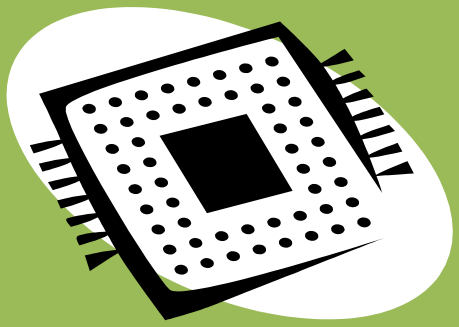
华中科技大学

2024

逻辑与计算机系统设计 ·实验报告·

专　　业：	陈宇航
班　　级：	本硕博 2301
学　　号：	U202315752
姓　　名：	陈宇航
电　　话：	13428122178
邮　　件：	2663767199@qq.com
完成日期：	2025-06-10



计算机科学与技术学院

1 运动码表系统设计

1.1 设计要求

利用 logisim 平台中现有运算部件构建一个运动码表系统，主要包括 7 段数码管驱动电路、2 选 1 选择器（1 位或 16 位）、无符号比较器设计（4 位或 16 位）、并行加载寄存器（4 位或 16 位）、BCD 计数器状态机及输出函数、BCD 计数器（1 位十进制）、码表计数器（4 位十进制）、码表显示驱动、码表控制器状态机及输出函数、码表控制器、码表数据通路的设计。系统集成后，可支持开始计时、停止计时、存储时间、复位、保存或显示最短时间等功能。

1.2 方案设计

1.2.1 7 段数码管驱动电路设计

首先观察数码管，共有 7 段（这里不考虑小数点），如图 1-1 所示。这 7 段中每一段都由一个独立信号控制，信号为 1 时代表该管亮起，为 0 时代表不亮起。特别注意，8 对应的引脚表示那位小数点。

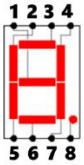


图 1-1：7 段数码管结构示意图

电路引脚及功能如表 1-1 所示。

表 1-1：7 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

因此，在设计数码管电路时，首先根据 4 位 BCD 码（8421 BCD 码）计算出对应输入十进制下的值。设 BCD 码从高位到低位分别为 X3、X2、X1、X0，则计算的方式是：

$$X = 2^3 \times X_3 + 2^2 \times X_2 + 2^1 \times X_1 + 2^0 \times X_0 = 8X_3 + 4X_2 + 2X_1 + X_0$$

接下来找出这个值在 7 段数码管显示下具体亮起的数码管，对这些亮起的数码管，信号赋为 1，其余的数码管信号赋为 0 即可。具体实验中，通过 Project→Analyze Circuit（分析组合逻辑电路）菜单下的 Table 功能，补全 7 段数码管驱动信号的真值表即可，如图 1-2 所示。

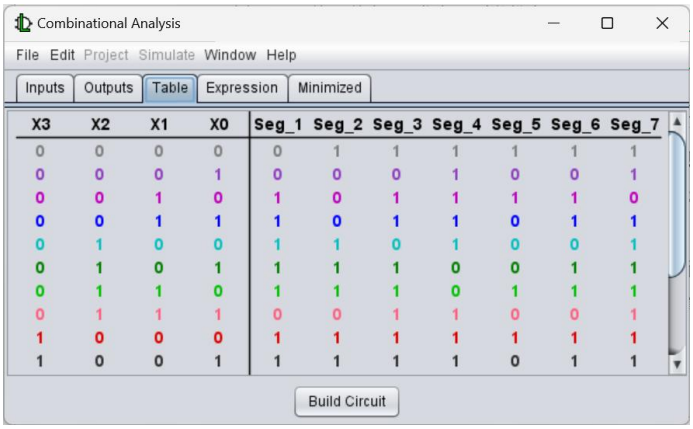


图 1-2：7 段数码管电路真值表

随后，点击“Build Circuit”自动生成电路即可实现预期的功能。Logisim 中的电路图如图 1-3 所示。

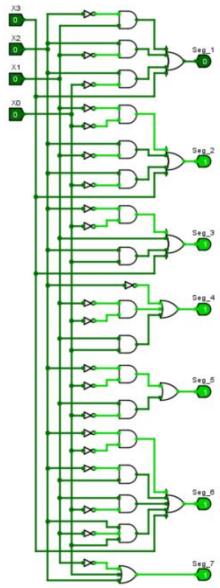


图 1-3：7 段数码管电路图

通过时钟信号控制输入有效的 BCD 码（0000~1001）进行测试，观察数码管显示状态，数码管可以稳定地按照预期设计显示正确的十进制输入图案，测试结果说明该电路正确且符合设计要求。

通过分析 Logisim 中生成的卡诺图可知，Seg_1 的输出存在险象（见图 1-4 Seg_1 的卡诺图）。当输入中 X0 = 0,X1 = 1, X3 = 0 时候，输出为 ~X2+X2 ，存在险象。因此可以在输出函数中添加冗余项~X0X1~X3 来消除险象再重新生成电路即可。经检查，其他输出函数未发现险象。

图 1- 4 Seg_1 的卡诺图

1.2.2 2 选 1 选择器设计（1 位）

本关要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 1-2 所示。

表 1-2：2 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
X1~X0	输入	1	2 路输入的第 0 路和第 1 路
Sel	输入	1	选择控制端
Out	输出	1	若 Sel=0，则选择 X0，否则选择 X1

由于输入的数据只有 1 位，这里只涉及到基本的逻辑表达式，即

$$Out = Sel \cdot X1 + \overline{Sel} \cdot X0$$

根据上面的表达式，很容易能画出电路结构图。电路图如图 1-5 所示。

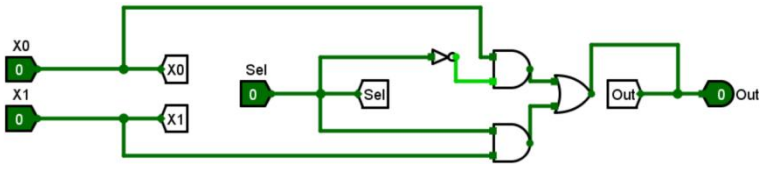


图 1-5：2 选 1 选择器（1 位）电路图

使用不同组合测试 1 位 2 选 1 选择器和 8 位 2 选 1 选择器中控制 Sel 是否选择正确的输入信号。经多次测试，当 Sel=0 时可以正确选择并输出 X0 的输入信号，当 Sel=1 时可以正确选择并输出 X1 的输入信号。两种选择器均能正确实现功能，说明两种电路设计均正确。

图 1- 6 二路选择器（1 位） 卡诺图

通过分析卡诺图（见图 3-17 二路选择器（1 位） 卡诺图）可知，1 位二路选择器的表达式存在险象，当 X1 X0 = 11 的时候，输出为 Sel + ~Sel，可以通过增加冗余项 X1 X0 来消除险象。修改后的电路图如下图 1-7 消除险象后的二路选择器电路图（1 位）。对于二路选择器（16 位）的险象处理也是同样的道理。

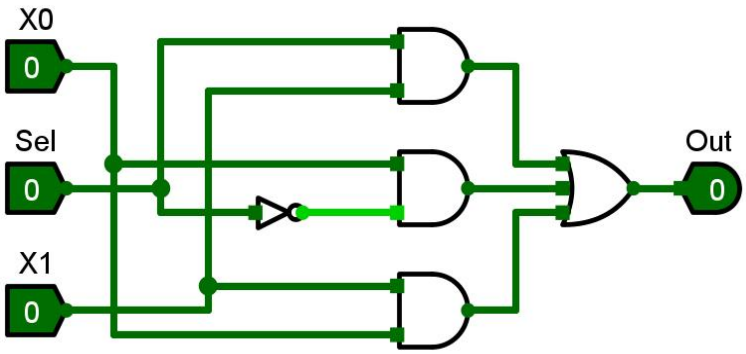


图 1- 7 消除险象后的二路选择器电路图（1 位）

1.2.3 2 选 1 选择器设计（16 位）

本关要在上一关的 1 位 2 路选择器的基础上，构建 16 位的 2 路选择器。很容易想到的思路是把输入的 16 位数据逐个拆开，拆成 16 个 1 位的数据，并使用 16 个 1 位 2 路选择器进行选择即可。这里，电路的引脚和功能和表 1-2 中的基本一致，唯一的区别在于 X0 和 X1 都是 16 位的。

利用分线器（Splitter）分离开 X0 和 X1 的 16 位数据，并分别置入 16 个 1 位 2 路选择器的输入端，将结果按位输出到 Out 即可。电路图如图 1-8 所示。

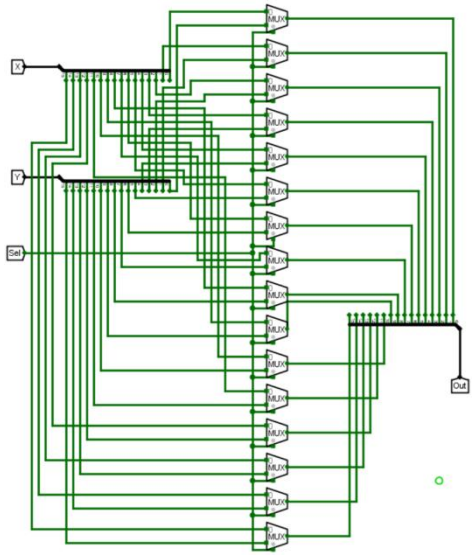


图 1-8：2 选 1 选择器（16 位）电路图

1.2.4 无符号比较器设计（4 位）

本关要设计实现四位无符号数的比较器。电路引脚及功能如表 1-3 所示。

表 1-3：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

对于四位无符号比较器，可以将四位分解成两个二位（高位，低位之分），一个 2 位无符号比较器先对两个高位进行比较，若高位可以比较出大小则比较器直接输出结果 Great/Less，若高位相同，则输出 Equal 并用第二个 2 位无符号比较器继续进行低位比较，可以比较出大小则输出 Great/Less，若到最低位数字都相等，则输出 Equal。因此通过对两个 2 位无符号比较器的输出进行改造，将输出增加相应的与门和非门，即可将两个 2 位无符号比较器改装成四位无符号比较器。实际上，可以利用逻辑表达

式的方式简化这一问题：

若 X=Y，则显然有$X3 = Y3, X2 = Y2, X1 = Y1, X0 = Y0$成立，即输入 X 和 Y 在每一位上都相等。

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，也就是 Y>X，则满足$Y3 > X3$，或$Y3 = X3$且$Y2 > X2$，或$Y3 = X3$且$Y2 = X2$且$Y1 > X1$，或$Y3 = X3$且$Y2 = X2$且$Y1 = X1$且$Y0 > X0$。

据此写出 Great、Equal 和 Less 的逻辑表达式（两个变量的与运算用空格表示，或运算用“+”表示，非运算用“~”表示）：

$$\begin{aligned}Great = {} & X0\ {\sim}Y3\ {\sim}Y2\ {\sim}Y1\ {\sim}Y0 + X1\ {\sim}Y3\ {\sim}Y2\ {\sim}Y1 + X1\ X0\ {\sim}Y3\ {\sim}Y2\ {\sim}Y0 + X2\ {\sim}Y3\ {\sim}Y2 \\ & + X2\ X0\ {\sim}Y3\ {\sim}Y1\ {\sim}Y0 + X2\ X1\ {\sim}Y3\ {\sim}Y1 + X2\ X1\ X0\ {\sim}Y3\ {\sim}Y0 + X3\ {\sim}Y3 \\ & + X3\ X0\ {\sim}Y2\ {\sim}Y1\ {\sim}Y0 + X3\ X1\ {\sim}Y2\ {\sim}Y1 + X3\ X1\ X0\ {\sim}Y2\ {\sim}Y0 \\ & + X3\ X2\ {\sim}Y2 + X3\ X2\ X0\ {\sim}Y1\ {\sim}Y0 + X3\ X2\ X1\ {\sim}Y1 + X3\ X2\ X1\ X0\ {\sim}Y0\end{aligned}$$

$$\begin{aligned}Equal = {} & {\sim}X3\ {\sim}X2\ {\sim}X1\ {\sim}X0\ {\sim}Y3\ {\sim}Y2\ {\sim}Y1\ {\sim}Y0 + {\sim}X3\ {\sim}X2\ {\sim}X1\ X0\ {\sim}Y3\ {\sim}Y2\ {\sim}Y1\ Y0 \\ & + {\sim}X3\ {\sim}X2\ X1\ {\sim}X0\ {\sim}Y3\ {\sim}Y2\ Y1\ {\sim}Y0 + {\sim}X3\ {\sim}X2\ X1\ X0\ {\sim}Y3\ {\sim}Y2\ Y1\ Y0 \\ & + {\sim}X3\ X2\ {\sim}X1\ {\sim}X0\ {\sim}Y3\ Y2\ {\sim}Y1\ {\sim}Y0 + {\sim}X3\ X2\ {\sim}X1\ X0\ {\sim}Y3\ Y2\ {\sim}Y1\ Y0 \\ & + {\sim}X3\ X2\ X1\ {\sim}X0\ {\sim}Y3\ Y2\ Y1\ {\sim}Y0 + {\sim}X3\ X2\ X1\ X0\ {\sim}Y3\ Y2\ Y1\ Y0 \\ & + X3\ {\sim}X2\ {\sim}X1\ {\sim}X0\ Y3\ {\sim}Y2\ {\sim}Y1\ {\sim}Y0 + X3\ {\sim}X2\ {\sim}X1\ X0\ Y3\ {\sim}Y2\ {\sim}Y1\ Y0 \\ & + X3\ {\sim}X2\ X1\ {\sim}X0\ Y3\ {\sim}Y2\ Y1\ {\sim}Y0 + X3\ {\sim}X2\ X1\ X0\ Y3\ {\sim}Y2\ Y1\ Y0 \\ & + X3\ X2\ {\sim}X1\ {\sim}X0\ Y3\ Y2\ {\sim}Y1\ {\sim}Y0 + X3\ X2\ {\sim}X1\ X0\ Y3\ Y2\ {\sim}Y1\ Y0 \\ & + X3\ X2\ X1\ {\sim}X0\ Y3\ Y2\ Y1\ {\sim}Y0 + X3\ X2\ X1\ X0\ Y3\ Y2\ Y1\ Y0\end{aligned}$$

$$\begin{aligned}Less = {} & {\sim}X3\ {\sim}X2\ {\sim}X1\ {\sim}X0\ Y0 + {\sim}X3\ {\sim}X2\ {\sim}X1\ Y1 + {\sim}X3\ {\sim}X2\ {\sim}X0\ Y1\ Y0 + {\sim}X3\ {\sim}X2\ Y2 \\ & + {\sim}X3\ {\sim}X1\ {\sim}X0\ Y2\ Y0 + {\sim}X3\ {\sim}X1\ Y2\ Y1 + {\sim}X3\ {\sim}X0\ Y2\ Y1\ Y0 + {\sim}X3\ Y3 \\ & + {\sim}X2\ {\sim}X1\ {\sim}X0\ Y3\ Y0 + {\sim}X2\ {\sim}X1\ Y3\ Y1 + {\sim}X2\ {\sim}X0\ Y3\ Y1\ Y0 \\ & + {\sim}X2\ Y3\ Y2 + {\sim}X1\ {\sim}X0\ Y3\ Y2\ Y0 + {\sim}X1\ Y3\ Y2\ Y1 + {\sim}X0\ Y3\ Y2\ Y1\ Y0\end{aligned}$$

然后自动生成电路即可。电路图如图 1-9 所示。

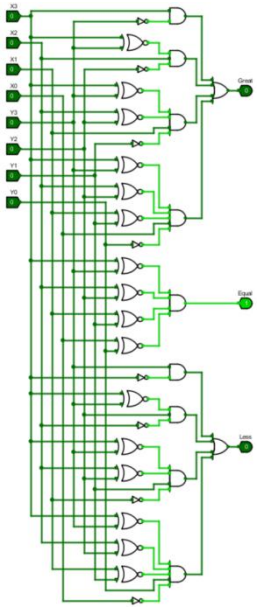


图 1-9：无符号比较器（4 位）电路图

1.2.5 无符号比较器设计（16 位）

本关要利用上面设计好的 4 位无符号比较器构建 16 位无符号比较器。这里，电路的引脚和功能和表 1-3 中的基本一致，唯一的区别在于 X 和 Y 都是 16 位的。

在实验中，可以把 16 位的输入分成四部分，每部分包含 4 位，并分别输入到四个 4 位无符号比较器中。然后，综合这四个 4 位无符号比较器输出的结果，得到整体的比较结果。

若 X=Y，则四个比较器的结果显然都为 Equal，即输入 X 和 Y 在每一位上都相等。

若 X>Y，则满足最高位比较器的结果为 Great，或最高位比较器的结果为 Equal 且次高位比较器的结果为 Great，或最高位比较器的结果为 Equal 且次高位比较器的结果为 Equal 且次次高位比较器的结果为 Great，或高位比较器的结果为 Equal 且次高位比较器的结果为 Equal 且次次高位比较器的结果为 Equal 且最低位比较器的结果为 Great。

若 X<Y，也就是 Y>X，与上面的分析类似。

根据上述分析即可连接电路。电路图如图 1-10 所示。

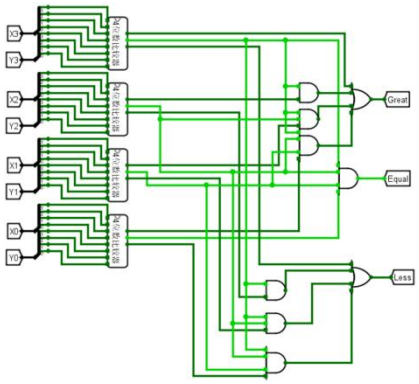


图 1-10：无符号比较器（16 位）电路图

1.2.6 并行加载寄存器设计（4 位）

下面要利用 D 触发器构成 4 位并行加载寄存器，要求上升沿触发，具有高电平使能端。寄存器是由具有存储功能的触发器组合而成的，其中一个触发器存储一位二进制代码。因此，存放 4 位二进制代码的寄存器，需要 4 个 D 触发器构成。电路引脚及功能如表 1-4 所示。

表 1-4：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 且 Clock 上升沿时 Q=Din

D 触发器的特点是其次态和现态一致，因此只需要把输入用分线器分开，分别输入 4 个 D 触发器的“D”输入端，并将这 4 个 D 触发器的次态输出按位输出到 Q 即可。注意 D 触发器需要接入时钟 Clock 和使能端 En，否则无法正常工作。电路图如图 1-11 所示。

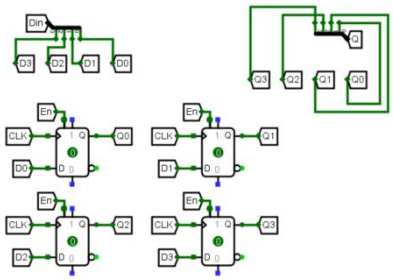


图 1-11：并行加载寄存器（4 位）电路图

1.2.7 并行加载寄存器设计（16 位）

本关要利用上一关设计好的 4 位并行加载寄存器构建 16 位并行加载寄存器，要求沿上升沿触发，且具有高电平使能端。这里，电路的引脚和功能和表 1-4 中的基本一致，唯一的区别在于输入 Din 和输出 Q 都是 16 位的。

分析封装好的 4 位寄存器，发现其输入有时钟、4 位输入数据和使能端。因此，在处理 16 位并行加载寄存器时，可以使用 4 个封装好的 4 位寄存器，按要求对 Q 和 Din 进行分线即可。电路图如图 1-12 所示。

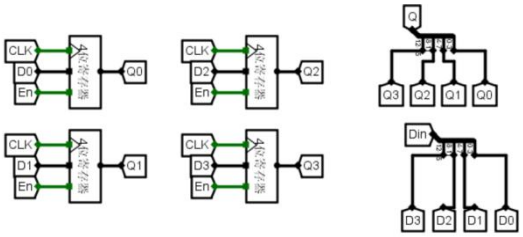


图 1-12：并行加载寄存器（16 位）电路图

1.2.8 BCD 计数器状态机设计

本关要设计 8421 BCD 码（十进制）计数器的状态机。电路引脚及功能如表 1-5 所示。

表 1-5：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	输入	4	次态输出 N，N=S+1

根据题目要求，在“同步时序电路状态转换表.xlsx”中完成状态转换真值表。由

于是计数器，因此现态为0时次态为1，现态为1时次态为2，以此类推，现态为$i(0 \le i \le 8)$时次态为$i+1$，现态为 9 时次态为 0（代表进位了）。由此即可画出状态转换真值表，如图 1-10 所示。注意这里没有输入信号，不需要填写表格中间的部分。

当前状态(现态)					输入信号								下一状态（次态）				
S3	S2	S1	S0	现态10进制	in1	in2	in3	in4	in5				次态10进制	N3	N2	N1	N0
0	0	0	1	1									2	0	0	1	0
0	0	1	0	2									3	0	0	1	1
0	0	1	1	3									4	0	1	0	0
0	1	0	0	4									5	0	1	0	1
0	1	0	1	5									6	0	1	1	0
0	1	1	0	6									7	0	1	1	1
0	1	1	1	7									8	1	0	0	0
1	0	0	0	8									9	1	0	0	1
1	0	0	1	9									0	0	0	0	0
0	0	0	0	0									1	0	0	0	1

图 1-13：BCD 计数器状态转换真值表

填写完成后，在 Excel 表格的“状态转换函数自动生成”下自动生成次态输出逻辑表达式。其中，$N3 = \sim S3\ S2\ S1\ S0\ +\ S3\ \sim S2\ \sim S1\ \sim S0$，$N2 = \sim S3\ \sim S2\ S1\ S0\ +\ \sim S3\ S2\ \sim S1\ +\ \sim S3\ S2\ \sim S0$，$N1 = \sim S3\ \sim S1\ S0\ +\ \sim S3\ S1\ \sim S0$，$N0 = \sim S3\ \sim S0\ +\ \sim S2\ \sim S1\ \sim S0$。然后在 Logisim 中自动生成电路即可，电路图如图 1-14 所示。

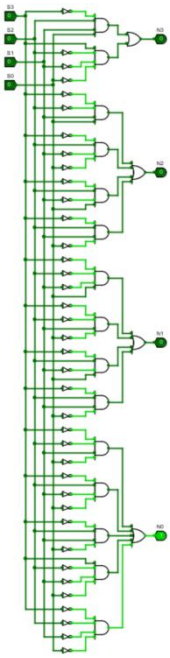


图 1-14：BCD 计数器状态机电路图

1.2.9 BCD 计数器输出函数设计

本关要设计 BCD 计数器的输出函数组合逻辑，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 1-6 所示。

表 1-6：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

具体来说，输出当且仅当当前状态 S=9 时（即 S3=S0=1，S2=S1=0）为 1，其余时刻均为 0。因此可以直接手动绘制电路。电路图如图 1-15 所示。

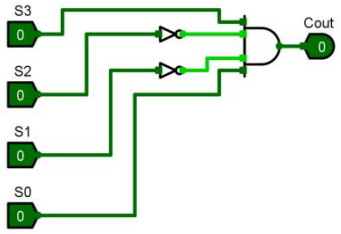


图 1-15：BCD 计数器输出函数电路图

1.2.10 BCD 计数器设计（1 位十进制）

下面要利用已经设计好的 BCD 计数器的状态机、输出函数，采用 D 触发器构建最终的 BCD 计数器。电路引脚及功能如表 1-7 所示。

表 1-7：BCD 计数器（1 位十进制）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 时输出为 1

首先利用分线器把 Q 分为 4 位，并将 Q3~Q0 的每一位都输入到状态机中，得到状态转换后的次态 QN3~QN0。然后将次态的 4 位数分别输入 4 个 D 触发器，使其输出覆盖现态的 Q3~Q0。最后，同样还要将新的“现态”Q3~Q0 输入到封装好的输出函数模块，从而得到进位输出。电路图如图 1-16 所示。

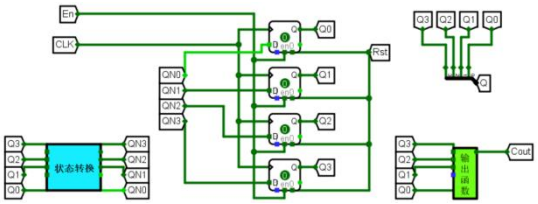


图 1-16：BCD 计数器（1 位十进制）电路图

1.2.11 码表计数器设计（4 位十进制）

本关要利用上一关已经设计好的 BCD 计数器，级联构建 4 位十进制计数器。由于码表的计时范围为 00.00~99.99，因此需要 4 位十进制计数器。这里，电路的引脚和功能和表 1-7 中的基本一致，唯一的区别在于没有进位输出 Cout 了。

设计过程中，需要考虑低位向高位进位的影响。第i个十进制计数器开始工作需要满足前$i-1$个十进制计数器的进位输出均为 1，因此需要用与门连接到第 2~4 个计数器的使能 En 端。电路图如图 1-17 所示。

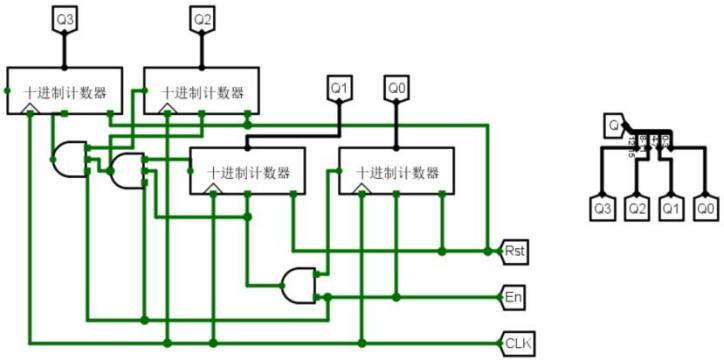


图 1-17：码表计数器（4 位十进制）电路图

1.2.12 码表显示驱动设计

本关要利用第 1 关的 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 1-8 所示。

表 1-8：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 码）
DispInfo	输出	32	4 个 8 段（7 段+小数点）数码驱动信号

首先用分线器把 Din 分为四个部分，每部分包含 4 位数字。然后分别把这四部分输入到封装好的数码管驱动模块中。注意到码表的显示格式为 xx.xx，也就是第二个数码管驱动电路的第 8 段数码驱动信号为 1，从而实现显示小数点的功能，而其余三个数码管驱动电路的第 8 段数码驱动信号均为 0。电路图如图 1-18 所示。

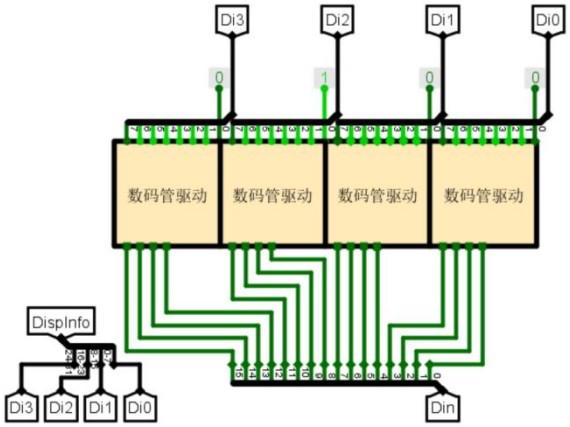


图 1-18：码表显示驱动电路图

1.2.13 码表控制器状态机设计

本关要设计码表控制器的状态机（即状态转换组合逻辑）。

电路引脚及功能如表 1-9 所示。

表 1-9：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，记录恢复为 99.99
newrecord	输入	1	新记录信号
S2~S0	输入	3	现态 S
N2~N0	输出	3	次态 N

首先分析码表可能存在的状态，主要有复位、清零、计时、计时停止、存储、查看等。根据码表功能，可以绘制出码表的状态转移图，如图 1-19 所示。

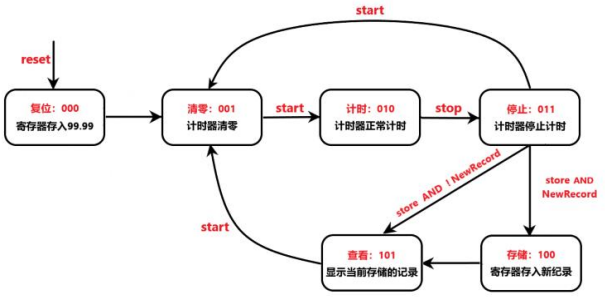


图 1-19：码表控制器状态机状态转移图

根据这一状态转移图，可以在“同步时序电路状态转换表.xlsx”中完成状态转换真值表，如图 1-20 所示。

当前状态(现态)					输入信号								下一状态（次态）				
S3	S2	S1	S0	现态10进制	start	stop	store	reset	NewRecord				次态10进制	N3	N2	N1	N0
								1					0	0	0	0	0
0	0	0	0	0				0					1	0	0	0	1
0	0	0	1	1	1			0					2	0	0	1	0
0	0	0	1	1	0			0					1	0	0	0	1
0	0	1	0	2		1		0					3	0	0	1	1
0	0	1	1	3	1			0					1	0	0	0	1
0	0	1	1	3			1	0	1				4	0	1	0	0
0	0	1	1	3			1	0	0				5	0	1	0	1
0	1	0	0	4				0					5	0	1	0	1
0	1	0	1	5	1			0					1	0	0	0	1
0	1	0	1	5	0			0					5	0	1	0	1
0	0	1	0	2		0		0					2	0	0	1	0
0	0	1	1	3	0		0	0					3	0	0	1	1

图 1-20：码表控制器状态转换真值表

需要注意的是，状态在转移过程中，除了图中显式标明的状态转移路径，还存在一些隐含的状态转移条件和状态维持条件。具体列举如下：

（1）只有给出 reset 指令时，才会转移到状态 0（复位），其它任何条件下都不会转移到状态 0；

（2）在状态为 1（清零）时，若不给出 start 指令，则会一直维持状态 1；

（3）状态为 4（存储）时，若不给出任何指令，则会直接跳转到状态 5（查看）。

填写完成后，在 Excel 表格的“状态转换函数自动生成”下自动生成次态输出逻辑表达式。其中，$N2, N1, N0$ 满足下面的逻辑表达式：

$$N2 = \sim start \sim reset\, S2 \sim S1 \;+\; \sim reset\, S2 \sim S1 \sim S0 \;+\; store \sim reset \sim S2\, S1\, S0$$

$$N1 = \sim start \sim store \sim reset \sim S2\, S1 \;+\; \sim reset \sim S2\, S1 \sim S0$$

$$+\; start \sim reset \sim S2 \sim S1\, S0$$

$$N0 = \sim start \sim reset \sim S1 \ + \ \sim reset \sim S1 \sim S0 \ + \ \sim start \sim store \sim reset \sim S2 \ S0$$
$$+ \ \sim start \sim reset \sim NewRecord \sim S2 \ S0 \ + \ \sim reset \ S2 \sim S1$$
$$+ \ stop \sim reset \sim S2 \sim S0 \ + \ start \sim reset \sim S2 \ S1 \ S0$$

然后在 Logisim 中自动生成电路即可，电路图如图 1-21 所示。

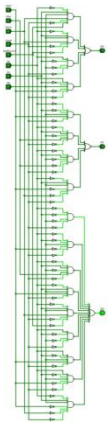


图 1-21：码表控制器状态转换电路图

1.2.14 码表控制器输出函数设计

本关要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 1-10 所示。

表 1-10：码表控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	当前状态 S（PS）
SDsel	输出	1	最好成绩记录的选择信号
SDen	输出	1	保存最好成绩记录的寄存器的使能信号
DPsel	输出	1	显示计时成绩记录的选择信号
TMen	输出	1	码表计时器使能信号
TMreset	输出	1	码表计时器复位信号

根据题目给定的测试用例，可以作出码表控制器的输出函数真值表，如图 1-22 所示。注意这里没有输入信号，不需要填写表格中间的部分。

当前状态(现态)					输入信号								输出				
S3	S2	S1	S0	现态10进制									SDSel	SDEN	DPSEL	TMen	TMReset
0	0	0	0	0										1	1		1
0	0	0	1	1											1		1
0	0	1	0	2											1	1	
0	0	1	1	3											1		
0	1	0	0	4									1	1	1		
0	1	0	1	5													

图 1-22：码表控制器输出函数真值表

填写完成后，在 Excel 表格的“输出函数自动生成”下自动生成输出函数的逻辑表达式。其中，5 个不同的输出分别满足以下逻辑表达式：

$SDsel = S2 \sim S1 \sim S0$，$SDen = \sim S1 \sim S0$，$DPsel = \sim S2 + \sim S1 \sim S0$，

$TMen = \sim S2\ S1 \sim S0$，$TMreset = \sim S2 \sim S1$

然后在 Logisim 中自动生成电路即可，电路图如图 1-23 所示。

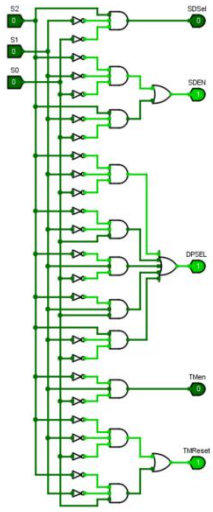


图 1-23：码表控制器输出函数电路图

1.2.15 码表控制器设计

跟据同步时序逻辑电路基本模型，利用已经设计好的码表控制器的状态机（状态转换组合逻辑）、输出函数，采用 D 触发器（或寄存器）构建最终的码表控制器，完成有限状态机的组装。

至此，基本已经可以整合码表的全部功能，包括开始计时、停止计时、存储计时记录、计时复位、选择最好成绩记录、保存最好成绩记录、显示计时记录、计时器复位等。电路引脚及功能如表 1-11 所示。

表 1-11：码表控制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	时钟脉冲
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，记录恢复为 99.99
newrecord	输入	1	新的最好成绩记录信号
SDsel	输出	1	最好成绩记录的选择信号
SDen	输出	1	保存最好成绩记录的寄存器的使能信号
DPsel	输出	1	显示计时成绩记录的选择信号
TMen	输出	1	码表计时器使能信号
TMreset	输出	1	码表计时器复位信号

这一关是对前面的模块的级联和汇总，按照封装后的状态转换模块和输出函数模块完成对应的连线即可。

在状态转换过程中，设 S2~S0 为现态，N2~N0 为次态，这六个引脚都是中间变量。电路图如图 1-24 所示。

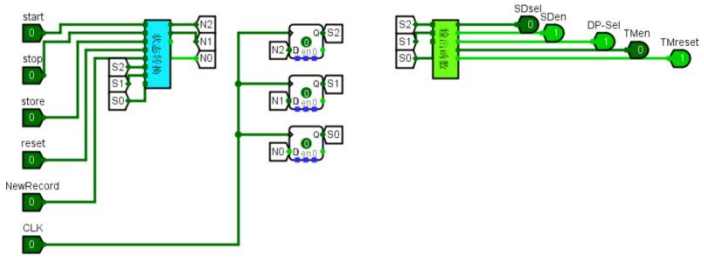


图 1-24：码表控制器电路图

1.2.16 运动码表数据通路设计（系统集成）

这是整个运动码表实验的最后一步，需要利用前面 15 关中设计好的选择器、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 1-12 所示。

表 1-12：运动码表数据通路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	时钟脉冲
TestMode	输入	1	在线评测开关，0 为本地模式，1 为在线模式
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，记录恢复为 99.99
Time	输出	16	计时成绩或成绩记录

首先把码表控制器的输入输出信号都按要求进行连接。具体有以下几项：

（1）将输入 Start1、Stop1、Store1、Reset1、CLK、NewRecord 都连接到码表控制器上；

（2）将时钟 CLK 连接到 16 位并行数据寄存器 SD 和计时器 TM 上；

（3）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；

（4）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；

（5）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；

（6）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。

接着要构建数据通路。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。由 16 位寄存器输出的数据需要送给最好成绩记录以及比较器，并与当前的计时记录进行比较，以输出新的最好成绩记录信号 NewRecord。

综合以上全部分析，可以连接出电路图如图 1-25 所示。

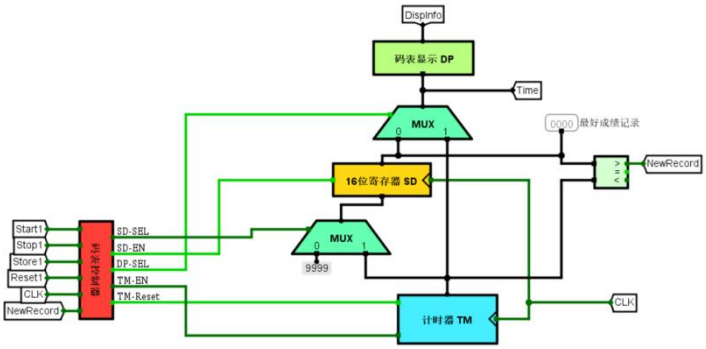


图 1-25：运动码表数据通路（系统集成）电路图

1.3 实验步骤

（1）基本电路模块构建与测试

根据 1.2 中的设计方案构建各个电路部件，包括数码管驱动电路、选择器、比较器、寄存器、BCD 计数器、码表计数器、码表显示驱动、码表控制器。

电路构建完成后，在 logisim 的测试电路下运行测试。首先运行“2 路选择器自动测试”电路，成功通过，显示“PASS”字样，如图 1-26 所示。

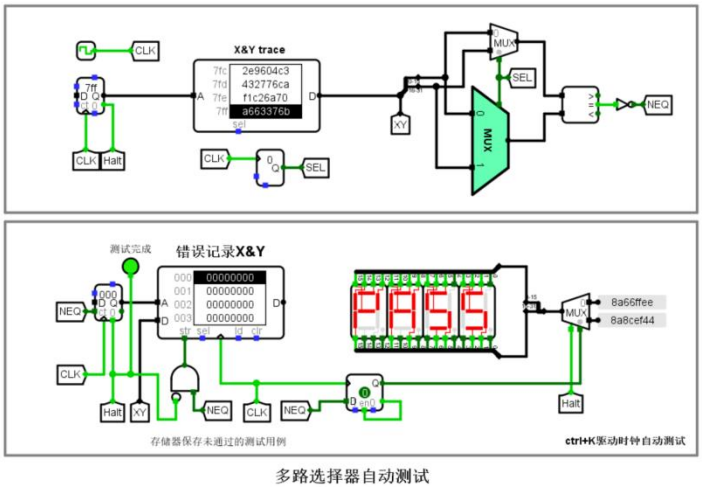


图 1-26：2 路选择器自动测试

然后运行“16 位无符号比较器自动测试”电路，成功通过，显示“PASS”字样，如图 1-24 所示。

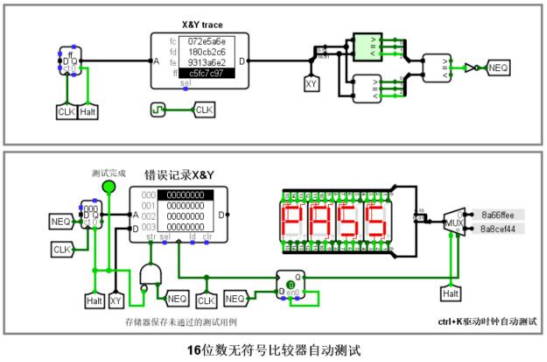


图 1-27：16 位无符号比较器自动测试

接着运行“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。

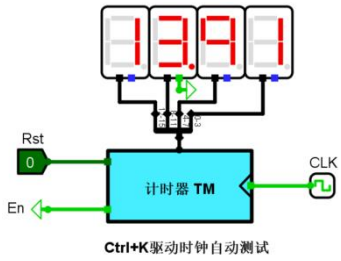


图 1-28：码表计数器自动测试

（2）运动码表功能测试

对所有模块整合后，打开“★运动码表”电路，构建测试集并手动运行测试。

首先按 Start 按钮，码表开始正常计时，如图 1-29 所示。

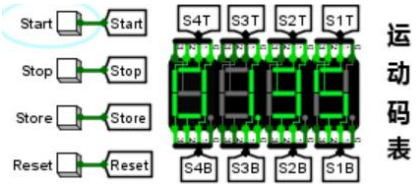


图 1-29：运动码表 Start 功能测试（开始计时）

接下来按 Stop 按钮，可以看到码表的计时记录维持不变，为停止状态，显示为“02.86”。按 Store 按钮保存本次记录为最好时间记录，可以看到数据通路电路中“最好成绩记录”输出由 9999 变为本次时间记录 286（单位：毫秒），如图 1-30 所示。

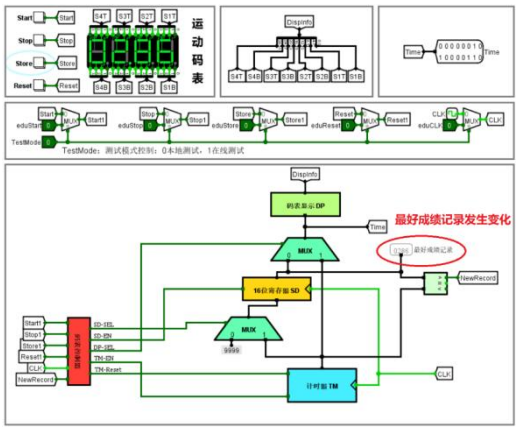


图 1-30：运动码表 Stop、Store 功能测试（停止计时和存储）

最后按 Reset 按钮，码表复位，显示为“00.00”，最好成绩记录变为 9999，如图 1-31、图 1-32 所示。

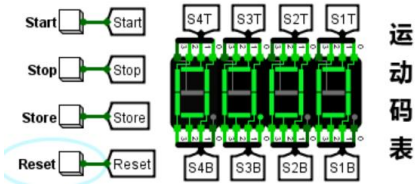


图 1-31：运动码表 Reset 功能测试（复位后码表显示）

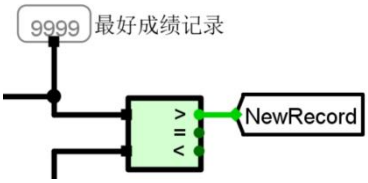


图 1-32：运动码表 Reset 功能测试（复位后最好成绩记录变为 9999）

1.4 故障与调试

1.4.1 引脚悬空时的信号传输问题

故障现象：在复杂电路中因为连线问题导致某些引脚没有接上（悬空），导致对应的引脚在输出结果时输出的是“xxxx”，如图 1-33 所示。

图 1-33：引脚悬空时信号无法正常传输

原因分析：如图 1-34 所示，在处理输入 Q 时利用分线器将其分为 4 位，分别为 Q1~Q4，但在输出结果时没有把 Q1~Q4 按位送给 Q，导致 Q 引脚悬空，无法读取每一位的值。这是由于在分线器分开后没有再次汇合，属于设计并行寄存器时的失误。

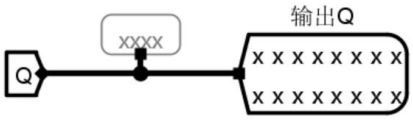


图 1-34：引脚悬空

解决方案：再引入一个分线器，把 Q1~Q4 按位送给 Q 即可解决引脚悬空的问题。

1.4.2 对时钟脉冲信号非法处理的问题

故障现象：在设计 4 位 BCD 计数器时，刚开始随着时钟脉冲的产生，评测结果正确，但过了一段时间后，后面的输出 Q 就开始出现错误结果，如图 1-35 所示。

图 1-35：对时钟脉冲信号非法处理后的评测结果

原因分析：如图 1-36 所示，在处理每个 D 触发器的使能端 En 时，简单地认为当且仅当 En 为高电平时才需要把时钟脉冲信号 CLK 输入每个 D 触发器的时钟端口，否则只需往时钟端口输入 0 即可。后来经过老师指导后发现不允许对时钟信号做任何操作，只能直接连接到触发器的时钟端口上。

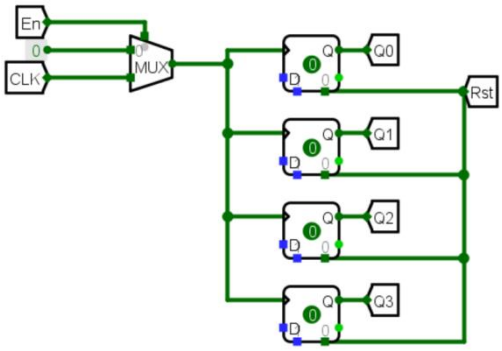


图 1-36：对时钟脉冲信号的非法处理（见左上角）

解决方案：只需要对 D 触发器增加一个单独的使能端 En，并将时钟和使能输入分别连接 4 个 D 触发器相应的输入端即可。

1.5 测试与分析

运动码表的测试与分析已经在上面 1.3 节完成，此处不再赘述。最终在头歌在线平台提交测评，成功通过了 16 关，获得满分。

1.6 实验总结

1） 本次实验中，我利用了 logisim 平台中现有运算部件构建一个运动码表系统。总体方案上，首先设计了局部的模块，包括 7 段数码管驱动电路、2 选 1 选择器（1 位或 16 位）、无符号比较器设计（4 位或 16 位）、并行加载寄存器（4 位或 16 位）、BCD 计数器状态机及输出函数、BCD 计数器（1 位十进制）、码表计数器（4 位十进制）、码表显示驱动、码表控制器状态机及输出函数、码表控制器、码表数据通路的设计。最后，对上述模块进行集成。

2） 本次实验的运动码表系统最终实现了开始计时、停止计时、存储时间、复位、保存或显示最短时间等功能。

1.7 实验心得

在本次实验中，我通过对码表各个基本模块的构建以及对运动码表数据通路的设计，最终完成了一个完整的运动码表系统。在这个过程中，我深刻理解了数字电路设计的基本原理和方法，包括时序逻辑、组合逻辑、状态机设计、时钟信号的使用等。通过实践，我更加熟悉了各个基本模块的功能和设计方法，如寄存器的并行加载、计数器的状态转换、数码管的驱动电路等。同时，我也学习到了如何将这些基本模块有机地结合起来，构建出一个完整的系统。在这个过程中，我不断调试和优化电路，解决各种问题和故障，提高了自己的实践能力和解决问题的能力。我主要遇到的问题是引脚悬空和对时钟脉冲的非法处理两种情形，不过经过仔细检查后都得以解决。通过

这次实验，我不仅掌握了数字逻辑电路设计的基本技能，还培养了自己的创新意识和实践能力，为今后的学习打下了坚实的基础。

2 CPU 设计实验

2.1 设计要求

本项目旨在运用Logisim平台内置的运算组件，设计并实现一款32位的MIPS CPU处理器。该处理器需要涵盖三种不同的实现方案：单周期硬布线控制器 CPU、多周期微程序控制器 CPU 以及多周期硬布线控制器 CPU。所构建的 CPU 必须能够全面支持表 2-1 中所罗列的全部指令，具体指令的功能定义与编码格式需参考课程提供的 MIPS 基准指令集手册。衡量设计成功与否的标准是，最终完成的 CPU 能够无误地执行指定的冒泡排序测试程序（sort.hex），并且可以在内存地址为 80 的位置观测到数据按降序排列的正确排序结果。

通过查阅 MIPS 基准指令集手册，我们能够确定上文所述 8 条指令的详细编码信息，并将其整理如表 2-2 所示，以供后续实验使用。具体来说，我们提取了所有指令操作码（OP，指令码前 6 位）以及 R 型指令的功能码（Func，指令码后 6 位）。分析表 2-2 可知，在这些指令中，仅 ADD、SLT 和 SYSCALL 三条指令被归类为 R 型指令。它们的共同特征是 OP 字段为 0，而具体的操作类型则依赖于 Func 字段来区分。其余指令的操作类型直接由其各自的 OP 字段决定。此外，指令码的第 21-25 位、16-20 位和 11-15 位分别用于指定源寄存器 rs、源寄存器 rt 和目标寄存器 rd。

表 2-1：MIPS CPU 支持的指令

编号	MIPS 指令	功能描述	备注
1	add $rd, $rs, $rt	R[$rd]←R[$rs] + R[$rt]	溢出产生异常，且不修改 R[$rd]
2	slt $rd, $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] + SignExt(imm)	溢出产生异常
4	lw $rt, imm($rs)	R[$rt]←Mem(R[$rs] + SignExt(imm))	无
5	sw $rt, imm($rs)	Mem(R[$rs] + SignExt(imm))←R[$rt]	无

6	beq $rs, $rt, imm	if(R[$rs] = R[$rt])　PC←PC + SignExt({imm, 00})	无
7	bne $rs, $rt, imm	if(R[$rs] != R[$rt])　PC←PC + SignExt({imm, 00})	无
8	syscall	系统调用，这里用于停机	无

表 2-2：MIPS CPU 支持的指令的指令码 OP 及 Func

编号	MIPS 指令	OP（前 6 位）	Func（后 5 位）
1	add $rd, $rs, $rt	000000	100000
2	slt $rd, $rs, $rt	000000	101010
3	addi $rt, $rs, imm	001000	非 R 型指令
4	lw $rt, imm($rs)	100011	非 R 型指令
5	sw $rt, imm($rs)	101011	非 R 型指令
6	beq $rs, $rt, imm	000100	非 R 型指令
7	bne $rs, $rt, imm	000101	非 R 型指令
8	syscall	000000	001100

2.2 方案设计

2.2.1 单周期 MIPS CPU（硬布线）及单周期硬布线控制器设计

本节将设计一个 32 位的单周期 MIPS 处理器。该处理器的构建整合了先前在运算器实验和存储系统实验中已完成的运算器、寄存器文件和存储器等模块，并结合了 Logisim 平台提供的其他功能组件。处理器的整体数据通路结构如图 2-1 所示。

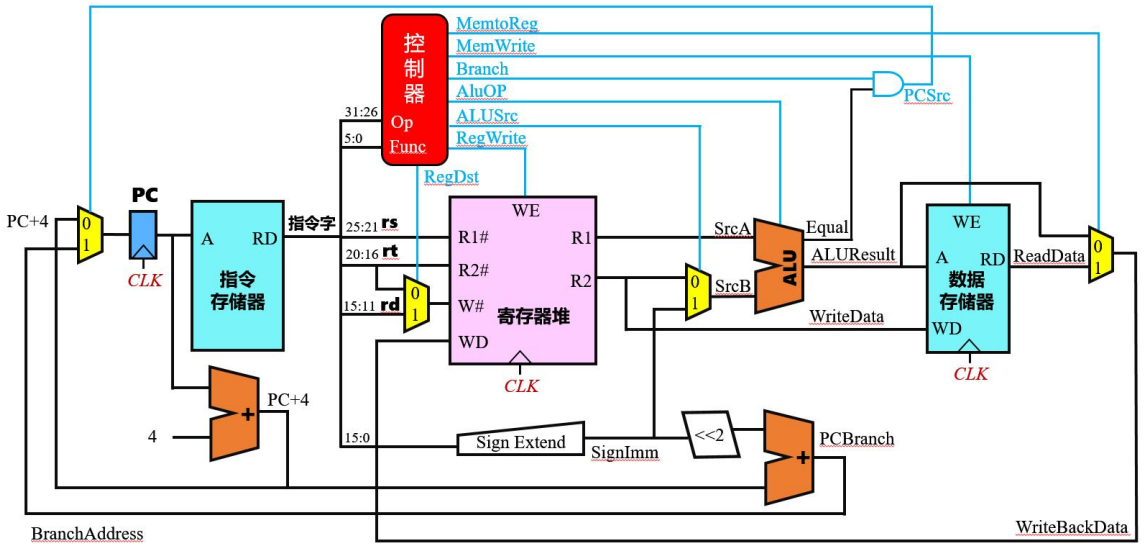


图 2-1：单周期 MIPS CPU 处理器数据通路

电路引脚及功能如表 2-3 所示。

表 2-3：单周期 MIPS CPU 处理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	时钟信号
PC	输出	32	程序寄存器的值
IR	输出	32	当前指令字
RegWrite	输出	1	寄存器文件写使能控制信号
RDin	输出	32	寄存器文件写入端口的数据
MemWrite	输出	1	存储器写使能控制信号
MDin	输出	32	存储器写入端口的数据

该处理器的整体结构由以下几个核心功能模块组成：

(1) PC (指令计数器): 其作用是保存下一条待执行指令的内存地址。

(2) 指令存储器 (Instruction ROM): 用于存放 CPU 将要执行的指令序列。

(3) 数据存储器 (Data RAM): 负责存取指令在执行期间所需要读写的数据。

(4) 立即数符号扩展器 (S-EXT): 将指令中的 16 位立即数转换为 32 位。由于该立即数可能为负值，因此必须采用符号位扩展的方式。

(5) 控制器: 作为 CPU 的指挥中心，负责生成数据通路所需的控制信号。它的工

作原理是：首先对指令的 OP 和 Func 字段进行译码，根据译码结果确定每条指令对应的控制信号值，进而推导出各控制信号的布尔逻辑表达式，并最终实现控制逻辑。此模块中也包含了 ALU 控制器，用以处理 ALU_OP 信号与指令 OP、Func 字段之间的映射关系。

(6) 寄存器文件 (RegFile): 包含了 32 个 MIPS 架构的通用寄存器。 (7) ALU (算术逻辑单元): 执行算术和逻辑运算，在此设计中主要提供加法与减法功能，并输出计算结果。

在单周期硬布线控制器设计完成后，只需参照图 2-1 的数据通路图，将上述各个部件正确连接即可。具体的操作细节在实验步骤中有详细说明。

2.2.2 多周期 MIPS CPU（微程序）及多周期微程序控制器设计

本部分的目标是构建一个 32 位的多周期 MIPS CPU 处理器。我们将采用微程序控制器的设计方案来实现其控制器，并参照多周期 MIPS 处理器的数据通路图进行搭建。具体的处理器数据通路如图 2-2 所示。

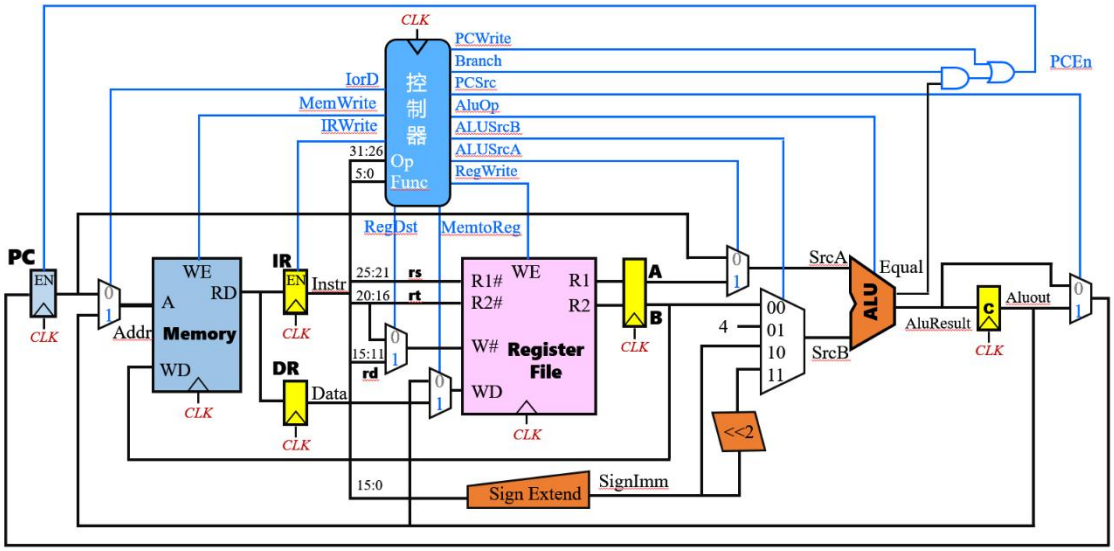


图 2-2：多周期 MIPS CPU 处理器（微程序）数据通路

在此设计中，电路的引脚定义与功能和表 2-3 中所列的保持完全一致。其整体结构由以下核心模块构成：

(1) PC (指令计数器): 负责记录当前指令的存放地址。

(2) 存储器 (Mem): 采用 RAM 实现，该模块统一作为指令存储器和数据存储器使

用，实现了二者的功能合并。

(3) IR (指令寄存器): 用于暂存从存储器中取出、正待 CPU 执行的指令。

(4) DR (数据寄存器): 用于暂存指令执行期间需要用到的数据。

(5) S-EXT (立即数扩展器): 通过符号扩展的方式，将指令中的 16 位立即数转换为 32 位数值。

(6) 控制器: 负责生成数据通路的控制信号。它根据指令译码结果输出相应的控制逻辑，并定义 ALU_OP 与指令 OP、Func 字段之间的逻辑关系。

(7) 寄存器文件 (RegFile): 包含了 MIPS 架构所需的 32 个通用寄存器。

(8) ALU (算术逻辑单元): 执行加法或减法等算术运算，并输出其结果。

(9) 临时寄存器 (A, B, C): 三个独立的寄存器，分别用于临时存放来自寄存器文件(RegFile)的两个输出（R1, R2）以及 ALU 的运算结果。

在完成多周期微程序控制器的设计后，便可以依据图 2-2 所示的数据通路图将上述所有部件连接起来。具体的操作流程在实验步骤中有详细阐述。

2.3 实验步骤

2.3.1 单周期硬布线 MIPS CPU 实现（指令译码、控制信号输出和 ALU 控制器逻辑）

设计的首要任务是完成单周期硬布线控制器。我们利用先前已列出的 8 条指令的操作码（OP）和功能码（Func），将指令译码逻辑的设计分为两个主要步骤：

(1) 首先，依据指令的 OP 字段是否为 0，将其划分为 R 型指令与非 R 型指令两大类别。

(2) 接着，对这两类指令分别处理： * ① 对于非 R 型指令，可以直接根据其 OP 码来生成对应的指令译码信号。 * ② 对于 R 型指令，则需根据其 Func 字段的具体值来确定译码信号，并且需要将一个名为 R_TYPE 的标记信号置为 1。

该逻辑的电路实现如图 2-3 所示。

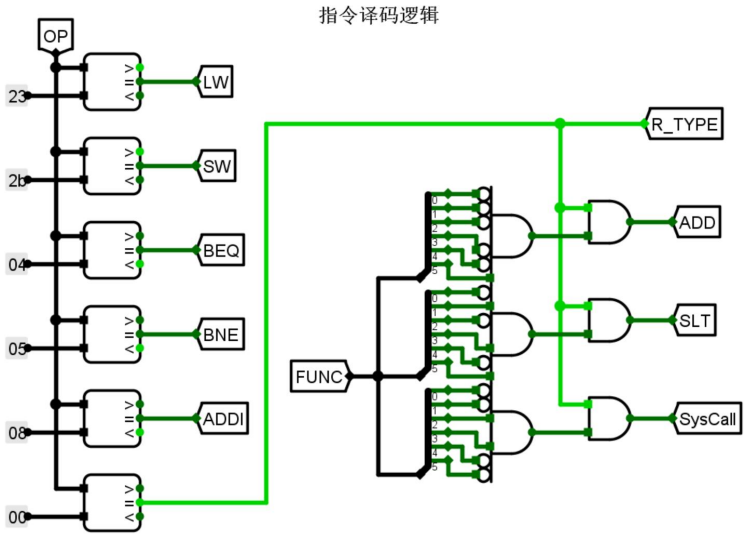


图 2-3：指令译码逻辑模块电路图

接下来，我们需要根据指令译码信号来构建控制器输出控制信号的逻辑。此过程首先要求我们明确数据通路中各组件的功能及其输入数据的来源。然后，依据每个组件输入来源的种类数量和选择条件，便可以推导出对应控制信号的逻辑表达式。

(1) 数据通路设计: 我们参照图 2-1 并结合需要支持的 8 条指令的功能特性，来确定主要功能模块的输入源，并在此基础上完成 CPU 数据通路的设计。

具体到每个功能部件，我们需要统计其各个输入端口的来源种类数。如果某个端口的输入来源只有一种，那么在构建控制逻辑时，直接进行连接即可。反之，若输入来源多于一种，则必须引入多路选择器（Mux）来决定当前有效的信号源。具体情况在表 2-4 中进行了总结。 从表中可以清晰地看到，IM、R1#、R2#、ALU.A、S-EXT、DM.A 以及 DM.Din 这些端口的输入源都只有一个，因此它们不需要配置任何选择器。

(2) 控制信号逻辑表达式推导: 对于余下的几个拥有多个输入源的端口（即 PC、W#、Reg.Din、ALU.B），我们需要分析在执行不同指令时，它们各自的数据选择条件以及由此产生的控制信号，具体分析如表 2-5 所示。最终，要获得某个控制信号的逻辑表达式，只需将所有能够使其有效（即产生该信号）的条件进行“逻辑或”运算即可。

表 2-4：CPU 数据通路设计

编号	MIPS 指令	PC	IM	RegFile				ALU		S-EXT	DM	
				R1#	R2#	W#	Din	A	B		A	Din

1	add	PC + 4	PC	rs	rt	rd	ALU	RD1	RD2	无	无	无
2	slt	PC + 4	PC	rs	rt	rd	ALU	无	无	无	无	无
3	addi	PC + 4	PC	rs	无	rt	ALU	RD1	S-EXT	i[15:0]	无	无
4	lw	PC + 4	PC	rs	无	rt	DM.Dout	RD1	S-EXT	i[15:0]	ALU	无
5	sw	PC + 4	PC	rs	rt	无	无	RD1	S-EXT	i[15:0]	ALU	RD2
6	beq	PC + 4 + offset	PC	rs	rt	无	无	RD1	RD2	i[15:0]	无	无
7	bne	PC + 4 + offset	PC	rs	rt	无	无	RD1	RD2	i[15:0]	无	无
8	syscall	PC	PC	无	无	无	无	无	无	无	无	无
来源种数		2	1	1	1	2	2	1	2	1	1	1

表 2-5：单周期硬布线控制器控制信号表

指令/多路选择器控制信号	**RegDst**	**RegWrite**	**MemToReg**	**MemWrite**	**AluSrc**	**Beq**	**Bne**	**Halt**
add	1	1	0	0	0	0	0	0
slt	1	1	0	0	0	0	0	0
addi	0	1	0	0	1	0	0	0
lw	0	1	1	0	1	0	0	0
sw	0	0	0	1	1	0	0	0
beq	0	0	0	0	0	1	0	0
bne	0	0	0	0	0	0	1	0
syscall	0	0	0	0	0	0	0	1

下面我们对表 2-5 中除 Beq 和 Bne 指令外的所有控制信号，在其值为 1 时的作用进行阐释，具体如下：

① RegDst = 1: 表明写入寄存器堆（RegFile）的目标地址（W#）由指令的 rd 字段指定；若为 0，则目标地址由 rt 字段指定。

② ② RegWrite = 1: 激活寄存器堆的写使能信号，允许将数据总线（WD）上的

数据写入到 W#所指向的寄存器中。

③ ③ MemToReg = 1: 表示写回寄存器堆的数据来源是存储器（Memory）的输出，而非 ALU 的计算结果。

④ ④ MemWrite = 1: 激活对存储器的写操作，允许将数据（此处指 ALU 的运算结果）写入内存。

⑤ ⑤ AluSrc = 1: 用于选择 ALU 的 B 操作数。当此信号为 1 时，B 操作数来自立即数扩展器（S-EXT）的输出；若为 0，则来自寄存器堆的读端口 2（R2）的输出

⑥ ⑥ Halt = 1: 这是一个停机控制信号，当其有效（为 1）时，处理器将停止运行。

⑦ 综合以上逻辑，便可构建出由指令译码信号生成控制器输出信号的完整电路，其最终实现如图 2-4 所示。

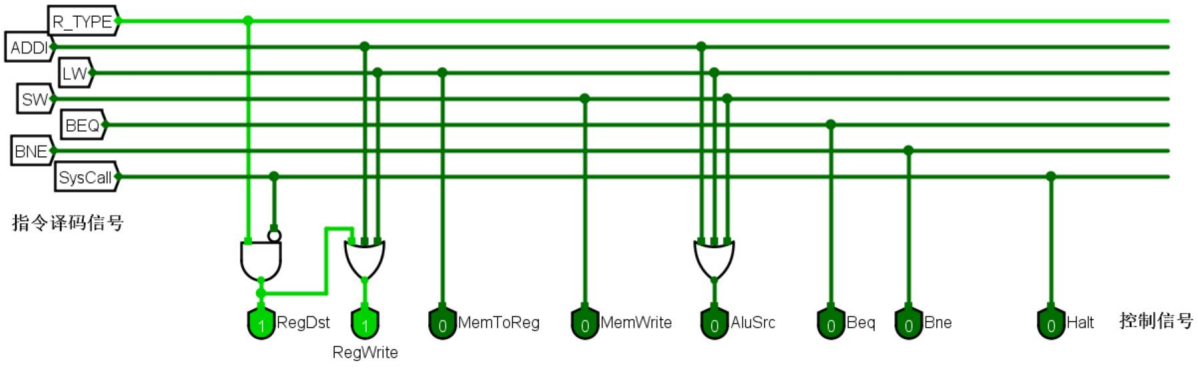


图 2-4：根据指令译码信号实现控制器输出控制信号电路图

在 ALU 控制器逻辑模块的设计中，我们首先确定 ALU_OP 信号只有两种可能的取值：5（代表加法）和 11（代表减法）。其具体取值依赖于 Func 字段：当 Func 字段的值为 2a（十六进制）时，表明当前指令是 ADD，此时应将 ALU_OP 设为 5。在所有其他情况下，ALU_OP 均设为 11，以执行减法操作。基于此逻辑，我们构建了 ALU 控制器，其电路图如图 2-5 所示。

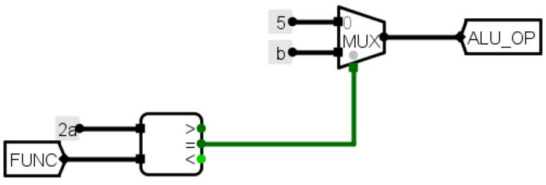


图 2-5：ALU 控制器逻辑（单周期硬布线）电路图

设计的最后一步是根据图 2-1 的连接方式，完成单周期硬布线 MIPS CPU 的整体电

路组装。此外，为了能够统计 sort.hex 测试程序的总执行周期，我们增加了一个计数器电路：将 Halt 停机信号进行反相操作，然后将其输出作为计数器的时钟输入，具体连接方式如图 2-6 所示。

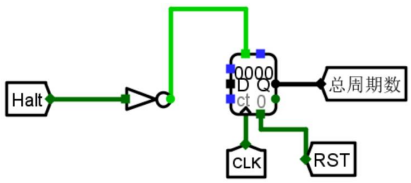


图 2-6：单周期硬布线 MIPS CPU 指令计数器电路图

2.3.2 多周期微程序 MIPS CPU 实现（ALU 控制器逻辑、微程序地址转移逻辑）

现在我们转向多周期 CPU 的设计。该设计在架构上与单周期模型相比，存在几项根本性的不同：

(1) 首先，多周期 CPU 采用统一的存储器模型，将获取指令和读写数据的功能整合到单个 RAM 模块中。同时，为了在不同时钟周期之间传递状态，像 ALU 和存储器这样的主要单元之后都配备了缓冲寄存器来锁存结果。

(2) 其次，硬件资源得到了更高效的利用。例如 ALU 和寄存器堆（RegFile），它们可以在同一条指令的不同执行阶段被重复调用。

(3) 再次，程序计数器（PC）的控制逻辑发生了变化。它不再是每个时钟周期都固定递增，以此来适应不同指令可变的执行周期长度。

(4) 最后，ALU 的功能也进行了扩展。其输出现在有三种可能：为 beq 或 bne 指令计算分支目标地址，为算术指令提供运算结果，或为 lw/sw 指令生成访存地址。这一变化直接影响了 ALU 控制器的设计，即 ALU_OP 信号的选择逻辑，其具体实现如图 2-7 所示。

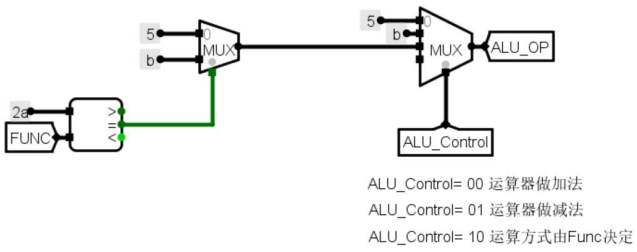


图 2-7：ALU 控制器逻辑（多周期微程序）电路图

我们根据表 2-1 中定义的各条指令功能，并结合课堂上讲授的多周期数据通路执

行流程，将所有指令共有的取指（Fetch）阶段以及 8 条指令各自的执行操作，详细地划分到不同的机器周期中。具体的分解结果汇总于表 2-6。

表 2-6：取值阶段与各指令在不同机器周期内的数据通路

指令	阶段	数据通路	状态	地址
取值阶段	取指令	IR←(M[PC]), PC←(PC)+4	S0	0
	译码并取操作数	A←(R[I[25:21]]), B←(R[I[20:16]]), C←(PC)+(S-EXT(I[15:0])<<2)	S1	1
ADD	加运算	C←(A)+(B)	S2	2
	写回	R[I[15:11]]←(C)	S3	3
SLT	操作数比较	C←((A)<(B))	S2	2
	写回	R[I[15:11]]←(C)	S3	3
ADDI	加运算	C←(A)+S-EXT(I[15:0])	S4	4
	写回	R[I[20:16]]←(C)	S5	5
LW	计算地址	C←(A)+S-EXT(I[15:0])	S6	6
	仿存	DR←(M[PC])	S7	7
	写回	R[I[20:16]]←(DR)	S8	8
SW	计算地址	C←(A)+S-EXT(I[15:0])	S9	9
	仿存	DR←(M[PC])	S10	10
BEQ	送目标地址	IF(A==B)　PC←(C)	S11	11
BNE	送目标地址	IF(A!=B)　PC←(C)	S12	12
SYSCALL	停机	--	S13	13

上表中的每一行都代表一条独立的微指令，我们为它们分别指定了从 S0 到 S13 的状态编码，并将这些编码补充标注回原表中。在控制逻辑上，我们采用了“下址字段法”与一个“顺序控制位 P”相结合的模式。该控制位 P 的作用是：当 P 为 1 时，下一条微指令的地址由指令执行的初始阶段（第一阶段）直接给出；当 P 为 0 时，下一条微指令的地址则由当前微指令的“下址字段”来指定。通过这种方式，我们便可以最终确定各条指令在不同执行阶段所对应的全部控制信号，其结果如图 2-8 所示。

微指令功能	状态	微指令地址	IorD	PcSrc	AluSrcA	AluSrcB	MemToReg	RegDst	IrWrite	PcWrite	RegWrite	MemWrite	MemRead	BEQ	BNE	AluControl	P	下址字段
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00	0	0001
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000
LW1	2	0010	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00	0	0100
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00	0	0000
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00	0	0000
R1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10	0	1000
R2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	00	0	0000
Beq	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	01	0	0000
Bne	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	01	0	0000
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	1100
ADDI2	12	1100	0	0	0	00	0	0	0	0	1	0	0	0	0	00	0	0000
SYSCALL	13	1101	0	0	0	00	0	0	0	0	0	0	0	0	0	00	0	1101

图 2-8：多周期微程序 MIPS CPU 指令控制信号表

图中各控制信号的说明如下：

(1) IorD: 内存访问选择信号，1 为数据访问，0 为指令访问。

(2) PcSrc: PC 更新方式选择，1 为跳转，0 为顺序执行。

(3) AluSrcA: ALU 的 A 操作数来源选择（PC 或寄存器)。

(4) AluSrcB: ALU 的 B 操作数来源选择，根据指令类型可以是寄存器（R 型）、立即数（sw/lw/addu）或偏移地址（bne/beq）。

(5) MemToReg: lw 指令专用的数据来源控制，当有效时，从内存加载数据到寄存器。

(6) RegDst: 指定要写入的寄存器堆目标地址（W#）。

(7) IrWrite/PcWrite/RegWrite: 三个分别是指令寄存器、PC 和寄存器堆的写控制信号。

(8) MemWrite/MemRead: 内存的写/读使能信号。

(9) BEQ/BNE: 分别对应 BEQ 和 BNE 指令的译码标志。 (10) AluControl: 决定 ALU 执行加法或减法运算。

利用以上这些状态信息和状态间的转移关系，下一步便是构建微指令的地址转移逻辑。图 2-8 为我们提供了指令的入口地址信息，在此基础上设计的完整地址转移方案，最终体现在图 2-9 的逻辑图中。

机器指令译码信号										微程序入口地址					
R_Type	ADDI	LW	SW	BEQ	BNE	SYSCALL	xxx1	xxx2	xxx3	入口地址10进制	S4	S3	S2	S1	S0
1										7	0	0	1	1	1
	1									11	0	1	0	1	1
		1								2	0	0	0	1	0
			1							5	0	0	1	0	1
				1						9	0	1	0	0	1
					1					10	0	1	0	1	0
						1				13	0	1	1	0	1

图 2-9：多周期微程序 MIPS CPU 微指令地址转移逻辑

接下来，在 Excel 表格中“地址逻辑自动生成”下自动生成地址转移逻辑表达式。具体表达式为：$S3 = ADDI + BEQ + BNE + SYSCALL$，$S2 = R\_Type + SW + SYSCALL$，$S1 = R\_Type + ADDI + LW + BNE$，$S0 = R\_Type + ADDI + SW + BEQ + SYSCALL$。

至此可以生成控制存储器所需的微指令并用 16 进制表示。在“微指令自动生成.xlsx”中填写每条微指令的下址字段和 P 后即可自动生成程序的微指令，如图 2-10 所示。

微指令	十六进制
00001001100100000001	13201
00011000000000010000	30010
00110000000000000011	60003
10000000001000000100	100204
00001000100000000000	8800
00110000000000000110	60006
10000000010000000000	100400
00100000000001001000	40048
00000010010000000000	4800
01100000000100100000	C0120
01100000000010100000	C00A0
00110000000000001100	6000C
00000000100000000000	800
00000000000000001101	D

图 2-10：多周期微程序 MIPS CPU 微指令编码

设计的最后一步，是依据图 2-2 的连接图，将多周期微程序 MIPS CPU 的各个模块组装起来。此外，为了方便地统计 sort.hex 程序的总执行周期数，我们设计了一个计数电路：将当前正在执行的微指令地址接入一个计数器，并设定当微指令地址为 d 时（此地址对应 syscall 指令），计数过程便结束。该计数逻辑的实现如图 2-11 所示。

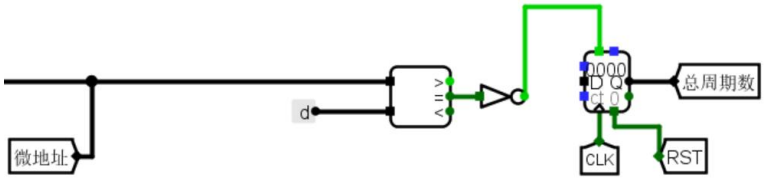


图 2-11：单周期硬布线 MIPS CPU 指令计数器电路图

2.4 故障与调试

2.4.1 指令加载失败问题

故障现象：评测时一开始的 IR 为 0，没有正确加载指令，如图 2-12 所示。

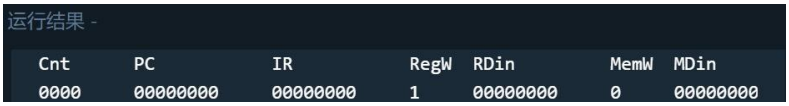


图 2-12：指令加载失败

原因分析：在构建指令存储器时使用了 RAM，导致本地测试正常，但线上评测环境不支持。

解决方案：将 RAM（随机存储器）更换为 ROM（只读存储器）即可。

2.4.2 数据地址不正确

故障现象：在单周期硬布线 MIPS CPU 下，冒泡排序 sort.hex 程序执行完成后，没有在预期的第 80 号存储单元形成降序排序的数据，而在第 200 号到第 21c 号存储单元之间可以观察到降序排序的数据，且相邻数据地址相差 4。如图 2-13 所示。

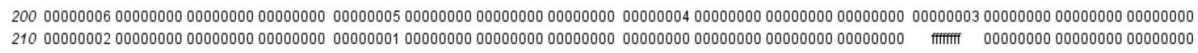


图 2-13：降序排序数据地址不正确

原因分析：对于数据存储器所用的 RAM，只支持一次访问读出 32 位数据，直接给出字地址导致内存分布异常。

解决方案：将字地址除以 4，相当于一次访问读出 8 位数据且相邻，即可解决这个问题。实验中的操作是使用分线器取出字地址的低 6 位，并将高 2 位置为 0 作为字节地址，然后送入 RAM 存储器的地址输入，就能在第 80 号存储单元处得到正确排序的数据结果，如图 2-14 所示。

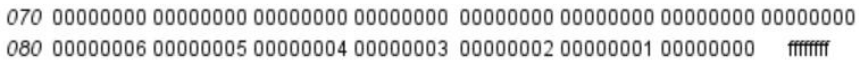


图 2-14：将地址修正后的降序排序数据

2.5 测试与分析

2.5.1 单周期硬布线 MIPS CPU 功能测试

实验操作的第一步是将冒泡排序程序 sort.hex 加载至指令存储器，这一过程通过“Load Image”功能完成，具体操作可参见图 2-15。加载完成后，我们按下 Ctrl+K 以启动时钟，让程序开始执行。在执行过程中，我们监测到最终消耗的总周期为 224 个，这一数字与给定的参考结果完全吻合，验证情况如图 2-16 所示。同时，程序执行

结束时数据存储器中的内容被记录下来，并展示于图 2-14。

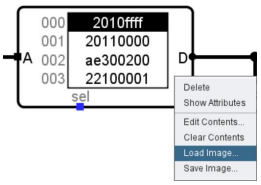


图 2-15：加载冒泡排序程序

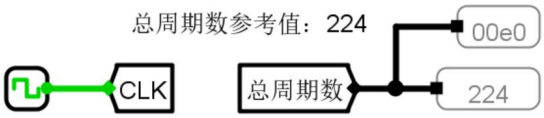


图 2-16：单周期硬布线 CPU 冒泡排序程序总周期数

2.5.2 多周期微程序 MIPS CPU 功能测试

基本步骤与上面一致，这里还需要在控制存储器中载入提前准备好的微指令编码。最终总周期数为 891，与参考结果一致，如图 2-17 所示。数据存储器的内容如图 2-14 所示。

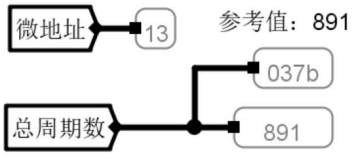


图 2-17：多周期微程序 CPU 冒泡排序程序总周期数

2.6 实验总结

本次实验围绕 32 位 MIPS 处理器的设计与实现展开，主要完成了以下工作：

首先，我们构建了完整的 CPU 数据通路，并针对单周期与多周期两种不同模型的特点，对通路细节进行了完善。这包括实现了数据通路中必要的寄存器读写、内存访问、ALU 运算、立即数处理、指令解析以及通过多路选择器进行数据路由等功能。

其次，在上述数据通路的基础上，我们成功实现了两种控制器：一种是单周期硬布线控制器，重点在于其指令译码和控制信号的组合逻辑实现；另一种是多周期微程序控制器，我们为其设计了微指令格式、状态编码以及核心的微程序地址转移逻辑。

最终，我们将这些模块整合，成功构建出可运行的 32 位 MIPS 处理器（含单周期和多周期两种版本），它能够支持包含 add、slt 等在内的 8 条指令，并能正确执行给

定的冒泡排序测试程序，达到了预期的设计目标。

2.7 实验心得

1） 对 logisim 的使用进一步熟练，如利用表达式一键生成组合逻辑电路等功能，使设计电路的效率大大提升。

2） 通过本实验，我对 CPU 的设计以及中断机制的实现有了一个更加深刻的认识，对课本上面的内容学习的更加深刻，更加熟练，通过实验，我巩固了支持中断的现代时序的状态机的实现，对微程序的各个控制信号的使用更加熟练，明白了 CPU 是如何通过实现各个部件的功能从而组合在一起实现 CPU 的功能的，明白了 CPU 的架构，同时通过实现这五条指令，我对 MIPS 指令的运行有了更加深刻的认识。

3） 通过本次实验，加深了对数据通路、总线和控制字段的理解，在实验过程中我也更加锻炼了自己发现问题解决问题的能力，遇到问题后可以冷静的分析问题可能存在的地方，并能够快速的寻找解决方案，在实验过程中我也更加细心，在设计过程中，一个很小的错误就可能导致错误的结果，这又会耗费很多的精力寻找问题所在，所以在设计过程中细心是很重要的。

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：陈宇航

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： _____