

华中科技大学

2023

逻辑与计算机系统设计 · 实验报告 ·

专 业： 计算机科学与技术

班 级：

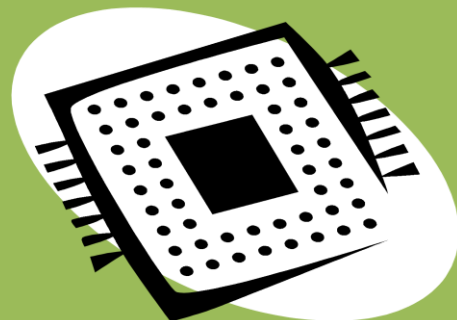
学 号：

姓 名： [作者]

电 话：

邮 件：

完成日期： 2023-06-26



计算机科学与技术学院

# 华中科技大学课程实验报告

---

1	运动码表系统设计 .....	4
1.1	设计要求 .....	4
1.2	方案设计 .....	5
1.2.1	运动码表的总体设计 .....	5
1.2.2	运动码表的模块说明 .....	6
1.3	实验步骤 .....	11
1.3.1	7 段数码管驱动电路 .....	11
1.3.2	2 路选择器（1 位） .....	13
1.3.3	2 路选择器（16 位） .....	14
1.3.4	4 位无符号比较器 .....	15
1.3.5	16 位无符号比较器 .....	16
1.3.6	4 位并行加载寄存器 .....	17
1.3.7	16 位并行加载寄存器 .....	18
1.3.8	BCD 计数器状态转换 .....	18
1.3.9	BCD 计数器输出函数 .....	20
1.3.10	4 位 BCD 计数器 .....	20
1.3.11	码表计数器 .....	21
1.3.12	码表显示驱动 .....	22
1.3.13	码表控制器状态转换 .....	23
1.3.14	码表控制器输出函数 .....	25
1.3.15	码表控制器 .....	26
1.3.16	运动码表 .....	27
1.4	测试与分析 .....	28
1.4.1	7 段数码管驱动电路测试 .....	28
1.4.2	2 路选择器（16 位）测试 .....	29
1.4.3	16 位无符号比较器测试 .....	29
1.4.4	码表计数器测试 .....	29
1.4.5	运动码表测试 .....	30

# 华中科技大学课程实验报告

---

2	CPU 设计实验 .....	31
2.1	设计要求 .....	31
2.2	方案设计 .....	32
2.2.1	单周期 CPU 功能部件 .....	32
2.2.2	多周期 CPU 功能部件 .....	33
2.2.3	地址转移逻辑 NPC .....	34
2.2.4	立即数扩展器 .....	34
2.2.5	操作控制器 .....	35
2.3	实验步骤 .....	35
2.3.1	指令解析 .....	35
2.3.2	实现地址转移逻辑 .....	37
2.3.3	单周期硬布线控制器 .....	37
2.3.4	单周期 CPU 总体结构图 .....	39
2.3.5	从单周期到多周期 .....	39
2.3.6	多周期控制信号 .....	40
2.3.7	多周期微程序控制器 .....	43
2.3.8	多周期硬布线控制器 .....	45
2.3.9	多周期 CPU 总体结构图 .....	46
2.4	故障与调试 .....	46
2.4.1	单周期数据存储器地址错误 .....	46
2.4.2	多周期 CPU 循环震荡 .....	47
2.5	测试与分析 .....	48
2.5.1	单周期 CPU 执行 sort.hex .....	48
2.5.2	多周期 CPU 执行 sort.hex .....	48
3	总结与心得 .....	50
3.1	实验总结 .....	50
3.1.1	运动码表实验总结 .....	50
3.1.2	CPU 实验总结 .....	50
3.2	实验心得 .....	51

# 华中科技大学课程实验报告

---

## 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 32 位运算器，可支持算数加、减、乘、除，逻辑与、或、非、异或运算、逻辑左移、逻辑右移，算术右移运算，支持常用程序状态标志（有符号溢出 OF、无符号溢出 CF，结果相等 Equal），运算器功能以及输入输出引脚见下表，在主电路中详细测试自己封装的运算器。

表 1-1 片引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零

表 1-2 运算符功能

ALU OP	十进制	运算功能	
0000	0	Result = X << Y	逻辑左移 (Y 取低五位) Result2=0
0001	1	Result = X >>> Y	逻辑右移 (Y 取低五位) Result2=0
0010	2	Result = X >> Y	算术右移 (Y 取低五位) Result2=0
0011	3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 有符号	
0100	4	Result = X/Y;	Result2 = X%Y 无符号
0101	5	Result = X + Y	Result2=0 (Set OF/CF)
0110	6	Result = X - Y	Result2=0 (Set OF/CF)
0111	7	Result = X & Y	Result2=0

# 华中科技大学课程实验报告

1000	8	Result = $X   Y$ Result2=0
1001	9	Result = $X \oplus Y$ Result2=0
1010	10	Result = $\sim(X   Y)$ Result2=0
1011	11	Result = $(X < Y) ? 1 : 0$ Signed Result2=0
1100	12	Result = $(X < Y) ? 1 : 0$ Unsigned Result2=0
1101	13	Result = Result2=0
1110	14	Result = Result2=0
1111	15	Result = Result2=0

## 1.2 方案设计

### 1.2.1 运动码表的总体设计

(一) 构成：7 位数码管驱动器，2 路选择器（16 位），16 位无符号比较器，16 位并行加载寄存器，4 位 BCD 计数器，码表计数器，码表显示驱动，码表控制器。

(二) 模块划分框图：

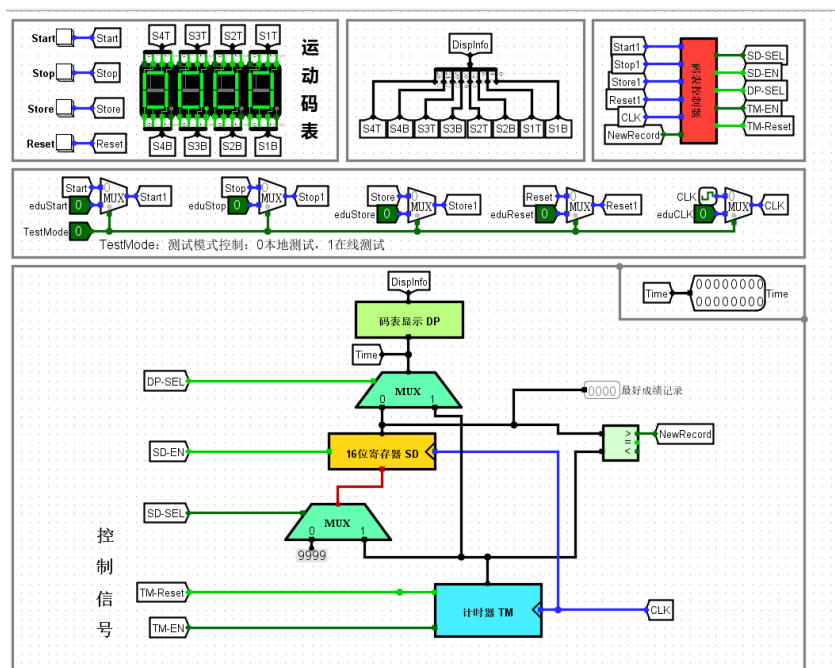


图 1-1

流程说明：首先按下 Start 键，系统开始计时，若按下 stop 键，系统停止计时，码

# 华中科技大学课程实验报告

表显示器显示当前时间；若按下 store 键，则 16 位寄存器会将当前时间记录下来，并与之前所记录的成绩作比较，若要优于之前的成绩，NewRecord 会显示高电平 1，则会将最好成绩替换。

## 1.2.2 运动码表的模块说明

### 1) 7 位数码管驱动电路：

由四个数码管显示驱动器组成

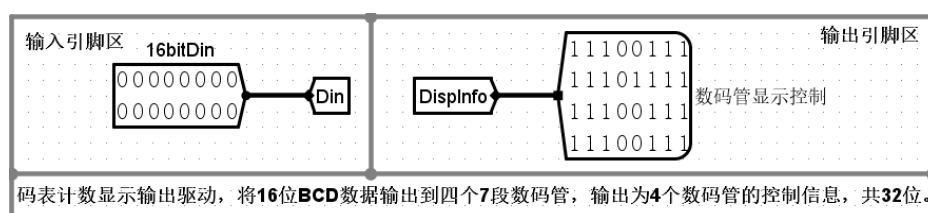


图 1-2

对于每个数码管驱动器，由真值表对应自动生成逻辑电路。真值表如下：

X3	X2	X1	X0	Seg_1	Seg_2	Seg_3	Seg_4	Seg_5	Seg_6
0	0	0	0	0	1	1	1	1	1
0	0	0	1	0	0	0	1	0	0
0	0	1	0	1	0	1	1	1	1
0	0	1	1	1	0	1	1	0	1
0	1	0	0	1	1	0	1	0	0
0	1	0	1	1	1	1	0	0	1
0	1	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0	0
1	0	0	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1
1	0	1	1	1	1	1	1	0	1
1	1	0	0	1	1	1	1	0	1
1	1	0	1	1	1	1	0	0	1
1	1	1	0	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	1

图 1-3

### 2) 16 位无符号比较器：

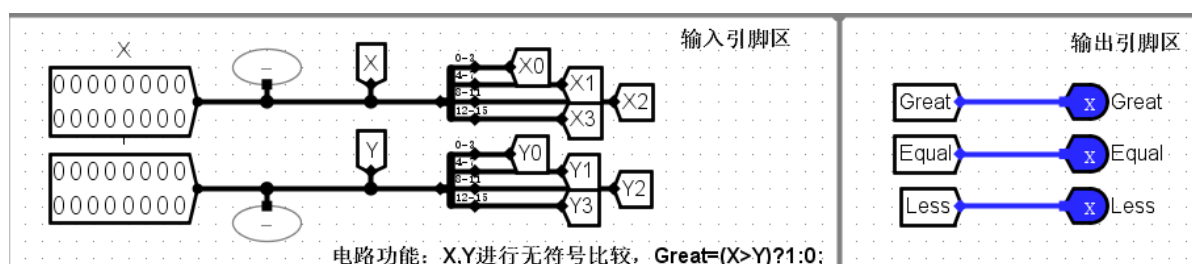


图 1-4

# 华中科技大学课程实验报告

由 4 个 4 位无符号比较器组成。

对于每个 4 位无符号比较器有对应逻辑表达式生成：



图 1-5

3) 2 路选择器 (16 位)：

由 16 个 2 路选择器 (1 位) 组成。

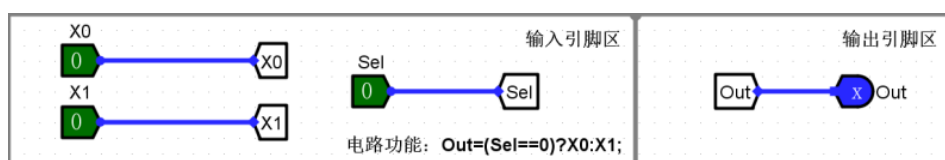


图 1-6

对于每个 2 路选择器：



X0	Sel	X1	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

图 1-7

4) 16 位并行加载寄存器:

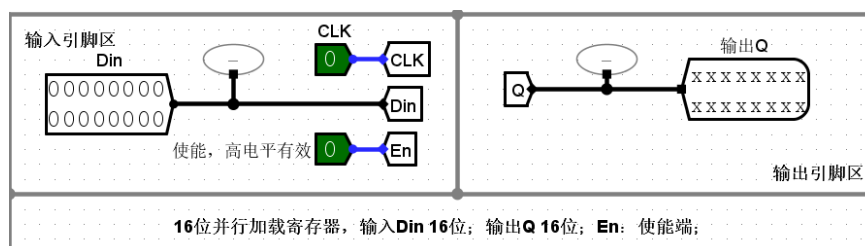


图 1-8

由 4 个 4 位并行加载寄存器构成;

对于每个并行加载寄存器: 由 4 个 D 触发器功能实现

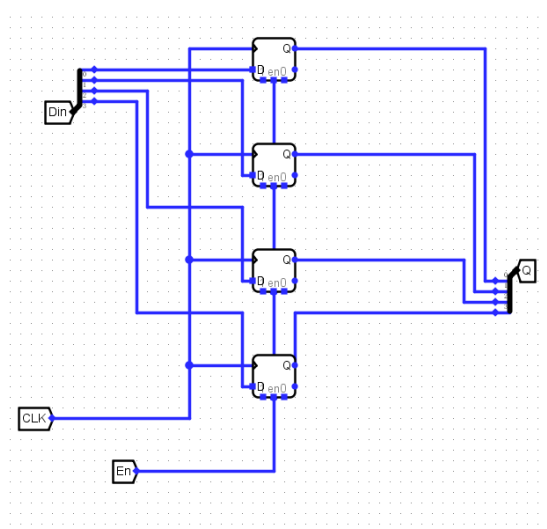


图 1-9

5) 4 位 BCD 计数器:

# 华中科技大学课程实验报告

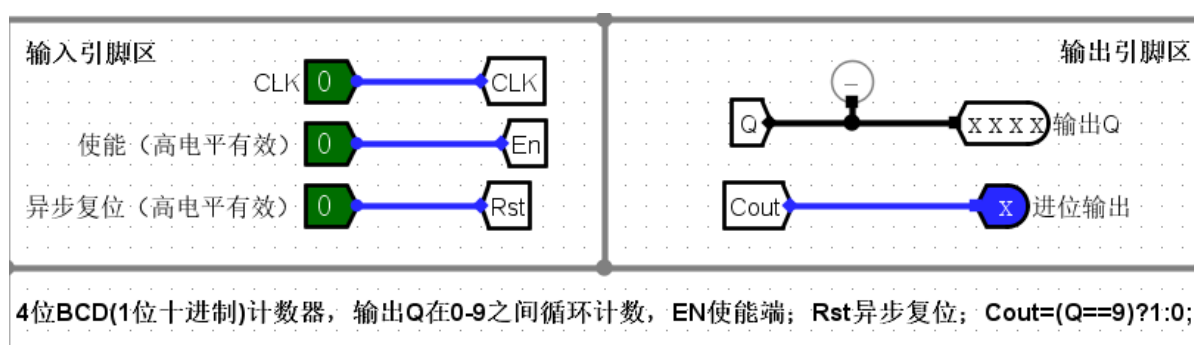


图 1-10

由“状态转换”、“输出函数”两个模块共同构成：

S3	S2	S1	S0	N3	N2	N1	N0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

图 1-11 状态转换

S3	S2	S1	S0	Cout
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

图 1-12 输出函数

6) 码表计数器：

由 4 个 BCD 计数器构成

# 华中科技大学课程实验报告

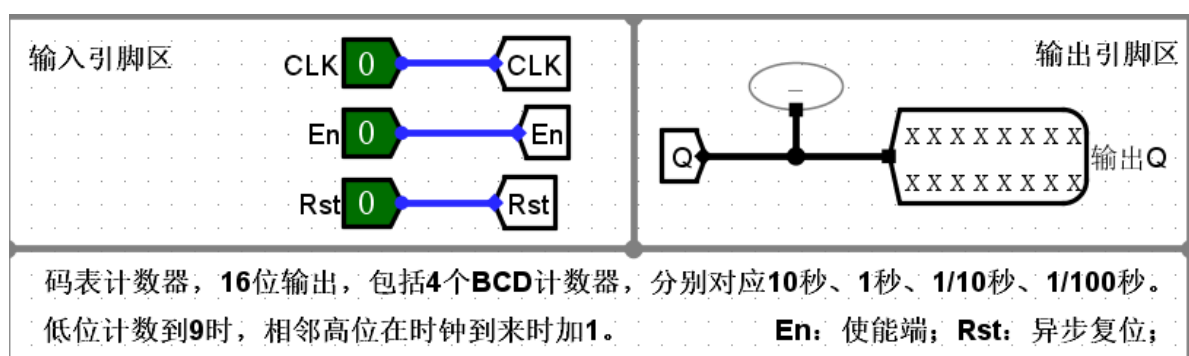


图 1-13

构成方式:

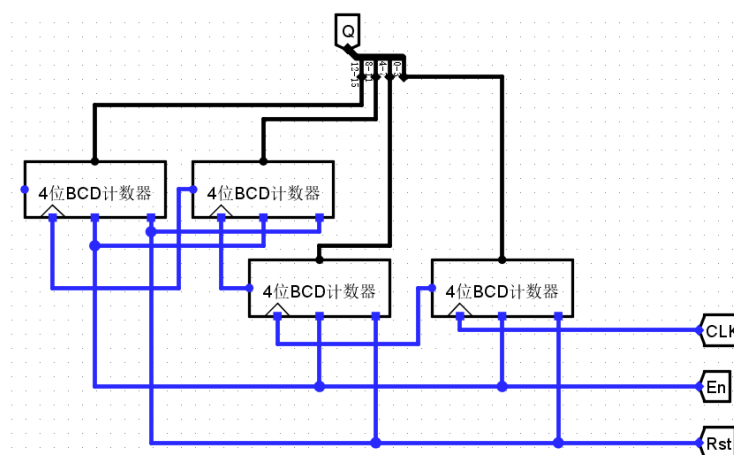


图 1-14

7) 码表控制器:

由“状态转换”、“输出函数”两个模块构成

状态转化表达式:



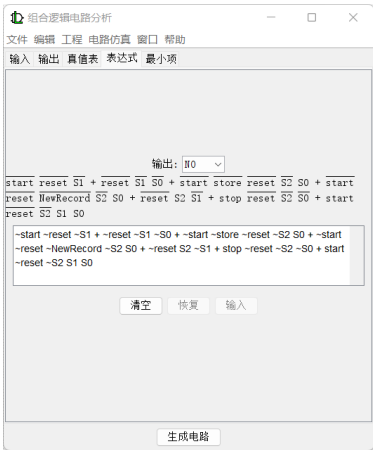


图 1-15

输出函数表达式:

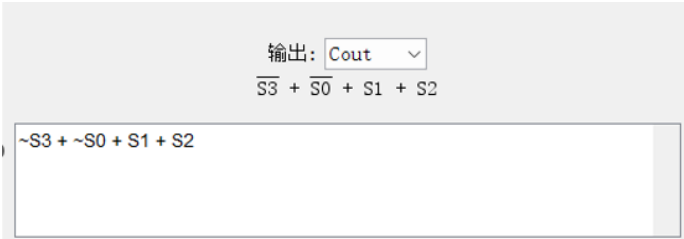


图 1-16

### 1.3 实验步骤

#### 1.3.1 7 段数码管驱动电路

##### a. 内部结构电路

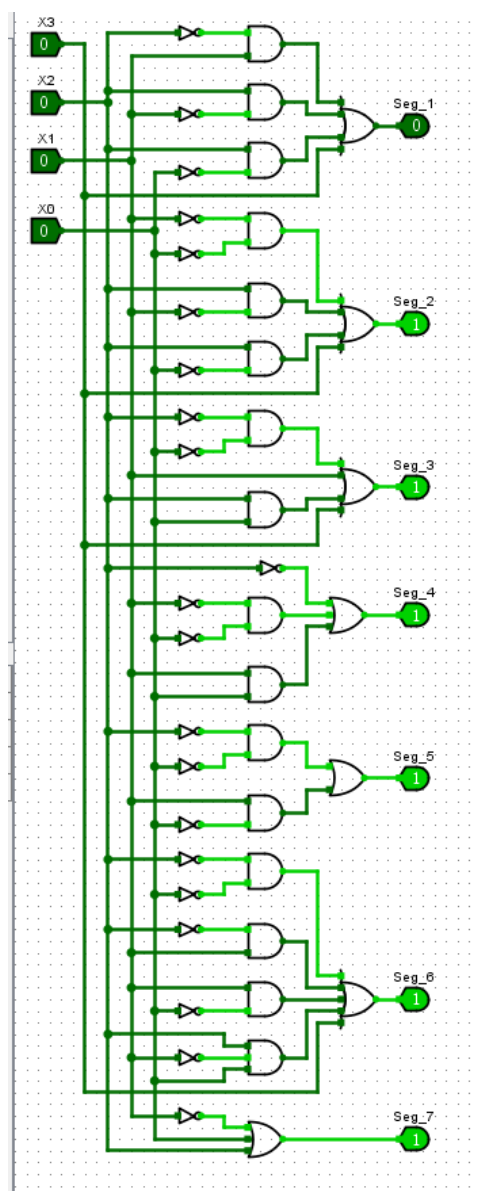


图 1-17

b.封装电路图

其中输入引脚有 x1、x2、x3、x4，输出引脚有 Seg\_1~Seg\_7

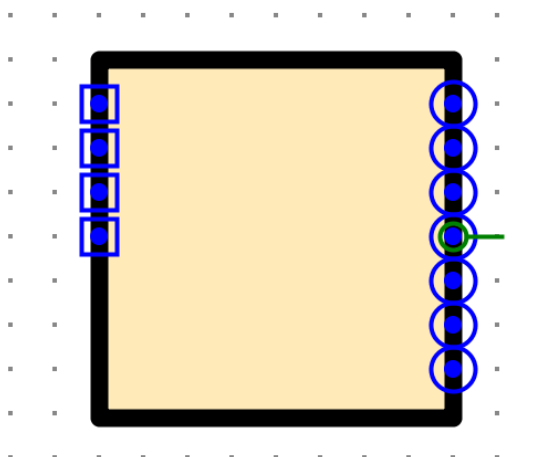


图 1-18

## 1.3.2 2 路选择器（1 位）

### a. 内部结构电路

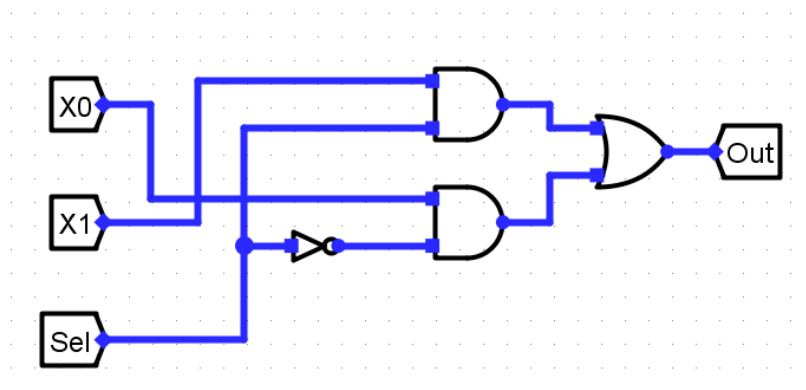


图 1-19

### b. 封装电路

其中输入引脚有 x0、x1、sel，输出电路有 Out

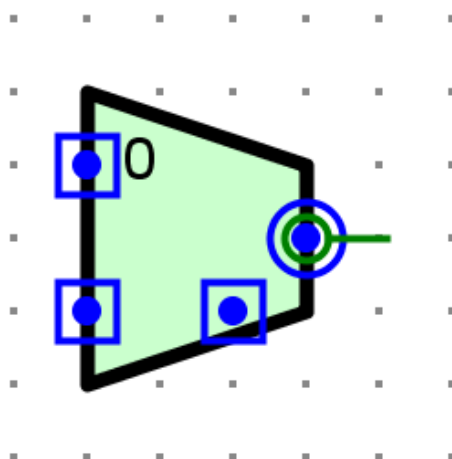


图 1-20

## 1.3.3 2 路选择器（16 位）

### a. 内部结构电路

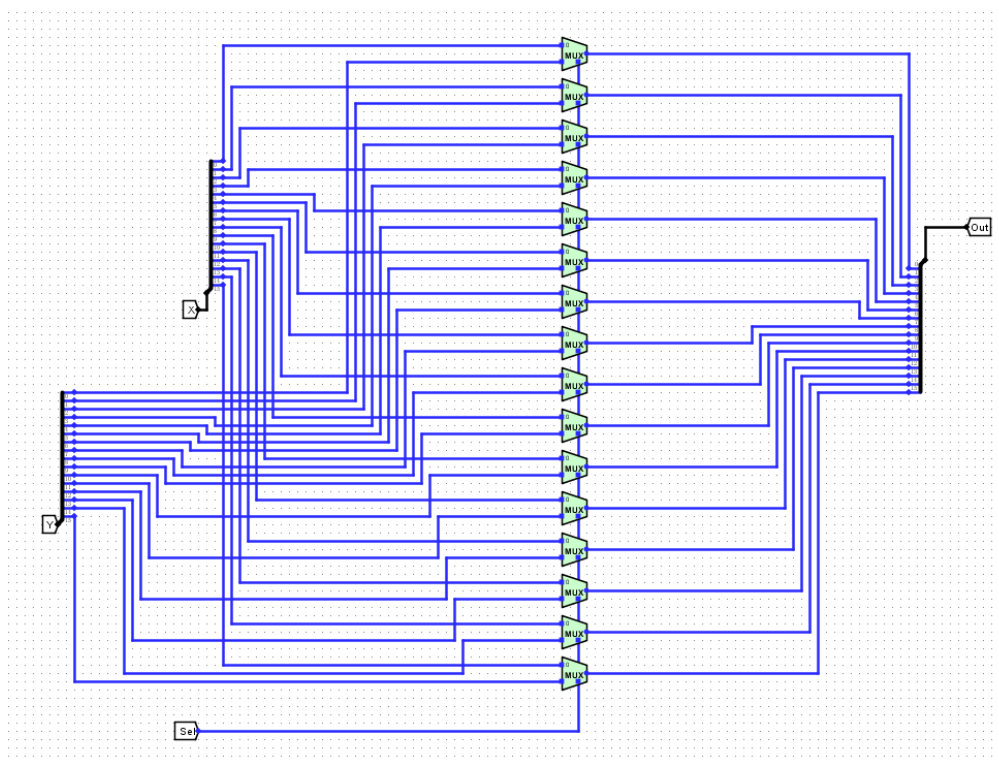


图 1-21

### b. 封装电路

# 华中科技大学课程实验报告

其中输入引脚有，x、y、sel，输出引脚有 out

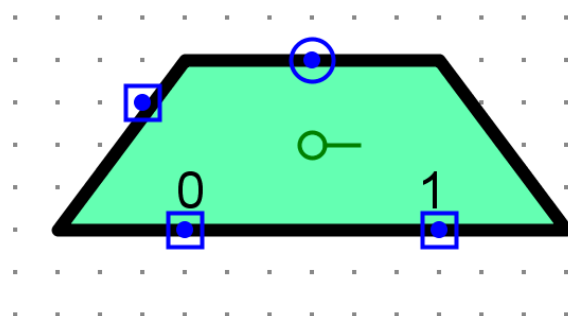


图 1-22

## 1.3.4 4 位无符号比较器

### a.内部结构电路

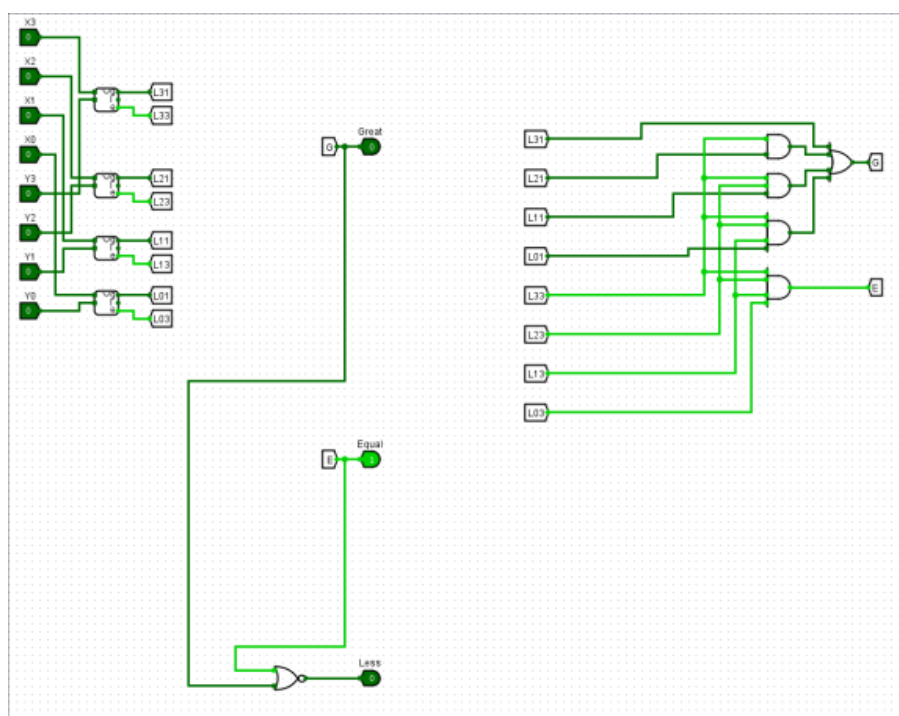


图 1-23

### b.封装电路

其中输入引脚有 x0~x3,y0~y3，输出引脚有 Great，Equal，Less



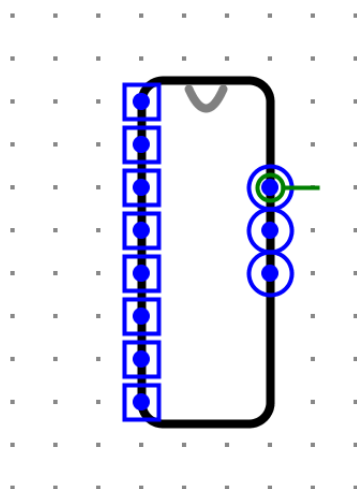


图 1-24

## 1.3.5 16 位无符号比较器

### a. 内部结构电路

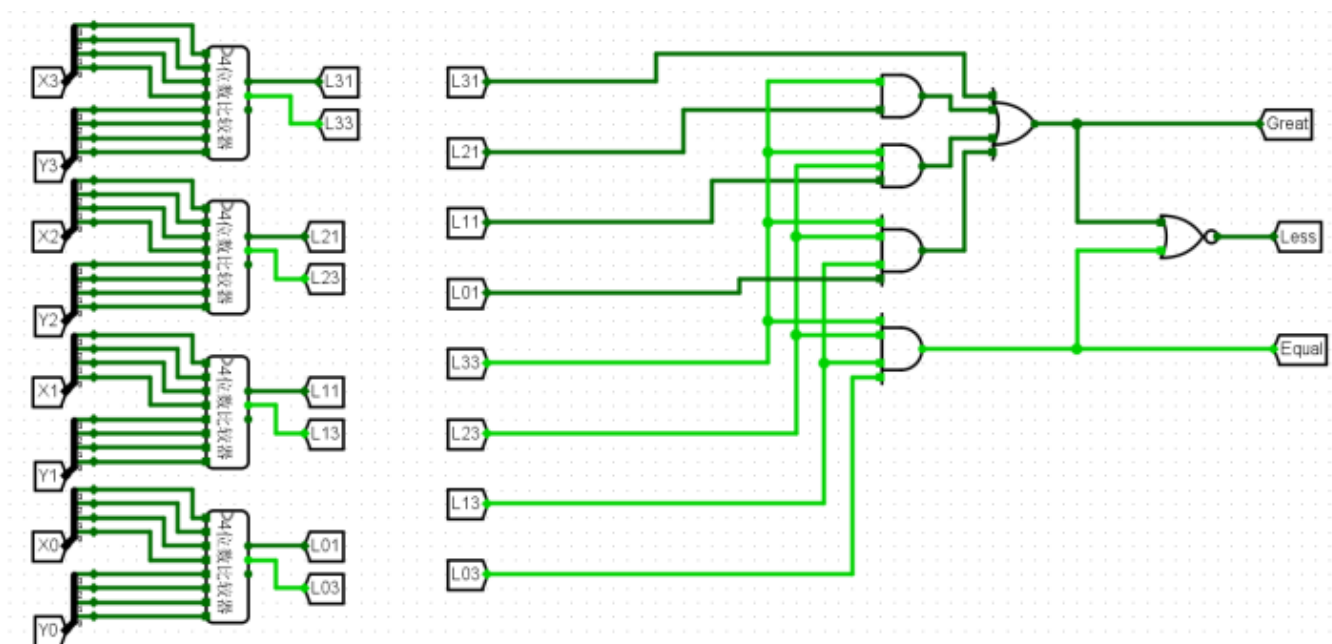


图 1-25

## b.封装电路

其中输入引脚有 x、y，输出引脚有 Great, Equal, Less

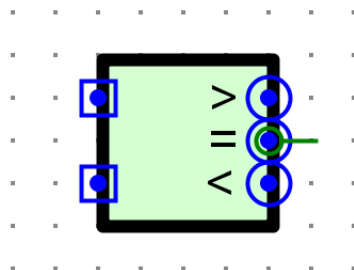


图 1-26

## 1.3.6 4 位并行加载寄存器

### a.内部结构电路

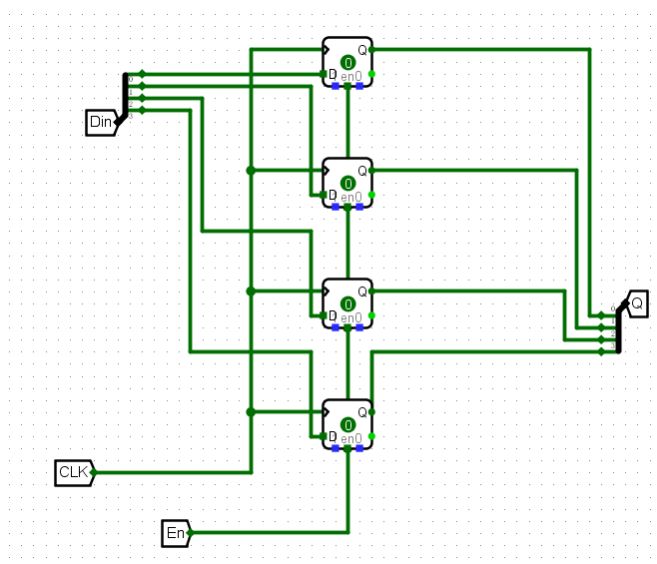


图 1-27

## b.封装电路

其中输入引脚有 En, CLK, Din, 输出引脚有 Q

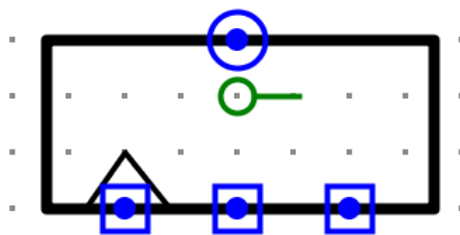


图 1-28

## 1.3.7 16 位并行加载寄存器

### a. 内部结构电路

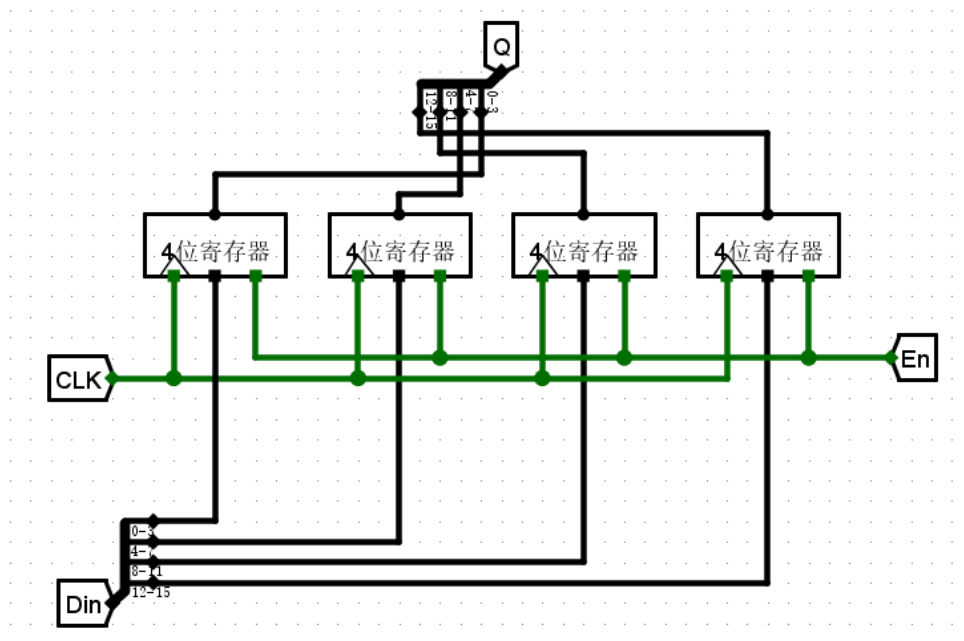


图 1-29

### b. 封装电路

其中输入引脚有 En, CLK, Din, 输出引脚有 Q

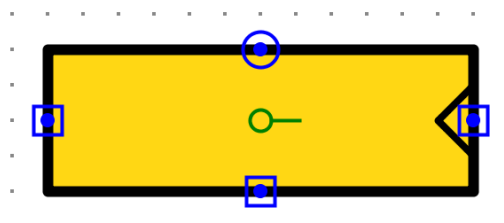


图 1-30

## 1.3.8 BCD 计数器状态转换

### a. 内部结构电路

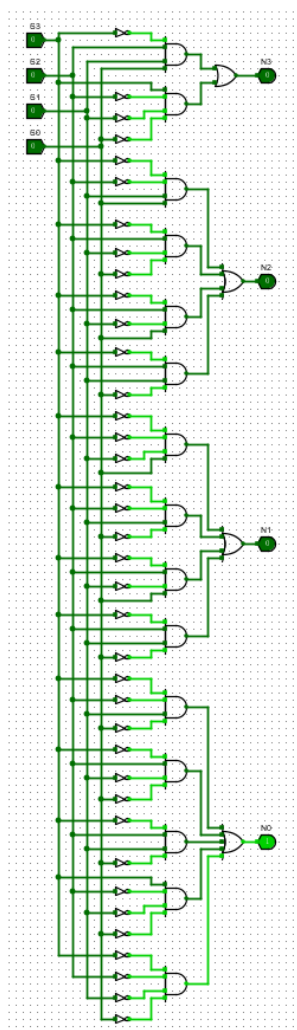


图 1-31

## b.封装电路

其中输入引脚有 S0~S3,输出引脚有 N0~N3

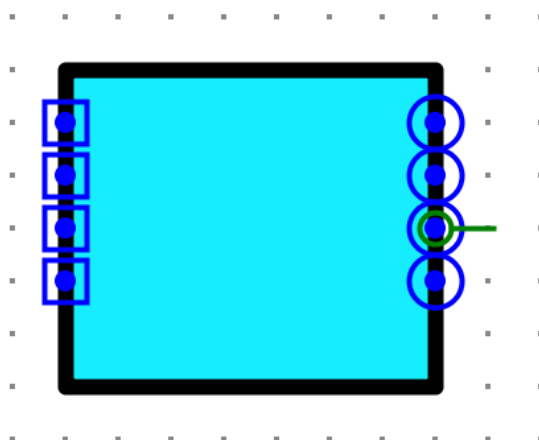


图 1-32

## 1.3.9 BCD 计数器输出函数

### a. 内部结构电路

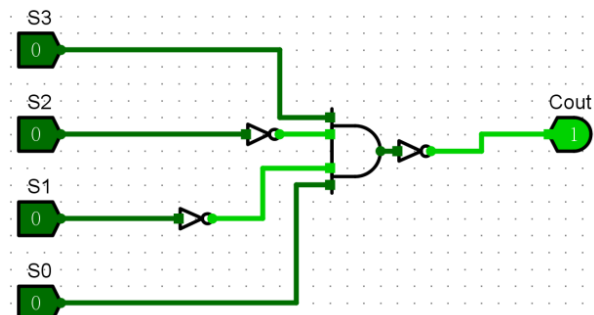


图 1-33

### b. 封装电路

其中输入引脚有 S0~S3，输出引脚有 Cout

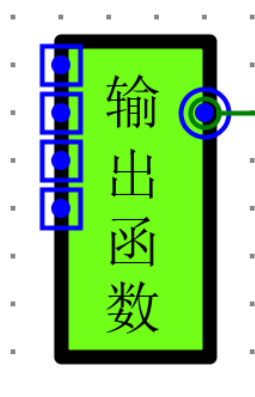


图 1-34

## 1.3.10 4 位 BCD 计数器

### a. 内部结构电路

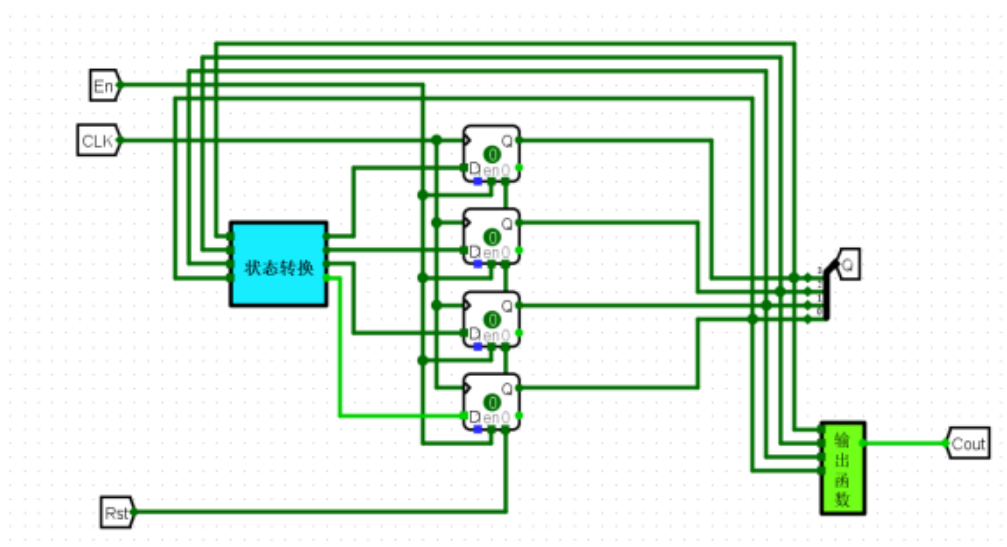


图 1-35

## b. 封装电路

其中输入引脚有 En、CLK、Rst，输出引脚有 Q、Cout

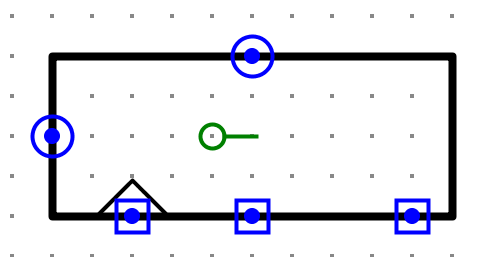


图 1-36

## 1.3.11 码表计数器

### a. 内部结构电路

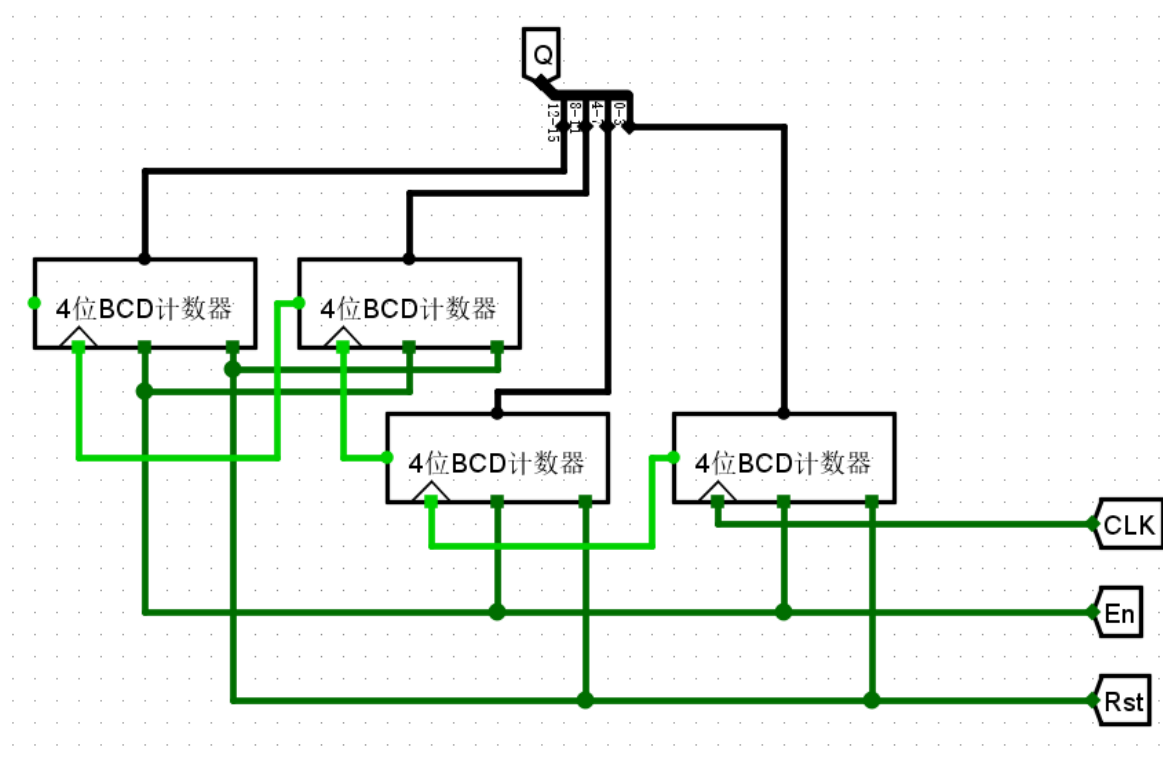


图 1-37

## b. 封装电路

其中输入引脚有 CLK、En、Rst，输出引脚有 Q

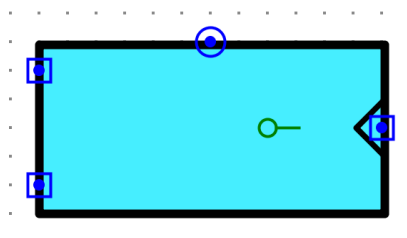


图 1-38

## 1.3.12 码表显示驱动

### a. 内部结构电路

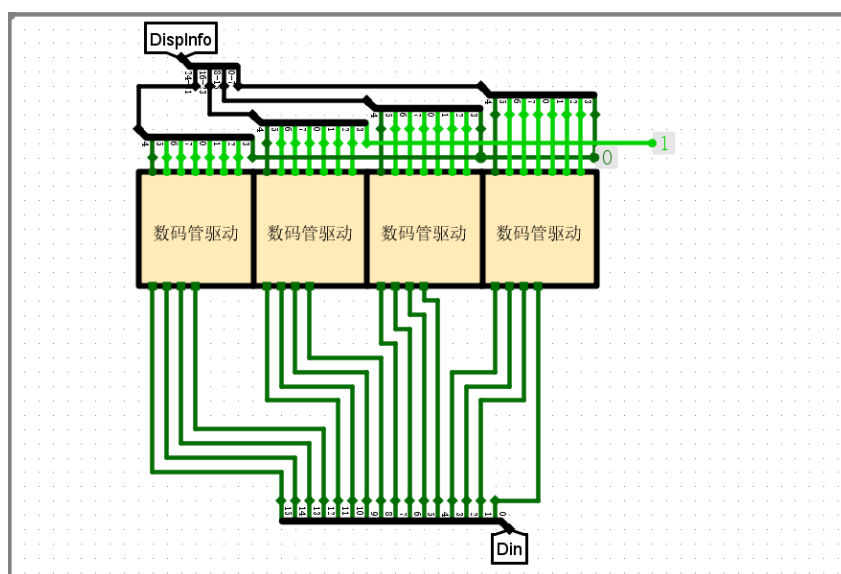


图 1-39

## b. 封装电路

其中输入引脚有 Din，输出引脚有 Dispinfo

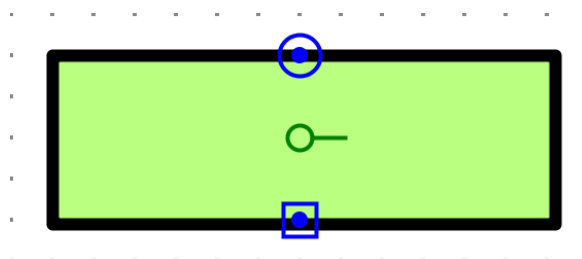


图 1-40

## 1.3.13 码表控制器状态转换

### a. 内部结构电路



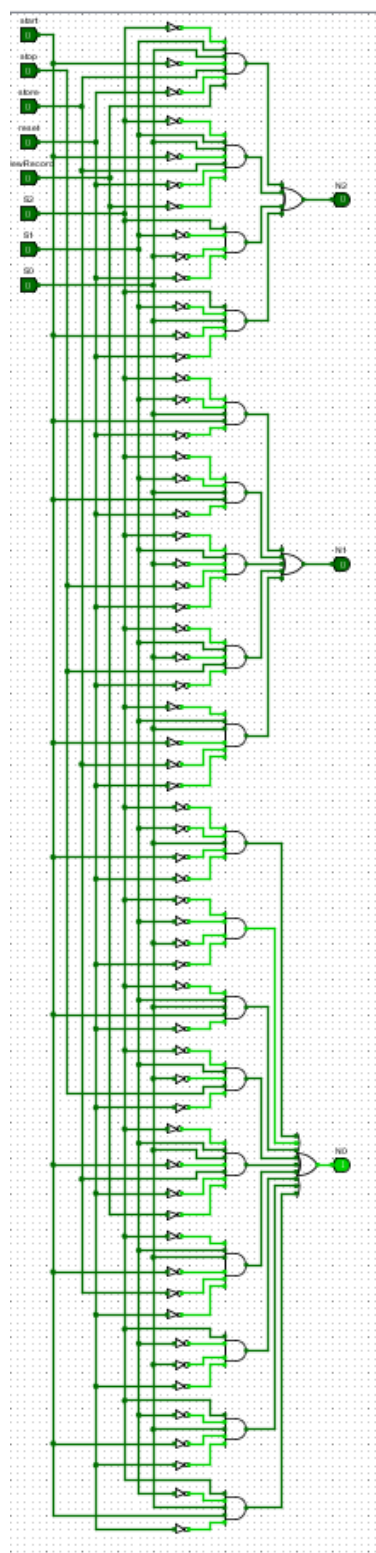


图 1-41

## b.封装电路

其中输入引脚有 start,stop,store,reset,newRecord,S0,S1,S2,输出引脚有 N0、N1、N2

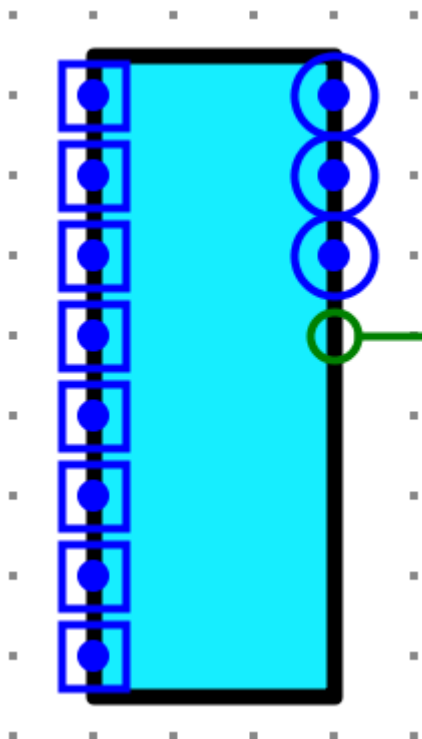


图 1-42

## 1.3.14 码表控制器输出函数

### a. 内部结构电路

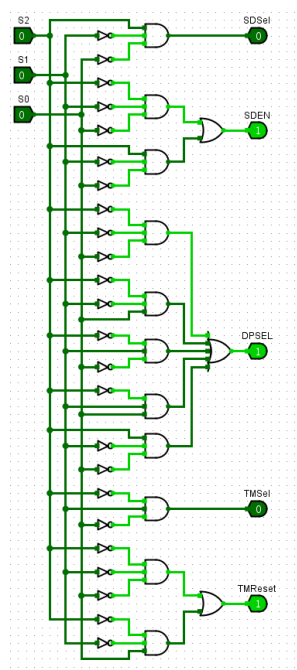


图 1-43

## b.封装电路

其中输入引脚有 S0、S1、S2，输出引脚有 SDSel,SDEN,DPESL,TMsel,TMReset

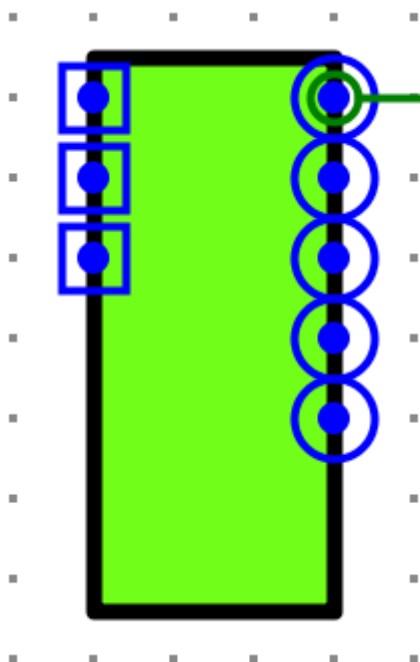


图 1-44

## 1.3.15 码表控制器

### a.内部结构电路

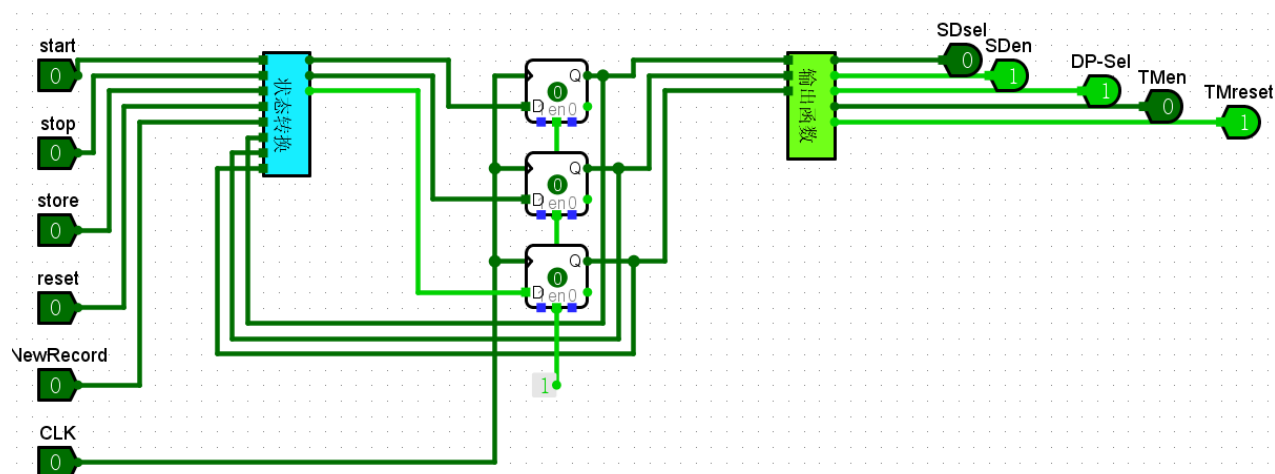


图 1-45

## b.封装电路

其中输入引脚有 start,stop,store,reset,NewRecord,CLK, 输出引脚有



## b.封装电路

其中输入引脚有 SDSel,SDEN,DPESEL,TMsel,TMReset, NewRecord,CLK,输出引脚有 Time, Dispinfo

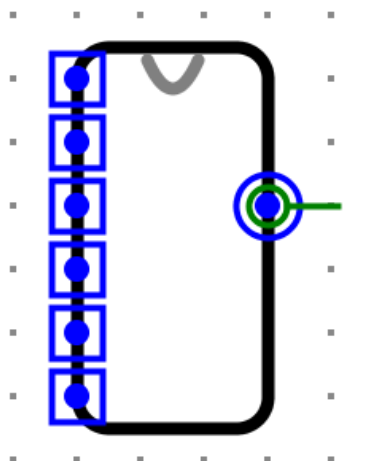


图 1-48

## 1.4 测试与分析

### 1.4.1 7 段数码管驱动电路测试

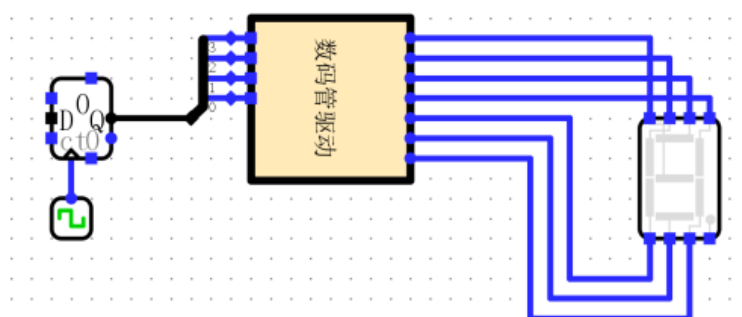


图 1-49 侧视图

测试用例：Ctrl+T 单步计时显示 0~9；

测试结果正确无误，单步运行顺利无误。说明连线正确，电路构成无误。

## 1.4.2 2路选择器（16位）测试

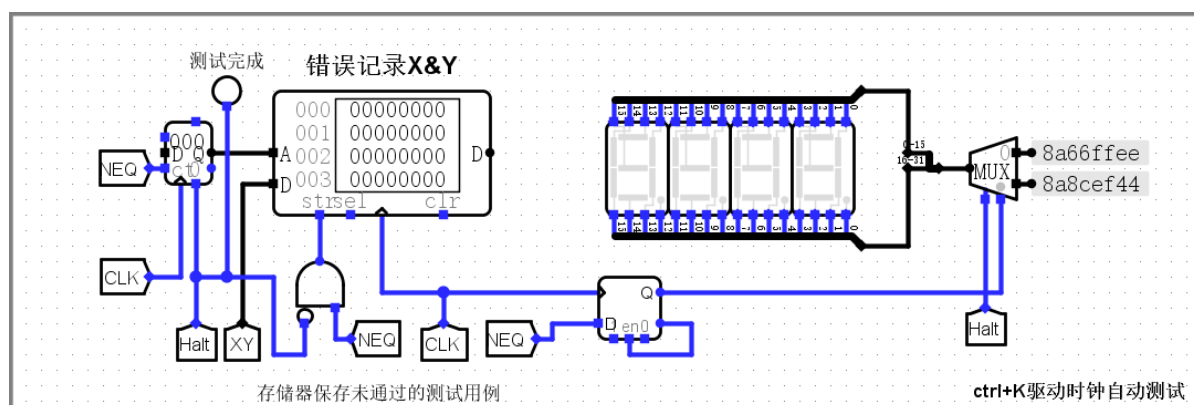


图 1-50 测试图

测试通过，无错误记录。自动测试成功。说明设计电路正确，功能完整无误。

## 1.4.3 16位无符号比较器测试

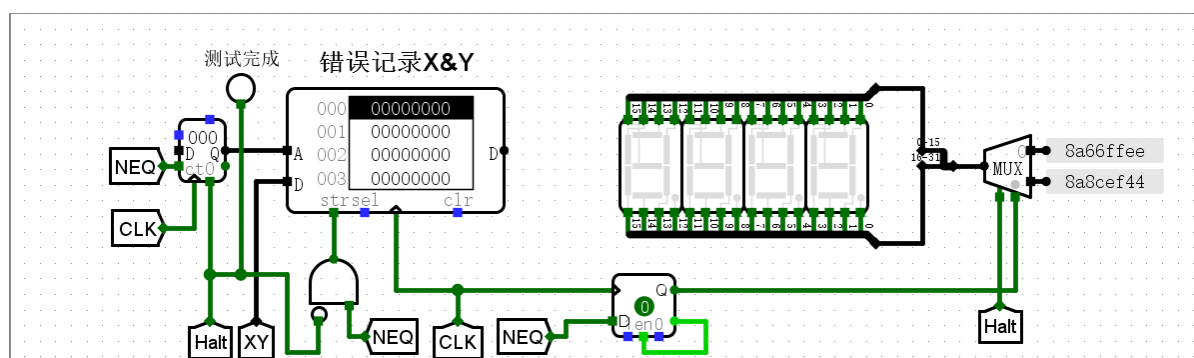


图 1-51 测试图

测试通过，无错误记录。自动测试成功。说明设计电路正确，功能完整无误。

## 1.4.4 码表计数器测试

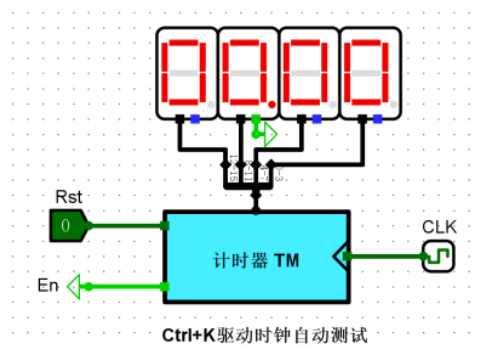


图 1-52 侧视图

# 华中科技大学课程实验报告

测试通过，无错误记录。自动测试成功。说明设计电路正确，功能完整无误。

## 1.4.5 运动码表测试

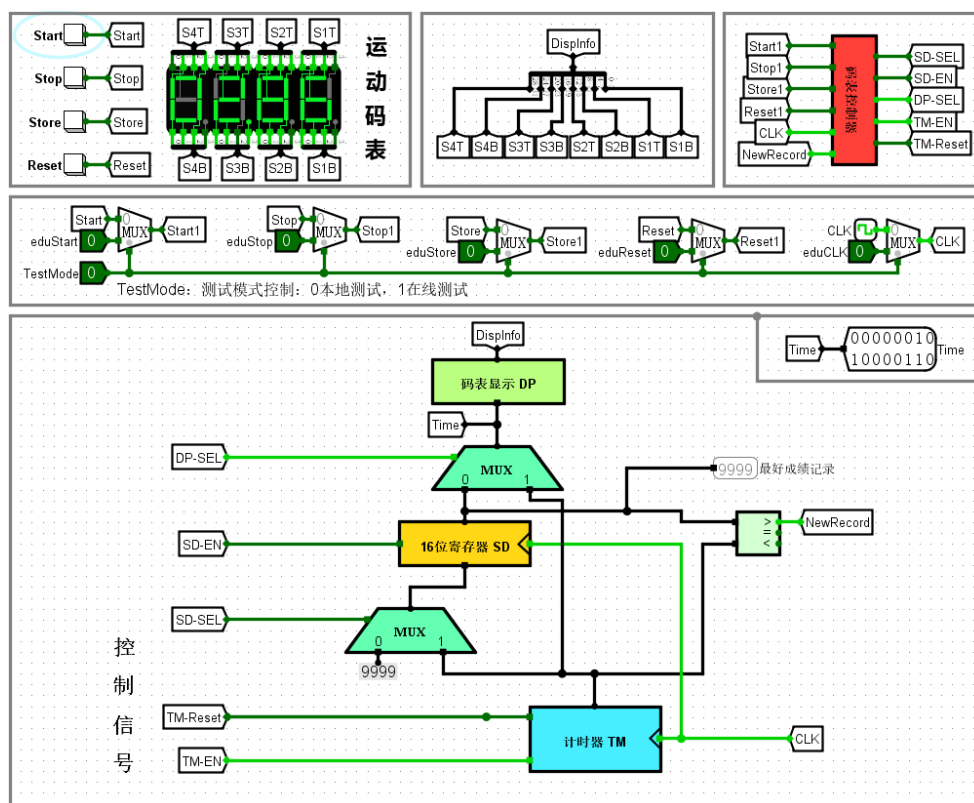


图 1-19 测试图

测试通过，无错误记录。自动测试成功。说明设计电路正确，功能完整无误。

## 2 CPU 设计实验

### 2.1 设计要求

利用 logisim 平台中现有运算部件构建一个 32 位运算器，可支持算数加、减、乘、除，逻辑与、或、非、异或运算、逻辑左移、逻辑右移，算术右移运算，支持常用程序状态标志（有符号溢出 OF、无符号溢出 CF，结果相等 Equal），运算器功能以及输入输出引脚见下表，在主电路中详细测试自己封装的运算器。

表 2-1 片引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零

表 2-2 运算符功能

ALU OP	十进制	运算功能
0000	0	Result = X << Y 逻辑左移 (Y 取低五位) Result2=0
0001	1	Result = X >>> Y 逻辑右移 (Y 取低五位) Result2=0
0010	2	Result = X >> Y 算术右移 (Y 取低五位) Result2=0
0011	3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 有符号
0100	4	Result = X/Y; Result2 = X%Y 无符号
0101	5	Result = X + Y Result2=0 (Set OF/CF)
0110	6	Result = X - Y Result2=0 (Set OF/CF)
0111	7	Result = X & Y Result2=0



# 华中科技大学课程实验报告

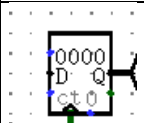
1000	8	Result = $X \mid Y$ Result2=0
1001	9	Result = $X \oplus Y$ Result2=0
1010	10	Result = $\sim(X \mid Y)$ Result2=0
1011	11	Result = $(X < Y) ? 1 : 0$ Signed Result2=0
1100	12	Result = $(X < Y) ? 1 : 0$ Unsigned Result2=0
1101	13	Result = Result2=0
1110	14	Result = Result2=0
1111	15	Result = Result2=0

## 2.2 方案设计

### 2.2.1 单周期 CPU 功能部件

- (1) 指令计数器 PC，用于存放当前指令执行位置的地址；
- (2) 指令存储器 IM，储存 CPU 要执行的指令；
- (3) 数据存储器 DM，根据设计需求，所有指令都必须在一个时钟周期之内完成，只采用一个存储器不可能在一个时钟周期内同时完成对指令和数据的操作；
- (4) 立即数扩展器 S-EXT，用于将 I 型指令中的 16 位立即数扩展为 32 位；
- (5) 控制器，产生控制信号，控制指令执行的数据通路；
- (6) 寄存器堆 RegiFile，提供 32 个 MIPS 通用寄存器；
- (7) 算术逻辑单元 ALU，产生运算结果。

以上所提到的功能部件在 logisim 中的外观如下表 2-3 所示

单周期 CPU 功能部件	部件外观 (logisim)	单周期 CPU 功能部件	部件外观 (logisim)
指令计数器 PC		寄存器	


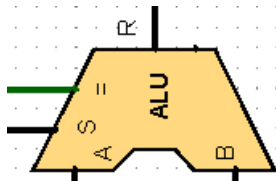
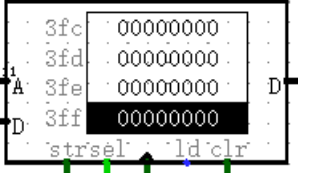
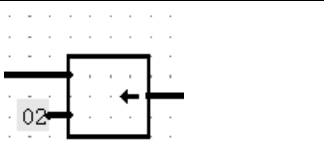
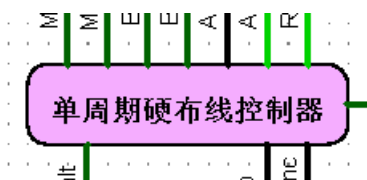
指令存储器 IM		算术逻辑单元 ALU	
数据存储器 DM			
立即数扩展			
控制器			

表 2-3 单周期 CPU 功能部件

2.2.2 多周期 CPU 功能部件

- (1) 指令计数器 PC，用于存放当前指令执行位置的地址；
- (2) 存储器 Mem，包括数据存储器以及指令存储器；
- (3) 指令寄存器 IR，存放当前指令；
- (4) 数据存储器 DR，存放要操作的数据；
- (5) 寄存器堆 RegiFile，提供 32 个 MIPS 通用寄存器；
- (6) 立即数扩展器 S-EXT，用于将 I 型指令中的 16 位立即数扩展为 32 位；
- (7) 控制器，产生控制信号，控制指令执行的数据通路；
- (8) 算术逻辑单元 ALU，产生运算结果；
- (9) 另外，增加三个寄存器 A、B、C 暂存 RegiFile 和 ALU 的数据输出。

以上所提到的功能部件在 logisim 中的外观如下表 2-4 所示

多周期 CPU 功	部件外观 (logisim)	多周期 CPU 功	部件外观 (logisim)
-----------	----------------	-----------	----------------

# 华中科技大学课程实验报告

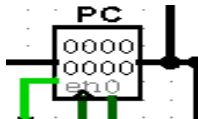
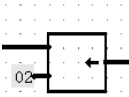

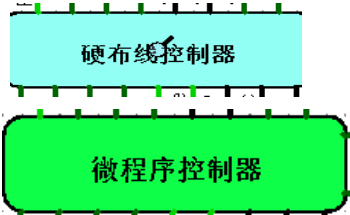
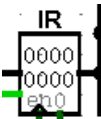
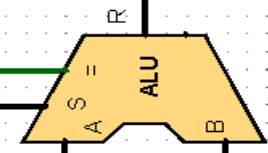


能部件		能部件	
指令计数器 PC		立即数扩展器	
指令存储器 Mem		控制器	
指令寄存器 IR		算术逻辑单元 ALU	
数据存储器 DR			
寄存器堆 RegiFile			

表 2-4 多周期 CPU 功能部件

## 2.2.3 地址转移逻辑 NPC

执行完一条指令后，下一步该执行哪条指令，根据实验要求的八条核心指令分析，有以下几种情况：

- (1) 顺序执行，继续执行相邻的下一条语句；
- (2) 遇到有条件分支，跳转到当前地址加上偏移地址。

两种可能的跳转，一次只能选择一个值，需要 1 个控制信号，由操作控制器生成。

## 2.2.4 立即数扩展器

立即数扩展有两种，都是扩展成 32 位。

- (1) 条件跳转指令中 16 位偏移地址的扩展，偏移量可正可负，需要用符号扩展；

- (2) 在 I 型指令中，操作数为 16 位的立即数，操作数可正可负，需要用符号扩展。

## 2.2.5 操作控制器

操作控制器根据指令的操作码和 func 产生控制信号。使用工程化方法生成。

- (1) 列出所有指令下，各个数据通路的值；
- (2) 将每一项数据都进行合并，当一项数据有多种情况，表明该数据项需要用控制信号进行选择，否则，不需要选择，可以直连；
- (3) 控制信号综合，列出各指令下，每个控制信号的值为 1 或 0，从而可以得出控制信号的逻辑表达式，就可以生成电路了；
- (4) 多周期控制器可分为微程序控制器和硬布线控制器，由于篇幅较长，将在实验步骤中给出。

## 2.3 实验步骤

### 2.3.1 指令解析

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，图 2-1 和图 2-2 所示，指令译码逻辑如图 2-3 所示：

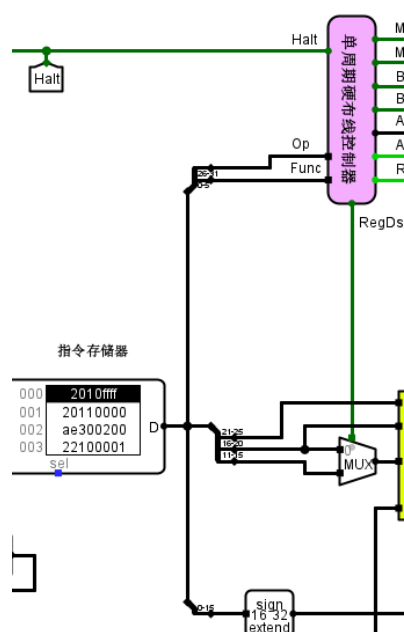


图 2-1 单周期指令解析

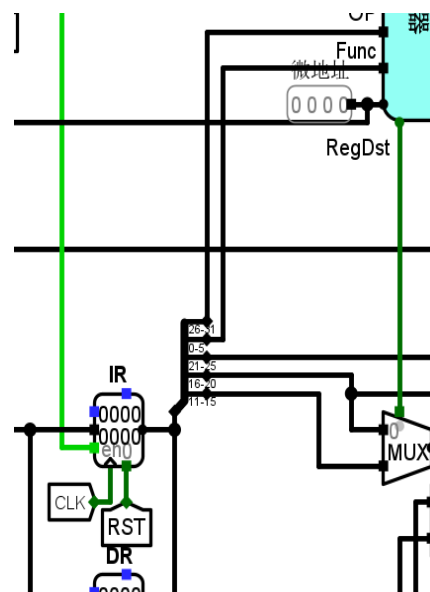


图 2-2 多周期指令解析

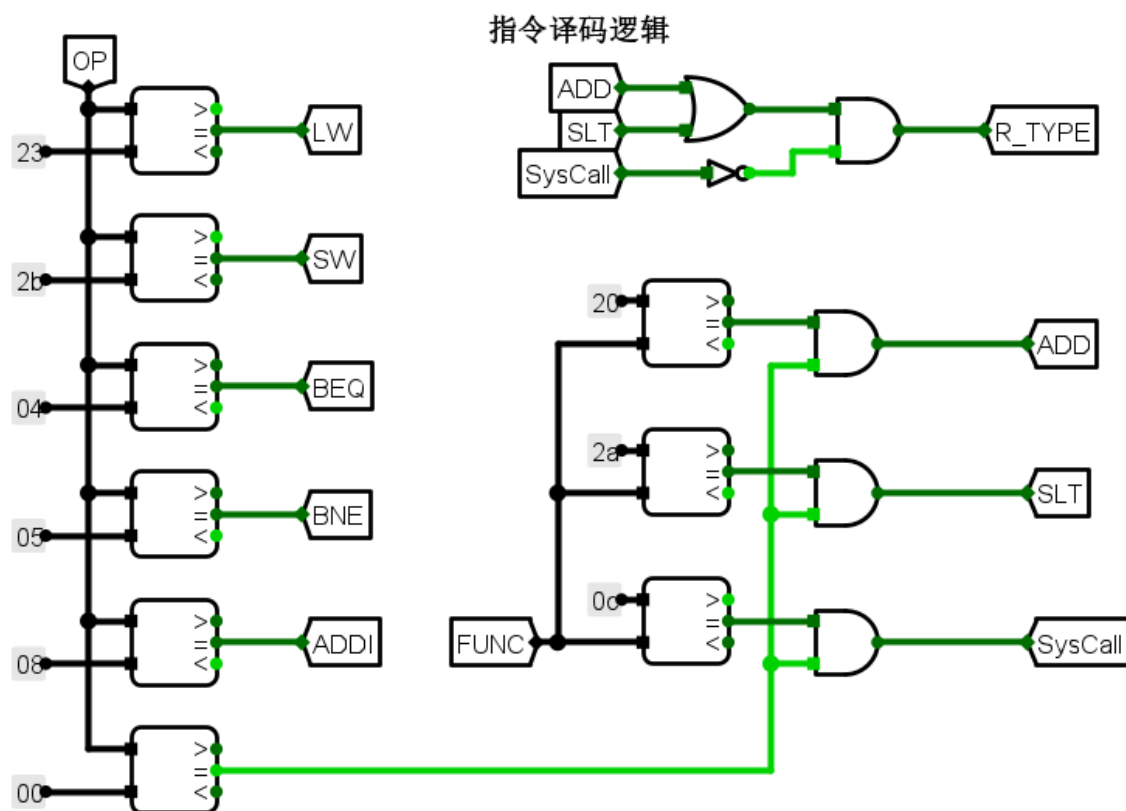


图 2-3 指令译码逻辑

# 华中科技大学课程实验报告

## 2.3.2 实现地址转移逻辑

PC+4 接到 MUX 的 0 输入端，16 位立即数扩展结果左移 2 位，加上 PC+4，接到 MUX 的 1 输入端，选择信号为 branch。实验要求的分支转移指令为 beq 和 bne，两条指令均为 I 型指令，由于两条指令的特殊性，可利用 ALU 的 equal 信号，最终  $branch = beq * equal + bne * /equal$  形成分支信号。

## 2.3.3 单周期硬布线控制器

### (1) 数据通路综合

- 构建数据通路。绘制主要功能部件输入来源表，该表主要用于描述控制类信号，仅保留数据类信号，具体如表 2-5，最左侧为指令助记符，第一行为数据项。
- 输入源合并。计算表 2-5 中每一个数据项的输入种类，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。

表 2-5 单周期功能部件输入来源表

指令	PC	IM	RegiFile (RF)				S-EXT	ALU		DM	
			R1#	R2#	W#	Din		A	B	Addr	Din
add	PC+4	PC	rs	rt	rd	ALU		RF.D1	RF.D2		
slt	PC+4	PC	rs	rt	rd	ALU		RF.D1	RF.D2		
addi	PC+4	PC	rs		rt	ALU	IMM[15:0]	RF.D1	S-EXT		
lw	PC+4	PC	rs		rt	DM.Dout	IMM[15:0]	RF.D1	S-EXT	ALU	
sw	PC+4	PC	rs	rt			IMM[15:0]	RF.D1	S-EXT	RF.D2	ALU
beq	PC+4+offset/PC	PC	rs	rt			IMM[15:0]	RF.D1	RF.D2		
bne	PC+4+offset/PC	PC	rs	rt			IMM[15:0]	RF.D1	RF.D2		
syscall	PC	PC									
合并	2输入	1输入	1输入	1输入	2输入	2输入	1输入	1输入	2输入	2输入	1输入

- 列出所有功能部件、多路选择器控制信号、运算操作选择的产生条件，如表 2-6 所示，横坐标给出的是不同指令的译码信号，表中有 1 的位置表示当前指令会产生对应的信号，利用译码电路生成各指令译码信号，然后以行为单位将各个产生信号的条件相加（逻辑或）即可得到控制信号的逻辑表达式。

- RegDst 为 1 表示 RegiFile 写回地址由 R 型指令 rd 字段给出，否则由 I 型指令 rt 字段给出；
- RegWrite 为 1 打开 RegiFile 写使能，表示数据写回 RegiFile；

# 华中科技大学课程实验报告

- c) AluSrcB 为 1 表示 ALU 的第二个操作数将由立即数扩展器 S-EXT 给出，否则由 RegiFile 的第二个输出给出；
- d) bne 为 1 表示指令为 bne；
- e) beq 为 1 表示指令为 beq，beq 与 bne 信号与 ALU equal 信号结合可控制 PC 数据来源；
- f) MemWrite 为 1 打开数据存储器 DM 写使能，将 ALU 运算结果写入数据存储器 DM；
- g) MemToReg 为 1 表示从数据存储器 DM 中选数据送入 RegiFile；
- h) Halt 为停机信号，为 1 时系统停机；
- i) 加法信号，表示 ALU 应对两个操作数执行加法操作；
- j) 比较信号，表示 ALU 应对两个操作数执行比较操作。

表 2-6 单周期硬布线控制器信号表

控制信号	1	2	3	4	5	6	7	8
	add	slt	addi	lw	sw	beq	bne	syscall
RegDst	1	1						
RegWrite	1	1	1	1				
AluSrcB			1	1	1			
bne							1	
beq						1		
MemWrite					1			
MemToReg				1				
Halt				1				1
加法	1		1	1	1			
比较		1						

(3) 根据以上分析，最终形成的单周期硬布线控制器电路如下：

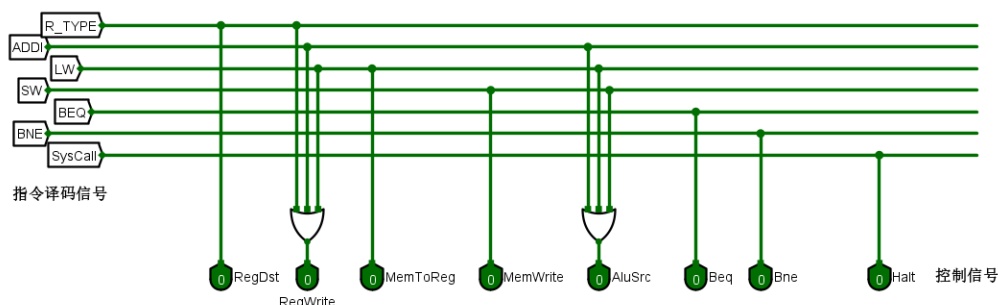


图 2-4 MIPS 单周期 CPU 控制器电路

## 2.3.4 单周期 CPU 总体结构图

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。

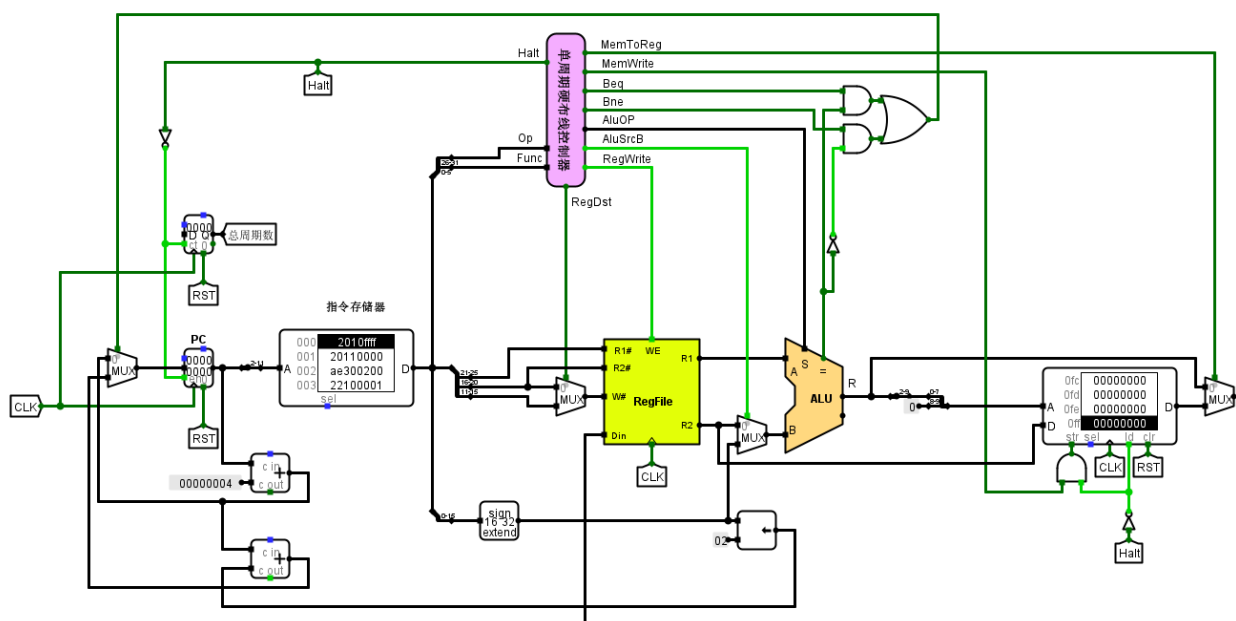


图 2-5 MIPS 单周期 CPU 总体结构图

## 2.3.5 从单周期到多周期

多周期处理器中，指令的执行需要占用多个时钟周期，不同的指令所需的时钟周期数不一定相同，相同指令在不同时钟节拍下产生的控制信号也不同，多周期 CPU 的整体架构也与单周期 CPU 有些许不同，从单周期到多周期处理器的转变涉及多个方面的改变和增加。以下是从单周期到多周期的详细内容说明：

1) 存储器的合并：在多周期处理器中，不再区分指令存储器和数据存储器，而是将指令和数据都保存在同一个存储器中。这样可以减少硬件的复杂性和成本。

2) 功能单元的重复利用：在多周期处理器中，部分功能单元（如 ALU 和寄存器文件）可以在一条指令执行过程的不同周期中多次使用。这意味着同一个功能单元在不同的时钟周期可以执行不同的操作，从而提高了处理器的效率和性能。

3) 数据寄存器的增加：多周期处理器中的主要功能单元的输出端都增加了寄存



# 华中科技大学课程实验报告

器。这些数据寄存器用于在后续时钟周期中存储需要使用的数据。例如，引入了数据寄存器 DR 用于存放从存储器读取的数据，指令存储器 IR 用于存放从存储器读出的指令，以及寄存器 A、B、C 用于保存寄存器文件和 ALU 的输出。

4) ALU 输出结果的变化：在多周期处理器中，ALU 的输出结果有三种情况：a) 分支目标地址：由 beq 和 bne 等分支指令给出，该地址将被写入 PC（程序计数器）中，用于指令跳转。b) 指令的运算结果：将被写入寄存器文件中，用于保存操作的结果。c) 存储器地址：对于访问存储器的指令（如 lw 和 sw），ALU 提供存储器地址。

5) PC 的控制：在多周期处理器中，PC 不再仅由时钟周期控制，而是增加了专门的写操作控制信号。这是因为不同的指令所需的时钟周期数不同，因此需要有专门的控制信号来控制 PC 的写入操作，确保在适当的时钟周期更新 PC 的值。

通过这些改变和增加，多周期处理器能够更灵活地执行指令，提高处理器的效率和性能。同时，由于多周期处理器中的时钟周期数和控制信号变得更加复杂，相应的硬件设计和控制逻辑也会更加复杂。

## 2.3.6 多周期控制信号

多周期 CPU 控制下，不同指令对应不同的时钟周期数，因此不能像单周期控制器一般一次性给出一条指令对应的所有控制信号，而每一条指令的执行又可以拆解为三部分“取指->译码->执行”三个阶段，其中取指和译码两个阶段所有指令对应的数据通路相同，因此现在的任务退化为分析每条指令执行阶段需要的时钟周期并给出每个时钟周期下所有的控制信号，指令的执行控制应遵循以下几个原则：

- (1) 有序进行；
- (2) 不会破坏系统中保存的结果，仅仅对指令本身功能要求对系统状态做出改变；
- (3) 保证不同周期下指令的每一步执行所用到的数据都来自正确来源，保证每一步的运算结果保存到正确输出。

基于此原则，给出取指、译码阶段以及每条指令的执行流程表，如下各表所示：

表 2-7 取指阶段操作流程

指令阶段	操作流程
------	------

# 华中科技大学课程实验报告

取指令	$IR \leftarrow (MEM[PC])$ $PC \leftarrow (PC) + 4$
译码及取操作数	$A \leftarrow (R[IR[25:21]])$ $B \leftarrow (R[IR[20:16]])$ $C \leftarrow (PC) + (S-EXT(IR[15:0]) \ll 2)$

表 2-8 add 指令执行操作流程

指令阶段	操作流程
加运算	$C \leftarrow (A) + (B)$
写回	$R[IR[15:11]] \leftarrow (C)$

表 2-9 slt 指令执行操作流程

指令阶段	操作流程
比较运算	$C \leftarrow ((A) < (B))$
写回	$R[IR[15:11]] \leftarrow (C)$

表 2-10 lw 指令执行操作流程

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

表 2-11 sw 指令执行操作流程

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

表 2-12 beq 指令执行操作流程

指令阶段	操作流程
送目标地址	$If(A == B) PC \leftarrow (C)$

表 2-13 bne 指令执行操作流程

指令阶段	操作流程
送目标地址	$If(A \neq B) PC \leftarrow (C)$

# 华中科技大学课程实验报告

表 2-14 addi 指令执行操作流程

指令阶段	操作流程
加运算	$C \leftarrow -(A) + S\text{-EXT}(IR[15:0])$
写回	$R[IR[20:16]] \leftarrow -(C)$

表 2-15 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

综合以上八条操作流程, 可以给出所有指令不同阶段的控制信号表, 如表 2-16 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。

# 华中科技大学课程实验报告

表 2-16 指令控制信号表

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
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00
LW1	2	0010	00	0	1	10	0	0	0	0	0	0	0	0	0	00
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00
R1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	10
R2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	10
BEQ	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	00
BNE	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	00
ADD1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00
ADD2	12	1100	0	0	0	00	0	0	0	0	1	0	0	0	0	00
SYSCALL	13	1101	1	0	0	00	0	0	0	0	0	0	0	0	0	11

## 2.3.7 多周期微程序控制器

- (1) 状态编码，8 条指令共 13 个执行阶段，采用 4 位二进制编码表示微指令地址，如下表 2-17：

表 2-17 状态编码表

微指令功能	状态	微指令地址
取指令	0	0000
译码	1	0001
LW1	2	0010
LW2	3	0011
LW3	4	0100
SW1	5	0101
SW2	6	0110
R1	7	0111
R2	8	1000
BEQ	9	1001
BNE	10	1010
ADD1	11	1011
ADD2	12	1100
SYSCALL	13	1101

- (2) 微指令地址转移逻辑，采用下址字段法，配合判断状态 P，取指阶段 P 为 1，表明控制存储器的地址由指令的第一个阶段的微程序地址给出，执行阶段 P 为 0，表明下一条微指令地址由下址字段给出，得出取指阶段指令的地

# 华中科技大学课程实验报告

址转移逻辑如下表 2-18:

表 2-18 微指令地址转移逻辑

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

(3) 生成的地址转移逻辑表达式为:

$$S3 = \text{ADDI} + \text{BEQ} + \text{BNE} + \text{SYSCALL};$$

$$S2 = \text{R\_Type} + \text{SW} + \text{SYSCALL};$$

$$S1 = \text{R\_Type} + \text{ADDI} + \text{LW} + \text{BNE};$$

$$S0 = \text{R\_Type} + \text{ADDI} + \text{SW} + \text{BEQ} + \text{SYSCALL}。$$

(4) 控制存储器 16 进制微指令生成, 根据 2.3.6 的分析, 8 条指令执行需要的控制信号需要 16 位编码, 而采用下址字段法, 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 0 表示取指阶段, 控制存储器地址由指令的微指令转移逻辑产生, P 为 1 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 (即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, 按 2.3.6 的控制信号编码, 产生的微指令如下表 2-19:

# 华中科技大学课程实验报告

表 2-19 微指令编码

微指令功能	状态	微指令地址	微指令	十六进制
取指令	0	0000	000010011001000000001	13201
译码	1	0001	0001100000000000010000	30010
LW1	2	0010	0001100000000000000001	60003
LW2	3	0011	1000000000010000000100	100204
LW3	4	0100	0000010001000000000000	8800
SW1	5	0101	0011000000000000000110	60006
SW2	6	0110	1000000000100000000000	100400
R1	7	0111	0010000000000001001000	40048
R2	8	1000	0000001001000010000000	4840
BEQ	9	1001	0110000000000100000000	C0100
BNE	10	1010	0110000000000100000000	C0080
ADD1	11	1011	00110000000000000001100	6000C
ADD2	12	1100	0000000001000000000000	800
SYSCALL	13	1101	1000000000000001101101	10006D

## 2.3.8 多周期硬布线控制器

- (1) 状态编码，和微程序控制器编码相同；
- (2) 状态转移，，增加有限状态机 FSM 实现现态到次态的转换，次态即下一条微指令在控制存储器中的地址，而 FSM 采用纯组合逻辑电路实现，中间是具体的连线，这也是“硬布线”的由来，FSM 状态转换如表 2-20；

表 2-20 FSM 状态转换

现态 10进制	S3	S2	S1	S0	R_Type	LW	SW	BEQ	BNE	SYSCALL	ADDI	次态 10进制	N3	N2	N1	N0
0	0	0	0	0	X	X	X	X	X	X	X	1	0	0	0	1
1	0	0	0	1	1	X	X	X	X	X	X	7	0	1	1	1
1	0	0	0	1	X	1	X	X	X	X	X	2	0	0	1	0
1	0	0	0	1	X	X	1	X	X	X	X	5	0	1	0	1
1	0	0	0	1	X	X	X	1	X	X	X	9	1	0	0	1
1	0	0	0	1	X	X	X	X	1	X	X	10	1	0	1	0
1	0	0	0	1	X	X	X	X	X	1	X	13	1	1	0	1
1	0	0	0	1	X	X	X	X	X	X	1	11	1	0	1	1
2	0	0	1	0	X	X	X	X	X	X	X	3	0	0	1	1
3	0	0	1	1	X	X	X	X	X	X	X	4	0	1	0	0
4	0	1	0	0	X	X	X	X	X	X	X	0	0	0	0	0
5	0	1	0	1	X	X	X	X	X	X	X	6	0	1	1	0
6	0	1	1	0	X	X	X	X	X	X	X	0	0	0	0	0
7	0	1	1	1	X	X	X	X	X	X	X	8	1	0	0	0
8	1	0	0	0	X	X	X	X	X	X	X	0	0	0	0	0
9	1	0	0	1	X	X	X	X	X	X	X	0	0	0	0	0
10	1	0	1	0	X	X	X	X	X	X	X	0	0	0	0	0
11	1	0	1	1	X	X	X	X	X	X	X	12	1	1	0	0
12	1	1	0	0	X	X	X	X	X	X	X	0	0	0	0	0
13	1	1	0	1	X	X	X	X	X	X	X	13	1	1	0	1

- (3) 硬布线控制存储器，地址编码 4 位，由于状态转移已经由 FSM 给出，因此不再需要下址字段和判断字段 P，可减少为 16 位二进制编码，当然本实验中为减少工作量，可同样采用 21 位二进制编码，弃用下址字段和 P 即可，从而硬布线控存和微程序控存相同，在这里不再重复贴出。

## 2.3.9 多周期 CPU 总体结构图

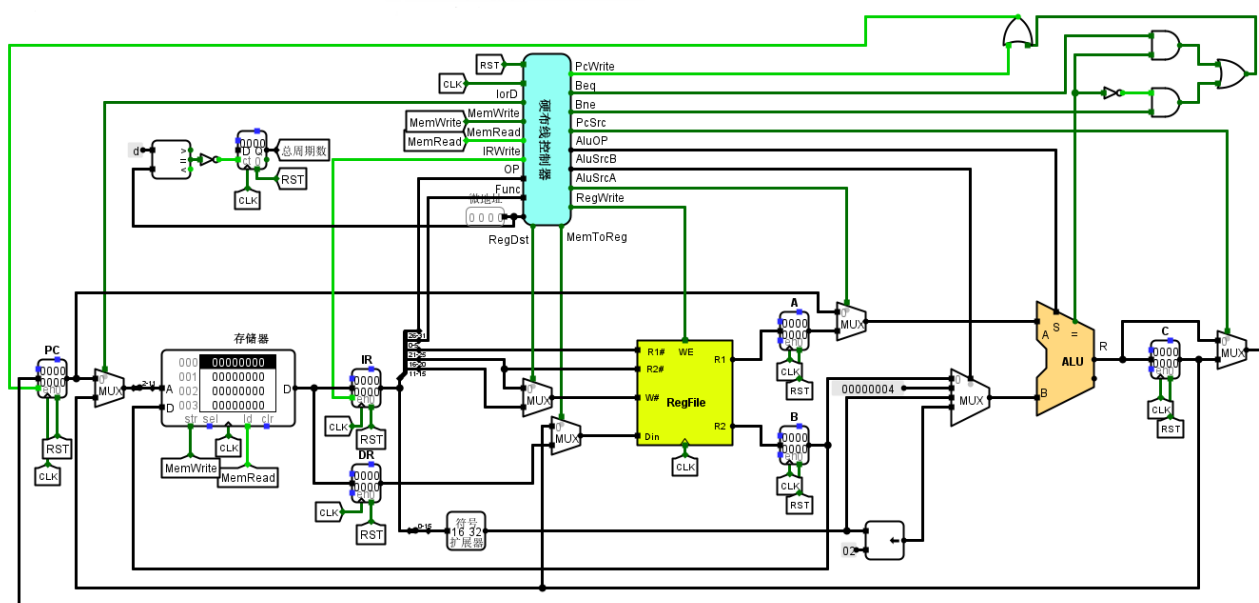


图 2-6 MIPS 多周期 CPU 总体结构图

## 2.4 故障与调试

### 2.4.1 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。

200	00000006	00000000	00000000	00000000
204	00000005	00000000	00000000	00000000
208	00000004	00000000	00000000	00000000
20c	00000003	00000000	00000000	00000000
210	00000002	00000000	00000000	00000000
214	00000001	00000000	00000000	00000000
218	00000000	00000000	00000000	00000000
21c	ffffffff	00000000	00000000	00000000

图 2-7 存储地址错误

**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，如下图：

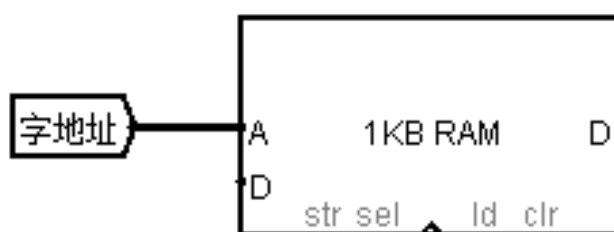


图 2-8 直接字地址访问

**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，如下图：

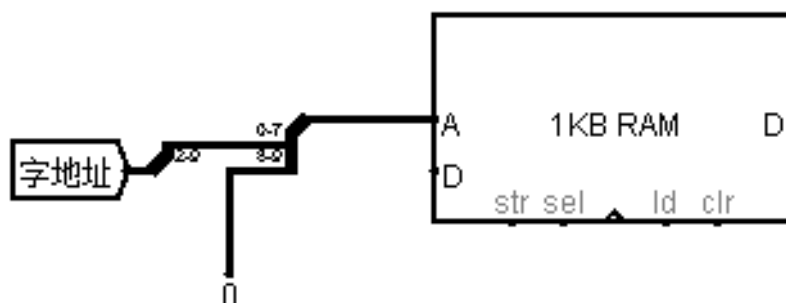


图 2-9 字节地址访问

## 2.4.2 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 syscall 指令与 R 型指令互斥，如下图：

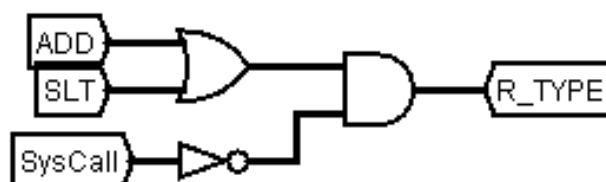


图 2-10 syscall 与 R 型互斥

## 2.5 测试与分析

### 2.5.1 单周期 CPU 执行 sort.hex

(1) 内存布局，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：

078	00000000	00000000	00000000	00000000
07c	00000000	00000000	00000000	00000000
080	00000006	00000005	00000004	00000003
084	00000002	00000001	00000000	fffffffa
088	00000000	00000000	00000000	00000000
08c	00000000	00000000	00000000	00000000

图 2-11 单周期执行 sort.hex 后内存布局

(2) 时钟周期，执行完毕后，系统停机，sort.hex 的时钟周期数为 224，如图：

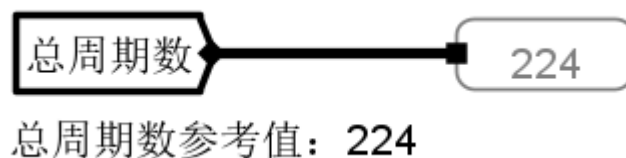


图 2-12 单周期执行 sort.hex 所需时钟周期数

### 2.5.2 多周期 CPU 执行 sort.hex

(1) 内存布局，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：

000	2010ffff	20110000	ae300200	22100001	22310004	ae300200	22100001	22310004
008	ae300200	22100001	22310004	ae300200	22100001	22310004	ae300200	22100001
010	22310004	ae300200	22100001	22310004	ae300200	22100001	22310004	ae300200
018	00008020	2011001c	8e130200	8e340200	0274402a	11000002	ae330200	ae140200
020	2231ffff	1611ffff	22100004	2011001c	1611ffff	2002000a	0000000c	00000000
028	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
030	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
038	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
040	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
048	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
050	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
058	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
060	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
068	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
070	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
078	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
080	00000006	00000005	00000004	00000003	00000002	00000001	00000000	ffffff
088	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

图 2-13 多周期 CPU 执行 sort. hex 后内存布局

(2) 时钟周期，执行完毕后，系统停机，sort.hex 的时钟周期数为 891，如图：

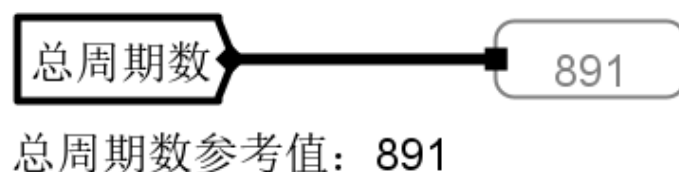


图 2-14 多周期 CPU 执行 sort. hex 所需时钟周期数

## 3 总结与心得

### 3.1 实验总结

#### 3.1.1 运动码表实验总结

本次实验主要完成了如下几点工作：

- 1) 实现了 7 段数码管驱动电路；
- 2) 实现了 1 位和 16 位的 2 路选择器；
- 3) 实现了 4 位和 16 位的无符号比较器；
- 4) 实现了 4 位和 16 位的并行加载寄存器；
- 5) 实现了 4 位 BCD 计数器；
- 6) 实现了码表计数器和码表控制器；
- 7) 实现了码表组件的集成，最终成功实现了运动码表。

#### 3.1.2 CPU 实验总结

本次实验主要完成了如下几点工作：

- 8) 实现了 32 位定长指令的解析；
- 9) 实现了立即数扩展；
- 10) 实现了程序执行指令数目的记录；
- 11) 实现了寄存器读写；
- 12) 实现了数据存储器的读写；
- 13) 实现了操作数的计算；
- 14) 实现了控制信号的生成；
- 15) 实现了各部件多输入来源的多选一；
- 16) 实现了 MIPS 多周期 CPU 控制存储器；
- 17) 实现了 MIPS 多周期 CPU 微程序地址转移逻辑；
- 18) 实现了 MIPS 多周期 CPU 硬布线控制器状态机；
- 19) 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。

## 3.2 实验心得

在完成计算机组成原理实验中运动码表的设计和 CPU 的设计过程中，我获得了以下几点收获和心得体会：

1) 工程化方法和调试技巧： 通过使用 Logisim 进行电路设计和实现，我学会了使用工程化方法生成电路。这包括了模块化设计、模块连接和调试技巧。在实验过程中，我掌握了常用的调试方法，例如单步执行、信号跟踪和查看内部状态等，这有助于我快速定位和解决问题。

2) CPU 的运行机理和模块划分： 通过实验，我宏观上掌握了 CPU 的运行机理和各个模块之间的关系。我了解了指令的执行流程、数据通路以及控制信号的生成方式。同时，我学会了从微观的角度准确地生成每一个控制信号，确保 CPU 能够正确执行指令。

3) 对 MIPS 核心指令的理解： 在实验中，我熟悉了 MIPS 核心指令集，并尝试实现了其中的一些指令。通过实际的设计和编码过程，我对 MIPS 指令的功能和实现思路有了更深入的了解。这让我更好地理解计算机体系结构和指令级并行的原理。

4) 知识理解的加深： 实验过程中，我深刻体会到了对课程内容理解的加深。通过实际动手设计和实现 CPU，我不仅仅是被动地接受知识，而是主动地运用和巩固所学的知识。在实验过程中，我必须真正理解课堂上的内容，才能顺利完成实验任务。这种实践与理论结合的方式使我对计算机组成原理的知识有了更深层次的掌握。

总而言之，通过这个实验，我不仅学会了 Logisim 工具的使用和电路设计的基本技巧，还深入理解了 CPU 的运行机理、控制信号的生成以及 MIPS 指令的实现思路。这不仅增强了我对计算机组成原理的理解，也提高了我的问题解决能力和工程实践能力。这样的实验经历对我的学习和未来的职业发展都具有积极的影响。

• 指导教师评定意见 •

---

### 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：

马耀辉

### 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：\_\_\_\_\_