华中科技大学

2024

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：	CS2201
学　　号：	U202215357
姓　　名：	王文涛
电　　话：	13607252896
邮　　件：	2380169004@qq.com

目 录

1 课程设计概述

1.1 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以 CPU 设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

1.2 设计任务

本课程设计的总体目标是利用 FPGA 以及相关外围器件，设计五段流水 CPU，要求所设计的流水 CPU 系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过 LED、数码管等适时的进行显示，方便监控和调试。尽可能利用 EDA 软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

（4） 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10）支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（12）能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。

（13）能运行教师提供的标准测试程序，并自动统计执行周期数

（14）能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

#	指令助记符	简单功能描述	备注
1	ADD	加法	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
2	ADDI	立即数加	
3	ADDIU	无符号立即数加	
4	ADDU	无符号数加	
5	AND	与	
6	ANDI	立即数与	
7	SLL	逻辑左移	
8	SRA	算数右移	
9	SRL	逻辑右移	
10	SUb	减	
11	OR	或	
12	ORI	立即数或	
13	NOR	或非	

#	指令助记符	简单功能描述	备注
14	LW	加载字	
15	SW	存字	
16	BEQ	相等跳转	
17	BNE	不相等跳转	
18	SLT	小于置数	
19	STI	小于立即数置数	
20	SLTU	小于无符号数置数	
21	J	无条件转移	
22	JAL	转移并链接	
23	JR	转移到指定寄存器	If $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$a0 值
25	MFC0	访问 CP0	中断相关，可简化，选做
26	MTC0	访问 CP0	中断相关，可简化，选做
27	ERET	中断返回	异常返回，选做
28	XOR	异或	
29	AUIPC	加载 PC 高位	
30	LH	加载高字节	
31	BLT	小于跳转	

2 总体方案设计

2.1 单周期 CPU 设计

我们采用的方案是用硬布线控制，执行速度快，将指令存放内存与数据存放内存分开。

总体结构图如图 2.1 所示。

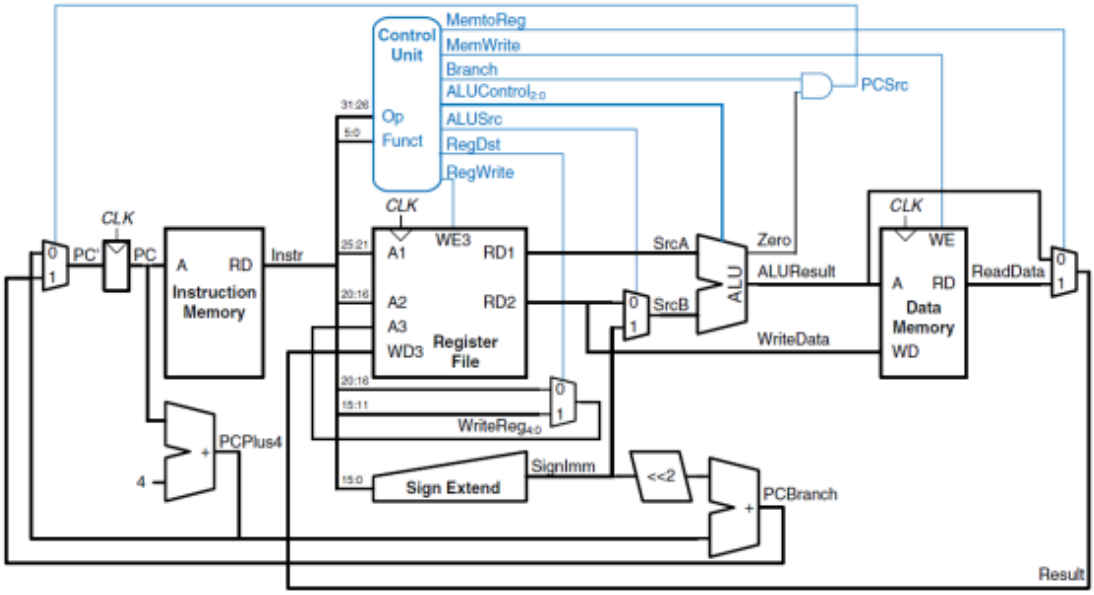

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

使用一个寄存器保存 PC 的值，寄存器设有使能端与复位端，执行 halt 时关闭使能端。根据 B 型指令与 J 型指令决定是否跳转，如果跳转，PC 值设为计算出的跳转地址；如果不跳转，将 PC 值加 4。

2. 指令存储器 IM

IM 中储存指令的机器码，根据程序计数器输出的指令地址在 ROM 中读取相应的指令。由于 ROM 按 4 字节编址，需要先将 PC 右移两位。

3. 运算器

ALU 单元负责各种算术和逻辑运算。包含三个输入引脚和五个输出引脚，各引脚的功能如表 2.1 所示。

根据 ALU_OP 的值执行相应的操作。

表 2.1 算术逻辑运算单元引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

寄存器堆用于存储 CPU 的寄存器，共 32 个寄存器。

实验中的 RF 组件包括使能端 WE，时钟端 Clk。时钟端采用下降沿触发。

输入引脚 R1#、R2#表示读寄存器的编号，W#表示写寄存器的编号。寄存器编号均用 5 位表示。Din 表示写入的数据。

输出引脚 R1，R2 表示读取的寄存器的内容。

2.1.2 数据通路的设计

基础实验需要实现取指令数据通路，R 型指令数据通路，I 型运算指令数据通路，访存指令数据通路，分支指令数据通路，Ecall 指令数据通路。由于在 CCAB 中有 U 型指令，还需要实现 U 型指令数据通路。

1. 取指令数据通路：使用一个二路选择器，根据是否进行跳转，选择 PC+4 还

是跳转地址。

2. R 型指令数据通路:，R 指令所有的操作数均是寄存器，执行过程中涉及的功能部件包括寄存器文件和 ALU。指令执行过程中的 ALU 的两个源操作数均来自于寄存器文件输出，根据指令中 rs1 和 rs2 的值，取出寄存器中的值，在 ALU 中进行运算，将 AluOp 设置为不同的值，就可以进行不同的运算。运算结果写入目的寄存器 rd 中。
3. I 型运算指令数据通路：根据立即数类型（B 型，J 型 S 型，U 型，I 型），从指令中取出立即数，作为 ALU 的第二个操作数。其他的部分与 R 型指令数据通路相同。
4. 访存指令数据通路：访存指令属于 I 型指令，访存地址等于变址寄存器 rs1 的值加上 12 位立即数， 地址运算通过 ALU 完成，所以需要将 rs1 的值送入 ALU，同时要将 12 位立即数进行 32 位符号扩展后送入 ALU，二者进行加法运算得到最终的访存地址。如果是加载指令，则数据存储器的结果将会送回到 寄存器文件的 WD 端，写入寄存器编号为 rd。如果是存储指令，则 rs2 寄存器的值会通过 R2 端口输出到数据存储器的写入端口 WD。
5. 分支指令数据通路：分支指令分为条件分支指令（B 型）和无条件分支指令（J 型）。条件分支指令根据 ALU 的计算结果判断是否跳转。无条件分支指令直接跳转。B 型指令和 jal 指令将立即数左移一位作为 offset 与 PC 相加得到跳转地址。Jalr 将 ALU 的计算结果作为偏移地址。
6. Ecall 指令数据通路：当 a7 寄存器为 34 时，显示 a0 寄存器的数据；否则，实现停机功能。因此当执行 ecall 指令时，将 rs1 输入端设置为 a7 寄存器号 17，rs2 输入端设置为 a0 寄存器号 10，获取对应寄存器数据。R1 为 34 时，将 a0 中的数据放入 LED 寄存器中，否则执行停机指令 halt。
7. U 型指令数据通路：执行 AUIPC 时，将 PC 与处理后的立即数相加，写回到目的寄存器号 rd 中。

2.1.3 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 2.2。

表 2.2 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_OP	0~12	ALU 单元操作号
MemtoReg	0/1	是否将内存中的数据写入寄存器
MemWrite	0/1	是否向内存中写入数据
ALU_Src	0/1	是否使用立即数而不是 B 寄存器
RegWrite	0/1	是否向寄存器中写入数据
ecall	0/1	是否执行系统调用
S_Type	0/1	是否执行 S 型指令
BEQ	0/1	是否执行 BEQ
BNE	0/1	是否执行 BNE
Jal	0/1	是否执行 Jal
Jalr	0/1	是否执行 Jalr

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图 2.2 所示。

指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	Sw	BEQ	BNE	Jal	Jalr	Auipc	Lh	Blt	CSRRSI	CSRRCI
add	0	0	c	5				1											
sub	32	0	c	6				1											
and	0	7	c	7				1											
or	0	6	c	8				1											
slt	0	2	c	11				1											
sltu	0	3	c	12				1											
addi		0	4	5			1	1											
andi		7	4	7			1	1											
ori		6	4	8			1	1											
xori		4	4	9			1	1											
slti		2	4	11			1	1											
slli		1	4	0			1	1											
srli		5	4	2			1	1											
srai	32	5	4	1			1	1											
lw		2	0		1		1	1											
sw		2	8			1	1			1									
ecall		0	1c						1										
beq		0	18	6							1								
bne		1	18	6								1							
jal			1b				1	1					1						
jalr			19	6				1						1					
CSRRSI		6	1c															1	
CSRRCI		7	1c																1
URET		0	1c																
XOR	0	4	c	9				1											
AUIPC			5	5			1	1							1				
LH		1	0		1		1	1								1			
BLT		4	18	11													1		

图 2.2 主控制器控制信号框架

2.2 中断机制设计

2.2.1 总体设计

为已实现的 CPU 增加中断处理机制，支持 3 个 Logisim 按钮触发的中断源，分别对应编号为 1、2、3 的三个按钮，中断优先级 1<2<3，高优先级中断应该正确中断低优先级中断服务子程序。这里只讲解更复杂的多级终端处理机制，实验采用硬件实现。

首先增加中断按键信号采样电路，中断使能寄存器 IE、异常程序计数器 EPC

2.2.2 硬件设计

首先设计中断按键信号采样电路，由于通过按键触发的中断源信号只会持续一瞬间，需要将通过断信号保存到 D 触发器中，当中断处理程序返回 uret 时再将信号清空。为了实现高优先级中断正确中断低优先级中断服务子程序，需要设计中断屏蔽，将比执行的中断优先级更低的中断信号频闭掉。按键信号采样电路输出中断处理信号（表示进行中断）和当前的中断号。

实现 EPC 寄存器，为了保存中断处理程序被中断的 PC 值，需要三个寄存器保存最多可能出现的 3 个中断地址。当进行中断向寄存器中存入新的返回地址，中断返回时将下一个 EPC 寄存器中的地址返回到当前寄存器。

与 EPC 类似，需要使用三个寄存器储存中断号，以便返回被中断的中断处理程序。

实现 IE 寄存器，中断响应时需要关中断，将 IE 寄存器置零；中断返回时需要开中断，将 IE 寄存器置 1。

最后根据中断号返回中断处理程序的地址。

2.2.3 软件设计

需要增加指令 uret，CSRRSI 与 CSRRCI。当中断处理程序执行完后返回 uret。CSRRSI 指令和 CSRRCI 指令用于给 IE 寄存器置 1 或者置 0。

2.3 流水 CPU 设计

2.3.1 总体设计

采用五段流水 CPU 设计，将指令过程分成 5 个阶段 IF，ID，EX，MEM，WB。不同阶段之间设置缓冲接口部件，构建各阶段之间的接口部件，流水线向后续段传递数据信息，控制信息，向前段传递反馈信息。

2.3.2 流水接口部件设计

五段流水 CPU 需要四个流水寄存器 IF/ID、ID/EX、EX/MEM、MEM/WB。流水寄存器用来储存上一段产生的数据以及需要往下传递的控制信号。寄存器采用上升沿触发，并且设有使能端和清空端，为了实现停机指令 halt 和重置指令 rst。

2.3.3 理想流水线设计

理想流水线设计不考虑分支指令，直接将单周期 CPU 分成 5 段。再 EX 段处理得到分支地址，以及完成系统调用。将产生的 halt 指令通过流水寄存器传递到最后的 WB 段再执行，为了保证停机时指令已执行。

2.4 气泡式流水线设计

气泡式流水线需要处理数据冲突，增加气泡处理机制。如果 ID 段要执行的指令需要用到 EX 段和 MEM 段才写回的数据，产生阻塞信号 stall，将 PC 寄存器暂停，并清空 ID/EX 寄存器数据，插入气泡。

还需要处理分支冲突，当 EX 段检测到需要跳转时，将得到的分支地址传到 IF 段的 PC 中，并将 IF/ID 和 ID/EX 寄存器清空。

2.5 重定向流水线设计

为了减少气泡，提高 CPU 效率，实现重定向处理机制。

在 ID 段检测到数据冲突之后，将需要使用的后续段中还未写回的数据直接传回 EX 段中代替原本的 R1 和 R2。

对于 Load-Use 相关的数据冲突，由于产生冲突的是访存指令，需要等待数据存储器读取数据，导致 EX 段的延迟增加，使流水线的频率大幅降低。因此检测到

Load-Use 冲突之后，产生阻塞信号 stall，将 IF 段和 ID 段寄存器暂停，暂停 PC 插入一个气泡。

2.6 动态分支预测机制

为了减少分支导致的气泡，实现动态分支预测。

使用一个 BTB 表来记录分支指令的分支跳转历史，并以此来“预测”下一个 PC 值。

当 PredictJump(预测跳转)与 BranchTaken(实际跳转)不一致时，生成 PredictErr 信号，表示动态分支预测错误，此时需要将 IF/ID 和 ID/EX 两个流水寄存器清空，通过插入气泡的方式重新执行分支指令。

BTB 表采用全相联 cache 来实现。cache 行包括有效位 Valid、分支指令地址、分支目标地址、分支预测历史位、淘汰计数位。分支预测历史位是一个按有限状态机变化的数据，用于统计历史跳转情况并以此来决定是否跳转;淘汰计数位使用 LRU 算法在 cache 槽满后对 cache 行进行淘汰替换。

3 详细设计与实现

指令周期流程图要在此部分出现、微程序流程图、微指令代码表、实验接线图等均需要在适当的位置和模块中表达出来。本章具体实现细节尽量多用图表方式展示，但要做到图文并茂，不能全文都是图。

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt 控制信号为 1，使寄存器停机。如图 3.1 所示。

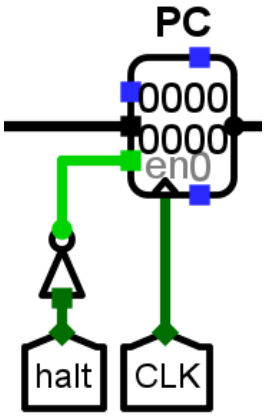

图 3.1 程序计数器（PC）

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) 数据寄存器（DM）

使用一个随机存储器 RAM 实现数据存储器（DM）。设置该随机存储器的地址位宽为 10 位，数据位宽为 32 位。时钟端采用上升沿触发，使能端由 MemWrite 提供。如图 3.3 所示。

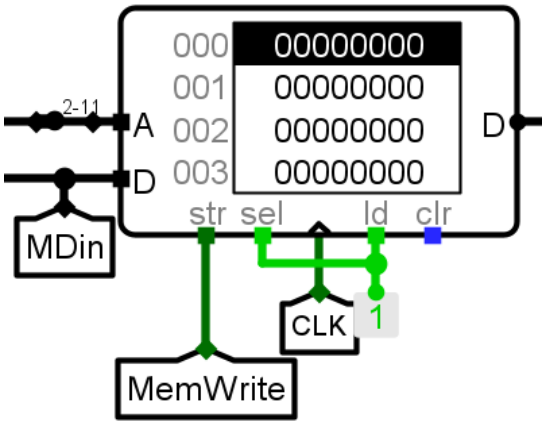

图 3.3 数据存储器（DM）

4) 寄存器堆（RF）

时钟信号采用下降沿触发，R1#和 R2#是要读取的寄存器编号，W#是要写入的寄存器编号，Din 是写入寄存器的值，WE 是写使能信号，R1 和 R2 是读取的寄存器数据。如图 3.4 所示。

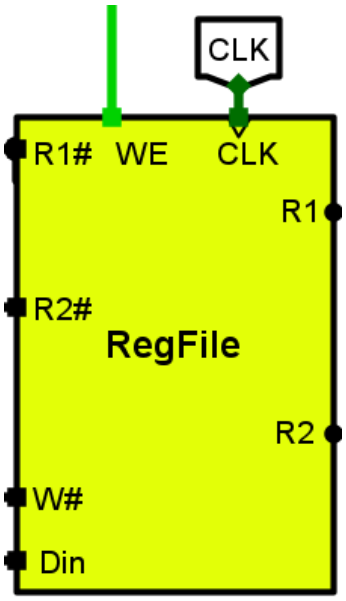

图 3.4 寄存器堆（RF）

3.1.2 数据通路的实现

程序计数器输出 PC 从指令存储器中读取指令，指令传到主控制器，输出控制信号。

对于 R 型指令，根据主控制器输出的 rs1 和 rs2 在寄存器堆中得到 R1 和 R2，经过 ALU 得到计算结果。写回的值 Rdin 即为计算结果。

对于 I 型指令，将指令输入立即数生成模块，得到立即数，输入 ALU 作为操作数 B，其他部分与 R 型指令相同。

对于分支指令，如果是 B 型或者 jal，将立即数左移一位作为偏移 offset，然后与 PC 相加作为跳转地址。如果是 jalr 指令，将 ALU 的结果作为跳转地址。

对于访存指令，将 ALU 的结果的 2 到 11 位作为地址从数据存储器中得到数据。如果要写回就作为 Rdin 写回寄存器。

最终的数据通路如图 3.5 所示。

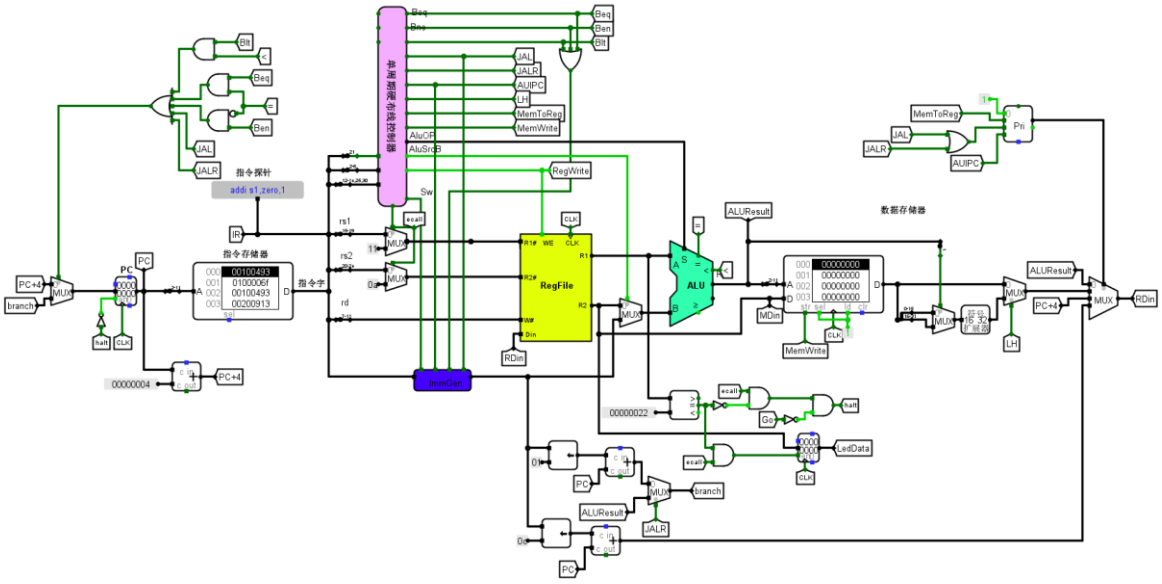

图 3.5 总体数据通路

3.1.3 控制器的实现

根据控制器的设计思路填写单周期硬布线控制器表达式如图 2.2。根据 excel 自动生成的表达式在 logsim 中生成运算器控制器和控制信号生成模块如图 3.6 所示。最终得到的封装电路如图 3.5 中所示。根据输入的指令中的 op_code 和 funct 生成相应的控制信号，根据 IR21 区分 ecall 和 uret。

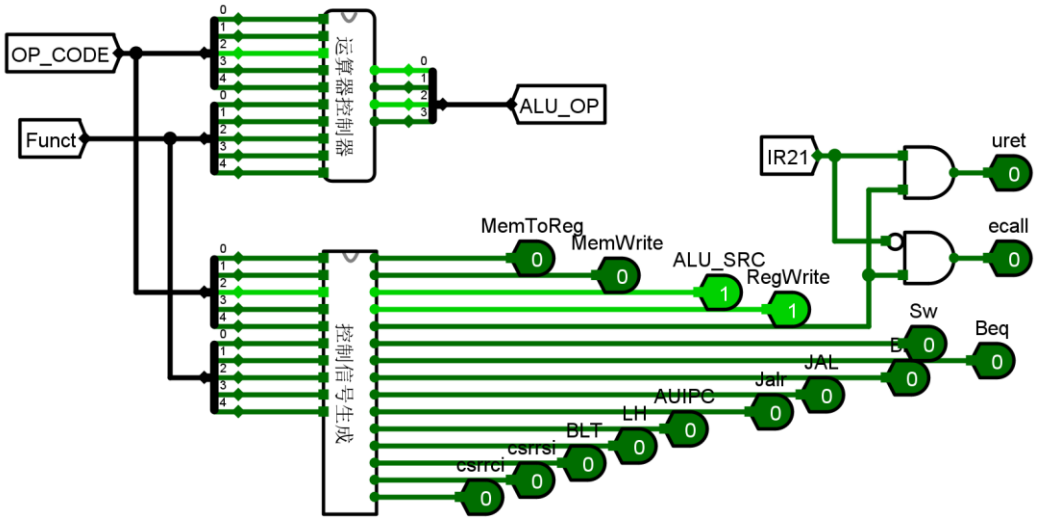

图 3.6 主控制器原理图

3.2 中断机制实现

3.2.1 单级中断机制实现

首先实现中断信号采样电路如图 3.7 所示。接收到中断请求信号时，持续输出当前请求的中断号和终端信号。输出信号灯信号 W1，W2，W3。当收到 URET 信号时，结束当前中断，将相应信号灯信号置 0.

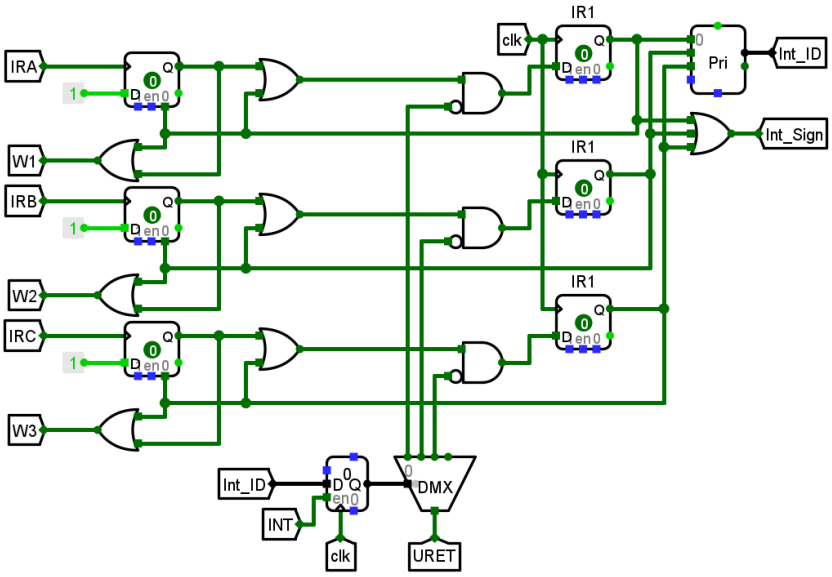

图 3.7 单级中断信号采样电路

实现 EPC 寄存器，当存在中断请求信号 INT 时，在时钟端上升沿将跳转前地址存入 EPC 寄存器。实现 IE 寄存器，中断使能信号 IE 表示能否进行中断，当存在中断信号 int_Sign 时将 IE 置为 1，表示无法处理新的中断；返回 URET 时，将 IE 置为 0 表示可以进行中断，在时钟端上升沿将跳转前地址存入 EPC 寄存器。当 IE 为 1 且又中断信号 int_Sign 时，输出中断请求信号 INT。如图 3.8 所示。

最后根据中断号生成中断入口地址，如图 3.8 所示。

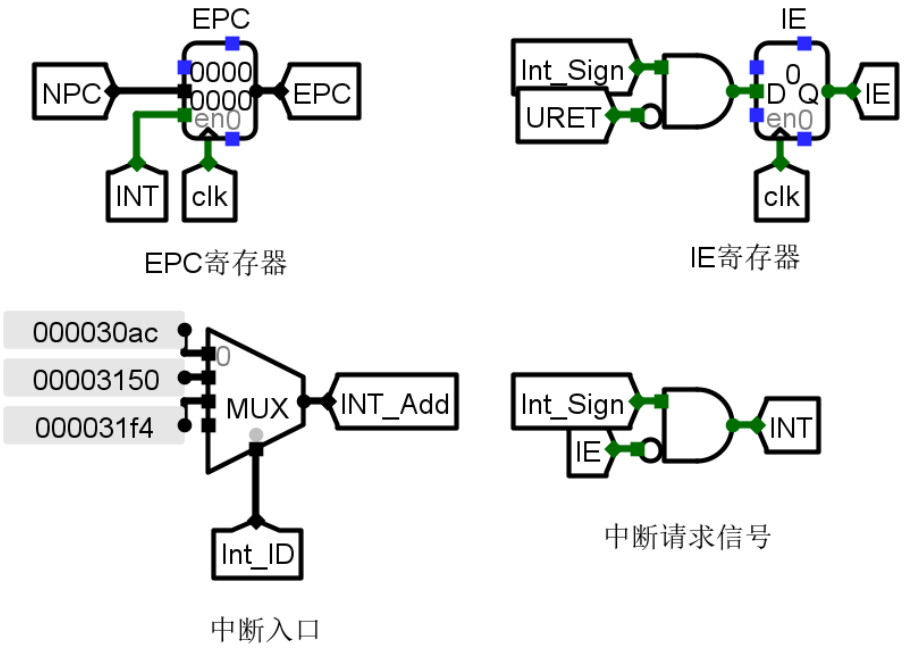

图 3.8 单级中断处理逻辑

3.2.2 多级中断机制实现

首先实现中断信号采样电路如图 3.9 所示。增加了优先级屏蔽模块，当前的中断号 int_ID 为高优先级时会屏蔽低优先级的中断请求，保证要执行的中断号 next_int_ID 不变。

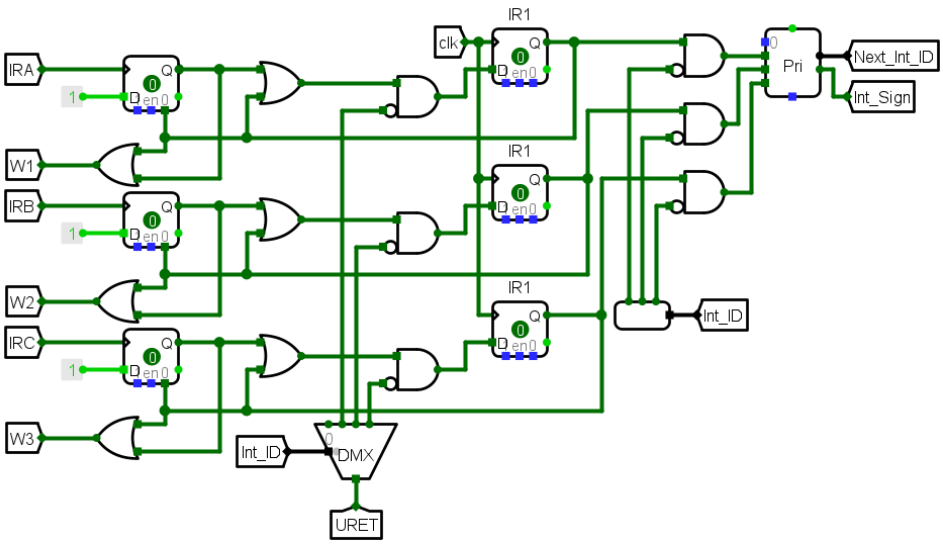

图 3.9 多级中断信号采样电路

使用硬件实现一个保存 EPC 的栈，栈顶是当前保存的 EPC，当有 INT 信号时，将新的 PC 值压入栈，EPC 变成新的 PC 值；当有 URET 信号时，当前的 EPC 弹出，变成上一个寄存器中的值。电路实现如图 3.10 所示。

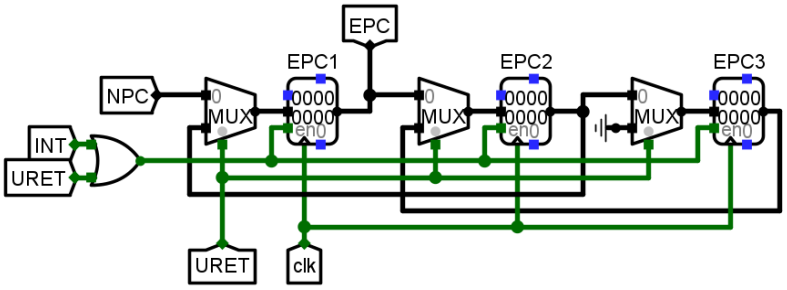

图 3.10 EPC 寄存器栈

与 EPC 类似，中断号也需要保存在栈中，电路实现如图 3.11 所示。

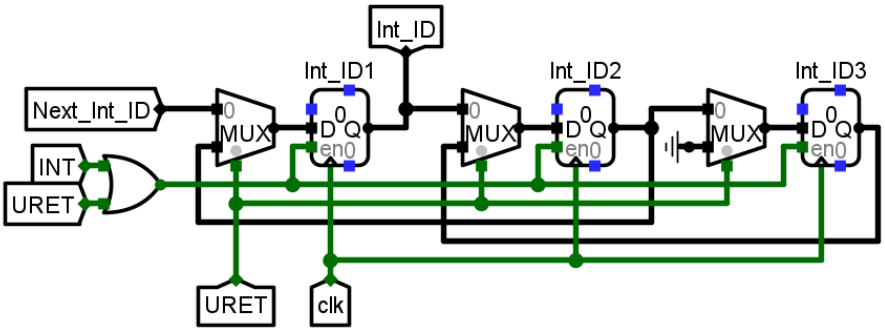

图 3.11 中断号寄存器栈

在多级中断中需要使用 csrrci 和 csrrsi 指令控制 IE 寄存器的开关，执行 csrrci 时，将 IE 置为 1，执行 csrrsi 时，将 IE 置为 0。

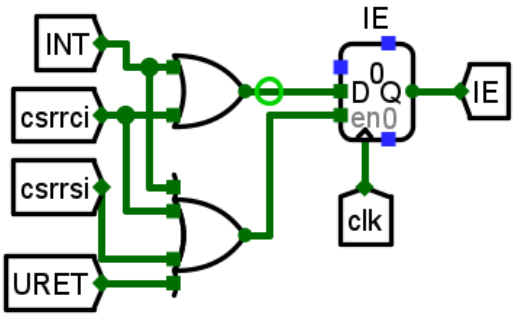

图 3.12 中断使能寄存器

其他部分与单级中断处理机制相同。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

寄存器采用上升沿触发，并且设有使能端和清空端。每个流水线寄存器储存不同的数据。以图 3.13 EX/MEM 寄存器为例，通过一个二路选择器，当 rst 为 1 时，将

寄存器中的值全部置为 0。

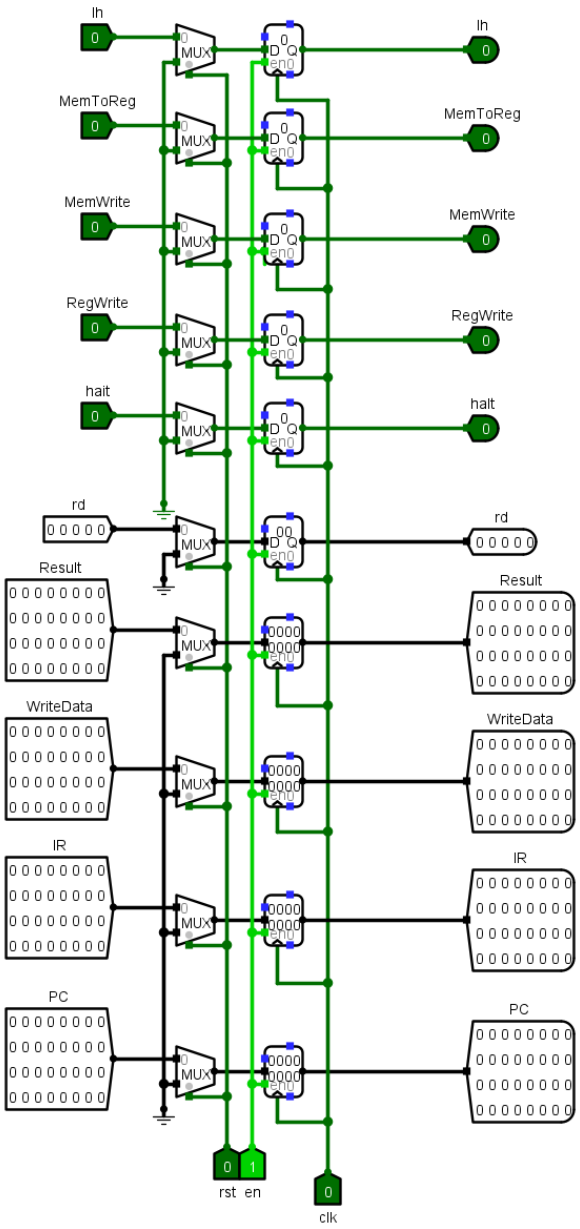

图 3.13 EX/MEM 寄存器

3.3.2 理想流水线实现

将单周期 CPU 拆分成了 5 段，之间通过流水寄存器相连。在取指段，有程序计数器（PC），将取出的指令通过 IF-ID 接口部件传送至译码段。在译码段，指令解析器将指令字段解析，并将其送至操作控制器和立即数生成器。操作控制器输出的控制信号中，部分信号在本段发挥作用，余下的信号通过集线器整合后传送至下一段。寄存器输出值和立即数扩展器的输出也 在该段传递至后续阶段。在执行段，使用分线

器提取本段所需的控制信号，其余信号通过集 线器打包后继续传递。ALU 的运算结果、寄存器输出以及 PC+4 的值向后传递。在访存段，寄存器的第二个输出作为数据存储器（DM）的输入，ALU 的运算结果作为访存地址。控制信号中，访存模式和写使能信号通过分线 器提取，其余信号继续向后传递。访存输出也继续传递至后续阶段。在写回段，ALU 输出、寄存器输出以及 PC+4 的值均传送回寄存器的输入端。选择信号从控制信号中提取。此外，停机信号在该段生成并生效。

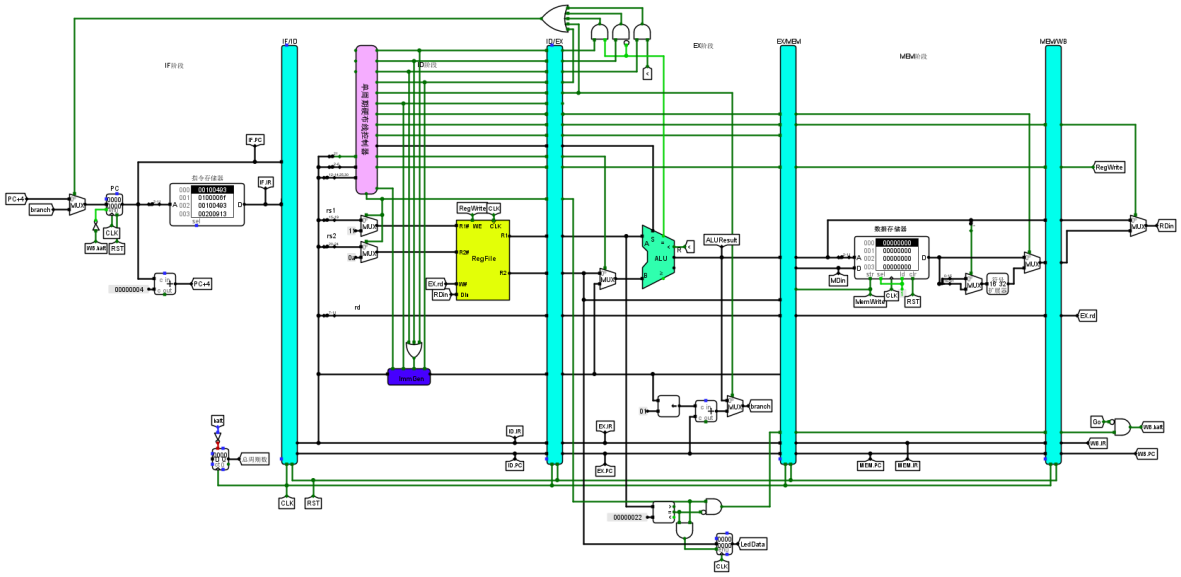

图 3.14 理想流水线 CPU

3.4 气泡式流水线实现

气泡处理机制的实现如图 3.15 所示。根据 ID 段的 IR 得到指令是否使用了 R1 和 R2，然后根据 EX 段和 MEM 段是否要写回寄存器来决定是否输出阻塞信号 stall。如果输出 stall，则将 PC 寄存器停机，生成一个气泡，并且将 ID/EX 寄存器清空。如图 3.16 所示。

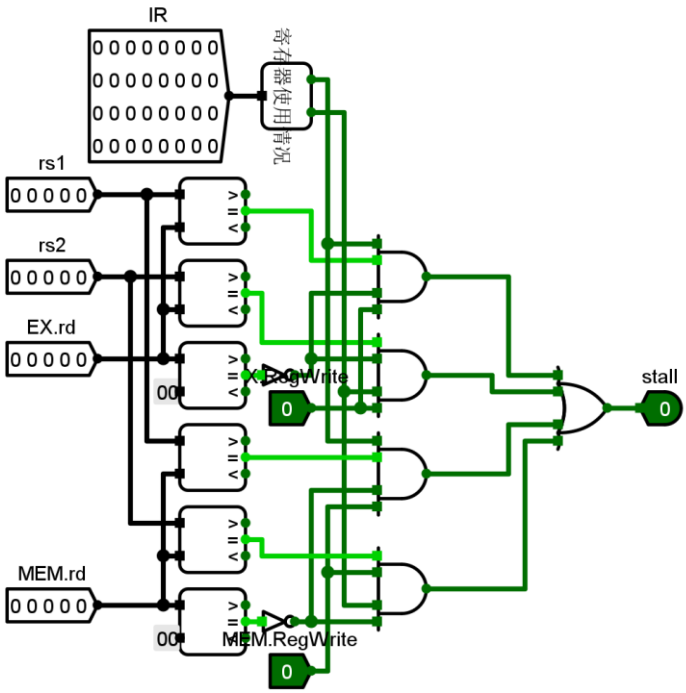

图 3.15 气泡处理

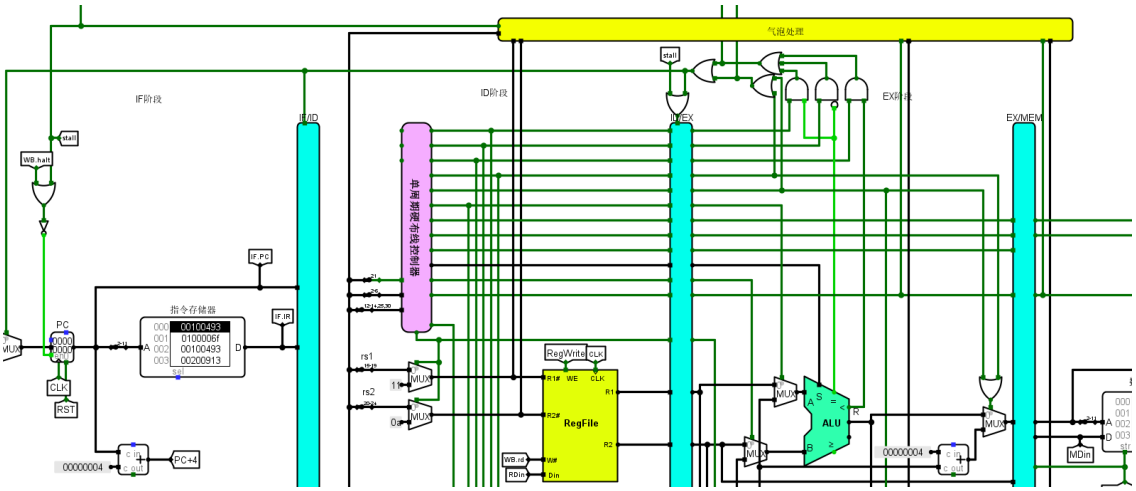

图 3.16 气泡流水线

3.5 重定向流水线实现

重定向处理机制的实现如图 3.17 所示。更改气泡处理机制，使输出的 rs1_f 和 rs2_f 的能表示 ID 段的寄存器与后面哪一段发生冲突。0 表示无冲突，1 表示与 MEM 段冲突，此时需要将 WB 段的写回数据重定向到输入端，2 表示与 EX 段冲突，此时需要将 MEM 段的写回数据重定向到输入端。重定向处理如图 3.18 所示。

涉及到访存，即 MemToReg 为 1 时需要输出 LoadUse 信号，避免最长数据通路变为 EX+MEM，从而影响性能。塞入一个气泡，并将 IF/ID 寄存器和 ID/EX 寄存器

清空。

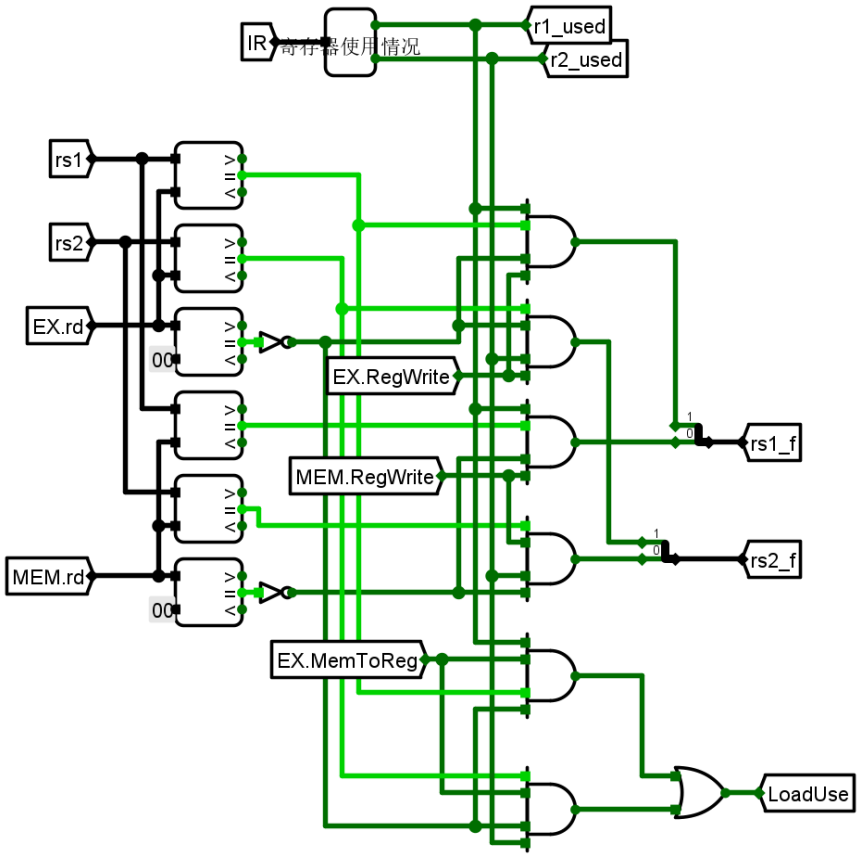

图 3.17 重定向处理

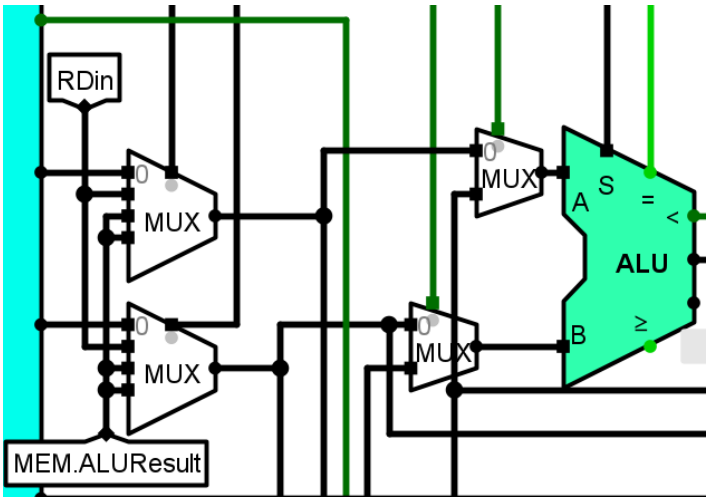

图 3.18 重定向数据通路

3.6 动态分支预测机制实现

动态分支预测使用分支目标缓冲器（BTB）记录分支指令的地址及其对应的目标

地址，避免每次都计算目标地址，从而提升分支跳转性能。BTB 的 cache 部分如图 3.19 所示。使用四个寄存器分别表示 cache 行是否可用，分支指令地址，置换标记（用于 URL 淘汰算法），分支目标地址。

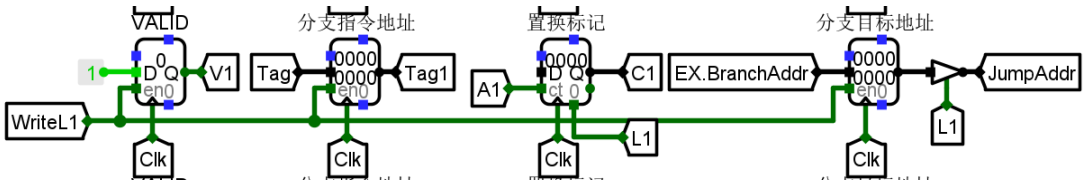

图 3.19 cache 槽

在 IF 段将 PC 与 cache 中储存的分支指令进行比较，如果命中，将置换标记清空，输出命中 Hit。然后在 EX 段如果跳转信号 branch_taken 为 1，则将指令与 cache 中储存的分支指令进行比较，如果没有找到，就输出 Miss，需要将这条分支指令写入 cache 中。

每个 cache 行还需要有一个分支预测历史寄存器，00 表示预测不跳转时未跳转，01 表示表示预测不跳转时跳转，10 表示预测跳转时跳转，11 表示预测跳转时未跳转。在 EX 段指令命中时，判断跳转结果是否与预测一致，更新分支预测历史寄存器。

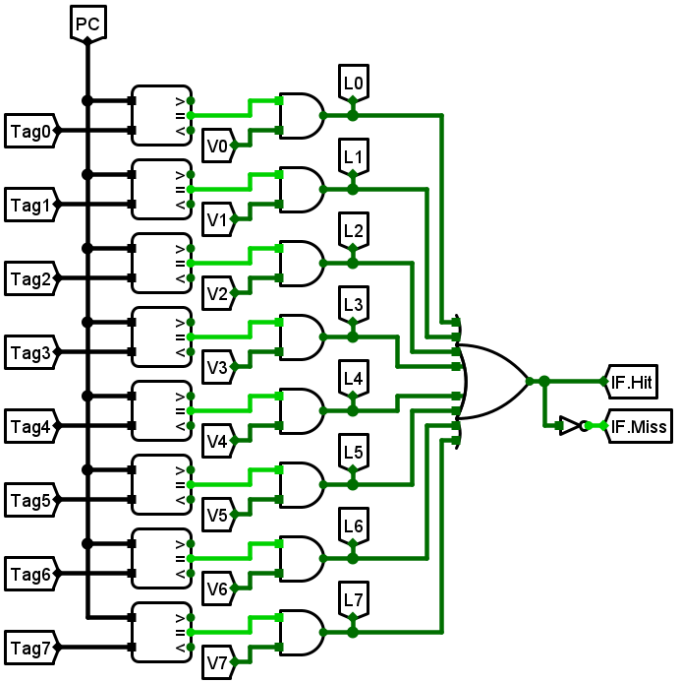

图 3.20 IF 段预测

在 CPU 中的数据通路如图 3.21 所示，预测正确时，不对分支进行调整，流水线继续执行。 预测失败时，需要修改分支，将正确的调整地址送入 IF 段的 PC 寄存

器，并清空 IF/ID、ID/EX 流水寄存器，以确保正确的指令流执行。

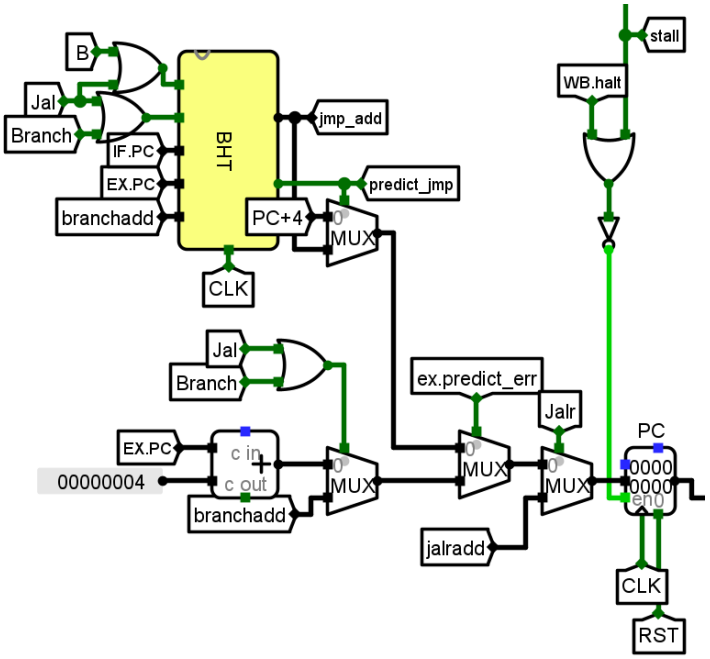

图 3.21 动态分支预测数据通路

4 实验过程与调试

4.1 测试用例和功能测试

使用测试用例 risc-v-benchmark_ccab 测试单周期 CPU，气泡流水线、重定向流水线、动态分支预测流水线。使用测试用例 risc-v 单级中断测试程序测试单级中断，使用测试用例 risc-v 多级中断测试(EPC 硬件堆栈保护) 测试多级中断。

4.1.1 测试用例 1

使用测试用例 risc-v-benchmark_ccab 依次对进行测试单周期 CPU，气泡流水线、重定向流水线、动态分支预测流水线进行测试，结果如图 4.1，图 4.2，图 4.3，图 4.4 所示。LED 输出的数据与总周期数，无条件分支数，条件分支成功数，插入气泡数，LoadUse 次数均正常。

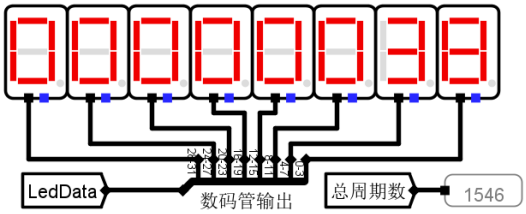

图 4.1 单周期 CPU 测试

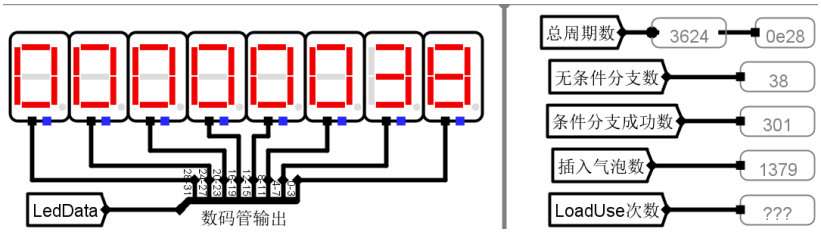

图 4.2 气泡流水线 CPU 测试

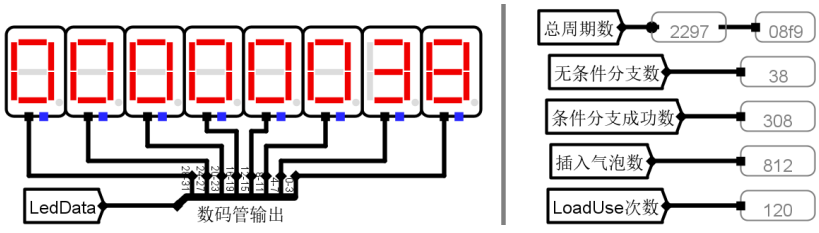

图 4.3 重定向流水线 CPU 测试

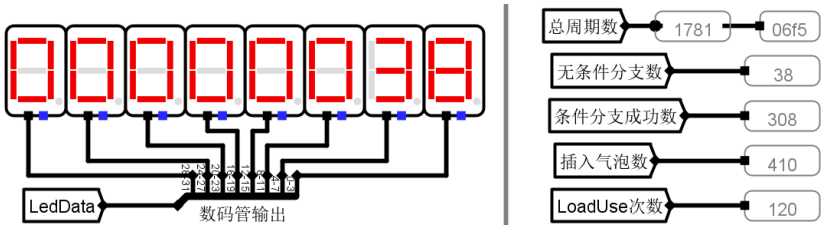

图 4.4 分支预测流水线 CPU 测试

4.1.2 测试用例 2

使用测试用例 risc-v 单级中断测试程序测试单级中断电路。按下按钮 1，程序成功进入中断程序 1，如图 4.5 所示。接着按下按钮 2，中断指示灯 W2 亮起，如图 4.6 所示，中断程序 2 进入等待状态，当中断程序 1 返回后，自动进入中断程序 2。

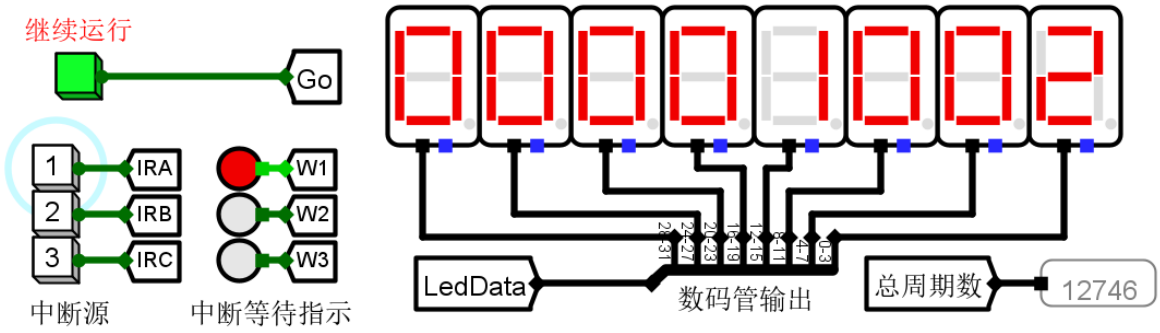

图 4.5 单级中断测试 1

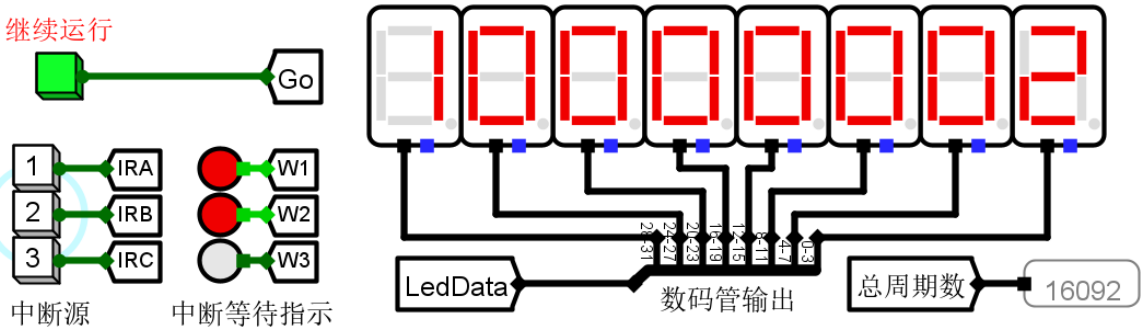

图 4.6 单级中断测试 2

4.1.3 测试用例 3

使用测试用例 risc-v 多级中断测试(EPC 硬件堆栈保护) 测试多级中断电路。按下按钮 1，程序成功进入中断程序 1，如图 4.7 所示。接着按下按钮 2，中断指示灯 W2 亮起，如图 4.8 所示，中断程序 1 被打断，直接进入进入中断程序 2，中断程序 2 执

行完毕后，自动从中断程序 1 被打断处继续执行。

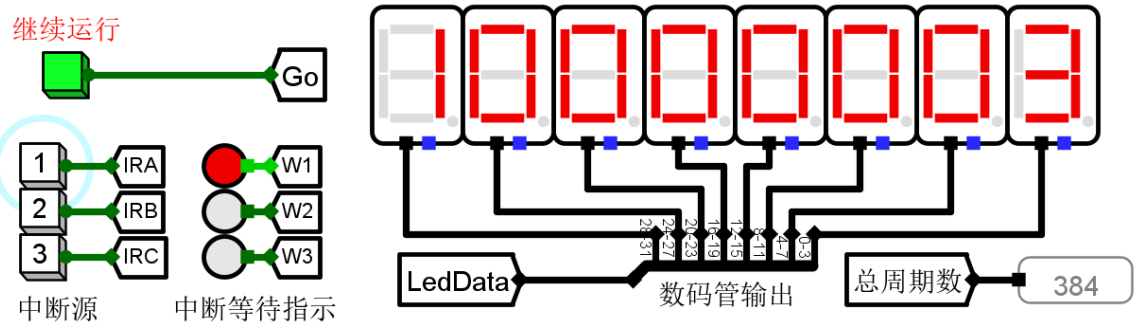

图 4.7 多级中断测试 1

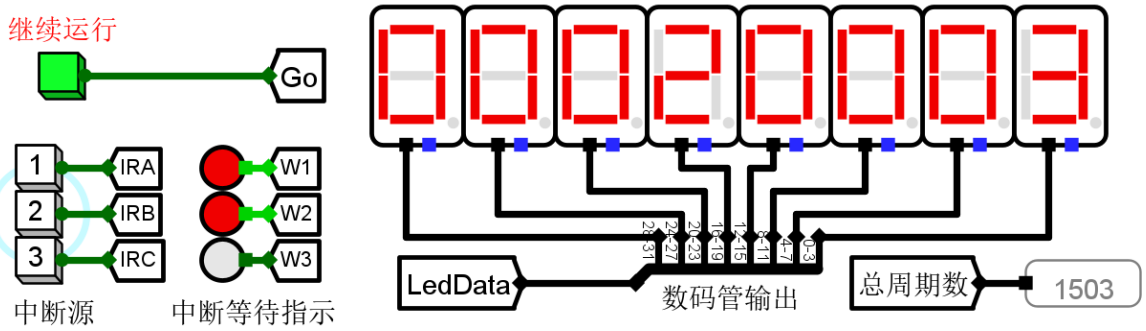

图 4.8 多级中断测试 2

4.2 性能分析

各 CPU 运行基础 benchmark 程序总周期数如表 4.1 所示。

表 4.1 各 CPU 运行总周期数

CPU	benchmark 程序运行总周期数
单周期	1546
气泡流水线	3624
重定向流水线	2297
动态分支预测	1781

单周期 CPU 的总周期数最少，但由于单周期 CPU 时钟周期比流水 CPU 的更长，实际性能是最差的。气泡流水线由于插入的气泡太多，PC 经常阻塞，总周期最多，性能次差。重定向流水线优化了数据冲突处理，减少了气泡数量，周期数减少，性能较好。支持动态分支预测的重定向流水线性能最佳，降低了分支冲突处理耗费的时延，减少气泡数量，显著减少了总周期数。

4.3 主要故障与调试

4.3.1 停机信号 halt 出错

气泡流水线程序暂停时，输出与预期不一致。

故障现象：程序暂停时 LED 输出与预期不一致。

原因分析：错误使用了 EX 段得到的 halt 信号，导致 PC 被暂停时，MEM 和 WB 段的指令停在流水线中未执行。

解决方案：将 halt 通过流水寄存器向后传递到 WB 段，使用 WB 段的 halt 信号。

4.3.2 LH 指令错误

CCAB 中的 LH 指令出现问题。

故障现象：运行测试 LH 指令的代码时，输出与预期不一致。

原因分析：数据寄存器按 4 字节编址而不是按字节编址。而 LH 指令要求内存地址必须是 2 的倍数，当 LH 操作的内存地址不是 4 的倍数时，输入数据寄存器的地址的精度会不够，区分不出来。

解决方案：判断输入数据寄存器的第一位是否是 1，如果是 1，说明不是 4 的倍数，输出 16-31 位；否则是 4 的倍数，输出 0-15 位。

4.4 实验进度

时间跨度太长，已经忘了。

5 设计总结与心得

5.1 课设总结

在这次硬件综合训练中，我作了如下几点工作：

1） 设计并实现了支持 24 条基础指令和 4 条扩展指令的单周期 CPU。

2） 设计并实现了理想流水线 CPU。

3） 设计并实现了气泡流水线 CPU。

4） 设计并实现了重定向流水线 CPU。

5） 设计并实现了基于重定向流水线的支持动态预测的流水线 CPU。

6） 实现了单级中断

7） 实现了多级嵌套中断

8） 实现了单周期上开发板

9） 实现了流水线上开发板

10） 完成了团队任务“米字棋”，实现了部分汇编代码，以及 logsim 上用硬件实现中断处理和渲染的部分。

5.2 课设心得

这次课程设计难度很大，时间跨度也很广（从学期开始到学期结束），既有个人任务又有团队任务。耗费了大量时间完成了所有的任务之后，我获得了巨大的成就感，对 CPU 的工作原理有了深刻的理解。

再任务刚开始时，我感到一头雾水，不知从何下手。再看完任务指导书再复习了理论课的知识之后逐渐上手。理解了 CPU 处理指令的流程，一步步完成了各种指令的数据通路。为了实现各种指令，开始一条条查看指令的功能，特别是对立即数的处理。一开始我不理解 ecall 指令的功能，在看来汇编代码之后才想到 ecall 是在执行系统调用。完成基础指令之后，我对这些指令的功能都变得了如指掌，也为我实现之后的团队任务中的汇编代码提供了帮助。

在实现 ccab 的时候，我发现 AUIPC 不能放到现有的数据通路之中，又增加了 U 型指令的数据通路。实现新的功能时，出现了许多与预期不一致的输出，经过一步步

调试，查看寄存器和数据存储器的值，最终找到了问题所在。这个过程很折磨，需要很长的时间，但是也让我对 CPU 的理解进一步加深。

这个过程中，logsim 本身的一些问题也多次困扰我。有时一些线会藏在模块的背后看不见导致输出不对，而这种情况很难第一时间发现，导致浪费了大量时间。

在实现各种流水线处理冲突的机制时，我在网上查找了很多资料才弄清楚，如果在实验指导手册中进行更加清楚的讲解会比较好。

在实现 CPU 上板时，我遇到了比预期多得多的困难。很大一部分原因是 verilog 代码的调试很不方便（可能是缺乏了这方面的经验），很难找到问题处在哪里。

最后完成团队任务时，我感受到了没有操作系统也没有库函数的情况下运行一段程序有多么困难，特别是 CPU 对于指令的支持不全。一开始尝试用能够支持的指令来代替，但是又会出现许多新的问题，最后还是选择了增加 CPU 支持的指令。这次团队任务也让我复习了汇编的知识。

最后，十分感谢各位老师为实验做出的付出。这种十分难得大型实验对学生的成长是大有裨益的。

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 5 版).北京：机械工业出版社.

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

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

[4] 谭志虎，周健，周游.计算机组成原理实验指导（基于 RISC-V 在线实训）. 北京：人民邮电出版社，2024 年.

[5] 曹强，施展.计算机系统结构（微课版）. 北京：人民邮电出版社，2024 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：王文涛