华中科技大学

2022

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：	CE1901
学　　号：	U201914870
姓　　名：	刘红阳
电　　话：	17307228012
邮　　件：	3184035501@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

（1） 支持规定的 32 位 mips/risc-v 指令集（指令集任选），具体见表 1；

（2） 在扩展指令集中支持 2 条 C 类运算指令，1 条 M 类存储指令，1 条 B 类 分支指令，具体任务每位同学不一样，任务要求由指导教师制定；

（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 3 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） 能自动统计各类无条件分支指令数目，条件分支成功次数、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 基本指令集

#	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	

#	指令助记符	简单功能描述	备注
12	ORI	立即数或	
13	NOR	或非	
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	SLLV	逻辑可变左移	**学生特殊需要实现指令**
29	XORI	异或立即数	**学生特殊需要实现指令**
30	SB	存储字节	**学生特殊需要实现指令**
31	BGTZ	大于 0 转移	**学生特殊需要实现指令**

2 总体方案设计

2.1 单周期 CPU 设计

MIPS 单周期 CPU 设计通过将指令存储器和数据存储器分开来实现单周期。主要组成部件包括程序计数器 PC、指令存储器、逻辑控制单元、寄存器堆、逻辑算术运算单元和数据寄存器。当时钟信号到来时，根据 PC 寄存器中值取出指令，指令通过逻辑控制单元产生控制信号，同时根据指令译码取出寄存器堆中的值送入逻辑算术运算单元，并将结果写回。

总体结构图如图 2.1 所示。

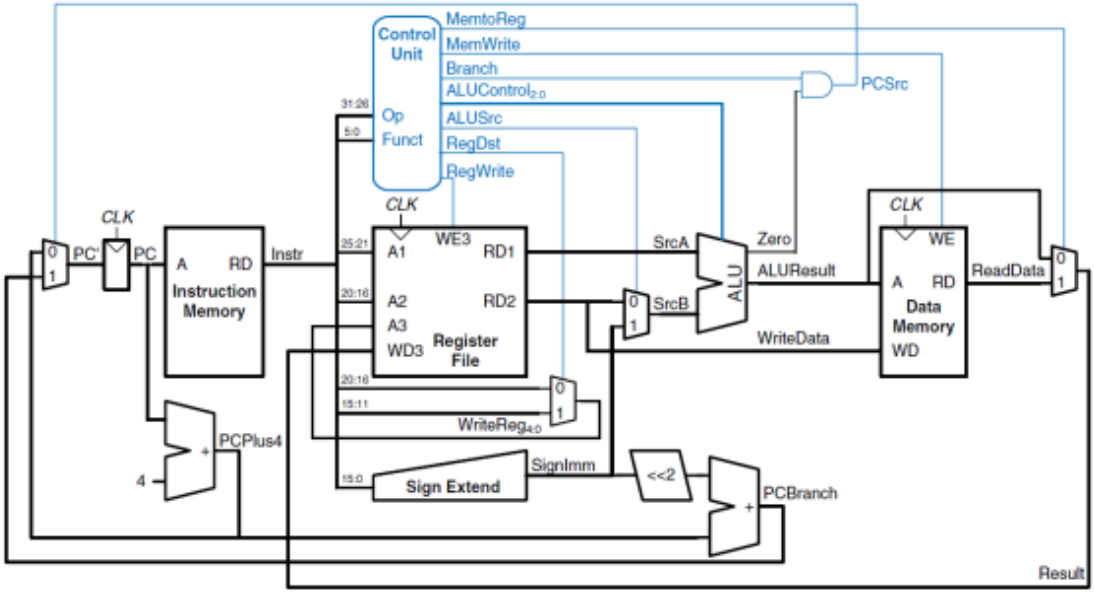

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

PC 寄存器由同步时钟信号 clk 控制，记录下一条指令的地址。

2. 指令存储器 IM

指令存储器根据 PC 值取出对应的指令，且不受时钟控制。

3. 运算器

表 2.1 算术逻辑运算单元功能描述

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
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

表 2. 2 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
Shamt	输入	5	只对左移右移运算有效，表示移动位数
ALU_OP	输入	4	运算器功能码，具体功能见表 2.1
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
Equal	输出	1	Equal=(x==y)?1:0，对所有操作有效

4. 寄存器堆 RF

由 32 个寄存器组成，可一次读取两个寄存器，且不受时钟信号控制。

表 2.3 寄存器堆的引脚与功能描述

引脚	输入/输出	位宽	功能描述
R1#	输入	5	读出寄存器堆中寄存器数据的编号
R2#	输入	5	读出寄存器堆中寄存器数据的编号
W#	输入	5	写入寄存器堆中的寄存器编号
WD	输入	32	写入寄存器的数据
WE	输入	1	写使能，高电平有效
R1	输出	32	根据 R1#的值读出对应寄存器数据
R2	输出	32	根据 R2#的值读出对应寄存器数据

2.1.2 控制器的设计

首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如表 2.4。

表 2.4 主控制器控制信号的作用说明

#	控制信号	信号说明	产生条件（信号为1）
1	RegWrite	寄存器写使能	寄存器写回信号
2	MemWrite	写内存控制信号	sw指令　未单独设置MemRead信号
3	AluOP	运算器操作控制符（4位）	R型指令根据Func选择
4	MemToReg	寄存器写入数据来自存储器	lw指令
5	RegDst	写入寄存器编号rt/rd选择	R型指令
6	AluSrcB	运算器B输入选择	lw指令，sw指令，立即数运算类指令
7	SignedExt	立即数符号扩展	ADDI、ADDIU、SLTI、LW、SW
8	JR	寄存器跳转指令译码信号	JR指令
9	JAL	JAL指令译码信号	JAL指令，选择寄存器写回编号，写回值
10	JMP	无条件分支控制信号	J、JAL、JR指令，选择无条件分支地址
11	Beq	Beq指令译码信号	Beq指令，用于有条件分支控制
12	Bne	Bne指令译码信号	Bne指令，用于有条件分支控制
13	Syscall	Syscall指令译码信号	根据$V0寄存器的值，决定是停机还是输出

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。并根据表格自动生成各个控制信号逻辑表达式。输入 logisim 生成相应的逻辑电路。

表 2.5 主控制器控制信号框架

#	指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	SLLV	BGTZ	SB	RsUsed	RtUsed
1	SLL	0	0	0				1			1										1
2	SRA	0	3	1				1			1										1
3	SRL	0	2	2				1			1										1
4	ADD	0	32	5				1			1									1	1
5	ADDU	0	33	5				1			1									1	1
6	SUB	0	34	6				1			1									1	1
7	AND	0	36	7				1			1									1	1
8	OR	0	37	8				1			1									1	1
9	NOR	0	39	10				1			1									1	1
10	SLT	0	42	11				1			1									1	1
11	SLTU	0	43	12				1			1									1	1
12	JR	0	8	x										1	1					1	
13	SYSCALL	0	12	x					1											1	1
14	J	2	X	x											1						
15	JAL	3	X	x				1							1	1					
16	BEQ	4	X	x								1								1	1
17	BNE	5	X	x									1							1	1
18	ADDI	8	X	5			1	1		1										1	
19	ANDI	12	X	7			1	1												1	
20	ADDIU	9	X	5			1	1		1										1	
21	SLTI	10	X	11			1	1		1										1	
22	ORI	13	X	8			1	1												1	
23	LW	35	X	5	1		1	1		1										1	
24	SW	43	X	5		1	1			1										1	1
25	SLLV	0	4	0				1			1						1			1	1
26	XORI	14	x	9			1	1												1	
27	BGTZ	7	x	X														1		1	
28	SB	40	x	5		1	1			1									1	1	1

2.2 中断机制设计

2.2.1 单级中断设计

添加 EPC 寄存器和 IE 使能寄存器，EPC 用来存储中断返回地址，IE 中断使能寄存器用来开中断和关中断。并通过优先选择器来确定中断源并将对应中断服务程序地址送入 PC。

2.2.2 多级嵌套中断设计

添加中断使能寄存器 IE、EPC 寄存器堆和中断服务寄存器堆 ISR，同时还需要设计好每个中断对应的中断屏蔽字。设计寄存器记录当前正在处理的中断源。同时对开中断指令和关中断指令做出完美的逻辑设计，改变 IE 寄存器的值。

2.3 流水 CPU 设计

2.3.1 总体设计

流水线 CPU 数据通路从左到右依次分成 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。每个段之间添加一个流水寄存器，根据其所连接的功能段的名称分别命名为 IF/ID、ID/EX、EX/MEM、MEM/WB，数据通路被被 4 个流水寄存器划分为五段流水线，这些流水寄存器采用统一时钟信号 CLK 进行同步，每来一个时钟，就会有一条新的指令进入流水线取指令 IF 段，同时流水寄存器就会锁存前段加工处理完成的数据和控制信号，为下一段的功能部件提供数据输入，指令流水线各功能段通过流水寄存器完成一次数据传送。

2.3.2 流水接口部件设计

不同的流水寄存器锁存传递的数据信息并不相同，但是他们总体设计结构基本相同。每个流水寄存器都是由一系列不同位数的寄存器组成，它们由同一时钟信号控制，共享使能端和同步清零信号。在指令流水线中，通过流水线寄存器传递的不仅仅是当前指令当前阶段待加工的数据，还需要向后段传递数据如何进行加工的操作控制信号，这些控制信号要应与各功能段处理的指令同步，每一个操作控制信号均只在某一功能段使用一次，操作控制信号使用完毕后就不再向后段继续传递。

2.3.3 理想流水线设计

理想流水线不考虑数据冲突、控制冲突和结构冲突，程序只顺序执行，流水寄存器在时钟信号的控制下，不断的将前一阶段的结果和控制信号向后传递。

2.4 气泡式流水线设计

✧ 结构冲突处理

计算分支目标地址、运 算器运算都需要使用运算器，取指令和读数据都要使用存储器，这些都是结构冲突。解决方案是增设加法部件避免运算冲突，增设指令存储器避免访存冲突。

✧ 控制冲突处理

分支指令会引起控制冲突，要解决控制冲突，在执行程序分支跳转时必须清除流水线中的分支指令后续的若干条误取指令。解决方案是将分支跳转信号 BranchTaken 送到 IF 段多路选择器选择控制端，选择分支目标地址送程序计数器 PC，同时将 BranchTaken 信号送入 IF/ID、ID/EX 流水寄存器同步清零端，清除误取的指令。

✧ 数据冲突处理

在 MIPS 五段流水线中，ID 段从寄存器堆取操作数时可能会发生数据相关，只需要考虑 ID 段指令 EX、MEM、WB 段的前三条指令之间的数据相关性。ID 段和 WB 段的数据相关可以采用先写后读的方式解决，寄存器堆写入控制采用下跳沿触发。增加硬件逻辑实现 ID 段与 EX、MEM 段指令的数据相关性检测，MIPS 指令包括 0~2 个源操作数，分别是 rs、rt 字段对应的寄存器，其中 0 号寄存器恒零，不需要考虑相关性。要想确认 ID 段指令使用的源寄存器是否在前两条指令中写入，只需要检查 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。

2.5 数据重定向流水线设计

除 Load 类访存指令外，大多数指令得目的操作数都已实际存放在 EX/MEM、MEM/WB 流水寄存器中，可以直接将正确的操作数从其所在位置重定向到 EX 段合适的位置。增加多路选择器 FwdA 和 FwdB，和选择控制信号 RS_F，来控制选择算术逻辑运算单元 ALU 的操作数。如果相邻两条指令存在数据相关，且前一条指令是访存指令时（称为 Load-Use 相关），此时要插入一个气泡来解决。故需要设计相关逻辑电路判断是否为 load-Use 相关。

3 详细设计与实现

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

使用一个 32 位寄存器实现程序计数器 PC，触发方式为上升沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和 Go 信号相或。当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，与 Go 信号相或，屏蔽时钟信号，使整个电路停机。直到点击 Go 按钮，程序才继续向前运行，如图 3.1 所示。

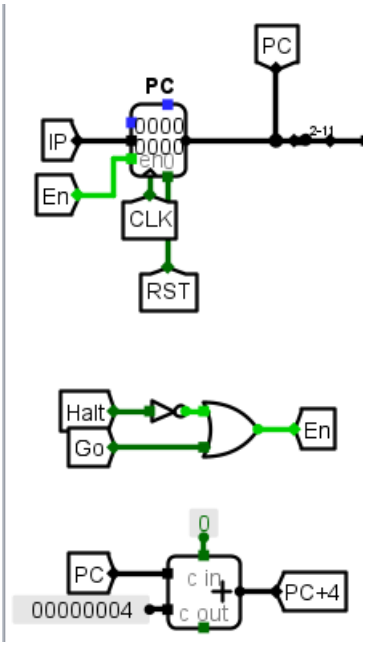

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

使用一个只读存储器 ROM 实现指令存储器（IM）。设置该只读存储器的地址位宽为 10 位，数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位，而 ROM 地址线宽度有限，仅为 10 位，故将 32 位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器只取 32 位指令地址的 2-11 位作为指令存储器的输入地址。如图 3.2 所示。

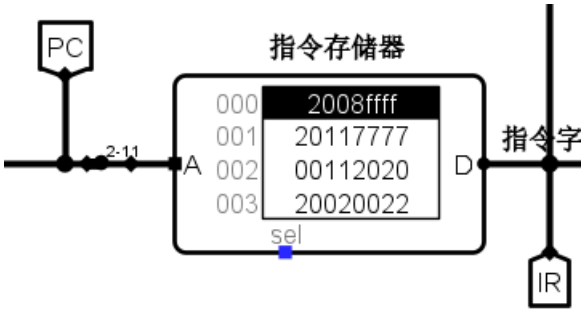

图 3.2 指令存储器（IM）

3) 通用寄存器堆

RegFile 为通用寄存器堆，里面有编号 0~31 的 32 位寄存器。在读寄存器时，R1# 和 R2#（5 位）输入对应寄存器编号，R1 和 R2（32 位）输出对应寄存器的值，该部分为组合逻辑，不受时钟控制。将数据写入寄存器时，W#（5 位）输入需要写入寄存器的编号，Din（32 位）输入要被写入的数据，当写使能 WE 为 1 时，在时钟信号 clk 的控制下，将对应数据写入寄存器，该部分是时序逻辑。

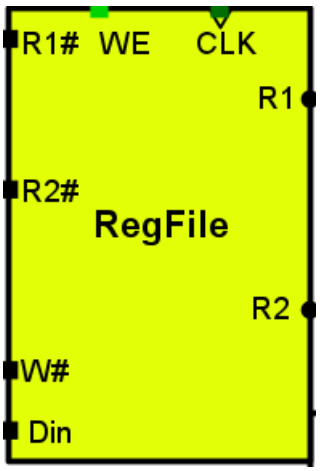

图 3.3 通用寄存器堆

4) 算术逻辑运算单元

AluOP（5 位输入）为运算器运算的功能号，其功能见表 2.1，A 和 B（32 位输入）为运算器的两个操作数，shamt 为位移运算时位移的位数。R(32 位)输出运算的结果。

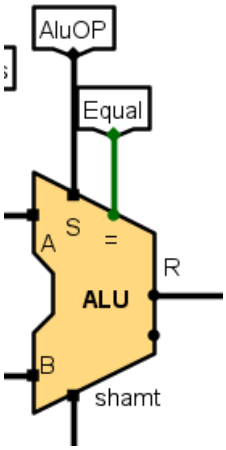

图 3.4 ALU 算术逻辑运算单元

3.1.2 数据通路的实现

1) 指令字分解

根据 MIPS 指令字的特点，通过分线器将指令字不同的字段分解出来，如图 3.5 所示

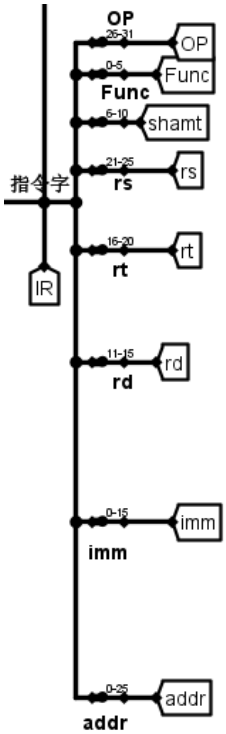

图 3.5 指令字分解

2) 根据指令字分解出的字段和单周期硬布线控制器产生的控制信号，可以设计出如图 3.6 所示的电路。RegFile 为通用寄存器堆，读出编号为 rs、rt 对应寄存器的值，由于本实验增加了一个特殊指令 syscall，其功能是当寄存器$v0（编号为 2）的值不等于 10 时，则将寄存器$a0(编号为 4)的值显示出来，故需要需要在 R1#和 R2#输入端增加多路选择器，并用 sycall 信号作为控制信号。RegDst 作为多路选择器的控制信号，控制写入寄存器的编号是 rt 还是 rd，但是当指令为 jal 时，需要将 PC+4 的值存入 31 号寄存器，故需要由 jal 控制的多路选择器控制要写入寄存器的编号和数据。AluSrcb 信号控制多路选择器选择 ALU 的第二个操作数是扩展后的立即数。sllV 信号只对移位指令有效，是用来控制移位位数是指令字的 shamt 字段还是 GPRrs 的低五位字段。ALU 计算出结果，取 2-11 位作为地址读出数据存储器的值。MemToreg 作为多路选择器的控制信号，为 0 时直接将 Alu 的运算结果作为 RDin 送入寄存器堆的写入数据，为 1 时，将从数据存储器读出的值送到寄存器中去。SB 为按字节存储指令。设计思路是取地址的 2-11 位作为地址读出数据存储器的 32 位数据，然后用地址的 0-2 位作为选择信号，选择 32 位数据哪个字节的 8 位数据修改为 GPRrt[0:8]的值。SB 组件具体见图 3.7，sdata 为数据存储器读出的原 32 位数据，Sel 为地址，cdata 是需要写入存储器中要改变的 8 位数据。BranchAddr 为分支指令的目标地址。

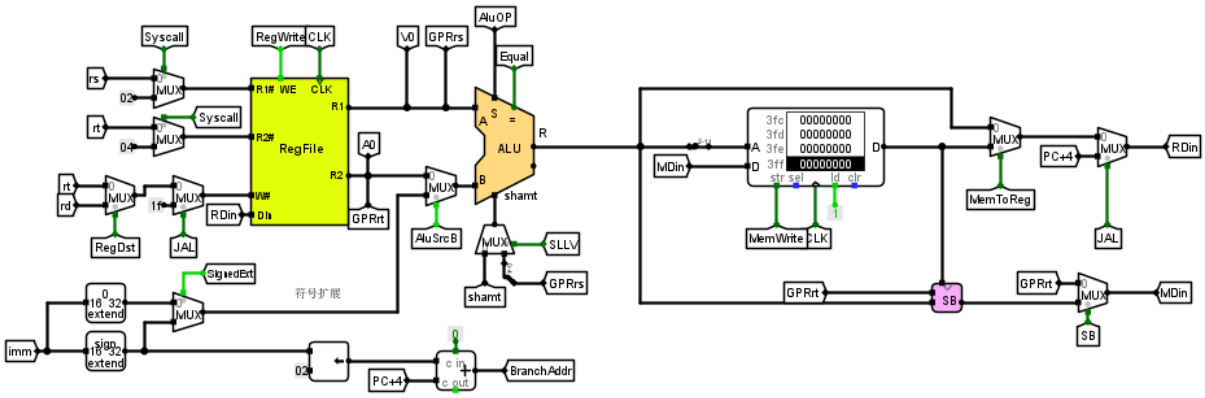

图 3.6 单周期 CPU 顶层设计

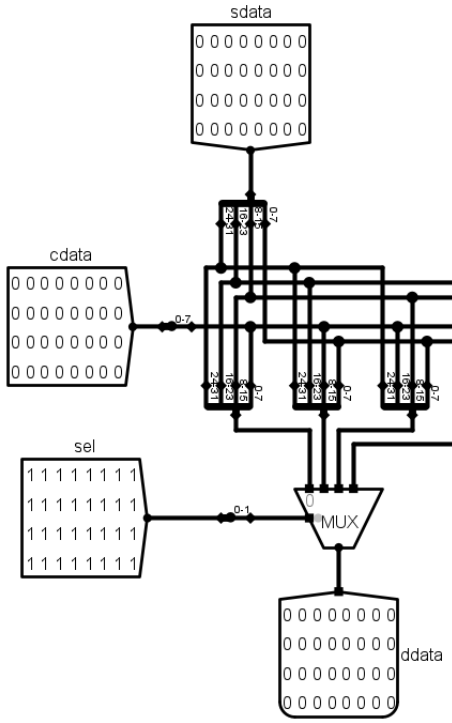

图 3.7　SB 组件的具体实现

3) 分支指令实现部件如图 3.8，多路选择器控制送入 pc 寄存器的地址 IP。

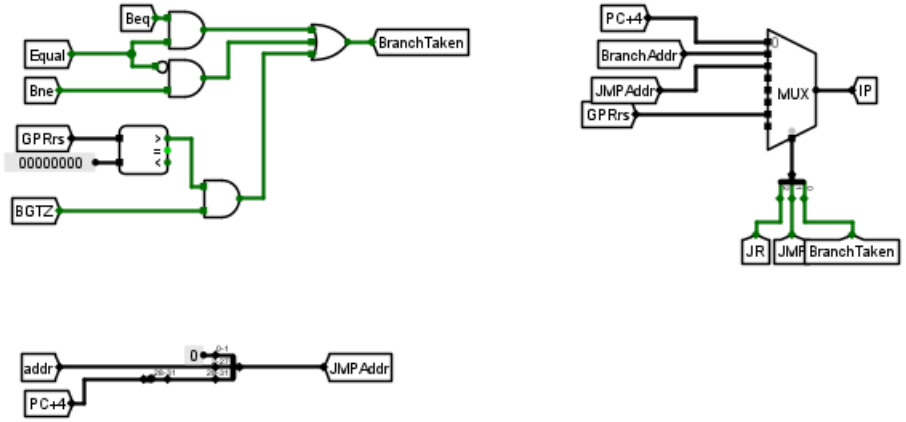

图 3.8 分支指令实现部件

4) Syscall 指令实现如图 3.9，当 V0 不为 10 时，显示寄存器的使能端 LED_EN 为 1，当时钟到来时将数据 A0 存入寄存器并显示。当 V0 为 10 时，停机信号 halt 为 1，halt 信号取反后借入 PC 寄存器使能端，完成 cpu 停机。

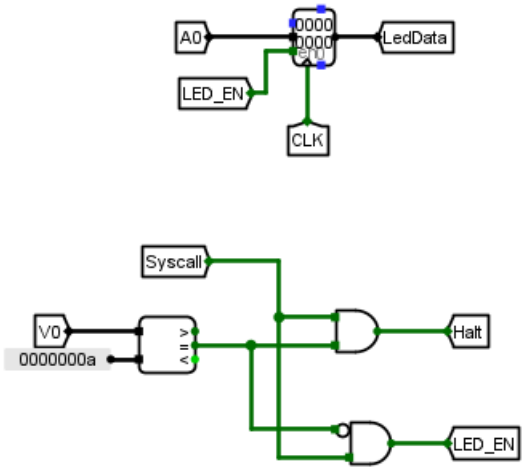

图 3.9 syscall 指令实现逻辑

3.1.3 控制器的实现

OpCode 和 FUNCT 字段作为输入，产生的控制信号作为输出，可以填写如表 3.1 所示的表格，根据表格自动生成的控制信号与 OpCode 和 FUNCT 逻辑关系，生成对应的逻辑电路。

表 3.1 主控制器控制信号

#	指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	SLLV	BGTZ	SB	RsUsed	RtUsed
1	SLL	0	0	0				1			1										1
2	SRA	0	3	1				1			1										1
3	SRL	0	2	2				1			1										1
4	ADD	0	32	5				1			1									1	1
5	ADDU	0	33	5				1			1									1	1
6	SUB	0	34	6				1			1									1	1
7	AND	0	36	7				1			1									1	1
8	OR	0	37	8				1			1									1	1
9	NOR	0	39	10				1			1									1	1
10	SLT	0	42	11				1			1									1	1
11	SLTU	0	43	12				1			1									1	1
12	JR	0	8	x										1	1					1	
13	SYSCALL	0	12	x					1											1	1
14	J	2	X	x											1						
15	JAL	3	X	x				1							1	1					
16	BEQ	4	X	x								1								1	1
17	BNE	5	X	x									1							1	1
18	ADDI	8	X	5			1	1		1										1	
19	ANDI	12	X	7			1	1												1	
20	ADDIU	9	X	5			1	1		1										1	
21	SLTI	10	X	11			1	1		1										1	
22	ORI	13	X	8			1	1												1	
23	LW	35	X	5	1		1	1		1										1	
24	SW	43	X	5		1	1			1										1	1
25	SLLV	0	4	0				1			1						1			1	1
26	XORI	14	x	9			1	1												1	
27	BGTZ	7	x	X														1		1	
28	SB	40	x	5		1	1			1									1	1	1

3.2 中断机制实现

3.2.1 单周期+单级中断

单周期单级中断机制实现如图 3.10，初始状态时中断使能寄存器被置为 1，表示开中断，当发生中断时，Int 信号被置为 1，进入中断响应周期，当时钟信号到来时，中断返回地址被送入 EPC 寄存器而中断服务程序地址送入 PC 寄存器，同时 IE 寄存器关中断，执行中断服务程序时，由于 IE 寄存器始终为 0，故执行中断服务程序不会被中断。直至中断返回指令 eret 执行时，IE 寄存器才被置为 1，开中断，同时将中断返回地址 ret_addr 送入 PC 寄存器，并将对应中断信号清 0。若有多个中断等待，通过优先编码器选择优先级最高的中断，进行中断响应。

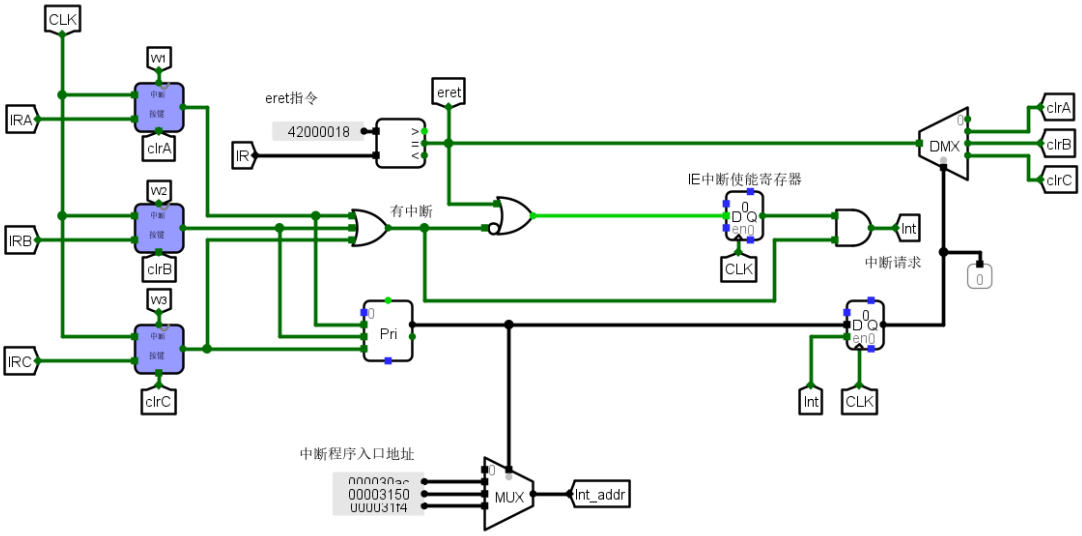

图 3.10 单级中断实现

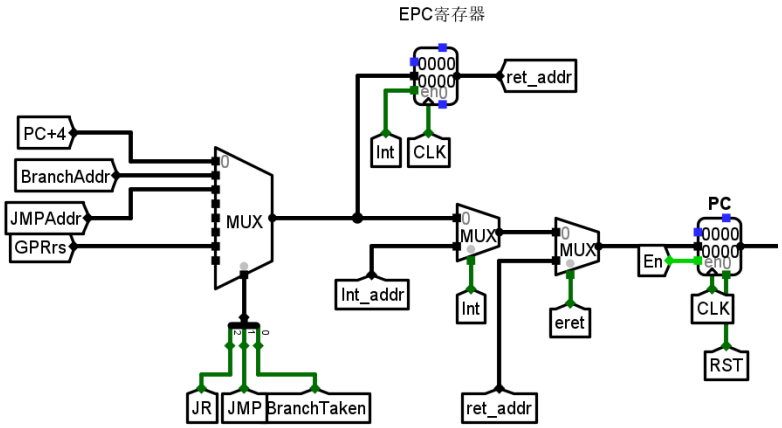

图 3.11 单级中断实现

3.2.2 单周期+多级中断

多级嵌套中断设计如图 3.12 所示，为了实现多级嵌套中断，我们用硬件实现了 EPC 堆栈。初始状态时中断使能寄存器被置为 1，表示开中断，当发生中断时，Int 信号被置为 1，进入中断响应周期，关中断并根据中断源 newstate 将中断返回地址送入对应的 EPC 寄存器。当时钟信号上升沿到来时，将中断源 newstate 送入中断源寄存器。同时将中断服务程序入口地址送入 PC 寄存器。进入中断服务程序后保存现场，当执行 mfco 开中断指令时，IE 中断使能寄存器被置为 1，表示开中断。nowstate 表示正在执行中断服务程序的中断号，nowstate 作为多路选择器的选择信号，将当前状态下的中断屏蔽字输出，并和中断信号与操作，由于中断使能寄存器为 1，此时未被屏蔽的中断源可以继续相应。ISR 是中断服务寄存器，他是用来记录被嵌套的中断源。当一个中断服务程序执行完毕。会执行 eret 指令，它会开中断，使得 IE 为 1，同时清除当前执行完毕的中断。如果还有中断没有处理完，则将等待的中断地址送入 PC 寄存器。

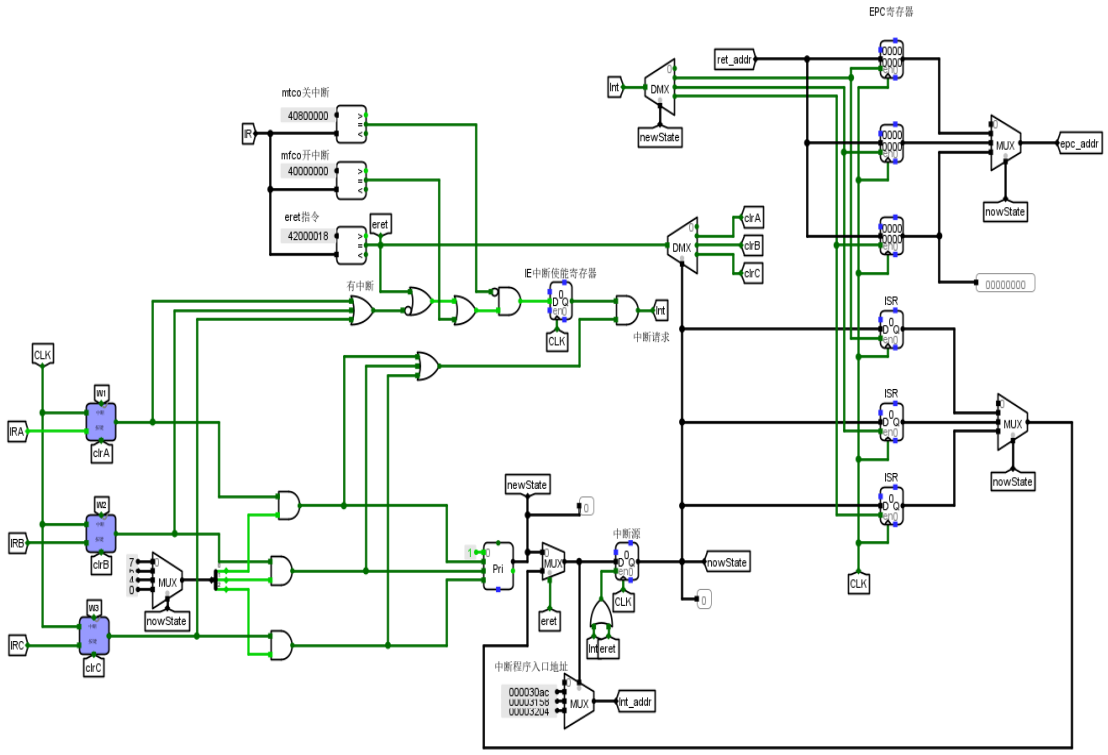

图 3.12 多级嵌套中断实现

3.2.3 流水线+单级中断

气泡流水线单级中断和单周期单级中断几乎完全相同，唯一不同之处如图 3.13 红色框内之处。由于发生中断时，PC 寄存器可能由于 EX 段的数据冲突而被暂停，导致正确的中断程序入口地址并没有送入 PC 寄存器，解决方案是 IE 中断使能寄存器的使能端要和 PC 寄存器的使能端接同样的信号 En。当 PC 寄存器被暂停时，中断使能寄存器也必须暂停。

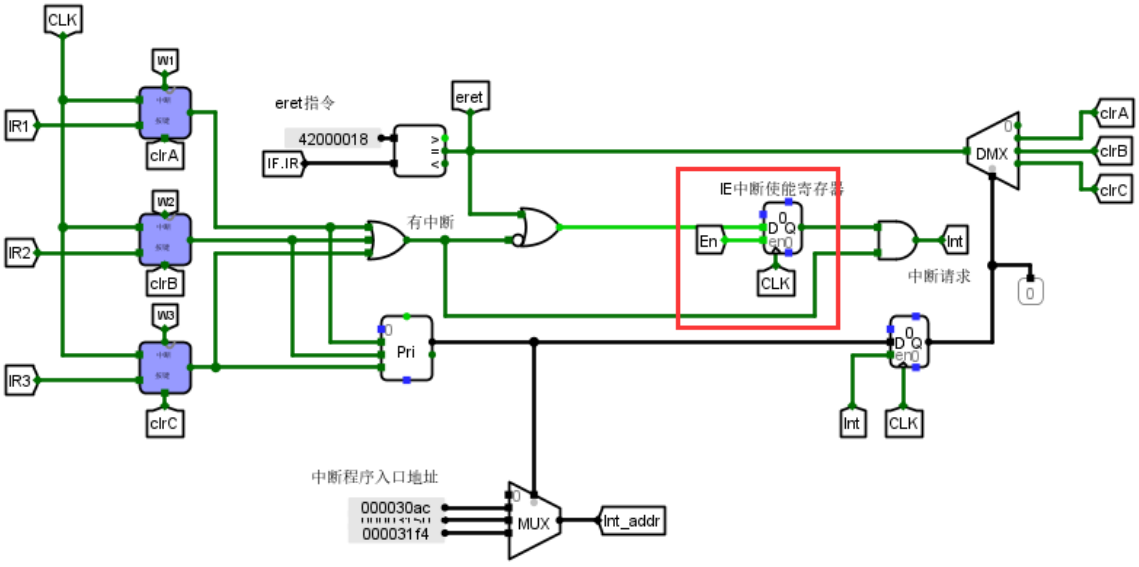

图 3.13 流水线中断单级中断实现

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口部件实现如图 3.14 所示，由一系列不同位数的寄存器排列组成，寄存器用来存储各类需要传递的数据和控制信号。同步清零端通过使用二路选择器来实现，当 reset 为 1 时，在时钟上升沿将 0 送入寄存器。所有寄存器由统一时钟信号 clk 控制,并且拥有公共的使能端 en。

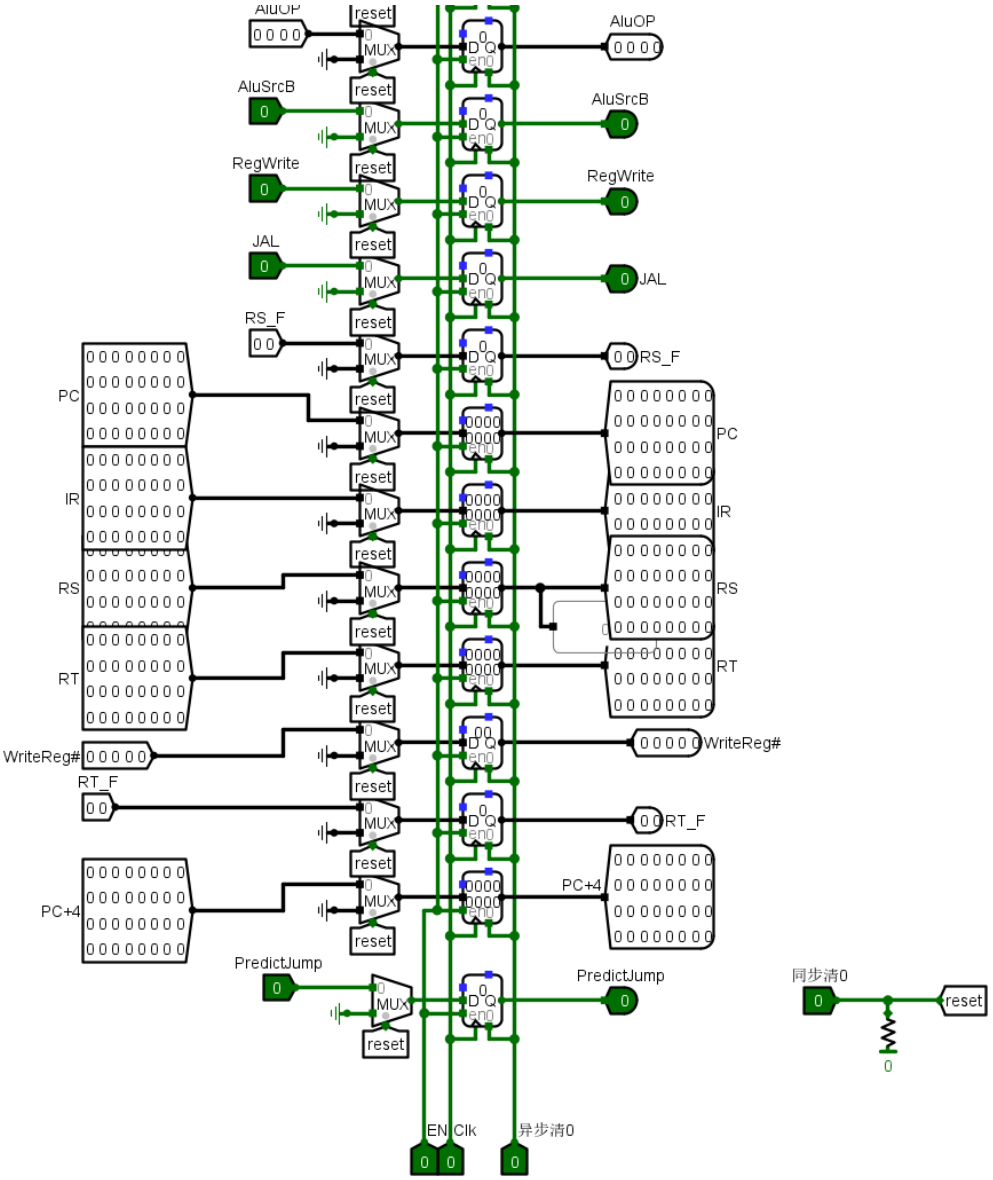

图 3.14 流水接口部件实现

3.3.2 理想流水线实现

理想流水线顶层设计如图 3.15， CPU 数据通路从左到右依次分成 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。每个阶段用一个流水寄存器分开。总体设计跟单周期类似，但是需要注意的是写数据时，寄存器堆写寄存器编号 W#的输入来源是根据 ID 段的指令字由 RegDst 信号控制多路选择器进行选择的，而写数据 WD 却来自于 WB 段，这样会数据混乱。解法办法是调整 ID 段多路选择器输出的写寄存器编号的输出位置，其不再送寄存器堆的 W#端，而是直接送 ID/EX 流水寄存器锁存，并逐段依次向后传递到 WB 段，最后再由 WB 段的 MEM/WB 流水寄存器送回到寄存器堆的写寄存器编号 W#端口。

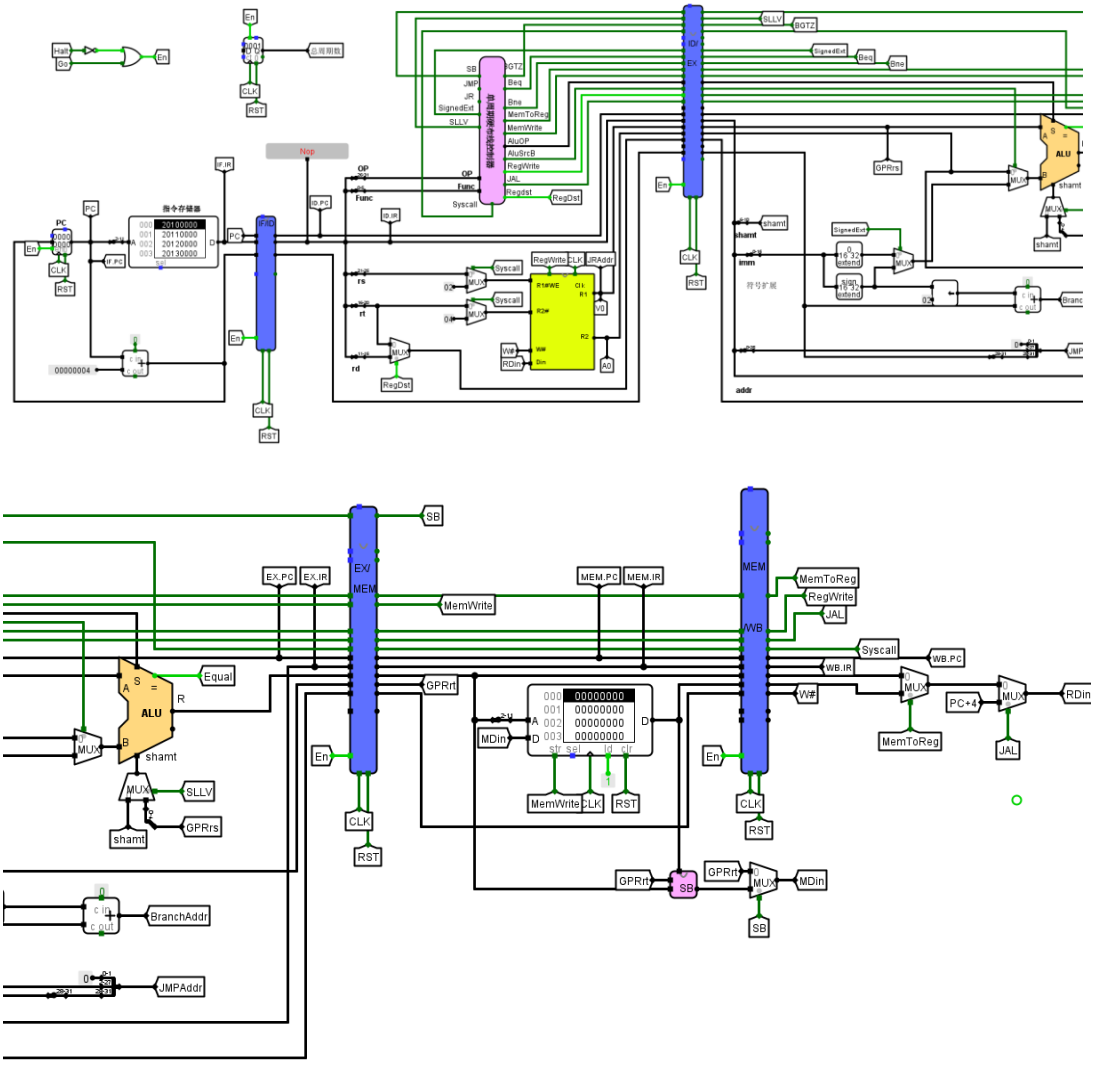

图 3. 15 理想流水线顶层设计

3.4 气泡式流水线实现

气泡流水线相对于理想流水线主要增加了数据相关处理逻辑，解决数据冲突和控制冲突。寄存器堆写入控制采用下跳沿触发，而所有流水寄存器采用上跳沿触发，这样就解决 ID 段和 WB 段之间的数据相关。数据相关处理逻辑顶层设计如图 3.16，输入为 ID 段指令 ID.IR、EX.RegWrite、EX_W#、BranchTaken、MEM_RegWrite、MEM_W# 信号，输出信号为 stall 和 flush。

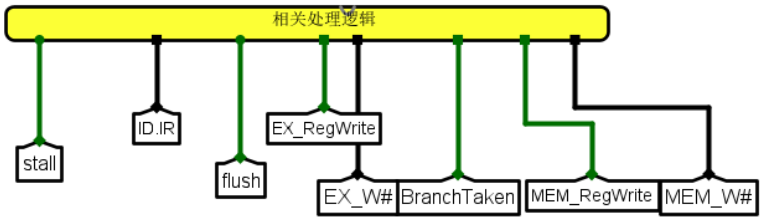

图 3.16 数据相关处理逻辑顶层设计

要想确认 ID 段指令使用的源寄存器是否在前两条指令中写入，只需要检查 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，因此流水线中的数据相关检测逻辑如下：

DataHazzard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#) +
RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#) +
RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#) +
RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)

- ✧ rs、rt 分别表示指令字中的 rs、rt 字段，
- ✧ RsUsed、RtUsed 分别表示 ID 段指令需要读 rs、rt 字段对应的寄存器
- ✧ EX.RegWrite 表示 EX 段的寄存器堆写使能控制信号 RegWrite，
- ✧ #MEM.WriteReg 表示 MEM 段的写寄存器编号 WriteReg

数据相关处理逻辑的具体设计实现如图 3.17，当数据冲突（dataHazzard 为 1）或者分支跳转（branchTaken 为 1）时，flush 置为 1，将清空 ID/EX 流水寄存器。需要特别注意的是 syscall 指令也应该是 Rs_Used = 1、Rt_Used = 1 且 rs = 2、rt = 4。

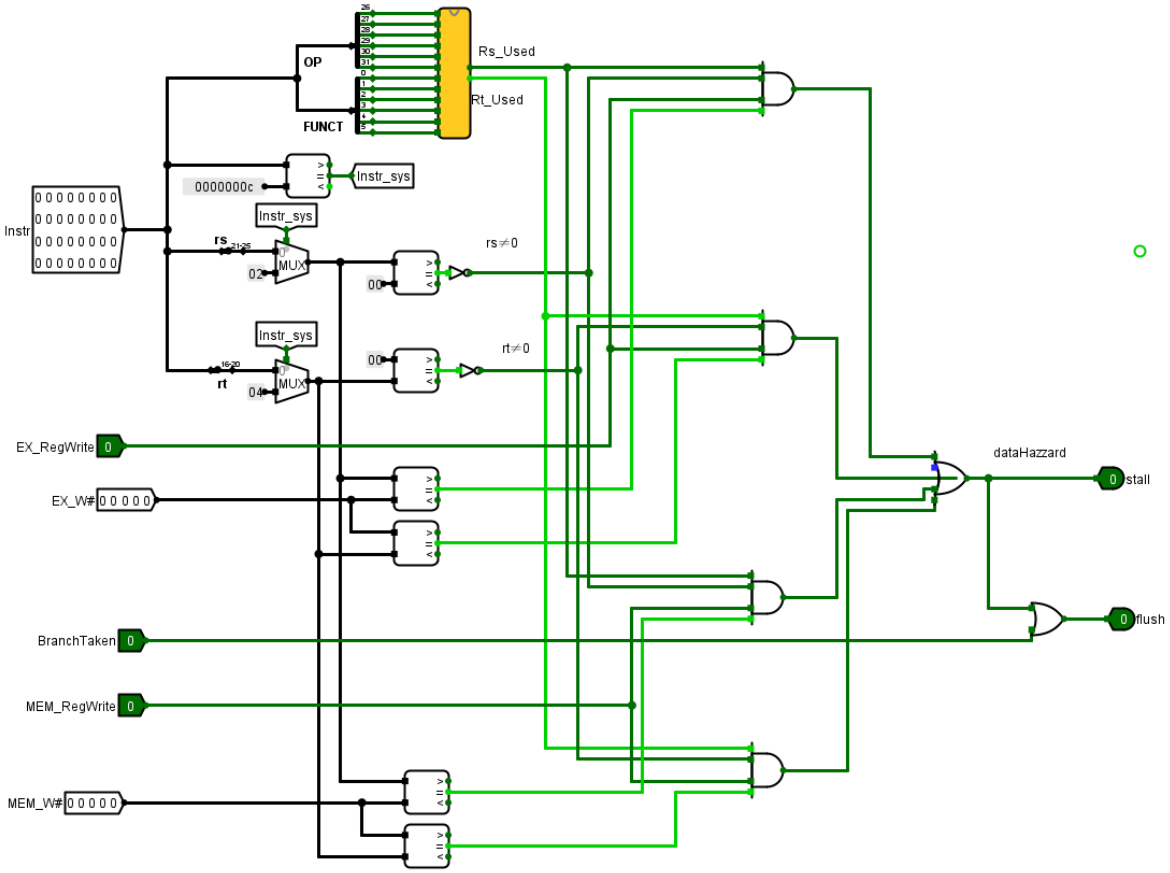

图 3.17 数据相关处理逻辑具体实现

有了数据相关检测逻辑，只需考虑如何暂停 IF、ID 段指令的执行以及如何插入气泡的问题，插入气泡可以参考控制相关中的流水清空信号 Flush，当发生数据相关时给 ID/EX 流水寄存器一个同步清空信号 Flush 即可；而要暂停 IF、ID 段指令执行，只须保证程序计数器 PC 的和 IF/ID 流水寄存器的值不变即可，要做到这一点，只需要控制寄存器使能端即可，当使能端为 1 时，寄存器正常工作，为 0 时则忽略时钟输入，寄存器值保持不变。进一步综合控制冲突处理逻辑可知流水线清空信号 IF/ID.Flush、ID/EX.Flush，以及流水线阻塞暂停信号 Stall 的逻辑表达式如下：

Stall = DataHazzard # 数据相关时要阻塞暂停 IF、ID 段指令的执行

PC.EN = IF/ID.EN =~Stall

IF/ID.CLR = BranchTaken # 出现分支跳转时要清空

IF/ID ID/EX.CLR = Flush = BranchTaken + DataHazzard # 出现分支或数据相关时要清空 ID/EX

气泡流水线顶层设计如图 3.18 所示，需要注意的是 PC 寄存器和 IF/ID 流水寄存器的使能端 en，除了受 stall 信号控制外，还应受到停机信号 Halt 和 Go 信号的控制，所以 En 的控制逻辑应为 En = (~Halt & ~stall) + Go，而其他流水寄存器不受 stall 信号控制，只受停机信号 Halt 和 Go 信号的控制，所以它们的使能端信号 h_en = ~stall + Go。

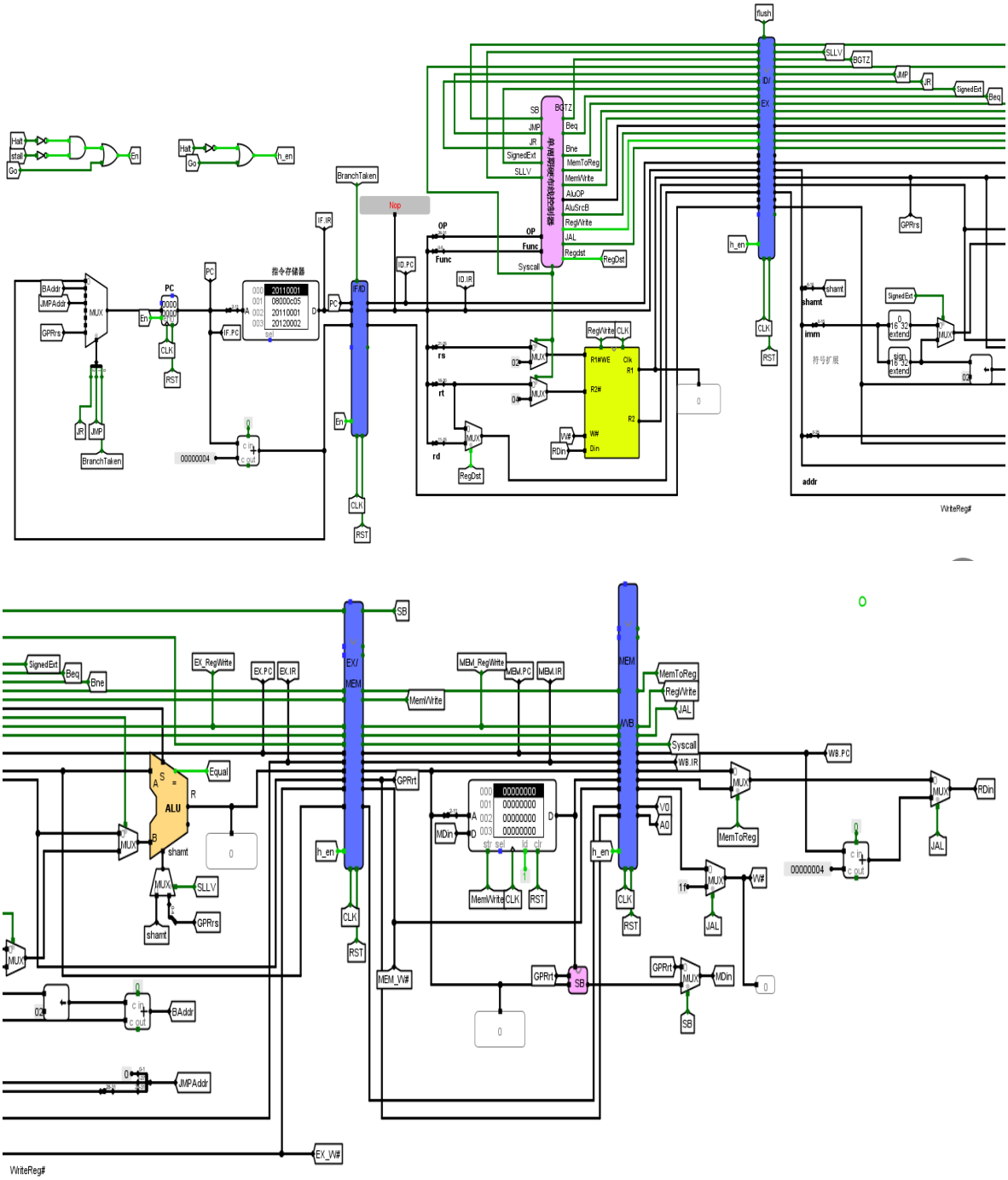

图 3.18 气泡流水线顶层设计

3.5 重定向流水线实现

重定向流水线相对于理想流水线主要增加了重定向相关处理逻辑，解决数据冲突和控制冲突。寄存器堆写入控制采用下跳沿触发，而所有流水寄存器采用上跳沿触发，这样就解决 ID 段和 WB 段之间的数据相关。数据相关处理逻辑顶层设计如图 3.19，输入信号为 ID 段指令 ID.IR、EX.RegWrite、EX_W#、MemRead、MEM_RegWrite、MEM_W#，输出信号为 stall、flush、RS_F 和 RT_F。

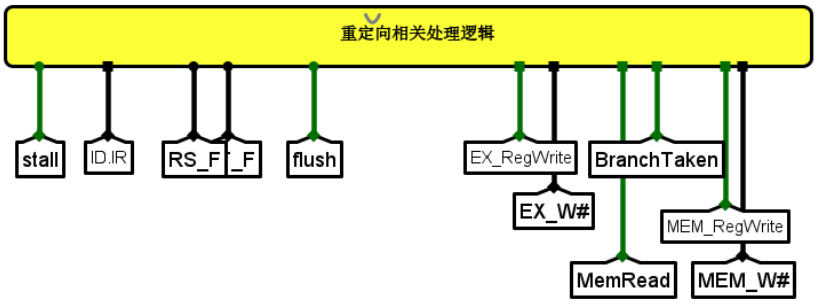

图 3.19 重定向相关处理逻辑顶层设计

对于 Load-Use 数据相关，不能采用重定向方式解决数据冲突，采用重定向会增加关键路径延迟，降低 CPU 的时钟频率，解决办法是在 Load-Use 的两条相邻指令之间强制插入一个气泡以消除这种相关。所以需要设计相关处理逻辑检测出 Load-Use 相关，其逻辑表达式如下：

LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#) + RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)

由于 MemRead 信号和 MemToReg 信号是同步的，所以可以用 MemToReg 信号代替 MemRead 信号。当发生 Load-Used 相关时，需要暂停 IF、ID 段指令执行、并在 EX 段插入气泡，需要控制 PC 使能端 EN、IF/ID 使能端 EN、ID/EX 清零端 CLR，而 EX 段执行分支指令时会清空 ID 段、EX 段中的误取指令，会使用 IF/ID 清零端 CLR、ID/EX 清零端 CLR。综合两部分逻辑，可以得到相关处理逻辑阻塞信号 Stall、清空信号 Flush，各控制端口的逻辑： Stall = LoadUse

F/ID.CLR = BranchTaken

IF/ID ID/EX.CLR = Flush = BranchTaken + LoadUse

ID/EX PC.EN = ~Stall

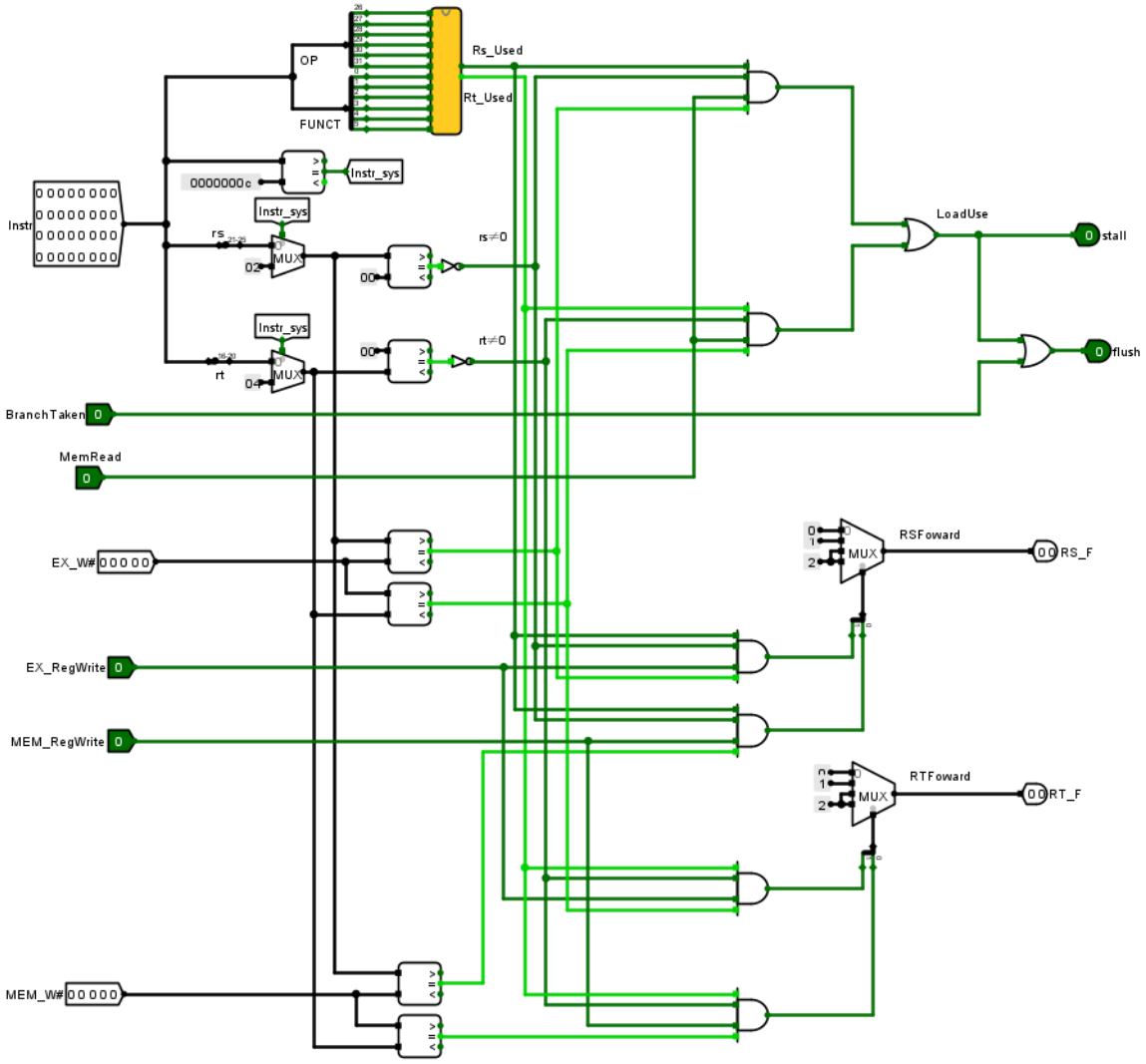

图 3. 20 重定向相关处理逻辑具体实现

如图 3.20 重定向相关处理逻辑的具体实现，除 Load-Use 数据相关，其他数据相关都可以采用重定向方式以无阻塞的方式解决，相关处理逻辑只需要在 ID 段生成两个重定向选择信号 RsFoward、RtFoward 传输给 ID/EX 流水寄存器即可，以 RsFoward 为例，其赋值逻辑如下：

```
IF (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))
    RsFoward = 2            # ID 段与 EX 段数据相关
else IF (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))
    RsFoward = 1            # ID 段与 MEM 段数据相关
else
    RsFoward = 0            # 无数据相关
```

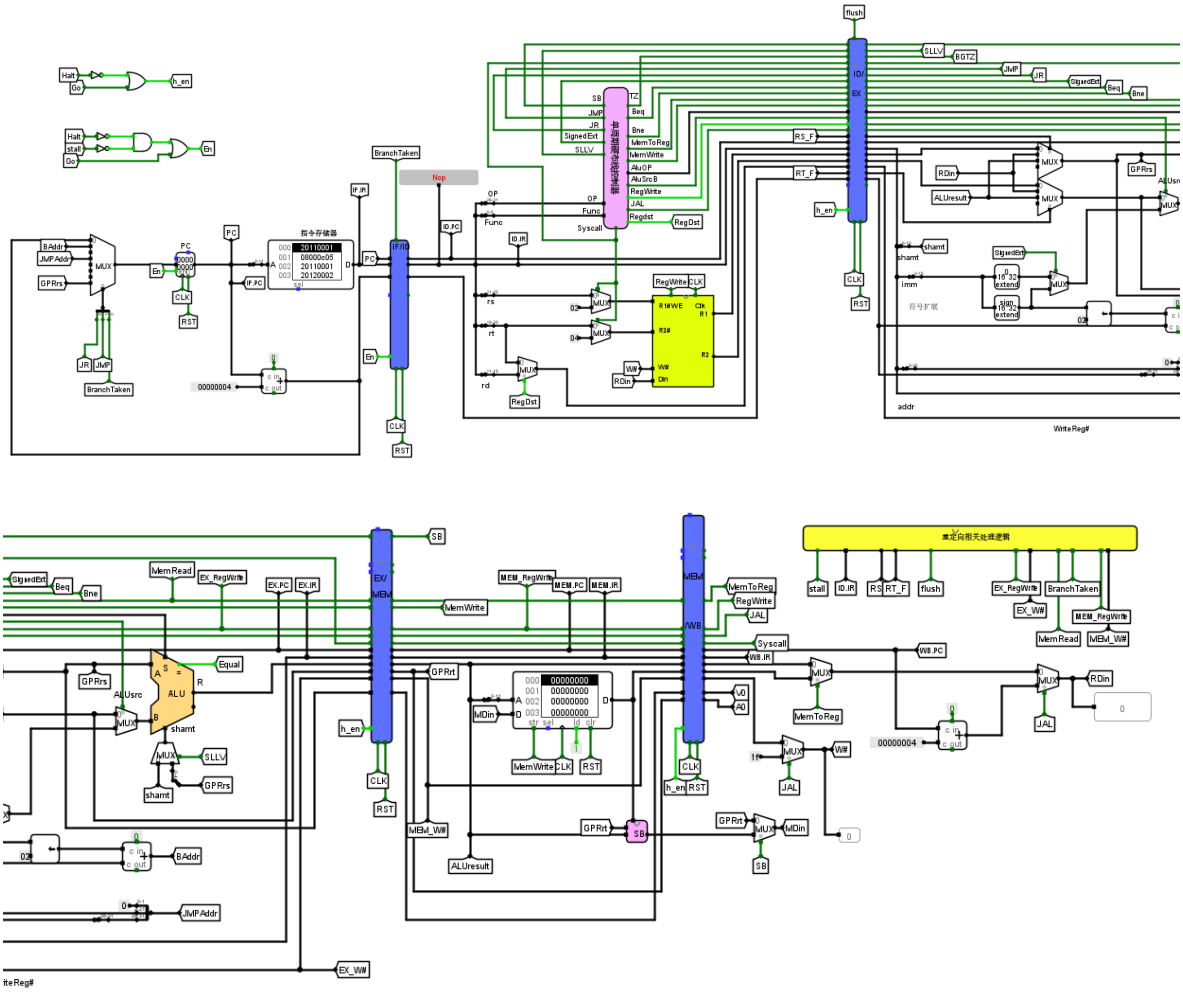

图 3.21 重定向流水线顶层设计

3.6 动态分支预测机制实现

动态分支预测依据分支指令的分支跳转历史，不断的对预测策略进行动态调整。它利用的是程序中分支指令的分支局部性。其主要实功能部件是分支预测缓冲器（Branch Prediction Buffer)，它用于存放分支指令的分支跳转历史统计信息。BTB 表每个表项主要包括 valid 位、分支指令地址、分支目标地址、历史跳转信息描述位（预测状态位)、置换标记五项。其中 valid 位用于标记当前表项是否有效。历史跳转信息描述位为 2 位，它的高位为 1 时表示预测跳转，为 0 表示预测不跳转。置换标记是当 BTB 表满时，用来淘汰数据的标志。BTB 表本质上是一个全相联的 cache，表项为 8， 用于缓存经常访问的分支指令的分支跳转历史统计信息。 典型的双位预测位状态转换图如图 3.22 所示。当状态位为 00、01 时预 测不跳转，为 10、11 时预测

跳转，当前分支指令是否发生跳转将会决定状态的变迁。

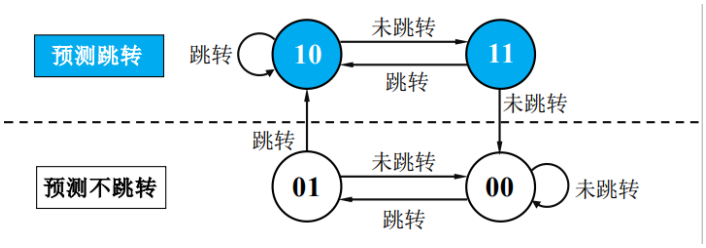

图 3.22 双预测位状态转换图

BTB 具体实现如图 3.23 所示， IF 段 PC 值在 BTB 表中进行全相联比较，若数据命中，根据对应表项中的分支预测历史位的值输出预测跳转信息 PredictJump，预测历史位为 10、11 时 PredictJump=1，表示程序要跳转执行，同时 BTB 表还要输出该指令的分支目标地址。如未命中，PredictJump=0，程序顺序执行，这是 BTB 读逻辑。当 EX 段执行分支指令时，也就是 EX.Branch=1 时，BTB 表会根据 EX.PC 全相联查找。如数据缺失，WriteLx 就会被置为 1，将当前分支指令的信息载入 BTB 表， 如果 BTB 表已满，会根据置换标记中数据采用 LRU 算法进行淘汰置换；如果数据命中，会将置换标记清零，并根据 EX 段指令实际分支跳转情况 EX.BranchTaken 的值依照预测位状态机更新分支预测历史位值，这是 BTB 写逻辑。

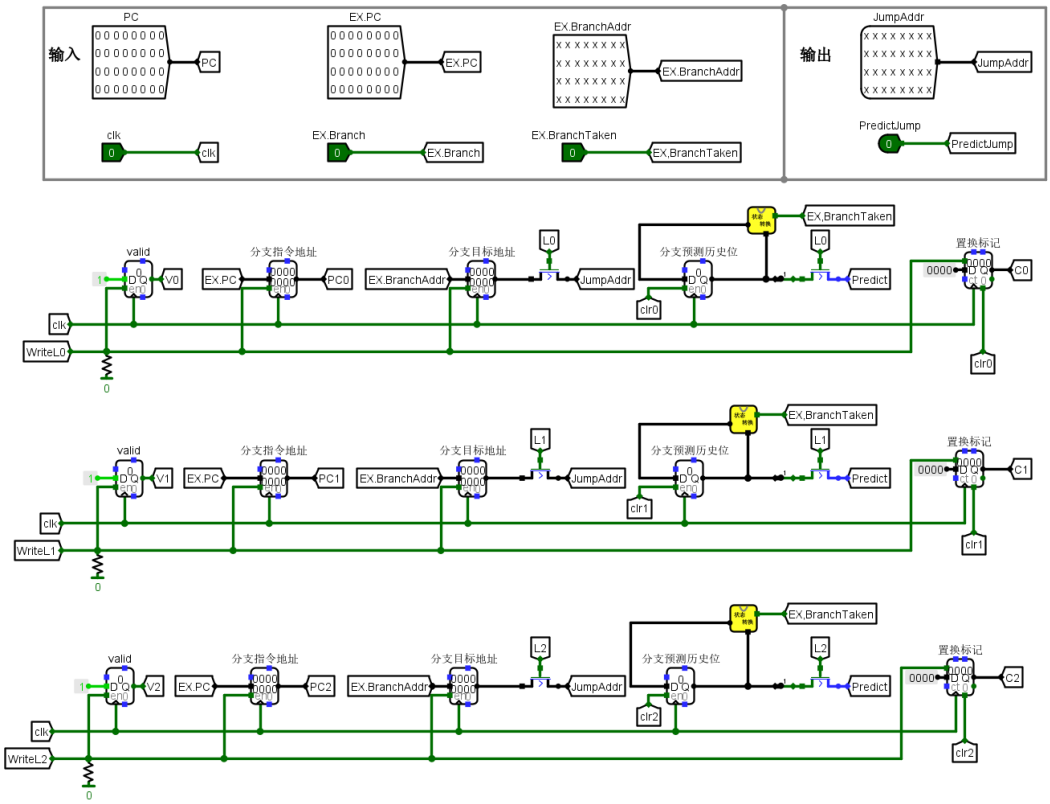

图 3.23 BTB 核心设计

BTB 读逻辑如图 3.24 所示，当有效位 V…为 1 时且 PC 命中，则将对应行的分支预测信号输出，若未命中，hit = 0,PredictJump = 0。

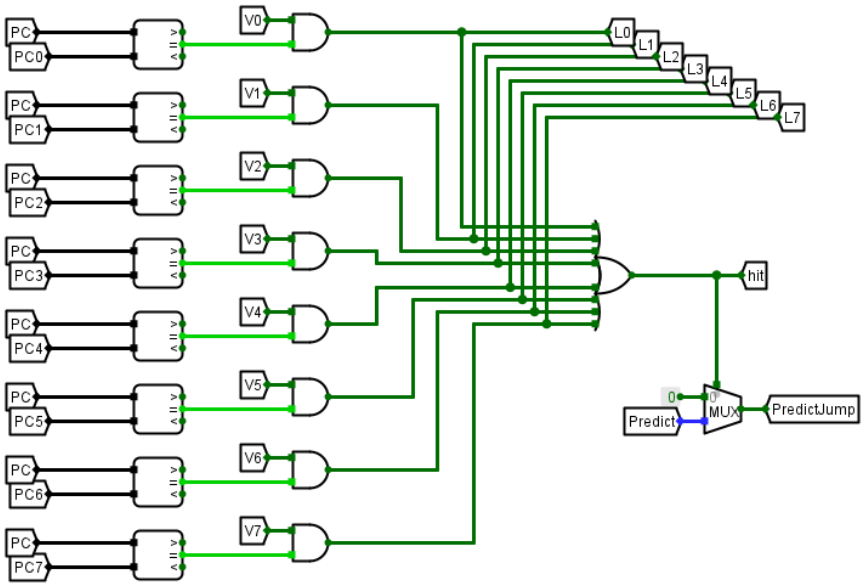

图 3. 24 BTB 读逻辑

BTB 写逻辑如图 3.25 所示，当 EX 段为分支指令即 EX.Branch 为 1 时，若命中 BTB，则对应行的 clrx 置为 1，将置换标志位清 0，同时 clrx 作为分支预测历史位的使能位，此时在时钟信号 clk 的控制下会更新分支预测历史位。若未命中，w_miss 为 1，之后根据淘汰选择机制，选择合适的一行，将分支信息写入 BTB 表中。

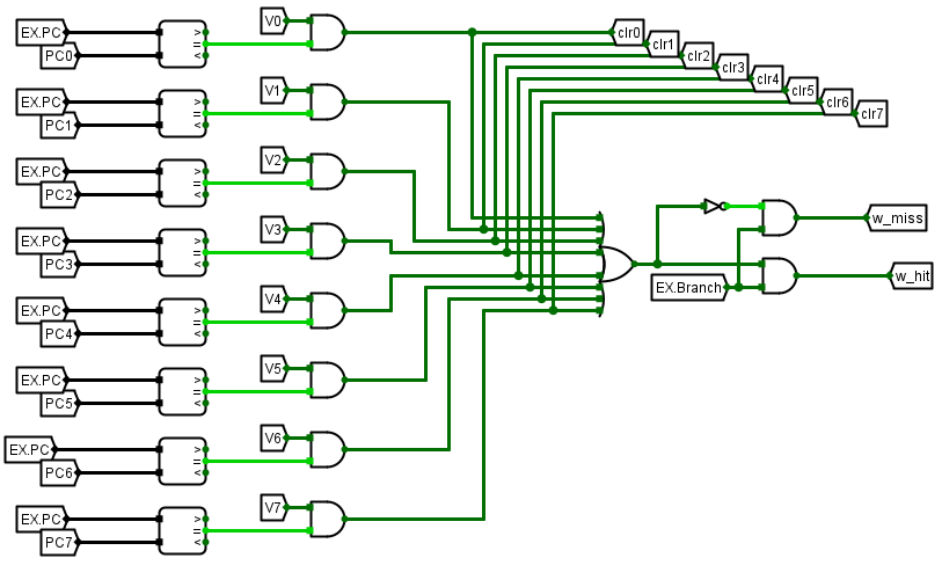

图 3. 25 BTB 写逻辑

BTB 淘汰逻辑如图 3.26 所示，每一行的有效位取反后跟该行的置换标志位通过分线器构成一个 17 位的数据，每一行的数据跟它们的行号一起经过无符号比较器，选择出最大数据和对应的行号，进行淘汰置换。若 BTB 未满，则至少有一行有效位 Vx 为 0，取反后为 1，且它位于 17 位数据最高位，所以它最大，该行一定会被置换写入新的数据；若 BTB 已满，则所有的有效位为 1，取反后为 0，所以置换标志位最大的那一行将会被置换掉，如此实现 LRU 置换算法。

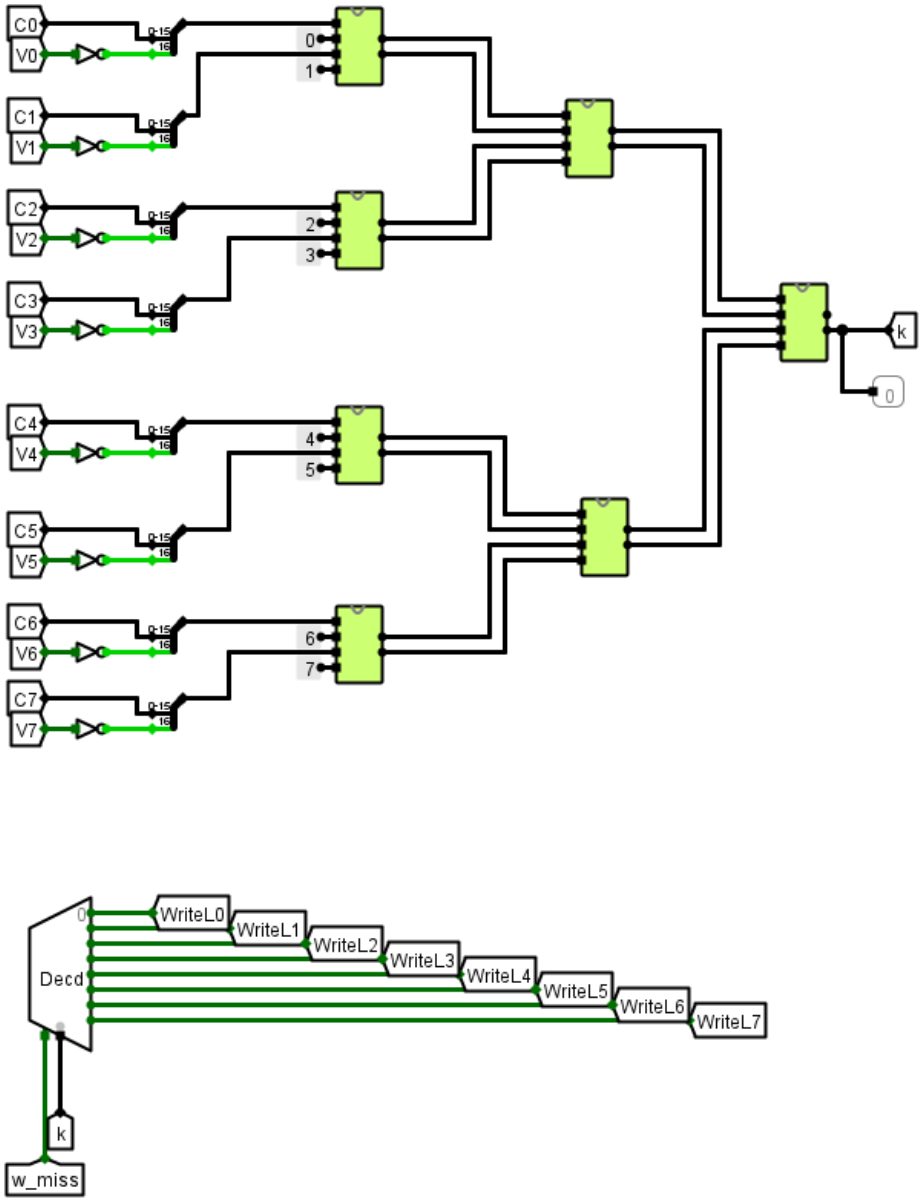

图 3.26 BTB 淘汰逻辑

分支预测顶层设计如图 3.27 所示，BTB 表放在 IF 段，利用 PC 的值作为关键字进行全相联比较，若 PredictJump=1，表示预测跳转，则将分支跳转地址送入 PC 寄存器；若 PredictJump=0，表示预测跳转，则将 PC+4 送入 PC 寄存器。若预测失误，PredictErr 为 1，则将 IF/ID 和 ID/EX 流水寄存器清零，清除误取的指令，并将正确的分支地址 BranchAddr 送入 PC 寄存器，同时更新 BTB 表中数据。

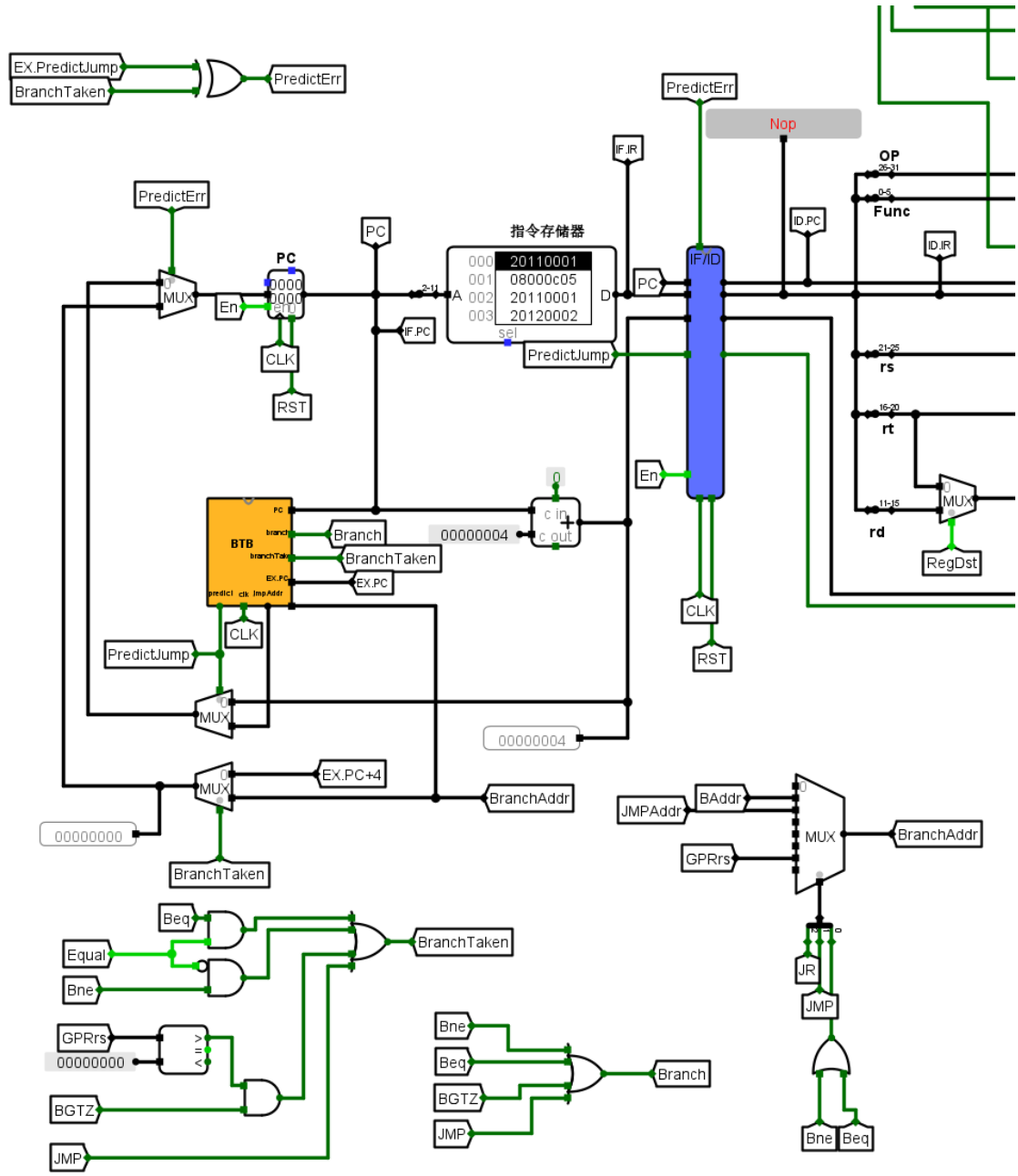

图 3.27 分支预测顶层设计

3.7 团队任务——音乐播发器的实现

一般说来，蜂鸣器演奏音乐只能是单音频率，它不包含相应幅度的谐波频率，即不考虑音色。因此蜂鸣器奏乐只需弄清楚两个概念即可，也就是“音调”和“节拍”。音调表示一个音符唱多高的频率，节拍表示一个音符唱多长的时间。十二平均律就规定了每一个音符的标准频率。十二平均律，是一种音乐定律方法，将一个纯八度平均分成十二等份，每等分称为半音，是最主要的调音法。十二平均律中各音的频率： C: 262 Hz、#C: 277 Hz、D: 294 Hz、#D: 311 Hz、E: 330 Hz、F: 349 Hz、#F: 370 Hz、G: 392 Hz、#G: 415 Hz、A: 440 Hz、#A: 466 Hz、B: 494 Hz。

根据音乐的简谱，找出对应的音符和节拍，然后换算成对应的频率，和发声时间与不发声时间。如图 3.28 所示，存储的是《两只老虎》的简谱，音乐存储器中存储的是按字节存储的数据，其高 4 位代表是代表音符的时值，低四位代表的是音符号，然后根据音符号从频率寄存器读出对应频率，输入到蜂鸣器，音符的时值每次减一，减到 0 后，将读取写一个音符，并将新的节拍号存入寄存器。

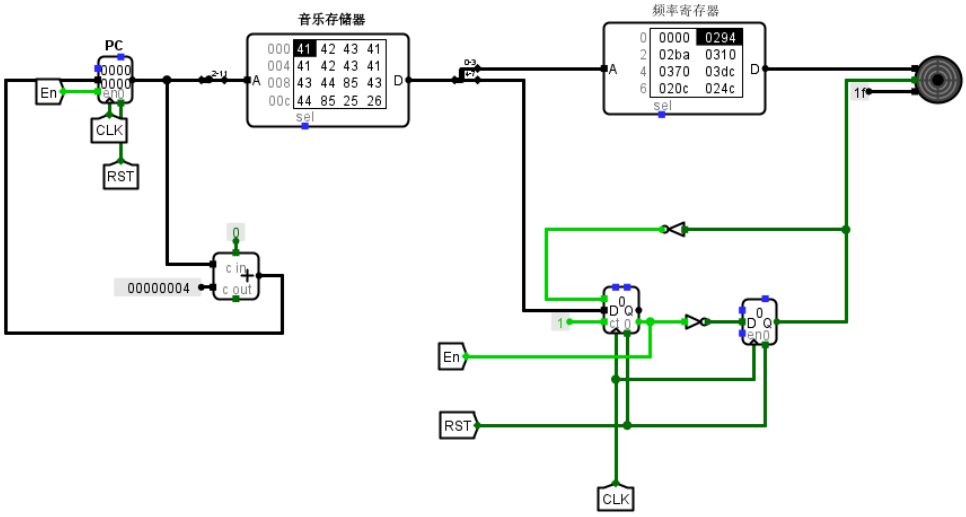

图 3.28 音乐播放器设计图

4 实验过程与调试

4.1 测试用例和功能测试

由于实验测试程序都是在显示器上动态显示测试结果不好展示，实验基本测试可以参照 educoder 上实验测试结果。这里只展示个人 CCMB 指令的测试结果。

4.1.1 测试用例 1

SLLV 移位测试结果如图 4.1 所示，显示器依次输出 0x00000876 0x00008760 0x00087600 0x00876000 0x08760000 0x87600000 0x76000000 0x60000000 0x00000000 测试结果合乎预期。

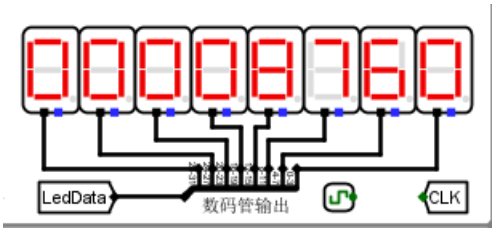

图 4.1 SLLV 移位测试

4.1.2 测试用例 2

xori 测试结果如图 4.2 所示，显示器依次输出 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 0x00008888 0x00007777 测试结果合乎预期。

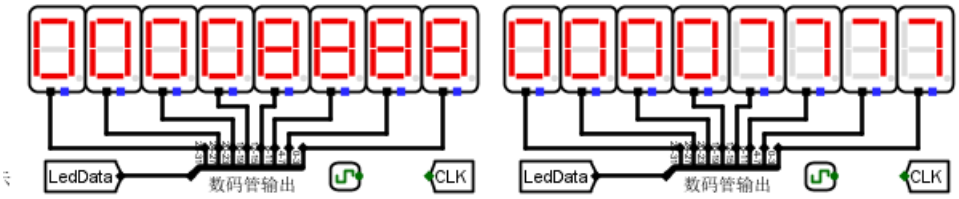

图 4.2 Xori 指令测试

4.1.3 测试用例 3

SB 测试结果如图 4.3 所示，显示器依次输出 0x00000000 0x00000001 0x00000002 0x00000003 0x00000004 0x00000005 0x00000006 0x00000007 0x00000008 0x00000009 0x0000000a 0x0000000b 0x0000000c 0x0000000d 0x0000000e 0x0000000f 0x00000010 0x00000011 0x00000012 0x00000013 0x00000014 0x00000015 0x00000016 0x00000017 0x00000018 0x00000019 0x0000001a 0x0000001b 0x0000001c 0x0000001d 0x0000001e 0x0000001f 0x03020100 0x07060504 0x0b0a0908 0x0f0e0d0c 0x13121110 0x17161514 0x1b1a1918 0x1f1e1d1c 测试结果合乎预期。

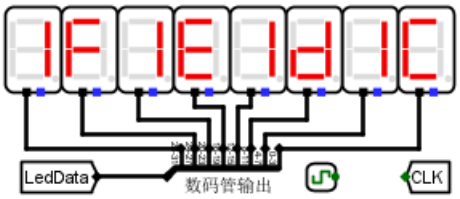

图 4.3 SB 指令测试

4.1.4 测试用例 4

bgtz 测试如图 4.4 所示，显示器依次输出 0x0000000f 0x0000000e 0x0000000d 0x0000000c 0x0000000b 0x0000000a 0x00000009 0x00000008 0x00000007 0x00000006 0x00000005 0x00000004 0x00000003 0x00000002 0x00000001 结果合乎预期。

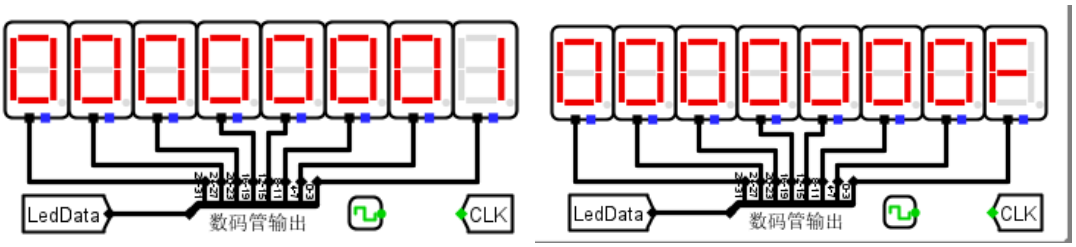

图 4.4 bgtz 指令测试

4.2 性能分析

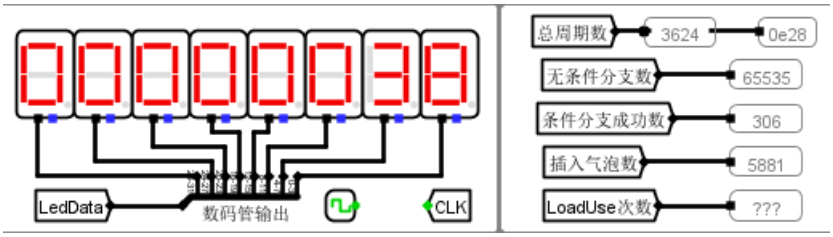

图 4.5 气泡流水线 branchmark 程序测试

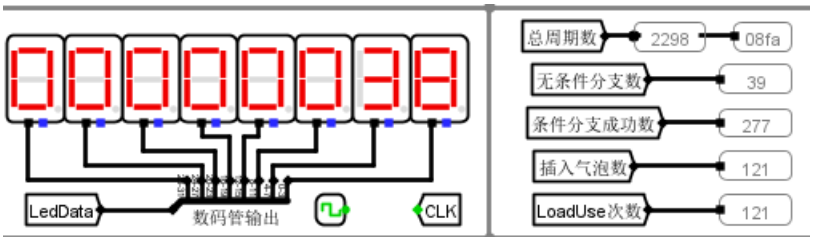

图 4.6 重定向流水线 branchmark 程序测试

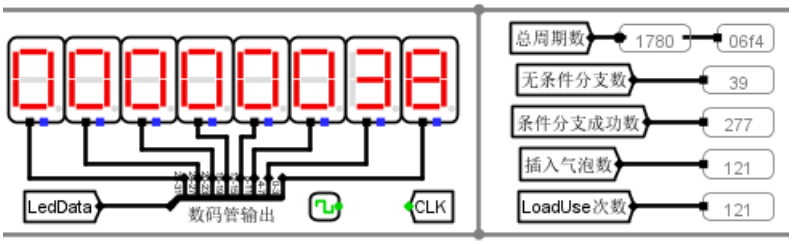

图 4.7 动态分支预测 branchmark 程序测试

如上图所示，气泡流水线测试 branchmark 程序时钟总周期数为 3624，由于程序运行过程中，存在大量插入的气泡，大大影响了流水线的性能。重定向流水线测试 branchmark 程序时钟总周期数为 2298，采用重定向机制，减少了插入的气泡，机器性能得到了极大提高。动态分支预测测试 branchmark 程序时钟总周期数为 1780，性能最好。动态分支预测为了设计方便，其分支预测初设设置都为 00，但是对于无条件分支指令，初始值如果是 00，则预测失败次数是 2 次，故如果把无条件分支指令区分开，使其初设值为 10，这样就会使得性能更加优秀。

4.3 主要故障与调试

4.3.1 气泡流水线故障

理想流水线：syscall 指令执行问题。

故障现象：执行 syscall 指令时,程序没有按照预期设计顺序执行。

原因分析：，最开始在流水线中设计 syscall 指令时如 4.8 所示，因为 syscall 指令可能会导致停机，所以设计方案是 syscall 信号运行到 MeM/WB 流水寄存器之后读出，以保证 syscall 指令执行前的指令都执行完毕，之后 sycall 信号经过多路选择器读出$a0 和$v0 寄存器的值，判断是否停机。但是这样做会导致但 v0 不等于 10 时，syscall 指令应该只是显示，而不是停机，若此时 ID 段指令又要读寄存器数据，就会出现错误，错误地将$a0 和$v0 寄存器的值送入到了 ID/EX 流水寄存器，而没有将正确地将编号为 rs 和 rt 的寄存器的值送入到 ID/EX 流水寄存器。

解决方案：当 syscall 指令在 ID 段时，将$a0 和$v0 寄存器的值依次向后面的流水寄存器传递，直到最后数据存储在 MeM/WB 流水寄存器中之后，再根据$v0 的值判断是否需要停机，解决后的设计图如图 4.9 所示。

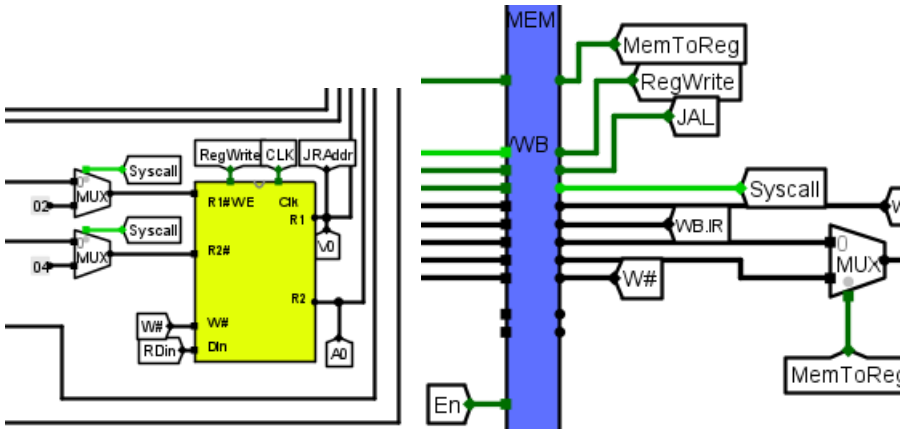

图 4.8 syscall 指令原始设计图

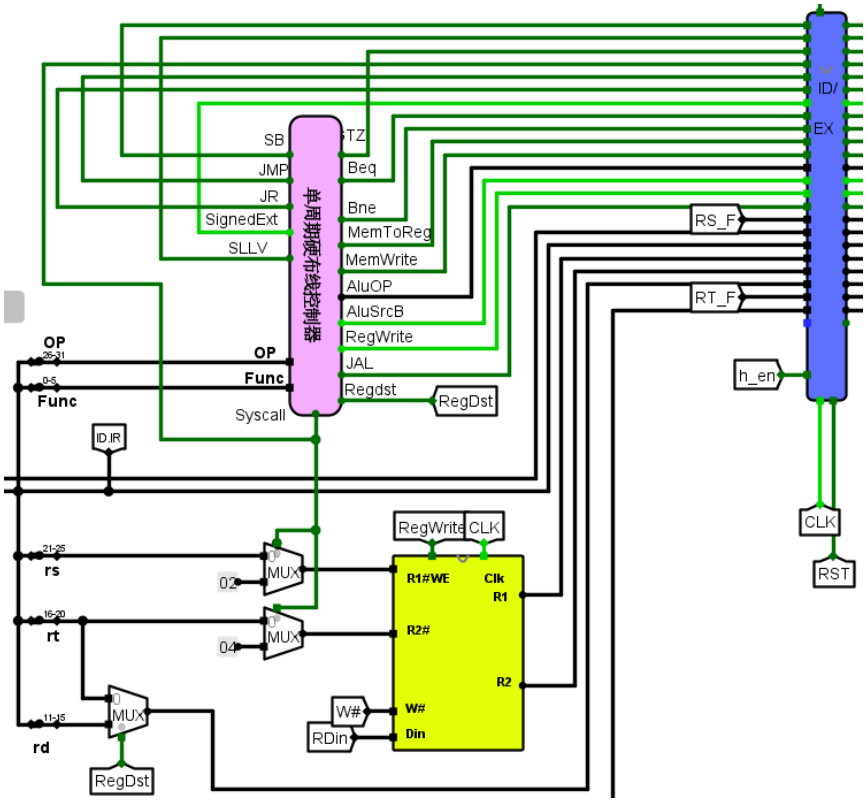

图 4.9 syscall 指令正确设计图

4.3.2 单级中断故障

故障现象：中断返回指令 eret 指令执行时，不能正确的清除对应的中断信号。

原因分析：初始单级中断设计图如图 4.10，由于没有采用单独的寄存器来记录当前正在执行的中断源，如果执行一个中断服务程序期间又发生了其他中断，经过优先编码器后，当 eret 指令执行时会清除优先级最高的那个中断等待信号，这显然是错误的。

解决方案：添加一个寄存器，当中断发生时，存储当前正在处理的中断号，但中断返回时，用该寄存器存储的值正确清除对应的中断等待信号。

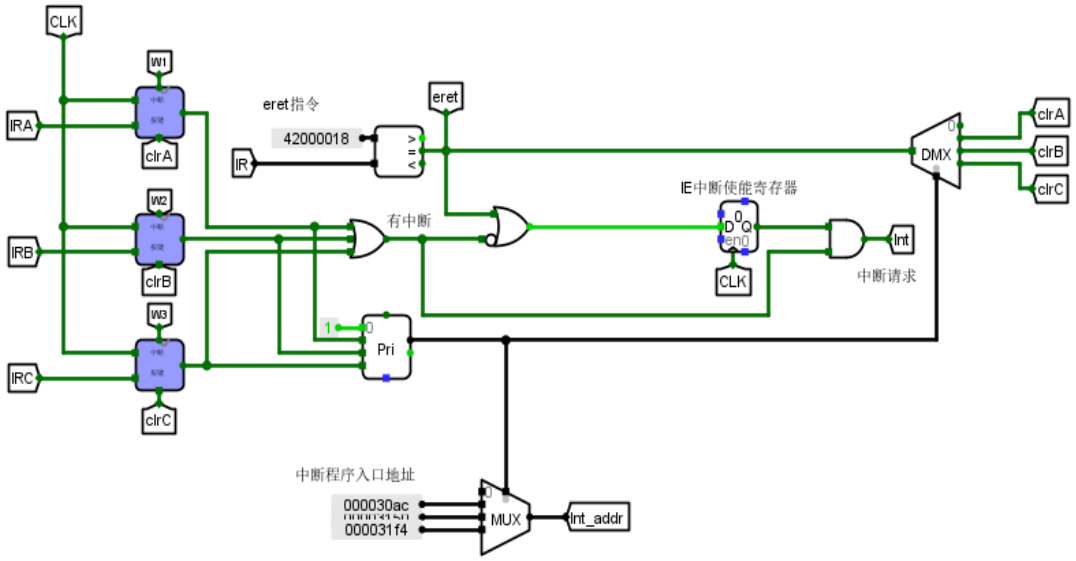

图 4. 10 错误的单级中断设计

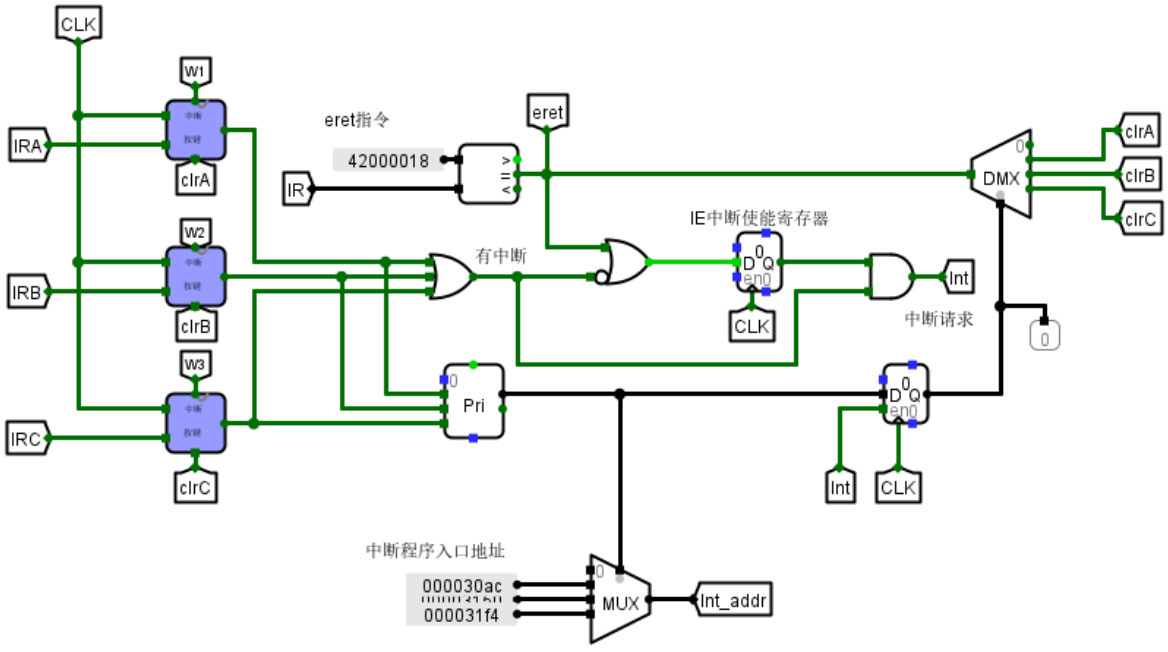

图 4. 11 正确的单级中断设计

4.4 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 相关理论知识，阅读课设任务书，阅读 MIPS 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。完成部分 Logism 单周期 CPU 故障报告。
第三天	完成 Logism 单周期 CPU 的故障报告，并且通过了 Logism 单周期 CPU 的检查。。
第四天	学习流水线原理，完成了理想流水线，并通过 educoder 测试。
第五天	学习了流水线中控制冲突，结构冲突和数据冲突的解决办法，完了了数据相关处理逻辑的设计，通过了气泡流水线的检查。
第六天	完了了数据相关处理逻辑的设计，对 load-Use 问题有了深入了解，完成了重定向流水线的设计。
第七天	完成了单周期+单级中断电路的设计。
第八天	完成了单周期+多级嵌套中断电路的设计。
第九天	完成了流水线+单级中断的电路设计。
第十天	完成了动态分支预测流水线的设计，熟练掌握了动态分支预测的原理。
第十一天	完成小组任务分工，利用 logsim 中的蜂鸣器设计一个音乐播放器。
第十二天	完成音乐播放器的设计并撰写报告。

5 设计总结与心得

5.1 课设总结

通过本次课设。我对 cpu 的体系结构有了更深入的掌握，这次实验，也让我对 logisim 的掌握更加熟练。现在总结一下本次课设完成的任务：

1） 实现了单周期 MIPS CPU

2） 实现了 CPU 单级中断和多级嵌套中断机制

3） 实现了经典的 MIPS 五段流水线，

4） 实现了气泡流水线和重定向流水线来解决数据冲突

5） 实现了动态分支预测机制

5.2 课设心得

本次课程设计难度较大，但也收获颇多，回顾整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。第个任务是使用 Logism 设计单周期 CPU，该任务书上有相应的整体设计图，所以整个过程还算比较迅速，主要需要注意的地方就是 syscall 指令的实现，还有就是个人特殊指令 ccmb 指令，需要在了解 cpu 的架构上，根据需要添加所需要的部件 。理想流水线和气泡流水线以及重定向流水线教材上都有顶层设计图，只需要了解他们解决冲突的原理，设计冲突检测的部件然后按照课本上的顶层设计连线就可以了。本次课程设计，难度最大的还是多级嵌套中断和动态分支预测机制，这些教材上都没有设计好的电路，都需要自己动手实现，多级嵌套我用的是用硬件实现堆栈，记录中断源，这对电路的设计和信号控制是一个巨大的挑战，最后动态分支预测，本质上是实现一个 cache，这与上学期的实验相似，主要难度在于写策略和替换规则 LRU 的实现。

本次课程设计与理论化紧密结合，非常锻炼学生学习能力以及对课内知识掌握情况的考察。本次课设我获益匪浅，在这里我衷心地感谢老师对我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

参考文献

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

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

[3] 谭志虎，秦磊华，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021 年.

[4] 谭志虎，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.

[7] 音乐数据的编码与解码 (rt-thread.org)

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：刘红阳