***2024***



**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： | 图灵2301 |
| 学 号： | U202310037 |
| 姓 名： | 李语心 |
| 电 话： | 18040376513 |
| 邮 件： | [2742719109@qq.com](mailto:2742719109@qq.com) |
| 完成日期： | 2023-05-30 |



# 运动码表系统设计

## 设计要求

本运动码表系统设计以Logisim为平台，旨在锤炼数字逻辑的硬核实力，要求深刻领悟数字系统离散、严谨的特质，采用模块化与层次化设计，确保逻辑清晰。需熟练掌握组合逻辑设计，梳理输入输出关系，化简表达式，力求简洁高效；同步时序电路设计则要了解状态机，时钟等。还要了解系统集成与联调，验证模块接口与功能。硬件上，系统包含四个按键（Start清零计时、Stop暂停显示、Store更新最快记录、Reset复位至00.00与记录99.99）和四个7段数码管，设计过程中，需注重细节把控，模拟多种使用场景，确保按键响应灵敏、显示准确无误，同时优化电路规模，降低功耗。

## 方案设计

本设计采用模块化思想和层次化设计，将系统分解为时钟、计数器、显示驱动和控制器等模块，并通过数据通路连接这些模块，以实现功能完善、性能稳定的计时系统。总电路图如下：

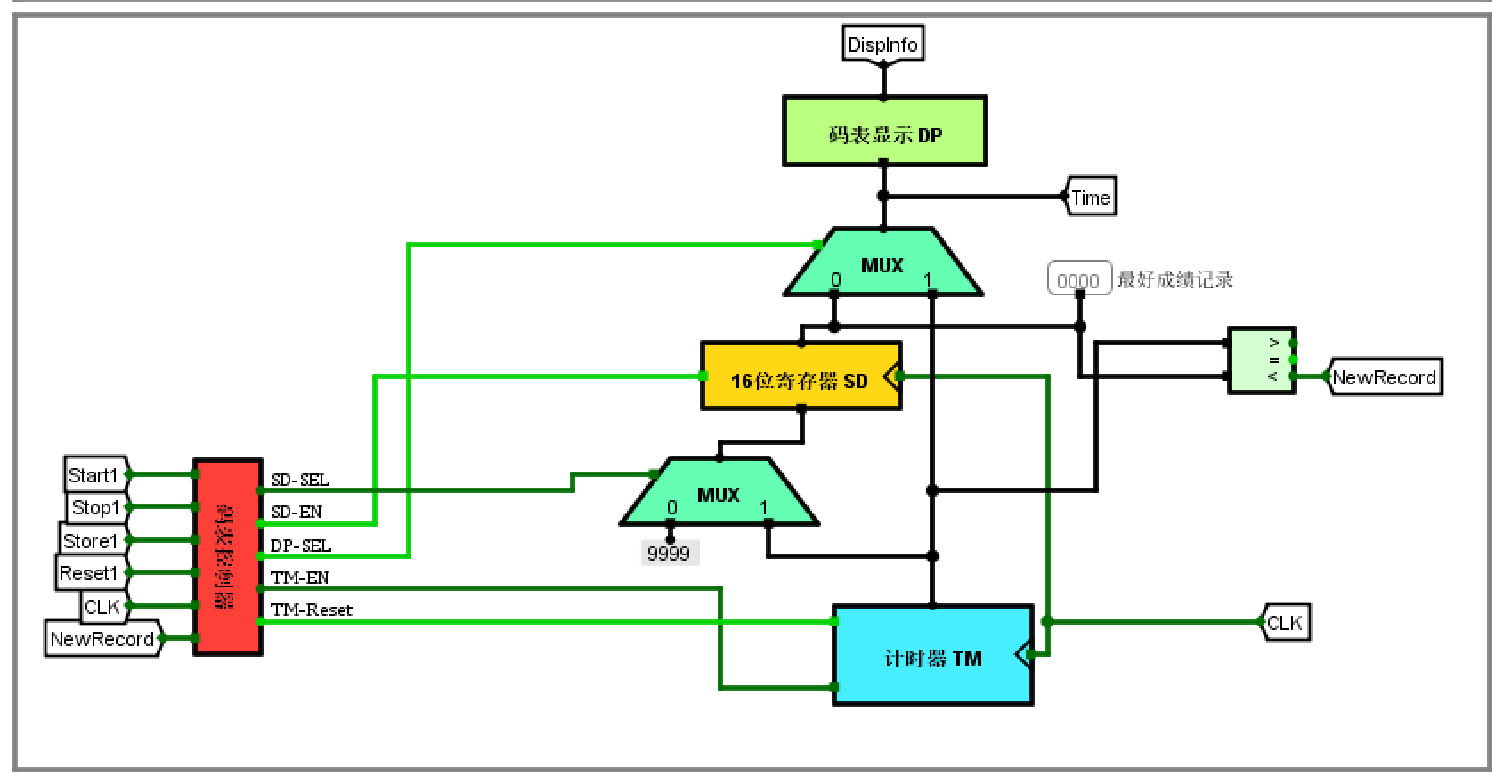


图1-1 总体结构图

### 数码管驱动

数码管驱动，也就是显示驱动模块将计数器的BCD输出转化为数码管，我们采用7段数码管驱动电路，通过真值表产生逻辑电路。

这里没有使用多路复用技术支持多位显示，而是直接显示所有位，感觉不适合真实实现。

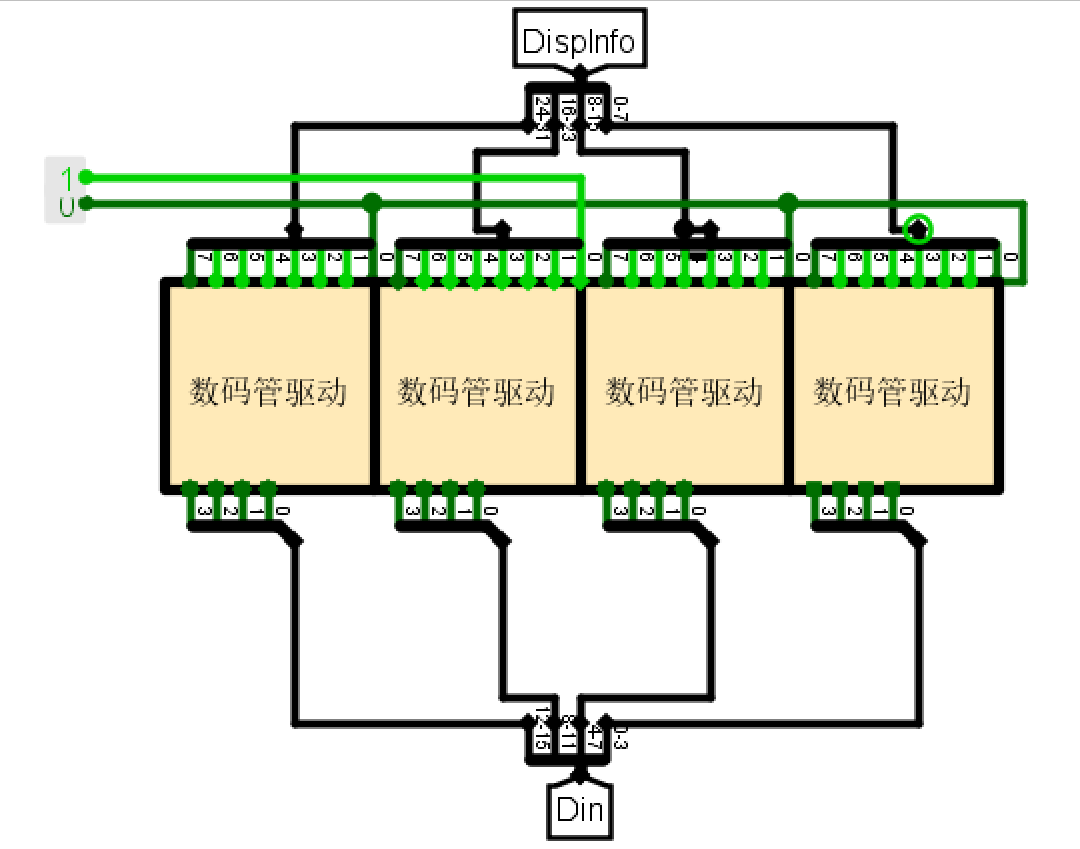


图1-2数码管驱动结构图

### 比较器

比较器的核心功能是比较两个输入数字（A和B），输出比较结果（A>B、A=B、A<B）。为实现高效设计，我们采用了层次化方法。

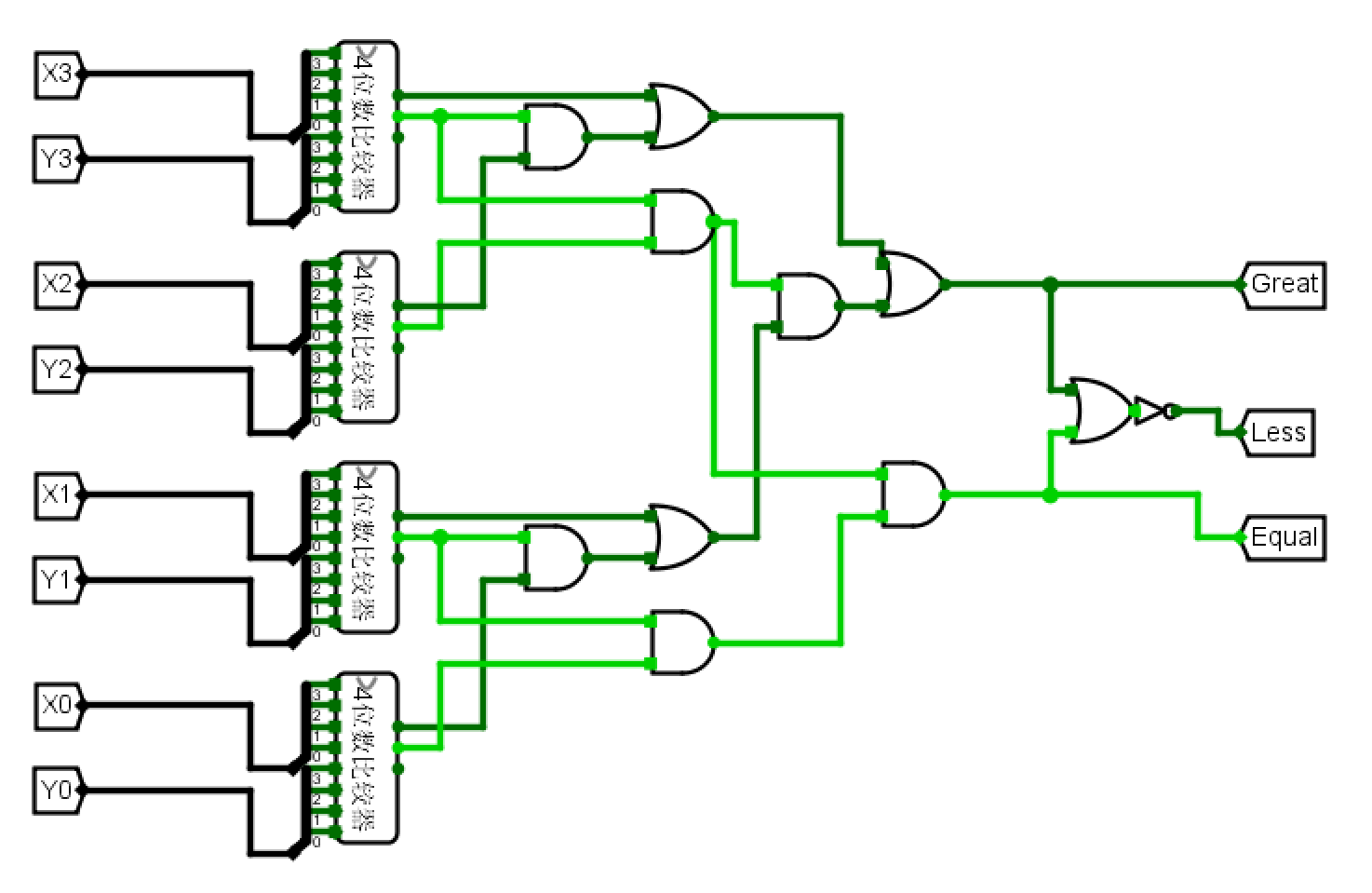
在底层用简单的4位比较器，作为基本单元。然后扩展为多位比较器（如16位），通过级联底层单元实现。这里4位比较器可以使用分治的思想，在分成2个二位的，但是这里我们直接合并成了逻辑电路，提高器件的使用率，减少延迟。但层次化设计不仅简化了复杂比较器的实现，还便于调试和优化，所以4位合并位16位使用了层次化设计。

图1-3 16位无符号比较器结构图

### 计数器

码表计数器基于BCD（二进制编码十进制）计数器，通过并行加载寄存器和状态机逻辑实现。设计思路是将计数器分解为模块化的寄存器和控制逻辑，利用时钟信号驱动递增计数，支持十进制输出以便于显示。过程上，我们先设计1位BCD计数器，基于并行加载寄存器存储4位二进制代码（表示0-9），再扩展为4位十进制计数器，通过级联实现多位计时。而我们之所以采用BCD计数器，是因为其直接输出十进制数字，与7段数码管显示兼容。

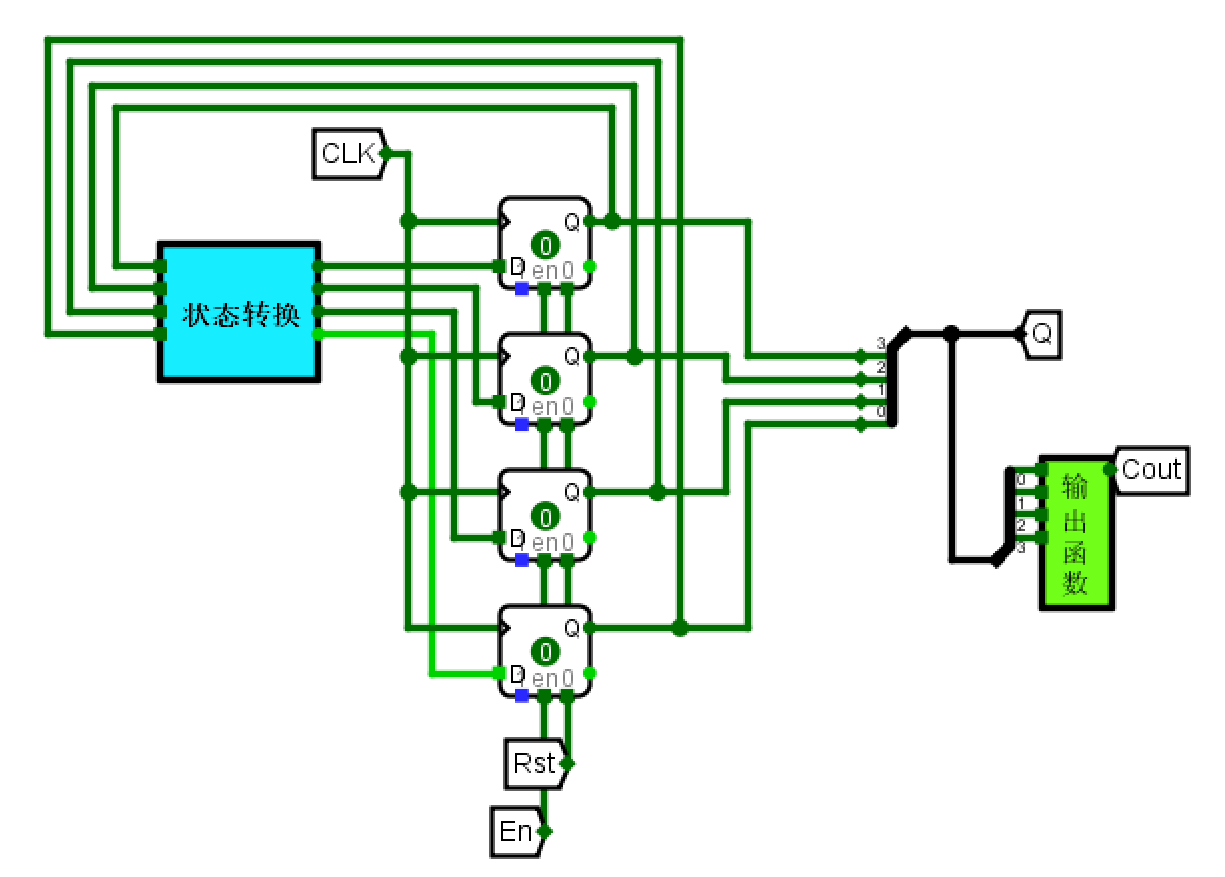
我们使用状态机控制计数器的递增（i -> i+1）。我们使用Moore型FSM，即状态机的输出（cout）只与现态有关。

图1-4 状态机结构图

此系统中并行加载寄存器，用于快速加载初始值或复位，时钟信号驱动寄存器更新，状态机控制寄存器状态和输出函数，计数器输出BCD码至显示驱动模块，形成从时钟到显示的单向数据流。

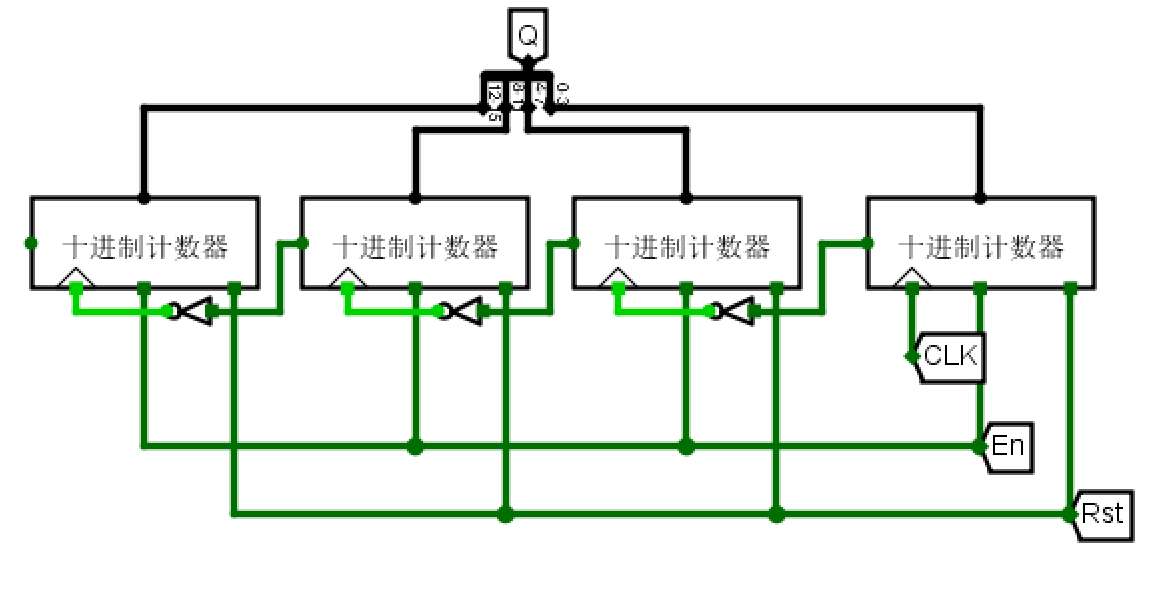


图1-5计数器结构图

### 码表控制器

码表控制器是运动码表数据通路系统的核心管理模块，负责根据用户输入（如开始、停止、复位和存储按键）还有比较器的输出信号协调计时功能并生成控制信号。设计思路基于有限状态机（FSM），将系统状态划分为“停止”、“计时”、“复位”和记录等。这里注意停止计时之后还要通过比较器的判断来判断是否把计数器寄存器里的内容计入记录寄存器里。

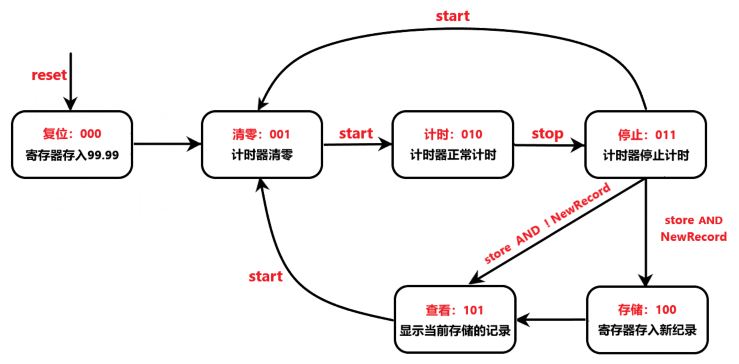
码表控制器通过接收用户输入，输出控制信号驱动计数器启停或复位，计数器结果再通过显示驱动呈现，形成从用户输入到数据输出的闭环控制流，把各模块联系在一起。

图1-6 码表控制器状态机流程图

### 运动码表

最后我们只需完成数据通路即可。运动码表数据通路设计将时钟、4位BCD计数器、7段数码管显示驱动和状态机控制器互联。时钟触发计数器，计数器输出至显示驱动，控制器根据输入调节计数器，形成数据-控制闭环。

这里需要注意各控制信号的含义和时序电路控制的流程。

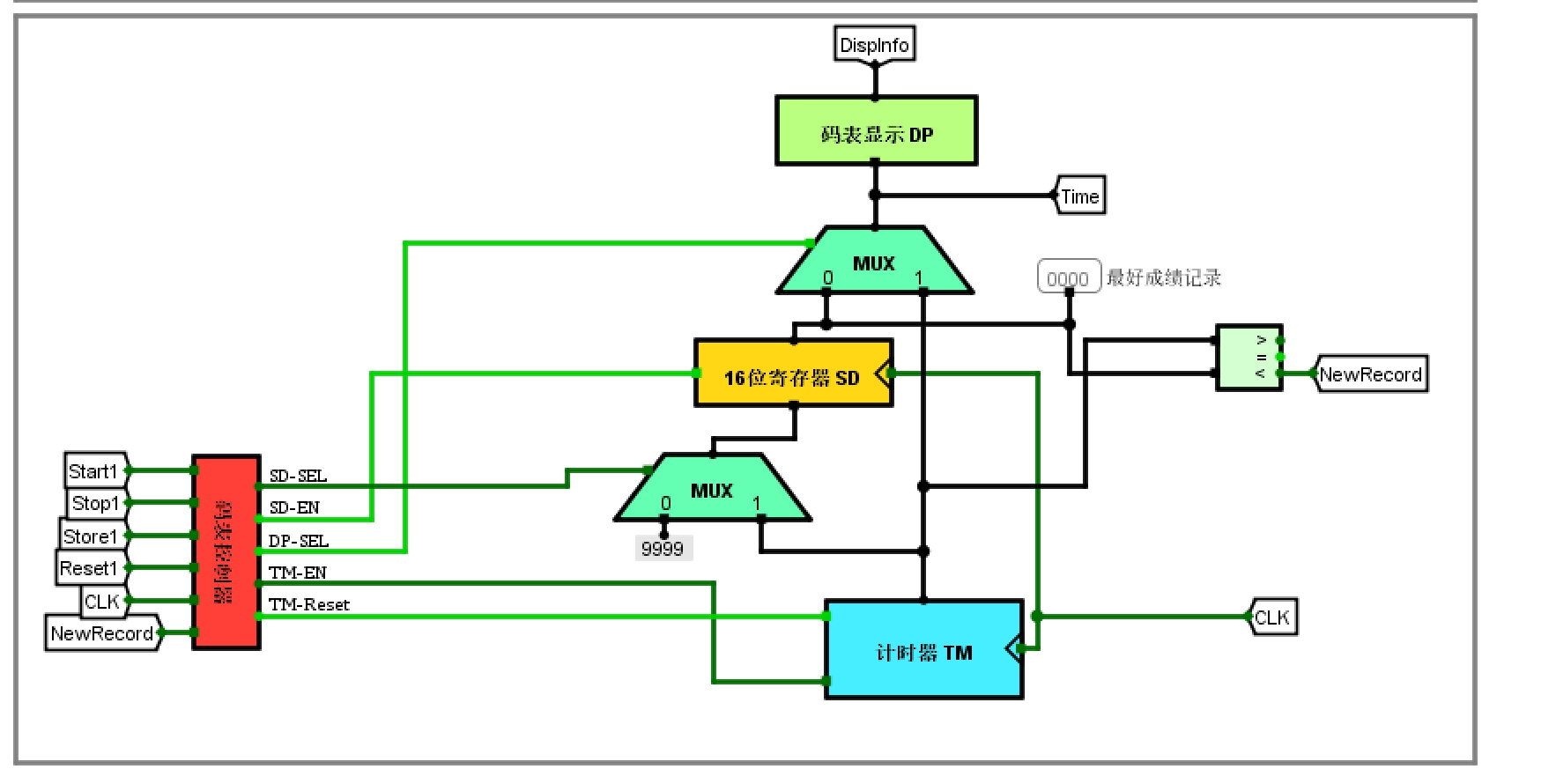


图1-7运动码表数据通路结构图

## 实验步骤

采用了模块化思想和层次化设计。

### 7段数码管驱动电路设计

输入为4位BCD码（B3 B2 B1 B0），输出为7段信号（a, b, c, d, e, f, g）。使用真值表列出BCD码（0-9）对应的7段输出（共阳或共阴，假设共阴）。

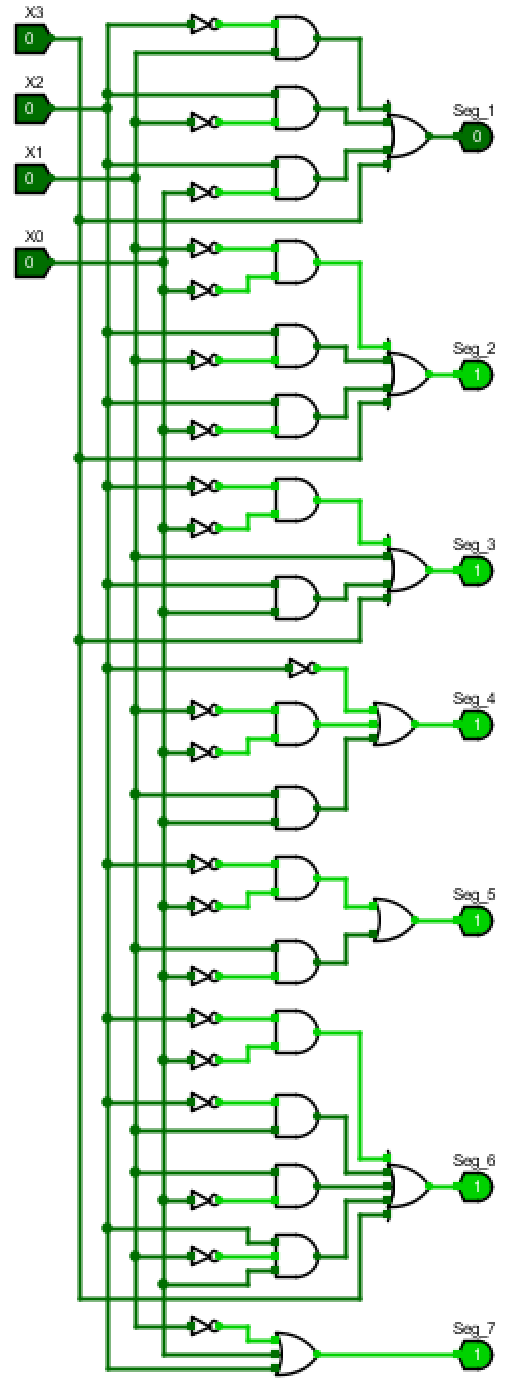


图1-8 7段数码管驱动电路设计图

### 2选1选择器设计（1位）

设计一个1位2选1多路选择器（MUX），根据选择信号S选择输入A或B。使用逻辑电路，S=0时输出A，S=1时输出B。



图1-9 一位2选1选择器设计图

### 2选1选择器设计（16位）

扩展1位2选1选择器为16位，处理16位数据。将16个1位2选1选择器并行组合，共享同一选择信号S。其电路图如下：

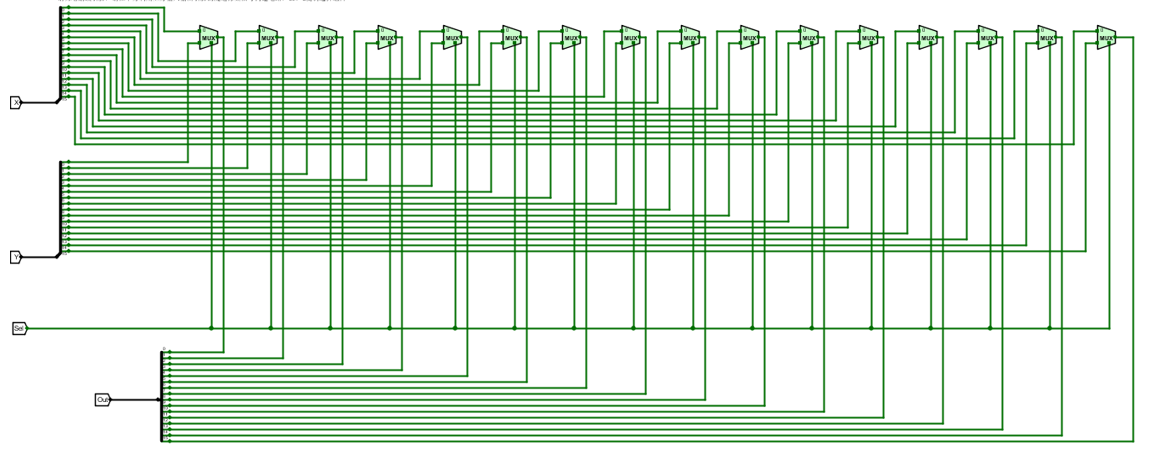


图1-10 16位2选1选择器结构图

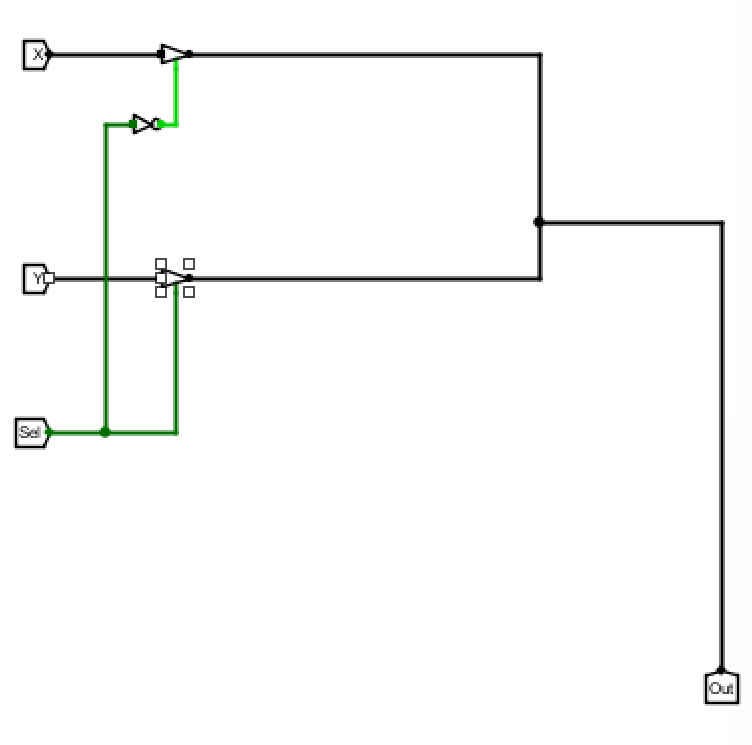
还可以使用三态门来实现，其电路图如下：

图1-11 16位2选1选择器结构图2

### 无符号比较器设计（4位）

设计一个4位无符号比较器，比较A和B，输出A>B, A=B, A<B。使用逻辑表达式，推导时使用了分治的思想。其中“小于”可以通过逻辑关系从“大于”和“等于”中直接得到，即less=!(great||equal)。其电路图如下：

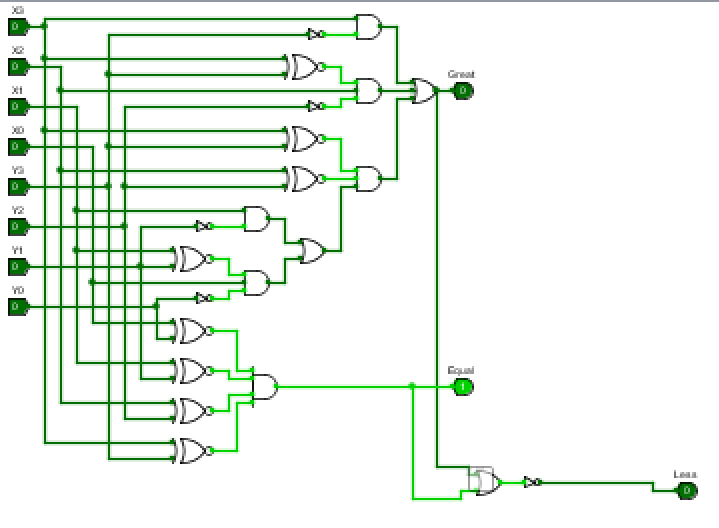


图1-12 4位无符号比较器结构图

### 无符号比较器设计（16位）

级联4个4位比较器，进一步使用分治思想。其电路图如下：

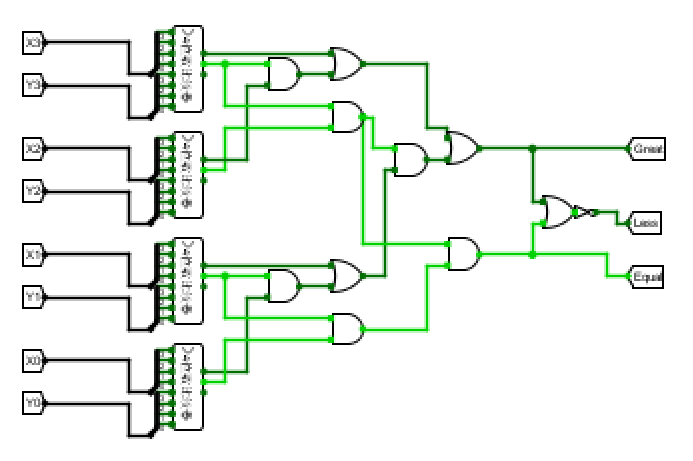


图1-13无符号比较器结构图（16位）

### 并行加载寄存器（4位）

设计一个4位寄存器，支持并行加载和保持数据。使用D触发器和选择器实现加载/保持。其电路图如下：

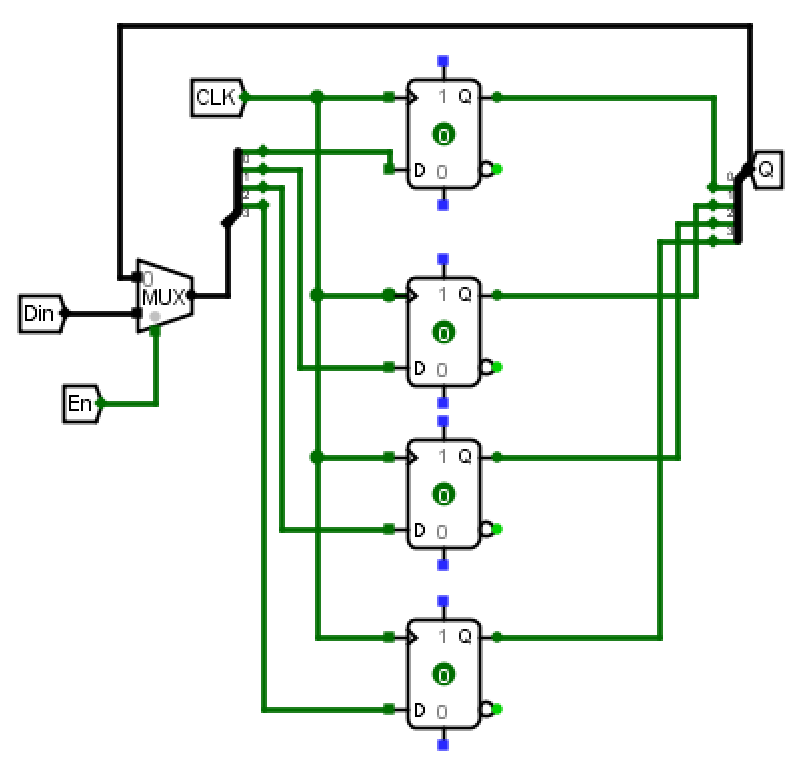


图1-14 并行加载寄存器结构图（4位）

这里的enable信号使用二路选择器实现，即：输入 D0为当前寄存器的输出（保持当前数据）。输入 D1为新的输入数据（加载新数据）。

### 并行加载寄存器（16位）

使用4个4位寄存器并行组合，扩展为16位并行加载寄存器。

其电路图如下：

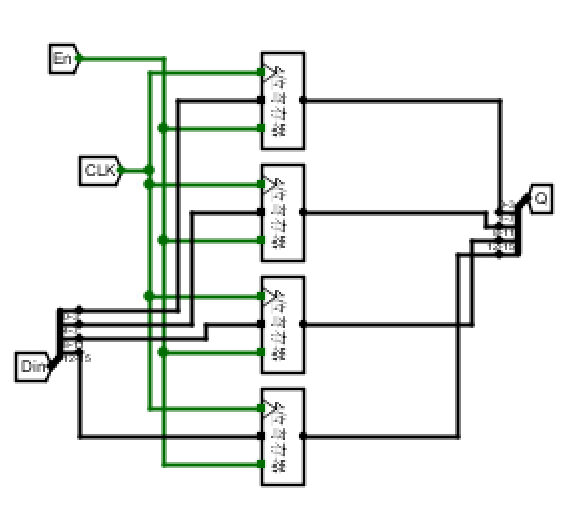


图1-15 并行加载寄存器（16位）

### BCD计数器状态机设计

设计一个1位十进制BCD计数器的状态机（0-9循环）。寄存器存储当前状态，加法器计算下一状态。 状态转移图如下：

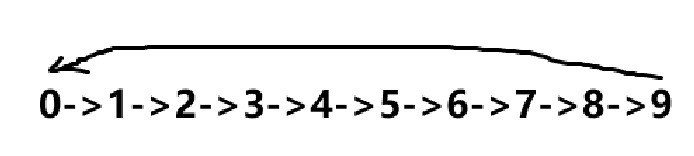


图1-16 BCD计数器状态机转移图

### BCD计数器输出函数设计

设计BCD计数器的输出逻辑（当前BCD值和进位信号）。从状态机获取状态，使用比较器生成cout，进位输出，S=9时，Cout=1。此输出函数用于级联成码表计数器时进位。其电路图如下：

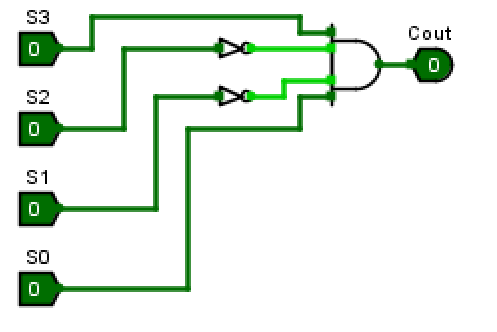


图1-17 BCD计数器输出函数结构图

### BCD计数器设计（1位十进制）

集成状态机和输出函数，完成1位BCD计数器。其电路图和封装图如下：

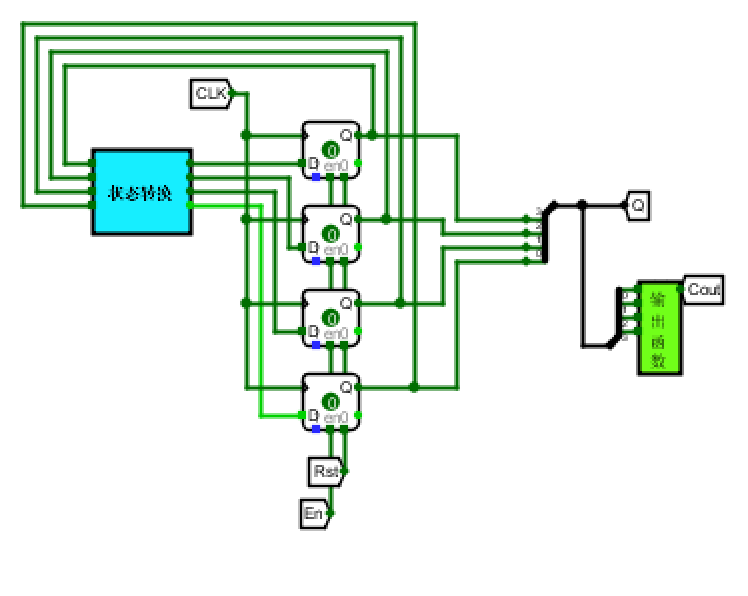


图1-18 BCD计数器设计（1位十进制） 电路图

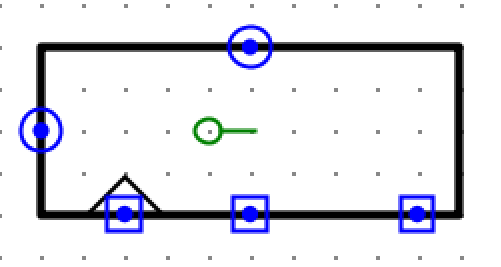


图1-19 BCD计数器设计（1位十进制） 封装图

### 码表计数器设计（4位十进制）

级联4个BCD计数器，设置进位条件。这里把cout作为时钟信号来实现进位，注意要对进位函数取反，在状态从9到10的时候形成上升沿。

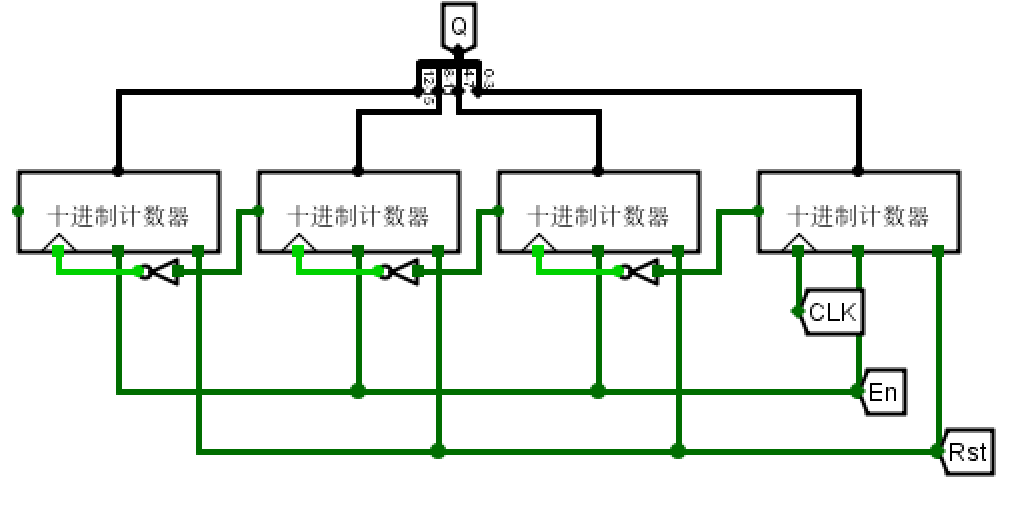


图1-20 码表计数器设计（4位十进制）

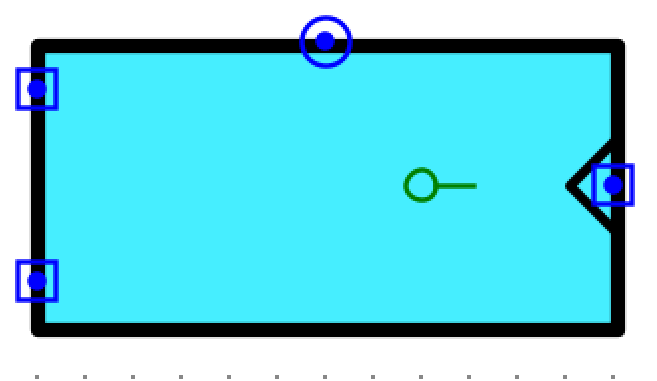


图1-21码表计数器封装图（4位十进制）

### 码表显示驱动设计

将4位BCD计数器输出转换为7段数码管显示，直接连接线路即可。总体结构如下：

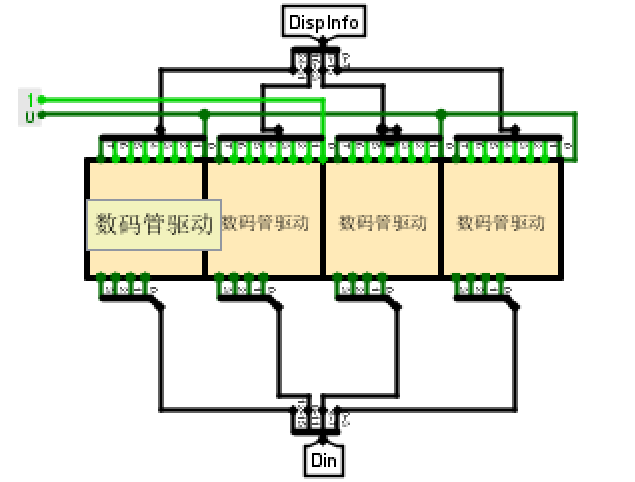


图1-22码表显示驱动设计电路图

### 码表控制器状态机设计

使用有限状态机（FSM），将系统状态划分为“停止”、“计时”、“复位”和记录等。这里注意停止计时之后还要通过比较器的判断来判断是否把计数器寄存器里的内容计入记录寄存器里。

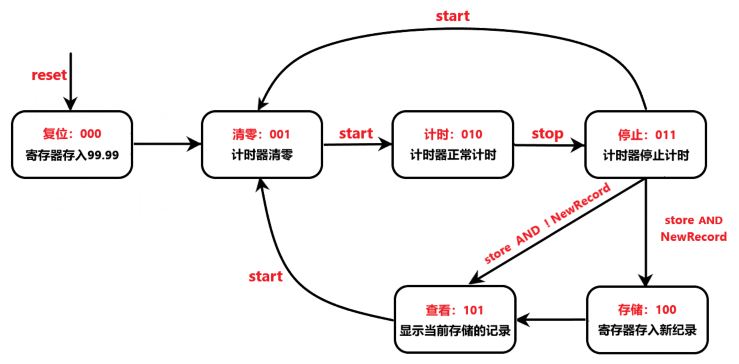


图1-23 码表控制器状态机转移图

### 码表控制器输出函数设计

设计控制器输出信号。状态机类型为moore型FSM，所以其输出仅取决于当前状态。输出信号位宽和说明如下：



图1-24 输出信号说明图

### 码表控制器设计

集成状态机和输出函数，完成码表控制器。



图1-25 码表控制器设计图

### 运动码表数据通路设计（系统集成）

集成所有模块，完成运动码表系统。运动码表数据通路设计将时钟、4位BCD计数器、7段数码管显示驱动和状态机控制器互联。时钟触发计数器，计数器输出至显示驱动，控制器根据输入调节计数器，形成数据-控制闭环。

这里需要注意各控制信号的含义和时序电路控制的流程。

电路图如下：

### 

图1-26 运动码表数据通路

## 故障与调试

1.4.1 16位2选1选择器连线错误

**故障现象**： 16位2选1选择器输出结果异常，部分位输出错误或为不定值（X）。

**原因分析**：16位2选1选择器由16个1位选择器（mux\_1bit）组成，通过Splitter拆分和合并16位信号。在Logisim中连接时，由于引脚数量较多，A[15:0]和B[15:0]的某些位接错了，导致选择逻辑失效。

**解决方案**：检查Splitter配置，确保A[15:0]和B[15:0]的位序正确。测试时输入固定值（如A=0xFFFF, B=0x0000），逐步切换S，验证Y[15:0]输出。

1.4.2 4位无符号比较器逻辑表达式错误

**故障现象**：比较器输出GT、EQ、LT不正确，例如A=5, B=3时GT=0（应为1）。

**原因分析**：4位比较器的逻辑表达式复杂度较高，手动推导时容易出错，例如遗漏项或括号错误。此外，逐位比较逻辑未正确实现高位优先，导致低位比较干扰结果。

**解决方案**：

采用分治思想，将4位比较器分解为2位子比较器，先比较高2位[A3:A2]，若相等再比较低2位[A1:A0]。测试关键用例（如A=4, B=4; A=7, B=2），确保GT、EQ、LT正确。

1.4.3 4位并行加载寄存器Enable端逻辑错误

**故障现象**：当Enable（Load）信号为0时，寄存器输出Q[3:0]被清零为0000，而不是保持原有值。

**原因分析**：并行加载寄存器应在Load=1时加载D[3:0]，Load=0时保持Q[3:0]。但在设计中，没有使用MUX，导致数据被清零。根本上来说是因为4位并行加载寄存器的使能逻辑设计缺陷，Load信号（Enable）在未使能时错误地强制D触发器输入为零，导致寄存器清零而非保持值。

**解决方案**：修改寄存器电路，确保Load=0时，D触发器输入通过2选1选择器连接到各自的Q输出，保持当前状态。

1.4.4 码表计数器进位逻辑错误

**故障现象**： 4位BCD计数器表现出错误的进位行为，从8到9时产生进位信号，而不是从9到0时，且在9到10时无法正确进位。

**原因分析**：每个BCD数字（0-9）的进位逻辑应在数字达到9且计数使能（CE）有效时触发进位输出，重置为0并触发下一位数字。然而，设计中会在8到9时提前触发进位。

**解决方案**：对cout取反。

1.4.5 7段数码管小数点显示问题

**故障现象**： 7段数码管输出信号过不了测试。

**原因分析**： 7段数码管包含小数点（dp）输入应为常数值来当MM:SS分隔符，在第二和第三位数字之间。

**解决方案**：分钟和秒之间的7段数码管小数点输入为常数1。然后其他显示器的小数点用Constant组件连接到0（关闭）。

1.4.6 运动码表数据通路控制信号和比较逻辑错误

**故障现象**：有三个问题：数据通路里面显示时间（time）不正确；控制信号（例如CE、Reset）行为不对；新纪录逻辑在当前时间大于前纪录时错误更新，而非小于前纪录时。

**原因分析：**

1. time的定义不明确。

2. 控制信号连接错误。需要仔细查看定义。

3. 纪录比较逻辑反了，当当前时间大于存储纪录时错误置newrecord=1，而应在当前时间小于纪录时置1（表示更快时间）。

**解决方案：** 修改线路连接，纠正以上错误。

因为实验报告模板在实验完成后才发出，所以没有当时故障的图例。

## 实验总结

本次实验主要完成了以下工作：

1. **熟悉实验平台**：配置Logisim环境，了解各个部分。
2. **设计组合逻辑模块**：实现16位2选1选择器、4位无符号比较器等模块，梳理输入输出关系，化简逻辑表达式，提升模块效率。
3. **开发时序逻辑模块**：设计4位并行加载寄存器、码表计数器等，基于状态机与时钟信号，确保同步时序逻辑准确。
4. **集成系统模块**：采用模块化与层次化设计，连接计数器、控制器与显示模块，验证接口兼容性与功能完整性。
5. **调试模块功能**：逐位检查信号连接，修正选择器连线错误、比较器逻辑错误等问题。
6. **优化电路设计**：精简逻辑电路规模，降低功耗，提高了器件利用。
7. **模拟使用场景**：测试按键功能（Start、Stop、Store、Reset），验证计时、暂停、记录更新与复位功能，覆盖多种输入组合。
8. **验证系统性能**：模拟真实计时场景，确认时间显示与记录更新的准确性。
9. **撰写实验报告**：记录设计过程、错误分析与解决方案，总结模块化设计的经验与教训。

通过以上工作，我加强了对数字逻辑设计与系统集成的理解，提高了逻辑分析与问题解决能力，为后续复杂数字系统设计奠定了基础。

## 实验心得

实验的整体质量非常高，内容安排合理、结构清晰。非常感谢设计者所付出的努力与心血，使我们在实践过程中能够将更多精力投入到对核心内容的理解与应用上，显著减少了重复性体力劳动所占据的时间，也增强了我们的学习体验。

通过这次课程设计，我有机会亲自实践数字电路的设计与搭建过程，这不仅加深了我对课堂所学理论知识的理解，也让我在实际操作中发现了理论与实践之间的联系与差异。

# 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 单周期处理器，一个多周期 微程序控制器MIPS 处理器，和一个多周期硬布线MIPS处理器。要求支持8条 MIPS 核心指令，最终设计实现的 MIPS 处理器能运行实验包中的冒泡排序测试程序 sort.asm。 8条 MIPS 核心指令如下所示：

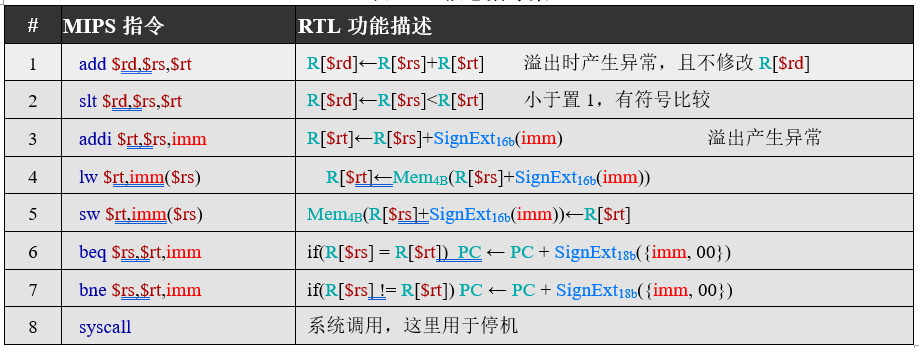


图2-1 MIPS 核心指令及描述

## 方案设计

### 单周期MIPS CPU设计

单周期结构中，所有指令在一个时钟周期内完成取指、译码、执行、访存和写回操作，强调控制路径与数据通路的紧密集成。设计遵循简化的 MIPS32 架构，支持基本的算术逻辑运算（如 ADD，SLT）、数据传输指令（如 LW、SW）以及控制转移指令（如 BEQ）。能够实现寄存器和内存之间的数据传输。整体结构包括程序计数器（PC）、指令存储器、寄存器堆、ALU、数据存储器、控制单元和必要的多路选择器。该设计的目标是在保证功能正确的基础上，简洁明了地展现 RISC 架构的设计理念。总的数据通路如下所示：

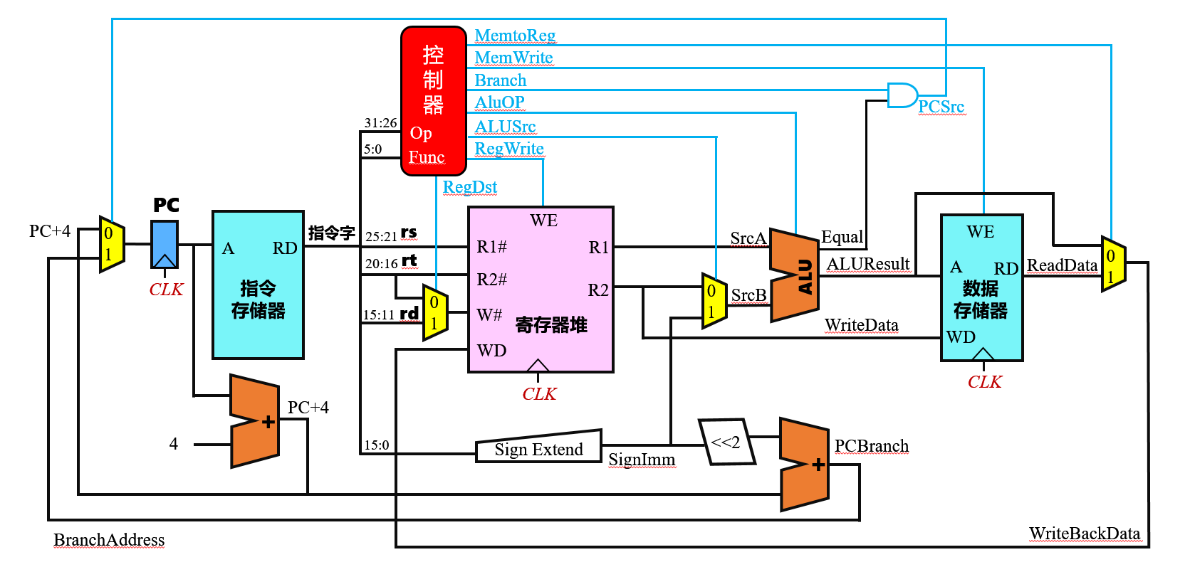


图2-2 单周期MIPS CPU数据通路

本单周期 MIPS CPU 采用模块化结构划分数据通路与控制路径，**使用复杂数据通路，将指令存储器和数据存储器分离，将PC的运算器和数据的运算器并利用存储器和寄存器的异步读取**，以保证能在一个周期内完成。控制单元根据指令类型生成相应的控制信号，驱动各模块实现算术逻辑、数据传输及控制转移等操作。

但是它也有很多缺点，比如性能低下，由于所有指令在一个统一的时钟周期内完成，不同复杂度的指令（如加法与内存访问）必须统一适配最长路径延迟，导致整个系统时钟频率受限，无法针对简单指令进行加速。资源利用率低，每个周期只执行一条指令，无法实现指令重叠处理，浪费了硬件的潜在吞吐能力。扩展性差，不利于加入复杂指令或改进机制，如中断处理、异常机制和多周期优化等。 因为，我们需要实现多周期 MIPS 处理器，如下。

### 多周期 MIPS 处理器

多周期 MIPS 处理器通过将每条指令的执行过程划分为多个独立的时钟周期来完成，从而将原本在单周期中同时进行的多个操作按步骤依次执行。每个周期中只执行一个子操作，例如：取指令、译码、运算、读取或写入。由于指令的不同阶段在不同时钟周期执行，因此处理器可以复用关键硬件资源，如 ALU、寄存器堆和内存模块，不需要为每个阶段配置专用硬件，从而显著降低了硬件复杂度与芯片面积。多周期 MIPS 处理器数据通路如下所示：

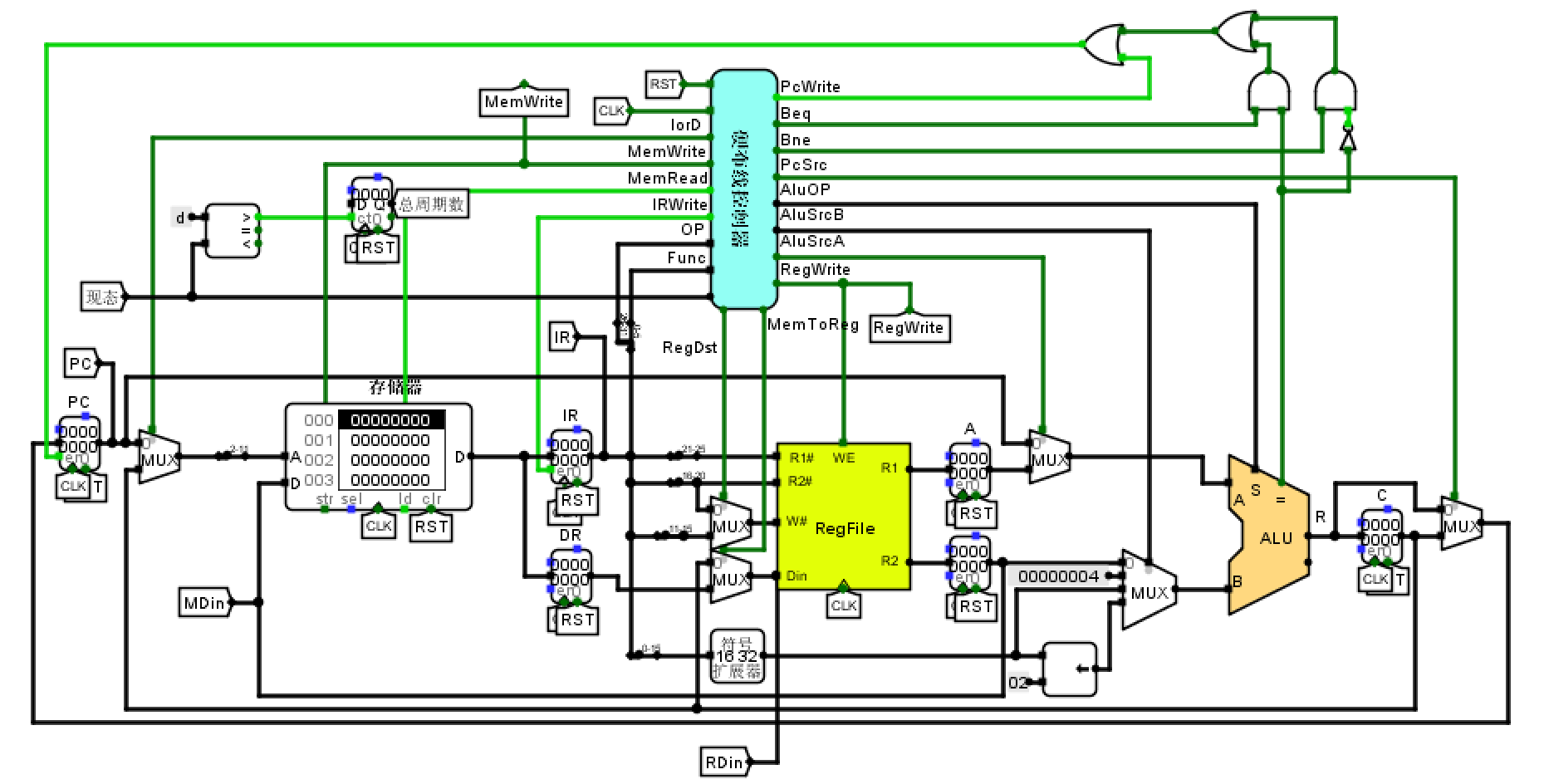


图2-3 多周期MIPS CPU数据通路

其中控制器负责控制所有操作，有两种实现方法：硬布线和微程序。**硬布线**采用状态机（通常是有限状态机，FSM）设计，根据当前状态与指令类型生成控制信号。而**微程序**则将每一时钟周期的所有控制信号的编码为一套“微指令”，存储在微指令存储器中，由微程序计数器顺序调度，具有较高的可扩展性和易于修改的优势。通俗来讲，就是一个用组合逻辑电路生成控制信号，一个用ROM存储控制信号。两种方法各有优劣，设计时需权衡性能、复杂度与可维护性。

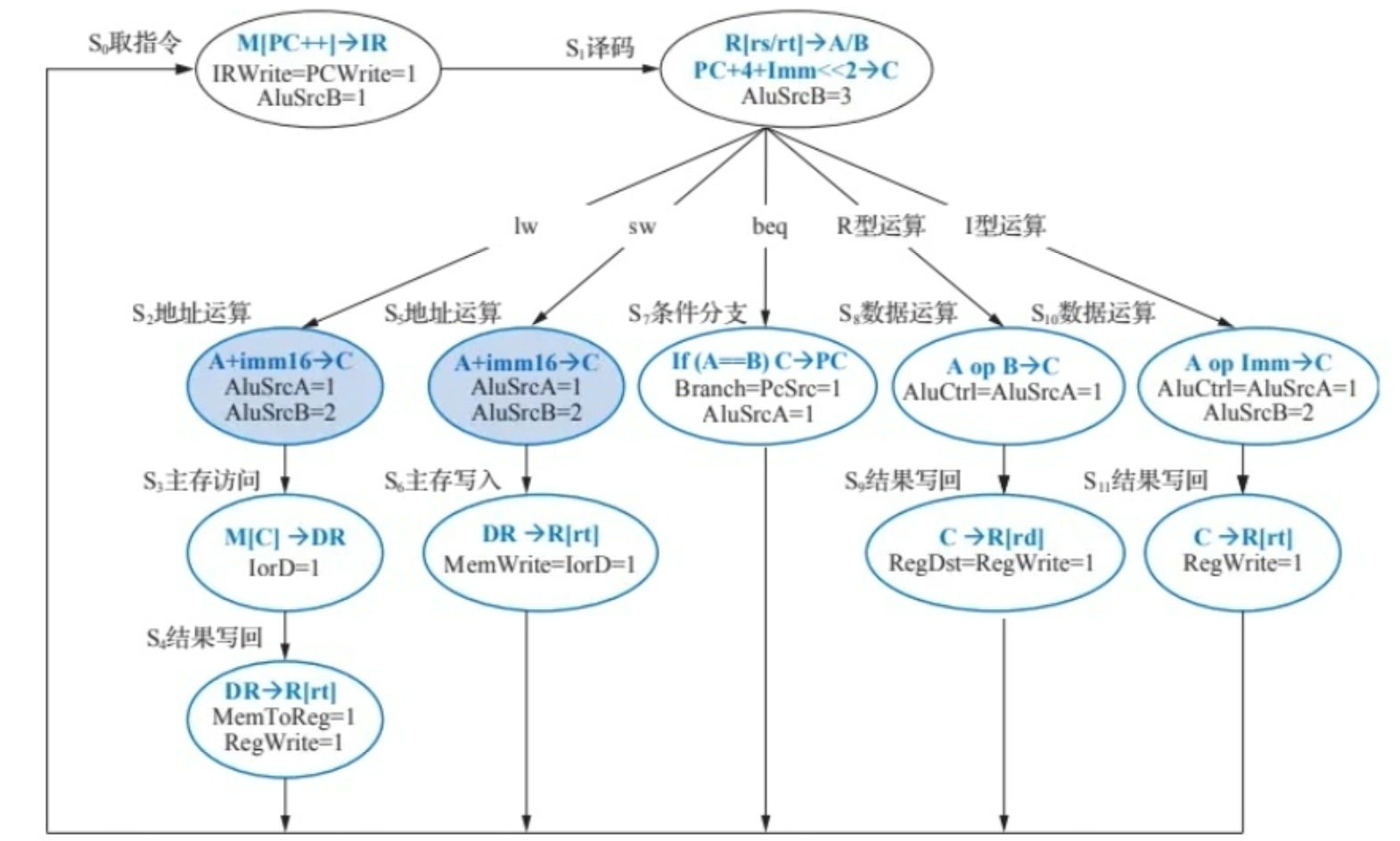
两种方案都使用以下状态转移图：

图2-4 多周期MIPS CPU状态转移图

## 实验步骤

### 实现控制器的指令解析

在指令解析过程中，我们首先对32 位的输入操作码通过分线器进行分解。其中，最高的 6 位用于表示操作码（op），第 21 至 25 位对应源寄存器 rs 的编号，第 16 至 20 位为目标寄存器 rt 的编号，第 11 至 15 位为目的寄存器 rd 的编号，而最低的 6 位（即第 0 至 5 位）则作为功能码（func）使用。此外，最低的 16 位还可用作 I 型指令中的立即数操作数。

然后我们通过组合逻辑电路（比较器）对OP进行解析，判断其是哪一种指令，如果OP为0，则不是I型指令，再判断通过func判断是R型还是系统调用。具体结构可由下图表示：

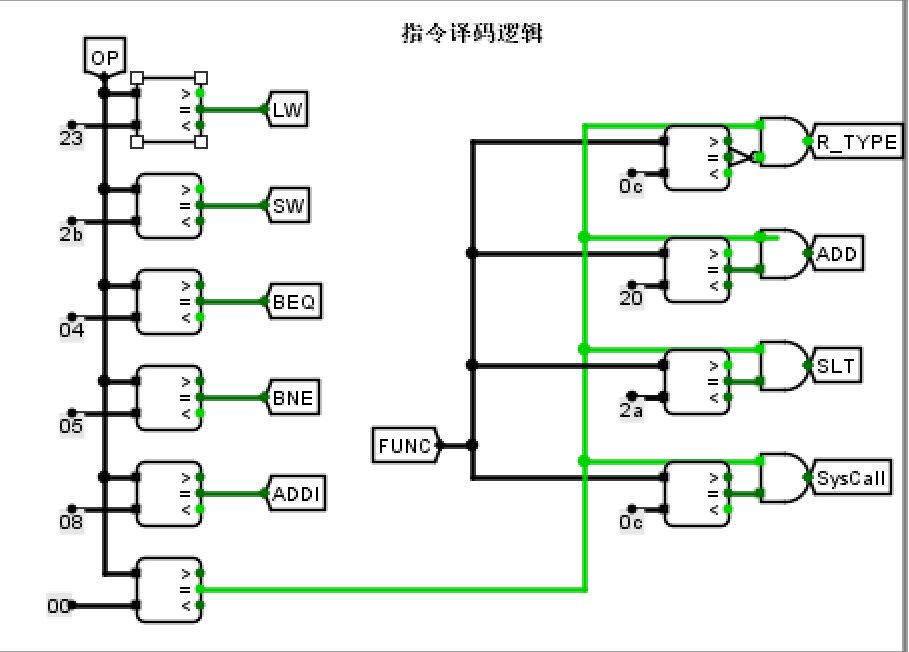


图2-5 指令译码逻辑电路图

值得注意的是，三个CPU的指令译码逻辑是相同的。

### 单周期硬布线控制器

单周期硬布线控制器通过不同指令类型（如R型指令、ADDI、LW、SW、BEQ、BNE、SysCall）的输入信号，生成多种控制信号，进而控制CPU数据通路中的操作，例如决定数据是否写入寄存器（RegWrite）、数据来源是内存还是ALU（MemToReg）、ALU操作数选择（ALUSrc）、是否进行内存写入（MemWrite）以及分支条件（Beq、Bne），最终共同协调CPU根据当前指令正确执行相应的功能。

下图为存储器和寄存器读写控制信号、多路选择器控制信号以及运算操作选择的产生条件：



图2-6 控制信号说明图

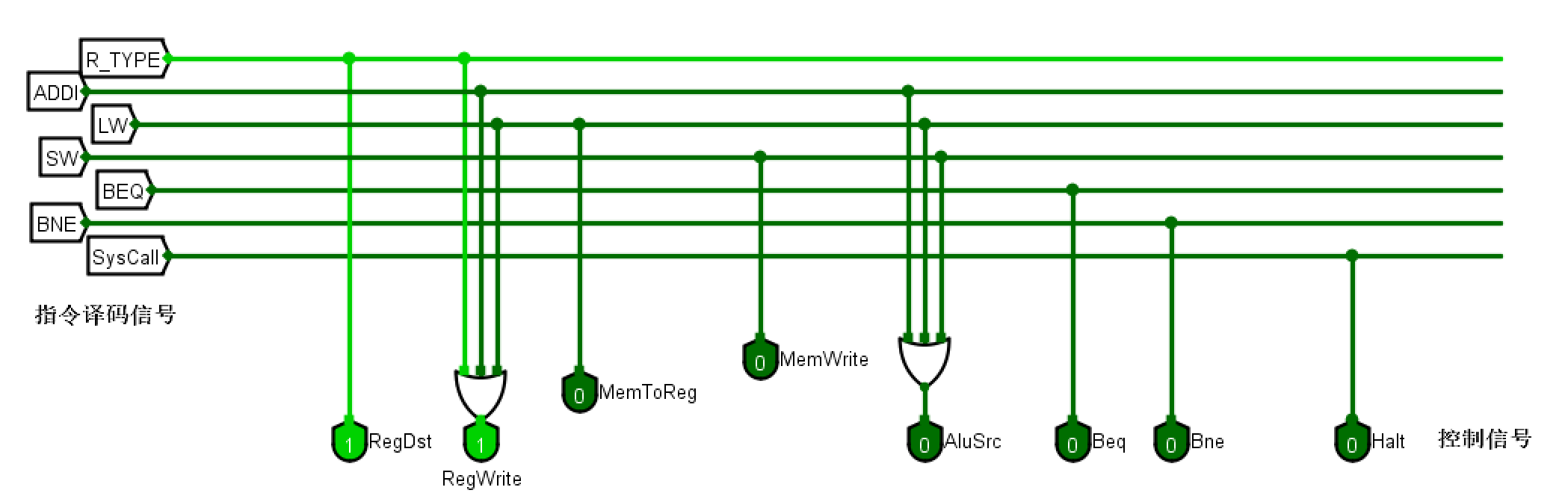
由此可以推导出每个指令对应的控制信号：  


图2-7 指令解析控制信号电路图

### 单周期MIPS数据通路

单周期MIPS处理器的数据通路在一个时钟周期内完成一条指令的取指、译码、执行、访存和写回操作。数据通路的核心组件包括：程序计数器（PC寄存器）用于存储当前指令地址；指令存储器（下图RAM部分，作为指令存储器）用于取出指令；寄存器文件（RegFile，下图黄色部分）用于读写通用寄存器；算术逻辑单元（ALU，橙色部分）执行算术和逻辑运算；数据存储器（下图ROM部分）用于访存操作。

通过对照数据通路将各部件以及控制信号之间进行连接即可，具体结构可由下图表示：

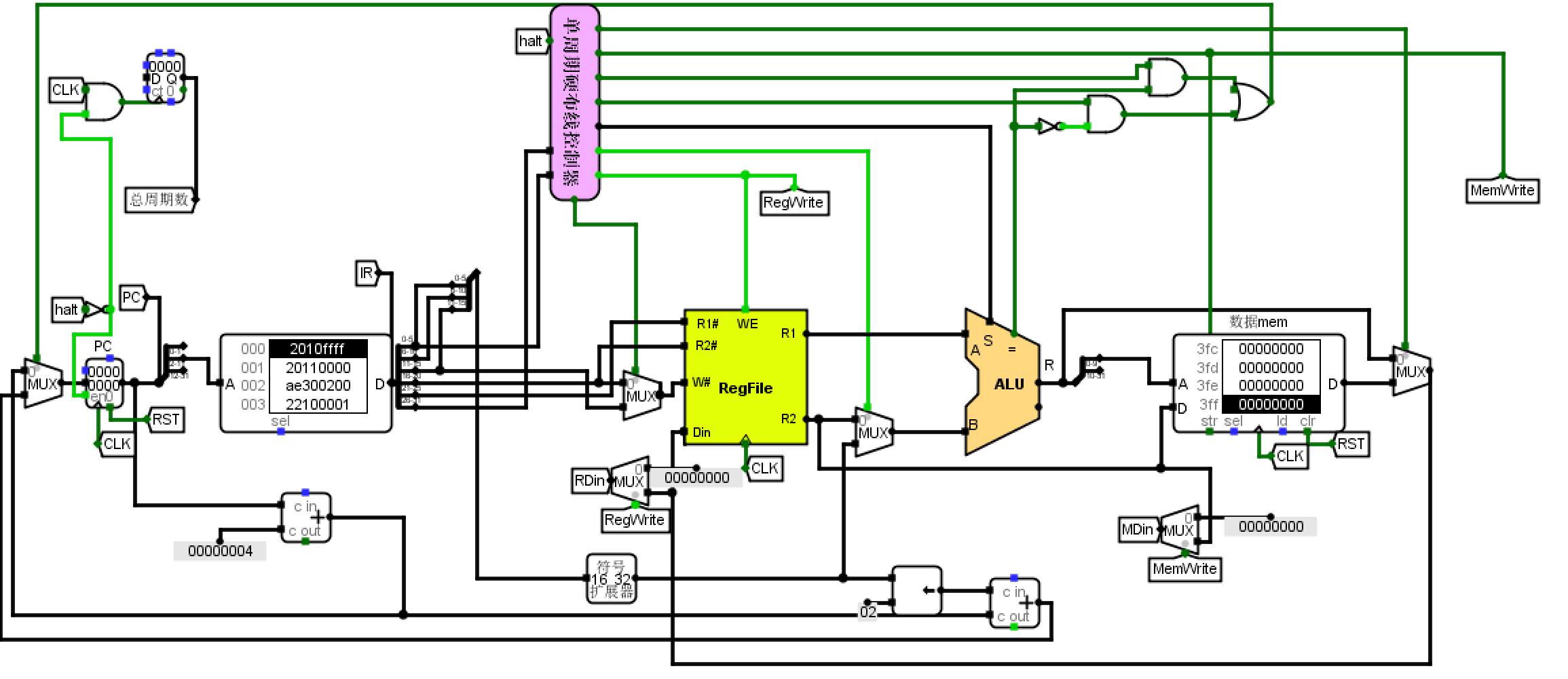


图2-8 单周期MIPS数据通路电路图

### 多周期微程序控制器

在多周期 CPU 的控制结构中，不同的指令通常需要不同数量的时钟周期才能完成执行。因此，无法像单周期控制器那样一次性生成整条指令所需的全部控制信号。所以在微程序控制器中，每一条机器指令的执行过程都被分解成一系列更基本的、不可再分的**微操作**。这些微操作的序列就构成了对应机器指令的微程序。这些微程序被固化在控制存储器中，而这种控制存储器通常就是ROM（Read-Only Memory）。每个微指令如下构成：

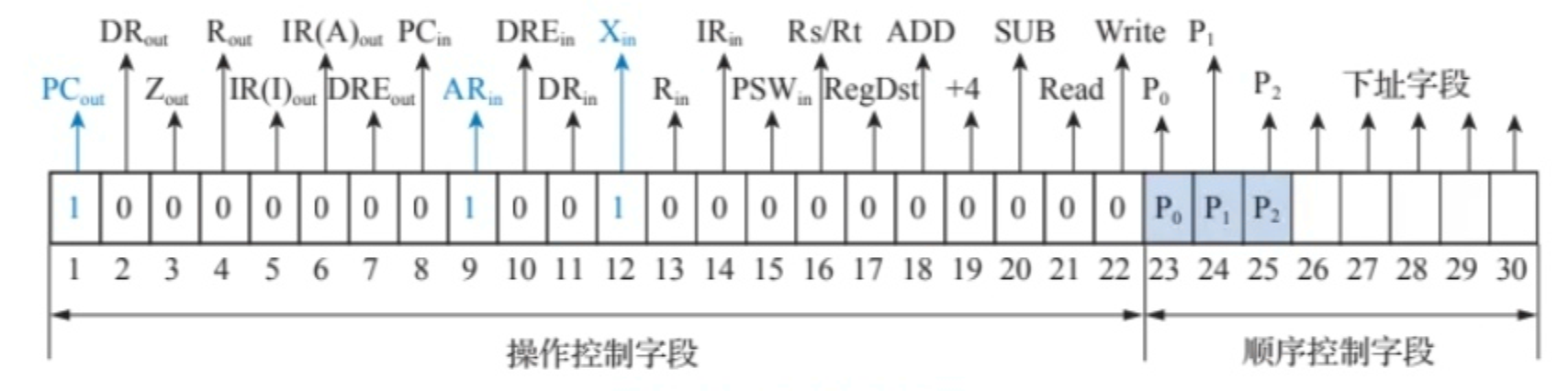


图2-9 微指令结构图

微程序在S1译码之后（P=1时）进行地址转移（而非直接转移到下址），所以需要一个组合逻辑进行转移：

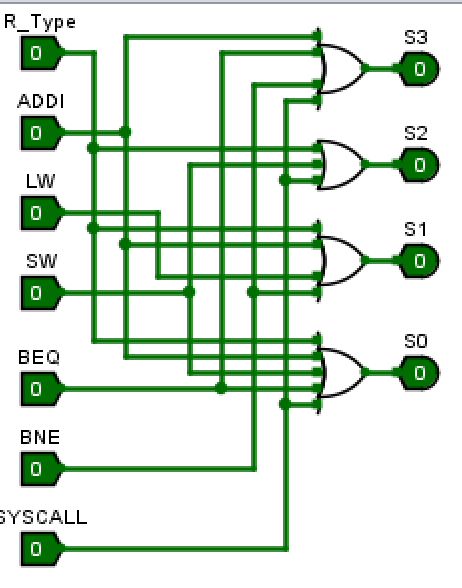


图2-10 地址转移电路图

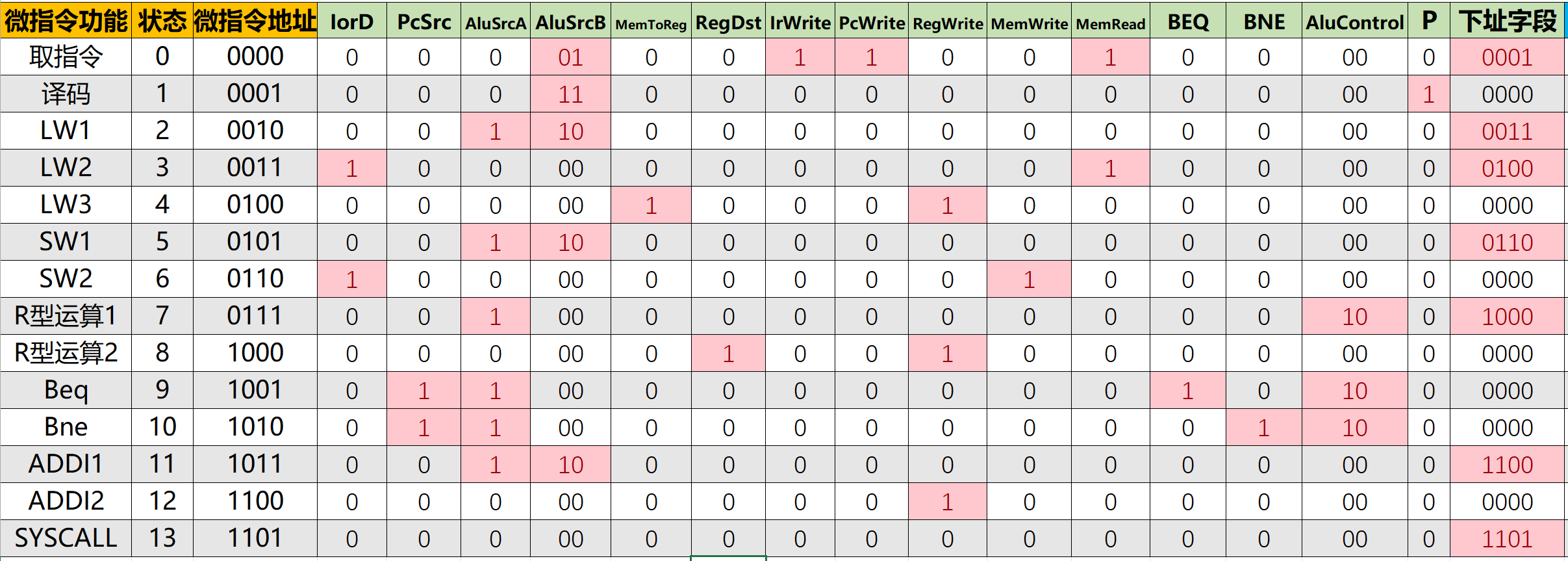
通过分析状态转移图，可以系统性地推导出实现所有机器指令所需的完整微指令数据，如下所示：

图2-11 微指令数据图

在每个时钟周期，微地址寄存器(μAR)中的微地址被送到控制存储器，取出对应的微指令。这条微指令被分成几个部分：操作控制字段直接产生微操作控制信号，驱动数据通路执行相应的微操作；同时，微指令的下地址字段和判别测试字段与来自指令寄存器的信息以及外部状态条件一起，送入地址转移逻辑，生成下一条微指令的地址。这个新生成的地址在下一个时钟脉冲到来时被载入μAR，从而开始下一个微周期的执行。这个循环不断进行，直到完成当前机器指令的所有微操作，然后跳转到下一条机器指令的微程序入口。

微指令控制器组成如下：

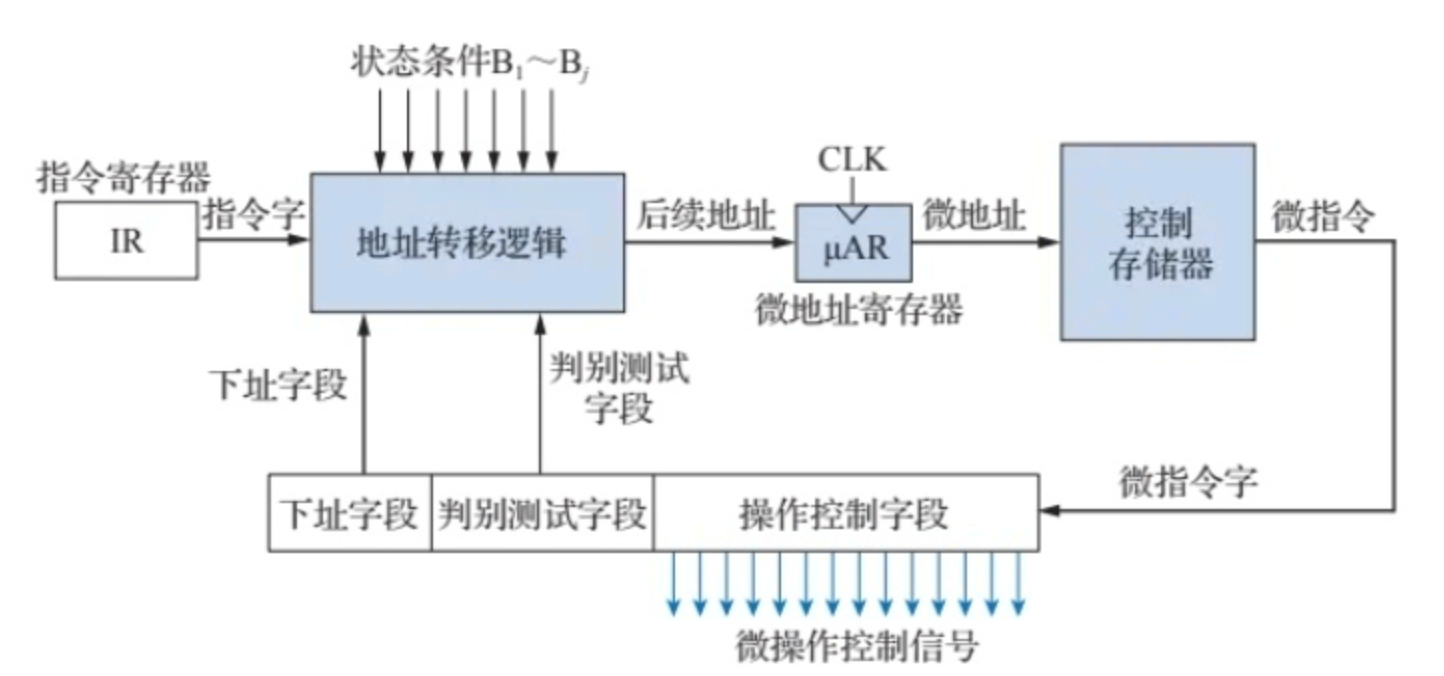


图2-12 微指令控制器组成

由此可以画出微指令控制器，具体结构如下：

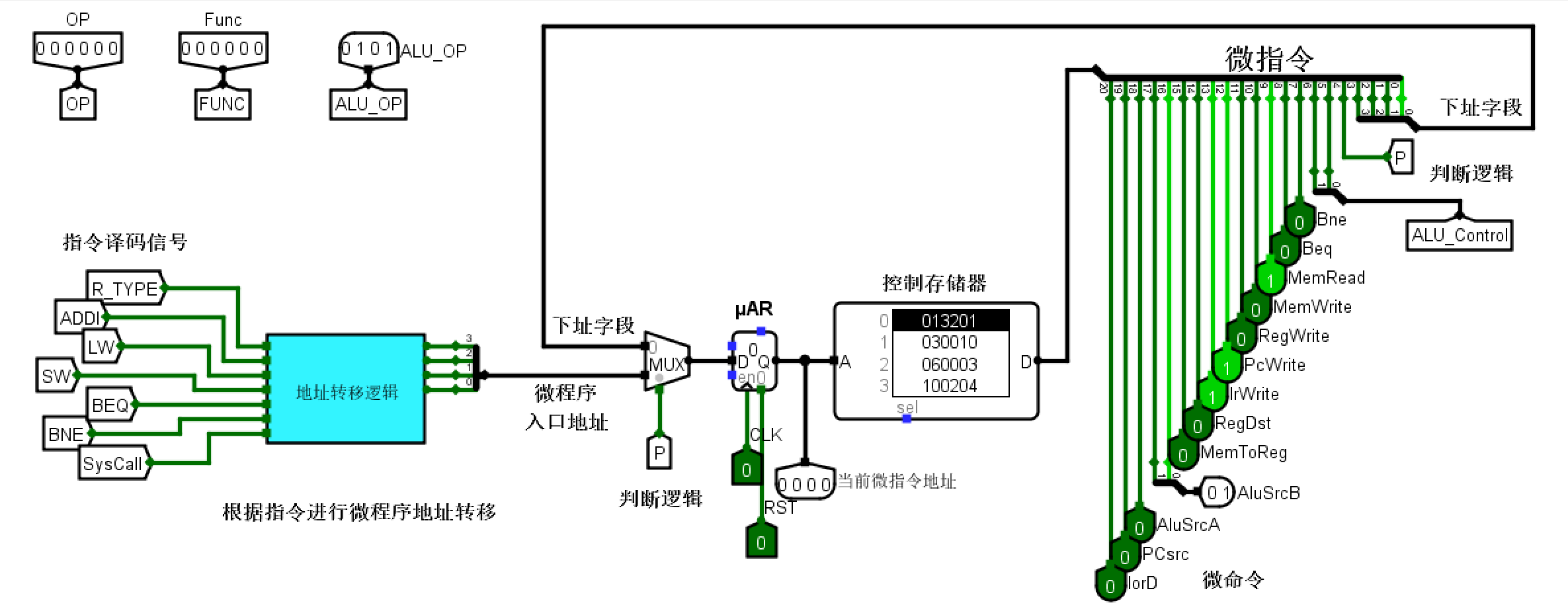


图2-13 微指令控制器电路

### 多周期硬布线控制器设计

多周期硬布线控制器是一种通过有限状态机和组合逻辑电路来分阶段生成控制信号的CPU控制器，它在性能上优于微程序控制器，但在灵活性和设计复杂性上不如微程序控制器。

其FSM设计如下。其中，在多周期硬布线控制器的大多数状态的状态转移的方式相对简单直接，通常只有一种预设的转移路径：要么是指令执行流程中的下一个确定周期（例如，从“执行”周期转移到“写入”周期），要么是当前指令执行完毕后无条件地回到“取指令”环节，为下一条指令的获取做准备。

然而，译码周期是一个显著的例外。在这个关键阶段，控制器不仅要对取出的指令进行解析，然后根据指令的类型和操作码来决定后续的状态转移。

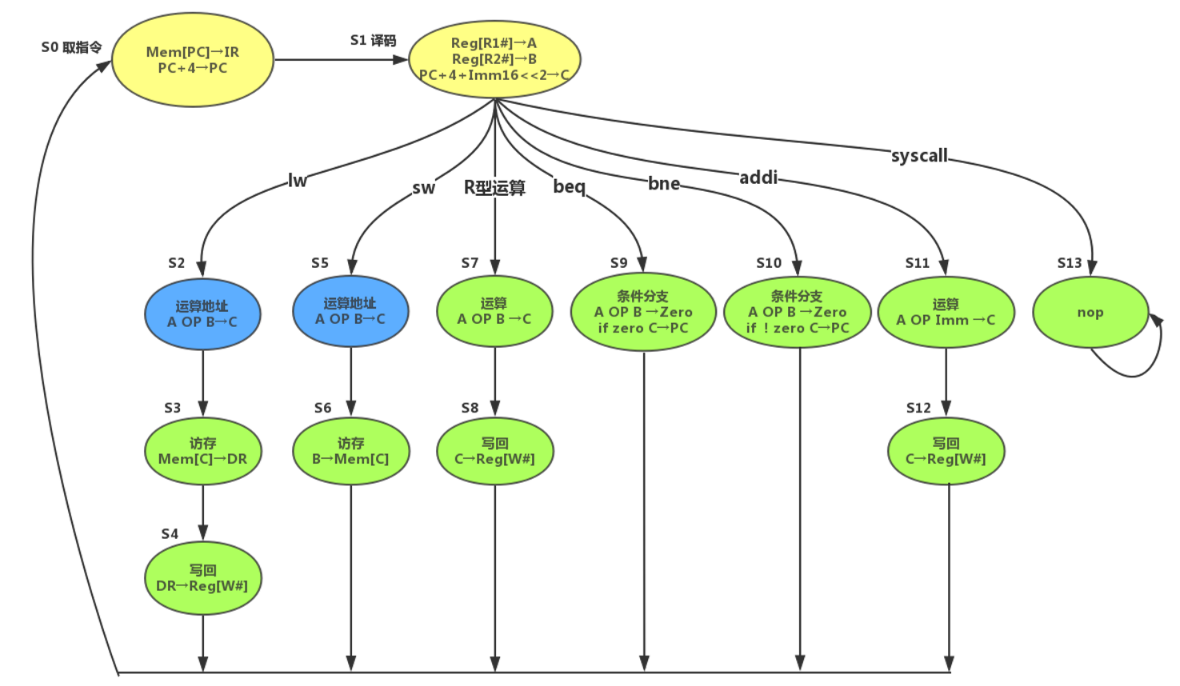


图2-14 控制器状态转移图

由此可得到状态转移表：

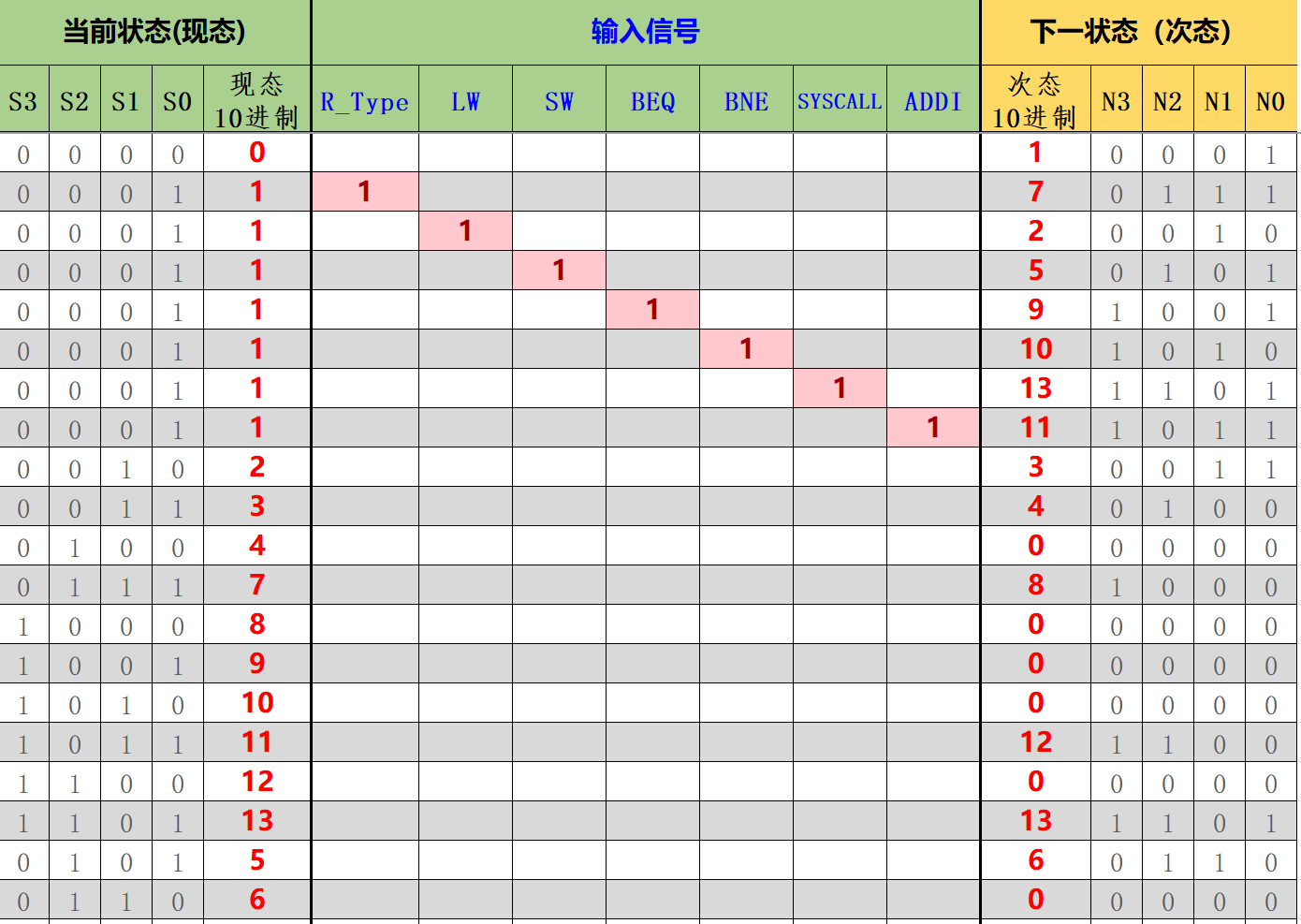


图2-15 状态转移表

### 多周期MIPS数据通路设计

多周期MIPS处理器通过控制器（蓝色部分，多周期微程序控制器MIPS只需将此更换为微程序控制器）根据指令操作码和当前执行阶段（状态）生成精细的控制信号，这些信号在不同时钟周期内协同驱动下层的数据通路，包括程序计数器（PC寄存器）、指令和数据存储器（下图中的RAM）、寄存器文件（黄色部分）、ALU以及IR，DR，A，B，C寄存器等核心组件，从而将8个MIPS指令的执行分解为多个时钟周期完成，实现了比单周期CPU更高的时钟频率和部件利用率。

其中有比单周期cpu更多的寄存器：**指令寄存器（IR）**用于在取指阶段之后，在后续周期内保存当前正在执行的机器指令，方便后续进行译码操作；**数据寄存器（DR）**则用于在访存阶段暂存从内存中读取的数据（例如LW指令的结果）或准备写入内存的数据（例如SW指令的数据），作为内存和寄存器文件之间的数据缓冲（由于指令和数据存储在同一个RAM中，所以我们需要判断其应该存入IR还有存入DR，所以会有IRWrite控制信号，防止IR被数据覆盖）；而**A和B寄存器**则分别用于在译码/读寄存器阶段之后，保存从寄存器文件读取的两个操作数，以便在随后的执行阶段将它们作为ALU的输入；最后，**C寄存器**（或AluResult寄存器）用于在执行阶段之后，暂存ALU的运算结果，这个结果可能在随后的访存阶段作为地址或数据写入内存，或者在写回阶段写入寄存器文件。

具体结构如下（多周期微程序控制器MIPS只需将此更换为微程序控制器）：

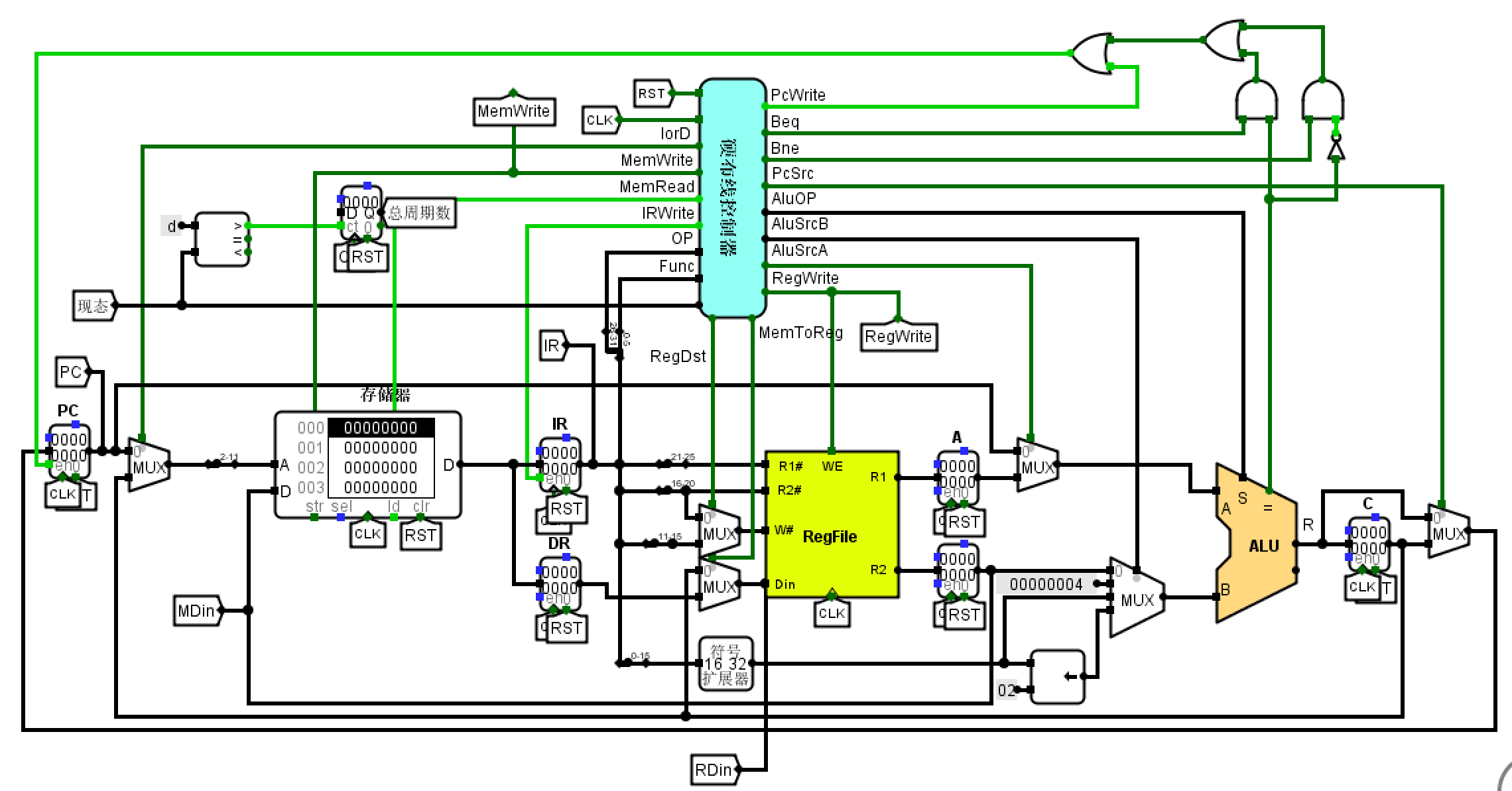


图2-16 多周期MIPS数据通路

## 故障与调试

### 单周期CPU的指令寄存器地址传入错误问题

**故障现象：**指令ROM访存地址的不匹配。

**原因分析：**PC的值代表的是内存中的字节地址。然而，指令ROM采用的是通过存储单元（32位）寻址，即每个地址单元存储一个完整的指令（4个字节）。

当PC的值的最后几位直接用于访问指令ROM时，就会出现问题。所以要去掉PC的低2位会错误地选择到同一个4字节指令中的不同字节，而不是指令本身。

**解决方案：**为了解决PC按字节寻址与指令ROM按存储单元寻址之间的不匹配，需要通过分线器将PC的低2位截断（用PC的2-11位，而不是0-9位）。这样通过PC得到该指令的起始地址。

错误结构如图：

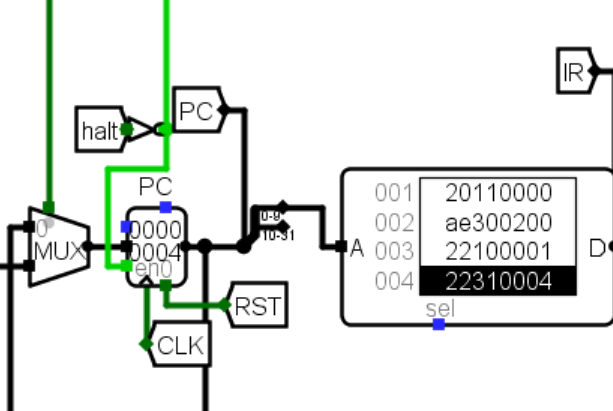


图2-17 单周期CPU的指令寄存器地址传入错误问题的错误结构

正确结构：

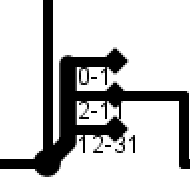


图2-18 单周期CPU的指令寄存器地址传入错误问题修改后

### 指令ROM为空问题

**故障现象：**运行时所有指令都为**0。**

**原因分析：**Logisim中的ROM模块被设计为只读存储器，这意味着它在仿真运行时不能被电路写入数据。它的主要用途是存储预设的、固定不变的程序代码或查找表数据。因此，ROM中的数据必须在电路设计阶段手动导入或编辑，而不是在仿真过程中通过输入引脚写入。

**解决方案：**手动加载指令数据 sort.hex。

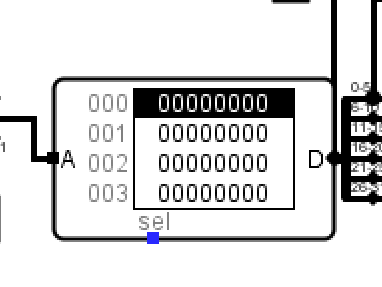


图2-19 指令ROM为空问题错误结构

### 扩展器扩展方案错误

**故障现象**：当原始立即数（Imm）为负数时，扩展后的数值是错误的，导致后续运算或逻辑判断出现意想不到的结果。

**原因分析**：立即数（Imm）通常是指令中的一个短字段，用于表示一个常量。在许多处理器架构中，这个短立即数需要被扩展到更长的位宽（例如从16位扩展到32位），以便与寄存器中的数据进行算术或逻辑运算。MIPS指令设计规定立即数是有符号的，所以在扩展时必须进行符号扩展。

错误结构如图：

### 

图2-20 扩展器扩展方案错误电路

## 测试与分析

### 单周期CPU

加载 sort.hex 后即可运行，执行完毕后，系统停机，可以看得时钟周期数为 224，也刚读取到syscall指令，如图：



图2-21 单周期CPU测试图

### 多周期CPU（微程序控制器和硬布线控制器）

（1）把 sort.hex 加载到RAM最开始（因为PC最开始为0），如图：

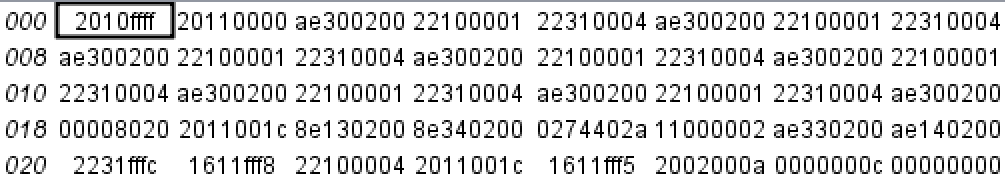


图2-22 对RAM填充代码

（2）运行之后可以看到，总周期数位891之后clk还在变化，但是cpu停止运行，如图：

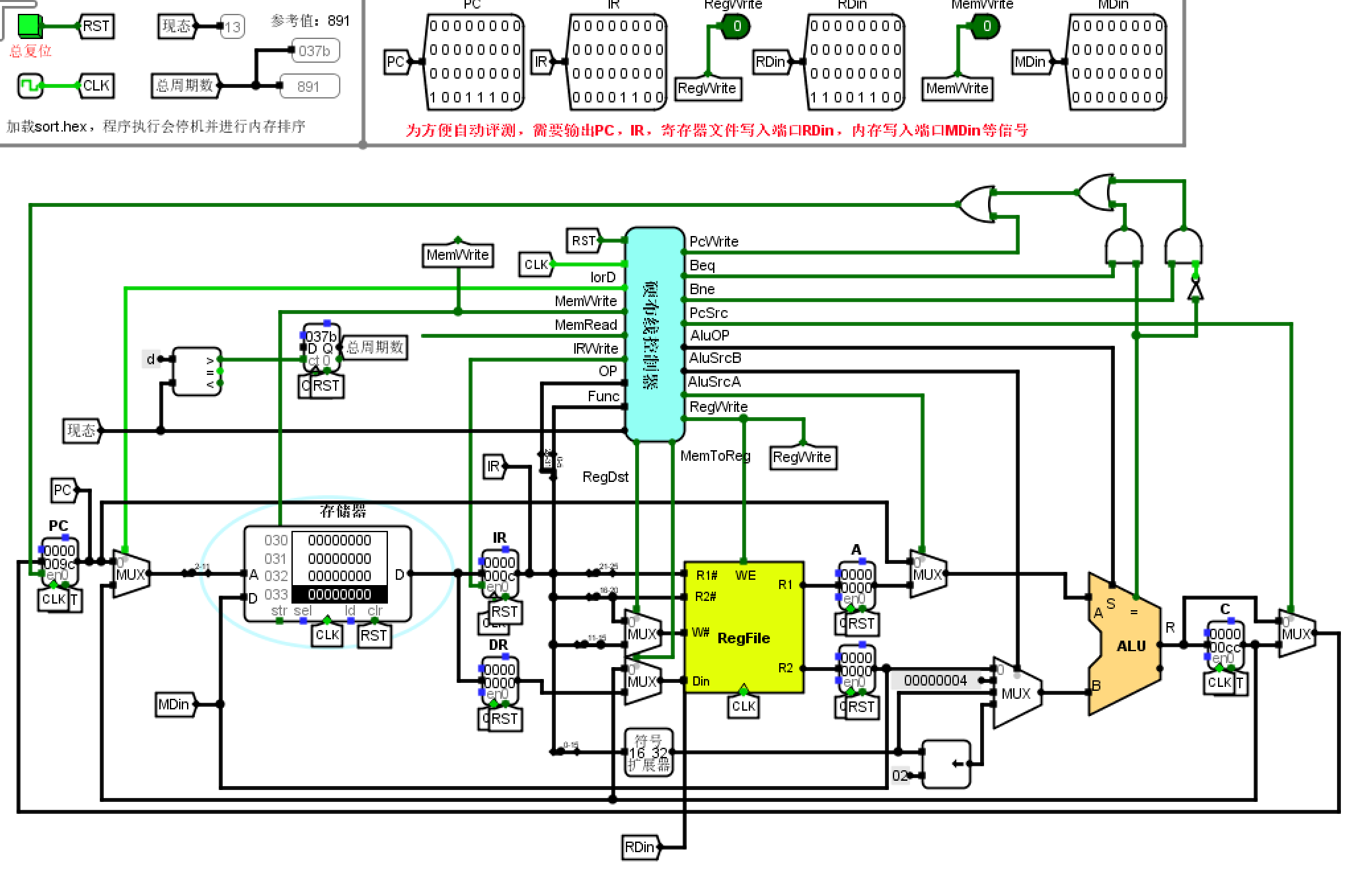


图2-23 多周期CPU测试

## 实验总结

本次实验主要完成了如下几点工作：

1） 方案总结：设计并实现了单周期 MIPS CPU和多周期 MIPS CPU，能够支持 8 条 MIPS 核心指令（ADD, SLT, LW, SW, BEQ, ADDI, BNE, SysCall），成功运行冒泡排序测试程序 sort.asm。单周期CPU采用模块化结构划分数据通路与控制路径，利用存储器和寄存器的异步读取以保证在一个周期内完成所有操作。多周期 MIPS 处理器通过将指令执行划分为多个时钟周期，实现了关键硬件资源的复用，降低了硬件复杂度。此外，多周期 MIPS 处理器实现了两种控制器方案：微程序控制器和硬布线控制器，并分析了各自的优缺点及设计原理。

2） 功能总结： 实现了控制器的指令解析功能，能够对 32 位输入操作码进行分解，并根据操作码（op）和功能码（func）判断指令类型（R型、I型或系统调用），该译码逻辑在三种 CPU 设计中通用；实现了单周期硬布线控制器，根据不同指令类型生成 RegWrite, MemToReg, ALUSrc, MemWrite, Beq, Bne 等控制信号，以协调 CPU 正确执行相应功能；实现了单周期 MIPS 数据通路，集成了 PC 寄存器、指令存储器、寄存器文件、ALU 和数据存储器，在一个时钟周期内完成指令的取指、译码、执行、访存和写回操作；实现了多周期微程序控制器，通过将每条机器指令分解为一系列微操作，并将微指令固化在控制存储器（ROM）中；实现了多周期硬布线控制器，采用有限状态机和组合逻辑电路分阶段生成控制信号，并在译码周期根据指令类型和操作码决定后续状态转移；实现了多周期 MIPS 数据通路，包含了 IR、DR、A、B、C 寄存器等比单周期更多中间寄存器，合并了存储器和运算器，支持更精细的指令执行阶段控制。

3）进行了故障与调试：解决了单周期 CPU 的指令寄存器地址传入错误问题，指令 ROM 为空问题，扩展器扩展方案错误问题等。

4） 进行了测试与分析： 对3个CPU分别加载sort.hex进行测试，cpu停止之后总周期数正确，提交之后能够通关。

## 实验心得

本次 CPU 设计实验让我对计算机组成原理有了更深入的理解和实践体验。通过从零开始构建单周期和多周期 MIPS 处理器，掌握了 MIPS 指令的执行流程，也深入理解了数据通路和控制信号的协同工作原理，并且深刻体会到了理论知识与实际电路设计之间的差距和联系。

实验总体质量很高，帮我节省了很多体力活，但还是一点点操作细节确实带来了无意义时间耗费。例如，在 Logisim 中连接分线器（splitter）时，需要逐位进行点击。但总体上瑕不掩瑜，应该是本科生涯中质量前几的实验之一。

|  |
| --- |
| 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: 李语心** |
| 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:** |