华中科技大学

2020

计算机组成原理　　课程设计报告

题　　目：	计算机组成原理课程设计
专　　业：	计算机科学与技术
班　　级：	ACM1701
学　　号：	U201714780
姓　　名：	刘晨彦
电　　话：	15927172332
邮　　件：	805132500@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	
13	NOR	或非	

#	指令助记符	简单功能描述	备注
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	LUI	立即数加载至高位	
29	SLTIU	小于立即置 1（无符号）	
30	LB	加载字节	
31	BLTZ	小于 0 转移	

2 总体方案设计

2.1 单周期 CPU 设计

本次我们采用的方案是经典的单周期硬布线 MIPS CPU 设计方案，即每条指令的执行时间固定，指令采用硬布线译码方式以及指令存储器和数据存储器分开的设计特点。

总体结构图如图 2.1 所示。

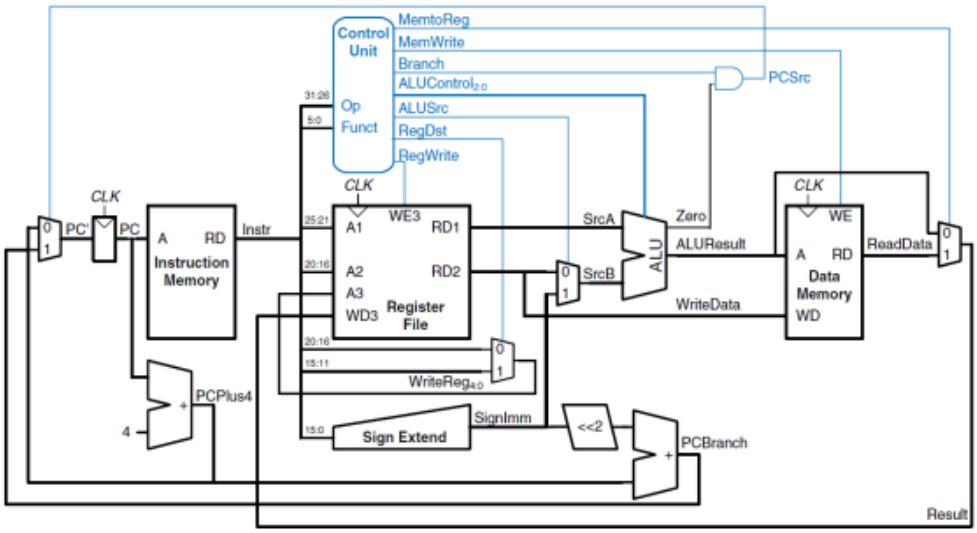

图 2.1 总体结构图

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

程序计数器简单的采用了寄存器原件，在上升沿进行更新，同时使能端由系统停机信号控制，即当出现 Syscall 指令且寄存器$v0 的值为 10 时，PC 寄存器和系统一起停止工作。

2. 指令存储器 IM

指令存储器由只读存储器元件实现，地址位宽为 10，数据位宽为 MIPS 单条指令的长度，即 32 位。

3. 运算器

表 2.1 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
A	输入	32	操作数 X
B	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码
Shamt	输入	5	位移位数
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
=	输出	1	当 A、B 相等时为 1，其余时刻为 0

4. 寄存器堆 RF

寄存器堆使用外部 Logisim 库“CS3410 Component”中的 Register File 实现，支持同时读取两个寄存器内的数据，并且向一个寄存器内写入数据。

2.1.2 数据通路的设计

数据通路的主要流向是：IF 段指令流出指令存储器，ID 段读取寄存器，EX 段流入 ALU 计算，MEM 段写入数据存储器，WB 段数据写回寄存器。通过对各条指令的流动情况分析，能够得到对应的控制信号，设计单周期硬布线信号控制器。

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表 2.2。

表 2.2 主控制器控制信号的作用说明

控制信号	取值	说明
JMP	0	当前指令非 J 型指令
	1	当前指令为 J 型指令，需要进行转跳
JR	1	当前指令为 JR
JAL	1	当前指令为 JAL
SignedExt	1	数据需要进行符号拓展
BLTZ	1	当前指令为 BLTZ
LUI	1	当前指令为 LUI
BEQ	1	当前指令为 BEQ 指令
BNE	1	当前指令为 BNE 指令
LB	1	当前指令为 LB 指令
MemToReg	0	ALU 计算结果送回寄存器
	1	RAM 数据送回寄存器
MemWrite	1	RAM 写使能信号
ALUOP	$(0\sim13)_{10}$	ALU 的运算符号
AluSrcB	0	寄存器 R2 数据送入 ALU 运算器 B 端口
	1	拓展立即数送入 ALU 运算器 B 端口
RegWrite	1	寄存器堆写使能信号
RegDst	0	rt 作为写入寄存器 W#编号
	1	rd 作为写入寄存器 W#标号
Syscall	1	当前指令为 Syscall 指令
ERET	1	当前为中断返回指令
关中断	1	当前为关中断指令
开中断	1	当前为开中断指令

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 2.3 所示。

表 2.3 主控制器控制信号框架

指令	AluOP	MenToReg	MenWrite	AluSrc	RegWrite	RegDst	JMP	SingedExt
SLL	0	0	0	0	1	1	0	0
SRA	1	0	0	0	1	1	0	0
SRL	2	0	0	0	1	1	0	0
ADD	5	0	0	0	1	1	0	0
ADDU	5	0	0	0	1	1	0	0
SUB	6	0	0	0	1	1	0	0
AND	7	0	0	0	1	1	0	0
OR	8	0	0	0	1	1	0	0
NOR	10	0	0	0	1	1	0	0
SLT	11	0	0	0	1	1	0	0
SLTU	12	0	0	0	1	1	0	0
JR	X	0	0	0	0	0	1	0
SYSCALL	X	0	0	0	0	0	0	0
J	X	0	0	0	0	0	1	0
JAL	X	0	0	0	1	0	1	0
BEQ	X	0	0	0	0	0	0	0
BNE	X	0	0	0	0	0	0	0
ADDI	5	0	0	1	1	0	0	0
ANDI	7	0	0	1	1	0	0	0
ADDIU	5	0	0	1	1	0	0	0
SLTI	11	0	0	1	1	0	0	0
ORI	8	0	0	1	1	0	0	1
LW	5	1	0	1	1	0	0	0
SW	5	0	1	1	0	0	0	0
LUI	0	0	0	1	1	0	0	0
SLTIU	12	0	0	1	1	0	0	0

指令	AluOP	MenToReg	MenWrite	AluSrc	RegWrite	RegDst	JMP	SingedExt
LB	5	1	0	1	1	0	0	0
BLTZ	11	0	0	0	1	0	0	0

除上表所述控制信号外，还有 SYSCALL、BEQ、BNE、JR、JAL、LUI、BLTZ、LB 控制信号，此类控制信号仅为对应指令所设计，其他指令不会产生该类控制信号，故不纳入表中。

2.2 中断机制设计

2.2.1 总体设计

由于课程设计内容中不包括中断矢量表，因此中断程序地址通过常量的方式写在硬件中。对单级中断，中断过程不能被打断，仅当一个中断过程完成后，才会选择当前等待的中断中优先级最高的中断进入。对多级中断，中断可以被优先级更高的中断所打断，当中断完成后，CPU 继续执行被打断的优先级最高的中断或返回正常程序。

2.2.2 硬件设计

首先设计中断信号采样电路。当中断产生时，等待指示灯立即亮起。在下一个时钟上升沿，产生对应中断号的中断屏蔽信号。当中断的同步清零信号亮起时，中断屏蔽信号会在下一个时钟上升沿消失。

对各中断屏蔽信号，通过优先编码器来控制中断的优先级。

设置信号量“中断信号”，当前不在中断程序且出现中断屏蔽信号时，“中断信号”为 1，同时记录中断编号。

设置信号量 IE，当 IE 为 1 时，当前正处在中断中。IE 由寄存器控制，当前不在中断程序且出现中断屏蔽信号时寄存器输入 1，当程序到达 ERET 指令时，寄存器 IE 清零。

当前不在中断程序且出现中断屏蔽信号时，中断号送入多路选择器选择中断程序首地址；当出现 ERET 信号时，将寄存器 EPC 锁存地址送入 PC。其余时刻的 PC 值，包括 PC+4 和正常程序转跳，由 CPU 其他部件计算更新。

当出现 ERET 指令时，根据记录的中断编号，将相应的中断清零信号置 1，标志

该中断完成。

对于多级中断，则有以下修改：

首先通过硬件实现 EPC 堆栈和中断编号记录堆栈，以实现中断嵌套。

其次 IE 在关中断和“中断信号”两种情况下置为 1，在 ERET 和开中断两种情况下置为 0。

中断清零信号现在由中断编号和开中断信号共同控制。由于中断清零信号会在中断最终完成前亮起，故需要屏蔽比当前中断号更低的中断屏蔽信号。

2.2.3 软件设计

在使用 Mars4_5 将汇编程序转换为二进制指令后，观察各中断地址，将终端地址写入硬件中以实现中断地址转跳。

2.3 流水线 CPU 设计

2.3.1 总体设计

流水线 CPU 设计的原则就是将 CPU 分割成几个时间上串行的段，每个段可以独立的实现指令的一部分要求，通过流水线的方式提高 CPU 处理效率。课程设计选择了经典的五段式流水 CPU，将程序分为 IF 段、ID 段、EX 段、MEM 段、WB 段，功能分别为：取指、译码、执行、访存、写回。

2.3.2 流水接口部件设计

流水线接口部件设计需要考虑一条指令的各个控制信号分别作用于哪一段，需要将这些控制信号，通过寄存器的方式和程序一起在流水线上向后传递，直到到达作用段，控制流水线部件，实现指令。

除了考虑接口部件需要传递的控制信号外，还需要完成同步清零，以防止毛刺等问题的出现。其余接口还包括时钟信号和复位信号。

在理想 IF/ID 接口部件中，需要传递指令 IR、PC 和 PC+4。

在理想流水 ID/EX 接口部件中，需要传递信号：BEQ、BNE、MenToReg、MemWrite、AluOP、AluSrcB、RegWrite 和 Syscall，以及信号和地址：R1#、R2#、写入寄存器编号、拓展立即数、PC 和 PC+4。

在理想 EX/MEM 接口部件中，需要传递信号 MenToReg、MemWrite、RegWrite、和 Syscall，以及数据：ALU 计算结果 R、R2#、写入寄存器地址和 PC

在理想 MEM/WB 接口部件中，需要传递信号 MenToReg、RegWrite 和 Syscall，以及数据和地址：ALU 计算结果 R、RAM 读出的数据、写入寄存器编号和 PC。

在气泡和重定向 IF/ID 接口部件中，需要传递指令 IR、J 型指令地址 Jaddr、PC 和 PC+4。

在气泡和重定向 ID/EX 接口部件中，需要传递信号 LB、Syscall、RegWrite、JAL、MenToReg、MenWrite、LUI、BLTZ、BEQ、BNE、JMP、JR、AluOP 和 AluSrcB，以及数据或地址：R1#、R2#、写入寄存器编号、位移量、扩展立即数、J 型指令转跳地址 Jaddr、PC 和 PC+4。

在气泡和重定向 EX/MEM 接口部件中，需要传递信号 LB、Syscall、RegWrite、JAL、MenToReg 和 MenWrite，以及数据或地址：R1#、R2#、ALU 计算结果 R、写入寄存器编号、PC 和 PC+4。

在气泡和重定向 MEM/WB 接口部件中，需要传递信号 LB、Syscall、RegWrite、JAL 和 MenToReg，以及数据或地址：R1#、R2#、ALU 计算结果 R、RAM 读出的数据、写入寄存器编号、ALU 计算结果低两位、PC 和 PC+4。

2.3.3 理想流水线设计

理想指令流水线设计中秉持从简原则，删除了 J 型指令对 R1#和 R2#端口的选择器件。由于未来的转跳指令在 EX 段执行，故转跳的判断和目的地址都传递到 EX 段。又由于写回操作在最后一段，因此需要将写入寄存器编号、RegWrite 信号等原先直接连接到 RegFile 的信号量传递到 WB 段。

2.4 气泡式流水线设计

气泡式流水线需要解决两种冲突：转跳冲突和数据冲突。

对于转跳冲突，当送入 PC 寄存器的地址不是 IF.PC+4 时，则可以判定为转跳，在 IF/ID 流水部件和 ID/EX 流水部件插入气泡。

对于数据冲突，则稍微复杂一些。首先根据每条指令的特点，判断其是否使用了寄存器堆 R1 口和 R2 口来读取寄存器内的数据。将 ID 段指令对寄存器的使用情况、EX 段对寄存器的使用情况以及 MEM 对寄存器的使用情况。若出现写后读的情况，

则需要在 ID/EX 段流水部件中插入一个气泡，同时控制 PC 寄存器和 IF/ID 流水接口部件使能为低电平（高电平有效）。具体的判断逻辑则会在第三节具体实现中进行详细描述。

2.5 重定向流水线设计

重定向和气泡流水线在处理分支冲突时的解决办法一致，都是在 IF/ID 流水接口部件和 ID/EX 流水接口部件中插入气泡来实现的。不同之处在于重定向流水线对于数据冲突的处理。

对于数据冲突，重定向流水线的策略是，将 MEM 段的数据或 WB 段的数据先送入之后指令需要使用的位置上，再按照正常顺序送入寄存器堆。因此除了 LW 指令由于取数据所需时间较长达不到满足的维持时间以外，其余写后读冲突均可得到解决。LW 指令所带来的写后读冲突，则仍然可以使用插入气泡的方式解决。对于哪些位置的数据送到哪里、如何选择数据等具体的实现逻辑，则会在第三节具体实现中进行详细描述。

2.6 动态分支预测机制

动态分支预测的关键在于对于转跳指令的判断。动态分支预测流水线电路基于重定向流水线修改而来。动态分支预测的两个功能是：查询 IF 段指令是否会带来指令转跳，并判断是否要在下一个时钟周期给出转跳目标地址，或是简单给出 IF.PC+4。第二个功能是，根据 EX 段指令及其地址和 ID 段指令地址，判断程序是否出现分支冲突，并根据情况对 BHT 进行写入或修改，提高 BHT 的判断准确度。

在实际设计中，将正常情况下的转跳指令分支地址计算和 BHT 功能集成在一起，以方便对 BHT 写入转跳目的地址。BHT 的主体是全相连并发比较 Cache，通过对 IF.PC 的比较给出最终下一个时钟的指令地址，通过对 EX.PC 的比较给出对 BHT 的更新。具体的实现逻辑会在第三节详细设计中给出。

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）

使用一个 32 位寄存器实现程序计数器 PC，触发方式为上升沿触发，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。Pause 为停机信号，当停机指令出现，且寄存器$v0 的值为 0xA 时，Pause 为 1。通过非门后，仅当需要停机时以低电平屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

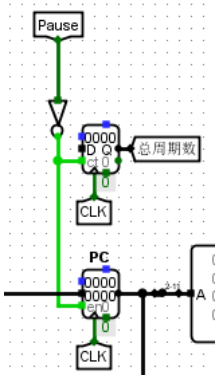

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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。

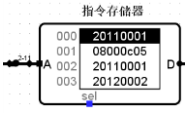

图 3.2 指令存储器（IM）

3.1.2 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现

方法为，对于每一条指令，将其改写成 RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据 RTL 功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 3.1 所示。

表 3.1 指令系统数据通路表

#	指令	PC	IM	RF				ALU			DM	
				R1#	R2#	W#	Din	A	B	OP	Addr	Din
1	SLL	PC+4	PC		rt	rd	alu	r2	立即数	0		
2	SRA	PC+4	PC		rt	rd	alu	r2	立即数	1		
3	SRL	PC+4	PC		rt	rd	alu	r2	立即数	2		
4	ADD	PC+4	PC	rs	rt	rd	alu	r1	r2	5		
5	ADDU	PC+4	PC	rs	rt	rd	alu	r1	r2	5		
6	SUB	PC+4	PC	rs	rt	rd	alu	r1	r2	6		
7	AND	PC+4	PC	rs	rt	rd	alu	r1	r2	7		
8	OR	PC+4	PC	rs	rt	rd	alu	r1	r2	8		
9	NOR	PC+4	PC	rs	rt	rd	alu	r1	r2	10		
10	SLT	PC+4	PC	rs	rt	rd	alu	r1	r2	11		
11	SLTU	PC+4	PC	rs	rt	rd	alu	r1	R2	12		
12	JR	$31	PC	rs								
13	SYSCALL	NULL/PC+4	PC	2	4							
14	J	$(PC)_{28\text{-}31}$<<28|$(IR)_{0\text{-}25}$<<2	PC									
15	JAL	$(PC)_{28\text{-}31}$<<28|$(IR)_{0\text{-}25}$<<2	PC									
16	BEQ	PC+4/imm<<2+PC+4	PC	rs	rt			r1	r2			
17	BNE	PC+4/imm<<2+PC+4	PC	rs	rt			r1	r2			
18	ADDI	PC+4	PC	rs		rt	alu	r1	立即数	5		
19	ANDI	PC+4	PC	rs		rt	alu	r1	立即数	7		
20	ADDIU	PC+4	PC	rs		rt	alu	r1	立即数	5		
21	SLTI	PC+4	PC	rs		rt	alu	r1	立即数	11		

#	指令	PC	IM	RF				ALU			DM	
				R1#	R2#	W#	Din	A	B	OP	Addr	Din
22	ORI	PC+4	PC	rs		rt	alu	r1	立即数	8		
23	LW	PC+4	PC	rs		rt	alu	r1	立即数	5	R_{2-9}>>2	
24	SW	PC+4	PC	rs	rt			r1	立即数	5	R_{2-9}>>2	r2
25	LUI	PC+4	PC	rs		rt	alu	r1	立即数	0		
26	SLTIU	PC+4	PC	rs		rt	alu	r1	立即数	12		
27	LB	PC+4	PC	rs		rt	alu	r1	立即数	5	R_{2-9}>>2	
28	BLTZ	PC+4/imm<<2+PC+4	PC	rs				r1	0	11		

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，表，将各个主要功能部件进行连接，根据数据通路合并表的最终结果，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建。

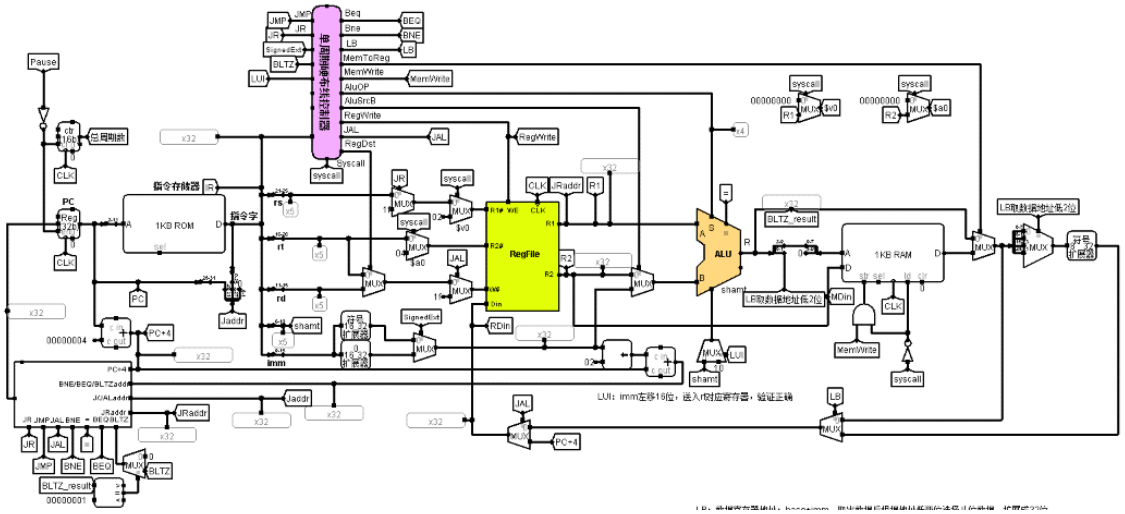

图 3.3 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

根据总体方案设计中表 2.4 相关内容，在 Logism 具体实现。

硬布线控制器信号生成电路如下图所示：

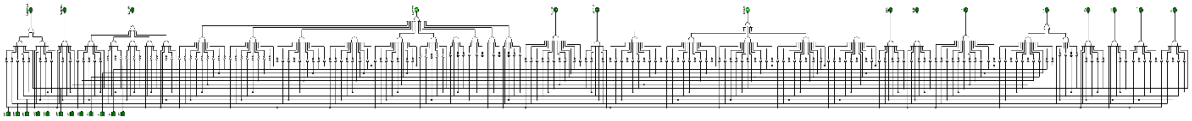

图 3.4 单周期硬布线电路原理图

硬布线运算控制器自动生成电路如下图所示：

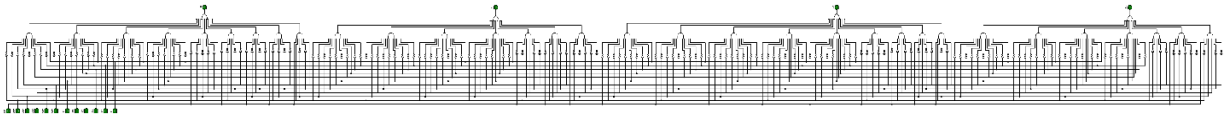

图 3.5 单周期硬布线运算控制器电路原理图

3.2 中断机制实现

3.2.1 单周期 MIPS CPU 单级中断的实现

单周期 MIPS CPU 单级中断的实现主要有以下几个组成部分：中断按键参考电路、中断编号优先编码器、实际有效中断信号产生逻辑、中断清零信号及信息记录信号产生逻辑、中断返回地址 EPC 寄存器、中断使能 IE 寄存器、中断编号寄存器和中断地址选择器。

单周期 MIPS CPU 单级中断电路的总体设计如下图所示：

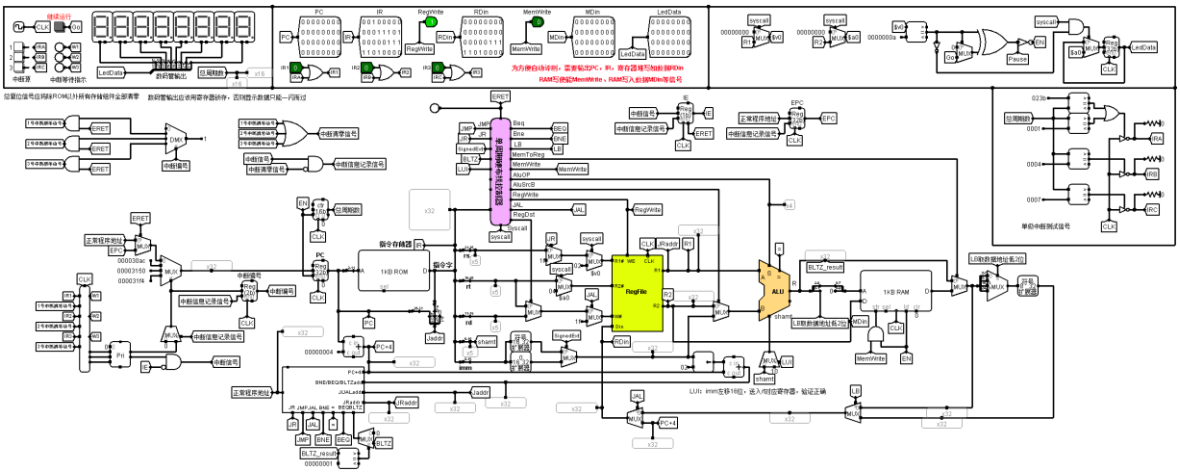

图 3.6 单周期 MIPS CPU 中断总体电路设计

其中，中断按键参考电路按照给出模板实现。

1. 中断编号优先编码器的实现

对于多个中断编号同时出现中断请求时，需要通过优先编码器赋予优先级，以确定首先执行哪一个中断。中断优先编码器的实现如下图所示：

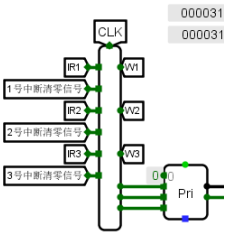

图 3.7 中断优先编码器

2. 实际有效中断信号产生逻辑的实现

当 CPU 正处在中断中时，可能会产生新的中断请求，但单级中断会使其等待，不会对 CPU 产生立即的影响。因此需要通过中断使能信号和优先编码器组选择端通过与操作实现：当当前为正常程序时，产生的中断会使 CPU 进入中断程序，当当前处在中断中时，则不会打扰当前中断。有效中断信号产生逻辑实现如下图所示：

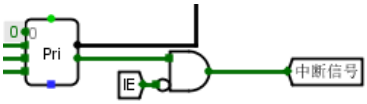

图 3.8 实际有效中断信号产生逻辑

3. 中断清零信号及信息记录信号产生逻辑的实现

由于当产生有效中断信号时，需要记录中断返回地址 EPC、修改 IE 信号、记录中断编号等工作，因此需要特别适用信号量作为使能信号。根据电路逻辑可知，当产生实际有效中断信号、且当前没有中断清零信号时，中断信号记录信号为 1。若不考虑当前中断清零信号，则会产生：该对应中断信号尚未清零，又被当做下一个中断而进入的情况。

中断清零信号则是由三个中断清零信号相或得到。当出现 ERET 指令，则当前中断编号对应中断清零信号则亮起。如下图所示：

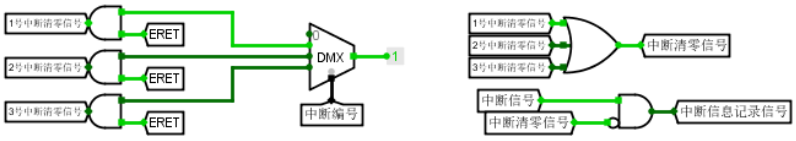

图 3.9 中断清零信号及信息记录信号产生电路

4. 中断返回地址 EPC 寄存器的实现

将没有中断时程序将要送入 PC 寄存器的地址记为正常程序地址，将该地址在中

断信息记录信号的使能下送入 EPC 寄存器，如下图所示：

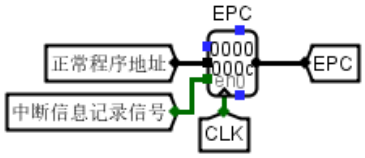

图 3.10 中断返回地址 EPC 寄存器实现电路

5. 中断使能 IE 寄存器的实现

将实际有效中断信号在中断记录信号的使能下送入 IE 寄存器，如下图所示：

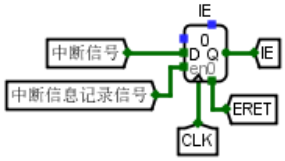

图 3.11 中断使能 IE 寄存器实现电路

6. 中断编号寄存器的实现

将中断编号在中断记录信号的使能下送入中断信号寄存器，如下图所示：

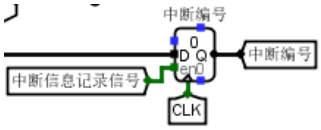

图 3.12 中断信号寄存器实现电路

7. 中断程序地址选择器的实现

根据电路逻辑可知，CPU 在非中断程序和中断程序中时，送入 PC 寄存器的地址都是由非中断电路产生的正常地址。当出现 ERET 指令时，送入 PC 寄存器的时 EPC 寄存器保存的中断返回地址。仅当出现有效中断信号时，给出对应中断编号的中断程序首地址。具体实现如下图。

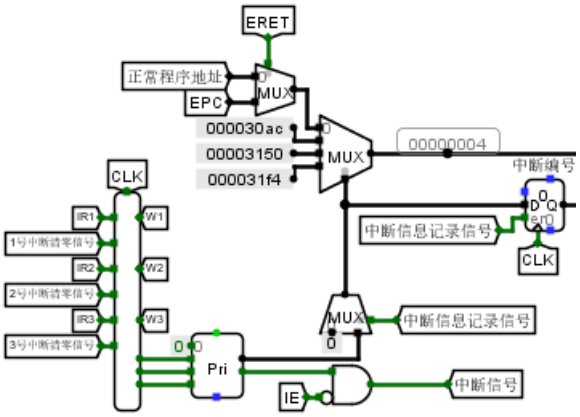

图 3.13 中断程序地址选择器电路

3.2.2 单周期 MIPS CPU 多级中断的实现

单周期 MIPS CPU 多级中断的实现主要有以下几个组成部分：中断按键参考电路、中断编号优先编码器、实际有效中断信号产生逻辑、中断清零信号及信息记录信号产生逻辑、中断返回地址 EPC 寄存器、中断使能 IE 寄存器、中断编号寄存器和中断地址选择器。

单周期 MIPS CPU 的多级中断电路的总体设计如下图所示：

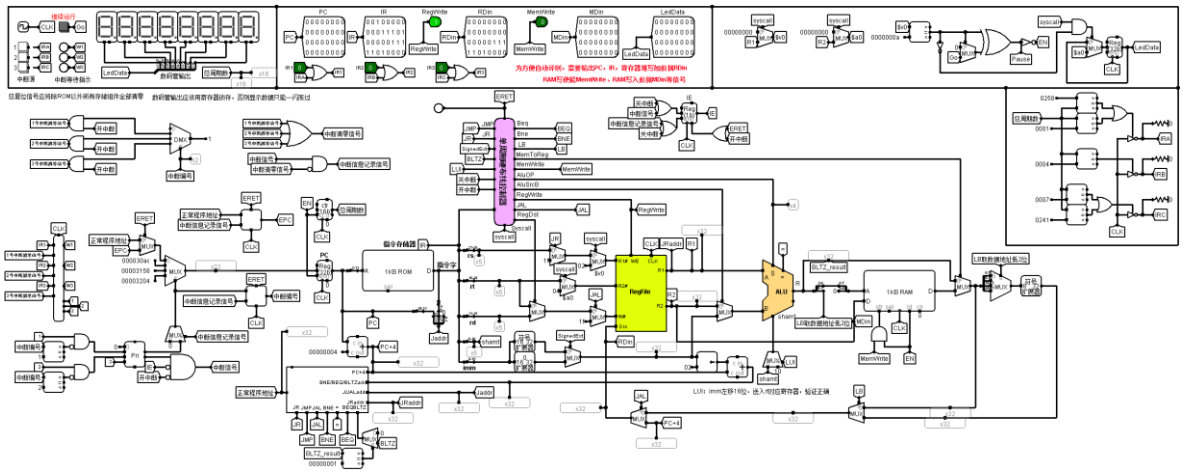

图 3.14 单周期 MIPS CPU 多级中断的总体电路

中断的开中断和关中断信号，都由硬布线控制器根据指令产生。

其中，中断按键参考电路按照给出模板实现，中断信息记录信号的设计逻辑、中断地址选择器的设计逻辑和单级中断一致。

1. 中断编号优先编码器的实现

为了支持多级中断，中断清零信号会在开中断时亮起，若采用和单级中断相同的

设计，会导致低优先级中断打断高优先级中断，因此需要在优先编码器前进行低优先级中断的屏蔽，如下图所示。

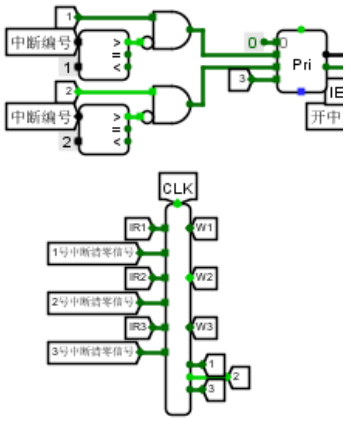

图 3.15 多级中断优先编码器电路

2. 实际有效中断信号产生逻辑的实现

多级中断的有效中断信号由：中断使能信号、优先编码器组选择端、开中断信号三者相与得到，如下图所示：

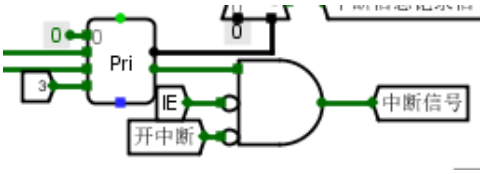

图 3.16 多级中断的有效中断信号产生电路

3. 中断清零信号产生逻辑的实现

中断清零信号现在由开中断信号和中断编号共同控制，若仍然由 ERET 信号控制，则会使单个中断在每次执行到开中断时，把自己误认为是要打断的中断，形成“自己打断自己”的情况。具体实现如下图所示：

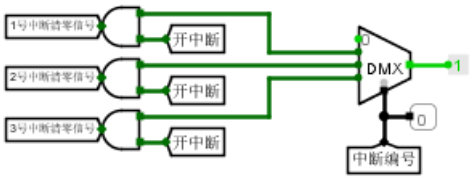

图 3.17 多级中断清零信号的产生电路

4. 中断返回地址 EPC 寄存器的实现

由于存在三级中断，因此最多需要记录三个中断返回地址，采用的实现方法是简易的硬件堆栈，当出现中断信息写入信号且不存在 ERET 信号时，三个寄存器数据向后传送。当出现 ERET 信号时，则将三个寄存器的数据向前传递，并在最后一个寄存器内填充 0。实现如下图所示。

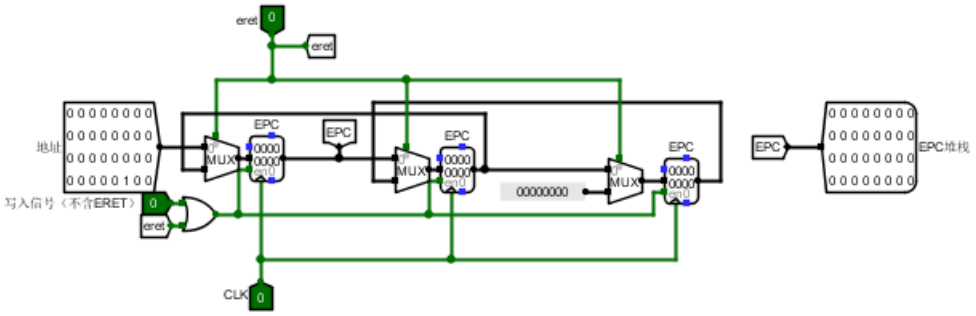

图 3.18 EPC 硬件堆栈电路

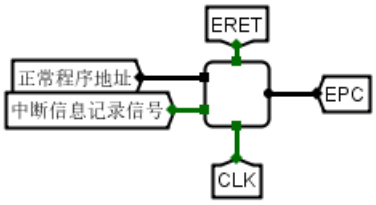

图 3.19 多级中断返回地址堆栈电路

5. 中断使能 IE 寄存器的实现

根据设计逻辑可知，当关中断或出现有效中断信号时，需要在中断记录信号和关中断信号相或的使能下将 IE 信号置为 1，在出现 ERET 指令或开中断指令时，将 IE 信号置为 0。实现如下图所示：

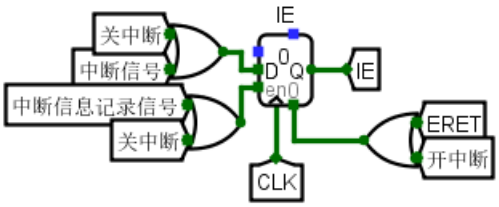

图 3.20 多级中断使能寄存器电路

6. 中断编号寄存器的实现

由于存在三级中断，因此最多需要记录三个中断编号，采用的实现方法是简易的硬件堆栈，当出现中断信息写入信号且不存在 ERET 信号时，三个寄存器数据向后传送。当出现 ERET 信号时，则将三个寄存器的数据向前传递，并在最后一个寄存器内填充 0。实现如下图所示。

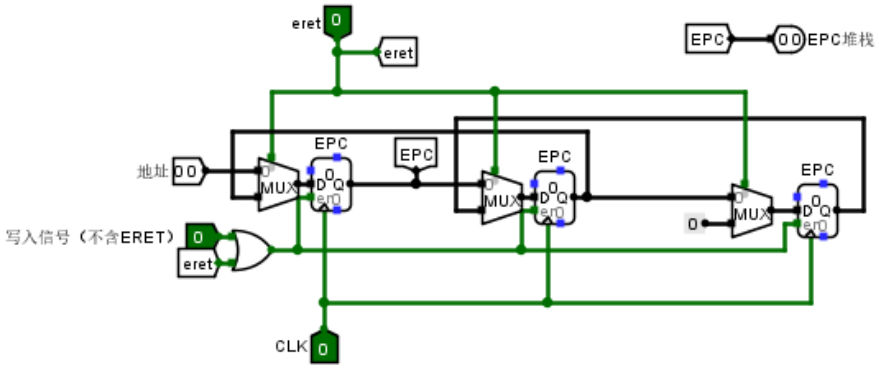

图 3.21 中断编号硬件堆栈电路

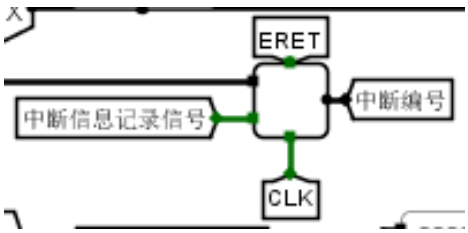

图 3.22 多级中断编号寄存器堆栈电路

3.2.3 流水线单级中断的实现

流水单级中断主要有以下几个组成部分：中断按键参考电路、中断编号优先编码器、实际有效中断信号产生逻辑、中断清零信号及信息记录信号产生逻辑、中断返回地址 EPC 寄存器、中断使能 IE 寄存器、中断编号寄存器和中断地址选择器。

流水线单级中断的总体设计电路如下图所示，气泡流水线的设计和实现在下文详述。

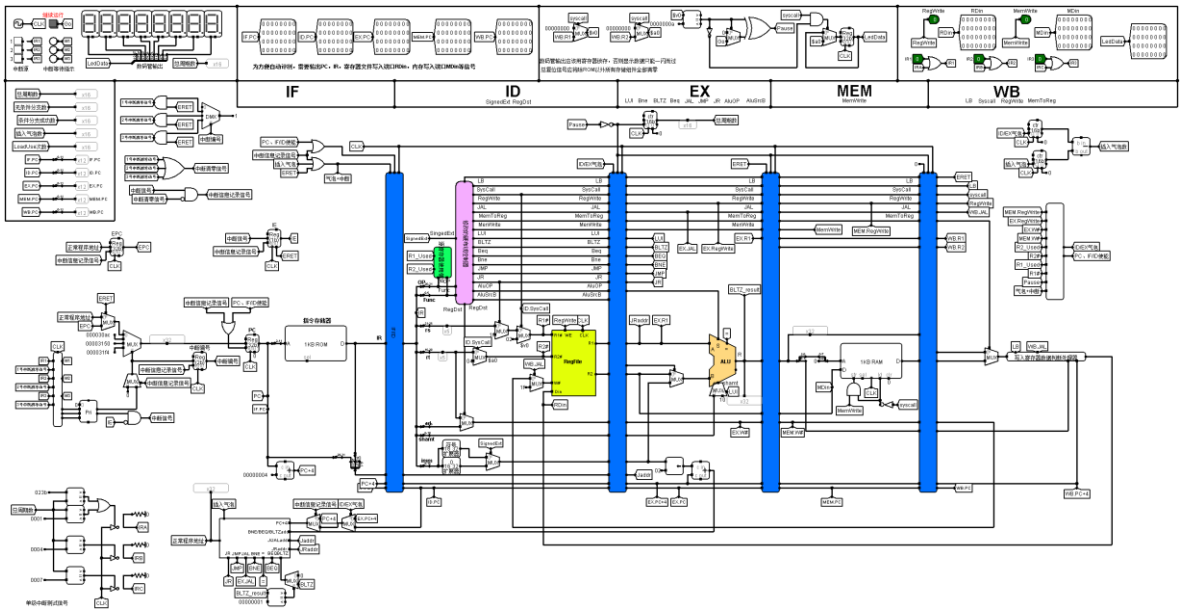

图 3.23 流水线单级中断的总体设计电路

上述部件的实现均和单周期 MIPS CPU 单级中断的设计一致。主要的不同在于对于流水线的处理。

对于出现中断时，需要在 IF/ID 和 ID/EX 两个流水部件插入气泡，让 EX、MEM 以及 WB 段的指令流出流水线。送入中断返回寄存器 EPC 的地址应当进行判断，若 EX 段正处在气泡中，则将 ID 段指令地址送入，否则送入 EX 段指令下一条指令的地址。

1. 流水部件插入中断气泡的实现

将中断信息记录信号通过或门，与其他信号一起，作为 IF/ID 段流水寄存器使能和清零信号的决定信号之一。实现如下图所示：

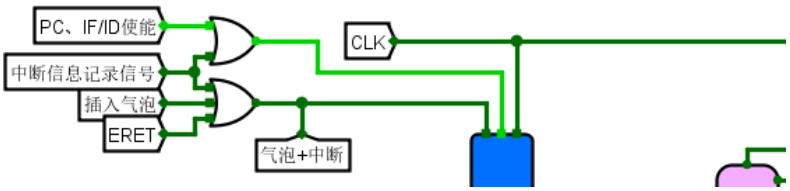

图 3.24 流水部件插入中断气泡电路

关于 IF/EX 段流水部件的气泡插入实现，由程序冲突引起的气泡和中断气泡决定，具体实现在气泡流水线中实现。

2. 选择 ID 或 EX 地址送入中断返回寄存器 EPC 的实现

送入 ID 段地址或 EX 段地址由中断时刻 ID/EX 是否存在气泡决定。若存在气泡，则送入 ID 段指令地址，否则送入 EX 段指令的下一条指令的地址。实现方式如下图右上角所示。图中的封装电路为指令地址跳转器，将在下文进行详述，其功能是给出正确的下一条指令地址。

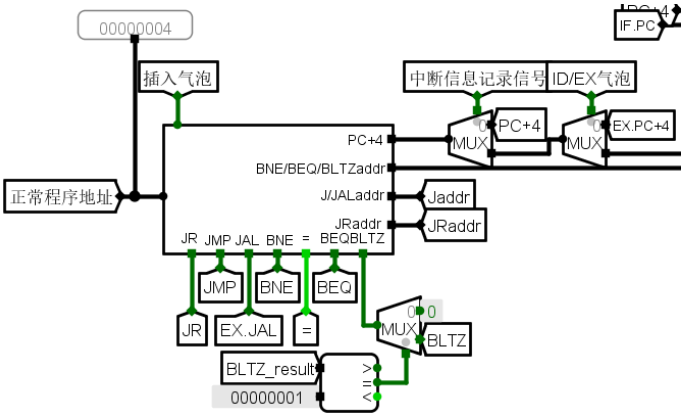

图 3.25 选择指令地址送入 EPC 寄存器的电路

3.3 理想流水线实现

理想流水线中，不包含指令冲突和转跳。实现电路如下图所示：

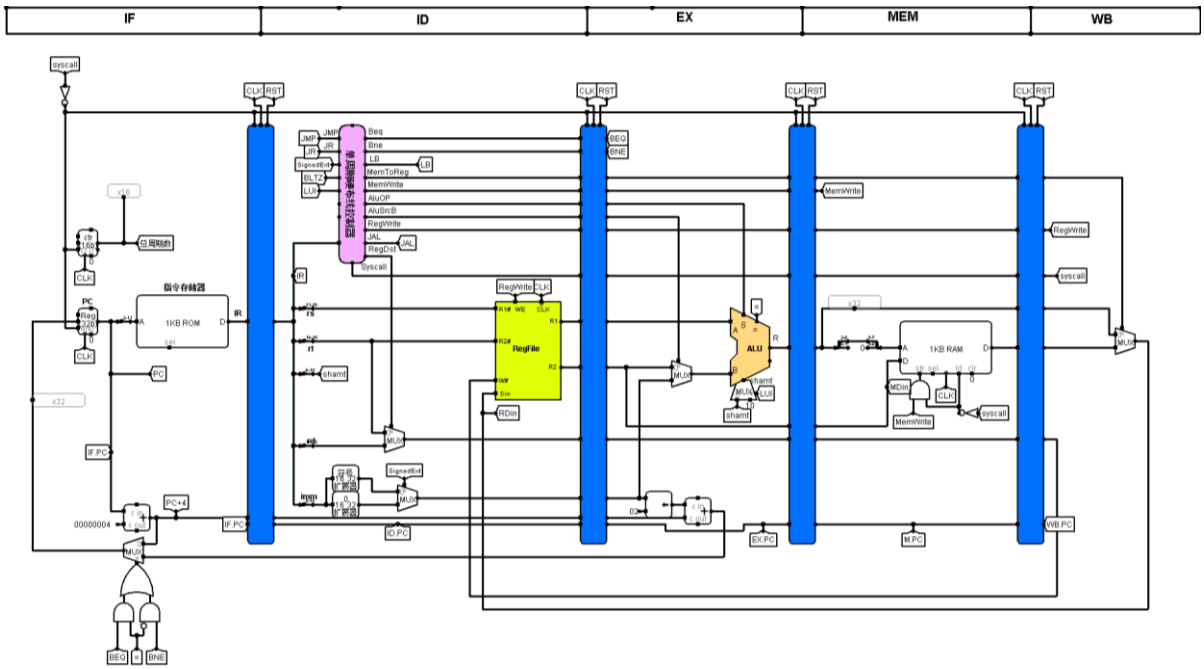

图 3.26 理想流水线的实现

3.3.1 理想流水部件的实现

流水部件的设计主要取决于各段间需要传递的内容。由于理想流水线中没有冲突等问题，需要处理的就是将转跳相关的信号传递到 EX、写入数据存储器的信号传递到 MEM 段、写回和停机相关信号传递到 WB 段。

IF/ID 段需要传递的内容有：指令 IR、IF 段指令地址 IF.PC、IF 段指令的下一条指令地址 IF.PC+4。

ID/EX 段需要传递的内容有：AluOP、#R1、#R2、拓展后的 Imm、ID 段指令地址 ID.PC、ID 段指令的下一条指令地址 ID.PC+4。需要传递的信号有：Beq、Bne、MemToReg、MemWrite、AluSrcB、RegWrite、RegDst、SysCall。

EX/MEM 段需要传递的内容有：ALU 计算结果 R、#R2、EX 段指令地址 EX.PC。需要传递的信号有： MemToReg、MemWrite、RegWrite、RegDst、SysCall。

MEM/WB 段需要传递的内容有：ALU 计算结果 R、数据存储器输出、MEM 段指令地址 MEM.PC。需要传递的信号有： MemToReg、RegWrite、RegDst、SysCall。

除了需要传递的信号和数据的数量不同外，流水部件的设计是相同的。

流水部件通过 Reset 信号控制数据选择器，当清零信号为高电平，会使数据选择器选择全零信号，输入到寄存器内。使能信号和时钟信号作用于所有寄存器。以 IF/ID 流水部件为例，其实现如下图所示：

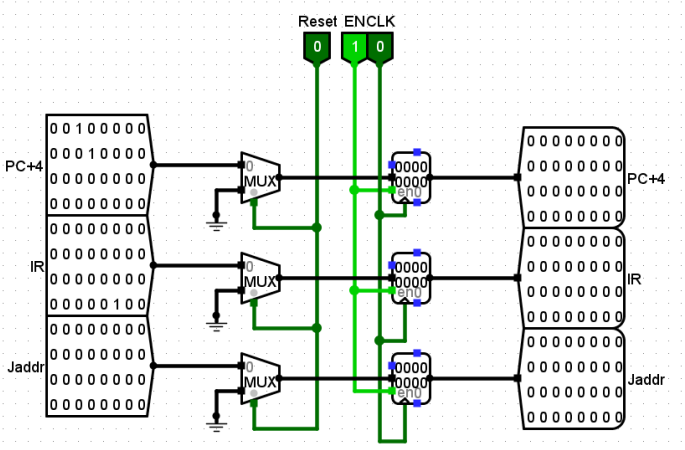

图 3.27 流水部件的实现电路

3.4 气泡式流水线实现

气泡流水线和理想流水线的区别在于，气泡流水线能够同时处理转跳冲突和数据

冲突。转跳冲突通过转跳地址判断器实现，而数据相关检测有数据相关检测器实现。气泡流水线电路如下图所示：

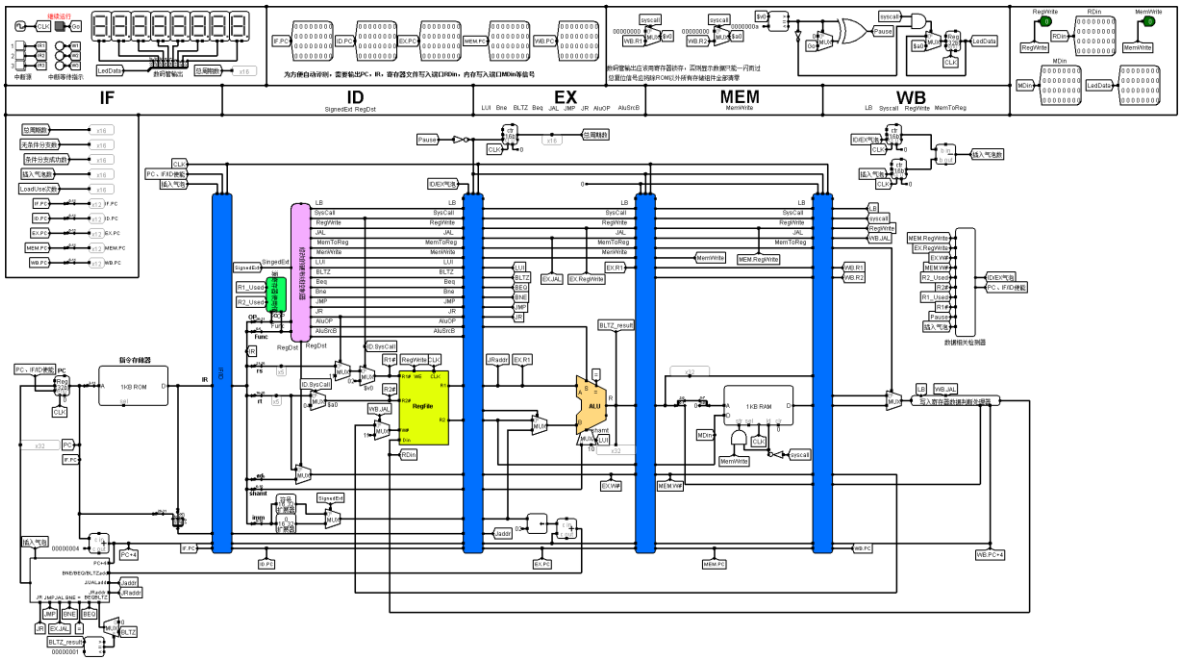

图 3. 28 气泡流水线电路

3.4.1 转跳地址判断器的实现

转跳地址判断器需要的输入包括：各类转跳地址、各类转跳信号。其输出包括实际的转跳地址和气泡信号。

各类转跳地址包括：IF 段指令的下一条地址 IF.PC+4、EX 段计算得到的 BNE/BEQ/BLTZ 指令的地址（其计算方式相同）、IF 段计算得到并传递到 EX 段的 J 指令目标地址 Jaddr 和保存在寄存器中，传递到 EX 的 JR 指令目标地址 JRaddr。

各类转跳信号有：JR、JMP、EX.JAL、BNE、ALU 的等于信号、BEQ 和 rs 小于 0 下时 BLTZ。

当判断器识别出转跳指令时，气泡信号亮起。具体实现如下图所示：

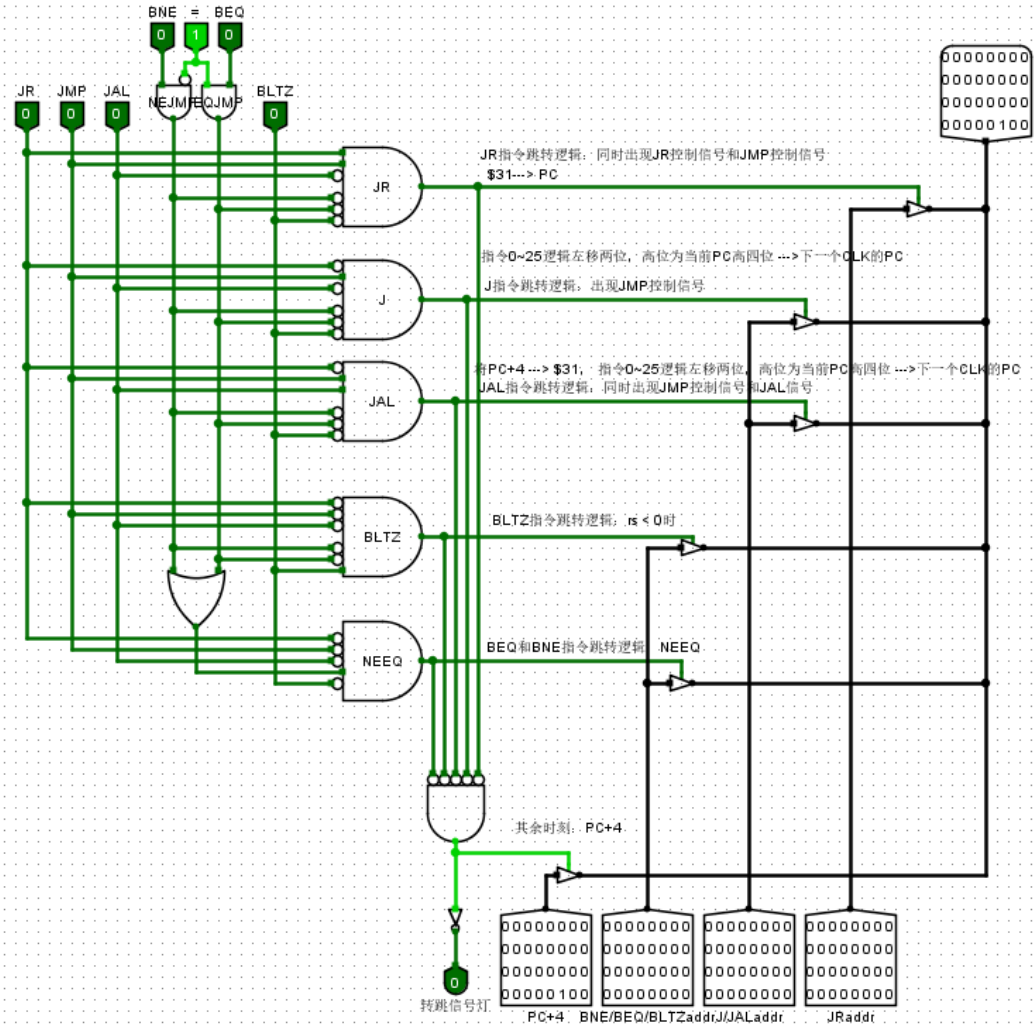

图 3.29 转跳地址判断器电路

其在气泡流水线中的封装和使用如下图所示：

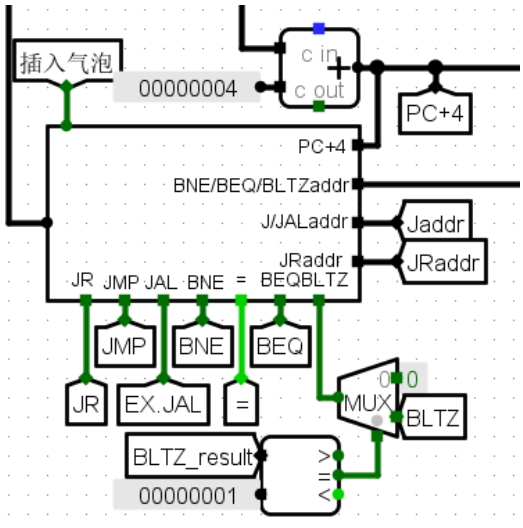

图 3.30 转跳判断器的封装和使用

3.4.2 寄存器使用情况判断的实现

指令对寄存器的使用情况如下表所示：

表 3.2 指令对寄存器使用状况表

指令	R1_Used	R2_Used
SLL		1
SRA		1
SRL	1	1
ADD	1	1
ADDU	1	1
SUB	1	1
AND	1	1
OR	1	1
NOR	1	1
SLT	1	1
SLTU	1	1
JR	1	
SYSCALL	1	1
J		
JAL		
BEQ	1	1
BNE	1	1
ADDI	1	
ANDI	1	
ADDIU	1	
SLTI	1	
ORI	1	
LW	1	

指令	R1_Used	R2_Used
SW	1	1
LUI		
SLTIU	1	
LB	1	
BLTZ	1	

根据上表，可以自动生成源寄存器使用情况电路，如下图所示：

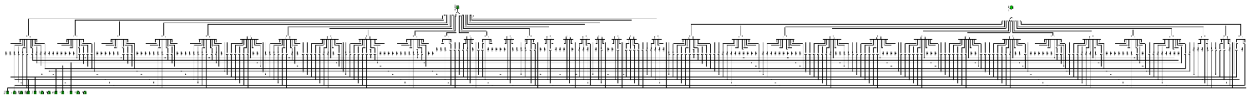

图 3.31 源寄存器使用情况电路

对该电路进行封装，得到如下电路：

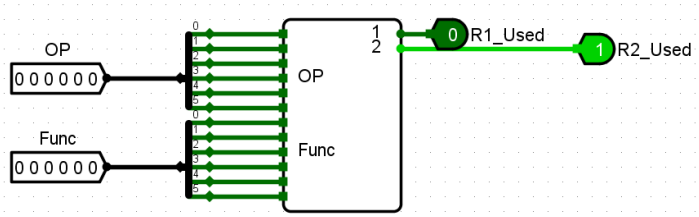

图 3.32 源寄存器使用情况封装电路

3.4.3 数据相关检测器的实现

数据相关检测器的功能是：根据 EX 和 MEM 段所要写入的寄存器使用情况，判断是否存在数据冲突。当出现冲突时，插入气泡。

数据相关检测器需要的输入信号有：MEM.RegWrite、EX.RegWrite、EX.W#（写回寄存器编号）、MEM.W#、R2 使用信号 R2_Used、R1 使用信号 R1_Used、R1 寄存器编号、R2 寄存器编号、暂停信号 Pause、转跳判断器给出的气泡信号。

实现逻辑是：当使用的寄存器不为$0，写入信号为 1，且正在使用的寄存器和要写入的寄存器相同时，产生数据中途。当产生数据冲突时，ID/EX 段流水部件要插入气泡，PC 和 IF/ID 段流水部件的使能信号为低电平。当不产生数据冲突时，ID/EX 段气泡取决于转跳判断器给出的气泡信号，PC 和 IF/ID 段流水部件的使能信号取决于 Pause 信号。具体实现如下图所示：

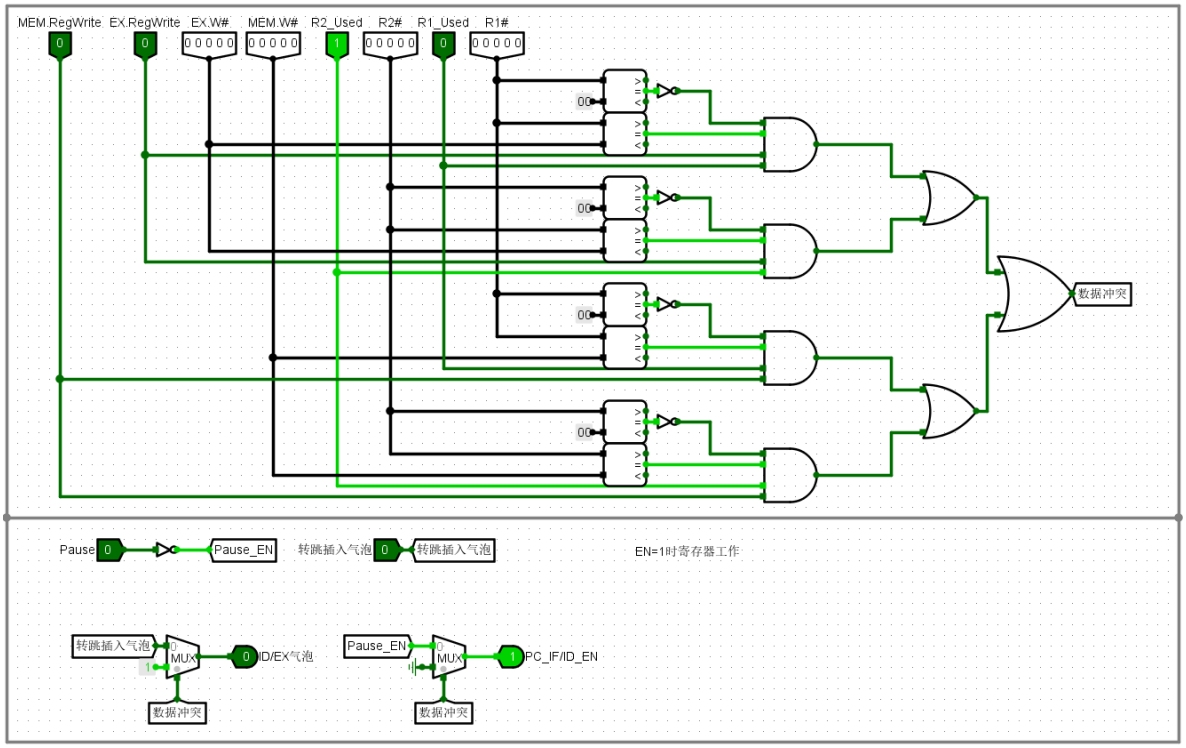

图 3.33 数据相关检测器电路

数据相关检测器的封装和应用如下图所示：

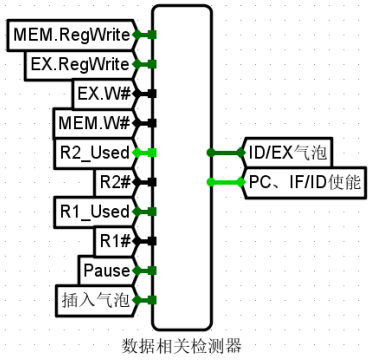

图 3.34 数据相关检测器的封装和应用

3.5 重定向流水线实现

重定向流水线的特点在于，对于一些出现数据冲突的指令，将数据先送到指令所需对应位置，再写回寄存器中。这些数据来源于：传递到 MEM 段的 ALU 运算结果、WB 段中从数据存储器和传递到 WB 段的 ALU 运算结果。

重定向流水线的电路图如下图所示：

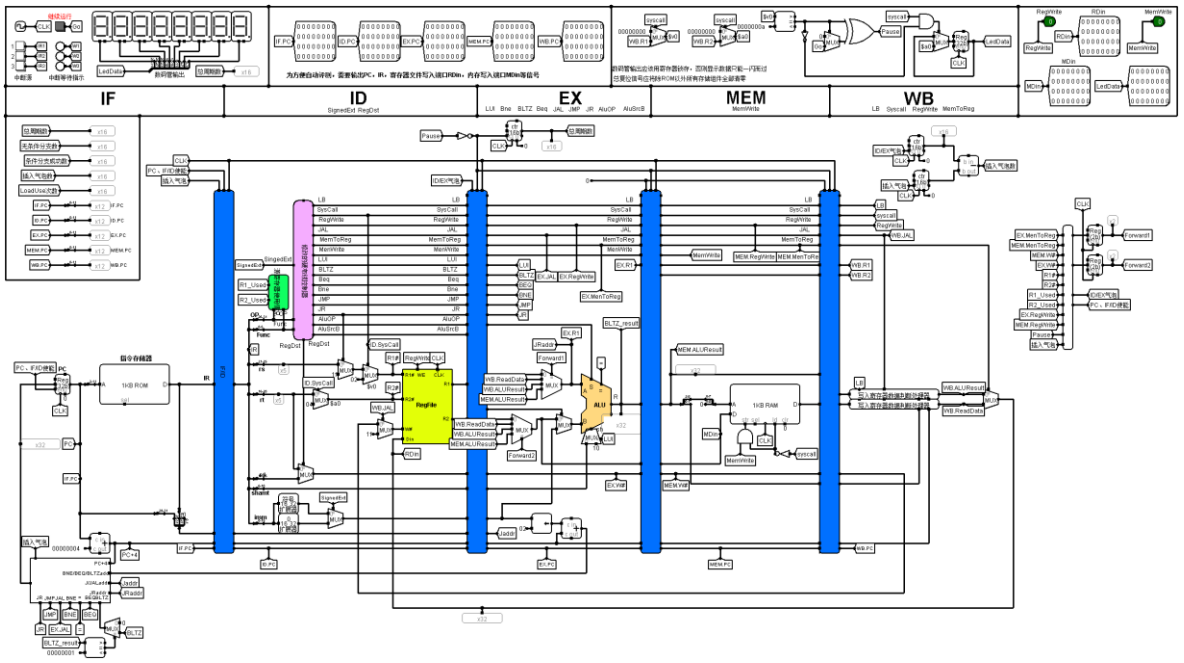

图 3. 35 重定向流水线的实现

3.5.1 重定向数据选择器的实现

当数据出现冲突时，需要将数据先送入正确的运算位置，再送回寄存器中。重定向的位置位于 ALU 的 A、B 端口。根据数据重定向的来源，使用 2 位的多路选择器即可实现。对电路的修改如下图所示：

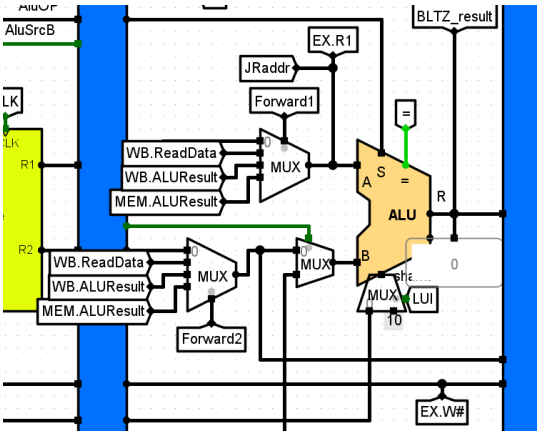

图 3. 36 重定向多路选择器的实现

3.5.2 重定向数据冲突检测和定向器的实现

数据相关检测器的功能是：根据 EX 和 MEM 段所要写入的寄存器使用情况，判断是否存在数据冲突。当出现非 LW 指令的冲突时，进行数据重定向，否则插入气泡。

数据相关检测器需要的输入信号有：MEM.MemToReg、EX. MemToReg 、

MEM.RegWrite、EX.RegWrite、EX.W#（写回寄存器编号）、MEM.W#、R2 使用信号 R2_Used、R1 使用信号 R1_Used、R1 寄存器编号、R2 寄存器编号、暂停信号 Pause、转跳判断器给出的气泡信号。

插入气泡的逻辑是：当 EX 段 MemToReg 信号为 1，且产生非$0 寄存器的冲突时，插入气泡。

数据重定向的逻辑是：根据输入的信号，给出 ALU 的 A、B 端口的选择器选择信号。

具体实现如下图所示：

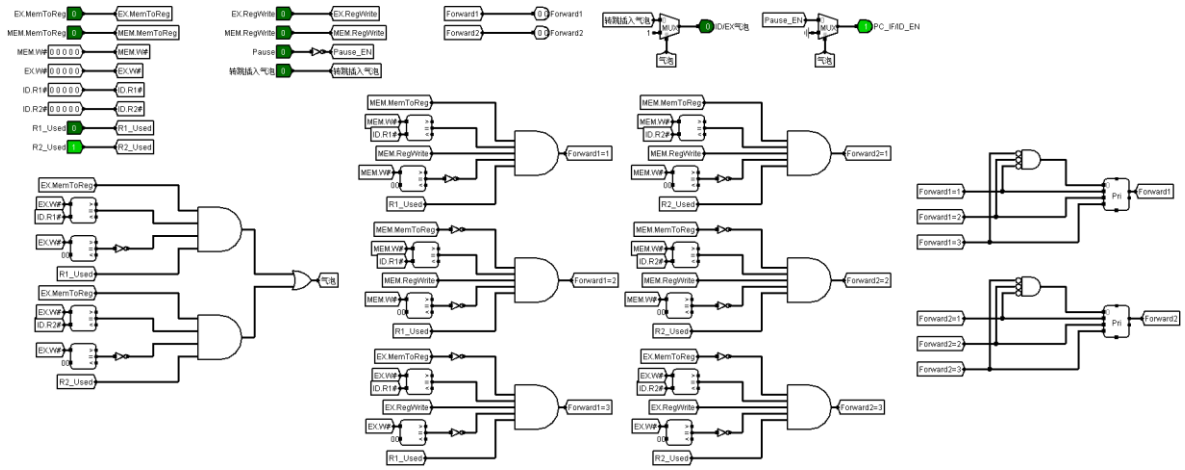

图 3.37 重定向数据冲突检测和重定向

3.6 动态分支预测机制实现

在具体实现中，将转跳地址判断器和 BHT 表进行了综合，实现了动态分支预测机制。

动态分支预测机制的主要功能是：当 IF 段指令地址进入动态分支预测表，检测其是否命中。若不命中，则交由转跳地址判断器部分，得出在没有动态分支预测机制下会给出的下一条指令地址。若 IF 段指令地址命中 BHT，则观察历史预测值。当历史预测值大于 1 时，认为程序会跳转，给出存储在 BHT 内的目标指令地址。当历史预测值小于等于 1 时，认为程序不会转跳，给出的指令地址为 IF.PC+4。

同时，对 EX 段指令地址也要进行检查。根据 EX 段指令地址和 ID 段指令地址的关系是否为 ID.PC = EX.PC + 4，判断 BHT 是否进行了转跳。同时，根据转跳地址判断器给出的正常指令地址和 IF.PC + 4 进行比较，若一致，说明当前不需要转跳，若不一致，说明 EX 段的指令为转跳指令。

根据“需要转跳与否”和“BHT 执行转跳与否”，可以得到四种情况：“需要转跳但 BHT 没有转跳”、“需要转跳且 BHT 已经执行了转跳”、“不需要转跳且 BHT 没有转跳”、“不需要转跳但 BHT 已经执行了转跳”。此四种情况对应不同的处理。

需要转跳但BHT没有转跳：向流水线插入气泡，将转跳目标指令地址送入PC中。当该指令地址（EX.PC）不命中 BHT，则将其根据 LRU 算法写入 BHT 表中。当指令地址命中 BHT，则修改对应位置的预测历史值。

需要转跳且 BHT 已经执行了转跳：分支预测成功，直接在 BHT 表中修改对应 EX.PC 位置的预测历史值。

不需要转跳且 BHT 没有转跳：当该指令地址（EX.PC）不命中 BHT 时，不做任何动作。当其命中时，修改对应 EX.PC 位置的预测历史值。

不需要转跳但 BHT 已经执行了转跳：此时插入气泡，将 IF.PC+4 的地址送入 PC 寄存器。同时在 BHT 表中修改对应 EX.PC 位置的预测历史值。

动态分支预测机制的全局设计如下图所示：

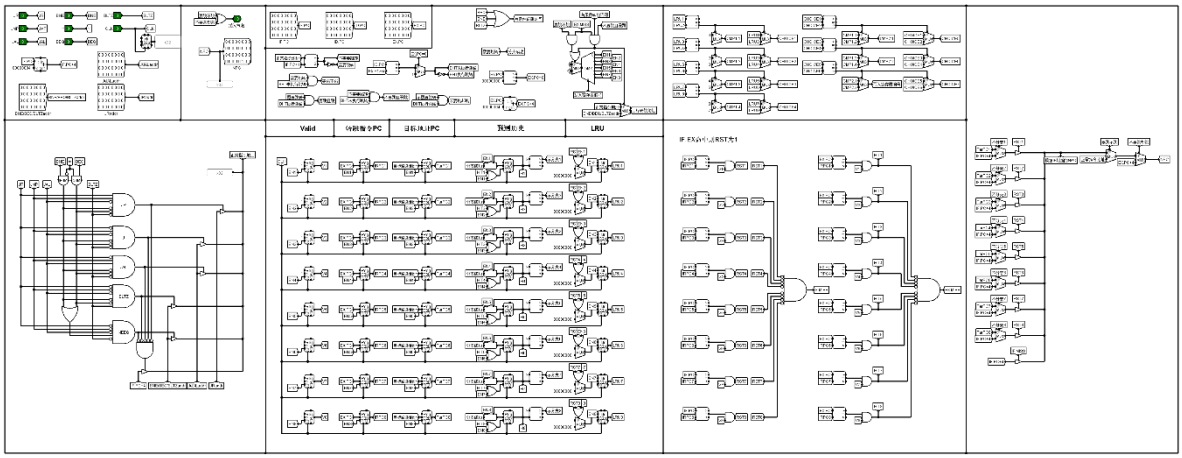

图 3.38 BHT 全局电路

3.6.1 BHT 的实现

BHT 采用全相连 Cache 的设计思路，每一行有：Valid 寄存器、转跳指令地址 PC 寄存器、目标指令地址 PC 寄存器、预测历史状态和 LRU 寄存器。其中预测历史状态转换的实现，会在 3.6.4 节中描述。

实现电路如下图所示：

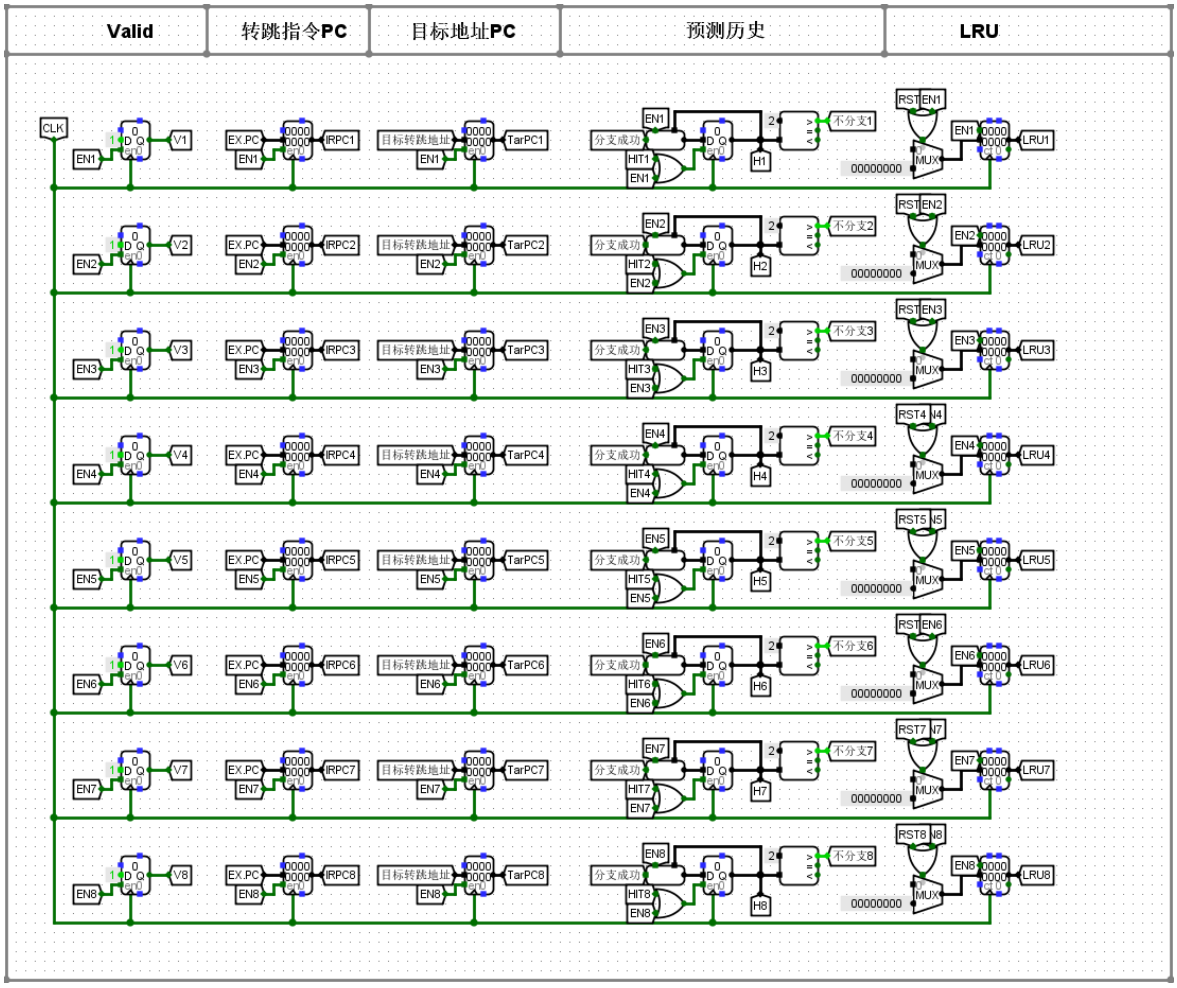

图 3.39 BHT 电路

3.6.2 BHT 命中检测的实现

BHT 命中检测由两部分构成：IF 段指令地址的命中检测 EX 和段指令地址的命中检测。检测命中要求为：指令地址存在于 BHT 的转跳指令地址 PC 寄存器之中，且对应 Valid 寄存器值为 1。当 8 行全不命中时，则不命中。实现电路如下图所示：

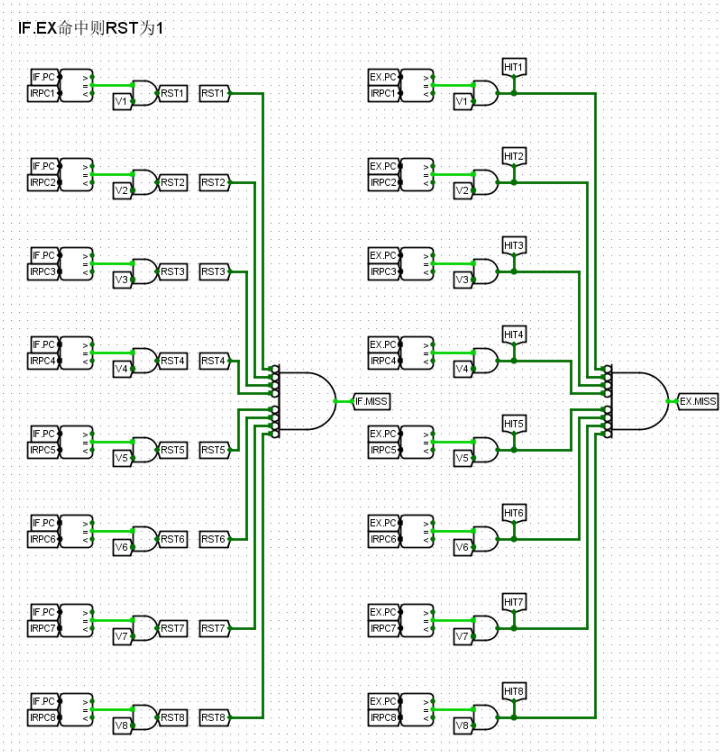

图 3.40 BHT 命中检测电路

3.6.3 LRU 的实现

通过对 BHT 中八行 LRU 寄存器的两两比较，得到其中最小值所在的行号。将对应行的使能信号全部调整为高电平。实现电路如下图所示：

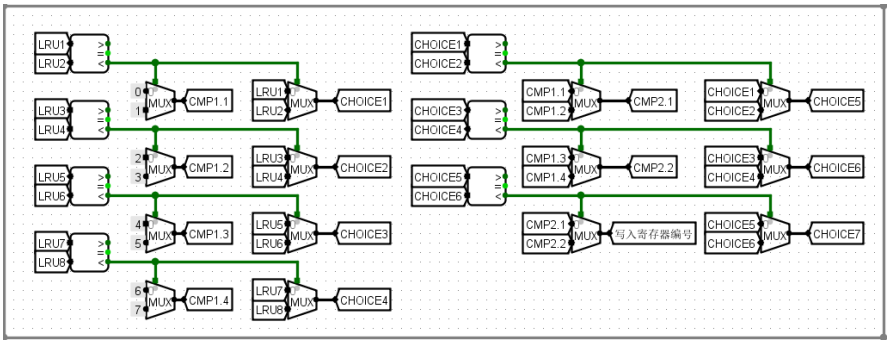

图 3.41 LRU 判断逻辑电路

3.6.4 预测历史状态转换器的实现

该转换器的实现采用了优先状态机的设计思想，针对前序状态下的四种情况，给出对应分支成功或失败下的更新状态。电路实现如下图所示：

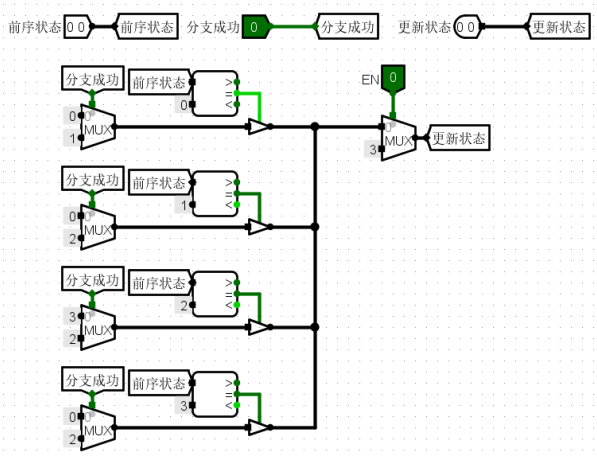

图 3.42 预测历史状态转换器电路

其电路封装和使用如下图所示：

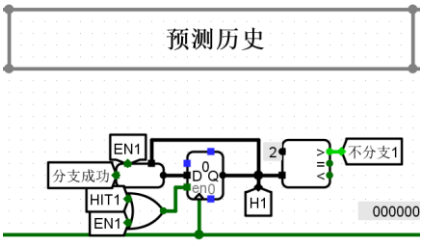

图 3.43 预测历史状态转换器封装和应用

3.6.5 四种状态判断的实现

四种状态的判断逻辑已经在上文详述，其具体实现如下图：

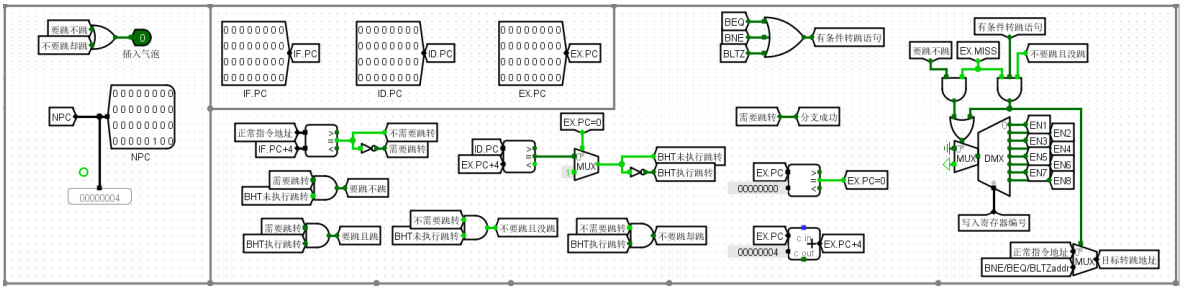

图 3.44 四种状态判断逻辑的实现

3.6.6 实际转跳判断的实现

该判断用于给出最终决定送入 PC 寄存器的地址。当给出分支信号时，给出对应 BHT 行的目标指令地址，当地址不命中时，给出 IF.PC＋4。当出现“需要转跳但 BHT 没有转跳” 的情况时，使用转跳地址判断器得出的地址。当出现“不需要转跳但 BHT 已经转跳”的情况时，给出 EX.PC+4。具体实现如下图所示：

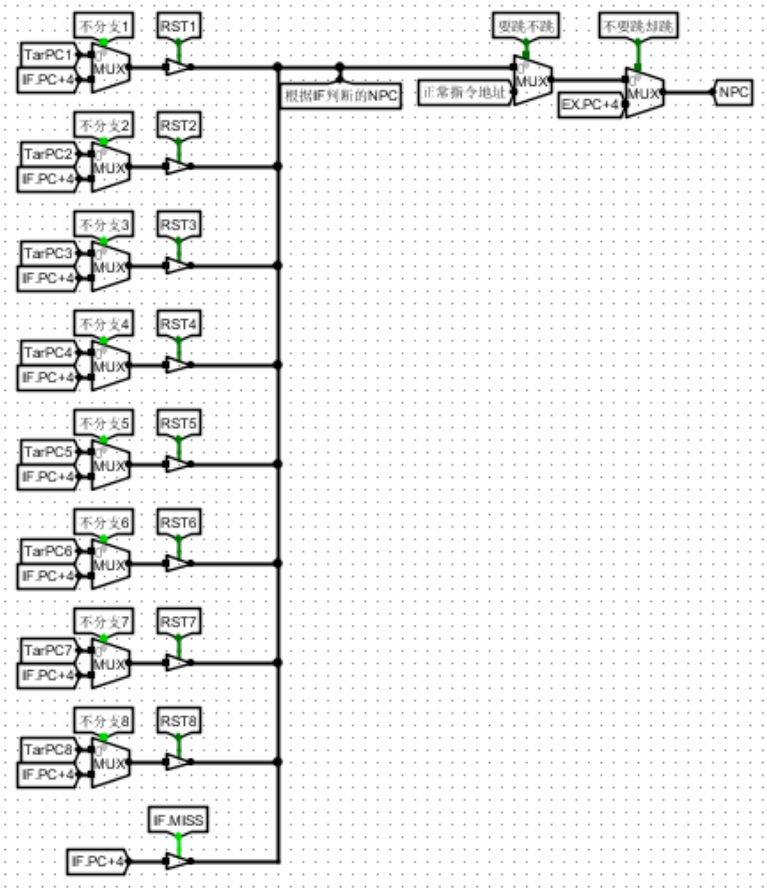

图 3.45 实际转跳判断电路

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 测试用例

测试用例 1 使用 Benchmark，不包含中断、能够停机且不包含拓展的自定义四条指令。

测试用例 2 使用单级中断测试程序。

测试用例 3 使用多级中断测试程序。

测试用例 4 使用 Benchmark+ccmb，不包含中断、能够停机且包括拓展的自定义四条指令。

测试用例 5 使用理想流水线测试程序。

4.1.2 功能测试

1. 单周期 MIPS CPU 测试

单周期 MIPS CPU 使用测试样例 1 进行测试，运行周期数如下图所示，为 1545：

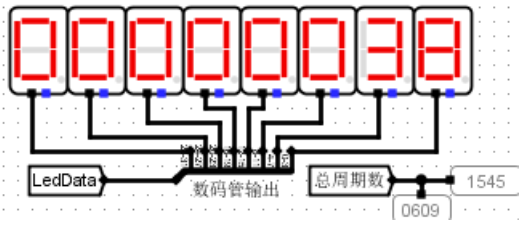

图 4.1 单周期 MIPS CPU 周期数测试

在 Educoder 平台测试，测试结果如下图所示：

图 4.2 单周期 MIPS CPU Educoder 平台测试结果

根据上述测试可知，通过正确性验证。

2. 理想流水线测试

理想流水线使用测试样例 5 进行测试。运行周期数如下图所示，为 20：

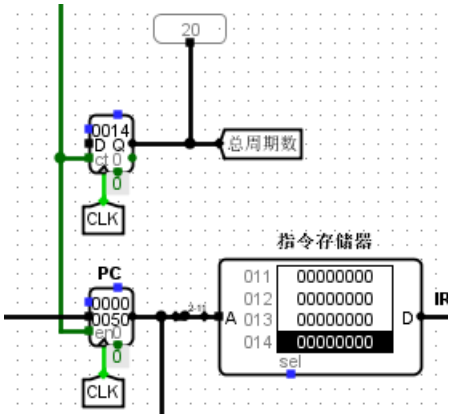

图 4.3 理想流水线周期数测试

在 Educoder 平台测试，测试结果如下图所示：

图 4.4 理想流水线 Educoder 平台测试结果

根据上述测试可知，通过正确性验证。

3. 气泡流水线测试

气泡流水线测试使用测试样例 1。运行周期数为 3623，如下图所示：

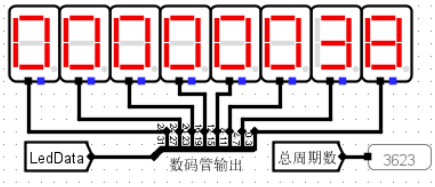

图 4.5 气泡流水线周期测试

在 Educoder 平台测试，测试结果如下图所示：

图 4.6 气泡流水线 Educoder 平台测试结果

根据上述测试可知，通过正确性验证。

4. 重定向流水线测试

重定向流水线测试使用测试样例 1。运行周期数为 2297，如下图所示：

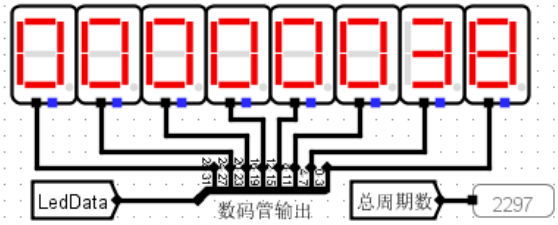

图 4.7 重定向流水线周期测试

在 Educoder 平台测试，测试结果如下图所示：

图 4.8 重定向流水线 Educoder 平台测试结果

根据上述测试可知，通过正确性验证。

5. 单周期 MIPS 单级中断测试

气泡流水线测试使用测试样例 2。在 Educoder 平台测试，测试结果如下图所示：

图 4.9 单周期 MIPS 单级中断 Educoder 平台测试结果

根据测试可知，通过正确性验证。

6. 单周期 MIPS 多级中断测试

气泡流水线测试使用测试样例 3。在 Educoder 平台测试，测试结果如下图所示：

图 4.10 单周期 MIPS 多级中断 Educoder 平台测试结果

根据上述测试可知，通过正确性验证。

7. 气泡流水线单级中断测试

气泡流水线测试使用测试样例 2。测试过程中，依次进入 1 号中断、3 号中断、2 号中断和 1 号中断，最后恢复到正常程序。

根据测试可知，通过正确性验证。

8. 重定向流水线动态分支预测测试

重定向流水线动态分支预测测试使用测试样例 1。运行周期数为 1765，如下图所示：

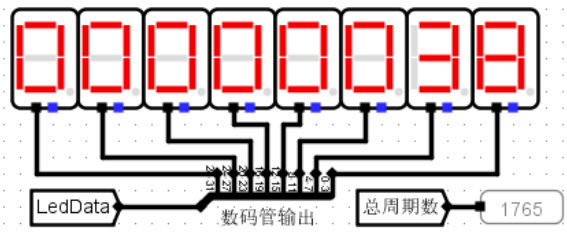

图 4.11 重定向流水线动态分支预测周期测试

9. 四条扩展指令测试

使用单周期 MIPS CPU，采用测试用例 4。LUI 指令测试中的显示截图如下图所示：

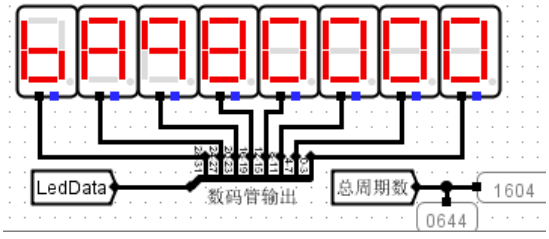

图 4.12 LUI 指令测试

SLTIU 指令测试中的显示截图如下图所示：

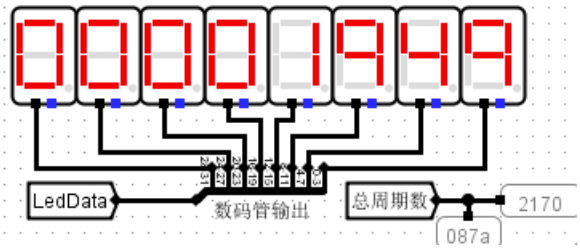

图 4.13 SLTIU 指令测试

LB 指令测试中的显示截图如下图所示：

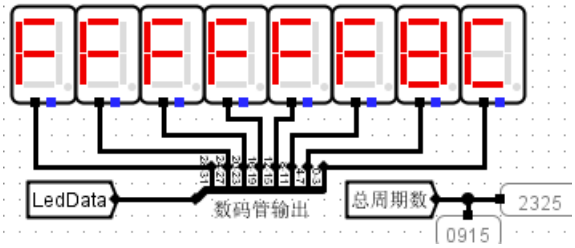

图 4.14 LB 指令测试

BLTZ 指令测试中的显示截图如下图所示：

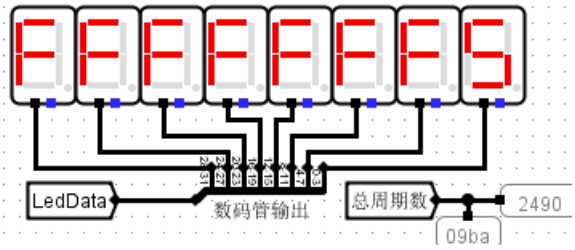

图 4.15 BLTZ 指令测试

根据上述测试观察，可知正确性得到验证。

4.2 性能分析

在测试中，单周期 MIPS CPU 对测试样例 1 的周期数为 1545，气泡流水线对测试样例 1 的周期数为 3623，重定向流水线对测试样例 1 的周期数为 2297，使用动态分支预测的重定向流水线对测试样例 1 的周期数为 1765。

重定向流水线和使用动态分支的流水线能够降低周期数的原因在于，其能够减少插入的气泡，提高流水线的效率。

虽然单周期 CPU 所用周期数更少，但是为了完成耗时较长的指令，要求每个周期的耗时相对更长。

4.3 主要故障与调试

重定向流水线：重定向多路选择器电路设计问题

故障现象：当数据冲突产生重定向时，数据存储器数据输入端的数据不是重定向的数据。

原因分析：如下图所示，当前 EX 段中从 R2#到运算器 ALU 的 B 端口之间的实现方式是：先使用二路选择器，由 AluSrcB 进行立即数或寄存器数的选择，然后使用四路选择器，完成重定向的数据选择。

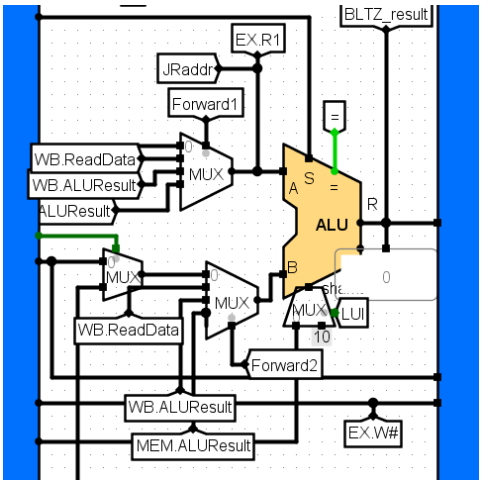

图 4.16 故障前电路

然而，重定向的意义是，将还未来得及写入寄存器的数据直接定向到使用的位置，因此重定向的数据应该和 R2#端出来的数据一起进行选择，而当前的电路会出现立即数被重定向数据所阻塞的问题，同时正确的重定向的数据无法流入数据存储器。修改后的电路如下图所示：

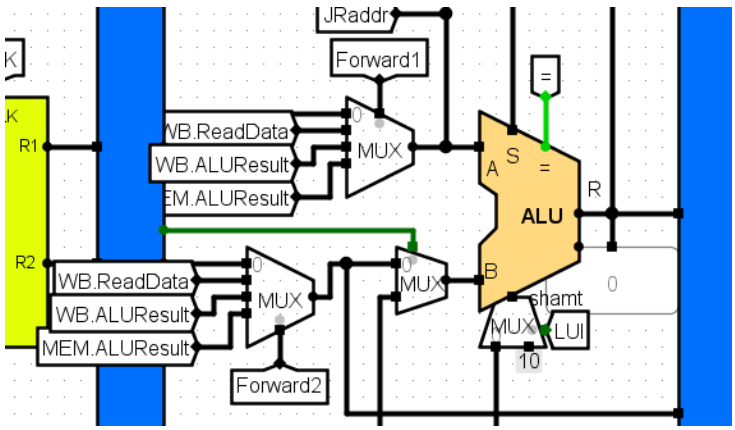

图 4.17 修改后电路

4.4 实验进度

表 4.1 课程设计进度表

时间	进度
2 月 17 日	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，开始课程设计。
2 月 18 日	完成单周期 MIPS CPU 电路，支持 24 + 4 条指令
2 月 19 日	完成理想流水线电路
2 月 21 日	完成气泡流水线电路，能够处理分支冲突和数据冲突
2 月 24 日	完成重定向流水线电路
2 月 27 日	完成单周期单级和多级中断
2 月 28 日	在 Educoder 平台完成单周期、理想、气泡、重定向、单周期单级/多级中断的测试 完成流水线单级中断
3 月 1 日	完成动态分支预测的重定向电路，除团队合作外所有任务完成
3 月 2 日	团队讨论，选题并讨论可行性
3 月 3 日	给出设计方案，组内任务分配
3 月 5 日	合并组内工作，进行总体调试
3 月 6 日	调试结束，团队任务完成

5 团队项目

5.1 团队项目介绍

团队项目选择通过外接 LED 点阵和按键，实现 Flappy Bird 游戏的核心功能。游戏玩法为：画面中有鸟和障碍物两类物体，其中鸟的横坐标保持不变，玩家通过点击按钮，控制鸟向上移动；否则鸟将会以自由落体的速度向下移动。障碍物之间相隔一定距离，从右向左移动，且每个障碍物具有固定长度的开口供鸟通过。当鸟撞上障碍物时，则表示游戏结束。

5.2 团队成员与分工

团队成员有：刘晨彦、张瀚元、邓小阳、黄栋。团队分工如下：

黄栋：修改电路

邓小阳：代码编写、电路修改

刘晨彦：项目设计、项目测试

张瀚元：代码编写、数据设置

5.3 团队项目分析

团队项目需要实现的部分有：控制 LED 点阵的显示和刷新、障碍物的移动和产生、鸟位置控制、碰撞检测等。

5.3.1 LED 点阵的显示和刷新

LED 点阵由 16 列组成，每列有 32 行。为了独立控制每一列，设置 16 个 32 位数据寄存器。由于 CPU 电路较为简单，不能通过接口的形式向外接硬件传递消息。但可以通过读取数据存储器的写信号、写入地址和写入数据，在将数据写入数据存储器的指定地址位置时，同时将数据保存到对应 LED 显示寄存器中。由于电路频率受到软件限制，不能够忽略 LED 逐行刷新的时间。为了避免刷新所导致的“一半新画面、一半旧画面”，额外设置了一条写入指令。当各数据寄存器检测到该写入指令，就会同步将显示数据刷新到 LED 点阵中。

5.3.2 障碍物的移动和产生

由于障碍物的显示数据已经保存在数据存储器的指定地址中，移动则可以通过读取对应数据存储器的地址，并送入相邻列对应的数据寄存器的地址中。

障碍物的产生则通过读取存储在数据存储器内的随机数产生：由于障碍物的开口长度固定，通过随机数来控制开口的高度，实现不同障碍物的产生。

5.3.3 鸟位置控制

鸟位置主要有两种控制：下落和上升

当按钮不被触发，鸟每次刷新下落一次。下落的长度为：2 格、4 格、8 格，直至落至 LED 点阵底部不再下落。通过分配寄存器，记录鸟的下落时间，根据下落时间可以得到具体的下落长度。鸟在 LED 点阵中通过点来表示，下落可以通过右移指令实现。

鸟的上升则需要通过外部按钮按下触发。为此需要设置中断，当按钮按下进入中断。一次中断内，使鸟的位置上升 2 格，通过左移指令实现。

5.3.4 碰撞检测

检测碰撞则可以通过综合障碍物的显示和鸟的显示完成：在鸟所在列，读取障碍物位置和鸟的位置，进行或操作，得到该列的显示信息。当或操作后得到的显示信息和障碍物信息一致，说明鸟和障碍物相撞。程序应该进入游戏结束过程。否则将或操作后的显示信息写入对应位置。在其他列中，需要从障碍物显示信息所在的数据存储器地址中读取，并送入对应 LED 点阵显示信息所在的数据寄存器地址中。

5.4 团队项目实现

5.4.1 外部器件

外部器件包括：按钮和 16×32 的 LED 点阵。

5.4.2 寄存器和存储分配

LED 点阵显示信息位于数据存储器的 0x00 ~ 0x0F，障碍物信息位于数据存储器的 0x10 ~ 0x1F。随机数保存在数据存储器的 0x20 ~ 0x8F。

鸟位置寄存器为$t0，鸟的落体时间寄存器为$t0。

5.4.3 代码实现

设定鸟所在列为右起第 11 列。代码逻辑如下：

1 初始化鸟位置寄存器$t0 的初始值为$2^{16}$，即鸟位于 LED 点阵水平居中位置

2 循环：

2.1 将数据存储器地址 0x1e 的值送入地址 0x1f 内，将地址 0x1d 内的值送入地址 0x1e 中……将地址 0x10 内的值送入地址 0x11

2.2 当其为首次进入循环时，转到 2.6

2.3 若地址 0x15 内的值不为 0，则说明可以产生新的障碍物，转到 2.6

2.4 否则将 0 送入 0x10，表示产生的是障碍间的空隙

2.5 转到 2.7

2.6 读取数据存储器的一个随机数x。将 0xff 左移x位。左移后的 0xff 和 0xffffffff 异或，得到新的障碍信息，送入 0x10

2.7 读取落体时间寄存器$t1 的值$y$。读取鸟位置寄存器$t0 的值x。将$x \gg 2^y$的值送回鸟位置寄存器。

2.8 落体时间寄存器$t1 的值加 1

2.9 将数据存储器地址 0x1f 的值送入地址 0x0f 内，将地址 0x1e 内的值送入地址 0x0e 中……将地址 0x1c 内的值送入地址 0x0c，将地址 0x1a 内的值送入地址 0x0a 中……将地址 0x10 内的值送入地址 0x00

2.10 取出地址 0x1b 内的值（鸟所在列的障碍/空隙信息），与鸟位置寄存器$t0 内的值进行与操作。结果与地址 0x1b 的值相同，则表示碰撞，转到 3。

2.11 否则将结果写入 0x0b。

2.12 将 0 写入数据存储器的 0x90 位置上，进行统一刷新。回到 2.1

3 将 OVER 字样写入 0x00 ~ 0x0f。停机

4 中断程序：

4.1 读取鸟位置寄存器$t0 的值$x$，将$x \ll 2$的值送回鸟位置寄存器中

4.2 若鸟位置寄存器的值为 0，则表明鸟的位置高于 LED 点阵画面，寄存器的值恢复为2^{31}

4.3 将落体时间寄存器的值清空为 0

4.4 取出地址 0x1b 内的值（鸟所在列的障碍/空隙信息），与鸟位置寄存器$t0 内的值进行与操作。结果与地址 0x1b 的值相同，则表示碰撞，转到 3。

4.5 单独刷新鸟所在列

4.6 中断返回

5.4.4 电路实现

在单周期 CPU 电路中，新增 32 个寄存器，用于更新每列的显示信息和同步刷新，如下图所示：

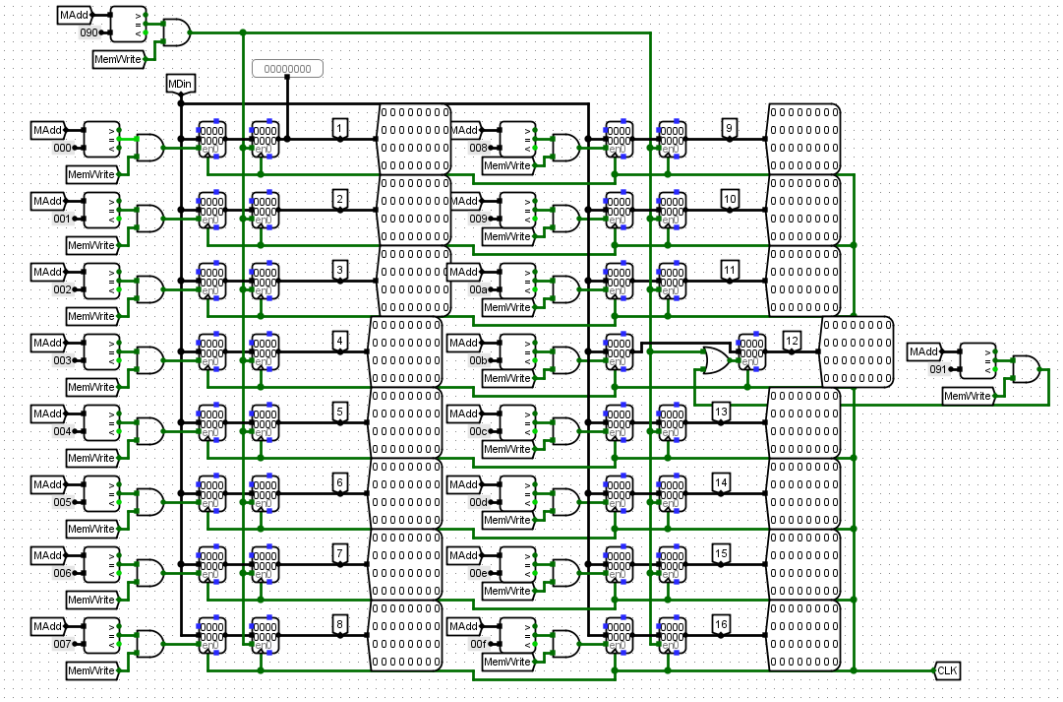

图 5.1 LED 显示信息寄存器电路

将单周期 CPU 进行封装，如下图所示：

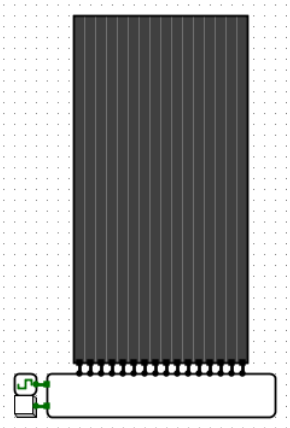

图 5.2 项目封装

电路实现中还包括在硬布线控制器和 ALU 中实现的 sllv、slrv 和 xor 指令。

5.5 团队项目测试

5.5.1 主要故障 1

故障表现：游戏开始时不能产生障碍物，LED 点阵一片空白

故障分析和解决：故障应该位于障碍产生的位置处。检查代码发现，由于没有加上“首次进入循环需要产生一个障碍”的判断，导致一直没有障碍产生，因此也无法根据前一个障碍产生下一个障碍

5.5.2 主要故障 2

故障表现：无法显示代表鸟的点

故障分析和解决：猜测是由于鸟所在列的计算产生错误。检查该部分代码，发现所使用的计算为：障碍信息和鸟位置进行异或操作，且电路尚不支持异或操作。因此首先修改电路硬布线控制器，支持异或操作，然后将异或操作修改为或操作。

5.5.3 主要故障 3

故障表现：当点击按钮后，画面混乱，无法正常控制鸟的位置

故障分析和解决：由于故障发生在向代码添加了鸟所在列独立刷新的代码之后，故猜测问题出现在中断上。通过阅读汇编指令，发现添加的代码位置在恢复现场之后，打乱了原先的顺序。通过调整代码顺序，解决该故障。

5.5.4 游戏测试

游戏过程能够满足设计要求，游戏过程如下图所示：

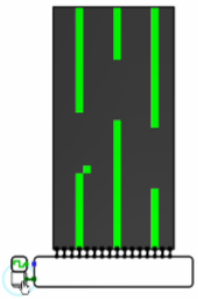

图 5.3 游戏过程截图

当游戏结束，显示为：

图 5.4 游戏结束截图

根据截图可知，能够实现基本功能。

6 设计总结与心得

6.1 课设总结

本次课程设计的过程中，主要完成了如下几点工作：

1） 按照单周期 MIPS CPU 框架图，实现了单周期 MIPS CPU。

2） 基于 24 条基本指令、4 条额外指令、ERET 指令、开中断、关中断的特性，实现了单周期硬布线控制器所支持的共 31 条指令。

3） 基于理想流水线框架图，实现了理想流水线。

4） 基于插入气泡的原理，实现了能够解决数据冲突和跳转冲突的气泡流水线。

5） 基于重定向的原理，实现了重定向流水线。

6） 基于单级中断的原理，设计了单级中断电路，完成了单周期 MIPS CPU 的单级中断。

7） 基于多级中断的原理，设计多级中断电路和 EPC 硬件堆栈，完成了单周期 MIPS CPU 的多级中断

8） 在单周期 MIPS CPU 单级中断的基础上，结合流水线的特性，实现了气泡流水线的单级中断。

9） 学习了动态分支预测的原理，根据其原理设计动态分支预测电路，能够显著提高流水线效率。

10） 设计了团队任务的实现方案。

11） 综合团队工作，完成任务测试和修改。

6.2 课设心得

本次课设是在特殊时刻下完成的，课设要求也随之有所变化。但尽管如此，指导材料足够细致易懂，能够在理解设计原理的情况下摸索完成，难度虽然存在，但并不是难以解决，通过摸索，能够更加理解实现的原理，加深学习的效果。

本次课设最大的难点在于中断和动态分支预测中。中断的原理和实际实现之间的差距比较大，仅仅了解中断的原理不足以设计出中断电路。不论是单级中断的屏蔽还

是多级中断，都是在不断尝试中逐渐完善设计方案，使之能够正常运转。因此虽然最后成功实现了所有中断，但是对于其实现更像是不断试错的结果，而无法总结出很好的设计实现方案。

动态分支预测我认为是实验中最具挑战的部分，尤其是电路设计的部分。在设计的过程中，花了大量的时间和思路尝试理清 BHT 中 IF 和 EX 段各自命中和不命中所对应的逻辑。且具体绘制电路时也相当耗时耗力，尤其是在最后检查运行情况的过程中，需要每个周期进行逐一检查，相当费时费力。

团队合作的主要问题在于，由于大家只能通过网络沟通，导致效率低下，且想法受到硬件的严重限制，导致很多的想法都不能够实现，只能完成一些简单的设计。

对于本次课程设计，我的建议是，如果日后还会需要进行团队设计的话，希望能额外提供一些 Logisim 硬件，使同学的想法能够得到很好地展现而不是受限于非常简陋的硬件。

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 4 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] 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.

[4] 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.

[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：