华中科技大学

2024

硬件综合训练　　课程设计报告

题　　目：	5 段流水 CPU 设计
专　　业：	计算机科学与技术
班　　级：	计卓 2201
学　　号：	U202215335
姓　　名：	张璐晟
电　　话：	18815880189
邮　　件：	1002682533@qq.com

目 录

1 课程设计概述

1.1 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以 CPU 设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、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。

1.2 设计任务

本课程设计的总体目标是利用 FPGA 以及相关外围器件，设计五段流水 CPU，要求所设计的流水 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的功能，对主要的数据流和控制流通过 LED、数码管等适时的进行显示，方便监控和调试。尽可能利用 EDA 软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

（4） 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10） 支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（12） 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。

（13） 能运行教师提供的标准测试程序，并自动统计执行周期数

（14） 能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

#	指令助记符	简单功能描述	备注
1	ADD	加法	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
2	ADDI	立即数加	
3	ADDIU	无符号立即数加	
4	ADDU	无符号数加	
5	AND	与	
6	ANDI	立即数与	
7	SLL	逻辑左移	
8	SRA	算数右移	
9	SRL	逻辑右移	
10	SUb	减	
11	OR	或	
12	ORI	立即数或	

#	指令助记符	简单功能描述	备注
13	NOR	或非	
14	LW	加载字	
15	SW	存字	
16	BEQ	相等跳转	
17	BNE	不相等跳转	
18	SLT	小于置数	
19	STI	小于立即数置数	
20	SLTU	小于无符号数置数	
21	J	无条件转移	
22	JAL	转移并链接	
23	JR	转移到指定寄存器	If $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$a0 值
25	MFC0	访问 CP0	中断相关，可简化，选做
26	MTC0	访问 CP0	中断相关，可简化，选做
27	ERET	中断返回	异常返回，选做
28	SB	从特定位置取一个字节到内存	
29	AUIPC	取立即数加载至 pc	
30	SLTIU	比较大小	
31	BLTU	无符号比较小于跳转	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 本次采用的是硬布线控制方案，将指令和数据分开存储，使用硬布线将二者连接起来。本次设计采用同步时序逻辑电路，CPU 内元件均听从同一时钟周期的命令。在一个周期内，控制器读取指令，并将其转化，得到 ALU 所需指令，通过指令其余部分获得对应的目的寄存器或立即数，将得到的数字传入 ALU，计算后写入内存或寄存器。

总体结构图如图 2.1 所示。

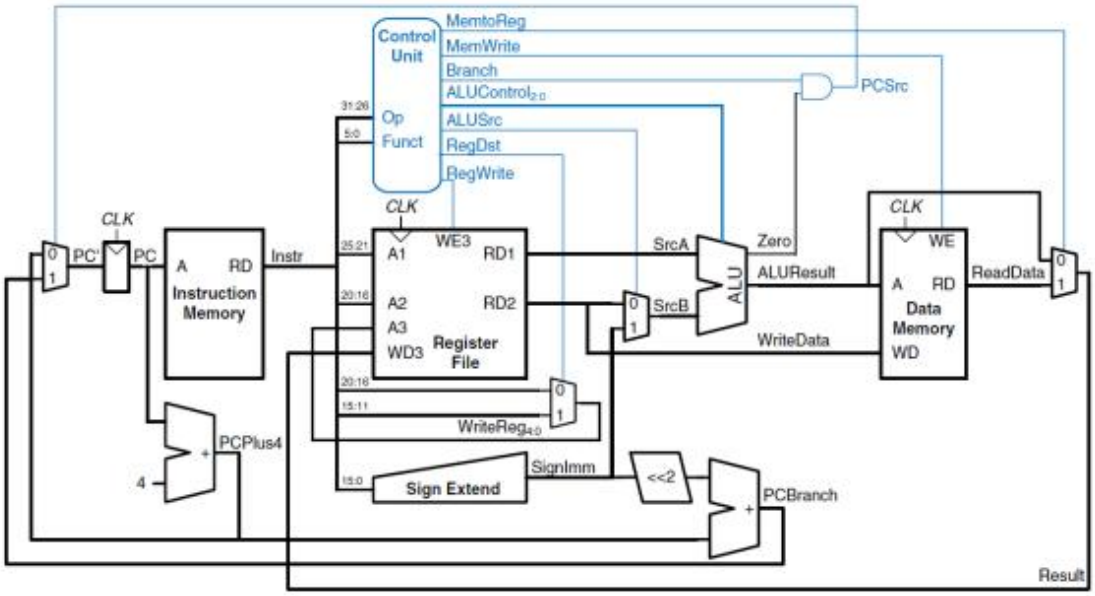

图 2.1 总体结构图

2.1.1 主要功能部件

运算器部分，具体设计思路如下。

1. 程序计数器 PC

程序计数器是一个寄存器，其作用是用于保存下一条指令的地址。其输入从多种可能的下条指令的地址中选取，经时钟周期后，输出下条指令的地址到指令存储

器中，并根据输入情况更新数据。

2. 指令存储器 IM

指令存储器中存储了计算机所需的所有指令，其执行指令地址由 PC 的输出决定。由于 RISC-V 为 4 字节的定长指令，所以 IM 所取指令的地址仅由 PC 输出地址的 2-11 位决定。

3. 运算器

运算器的功能是，接受控制器传来的 ALU_OP 指令，根据指令内容，对操作数进行不同的运算。

运算器的输入与功能对应如表 2.1 所示，输入输出接口如表 2.2 所示。

表 2.1 算术逻辑运算单元 ALU_OP 与功能描述

输入信号	功能描述
0	逻辑左移
1	算术右移
2	逻辑右移
3	无符号乘法，高位使用 result2 存储
4	无符号除法，result2 存储余数
5	加法
6	无符号乘法，高位使用 result2 存储
7	无符号除法，result2 存储余数
8	加法
9	无符号乘法，高位使用 result2 存储

输入信号	功能描述
10	无符号除法，result2 存储余数
11	加法
12	无符号乘法，高位使用 result2 存储

表 2.2 算术逻辑运算单元引脚与功能描述

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
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

寄存器堆由 32 个 32 位寄存器组成，在 logisim 中已经封装打包好，其输入引脚与输出引脚的名称与功能如下：

表 2.3 寄存器堆对应内容

引脚	输入/输出	位宽	功能描述
R1#	输入	5	源操作寄存器 1 编号
R2#	输入	5	源操作寄存器 2 编号
W#	输入	5	写入寄存器编号

WE	输入	1	现在要写入寄存器
CLK	输入	1	时钟使能信号
RDin	输入	32	写入寄存器内容
R1	输出	32	寄存器 R1#存储内容
R2	输出	32	寄存器 R2#存储内容

2.1.2 数据通路的设计

为了更有效地设计后续流水线 CPU，我们将单周期 CPU 细分为五个逻辑紧密相连的阶段：取指、译码、执行、存储（或访存）、以及写回。以下是各阶段数据通路设计的详细逻辑关联：

(1) 单周期 CPU 取指数据通路设计

在单周期 CPU 的设计初期，我们聚焦于取指阶段，其核心在于确定并执行下一条指令的地址。此过程的核心组件是程序计数器（PC），它负责追踪当前指令的地址。系统启动时，PC 被初始化为 0。随着每个时钟周期的推进，PC 通常会增加 4（假设指令长度为 4 字节），以指向下一条顺序指令。然而，这一简单的递增逻辑可能受到前一条指令类型的影响，例如跳转或分支指令，它们可能要求 PC 根据算术逻辑单元（ALU）的计算结果或特定指令模式跳转到非顺序地址。因此，取指阶段需综合考虑这些条件，准确计算出下一条指令的地址，并更新 PC 寄存器。

(2) 单周期 CPU 译码数据通路设计

完成取指阶段后，我们进入译码阶段，其核心在于识别指令类型并生成相应的数据和控制信号。

控制信号生成：指令寄存器（IR）中的指令码被送入硬布线控制器，该控制器解析指令码并输出一系列控制信号，这些信号指导 CPU 各部分如何操作。

数据信号处理：

R 型指令：从 IR 中解析源寄存器 RS1、RS2 和目标寄存器 RD 的标识，并访问寄存器堆以获取 RS1 和 RS2 中的数据。

I 型和 S 型指令：从 IR 中提取立即数，并将其扩展至 32 位，准备用于后续的算术或逻辑操作。

B 型和 J 型指令：从 IR 中提取偏移量或跳转目标地址，用于计算下一条指令的地址。

U 型指令：处理较为简单，主要关注从 IR 中提取正确的立即数和 PC 位数。

通过上述分类处理，译码模块能够针对不同类型的指令生成精确的数据和控制信号，确保 CPU 能够准确无误地执行每条指令。这一设计不仅增强了 CPU 的灵活性，也为后续流水线 CPU 的设计奠定了坚实的基础，使得指令的执行流程更加清晰、高效。

(3) 单周期 CPU 执行数据通路设计：

在成功完成单周期 CPU 译码模块的设计后，我们转而专注于执行模块的设计。执行模块的核心职责涵盖执行指令所需的算术逻辑运算以及响应系统中断（如 ecall 指令）。首先，我们依据 AluSrcB 控制信号来确定 ALU 的 B 输入源，即是来自寄存器的值还是立即数。紧接着，根据 AluOp 控制信号，我们明确 ALU 需要执行的具体运算类型。最终，ALU 的计算结果被输出，以供后续处理阶段使用。

针对系统中断处理，我们设计了一个专用寄存器，用于暂存即将在 LED 上显示的数据。我们持续监测 A 引脚的输入值，若其等于 34（即 0x22，这个数据是 ecall 指令到达时对应的值），则将该寄存器设置为预定的显示数值。若 A 引脚的输入值不等于 34，则将 halt 标志置为 1，此举将导致时钟信号暂停，进而激活系统中断处理机制。

通过上述设计，执行模块在确保 CPU 高效执行算术逻辑运算的同时，亦能灵活响应并妥善处理系统中断请求。

(4) 单周期 CPU 存储与写回数据通路设计：

在单周期 CPU 执行模块设计圆满结束后，我们着手开展存储与写回模块的设计工作。该模块的核心任务是将数据保存到主存储器中，或将运算结果写回到寄存器堆。

我们根据指令类型，对从指令中解析出的立即数执行必要的操作。操作完成后，

处理后的数据被写入数据存储器，以实现数据的持久保存。

依据指令的具体类型，我们决定是将 ALU 的计算结果、数据存储器的输出值，还是下一条指令的地址作为写回数据。这些数据随后被精确地写入寄存器堆，以更新寄存器的状态。

通过精心策划的存储与写回流程，我们确保了 CPU 在单周期内能够高效地管理数据存储与寄存器状态更新，从而维护了整个计算流程的连贯性和数据的一致性。

2.1.3 控制器的设计

首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如下所示。

表 2.4 控制器控制信号

控制信号	取值	说明
Rs1	0、1	是否需要使用操作数 R1
Rs2	0、1	是否需要使用操作数 R2
AluOP	0-12	使用 ALU 的指令编号
BEQ	0、1	是否为 BEQ 指令
BNE	0、1	是否为 BNE 指令
MemToReg	0、1	是否需要从内存写入寄存器
MemWrite	0、1	是否需要写入内存
AluSrcB	0、1	第二操作数是为寄存器 2 中值（0）还是立即数（1）
RegWrite	0、1	是否需要写入寄存器
SLTIU	0、1	是否为 SLTIU 指令
JALR	0、1	是否为 JALR 指令
JAL	0、1	是否为 JAL 指令
BLTU	0、1	是否为 BLTU 指令

URET	0、1	是否需要中断返回
S_Type	0、1	是否为 S 型指令
Ecall	0、1	是否为 ecall 指令
Sb	0、1	是否为 sb 指令
Auipc	0、1	是否为 auipc 指令

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号，对于与本条指令无关的控制信号，控制信号的取值一律为 0，以简化控制器电路的设计。该控制信号表的框架如表 2.5 所示。

表 2.5 主控制器控制信号框架

指令	ALU_OP	MemToReg	ALU_Srcb	RegWrite	Ecall	S_Type	RS1_used	RS2_used
ADD	5		1	1			1	1
SUB	6			1			1	1
AND	7			1			1	1
OR	8			1			1	1
SLT	11			1			1	1
SLTU	12			1			1	1
ADDI	5		1	1			1	
ANDI	7		1	1			1	
ORI	8		1	1			1	
XORI	9		1	1			1	
SLTI	11		1	1			1	
SLLI	0		1	1			1	
SRLI	2		1	1			1	
SRAI	1		1	1			1	

LW		1	1	1			1	
SW			1			1	1	1
ECALL					1			
BEQ	6						1	1
BNE	6						1	1
JAL								
JALR			1				1	
URET					1			
SLTIU	12			1			1	
AUPIC				1				
SB	5	1	1					
BLTU	12						1	1

2.2 中断机制设计

2.2.1 总体设计

中断是计算机运行过程中的一种重要机制，当遇到需要主机干预的意外情况时，计算机能够自动暂停当前运行的程序，转而处理新情况，并在处理完毕后返回原程序继续执行。根据中断的性质和层次，中断可以分为单级中断和多级中断，而在硬件实现上，还可以进一步区分为 EPC 内存堆栈保护和 EPC 硬件堆栈保护的中断类型。

在本次实验设计中，我们专注于可屏蔽的外部中断。实验支持多种中断模式，包括单周期单级中断、单周期多级中断以及流水单级中断。

对于所有外部中断的处理，通常遵循以下步骤：首先保存下一条即将执行的指令地址，以保护当前程序的执行状态；然后保护现场的寄存器内容，以确保中断处理过程中不会丢失重要数据；接着跳转到中断源执行相应的中断服务程序。当中断处理完毕后，根据 ecall 指令恢复中断现场的寄存器内容，并将指令跳转到之前保存的下一条指令地址，以继续执行原程序。

在处理单级中断时，由于中断服务程序执行期间不会接受新的中断请求，因此可以使用一个寄存器来记录当前是否处于中断执行状态，并保存中断返回地址。

对于多级中断的处理，则需要考虑不同等级的中断优先级。在执行多级中断服务程序时，低级和同级的中断请求应被屏蔽，但允许更高级的中断打断当前服务。为了实现这一功能，可以根据中断服务程序的执行进度（如是否正在保存和恢复现场）来灵活调整中断指示标志。具体实现上，可以使用 CSSRCI 和 CSSRSI 两条指令来控制中断的屏蔽和允许状态。

2.2.2 硬件设计

无论是单周期 CPU 的多级中断还是流水线 CPU 的多级中断，我们都需要一系列寄存器来保存中断相关的信息。下面是这些寄存器的名称和存储内容的含义。

1. 中断使能寄存器（IE）：为 1 时表示当前允许中断，否则当前不允许中断。

2. 中断请求寄存器（IR）：为 1 代表对应的中断有请求，否则代表对应的中断无请求。

3. 中断返回地址寄存器（EPC）：存储中断执行前的 PC 寄存器的值。

4. 当前中断号寄存器（IR_id）：存储当前正在执行的中断的中断号。

由于多级中断支持中断嵌套，所以我们需要为每一个不同的中断设置一组上述的中断信息寄存器。同时应设置判断当前中断是否允许优先执行的数据通路。

2.2.3 软件设计

对于多级中断，我们需要分别设计中断 PC 计算、中断使能维护、EPC 寄存器堆维护、中断返回信号生成以及中断优先级确定的数据通路。

其中，中断 PC 计算数据通路需要根据当前的中断请求计算出当前将要执行的中断服务程序的首地址；中断使能维护数据通路需要根据当前的情况更新 IE 寄存器的值；EPC 寄存器堆维护数据通路需要在中断前保存中断的返回地址并在中断结束后将保存的返回地址输入 PC；中断返回信号生成数据通路需要在当前中断结束之后清除该中断的中断请求，以更新中断请求队列。中断优先级确定数据通路需要确定当前的中断是否允许优先执行。

2.3 流水 CPU 设计

2.3.1 总体设计

本次实验的目标是实现一个五段流水线 CPU，该 CPU 将指令执行过程划分为取指(IF)、译码(ID)、执行(EX)、访存(MEM)和写回(WB)五个阶段，并通过四组寄存器保存各阶段所需信息，以实现指令执行的相对独立性，这是理想流水线设计的核心。

然而，由于无条件和有条件跳转指令的存在，某些指令的效果需依赖于前一条指令的结果，这破坏了指令间的隔离，导致流水线效率降低。为解决这一问题，我们根据消除影响逻辑和方式的不同，将流水线分为气泡流水线和重定向流水线两大类，并通过动态分支预测技术进一步加速重定向流水线。在实验中，我们逐步设计了理想流水线 CPU、气泡流水线 CPU 和重定向流水线 CPU，并且基于此研究动态分支预测流水线和含中断的流水线。

理想流水线 CPU 虽然实现了流水线的基本原理，但由于无法处理跳转指令和数据相关，缺乏实用价值。气泡流水线 CPU 通过引入气泡来延迟计算数据相关的指令，但这种方法会导致效率下降。重定向流水线 CPU 则通过数据重定向技术解决了理想流水线 CPU 的问题，大大减少了气泡的添加，提高了流水线的效率。

2.3.2 流水接口部件设计

对于流水线设计而言，核心思想是设置寄存器，根据特殊信号以及时钟信号判断是否将数据进一步写入。根据五个阶段所需的信息不同，设计了如下的流水线接口：

表 2.4 不同阶段所需的流水线接口

阶段	所需流水线接口
IF/ID	IR、PC、PC+4、
ID/EX	BEQ、BNE、MemToReg、MemWrite、AluSrcB、RegWrite、S_Type、Ecall、JAL、JALR、SB、AUIPC、URET、BLTU、SLTIU、 ALUOP、RD、R1、R2、IMM_1、IMM_2、PC、PC+4、IR、RS1_Forward、RS2_Forward

EX/MEM	JAL、JALR、MemToReg、MemWrite、SB、auipc、sltiu、RegWrite、Halt、RD、Result、WriteData、IR、PC、PC+4
阶段	所需流水线接口
MEM/WB	JAL、JALR、MemToReg、RegWrite、Halt、RD、ALUResult、ReadData、IR、PC、PC+4、Auipc

2.3.3 理想流水线设计

理想流水线 CPU 的设计基于单周期 CPU 的架构，通过将取指、译码、执行、存储和写回这五个阶段分离，并利用四组流水线接口来实现。在这种设计中，每条指令的执行被划分为独立的阶段，各阶段之间通过流水线接口部件进行连接，确保前一阶段的输出信号能够作为下一阶段的输入信号。

特别地，对于停机指令，其在 EX 阶段处理完毕后，需要经过两个周期才能传送到 WB 阶段，以确保在执行停机操作之前，所有指令都已完整执行。这样的设计允许理想流水线 CPU 在不考虑同时进入流水线的指令间相互作用的情况下，直接将单周期 CPU 的各个步骤拆分并连接，实现流水线的高效运作。

2.4 气泡式流水线设计

针对气泡流水线 CPU 的设计，我们在理想流水线 CPU 的基础上增加了气泡冲突处理模块。该模块根据当前 CPU 的状态来判断是否发生冲突，一旦检测到冲突，就会插入一个气泡，即清空 IF/ID 和 ID/EX 这两个流水接口，以确保流水线能够正确处理数据冲突，维持指令执行的正确性。通过这种方式，气泡流水线 CPU 能够在保持流水线连续性的同时，解决数据冲突问题，尽管这可能会牺牲一定的效率。

2.5 数据转发流水线设计

数据转发流水线（重定向流水线）在气泡流水线的基础上进行了优化，减少了气泡的插入，从而提高了流水线 CPU 的效率。这种流水线通过汇总数据可能的全部路径，并根据指令的执行情况选择一条分支来获取数据。然而，当相邻两条指令存在数据相关，特别是前一条指令是访存指令时，会出现一种特定的冲突（称为 Load_Use），此时仍然需要使用气泡来避免冲突。为了支持这种重定向，我们需要

对原有的冲突处理模块进行修改，以便在两条数据相关的指令之间进行数据重定向。尽管如此，对于数据相关且上一条指令为访存指令的情况，我们仍然需要插入气泡来确保正确处理冲突。通过这种方式，重定向流水线在保持流水线效率的同时，也能够妥善解决数据相关性问题。

3 详细设计与实现

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

① Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

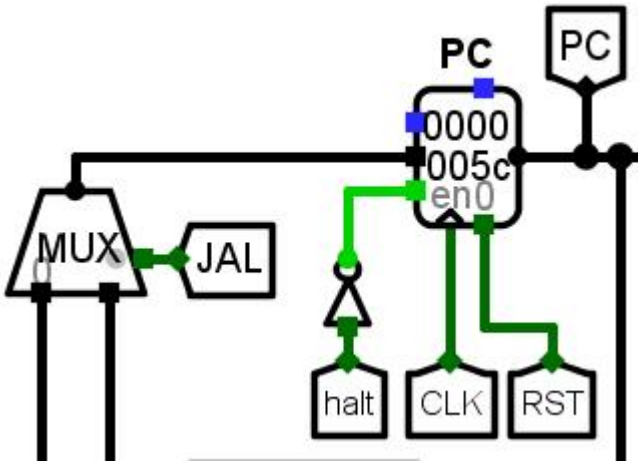

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

① Logism 实现：

使用一个只读存储器 ROM 实现指令存储器（IM）。设置该只读存储器的地址位宽为 10 位，数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位，而 ROM 地址线宽度有限，仅为 10 位，故将 32 位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器只取 32 位指令地址的 2-11 位作为指令存储器的输入地址。如图 3.2 所示。

图 3.2 指令存储器（IM）

3) 内存（MEM）

① Logisim 实现

电路中的内存和指令存储器一样，是由十位地址线（2-11 位）寻字，低两位寻字节的存储器，如下图 3.3.

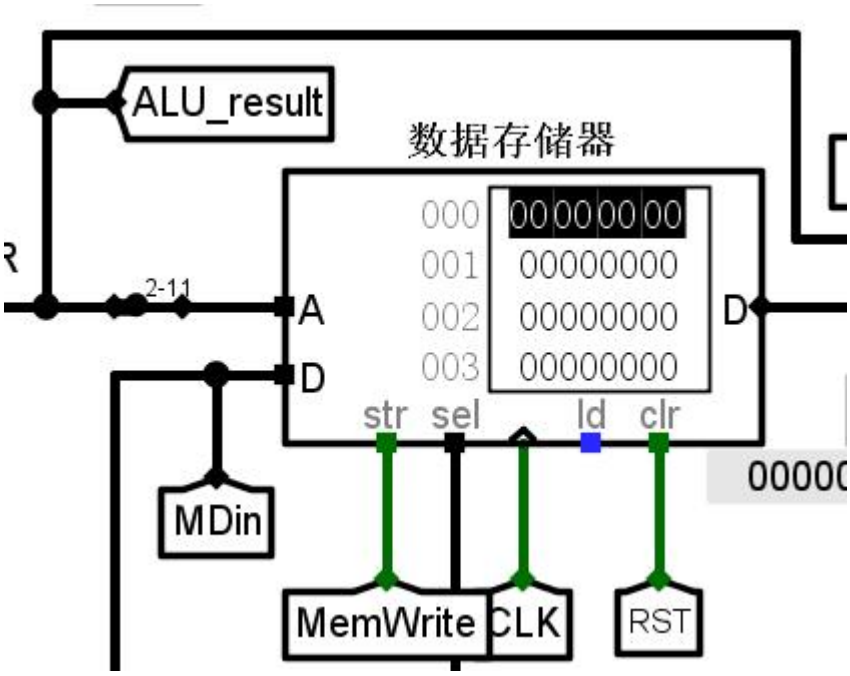

图 3.3 MEN

4) 寄存器（RegFile）

① Logisim 实现

RegFile 由 32 个 32 位的寄存器构成，其主要接口和功能都已经在总体设计方案中阐明。在设计包中 Logisim 的 RegFile 电路已经封装完

成，因而只需要直接调用即可。对于停机指令，可以在外侧使用简单逻辑电路进行判定具体输入。Losigim 实现如图 3.4 所示：

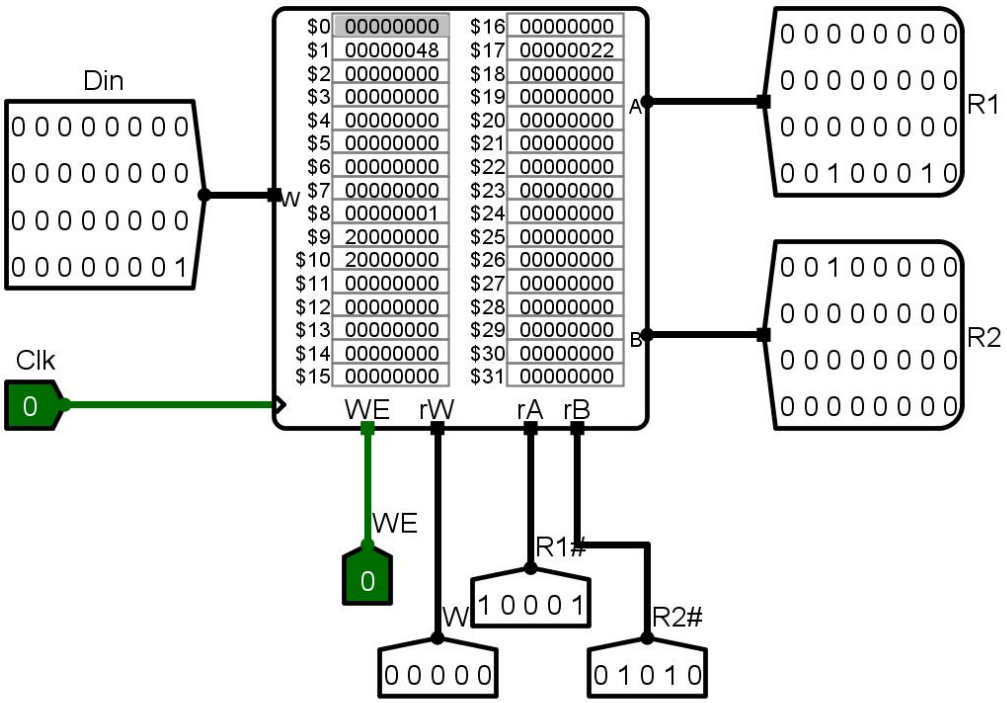

图 3.4 内存示意图

5) ALU

① Logisim 实现

和 RegFile 相同，ALU 也是已经提供好的子电路，接口也相对简单。

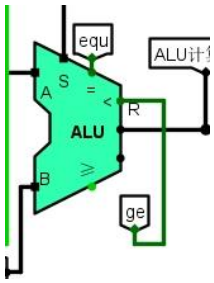

图 3.5 ALU 的 Logisim 实现

3.1.2 数据通路的实现

我们根据总体方案设计对应章节中的思路在 Logisim 上进行数据通路的实现。单周期 CPU 的数据通路也可大致分成取指、译码、执行、存储和写回五个阶段。我们通过选择合适的逻辑门、多路选择器、优先编码器等元件，实现了单周期 CPU 的数据通路，如图 3-6 所示。

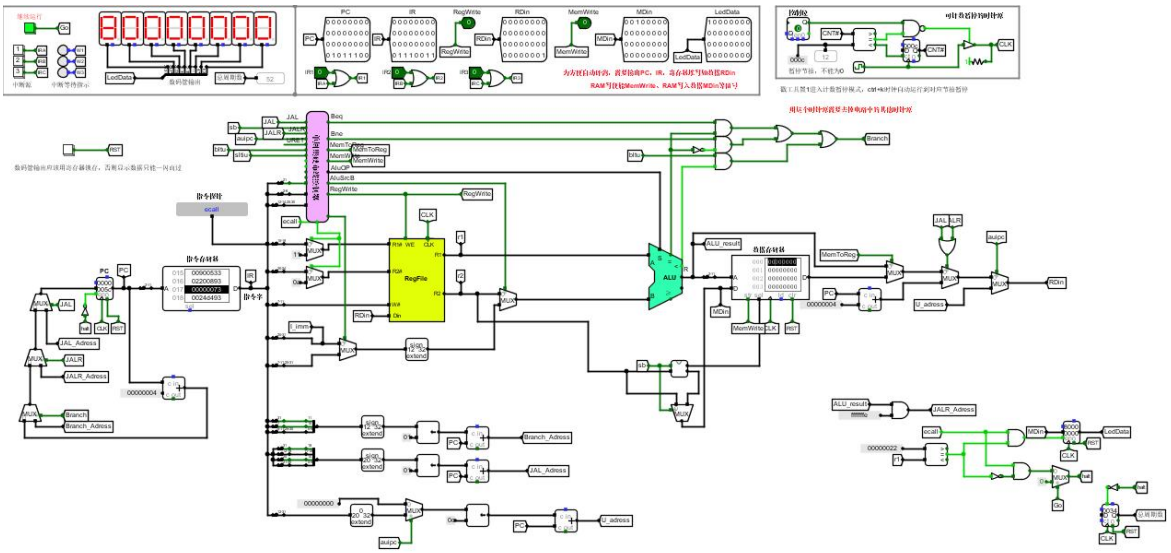

图 3.6 数据通路示意图

3.1.3 控制器的实现

① 主控制器：

主控制器对照表位于表 2.3。具体每个信号的电路通过填表，由 Logisim 自行生成。主控制器又由两个子电路构成：ALUOP 产生和其他控制信号产生器。最终总体结构图 3.7 所示：

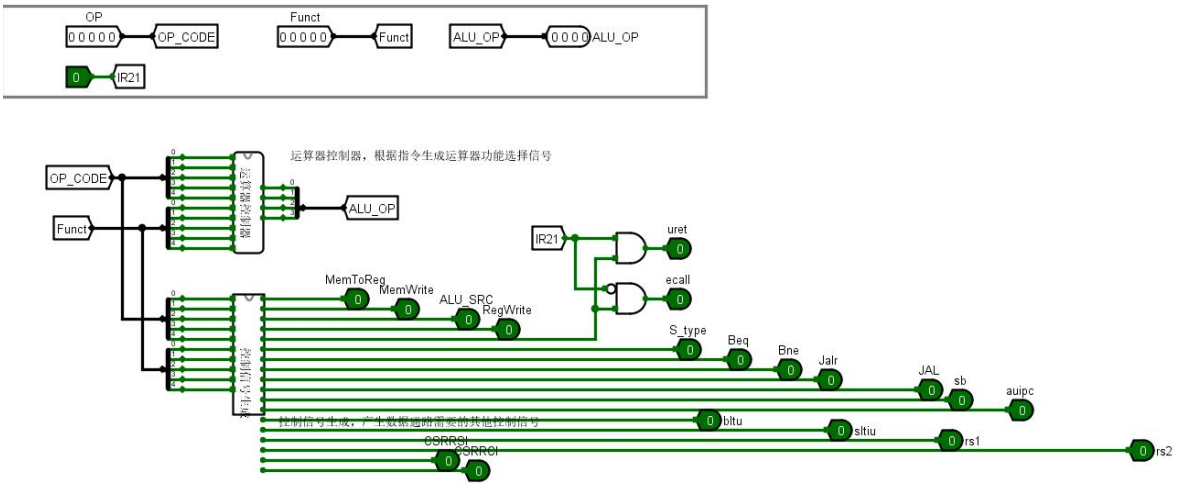

图 3.7 主控制器示意图

② Branch 控制器：

Branch 控制器利用 ALU 计算得到的控制信号和主控制器信号来联合控制跳转指令，原理图见图 3.8：

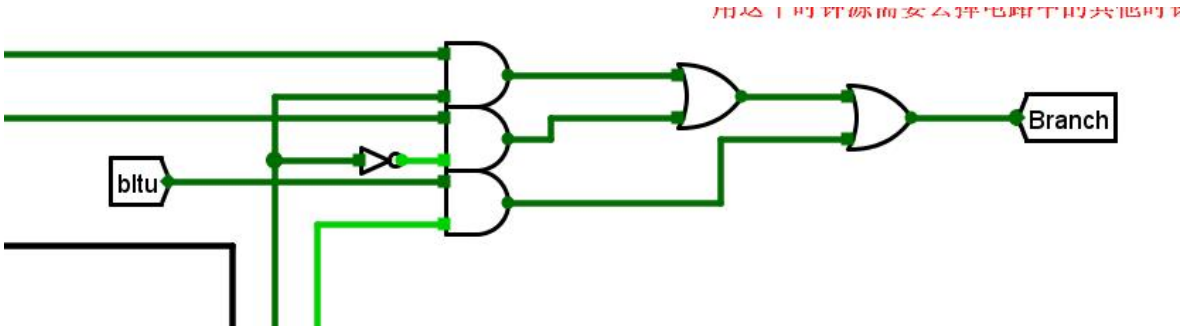

图 3.8 BRANCH 控制器

③ 停机控制器：

停机控制器主要实现停机和 LedData 的输出，根据寄存器中的值和 Ecall 信号来进行控制，原理图见图 3.9：

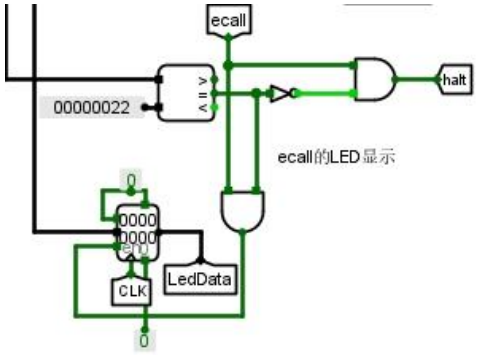

图 3.9 停机控制器 Logisim 实现

3.2 中断机制实现

3.2.1 单级中断

在处理中断时，无论是单级还是多级中断，确定下一条指令的跳转地址都是关键步骤。在单周期 CPU 的设计中，我们通过优先编码器和多个选择器，结合控制信号从 PC+4、中断源和 EPC（Exception Program Counter）中选择下一条指令的地址。中断源的具体地址可以通过 RARS 仿真器进行查看，而中断号的产生可以通过中断按键获取，中断地址通过寄存器保留记录。Logisim 原理图如下图所示。

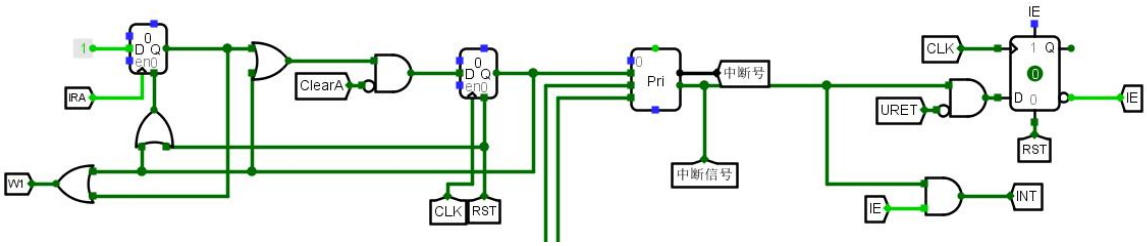

图 3.10 中断号获取

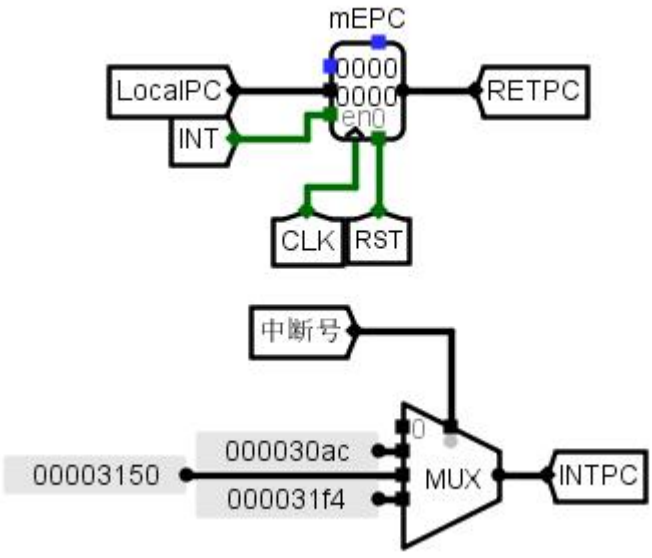

图 3.11 中断入口地址记录

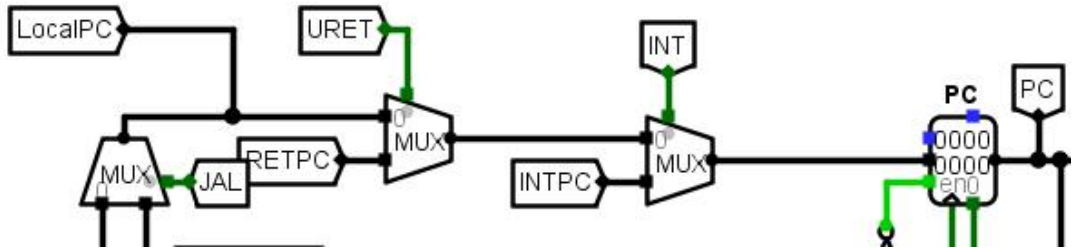

图 3.12 进入中断地址

3.2.2 多级中断

根据总体方案设计中的多级中断设计方案，我们依次在 Logisim 上实现了中断 PC 计算、中断使能维护、EPC 寄存器堆维护、中断返回信号生成以及中断优先级确定的数据通路。多级中断的整体电路如图 3.13 所示。

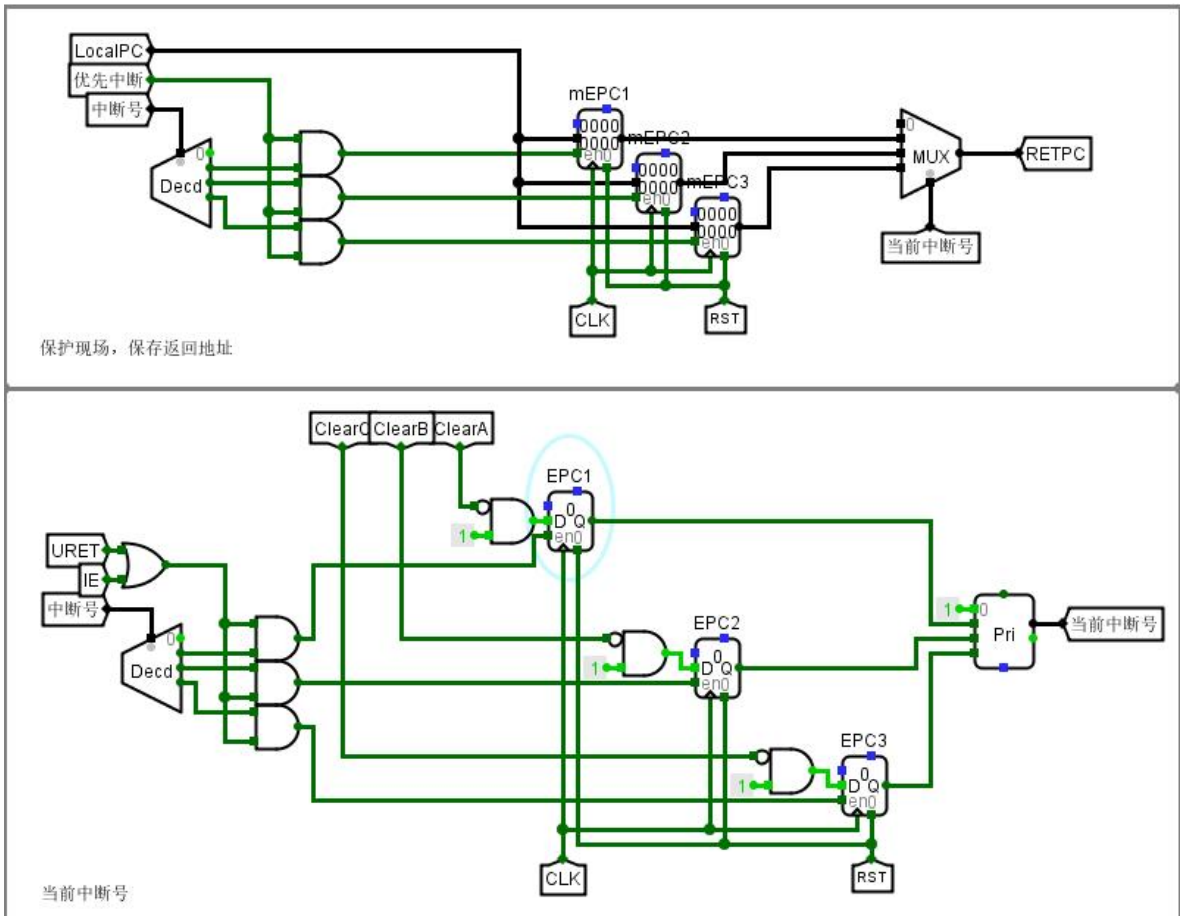

图 3.13 多级中断下对于中断优先级的处理

中断 PC 计算数据通路用于计算中断发生时 PC 应设置成的值。首先，我们使用一个优先编码器来对中断请求信号进行编码。接着，使用一个多路选择器根据中断请求信号选择对应的中断服务程序入口地址。然后，我们根据 CPU 当前

的状态判断应该把 PC 设置为何值。并输出中断产生信号。

EPC 寄存器（也就是 RETPC,记录地址)堆维护数据通路负责配置 EPC 寄存器的值，以支持多级中断。为此，我们配备了 3 个 EPC 寄存器以应对 3 个不同的中断号。该数据通路首先通过解码器识别当前执行的中断号，然后结合 CPU 的当前状态信息，利用多路选择器确定应存储到哪个特定中断号的 EPC 寄存器中的值。在中断处理完毕后返回时，数据通路还需依据 IR_id 寄存器中存储的返回中断号，从 EPC 中输出相应的值,大致上与单级中断是一致的。

为了管理多级中断中的中断嵌套和调度，我们构建了中断优先级确定数据通路。这个通路包括一个 IR_id 硬件栈来保存待执行的中断号，以及一个计数器来追踪栈顶指针。当新的中断请求到来时，如果其优先级高于当前栈顶中断或者栈为空，该中断会被压入栈中；否则，它将等待直到栈顶中断处理完毕。当栈顶中断完成并返回时，该中断会从栈中弹出。此外，如果新请求的中断优先级高于当前执行的中断，中断优先级确定数据通路将输出一个信号，指示当前指令的优先级更高，以便进行中断处理。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

该部分主要阐述了流水线接口的子电路设计。各流水线接口为若干寄存器，对每条输入和输出使用一个寄存器进行存储，并使用 reset 和 flush 等复位信号进行清空（插入气泡）。大致电路如图 3.12 所示： （以接口数最多的 ID 段展示)

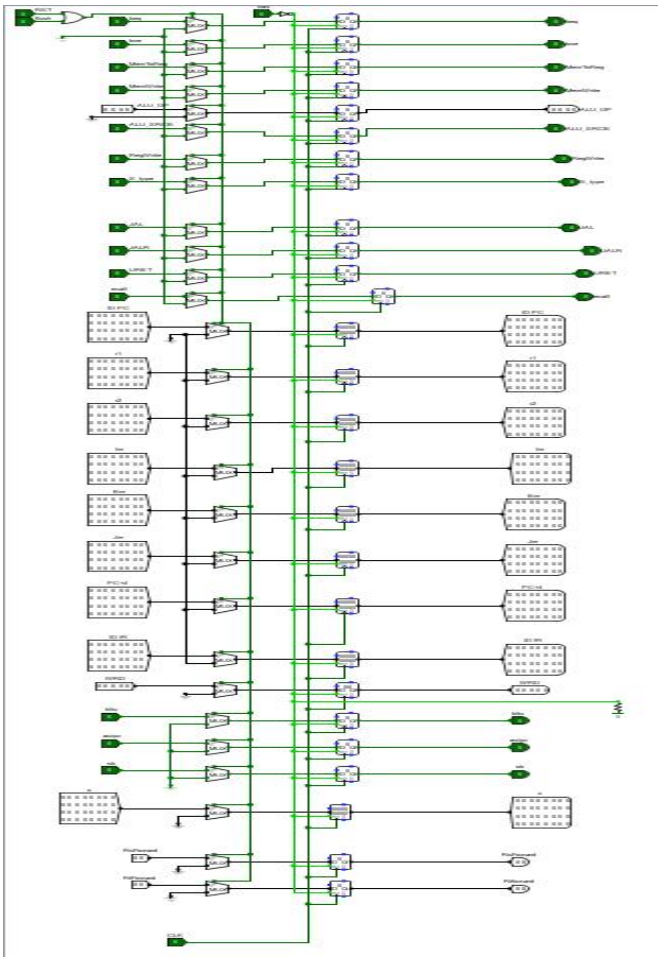

图 3.12 ID 段流水寄存器

3.3.2 理想流水线实现

我们在 Logisim 上构建了理想流水线 CPU。鉴于单周期 CPU 的设计已经按照取指、译码、执行、存储和写回这五个步骤进行，实现理想流水线 CPU 的过程相对直接，只需将这五个阶段分离，并在它们之间加入四个流水线接口组件即可。

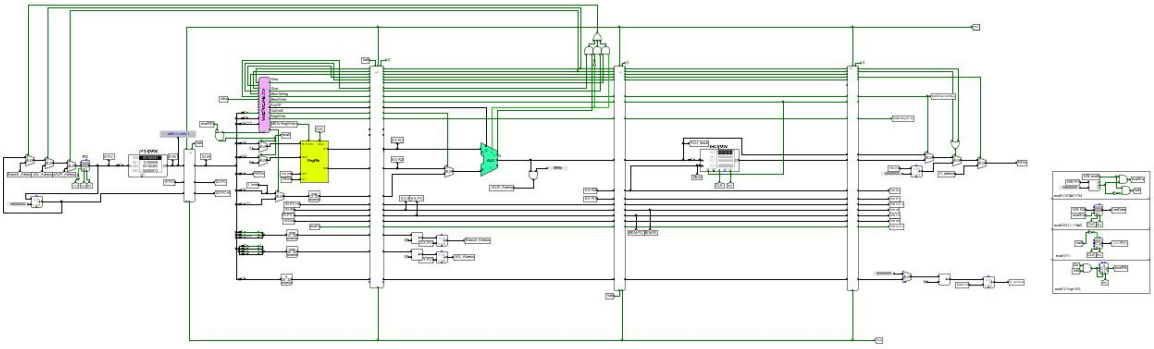

图 3.13 理想流水线电路图

3.4 气泡式流水线实现

气泡冲突处理电路是一个组合逻辑电路。它的输入是执行和存储阶段的 RD 和 RegWrite 等控制信号，输出是作用于流水接口部件的清空信号和作用于时钟的阻塞信号。由课本知识，通过寄存器使用情况设计冲突信号和清空信号，在其中，对于寄存器的使用情况可以通过指令的功能，类型来进行初步猜想，并通过实际运行情况加以验证。一般来说，寄存器使用情况和指令类型密切相关,同类型指令使用情况绝大部分情况下是一致的。

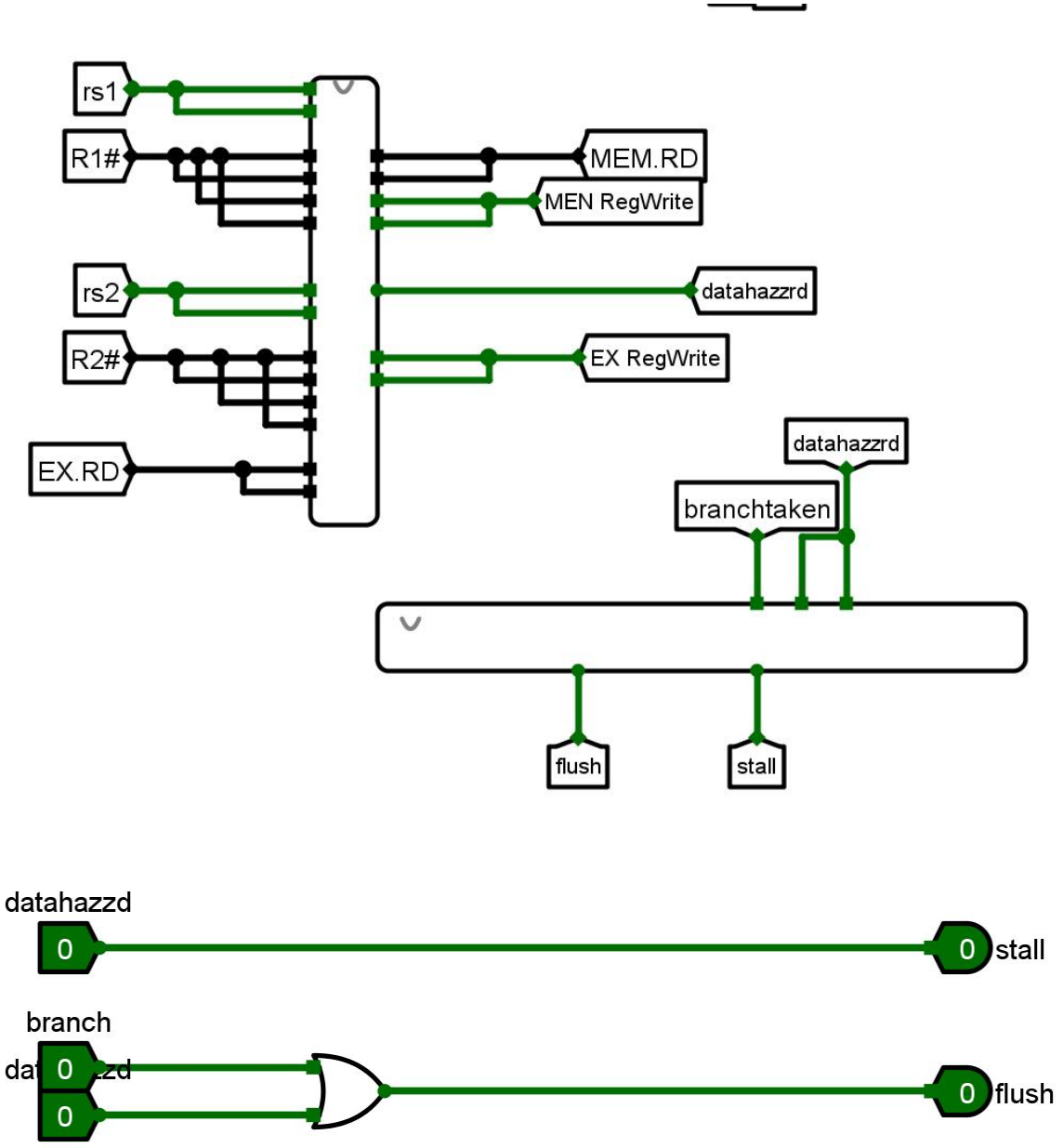

图 3.14 气泡处理逻辑

并且根据分析，我们在 ID/EX 段实现气泡操作，也就是在这一段内进行清零处理，因此，MEM，WB 之后都不需要清零操作。实际上需要清零的是 IF/ID,ID/EX。

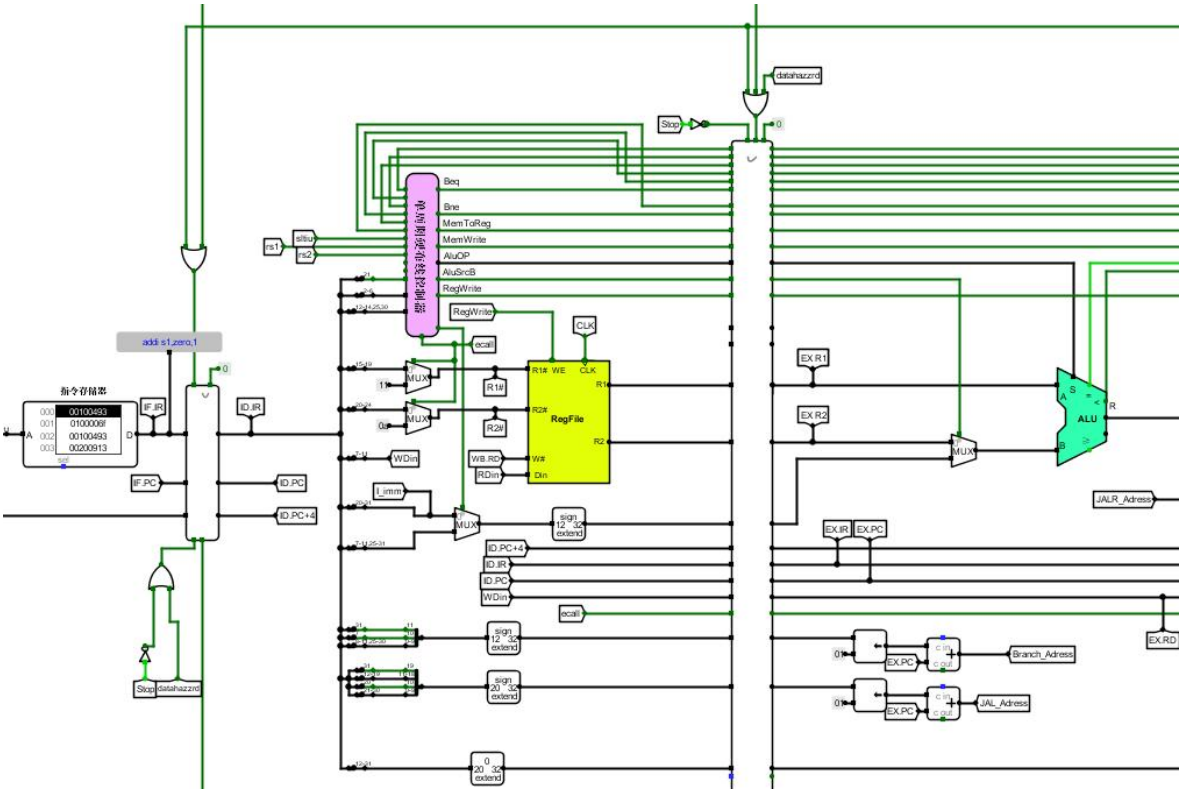

图 3.15 气泡流水线部分

3.5 数据转发流水线实现

根据总体方案设计，我们需要重新设计冲突处理电路，并其插入到已实现的气泡流水线 CPU 电路中，再对整体电路做一些修改即可。

对于重定向的冲突处理电路，我们删减了输出冲突和阻塞信号的一些情况，使得只有发生 load_use 时才会输出冲突和阻塞信号。对于其它的情况，我们增设 r1_forward 和 r2_forward 两个 2 位输出来支持数据重定向。当然，也是通过课本内容进行信号的判断处理操作。（课本里是 mimps 指令，同样可用于 risc-v，真好）

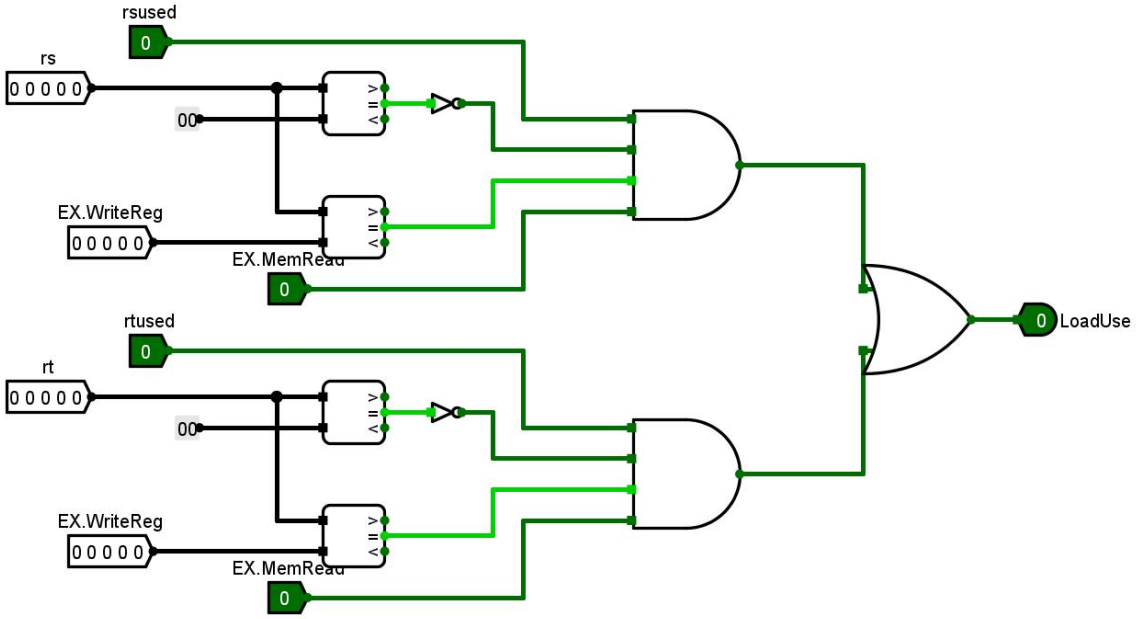

图 3.16 loadUse

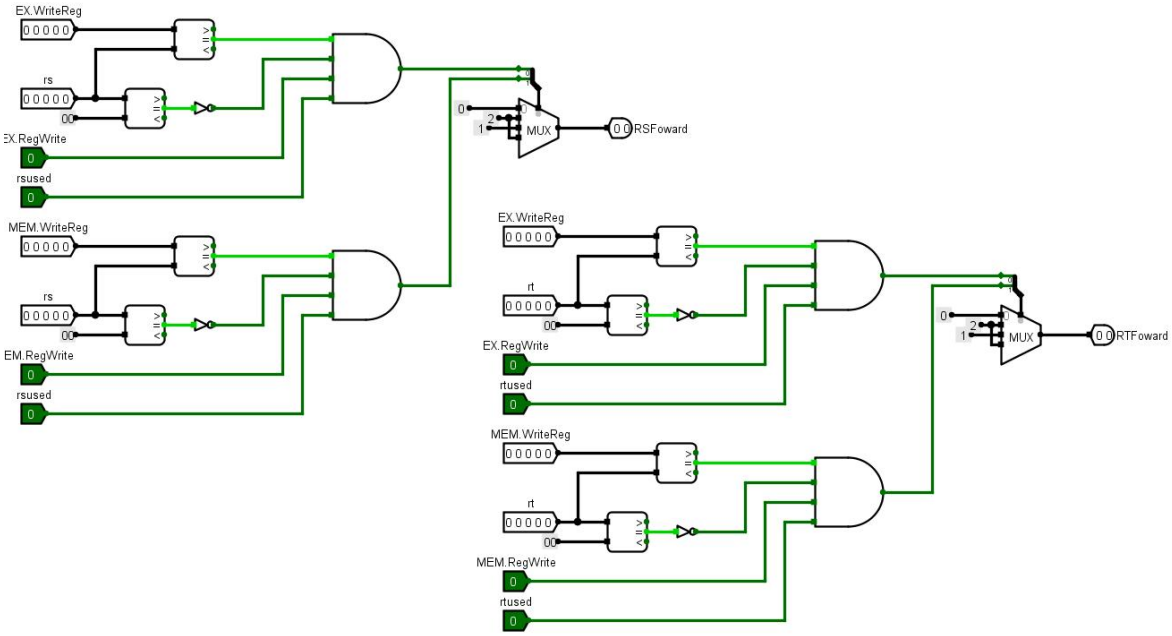

图 3.17 forward

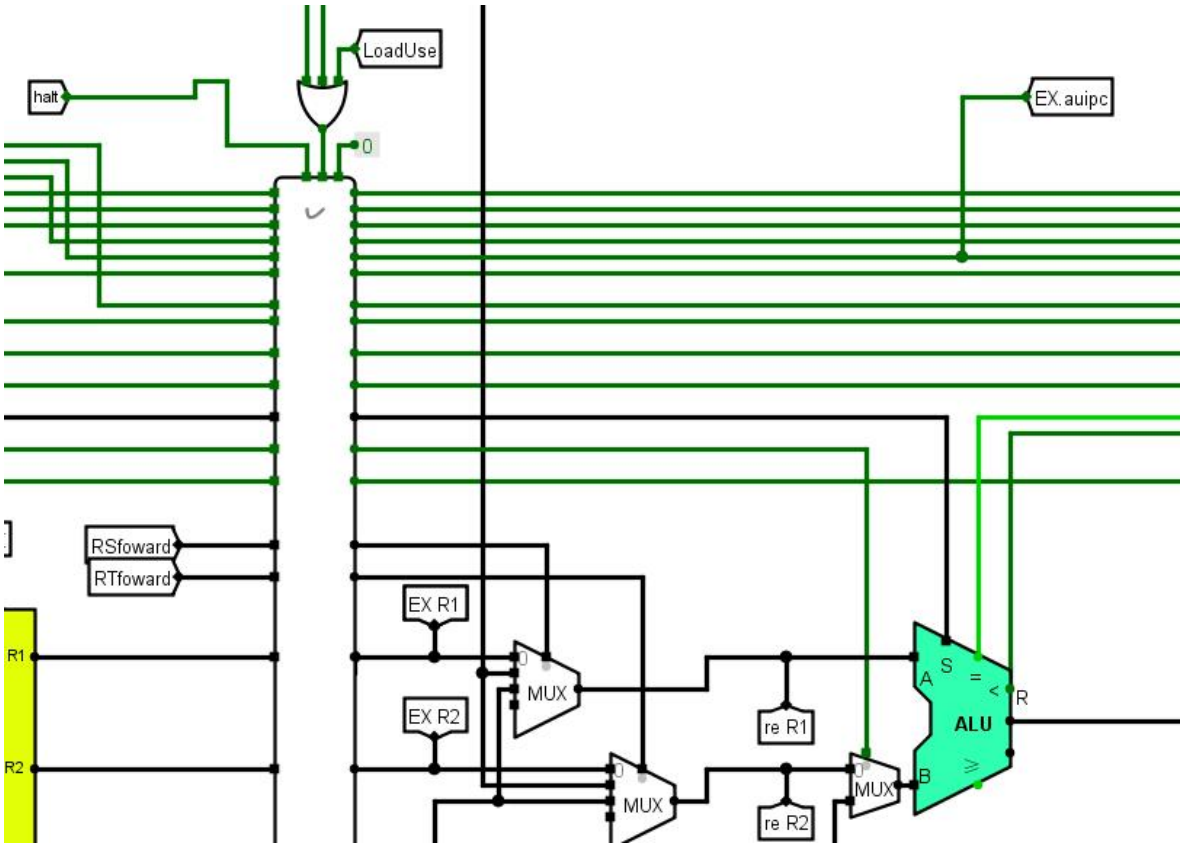

图 3.18 重定向流水线中与气泡最大的区别

4 实验过程与调试

4.1 测试用例和功能测试

本次实验中，单周期 CPU、气泡流水线 CPU、重定向流水线 CPU、单级和多级中断均在头歌平台通过仿真测试。

具体测试用例有 Benchmark.asm 和中断测试.asm 两个测试文件。以及 ccab 效果文件。

4.1.1 测试用例 1

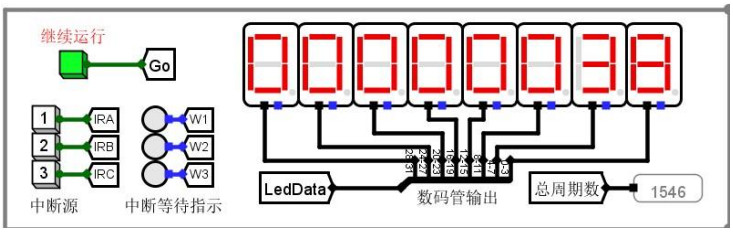

图 4.1 不含 ccab 效果单周期

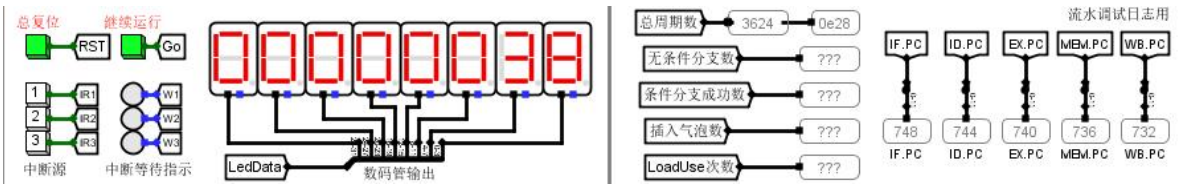

图 4.2 不含 ccab 气泡

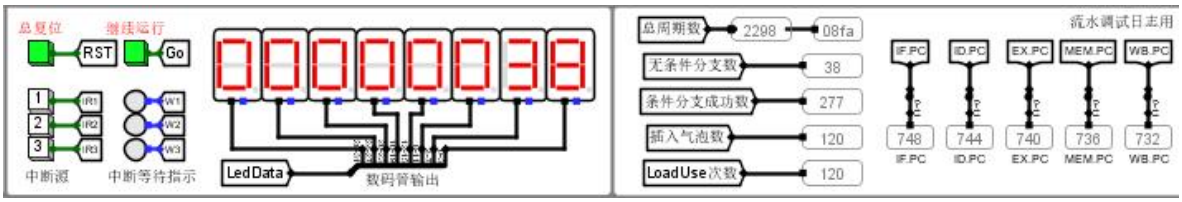

图 4.3 不含 ccab 重定向

4.1.2 测试用例 2

对于三种中断，对于中断按键操控的中断源均能在开中断时及时响应，并且在当前中断操作结束完成后及时响应中断请求队列中级别最高的中断请求，因而可以认为实验成功。

4.2 团队任务部分

我们团队设计任务是一个扫雷游戏极度简单版，我的工作是写 C 语言程序以及提供连接思路,同伴任务是在 logisim 里实现以及对汇编代码段的重构,连线。

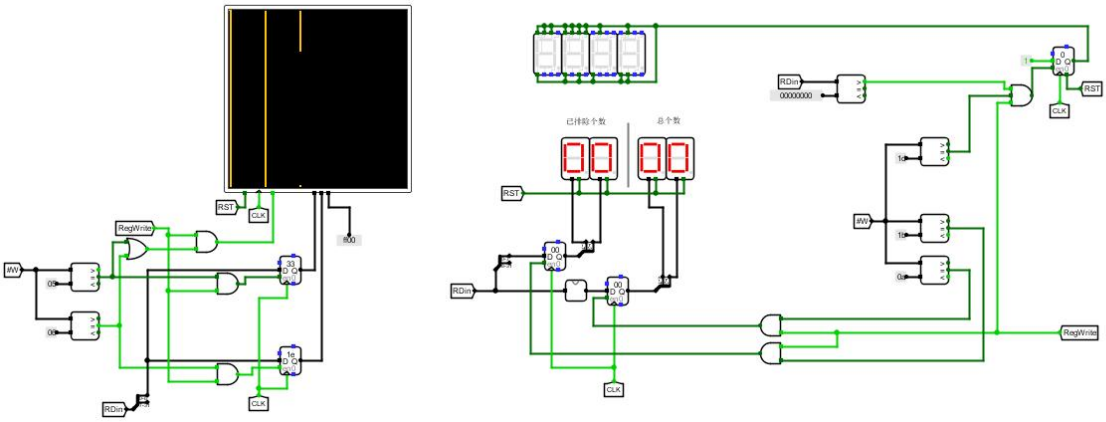

图 4.4 团队任务部分，详情见课程平台

4.3 性能分析

在单周期流水线 CPU 中，执行指令条数为 1546，但是每周期长度较长。

对于多周期的流水线，仅估算起见，每个周期长度可认为约为单周期 CPU 的周期长度的 1/5。对于气泡流水线，需要执行 3624 周期，相较于单周期流水线，时间缩短一半。对于重定向流水线，所需执行周期数为 2298，比气泡流水线又减少 1/3。

4.4 主要故障与调试

4.4.1 流水线数据故障

故障现象：执行 halt 指令时控制信号无法通过 ID/EX 接口。在进行到 0x200 周期之后会出现寄存器读数错误。

原因分析：寄存器最开始默认设置设置为上升沿刷新，但当 D 端有输入且 clk 变化时，数据来不及清零就被送出，此时导致本应该等一等的数据不见了。

解决方案：寄存器时钟端设置为下降沿触发。

4.4.2 重定向故障

故障现象：在重定向流水线中，运行到 0x260 节拍和 0x5ac 节拍左右时发生错误，此时寄存器使用情况不对应，导致七段数码显示管显示错误以及最后周期错误。

原因分析：这是由于 ecall 指令是既要使用 rs1 又要使用 rs2，在重定向时 load_use 错误导致的。

解决方案：修改 ecall 控制信号。

4.4.3 Go 故障

故障现象：在载入 ccab 后点击 GO 按键运行效果与裸机运行 ccab 单独指令天差地别。

原因分析：这是由于 ecall 指令是既要使用 rs1 又要使用 rs2，在重定向时 load_use 错误导致的。Go 按键的判断逻辑和清零逻辑需要考虑，因为指令运行前需要正确清除所有寄存器中的内容。

解决方案：完善 Go 的运行逻辑。

4.5 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 相关理论知识，阅读课设任务书，阅读 MIPS 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。完成部分 Logism 单周期 CPU 故障报告。
第三天	完成 Logism 单周期 CPU 的故障报告，并且通过了 Logism 单周期 CPU 的检查。
第四天	完成 Logisim 理想流水线
第五天	完成 Logisim 气泡流水线
第六天	开始 Logisim 重定向流水线

时间	进度
第七天	仍然在重定向流水线。
第八天	完成重定向流水线,并且开始中断.
第九天	完成中断,开始流水线中断.
第十天	完成流水线中断,着手设计团队任务.
第三十天	完成团队任务.

5 设计总结与心得

5.1 课设总结

在本次课设中，我完成了 Logisim 平台上 CPU 的相关设计。

1. 单周期 CPU
2. 理想流水线 CPU
3. 气泡流水线 CPU
4. 重定向流水线 CPU
5. 单周期 CPU 的多级中断支持
6. 重定向流水线 CPU 的多级中断支持
7. 含中断流水线（单级）

5.2 课设心得

这次课程设计标志着我首次涉足硬件领域，也是我第一次承担与硬件相关的大型项目。对于我这个硬件新手来说，面临的挑战是巨大的。然而，经过近两周的持续努力，我成功地完成了课程设计的所有要求，从基础开始，一步步设计并实现了一个 CPU。

课程组开始的内容设计与上个学期关系不算很大，因此需要思考框架，材料所提供的原理图是一个很好的入手点，思考几天基本上就把来龙去脉理清楚了。接下来需要做的就是如何完善分支，处理数据等等。

紧接着，理想流水线 CPU 的设计并没有什么难度，但是使用插入气泡、数据重定向技术对于流水线 CPU 进行冒险处理时，因为这些方法书本上并没有，老师提供的 PPT 上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，和小组内的成员进行相关探讨。

然而对于本次课程设计，我还有一些小小的建议和改进。就是对于 verilog 语言从头开始学到应用确实不容易，给多点参考资料就好了。以及头歌参考不严谨，非常非常不严谨，周期数对不上也给通过了，这不利于后续的改错和调试，因为总是不能很好的找到到底是哪个节拍出问题！

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。

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，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021年.

[4] 谭志虎，周健，周游.计算机组成原理实验指导（基于RISC-V在线实训）. 北京：人民邮电出版社，2024年.

[5] 曹强，施展.计算机系统结构（微课版）. 北京：人民邮电出版社，2024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：