***2025***



**逻辑与计算机系统设计 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | 本硕博2301 |
| 学 号： | U202315763 |
| 姓 名： | 王家乐 |
| 电 话： | 15391560195 |
| 邮 件： | chia.le@foxmail.com |
| 完成日期： | 2025-06-16 |



# 运动码表系统设计

## 设计要求

利用相关部件（选择器、比较器、寄存器、计数器、码表控制器等）构建运动码表系统数据通路。输入为Start、Stop、Store和Reset。输出为4个7段数码管显示数字，分别显示秒和百分秒。具体功能：（1）当按下Start时，计时器清零，重新开始计时；（2）当按下Stop时，计时器停止计时，显示计时数据；（3）当按下Store时，若当前计时数据小于系统记录，则更新系统记录，并显示当前计时数据；否则不更新系统记录，但显示系统记录；（4）当按下Reset时，复位，计时=0.00，系统记录=99.99。

表1-1 运动码表片引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| CLK | 输入 | 1 | 时钟脉冲 |
| TestMode | 输入 | 1 | 在线测评开关，0本地模式，1在线模式 |
| Start | 输入 | 1 | 开始计时信号 |
| Stop | 输入 | 1 | 停止计时信号 |
| Store | 输入 | 1 | 存储计时记录信号 |
| Reset | 输入 | 1 | 计时复位信号，记录恢复99.99 |
| Time | 输出 | 16 | 计时成绩或者成绩记录 |

## 方案设计

### 7段数码管驱动电路设计

观察图1-1数码管结构图，数码管共有8个控制信号，分别控制7段管和一个小数点（这里不考虑小数点），控制信号为1表示亮起，为0不亮。需要根据每个数字的四位BCD码来确定哪一段亮，哪一段不亮。

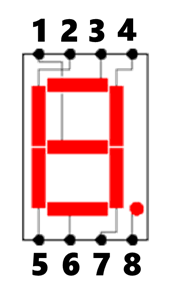


图1-1 7段数码管结构图

电路引脚及功能如表1-2所示。

表1-2 7段数码管电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| X3~X0 | 输入 | 1 | 4位BCD码输入 |
| Seg1~Seg7 | 输出 | 1 | 7位数码管驱动信号 |

构建电路时，先计算每个4位BCD码（8421BCD码）对应的十进制数值，再确定该数值在7段数码管中应该亮起的部分，将其对应的控制信号赋为1，其他控制信号赋为0。通过Logisim中的分析组合逻辑电路功能，补全真值表，如图1-2所示。

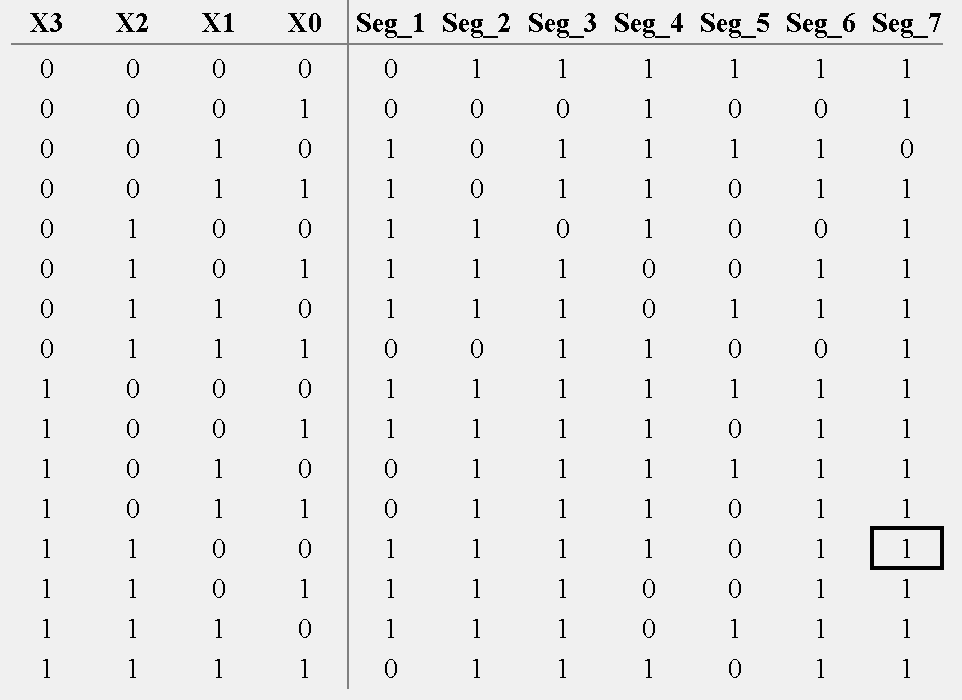


图1-2 7段数码管电路真值表

分析卡诺图可知，Seg\_1的输出存在险象（如图1-3所示）。当输入X0=0，X1=1，X3=0时，输出为~X2+X2，存在“0”型险象，可在输出函数中添加冗余项~X0X1~X3来消除。Seg\_2~Seg\_7不存在险象。

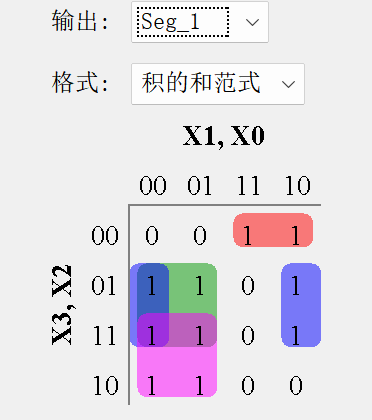


图1-3 Seg\_1卡诺图

最后点击“构建电路”即可自动生成电路，如图1-4所示。

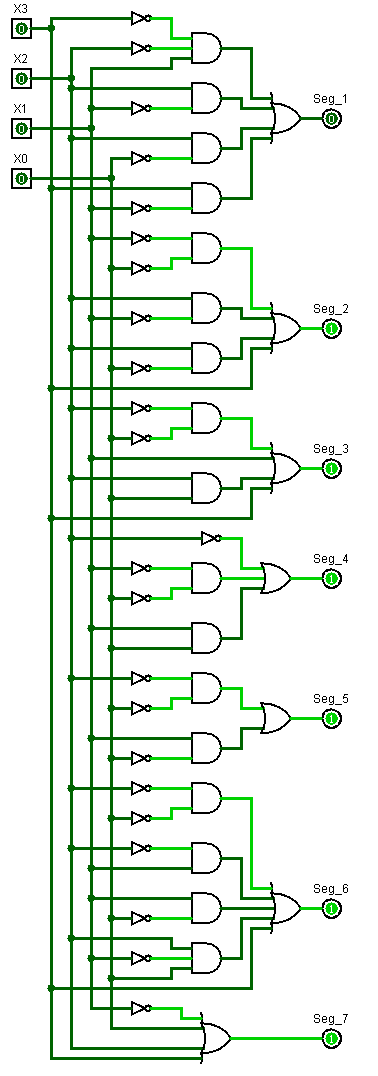


图1-4 7段数码管电路图

通过时钟控制信号输入BCD码，观察数码管状态，结果符合预期，如图1-5所示。

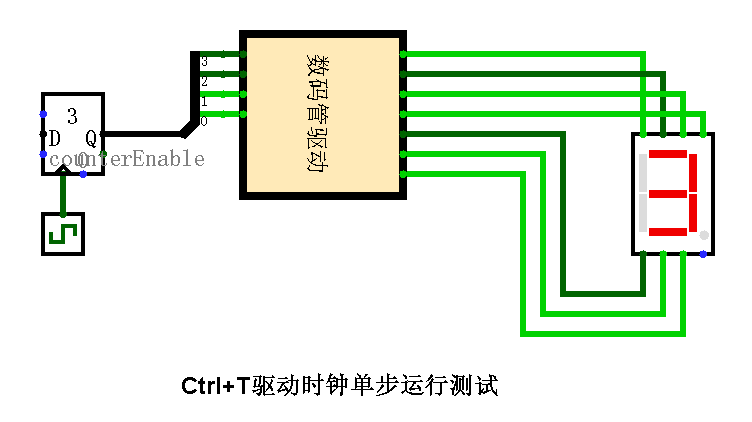


图1-5 7段数码管测试

### 2选1选择器设计（1位）

利用基本逻辑门构成1位的2路选择器，电路引脚及功能如表1-3所示。

表1-3 2选1选择器（1位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| X0 | 输入 | 1 | 2路输入的第0路 |
| X1 | 输入 | 1 | 2路输入的第1路 |
| Sel | 输入 | 1 | 选择控制端 |
| Out | 输出 | 1 | Out=(Sel==0)?X0:X1; |

该电路的逻辑函数表达式为，因此电路图如图1-6所示。

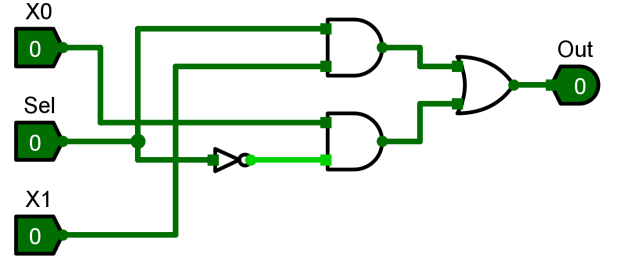


图1-6 2选1选择器（1位）电路图

### 2选1选择器设计（16位）

在1位2路选择器的基础上构建16位2路选择器，输入与输出都为16位。将数据拆分成16个1位数据，再使用16个1位2路选择器即可，电路图如图1-7所示。

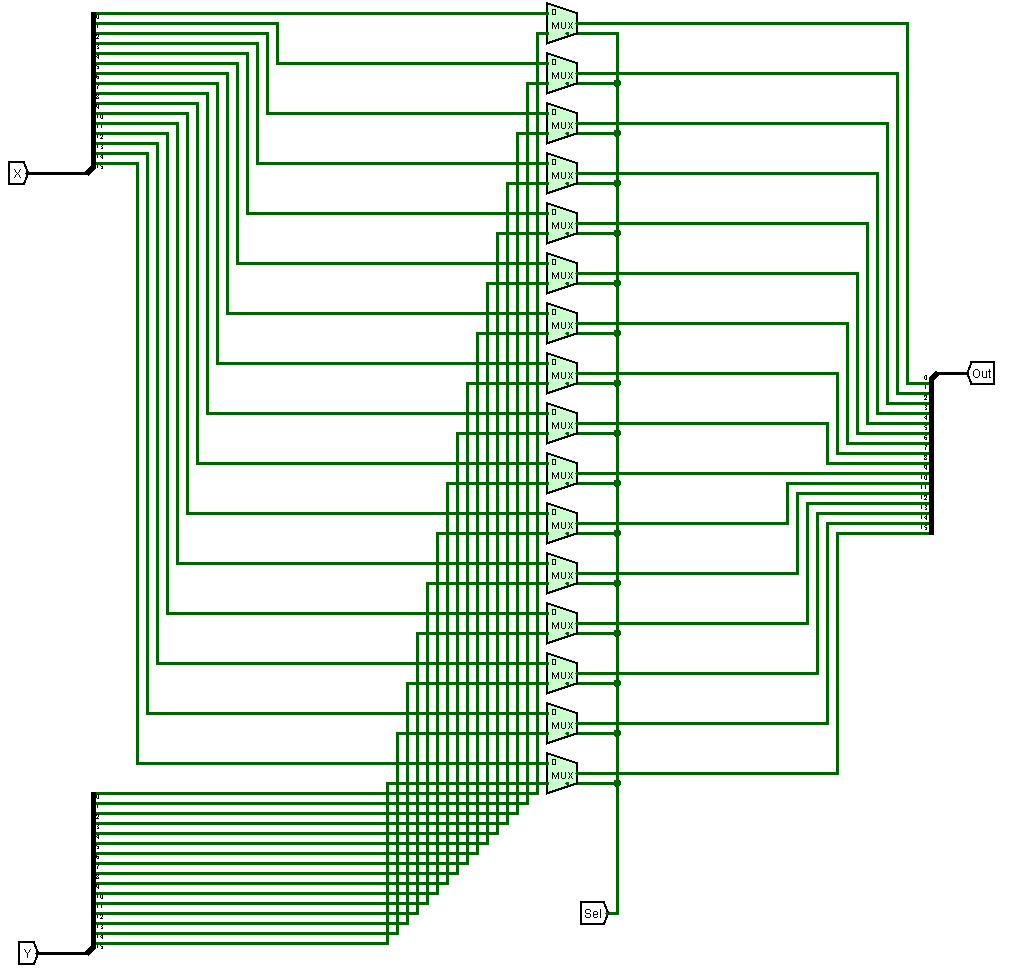


图1-7 2选1选择器（16位）电路图

### 无符号比较器设计（4位）

设计实现四位无符号比较器，电路引脚及功能如表1-4所示。

表1-4 无符号比较器（4位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| X3~X0 | 输入 | 1 | 4位输入X |
| Y3~Y0 | 输入 | 1 | 4位输入Y |
| Great | 输出 | 1 | X大于Y |
| Equal | 输出 | 1 | X等于Y |
| Less | 输出 | 1 | X小于Y |

若X>Y，则满足X3>Y3，或X3=Y3且X2>Y2，或X3=Y3且X2=Y2且X1>Y1，或X3=Y3且X2=Y2且X1=Y1且X0>Y0；若X=Y，则一定有X3=Y3且X2=Y2且X1=Y1且X0=Y0；若X<Y，同理不再赘述。因此逻辑表达式如下：*Great = X3 ~Y3 + ~(X3 ^ Y3) X2 ~Y2 + ~(X3 ^ Y3) ~(X2 ^ Y2) X1 ~Y1 + ~(X3 ^ Y3) ~(X2 ^ Y2) ~(X1 ^ Y1) X0 ~Y0；Equal = ~(X0 ^ Y0) ~(X1 ^ Y1) ~(X2 ^ Y2) ~(X3 ^ Y3)；Less = ~X3 Y3 + ~(X3 ^ Y3) ~X2 Y2 + ~(X3 ^ Y3) ~(X2 ^ Y2) ~X1 Y1 + ~(X3 ^ Y3) ~(X2 ^ Y2) ~(X1 ^ Y1) ~X0 Y0*。最终生成电路图如图1-8所示。

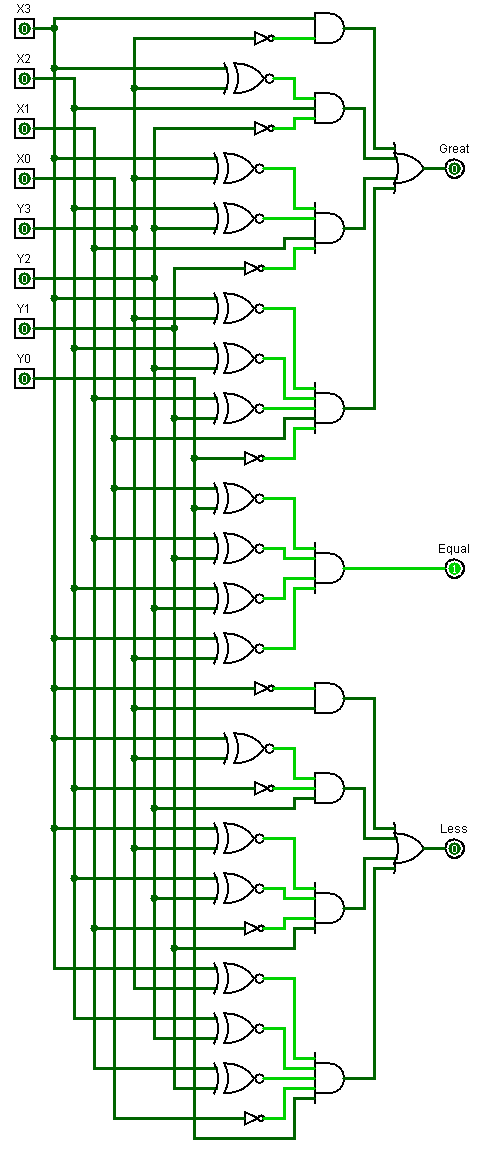


图1-8 无符号比较器（4位）电路图

### 无符号比较器设计（16位）

为了构建16位无符号比较器，我们将输入X和Y各拆分为四个4位段，并分别输入到四个4位无符号比较器中，每个比较器输出三个信号：Great，Equal，Less。比较过程从最高位段开始分析：如果该段的比较器输出Great为真，则整体X > Y；如果Less为真，则整体X < Y；如果Equal为真，则需依次检查下一个次高位段。重复上述判断逻辑。如果所有段均相等，则最终X = Y；否则，第一个非等段的结果（Great或Less）直接决定整体大小关系，低位段的结果仅在高位段相等时才生效。这种级联方式确保了高位优先原则，即高位段的比较结果权重更高，从而正确综合出16位比较结果。根据上述分析连接电路。电路图如图 1-9 所示。

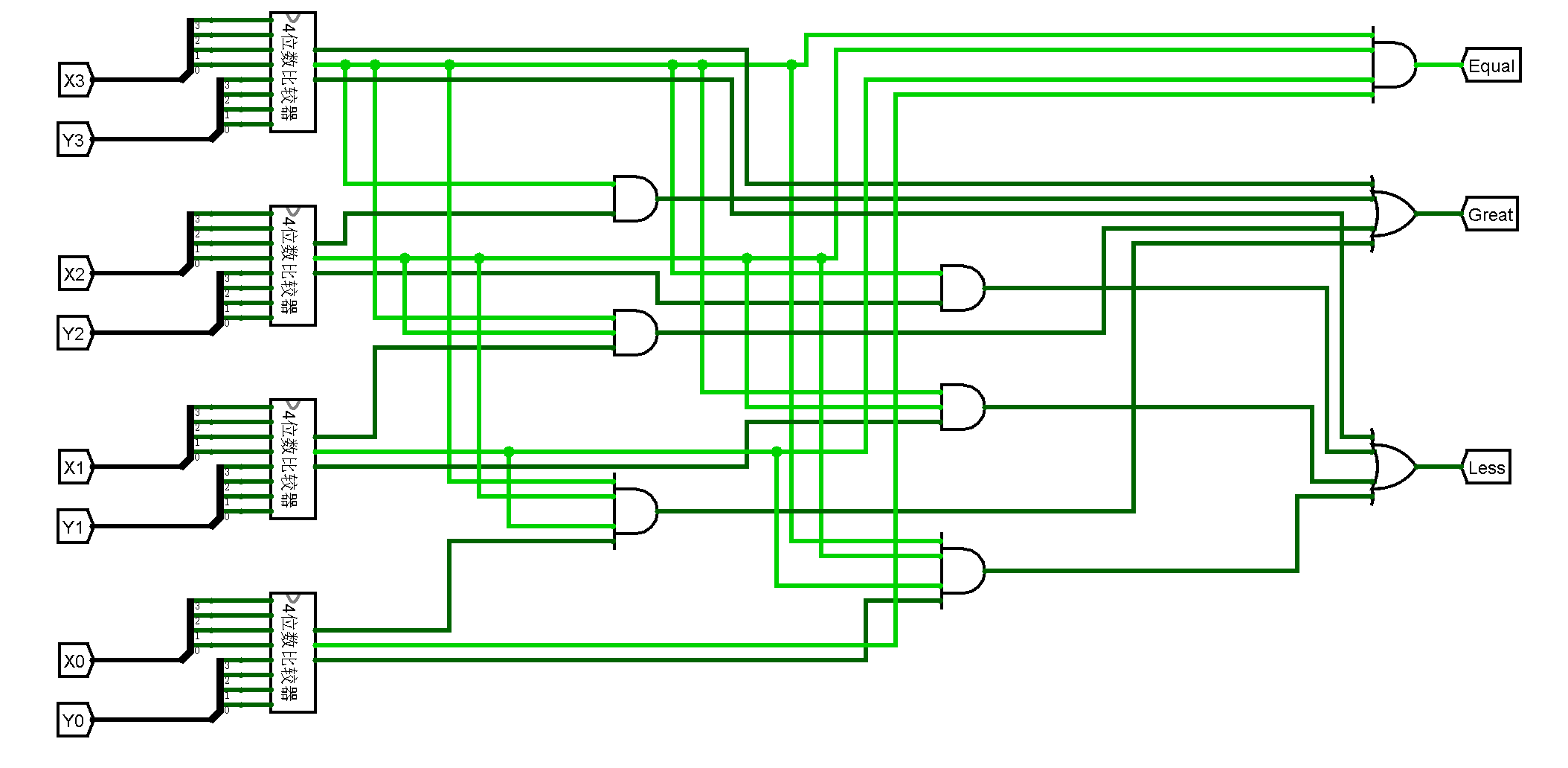


图1-9 无符号比较器（16位）电路图

### 并行加载寄存器（4位）

使用四个D触发器来构建4位寄存器，上升沿触发，具有高电平使能端，每个触发器存储一个二进制位，电路引脚及功能如表1-5所示。

表1-5 并行加载寄存器（4位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| Clock | 输入 | 1 | 时钟脉冲，上升沿有效 |
| Din | 输入 | 4 | 寄存数据输入端 |
| En | 输入 | 1 | 使能输入端，高电平有效 |
| Q | 输出 | 4 | 寄存数据输出端：En=1、Clock↑时Q=Din |

D触发器现态与输入一致，因此只需用分线器将Din分开为单独的二进制位，再连接到4个D触发器，将这4个D触发器的现态连接到Q即可，同时每个触发器接入时钟Clock和使能En。电路图如图1-10所示。

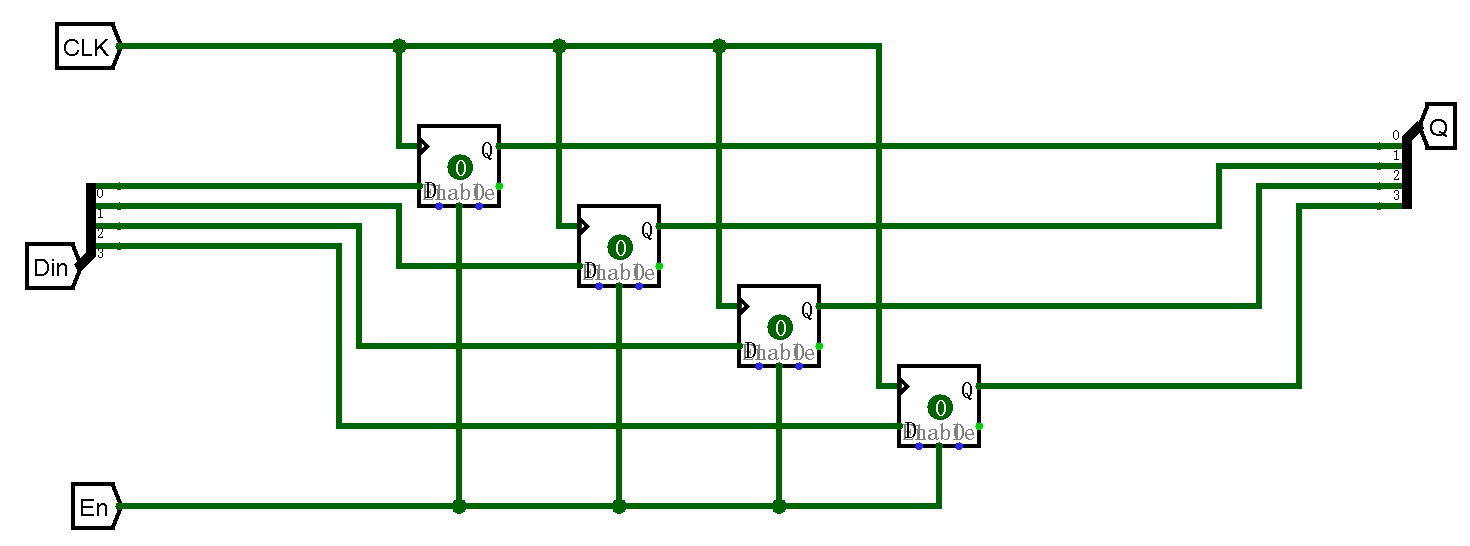


图1-10 并行加载寄存器（4位）电路图

### 并行加载寄存器（16位）

构建16位并行加载寄存器，我们可以使用四个封装好的4位寄存器，只需对输入数据Din与输出数据Q进行分线。电路图如图1-11所示。

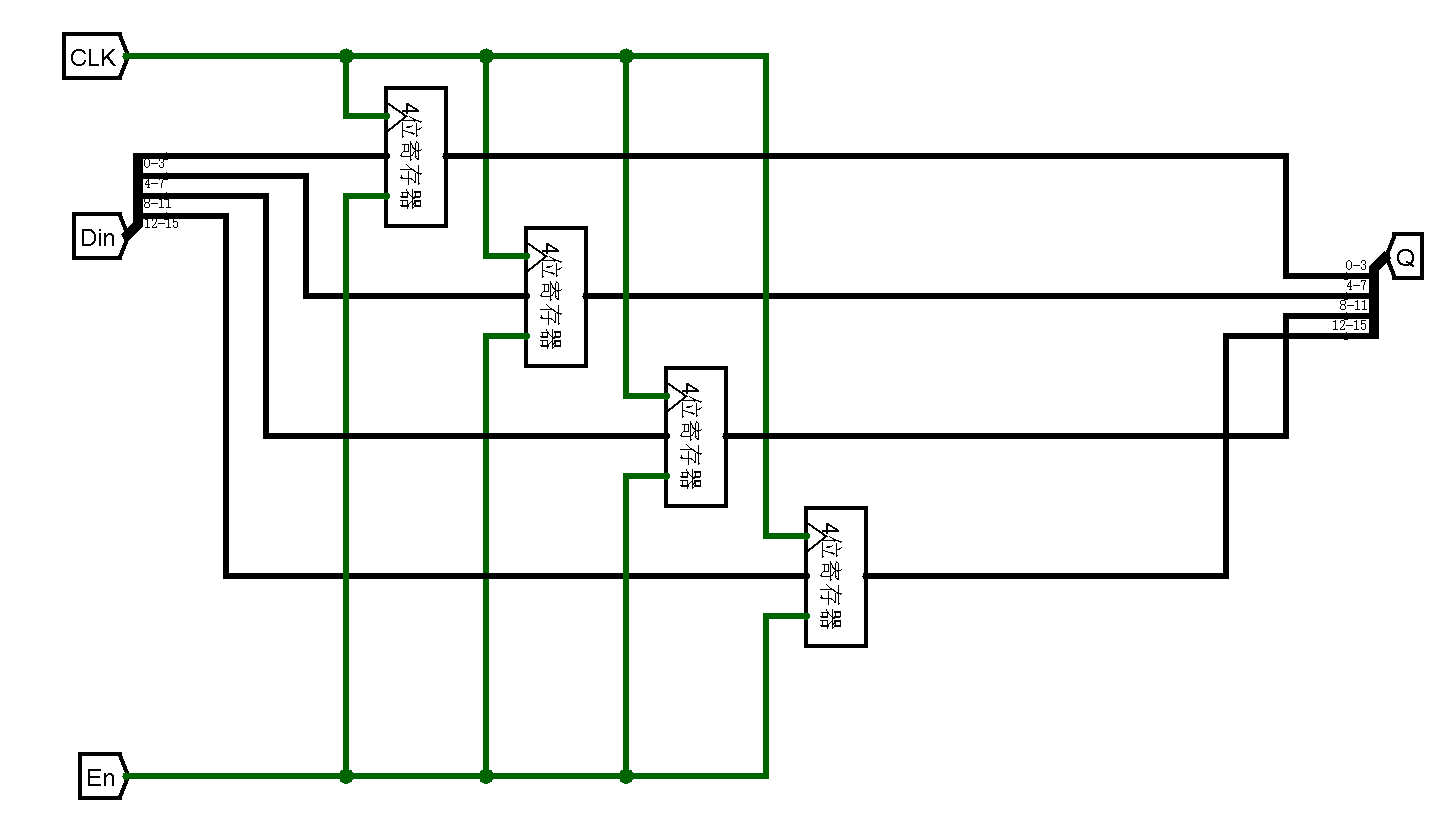


图1-11 并行加载寄存器（16位）电路图

### BCD计数器状态机设计

设计8421BCD码计数器状态机，电路引脚及功能如表1-6所示。

表1-6 BCD计数器状态机电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| S3~S0 | 输入 | 4 | 当前状态S |
| N3~N0 | 输出 | 4 | 次态输出N，N=S+1 |

现态为0时次态为1，现态为1时次态为2，以此类推，现态为9时次态为0（代表进位），由此补全“同步时序电路状态转换表.xlsx”的真值表，如图1-12所示。



图1-12 BCD计数器状态转换真值表

接下来自动生成状态转换函数：*N3 = ~S3 S2 S1 S0 + S3 ~S2 ~S1 ~S0；N2 = ~S3 ~S2 S1 S0 + ~S3 S2 ~S1 ~S0 + ~S3 S2 ~S1 S0 + ~S3 S2 S1 ~S0；N1 = ~S3 ~S2 ~S1 S0 + ~S3 ~S2 S1 ~S0 + ~S3 S2 ~S1 S0 + ~S3 S2 S1 ~S0；N0 = ~S3 ~S2 ~S1 ~S0 + ~S3 ~S2 S1 ~S0 + ~S3 S2 ~S1 ~S0 + ~S3 S2 S1 ~S0 + S3 ~S2 ~S1 ~S0*。自动生成电路图如图1-13所示。

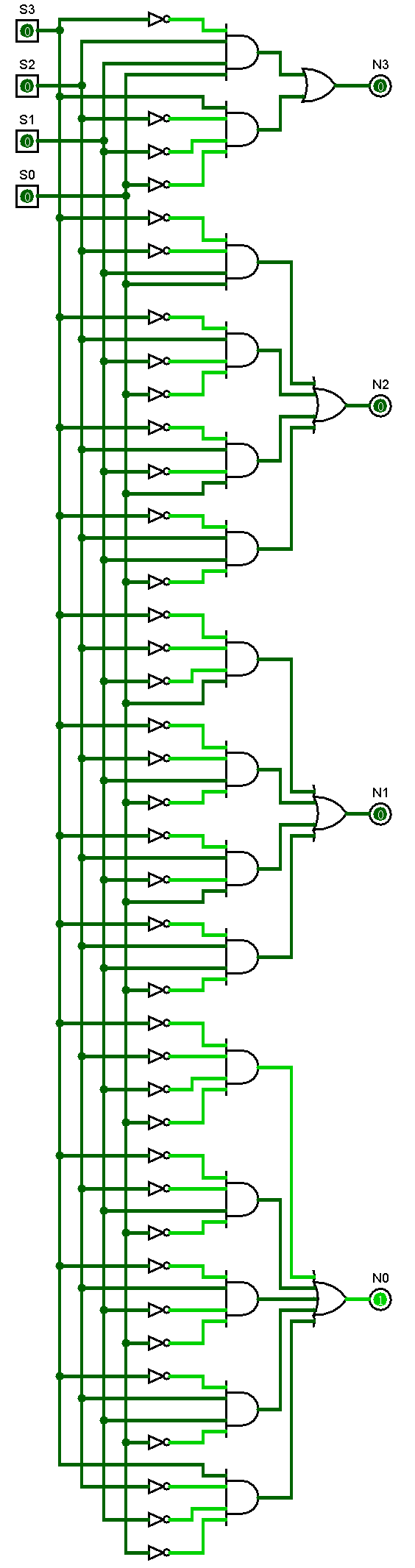


图1-13 BCD计数器状态机电路图

### BCD计数器输出函数设计

生成计数器的进位输出信号，该输出信号仅与状态信号有关（即Moore型输出），电路引脚及功能如表1-7所示。

表1-7 BCD计数器输出函数电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| S3~S0 | 输入 | 4 | 当前状态S |
| Cout | 输出 | 1 | 进位输出，S=9时，Cout=1 |

当S=9（S3=1，S2=0，S1=0，S0=1）时，输出为1，其余时刻均输出为0，画出电路图如图1-14所示。

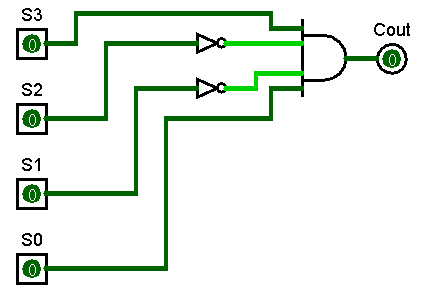


图1-14 BCD计数器输出函数电路图

### BCD计数器设计（1位十进制）

根据BCD计数器状态机和输出函数以及D触发器构建BCD计数器，电路引脚及功能如表1-8所示。

表1-8 BCD（1位十进制）计数器电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| CLK | 输入 | 1 | 时钟输入 |
| Rst | 输入 | 1 | 异步复位信号，为1时Q=0 |
| En | 输入 | 1 | 使能信号，为1时进行计数，Q=Q+1 |
| Q | 输出 | 4 | 计数器计数输出 |
| Cout | 输出 | 1 | 进位输出，计数到9时输出1 |

将Q连接到状态机上，得到转换后的次态，再将次态连接到4个D触发器，并让触发器的输出覆盖现态，同时将现态连接到封装好的输出函数模块，得到进位输出Cout，电路图如图1-15所示。

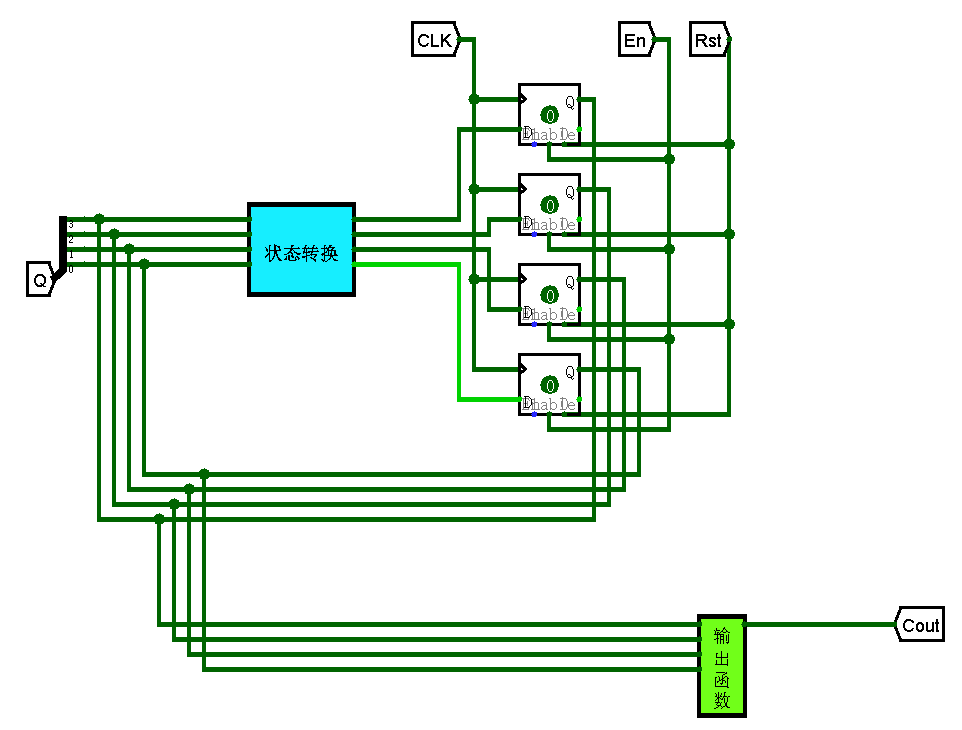


图1-15 BCD计数器（1位十进制）电路图

### 码表计数器设计（4位十进制）

利用前面的1位十进制计数器构建4位十进制计数器。不能和设计1位十进制计数器一样，使用相同的使能端，因为使能端为高电平时，时钟变化时，4个1位十进制计数器都会开始计数，这显然与功能不符。当前4位BCD计数器的使能端应由En和所有比它低的4位BCD计数器的进位输出进行与运算得出来的，这样低位计数器进位时，高位计数器才能加1。电路图如图1-16所示。

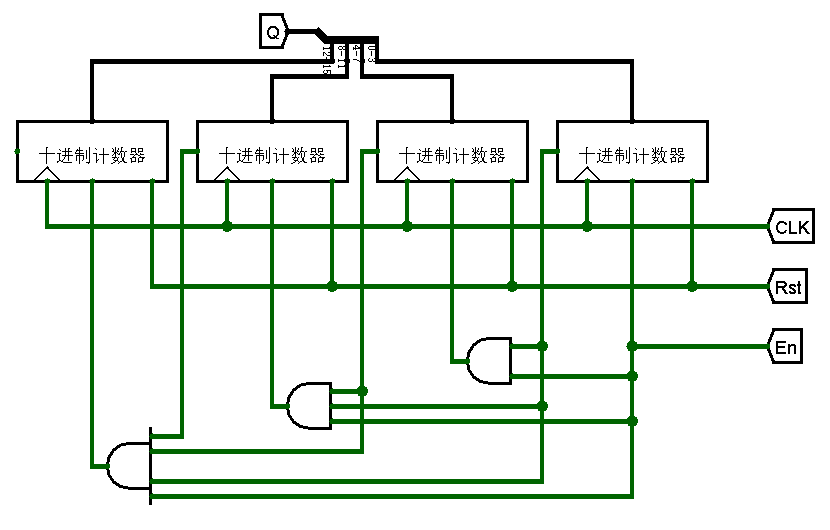


图1-16 码表计数器（4位十进制）电路图

### 码表显示驱动设计

利用7段数码管驱动电路级联构建4位十进制显示驱动电路，电路引脚及功能如表1-9所示。

表1-9 BCD码表显示驱动电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| Din | 输入 | 16 | 4位十进制（16位BCD） |
| DispInfo | 输出 | 32 | 4个8段（7段+小数点）数码驱动信号 |

用分线器将Din分成4个4位数据，再分别连接到4个数码管驱动模块。码表的显示格式位XX.XX，因此第二个数码管驱动的第8段驱动信号位1，实现小数点功能，电路图如图1-17所示。

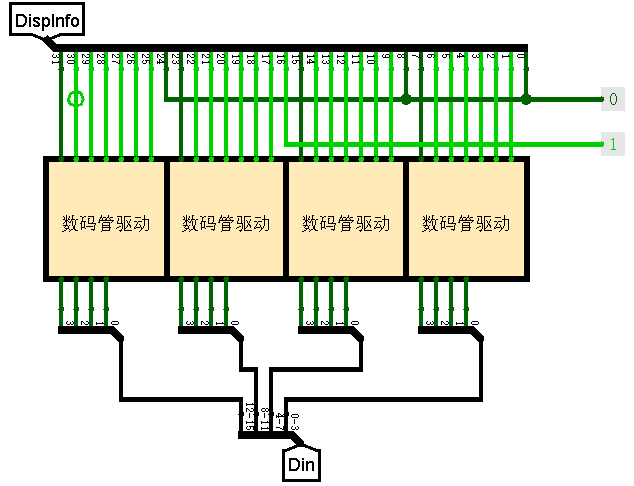


图1-17 码表显示驱动电路图

### 码表控制器状态机设计

设计码表状态转换的组合逻辑，电路引脚及功能如表1-10所示。

表1-10 码表控制器状态机电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| start | 输入 | 1 | 开始计时 |
| stop | 输入 | 1 | 停止计时 |
| store | 输入 | 1 | 存储计时记录 |
| reset | 输入 | 1 | 计时复位，记录恢复99.99 |
| newrecord | 输入 | 1 | 新最好成绩记录 |
| S2~S0 | 输入 | 3 | 现态S |
| N2~N0 | 输出 | 3 | 次态N |

分析码表可能的状态，根据其功能绘制码表状态转移图如图1-18所示。

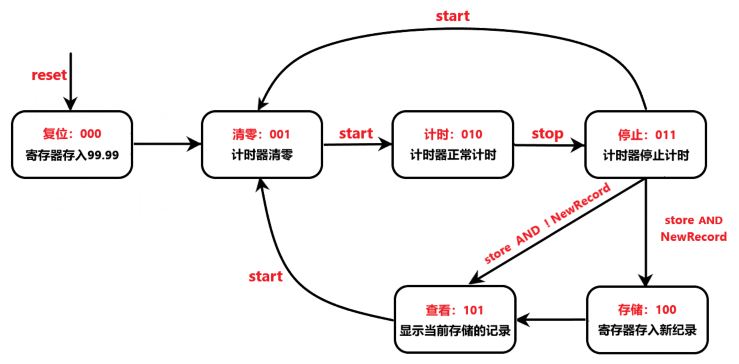


图1-18 码表控制器状态机状态转移图

根据状态转移图完成状态转换真值表，填表过程有一些隐含条件需要注意：只有接收reset信号才会转移到状态0；状态为4时，若不给出任何指令，则会直接跳转到状态5。真值表如图1-19所示。

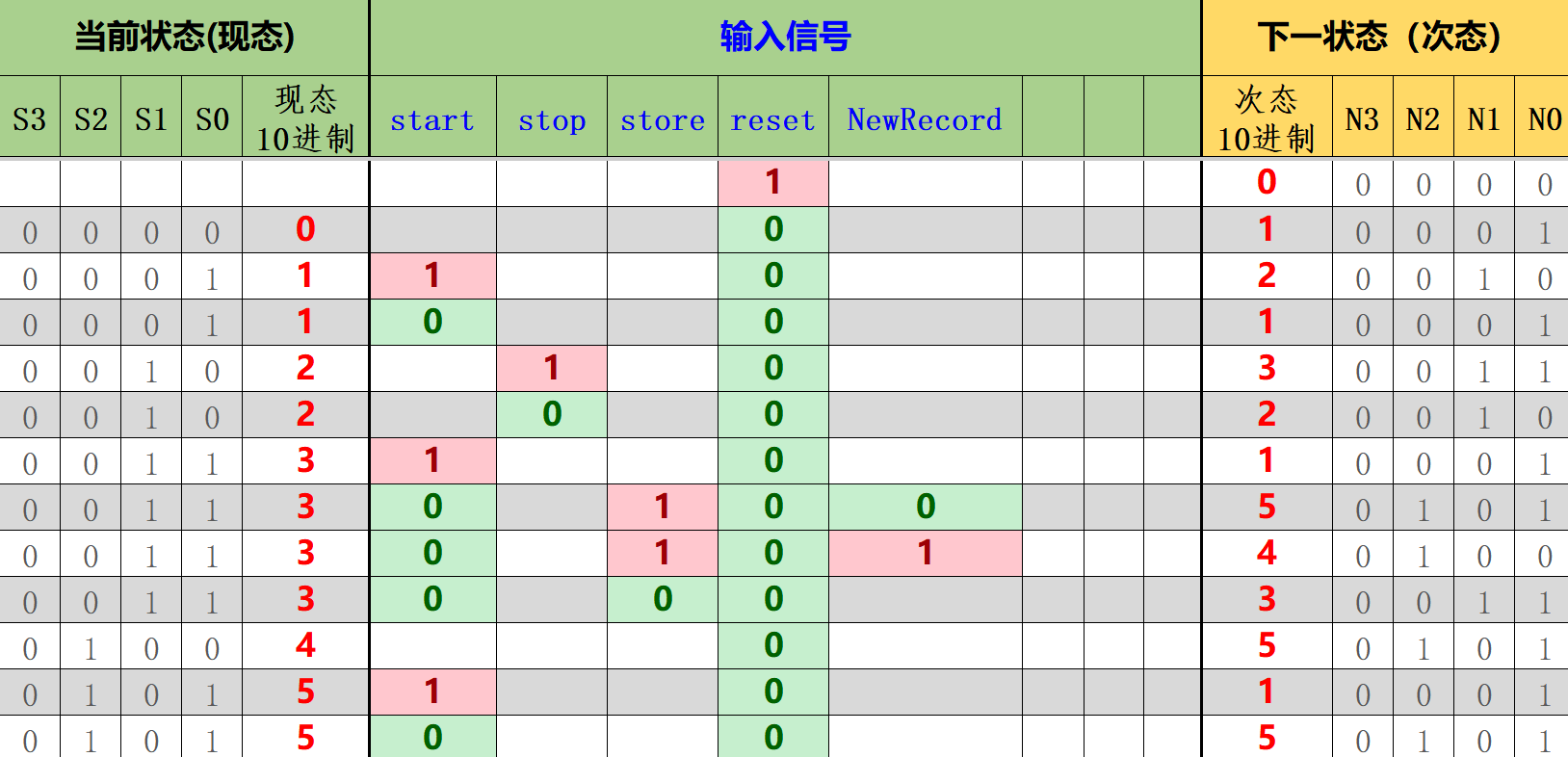


图1-19 码表控制器状态转换真值表

自动生成状态转换函数：*N2 = ~S2 S1 S0 ~start store ~reset ~NewRecord + ~S2 S1 S0 ~start store ~reset NewRecord + S2 ~S1 ~S0 ~reset + S2 ~S1 S0 ~start ~reset；N1 = ~S2 ~S1 S0 start ~reset + ~S2 S1 ~S0 stop ~reset + ~S2 S1 ~S0 ~stop ~reset + ~S2 S1 S0 ~start ~store ~reset；N0 = ~S2 ~S1 ~S0 ~reset + ~S2 ~S1 S0 ~start ~reset + ~S2 S1 ~S0 stop ~reset + ~S2 S1 S0 start ~reset + ~S2 S1 S0 ~start store ~reset ~NewRecord + ~S2 S1 S0 ~start ~store ~reset + S2 ~S1 ~S0 ~reset + S2 ~S1 S0 start ~reset + S2 ~S1 S0 ~start ~reset。*

电路图如图1-20所示。

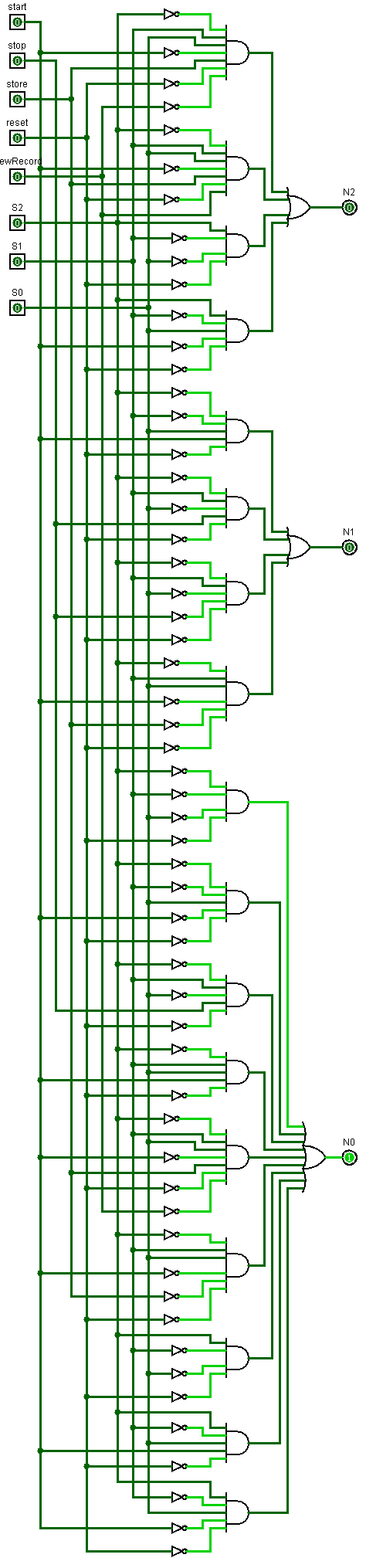


图1-20 码表控制器状态转换电路图

### 码表控制器输出函数设计

设计码表控制器的输出函数组合逻辑（Moore 型输出），需要结合运动码表系统数据通路完成码表控制器输出函数真值表，电路引脚及功能如表1-11所示

表1-11 码表制器输出函数电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| S2~S0 | 输入 | 3 | 现态S |
| SDsel | 输出 | 1 | 好成绩记录的选择 |
| SDen | 输出 | 1 | 保存成绩记录的寄存器使能 |
| DPsel | 输出 | 1 | 显示成绩记录的选择 |
| TMen | 输出 | 1 | 码表计时器使能 |
| TMreset | 输出 | 1 | 码表计时器复位 |

完成输出函数真值表，如图1-21所示。

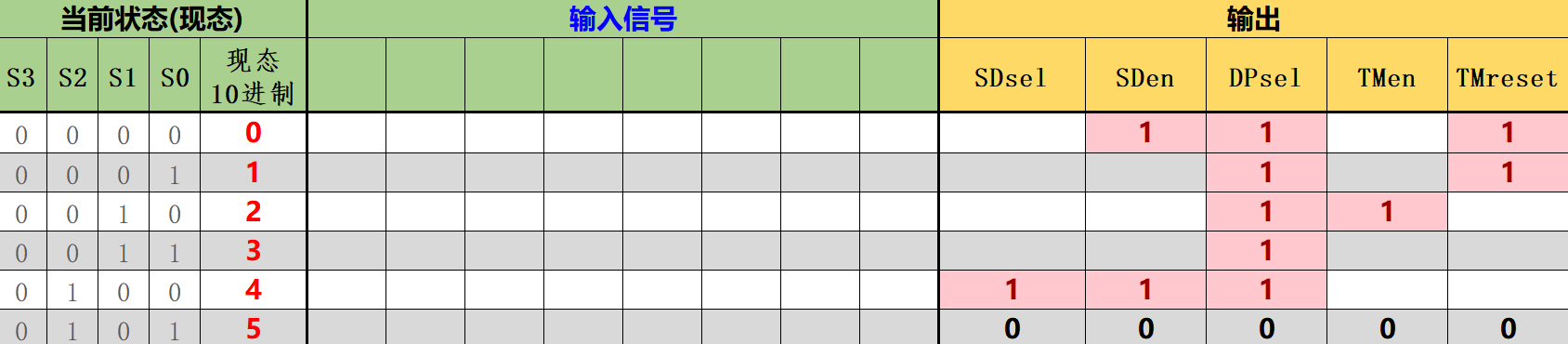


图1-21 码表控制器输出函数真值表

自动生成输出函数逻辑表达式：*SDsel = S2 ~S1 ~S0；SDen = ~S2 ~S1 ~S0 + S2 ~S1 ~S0；DPsel = ~S2 ~S1 ~S0 + ~S2 ~S1 S0 + ~S2 S1 ~S0 + ~S2 S1 S0 + S2 ~S1 ~S0；TMen = ~S2 S1 ~S0；TMreset = ~S2 ~S1 ~S0 + ~S2 ~S1 S0*。电路图如图1-22所示。

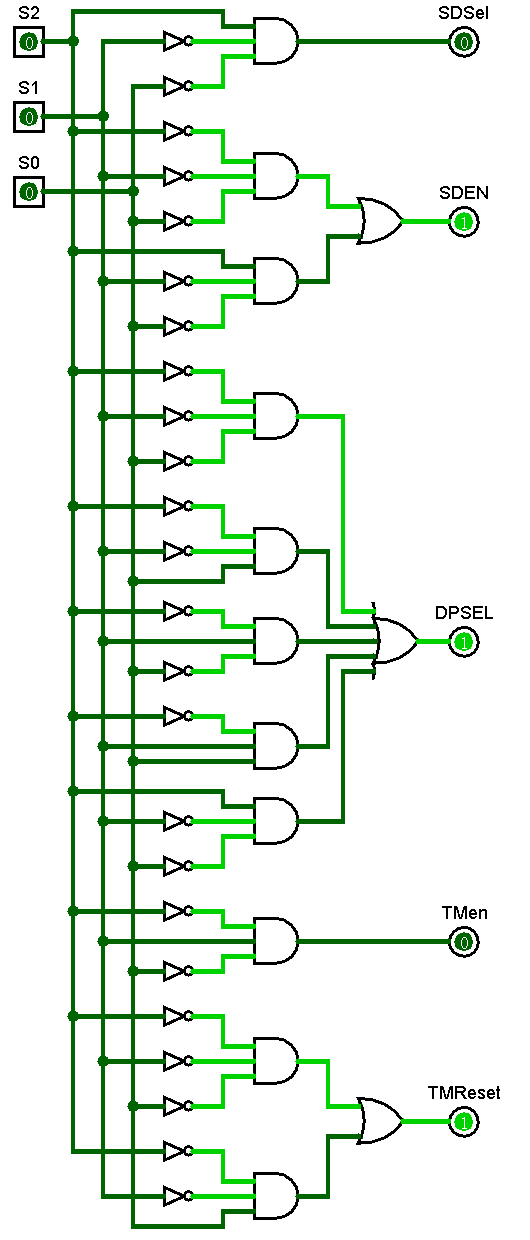


图1-22 码表控制器输出函数电路图

### 码表控制器设计

将码表控制器状态机和输出函数组合到一起，封装为码表控制器如图1-23所示。

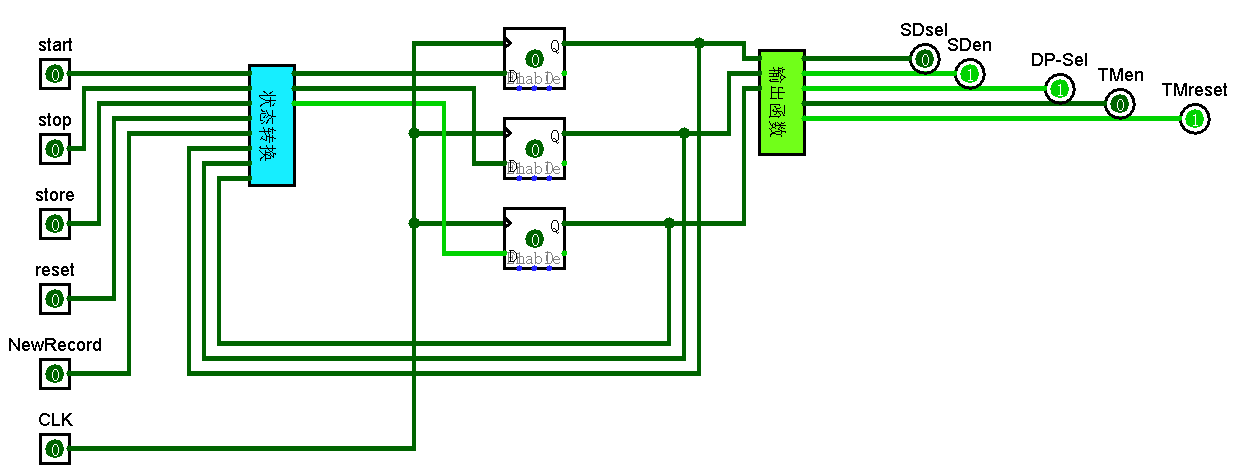


图1-23 码表控制器电路图

### 运动码表数据通路设计（系统集成）

利用前面设计好的选择器、比较器、寄存器、计数器、码表控制器等部件构建运动码表系统数据通路。注意初始记录99.99和最好成绩记录必须送入选择器0号端口。电路引脚及功能如表1-1所示。

将输入Start1、Stop1、Store1、Reset1、CLK、NewRecord都连接到码表控制器上；将时钟CLK连接到16位并行数据寄存器SD和计时器TM上；将码表控制器的输出TM-EN和TM-Reset连接到计时器TM上，确保计时器的正常运行和复位功能;将码表控制器的输出DP-SEL连接到码表显示信号的选择器上，确定显示的计时记录来源；将码表控制器的输出SD-EN连接到16位寄存器上，确定寄存器是否保存最好成绩记录；将码表控制器的输出SD-SEL连接到最好成绩记录的选择器上，确定是否选择最

短计时记录。计时器的输出需要连接到最好成绩记录的选择器和码表显示信号的选择器上，从而为各类功能提供当前的计时记录。电路图如图1-24所示。

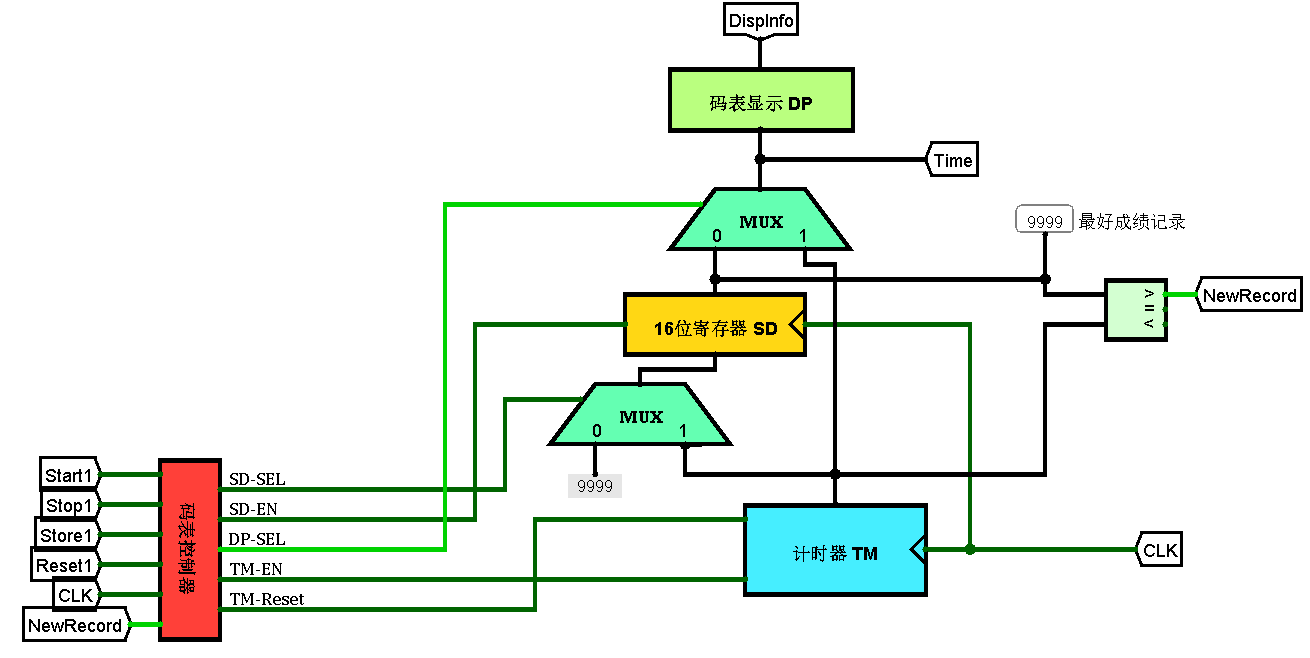


图1-24 码表控制器数据通路电路图

## 实验步骤

（1） 仔细阅读头歌平台各关卡中的实验提示，并了解各部分子电路的功能。观

看实验指导，整理好原理，并与所学建立联系。

（2） 在Logisim和对应的excel表格中完成相应的内容。

（3） 用记事本打开StopWatch.circ，将代码复制粘贴到头歌平台上进行测评。

## 故障与调试

### 4位十进制计数器技术问题

**故障现象：**计数时进位出现错误，从0090直接到0191再到0292。

**原因分析：**高位数字的使能信号输入有误，如图1-25所示，直接使用低位数字的进位输出作为使能，这样会导致进位只与相邻的低位有关，显然不符合设计要求。

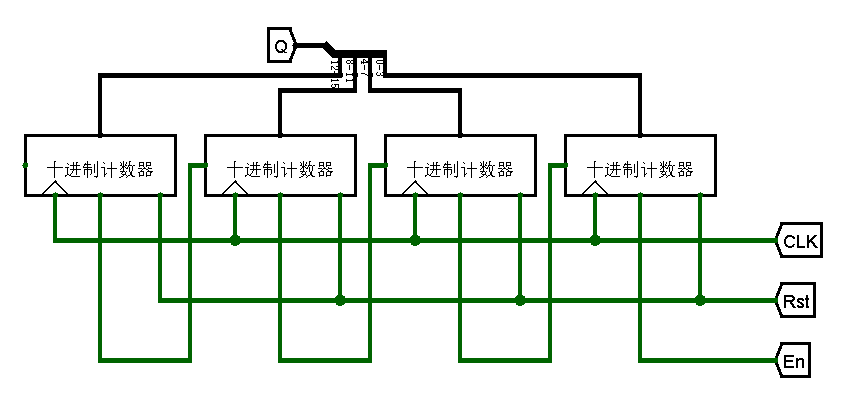


图1-25 4位十进制计数器错误计数电路图

**解决方案：**将高位数字的使能信号改为由En和所有比它低的十进制计数器的进位输出进行与运算。这样当低位都为9时，高位才会加1。

## 测试与分析

### 2路选择器测试

运行“2 路选择器自动测试”电路，成功通过显示“PASS”，如图 1-26 所示。

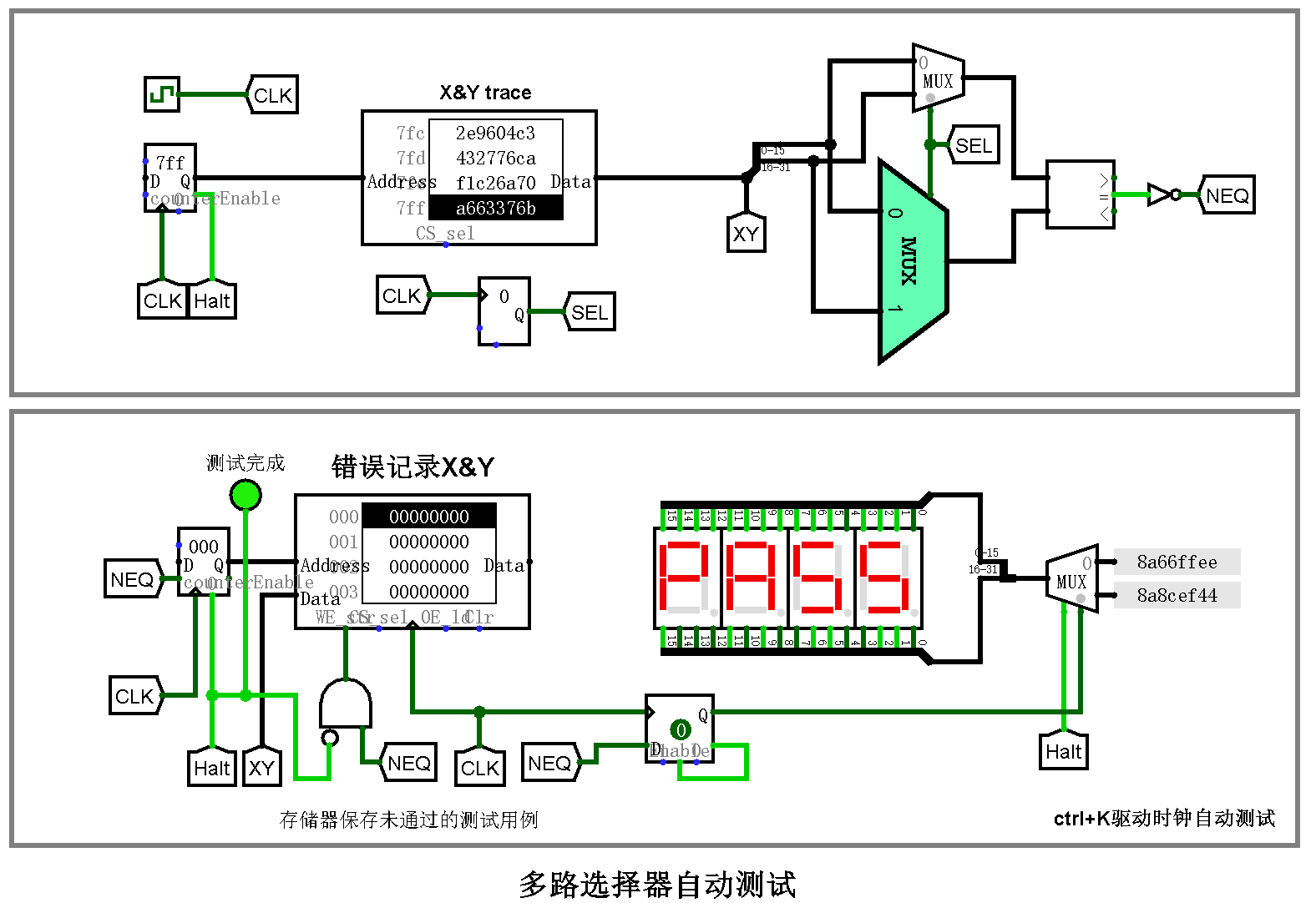


图1-26 2路选择器测试

### 16位无符号比较器测试

运行“16位无符号比较器测试”电路，成功通过显示“PASS”，如图 1-27 所示。

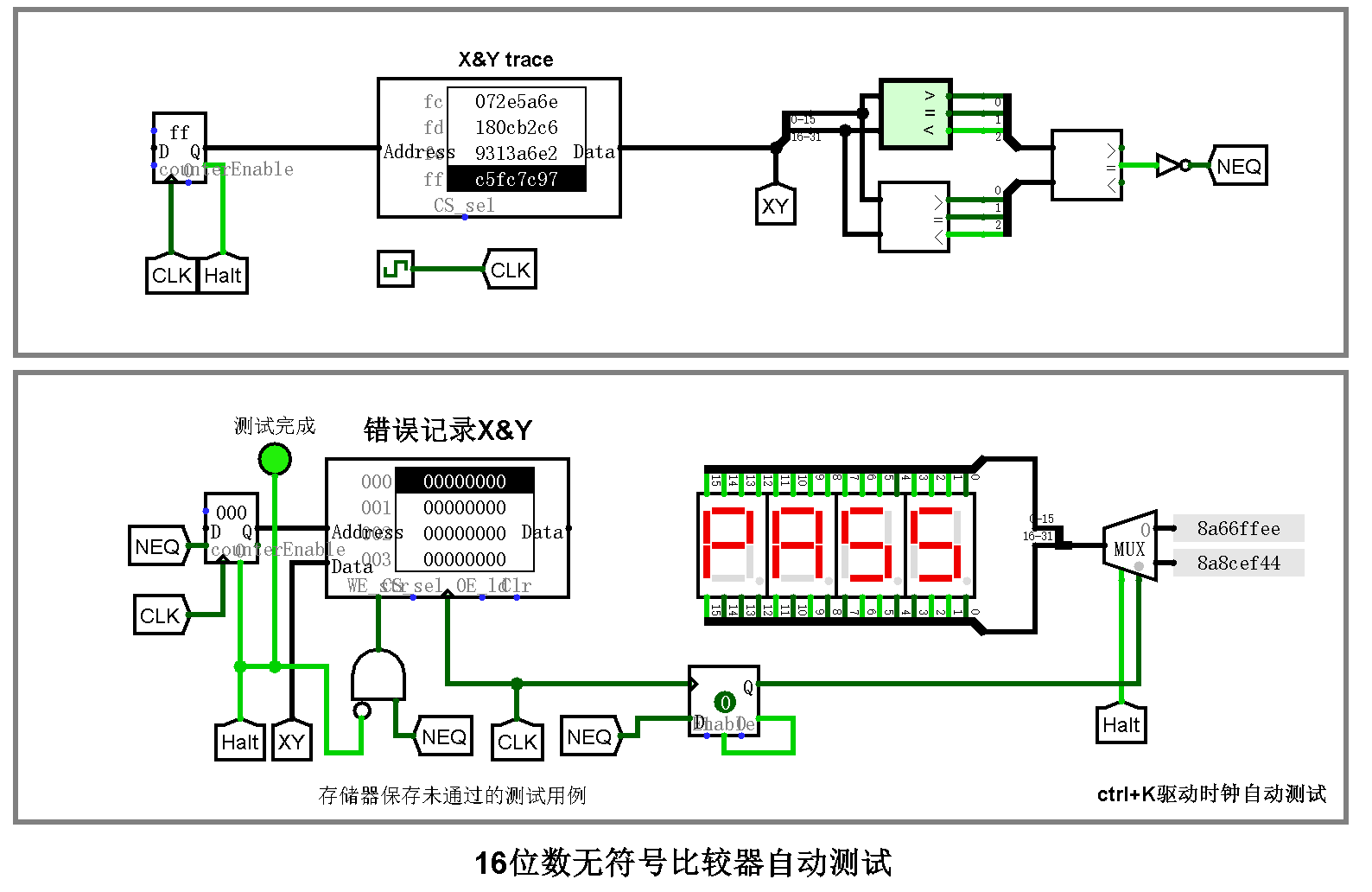


图1-27 16位无符号比较器测试

### 码表计数器测试

运行“码表计数器自动测试”电路，将时钟频率设为 100Hz，按 Ctrl+K 键启

动码表，计时功能正常，如图 1-28 所示。

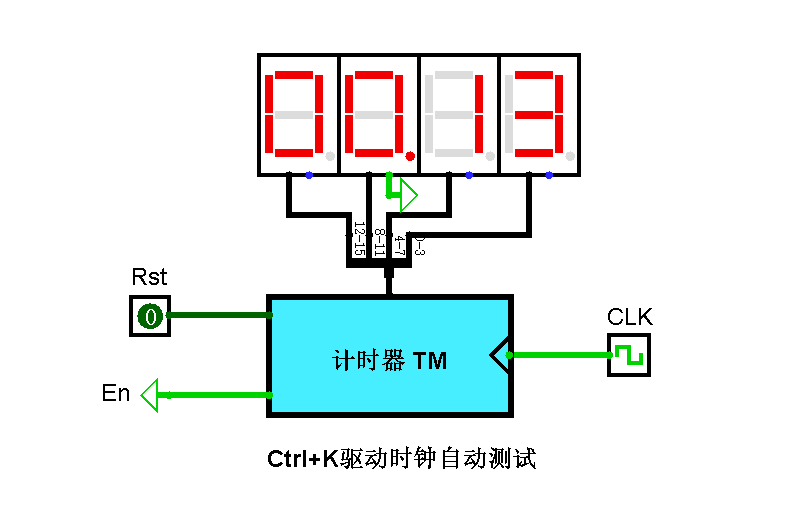


图1-28 码表计数器测试

### 运动码表Start功能测试

点击Start按钮，码表开始计时，如图1-29所示。

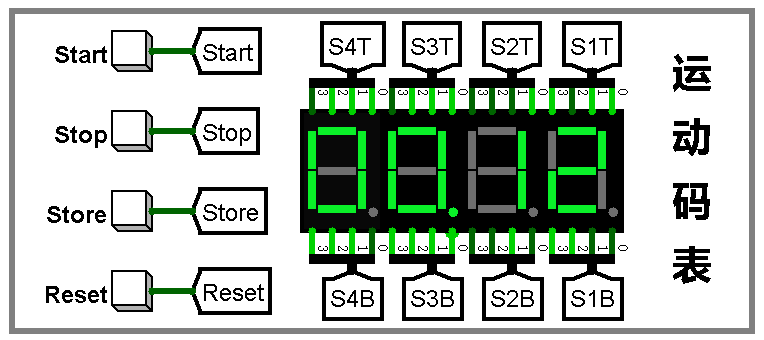


图1-29 码表Start功能器测试

### 运动码表Stop、Store功能测试

点击Stop按钮，计时停止，此时码表显示“01.28”。再按Store按钮，可以发现最好成绩更新为0128，如图1-30所示。

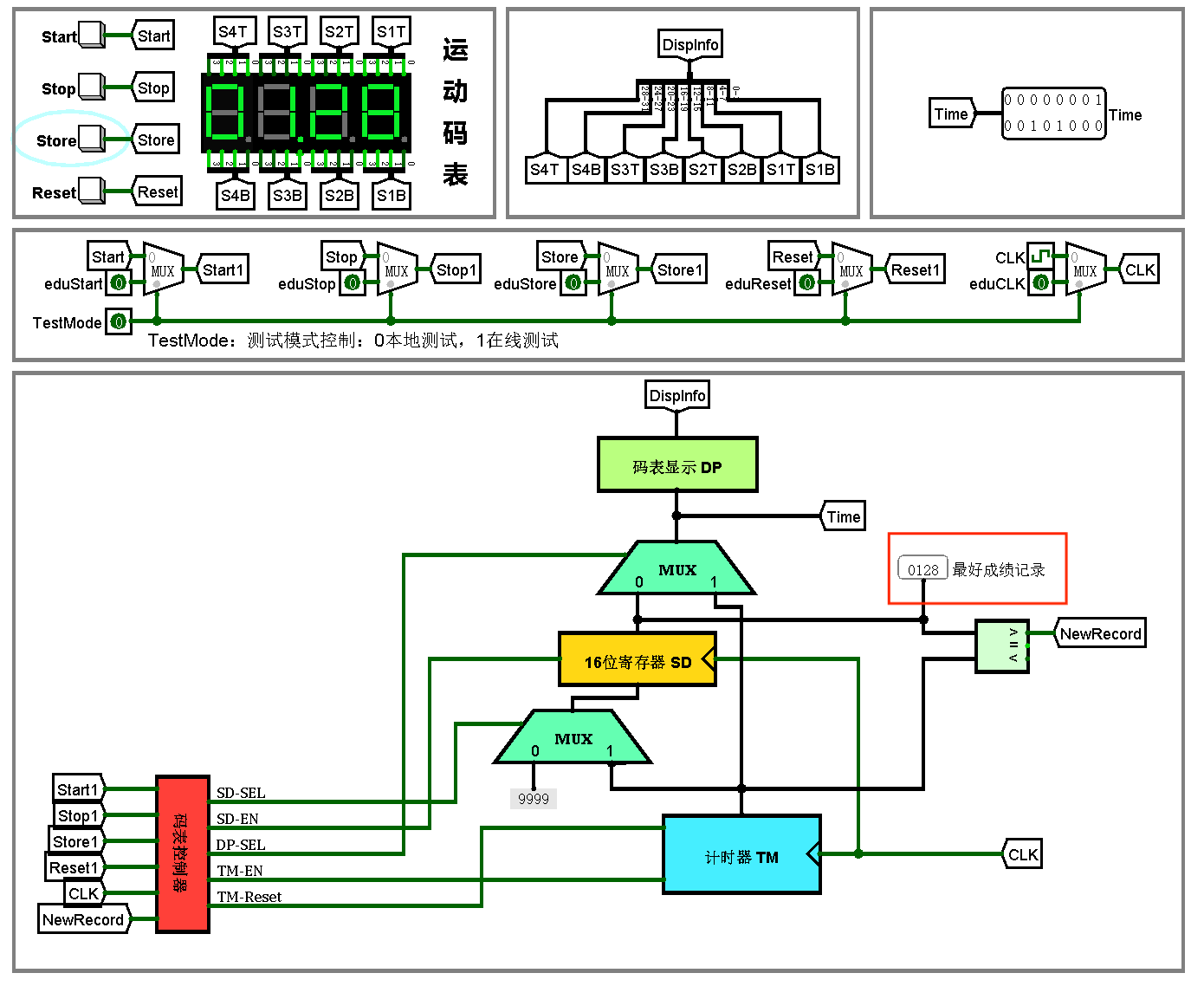


图1-30 码表Stop、Store功能器测试

### 运动码表Reset功能测试

点击Reset按钮，码表复位为“00.00”，最好成绩变为9999，如图1-31所示。

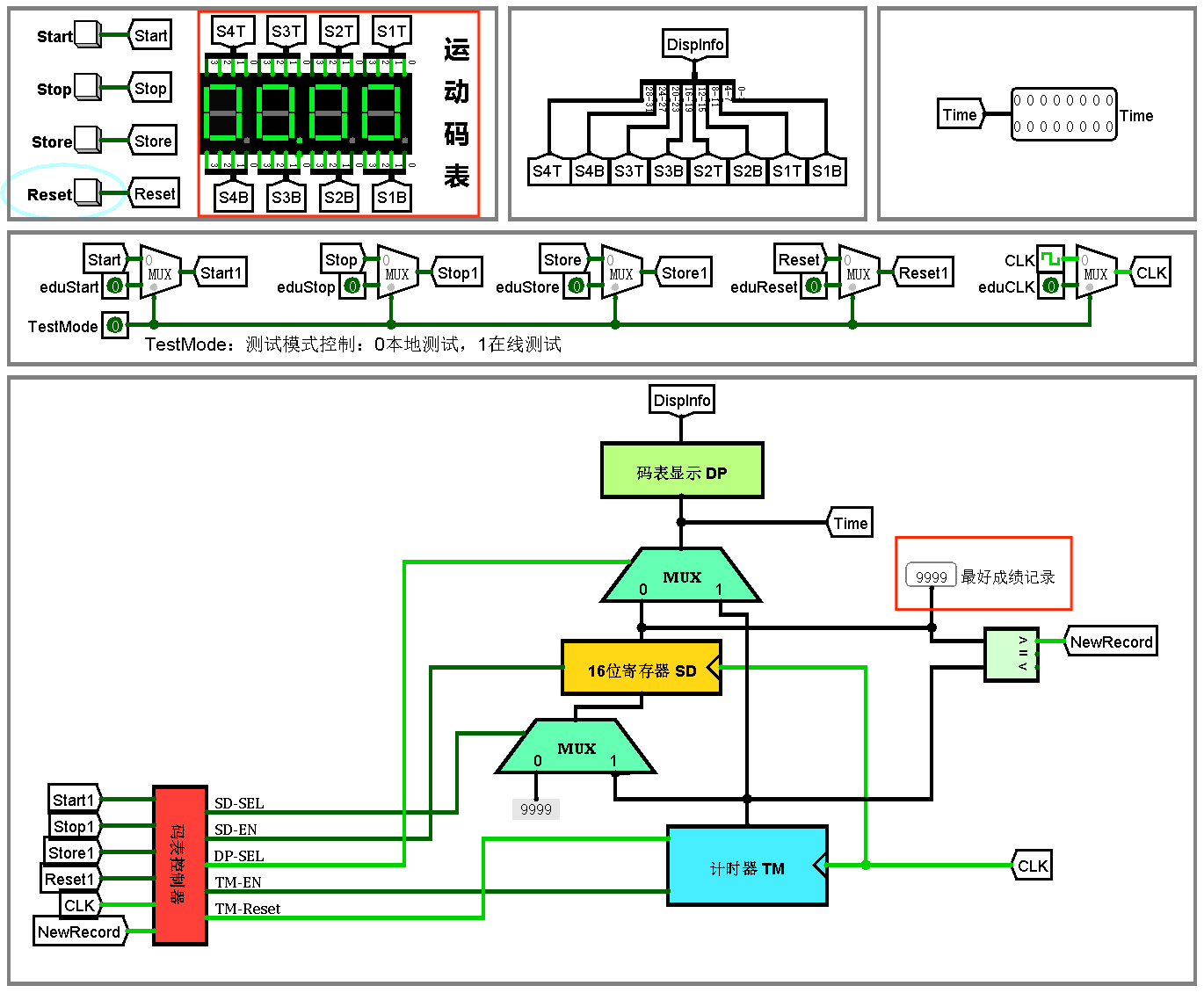


图1-31 码表Reset功能器测试

## 实验总结

本次实验完成了运动码表系统的全流程设计与实现，通过模块化构建核心功能电路（包括7段数码管驱动、多路选择器、无符号比较器、并行寄存器及BCD计数器），并基于状态机理论设计了码表控制器，最终集成数据通路实现计时启停、记录存储与动态显示功能。重点攻克了4位计数器高位使能逻辑的级联优化问题，确保进位同步性；通过本地功能测试与平台自动化验证，系统精准响应Start（清零计时）、Stop（冻结显示）、Store（记录比对更新）及Reset（复位初始化）指令，各项指标均符合设计要求，为硬件系统设计提供了完整实践范例。

## 实验心得

通过对码表基础模块的逐层构建与运动码表数据通路的系统设计，最终实现了一个功能完整的计时系统。这一过程让我深刻体会到数字电路设计的核心逻辑：从时序控制到组合逻辑，再到模块化集成，每一步都需要严谨的理论支撑和细致的实践验证。特别是在设计4位十进制计数器时，高位使能信号的级联逻辑曾导致进位异常，通过重新梳理使能链，我才真正理解了级联电路中信号传递的同步性要求。这些挑战培养了我从故障现象反向追踪设计缺陷的系统性思维能力。这次实践不仅夯实了我的数字电路基础，更让我体会到"纸上理论"与"板上信号"之间的桥梁——唯有通过反复调试，才能将抽象的逻辑转化为可靠的硬件生命。

# CPU设计实验

## 设计要求

利用运算器实验，存储系统实验中构建的运算器、寄存器文件、存储系统等部件以及Logisim中其它功能部件，构建一个32位MIPS CPU处理器，包括单周期硬布线CPU、多周期微程序CPU以及多周期硬布线CPU，该处理器应支持核心指令集中列出的所有指令，见表2-1。最终设计完成的CPU应能运行标准测试程序。

表2-1 核心指令集

|  |  |  |
| --- | --- | --- |
| 指令 | 指令格式 | 功能描述 |
| Add | add $rd, $rs, $rt | R[$rd]←R[$rs]+R[$rt] |
| Set Less Than | slt $rd, $rs, $rt | R[$rd]←R[$rs]<R[$rt] |
| Add Immediate | addi $rt, $rs, imm | R[$rt]←R[$rs]+SignExt(imm) |
| Load Word | lw $rt, imm($rs) | R[$rt]←Mem(R[$rs]+SignExt(imm)) |
| Store Word | sw $rt, imm($rs) | Mem(R[$rs]+SignExt(imm))←R[$rt] |
| Branch on Equal | beq $rs, $rt, imm | If(R[$rs]=R[$rt]) PC←PC+SignExt({imm,00}) |
| Branch on Not Equal | bne $rs, $rt, imm | If(R[$rs]!=R[$rt]) PC←PC+SignExt({imm,00}) |
| syscall（display or exit） | syscall | 系统调用，这里用于停机 |

## 方案设计

### MIPS指令译码器

将32位输入操作码用分线器接出，最高6位为操作码op，21-25位为rs寄存器编号，16-20位为rt寄存器编号、11-15位为rd寄存器编号， 0-5位为功能码func；取低16位，作为I型指令中的立即数操作数。**单周期和多周期指令译码方式相同**。MIPS指令译码器引脚如表2-2所示。

表2-2 MIPS指令译码器引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| OP | 输入 | 6 | MIPS指令中的OP字段 |
| FUNC | 输入 | 6 | R型指令中的funct字段 |
| LW | 输出 | 1 | 当前指令为lw指令则输出1 |
| SW | 输出 | 1 | 当前指令为sw指令则输出1 |
| BEQ | 输出 | 1 | 当前指令为beq指令则输出1 |
| BNE | 输出 | 1 | 当前指令为bne指令则输出1 |
| ADDI | 输出 | 1 | 当前指令为addi指令则输出1 |
| ADD | 输出 | 1 | 当前指令为add指令则输出1 |
| SLT | 输出 | 1 | 当前指令为slt指令则输出1 |
| SysCall | 输出 | 1 | 当前指令为syscall指令则输出1 |
| R\_TYPE | 输出 | 1 | R型运算类指令译码信号 |

查阅MIPS32指令手册可知各指令对应的OP与FUNC，再使用多个比较器判断当前指令类型，如图2-1所示。

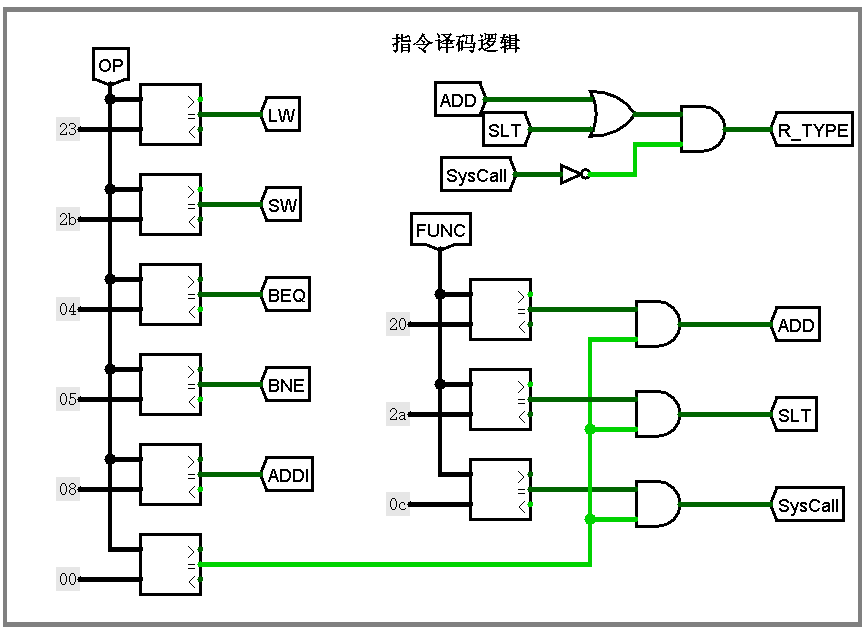


图2-1 MIPS指令译码器

### 单周期MIPS CPU设计

首先设计单周期CPU的数据通路，使用“指令-部件输入-部件输出”邻接关系表，绘制主要功能部件输入，若有多个输入来源，则需要控制信号配合多路选择器进行选择，具体如图2-2所示。

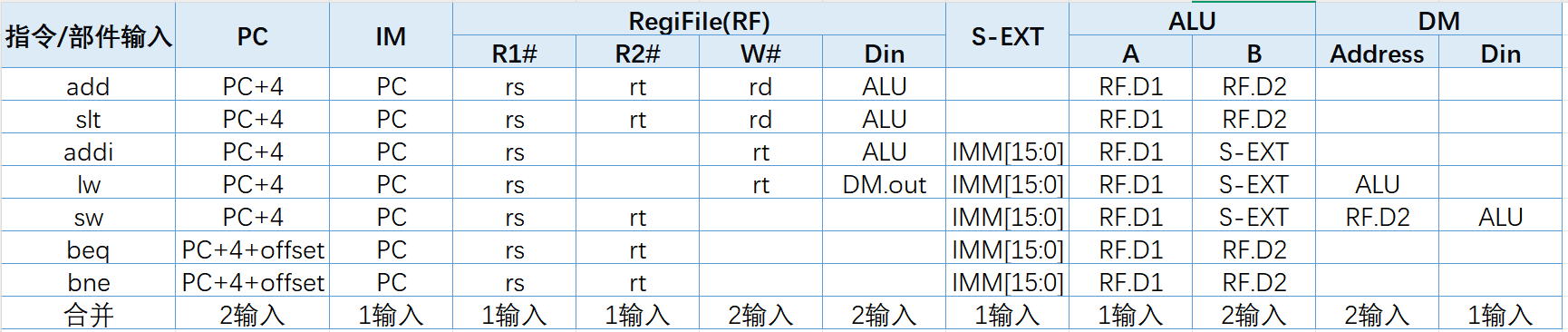


图2-2 单周期CPU数据通路

接着设计单周期硬布线控制器，先根据指令译码信号中的SLT信号确定相应的ALU\_OP信号。当SLT=0时，为加法运算，查表可知ALU\_OP为0x5；当SLT=1时，为比较运算查表可知ALU\_OP为0xb。因此ALU\_OP控制逻辑如图2-3所示。

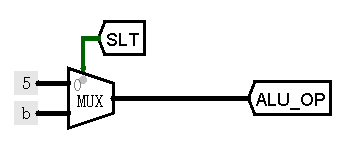


图2-3 单周期ALU\_OP控制逻辑

再根据R\_TYPE、ADDI、LW、SW、BEQ、BNE、SysCall译码信号实现控制器输出控制信号的逻辑，各控制信号产生条件如图2-4所示。



图2-4 控制信号产生条件

由控制信号产生条件连接电路图如图2-5所示。

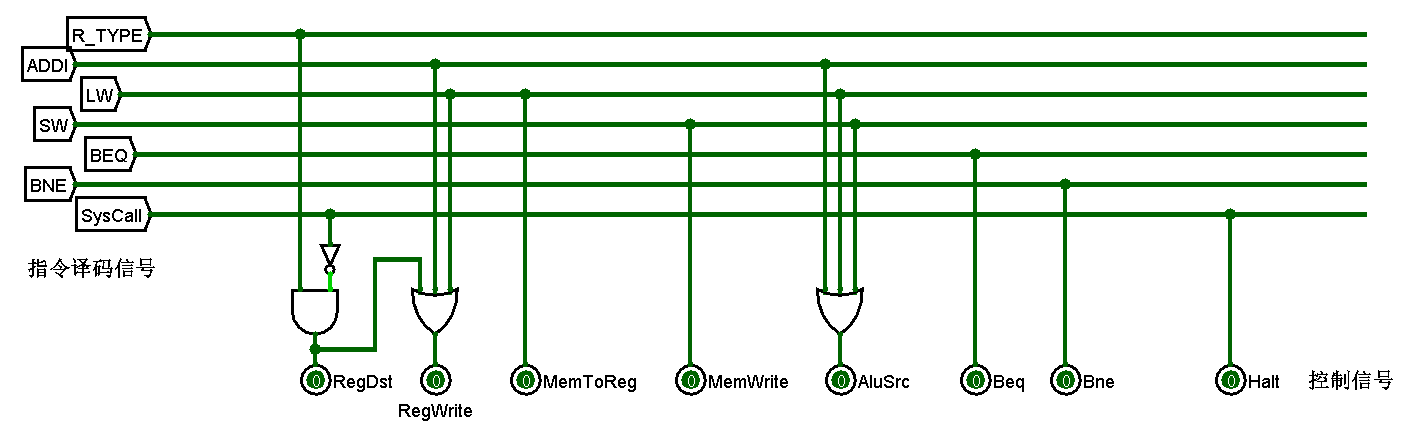


图2-5 控制信号电路图

通过上面的分析，我们已经得到单周期CPU的所有功能部件，我们也知道了所有指令的数据通路，以及如何产生控制信号控制指令数据的传递，因此把这些部件和控制信号连接起来就得到了单周期CPU总体结构图，如图2-6所示。

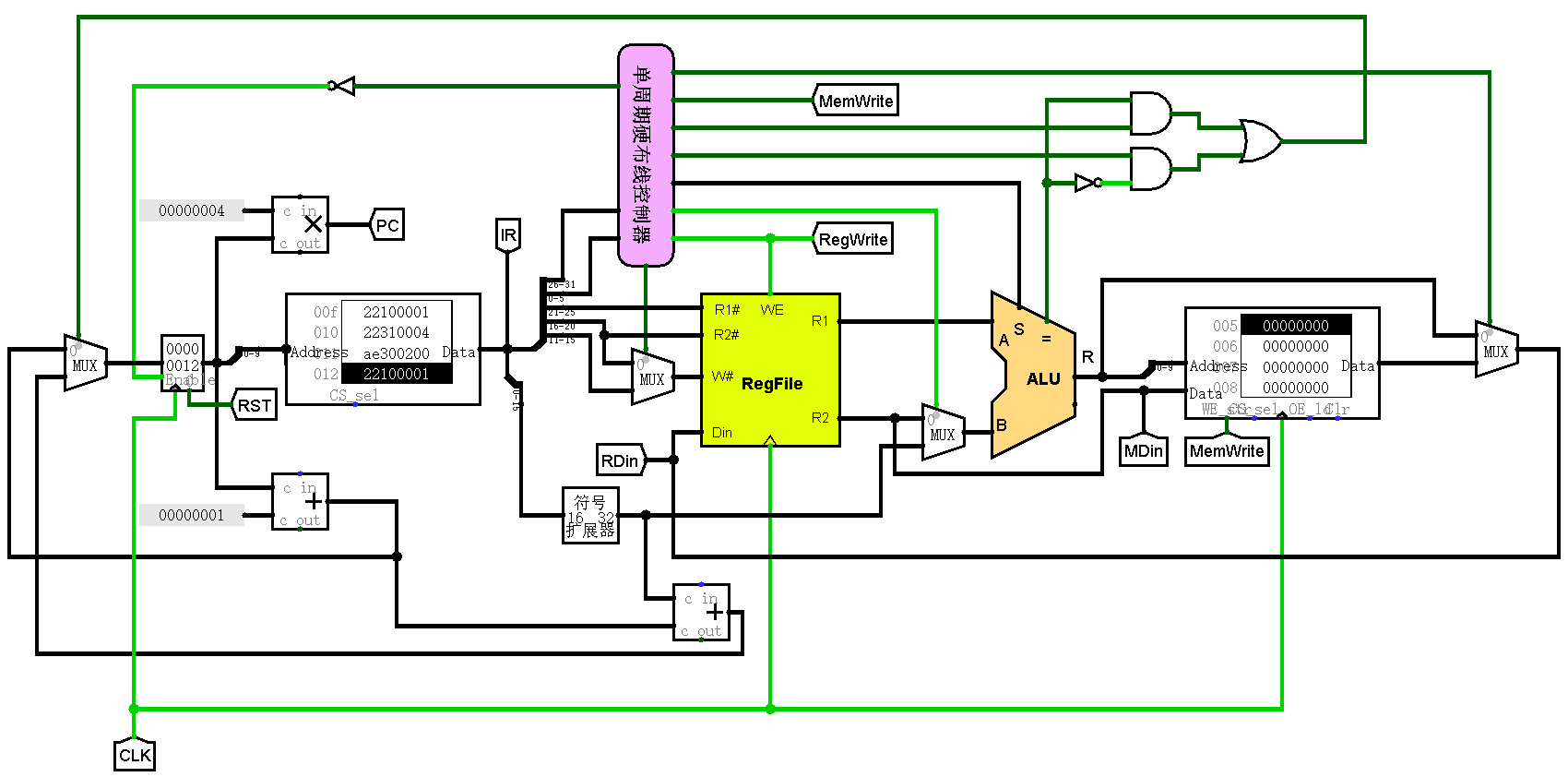


图2-6 单周期MIPS CPU总体结构图

### 从单周期到多周期

多周期CPU控制下，不同指令对应不同的时钟周期数，不能像单周期控制器一次性给出一条指令对应的所有控制信号，而每一条指令可拆解为三部分“取指、译码、执行”，其中取指和译码两个阶段对应的数据通路相同，分析每条指令执行阶段需要的时钟周期及控制信号如下各表所示（A、B、C为寄存器，保存RegiFile和ALU的输出）。

表2-3 取指阶段操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 取指令 | IR←(MEM[PC])  PC←(PC)+4 |
| 译码及取操作数 | A←(R[IR[25:21]])  B←(R[IR[20:16]])  C←(PC)+(S-EXT(IR[15:0])<<2) |

表2-4 add指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 加运算 | C←(A)+(B) |
| 写回 | R[IR[15:11]]←(C) |

表2-5 slt指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 比较运算 | C←((A)<(B)) |
| 写回 | R[IR[15:11]]←(C) |

表2-6 lw指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 计算地址 | C←(A)+ S-EXT(IR[15:0]) |
| 访存 | DR←(MEM[PC]) |
| 写回 | R[IR[20:16]]←(DR) |

表2-7 sw指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 计算地址 | C←(A)+ S-EXT(IR[15:0]) |
| 访存 | DR←(MEM[PC]) |

表2-8 beq指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 送目标地址 | If(A==B) PC←(C) |

表2-9 bne指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 送目标地址 | If(A!=B) PC←(C) |

表2-10 addi指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 加运算 | C←(A)+ S-EXT(IR[15:0]) |
| 写回 | R[IR[20:16]]←(C) |

表2-11 syscall指令执行操作流程

|  |  |
| --- | --- |
| 指令阶段 | 操作流程 |
| 空操作，停机 | 锁住PC |

### 微程序地址转移逻辑设计

根据每个指令需要的不同周期数以及每个周期的操作，可以给出所有指令不同阶段的控制信号表，如图2-7所示，控制信号说明如下：

1. PCWrite：PC写使能控制，取指令周期，分支指令执行；
2. IorD：指令还是数据，0表示指令，1表示数据；
3. Irwrite：指令寄存器写使能；
4. MemWrite：写内存控制信号；
5. MemRead：读内存控制信号；
6. Beq：beq指令译码信号；
7. Bne：bne指令译码信号；
8. PcSrc：PC输入源，0表示顺序寻址，1表示跳跃寻址；
9. AluControl：ALU控制信号，即加法或比较运算；
10. AluSrcA：ALU第一输入选择，PC还是寄存器输出；
11. AluSrcB：ALU第二输入选择，R型指令输入为寄存器输出，取指阶段为4，sw、lw、addi指令为立即数，跳转指令bne、beq相应的偏移地址；
12. RegWrite：寄存器RegiFile写使能；
13. RegDst：RegiFile的第二个寄存器编号由R型指令rd给出；
14. MemToReg：lw指令，写入寄存器的数据来自存储器。

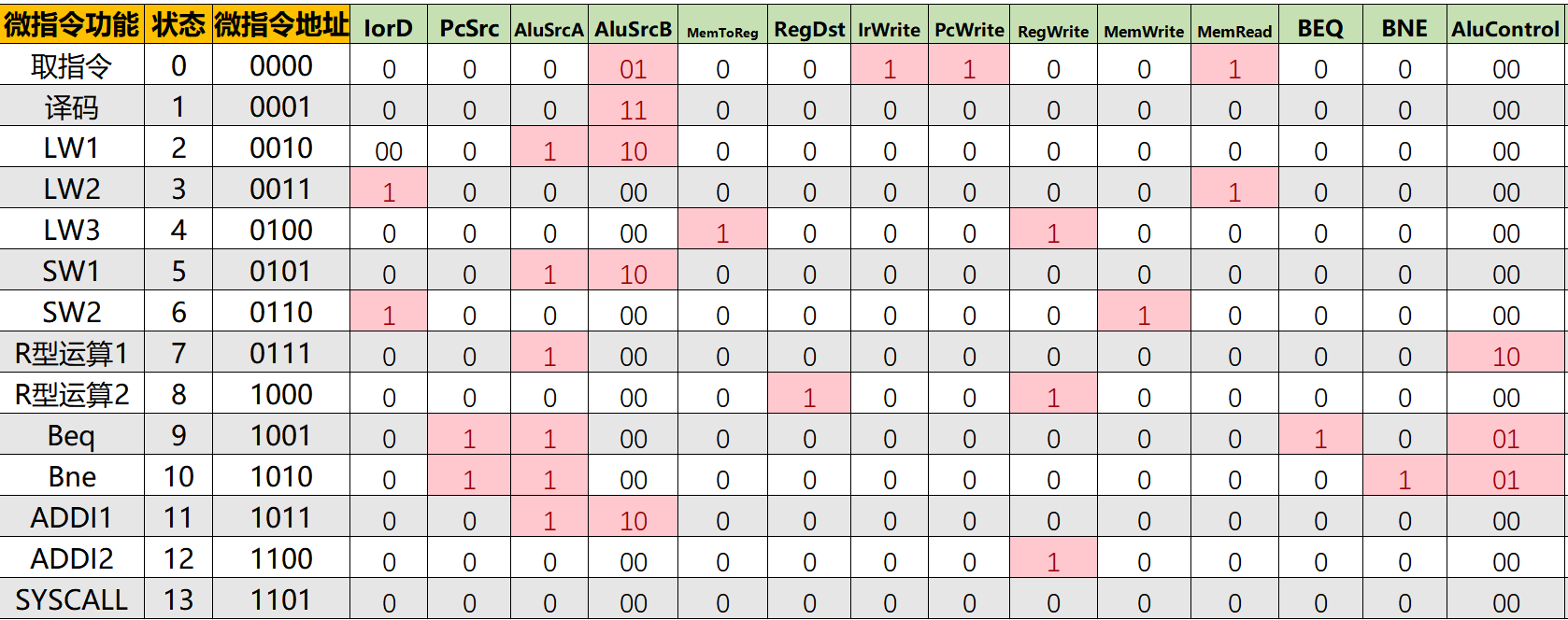


图2-7 指令控制信号

微指令地址转移逻辑采用下址字段法，配合判断状态P，译码阶段P为1，表明控制存储器的地址为指令的第一个阶段的微程序地址；执行阶段P为0，表明下一条微指令地址由下址字段给出，得出译码阶段指令的入口地址如图2-8所示。



图2-8 微程序入口地址

由此得到入口地址逻辑表达式：

*S3 = ADDI + BEQ + BNE + SYSCALL*

*S2 = R\_Type + SW + SYSCALL*

*S1 = R\_Type + ADDI + LW + BNE*

*S0 = R\_Type + ADDI + SW + BEQ + SYSCALL*

译码阶段微程序入口地址逻辑电路图如图2-9所示。

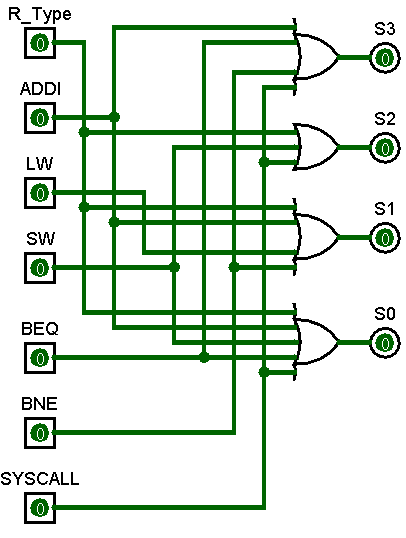


图2-9 微程序入口地址电路图

执行8条指令需要的控制信号为16位编码，且采用下址字段法，下址所需编码位数和微指令地址位数相同，即4位，一位判断标志位P，P为1表示译码阶段，控制存储器地址由指令的微指令转移逻辑产生，P为0表示控制存储器地址由下址字段产生，即表示指令的不同执行阶段（每条指令对应多条微指令），因此微指令编码21位，地址位编码4位，由此生成的微指令转移逻辑及编码如图2-10所示。



图2-10 微指令转移逻辑及编码

### 多周期MIPS微程序CPU设计

首先设计多周期微程序控制器。其中指令译码器与2.2.1相同，不再赘述。根据指令译码信号进行微程序地址转移并得到微命令控制信号，电路图如图2-11所示。

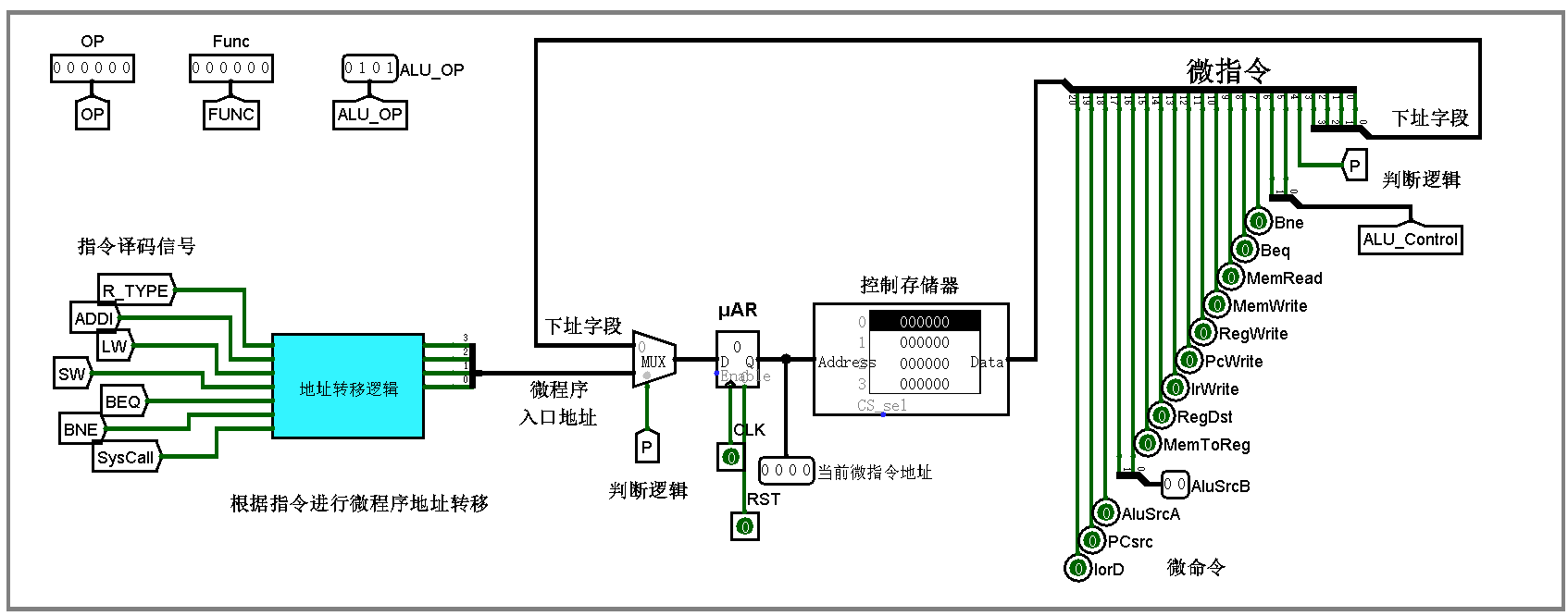


图2-11 微命令产生逻辑

再根据微命令控制信号中的ALU\_Control信号确定相应的ALU\_OP信号。当ALU\_Control=00时，为加法运算，查表可知ALU\_OP为0x5；当ALU\_Control=01时，为减法运算，查表可知ALU\_OP为0x6；当ALU\_Control=10时，ALU\_OP由FUNC决定，若为SLT指令，则进行比较运算，查表可知ALU\_OP为0xb，否则为加法运算。因此ALU\_OP控制逻辑如图2-12所示。

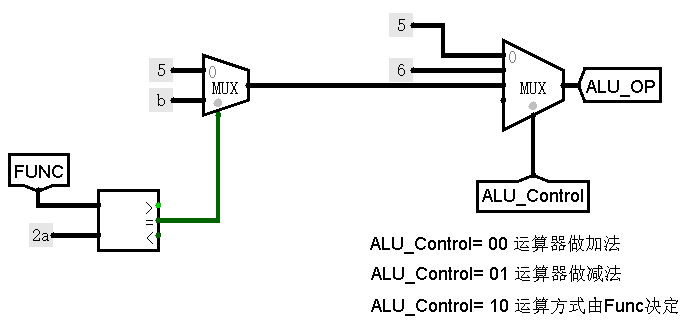


图2-12 多周期ALU\_OP控制逻辑

将以上部件和控制信号连接起来，得到多周期MIPS微程序CPU如图2-13所示。

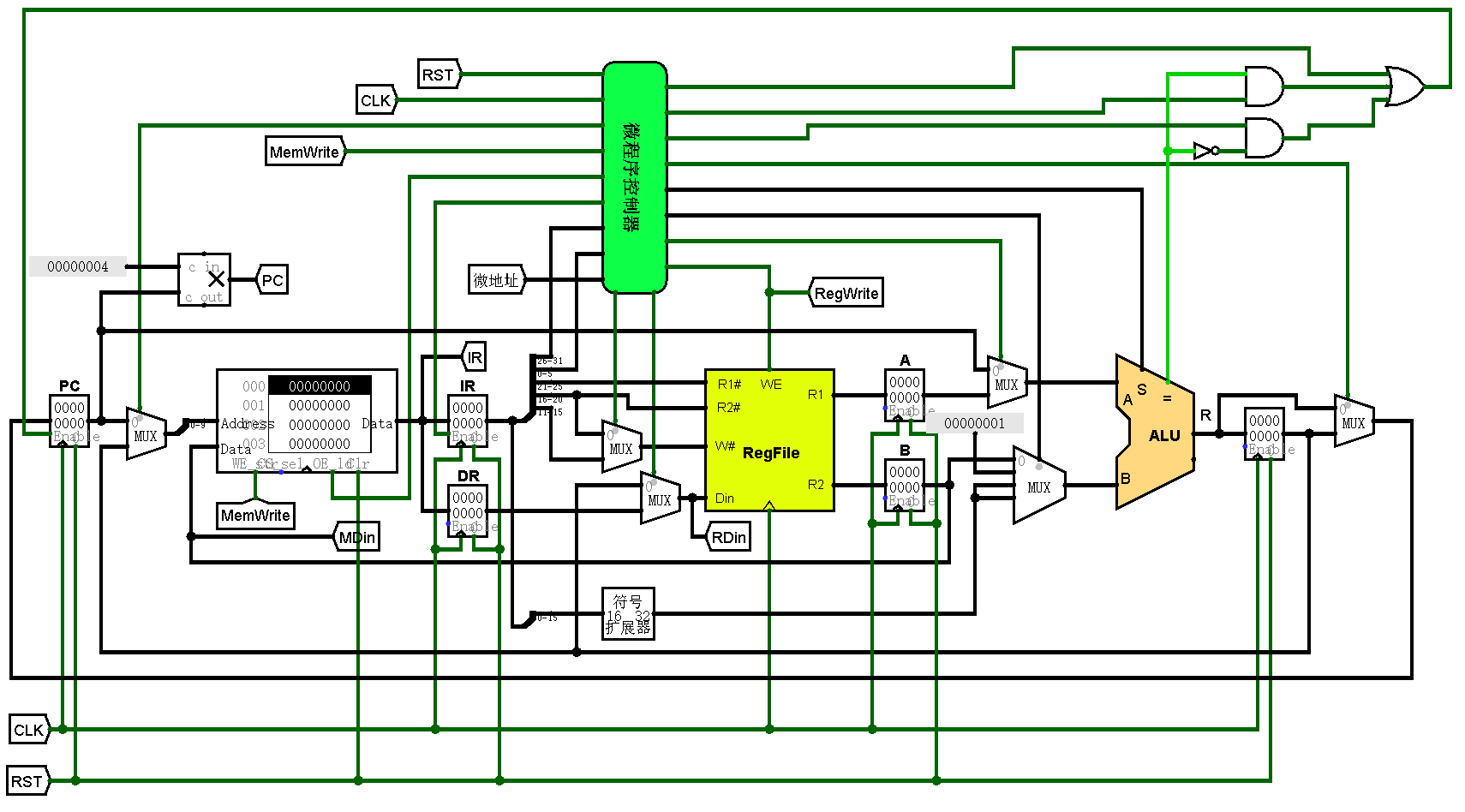


图2-13 多周期MIPS微程序CPU总体结构图

### 多周期硬布线控制器状态机设计

状态编码与微程序状态控制器相同，且状态转移图如图2-14所示。

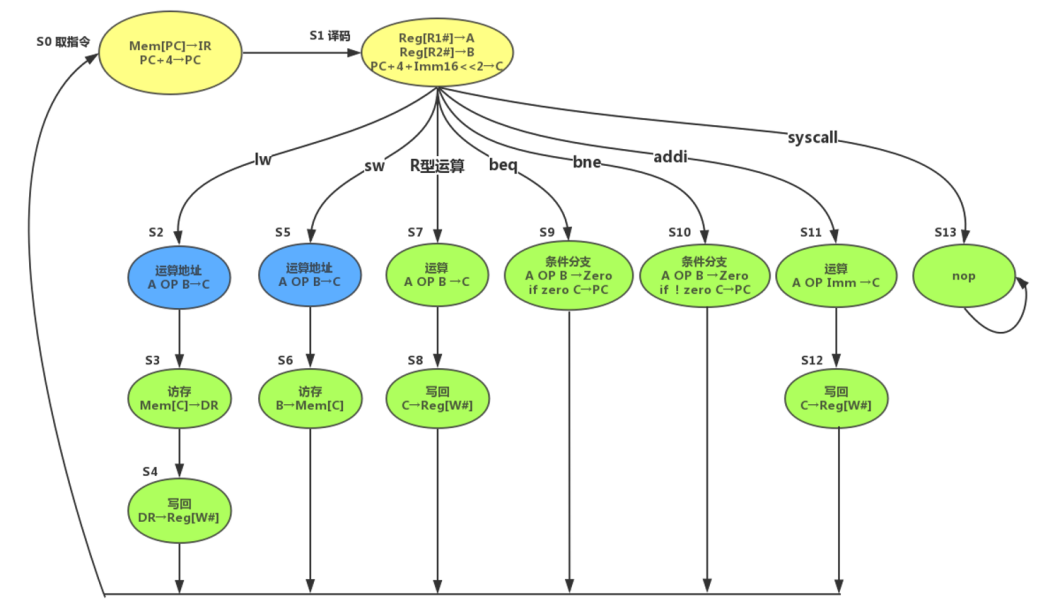


图2-14 多周期硬布线控制器状态转移图

使用FSM有限状态机来实现状态转移，次态即下一条微指令在控制存储器中的地址，FSM采用纯组合逻辑电路实现，FSM状态转换如图2-15所示。

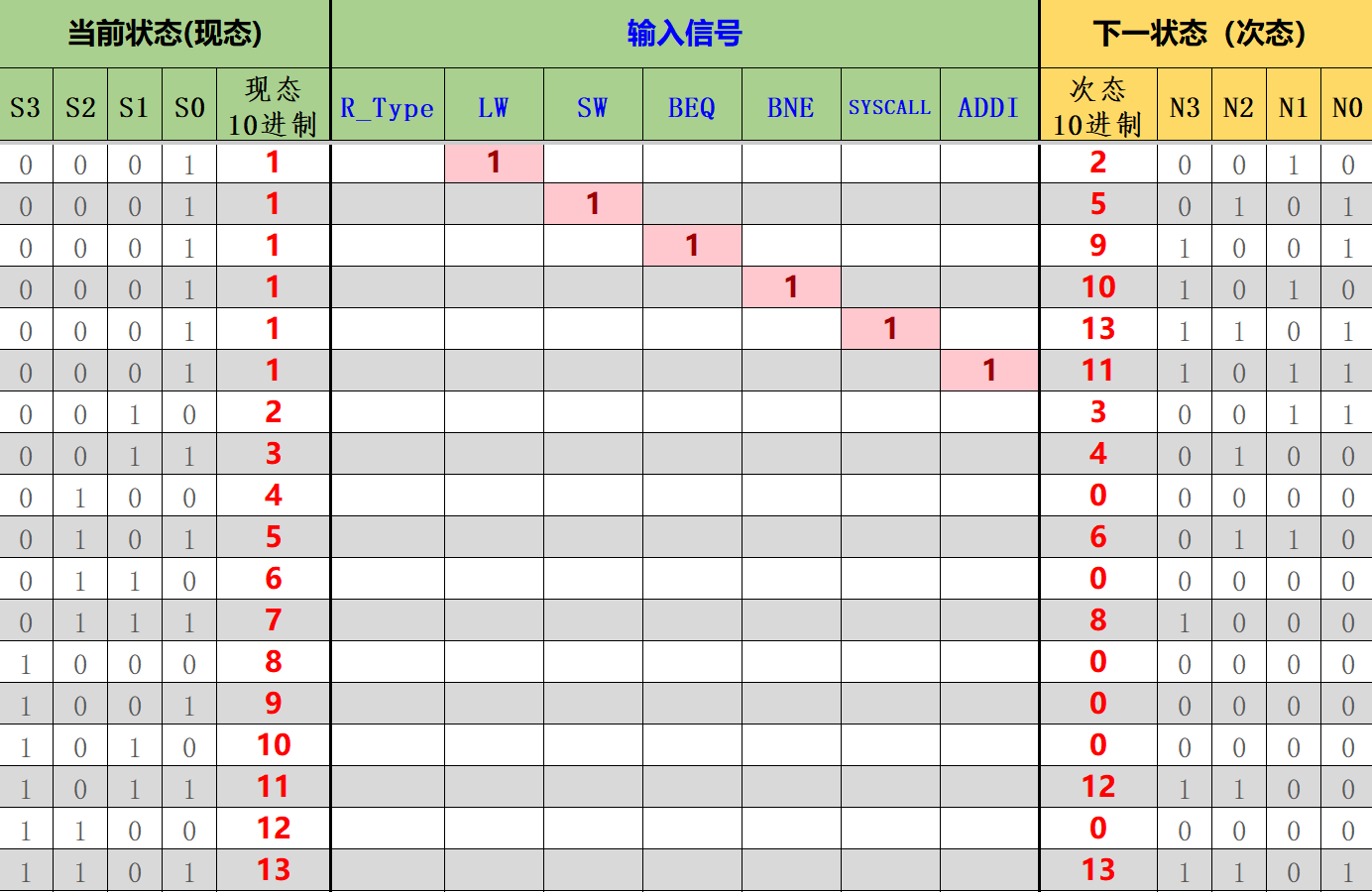


图2-15 FSM状态转换

生成状态转换逻辑表达式：

*N3 = ~S3 ~S2 ~S1 S0 BEQ + ~S3 ~S2 ~S1 S0 BNE + ~S3 ~S2 ~S1 S0 SYSCALL + ~S3 ~S2 ~S1 S0 ADDI + ~S3 S2 S1 S0 + S3 ~S2 S1 S0 + S3 S2 ~S1 S0*

*N2 = ~S3 ~S2 ~S1 S0 R\_Type + ~S3 ~S2 ~S1 S0 SW + ~S3 ~S2 ~S1 S0 SYSCALL + ~S3 ~S2 S1 S0 + ~S3 S2 ~S1 S0 + S3 ~S2 S1 S0 + S3 S2 ~S1 S0*

*N1 = ~S3 ~S2 ~S1 S0 R\_Type + ~S3 ~S2 ~S1 S0 LW + ~S3 ~S2 ~S1 S0 BNE + ~S3 ~S2 ~S1 S0 ADDI + ~S3 ~S2 S1 ~S0 + ~S3 S2 ~S1 S0*

*N0 = ~S3 ~S2 ~S1 ~S0 + ~S3 ~S2 ~S1 S0 R\_Type + ~S3 ~S2 ~S1 S0 SW + ~S3 ~S2 ~S1 S0 BEQ + ~S3 ~S2 ~S1 S0 SYSCALL + ~S3 ~S2 ~S1 S0 ADDI + ~S3 ~S2 S1 ~S0 + S3 S2 ~S1 S0*

由于电路图尺寸过大，这里不再给出。

### 多周期MIPS硬布线控制器CPU设计

多周期MIPS硬布线控制器CPU的指令译码逻辑同2.1.1，ALU控制逻辑同多周期微程序CPU，因此这里不再赘述。状态转移已经由FSM给出，因此不再需要下址字段和判断字段P，硬布线控制存储器可减少为16位二进制编码，本实验中仍然采用21位二进制编码，弃用下址字段和P即可，多周期硬布线控制器如图2-16所示。

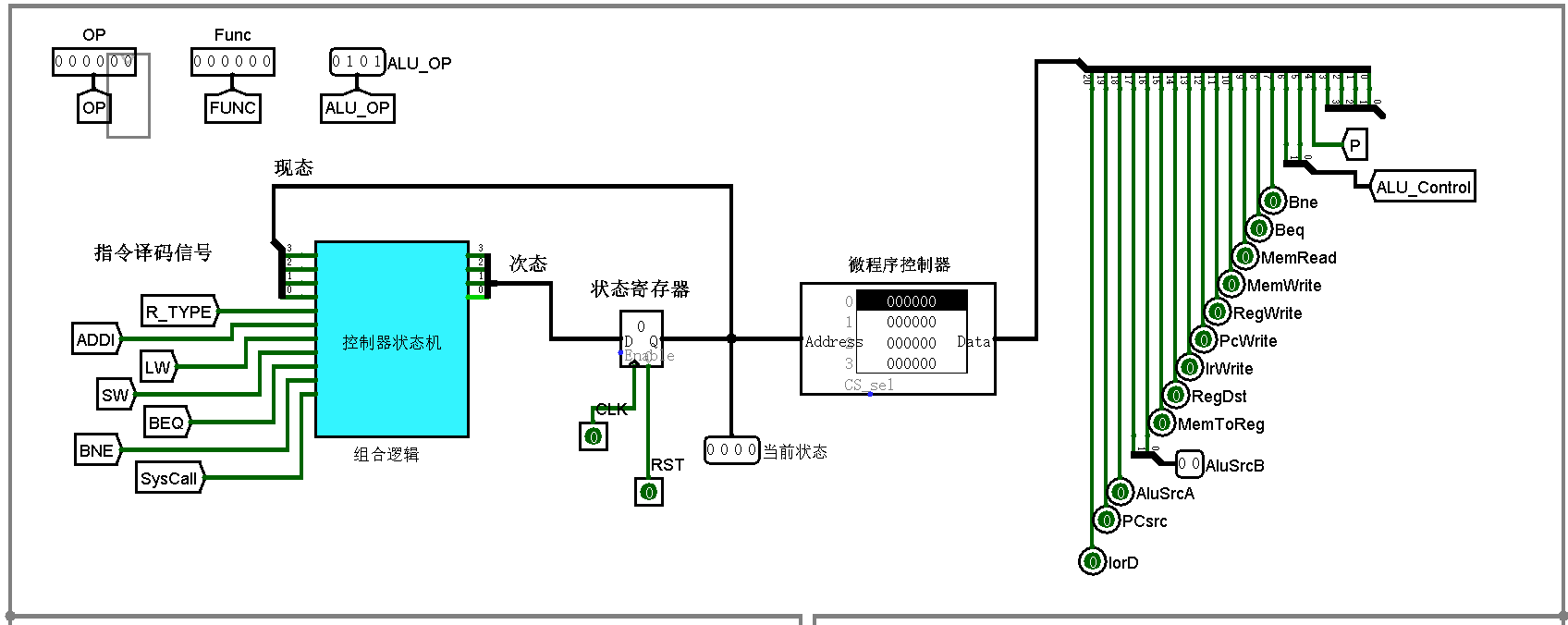


图2-16 多周期硬布线控制器

连接各部件和控制信号，得到多周期MIPS硬布线控制器CPU如图2-17所示。

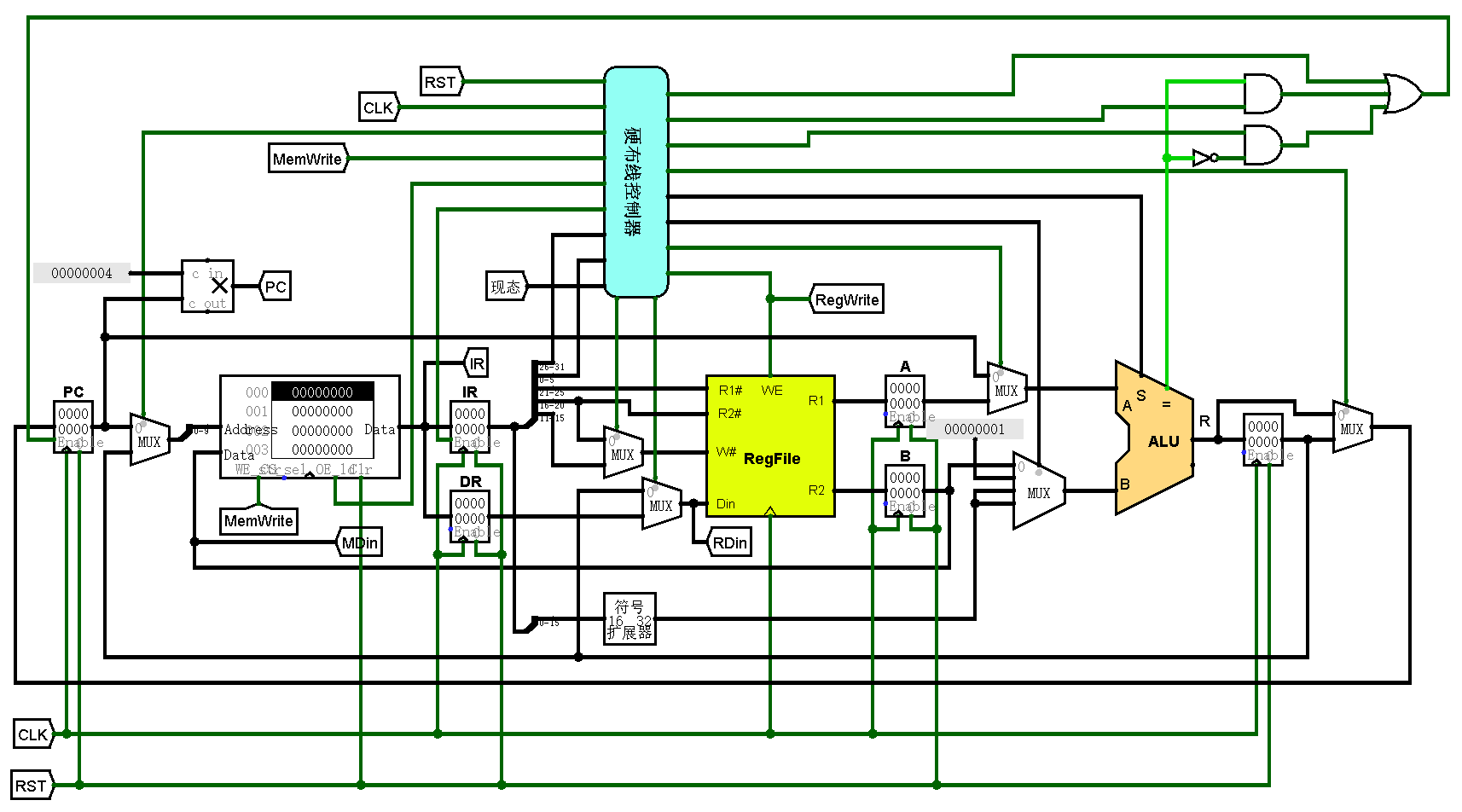


图2-17 多周期MIPS硬布线控制器CPU总结构图

## 实验步骤

（1） 仔细阅读头歌平台各关卡中的实验提示，并了解各部分子电路的功能。观

看实验指导，整理好原理，并与所学建立联系。

（2） 在Logisim和对应的excel表格中完成相应的内容。

（3） 记事本打开cpu.circ，将代码复制粘贴到头歌平台上进行测评。

## 故障与调试

### 单周期数据存储器地址错误

**故障现象：**冒泡排序sort.hex程序执行完毕后，没有在指定的80号存储单元形成降序排列的数据，而是在200号存储单元形成降序排列数据，而且两个数据地址之差为4。

**原因分析：**Logisim中RAM只支持一种访问模式，一次访问读出32位数据，直接给出字地址使得内存布局错误。

**解决方案：**字地址除以4即可得到正确的内存布局，即高两位补零作为字节地址送入存储器地址即可。

### 多周期CPU循环震荡

**故障现象：**多周期CPU运行sort.hex程序时，能正确得到相应的内存布局，但是遇到停机指令无法停机，PCEn使能信号产生振荡，PC循环变化；

**原因分析：**指令译码错误，将syscall指令也当成了R型指令，导致控制器中没有产生正确的PCWrite信号，也没有进入syscall的死循环，从而使得PCEn信号出现震荡，系统无法停机；

**解决方案：**区分syscall指令和R型指令，使之互斥，如图2-18所示。

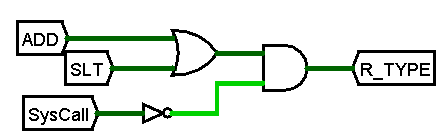


图 2-18 syscall与R型互斥

## 测试与分析

### 单周期MIPS CPU执行sort.hex

运行sort.hex后，数据存储器在80号单元开始处出现6,5,4,3,2,1,ffff的有符号降序数据，如图2-19所示。

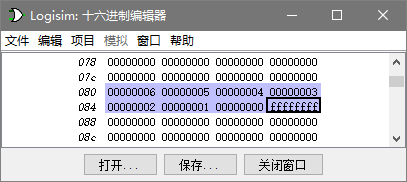


图2-19 单周期CPU执行sort.hex后内存布局

### 多周期MIPS CPU执行sort.hex

运行sort.hex后，数据存储器在80号单元开始处出现6,5,4,3,2,1,ffff的有符号降序数据，如图2-20所示。

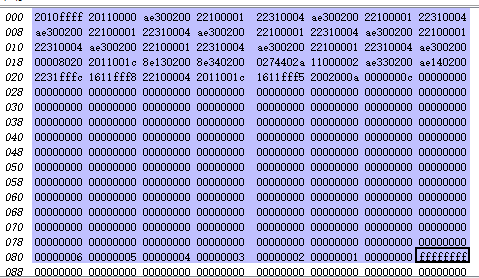


图2-20 多周期CPU执行sort.hex后内存布局

## 实验总结

本次实验主要完成了如下几点工作：

1. 实现了32位定长指令的解析；
2. 实现了寄存器读写；
3. 实现了数据存储器的读写；
4. 实现了操作数的计算；
5. 实现了控制信号的生成；
6. 实现了各部件多输入来源的多选一；
7. 实现了MIPS多周期CPU控制存储器；
8. 实现了MIPS多周期CPU微程序地址转移逻辑；
9. 实现了MIPS多周期CPU硬布线控制器状态机；
10. 完成了MIPS CPU数据通路综合，最终的CPU能完整支持MIPS核心指令集。

## 实验心得

本次CPU设计实验是对数字电路及计算机组成原理的高度综合实践，通过实现单周期硬布线CPU、多周期微程序CPU以及多周期硬布线CPU三种架构，我深刻体会到了CPU设计的复杂性和系统性。每一种架构都有其设计逻辑与控制重点，单周期CPU强调指令在一个周期内完成，要求通路清晰、控制信号精准；而多周期架构则更贴近真实处理器的执行方式，通过状态机的调度控制指令多阶段执行，更具灵活性与扩展性。实验中通过状态转移图、控制信号逻辑推导、电路拼装、平台测试等步骤，不断磨合抽象的计算机组成原理与电路实现之间的接口。这不仅锤炼了我在逻辑设计、调试分析等方面的能力，也让我更加理解“控制+数据通路”是CPU设计的根基。每一个模块的独立设计、正确集成与最终联调，都是一次从“知其然”到“知其所以然”的飞跃。本次实验为我日后深入学习计算机体系结构和硬件系统设计奠定了坚实的基础，也让我体会到数字世界中“每一个信号皆有路径，每一条路径皆有意义”的工程之美。

|  |
| --- |
| 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。  **b66436a16e0131fbf793f567ce058f76**特此声明！  **作者签字: 王家乐** |
| 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） | 课程目标1  工具应用  （10分） | 课程目标2  设计实现  （70分） | 课程目标3  验收与报告  （20分） | 最终评定  （100分） | | 得分 |  |  |  |  | |
| **指导教师签字:** |