华中科技大学

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：	CS2009
学　　号：	U202015557
姓　　名：	周帅君
电　　话：	15979950220
邮　　件：	zhoushuaijun2333@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 LOGISIM 仿真平台，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。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

（1） 支持表 1.1 规定的 32 位 RISC-V 指令集；

（2） 在 CCAB 扩展指令集中支持 2 条 C 类运算指令，1 条 M 类储存指令，1 条 B 类分支指令，具体任务每位同学不一样，本人任务为表 1.1 中 25~28；

（3） 支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（4） 支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（5） 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。

（6） 能运行教师提供的标准测试程序，并自动统计执行周期数

（7） 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

#	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	

#	RISC-V	指令类型	简单功能描述	备注
14	BEQ	B	相等跳转	
15	BNE	B	不相等跳转	
16	SLT	R	小于置数	
17	STI	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 等待 go 按键暂停 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，可简化关中断
24	URET	I	中断返回	清中断，mEPC 送 PC，开中断
25	SLL	R	逻辑左移	指令格式参考 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
26	SLTIU	I	无符号小于立即数置数	
27	LB	I	加载字节	
28	BLT	B	小于跳转	

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哈佛结构，即指令储存器与数据储存器相分离的结构。设计上采用简单迭代法，先设计出一条 R 型指令的基本数据通路，完成后再在这个基础上增加新的数据通路，直至所有指令都能正常运行。控制器采用硬布线控制器。

总体结构图如图 2.1 所示。

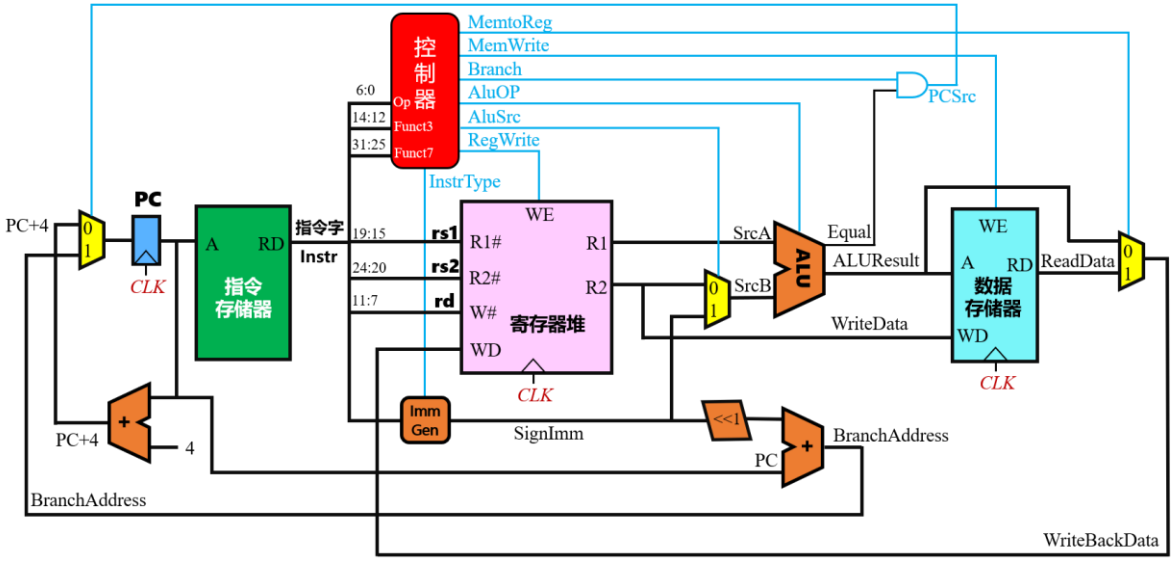

图 2.1 总体结构图

2.1.1 主要功能部件

单周期 CPU 的主要功能部件分为程序计数器 PC、指令储存器 IM、寄存器堆 RF、运算器 ALU 和数据存储器 DM。

1. 程序计数器 PC

程序计数器由一个 32 位寄存器实现，输入由是否是跳转类指令决定是分支目标地址还是 PC+4。

2. 指令存储器 IM

由只读存储器实现，一个地址对应四个字节即一条指令的数据长度。

3. 运算器 ALU

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**错误!未找到引用源。**
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	操作数比较结果是否为“<”
≥	输出	1	操作数比较结果是否为“≥”
=	输出	1	操作数比较结果是否为“=”

表 2.2 运算器功能码对应功能

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

ALU_OP	十进制	运算功能
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

4. 寄存器堆 RF

包含 32 个 32 位寄存器，由 CS3410 库提供 Register File。

5. 数据存储器 DM

由随机存储器实现，一个地址对应四个字节的数据。

2.1.2 数据通路的设计

表 2. 3 指令系统数据通路框架

指令	RF				ALU			DM	
	R1#	R2#	W#	Din	A	B	OP	Addr	Din
add	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	5		
sub	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	6		
and	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	7		
or	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	8		
slt	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	11		
sltu	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	12		
addi	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	5		
andi	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	7		
ori	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	8		
xori	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	9		
slti	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	11		
slli	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	0		
srli	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	2		
srai	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	1		
lw	[15:19]		[7:11]	DMout	x[r1]	[20:31]	5	ALU result	

指令	RF				ALU			DM	
	R1#	R2#	W#	Din	A	B	OP	Addr	Din
sw	[15:19]	[20:24]			x[r1]	[7:11,25:31]	5	ALU result	x[r2]
ecall	0x11	0xa			x[r1]	0x22	11		
beq	[15:19]	[20:24]			x[r1]	x[r2]	11		
bne	[15:19]	[20:24]			x[r1]	x[r2]	11		
jal			[7:11]	PC+4					
jalr	[15:19]		[7:11]	PC+4	x[r1]	[20:31]	5		
sll	[15:19]	[20:24]	[7:11]	ALU result	x[r1]	x[r2]	0		
sltiu	[15:19]		[7:11]	ALU result	x[r1]	[20:31]	12		
lb	[15:19]		[7:11]	DMout	x[r1]	[20:31]	5	ALU result	
blt	[15:19]	[20:24]			x[r1]	x[r2]	11		

除此之外，还需要为跳转类指令 beq、bne、blt、jal、jalr 增加设置 PC 的通路；为指令 jal、jalr 增加 PC 写回寄存器堆的通路；为指令 ecall 增加 x[r2]输出到 LedData 的通路，并用寄存器锁存，以及一条判断是否停机的通路；为指令 lb 增加字节选择的路径。

2.1.3 控制器的设计

首先对控制信号进行统计，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 2.4 所示。

表 2. 4 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_OP	0x0~0x1100	不同的取值对应不同的运算器 ALU 功能，如**错误!未找到引用源。**
MemtoReg	0	选择 ALU result 作为写回寄存器堆的数据
	1	选择数据存储器中 DM 读取的数据作为写回寄存器堆的数据
MemWrite	0	不向数据存储器中 DM 中写入数据
	1	向数据存储器中 DM 中写入数据
ALU_src	0	选择 x[r2]作为运算器 B 输入

控制信号	取值	说明
	1	选择立即数作为运算器 B 输入
RegWrite	0	不向寄存器堆 RF 写入数据
	1	向寄存器堆 RF 写入数据
ecall	0	R1#选择[15:19]，R2#选择[20:24]，立即数不选 0x22，禁用 Led 数据寄存器
	1	R1#选择 0x11，R2#选择 0xa，立即数选择 0x22，使能 Led 数据寄存器
S_type	0	立即数选择[20:31]
	1	立即数选择[7:11,25:31]
BEQ	0	不进行 beq 指令判断阶段
	1	进行 beq 指令判断阶段
BNE	0	不进行 bne 指令判断阶段
	1	进行 bne 指令判断阶段
jal	0	PC 不选择 jal 通路的计算结果，寄存器堆数据不选择 PC+4
	1	PC 选择 jal 通路的计算结果，寄存器堆数据选择 PC+4
jalr	0	PC 不选择 jalr 通路的计算结果，寄存器堆数据不选择 PC+4
	1	PC 选择 jalr 通路的计算结果，寄存器堆数据选择 PC+4
lb	0	选择写回数据
	1	选择根据 ALU result[0:1]确定的写回数据的一个字节
blt		不进行 blt 指令判断阶段
		进行 blt 指令判断阶段

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

表 2.5 主控制器控制信号框架

指令	ALU_OP	MemtoReg	MemWrite	ALU_src	RegWrite	ecall	S_type	BEQ	BNE	jal	jalr	lb	blt
add	5				1								
sub	6				1								
and	7				1								

指令	ALU_OP	MemtoReg	MemWrite	ALU_src	RegWrite	ecall	S_type	BEQ	BNE	jal	jalr	lb	blt
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	11			1		1							
beq	11							1					
bne	11								1				
jal					1					1			
jalr	5			1	1						1		
sll					1								
sltiu				1	1								
lb		1		1	1							1	
blt													1

2.2 中断机制设计

2.2.1 总体设计

对于单周期单级中断，需要在单周期 CPU 的基础上增加中断信号采样，根据中断号确定中断入口地址，并将中断断点 PC 暂存，中断返回时恢复现场，送回断点 PC。

对于单周期多级中断，需要在单周期单级中断 CPU 基础上增加中断优先级判断，高级中断应能中断低级中断的服务程序，反之不能，需要等待高级中断服务程序完成。断电保存也应能支持保存多个断点。

2.2.2 硬件设计

对于单周期单级中断，可参考中断按键参考电路，为每个每个中断号提供一个中断采集电路，中断信号通过优先编码器被中断处理电路识别中断号，将中断入口地址送入 PC，并让中断使能寄存器 IE 关中断。中断断点通过异常程序计数器 mEPC 保存。中断返回恢复现场时，同时将中断使能寄存器恢复为开中断状态。

对于单周期多级中断，增加一个中断优先级判断电路。为使设计方便，将无中断视为 0 级中断服务程序。通过中断优先级电路的判断，若是高级的中断则直接通过中断使能寄存器 IE 开中断，保存断点，将中断入口地址送入 PC；反之则将中断信号通过寄存器暂存。嵌套的中断所产生的断点也通过堆栈的方式保存，堆栈通过硬件堆栈实现，即将不同层的数据存储在不同的寄存器中，通过计数器进行选择，寄存器单调增减的特性正好符合堆栈的存储方式。

2.2.3 软件设计

利用 risc-v 汇编模拟运行软件 RARS 获取中断入口地址。

2.3 流水 CPU 设计

2.3.1 总体设计

CPU 流水线设计为 5 段，分别为取指令阶段 IF、译码阶段 ID、执行阶段 EX、存储阶段 MEM 以及写回阶段 WB。取指令阶段通过 PC 从指令储存器 IM 中取出指令字；译码阶段根据指令字准备后续执行所需的操作数和控制信号；执行阶段根据译码阶段提供的操作数和指令信号使用运算器 ALU 进行计算，理想流水暂不涉及跳转指令的处理；存储阶段根据执行阶段的运行结果对数据存储器 DM 进行读取或写入；写回阶段将储存阶段读取的数据写回寄存器堆 RF 中。前后段之间相差一个节拍，相邻间的控制信号通过流水接口部件传递。

2.3.2 流水接口部件设计

流水接口使用多个寄存器暂存、传递数据，每个寄存器负责一项数据，与电路采用相同的同步时钟。

不同阶段间需要传递的数据不尽相同，总的来说，都需要传递的数据为当前阶段的 PC、IR 值，主要用于正确性检查，（除 IF/ID 接口外）后续阶段所需的控制信号。

对于 ID/EX 接口，需要额外传递跟据 R1#和 R2#从寄存器堆 RF 取出的、指令中的立即数，作为操作数；写回阶段需要用到的写回寄存器号 W#。对于 EX/MEM 接口，需要额外传递运算器 ALU 运算结果；R2#号寄存器数据；写回阶段需要用到的写回寄存器号 W#。对于 MEM/WB 接口，需要额外传递运算器 ALU 运算结果；从数据存储器 DM 读取的数据；写回阶段需要用到的写回寄存器号 W#。

2.3.3 理想流水线设计

将流水接口部件设计、封装好后，用相应接口部件在单周期 CPU 的基础上按阶段分割。IF/ID 段分割于从指令存储器取出指令字与对指令字进行解析之间；ID/EX 段分割于准备好控制信号，R1#内容、R2#内容与立即数间；EX/MEM 段分割于运算器 ALU 得出结果与访问数据存储器 DM 间；MEM/WB 段分割于根据 ALU 运算结果和数据存储器 DM 得出写回数据间。

2.4 气泡式流水线设计

2.4.1 总体设计

气泡流水线通过在执行阶段插入气泡来解决数据相关冲突和分支指令引起的控制冲突。

当检测到 ID 段指令与 EX 段指令存在数据相关冲突，即 ID 段指令使用的源寄存器与 EX 段的目的寄存器存在相同时，暂停 IF 与 ID 段时钟，停止读取指令。EX 段指令继续随时钟周期执行，执行过程中在 EX 段插入气泡，即空指令，待该指令完成写回阶段数据写回寄存器堆，再继续流水线的运行。

由于分支指令在 EX 段执行，当 EX 段检测到分支指令时，会将跳转 PC 传到 IF 段，下一节拍在 IF 段会取出正确的跳转目的地址，所以此时需要清空 EX、ID 的错误指令，及插入气泡。

2.4.2 硬件设计

需要增加新的数据相关检测逻辑，具体为：

$$\begin{aligned} DataHazzard = {} & r1used\ and\ (r1\#! = 0)\ and\ EX.RegWrite\ and\ (r1\# == EX.WriteReg\#) \\ & + r2used\ and\ (r2\#! = 0)\ and\ EX.RegWrite\ and\ (r2\# == EX.WriteReg\#) \\ & + r1used\ and\ (r1\#! = 0)\ and\ MEM.RegWrite\ and\ (r1\# == MEM.WriteReg\#) \\ & + r2used\ and\ (r2\#! = 0)\ and\ MEM.RegWrite\ and\ (r2\# == MEM.WriteReg\#) \end{aligned}$$

其中，r1used 与 r2used 信号由新增加的源寄存器使用情况电路判断出，该电路输入为指令字的 opcode 与 funct 字段。DataHazzard 表示存在数据相关冲突，需要暂停新指令的读取，并在 EX 段插入气泡。

还需要新增数据相关处理逻辑，具体为：当检测到数据相关时，阻塞 IF、ID 段指令的执行；在冲突指令执行完前，在 EX 段插入气泡。

特别地，需要将寄存器堆的写入时刻设置为时钟下跳沿，这样，在数据冲突指令读取数据前就能完成冲突数据的写回，从而避免冲突。

当 EX 段检测到分支指令且执行跳转时，将 IF、ID 段指令清空，即插入气泡避免执行错误指令。

2.5 重定向流水线设计

2.5.1 总体设计

重定向流水线通过将与 EX 段冲突的暂存在 MEM 段或 WB 段接口寄存器中的数据直接送回 EX 段，从而避免插入气泡，提升流水线性能。

暂存在 MEM 段的目的操作数以 ALUresult 的形式暂存，暂存在 WB 段的目的操作数以 RDataIn 的形式暂存。当 EX 段的源操作数与这些存在数据冲突时，将这些数据直接送回 EX 段，替换错误读取源操作数。

特别地，当前一条指令是访存指令且与后一条指令存在数据冲突时（Load-Use 相关），不能使用重定向方法，需通过插入气泡的方式解决，即当检测到 Load-Use 相关时停止读取新的指令，在 EX 段插入一个气泡，即可等待内存数据读取完成后再进行数据重定向。

2.5.2 硬件设计

需要增加新的重定向相关处理逻辑，用于判断 Load-Use 相关和数据相关，具体

为：

$$LoadUse = r1used\ and\ (r1\#! = 0)\ and\ EX.MemRead\ and\ (r1\# == EX.WriteReg\#)$$
$$+r2used\ and\ (r2\#! = 0)\ and\ EX.MemRead\ and\ (r2\# == EX.WriteReg\#)$$

$$if\big(r1used\ and\ (r1\#! = 0)\ and\ EX.RegWrite\ and\ (r1\# == EX.WriteReg\#)\big)$$
$$r1forward = 2$$
$$else\ if\big(r1used\ and\ (r1\#! = 0)\ and\ MEM.RegWrite\ and\ (r1\# == MEM.WriteReg\#)\big)$$
$$r1forward = 1$$
$$else\ r1forward = 0$$

$$if\big(r2used\ and\ (r2\#! = 0)\ and\ EX.RegWrite\ and\ (r2\# == EX.WriteReg\#)\big)$$
$$r2forward = 2$$
$$else\ if\big(r2used\ and\ (r2\#! = 0)\ and\ MEM.RegWrite\ and\ (r2\# == MEM.WriteReg\#)\big)$$
$$r2forward = 1$$
$$else\ r2forward = 0$$

其中 riforward 为 2 表示 ID 段与 EX 段数据相关，为 1 表示 ID 段与 MEM 段数据相关，为 0 表示无数据相关。EX 段运算器 ALU 使用的操作数应根据数据相关获取。

同时，分支跳转类指令依然采用插入气泡的方式处理。

2.6 动态分支预测机制设计

2.6.1 总体设计

动态分支预测通过对跳转指令是否跳转以及跳转位置进行预测，来减少指令误取，减少气泡插入，从而提高流水线性能。

为实现分支预测，需要使用缓存来统计分支跳转历史信息，一般使用分支预测缓冲器 BTB。BTB 表主要包括有效位、分支指令地址、分支目标地址、分支预测历史位和置换标记。IF 段读取分支指令时读取 BTB 信息分支预测和预取，EX 段执行分支指令时根据实际情况更新 BTB 表。

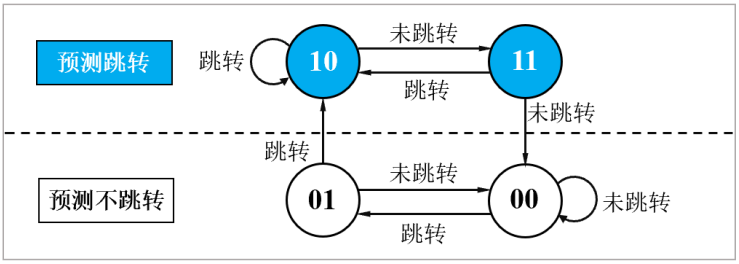

图 2.2 双位预测状态转换图

分支预测历史位是当前分支指令的历史跳转情况的统计，采用两位二进制编码表示。预测状态转换图如图 2.2 所示。

2.6.2 硬件设计

动态分支预测 CPU 在重定向 CPU 的基础上设计。

BTB 表设计为 8 行全相联 cache，数据淘汰方式为近期最少使用（LRU）算法。具体输入输出引脚如表 2.6 所示。

表 2. 6BTB 表逻辑引脚说明

引脚	输入/输出	位宽	功能描述
CLK	输入	1	时钟节拍信号
PC	输入	32	程序计数器地址，是 BTB 表的查找关键字
EX.branch	输入	1	EX 段分支指令信号，为 1 时更新 BTB 表
EX.branchTaken	输入	1	EX 段分支是否跳转，为 1 跳转，为 0 不跳转
EX.PC	输入	32	EX 段指令对应的 PC 地址
EX.branchAddr	输入	32	EX 段分支指令的目标地址
predictJump	输出	1	预测跳转信号，向右依次传给 EX 段，为 1 表示预测跳转
JumpAddr	输出	32	预测跳转的目标地址

当 IF 段取出指令时，会使用 BTB 表进行匹配，若 PC 命中表项，即 BTB 表中有该指令的跳转记录，则下一节拍根据预测进行取指令，否则程序顺序执行。当分支指令进行至 EX 段时，会得出是否跳转以及跳转地址的真实结果，此时再对 BTB 表进行匹配，若 EX.PC 命中则更新 BTB 表分支预测历史位，否则添加新表项。同时若预测失败，测需要在 IF 段重新读取正确跳转目的地址，并清空误取指令，插入气泡。

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）

PC 寄存器由 32 位寄存器实现。PC 的产生方式有顺序执行的 PC+4、jal 和 b 类指令的立即数、jalr 指令的计算结果。通过多路选择器，以 jal、b 类指令、jalr 的指令信号作为选择信号进行选择。Halt 为停机信号，通过非门实现当为 1 时停机，ecall 指令的停机功能也通过此实现。CLK 为时钟信号，PC 寄存器根据该信号上升沿触发。RST 为清零信号。如图 3.1 所示。

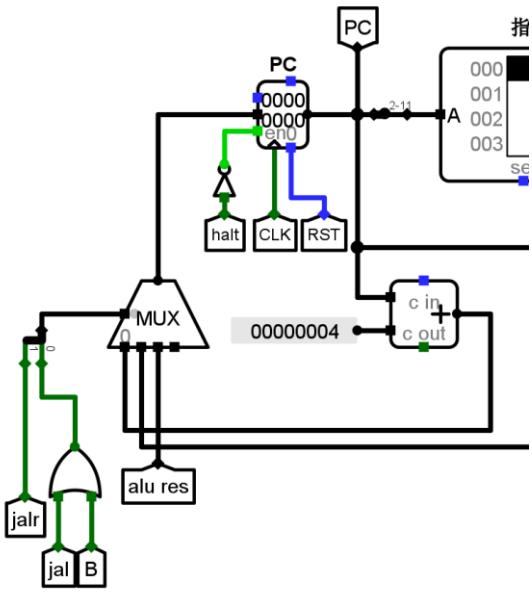

图 3.1 程序计数器（PC）实现

2) 指令存储器（IM）

图 3.2 指令存储器（IM）实现

指令储存器通过地址位宽 10 位、数据位宽 32 位的只读存储器 ROM 实现。由于该 ROM 通过字（四字节）编址，所以 PC 从 2 位开始取值，取 2 至 11 位寻址。如图 3.2 所示。

3) 运算器（ALU）

借助 Logisim 的运算部件实现。运算功能选择通过多路选择器实现。如图 3.3 所示。

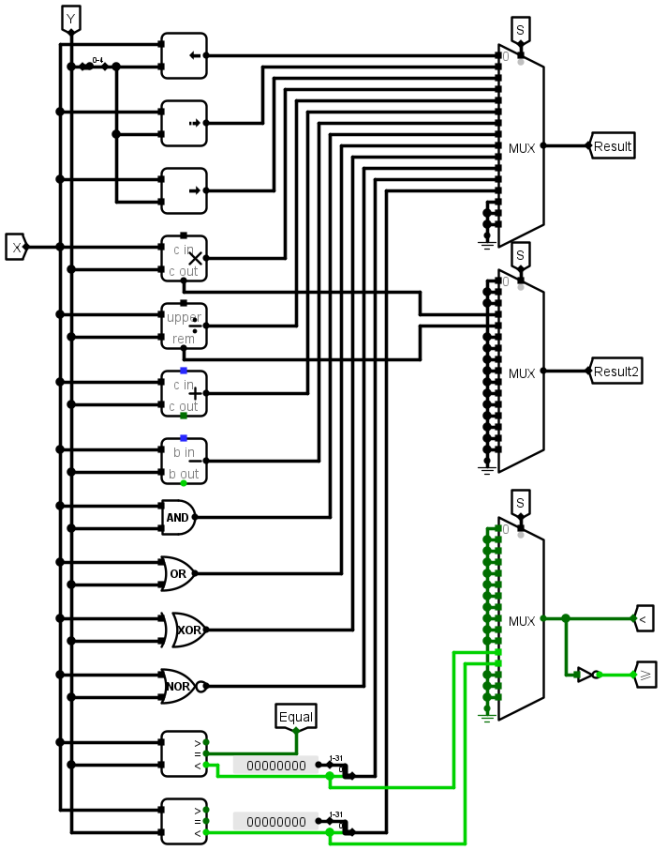

图 3.3 运算器（ALU）实现

4) 寄存器堆（RF）

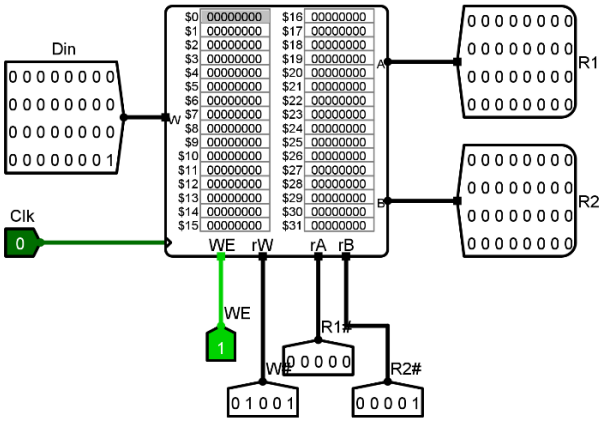

图 3.4 寄存器堆（RF）实现

寄存器堆通过 CS3410 库的 Register File 实现。由 32 个 32 位寄存器组成。如图 3.4 所示。

5) 数据存储器（DM）

通过地址位宽 10 位、数据位宽 32 位的随机存储器 RAM 实现。由于 RAM 按字（4 字节）编址，所以寻址地址从 2 位开始取值，取 2 至 11 位寻址。CLK 为时钟信号。MemWrite 为存储信号。如图 3.5 所示。

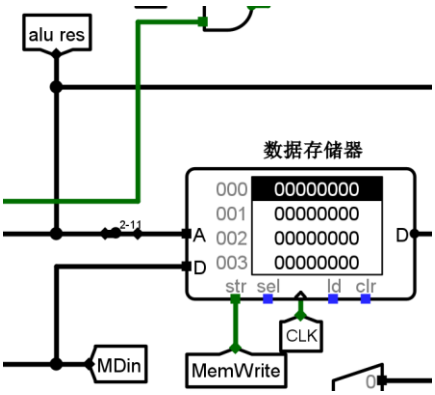

图 3.5 数据存储器（DM）实现

3.1.2 数据通路的实现

数据通路采用简单迭代法实现，先完成一条基础指令（如 R 型指令）的数据通路，然后再在其之上不断增加新的数据通路，按照 2.1.2 节中表 2.3 与附加条件设计，完成全部通路设计。如图 3.6 所示。

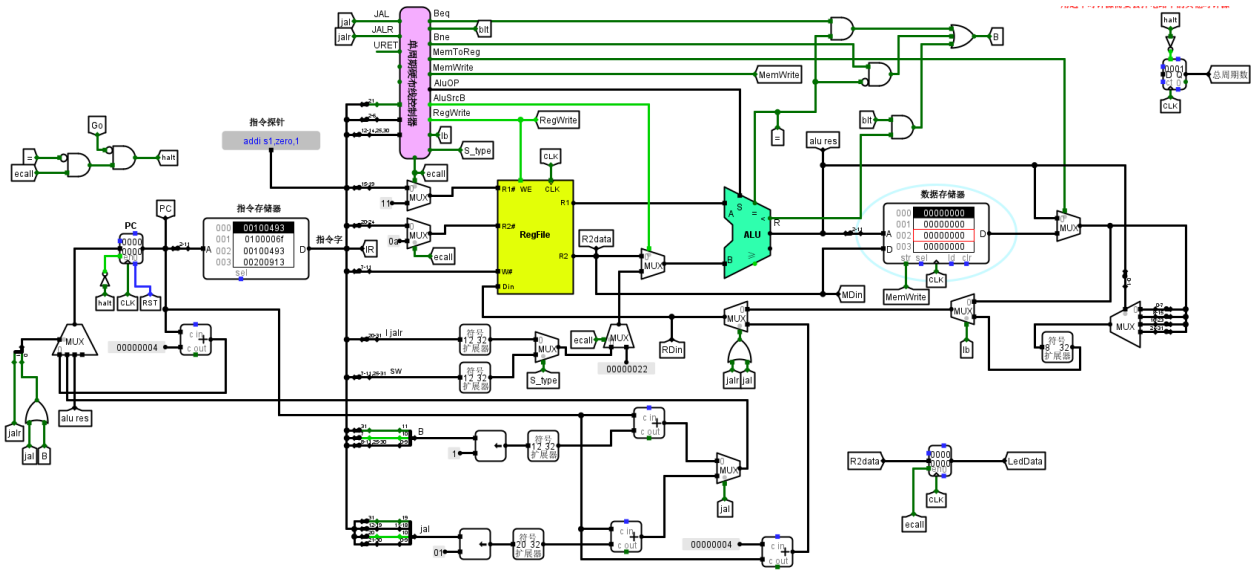

图 3.6 数据通路实现

3.1.3 控制器的实现

按照表 2.5 在硬布线控制器表达式自动生成 excel 表中填表，获取控制信号的表达式，借助 Logisim 的组合逻辑电路分析功能，生成硬布线控制器。为方便自动生成电路，为 ALU_OP 单独用一个电路输出，再将所有控制信号整合输出，如图 3.7 所示。

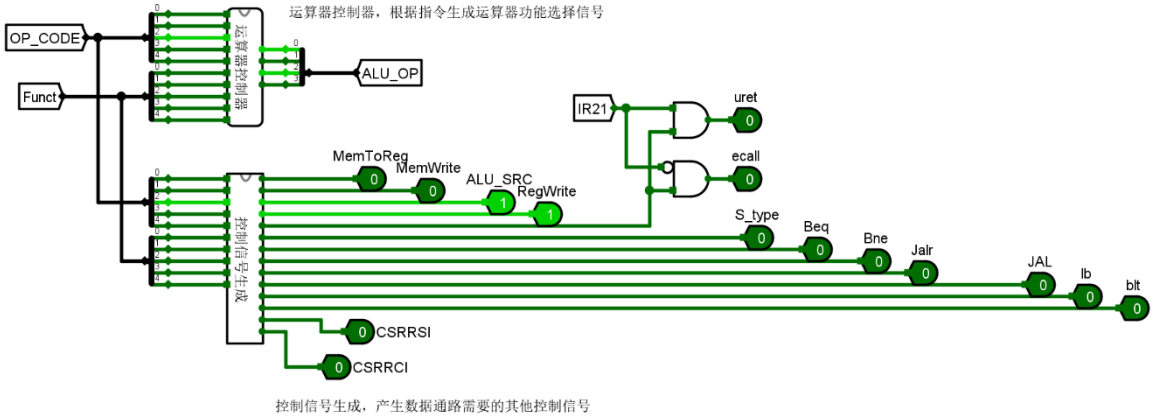

图 3.7 硬布线控制器实现

3.2 中断机制实现

3.2.1 单周期单级中断实现

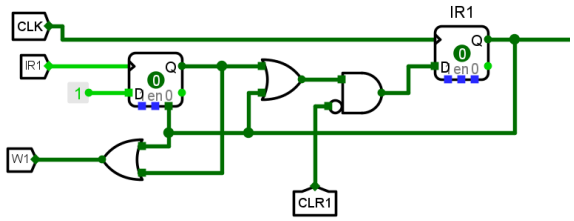

图 3.8 中断信号采样电路（以中断信号 1 为例）

单周期单级中断 CPU 在单周期 CPU 基础上进行增改。

增加中断信号采样电路对中断信号进行收集，由 D 触发器组成的电路实现，如图 3.8 所示；所收集的中断信号通过优先编码器进行解析后通过多路选择器进行中断入口选择，如图 3.9 所示，其中中断入口的地址通过 RARS risc-v 汇编模拟软件获取；同时优先编码器解析的存在中断请求信号传递至中断使能寄存器关中断，如图 3.10 所示，其中将 IE 寄存器的信号取非后和中断请求信号进行与操作，将这个信号作为是否执行中断服务程序的信号为采用的一个小技巧，可以通过此正确控制异常程序计数器 mEPC 和中断采样电路；异常程序计数器 mEPC 通过 32 位寄存器实现保存中断

断点，如图 3.11 所示；uret 中断返回时，中断采样电路信号也应被清除，如图 3.12 所示；主电路 PC 值的选择增加中断入口地址和中断返回地址选项，通过优先编码器和多路选择器选择，如图 3.13 所示。同时，为适配软件，需新增 uret 指令回路，该指令与 ecall 指令仅第 21 位存在不同，可以与 ecall 进行相同解析，然后再对第 21 位进行判断分辨。

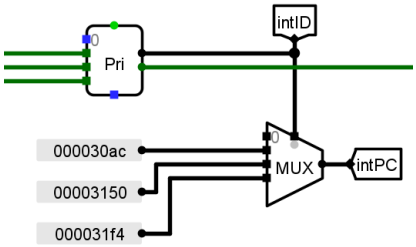

图 3.9 中断入口解析

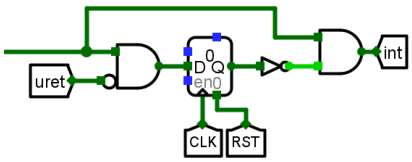

图 3.10 中断使能实现

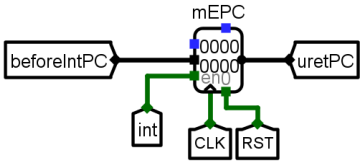

图 3.11 异常程序计数器 mEPC 实现

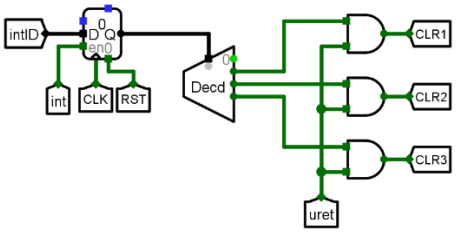

图 3.12 中断采样电路信号清空实现

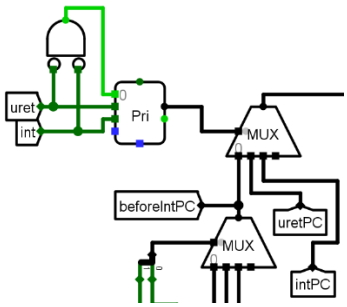

图 3.13 中断入口和中断返回实现

3.2.2 单周期多级中断实现

单周期多级中断在单级中断的基础上增加和修改。

中断使能部分，考虑到高级中断应该中断低级中断，所以有多个中断请求时，应进行中断等级比较，中断使能寄存器也应能够响应关中断指令 CSRRCI、开中断指令 CSRRSI，如图 3.14 所示，其中一个小技巧是将中断返回信号和开中断信号作为中断使能寄存器 IE 的时钟输入而不是数据输入，可以避免 IE 需要与时钟节拍同步的麻烦。断点保存部分，需要保存多个断点，利用计数器设计硬件堆栈实现，如图 3.15 所示。同时，无法中断高级中断的低级中断请求需要进行暂存，如图 3.16 所示。为适配软件部分还应增加 CSRRCI 和 CSRRSI 的指令通路，解析这两个指令的指令字后输出相应信号即可。

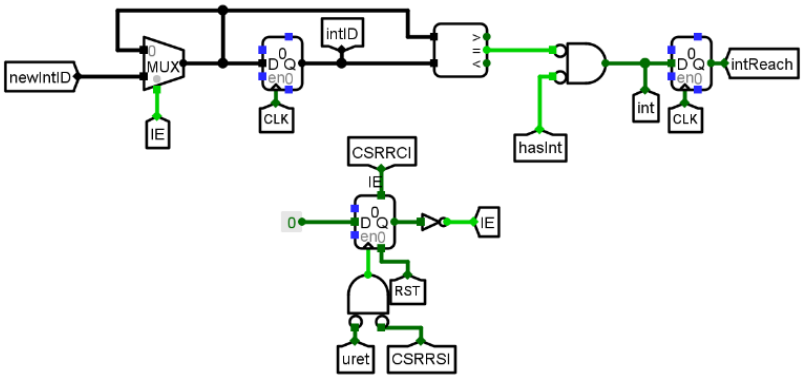

图 3.14 中断使能实现

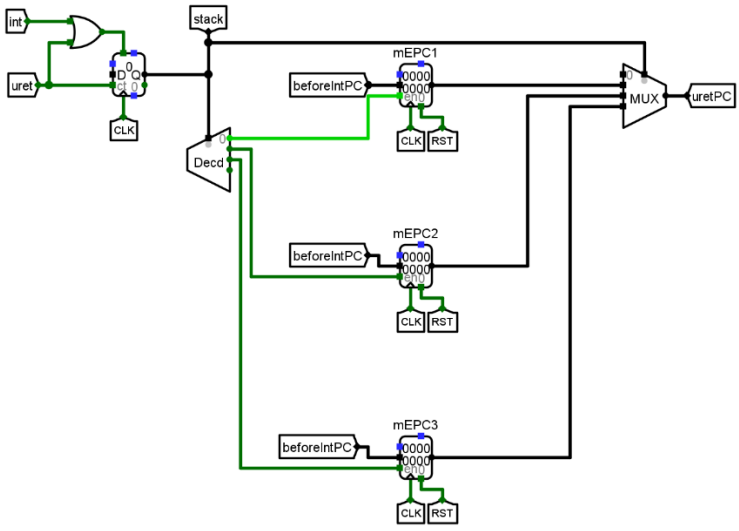

图 3.15 硬件堆栈实现断点保存

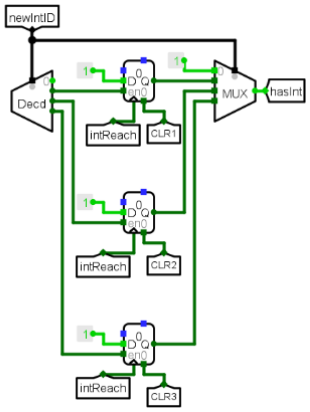

图 3.16 未执行中断保存实现

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口的每项数据都使用一个寄存器暂存，如图 3.17 所示。其中一个小技巧是使用输入 0 来达到同步清零的目的，避免寄存器异步清零带来的毛刺。

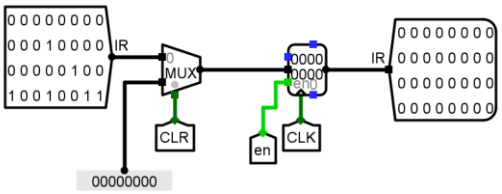

图 3.17 流水接口数据暂存实现（以 IR 为例）

3.3.2 理想流水线实现

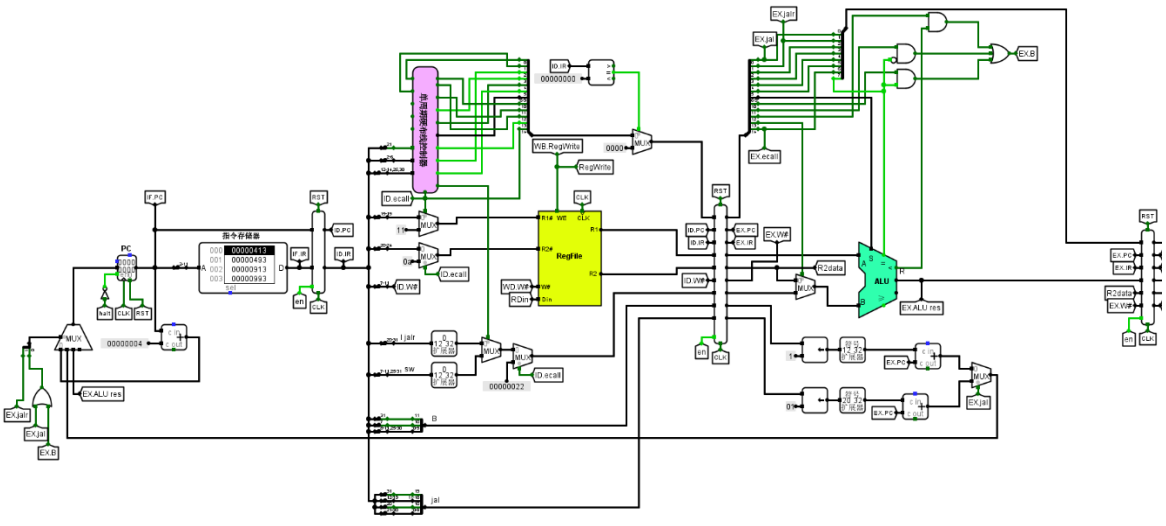

图 3.18 理想流水线（IF、ID、EX 段）

按照 2.3.3 节分割设计，在分割点插入流水接口部件，如图 3.18 和图 3.19 所示。

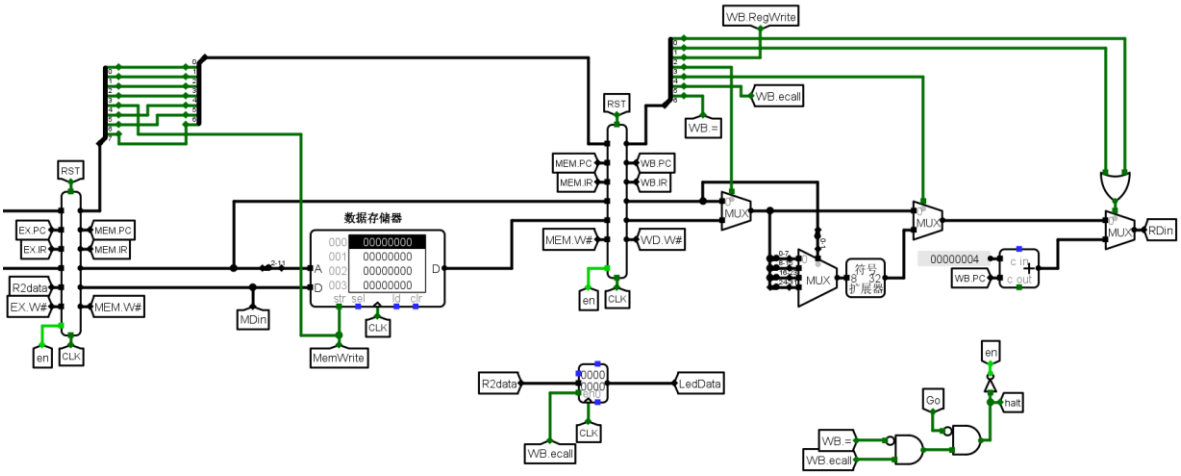

图 3.19 理想流水线（MEM、WB 段）

3.4 气泡式流水线实现

在理想流水线的基础上进行增改。

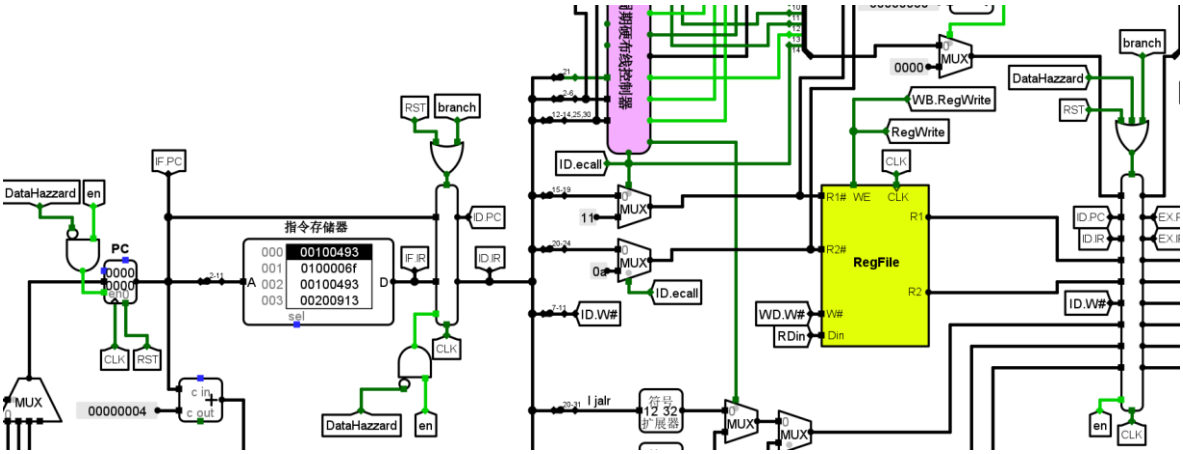

图 3.20 数据冲突处理实现

根据 2.4.2 节 DataHazzard 公式实现数据相关检测逻辑，如图 3.21 所示。其中 r1used 和 r2used 信号的获取借助硬布线控制器表达式自动生成表，如表 3.1 所示，然后使用 Logisim 组合逻辑电路分析自动生成电路。在 ID 段插入数据相关检测逻辑，如图 3.22 所示。根据 2.4.2 节添加数据相关冲突处理，通过对接口部件和 PC 寄存器输入清零信号和使能信号来实现气泡的插入，如图 3.20 所示。

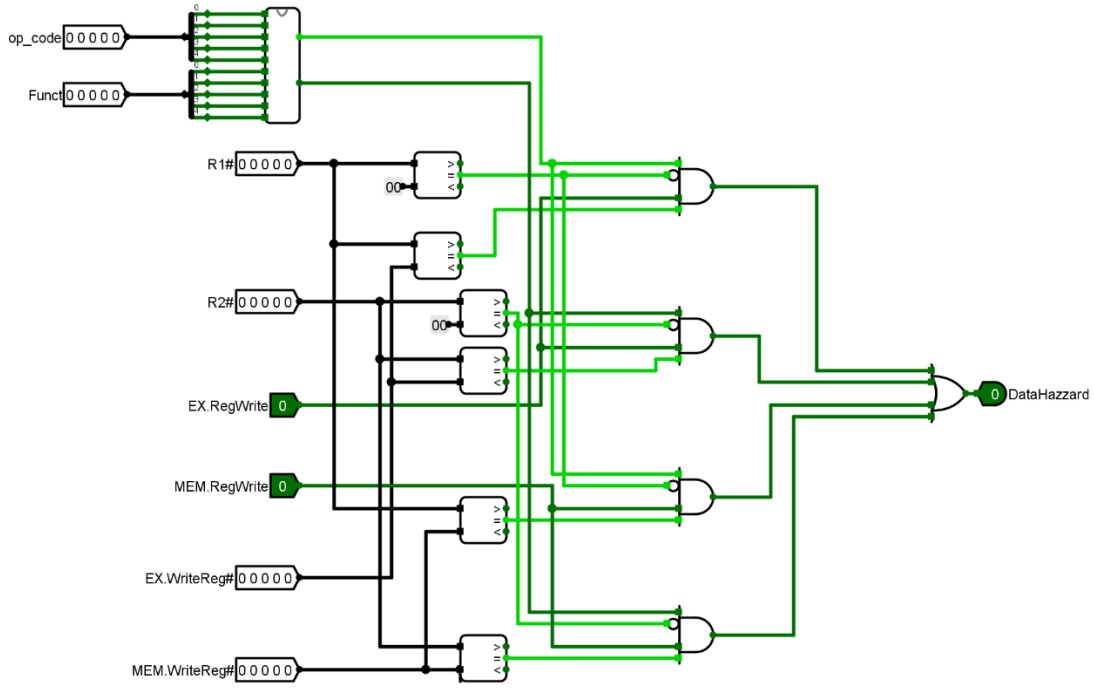

图 3.21 数据相关检测逻辑实现

表 3.1r1used 和 r2used 信号表

指令	r1used	r2used
add	1	1
sub	1	1
and	1	1
or	1	1
slt	1	1
sltu	1	1
addi	1	
andi	1	
ori	1	
xori	1	
slti	1	
slli	1	
srli	1	
srai	1	

指令	r1used	r2used
lw	1	
sw	1	1
ecall	1	1
beq	1	1
bne	1	1
jal		
jalr	1	
sll	1	1
sltiu	1	
lb	1	
blt	1	1

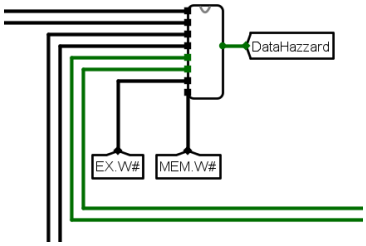

图 3.22 加入数据相关检测逻辑

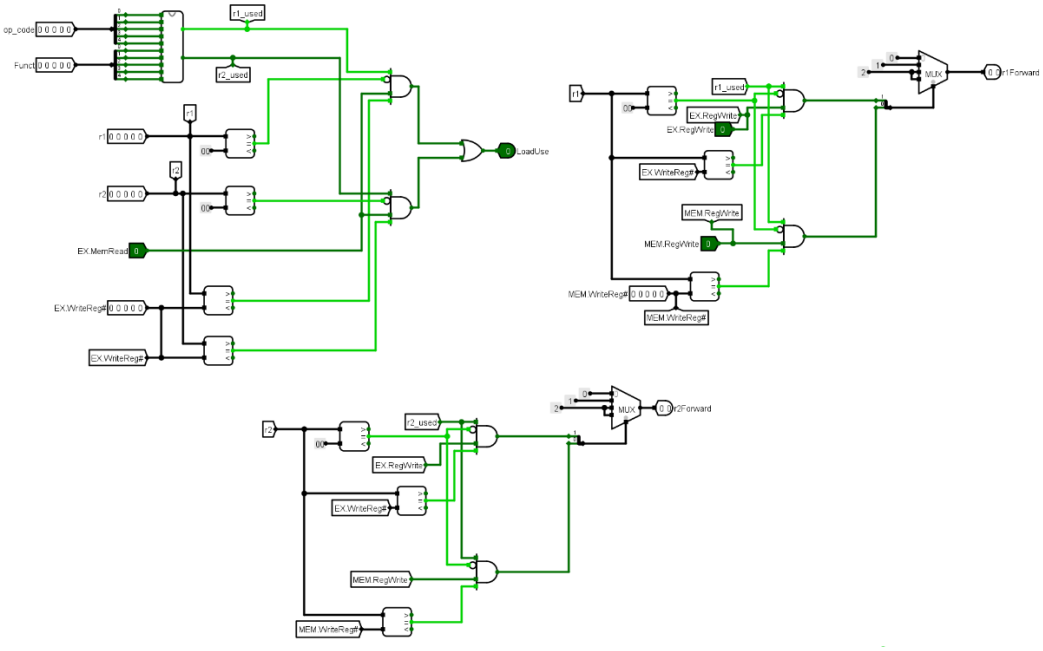

图 3.23 重定向相关处理逻辑实现

3.5 数据转发流水线实现

在理想流水线的基础上进行增改。

根据 2.5.2 节 LoadUse、r1forward、r2forward 公式实现重定向相关处理逻辑，如图 3.23 所示，其中 r1used 与 r2used 输出逻辑重用气泡流水线中所用。将重定向相关逻辑插入 ID 段，如图 3.24 所示。根据 2.5.2 节增加数据重定向通路，如图 3.25 所示。与气泡流水线类似，根据 2.5 节在流水接口和 PC 寄存器的清零端和使能端接入相应 LoadUse 和分支信号，如图 3.26 所示。

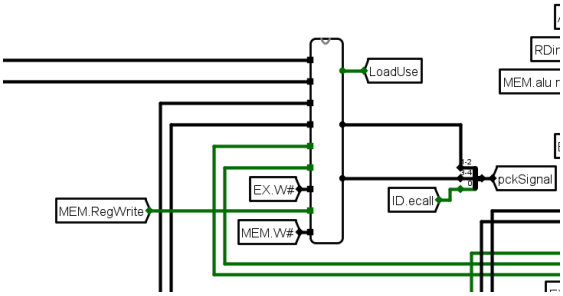

图 3.24 插入重定向相关逻辑

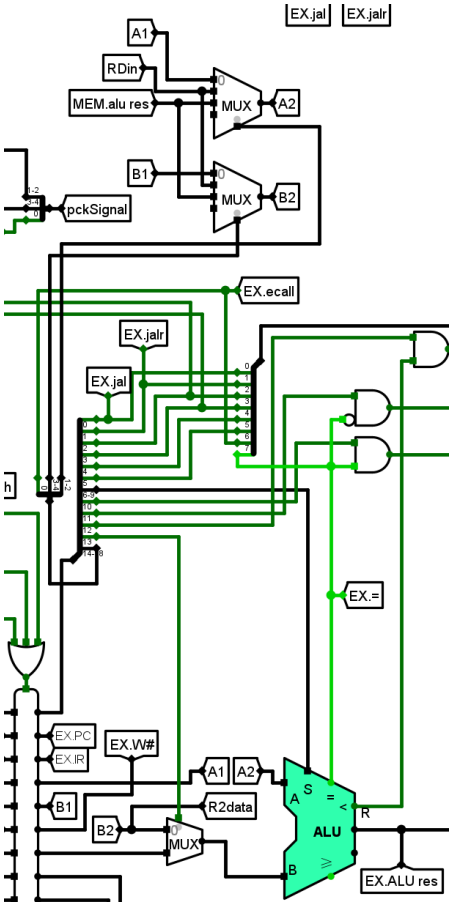

图 3.25 重定向数据通路

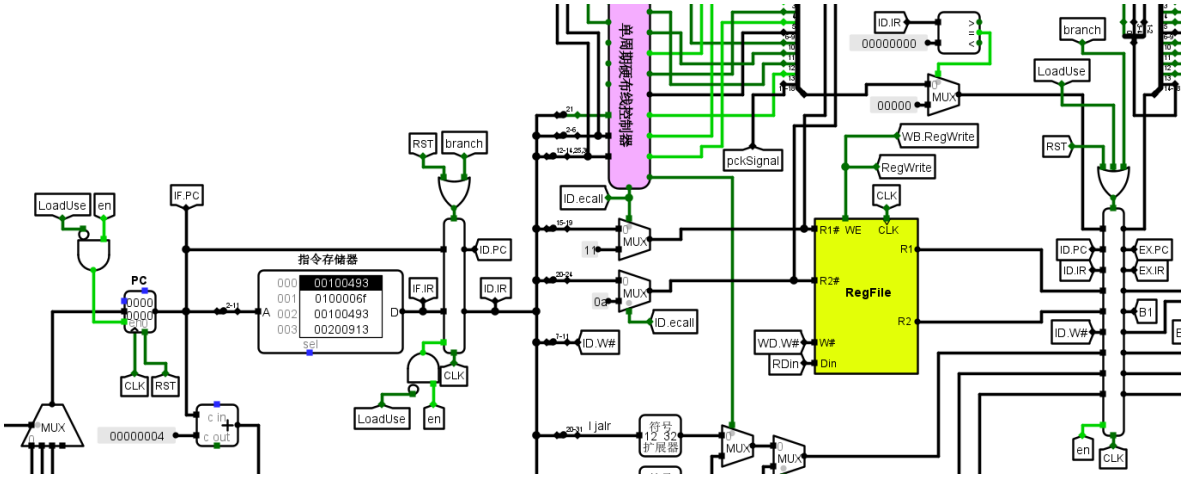

图 3.26 重定向中的气泡插入实现

3.6 动态分支预测机制实现

在重定向流水线基础上进行增改。

首先进行 BTB 设计。BTB 表为 8 行全相联 cache，表项为 valid 位、查找标签、淘汰计数位、分支目标地址、分支预测历史位，如图 3.27 所示。cache 淘汰采用 LRU 算法，使用计数器记录 cache 行未使用周期，当需要淘汰时最先淘汰计数器值最大的 cache 行，并清空计数器，如图 3.29 所示，其中计数器值的比较通过归并排序实现，每两组数据通过如图 3.28 所示比较逻辑比较。分支预测历史位的状态转换通过 JK 触发器实现的同步实现电路完成，并利用计数器和异步置一位将初始状态设置为 11，如图 3.30 所示。

将封装好的 BTB 表插入 IF 段，并更改 PC 选择逻辑，如图 3.31 所示。通过比较器增设跳转正确性判断，如图 3.32 所示，正确性信号与图 3.26 中 branch 信号相似，在检测到预测错误时向相应流水线接口输入清零信号，插入气泡清除错误指令。

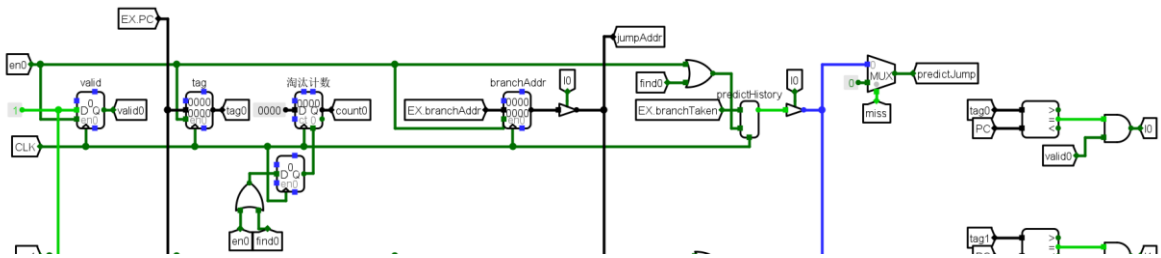

图 3.27 一个 cache 行

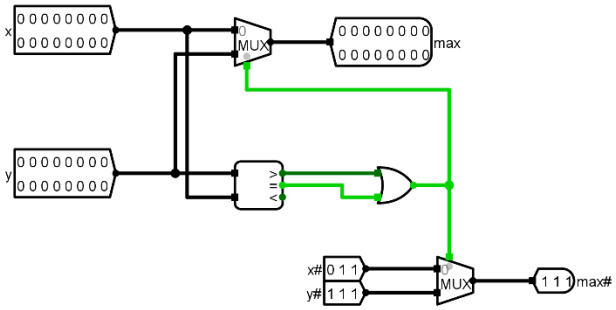

图 3. 28 归并排序所用比较逻辑

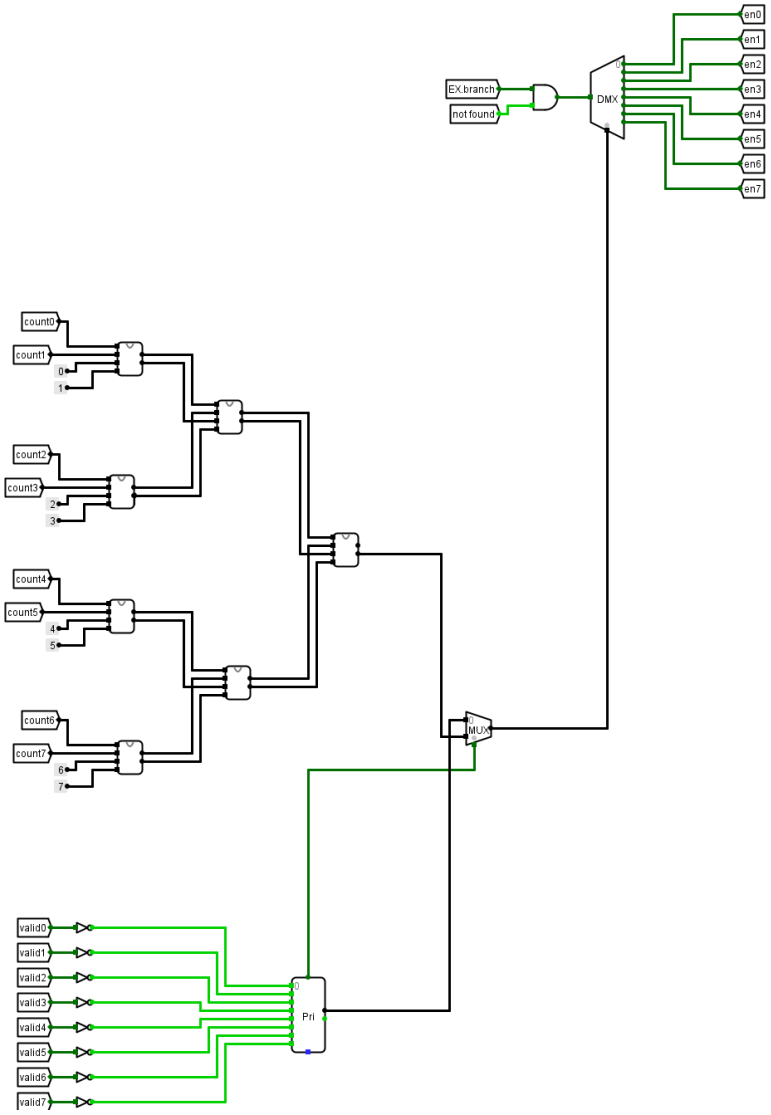

图 3. 29LRU 算法淘汰 cache 行

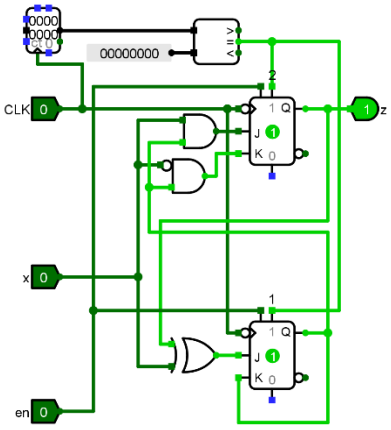

图 3.30 分支历史预测位转换逻辑

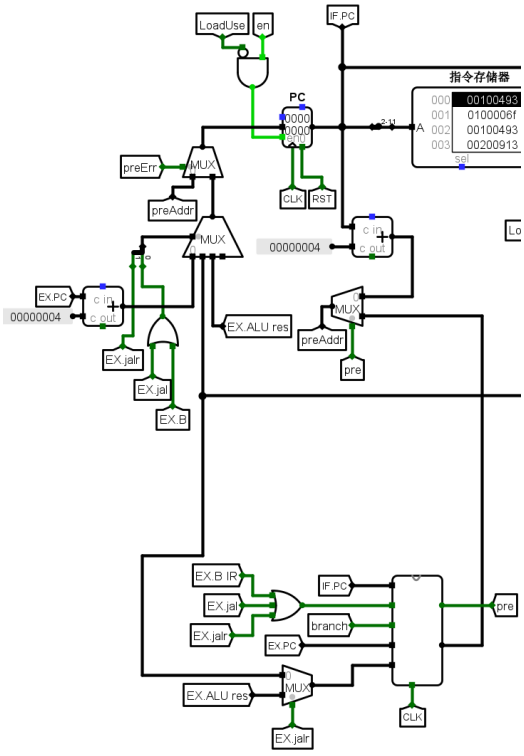

图 3.31 插入 BTB 表

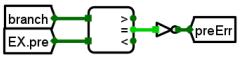

图 3.32 分支预测正确性判断

4 实验过程与调试

4.1 测试用例和功能测试

测试使用 risc-v 汇编程序，并将其在 RARS 中编码成对应的 16 进制文件后加载到指令储存器中执行。程序中对所有指令进行了功能性测试，通过 LED 显示检查程序正确性从而反映出指令通路的正确性。中断功能通过编写相应的中断服务程序和设置中断按钮检查其正确性。

通过头歌平台和线下测试通过。

4.2 性能分析

对于 benchmark 测试程序，单周期 CPU 周期数为 1546，但由于未流水分段，其关键路径时延较流水线 CPU 在真实情况下应长很多；气泡流水线周期数为 3624；重定向流水线周期数为 2297，相比于气泡流水线，由于减少了气泡的插入，周期数明显减少；动态分支预测流水线周期数为 1763，相比于重定向流水线，由于进一步减少了分支时的气泡的插入，周期数进一步减少。

总体来说，通过分析运行耗时因素，针对性进行部件改进设计，达到逐渐提高 CPU 性能的目的。

4.3 主要故障与调试

4.3.1 接口部件同步清零故障

气泡流水线：接口插入气泡问题。

故障现象：在流水接口部件通过同步清零的方式插入气泡时，数据寄存器不能正确地同步清零。

原因分析：最初的实现方案为：在接口内的数据寄存器的清零端接入时钟信号和清零信号的与信号。但此种方法会产生毛刺，无法真正达成时钟同步，导致新取入的指令也被清除。

解决方案：通过多路选择器在收到清零信号的时候对数据寄存器输入 0 的伪清零方式，达到同步清零，如图 3.17 所示。

4.3.2 BTB 淘汰计数故障

BTB 表：淘汰计数错误。

故障现象：BTB 表的淘汰技术器应该在数据更新时清零，但实际运行时清零与读取顺序受不稳定，进而导致后续逻辑错误。

原因分析：淘汰计数器读取和清零同时进行的方式由于毛刺影响而不稳定。

解决方案：在淘汰技术器的清零端通过寄存器将清零信号延迟一个节拍，从而使读取与清零顺序清晰如图 4.1 所示。

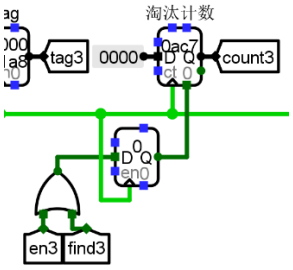

图 4.1 淘汰技术器清零延迟

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 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	学习理想流水线知识，完成流水接口部件的设计和实现。
第四天	完成理想流水线，进行测试与调试。
第五天	学习气泡流水线，完成数据相关检测逻辑，完成气泡流水线，进行测试与调试，完成故障报告。
第六天	学习重定向流水线，完成重定向相关检测逻辑，完成重定向流水线，进行测试与调试。
第七天	学习动态预测流水线，完成 BTB 表的设计与实现。
第八天	完成动态预测流水线，进行测试与调试，完成故障报告。
第九天	复习单周期单级中断知识，完成单级中断设计与实现，并进行测试与调试。

时间	进度
第十天	复习单周期多级中断知识，完成多级中断的设计与实现。

5 团队任务

5.1 项目设计

团队设计一个井字棋小游戏，能支持人人对战（PVP）与人机对战（PVE）两种模式。落子与功能按键均通过 logisim 的按钮实现。

5.2 项目分工

项目分为软件部分和硬件部分。硬件部分分为输入输出部件的设计，和新增指令通路与中断接口。软件部分分为 c++源程序的编写，和 risc-v 汇编程序的编译与硬件适配。每项任务分别由一人完成，一共四人。

5.3 个人任务

本人主要完成由 c++源程序向汇编程序的编译以及硬件适配。

risc-v 汇编借助现有的 risc-v 编译器进行编译生成，但一般的编译器均为现有的 risc-v CPU 设计，而此次课设设计的 CPU 为简单 CPU 且没有操作系统支持，所以在使用编译器编译后还需经过调整，如对全局变量进行手动分配内存；对使用指令进行简化，尽可能使用已实现指令通路的指令。

编译完成后进行联合调试，根据硬件和 c++源程序进行汇编程序调整。

6 设计总结与心得

6.1 课设总结

此次课设进行了 risc-v CPU 的设计与实现，主要作了如下几点工作：

1） 完成单周期 CPU 的设计与设计，支持规定指令和差别扩展指令。

2） 完成理想流水线的设计与实现。

3） 完成气泡流水线的设计与实现。

4） 完成重定向流水线的设计与实现。

5） 完成单周期单级中断 CPU 的设计与实现。

6） 完成单周期多级中断 CPU 的设计与实现。

7） 完成团队任务——井字棋小游戏的设计讨论，实现团队任务的软件部分汇编程序的编译与硬件适配。

6.2 课设心得

通过此次课设，我加深了对 CPU 结构的深入理解，能够自己动手实现简单的 CPU 和利用多种流水线进行性能优化。熟练了使用 logisim 进行硬件线路的设计与实现，加强了对硬件运行过程中毛刺、时延的理解。

此次课设让我感到很有成就感，虽然过程中有很多困难，能够通过自己学习实现 CPU 这样一个复杂的系统还是让人感到满满的充实感和收获感。

对于课程，我建议可以加强团队任务的引导。个人部分的文档都很详尽，但对于团队任务仅是通过历年项目样例难以把握自己团队的准确方向与定位，从而较容易陷入模仿和微创新的陷阱。建议可以对与团队任务可以增加一个基调，如建议使用软件的规模，常用的硬件拓展方向等。

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：周帅君