***2022***



**逻辑与计算机系统设计基础**

**实验报告**

|  |  |
| --- | --- |
| 题 目： | 多周期CPU设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科本硕博2001班 |
| 学 号： | U202015628 |
| 姓 名： | 柳子淇 |
| 电 话： | 15643411133 |
| 完成日期： | 22/6/23 |

目 录

[1 课程实验概述 2](#_Toc634)

[1.1 实验目的 2](#_Toc13063)

[1.2 设计任务 3](#_Toc25199)

[1.3 设计要求 3](#_Toc19388)

[1.4 技术指标 3](#_Toc14650)

[2 方案设计 4](#_Toc1215)

[2.1 运算器设计 4](#_Toc30120)

[2.2 存储器设计 10](#_Toc14899)

[2.3 寄存器文件设计 11](#_Toc12718)

[2.4 数据通路设计 13](#_Toc6931)

[2.5 控制器设计 27](#_Toc20946)

[3 实验过程与调试 31](#_Toc28026)

[3.1 测试用例设计及测试结果 31](#_Toc20676)

[3.2 设计中遇到的主要问题 32](#_Toc25171)

[4 实验总结与心得 35](#_Toc8406)

[4.1 实验总结 35](#_Toc23403)

[4.2 实验心得 35](#_Toc6719)

[4.3 问题回答 35](#_Toc24382)

[参考文献 36](#_Toc25662)

# 课程实验概述

## 实验目的

逻辑与计算机系统设计基础实验是独立开设的重要设计性、综合性实践环节，通过课程实验环节，帮助学生深刻理解数字系统、冯诺依曼结构计算机的工作原理，熟悉计算机各大功能部件的功能特性、时间特性、数据通路等知识；熟练掌握组合逻辑、时序逻辑电路和计算机核心部件-运算器、控制器的工作原理及设计方法，并建立整机的概念。《逻辑与计算机系统设计基础实验》是支撑培养学生解决计算机领域复杂工程问题能力的重要环节，学生只有基于对计算机工作原理深入的理解，才能建立满足特定要求的控制器原理模型，并在此基础上通过现代化工具设计简单的控制器，并与设计的其它功能部件一起构成简单的整机，从而强化学生计算机硬件系统设计与实现能力的培养。

该课程实验属于设计型实验，不仅锻炼学生常见组合逻辑电路、同步时序逻辑电路及简单计算机系统的设计能力，而且通过实验中所遇故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

本实验的具体目标包括：

**目标1：**掌握硬件及系统实验设计方法，能基于计算机组成原理理论知识，根据实验任务要求，设计硬件功能部件和简单系统；

**目标2：**掌握基于主流设计工具设计硬件模块或系统的流程和方法；

**目标3**：掌握CPU的设计方法，能根据任务书提出的特殊指令和功能要求，分析CPU设计的关键问题，制定满足特定指令要求的CPU设计可行性方案，并进行软硬协同的系统设计，能在设计中体现创新意识；

**目标4:** 能根据CPU设计系统方案功能验证、方案优化和故障分析的需要，研究技术路线、制定实验方案、搭建硬件和仿真平台，实施实验、采集和整理数据；

**目标5：**能分析实验数据、解释实验中出现的特殊现象，并通过数据分析和现象解释定位并解决实践过程中遇到的问题或对系统设计进行优化；

**目标6：**熟悉硬件设计主流工具(Logisim 、EDA软件、FPGA开发平台等）的功能、特点及使用方法，掌握基于主流设计工具设计控制器和简单计算机系统的流程和方法，能分析不同工具的局限性；

**目标7：**能根据设计任务要求组建团队，成员分工明确、合理。能理解多学科背景下的团队中各角色的定位与责任，能够胜任个人承担的角色任务，并能与团队其他成员有效沟通，保持团队的高效运行；

**目标8**：通过实验检查和验收过程中的问答、撰写实验报告等活动，培养学生与专业有关的沟通与表达能力。

## 设计任务

本课程实验的总体目标是利用开源仿真工具Logisim，设计运算器、存储器、寄存器文件、控制器，并构成支撑特定功能的简单计算机系统。要求所设计的CPU系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序功能。

## 设计要求

(1)给出运算器、存储器、寄存器文件、数据通路、控制器的设计方案和电路；设计方案思想表述清晰、层析分明，电路图突出设计主体；

(2)能清楚描述实验中遇到的主要问题及分析、定位与解决问题的思路与方法；

(3)报告格式规范、正文采用小四号字、1.5倍行距；每段首行缩进2个汉字，非首行不缩进；图、表按章分别编号并命名(参考教材中图、表编号方法)；图、表中的文字为五号字。

(4)如果完成了单周期与多周期CPU设计，报告只需要针对多周期方案即可。

## 技术指标

(1)支持MIPS指令；

(2)能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。

表 1.1 实验支持的指令集

| **#** | **指令助记符** | **简单功能描述** | **备注** |
| --- | --- | --- | --- |
| 1 | Lw | 以rs+E(imm)为地址,从主存中加载数据,放入rt号寄存器中 |  |
| 2 | Sw | 以rs+E(imm)为地址把rt号寄存器的数据存入主存中 |  |
| 3 | Bne | if(rs!=rt)PC=PC+E(imm) |  |
| 4 | Beq | if(rs!==rt)PC=PC+E(imm) |  |
| 5 | Syscall | 调用中断 | 用作停机指令 |
| 6 | Addi | Reg+imm |  |
| 7 | Add | Reg+Reg |  |

**说明：此表中给出你所设计的CPU支持的指令集，根据实际情况填写。**

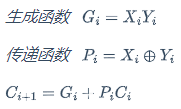
# 方案设计

## 运算器设计

### 基本单元——四位快速加法器

**一．设计思路与总览：**

观察串行进位的全加器：由于高位依赖低位，形成了串行进位链：



快速加法器要实现并行计算就要提前计算进位位。

对于进位位Ci，已经有了生成函数和传递函数，再根据Ci+1与Ci的递推关系，将表达式层层展开，最终用C0直接表示Ci。例如：



而对于每一位结果Si，有：



对于减法运算，我们还需要一个求补电路。这涉及补码的一种求法：从右向左看，第一个1不变，它左边的其余位取反，即得到了补码。然后再用加法器进行计算。

对于溢出检测，需要设计一个溢出检测电路。溢出检测的逻辑为：判断最高位，负数加负数得正数，或正数加正数得到负数，则OF=1，否则OF=0。

**总览：**

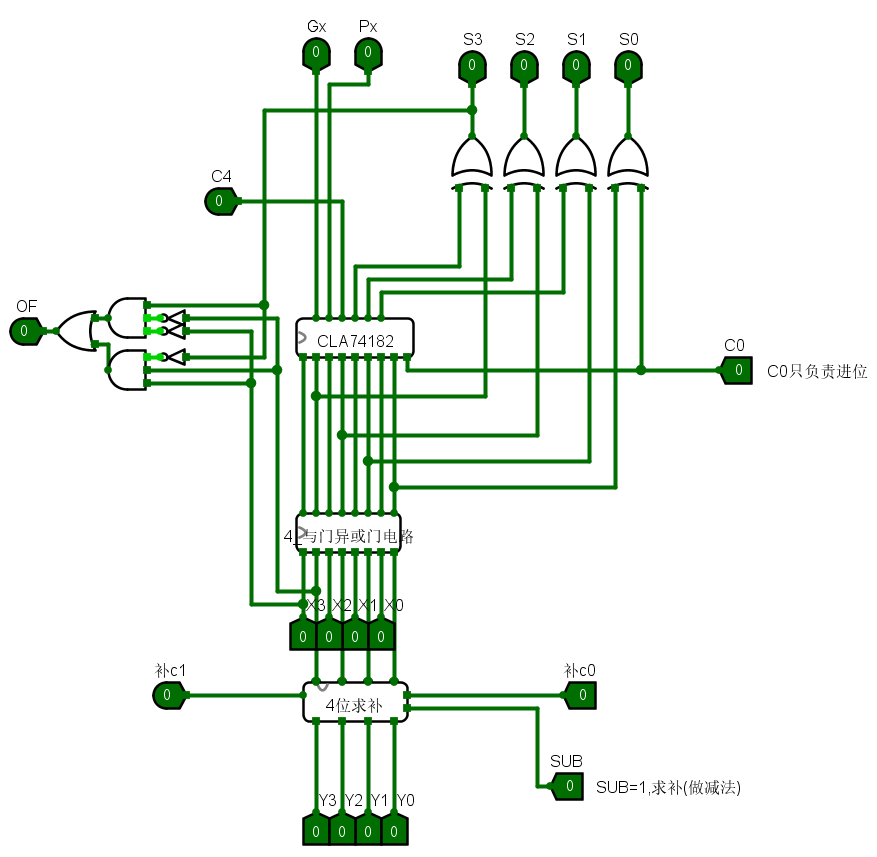


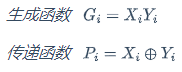
图 2.1.1-1 四位快速加法器

**注：**其中C0只负责作加法时的进位，SUB用于控制是否求补。补C0和补C1负责求补电路的进位，C1表示是否找到了末位1，C0接收上一片的C1。（用于迭代设计）

**二．具体实现：**

**1.四位与门异或门电路：**

用于将Xi，Yi生成相应的Pi，Gi。



**2.四位先行进位电路：**

由C0、Pi、Gi生成对应的C1、C2、C3、C4，以及迭代时需要用到的Px、Gx。

具体逻辑表达式：

*C1=P0 C0 + G0*

*C2=P1 P0 C0 + P1 G0 + G1*

*C3=P2 P1 P0 C0 + P2 P1 G0 + P2 G1 + G2*

*C4=P3 P2 P1 P0 C0 + P3 P2 P1 G0 + P3 P2 G1 + P3 G2 + G3*

*Px=P3 P2 P1 P0*

*Gx=P3 P2 P1 G0 + P3 P2 G1 + P3 G2 + G3*

由上式生成逻辑电路。

**3.求补电路:**

**两位求补电路：**

设计一个数位可拓展的求补电路。S表示是否求补，若S=0，则输出原输入，若S=1，则启用求补。C1表示是否找到了末位1，C0用于接收上一片的C1。



图 2.1.1-2 两位求补电路真值表

**四位求补电路：**

把两个两位求补电路连接，以适配四位快速加法器。并且设有C0、C1两个端口以供迭代设计。

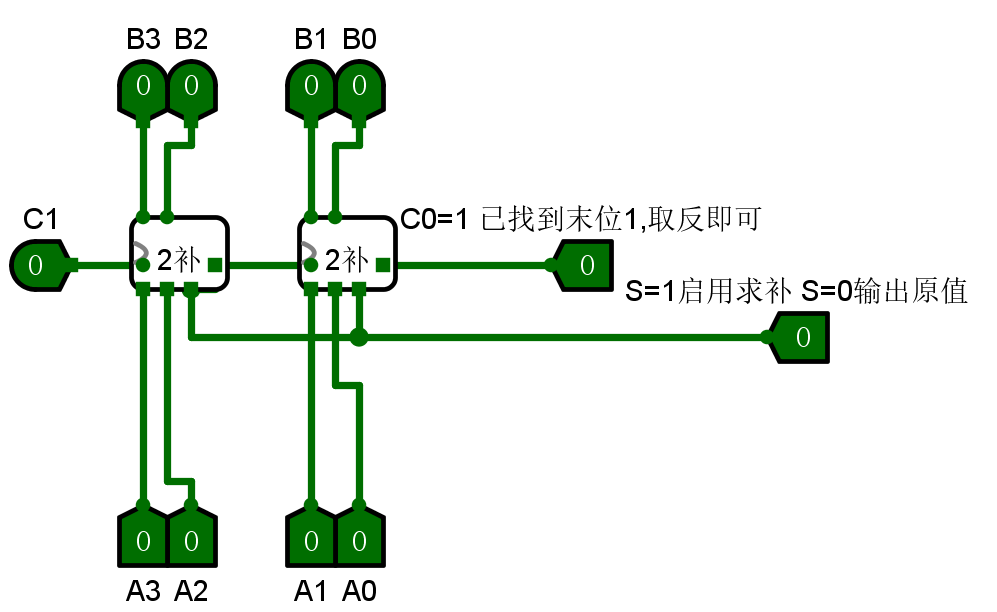


图 2.1.1-3 四位求补电路

### 迭代设计——32位快速加法器

**一．设计思路与总览：**

观察进位位的递推关系表达式:



我们可以进行一个抽象，把Gi看作成组进位生成函数，Pi看成成组进位传递函数。同样符合表达式的形式。所以之前设计的**四位先行进位电路**同样可以用于**四组间的先行进位**，这样得到一个16位快速加法器。

用两片16位快速加法器相连接实现32位快速加法器，然后进行对外封装，将输入输出端整理成32位宽的数据端口，最高进位位设为CF，溢出检测位设为OF。

后续拓展：为了实现BEQ,BNE指令，ALU需要拓展比较功能，在输入端紧接一个比较器。设ZF端口，若X=Y，则ZF=1，反之，ZF=0。

**二．具体实现：**

根据设计思路，将4片四位快速加法器通过CLA74182并成16位快速加法器；对于4位求补电路：根据之前的求补方法，求补电路间只需要传递“是否找到了末位1”的信息。所以直接串行连接各个C1、C0。

然后调整封装，最终如下图所示。

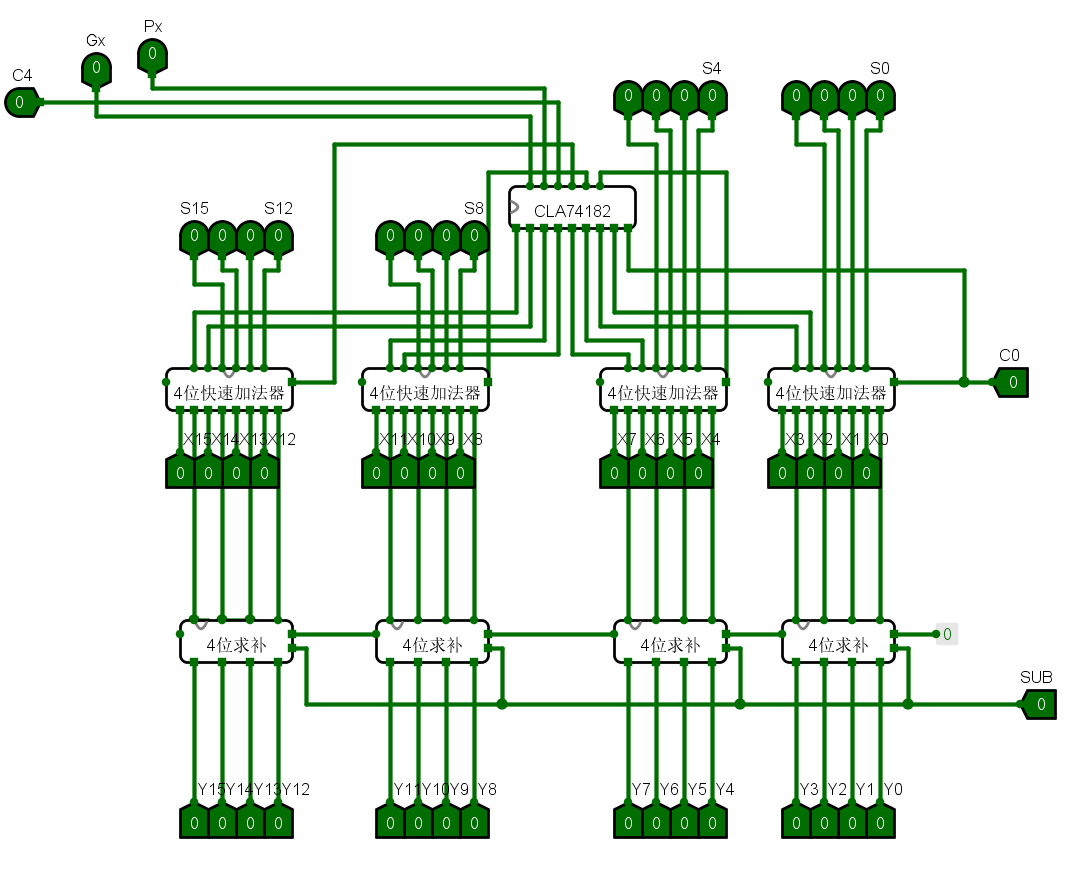


图 2.1.2-1 16位快速加法器

由16位到32位只需要将前一片进位位C1传递给下一片的C0即可。然后再对输入输入端口进行整合，封装成32bit宽的数据端口。如下图所示：

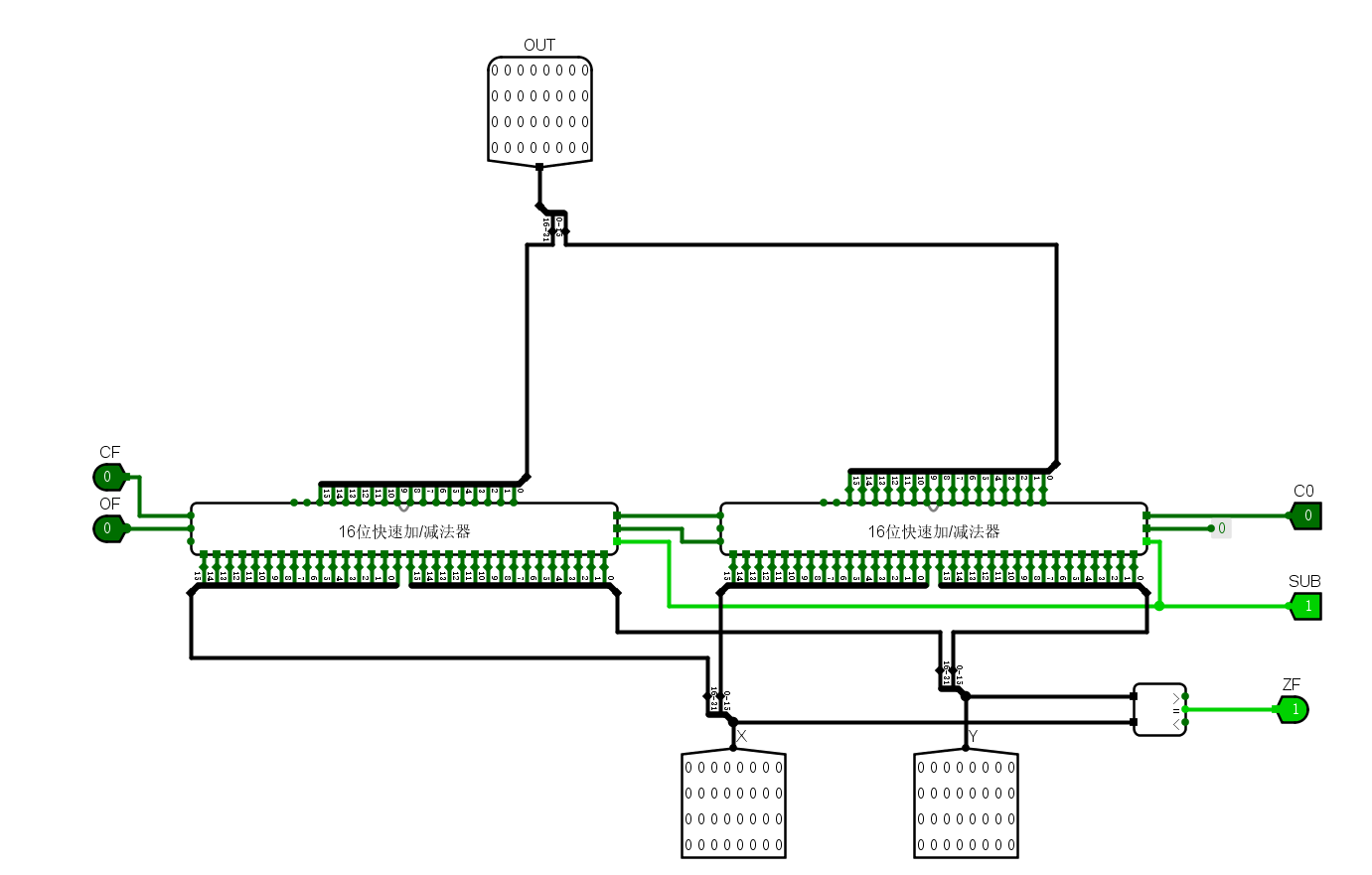


图 2.1.2-2 32位快速加法器

后续添加了比较功能，判断是否相等，增加ZF端口，若X=Y，则ZF=1；反之ZF=0。

## 存储器设计

调用Logisim中的存储库，查阅相关技术文档。用写信号W控制三态门，将原存储库重新封装成一个有数据输入、输出端口端口的存储库，作为主存使用。具体实现如下图所示。

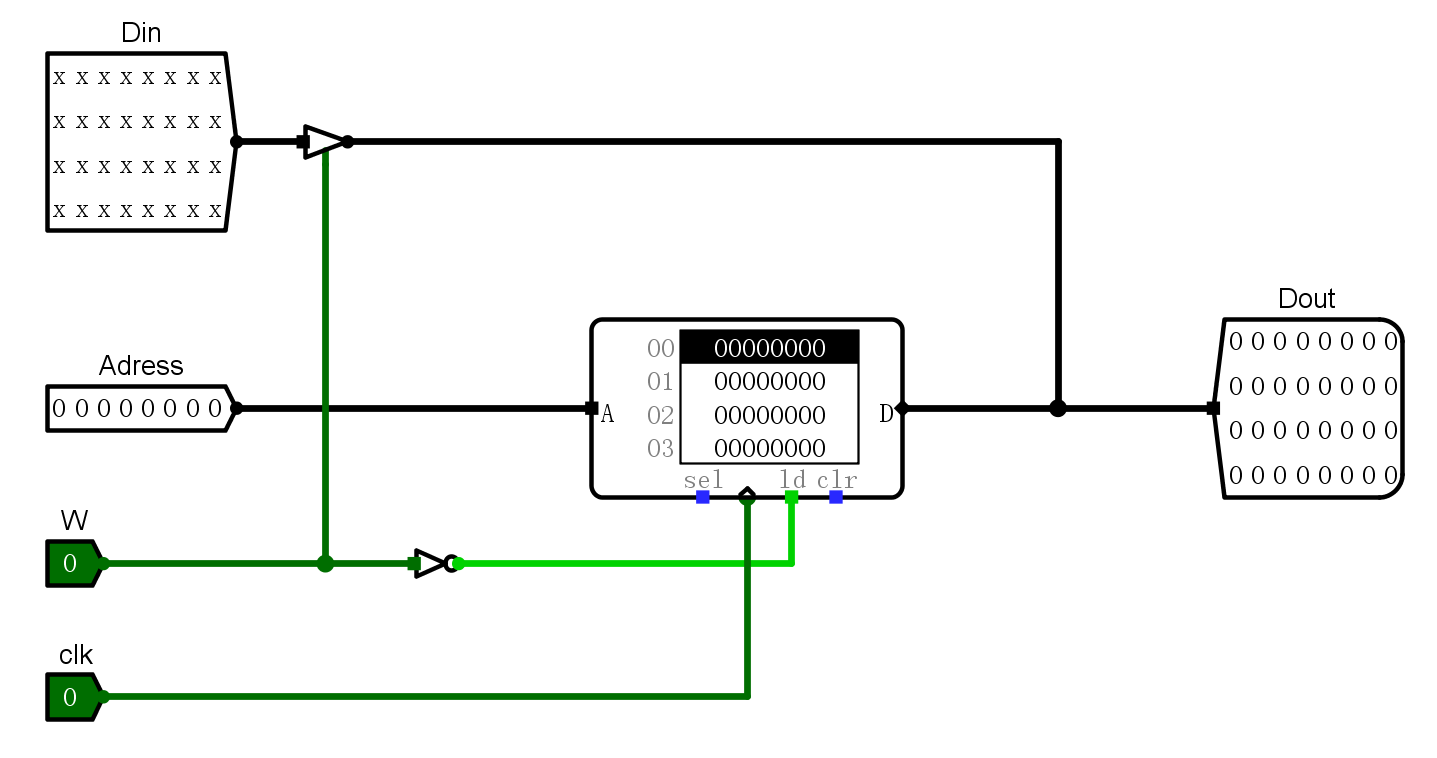


图 . 存储器

## 寄存器文件设计

设计了八个寄存器的寄存器堆。由单周期的寄存器堆化简而来。只保留解码器进行片选，来选择要写入的寄存器。用MUX来选择输出哪个寄存器内的值。之前的单周期专用通路的寄存器堆需要读写控制，可以同时读取两个寄存器，而在多周期实验中寄存器堆的读写控制交给了数据端连接总线的三态门。

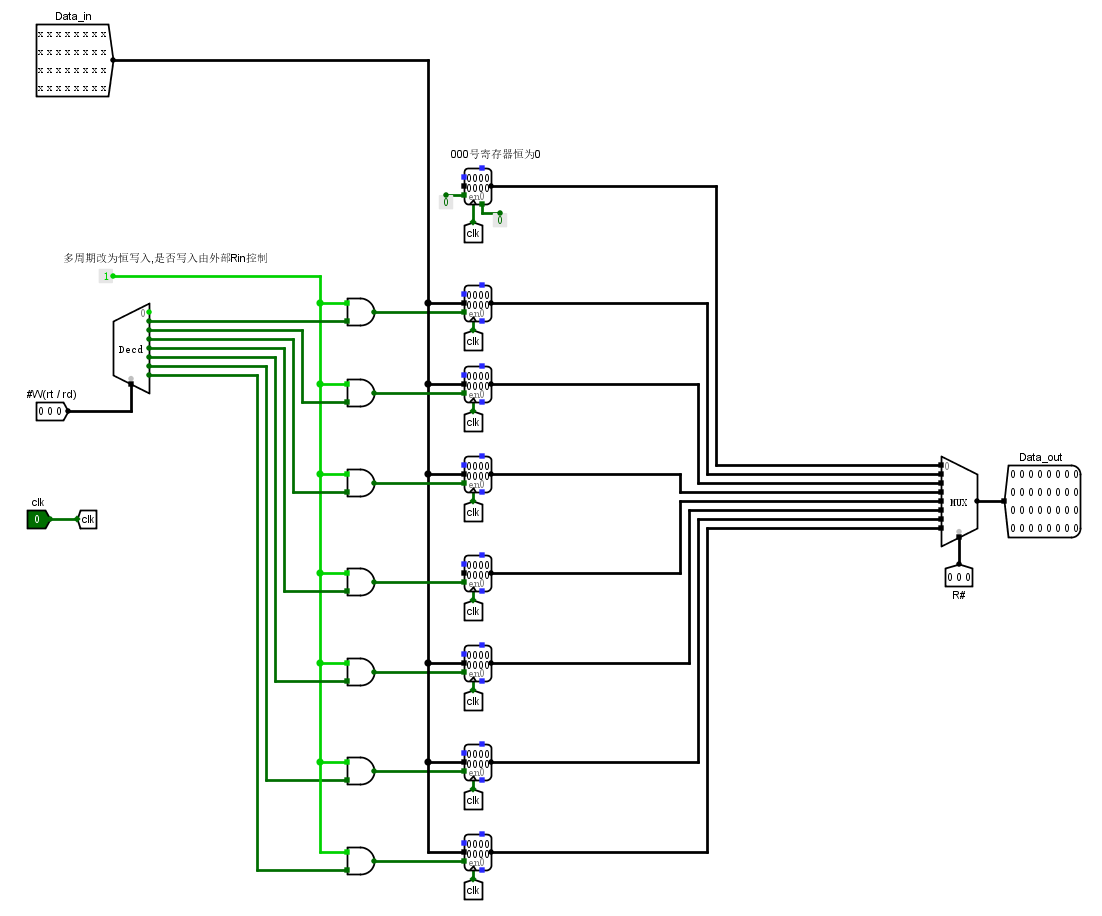


图 2.3 寄存器堆

## 数据通路设计

### 单总线结构

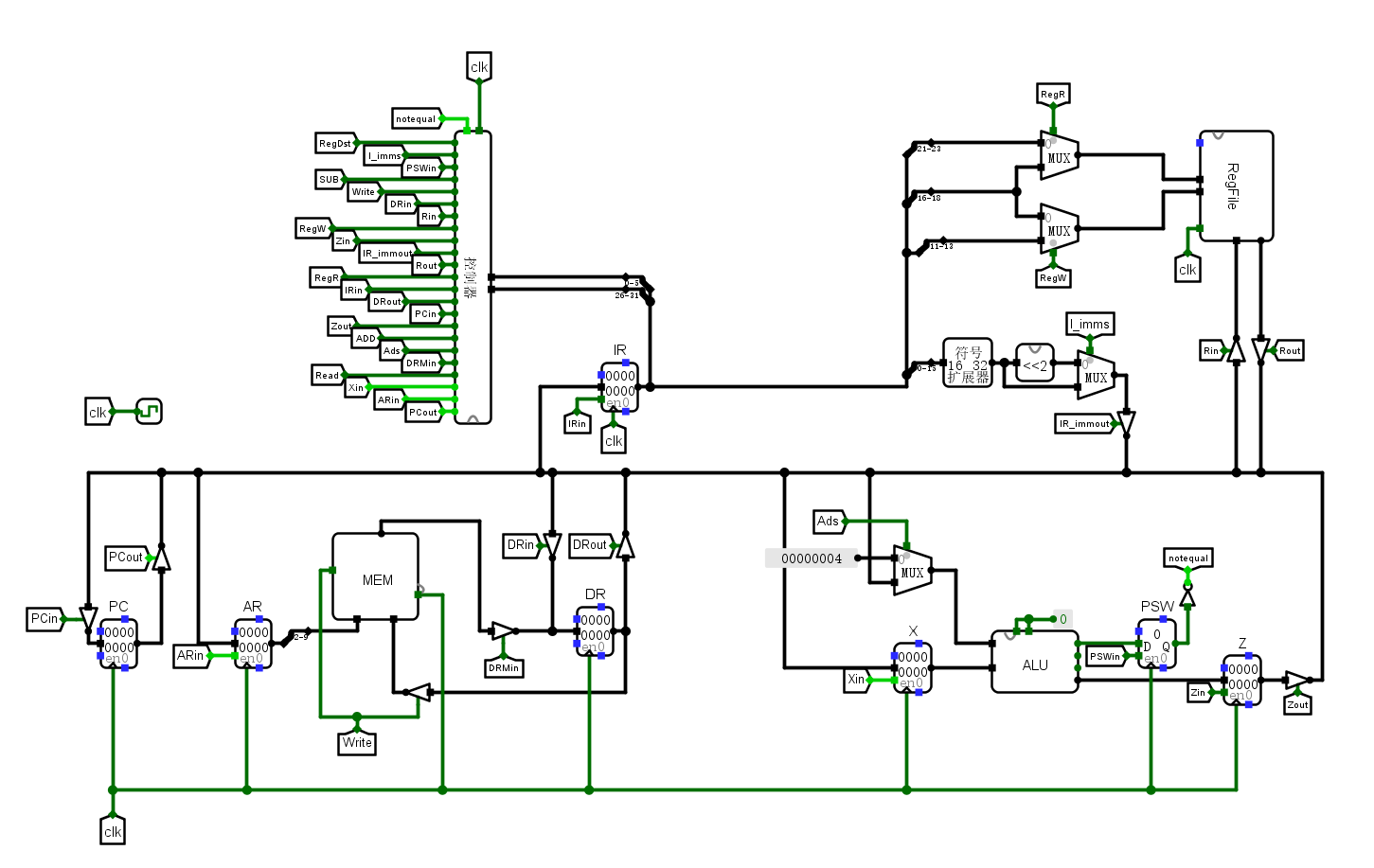


图 2.4.1单总线结构MIPS多周期CPU

### 控制信号与指令流程



表 2.4.3-1取指令阶段的控制信号

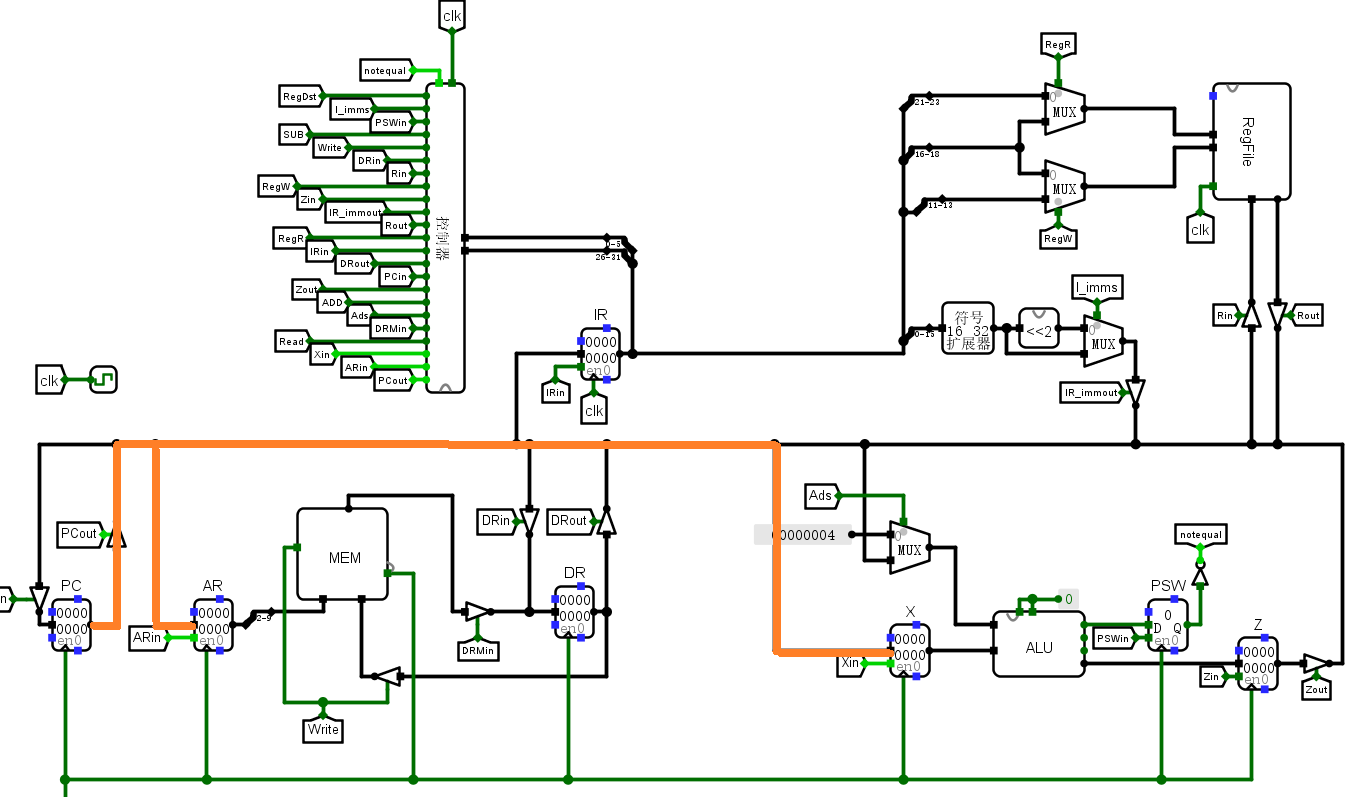
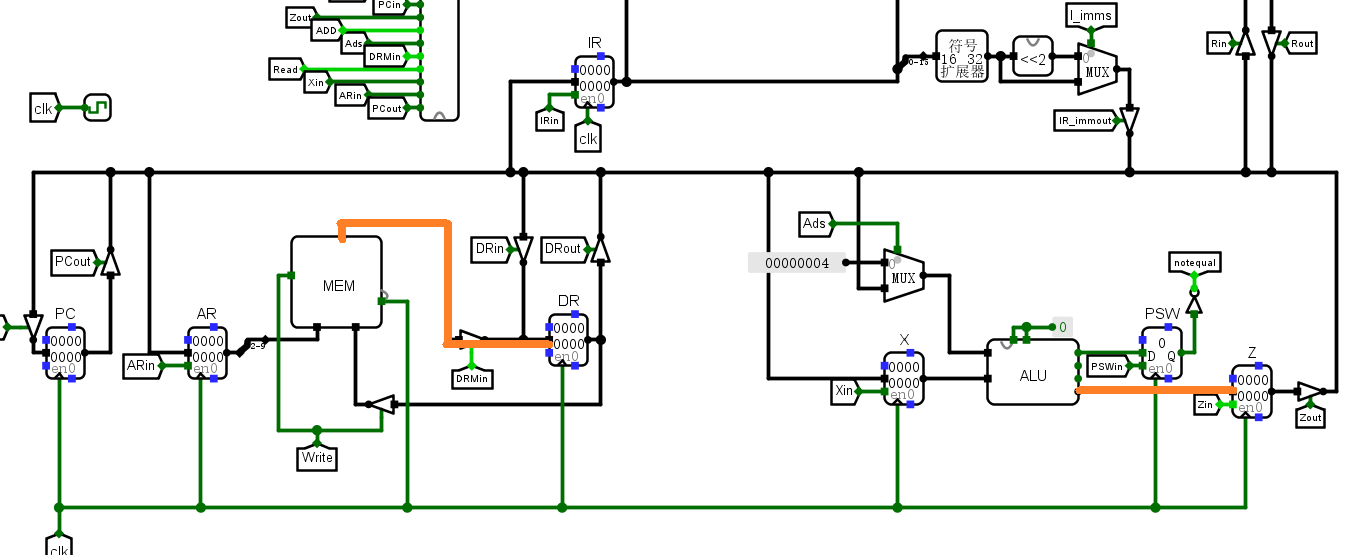


表 2.4.3-2 各指令各个阶段的控制信号

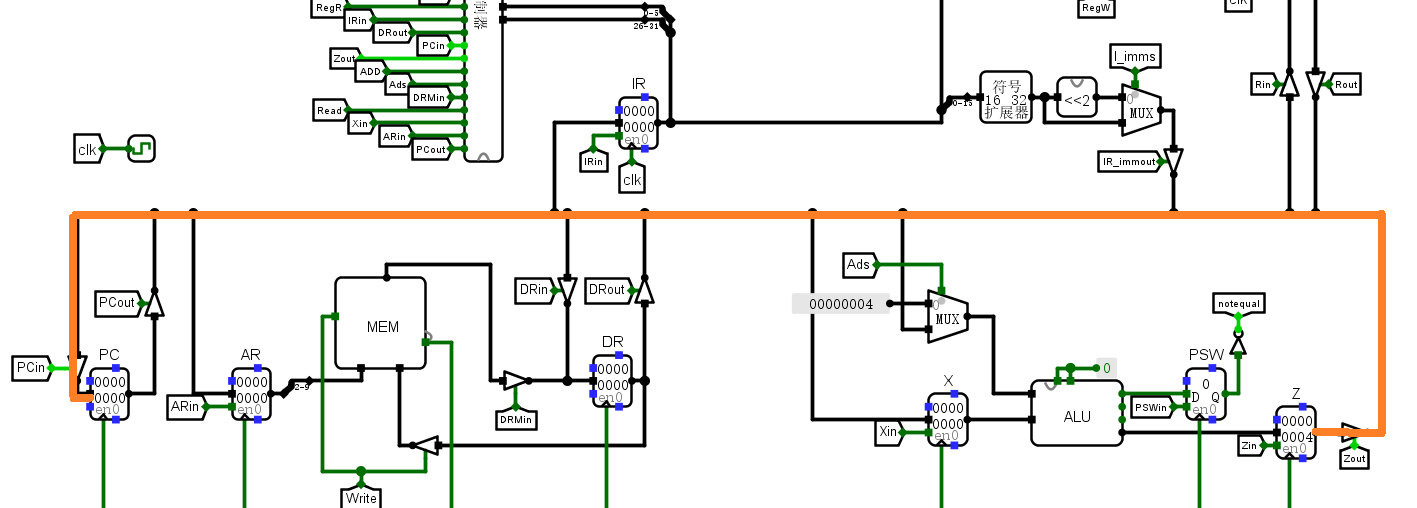
### 指令各个阶段的数据通路

#### 取指令阶段：

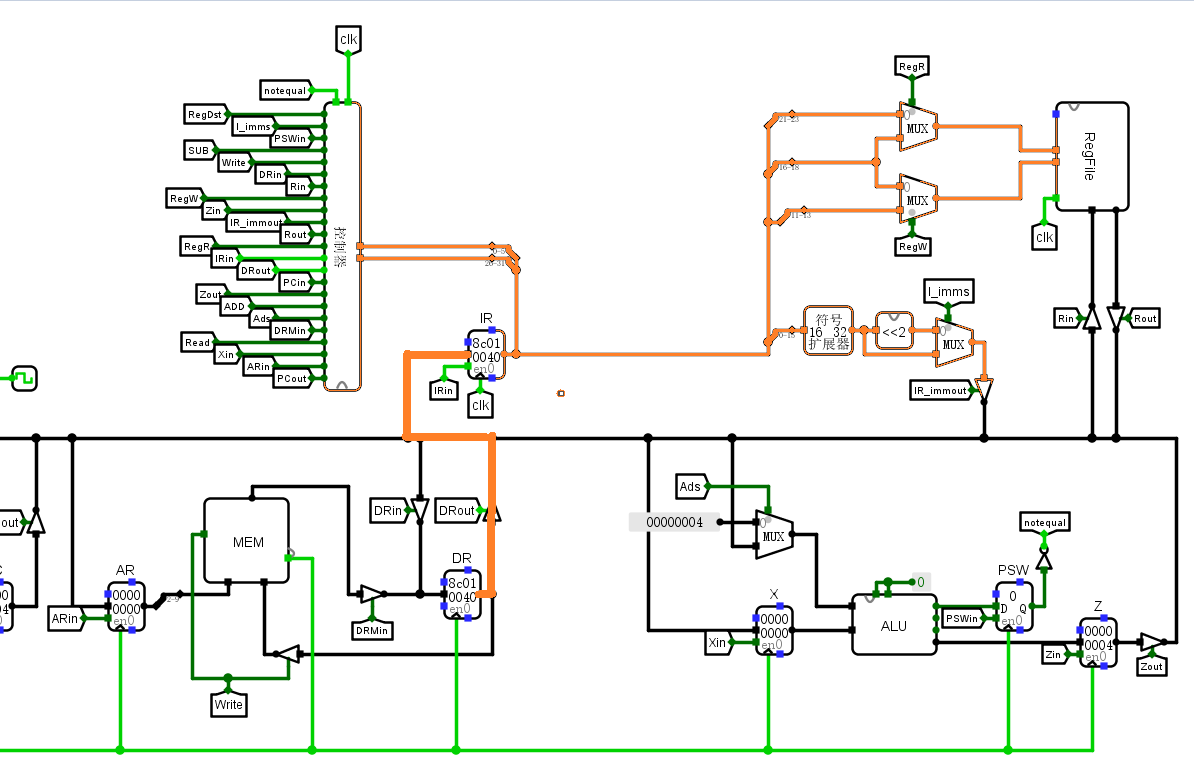
S0：

  
S1：

S2：



S3：

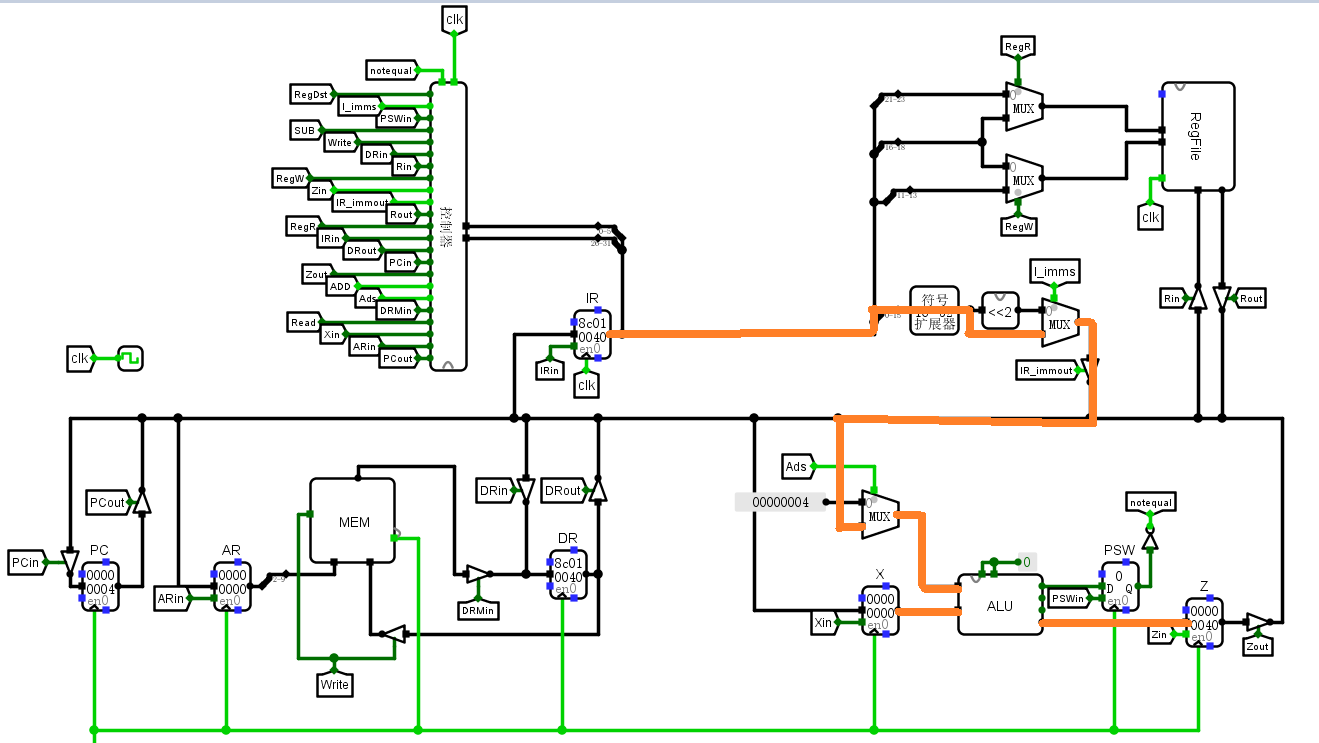


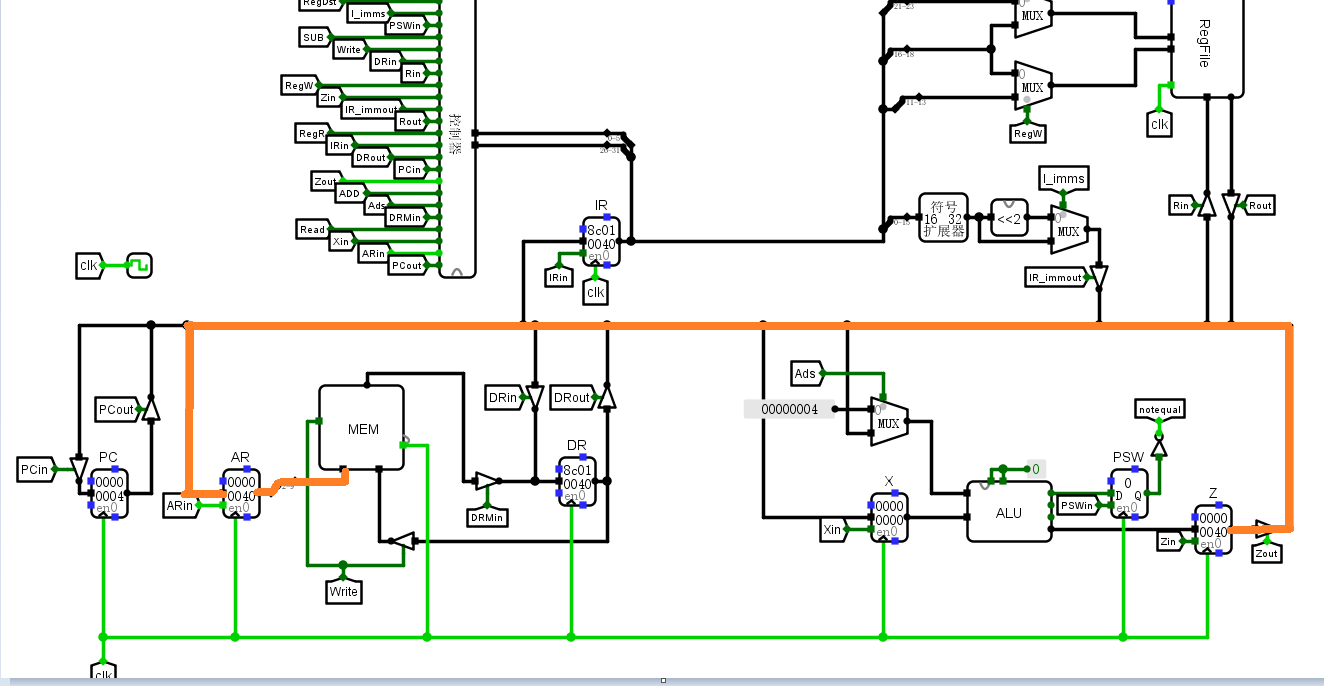
#### LW指令：

S4：



S5：

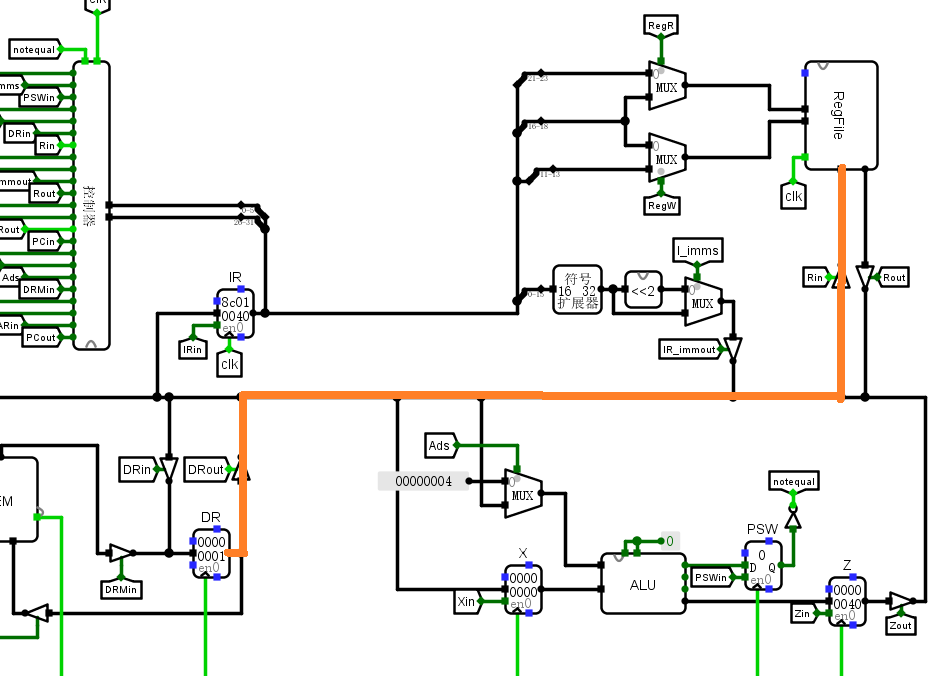


S6:

S7：



S8：

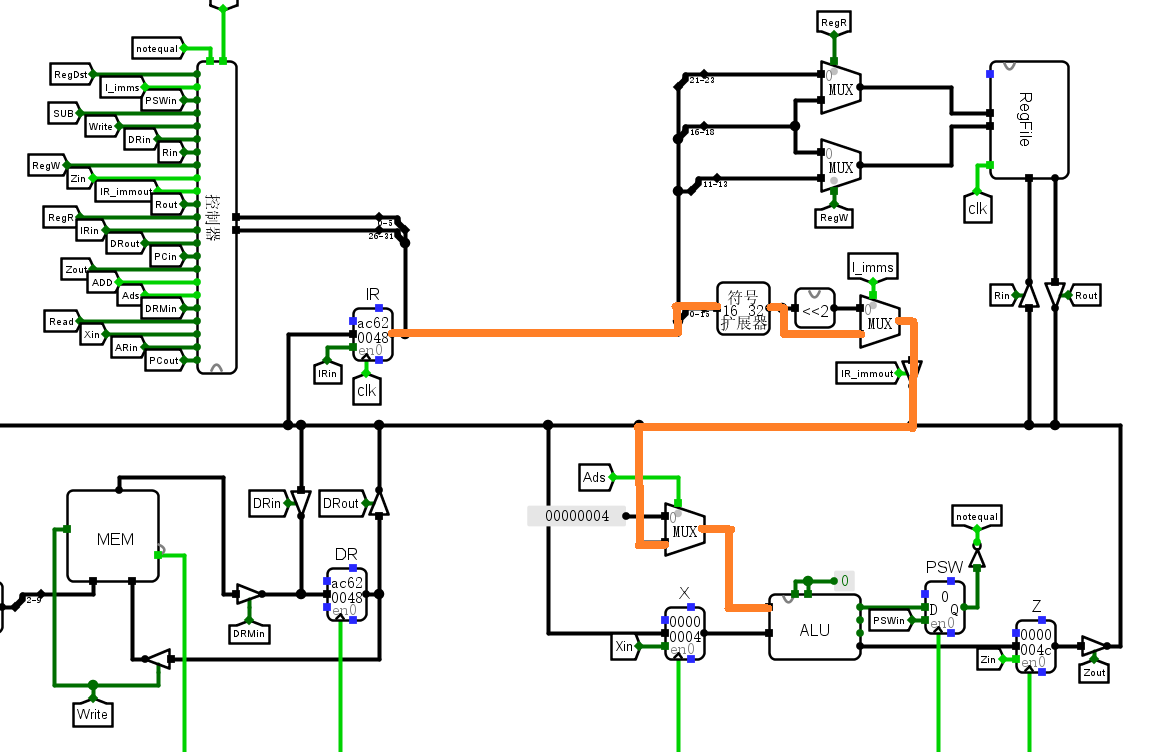


#### 指令SW

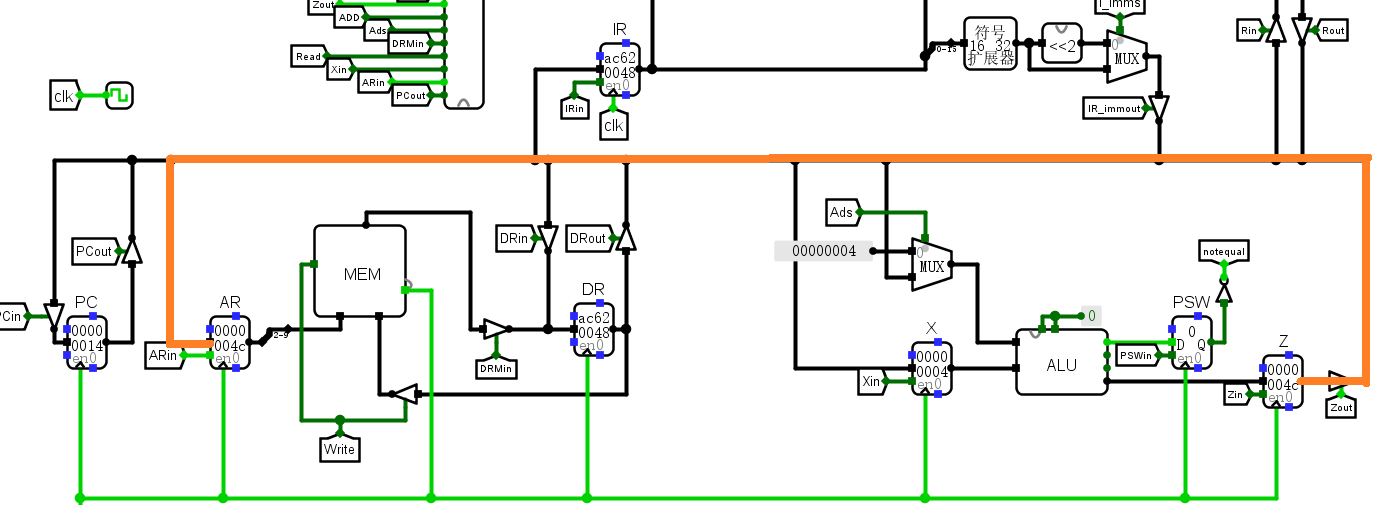
S9：



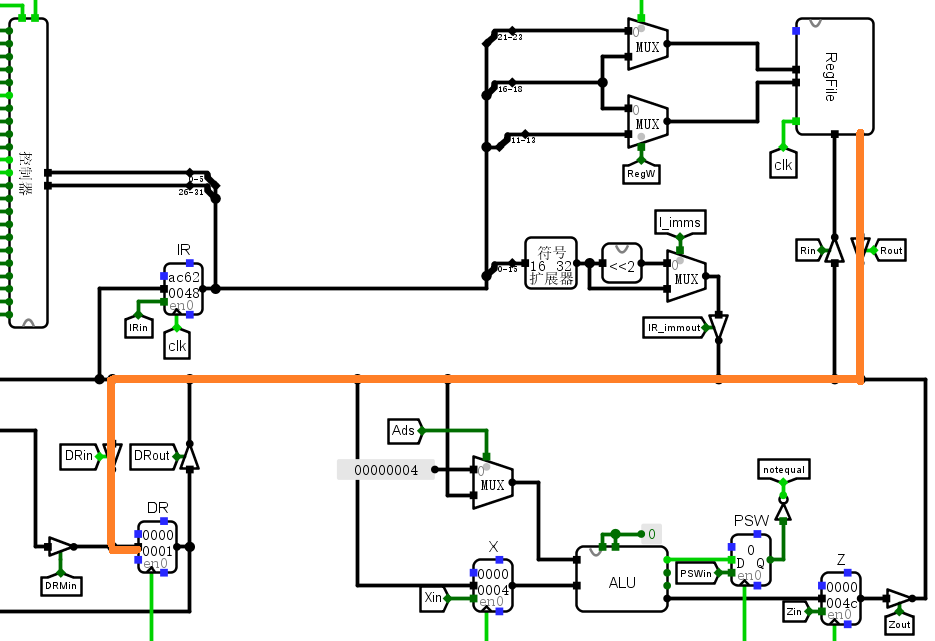
S10：



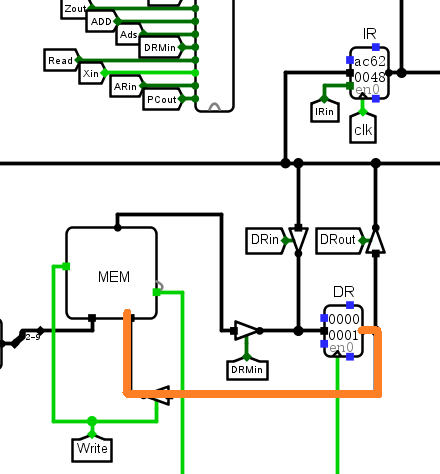
S11：



S12：

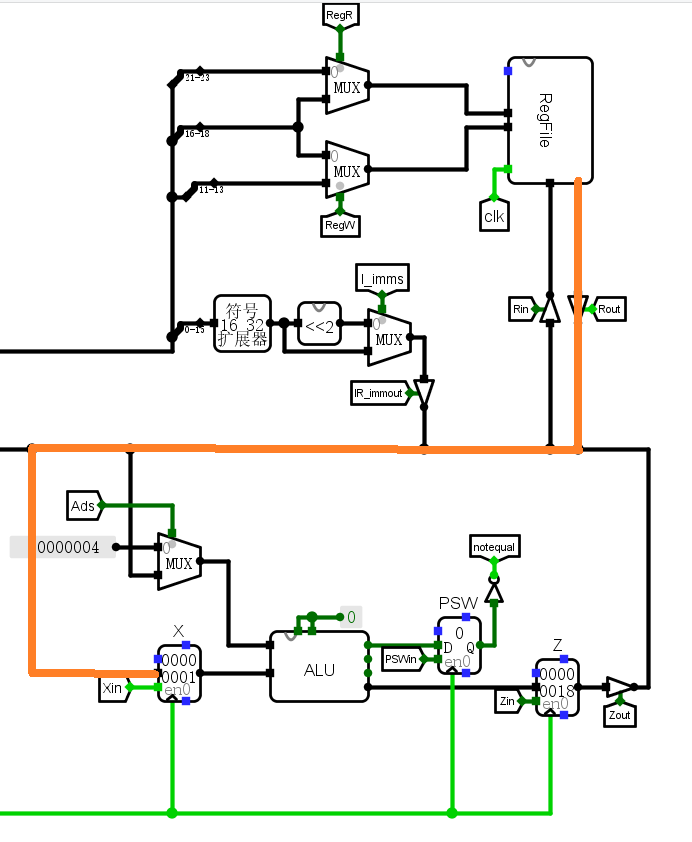


S13：

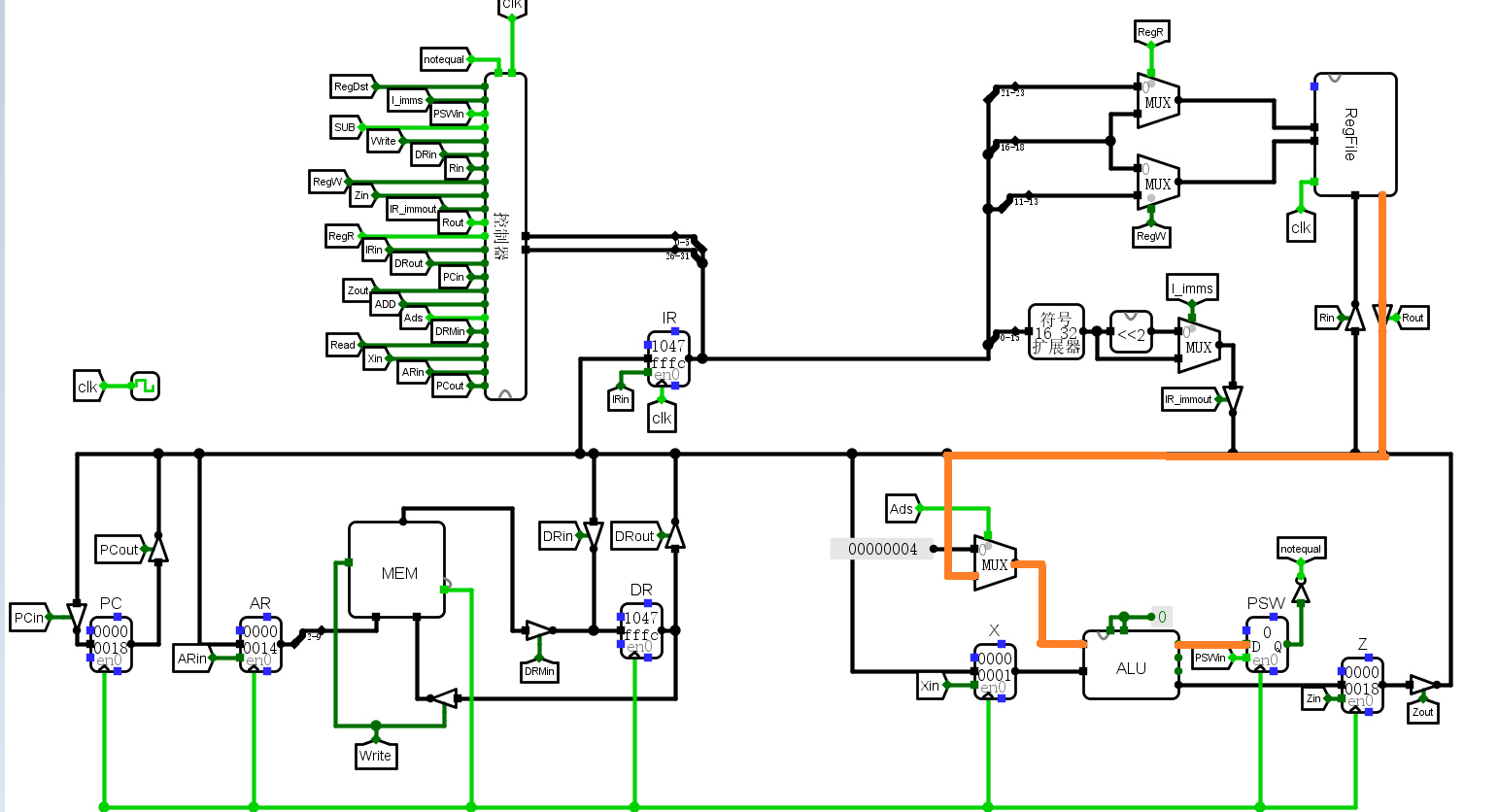


#### 指令BEQ：

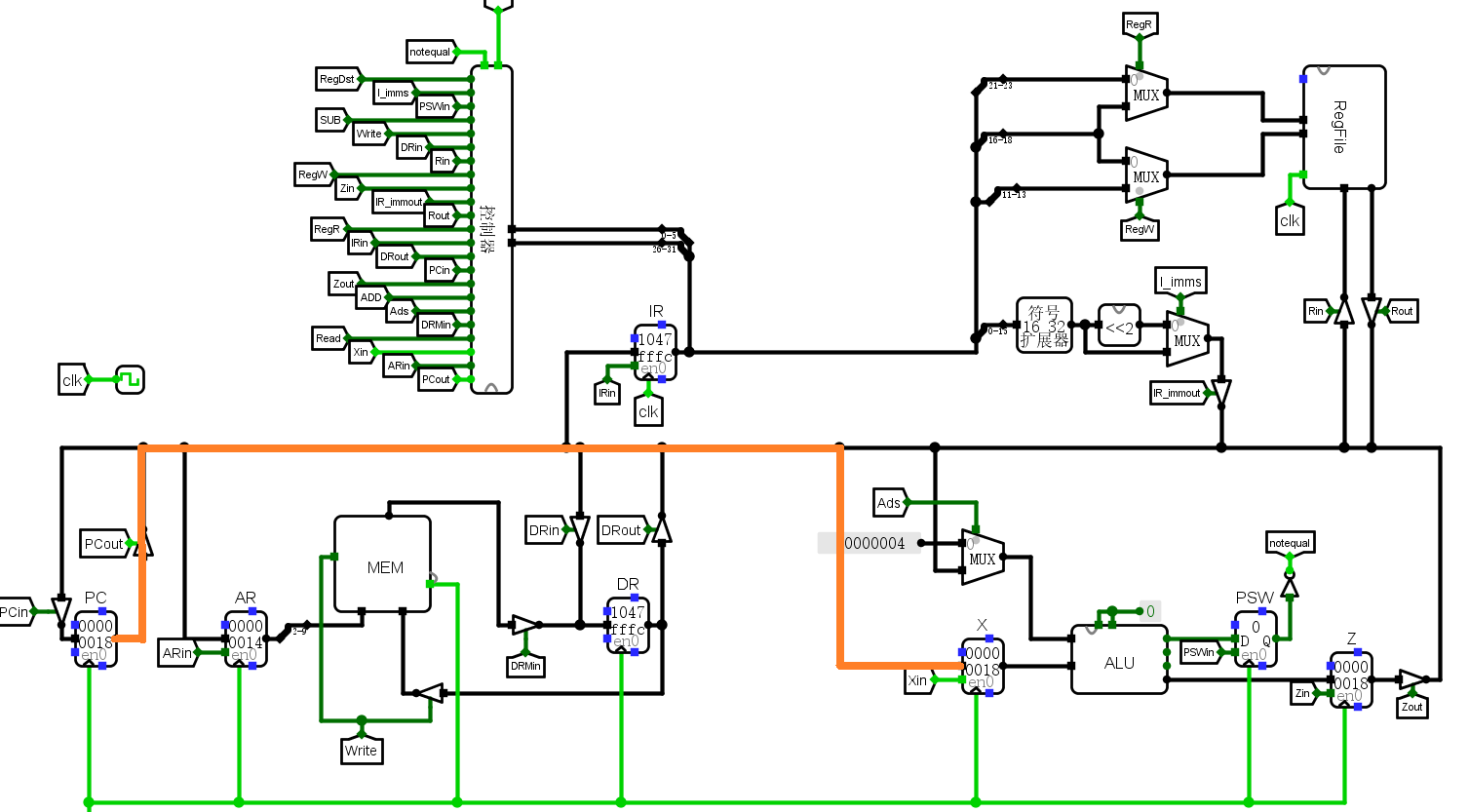
S14：



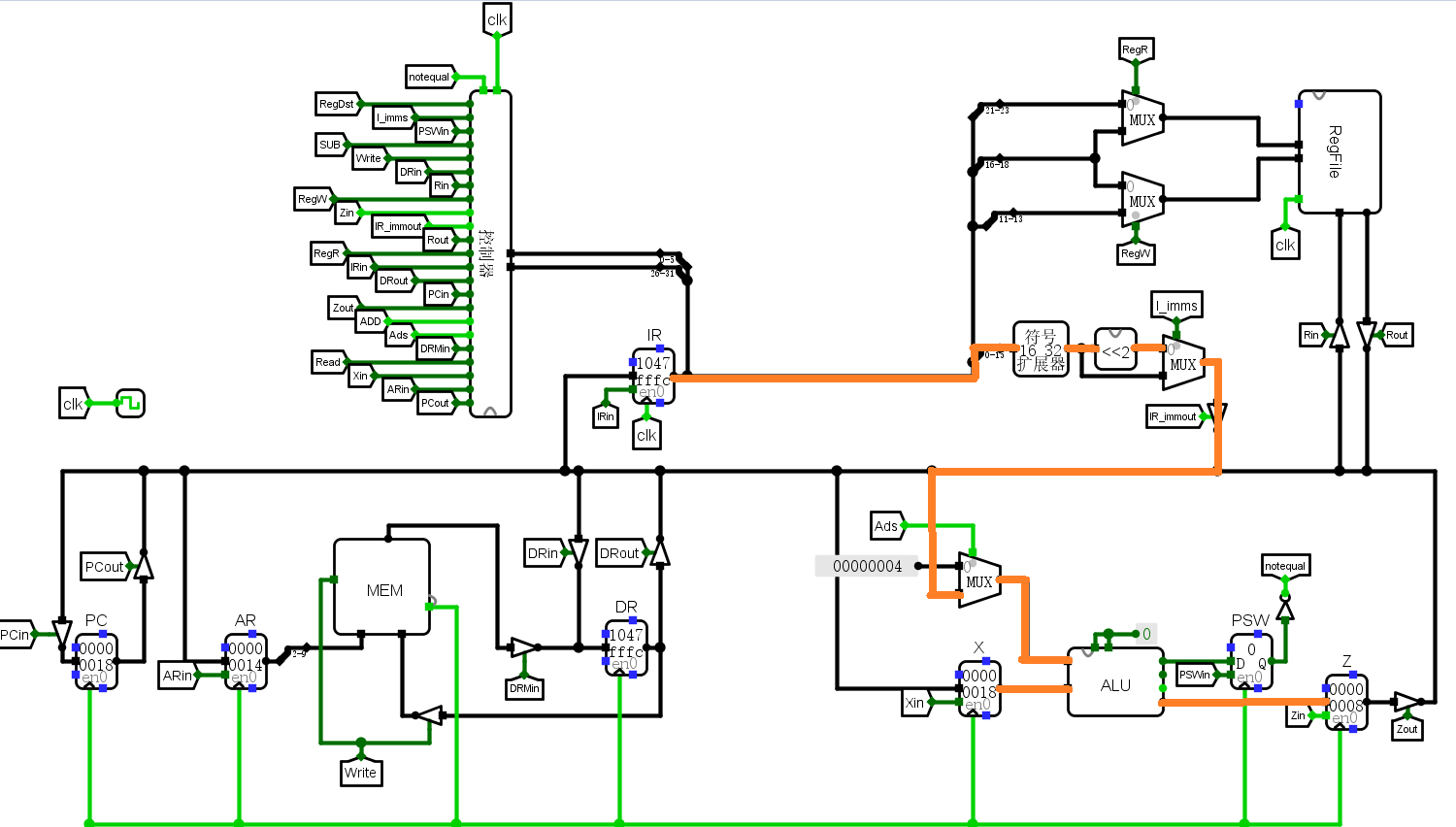
S15：



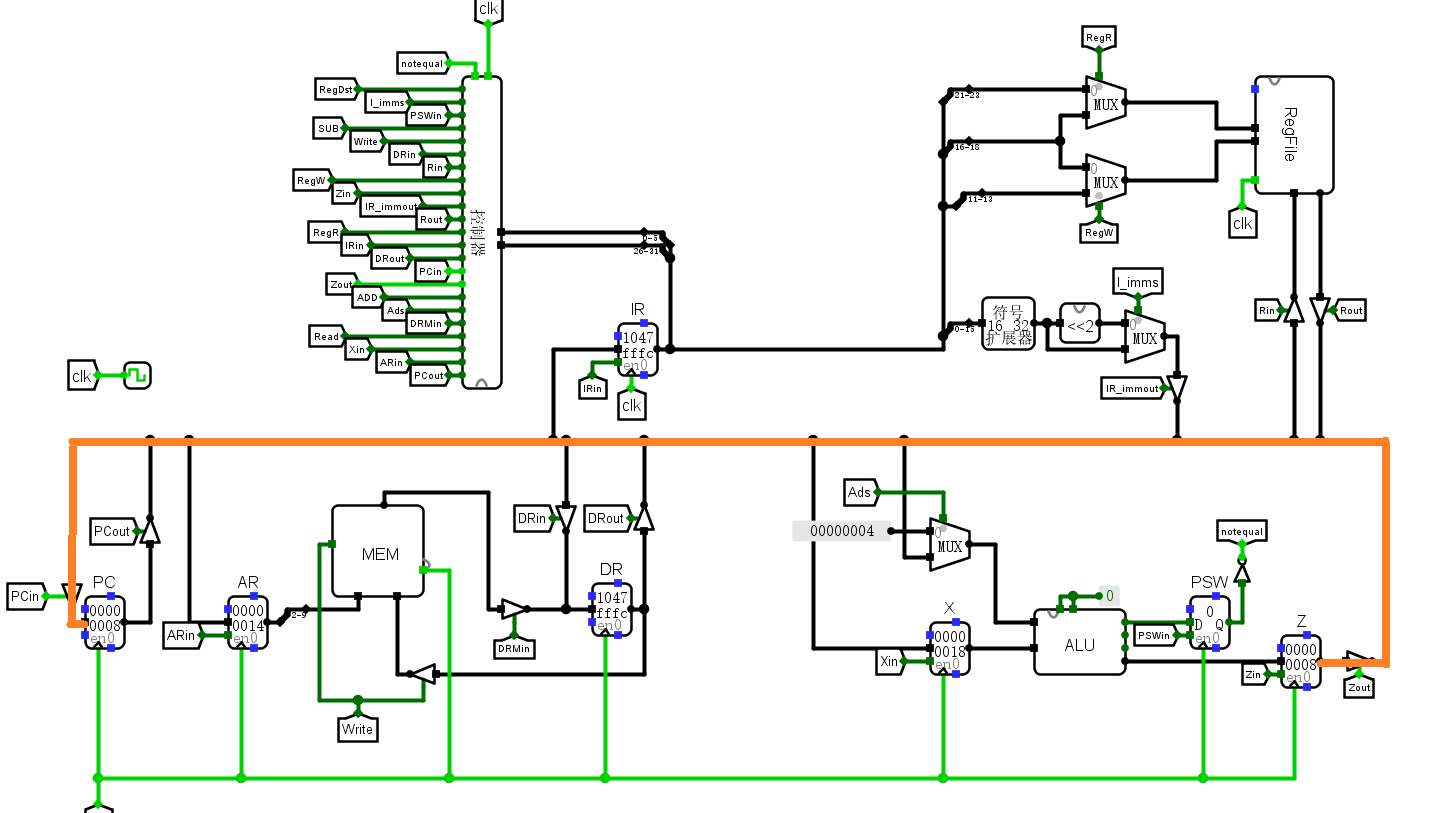
S16：



S17:

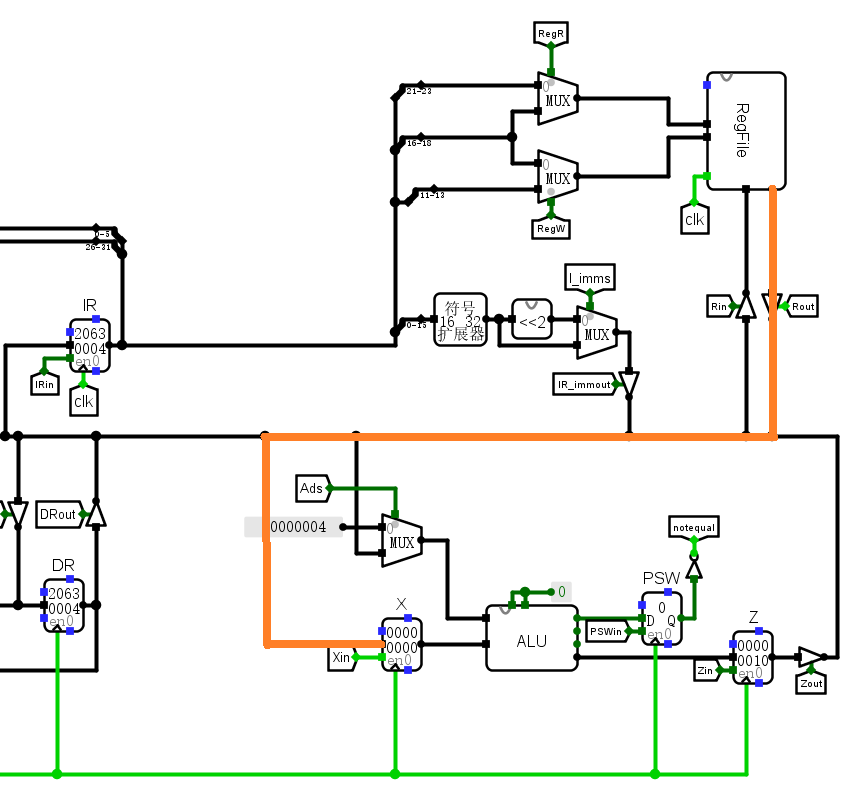


S18：

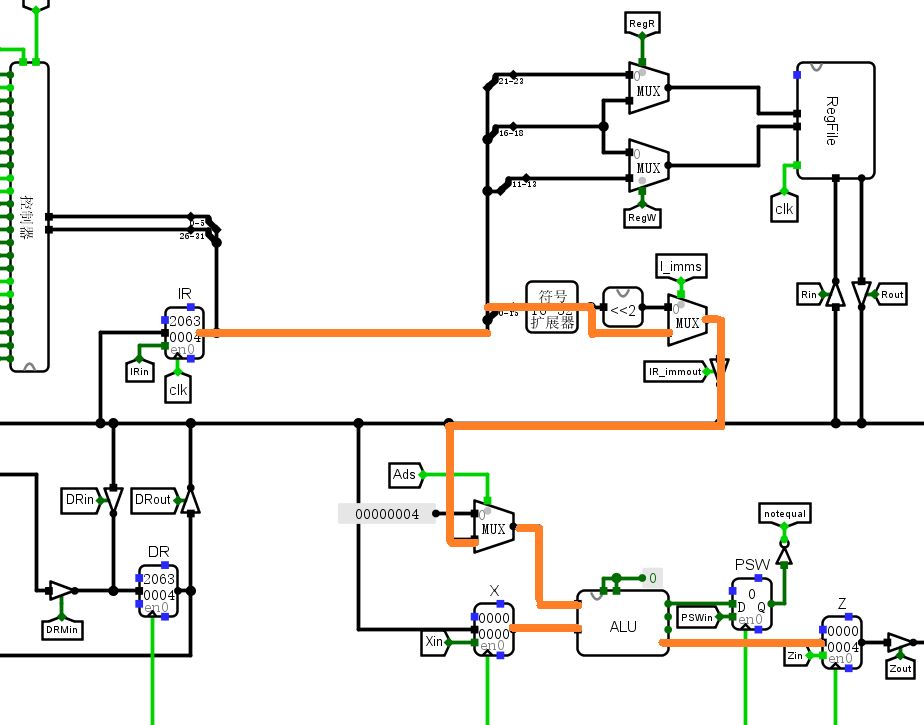


#### 指令ADDi：

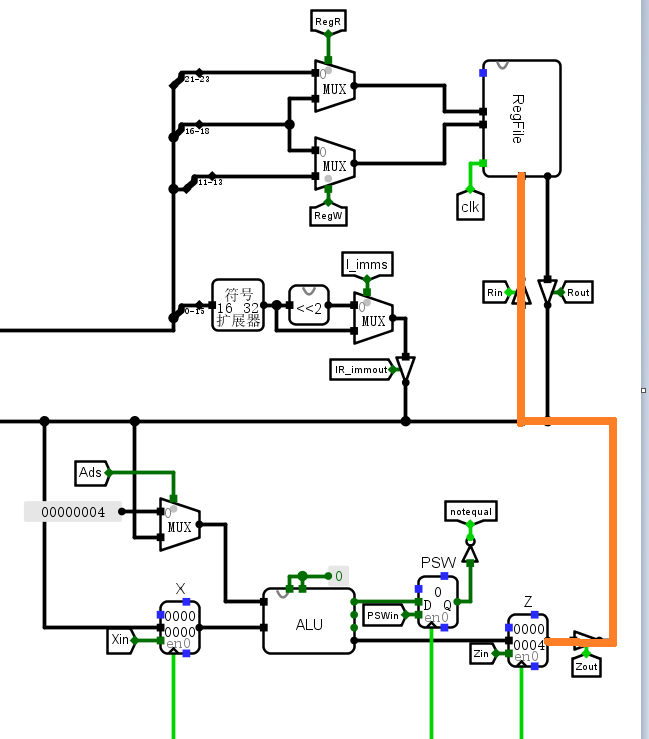
S19：



S20:

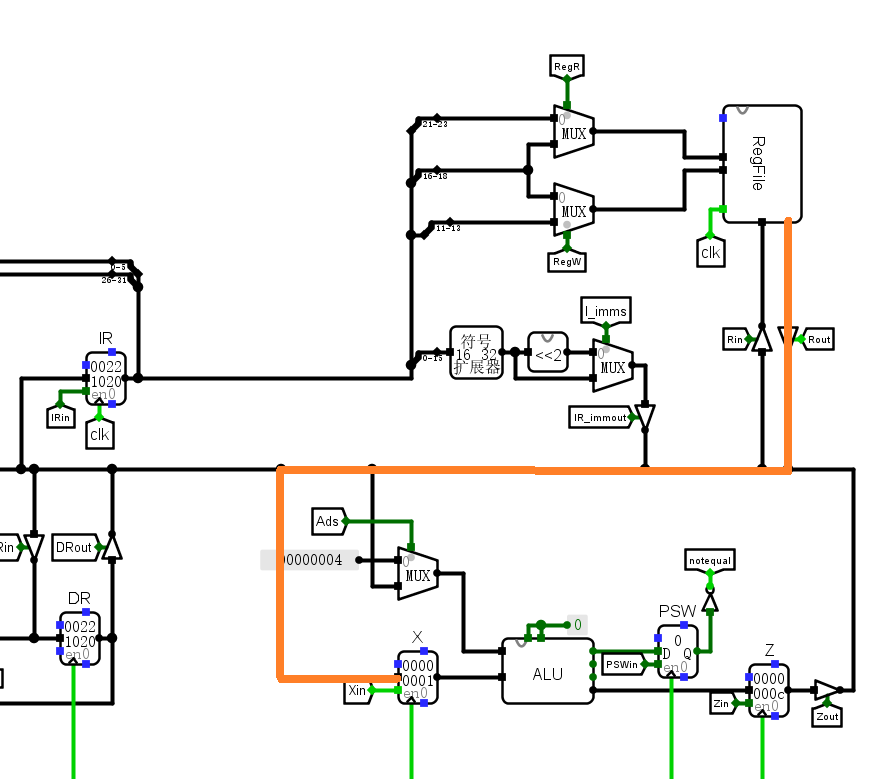


S21:

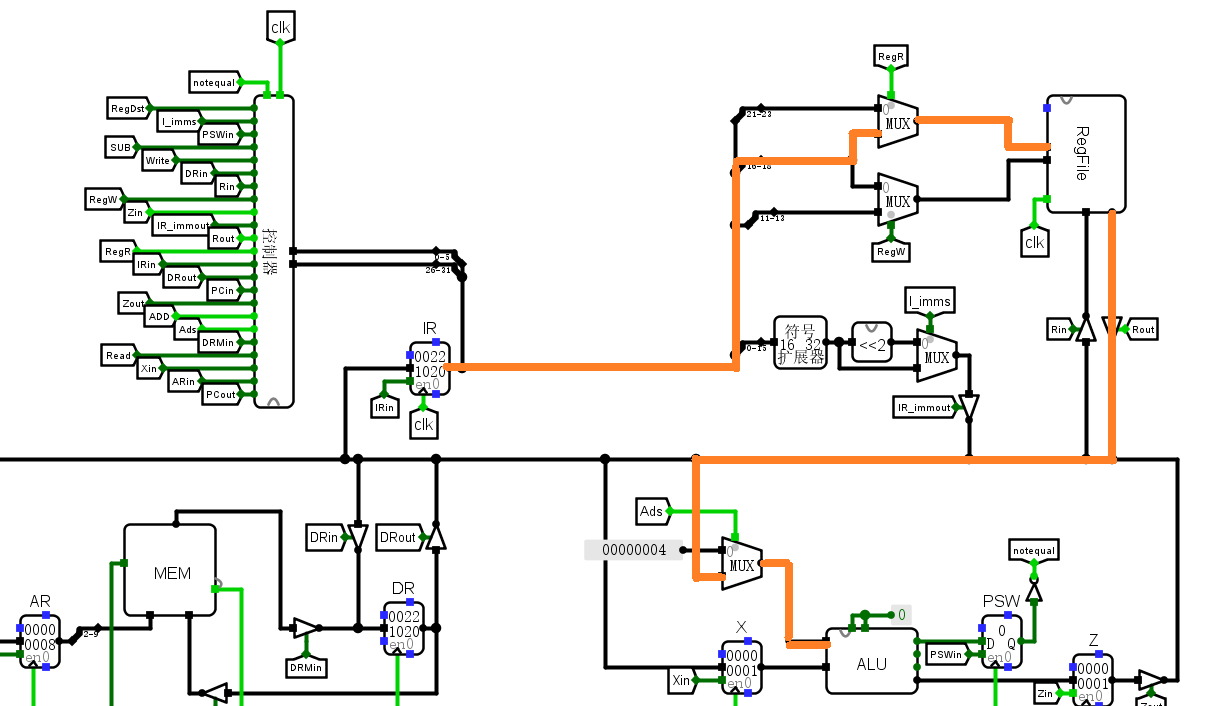


指令ADD：

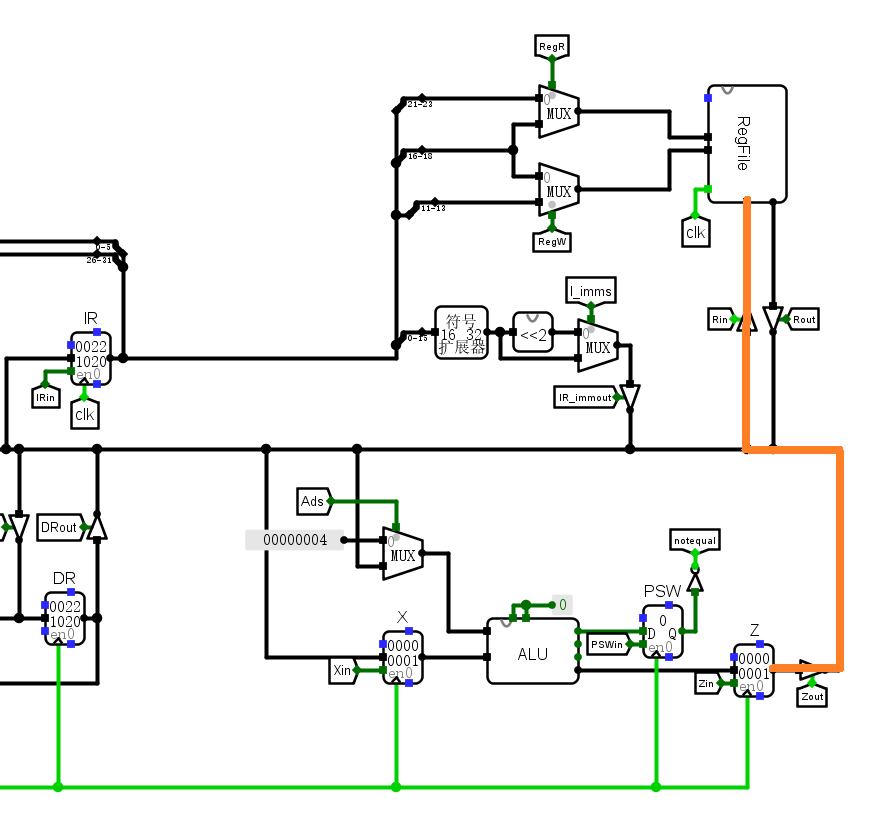
S22：



S23：



S24:



## 控制器设计

### 指令译码：

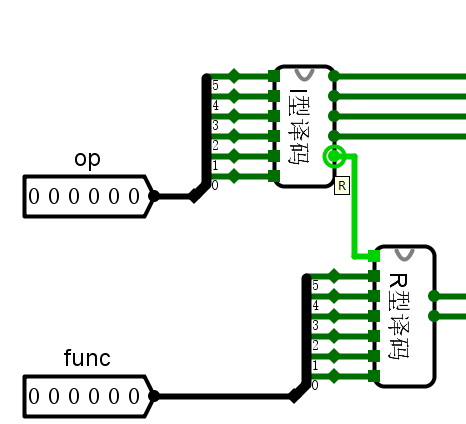


图 2.5.1 指令译码

如果op!=0，即I型指令，通过I型译码输出相应指令；如果OP=0，即为R型指令，R=1，启用R型译码。

两个I型、R型译码电路通过真值表生成。

### FSM状态转移电路：

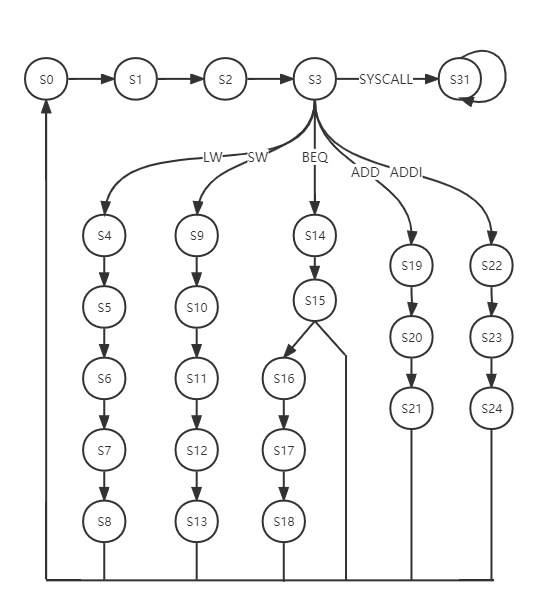


图 2.5.2 状态转移图

根据状态转移图，得到逻辑表达式：

*y4 = syscall + ~x3 ~x2 x1 x0 ~lw ~sw ~beq ~addi add + ~x3 ~x2 x1 x0 ~lw ~sw ~beq addi ~add + ~x4 x3 x2 x1 x0 equal + x4 ~x3 ~x2 ~x1 + x4 ~x3 x1 x0 + x4 ~x3 x2 ~x0*

*y3=syscall + ~x4 ~x3 x1 x0 ~lw ~sw beq ~addi ~add + ~x4 ~x3 x1 x0 ~lw sw ~beq ~addi ~add + ~x3 x2 x1 x0 + ~x4 x3 ~x2 x0 + ~x4 x3 ~x2 x1 + ~x4 x3 x2 ~x0*

*y2=syscall + ~x4 ~x2 x1 x0 ~lw ~sw ~beq ~addi add + ~x4 ~x2 x1 x0 ~lw ~sw beq ~addi ~add + ~x4 ~x2 x1 x0 lw ~sw ~beq ~addi ~add + ~x4 ~x3 x2 ~x1 + ~x4 x2 ~x0 + ~x3 x2 ~x0 + ~x4 x3 ~x2 x1 x0 + x4 ~x3 ~x2 x1 x0*

*y1=syscall + ~x4 ~x3 ~x1 x0 + ~x4 ~x2 ~x1 x0 + ~x3 ~x2 ~x1 x0 + ~x4 x1 ~x0 + ~x4 ~x3 ~x2 x1 ~lw ~sw ~beq ~addi add + ~x4 ~x3 ~x2 x1 ~lw ~sw ~beq addi ~add + ~x4 ~x3 ~x2 x1 ~lw ~sw beq ~addi ~add + ~x3 x2 x1 ~x0*

*y0=~x3 ~x1 ~x0 + syscall + ~x4 x1 ~x0 + ~x4 ~x3 ~x2 x1 ~lw ~sw ~beq addi ~add + ~x4 ~x3 ~x2 x1 ~lw sw ~beq ~addi ~add + ~x4 x2 ~x0 + ~x3 x2 ~x0*

利用上述逻辑函数表达式，在Logisim中生成状态转移电路。

### 控制信号输出电路：

根据2.4.3中的状态-控制信号对应表。得到控制信号输出电路的真值表如下图所示：

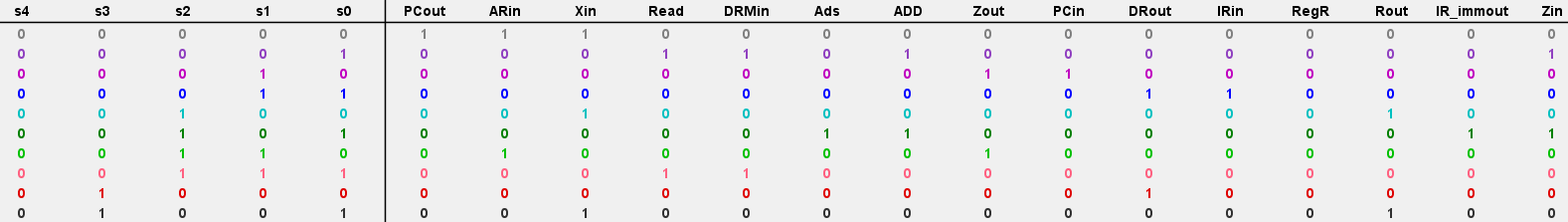


图 2.5.3 控制信号输出电路的真值表（部分）

由真值表生成电路，这样实现了由状态到具体的控制新号的一个转换。作为控制器的输出。

### 控制器

将上述三部分组合成最终的控制器如下图所示：

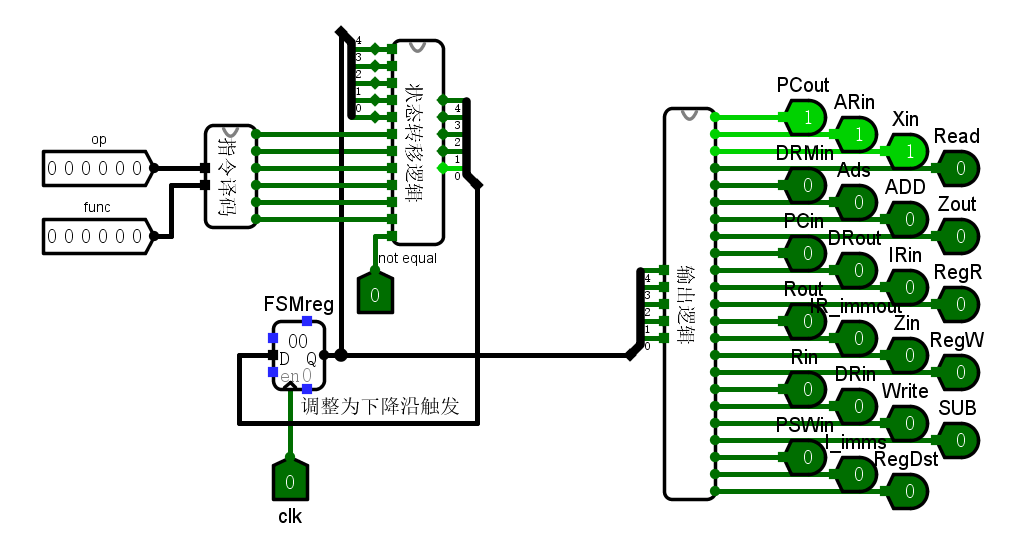


图 2.5.4 控制器

# 实验过程与调试

## 测试用例设计及测试结果

编写指令：

100011 00000 00001 0000 0000 0100 0000 #Mem(0+1) -> reg1 i

100011 00000 00111 0000 0000 0100 0100 #Mem(0+2) -> reg7 9

000000 00001 00010 00010 00000 100000 #reg1+reg2 -> reg2

001000 00011 00011 00000 00000 000100 #reg3+4 -> reg3

101011 00011 00010 0000 0000 0100 1000 #reg2 -> Mem[reg3+48]

000100 00010 00111 1111 1111 1111 1100 #reg2!=reg7? PC-3

00000000000000000000001010001100 #停机

指令功能：从主存中的数据段读取第一个数据（1）和第二个数据（9），接下来，依次向后面的单元写入1、2、3…9。停机。

接下来将上述指令转换为16进制，加载进主存。实验所用16进制代码文件：

v2.0 raw

8c010040

8c070044

221020

20630004

ac620048

1047fffc

28c

9\*0

1

9

## 设计中遇到的主要问题

### 两个16位加法器的合并

后续突然发现这个CLA74182多余的。因为只有两片，直接传递进位位就可以了，通过CLA74182再连接只会起反作用。

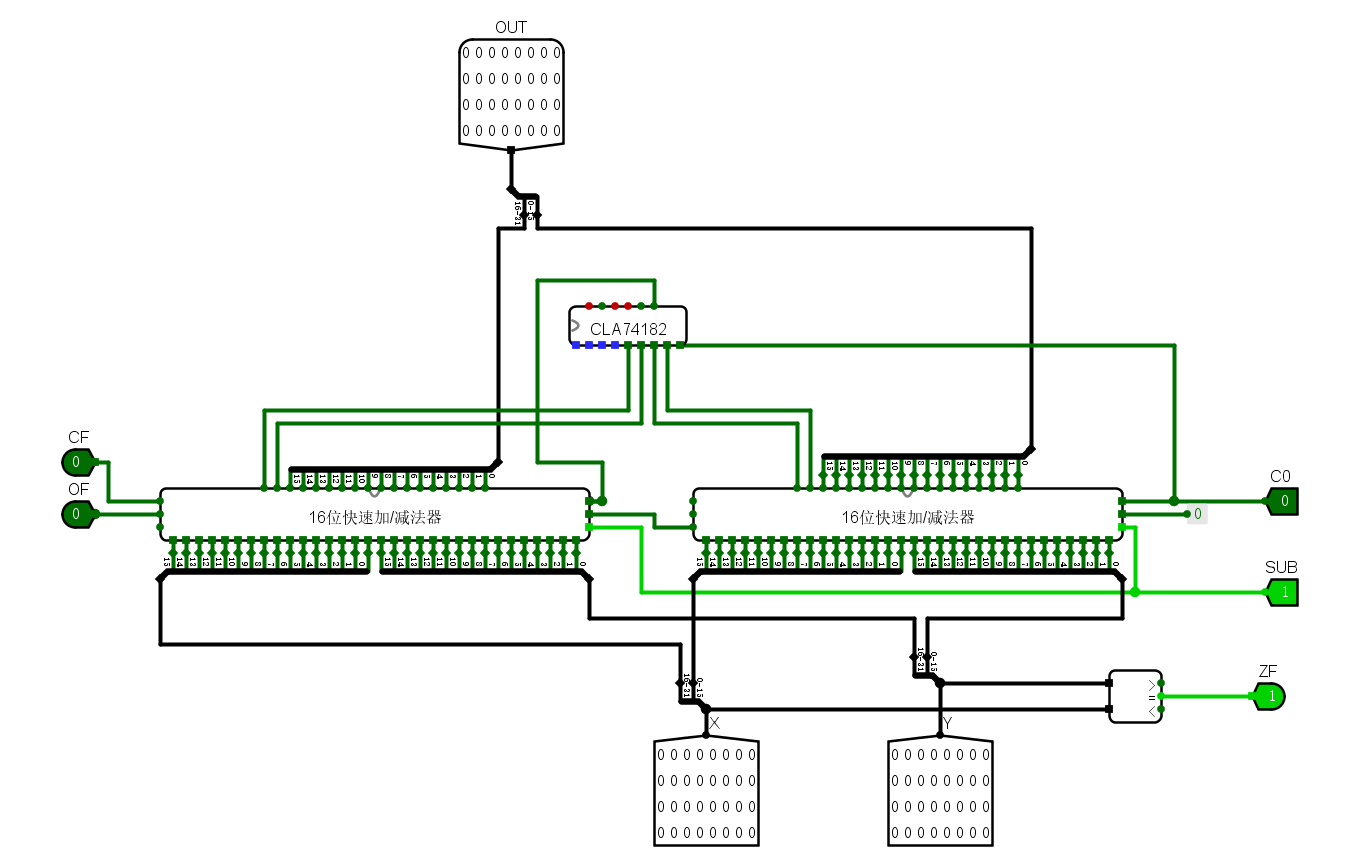


图 3.2.1两个16位加法器的合并(更改前)

删除，让低16位直接传进位给高16位。

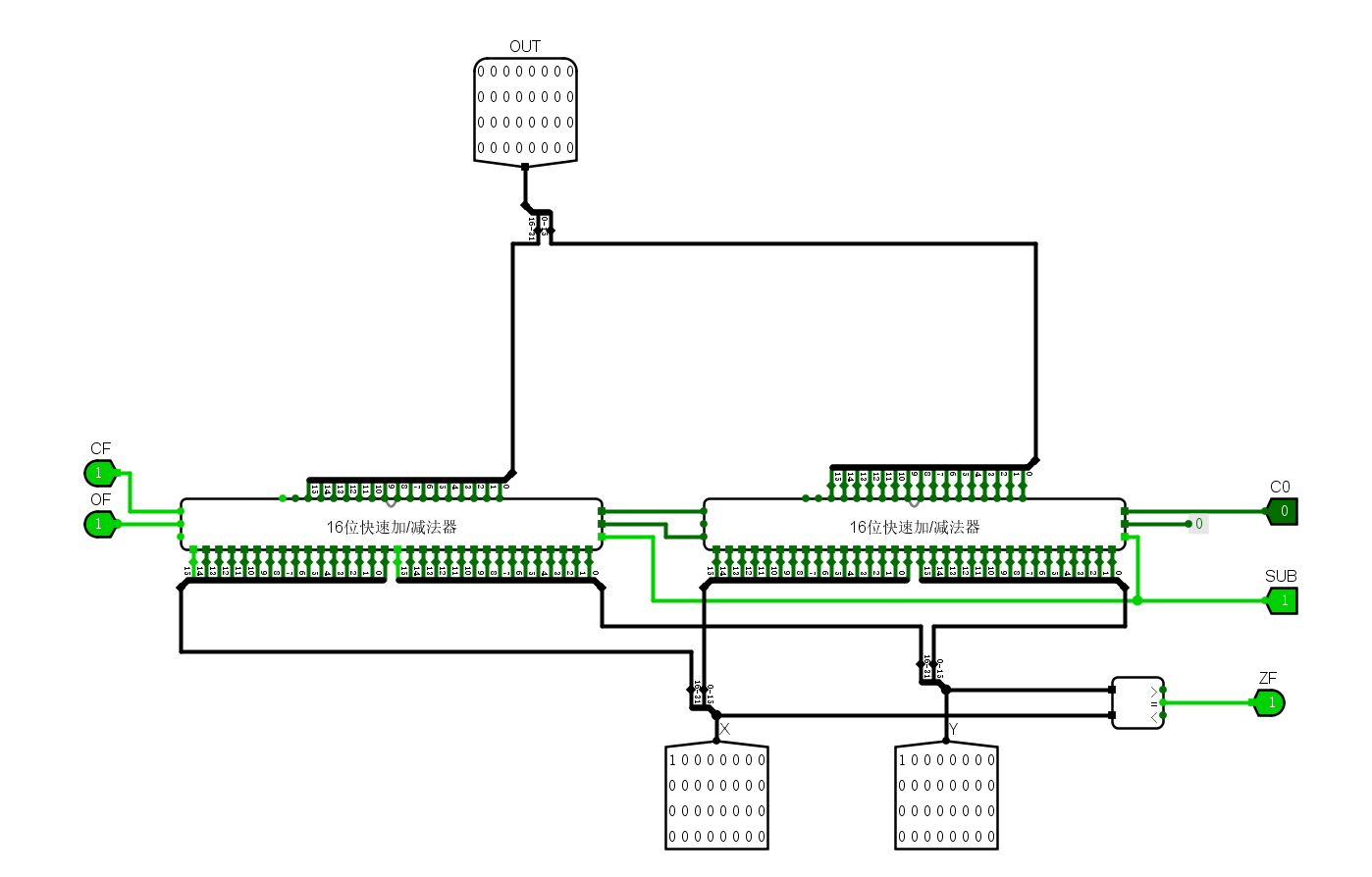


图 3.2.2 两个16位加法器的合并(更改后)

### 发现加法器中存在一个隐患

ALU中溢出检测存在一个隐患，如上图 3.2.2中所示数据， 在边界值有这样一个特殊情况，会是的OF为1。

通过汇编代码验证，这种情况在正常的计算机中标志位OF是不会变1的。这是由溢出检测时没有考虑到0的情况造成的。但是经过检查，后续代码不会涉及到这个特殊情况，所以仅作记录，没做改动。

### 控制器错误

忘记将状态机改为下降沿触发了，修改控制器状态机中的状态寄存器为下降沿触发。时钟下降沿时状态更新，其余所有寄存器均为时钟上升沿触发。

这样每次时钟下降沿时，各个控制信号已经准备好；等到上升沿时，数据进行传递，功能正常。

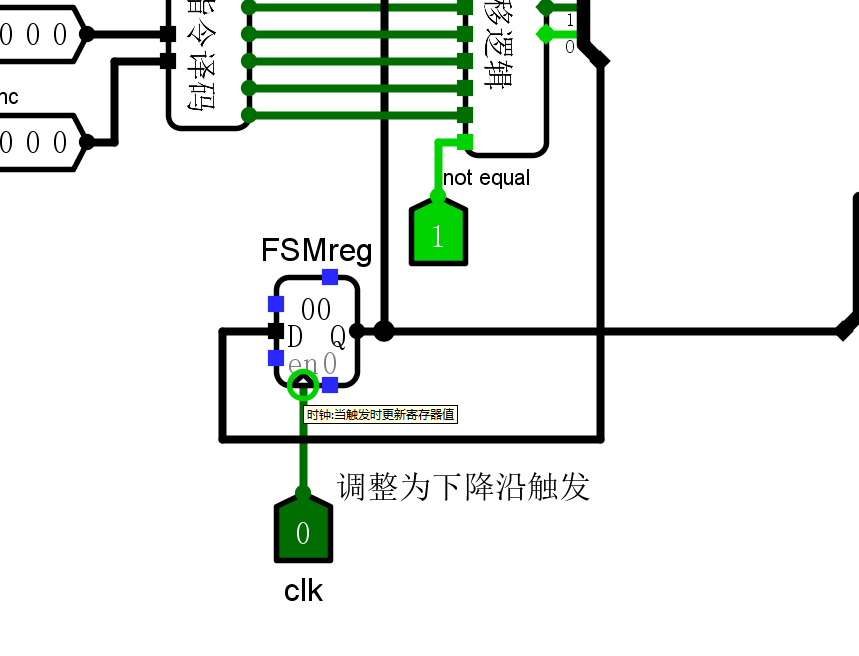


图 3.2.3 调整状态寄存器为时钟下降沿触发

# 实验总结与心得

## 实验总结

实验情况良好。在完成专用通路的单周期MIPSCPU后，又尝试了单总线结构的MIPSCPU，均使用了在之前实验中自己手工制作的功能部件。完成了一些基本的MIPS指令，编写了比较简单的小程序。体会到了CPU的工作原理。进一步理解了计算机组成原理。本实验是在参考老师PPT，和同学们讨论的基础上自己动手完成的。

## 实验心得

通过实验可以更直观地感受CPU的整体结构。而且自己实现各个功能部件有一种“万丈高楼平地起”的成就感。

## 问题回答

问题1：使用Logisim仿真工具进行本实验的优点与不足

Logisim仿真工具简单直观，易上手，拖动就可以布置电路。但自然地，它的功能也相对有限，不适用与复杂电路，某些部件的可选属性很少，不支持高度自定义。也有一些细节不够人性化，有时，在用真值表生成逻辑电路时，只能在五颜六色的表格里一个个点击，不方便而且容易出错。但是总的来说，用Logisim来完成规模不大的实验还是很适用的。

# 参考文献

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

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: (可使用电子签名)** |