華中科技大學

2022

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：	CS2003
学　　号：	U202015362
姓　　名：	计胜翔
电　　话：	13296538161
邮　　件：	2561332247@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	SLL	逻辑左移	
29	SLTIU	小于无符号立即数置数	
30	LBU	无符号字节加载	
31	BLTU	小于无符号数时跳转	

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硬布线控制，且指令存储器和数据存储器相分离的单周期处理器结构（哈佛结构）。同时，所有指令均在一个时钟周期内完成。

此外，我们采用简单迭代法实现单周期处理器。这种方法相对比较直观简单，我们先完成一条固定指令的基本数据通路，测试运行通过后再在这个基础上不断增加新的数据通路，支持新的指令，直至基本 24 条指令和 4 条扩展指令（中断相关指令在后续阶段完成）都能正常运行。当然，我们首先应该完成取指令和执行指令的逻辑。

总体结构图如图 2.1 所示。

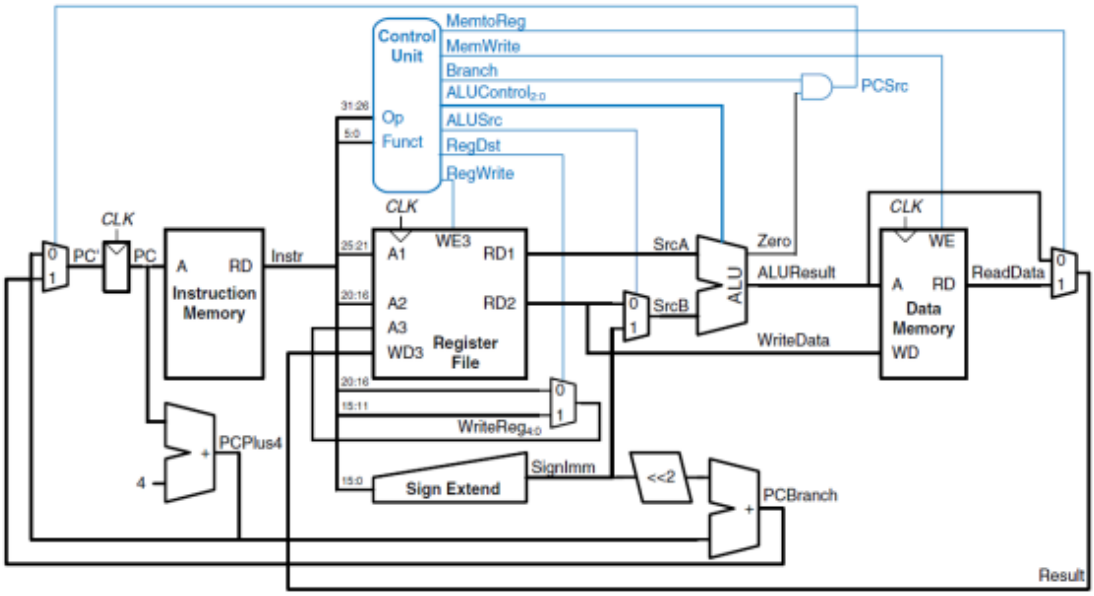

图 2.1 总体结构图

2.1.1 主要功能部件

运算器部分，具体设计思路如下：运算器由算数逻辑单元和各种寄存器组成，接受控制器的命令后执行算术运算、逻辑运算或逻辑测试，负责数据加工。

1. 程序计数器 PC

程序计数器 PC 保存将要执行的指令的字节地址，采用寄存器实现，位宽为 32 位。

CPU 取指令时，用 PC 的内容作为地址访问指令存储器 IM，然后修改 PC 的值形成下一条指令的地址。当程序顺序执行时，PC 加 4 即可；当出现分支跳转时，用分支指令提供的分支地址修改 PC 的值。

2. 指令存储器 IM

指令存储器 IM 用于保存程序的所有指令，本次采用只读存储器 ROM 实现。其输入输出引脚与功能描述见表 2.1。

表 2.1 指令存储器 IM 引脚与功能描述

引脚	输入/输出	位宽	功能描述
A	输入	10	指令地址
D	输出	32	指令

3. 数据存储器 DM

数据存储器 DM 用于存放数据，即可读出也可写入，采用 RAM 实现。其输入输出引脚与功能描述见表 2.2。

表 2.2 数据存储器 DM 引脚与功能描述

引脚	输入/输出	位宽	功能描述
CLK	输入	1	时钟信号
Addr	输入	10	需要访问数据的地址
MDin	输入	32	即将写入的数据值
MemWrite	输入	4	写使能信号
Dout	输出	32	对应地址单元的数据

4. 运算器

运算器接受两个 32 位的操作数，根据操作码决定对两个数进行何种运算，并输出结果，其输入输出引脚与功能描述见表 2.3。

表 2.3 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，具体功能见表 2.4
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

运算器的操作码 ALU_OP 各值对应的运算见表 2.4：

表 2.4 运算器操作码对应功能描述

ALU_OP	十进制	运算功能
0000	0	逻辑左移。 Result = X << Y（Y 取低五位），Result2=0
0001	1	算术右移 Result = X >>>Y（Y 取低五位），Result2=0
0010	2	逻辑右移 Result = X >> Y（Y 取低五位），Result2=0
0011	3	无符号乘法 Result = (X * Y)[31:0]； Result2 = (X * Y)[63:32]
0100	4	无符号除法。Result = X/Y；Result2 = X%Y
0101	5	加法。Result = X + Y
0110	6	减法。Result = X - Y
0111	7	按位与。Result = X & Y
1000	8	按位或。Result = X | Y
1001	9	按位异或。Result = X⊕Y
1010	10	按位或非。Result = ~(X |Y)
1011	11	符号比较。Result = (X < Y) ? 1 : 0

ALU_OP	十进制	运算功能
1100	12	无符号比较。Result = (X < Y) ? 1 : 0

5. 寄存器堆 RF

寄存器堆 RF 是指运算器内部的若干寄存器，包含 RISC-V 体系结构中的 32 个 32 位的通用寄存器。其输入输出引脚描述见表 2.5：

表 2.5 寄存器堆 RF 引脚与功能描述

引脚	输入/输出	位宽	功能描述
CLK	输入	1	时钟信号
R1#	输入	5	寄存器 R1#，来自于指令字中的 rs1 字段
R2#	输入	5	寄存器 R2#，来自于指令字中的 rs2 字段
W#	输入	5	目的寄存器
RDin	输入	32	写入目的寄存器的数据值
RegWrite	输入	1	写使能信号
R1	输出	32	寄存器 R1#的值
R2	输出	32	寄存器 R2#的值

2.1.2 数据通路的设计

RISC-V 指令主要分为 R 型、I 型、S 型、B 型、U 型、J 型，每个类型的指令的数据通路有所不同，具体指令的数据通路也会有细小差别，具体设计和实现见 3.1.2。

2.1.3 控制器的设计

控制器采用硬布线的方式，其输出的控制信号利用表格自动生成的组合逻辑表达式实现。

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 2.6。

表 2.6 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_OP	0000-1100	各种取值对应的功能见表 2.4
MemtoReg	0	寄存器写入数据不来自数据存储器
	1	寄存器写入数据来自数据存储器
MemWrite	0	不允许写数据存储器
	1	数据存储器允许写入
RegWrite	0	不允许写寄存器
	1	寄存器允许写入
S_Type	0	当前指令是 S 型指令
	1	当前指令不是 S 型指令
AluSrcB	0	运算器 B 输入来自寄存器
	1	运算器 B 输入来自指令中的立即数
JALR	0	当前指令不是 jalr
	1	当前指令是 jalr
JAL	0	当前指令不是 jal
	1	当前指令是 jal
Beq	0	当前指令不是 beq
	1	当前指令是 beq
Bne	0	当前指令不是 bne
	1	当前指令是 bne
ecall	0	当前指令不是 ecall
	1	当前指令是 ecall
lbu	0	当前指令不是 lbu
	1	当前指令是 lbu
bltu	0	当前指令不是 bltu
	1	当前指令是 bltu

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 2.7 所示。

表 2.7 主控制器控制信号框架

指令	ALU_OP	MemtoRe	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr	lbu	bltu
add	5				1								
sub	6				1								
and	7				1								
or	8				1								
slt	11				1								
sltu	12				1								
addi	5			1	1								
andi	7			1	1								
ori	8			1	1								
xori	9			1	1								
slti	11			1	1								
slli	0			1	1								
srli	2			1	1								
srai	1			1	1								
lw	5	1		1	1								
sw	5		1	1			1						
ecall						1							
beq								1					
bne									1				
jal					1					1			
jalr	5			1	1						1		
sll	0				1								
sltiu	12			1	1								
lbu	5	1		1	1							1	
bltu	12												1

2.2 流水 CPU 设计

2.2.1 总体设计

将指令过程分成 5 个阶段：取指令（IF）段、译码取数（ID）段、指令执行（EX）段、访存（MEM）段、写回（WB）段。其中 IF 段包括程序计数器 pc、指令存储器以及计算下条指令的地址逻辑；ID 段包括操作控制器、取操作数逻辑、立即数符号扩展模块；EX 段主要包括算术逻辑运算单元 ALU、分支地址计算模块；MEM 段主要包括数据存储器读写模块；WB 段主要包括寄存器写入控制模块。不同阶段之间设置缓冲接口部件。

总体结构图如图 2.2 所示。

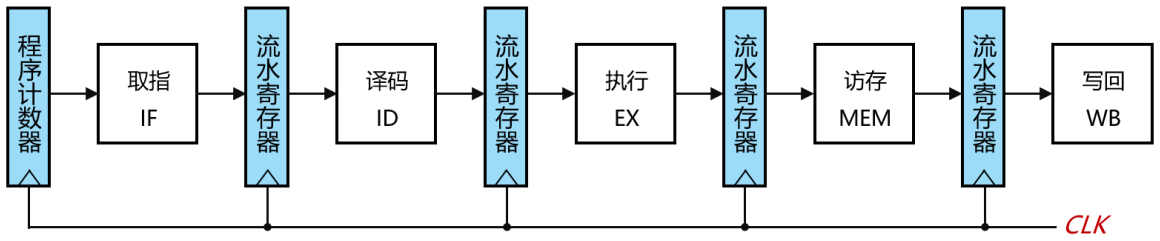

图 2.2 流水 CPU 总体结构图

2.2.2 流水接口部件设计

流水接口部件用于锁存本阶段处理完成的所有数据和结果，以保证本段的执行结果能在下一个时钟周期给下一个阶段使用，本课程设计中采用流水寄存器实现。流水寄存器在同一时钟的驱动下锁存流水线前段加工完成的数据以及控制信号，所以每个流水寄存器的输入输出引脚有所不同。此外，流水寄存器的使能端应接停机信号。

2.2.3 理想流水线设计

本阶段不考虑各种结构冲突、数据冲突和分支冲突，故称“理想流水”。我们需要增加 4 个流水寄存器，分别命名为 IF/ID、ID/EX、EX/MEM、MEM/WB，并且分支指令在 EX 段完成。

不同的流水寄存器锁存传递的数据信息并不相同。IF/ID 流水寄存器需要锁存从指令存储器取出的指令字以及 PC+4 的值；ID/EX 流水寄存器需要锁存从寄存器堆中取出的两个操作数 R[rs1]和 R[rs2]、立即数符号扩展的值、PC+4 等后段可能用到的操作数和控制信号；EX/MEM 流水寄存器需要锁存 ALU 运算结果、数据存储器待写入数据、写寄存器编号 WriteReg#等数据；MEM/WB 流水寄存器需要锁存 ALU 运算结果、数据存储器读出数据、写寄存器编号 WriteReg#等数据。

2.3 气泡式流水线设计

2.3.1 总体设计

在理想流水线的基础上增加对流水线冲突的处理。

对于结构冲突，在单周期 CPU 的设计中就存在，但由于我们采用的是哈佛结构，所以已解决此冲突。

对于控制冲突，解决方法是在执行程序分支跳转时必须清除流水线中分支指令后续的若干条误取指令，我们可以将分支跳转信号 BranchTaken 作为流水线清空信号 Flush 直接连接 IF/ID、ID/EX 流水寄存器的同步清零控制端。

对于数据冲突，通过插入气泡解决。当发生数据相关时，只需暂停 IF、ID 段指令的执行，同时插入气泡阻塞暂停流水线，从而消除数据相关性。

2.3.2 数据冲突处理设计

首先在流水线中增加硬件逻辑实现 ID 段与 EX、MEM 段指令的数据相关性检测。

然后对数据冲突进行处理，当发生数据相关时给 ID/EX 流水寄存器一个同步清零信号 Flush 即可，相当于插入气泡。而要暂停 IF、ID 段指令的执行，只需控制流水寄存器使能端即可，置 0 忽略时钟输入，使寄存器值保持不变。

2.4 重定向流水线设计

2.4.1 总体设计

在理想流水线的基础上增加对流水线冲突的处理。

对于结构冲突和控制冲突，解决方法和气泡式流水线相同，见 2.3.1。

对于数据冲突，使用重定向的方式解决。先不考虑 ID 段所取的寄存器操作数是否正确，等到指令实际使用这些寄存器操作数时（即 EX 段）再将正确的操作数从其所在位置重定向到 EX 段的寄存器操作数 R[rs1]和 R[rs2]。

2.4.2 数据冲突处理设计

除 Load-Use 相关外，其他数据相关都可以采用重定向方式，只需在 ID 段生成两个重定向选择信号并向后面的流水寄存器传输即可，作为 ID/EX.R1 和 ID/EX.R2 输出端重定向时的多路选择器的选择信号。而多路选择器的输入来源包括 ID/EX.R1、EX/MEM.AluResult 的重定向、WB 段 RDin 的重定向。

检测出发生 Load-Use 相关时，暂停 IF、ID 段指令的执行，并在 EX 段中插入气泡即可。

2.5 单级中断机制设计

2.5.1 总体设计

在单周期 CPU 的基础上增加单级中断处理机制，支持 3 个优先级不同的外部按键中断源。每个中断对应一个中断号。根据中断号确定执行哪段中断程序，用类似于中断向量表的方式存放中断程序入口地址。可以事先获取中断程序入口地址，以常量形式作为多路选择器的输入来源。

中断响应周期内要实现硬件关中断、将主程序断点保存至 mEPC 寄存器、将中断识别逻辑产生的中断服务程序入口地址送 PC。中断返回 uret 时，要实现将 mEPC 寄存器送 PC、开中断、发送中断结束信号熄灭当前中断请求的指示灯。因此需要增加与中断相关的寄存器，增加中断识别逻辑，增加一些数据通路。

2.5.2 硬件设计

1、中断使能信号 IE：

采用 D 触发器实现。用于开关中断，1 表示开中断，0 表示关中断，开关中断采用同步置位和复位方式。

2、异常程序计数器 mEPC：

用 32 位寄存器实现，上升沿触发，当有中断请求时，使能端置 1，写入当前 PC+4 的值作为中断返回地址。

3、中断识别：

采用优先编码器对不同中断请求信号进行编码，优先响应优先级高的。然后通过多路选择器选择相应的中断程序入口地址。

2.6 多级中断机制设计

2.6.1 总体设计

在单级中断的基础上修改实现多级中断，高优先级中断应该正确中断低优先级中断服务子程序。其中，mEPC 寄存器采用硬件堆栈的方式保护断点地址，以支持嵌套中断。我们需要比较新中断和当前中断程序的优先级，判断是否存在更高优先级的中断。

此外，要增加 csrrsi、csrrci、csrrw 指令数据通路，前 2 条指令分别对应开关中断指令即可。

2.6.2 硬件设计

1、硬件堆栈：

为 3 个中断号分别设置 3 个 mEPC 寄存器，通过多路选择器根据当前运行中断号选择哪个寄存器的值是当前中断程序的返回地址。

2、中断使能信号 IE：

硬件设计与单级中断基本相同，只需增加对于 csrrsi、csrrci 指令的支持。对于 csrrsi，IE 置 1，开中断；对于 csrrci，IE 置 0，关中断。

2.7 基于重定向流水的单级中断机制设计

结合重定向流水和单周期 CPU 单级中断实现。

2.7.1 中断陷入

有中断请求时，在时钟上升沿，中断请求信号 INT 会置为 1。将 INT 信号看作分支跳转信号，在 PC 输入端选择中断入口地址作为输入源，同时接入 ID/EX、EX/MEM 流水寄存器的同步清零端，在中断到来时插入 2 个气泡。

对于断点地址，正常情况下应保存 EX.PC，但若 ID/EX 流水寄存器为气泡，存在 3 种情况：前一个周期存在 Load_Use 相关、 前一个周期存在分支相关、前二个周期存在分支相关。需要根据不同情况选择正确的断点地址保存。

2.7.2 中断返回

中断返回指令为 uret，此时，只需在 PC 输入端选择 mEPC 中保存的值作为输入源即可。

2.8 动态分支预测机制

2.8.1 总体设计

在重定向流水线的基础上实现动态分支预测机制，根据跳转指令的历史跳转信息预测下一次是否要跳转，并不断地对预测策略进行动态调整。

本阶段中，动态分支策略采用分支预测缓冲器 BTB 表实现，且使用双位预测，预测位初始值设为 01。

2.8.2 硬件设计

BTB 采用八路全相连映射 CACHE 实现，cache 的替换策略采用 LRU 算法。

在 IF 段利用 PC 值在 BTB 中进行全相联比较，若命中则根据 BTB 表中的状态位预测是否跳转。

在 EX 段比较真实跳转信息和预测信息，判断是否预测成功，并准备更新 BTB 中对应的状态位信息。如果预测失败需要在流水插入气泡并在 IF 段开始执行另一条路径；如果当前跳转指令缺失，则在 BTB 表中插入新的跳转指令的信息。

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）

在 Logisim 中使用一个 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。Halt 为停机信号，取反后接入寄存器使能端，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，PC 忽略时钟信号，使整个电路停机。如图 3.1 所示。

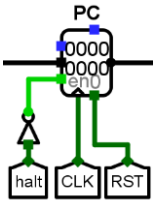

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

在 Logisim 中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。

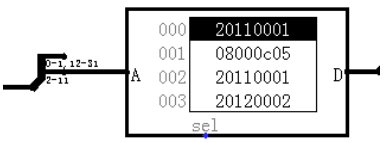

图 3.2 指令存储器（IM）

3) 数据存储器（DM）

使用 32 位的随机存储器 RAM 实现。与指令存储器 IM 相同，从 32 位地址中截取 2-11 位作为 DM 的输入地址。此外，还有控制信号 MDin、MemWrite 接入 DM 的相应端口。如图 3.3 所示。

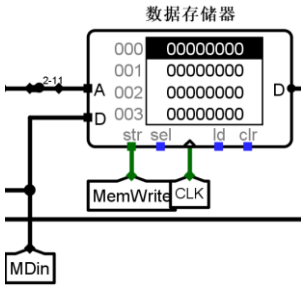

图 3.3 数据存储器（DM）

4) 运算器

运算器采用已经封装好的 ALU 运算单元实现。寄存器操作数 R[rs2]和立即数操作数经过一个 2 路选择器，根据 AluSrcB 信号选择输出后，再接入 B 操作数的输入端口。AluOp 作为运算器操作码输入。输出有 equal、<信号和运算结果。如图 3.4 所示。

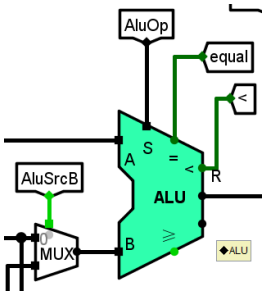

图 3.4 运算器（ALU）

5) 寄存器堆（RF）

使用 CS3410 库中提供的寄存器文件实现。寄存器写使能信号 RegWrite 接入 WE 端口，寄存器写入数据 RDin 接入 Din 端口。如图 3.5 所示。

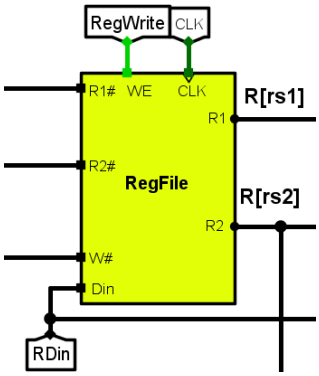

图 3.5 寄存器堆（RF）

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那一小节的详细内容，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
add	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	5		
sub	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	6		
and	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	7		
or	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	8		
slt	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	11		
sltu	PC+4	PC	rs1	rs2	rd	ALU	r1	r2	12		
addi	PC+4	PC	rs1		rd	ALU	r1	imm	5		
andi	PC+4	PC	rs1		rd	ALU	r1	imm	7		
ori	PC+4	PC	rs1		rd	ALU	r1	imm	8		
xori	PC+4	PC	rs1		rd	ALU	r1	imm	9		
slti	PC+4	PC	rs1		rd	ALU	r1	imm	11		
slli	PC+4	PC	rs1		rd	ALU	r1	imm	0		
srli	PC+4	PC	rs1		rd	ALU	r1	imm	2		
srai	PC+4	PC	rs1		rd	ALU	r1	imm	1		
lw	PC+4	PC	rs1		rd	MDout	r1	imm	5	ALU	
sw	PC+4	PC	rs1	rs2			r1	imm	5	ALU	R[rs2]
ecall	PC+4	PC	rs1	rs2							
beq	PC+4/PC+imm<<1	PC	rs1	rs2			r1	r2			
bne	PC+4/PC+imm<<1	PC	rs1	rs2			r1	r2			
jal	PC+立即数	PC			rd	PC+4	r1	imm			

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
jalr	R[rs1]+imm	PC			rd	PC+4	rs1	imm	5		
sll	PC+4	PC	rs1	rs2	rd	ALU	rs1	r2	0		
sltiu	PC+4	PC	rs1		rd	ALU	rs1	imm	12		
lbu	PC+4	PC	rs1		rd	MDout	rs1	imm	5	ALU	
bltu	PC+4/PC+imm<<1	PC	rs1	rs2			rs1	r2	12		

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，如图 3.6 所示。

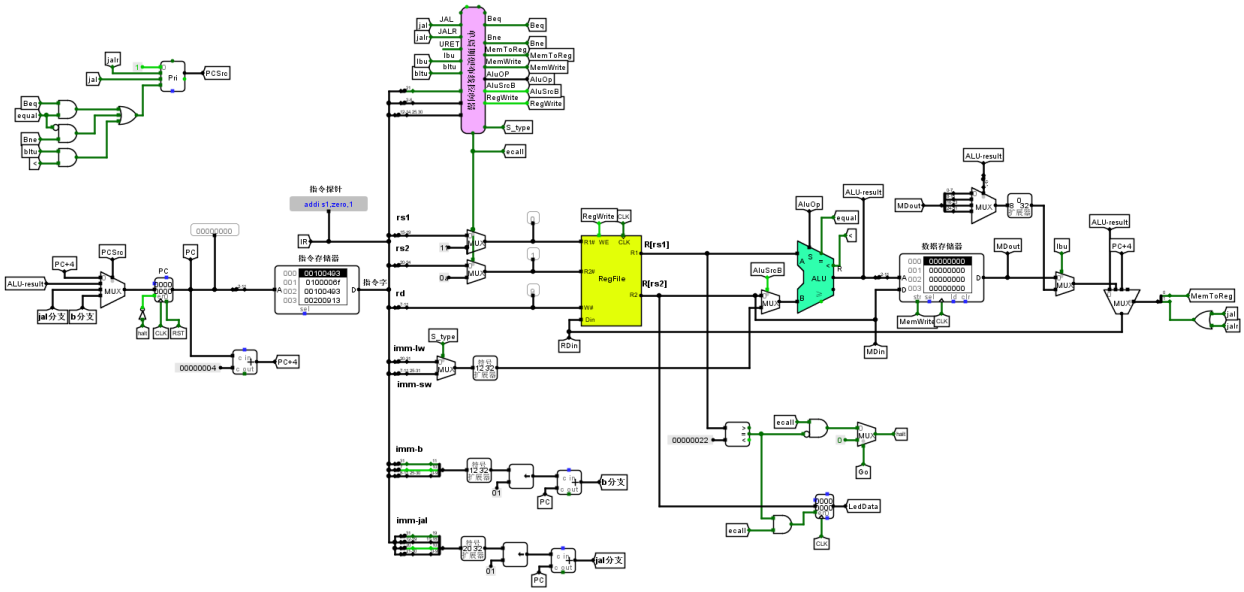

图 3.6 单周期 CPU 数据通路（Logisim）

3.1.3 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，在 RISC-V 单周期硬布线控制器表达式自动生成的 excel 表中填写每条指令对应的各种控制信号，最终如下所示。

#	指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr	lbu	bltu	Rs1Used	Rs2Used	uret	csrrsi	csrrci
1	add	0	0	c	5				1									1	1			
2	sub	32	0	c	6				1									1	1			
3	and	0	7	c	7				1									1	1			
4	or	0	6	c	8				1									1	1			
5	slt	0	2	c	11				1									1	1			
6	sltu	0	3	c	12				1									1	1			
7	addi		0	4	5			1	1									1				
8	andi		7	4	7			1	1									1				
9	ori		6	4	8			1	1									1				
10	xori		4	4	9			1	1									1				
11	slti		2	4	11			1	1									1				
12	slli	0	1	4	0			1	1									1				
13	srli	0	5	4	2			1	1									1				
14	srai	32	5	4	1			1	1									1				
15	lw		2	0	5	1		1	1									1				
16	sw		2	8	5		1	1			1							1	1			
17	ecall	0	0	1c						1												
18	beq		0	18								1						1	1			
19	bne		1	18									1					1	1			
20	jal			1b					1					1								
21	jalr		0	19	5			1	1						1			1				
22	CSRRSI		6	1c																	1	
23	CSRRCI		7	1c																		1
24	URET	2	0	1c																1		
25	sll	0	1	c	0				1									1	1			
26	sltiu		3	4	12			1	1									1				
27	lbu		4	0	5	1		1	1							1		1				
28	bltu		6	18	12												1	1	1			

图 3.7 指令控制信号表填写情况

根据指令 op 和 func 字段生成控制信号的电路如图 3.8 所示。

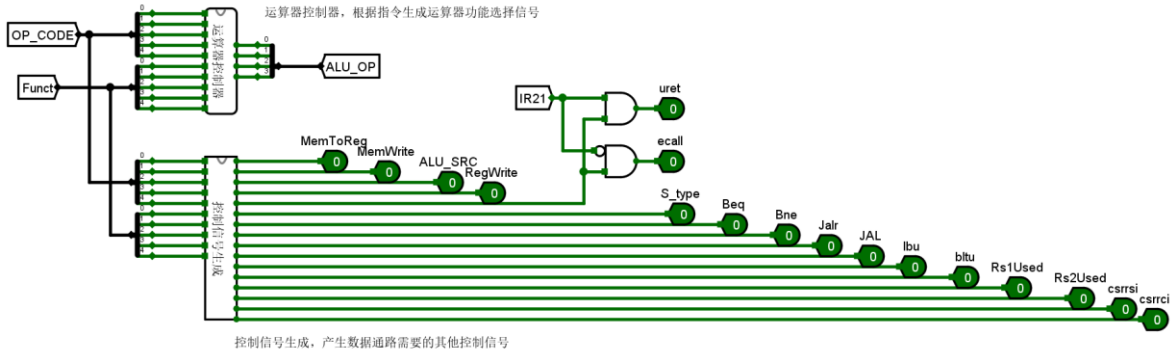

图 3.8 根据指令 op 和 func 字段生成控制信号

其中，2 个封装电路内部是组合逻辑电路，组合逻辑表达式由图 3.7 中表格自动生成的表达式得到，再利用 Logisim 的根据表达式自动生成组合逻辑电路的功能得到这 2 个封装电路。最终控制器的电路如图 3.9 所示。

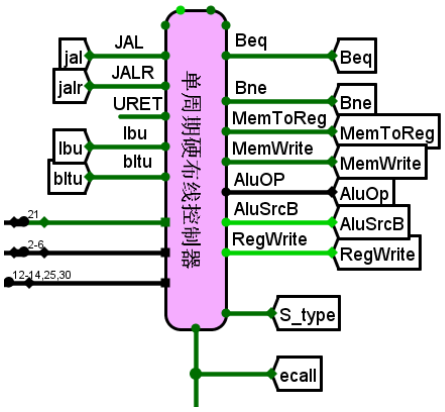

图 3.9 硬布线控制器实现

3.2 流水 CPU 实现

3.2.1 流水接口部件实现

因为每个流水寄存器结构相同，只是内部寄存器个数和传递的信号数据不同，所以下面以 IF/ID 流水寄存器为例。

控制信号包括同步清零信号 Flush、时钟信号 CLK、清零信号 RST 和停机信号 halt。输入为 IF 段的结果，输出为对应寄存器的值。时钟上升沿触发，将左侧数据锁存进寄存器中，供下一阶段 ID 段使用。流水寄存器内部电路如图 3.10 所示。

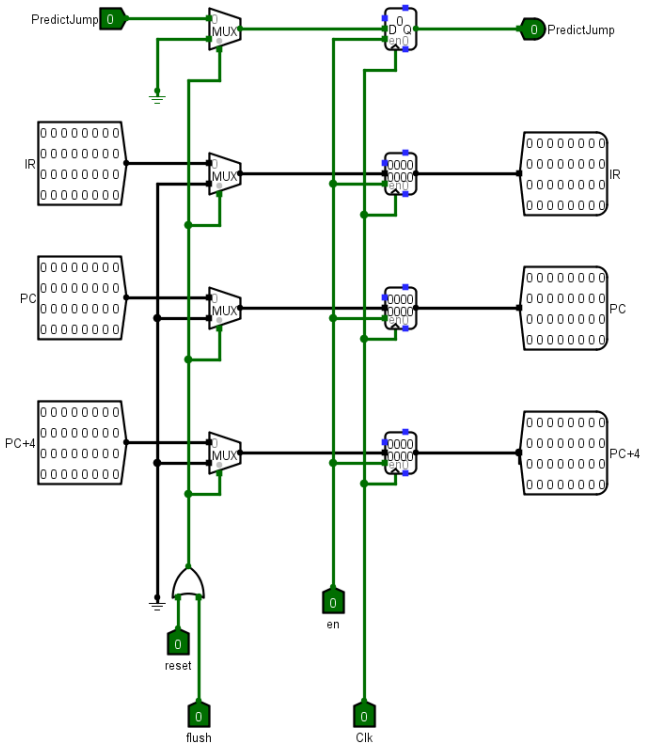

图 3.10 IF/ID 流水寄存器内部电路

每个寄存器位宽由锁存的数据位宽决定，但由统一的时钟信号控制。每个数据会与接地信号经过同步清零信号选择，多路选择器的输出再作为寄存器的输入，从而实现同步清零。而不是直接将同步清零信号接入寄存器的置 0 端。

3.2.2 理想流水线实现

理想流水线实现的总体电路如图 3.11 所示。

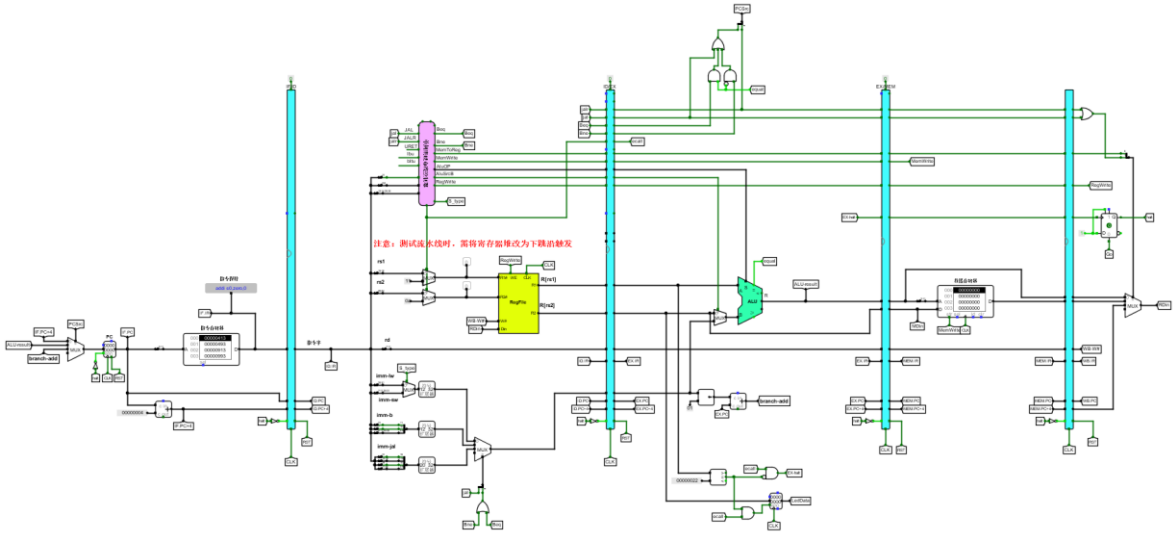

图 3.11 理想流水线实现

其中，分支跳转在 EX 段处理，生成控制信号 PCSrc，对 IF 段 PC 的输入进行选择。4 个流水寄存器使用统一时钟信号、停机信号和清零信号。

3.3 气泡式流水线实现

3.3.1 插入气泡实现

在理想流水线的基础上增加气泡处理逻辑即可，封装电路如图 3.12 所示。

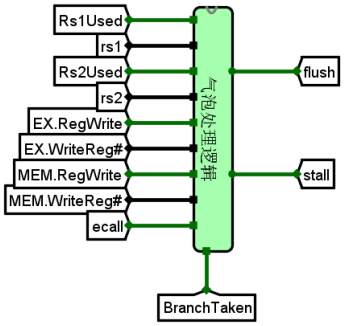

图 3.12 气泡处理逻辑封装电路

该部分电路生成同步清零信号 flush 和流水线阻塞暂停信号 stall。flush 信号接入 ID/EX 流水寄存器的同步清零端，用于插入气泡；stall 信号取反后接入 IF/ID 流水寄存器的使能端，当发生数据相关时阻塞暂停 IF、ID 段指令的执行。

3.3.2 数据相关检测实现

要检测数据相关性，检查 EX、MEM 段的寄存器堆写入控制信号 RegWrite 是否

为 1，且写寄存器编号 WriteReg#是否和源寄存器编号相同即可。实现电路如图 3.13 所示：

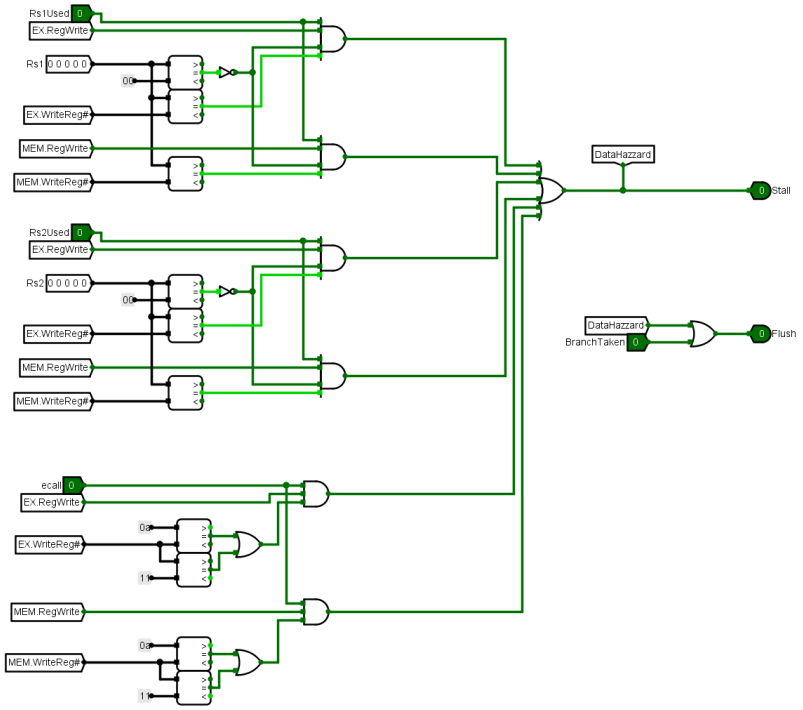

图 3.13 数据相关检测实现

3.4 重定向流水线实现

3.4.1 数据相关检测和处理实现

1、Load-Use 相关

若 ID 段、EX 段的相邻指令存在数据相关，且 EX 段指令为访存指令，即 EX.MemToReg 信号为 1，则存在 Load-Use 相关。实现电路如图 3.14 所示。

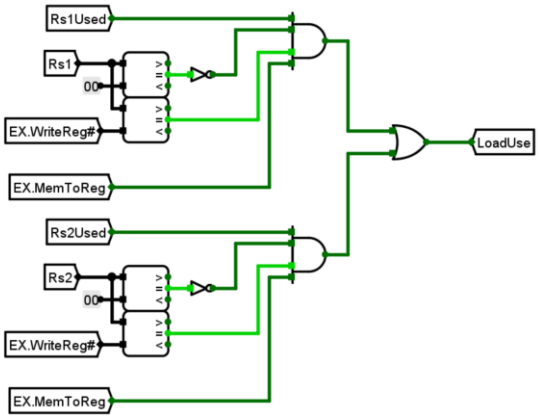

图 3.14 Load-Use 相关检测电路

由于 Load-Use 相关仍然要通过插入气泡解决，所以输出的 LoadUse 信号也是插入气泡的信号，即 stall 信号。

2、其他数据相关

其他数据相关的检测逻辑和气泡式流水线基本相同，但是处理逻辑不同。要在 ID 段生成两个重定向选择信号 Rs1Forward、Rs2Forward 并传输给 ID/EX 流水寄存器。以 Rs1Forward 为例，当 ID 段与 EX 段数据相关时，Rs1Forward=2；当 ID 段与 MEM 段数据相关时，Rs1Forward=1；当无数据相关时，Rs1Forward=0。实现电路如图 3.15 所示。

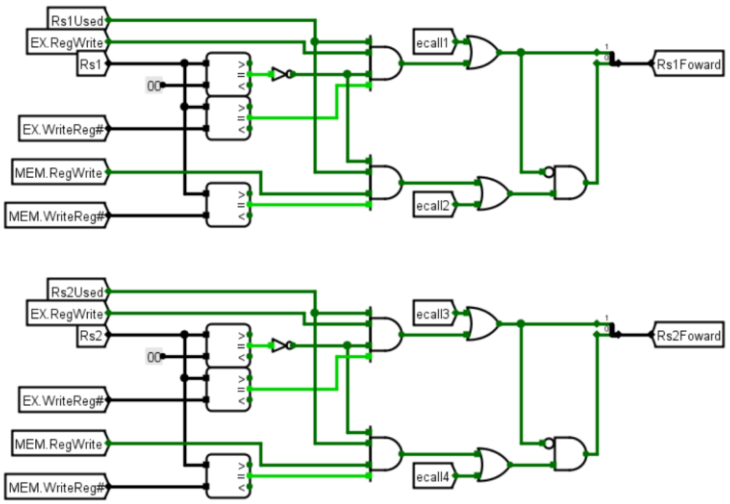

图 3.15 其他数据相关检测和处理电路

3.4.2 重定向实现

在流水线的 EX 段 ALU 的 A、B 操作数输入前各增加一个多路选择器，分别由 ID/EX 流水寄存器传输的 Rs1Forward、Rs2Forward 选择，而多路选择器的输入来源于原 r1 和 r2、Mem.AluResult、RDin。从而实现数据的重定向。实现电路如所示。

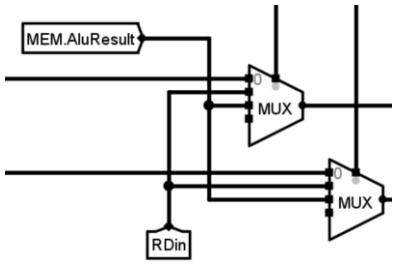

图 3.16 重定向实现

3.5 单级中断机制实现

1、中断使能信号 IE 实现：

采用 D 触发器，上升沿触发。无中断时，IE=1，开中断；当中断信号为 1 时，IE 为 0，关中断；当为 uret 指令时，IE=1，开中断。如图 3.17 所示。

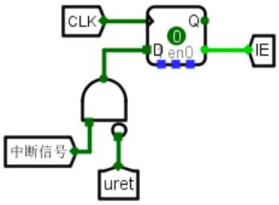

图 3.17 单级中断中断使能信号 IE 实现

2、异常程序计数器 mEPC 实现：

采用 32 位寄存器，上升沿触发。INT 信号接入寄存器使能端，当 INT 为 0，无中断响应时，忽略时钟输入；当 INT 为 1 时，保存断点地址。如图 3.18 所示。

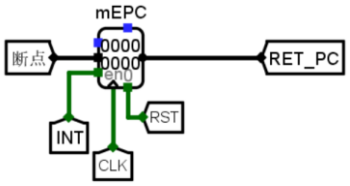

图 3.18 单级中断异常程序计数器 mEPC 实现

3、中断识别实现：

采用优先编码器，如图 3.19 所示。

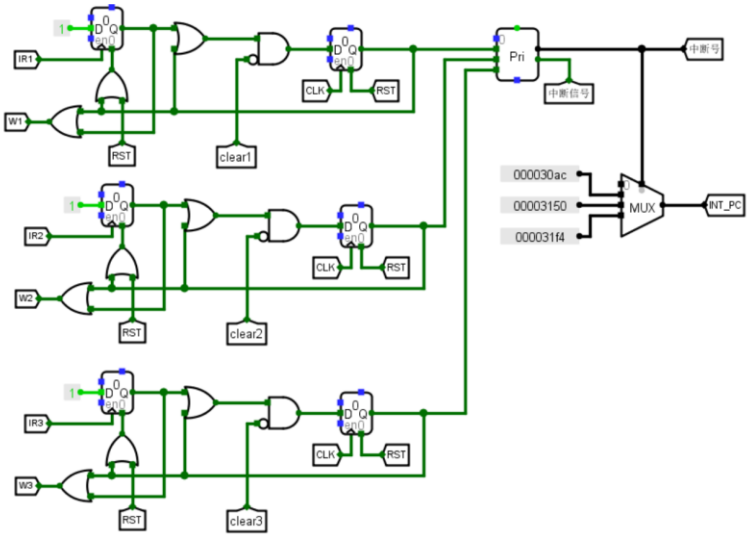

图 3.19 单级中断中断识别实现

左侧为 3 个不同中断源对应的中断等待信号产生电路，将 3 个中断等待信号通过优先编码器，根据优先级进行编码；右侧为当前中断号和查找中断程序入口地址电路，通过多路选择器实现。

4、中断等待信号清零实现：

根据当前中断号，通过译码器后与 uret 指令信号相与后，产生 3 个中断等待信号的清零信号，分别对应 3 个不同中断源。当当前中断执行到 uret 指令时，对应 clear 信号变为 1，从而清除图 3.19 中当前中断的等待信号。如图 3.20 所示：

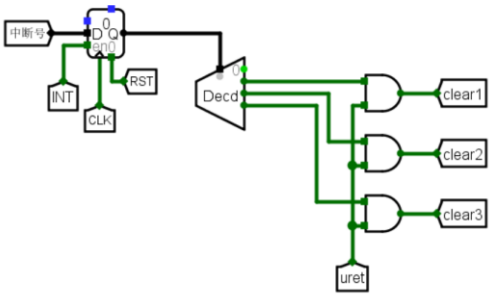

图 3.20 单级中断中断等待信号清零实现

3.6 多级中断机制实现

1、硬件堆栈实现：

使用 3 个 mEPC，对应 3 个不同优先级的中断。根据即将相应的中断号决定将断点地址保存至哪个 mEPC，使用译码器实现。再根据当前运行中断号，通过多路选择器对 3 个 mEPC 的值选择，输出当前中断的返回地址。如图 3.21 所示。

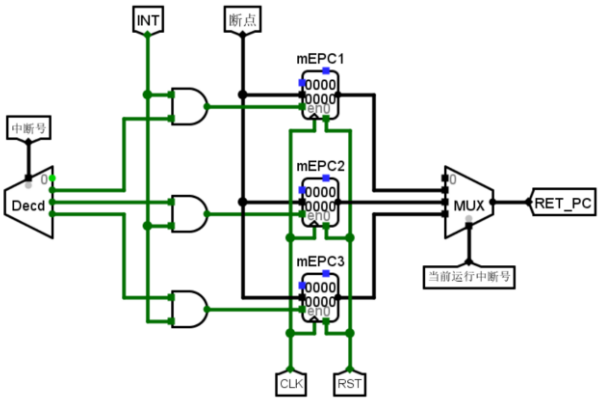

图 3.21 多级中断硬件堆栈实现

2、中断使能信号 IE 实现：

在单级中断中 IE 实现的基础上，增加对指令 csrrsi、csrrci 的支持。csrrsi 指令表示开中断，要将 IE 置为 1；csrrci 指令表示关中断，要将 IE 置为 0。如图 3.22 所示。

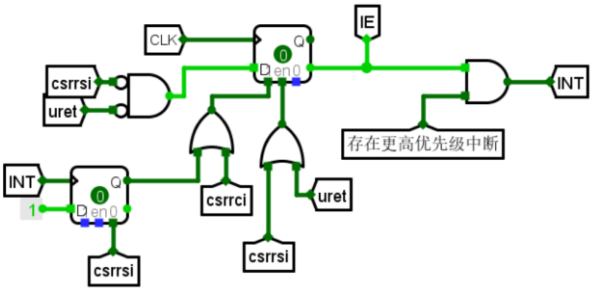

图 3.22 多级中断中断使能信号 IE 实现

3.7 基于重定向流水的单级中断机制实现

本部分只需综合重定向流水和单级中断即可。与重定向流水不同的是，需要在流水寄存器的同步清零端通过或门增加 INT 信号作为控制信号，在中断到来时插入 2 个气泡。

而本部分的重点在于断点地址的选择。理想情况下，即中断时 EX 段不是气泡，需要保存的断点为 EX.PC。但是由于分支相关和 Load-Use 相关会导致中断时 EX 段为气泡的情况。在这种情况下，只可能有下列三种情况：

（1）前一个周期出现了 Load-Use 相关，需要保存的断点为 ID.PC；

（2）前一个周期出现了分支相关，需要保存的断点为 IF.PC；

（3）前二个周期出现了分支相关，需要保存的断点为 ID.PC。

对于这些情况的判断和对断点来源的选择的实现电路如下所示。

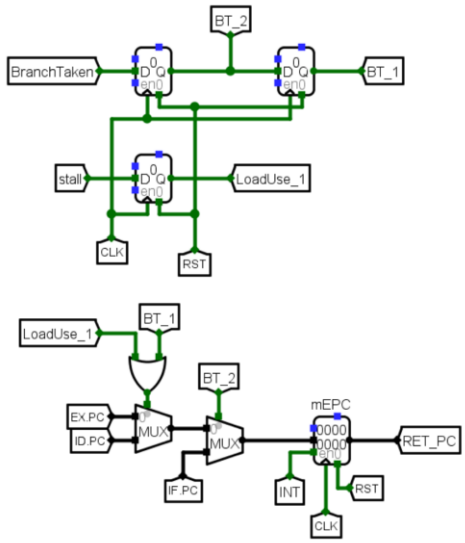

图 3.23 断点来源选择

此外，注意：uret 指令属于分支跳转指令，因此也在 EX 段进行处理跳转。

3.8 动态分支预测机制实现

3.8.1 BTB 实现

1、八路全相连映射 CACHE 实现

参照上学期组原实验。如图 3.24 所示。

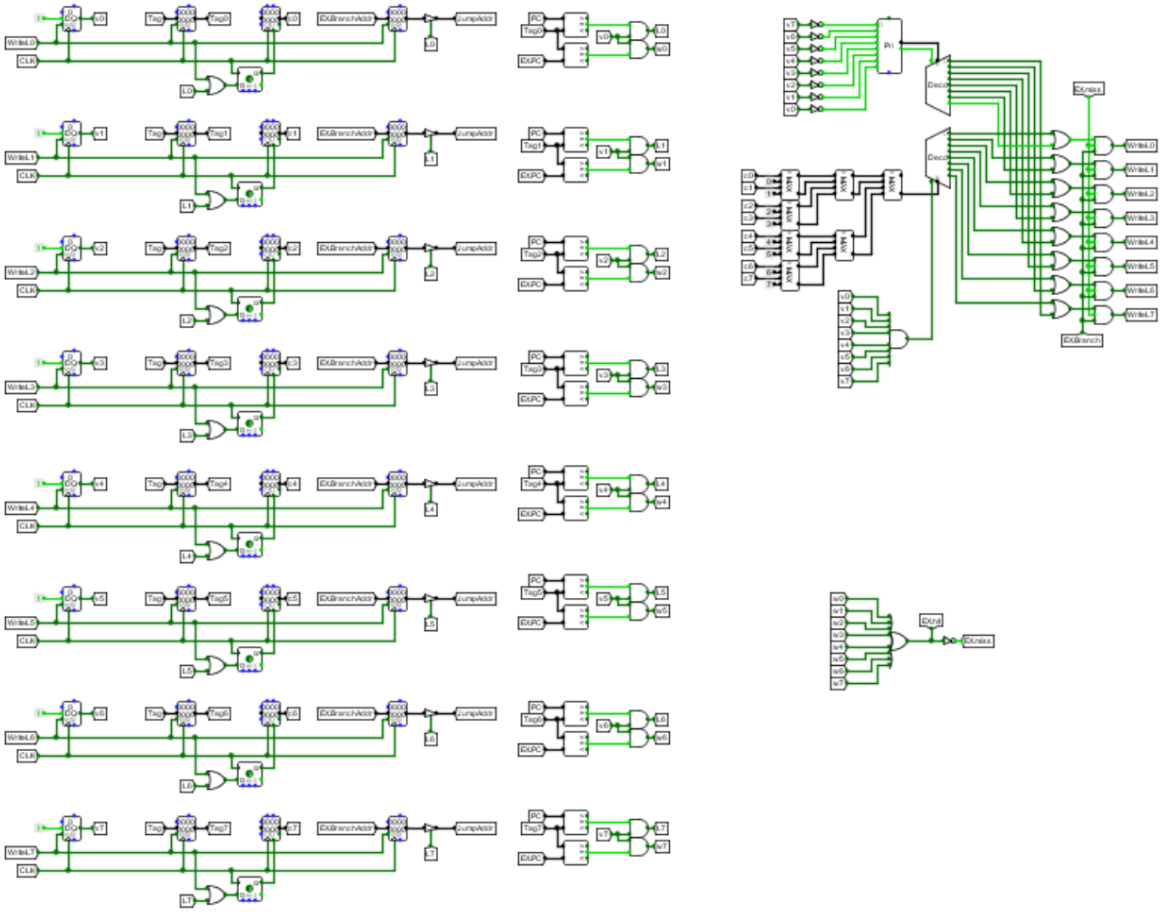

图 3.24 八路全相连映射 CACHE

右上方为 LRU 算法实现电路，采用 2 路归并比较找出计数值最大的 cache 行，将该行的写信号置为 1。

2、状态位转换实现：

采用双位预测，根据书中的双位预测状态转换图实现相关逻辑。如图 3.25 所示。high、low 表示当前状态位的高位和低位，nexthigh、nextlow 表示更新后状态位的高位和低位，jump 表示实际是否跳转。write 信号为 1 时，将状态位初始化为 10 或 01。

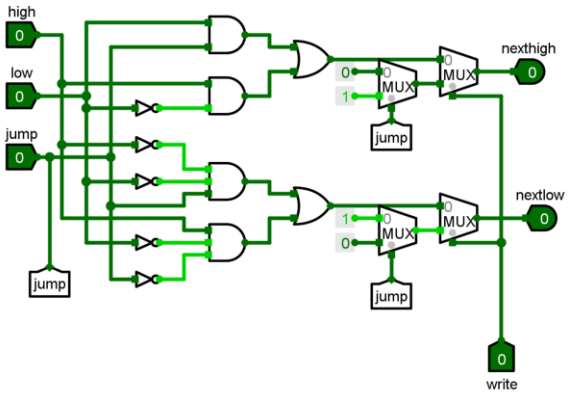

图 3.25 状态位转换实现

3、预测跳转实现：

共有 8 个预测状态位，对应 cache 的 8 行，每一行使用一个 2 位寄存器锁存当前状态位。先将上面的状态位转换电路封装，应用到每一行，该封装电路的输出作为寄存器的输入，在时钟上升沿更新预测状态位。然后每一行使用三态门，根据相应 cache 行的命中信号，对预测状态位进行控制输出，并根据状态位的高位预测是否跳转。高位为 1，则预测跳转，PredictJump 为 1；高位为 0，则预测不跳转，PredictJump 为 0。

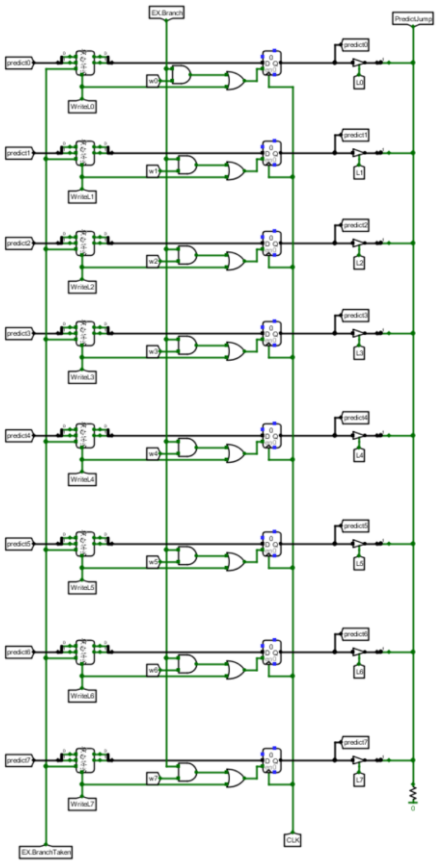

图 3.26 预测跳转实现

3.8.2 IF 段地址跳转实现

根据 BTB 输出的预测跳转信号 PredictJump，从顺序地址 IF.PC+4 和预测跳转地址 PredictAddr 中选择预测的下一条指令地址。

EX 段需要产生预测错误信号 EX.PredictErr，并接入 IF/ID、ID/EX 流水寄存器的同步清零端。

PC 的输入端，根据 EX.PredictErr 进行选择，若 EX.PredictErr 为 1，则输入正确的下一条指令的地址 EX.PC+4，同时清除 ID、EX 段的错误指令，插入气泡；若 EX.PredictErr 为 0，则输入预测地址。

如图 3.27 所示。

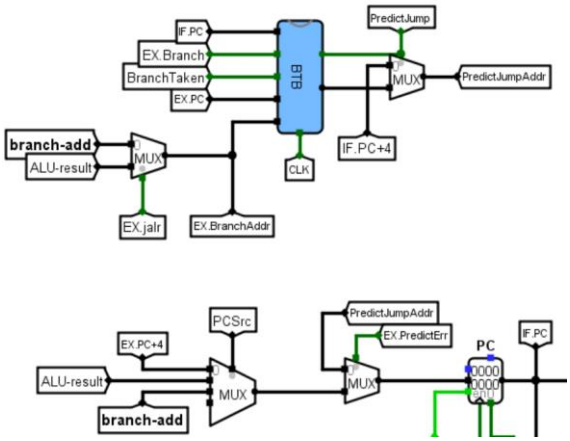

图 3.27 IF 段地址跳转实现

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 单周期 CPU 测试

将自己的 ccab 指令对应的测试程序加入到 benchmark.asm 中，生成新的基准测试程序 benchmark.hex，运行结果如图 4.1 所示。

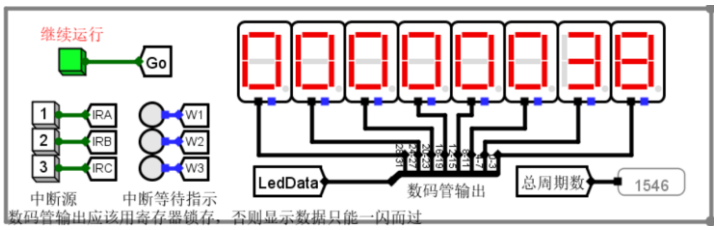

图 4.1 单周期 CPU 测试参数结果

内存排序结果如图 4.2 所示。

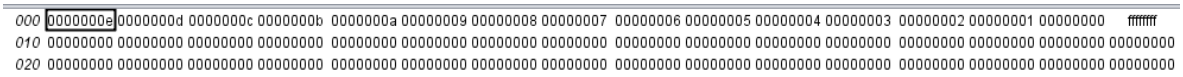

图 4.2 单周期 CPU 测试内存排序结果

上述测试结果与预期一致，任务完成。

4.1.2 理想流水线测试

在理想流水线中加载“risc-v-分支预测测试.hex”，运行结果如图 4.3 所示。

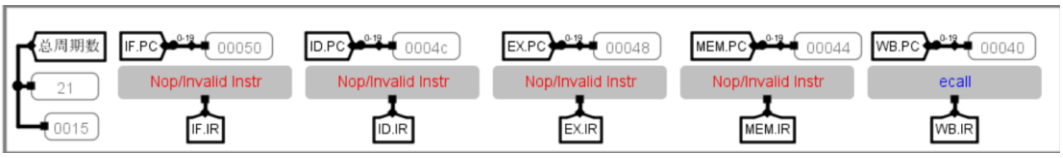

图 4.3 理想流水线测试参数结果

总周期数正确。内存结果如图 4.4 所示：

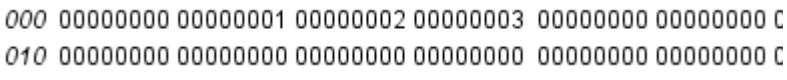

图 4.4 理想流水线测试内存结果

4.1.3 气泡流水线测试

在气泡流水线中加载“benchmark.hex”，运行结果如下，符合预期。

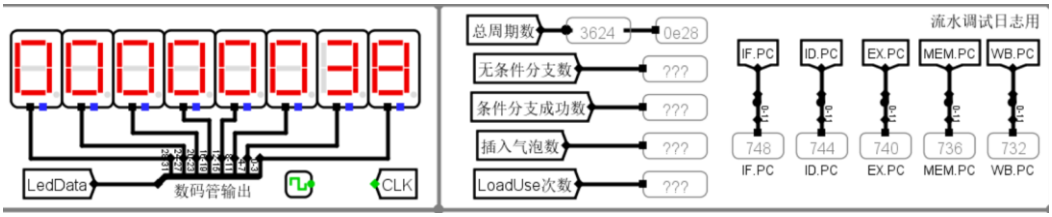

图 4.5 气泡流水测试结果

4.1.4 重定向流水线测试

在重定向流水线中加载“benchmark.hex”，运行结果如下，符合预期。

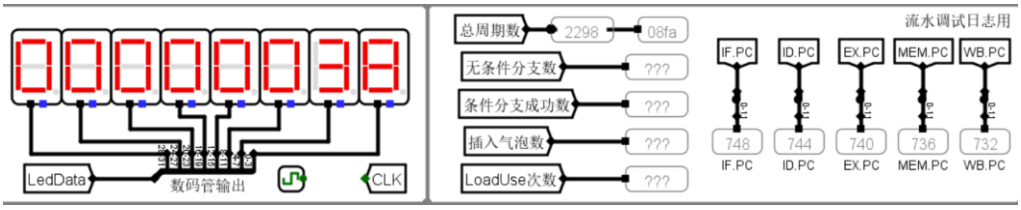

图 4.6 重定向流水测试结果

4.1.5 单级中断测试

在单周期+单级中断电路中加载“risc-v 单级中断测试程序.hex”，并利用单级中断信号测试模拟器模拟按键中断。中断响应顺序应为“1，3，2，1”，最后返回到主程序执行，测试结果符合预期。运行时截图如下。

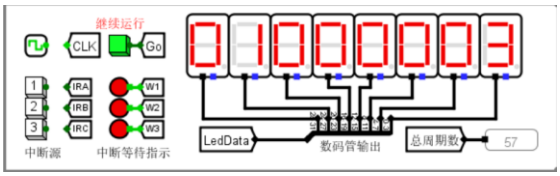

图 4.7 单级中断测试截图

4.1.6 多级中断测试

在单周期+多级中断电路中加载程序“risc-v 多级中断测试(EPC 硬件堆栈保护).hex”，并利用多级中断信号测试模拟器模拟按键中断。中断响应顺序应为“3，2，1，3，1”，最后返回到主程序执行，测试结果符合预期。运行时截图如下。

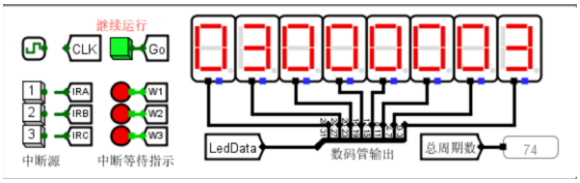

图 4.8 多级中断测试截图

4.1.7 基于重定向流水线的单级中断测试

和单周期单级中断类似，加载“risc-v 单级中断测试程序.hex”，并利用单级中断信号测试模拟器模拟按键中断。中断响应顺序应为“1，3，2，1”，最后返回到主程序执行，测试结果符合预期。运行时截图如下。

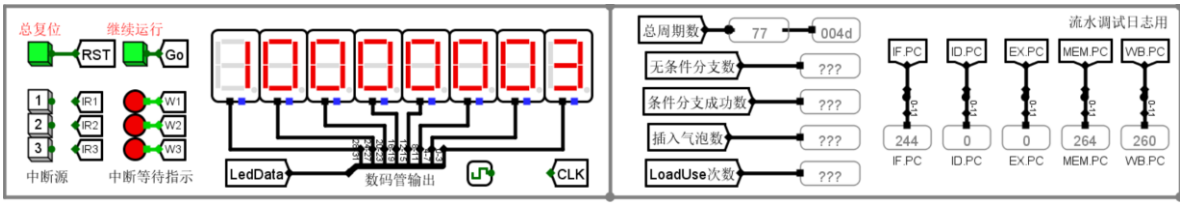

图 4.9 流水中断测试截图

4.1.8 动态分支预测测试

在重定向动态分支预测电路中加载程序“benchmark.hex”，最终总周期数应小于1800，测试结果的总周期数为 1760，符合要求。运行结果如下：

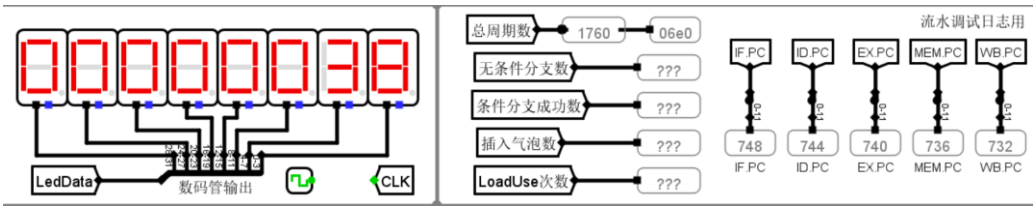

图 4.10 动态分支预测测试结果

4.2 性能分析

下表列出了对于测试基准程序“benchmark.hex”，不同方案的总时钟周期数。

表 4.1 各方案基准测试总周期表

方案	单周期 CPU	气泡流水线	重定向流水线	动态分支预测
总周期数	1546	3624	2298	1760

由上表可知，与基本的单周期 CPU 相比，流水线的总周期数都更多。而气泡流

水线因为插入了大量的气泡，所以总周期数最多；重定向流水线改进了对数据相关的处理，用旁路技术代替插入气泡，一定程度提高了流水的性能，但是分支相关和 Load_Use 相关还是要插入气泡；基于重定向流水线的动态分支预测技术，进一步优化了对分支相关的处理，使得流水周期大幅减少，流水性能得到很明显的提升。

4.3 主要故障与调试

4.3.1 理想流水线故障

ID 段寄存器堆写回数据问题。

故障现象：WB 段的写回数据无法正确写入 ID 段的寄存器堆。

原因分析：如图 4.11 所示。寄存器堆写寄存器编号 W#端口的输入来源是根据 ID 段的指令字得到的，而写数据 RDin 却来自 WB 段，也就是写入地址和写入数据分属不同的指令，从而造成数据紊乱。

解决方案：调整 ID 段寄存器堆的写寄存器编号 W#的输入来源，将 rd 直接送入 ID/EX 流水寄存器中，并逐段向后传递到 WB 段。最后由 WB 段的 MEM/WB 流水寄存器送回 ID 段寄存器堆的写寄存器编号 W#端口。

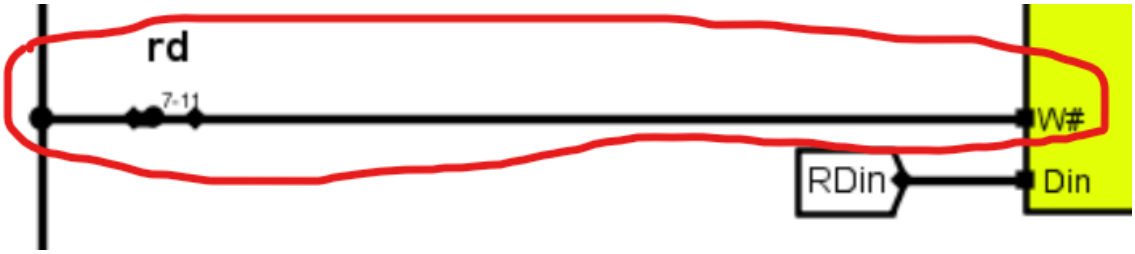

图 4.11 ID 段寄存器堆写回数据错误

4.3.2 单级中断故障

中断响应顺序错误问题。

故障现象：单级中断测试时，中断响应顺序为“1，2，1”，而不是预期的“1，3，2”。

原因分析：如图 4.12 所示。图中的中断号是优先编码器的输出，表示当前具有最高优先级的、中断等待信号为 1 的中断号，不一定是当前中断程序对应的中断号，导致产生的中断清零信号错误。

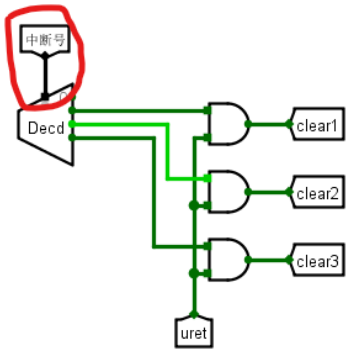

图 4.12 单级中断清零信号产生错误

解决方案：在响应中断时，将其对应的中断号锁存进一个寄存器中，该寄存器在中断程序过程中忽略时钟，值不再改变，从而保证当前中断号不受其他中断信号影响。如所示。

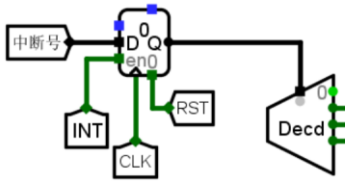

图 4.13 中断清零信号错误问题解决

4.3.3 流水中断故障

故障现象：中断响应结束后无法正常回到主程序，陷入 3 号中断程序的死循环。

原因分析：通过可计数暂停的时钟源定位到进入到中断产生的前几拍，然后单步调试。发现产生中断后，在进入中断程序前，还执行了原本 IF/ID、ID/EX 流水寄存器的指令，而没有将其清除。

解决方案：中断信号相当于分支跳转指令，所以在重定向流水中应通过插入气泡解决。将 INT 信号加入到 IF/ID、ID/EX 流水寄存器同步清零端的控制信号中。如图 4.14 所示。

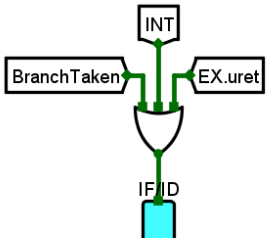

图 4.14 流水中断故障解决

4.4 实验进度

表 4.2 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 RISC-V 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。
第三天	复习关于指令流水线的知识点，设计流水寄存器并组装，完成理想流水线的基本构建。
第四天	调试理想流水线，通过测试。学习关于气泡流水线的知识，实现气泡相关逻辑，并加入流水线。
第五天	调试气泡流水线，通过测试。学习关于重定向流水线的知识，实现重定向相关逻辑，调试电路并通过测试。
第六天	复习关于单级中断的知识，实现单级中断部分的电路，并加入单周期 CPU 中，通过测试。
第七天	复习关于多级中断的知识，修改单级中断电路实现多级中断，通过测试。
第八天	在重定向流水线中加入单级中断，并通过测试。学习关于动态分支预测的知识。
第九天	构建 BTB 表，实现动态预测逻辑，通过相应测试。完成课程设计。

5 团队项目

5.1 项目内容

本项目是一个自制的音乐播放器，主要内容为利用 Logisim 实现音频的录制与播放。通过 midi keyboard 组件进行乐曲演奏，并将音符数据进行采样，通过单周期 CPU 进行存储，最后通过 sound emitter 组件播放数据存储器中录制的音频或外部存储的音频文件。

据此，可分为 3 个模块：音频采样模块、音频存储模块、音频播放模块。

5.2 项目分析

1、选题原因：

喜欢音乐，于是决定用 Logisim 基于本次课设的单周期 CPU 制作一款音乐播放器。

2、可行性分析：

乐曲可以转换为特定格式的音频信号进行存储，利用 midi keyboard 组件可以实现音频的输入，通过 sound emitter 组件可以实现不同音色的音频的播放，音频的录制与播放可以实现。

3、工作量分析：

团队成员有一定的乐理知识，工作量适中，可以正常完成。

5.3 项目设计与实现

将所有模块内容和电路与本课程设计的单周期 CPU 整合，主要是数据存储器 RAM 部分的接线处理。最终项目实现电路如下：

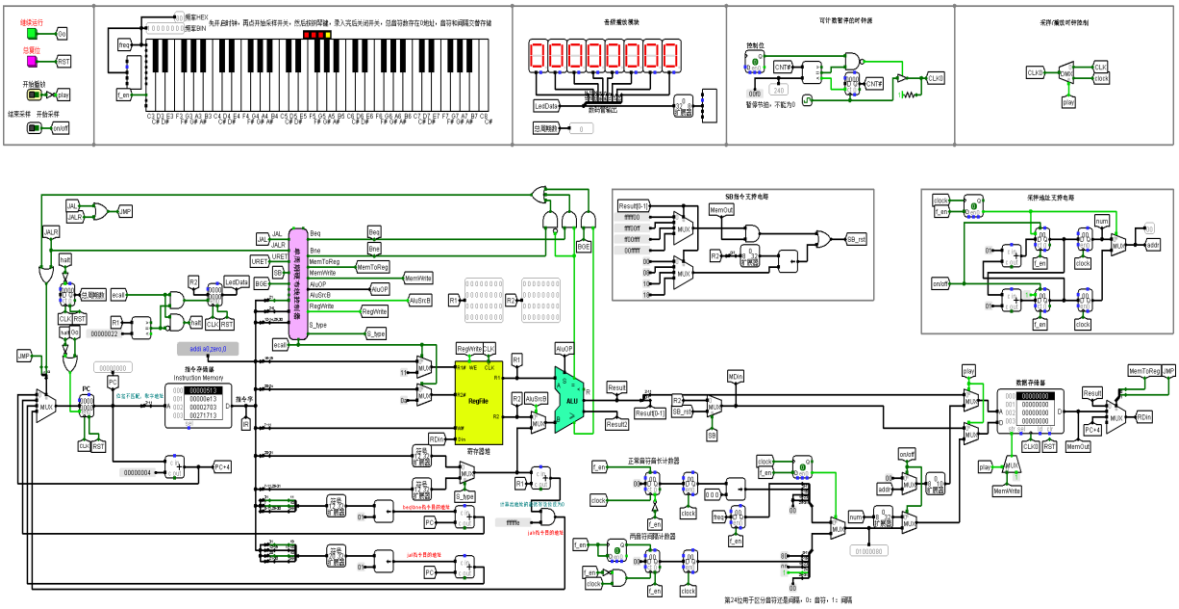

图 5.1 项目总体电路

5.3.1 音频存储模块

音频以一个一个音符的形式存储，每个音符占用 32 位（4 个字节），对于音符存储格式的设计如下：

表 5.1 音符存储格式表

位数	功能	说明
0-7 位	频率编码	若为普通音符则代表频率的 8 位二进制，如 00111100；若为休止符则为全 0。
8-15 位	音长编码	表示每个音符的持续时间。
16-23 位	音色编码	代表该音符对应的乐器音色。
24 位	休止符标识	为 1 时代表当前音符为休止符。
25-31 位	暂未使用	

5.3.2 音频采样模块

音频采样分为两部分：频率和音长。频率由 Midi Keyboard 组件在按下琴键时生成，不同琴键的频率不同；音长通过计数器计数实现，录制音频时，计数器从 0 开始从按下一个琴键开始计数，直到松开琴键才停止计数，计数器的值则表示音长。

为了使音乐有节奏，不仅要记录普通音符的长度，还要记录两音符之间的间隔（休

止符）长度。实现方法与普通音符的音长相同。

最后，以总音符个数为开头，所有采样数据以固定的 32 位格式按序存在数据存储器中。

音频采样电路如下：

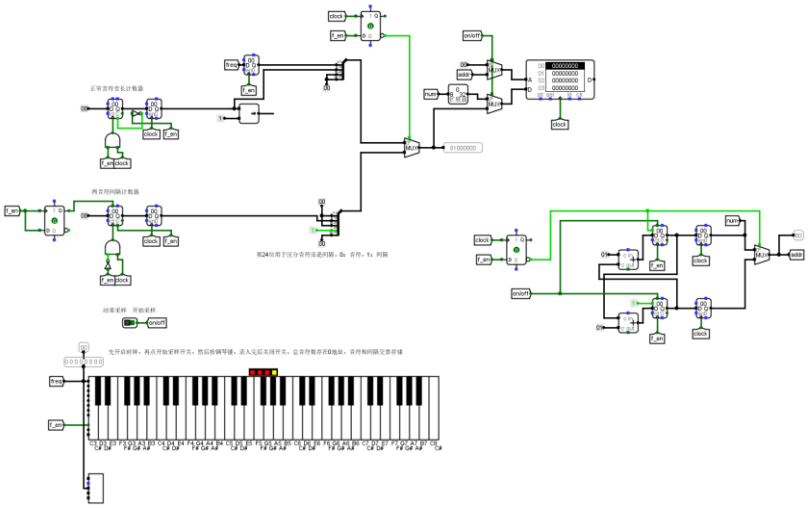

图 5.2 音频采样电路实现

5.3.3 音频播放模块

本部分主要通过软件实现，即编写汇编程序让 CPU 自动从数据存储器中读取音频数据并逐音符输出到 LedData 中。再利用 sound emitter 组件播放 LedData 表示的音符信息。

汇编代码的实现如下：

```
.text
loop:
    addi a0, zero, 0  # 重置a0，用于存储每个读取的音符
    addi t3, zero, 0 # 音符个数计数器
    lw a4, 0(zero) # 将内存中首字节存放的音符个数读取给a4寄存器
    slli a4, a4, 2
    addi a7, zero, 34
    # 播放音乐
    play:
        lw a0, 4(t3) # 读取内存中对应音符

        # 计算音长
        srli t4, a0, 10
        andi t4, t4, 0xff

        ecall  # 输出音符

        # 通过循环实现音长（若为间隔符，则实现间隔时长）
        delay:
            beq t4, zero, next
            addi t4, t4, -1
            jal zero, delay

        next:
            addi a0, zero, 0 # 停止播放音符

        addi t3, t3, 4 # 音符数加1
        bne t3, a4, play

    # system call for loop
    addi a7, zero, 10
    ecall
    jal zero, loop  # 循环播放
```

图 5.3 音频播放汇编代码实现

分析图中代码：首先读取内存开头的音符总数量；然后循环读取音符，将读取到的音符存储在 a0 寄存器中，通过 ecall 指令再将音符输出到 LedData 中，从而播放该音符，循环 delay 用于延长音符输出时间，即音长；在单次播放的结尾，通过 ecall 停机，等待用户在 CPU 中按下 Go 键重新播放音频。

5.4 小组分工与合作

小组成员及分工合作内容如下：

表 5.2 小组成员分工表

成员名	任务分工
张隽翊	组织团队项目，设计音频采样算法
计胜翔	修改、优化、测试音频采样电路
王广凯	设计音符存储格式，编写播放音频文件的汇编代码
王梓熙	资料搜集，汇编器脚本编写，PPT 制作，电路测试

6 设计总结与心得

6.1 课设总结

本次课设，我主要作了如下几点工作：

1） 设计了程序计数器 PC、运算器、控制器和数据通路等主要功能部件，实现了支持 24+4 条指令的单周期 CPU。

2） 设计了流水接口部件，完成了对单周期 CPU 的流水线改装，实现了理想流水线。

3） 设计了气泡相关检测和处理逻辑，完成了对流水冲突的处理，实现了气泡流水线。

4） 设计了重定向相关检测和处理逻辑，完成了对气泡流水线的优化，实现了重定向流水线。

5） 设计了单级中断的异常程序计数器 mEPC、中断识别逻辑等硬件，实现了单周期 CPU 的单级中断机制。

6） 设计了多级中断的硬件堆栈等，完成了对单级中断电路的改装，实现了单周期 CPU 的多级嵌套中断机制。

7） 完成了对重定向流水线和单级中断机制的结合，实现了基于重定向流水线的单级中断。

8） 设计了 BTB 表、状态位转换逻辑、预测跳转逻辑等，完成了对重定向流水线的性能优化，实现了动态分支预测机制。

9） 参与团队项目，设计了音符的存储格式，完成了相关 RISC-V 汇编程序编写，实现了从内存中读取音频并播放。

6.2 课设心得

本次课程设计共耗时 2 周，每阶段的难度递增。过程中遇到了许多难题，好在最后都解决了，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，第一个任务是使用 Logism 设计单周期 CPU，虽然难度不是很大，但由于刚开始有点懵，不知从哪开始，所以完成第一项任务用了 2 天半时间。紧接着，理想流水线 CPU 的设计烦在为了电路美观，要合理设计、不断调整

流水寄存器封装的各个端口的位置和布局。气泡流水、重定向流水有一定的难度，因为没有学过，但是好在书本中有详细的原理解释和实现方法，仔细阅读书本再稍加修改就不难完成了。接下来，虽然组原课上学过单级中断和多级中断的原理，但是书上并没有具体电路的设计和实现，这部分最终通过自己努力、网上查找资料和与同学交流完成。中断电路中有很多细节需要注意，debug 难度也较大。最后的动态分支预测任务的工程量最大，涉及到上学期实验中的 cache 相关内容，在 Logisim 中画电路较为费劲，但好在书上也有这部分的详细解释。

做团队项目时，小组成员在一起互相讨论问题、沟通想法、分工协作、协调进度，我很享受这种团队氛围，最终大家一起实现了一个录音和播放音乐的小项目。

整个课设过程考察了我们对基本知识的理解，锻炼了我们学习新知识的能力、创新设计能力、发现并解决问题的能力等，培养了我们的计算机系统设计能力。团队项目锻炼了我们的表达能力和协作能力。

整体来说，课设的体验还是很好的，感谢课程组老师的辛苦付出。唯一的一点建议是最好将各阶段的要求和检查标准明确写在任务书中，而不是只在 QQ 群中说，因为消息太多，可能漏看。

最后，感谢老师们对于我们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，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：计胜翔