華中科技大學

2020

计算机组成原理　　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：	CSIE 1701
学　　号：	U201715220
姓　　名：	罗 俊 杰
电　　话：	15327457352
邮　　件：	2061926973@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 LOGISIM 或 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

（1） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（2） 支持教师指定的 4 条扩展指令；

（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

#	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	SUBU	无符号减	扩展指令 1
29	XOR	异或	扩展指令 2
30	SB	存储字节	扩展指令 3
31	BLEZ	小于等于 0 转移	扩展指令 4

2 总体方案设计

2.1 单周期 CPU24+4

本阶段实现基于硬布线控制器的 MIPS 单周期 CPU，支持基本 24 条指令和 4 条扩展指令（中断相关指令在后续阶段完成），总体结构图如图 2.1 所示。

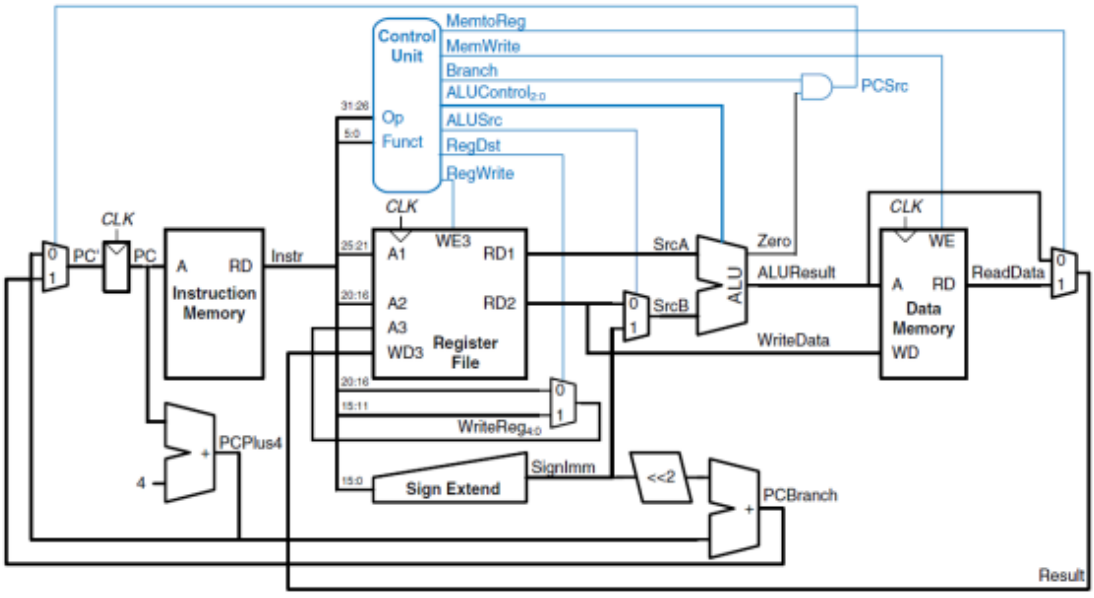

图 2.1 总体结构图

2.1.1 数据通路的设计

数据通路主要分为控制器通路、指令存储器通路、ALU 计算通路和数据存储器通路四个部分。下面从四个方面介绍具体的设计思路。

1. 三种指令的数据通路

MIPS 有 R、I、J 型三种指令类型，这些指令的执行需要用到公共的器件如 PC、ALU 等，但我们更关注其执行过程的不同的地方。为了完成各自的功能，需要设计专门的数据通路，比如为跳转指令生成地址计算的数据通路等。由于我们实现的是单周期 CPU，所以需要在关键器件前加多路选择器以满足不同的数据来源需求。具体的，需要在 PC、寄存器文件、ALU 等部件前加入多路选择器，选择信号由控制器发出。

2. 关于跳转指令

跳转指令分为条件跳转和无条件跳转。前者为 I 型指令，包括 BEQ、BNE 和 BLEZ

等；后者为 J 型指令，包括 JMP、JR 和 JAL。他们的数据通路只需在已有通路的基础上加入地址计算与地址选择的部分即可。

3. 四条拓展指令

拓展指令分别为 SUBU、XOR、BLEZ 和 SB。前二者属于 R 型指令，R 型数据通路是通用的；为实现 BLEZ 指令，我们用额外的比较器比较 RS 和 0 的大小以确定是否跳转；SB 指令根据 RS(OFFSET)计算地址的低二位决定 RT 低 8 位的字内偏移，然后和从数据寄存器读出来的原有的数据结合，写入到目标地址，完成字节的写入功能，之所以能这么做是因为从数据寄存器读出数据不需要时钟沿触发。

4. 关于 Syscall 指令

该指令完成简易系统调用的功能，需要设计专门的通路完成 LED 显示、停机等功能。

2.1.2 控制器的设计

控制器采用硬布线的方法、利用控制信号自动生成表格实现。

控制器根据指令的 OP、FUNC 字段，通过组合逻辑输出该指令执行所需要的所有控制信号，这些信号分为三类：多路选择器的选择信号，器件的写信号和 ALU 的功能选择信号。大部分信号的功能和产生条件在指导书已有详细说明，本报告不赘述，只是介绍自己增加的几个信号：

BLEZ：当前指令是否为 BLEZ 指令，是则为 1 否则为 0；

SB：当前指令是否为 SB 指令，是则为 1 否则为 0；

SignedExt：立即数是否需要符号拓展，是则为 1 否则为 0。

2.2 单级中断机制设计

2.2.1 总体设计

单级中断采用一个寄存器 EPC 保留返回地址，用一个 D 触发器控制中断使能信号 IE。指令方面只需要增加对 ERET 指令的支持即可。

在一条指令的上升沿后，检测到中断请求，立刻执行中断隐指令：将 PC 的值保存到 RET_PC 中，将中断使能 IE 置为 0，并通过该中断号对应的入口地址 INT_PC

取得该中断程序的第一条指令，那么下一个时钟上升沿就开始执行中断服务程序了。

执行到该中断程序的 ERET 指令时，我们需要将返回地址 RET_PC 置于 PC 中，需要清空该中断的请求信号，需要将 IE 的值置为 1，这些都是同步 ERET 的上升沿的。此后，CPU 开始执行被中断的主程序（如果当前没有别的中断请求）。

2.2.2 硬件设计

我们需要设计电路检测并保存所有的中断请求，执行优先级最高的中断并保留其中断号，以便返回时能够清空对应的中断请求信息，这些可以通过优先编码和优先解码器以及寄存器来实现；我们需要设计电路让 CPU 能够从主程序切换到中断程序，又从中断程序返回到正确的主程序地址，这些工作依赖中断请求信号 INT、中断使能信号 IE 和返回指令 ERET，并使用寄存器等器件来完成。

2.2.3 软件设计

软件方面只需增加对 ERET 指令的支持，控制器在检测到该指令后，给出“ERET”的信号，其他中断相关电路根据该信号完成中断返回的工作。

2.3 多级中断机制设计

2.3.1 总体设计

在单级中断的基础上实现多级中断，需要考虑中断程序被优先级更高的中断信号所中断的情况。我们需要用电路比较新的中断和当前中断程序的优先级，以决定是否需要进入新的中断程序。我们需要设计硬件栈来保存返回地址，以支持嵌套中断。我们还需要考虑进入中断服务程序后保护现场和恢复现场的原子性，在进入中断后关中断，保护现场后用指令 MFC0 开中断；在恢复现场前用指令 MTC0 关中断，执行指令 ERET 时同步开中断。

2.3.2 硬件设计

为比较新的中断请求和当前中断程序的优先级，需要设计电路保存正在运行（包括被中断的）中断程序号，可以用三个寄存器来保存三个中断程序的运行状态（是否运行）。用硬件栈保存返回地址，在进入新的中断程序前保存当前程序的返回地址，

在中断程序执行 ERET 时同步恢复返回地址到寄存器。

2.3.3 软件设计

较单级中断，需要新增对指令 MFC0 和 MTC0 的支持，此二者即软件开关中断的关键。当控制器检测到这两条指令的时候，给出“MFC0”和“MTC0”的信号，以便多级中断电路能及时开关中断。编写测试程序时也应注意，保护现场后执行 MFC0 开中断，恢复现场之前执行指令 MTC0 关中断。

2.4 理想流水 CPU 设计

2.4.1 总体设计

流水线由取指、译码、执行、访存和写回五段组成，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”。

取值阶段根据 PC 获取指令寄存器中的指令，向后传递；译码阶段由指令生成控制器的控制信号和寄存器操作数，向后传递；执行阶段利用 ALU 进行运算，或计算分支地址等；访存阶段进行向数据存储器存数据或从中读数据；写回阶段将 ALU 计算结果或从内存读出的数据写入到寄存器文件。

2.4.2 流水接口部件设计

流水接口部件用于分割流水阶段，暂存数据并向下一阶段提供数据。流水接口必须支持同步清零，使能控制。每一个接口部件都向后传递 PC 和 IR，方便流水线的实现。根据各阶段的功能特点增加接口部件的接口，ID/EX 接口最为复杂，随着流水的深入，接口数将减少。

2.4.3 理想流水线设计

跳转指令在 EX 阶段完成地址计算，流水也在 EX 阶段完成跳转。JAL 指令既有跳转功能，又要写寄存器文件，考虑到其数据相关性小，所以将其跳转安排在 EX 阶段而写寄存器文件安排在写回阶段。

写回阶段对寄存器文件的操作的数据来源一定是流水最后阶段，而不是当前的译码阶段。寄存器文件下降沿触发，以解决译码和写回阶段可能存在的结构冲突。

对于 Syscall 指令，若是停机，则传到 WB 阶段再停机，停机操作 PC 和流水接口的使能；为操作方便，若是显示，则安排在 ID 阶段，流水只传输是否停机的信号。

2.5 气泡流水 CPU 设计

2.5.1 总体设计

气泡流水线用流水阻塞和插入气泡的方式解决分支相关和数据相关。

对于结构相关，我们采用哈弗结构解决取指令和取数据的争用主存问题， 另加 PC、分支地址计算的加法器来解决 ALU 争用的问题。

对于分支相关，因为分支在 IE 段执行，我们只需在 IE 段检测到成功的分支信号后在 IF/ID 和 ID/EX 同步插入气泡即可。

对于数据相关，若是 ID 与 WB 相关，则将寄存器文件设置为下降沿触发来解决 ID 取操作数和 WB 写回的相关；若是 ID 与 MEM 相关，则暂停 PC 和 IF/ID，在 ID/EX 插入一个气泡即可；若是 ID 与 EX 相关，则采取 ID 与 MEM 相关时相同的解决方式，最后会变为 ID 与 WB 的相关。

2.5.2 硬件设计

在 EX 段增加跳转成功检测逻辑；在 ID 段增加数据相关检测逻辑，包括 ID 段当前所使用的源寄存器和 EX、MEM 使用的写寄存器的检测。

2.6 重定向流水 CPU 设计

2.6.1 总体设计

在气泡流水的基础上实现重定向，就是用旁路的方法处理数据相关的问题。

我们在 EX 阶段完成所有的重定向工作，也就是重定向进入 EX 的 R1 和 R2，这样，无论是 R1 和 R2 作为 ALU 的操作数，还是 R2 向后传递作为 MEM 阶段写入数据存储器的值，都一次性得到了正确的值。

我们需要在 ID 阶段检测相关情况，将得出的重定向选择信号传递到 EX 阶段，所以接口部件 ID/EX 需要增加两个流水接口。

在 ID 检测相关时，不仅要给出是否相关，还要给出是否存在 Load_Use 相关，还要给出 R1、R2 具体和 EX、MEM 哪一阶段相关，以为 Load_Use 相关插入气泡，为

其他相关选择正确的重定向来源。

另外需要注意一个细节问题，关于 Syscall：之前气泡流水在 ID 进行显示，在 WB 停机，所以在 ID 就进行了寄存器$v0 的判断，要在 ID 就重定向$v0。为了统一，将 SYSCALL 的显示推迟到 EX。

2.7 基于重定向流水的单级中断机制设计

2.7.1 总体设计

在重定向流水和单周期 CPU 单级中断的基础上实现流水中断，既利用了单周期单级中断的特点，又有新的涉及流水的问题需要考虑。

1. 中断陷入

和单周期单级中断一样，中断来临，当主程序的上升沿到来后，会触发 INT 上升沿，也就是在中断程序的第一个时钟周期内，会形成一个高电平信号 INT_Level，该信号随着第一个中断上升沿的到来而消失。

INT_Level 为 1 时，将 ID/EX、EX/MEM 设置同步清零信号，PC 选择中断地址 INT_PC，同时，将 PC 的跳转选择信号选择中断入口地址+4。如此第一个中断周期内，IF 的原 PC 无效，ID 指令无效，EX 指令无效（不管是跳转还是非跳转指令），随着第一个中断上升沿的到来，将插入两个气泡。

但是，对于连续中断的情况，如果前一个中断的 ERET 仍然处于 ID 阶段，则不能插入气泡，因为要保证恢复现场的原子性；若 ERET 刚处于 ID 或 EX 阶段，此时进入新的中断，则保持最初的返回地址不变。

保存 EX.PC。因为主程序的最后的上升沿到来后，EX 的指令已经更新，由于 INT 上升沿的延迟较大，所以 EX.PC 送到 EPC 寄存器后，INT 上升沿才触发寄存器，正确保存了断点。若 EX 为气泡，则考虑三种情况，即前一个周期存在 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。所以，需要选择正确的断点保存。

关中断。INT 为高电平后立马将 IE 寄存器置 0，这一操作又使得 INT 变为低电平，但是由于 INT 上升沿带来的 INT_Level 高电平会一直持续到第一个中断上升沿结束。

2. 中断返回

ERET 指令 IF 执行，不需要再插入一个气泡，IF 段的地址选择 RET_PC，当后续中断立即来临时，注意当前中断恢复现场的原子性。

同时，ERET 开中断，清除当前中断请求，为下一个中断做准备。

2.8 基于重定向流水的动态分支预测机制设计

2.8.1 总体设计

本阶段基于重定向流水实现动态分支预测机制，动态分支预测根据跳转指令的历史跳转信息预测下一次是否要跳转。

利用八路全相连映射 CACHE 实现 BHT，在 IF 阶段以 PC 为索引查找分支历史记录，如果命中则根据状态位决定下一条指令是取 PC+4 还是目标跳转地址，这部分工作是和 IF 并行的。还需要设计电路以在分支预测失败时能够返回分支预测点执行另一条分支路径。

在 EX 阶段要根据真实跳转信息和该指令处于 IF 阶段的预测信息判断是否预测成功，并准备更新 BHT 中对应行的状态信息，如果预测失败需要在流水插入气泡并在 IF 阶段开始执行另一条路径。如果当前跳转指令缺失，则需要准备在 BHT 中插入新的跳转指令的信息，如果 BHT 需要替换，采用 LRU 策略。

所以，BHT 需要实现三套逻辑。其一，IF 的读逻辑；其二 EX 的插入（替换）逻辑；其三，EX 阶段的更新逻辑（主要是更新对应行的状态位）。

2.9 团队项目设计

2.9.1 项目内容与特色

小组实现一个走迷宫的游戏。用32*32 的 LED 点阵作为游戏屏幕，主人公要从(0, 0)走到(31, 31)。 游戏开始是欢迎界面，可以选择两个不同的关卡，选择后载入相应的地图。 地图中间亮的灯表示墙壁，是走不通的，如果撞墙的话会有警告音。 当主人公走到终点，屏幕会显示 SUCCESS 字样，并响起小星星的音乐。

该项目实现的是大家很熟悉的小游戏，既有一定的趣味性，又很好的考验了团队成员利用所学内容解决实际问题的能力和交流协作的能力。

2.9.2 工程评估，选题原因，可行性分析

实现“走迷宫”的项目需要从硬件和软件两个方面进行，还要考虑软硬件协同的问题，在小组成员的精诚协作下，预估可以在 2 至 3 天时间完成该项目。之所以选择这个项目，其一是团队成员对“走迷宫”小游戏感兴趣，其二是我们有信心有能力在预定的时间内完成该项目。在已经实现的单周期单级中断 CPU 加以适当的改造，编写相关程序，设计音乐播放文件和电路也不是难事。

1. 硬件方面

   主控 CPU：负责游戏的主体逻辑

   音乐 DMA：负责播放结束的小星星音乐

   警告蜂鸣器及其寄存器：人物撞墙时发出警告音，寄存器只需存放频率，音量大小是常量

   音乐蜂鸣器及其寄存器：播放小星星音乐，寄存器只需存放频率，音量大小是常量

   六路中断逻辑：四个中断是上下左右，两个中断是关卡选择

   显示逻辑：包括 32 个 32 位寄存器、对应更新线路和 LED 点阵 。实现的时候只需要在将地图及人物数据写入内存的时候同时写入到备份的 32 个 32 位寄存器中（以便能在一个时钟周期内全部读出），需要更新 LED 点阵时调用 SYSCALL 指令，将备份寄存器中的数据更新到控制点阵输出的寄存器中即可。

2. 软件方面

   主控程序的总体逻辑：首先，载入欢迎界面，并提示 2 个关卡；然后进入空操作死循环等待关卡选择；当用户选择关卡（按 1 或 2 按钮）后，视情况进入2 个关卡中断服务程序中的一个，该程序会初始化屏幕界面，再次进入空操作死循环等待上下左右操作；如果有上下左右按键对应的中断，那就去执行相应的中断服务程序；上下左右中断服务程序的最后会检查主人公的位置是否是终点，如果是，就跳到游戏结束程序段 游戏结束程序会在屏幕上显示 SUCCESS 字样，并触发音乐 CPU 播放音乐

   音乐文件：将《小星星》的旋律编写成 16 进制文件存入 ROM 中，需要播放时根据 PC 取出送到蜂鸣器。

   根据以上分析，我们的项目在现实条件下是完全可行的。

3 详细设计与实现

3.1 单周期 CPU24+4

3.1.1 数据通路的实现

整体数据通路如下图所示。

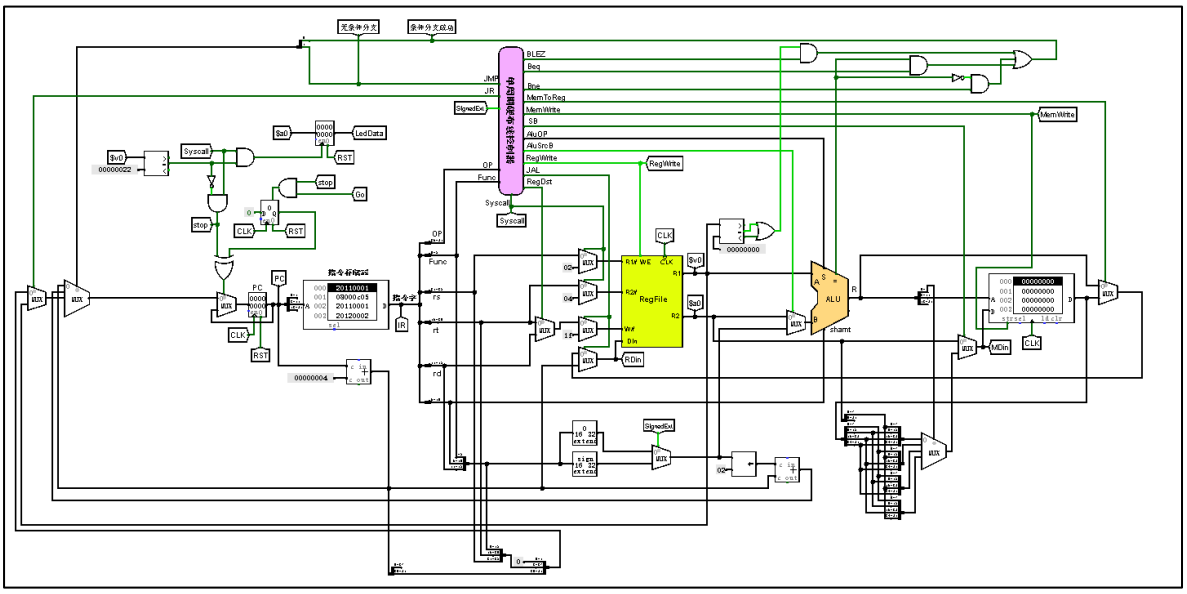

图 3.1 整体数据通路

具体的，Syscall 相关的数据通路如下所示。

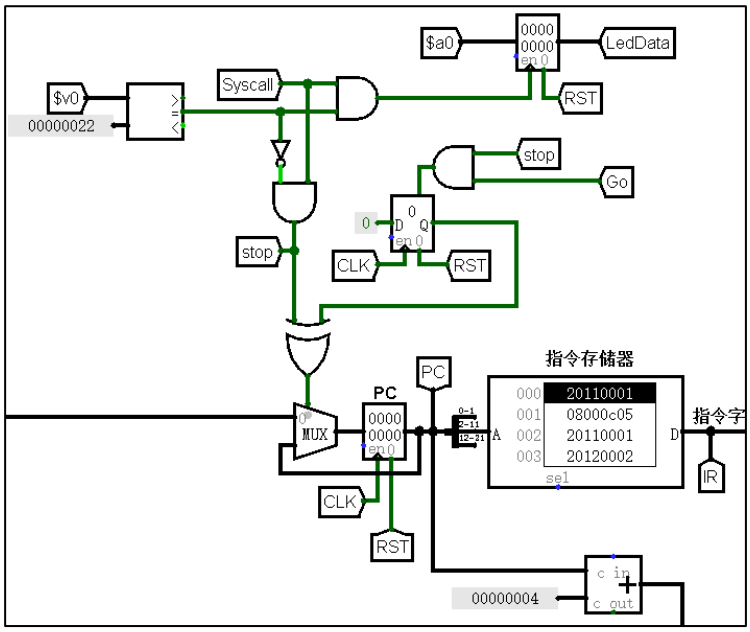

图 3.2 Syscall 相关通路

指令 SB 相关数据通路如下所示。左下部的多路选择器用来选择各种字内偏移情

况下的待写入数据。

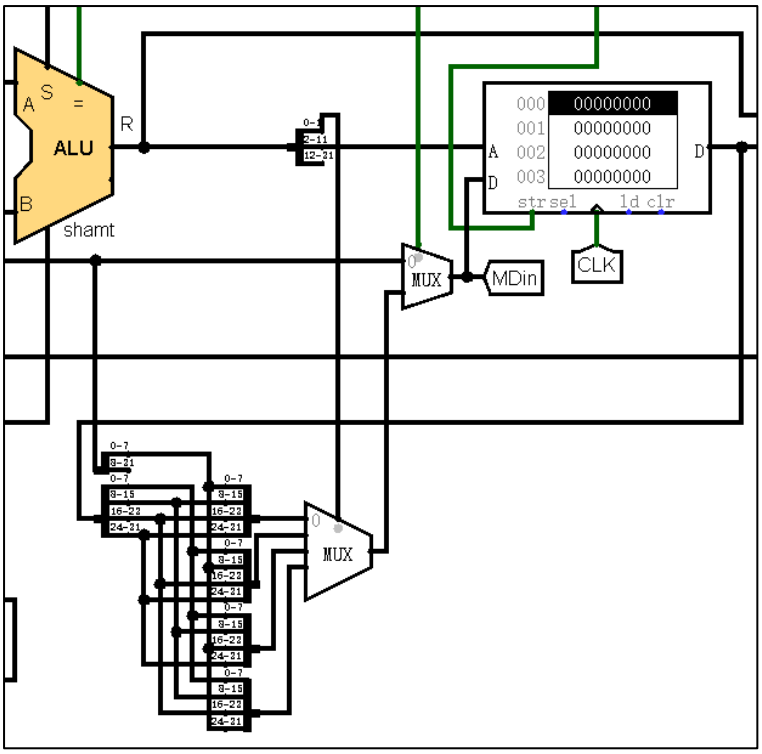

图 3.3 Syscall 相关通路

3.1.2 控制器的实现

根据各条指令的功能，填写控制信号自动生成表格，其部分内容如下所示。

#	指令	OpCode (十进制)	FUNCT (十进制)	OP5	OP4	OP3	OP2	OP1	OP0	F5	F4	F3	F2	F1	F0	ALU_OP	S3	S2	S1	S0	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst
1	SLL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				1			1
2	SRA	0	3	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1				1			1
3	SRL	0	2	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	1	0				1			1
4	ADD	0	32	0	0	0	0	0	0	1	0	0	0	0	0	5	0	1	0	1				1			1
5	ADDU	0	33	0	0	0	0	0	0	1	0	0	0	0	1	5	0	1	0	1				1			1
6	SUB	0	34	0	0	0	0	0	0	1	0	0	0	1	0	6	0	1	1	0				1			1
7	AND	0	36	0	0	0	0	0	0	1	0	0	1	0	0	7	0	1	1	1				1			1
8	OR	0	37	0	0	0	0	0	0	1	0	0	1	0	1	8	1	0	0	0				1			1
9	NOR	0	39	0	0	0	0	0	0	1	0	0	1	1	1	10	1	0	1	0				1			1
10	SLT	0	42	0	0	0	0	0	0	1	0	1	0	1	0	11	1	0	1	1				1			1
11	SLTU	0	43	0	0	0	0	0	0	1	0	1	0	1	1	12	1	1	0	0				1			1
12	JR	0	8	0	0	0	0	0	0	0	0	1	0	0	0	x	X	X	X	X							
13	SYSCALL	0	12	0	0	0	0	0	0	0	0	1	1	0	0	x	X	X	X	X					1		
14	J	2	X	0	0	0	0	1	0							x	X	X	X	X							
15	JAL	3	X	0	0	0	0	1	1							x	X	X	X	X				1			
16	BEQ	4	X	0	0	0	1	0	0							x	X	X	X	X						1	
17	BNE	5	X	0	0	0	1	0	1							x	X	X	X	X						1	
18	ADDI	8	X	0	0	1	0	0	0							5	0	1	0	1			1	1		1	
19	ANDI	12	X	0	0	1	1	0	0							7	0	1	1	1			1	1			
20	ADDIU	9	X	0	0	1	0	0	1							5	0	1	0	1			1	1		1	
21	SLTI	10	X	0	0	1	0	1	0							11	1	0	1	1			1	1		1	
22	ORI	13	X	0	0	1	1	0	1							8	1	0	0	0			1	1			
23	LW	35	X	1	0	0	0	1	1							5	0	1	0	1	1		1	1		1	

输出信号情况
为1时填1，其他不填！

真值表 | 控制信号表达式生成 | 运算器规格 | 控制信号产生条件

图 3.4 各指令所需控制信号

其完整内容请查看附属文件。

最后，实现的控制器其内部框架如下图所示。

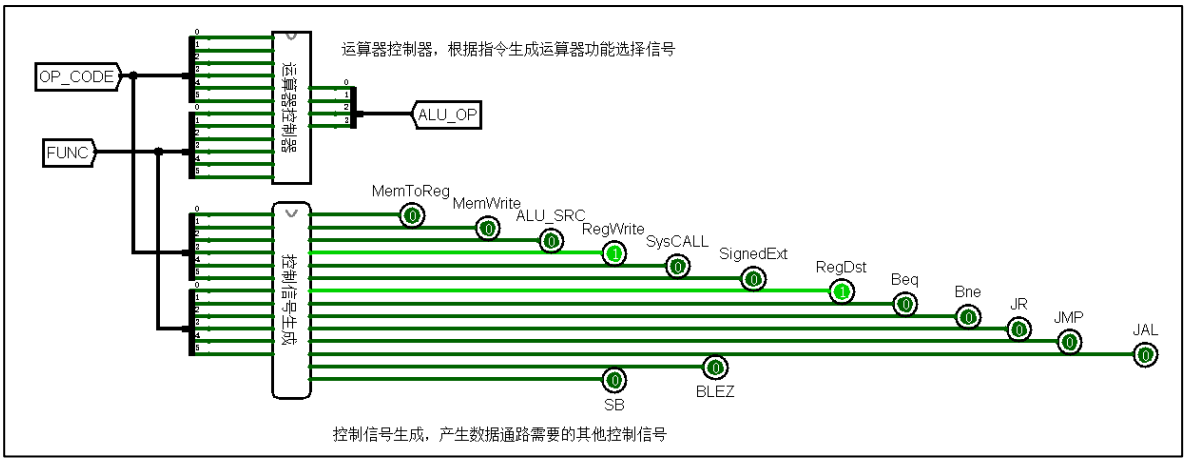

图 3.1 控制器内部框架

3.2 单级中断机制实现

3.2.1 中断的检测、识别、现场保护与返回

实现的电路图如下所示。

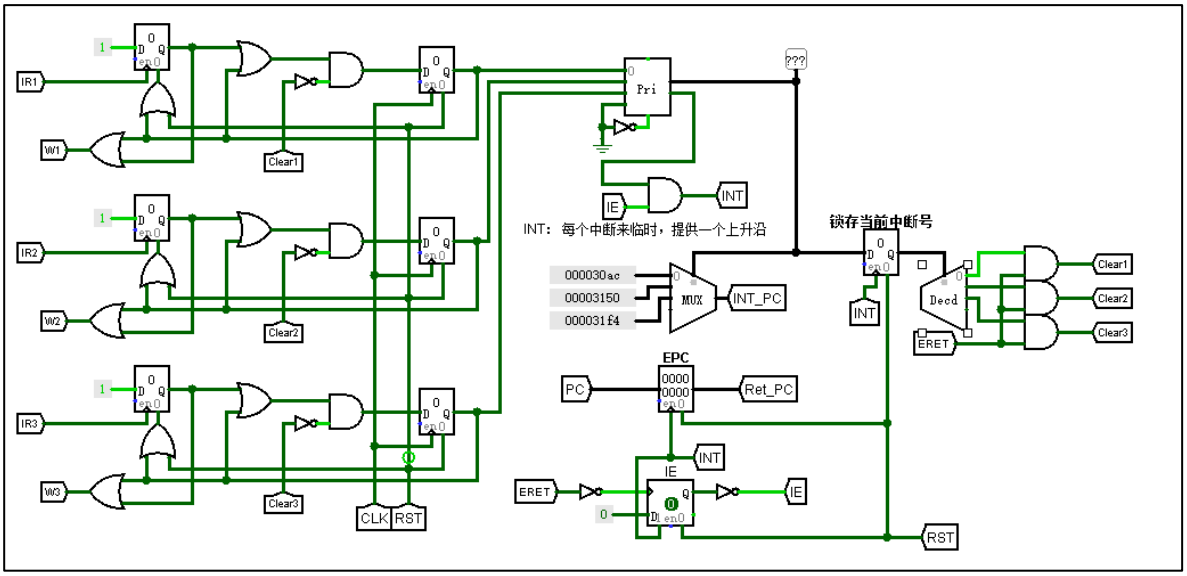

图 3.6 单级中断服务主要电路实现

其中，左边部分负责检测并保存所有的中断请求，并在中断返回的时候清空对应的中断信号；右上部分负责选择最高优先级的中断号并保存，将其地址送入 INT_PC 中，并在返回的时候生成 CLEAR 信号；右下部分负责保存返回地址到 RET_PC，并在执行某个中断程序时关中断，中断返回后开中断。特别注意 INT 信号，在每个中断来临时都提供一个短暂而稳定高电平（我们利用的是这个高电平的上升沿），即使多个中断同时发生，CPU 也会按照这些中断的优先级依次执行。

3.2.2 CPU 地址的切换

实现的电路图如下所示。

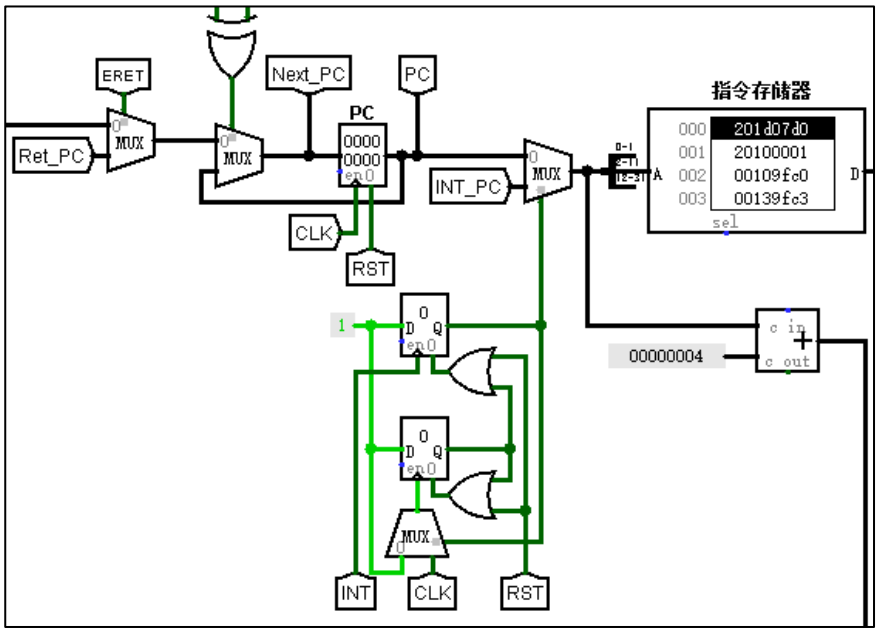

图 3.7 单级中断地址切换

其中，中下部分的电路负责在每一个 INT 的上升沿后维持一段高电平，直到下一个上升沿的到来，这个高电平用于选择中断的入口地址 INT_PC 为下一条指令的地址。返回的时候执行 ERET 指令，该指令的上升沿后，返回地址 RET_PC 被设置到了 PC 中，下一个上升沿将返回主程序执行。

3.3 多级中断机制实现

3.3.1 多级中断的检测和识别

多级中断的检测和识别电路如下所示。

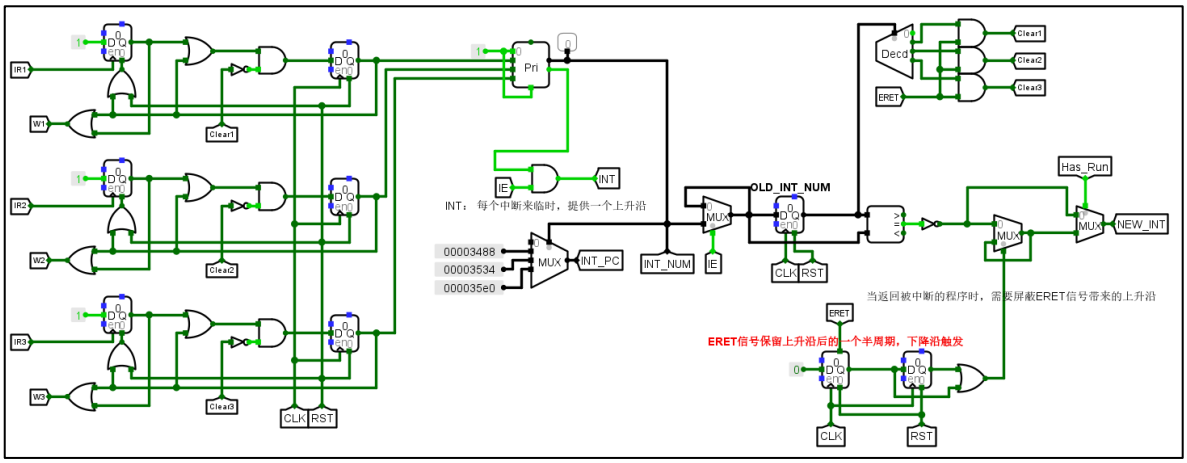

图 3.8 多级中断检测识别

在单级中断的基础上，增加了新的中断号 INT_NUM 和当前运行中断号 OLD_INT_NUM 的优先级比较电路，主程序的优先级最低。当进入新的中断程序时，NEW_INT 会提供一个上升沿，就向单级中断中 INT 会提供一个上升沿一样。

为支持嵌套中断，我们设计如下硬件栈保存多个返回地址。

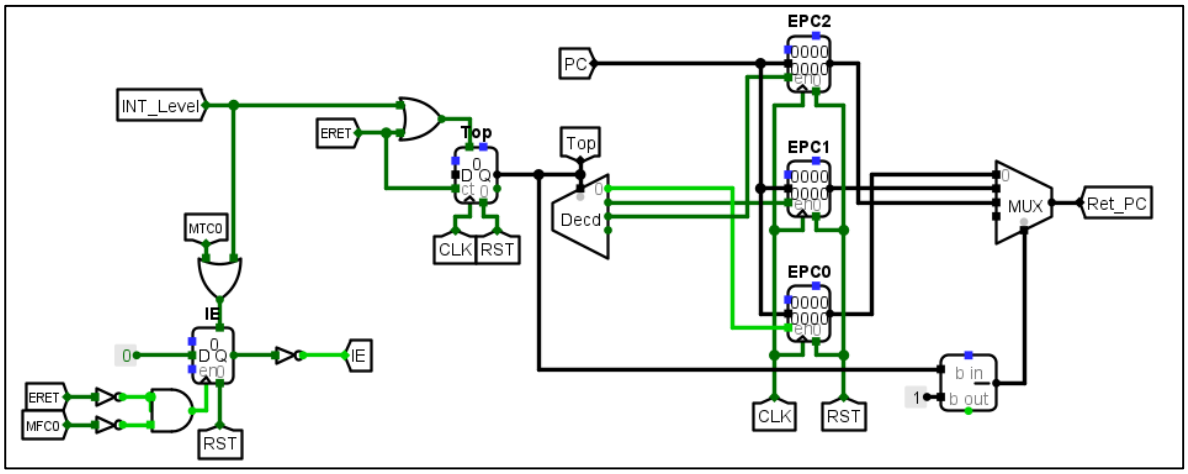

图 3.9 硬件栈和中断使能控制

在进入新的中断程序时，同步将返回地址送入栈顶寄存器，随后栈顶指针 TOP 加一；退出中断程序时 TOP-1 的地址存放在 RET_PC 中，在 ERET 的上升沿同步将 RET_PC 送到 PC。 上图左侧通过寄存器保存中断使能的值，INT_LEVEL 信号和 MTC0 信号负责将 IE 置为 0，ERET 信号和 MFC0 负责将 IE 置为 1。INT_LEVEL 一个短暂的高电平由 NEW_INT 的上升沿触发，持续到下一个时钟上升沿的结束，它涉及到多级中断的地址切换电路。

3.3.2 多级中断地址切换

多级中断的地址切换电路，如下所示。

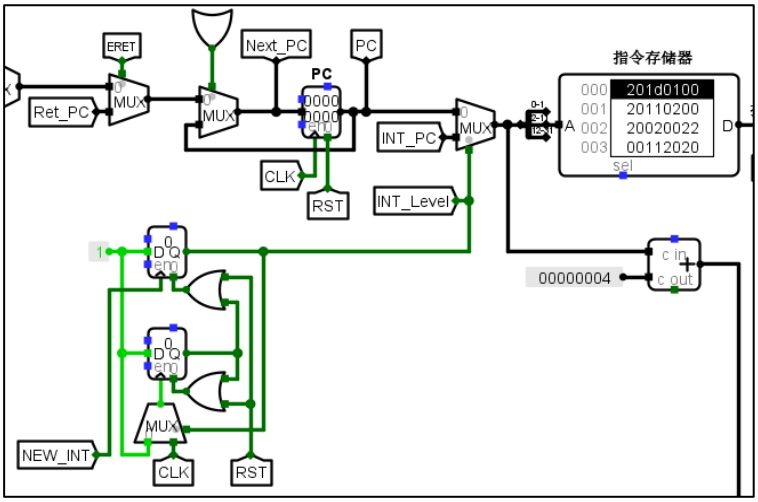

图 3.10 多级中断地址切换

这和单级中断的地址切换类似，只是进入新的中断是 NEW_INT 的上升沿触发的。

3.3.3 各中断程序的运行情况

为记录各中断程序的运行情况，我们设计如下电路。

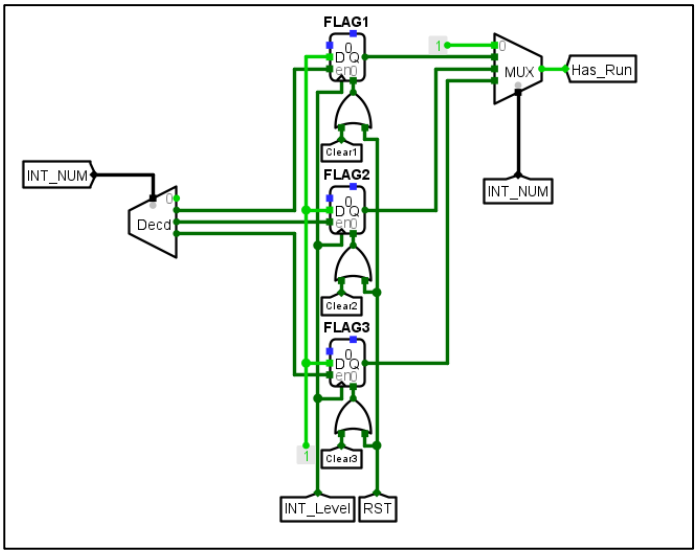

图 3.11 各中断程序的运行标志

3.4 理想流水 CPU 实现

3.4.1 总体电路

下面是理想流水的总体实现电路。

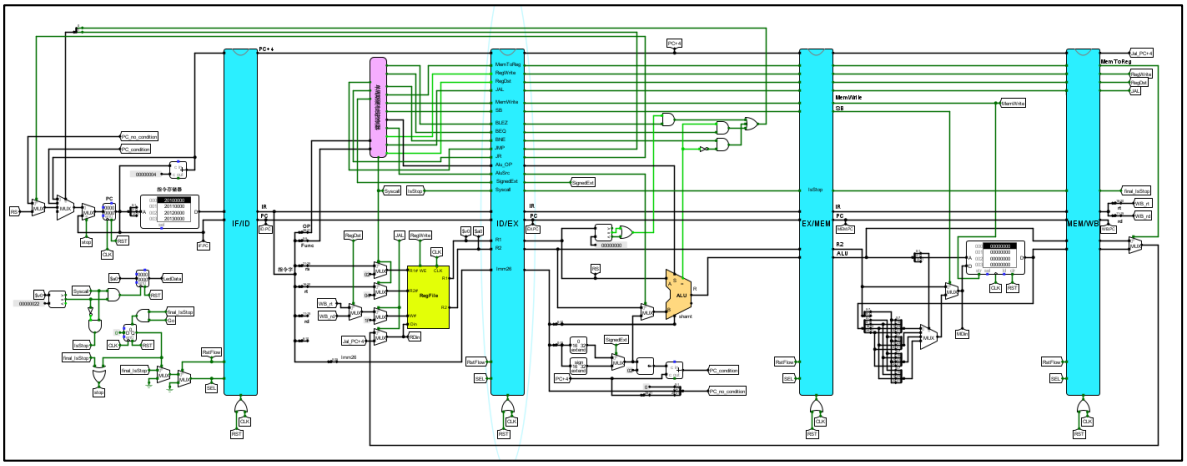

图 3.12 理想流水总体电路

取指阶段上半部分处理指令地址的问题，下半部分处理的是 Syscall、流水暂停和流水清空的问题。

3.4.2 流水接口部件实现

下面是 ID/EX 接口部件的实现，是所有接口部件中最复杂的，因为它要向后传递所有的控制信号和其他信息。

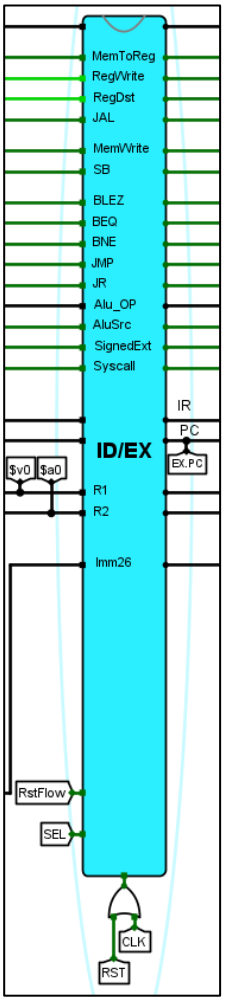

图 3.13 ID/EX 接口部件

其内部实现的部分电路如下所示，支持同步清零、使能控制。

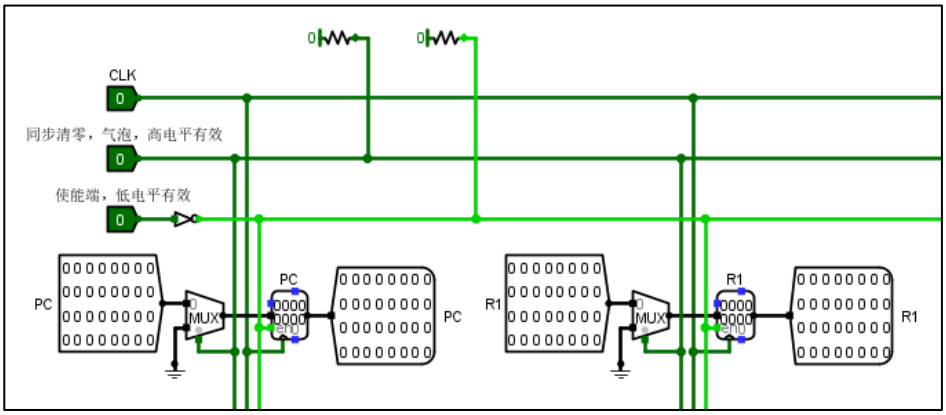

图 3.14 接口部件内部实现

3.5 气泡流水 CPU 实现

3.5.1 总体电路

气泡流水总体电路如下所示。

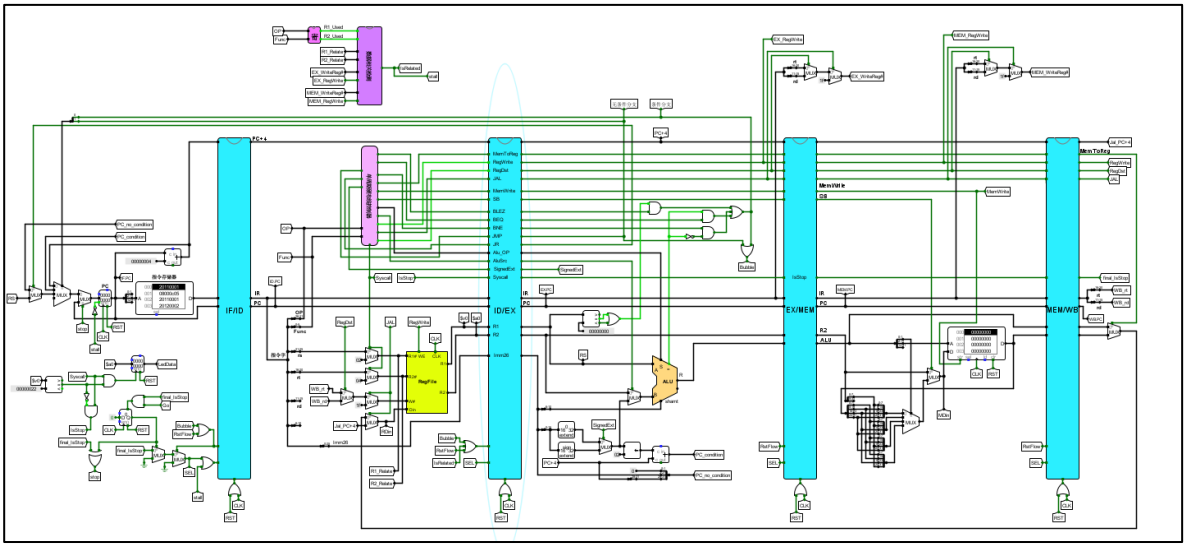

图 3.12 气泡流水总体电路

主要在理想流水的基础上加了分支检测和数据相关检测，以及暂停流水和插入气泡的电路

3.5.2 分支相关

在 EX 阶段检测是否跳转成功，如下图所示。

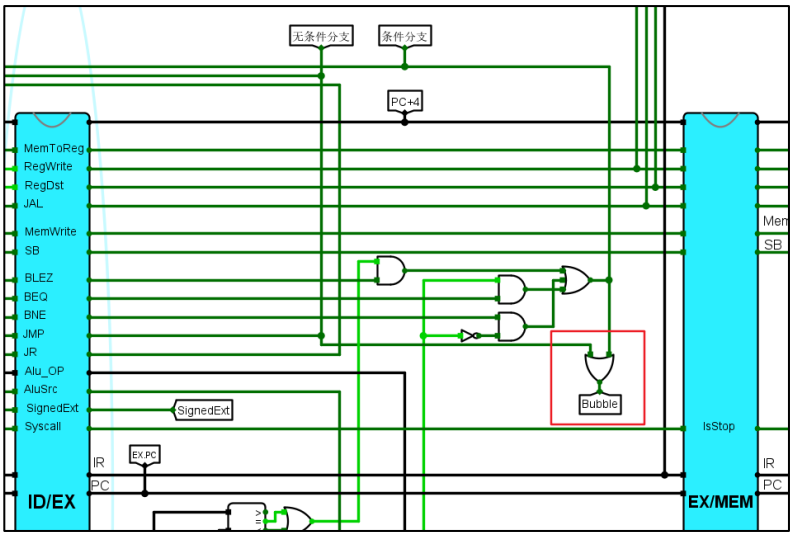

图 3.16 分支成功检测

利用 Bubble 信号，在 IF/ID 和 ID/EX 插入气泡（将其同步清零）。

3.5.3 数据相关

整体的数据相关检测电路如下所示。

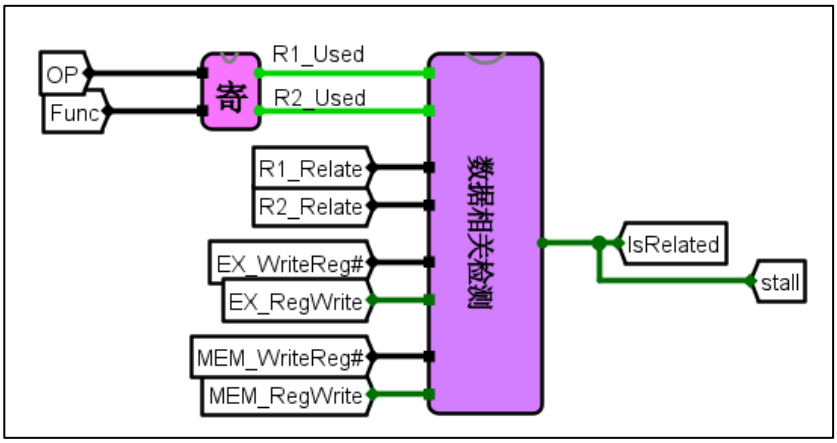

图 3.17 数据相关检测整体电路

其中，“寄”封装的电路负责检测 ID 段指令的源寄存器使用情况，R1_used 为 1 代表指令的第一个寄存器被使用，其编号为 R1_Relate，“寄”是根据指令的 OP 和 FUNC 利用控制器真值表自动生成的。

EX_RegWrite 为 1 代表 EX 段存在被写的寄存器，其标号为 EX_WriteReg#，其余类推。这些信号需要在 EX 阶段和 MEM 阶段检测，相关电路如下所示。

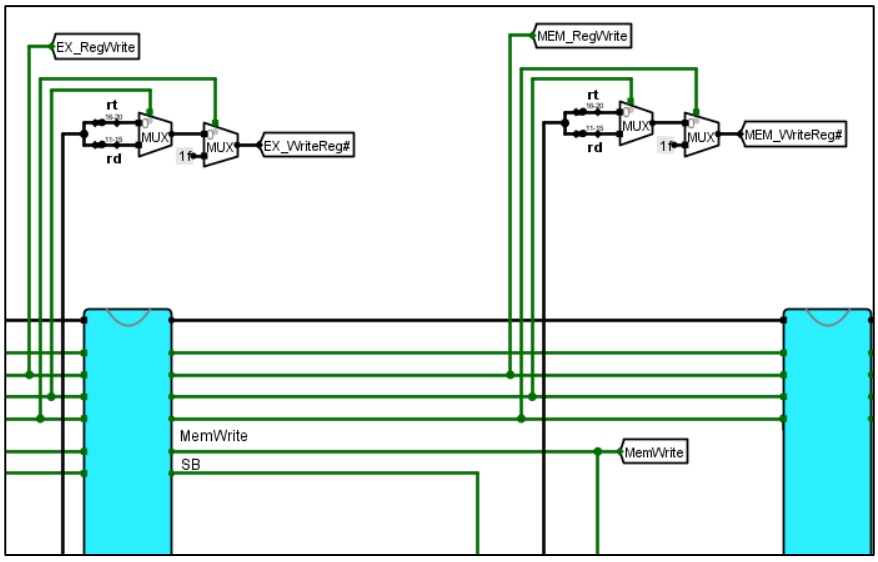

图 3.18 EX 和 MEM 写寄存器检测

而“数据相关检测”封装的电路利用输入信息产生是否存在数据相关的最终信号，其内部实现如下所示。

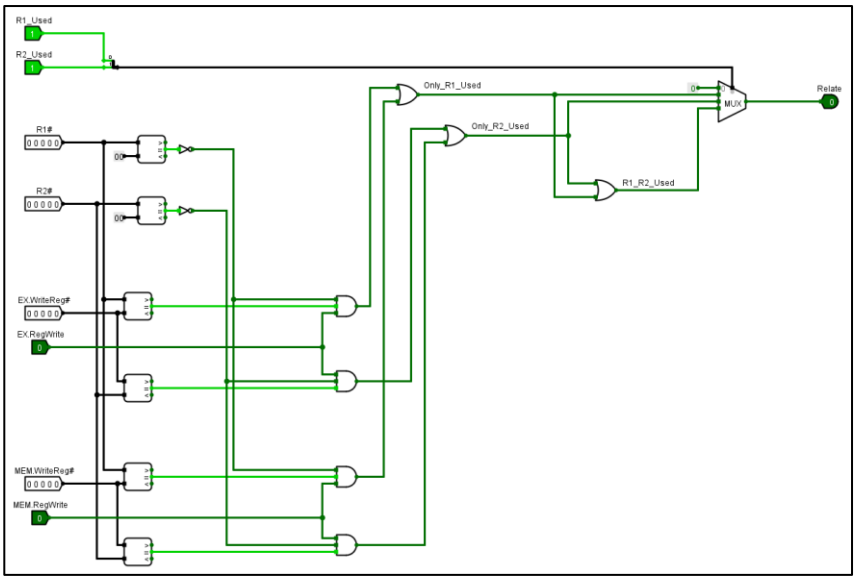

图 3.19 气泡流水“数据相关检测”内部实现

一个值得注意的细节是，数据相关检测时，寄存器编号为 0 不用考虑，主要因为 SLL 的 op 和 fun 均为 0，和初始 IR 为 0 相同，导致初始时或气泡时误检测到写寄存器编号为 0；

3.6 重定向流水 CPU 实现

3.6.1 总体电路

重定向流水总体电路如下所示。

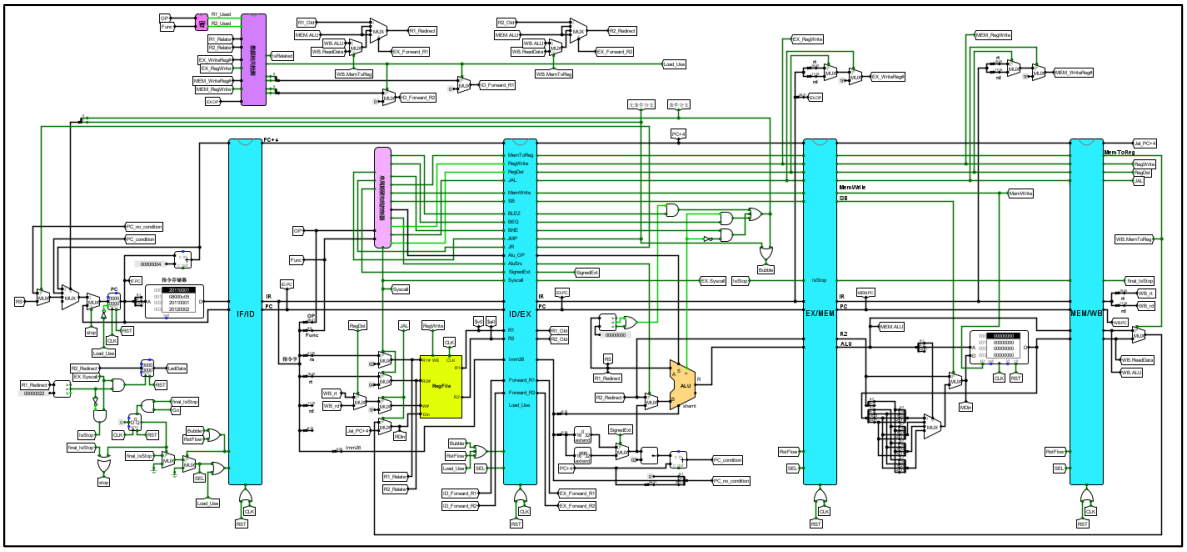

图 3.20 重定向流水总体电路

3.6.2 EX 重定向电路

我们在 EX 完成所有的重定向工作，相关电路如下所示。

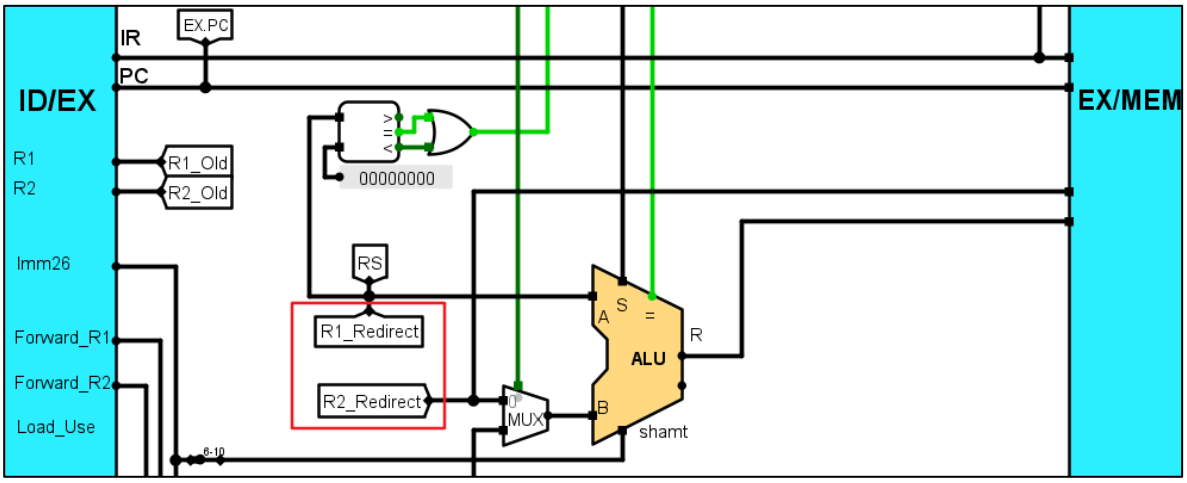

图 3.21 EX 重定向电路

根据前文所述，我们重定向进入 EX 的 R1 和 R2。

3.6.3 重定向选择信号的传递

我们需要将 ID 阶段的重定向选择信号传递到 EX 阶段，如下图所示。

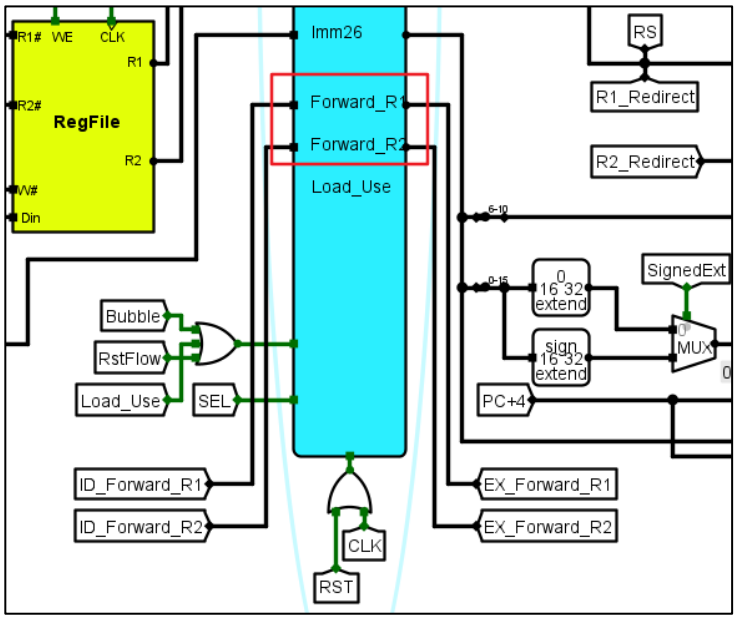

图 3.22 重定向选择信号的传递

ID_Forward_R1 是 ID 阶段为 R1 生成的重定向选择信号，通过流水接口传递到 EX 命名为 EX_Forward_R1。R2 类似。

3.6.4 重定向选择信号的产生和重定向来源

相关电路如下图所示。

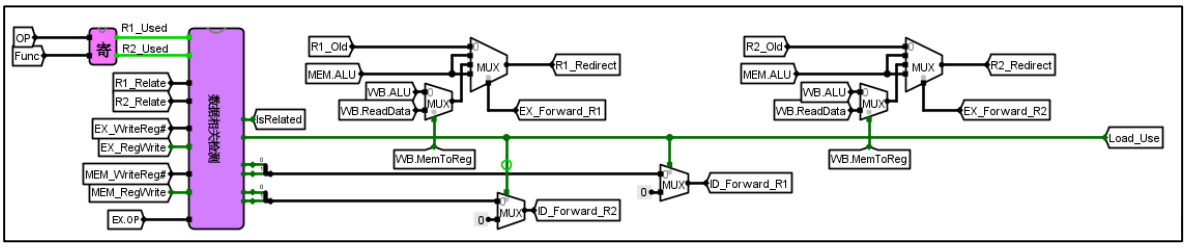

图 3.23 重定向选择信号的产生和重定向来源

对于“数据相关检测”封装电路，与气泡流水相比，多了 EX.op 的输入，因为我们要借此检测 Load_Use 相关；其输出多了 Load_Use 信号，在其为 1 的情况下，不需要考虑重定向而直接插入气泡即可。其下部四个输出从上到下分别代表 R1 与 EX 阶段数据相关、R1 与 MEM 阶段数据相关、R2 与 EX 阶段数据相关、R2 与 MEM 阶段数据相关。

根据这些信息，上图的右半部分生成 R1 和 R2 的重定向选择信号。

“数据相关检测”的内部实现如下所示，读者可对比前文气泡流水的“数据相关检测”加以理解。

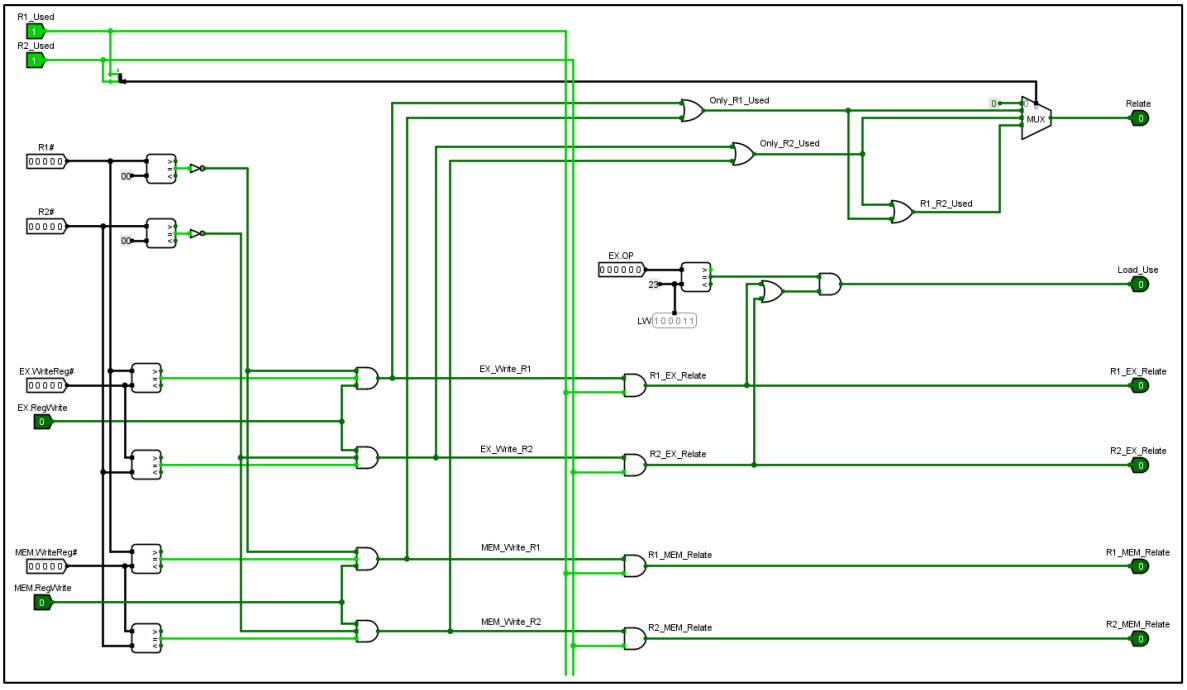

图 3.24 重定向流水“数据相关检测”内部实现

3.7 基于重定向流水的单级中断机制实现

3.7.1 总体电路

流水中断总体电路如下所示。

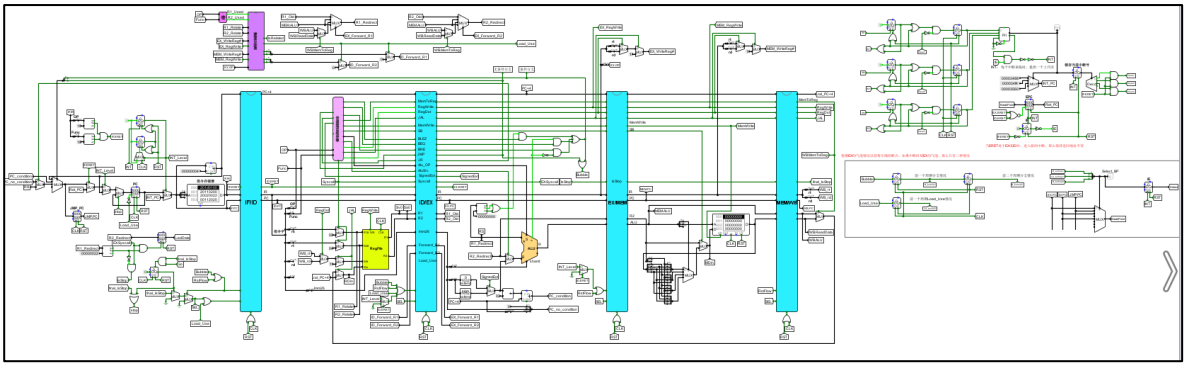

图 3.23 流水中断总体电路

中断的逻辑主要在右半部分。

3.7.2 中断的检测识别和返回

相关电路图如下所示。

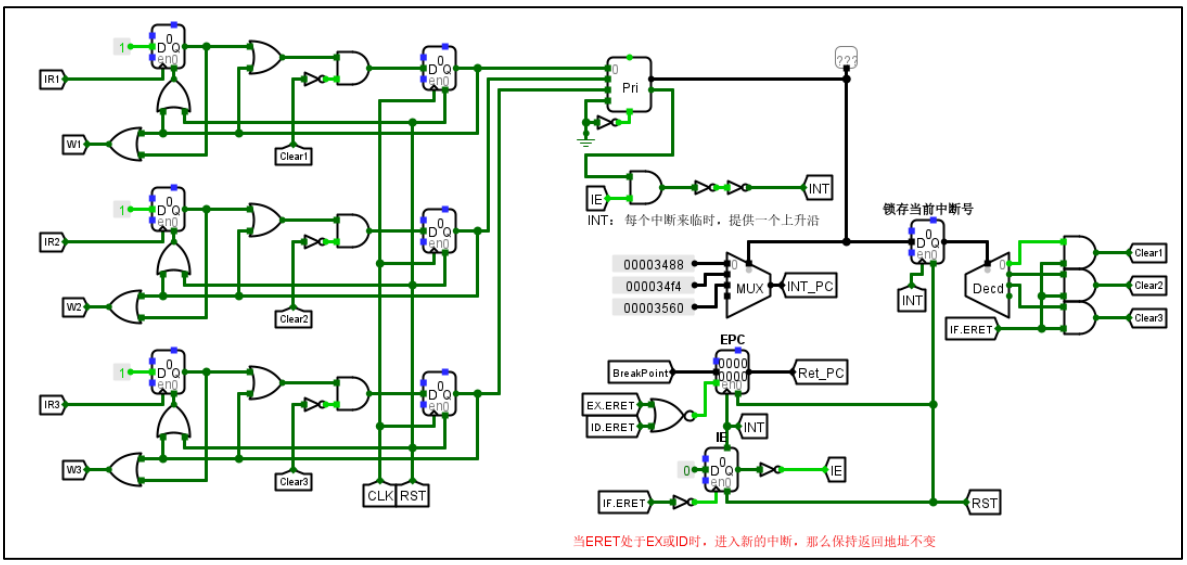

图 3.26 流水中断总体电路

值得注意的是，当 ERET 处于 EX 或 ID 时，若 IF 进入新的中断程序，那么保持返回地址不变。

3.7.3 断点的获取与保存

进入中断是需要保存断点(返回地址)，理想情况下，只需要保存 EX 的 PC 即可。

但是由于分支相关和 Load_Use 数据相关会导致中断时 EX 段为气泡的情况。在这种情况下，只可能有下列三种情况：

前一个周期出现了 Load_Use 相关，需要保存的断点为 ID 的 PC；前一个周期出现了分支相关，需要保存的断点为 JMP.PC（JMP.PC 保存了上一个跳转指令的目标地址）；前二个周期出现了分支相关，需要保存的断点为 ID 的 PC。

如果中断时 EX 不是气泡，那么需要保存的断点即 EX 的 PC。

实现的电路如下图所示。

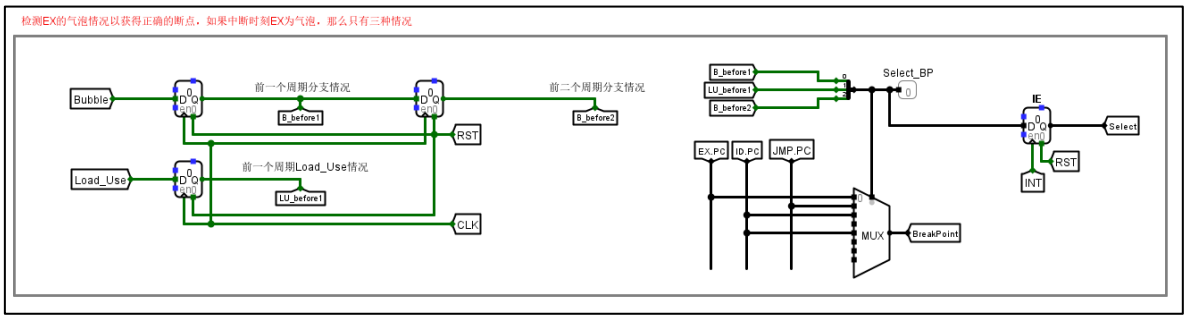

图 3.27 断点的获取与保存

3.7.4 IF 阶段的地址切换

陷入中断与从中断中返回的地址切换工作在 IF 段完成，如下图所示。

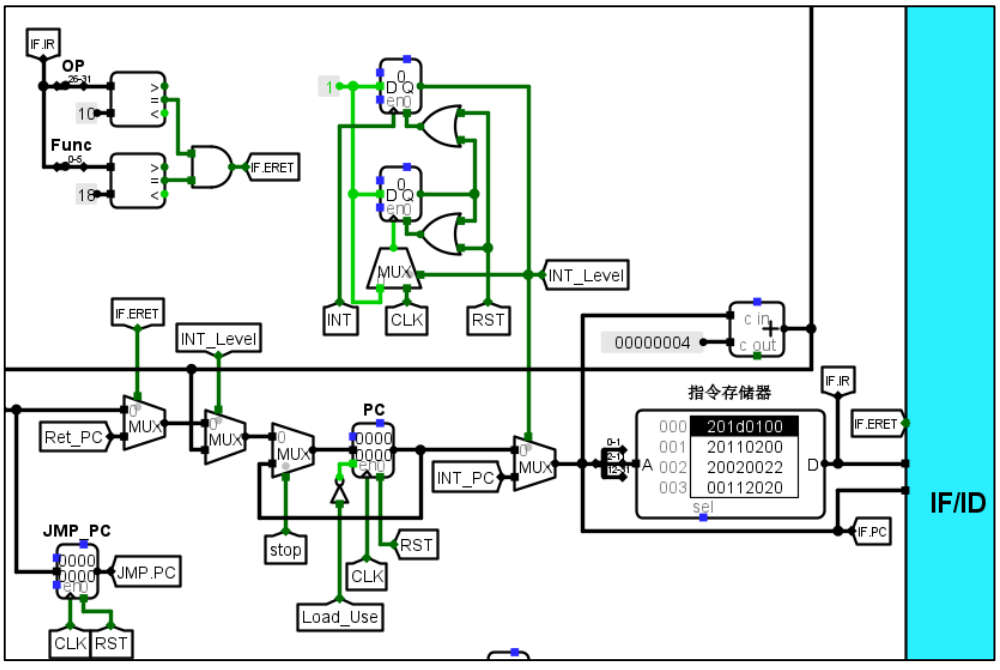

图 3.28 IF 阶段的地址切换

其中左上角的电路负责在 IF 阶段检测 ERET 指令，及时将返回地址送入 PC，避免插入气泡从而影响流水性能。

3.8 基于重定向流水的动态分支预测机制实现

3.8.1 总体电路图

动态分支预测的总体电路图如下所示。

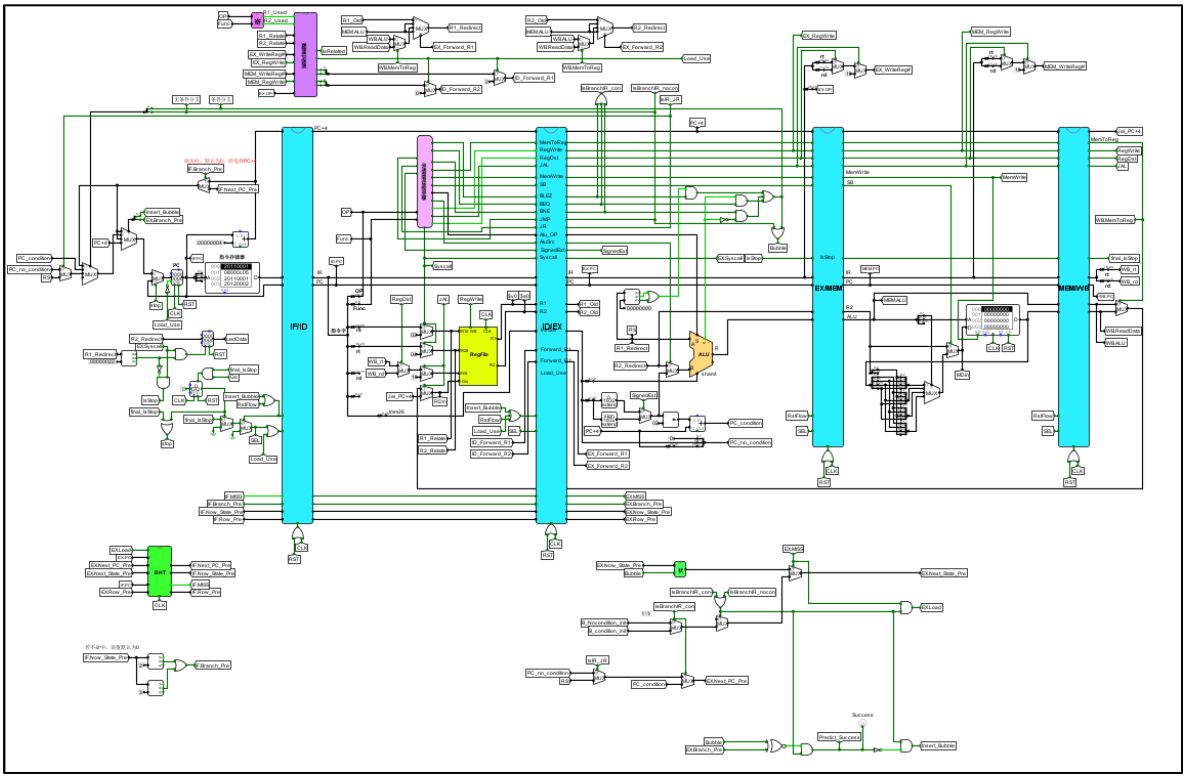

图 3.29 动态分支预测总体电路

3.8.2 IF 阶段的地址选择

相关电路如下所示。

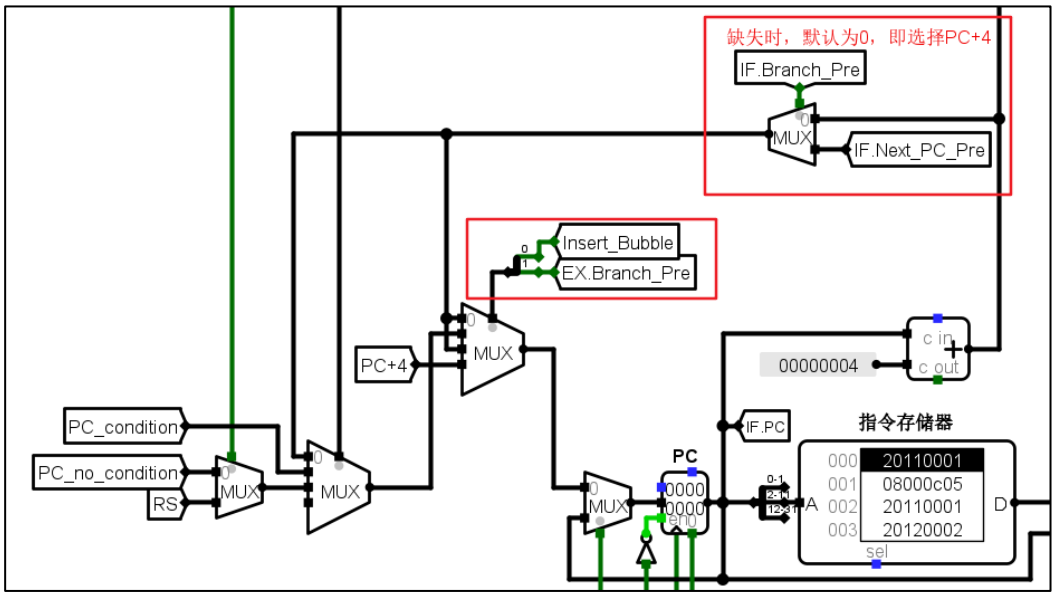

图 3.30 IF 阶段的地址选择

其中右上方的红框表示如果预测跳转成功则下一条指令的地址为该跳转指令的目标地址，左下方的红框表示如果 EX 阶段检测出预测错误则需要回到预测点执行另一条分支路径。

3.8.3 BHT 的实现

BHT 的封装以及相关预测逻辑如下所示。

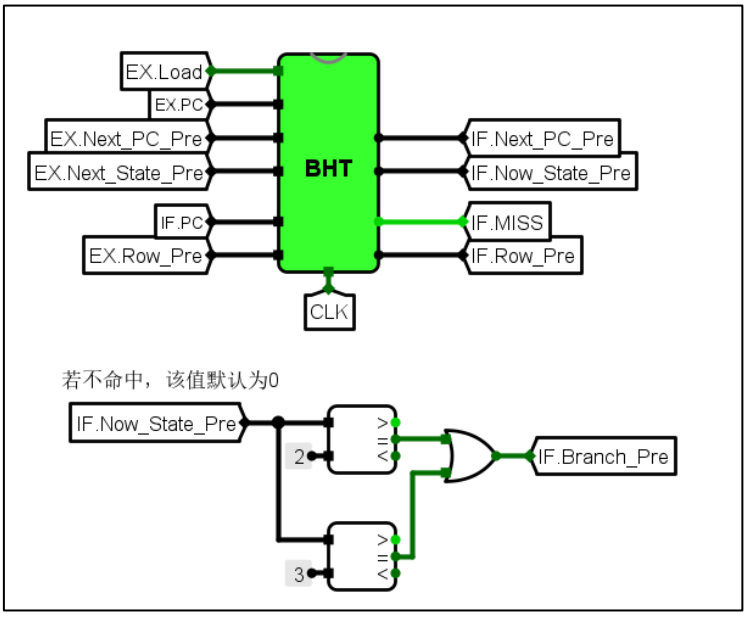

图 3.31 BHT 的封装及相关预测逻辑

上图中，以 IF 开头的信号属于 IF 阶段的查找逻辑（读逻辑），以 EX 开头的信号属于 EX 阶段的更新或插入（替换）逻辑。EX 阶段需要更新 BHT 时，EX.Row_Pre 代表需要更新的行，EX.Next_State_Pre 代表更新后的状态值；EX 阶段需要插入（更新）时，EX.Next_State_Pre 代表初始的状态值，根据该指令为条件跳转指令或无条件跳转指令而不同。

BHT 内部实现逻辑的部分电路如下所示，不再过多解释。

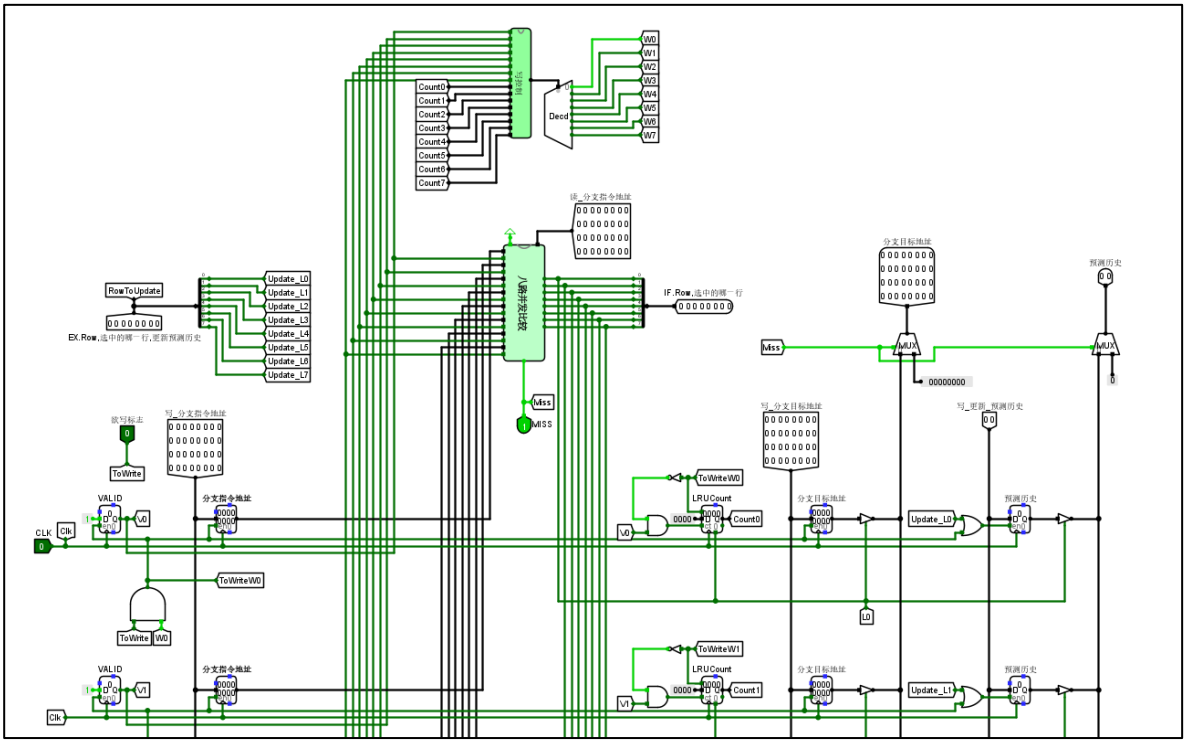

图 3.32 BHT 内部实现部分电路

3.8.4 IF 向 EX 传递的预测信息

IF 必须向 EX 传递预测的相关信息使得在 EX 阶段可以做出预测是否成功的判断，增加的流水接口如下所示。

图 3.33 预测信息流水接口

3.8.5 EX 预测结果的检测逻辑

EX 阶段检测预测是否成功，并且要准备 BHT 更新或插入（替换）的信息，电路如下所示。

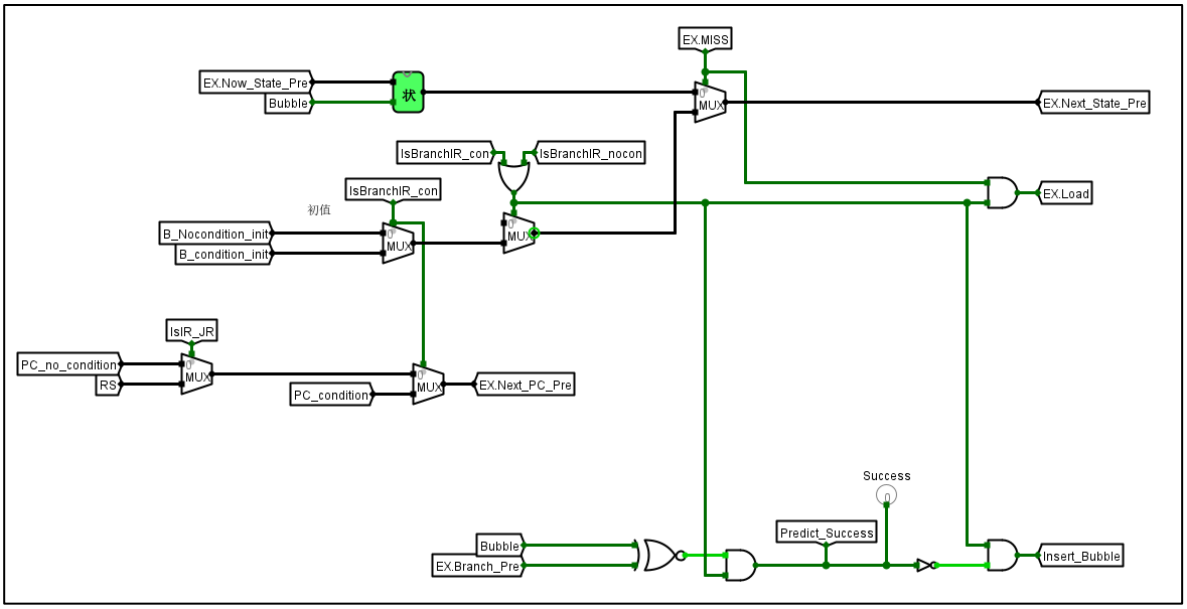

图 3.34 EX 阶段预测结果的检测

其中，当 EXLoad 为 1 时 BHT 将插入新的跳转指令，Insert_Bubble 为 1 时说明预测失败，需要插入气泡。

值得注意的是，如果 IF 阶段查找缺失，我们默认是不跳转的。那么如果有一条条件跳转指令，第一次来到 EX 阶段，之前在 IF 肯定是缺失的，如果实际上在 EX 阶段该指令确实不跳转，那么也不需要插入气泡，也属于预测成功。当然，下一个时钟将会把该条跳转指令插入到 BHT 中。

上图中“状”所封装的电路实现状态的转换，利用 LOGISIM 根据真值表自动生成电路的功能完成的，内部实现如下。

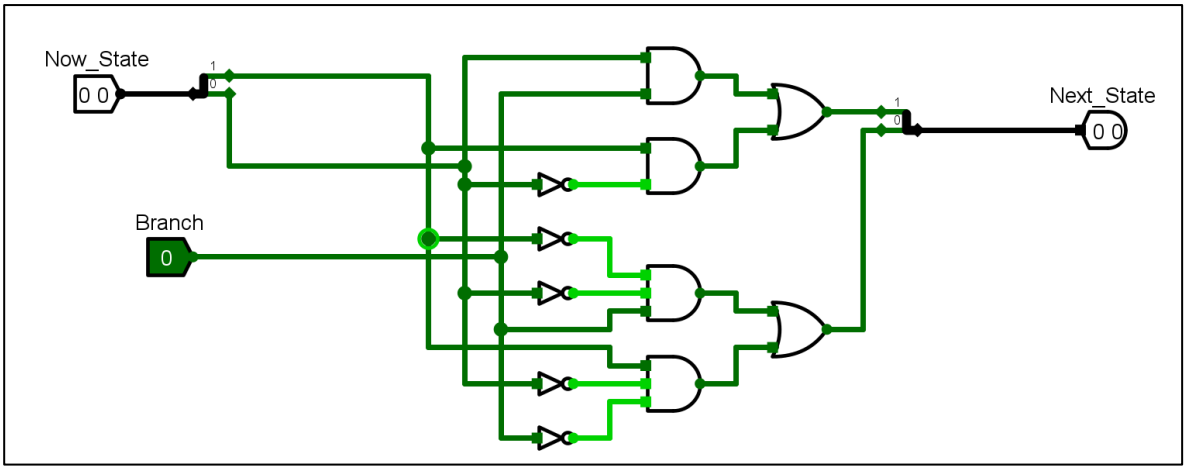

图 3.34 BHT 状态转换电路

3.9 团队项目实现

3.9.1 工作量与团队成员分工

主要工作与分工如下表所示：

表 3.1 团队项目分工表

成员名	负责的工作
于泽文	CPU 新增中断与显示逻辑等
尚琛展	程序主体框架与关卡中断程序等
李馨玥	音乐文件和音乐 DMA 等
罗俊杰（组长）	上下左右移动中断和最后组装等

3.9.2 开发工具与协作工具

开发平台基于 LOGISIM 和 MARS，交流协作工具包括 QQ 和 ZOOM。

3.9.3 完成情况及遇到的问题

小组成员密切合作，积极交流，都出色地完成了自己的工作，团队项目如期完成。

因为团队项目是经过所有团队成员细致的讨论后才开始具体实现的，所以达成比较高的一致性认知，实现过程相对比较顺利。但是还是存在一些地方出现了小的偏差，需要共同探讨，一起解决。

本人负责上下左右移动中断程序，逻辑相对比较复杂，难免出现问题，只能单步调试进而解决；最后组装电路时花了一定的时间设计实现键盘解析逻辑，最终根据约定的接口完成用户交互界面的设计和实现。

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 CPU24+4 测试

加载基准测试程序 benchmark.hex，CPU24+4 的运行结果如下所示。

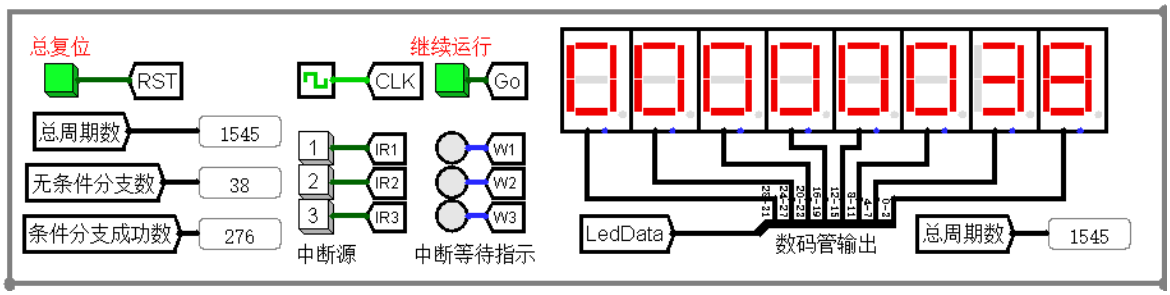

图 4.1 CPU24+4 测试参数

下图为内存排序结果。

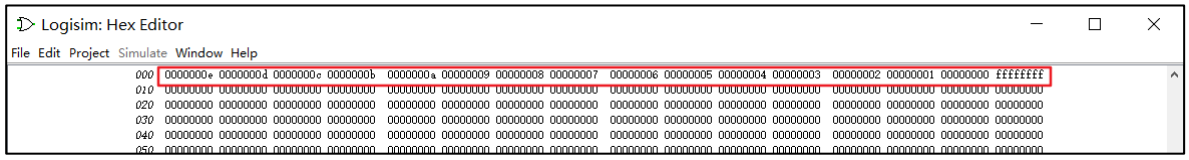

图 4.2 CPU24+4 测试内存结果

上述参数与结果和预期一致，任务完成。

4.1.2 单级中断测试

加载“中断演示主程序.hex”，主程序运行时快速依次按下中断“1，2，3”，执行顺序为“1，3，2”，最后返回到主程序执行，符合预期。下面是运行时截图。

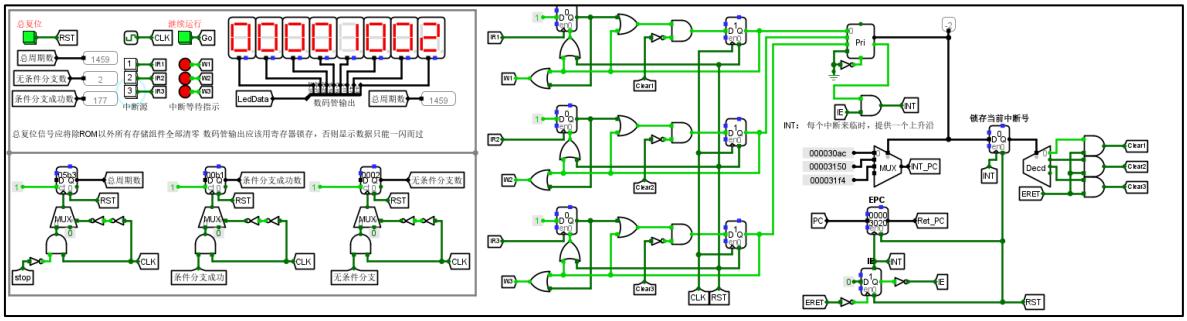

图 4.3 单级中断测试截图

4.1.3 多级中断测试

加载“多级中断演示主程序.hex”，主程序运行时依次按下中断“1，2，3”，执行顺序为“1，2，3，2，1”；依次按下“2，3，1”，执行顺序为“2，3，2，1”。结果与预期一致，功能实现。下面是运行过程的截图。

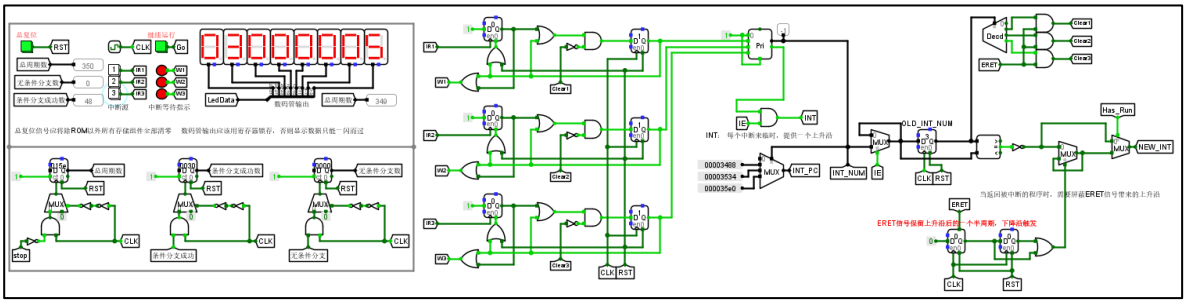

图 4.4 多级中断测试截图

4.1.4 理想流水测试

加载“理想流水线测试.hex”，运行参数如下所示。

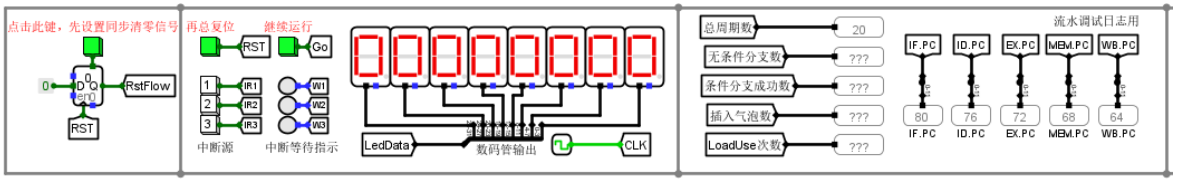

图 4.5 理想流水测试参数

总周期为 20，符合预期。下面是运行后的内存结果。

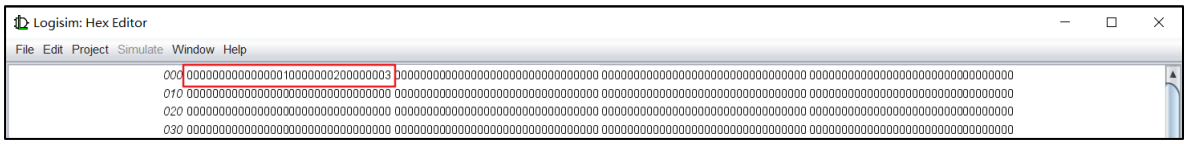

图 4.6 理想流水测试内存结果

4.1.5 气泡流水测试

加载“benchmark.hex”，运行结果如下，符合预期。

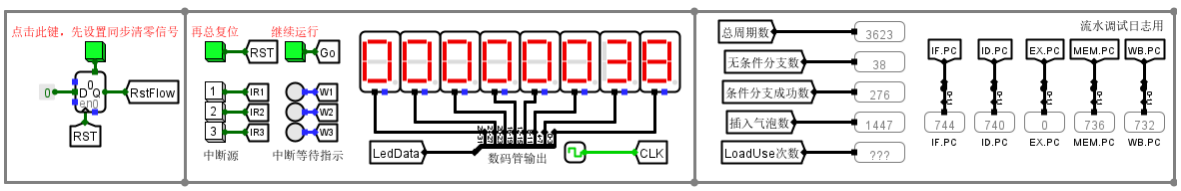

图 4.7 气泡流水测试结果

4.1.6 重定向流水测试

加载“benchmark.hex”，运行结果如下，符合预期。

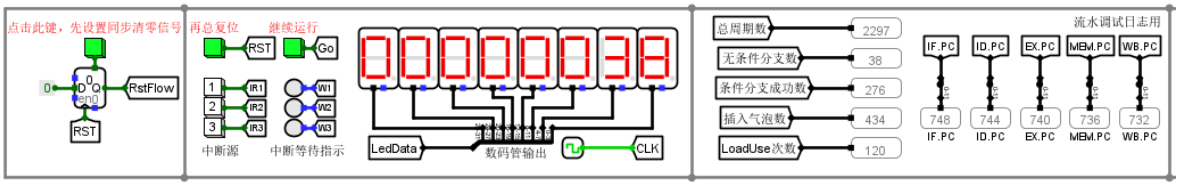

图 4.8 重定向流水测试结果

4.1.7 基于重定向流水的单级中断测试

和单周期单级中断类似，加载“中断演示主程序.hex”，主程序运行时快速依次按下中断“1，2，3”，执行顺序为“1，3，2”，最后返回到主程序执行，符合预期。下面是运行时截图。

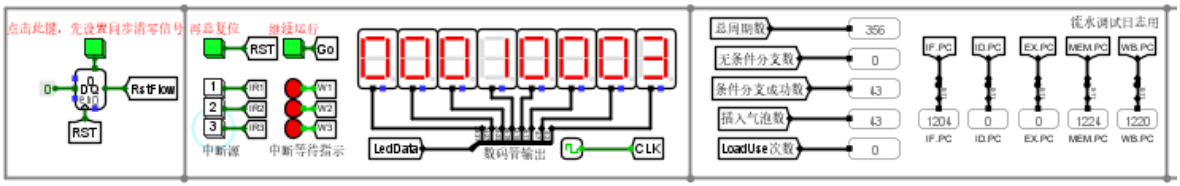

图 4.9 流水中断测试截图

4.1.8 基于重定向流水的动态分支预测机制测试

加载“benchmark.hex”，将条件分支初始状态 B_condiiton_init 设置为“11”，无条件分支初始状态 B_Nocondion_init 设置为“10”，运行结果如下。

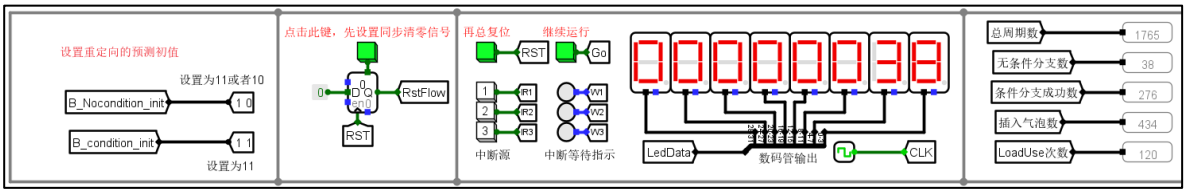

图 4.10 动态分支预测测试结果

可见，总周期为 1765，与基本的重定向流水的 2297 相比，性能提升不少。

4.1.9 团队项目测试

下面是游戏启动时的交互界面情况。

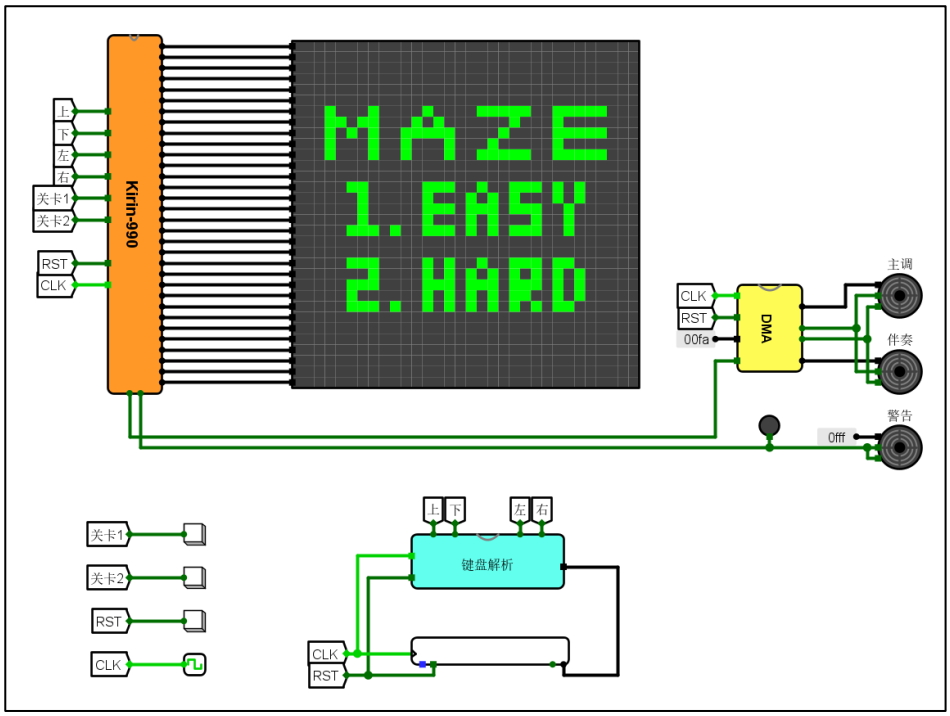

图 4.11 游戏启动界面

具体功能测试与介绍请参看“CSIE1701_关山口施工队_计算机组成原理课程团队项目答辩视频.mp4”，下面是视频的介绍逻辑。

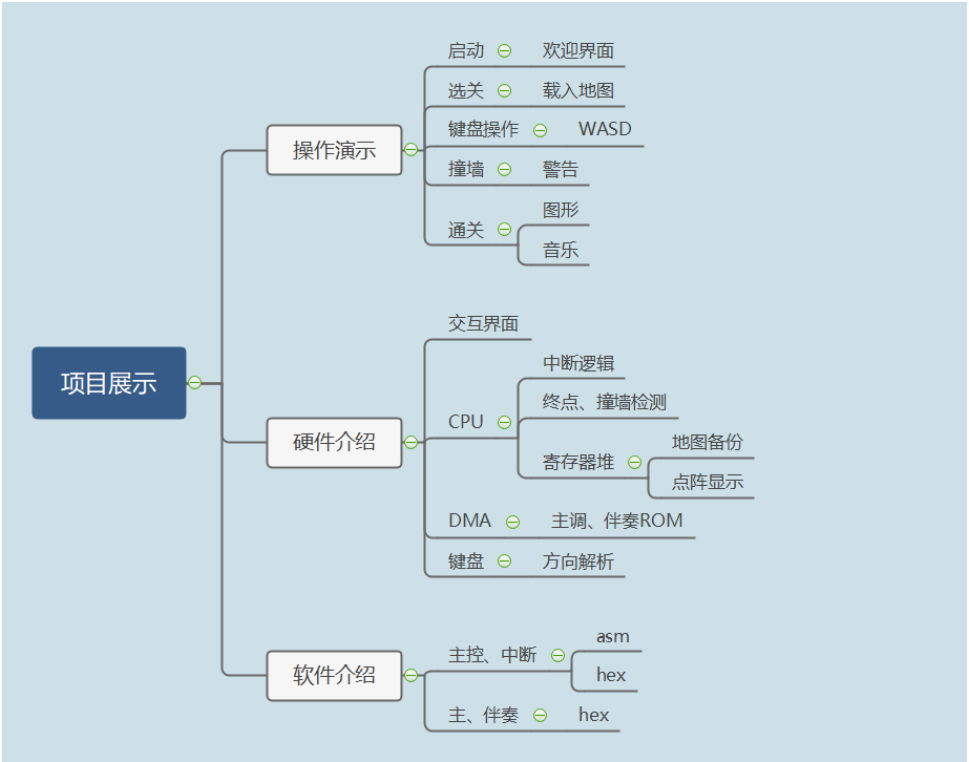

图 4.12 团队项目视频介绍逻辑

4.2 性能分析

下表列出了测试基准程序“benchmark.hex”时，各种方案的时钟周期数。

表 4.1 各方案基准测试周期表

方案	CPU24+4	气泡流水线	重定向流水线	动态分支预测
周期数	1545	3623	2297	1765

由上表可知，与基本的单周期 CPU24+4 方案相比，各流水解决方案的周期数都要多，但是流水方案的周期短，所以性能还是会有不同程度的提升。

气泡流水线因为数据相关和分支相关插入了大量的气泡，所以它的周期数是最多的；重定向流水线在气泡流水线的基础上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；在重定向流水线的基础上加入动态分支预测技术，进一步优化了对分支相关的处理，使得流水周期减少了 532 个时钟周期，流水性能得到很明显的提升。

如果想进一步提升流水性能，对于分支相关，可以把分支指令的执行提前到 ID 阶段完成，甚至可以采取延迟槽技术完全消除分支相关对流水性能的影响；对于 Load_Use 相关，可以考虑数据预取，这是更深层次的研究内容。

4.3 主要故障与调试

4.3.1 单级中断故障

故障现象：如果多个中断同时发生，只有第一个中断会正常执行，后续中断无法执行且其中断请求信号不会被清空。

原因分析：如下图所示，IE 的值是用一个寄存器控制的，IE 的值持续为 1，当有多个中断请求时，INT 的值持续为 1，不会产生一个上升沿，也就无法执行被保存的其他优先级较低的中断请求。

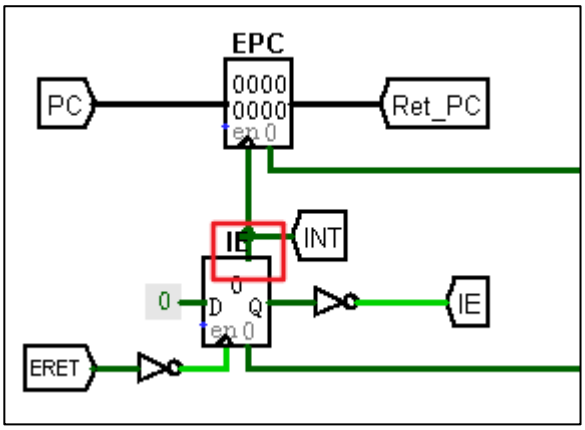

图 4.13 IE 控制错误

解决方案：用 D 触发器替换寄存器，并让 INT 信号控制其直接置 1 端，那么可以保证某个中断执行的时候 IE 为 0，中断返回后 IE 为 1。如果有多个中断同时发生，那么在第一个中断返回后，IE 的上升沿会产生 INT 的上升沿，可以继续执行下一个中断服务程序。改造后的电路已经在前文给出。

后面又发现之所以存在这个问题，是因为测试时打开该电路文件使用的是官网的 logisim2.7.1，而电路开发时使用的是华科改良版的 LOGISIM，如果用后者打开，寄存器拥有直接置 1 的端口，不会出现这个问题，不用换成 D 触发器。

4.3.2 气泡流水故障

故障现象：气泡流水偶尔工作异常。

原因分析：由 LW 指令测得 MemToReg 信号在 ID/EX 阶段传递中断，进而发现内部短路，如下图所示。

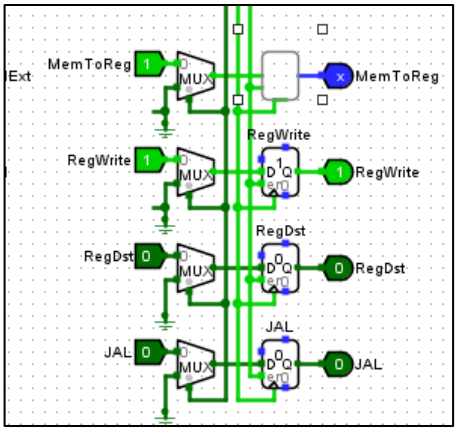

图 4.14 ID/EX 接口部件内部短路

解决方案：将短路线移除即可。

4.3.3 重定向流水故障

故障现象：重定向流水处理 Load_Use 相关时出现错误。

原因分析：重定向选择信号使用了 ID 阶段新产生的 EX_Load_Use，而不是传递到 EX 阶段的 EX_Load_Use，如下图所示。

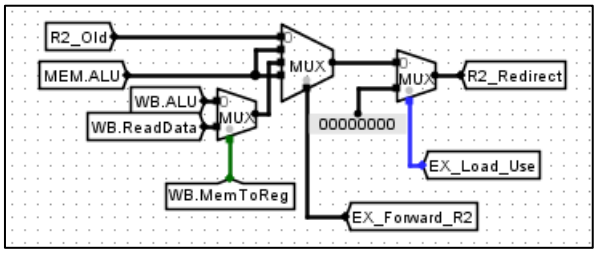

图 4.15 重定向流水 Load_Use 处理错误

解决方案：为了减少流水接口，在 ID 阶段优先考虑 Load_Use 信号，若其为 1，则直接将重定向选择信号置为 0（即不用考虑重定向），下一周期直接在 ID/EX 插入气泡。

4.3.4 动态分支预测功能故障

故障现象：动态分支预测电路无法正确执行基准测试程序。

原因分析：在 EX 阶段判断是否预测成功的逻辑不对，如下所示。

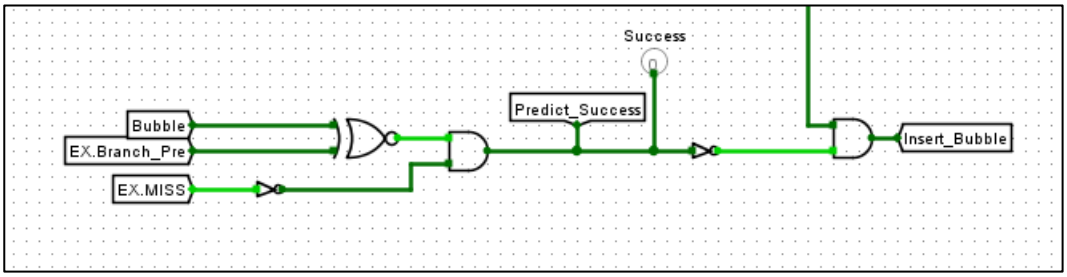

图 4.16 动态分支预测结果判断错误

解决方案：为 Predict_Success 信号不应该由 EX.MISS 决定。正如前文所述，当一条条件分支指令第一次来到 EX 阶段，我们认为这次预测是成功的（IF 阶段 BHT 缺失则默认不跳转），不应该插入气泡。改正后的电路已在前文给出。

4.3.5 动态分支预测性能故障

故障现象：动态分支预测电路虽然可以正确执行基准测试程序，但是较基本的重定向流水其性能提高很少。

原因分析：BHT 实现出错，具体的是计数器 LRUCount 异步清零，导致 LRU 替换算法无法正确找到被替换的 BHT 行，如下图所示。

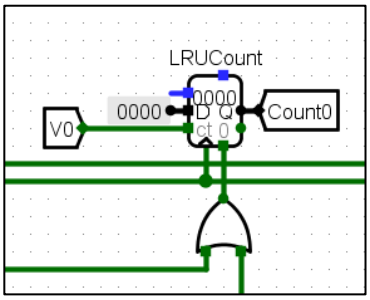

图 4.17 BHT 计数器异步清零错误

解决方案：在该行命中时，LRUCount 可以异步清零，当该行需要被替换时，LRUCount 必须同步清零，如下所示。

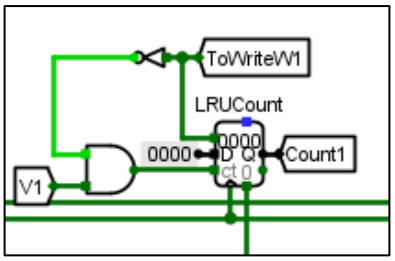

图 4.18 BHT 计数器正确的清零机制

4.4 实验进度

我组从一开始就制定了课设进度安排表，并在两周的课设时间内严格按照进度表执行。

表 4.2 小组课程设计进度表

关山口施工队日程

任务	天数	完成日期	累计分数	scz	yzw	lxy	ljj	备注
CPU24+4	2	2.17-2.18	50	√	√	√	√	
单级中断	1.5	2.19-2.20	60	√	√	√	√	
理想流水线	1	2.20-2.21	65	√	√	√	√	第一周
气泡流水线	1.5	2.21-2.22	75	√	√	√	√	
重定向流水线	1.5	2.23-2.25	85	√	√	√	√	
单周期上板FPGA	/(1)		90					返校实现
团队展示	2		总成绩15%	√	√	√	√	第二周
多级中断			/+5~15	√	√	√	√	
流水中断			/+5	√	√		√	
流水上板FPGA			/+15					扩展模块任意组合
分支预测			/+10	√	√		√	
备注								

然而，实际情况与进度安排表有细微差别，主要体现在团队展示的任务是在第二周完成多级中断、流水中断和动态分支预测之后完成的。

5 设计总结与心得

5.1 课设总结

本次课程设计内容较多，难度较大，我主要做了如下几点工作：

1） 设计并完成了 CPU24+4，该任务在上学期组成原理课程实验的基础上加以补充，相对较简单，但是也是整个课设的基石，很重要；

2） 设计并完成了单级中断机制，主要是理解单级中断的陷入和返回的原理，设计电路产生必要的控制信号；

3） 设计并完成了多级中断机制，在单级中断的基础上完成了嵌套中断的电路支持；

4） 设计并完成了理想流水线，主要工作是流水接口的设计、流水线的分割和停机、显示、清空流水等功能的实现；

5） 设计并完成了气泡流水线，用流水阻塞和插入气泡等方法解决各种冲突。

6） 设计并完成了重定向流水线，在气泡流水线的基础上加入旁路机制，关键是搞清重定向的数据来源和重定向选择信号的产生；

7） 设计并完成了基于重定向流水的单级中断机制，主要解决在单周期单级中断的基础上因流水而带来的问题；

8） 设计并完成了基于重定向流水的动态分支预测机制，深入理解其机制，设置 BHT 和相关预测逻辑进一步提升流水性能；

9） 担任小组组长，参与团队项目的讨论，完成上下左右移动的中断程序，设计完成用户交互界面，完成后期视频录制。

5.2 课设心得

本次课设历时 2 周，实现了从基本的单周期 CPU 到 MIPS 五段重定向流水动态分支预测机制的任务，并且利用设计的 CPU 完成了“走迷宫”的小游戏。这两周过得很充实，很有成就感，因为“咱们也是造过 CPU 的人了”。

从 CPU24+4，到单级中断、多级中断，到理想流水、气泡流水、重定向流水，再到流水中断、动态分支预测，实验难度逐渐增大，也越来越难 DEBUG，也不知道

熬了多少次夜，抠了多少次头才完成全部的任务。然而，正是因为课设的难度渐进性，所以做好基础的任务是对后续任务很好的铺垫，并且基于 LOGISIM 和 MARS 平台开发的所有细节都是可见的，只要有耐心，问题总是可以解决。

实验过程考验了我们对基本知识的理解，获取新知识的能力，创新设计能力，以及发现问题解决问题的能力，团队项目锻炼了我们的表达能力，协作能力。

担任“关山口施工队”课设小组的组长让我有更多的收获，让我知道如何营造良好的小组工作气氛，如何协调成员的关系，也知道作为组长就需要有大局意识，奉献意识。

整体来说，课设的体验还是不错的，得益于老师的辛苦付出。然而，我有一个小建议希望课程组能采纳。我希望在课设的开始就能制定一套标准，满足什么条件任务就算完成，而不是在实验过程中反复变动。在经验丰富的人看来这些变动微不足道，但是对于大多数同学而言，在短时间、高任务的情况下这些变动很是影响心态和进度。

另外，因疫情原因不能在校利用 FPGA 进行上板测试，很是遗憾。

最后，衷心感谢谭志虎老师、胡迪青老师的耐心指导，感谢小组成员的支持和帮助！

参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4] 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社,2018.

[5] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

[6] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字: 

2020 年 8 月 15 日星期六