大小，ALU\_Control选择为10。同时将寄存器C的值送到PC，即PcSrc要为1，同时还要给出beq信号为1。下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
11. BNE（1010）:和BNE字段一样，AluSrc选择1，AluSrcB选择10，ALU\_Control选择为10，同时PCSrc为1，给出bne信号为1。下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
12. ADDI1（1011）:实现A OP Imm→C，AluSrcA要为1，AluSrcB要为10。下一个执行的指令为ADDI2操作，故判别字段为0，下址字段为1100。
13. ADDI2（1100）:实现C→Reg[W#]，RegWrite要为1，下一个执行的指令为取指令操作，故判别字段为0，下址字段为0000。
14. SysCall（1101）:实现停机，只需要判别字段为0，下址字段为1101即可。

将16进制的微程序加载到控制存储器中，如下图1.12所示：

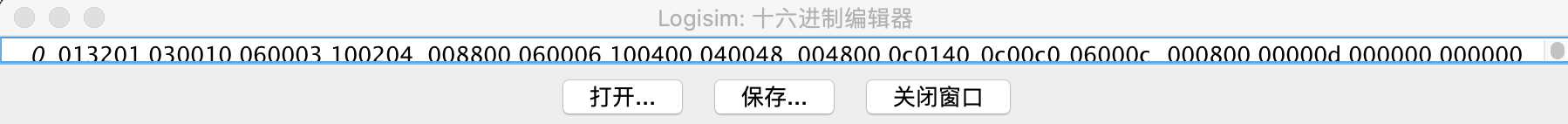


图 1.12 加载在控制存储器中的内容

### 多周期MIPS CPU（硬布线）的控制器

1. 实现控制器内部逻辑（指令译码和ALU控制）

多周期硬布线的控制器内部逻辑和微程序完全一致，我们这里只给出结果，不再详细叙述，如下图1.13所示：

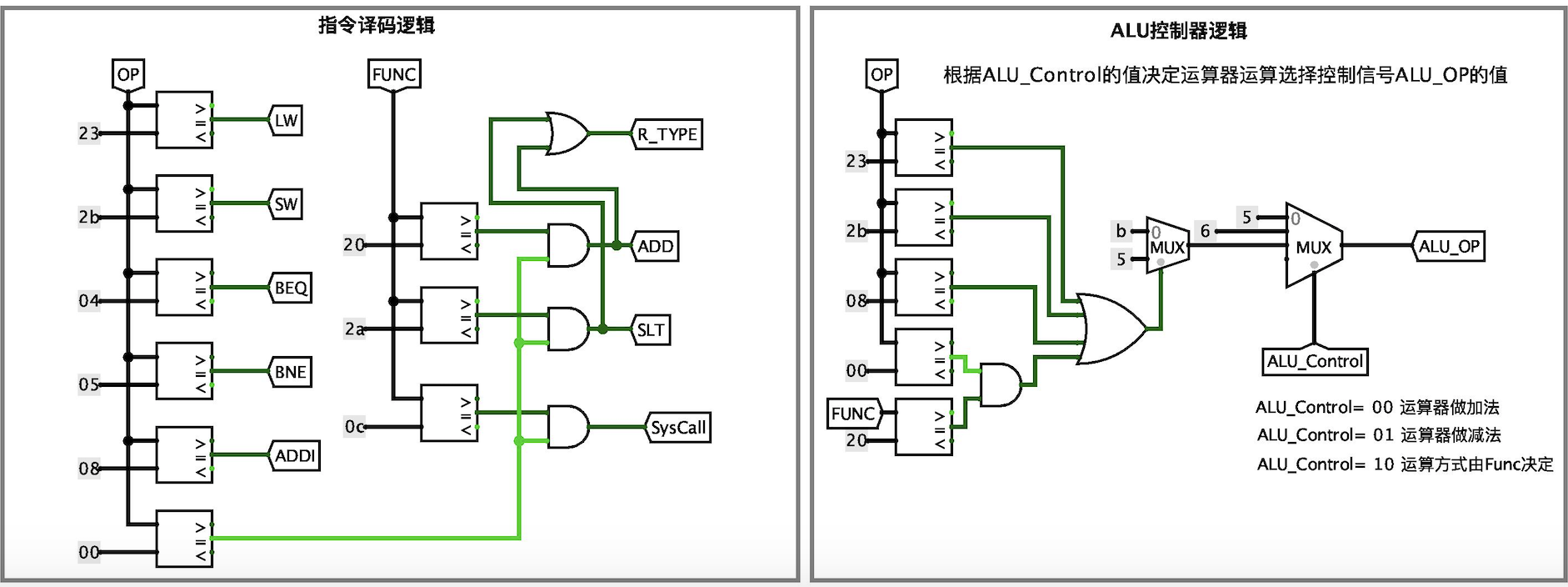


图 1.13 多周期硬布线控制器的内部逻辑电路图

1. 根据状态图生成转移逻辑

现在，我们将要完成多周期的FSM状态机的组合逻辑和控制存储器的逻辑，需要完善如下图1.14的逻辑电路：

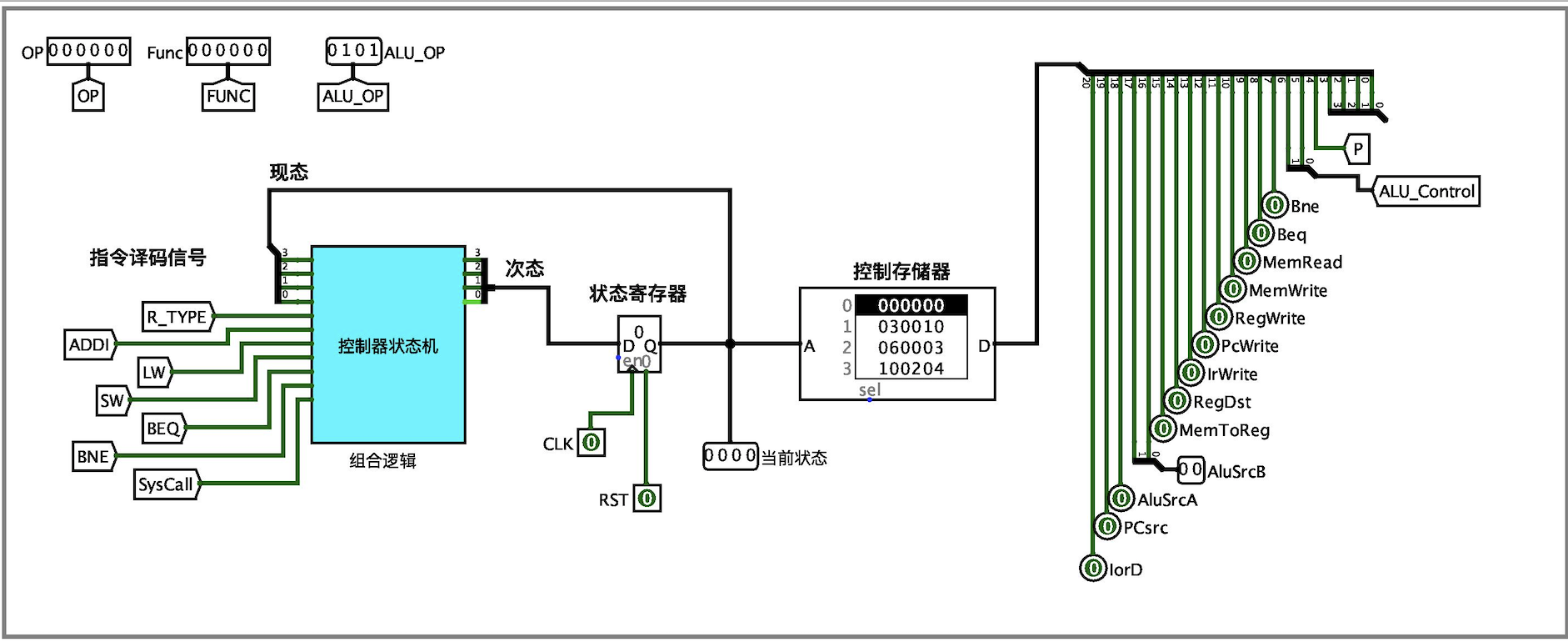


图1.14 多周期硬布线控制器的顶层视图

其状态图和微程序控制器的状态图完全一致，参见图1.7。

状态图的转移逻辑通过填写excel表以及电路分析得到，填写好的excel表如下图1.15所示：

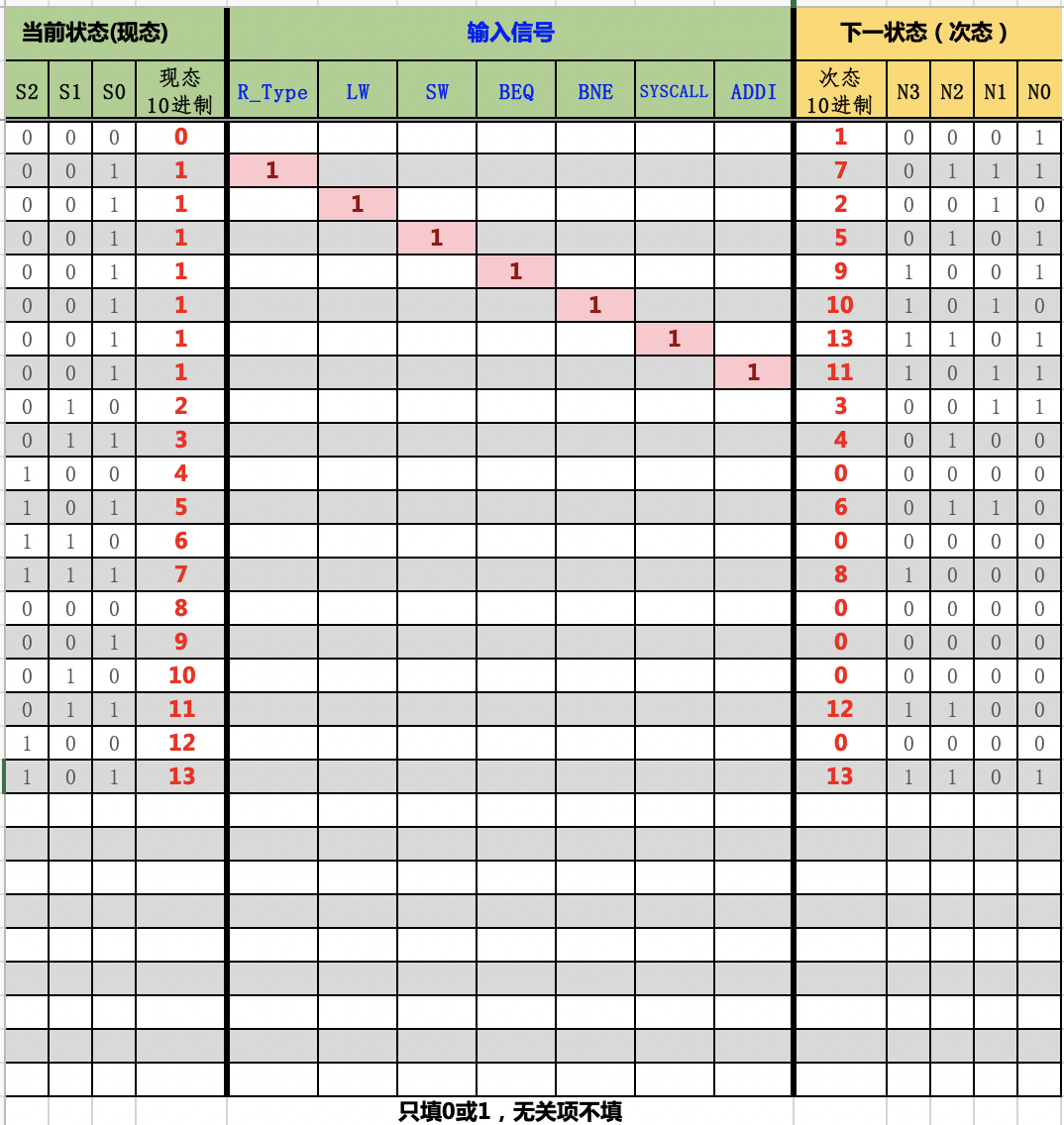


图1.15 多周期硬布线控制器转移逻辑excel图表

上图构建了每一条微指令与下一条微指令的跳转关系，根据状态图填写即可。这里不再展开叙述。然后再在excel表中的第二页将对应节点的逻辑表达式输入到电路“状态及FSM”进行电路分析即可。由于版面限制加上电路截图过长，我们这里给出N3、N2、N1、N0的表达式代替电路截图：

1. 控制存储器的设计

控制存储器的设计与微指令控制器完全一样，将生成的16进制微指令加载入控制存储器即可，如下图1.16所示：

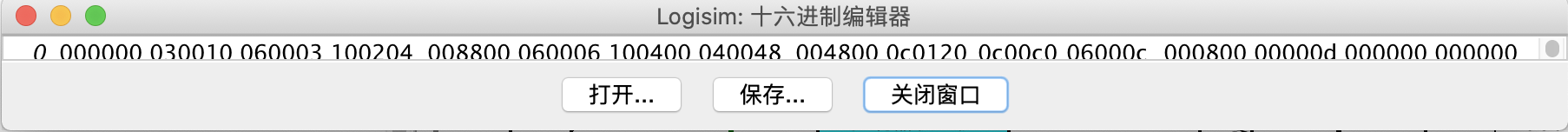


图1.16 多周期硬布线控制器加载在控存中的内容

## 故障与调试

### 单周期CPU无法停机

**故障现象：**单周期CPU无法停机，观察指令寄存器，其总是在地址为0x10到0x20左右的指令进行死循环，下图1.17展示了死循环中某一时刻的截图：

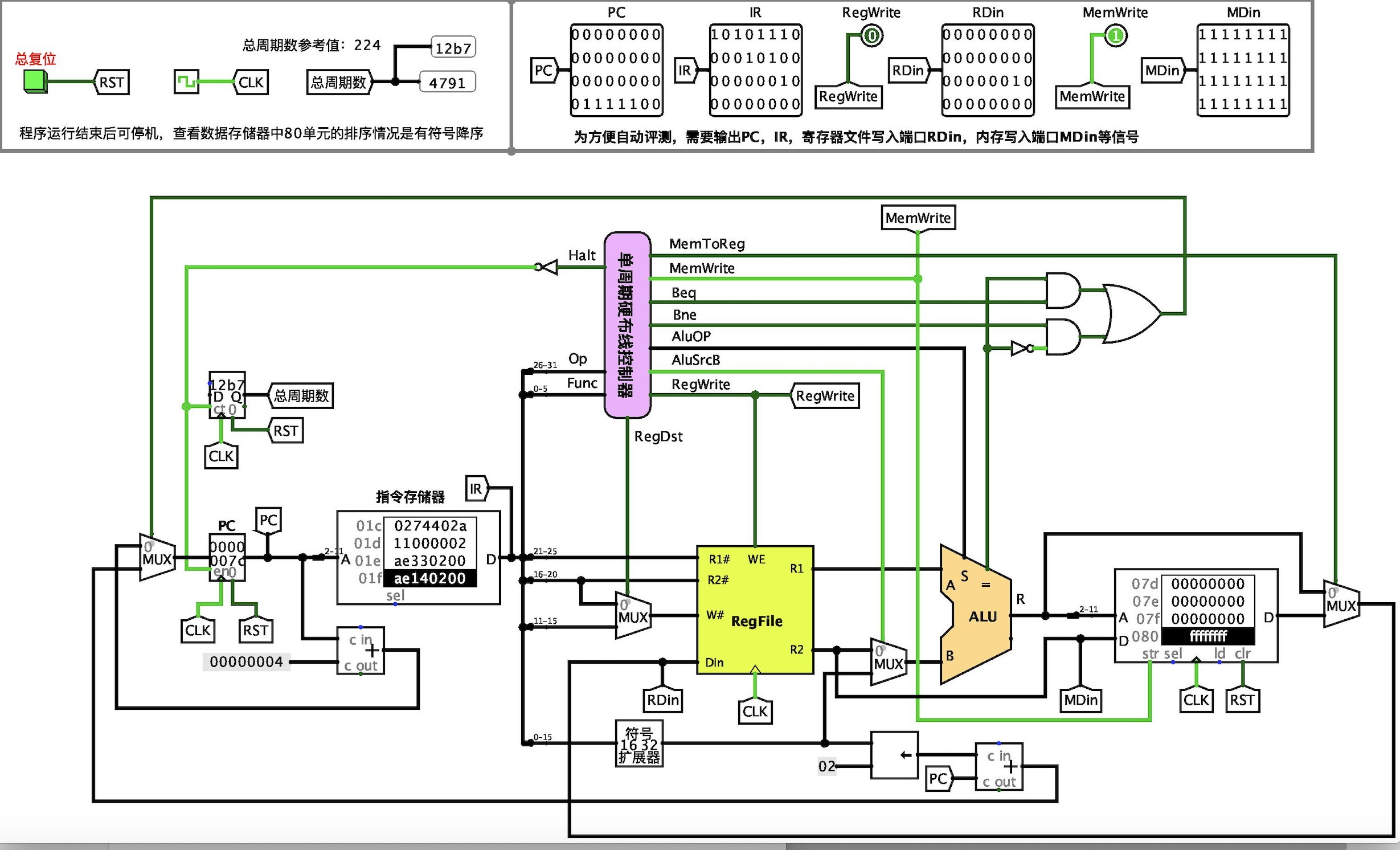


图1.17 单周期CPU进入死循环示意图

可以看出，总周期数在无限增长。停止时钟模拟，打开数据寄存器，发现冒泡排序结果不对，如下图1.18所示：

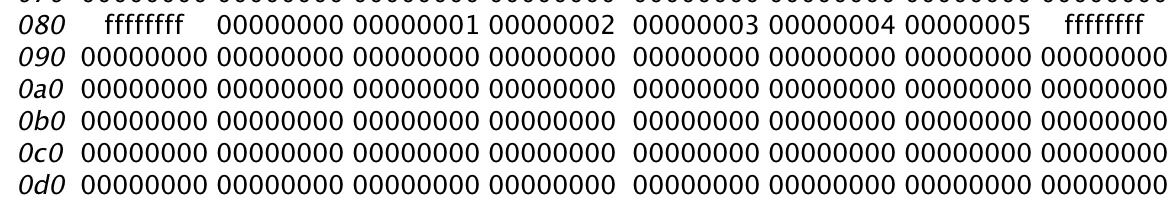


图1.18 故障一发生时的冒泡排序结果示意图

**原因分析：我们知道，要想要程序停机，最后执行的一条指令的地址是0x26，但是程序在该通路上运行的时候一直在0x10到0x20左右进行循环，证明PC的设置有问题。沿着这个思路进行下去，发现是在PC加上立即数的时候忘记加上4。**

**解决方案：在立即数和PC相加写入PC寄存器的那一路上该换成立即数和PC+4相加，写入PC寄存器，就有了我们在第二章节图1.1所示的通路图。**

### 多周期CPU的总周期数不正确

### **故障现象：**多周期MIPS CPU在停机的时候总周期数不是891，而是212，但是排序正常，如下图1.19和图1.20所示。

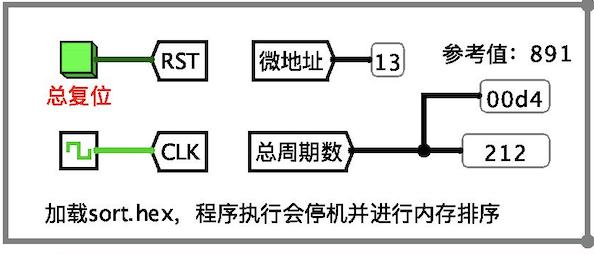


图1.19 多周期CPU停机时总周期数不对示意图

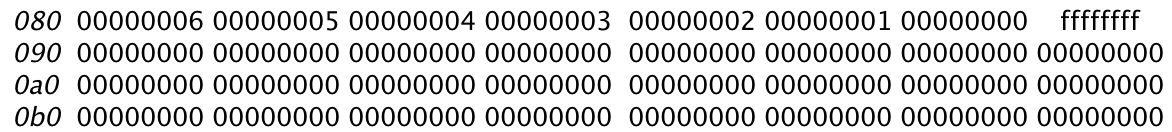


图1.20 故障二发生时的冒泡排序结果示意图

**原因分析：根据图上给出的信息来看，程序运行的是正确的，正常排序了，停机的微地址为13也是正确的。那么肯定是统计总周期数的时候出现了问题，于是想到了计数器出了问题，沿着这个思路排查下去，发现是计数器使能端连接的比较器除了问题，导致少进行了一些计数。**

**解决方案：将比较微地址的比较器数字类型由“关于2的补码”替换成“无符号”即可。**

## 测试与分析

### 单周期硬布线MIPS CPU的测试

将sort.hex加载到单周期CPU指令存储器，启动时钟模拟，点击复位按钮，开始测试。停机之后发现总周期数和参考值一致，截图如下图1.21所示：



图1.21 单周期硬布线CPU总周期数示意图

查看数据寄存器中80单元的排序情况，发现时有符号降序排序，证明冒泡排序程序测试成功，截图如下图1.22所示：

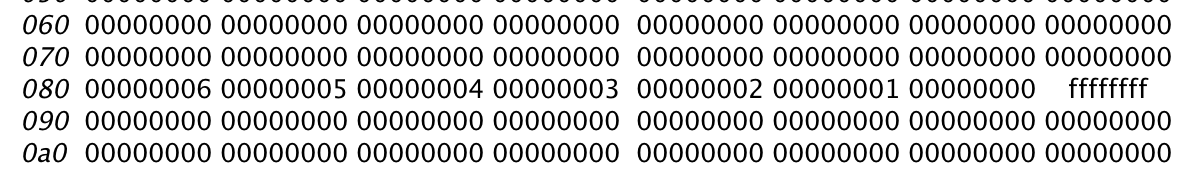


图1.22 单周期硬布线CPU冒泡排序示意图

### 多周期微程序MIPS CPU测试

将sort.hex加载到多周期微程序CPU的内存，启动时钟模拟，点击复位按钮，开始测试。停机之后发现总周期数和参考值一致，截图如下图1.23所示：

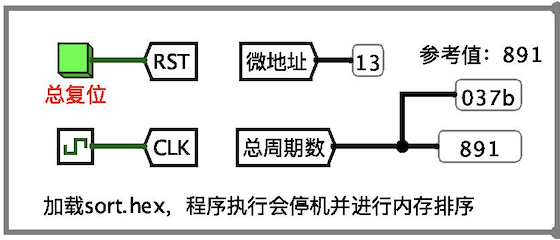


图1.23 多周期微程序CPU总周期数示意图

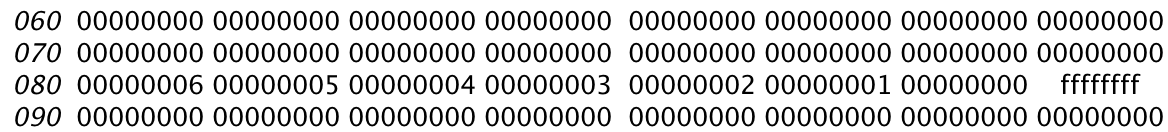
查看内存中的数字排序情况，为有符号降序排序，证明冒泡排序程序运行成功，截图如下图1.24所示：  


图1.24 多周期微程序CPU冒泡排序示意图

### 多周期硬布线MIPS CPU测试

将sort.hex加载到多周期硬布线CPU的内存，启动时钟模拟，点击复位按钮，开始测试。停机之后发现总周期数和参考值一致，截图如下图1.25所示：

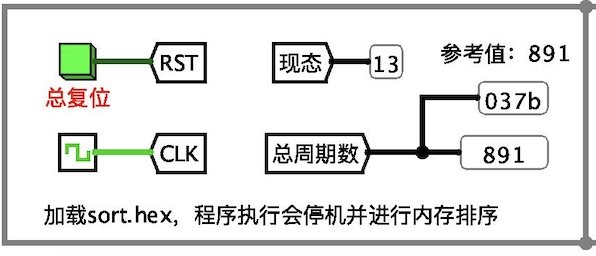


图1.25 多周期硬布线CPU总周期数示意图

查看内存中的数字排序情况，为有符号降序排序，证明冒泡排序程序运行成功，截图如下图1.26所示：

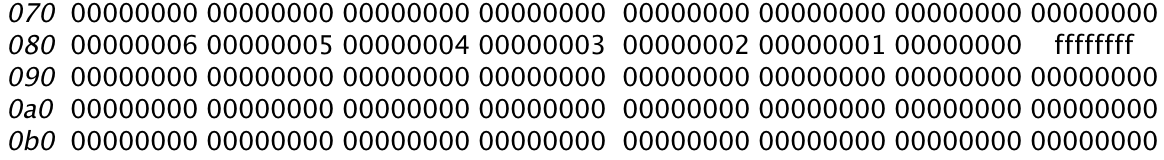


图1.26 多周期硬布线CPU冒泡排序示意图

# 总结与心得

## 实验总结

本次实验主要完成了如下几点工作：

1. 在logisim平台上完成了单周期CPU和多周期CPU的数据通路的构建。
2. 设计了单周期硬布线控制器、多周期微程序控制器以及多周期硬布线控制器。
3. 在每个控制器上，我们设计并实现了8条核心指令集，其中涉及到指令的译码、信号逻辑的控制、地址逻辑的转移、微指令的设计、状态逻辑的转移等等内容。
4. 在8条核心指令的基础上，在3个电路上分别完成了冒泡排序程序的测试。

## 实验心得

1. 本次实验让我们对CPU有了更加深入的理解，完成实验的过程是对整个CPU章节的一次很好的复习，也让我们更加关注了CPU中的具体实现的细节。通过完成这一次实验，收获主要有：其一，熟悉了单周期、多周期的CPU的数据通路构造；其二，也是实验中的关键，熟悉了控制器的构造原理，并完成了单周期的硬布线控制器、多周期的微程序控制器以及多周期的硬布线控制器。在设计完成控制器的过程中，我们对于指令在通路图上的执行及其状态的跳转了解更加深入。
2. 实验中也有许多小的细节需要注意。例如我在实验中遇到了如下的问题：在单周期实验中，我第一次跑出来的总周期数会一直增长，观察指令存储器发现是指令在死循环，后来在检查数据通路的时候发现是因为我在PC=PC+立即数的时候没有加4，导致了错误，后经改正正常。在多周期数据通路实现时，第一次跑的时候总周期数不是891，而是212，检查后发现，原来是比较微程序地址的时候比较器用的是补码，应该换为无符号数比较。
3. CPU实验以及整个组原实验给人的感觉是特别好的，实验内容非常贴合课程内容，实验平台以及理论知识的讲解做的都是最到位的。通过做组原实验，我发现了更多的logisim的用法以及其功能的强大，在这里感谢设计课程实验的老师以及提出过课程建设建议的同学们。对于CPU实验，提出的一点建议就是，现在我们跑的程序只有冒泡排序一个，如果可以，其实可以多设计一点测试程序，增加实验的多样性和趣味性（对应的可能会多增加一些指令的设计）。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第5版).北京:机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018年.
4. 秦磊华，吴非，莫正坤.计算机组成原理. 北京:清华大学出版社，2011年.
5. 袁春风编著. 计算机组成与系统结构. 北京:清华大学出版社，2011年.
6. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: 嵌入签名图片** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |