华中科技大学

2023

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：	CS2108
学　　号：	U202115568
姓　　名：	金钊
电　　话：	15518251995
邮　　件：	242795775@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**错误!未找到引用源。**前 24 条基本 32 位 RISC-V 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。

（15） **表 1 基本指令集**

#	RISC-V	指令类型	简单功能描述	备注
1	ADD	R	加法	指令格式与功能 请参考 RISC-V32 指令集英文手册，或参考 RARS 模拟器
2	ADDI	I	立即数加	
3	AND	R	与	
4	ANDI	I	立即数与	
5	SLLI	I	逻辑左移	
6	SRAI	I	算数右移	
7	SRLI	I	逻辑右移	
8	SUB	R	减	
9	OR	R	或	
10	ORI	I	立即数或	
11	XORI	I	或非/立即数异或	

12	LW	I	加载字	
13	SW	S	存字	
14	BEQ	B	相等跳转	
15	BNE	B	不相等跳转	
16	SLT	R	小于置数	
17	SLTI	I	小于立即数置数	
18	SLTU	R	小于无符号数置数	
19	JAL	J	转移并链接	
20	JALR	I	转移到指定寄存器	
21	ECALL	I	系统调用	if ($a7==34) LED 输出$a0 的值 else 停机等待 Go 按键按下
				注意显示逻辑需要考虑如何锁存过去的数据，否则数据一闪而过。
22	CSRRSI	I	访问 CSR 寄存器	中断相关，可简化为开中断
23	CSRRCI	I	访问 CSR 寄存器	中断相关，可简化为关中断
24	URET	I	中断返回	清中断，mEPC 送 PC，开中断
CCAB 扩展指令集				
25	SLL	I	移位	
26	AUIPC	U	组成 call 指令	
27	SB	S	存字节	
28	BGE	B	大于等于跳转	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 本次采用的方案是硬布线控制方案，采用将指令与数据分开存储的哈佛架构，使用硬布线将二者进行联系。为便于开发，本次采用同步时序，CPU 内全部元件听从同一时钟周期。在一个周期内，首先控制器读取指令并转化得到 ALU 所需指令，然后从指令的其余部分获得目的寄存器或者立即数，随后将转化得到的数字经过 ALU 计算后写入内存或寄存器。

总体结构图如图 2.1 所示。

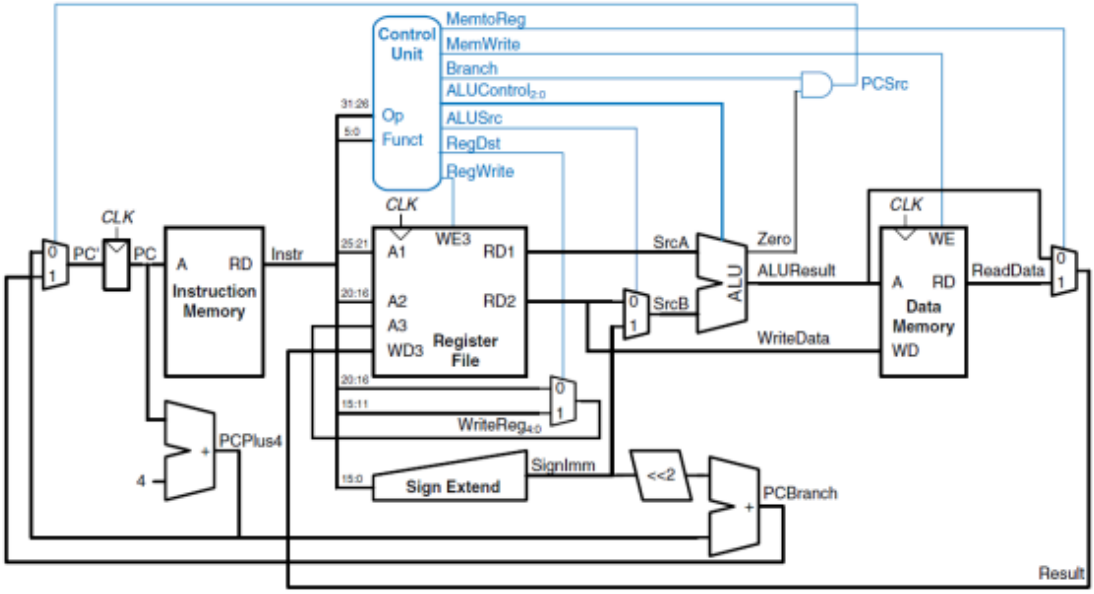

图 2.1 总体结构图

2.1.1 主要功能部件

以下为单周期 CPU 的几个核心部件的设计原理。

1. 程序计数器 PC

程序计数器为一个寄存器，用于存储下一条将要运行的指令地址。其输入从多种可能的下条地址中选取，经过时钟周期的控制输出即将执行的地址，传输给 IM。

2. 指令存储器 IM

指令存储器存储了程序所有可能运行的指令，具体执行指令地址由 PC 的输出地址决定。由于 RISC-V 为定长指令，每条指令均为 4 字节，因而 IM 中地址仅由 PC 输出地址中的 2-11 位决定。

3. 运算器

运算器负责根据控制器产生的 ALUOP 指令，将两个操作数进行运算。器输入和输出接口如下：

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
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

每个 ALUOP 对应的 ALU 功能如下表：

表 2.2 算术逻辑运算单元 ALUOP 与对应功能描述

输入 OP	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

输入 OP	功能描述
4	无符号除法，result2 存储余数
5	加法
6	减法
7	按位与
8	按位或
9	按位异或
10	按位或非
11	有符号比较，小于输出 1
12	无符号比较，小于输出 1

4. 寄存器堆 RF

寄存器堆是由 32 个 32 位寄存器构成，其输入引脚与输出引脚的名称与功能如下：

表 2. 3 寄存器堆输入输出引脚名称与对应功能描述

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

5. 内存 MEM

内存和指令存储器结构类似，也是由 10 条地址线（2-11 位）进行寻址，要求访问按字对齐。寻址的地址来源于寄存器中的地址或者 ALU 的计算结果。

由于需要实现 SB 指令，完成对特定字节的写入操作，因此选择使用 MIPS RAM 存储器。MIPS RAM 有一个 sel 输入，当 sel 是选择控制字段，线宽 4 位，当 sel=0001 时，D 端口的 0~7 位被选中，当 sel = 0010, D 端口的 8~15 位被选中，当 sel＝0100 时,D 端口 16~23 位被选中，当 sel=1000 时，D 端口 24~31 位被选中。另外 sel=0011，表示 0~15 位被选中，sel=1100，表示 16~31 位被选中，sel=1111 时，4 个字节同时被选中。这为 SB 指令的实现提供了很大的方便。

2.1.2 数据通路的设计

图 3 为取指令的数据通路,涉及到的功能部件包括程序计数器 PC、指令存储器、加法器等。取指令通路应能取出程序计数器 PC 锁存地址对应的的机器指令，并能在时钟上跳沿到来时实现 PC 自动加 4，从而进入下一条指令的指令周期。这里程序计数器 PC 可以采用寄存器实现，其输出作为指令存储器的地址输入，指令存储器经过一个存储周期后一次性取出 32 位的机器指令，故其数据宽度应为 32。需要注意的是，无论是在 Logisim 平台实现还是 FPGA 上实现，设计者均需考虑指令存储器地址位宽的问题，首先在 FPGA 实现中指令存储器地址宽度越大综合速度越慢，另外 PC 锁存地址是 32 位字节地址，指令存储器输入地址是字地址，二者不匹配；为简化电路设计，Logisim 中建议采用 ROM 实现。

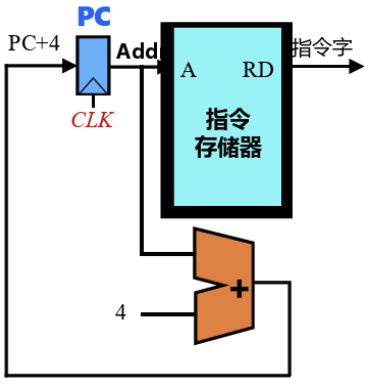

图 2.2 取指令数据通路

在顺序执行方式下，每一个时钟周期内 CPU 取指令后将 PC 寄存器的值加 4，形成下一条指令的地址。这里加“4”是因为 RISC-V32 中指令字长均为 4 字节，每条指令在存储器中占用 4 个字节的存储单元，而 PC 中存放的地址是字节地址。为避

免资源冲突，这里加法器应该是独立的器件，和 CPU 中的算术逻辑运算单元分开。

指令取出后即可进入指令的执行周期，接下来我们可以开始设计能支持最为简单的 R 型指令的数据通路，使得能运行一条指令的简单的 CPU 能运行起来。

2）构建 R 型指令数据通路

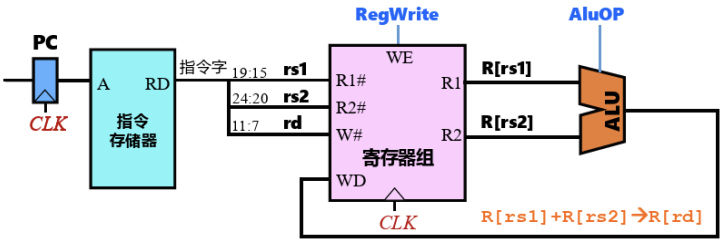

图 2.3 R 型指令数据通路

RISC-V32 中算术逻辑运算指令属于 R 型指令，R 指令所有的操作数均是寄存器，执行过程中涉及的功能部件包括寄存器文件和 ALU。R 型指令的数据通路如图 4 所示，指令执行过程中的 ALU 的两个源操作数均来自于寄存器文件输出，其中 R[rs1]是寄存器 R1#的值，R[rs2]是寄存器 R2#的值，R1#来自于指令字中的 rs1 字段， R2#来自于指令字中的 rs2 字段，运算结果写入目的寄存器 rd 中，将 AluOp 设置为不同的值，就可以进行不同的运算。U 型指令也需要向寄存器写入，但是参与运算的操作数不再是寄存器的值。这里选择使用专用的运算组件实现，复用向寄存器组写入的电路即可。

3） 构建 I 型运算指令数据通路

I 型运算类指令另外一个运算操作数来自 12 位立即数，12 位立即数经 32 位符号扩展器扩展后送入 ALU，二者进行运算得到最终的结果，运算结果写入目的寄存器 rd 中，同样将 AluOp 设置为不同的值，就可以进行不同的运算，数据通路如图 5 所示。

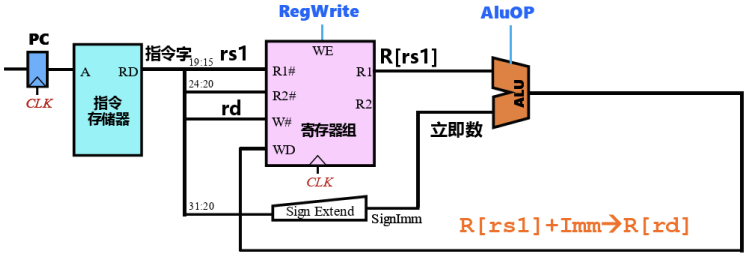

图 2.4 I 型运算指令的数据通路

4） 构建访存指令数据通路

访存指令属于 I 型指令，访存地址等于变址寄存器 rs1 的值加上 12 位立即数，地址运算通过 ALU 完成，所以需要将 rs1 的值送入 ALU，同时要将 12 位立即数进行 32 位符号扩展后送入 ALU，二者进行加法运算得到最终的访存地址，图 6 是访存指令操作的部分数据通路。该数据通路包括指令存储器、寄存器文件、符号扩展、ALU、数据存储器等功能部件。如果是加载指令，则数据存储器的结果将会送回到寄存器文件的 WD 端，写入寄存器编号为 rd。

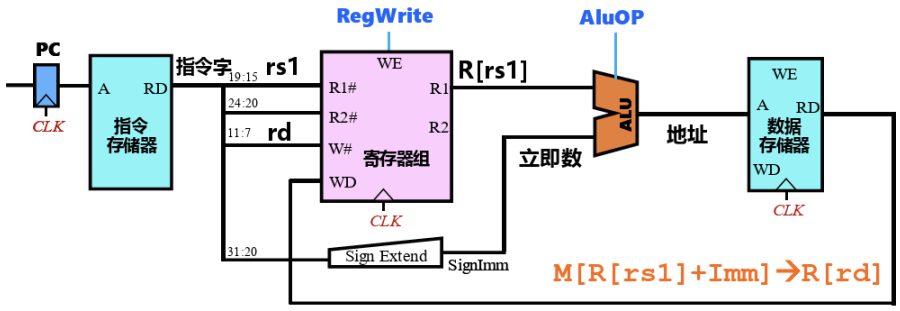

图 2.5 lw 指令的数据通路

如果是存储指令，则 rs2 寄存器的值会通过 R2 端口输出到数据存储器的写入端口 WD，如图 7 所示。

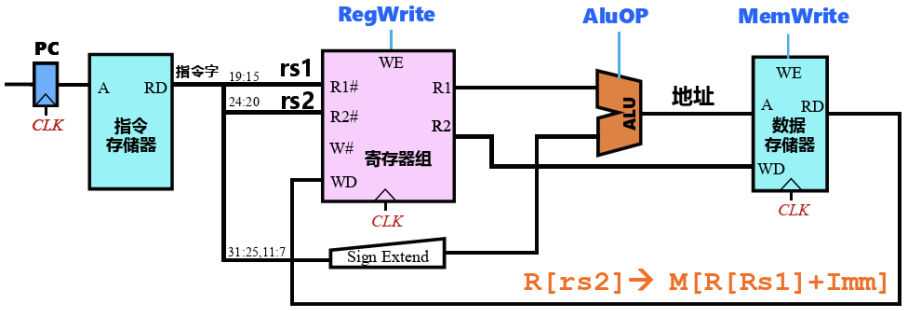

图 2.6 sw 指令的数据通路

对比不同类型指令的数据通路可发现，寄存器组写入寄存器编号 WD 的输入，ALU 第二个操作数输入，符号扩展器 SignExtend 的输入都包含多个不同的输入来源，为了将不同的数据通路统一到同一个电路中，我们需要在有多个输入来源的端 口增加多路选择器，如图 8 所示，每增加一个多路选择器就额外引入了一个控制信 号，这里分别增加了 S_Type(为 1 表示 S 指令)、AluSrc（为 1 选择立即数送 Src）、MemtoReg（为 1 将内存数据写回寄存器）3 个控制信号。这些控制信号都应该由硬 布线控制

器根据指令译码自动生成，在完成对应控制逻辑之前，我们可以先手动设 置对应的值，验证数据通路的功能是否正常。当数据通路能正常工作时，就可以在 控制器中增加相应的组合逻辑自动生成新增加这些控制信号。至此我们设计的 CPU 已经可以支持部分 R 型、I 型、S 型，U 型指令。跳转指令等需要专用电路进行计算等操作，在这里不再详细介绍。

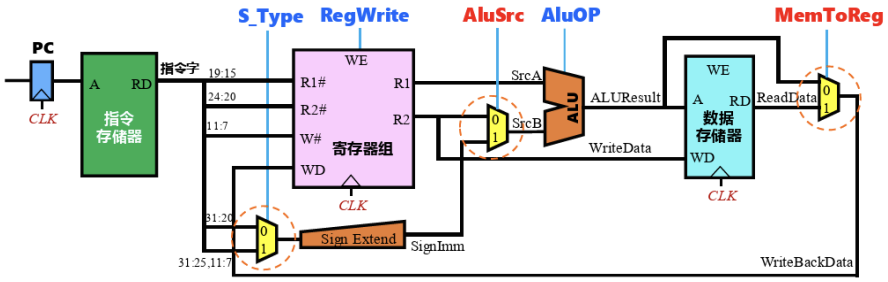

图 2.7 支持算术逻辑指令和访存指令的混合数据通路

2.1.3 控制器的设计

控制器需要从指令中得到源操作数的具体数值（如 I 型指令）或地址，以及对取出的数字进行的操作种类，和最后写入的寄存器编号或内存地址。其中最核心的就是控制信号，通过控制信号可以对整个电路的功能进行控制。控制器的控制信号定义与作用如表 2.3：

表 2.2 主控制器控制信号的作用说明

控制信号	取值	说明
R1_used	0、1	是否需要使用操作数 R1
R2_used	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

控制信号	取值	说明
BGE	0、1	是否为 BGE 指令
JALR	0、1	是否为 JALR 指令
JAL	0、1	是否为 JAL 指令
SLL	0、1	是否为指令
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

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。该控制信号表如图 2.3 所示。为简化表格，因而 BEQ 类指示是否为该条指令的输出被省略了。

表 2.3 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	

指令	ALU_OP	MemToReg	ALU_Srcb	RegWrite	Ecall	S_Type	RS1_used	RS2_used
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			
SRA	1						1	1
XOR	9						1	1
LBU	5	1					1	
BLT	11		1				1	1

2.2 中断机制设计

2.2.1 总体设计

中断类型大体可以分为内部中断和外部中断，本次实验设计的中断为可屏蔽的外部中断。本次实验中支持单周期单级中断、单周期多级中断和流水单级中断。

一般地，完成一次外部中断的过程如下：保存下一条指令地址，保护现场的寄存器，然后 PC 跳到中断源处，CPU 开始执行中断。然后根据 ecall 指令恢复中断现场的寄存器，指令跳转到存储的下一条指令的地址，

于单级中断，因为在执行中断服务程序的时候不会再接受新的中断请求，因而可以使用一个寄存器来保存当前是否处于执行中断的状态，以及中断返回地址。对于多级中断，需要考虑不同等级的中断，以及执行中断服务程序时不能被低级和同级中断打断，但是允许被更高级的中断所中断，因而需要根据执行中断服务程序的进度（即是否正在保存和恢复现场）来灵活调整中断指示标志，具体可以使用 CSSRCI 和 CSSRSI 两条指令来控制。此外，多级中断的实现还需要能够保存多次断点并顺序返回，这里可以选择使用硬件堆栈实现。

2.2.2 硬件设计

为了实现中断机制，需要完成中断信号产生电路，中断使能信号产生电路，断点保存以及中断返回电路。对于多级中断，还需要完成硬件堆栈。对于流水中断，还需要对流水组件做出适当的修改。其中中断信号产生电路，中断使能信号产生电路，断点保存相对简单，只需要通过寄存器和多路选择器和解复用器对相关信号做出适当选择即可。硬件堆栈的实现根据实验的需要，需要能够实现三级嵌套中断，可以使用加法器，多路选择器和寄存器来进行实现。流水中断的实现则需要跟据流水组件的设计进行。

2.2.3 软件设计

对于采用内存堆栈保护的多级中断实现，在中断程序保护现场时需要将 mepc 保存在软件堆栈中，而硬件堆栈则不需要这一步。硬件堆栈在触发中断隐指令后就会将 mepc 压入硬件堆栈保存。这就是二者在程序上的主要差别。其他方面则保持一致。

2.3 流水 CPU 设计

2.3.1 总体设计

本次实验实现的是五段流水线 CPU，即将一条指令的执行过程分成取指 IF、译码 ID、执行 EX、访存 MEM、写回 WB 五个阶段，通过四组寄存器保存每一段所需要的信息来实现每条指令五个阶段执行的相对独立性，这也是理想流水线的基本设计原理。但是在实际执行过程中，由于存在无条件和有条件跳转指令，因而有些指令的具体效果需要等待上条指令的结果，破坏了指令之间的隔离关系。因而在理想流水线上根据消除影响逻辑和方式不同，分成了气泡流水线和重定向流水线这两大类。此外，利用动态分支预测可以进一步减少重定向流水线的气泡，提高 CPU 的 IPC。

2.3.2 流水接口部件设计

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 2.5 不同阶段所需流水线接口

阶段	所需流水线接口
IF/ID	IR、PC、PC+4、PredictJump
ID/EX	BEQ、BNE、MemToReg、MemWrite、AluSrcB、RegWrite、S_Type、Ecall、JAL、JALR、SLL、AUIPC、URET、ALUOP、RD、R1、R2、IMM_1、IMM_2、PC、PC+4、IR、RS1_Forward、RS2_Forward、PredictJump
EX/MEM	JAL、JALR、MemToReg、MemWrite、LBU、RegWrite、Halt、RD、Result、WriteData、IR、PC、PC+4
MEM/WB	JAL、JALR、MemToReg、RegWrite、Halt、RD、ALUResult、ReadData、IR、PC、PC+4

需要指出的是，本次实验中需要使用到流水组件的有：理想流水线 CPU，气泡流水线 CPU，重定向流水线 CPU，动态分支预测 CPU。它们对于流水组件的接口需求是不同，这里给出的是功能最全的动态分支预测的流水组件接口信息，它可以向前兼容其他 CPU 的流水组件。

2.3.3 理想流水线设计

理想流水线由于不涉及同时进入流水线的指令之间的相互作用，因而只需要将单周期的 CPU 按照上述五个步骤直接进行拆分，使用四组流水线接口即可。对于停机指令，在 EX 段 CPU 处理完停机逻辑后，需要经过两个周期传送到 WB 段再最终执行停机操作，以保证全部指令都执行完毕再停机。

2.4 气泡式流水线设计

为了支持同在流水线的指令的相互作用关系，在理想流水线的基础之上需要增加由当前的指令操作信号控制的 DataHazzard 组合逻辑以判定是否需要插入气泡，插入气泡等效于清空流水线接口内寄存器的值。DataHazzard 的主要判定逻辑是根据源寄存器的使用状态和数据相关检测来判定是否存在冲突。具体而言，若判定需要插入气泡，则 ID/EX 接口和 IF/ID 接口均需要清空，以跳转到正确的指令处重新执行。由于指令的相互作用，气泡被插入了流水线，导致流水线的效率下降。

2.5 数据转发流水线设计

数据转发流水线（下称重定向流水线）以气泡流水线的基础之上进行了一定的优化，结构类似，但是减少了气泡的插入。重定向将数据可能的全部路径进行汇总，然后根据指令的执行情况获取数据，选择一条分支。但是若相邻两条指令存在数据相关，且前一条是访存指令（称为 Load_Use），这里就需要做出 trade off。对经过存储器的

数据进行重定向需要经过运算器和存储器两个组件，关键路径过长，不利于缩短 CPU 的周期，得不偿失，因此选择使用气泡插入空周期来保证数据的一致性。

2.6 动态分支预测机制

动态分支预测是在重定向流水线上进一步的优化。重定向只是在气泡流水线上减少了气泡的产生，但是并未减少分支误取得代价，导致了周期的浪费。动态分支预测使用一个 BHT 表来记忆过去的跳转指令，使用 LRU 算法来动态更新 Cache 槽的存储内容，同时使用双预测位的四状态的状态机来判定是否需要跳转（PredictJump），不断根据当前实际跳转情况来进行状态机的更新。具体而言，电路在 IF 段通过状态判断出预测结果，在 EX 段根据实际结果通过硬件来更新状态机。通过这两个方法来实现预取得正确的指令，减少分支误取的代价。

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。

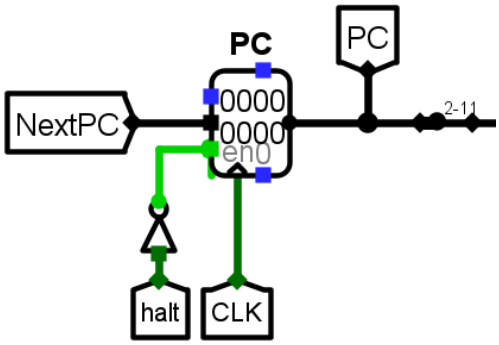

图 3.1 程序计数器（PC）

NextPC 是下一个时钟周期的 PC 的值。由于存在跳转指令，需要根据当前周期的指令情况做出选择。具体实现如下：

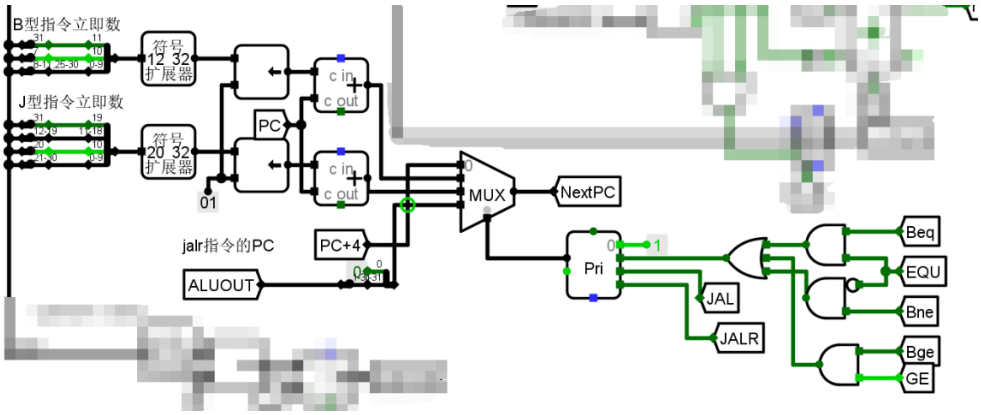

图 3.2 NEXTPC 生成电路。

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.所示。

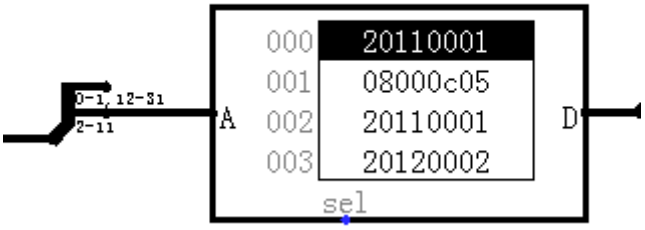

图 3.3 指令存储器（IM）

3) 寄存器（Regfile）

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 3.3 所示：

4)

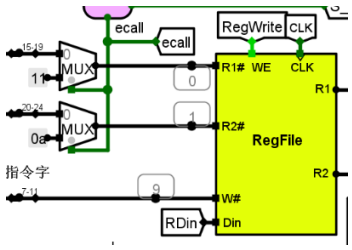

图 3.4 寄存器文件（RegFile）的 Logisim 实现

5) ALU

实验已经提供了 ALU 的电路，不需要学生自己去实现，在此省略。

3.1.2 数据通路的实现

数据通路基本由 PC->ROM->寄存器和控制器->ALU->MEM 的流程构成，有的指令不会完成完整的通路根据每条指令的不同，控制信号会生成不同的数据选择通路，从而获取指令产生的数据。数据通路的选择高度依赖控制器产生的控制信号，因此在这里无法详细说明。下图为 Logisim 中产生的数据通路：　　。

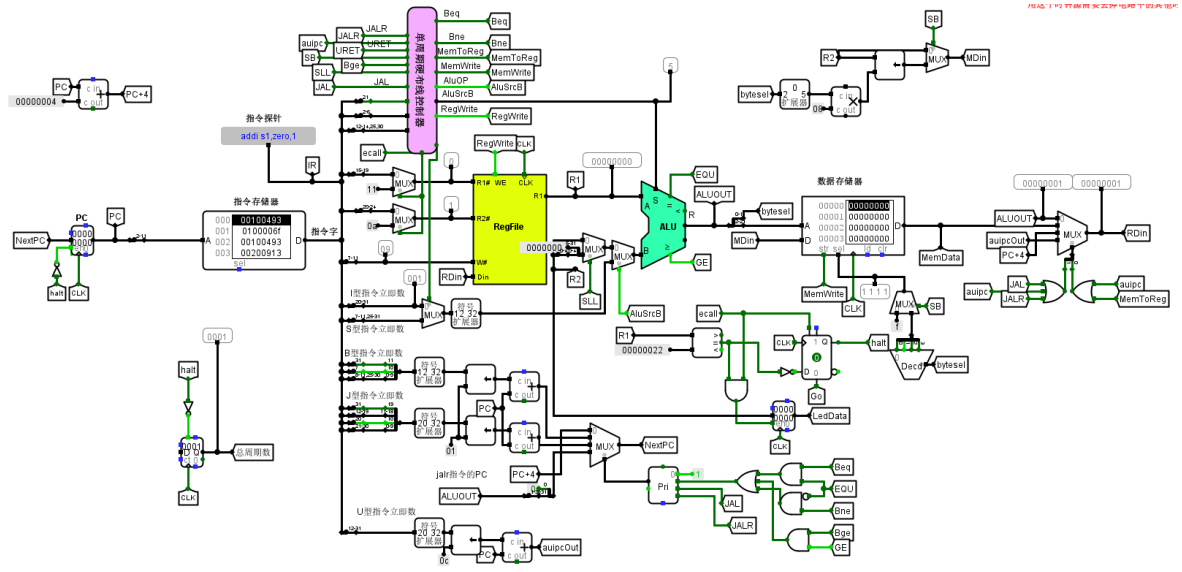

图 3.5 数据通路

3.1.3 控制器的实现

根据总体方案设计中控制器的设计小节的相关内容，完成提供的 excel 表格，在 logisim 中分析工程，自动生成电路即可。

主控制器主要由两部分组成，ALUOP 产生电路和控制信号产生电路。根据需要完成的指令，结合指令手册完成 excel 表格获取逻辑表达式，将表达式填入 logisim 即可。由于表格内容过多，自动生成的电路规模太大，在这里省略，仅展示最后完成的控制器的电路：

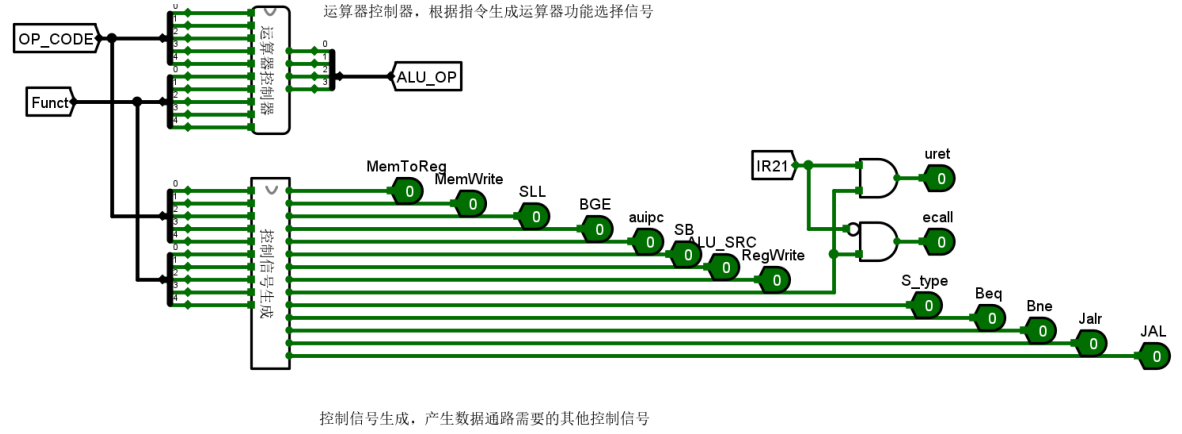

图 3.6 控制器电路

各类型数据通路在完成后并不是根据指令类型导通特定的数据通路，而是根据指令类型选择特定通路的数据。在本次实验完成的 CPU 还需要根据这些控制信号组合产生需要的信号。如对于分支指令，需要组合产生 PC 选择的信号。如图：

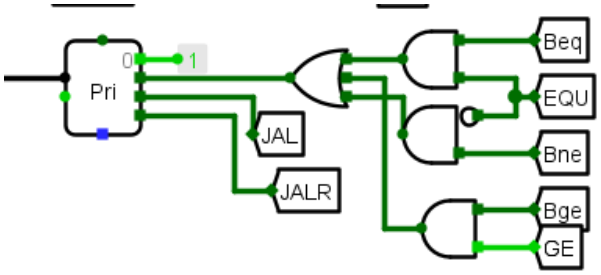

图 3.7 分支信号选择电路

向寄存器堆中写入的数据同样需要进行选择，根据指令的不同选择正确的数据通路的数据进行写入。寄存器写入数据的控制信号如图：

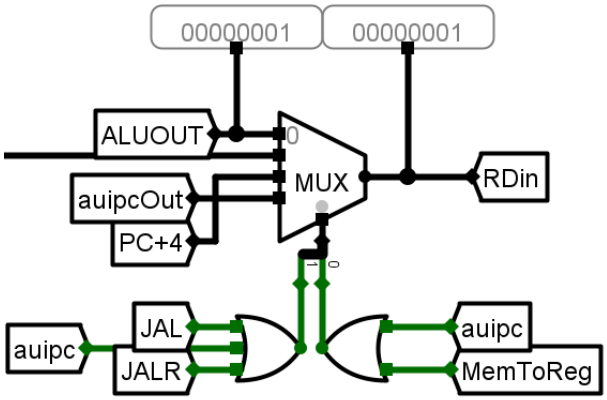

图 3.8 寄存器写入选择电路

为了实现 ecall 指令的停机功能，需要产生停机控制信号 halt。停机的条件是：if ($a7==34) LED 输出$a0 的值，else 停机等待 Go 按键按下。这里选择使用 D 触发器保存比较状态，将比较结果取反后作为信号输入，ecall 信号作为使能信号，Go 信号作为置位信号即可。

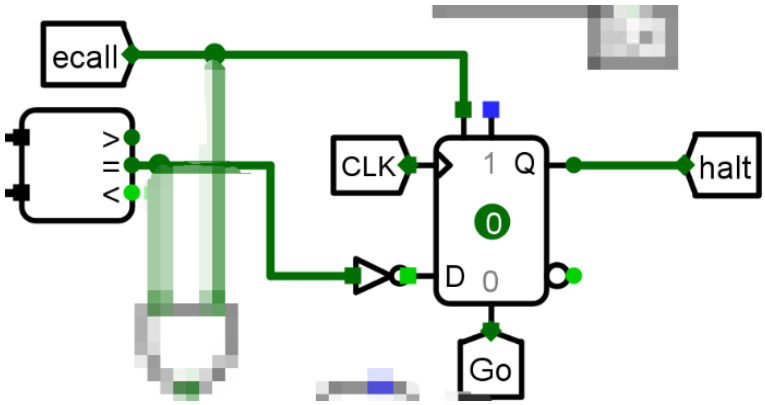

图 3.9 停机信号生成电路

3.2 中断机制实现

3.2.1 单级中断

单级中断和多级中断都需要涉及下一条指令的跳转地址。在单周期 CPU 的三种选择之上，使用优先编码器和多个选择器通过控制信号从 PC+4、中断源、EPC 中地址选择下一条地址，中断源地址可以通过 RARS 仿真器看到具体的地址。Logisim 原理图如图 3.8 所示：

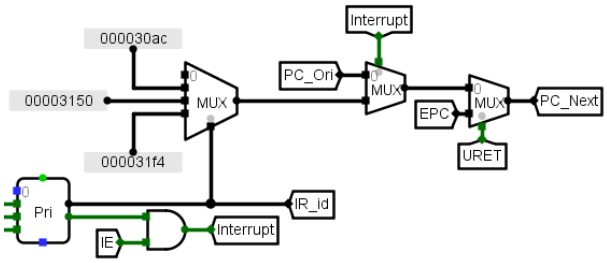

图 3.10 跳转地址选择电路（Logisim）

硬件实现上通过使用 EPC 和 IE 寄存器来保存当前中断的状态和返回地址，以及清空中断请求队列。电路如图 3.9 所示：

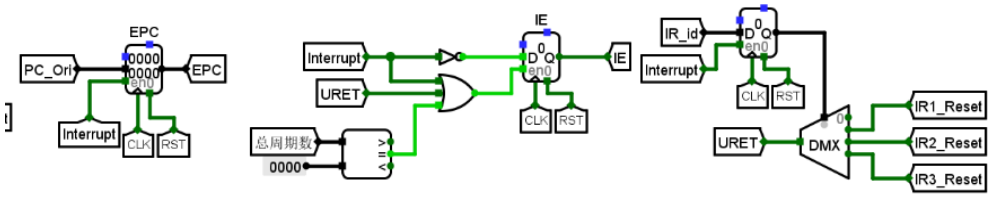

图 3.11 硬件保存中断电路（Logisim）

3.2.2 多级中断

相比于单级中断，多级中断需要能够保存三套 EPC 和 IR。此外，还要能够判断中断优先级来确定中断使能信号。这里采用硬件堆栈保存保存中断号。根据中断号，使用译码器和多路选择器来实现不同中断的 EPC 的获取。此外，还需要使用 CSRRCI 和 CSRRSI 来控制中断的开关，这个主要是通过对中断使能信号 IE 的控制实现的。

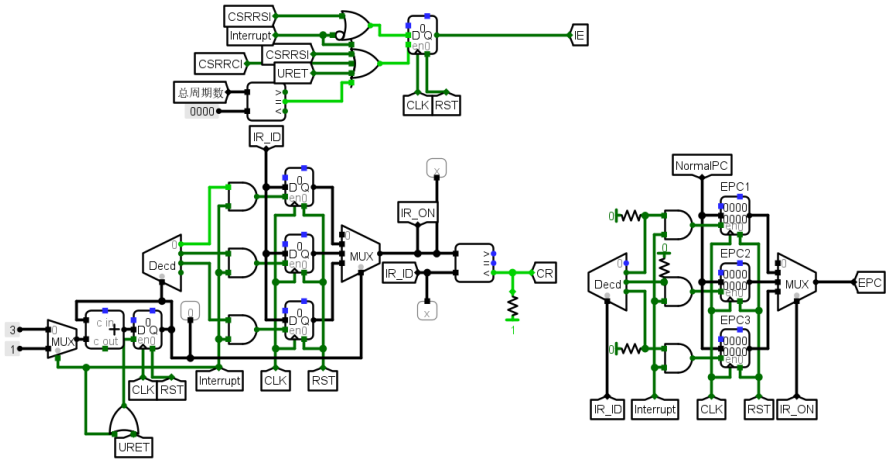

图 3. 12 多级中断的硬件堆栈和中断使能信号生成

3.2.3 流水中断

本次实验中实现的流水中断是在重定向流水线的基础上实现的多级中断（硬件堆栈保存）。为了实现流水中断，选择从清空 IF，ID 和 EX 段的指令，并将 EX 段的 PC 作为返回地址保存。因此在在重定向流水线的 IF/ID 流水组件和 ID/EX 流水组件 rst 信号添加 interrupt 触发，在 IF 段和 IF/ID 流水组件的 rst 信号添加 IF.URET 判断，将 EPC 修改为 EX.PC 即可实现流水中断机制。流水组件的修改如下图所示：

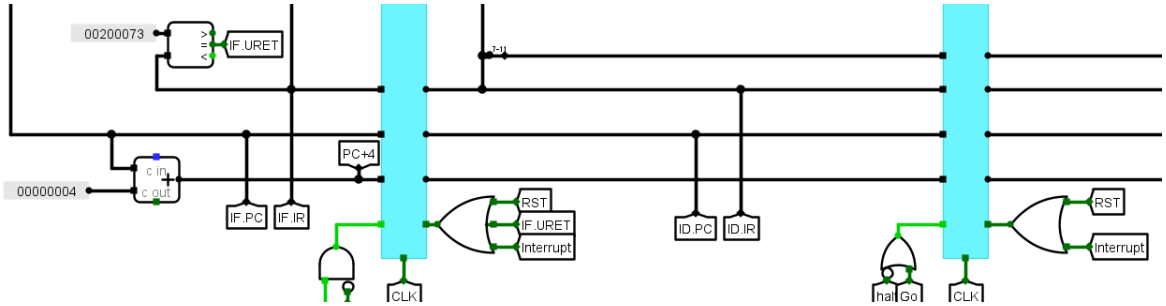

图 3. 13 流水中断控制信号

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

在流水线 CPU 中，流水接口组件的作用是实现指令流水线的各个阶段之间的数据传输和控制信号传递。它起到了连接不同阶段的桥梁作用，并协调各个阶段的工作，使得流水线能够顺利执行指令。本次实验中流水组件需要能够正确跟据时钟信号变化传输数据，根据 rst 完成流水清空，根据 halt 和 go 信号完成 CPU 的停机和启动。部分流水组件还需要根据其他控制信号完成清空和暂存操作。

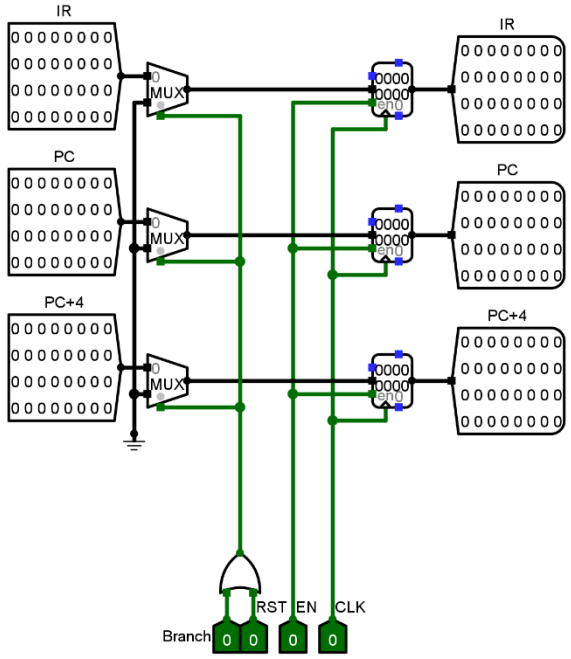

图 3.14 流水组件

上图为理想流水线的 IR/ID 段流水组件的实现，该流水组件只需要传输 IR，PC，PC+4 即可。使用多路选择器来实现流水清空功能。之所以使用多路选择器实现流水清空而不是使用寄存器的清零信号，是为了确保流水组件完成暂存功能，确保指令完成执行。其他阶段流水组件的设计结构基本相同，只是因传递的信号不同添加了不同的流水接口。由于电路规模较大，在此省略。

3.3.2 理想流水线实现

理想流水线不需要考虑数据冲突和分支指令，只需要将 CPU 分割成五段流水线顺序执行即可，实现较为简单。如下图所示：

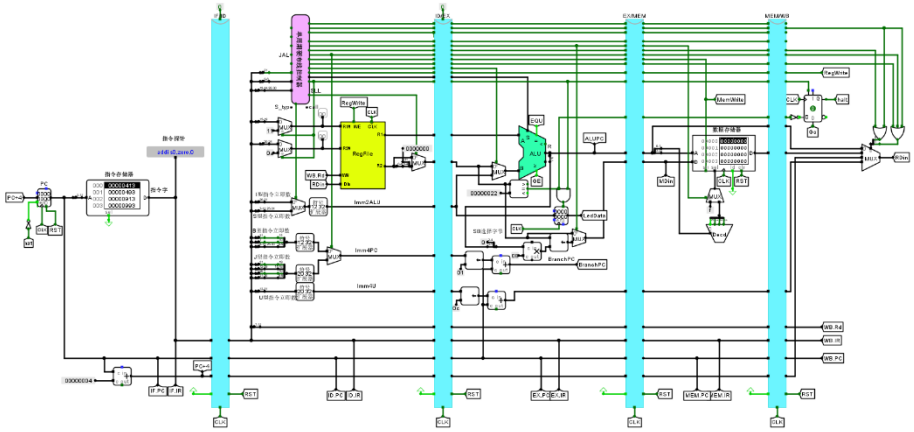

图 3.15 理想流水线的实现

3.4 气泡式流水线实现

气泡流水线需要考虑到分支指令和数据冲突的影响，通过插入空指令，来解决冲突。数据冲突的判断通过生成 DataHazzzard 信号来判断。这个信号将会判断是否同时存在寄存器写信号和寄存器使用信号，如果存在将会产生数据冲突，DataHazzard 信号为 1。分支的判断将会在 EX 段完成，这是为了将条件分支和无条件分支一起处理。当 CPU 需要进入分支时，BranchTaken 信号为 1

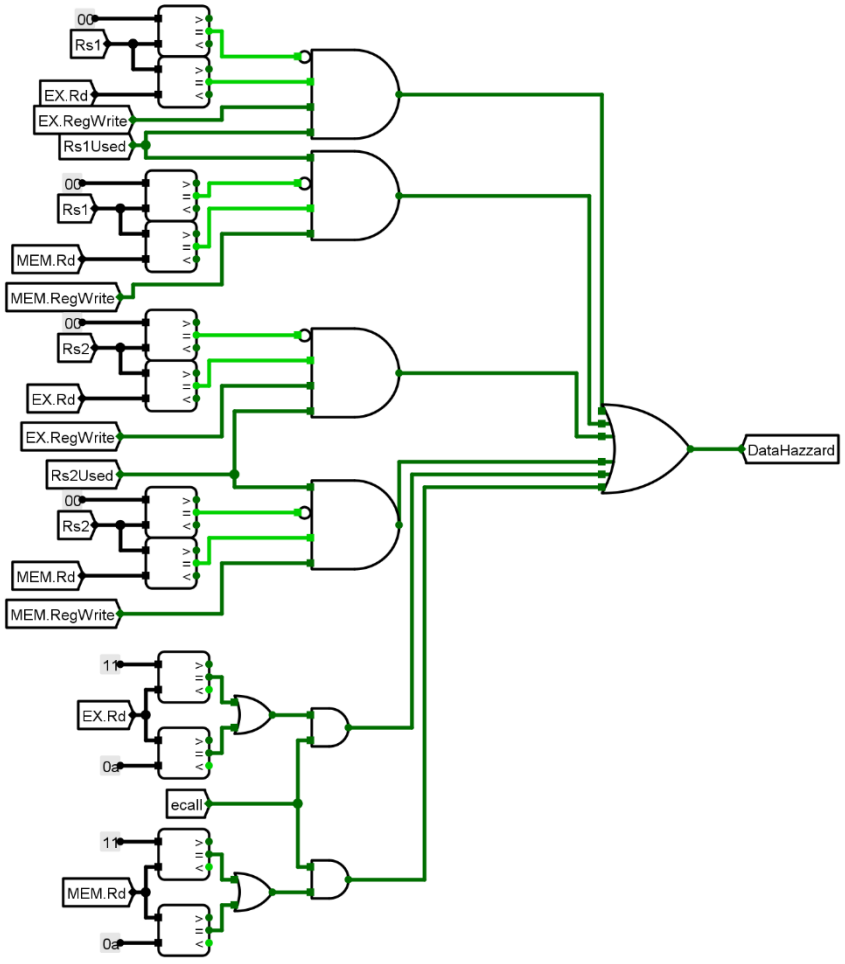

图 3.16 气泡流水线数据冲突信号生成

当 DataHazzard 信号或 BranchTaken 信号为 1 时，流水处理逻辑将会向 CPU 插入气泡，flush 信号将会刷新 ID/EX 段的流水组件，branchTaken 信号会刷新 IF/ID 段的流水组件，来取消指令的执行以解决数据冲突或分支误取。此外，数据冲突的还会产生 stall 信号，暂停一个时钟周期来确保指令完成指令执行，然后继续向后执行指令以避免数据冲突，确保指令按照正确的顺序和数据完成执行。整个气泡流水线的结构如下图所示：

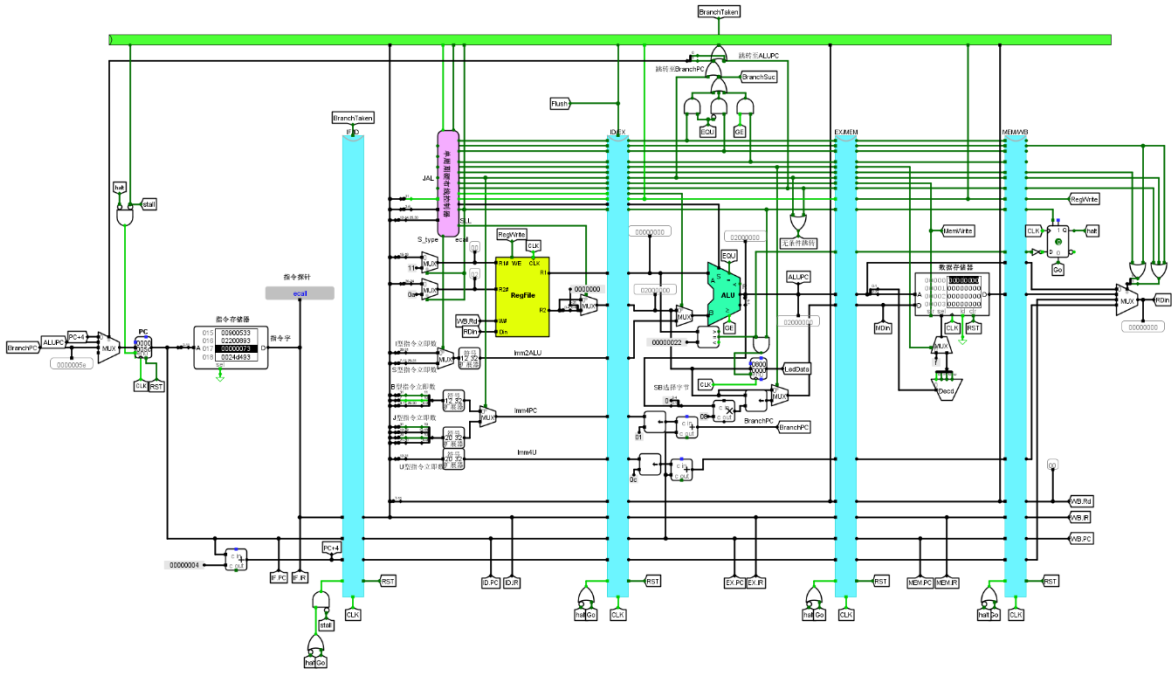

图 3.17 气泡流水线 CPU

3.5 数据转发流水线实现

通过直接将 MEM 段和 WE 段的数据转发到 EX 段，可以确保数据一致性，避免了插入气泡造成 CPU 流水线性能下降。但是数据转发不能解决全部的数据冲突，当出现 Load_Use（从内存向寄存器写入）时，为了避免流水周期过长，提高 CPU 的频率，依然选择插入气泡的方式解决。Load_Use 判定逻辑如下：

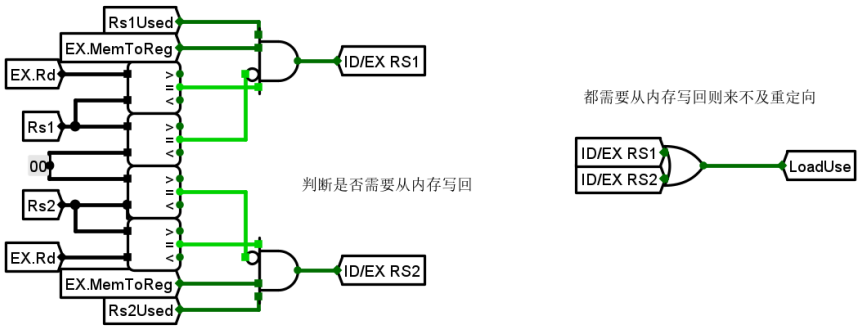

图 3.18 Load-Use 信号生成

在处理完 Load_Use 之后，还需要处理数据转发逻辑。当数据不需要经过访存即可获取的情况下，可以选择使用数据转发的方式解决数据冲突。如果 MEM，WE 段的使用的寄存器和 EX 段的寄存器相同，可以将数据重定向到 EX 段送入 ALU 执行。其中多路选择器的选择信号需要区分数据来自 MEM 段还是 EX 段。实现如下：

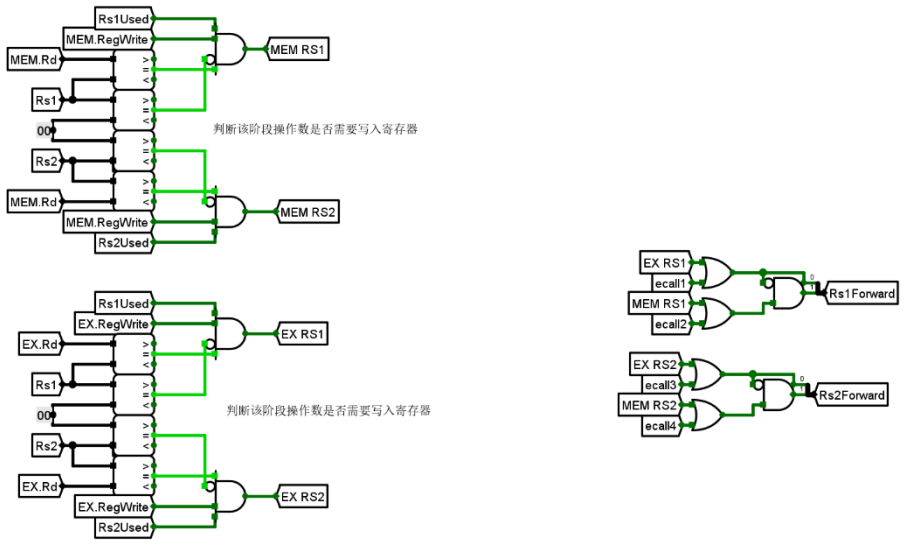

图 3.19 重定向信号生成

重定向流水线和气泡流水线在其他方面差异不大，这里不再赘述。下图为重定向流水线 CPU 的完整实现：

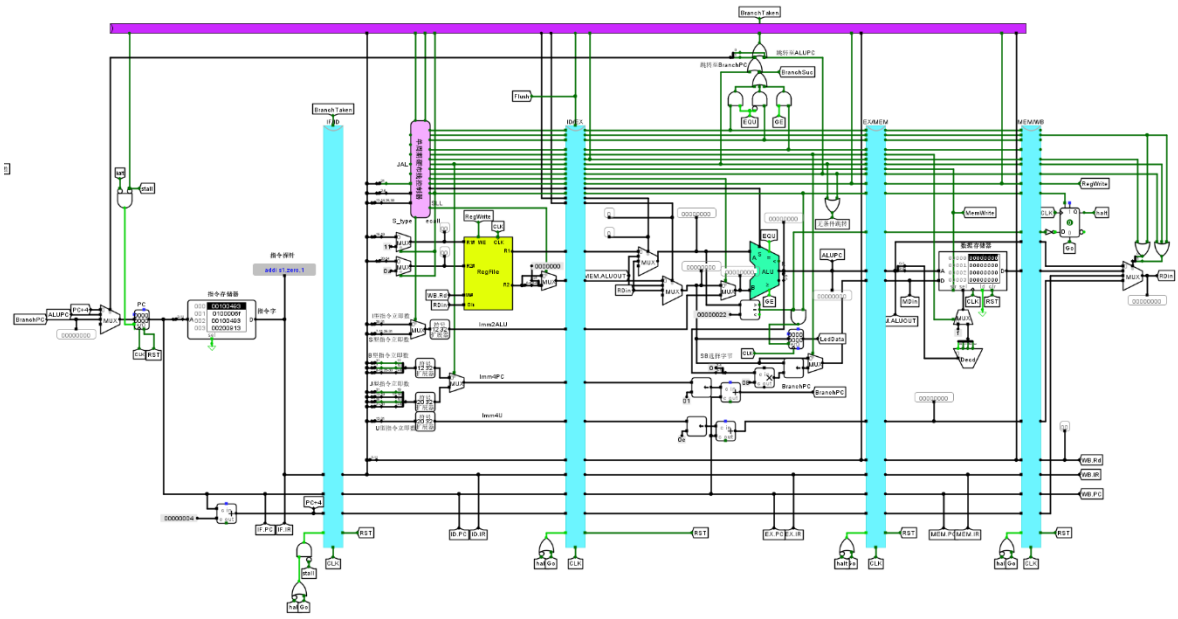

图 3.20 重定向流水线 CPU

3.6 动态分支预测机制实现

动态分支预测的实现主要依靠分支预测缓冲器，用于存放分支指令的分支跳转历史统计信息。BTB 的表项包括有效位，分支指令地址，分支目标地址，分支预测历史位，置换标记。这里选择使用 8 位 cache 槽存储以上信息，每一位 cache 槽如下图所示：

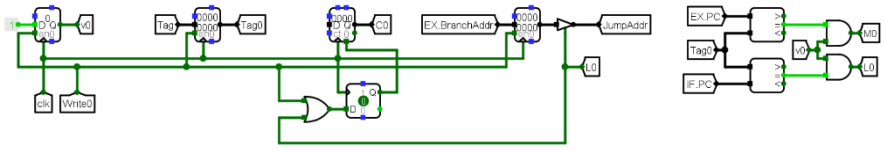

图 3.21 BHT 单个表项

根据优先写入空 cache 槽，优先覆盖在早被使用的跳转地址的原则，实现 LRU 逻辑的电路设计如下：

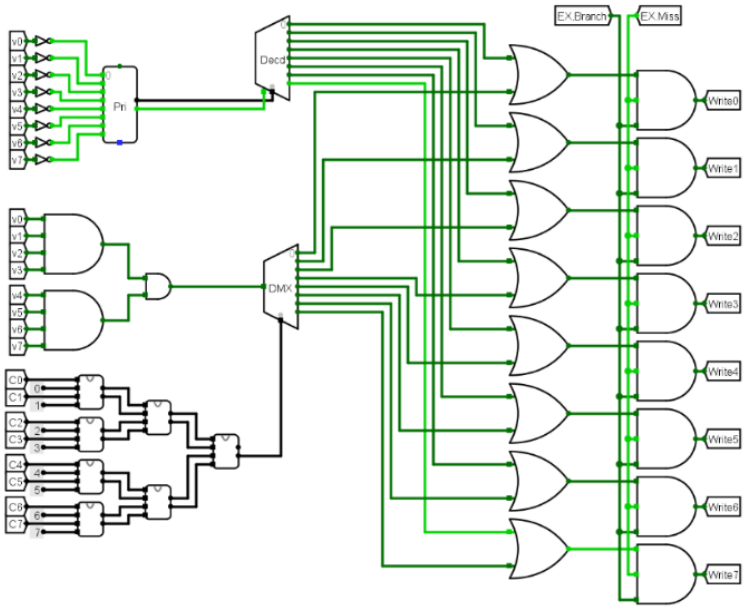

图 3.22 LRU 逻辑实现

由于使用双位分支预测历史，还需要将使用 FSM 对现有的分支预测历史位和预测结果进行转换。每条信息都需要设置一个，共计 8 个。

3.7 CCAB 拓展指令支持

AUIPC，BGE，SLL 指令只需要在指令类型对应的数据通路添加相应的控制信号，使用通用的组件即可完成功能。但是 SB 指令由于需要使用 ALU 的运算结果再次参与运算，这里选择使用专用的运算组件完成支持。

SB 指令是一个 S 型指令，将下 x[rs2]的低字节存入内存地址 x[rs1]+sign-extend(offset)处。SB 指令的格式为：SB rs2，offset(rs1)。使用 RTL 描述为：M[x[rs1]+sext(offset)=x[rs2][7:0]。在使用 ALU 计算获取地址后，还将 rs2 的低字节按照地址低两位的偏移量移动到指定位置并获取存储器的偏移量，然后将其存如存储器中。对 rs2 进行的处理过程如下图所示：

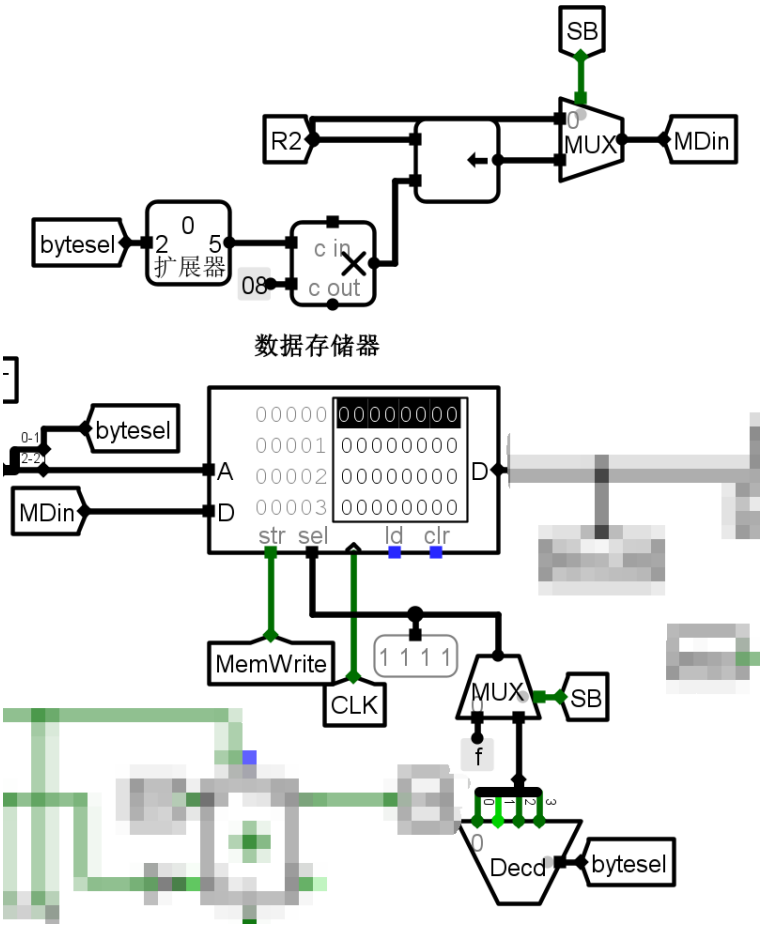

图 3. 23 SB 指令电路支持

上图为支持 SB 指令的专用的电路组件。其中 bytesel 为 ALU 输出的低两位，用于确定字节的地址偏移量。对 rs2 进行移位处理是为了将低字节移动到指定位置，以便存储区写入。

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 risc-v-benchmark_ccan.asm 和 risc-v 单级中断测试程序.asm，risc-v 多级中断测试(EPC 硬件堆栈保护).asm 三个测试文件。

4.1.1 测试用例 1 risc-v-benchmark_ccan.asm

以下为 Benchmark.asm 在单周期、气泡流水线、重定向流水线、动态分支预测电路的运行结果：

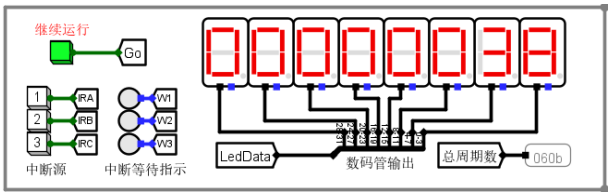

图 4.1 单周期 CPU 测试结果

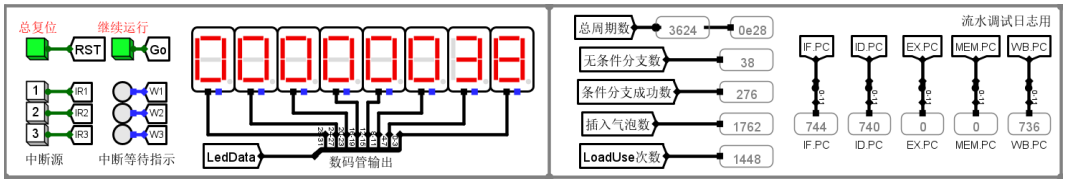

图 4.2 气泡流水线 CPU 测试结果

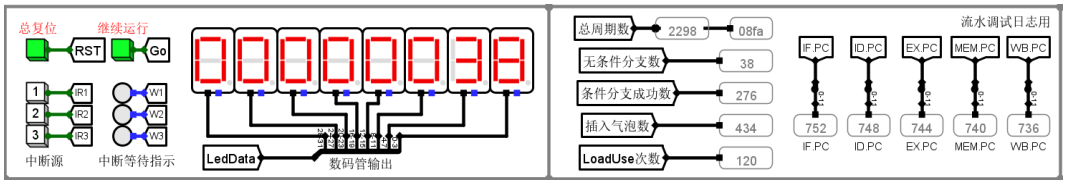

图 4.3 重定向流水线 CPU 测试结果

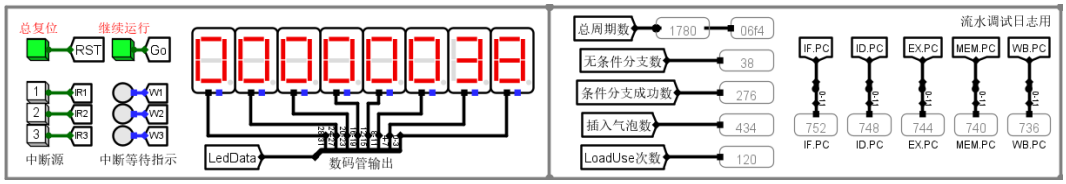

图 4.4 分支预测重定向流水线 CPU 测试结果

以上四个 CPU 在运行测试程序时的输出相同，符合测试要求。后续的 CCAB 测试集的运行符合预期，已通过线下测试，在这里不再赘述。

4.1.2 中断测试

中断测试主要依据 CPU 是否正确地响应中断以及中断是否正确返回判断。这需要根据 LED 的显示数据和 PC 的数值来动态地观察判断，在报告中难以体现。本次实验完成的单机中断，多级嵌套中断以及流水中断均通过线下测试，在这里不在赘述。

4.2 性能分析

从上述测试可以看出，从气泡流水线，重定向流水线到动态分支预测，CPU 的流水性能不断提升，完成相同测试集使用的总周期数不断减少，不断接近无流水线的单周期 CPU 的 IPC。考虑到使用流水线使得 CPU 的关键路径大大缩短，最大频率提升，可以看出采用流水线设计的 CPU 的性能得到明显提升。由于 Logisim 无法获得电路延迟等信息，本次实验无法对每个 CPU 的性能做出准确的定量信息。

4.3 主要故障与调试

4.3.1 BGE 指令停机故障

CCAB 扩展指令： BGE 指令支持异常。

故障现象：运行 BGE.asm 测试程序时，无法停机。

原因分析：没有正确理解 ALU 的功能。由于 Beq 和 Bne 指令使用的等于信号使用 ALU 的默认输出即可，无填写 ALUOP 表项即可实现比较功能，就误以为 ALU 的大于等于信号在默认情况下输出有符号比较结果。实际上比较信号输出在默认情况下小于信号接地，恒为 0；大于等于信号为小于信号的取反，恒为 1。

解决方案：在控制器表格中 ALUOP 的 BGE 表项填入 11（有符号比较）即可。

4.3.2 Halt 信号故障

单周期 CPU：halt 信号生成逻辑错误，提前半周期触发停机。

故障现象：在单周期 CPU 中，执行完 benchmark 提前在 1545 周期停机。

原因分析：在最初的设计中将 D 触发器设置为锁存模式，ecall 信号作为 D 触发器使能信号，将$a7 和 34 的比较结果直接作为输入，导致周期尚未结束时即可触发停机指令，CPU 提前停机在 1545 周期。

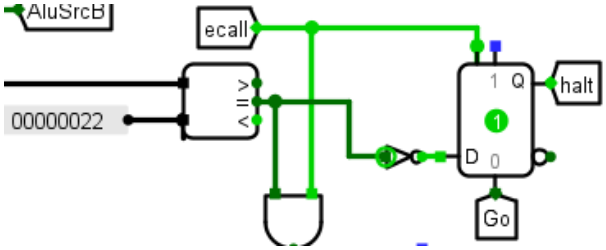

图 4.5 停机信号生成电路（旧）

解决方案：重新设置停机指令的触发，将 D 触发器设置为上升沿触发，并添加时钟信号，使得停机信号将在下一周期开始瞬间触发。修改后重新运行，CPU 正确停机在 1546 周期。

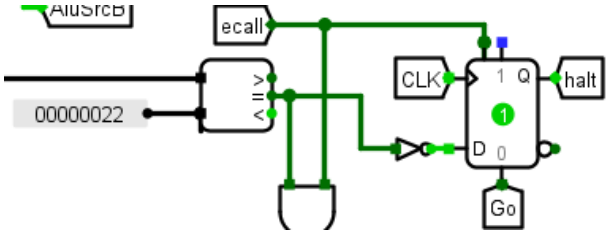

图 4.6 停机信号生成电路（新）

4.3.3 连线故障

故障现象：电路功能不符合预期。

原因分析：在完成本次实验的过程中，最为常见的问题不是电路设计存在逻辑错误，而是连接线路时出现了失误，导致电路并非按照预期连接出现错误。在一些接口比较多的电路中，会有大量的重复性操作，比如流水组件。在进行复制粘贴增加接口的过程中，一不小心就会出现错误。在对 CPU 功能进行升级的过程中，往往会对控制器，流水组件等增加接口，而在重新生成电路后，又常常导致电路封装改变进而导致原有的电路失效。或是选择直接在副本上操作放弃向前兼容，或是手动调整电路封装适应电路变化，都是十分繁琐的操作，许多错误就是在这些繁琐的工作中产生的。

解决方案：只能在连接电路是多加小心，出现错误后认真排查。希望未来能够有更加方便的软件使用。

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。使用 Verilog 实现了部分单周期 CPU 的重要部件，并通过仿真检查。
第四天	继续使用 Verilog 进行实现单周期 CPU 的工作，完成了所有部件的编写、控制器的编写，以及所有部件以及控制器的仿真测试，正在进行数据通路的拼接。
第五天	使用 Verilog 完成单周期 CPU 数据通路的连接，并且通过仿真测试。使用 Verilog 完成时钟分频以及七段数码管的代码编写，正在调试。
第六天	完成 CPU 电路的功能仿真和时序仿真，并成功将生成 bit 流烧入 FPGA 板内实现预计功能。
第七天	复习关于指令流水线的知识点，完成理想流水线的 verilog 代码，正在调试。
第八天	调试成功理想流水线 verilog 代码，并成功将 bit 流烧至 FPGA 板中。完成冒险处理中的数据冲突处理和分支处理代码编写，正在调试。
第九天	完成冒险处理中的数据冲突和分支处理，并成功烧入 FPGA 板内。完成数据重定向的 Verilog 代码的编写，正在调试。
第十天	完成数据重定向的 Verilog 代码并成功烧入 FPGA 板内。成功实现动态分支预测，预测成功率显著提高，并成功将代码烧入 FPGA 板内。

5 团队任务

5.1 选题与设计

本次团队任务中，我们小组使用 logisim 软件，利用之前实验中已经完成的 risc-v 单周期 CPU 和添加的其他硬件电路，实现一个简单的 Flappy Bird 小游戏。在 32×32LED 点阵屏幕上，屏幕第 16 列的一个点代表小鸟，列中有 3 个空格的柱子代表障碍物，小鸟触碰到屏幕下方或柱子则游戏结束。我主要负责地图生成的硬件电路和单周期多级中断 CPU 的修改。

5.2 个人分工

在地图的生成中，我选择了使用硬件电路实现。通过随机数产生器生成一个范围在 0~15 的随机数作为障碍物中心点，通过解码器输出，然后将其扩展到 32 位，左移 10 位令障碍处于屏幕中心，并上下添加两位获得一个 3 格宽的孔洞，取反后输出即可。如图 5.2 所示。

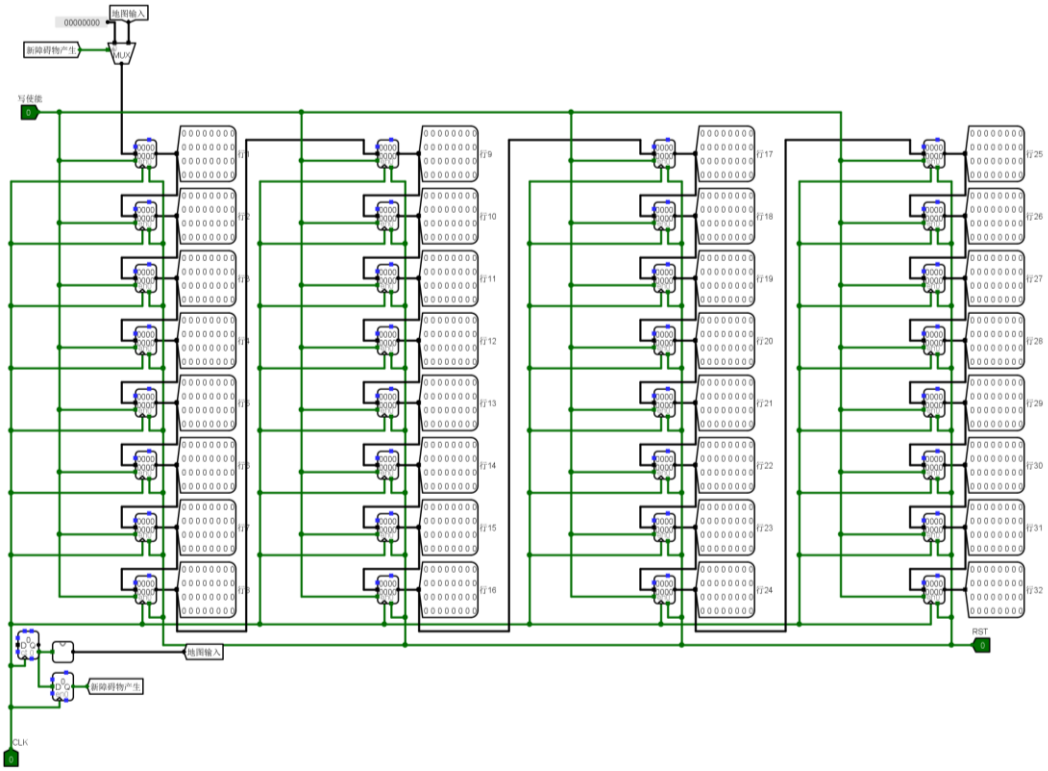

图 5.1 地图流水寄存电路

将时钟输入使用计数器分频控制障碍物生成电路生成的障碍物的间隔，我们选择

使用 4 进制计数器，即每四个时钟周期生成一个障碍物。将障碍物依次向后流水传输，即可获得障碍物地图。将所有输出按列接入 LED 点阵显示屏上，即可获得地图。

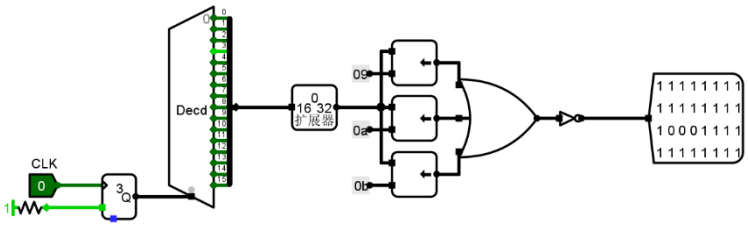

图 5.2 障碍物生成电路

小鸟下落是在按照理想条件做自由落体下落，这是由 1 号中断程序完成的，每 32 个时钟周期执行一次。2 号中断控制小鸟上升，它可以在打断 1 号中断执行。和我们设计的多级中断 CPU 不同的是，这里要求中断不能嵌套执行，1 号中断被 2 号中断打断后将会被终止执行。但是在嵌套的情况下，中断依然按照 1 号的返回地址返回。因此需要对中断机制做出一些修改。

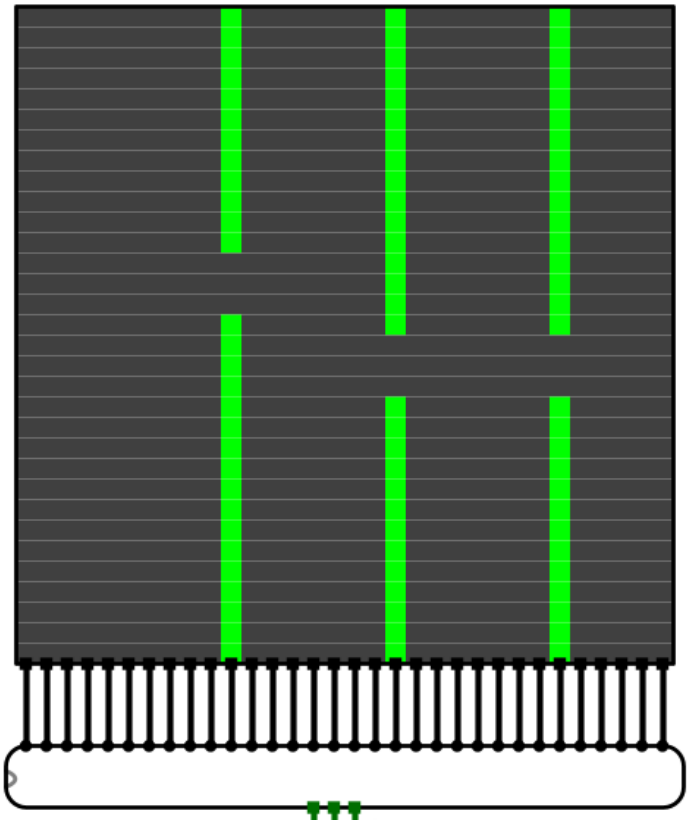

图 5.3 障碍物生成电路示范

首先对硬件堆栈进行修改。由于 2 号中断结束后堆栈一定为空，所以当栈深度为 2 时，也就是中断嵌套执行时，需要减 2 清空堆栈。

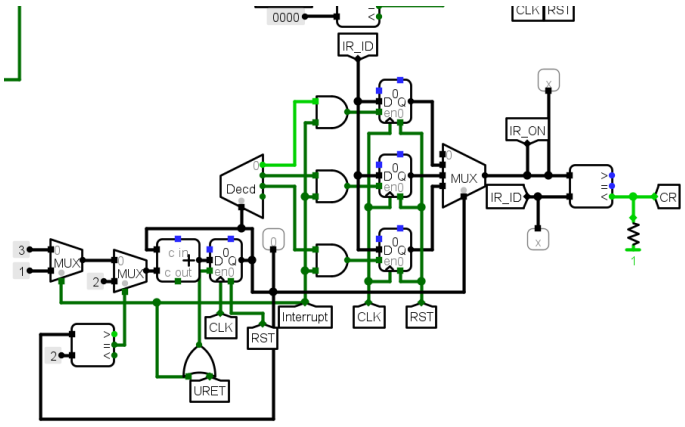

图 5.4 多级中断硬件堆栈修改

为了保证中断总能返回到主函数，需要对 EPC 进行修改。当中断嵌套执行时，2 号中断需要按照 1 号中断的断点返回到主函数继续执行。可根据 EPC 大小判断断点位置。因为中断嵌套时，2 号中断的返回地址非常大，据此可以判断该采用哪个返回地址。

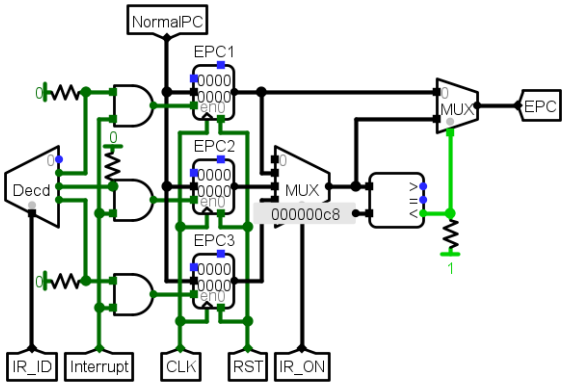

图 5.5 多级中断 EPC 生成修改

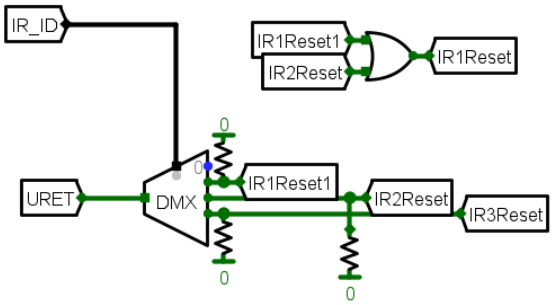

图 5.6 多级中断清中断信号修改

当中断嵌套执行时，2 号中断完成后会同时清空 2 号和 1 号中断。因此需要将 2 号中

断清中断信号作为 1 号中断的清中断信号的充分条件之一。

5.3 总体展示

除了我完成的部分之外，我们的团队任务还有小鸟移动的控制程序，BGM 模块，积分模块以及结算画面等任务。经过小组成员的共同协作，我们合力完成各个模块的内容并完成了组装。

以下是我们的团队任务的演示。第一张图片演示了游戏进行中的状态，第二张图片展示了游戏结束的画面。

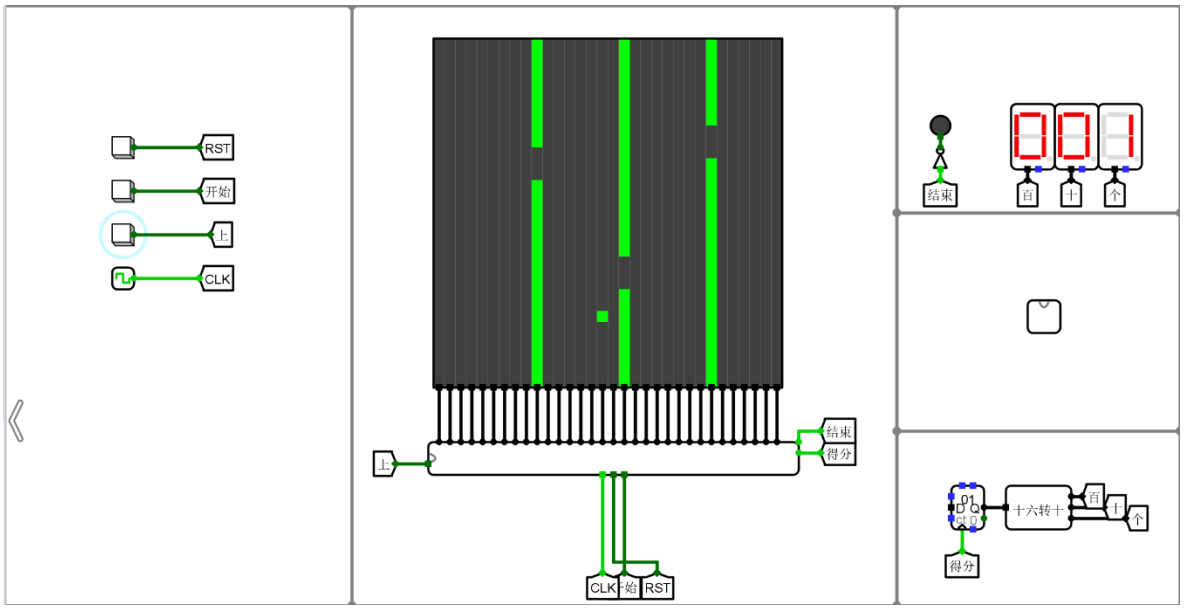

图 5.7 游戏进行演示

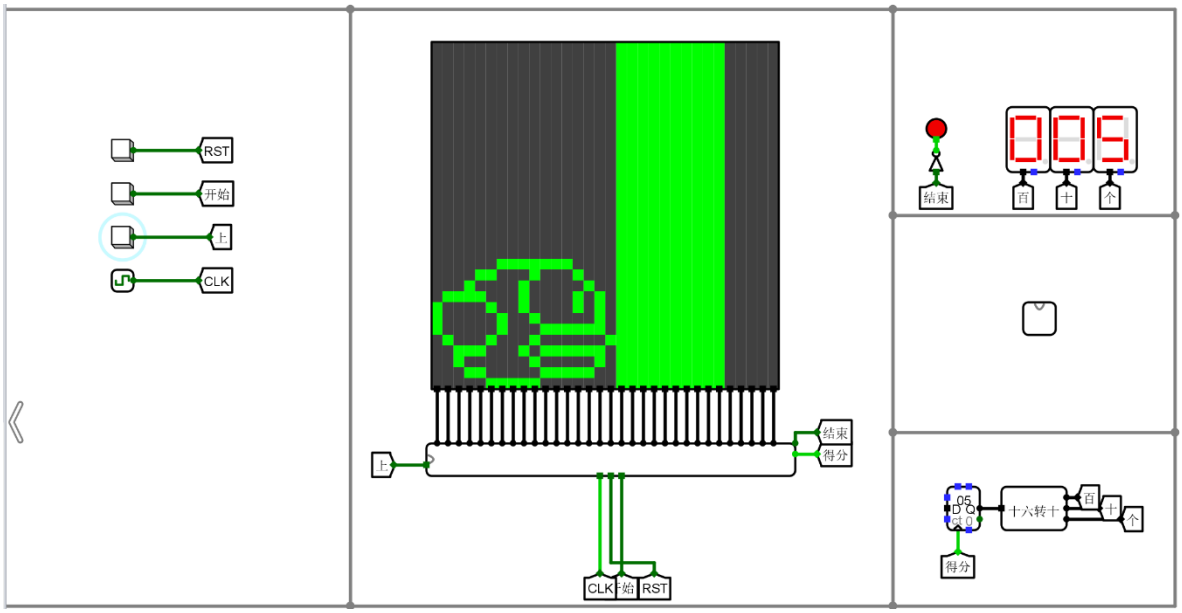

图 5.8 游戏结束演示

6 设计总结与心得

6.1 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1） 基础任务：设计了单周期 CPU、理想流水线 CPU、气泡流水线 CPU、重定向流水线 CPU、支持单级中断和多级中断的单周期 CPU，支持单级中断的流水线 CPU，以及动态分支预测的重定向流水线 CPU。

2） 功能总结：所有的 CPU 除理想流水线外，均支持 24 条基本指令和 4 条扩展 CCAB 指令。重定向流水线 CPU 减少了普通流水线 CPU 的气泡插入，加速了 CPU 的执行速度。动态分支预测技术利用 BHT 表进一步加速了重定向流水线 CPU，减少了分支误取的代价。对于多级中断，支持 3 条不同等级中断源的嵌套中断。

3） 团队任务：完成地图电路生成设计，使用简洁的硬件电路完成了功能。完成多级中断 CPU 的修改，使其支持高优先级中断的抢占执行和直接返回至首个断点的功能。

6.2 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力以及国庆节假期的辛苦加班才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课程设计刚刚开始的时候，遇到的最大的困难是不知道怎么逐步构建一个完整的，支持多条指令的 CPU 数据通路。在以往的实验中，我基本都是面对的被拆分好的小任务，然后跟随实验知道逐步各个模块并组装。然而在这次课设中，我一开始就需要设计完成一个具备完整功能的 CPU，乍一看让人难以上手。在仔细阅读任务书和其他相关资料后，我比葫芦画瓢逐步完成了 CPU 的搭建。但是一开始的由于对数据通路的认识不够清楚，出现了很多逻辑错误。经过不断的调试和改进，我逐渐研究清楚了 CPU 的结构和功能，为之后的实验进行打下了基础。

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。随后的动态分支预测的设计难度更高，老师并没有提供任何相关的内容，这又使得我不断地去网上搜寻资料文献，阅读全英文的学术论文。从这一切也可以看出团队在一个工作中的重要性，以及自我学习能力的必要性。

中断部分的实验也让我吃了一点苦头。中断如何嵌套保存中断号，如何让中断隐指令无延迟执行，这些问题都困扰了我很久。在查阅资料，学习学长的作品参考后，才完成了硬件堆栈多级中断的实现。后来的团队任务中，我又遇到了中断隐指令的延迟问题。在单级中断进行多次修改都没有消除一个时钟周期的延迟。最终选择了使用多级中断 CPU 进行改进完成。

然而对于本次课程设计，我还有一些小小的建议和改进。理想流水线 CPU 的实现相对简单，且功能上完全被气泡流水线和重定向流水线替代，可以直接提供示例而不要实现。可以直接提供功能兼容性好，电路重复度高组件成品，同学们只需要完成逻辑电路的链接即可。流水组件的连接重复度极高，当电路发生变动时，封装常常会发生大幅度改变，且由于电路规模较大，难以调整封装对应的接口。如果想要获取一个干净整洁电路，就需要花费大量的时间调整封装，且这样的工作会随着气泡流水线，重定向流水线，分支预测流水线的电路调整多次重复，机械地浪费大量时间。还有控制器电路也存在相同的问题。对此，我选择放弃兼容性，为每个 CPU 单开一个副本保存。**如果课设组能够提供具备完整接口的控制器和流水组件电路，同学们只需要完成不同流水线的控制逻辑电路和 CPU 控制器的表达式生成表的填写，无需再关心电路封装的调整。这将会大大降低同学们的负担，免去不必要的机械劳动。**

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。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

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] 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，周军龙，肖亮. 计算机组成原理实验指导与习题解析.北京：人民邮电出版社，2022.

[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：金钊