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：	CS2202
学　　号：	U202215383
姓　　名：	何志鑫
电　　话：	18000908716
邮　　件：	3406199643@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	LB	字节加载	
29	BLTU	无符号小于时分支	
30	SLL	逻辑左移	
31	AUIPC	PC 加立即数	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 采用硬布线控制器方案，主要包括以下功能部件：程序计数器 PC、指令存储器、硬布线控制器、寄存器堆、运算器和数据存储器。每个部件在 CPU 中承担特定功能，共同组成主要的数据通路，总体结构图如图 2.1 所示。通过逐步实现各功能部件及其数据通路，可以完成整个单周期 CPU 的设计。

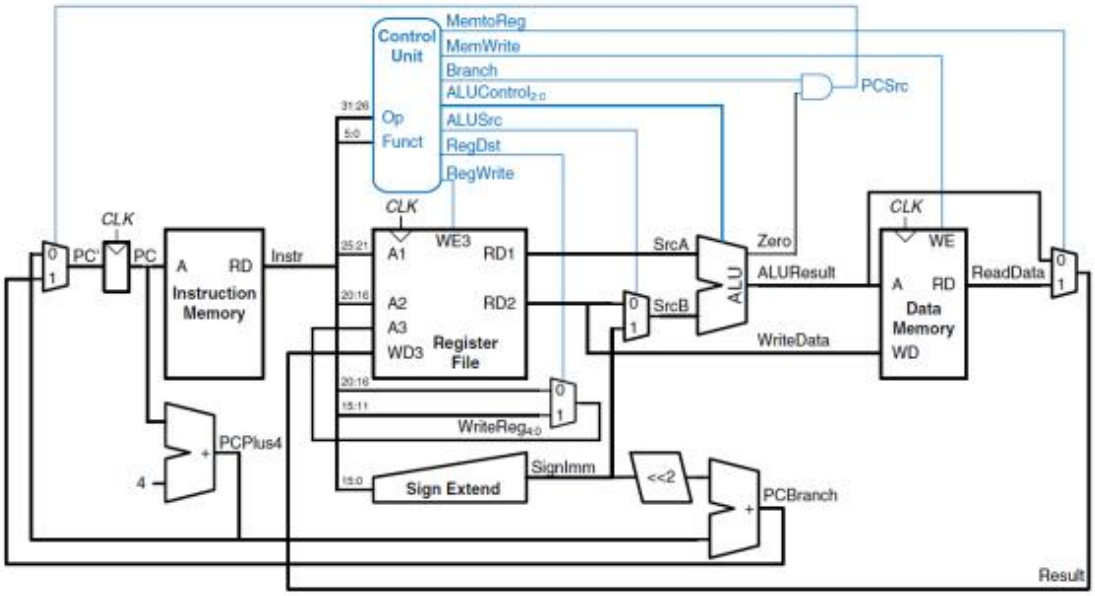

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

程序计数器 PC 的作用是确定 CPU 下一条运行指令的地址。在指令的取指阶段，处理器根据程序计数器中的地址从内存中取出指令。当指令被取出后，程序计数器的值会被更新，可能是顺序下一条指令，也可能是需要跳转去的指令。输入和输出的位宽都是 32 位。

2. 指令存储器 IM

指令存储器是一个 ROM，用于存储指令的内容。其输入输出引脚见表 2.1。

表 2.1 指令存储器的输入输出引脚表

类型	引脚名称	引脚含义	位宽
输入	A	指令地址	10
	sel	片选信号	1
输出	D	指令内容	32

3. 运算器

运算器用于进行结果的计算。输入输出引脚见表 2.2。

表 2.2 运算器的输入输出引脚表

类型	引脚名称	引脚含义	位宽
输入	A	操作数 1	32
	B	操作数 2	32
	S	功能选择	4
输出	R	计算结果	32
	=	是否相等	1
	≥	大于等于	1
	＜	是否小于	1

4. 寄存器堆 RF

寄存器堆用于模拟 RISC-V 需要的所有寄存器。输入输出引脚见表 2.3。

表 2.3 寄存器堆的堆输入输出引脚表

类型	引脚名称	引脚含义	位宽
输入	R1#	R1 地址	5
	R2#	R2 地址	5

类型	引脚名称	引脚含义	位宽
	W#	写入地址	5
	Din	写入内容	32
	WE	写使能	1
	CLK	时钟信号	1
输出	R1	R1 内容	32
	R2	R2 内容	32

5. 数据存储器

数据存储器用于模拟主存，是一个 RAM。使用 logisim 中的 RAM 器件即可。其输入输出引脚见表 2.4。

表 2.4 数据存储器的输入输出引脚表

类型	引脚名称	引脚含义	位宽
输入	A	输入地址	10
	D	输入数据	32
	str	写使能	1
	sel	片选信号	1
	ld	读使能	1
	clr	清空信号	1
	CLK	时钟信号	1
输出	D	输出数据	32

2.1.2 数据通路的设计

根据总体结构图，将单周期 CPU 分成取指（IF）、译码（ID）、执行（EX）、存储（MEM）和写回（RB）五部分，这也方便于后续流水线 CPU 的设计。

1. 取值部分数据通路设计

数据通路的取指部分是首要设计环节，其关键作用是获取下一条指令在指令寄

存器里的地址。取值部分的中心部件是 PC（程序计数器）寄存器，它记录着当前指令在指令寄存器的地址。起初，PC 寄存器被初始化为 0。在每个时钟周期里，PC 寄存器中的值会被读取，并且增加 4，这代表着按照顺序执行的下一条指令的地址。之后，会依据前一条指令的种类来决定下一条指令的地址是 PC + 4，还是由前一条指令本身或者 ALU（算术逻辑单元）运算得出的地址。最终，准确的指令地址会被重新写入 PC 寄存器，从而实现地址的更新。

2. 译码部分数据通路设计

译码部分的主要任务是区分指令的种类，并据此产生相应数据和控制信号。对于控制信号的产生，将指令字（IR）送入事先设计好的硬布线控制器，从而获得相应的控制信号。在处理指令相关数据方面，需根据不同指令类型进行区分处理：

① 对于 R 型指令，要从 IR 中解析出 RS1、RS2 和 RD 寄存器的编号，然后将 RS1 和 RS2 的编号送入寄存器堆，以便获取它们所对应的值。

② 对于 I 型和 S 型指令，要从 IR 中提取立即数，并将其扩展成 32 位，以便后续的运算使用。

③ 对于 B 型和 J 型指令，要从 IR 中提取地址偏移量，用以计算下一条指令的目标地址。

通过上述设计，译码部分可以精准地生成执行指令所需的数据和控制信号。

3. 执行部分数据通路设计

执行部分的主要任务是计算指令的执行结果以及处理系统中断（ecall）指令。

① 计算执行结果

根据 AluOp 控制信号，选择 ALU 执行的运算类型；根据 AluSrcB 控制信号，确定 ALU 的 B 引脚的输入是否为立即数；最后将 ALU 运算结果作为执行部分的结果输出。

② 系统中断（ecall）指令处理

处理系统中断之前，需要存储此刻的数据，设置一个寄存器，存储需要在 LED 上显示的数据。具体的处理方法是判断 ALU 的 A 引脚的输出值是否为 34。如果是，将寄存器设置为需要显示的数据；否则，将 halt 信号设置为 1，触发时钟暂停，停

止 CPU 运行。

通过上述设计，执行部分可以正确地完成指令运算、系统中断的处理。

4. 存储部分数据通路设计

存储部分的任务是将数据存储到主存中。根据指令类型，如果指令需要存储数据，将执行部分输出的结果输入数据存储器。

5. 写回部分数据通路设计

写回部分的任务是将数据写回到寄存器堆。有的指令需要将指定数据在下一周期写回到寄存器堆中，写回的数据可能包括 ALU 计算结果、存储器的输出值、下一条指令地址等。

2.1.3 控制器的设计

1. 运算器控制信号

运算控制信号用 AluOp 来表示。为了保证每条指令都能借助 ALU 正确地完成运算，必须为每条指令指定合适的 AluOp。而 ALU 具备 13 种不同的运算功能，故 AluOp 需要使用 4 位比特来表示。

为了实现根据指令生成 AluOp 的功能，单独设计了一个用于生成运算器控制信号的组合逻辑电路。在设计该电路时，首先确定每条指令对应的 AluOp，然后计算出 AluOp 每位比特对应的逻辑表示式。根据这些逻辑表达式就可以得到运算器控制信号生成电路。

2. 硬布线控制器

在单周期 CPU 中，除了 AluOp 控制信号外，还需要生成其他 15 个控制信号来完成控制，如 RegWrite、MemWrite 等。这些控制信号对于每条指令来说都必须唯一确定，以确保能正确地区分出不同指令以及实现指令的功能。

除了 AluOp，其他控制信号只能取 0 或 1 两个值，因此这些控制信号仅需 1 位来表示。具体控制信号如表 2.5。

表 2.5 主控制器控制信号及说明

控制信号	取值	说明
RegWrite	0/1	寄存器写使能
MemWrite	0/1	写内存控制信号
AluOP	0-12	运算器操作控制符（4 位）
MemToReg	0/1	寄存器写入数据来自存储器
S_Type	0/1	S 型指令译码信号
AluSrcB	0/1	运算器 B 输入选择
JALR	0/1	JALR 指令译码信号
JAL	0/1	JAL 指令译码信号
Beq	0/1	Beq 指令译码信号
Bne	0/1	Bne 指令译码信号
ecall	0/1	ecall 指令译码信号
U_Type	0/1	U 型指令译码信号
BLTU	0/1	BLTU 指令译码信号
LB	0/1	LB 指令译码信号
rs1_used	0/1	需要使用 rs1 寄存器
rs2_used	0/1	需要使用 rs2 寄存器
CSR	0/1	CSRRSI、CSRRCI 指令译码信号

得到具体控制信号后，就可以设计硬布线控制器电路（组合逻辑电路）。根据需要实现的指令，明确每位控制信号的对应值（如表 2.6），这样就可以计算出相应的逻辑表达式。最后就可以直接生成硬布线控制器的电路。

表 2.6 指令对应控制信号

指令	ALU_OP 取值	取值为 1 的控制信号
add	5	RegWrite, user1, user2
sub	6	RegWrite, user1, user2
and	7	RegWrite, user1, user2

指令	ALU_OP 取值	取值为 1 的控制信号
or	8	RegWrite, user1, user2
slt	11	RegWrite, user1, user2
sltu	12	RegWrite, user1, user2
addi	5	ALU_Src, RegWrite, user1
andi	7	ALU_Src, RegWrite, user1
ori	8	ALU_Src, RegWrite, user1
xori	9	ALU_Src, RegWrite, user1
slti	11	ALU_Src, RegWrite, user1
slli	0	ALU_Src, RegWrite, user1
srli	2	ALU_Src, RegWrite, user1
srai	1	ALU_Src, RegWrite, user1
lw	5	MemToReg, ALU_Src, RegWrite, user1
sw	5	MemWrite, ALU_Src, S_Type, user1, user2
ecall	/	ecall
beq	6	beq, user1, user2
bne	6	bne, user1, user2
jal	/	RegWrite, user1, user2
jalr	6	ALU_Src, RegWrite, jalr, user1
CSRRSI	8	ALU_Src, csr
CSRRCI	7	ALU_Src, csr
URET	/	ecall
lb	/	MemtoReg, ALU_Src, RegWrite, LB, rs1_used,rs2_used
bltu	12	BLTU, rs1_used, rs2_used
sll	0	RegWrite, rs1_used, rs2_used
AUIPC	5	RegWrite, U_Type

2.2 中断机制设计

2.2.1 总体设计

中断是指在计算机运行过程中，当发生某些意外情况需要主机干预时，计算机能够自动停止当前程序的执行，并转入处理新的情况的程序。在处理完新情况后，计算机会返回原先被暂停的程序继续运行。中断可以分为单级中断和多级中断两种类型，也可以依据中断保护方式的不同，分为 EPC 内存堆栈保护和 EPC 硬件堆栈保护两种形式。

单级中断：在执行中断服务程序期间，CPU 不允许其他中断请求插入，即在处理一个中断时，不会响应其他中断请求，直到当前中断服务程序执行完毕。

多级中断：允许高优先级的中断打断正在执行的低优先级中断服务程序，即在处理低优先级中断时，如果出现高优先级的中断请求，CPU 会暂停当前的低优先级中断服务程序，转而处理高优先级的中断。

EPC 内存堆栈保护：当发生中断时，CPU 会将中断发生前程序计数器（PC）的值压入内存中的堆栈进行保存，以便在中断处理完成后能够准确地返回到原先程序的中断点继续执行。

EPC 硬件堆栈保护：CPU 将中断发生前的 PC 值保存在专门的 EPC（Exception Program Counter）寄存器中，这种方式通过硬件机制来保护中断发生前的程序状态，提高了中断处理的效率和可靠性。

在实验设计中，采用了 EPC 硬件堆栈保护方式，并且分别设计了单周期 CPU 和流水线 CPU 的多级中断机制。所设计的 CPU 支持 3 个中断，分别对应中断号 1、2 和 3，其中中断号越大表示中断的优先级越高，即中断号 3 的优先级最高，中断号 1 的优先级最低。这种设计使得 CPU 能够根据中断的优先级灵活地处理多个同时发生的中断请求，提高了系统的响应能力和处理效率。

2.2.2 硬件设计

中断需要一系列寄存器来保存与中断相关的信息。以下是寄存器的名称及其存储内容的含义：

① 中断使能寄存器（IE）：当值为 1 时，表示当前允许中断；当值为 0 时，

表示当前不允许中断。

② 中断请求寄存器（IR）：当值为 1 时，表示对应的中断有请求；当值为 0 时，表示对应的中断没有请求。

③ 中断返回地址寄存器（EPC）：存储中断发生前的 PC 寄存器值，即程序中断时的地址。

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，需要为每个不同的中断设置一组上述的中断信息寄存器。同时，还需要设计数据通路来判断当前中断是否允许优先执行。

2.2.3 中断数据通路设计

对于多级中断，需要设计多个数据通路来支持中断的处理，包括中断 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。每个数据通路的功能如下：

① 中断 PC 计算数据通路：根据当前的中断请求计算出即将执行的中断服务程序的首地址。

② 中断使能维护数据通路：根据当前的状态更新中断使能寄存器（IE）的值，控制中断是否被允许。

③ EPC 寄存器堆维护数据通路：在中断发生前，将中断返回地址（即中断前的 PC 值）保存；在中断处理完成后，将保存的返回地址输入到 PC 中，以便恢复程序的执行。

④ 中断返回信号生成数据通路：当中断处理完成后，清除当前中断的中断请求，从而更新中断请求队列，确保系统能继续处理后续的中断。

⑤ 中断优先级确定数据通路：根据中断的优先级，判断当前的中断是否可以优先执行，确保高优先级的中断能打断低优先级的中断服务程序。

这些数据通路共同工作，实现了多级中断的处理和管理。

2.3 流水 CPU 设计

2.3.1 总体设计

本次实验需要搭建一个五阶段流水线 CPU。运作模式是把每条指令细分为取指、

译码、执行、访存和写回这五个阶段。在流水线操作中，当一条指令流转到下一阶段时，下一条指令紧跟着进入当前阶段，以此实现多条指令的同时处理。

实验通过分步迭代的方法，先后设计了理想流水线 CPU、气泡流水线 CPU 和重定向流水线 CPU，并且最终将重定向流水线 CPU 实现了上板测试。

① 理想流水线 CPU：基于流水线 CPU 的基本工作原理进行设计，实现了指令的分阶段处理。然而，这种 CPU 存在明显的局限性，无法处理跳转指令，也无法解决数据相关问题，即当一条指令需要依赖前一条指令的结果时，理想流水线 CPU 无法有效处理，这使得它在实际应用中不具备可行性。

② 气泡流水线 CPU：为了解决理想流水线 CPU 中的数据相关问题，气泡流水线 CPU 采取了在流水线中插入“气泡”的策略。所谓“气泡”，实际上是一个空操作，用于延迟那些需要等待前一条指令结果的指令的执行。虽然这种方法能够在一定程度上解决数据相关问题，但它也引入了额外的延迟，降低了流水线的整体效率。

③ 重定向流水线 CPU：针对气泡流水线 CPU 中因插入过多气泡而导致效率降低的问题，重定向流水线 CPU 引入了数据重定向技术。通过这种技术，指令可以直接利用已经修改的数据，无需再等待数据的最终写回，从而大大减少了气泡的插入数量，有效提升了 CPU 的运行效率。

通过这一系列的设计改进，重定向流水线 CPU 成功克服了理想流水线 CPU 和气泡流水线 CPU 的主要缺陷，显著提升了 CPU 的性能和实用性。

2.3.2 流水接口部件设计

五段流水线 CPU 需要 4 个流水接口部件，用于存储下一阶段所需的控制信号和数据，并将这些信号和数据转发到下一个阶段。这些接口部件的构造类似，都由一组寄存器和一套组合逻辑电路来决定寄存器输入的方式。

各个流水接口部件的输入输出引脚如表 2.7 所示。

表 2.7 指令对应控制信号

流水接口部件类型	引脚名称	引脚含义	流水接口部件类型	引脚名称	引脚含义
公有	reset	复位信号	ID/EX	imm1	立即数 1
	refresh	刷新信号		imm2	立即数 2
	en	使能信号		预测跳转	预测是否会跳转
	clk	时钟信号		r1_forward	重定向控制信号 1
	PC	当前指令地址		r2_forward	重定向控制信号 2
	PC+4	下一条指令地址	EX/MEM	MemToReg	主存写入寄存器堆
	IR	当前指令内容		MemWrite	写入主存
IF/ID	预测跳转	预测是否会跳转		RegWrite	写入寄存器堆
ID/EX	Beq	Beq 指令控制信号		Jal	Jal 指令控制信号
	Bne	Bne 指令控制信号		Jalr	Jalr 指令控制信号
	MemToReg	主存写入寄存器堆		uret	uret 指令控制信号
	MemWrite	写入主存		dasb	sb 指令控制信号
	AluSrcB	ALU 操作数来源		halt	暂停信号
	RegWrite	写入寄存器堆		res	ALU 计算结果
	Jal	Jal 指令控制信号		R2	ALU 操作数 2
	Jalr	Jalr 指令控制信号		rd_a	写回地址
	uret	uret 指令控制信号	MEM/WB	MemToReg	主存写入寄存器堆
	csr	特权指令控制信号		RegWrite	写入寄存器堆
	Bgeu	Bgeu 指令控制信号		Jal	Jal 指令控制信号
	S_type	S 型指令指示		Jalr	Jalr 指令控制信号
	dasb	sb 指令控制信号		uret	uret 指令控制信号
	ecall	ecall 指令控制信号		dasb	sb 指令控制信号
	AluOp	ALU 功能选择		halt	暂停信号
	R1	ALU 操作数 1		res	ALU 计算结果
	R2	ALU 操作数 2		rd_a	写回地址
	rd_a	写回地址		MemData	主存输出数据

2.3.3 理想流水线设计

设计理想的流水线 CPU，在已完成的单周期 CPU 基础上，将其工作流程分为五个阶段：取指、译码、执行、存储和写回。在这五个阶段之间插入设计好的四个流水接口部件即可。

每个流水接口部件的输入应连接到上一阶段生成的相关信号，而其输出应连接到下一阶段所需的对应信号。这样就可以实现数据和控制信号在各阶段间的有序传递，从而完成流水线的整体设计。

2.4 气泡式流水线设计

设计气泡流水线 CPU，需要在理想流水线 CPU 的基础上，增加一个气泡冲突处理模块。该模块的作用是根据 CPU 当前状态判断是否存在冲突。如果检测到冲突，则插入一个气泡，具体操作是清空 IF/ID 流水接口和 ID/EX 流水接口的内容。

气泡冲突处理模块的输入和输出引脚定义如表 2.8 所示。

表 2.8 气泡冲突处理的输入输出引脚

类型	引脚名称	引脚含义	位宽
输入	exrd	执行阶段的 RD	5
	memrd	存储阶段的 RD	5
	exRegWrite	执行阶段的 RegWrite	1
	memReWrite	存储阶段的 RegWrite	1
	exMemtoReg	执行阶段的 MemtoReg	1
	user1	是否使用 rs1	1
	user2	是否使用 rs2	1
	IR	指令内容	32
	ecall	ecall 控制信号	1
	branch	存在分支	1
输出	清空	清空信号	1
	阻塞	阻塞信号	1

2.5 重定向流水线设计

重定向流水线通过对两条数据相关的指令进行数据重定向，以减少气泡的插入，从而提高流水线 CPU 的效率。为了实现这一点，需要对先前设计的冲突处理模块进行修改，使其支持数据重定向。需要特别注意的是，当两条指令存在数据相关性且前一条指令是访存指令时，为了避免冲突，仍然需要插入气泡。这样可以确保数据的正确传递与处理。

重定向冲突处理模块的输入和输出引脚定义如表 2.9 所示。

表 2.9 重定向冲突处理的输入输出引脚

类型	引脚名称	引脚含义	位宽
输入	exrd	执行阶段的 RD	5
	memrd	存储阶段的 RD	5
	exRegWrite	执行阶段的 RegWrite	1
	memReWrite	存储阶段的 RegWrite	1
	exMemtoReg	执行阶段的 MemtoReg	1
	user1	是否使用 rs1	1
	user2	是否使用 rs2	1
	IR	指令内容	32
	ecall	ecall 控制信号	1
	branch	存在分支	1
输出	清空	清空信号	1
	阻塞	阻塞信号	1
	r1_forward	重定向控制信号 1	2
	R2_forward	重定向控制信号 2	2

2.6 动态分支预测机制

为了进一步减少气泡插入的数量并提高流水线 CPU 的效率，引入了动态分支预测机制。考虑到程序的分支跳转具有局部性，通过增加一个缓存（分支历史表）来记录每次分支跳转指令的执行结果，并利用这些记录来预测未来分支跳转的执行结果。

为了提高预测的准确性，采用了双预测位策略进行分支预测。同时，为了保证预测历史的及时更新，使用 LRU（最少最近使用）算法来更新分支历史表。通过这种方式，可以在取指阶段提前预测分支的结果，从而显著提高流水线 CPU 的整体效率。

其中分支历史表通过全相联存储器实现，其设计需要存储以下内容：有效位、分支指令地址和分支目标地址。此外，为了支持 LRU 更新算法，还需要设置一个淘汰计数器，用于记录和更新最近最少使用的条目。

分支历史表的输入输出引脚定义如表 2.10 所示。

表 2.10 分支历史表的输入输出引脚

类型	引脚名称	引脚含义	位宽
输入	exPC	分支指令地址	32
	exbranch_ad	分支目标地址	32
	ifPC	分支查询地址	32
	exbranch	表更新使能位	1
	是否跳转	是否跳转	1
	clk	时钟	1
输出	预测结果	预测跳转地址	32
	预测跳转	预测是否跳转	1

动态分支预测的预测过程应在重定向流水线 CPU 的取指阶段进行，而更新过程则放在执行阶段。在执行阶段，如果发现预测结果正确，则说明取指阶段的预取指令是有效的，此时无需插入气泡；如果预测错误，则需要插入气泡，并重新从取指阶段预取正确的指令。

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）

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。

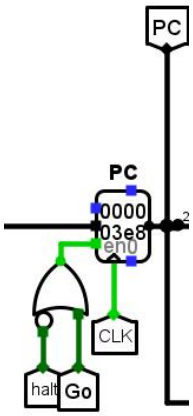

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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) 硬布线控制器

Logism 实现：

设计了一个组合逻辑电路作为硬布线控制器，用于生成单周期 CPU 中的所有控制信号。硬布线控制器的输入是指令内容 IR 的一部分，输出则是 CPU 所需的所有控制信号。具体实现如图 3.3 和图 3.4 所示。

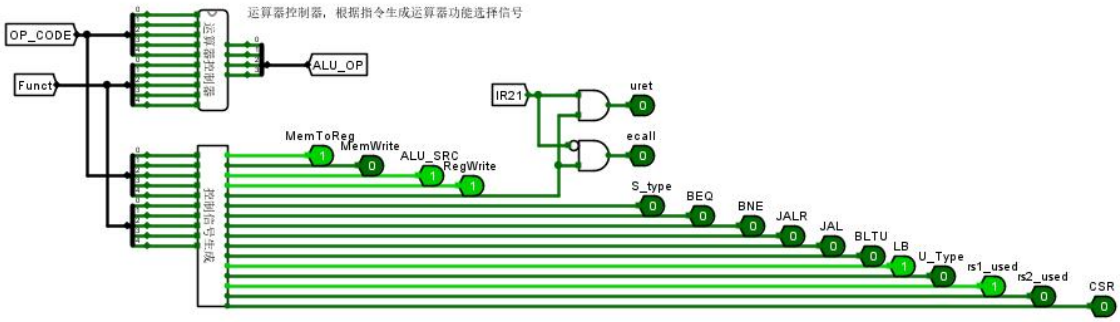

图 3.3 硬布线控制器电路

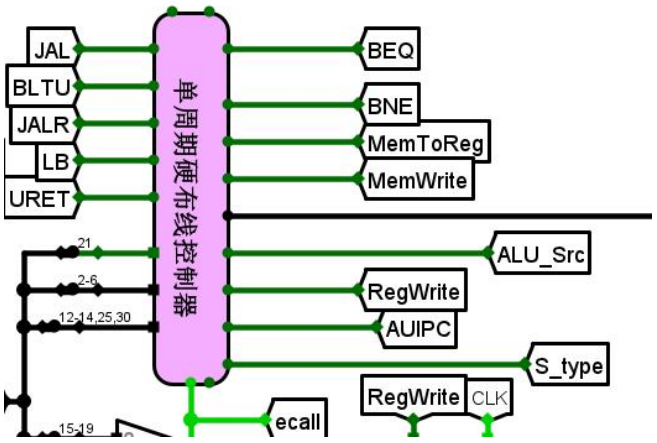

图 3.4 硬布线控制器使用

4) 寄存器堆

Logism 实现：

将 32 个 32 位寄存器封装为一个寄存器堆，其输入包括读地址、写地址以及相

关的控制信号，输出则是读出的数据内容。由于寄存器堆包含 32 个寄存器，每个寄存器为 32 位宽，因此地址线的宽度为 5 位，输出数据宽度为 32 位。Logisim 实现如图 3.5 和 3.6 所示。

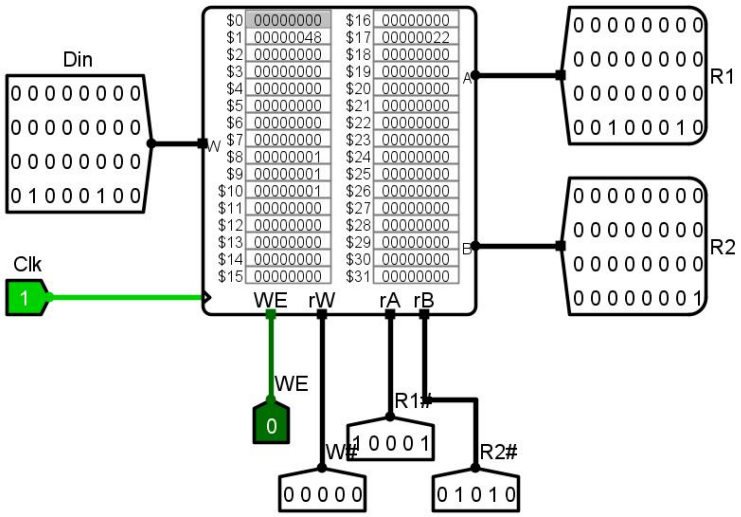

图 3.5 寄存器堆实现

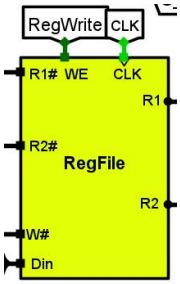

图 3.6 寄存器堆使用

5) 运算器

Logism 实现：

运算器用于执行指令的计算操作，其输入包括两个操作数和功能选择信号，输出包括各种计算和比较结果。操作数和结果的宽度均为 32 位，而功能选择信号 5 位。实现如图 3.7 和 3.8 所示。

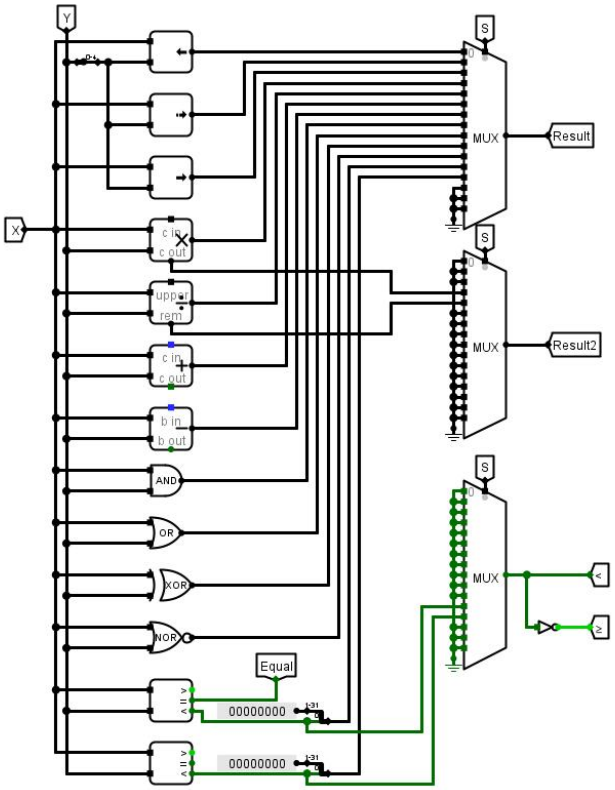

图 3.7 运算器实现

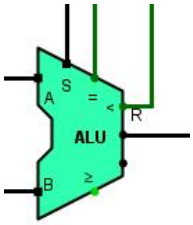

图 3.8 运算器使用

6) 数据存储器

Logism 实现：

数据存储器用于充当主存，其输入包括数据地址和若干控制信号。地址的高 10 位用于确定存储的字，低 2 位用于确定字节位置。输出为 32 位的存储内容。使用 Logisim 中的 RAM 实现，如图 3.9 所示。

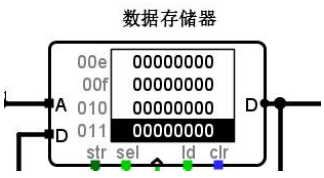

图 3.9 数据存储器

3.1.2 数据通路的实现

根据总体设计方案中的思路，在 Logisim 上实现了单周期 CPU 的数据通路。数据通路按照 CPU 的工作流程划分为五个阶段：取指、译码、执行、存储和写回。通过选择合适的逻辑门、多路选择器、优先编码器等元件，完成了单周期 CPU 数据通路的设计与实现，如图 3.10 所示。

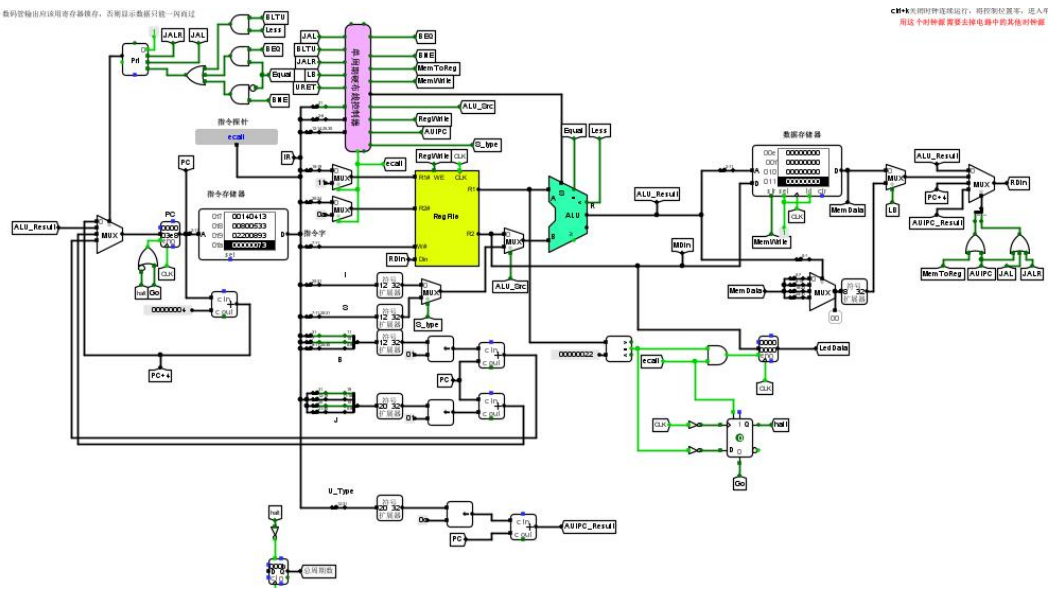

图 3.10 单周期 CPU 数据通路的实现

3.1.3 控制器的实现

根据总体设计方案中的思路，在 Logisim 上完成了控制器的实现。首先，通过填写 Excel 表生成了控制信号的逻辑表达式，然后分成运算器控制信号生成电路和其他控制信号生成电路，根据逻辑表达式自动生成电路，然后将两个模块组合起来，最终实现完整的硬布线控制器，如图 3.11 所示。

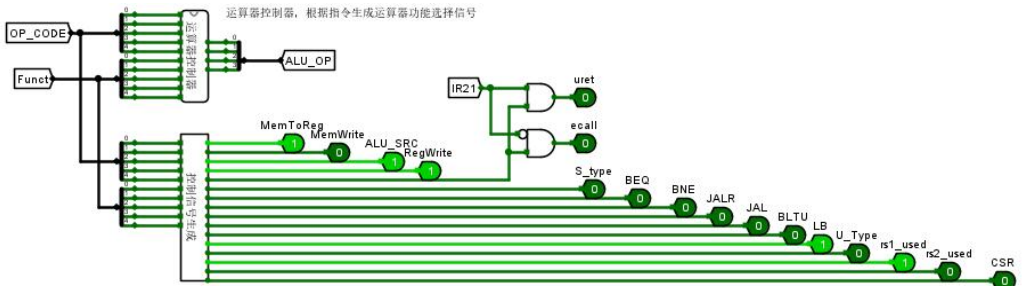

图 3.11 控制器的实现

3.2 中断机制实现

3.2.1 中断 PC 计算

中断 PC 计算数据通路的作用是计算中断发生时 PC 应该设置的值。具体实现如下：使用一个优先编码器对中断请求信号进行编码，以确定优先级最高的中断请求；使用一个多路选择器，根据中断请求信号选择相应的中断服务程序入口地址；根据 CPU 当前的状态，判断 PC 应该设置为的值；输出中断产生信号，以指示中断已被处理。实现如图 3.12 所示。

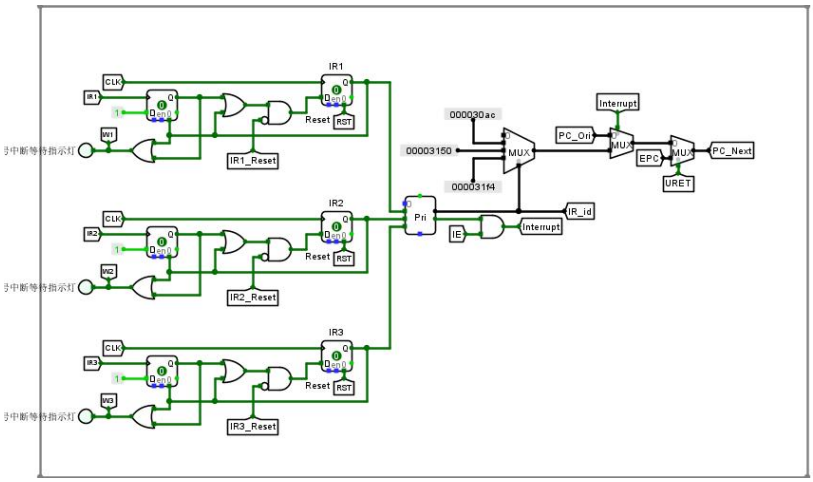

图 3.12 中断 PC 计算的实现

3.2.2 中断使能维护

中断使能维护数据通路用于设置中断使能寄存器的值，具体实现如下：设置一个 1 位寄存器，用于存储当前的中断使能信号；设计一个组合逻辑电路，根据 CPU 的当前状态判断是否需要更新中断使能信号，以及生成中断使能信号的新值；通过该逻辑电路控制寄存器的更新，确保中断使能信号能够正确反映 CPU 的状态。实现如图 3.13 所示。

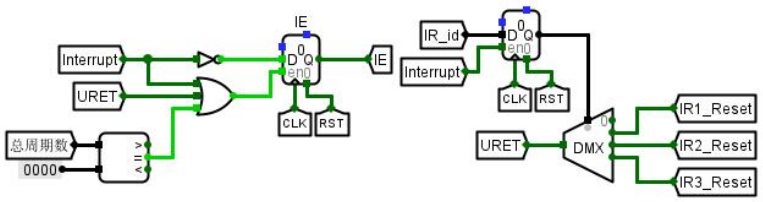

图 3.13 中断使能维护实现

3.2.3 EPC 寄存器堆维护

EPC 寄存器堆维护数据通路用于设置 EPC 寄存器的值。针对支持 3 个中断号的多级中断设计，需要设置 3 个 EPC 寄存器。其具体实现如下：使用一个解码器，根据当前中断请求信号获取当前要处理的中断号；结合 CPU 当前的状态信息，利用多路选择器决定需要存入的 EPC 寄存器的值，并更新对应中断号的 EPC 寄存器；在中断返回时，数据通路根据当前需要返回的中断号（由 IR_id 寄存器存储）输出对应 EPC 寄存器中的值。实现如图 3.14 所示。

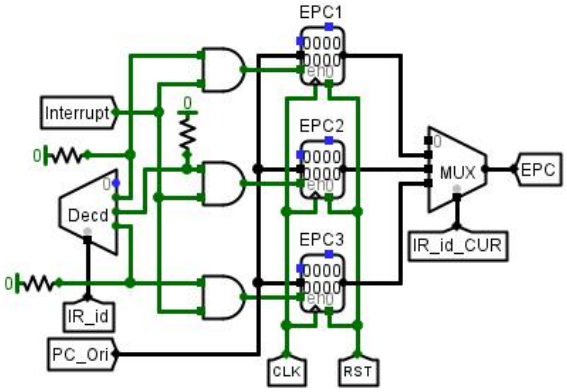

图 3.14 EPC 寄存器堆维护实现

3.2.4 中断返回信号生成

EPC 寄存器堆维护数据通路还用于在执行完中断服务程序后生成相应的中断返回信号。其具体实现如下：当当前指令为中断返回指令时，使用一个解复用器，根据当前的中断号，将对应中断的中断返回信号置为 1；该返回信号指示 CPU 完成当前中断服务程序，准备返回到中断发生前的程序状态。实现如图 3.15 所示。

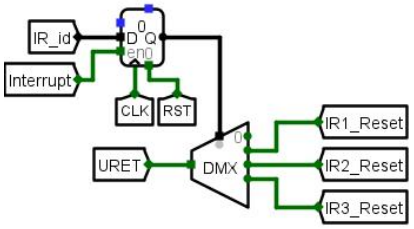

图 3.15 中断返回信号实现

3.2.5 中断优先级确定

中断优先级确定数据通路用于实现多级中断的调度，支持高优先级中断打断低优先级中断（嵌套中断）。具体实现步骤如下：

① 设置一个 IR_id 硬件栈，用于存储等待执行的中断号；

② 配备一个计数器来维护栈顶指针，确保正确管理硬件栈的压栈和弹栈操作；

③ 当有中断请求时，若该中断的优先级高于栈顶中断或栈为空，则将该中断号压栈并立即执行。否则，该中断需要等待栈顶中断返回后再执行；

④ 当栈顶中断返回时，硬件栈弹出该中断号，恢复上一个中断或主程序的状态；

⑤ 如果请求的中断优先级高于正在执行的中断，数据通路还需输出一个当前指令优先的信号，以表明当前指令应被优先执行。

实现如图 3.16 所示。

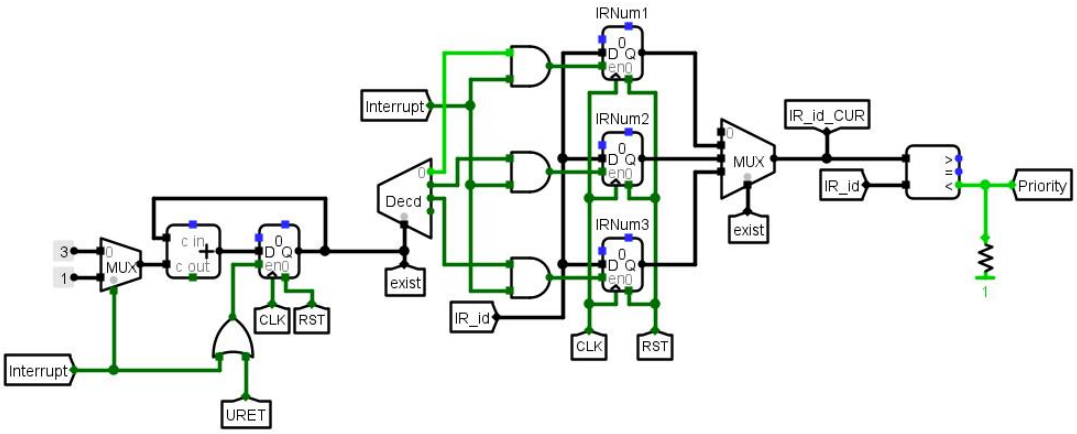

图 3.16 中断优先级确定实现

3.3 流水 CPU 实现

在本次实验中，在 Logisim 上实现了理想流水线 CPU，气泡流水线 CPU 和重定向流水线 CPU。

3.3.1 流水接口部件实现

在 Logisim 上实现了 IF/ID、ID/EX、EX/MEM 和 MEM/WB 共 4 个流水接口部

件。由于这些部件的结构和实现原则基本相同，因此这里只介绍 IF/ID 流水接口部件的实现。实现过程如下：

① 输入存储：对于每个输入信号，使用一个与其位数相同的寄存器来存储该信号的值；

② 输出连接：寄存器的输出端作为流水接口部件的输出信号；

③ 支持清空功能：为了支持流水线的清空操作，将输入信号与重置和清空信号进行逻辑与运算后，再将结果输入到寄存器中。

IF/ID 流水接口部件的 Logisim 实现如图 3.17 所示，其他三个流水接口部件的实现与此基本一致。

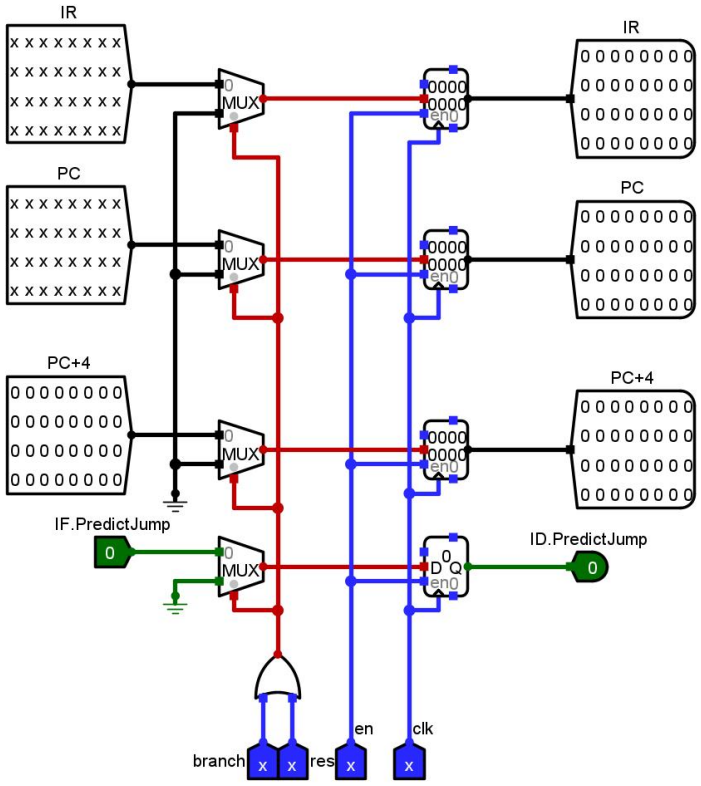

图 3.17 IF/ID 流水接口部件

3.3.2 理想流水线实现

在 Logisim 上完成了理想流水线 CP 的实现。由于单周期 CPU 是按照取指、译码、执行、存储和写回这五个阶段设计的，因此在实现理想流水线 CPU 时，仅需将单周期 CPU 的五个阶段拆分开，并在各阶段之间插入 4 个流水接口部件，就可

以实现流水线操作。实现如图 3.18 所示。

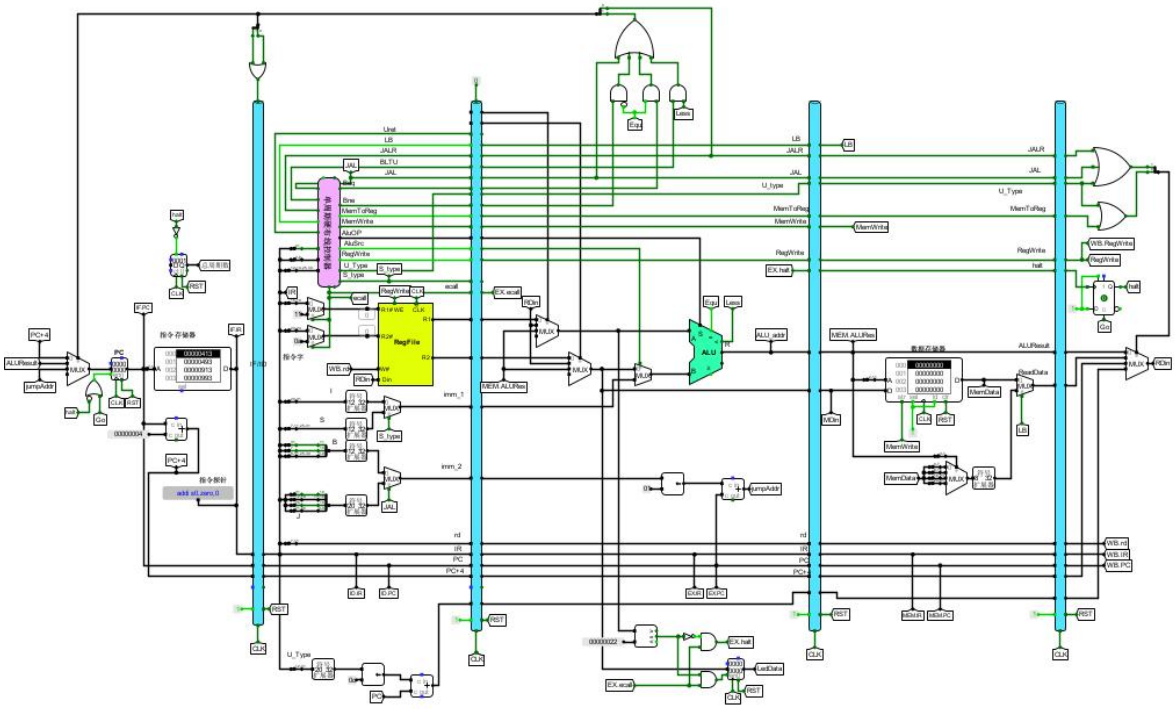

图 3.18 理想流水线 CPU 电路

3.4 气泡式流水线实现

在 Logisim 中实现了气泡冲突处理模块的电路，并将其插入到理想流水线 CPU 的电路中。气泡冲突处理电路是一个组合逻辑电路，具体实现步骤如下：

① 输入信号：输入包括执行阶段和存储阶段的 RD（目标寄存器地址）、RegWrite（寄存器写使能）等控制信号；

② 逻辑判断：通过组合逻辑电路判断是否发生数据相关冲突。如果冲突存在，则需要生成相应的信号；

③ 输出信号：

清空信号：用于清空流水接口部件中的相关数据，插入气泡。

阻塞信号：用于阻止时钟信号，暂停流水线的某些阶段运行。

实现如图 3.19 和 3.20 所示。

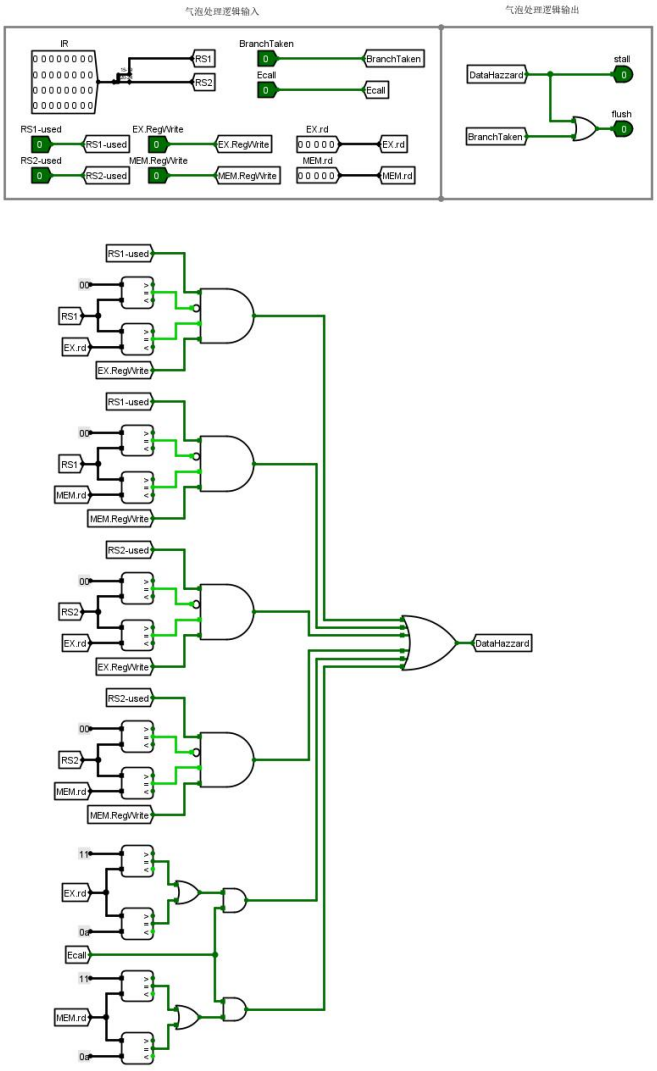

图 3.19 气泡插入实现

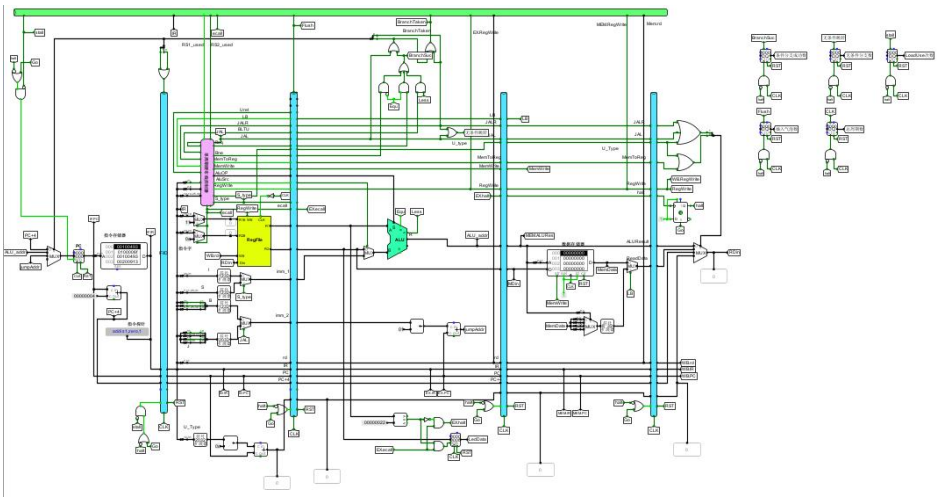

图 3.20 气泡流水线 CPU 实现

3.5 重定向流水线实现

对冲突处理电路进行了重新设计，并将其插入到已实现的气泡流水线 CPU 电路中，同时对整体电路进行了一些必要的修改。

重定向冲突处理电路的主要改进如下：

① 删减冲突信号的触发情况：仅当发生 load-use 数据相关时，才输出冲突信号和阻塞信号，从而减少气泡插入，提高流水线效率。

② 新增数据重定向支持：增设了两个 2 位输出信号（rs1_forward 和 rs2_forward），用于支持数据重定向功能。这些信号用于指示操作数 r1 和 r2 的重定向来源。根据重定向信号，流水线能够从更近的流水阶段获取操作数，而无需等待写回。

实现电路如图 3.21 和图 3.22 所示。

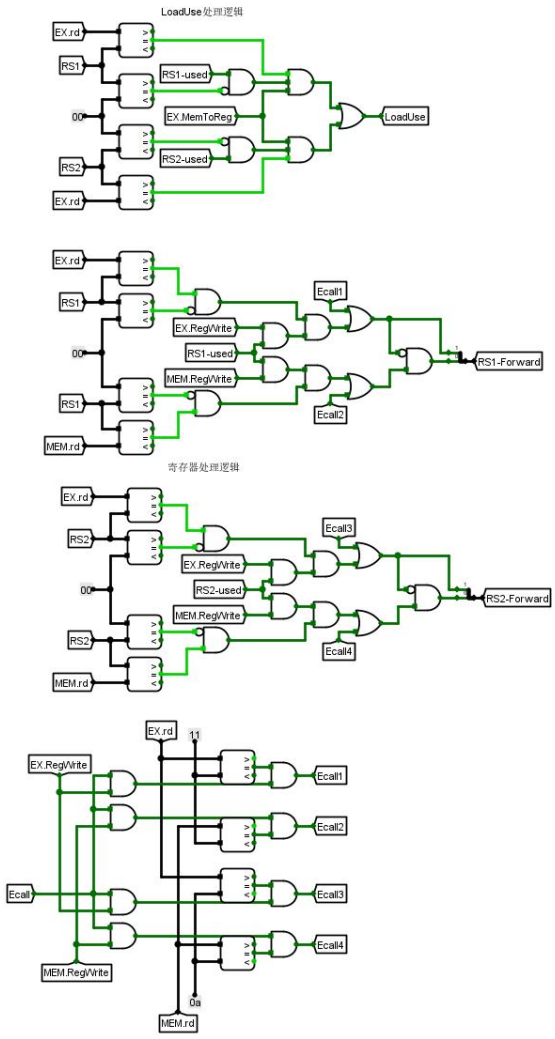

图 3.21 重定向逻辑实现

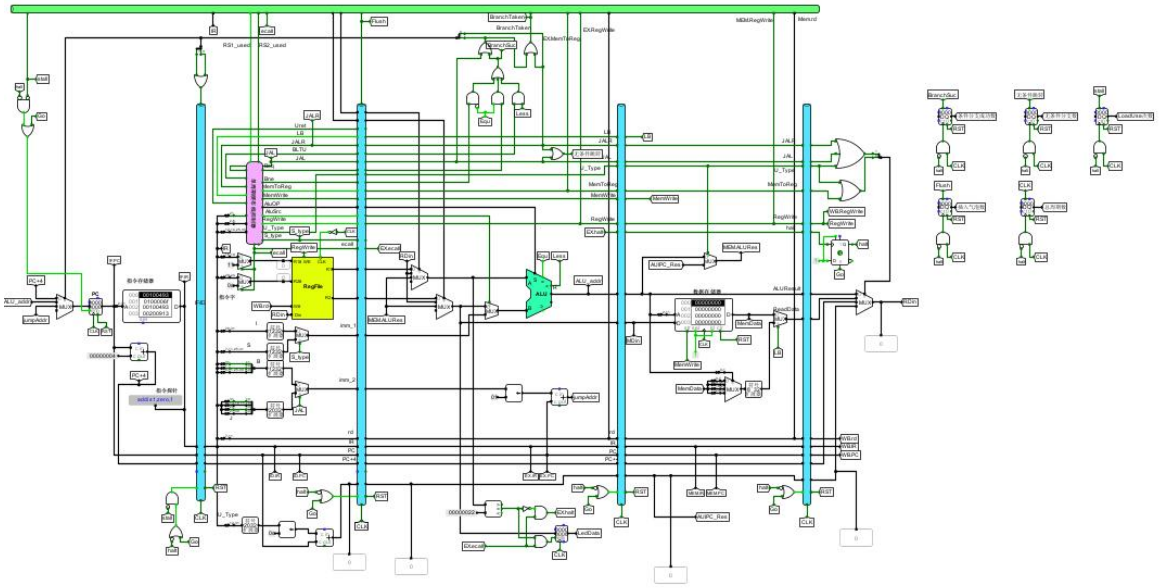

图 3.22 重定向流水线 CPU 实现

3.6 动态分支预测机制实现

实现动态分支预测机制步骤如下：

① 实现分支历史表

结构设计：分支历史表由 8 个 Cache 槽组成，每个槽包含有效位寄存器、分支指令地址寄存器、分支目标地址寄存器、淘汰计数器。

支持功能：LRU 淘汰算法，根据当前指令地址，输出预测的目标地址。

② 分支预测过程

预测阶段（IF 阶段）：在取指阶段，使用分支历史表对分支指令进行预测。将预测得到的目标地址作为 PC 寄存器的输入，通过多路选择器选择是否使用该预测地址。

验证阶段（EX 阶段）：判断分支指令的预测结果是否正确。如果预测错误，触发出错信号，重新设置 PC，清空流水线；如果预测正确，指令继续执行。

③ 最终实现

将动态分支预测电路和多级中断处理电路集成到流水线 CPU 中，形成最终的设计。

实现电路如图 3.23 和 3.24 所示。

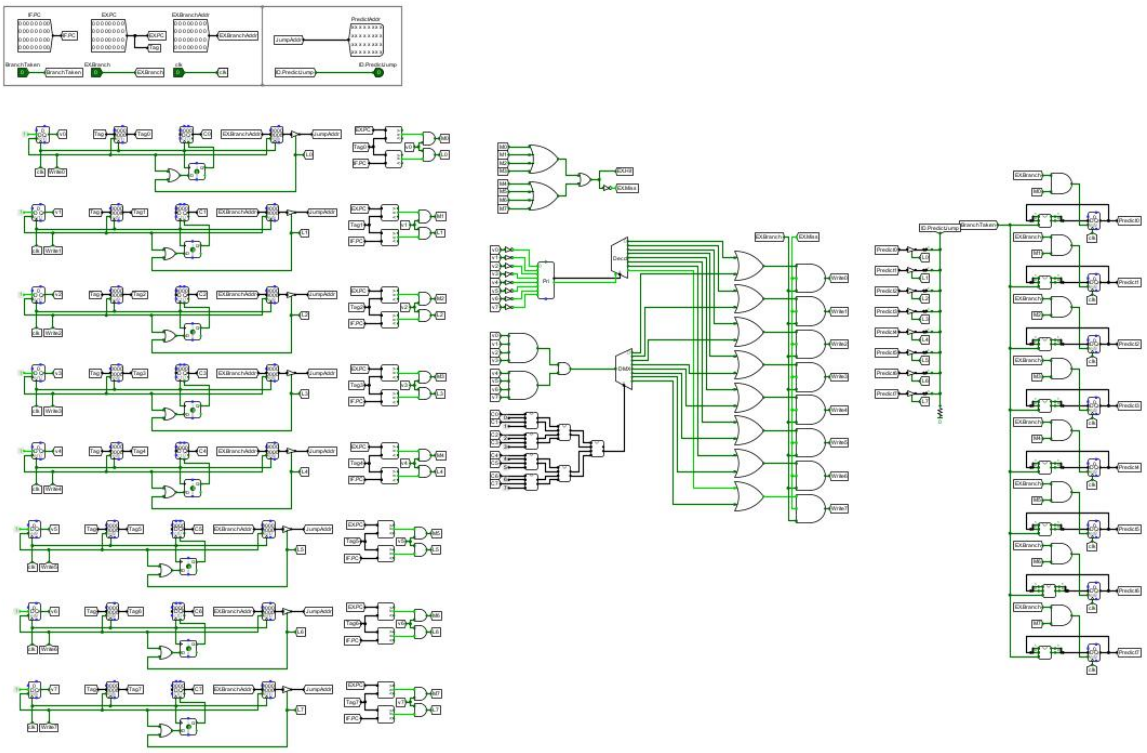

图 3.23 动态分支预测实现

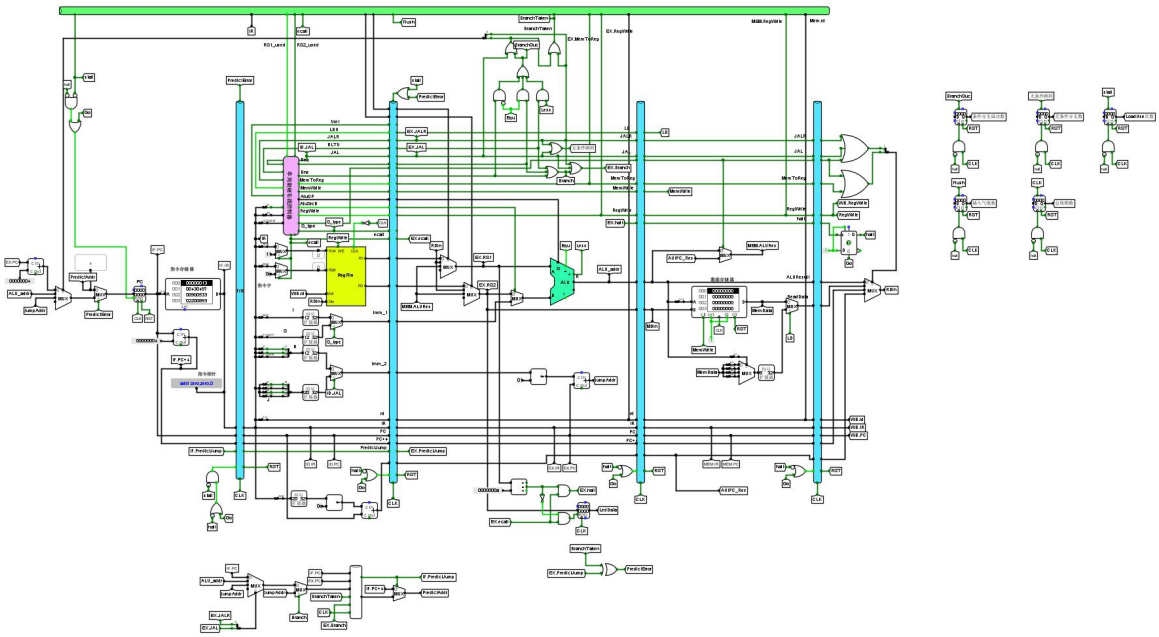

图 3.24 动态分支预测流水线 CPU 实现

4 实验过程与调试

4.1 测试用例和功能测试

本次实验中，各种 CPU 都通过了头歌平台的测试。除此之外，还使用了 3 个测试用例对各种 CPU 测试，包括 risc-v-benchmark_ccab、多级中断测试（EPC 硬件堆栈保护）、分支预测。所有 CPU 都通过了测试。

4.1.1 测试用例 - risc-v-benchmark_ccab

使用 risc-v-benchmark_ccab 测试用例对在 Logisim 实现的单周期 CPU、气泡流水线 CPU 和重定向流水线 CPU 进行了测试。测试结果如图 4.1。

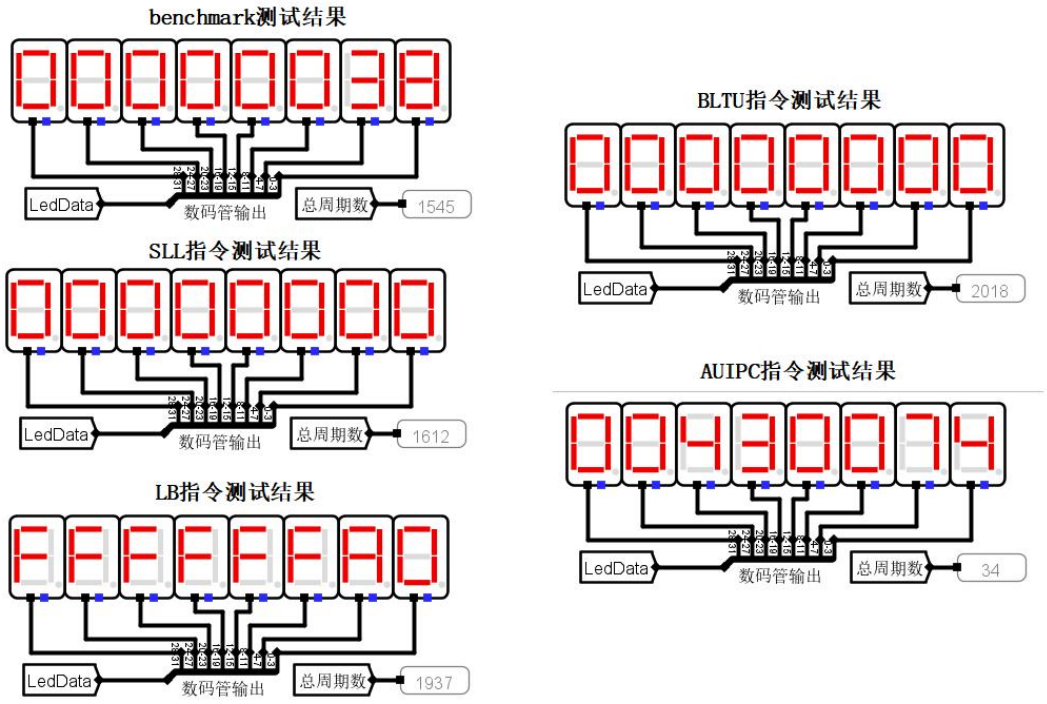

图 4.1 CPU 测试结果

4.1.2 测试用例 - 多级中断测试 (EPC 硬件堆栈保护)

使用多级中断测试 (EPC 硬件堆栈保护) 测试用例对单周期多级中断 CPU 和

重定向流水线多级中断 CPU 进行了测试。两者均通过了测试。由于中断过程难以以图片显示，这里不作展示。

4.1.3 测试用例 - 分支预测测试

使用分支预测测试用例对动态分支预测 CPU 进行了测试，测试结果如图 4.2 所示。

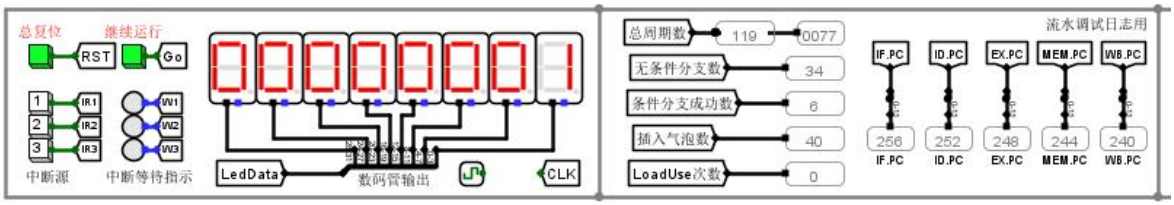

图 4.2 动态分支预测 CPU 测试结果

4.2 性能分析

在气泡流水线 CPU、重定向流水线 CPU、动态分支预测 CPU 三种 CPU 上运行了 risc-v-benchmark.hex 测试用例，并统计了所需周期数，如表 4.1 所示。

表 4.1 测试结果

CPU 类型	周期数
气泡流水线 CPU	3623
重定向流水线 CPU	2297
动态分支预测 CPU	1781

数据分析：

① 气泡流水线 CPU：周期数最多，因为在发生数据相关和分支跳转时，会插入大量气泡以确保正确性。

② 重定向流水线 CPU：通过数据重定向减少了部分气泡插入，显著降低了周期数，性能相比气泡流水线 CPU 提升明显。

③ 动态分支预测 CPU：引入动态分支预测机制，通过在取指阶段提前预测分支跳转的方向并减少错误预测导致的流水线清空次数，进一步优化了性能，周期数最低。

4.3 主要故障与调试

4.3.1 指令控制信号故障

故障现象：空指令时 RegWrite 的信号为 1。

原因分析：直接使用表格生成的逻辑表达式会导致在硬布线控制器中输入信号全为 0，也就是空指令时，RegWrite 的信号为 1。在运行某些测试集时会导致出错。

解决方案：单独修改硬布线控制器电路，在分析组合逻辑电路处将输入信号为 0 时的输出信号全部改成 0。

4.3.2 寄存器故障

故障现象：无法正确地写回数据到寄存器中。

原因分析：寄存器的触发方式为下降沿，但在电路中需要改成上升沿，这样在一个周期内，数据就可以在周期的后半周期进行写回寄存器。

解决方案：修改寄存器的触发方式为上升沿。

4.3.3 重定向故障

故障现象：重定向流水线 CPU 中，测试 AUIPC 指令时无法正确地进行重定向。

原因分析：当重定向源来自存储阶段运算器计算结果时，直接使用了执行阶段传递到存储阶段的运算器计算结果。但根据 AUIPC 指令功能，需要使用 AUIPC 部分电路在译码阶段计算出的结果，而不是执行阶段 ALU 计算的结果。

解决方案：当重定向源来自存储阶段运算器计算结果时，将来源修改成 AUIPC 计算结果，如图 4.3 所示。

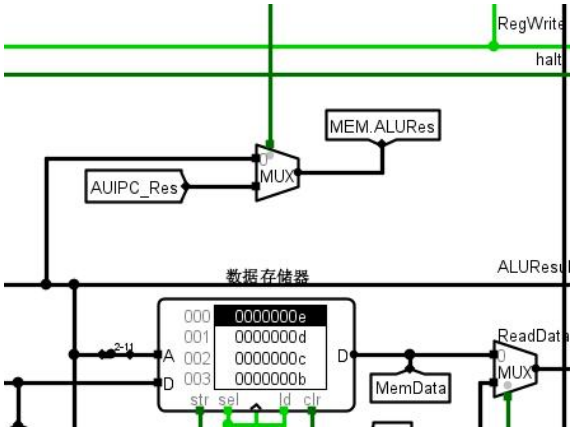

图 4.3 重定向故障解决方案

4.4 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	完成理想流水线的 Logisim 实现。
第五天	完成气泡流水线的 Logisim 实现。
第六天	完成重定向流水线的 Logisim 实现。
第七天	完成单周期 CPU 单机中断、多级中断的 Logisim 实现。
第八天	完成重定向流水线 CPU 多级中断的 Logisim 实现。
第九天	学习动态分支预测的相关知识，开始进行动态分支预测的实现。
第十天	完成基于重定向流水线 CPU 上的动态分支预测。

5 团队任务

5.1 概述

实现的内容：基于 Logisim 实现运行在 risc-v32 单周期 CPU 上的井字棋游戏。

组员：CS2202 陈薛嘉，CS2202 何志鑫，CS2202 赵亦安，CS2208 朱昊天

游戏规则：井字棋是一种两人对战游戏，玩家 A 和玩家 B 分别使用“X”和“O”作为标记， 在一个 3×3 的棋盘上轮流下棋。玩家 A 先手，每次只能在未被占据的空格中放置自 己的标记。游戏的目标是率先在横向、纵向或对角线上连成 3 个相同的标记，即可获胜。如果棋盘被填满且无人达成胜利条件，则游戏以平局结束。

游戏过程中，玩家不得覆盖对方的标记，也不能在已被占据的格子中放置标记，违规操作需重新开始 RST。

游戏界面如图 5.1 所示。

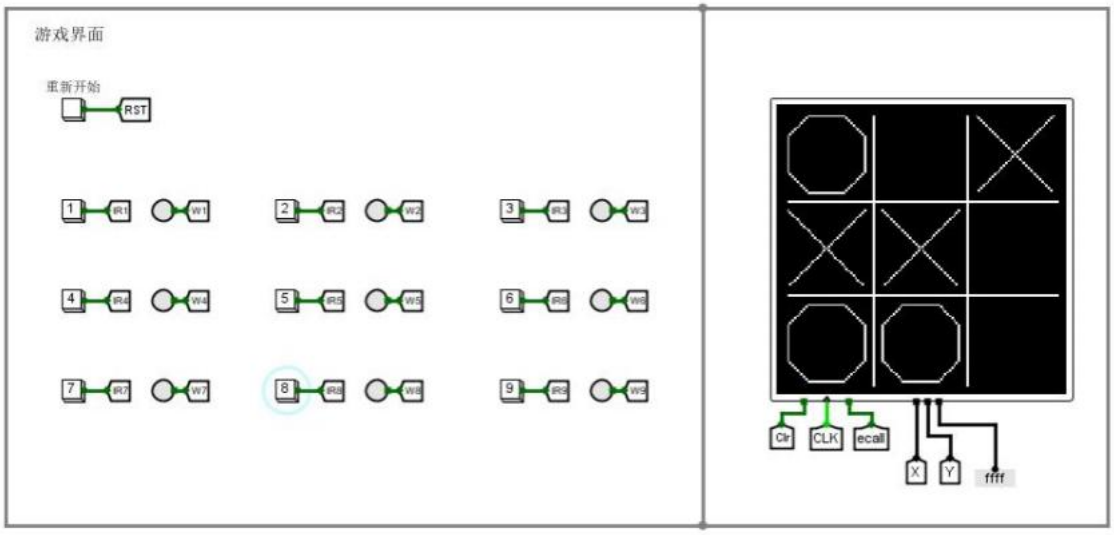

图 5.1 井字棋游戏界面

5.2 显示逻辑

整个游戏的显示界面如图 5.2 所示。

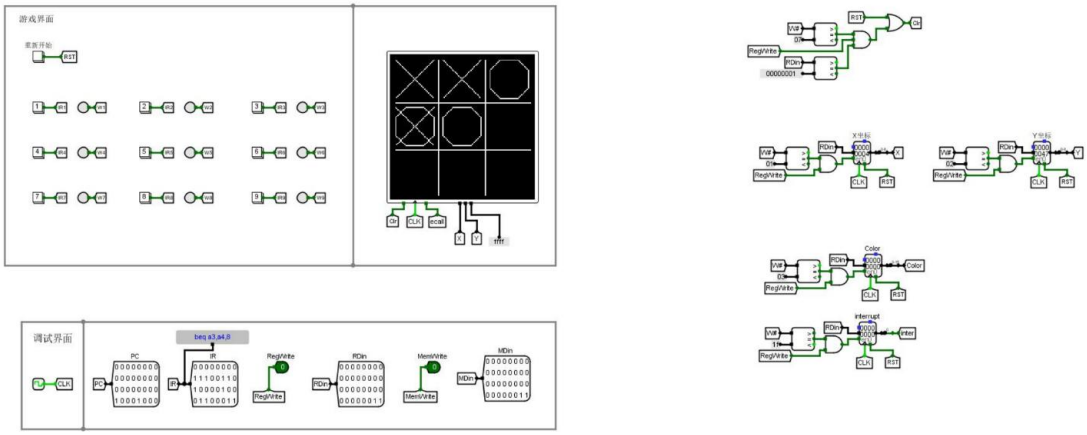

图 5.2 井字棋游戏显示逻辑

5.3 寄存器使用规范

1. $1（ra）,$2（sp）寄存器分别写入 LCD 的 X 坐标和 Y 坐标。（X，Y 坐标各位 7 位）

2. $3（gp）寄存器存放 LCD 的颜色（16 位），线段画白色（ffff），X 画黄色（fff0），○画绿色（f00）

3. $31（t6）用于开关中断，为 1 为关中断，为 0 为开中断

4. $18（s2），$19（s3）用于存放当前 X 或○的中心。

5. $11（a1）寄存器用于存放下棋的用户，0 表示 X 在下棋，1 表示○在下棋

6. $12（a2）寄存器用于记录下棋数，最多到第 9 次还没有结束游戏时，判断平局

7. $13（a3）寄存器用于记录当前棋盘状态，为 0 时表示正在下棋，为 1 时表示 X 胜利，为 2 时表示○胜利，3 表示平局

8. $5（t0）表示 X 下的格子，$6（t1）表示○下的格子。从 0~8 位依次表示 1~9 个格子

9. $7(t2)表示是否清空界面，置 1 为清空界面。用于结束比赛时，重新绘制 LCD

10. $14 $15 $16（a4，a5，a6）表示可待选择寄存器

6 设计总结与心得

6.1 课设总结

本次实验中总共在 Logisim 上设计实现并完成了任务：

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 的单机中断、多级中断支持
6. 重定向流水线 CPU 的多级中断支持
7. 重定向流水线 CPU 的动态分支预测支持
8. 团队任务 - 井字棋游戏

6.2 课设心得

这是一次与硬件相关的大型项目，对于硬件开发经验较少的我而言，这次课设的难度可想而知。然而，通过近两周的不懈努力，我从零开始设计并实现了一台 CPU，成功完成了课设的大部分任务。

在 Logisim 上设计和实现单周期 CPU 及理想流水线 CPU 时，由于在上学期的组成原理实验中积累了一些基础，加上指导手册内容详尽，完成得较为顺利。然而，在实现中断机制、气泡流水线 CPU 和重定向流水线 CPU 时，遇到了很多错误。但通过不断 debug，我还是逐一解决了问题。在个人任务最后的动态分支预测 CPU 设计与实现过程中，我大量查阅资料，大量的尝试，最后才完成动态分支预测。而在团队任务中，最后也和几位队友一起实现了井字棋的小游戏，进一步加强了对 Logisim 的掌握，也对 CPU 有更深的了解。总的来说，这次实验不仅让我更深刻地理解了组成原理中关于 CPU 的知识，也让我在实践中不断进步，收获颇丰。

同时，我也对课设提出以下建议：建议添加关于上板部分的资料和教程，对于没有选修 Verilog 的同学来说，这部分无从下手，而且网上资料较少，使得上板任务难以完成。

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: