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：	CS1907
学　　号：	U201915146
姓　　名：	向隆航
电　　话：	15874360548
邮　　件：	1171350311@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 21 条基本 32 位 RISC-V 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 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
2	ADDI	立即数加	
3	AND	与	
4	ANDI	立即数与	
5	SLL	逻辑左移	
6	SRA	算数右移	
7	SRL	逻辑右移	
8	SUb	减	
9	OR	或	
10	ORI	立即数或	
11	XORI	立即数异或	
12	LW	加载字	
13	SW	存字	

#	指令助记符	简单功能描述	备注
14	BEQ	相等跳转	
15	BNE	不相等跳转	
16	SLT	小于置数	
17	SLTI	小于立即数置数	
18	SLTU	小于无符号数置数	
19	JAL	转移并链接	
20	JALR	转移到指定寄存器	If $a7==10 halt(停机指令)
21	ECALL	系统调用	else 数码管显示$a0 值
22	CSRRS	访问 CP0	中断相关，可简化，选做
23	CSRRC	访问 CP0	中断相关，可简化，选做
24	URET	中断返回	异常返回，选做
25	SRA	算术右移	
26	XOR	异或	
27	SB	将最低位字节写入	
28	BGEU	无符号大于等于时分支	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 本次我们采用的方案是硬布线控制，所有的指令执行是定长的指令周期，即取所有指令里面最慢的那条指令进行同步，且实现指令和数据分开存储的方式，即哈佛结构，运算器和 PC 累加器分离。通过控制器根据读入的指令给出的相应控制信号，其余各功能部件根据得到的控制信号完成相应功能。同时在实施的过程中，全部采用 Logisim 仿真实现。

总体结构图如图 2.1 所示。

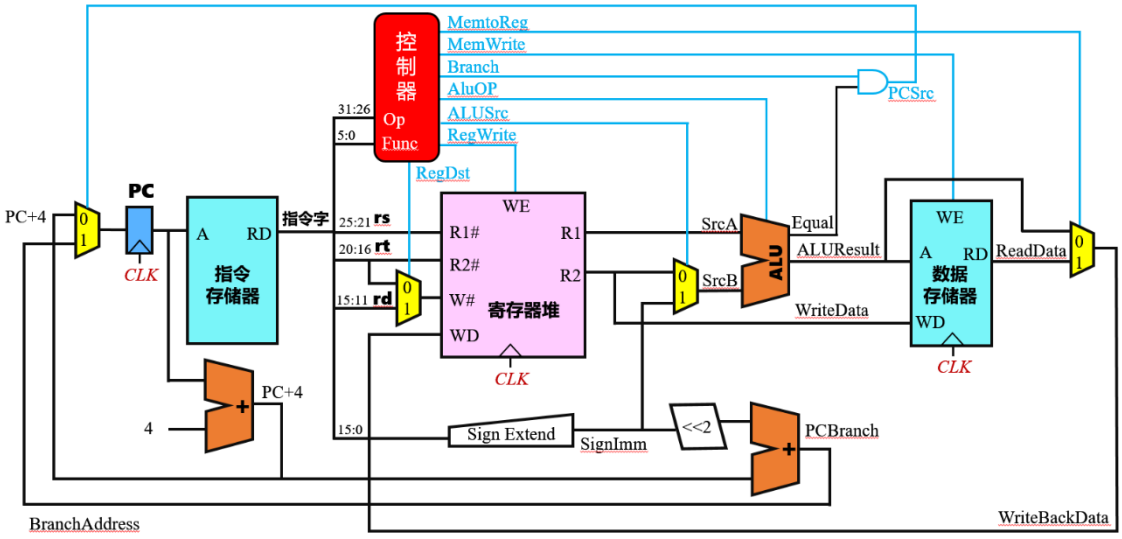

图 2.1 总体结构图

2.1.1 主要功能部件

主要的功能部件有：程序计数器 PC，指令存储器 IM，寄存器组，运算器 ALU，数据存储器 DM，单周期硬布线控制器等。具体设计思路如下：

1. 程序计数器 PC

程序计数器 PC 存储下一条指令在指令存器中的地址。每条指令 32 位，4 个字

节，所以一般情况下一条指令的地址就是 PC+4。其他情况根据相应指令输入下一条指令的地址。注意满足 ecall 指令的要求时，PC 不能继续读入新的指令地址达到停机的目的。

2. 指令存储器 IM

用于存储二进制的 RISC-V 指令。在取指令时根据 PC 寄存器输出端口的值来进行指令的读取，一条指令占 4 个字节，所以需要将读取的 PC 的值去掉末两位再取第 2-11 位作为地址进行读取。

3. 运算器

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
Begu	输出	1	Begu=(x>=y)?1:0,对所以操作有效
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

表 2.2 ALU_OP 和功能

ALU_OP	十进制	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)

ALU_OP	十进制	功能
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

4. 寄存器堆 RF

寄存器堆 RF 包含 32 个通用寄存器。

表 2.3 寄存器堆 RF

引脚	功能
R1#	读寄存器 1 的编号
R2#	读寄存器 2 的编号
W#	写寄存器的编号
Din	写入目的寄存器的数据内容
WE	控制器的写使能信号
Clk	时钟信号
R1	[R1]寄存器中的数据
R2	[R2]寄存器中的数据

2.1.2 数据通路的设计

表 2.4 指令系统数据通路框架

指令	PC	RF				ALU			DM	
		R1#	R2#	W#	Din	A	B	OP	Addr	Din
RR 型指令	PC+4	Rs1	Rs2	Rd	Alu result	R1	R2	OP		

指令	PC	RF				ALU			DM	
		R1#	R2#	W#	Din	A	B	OP	Addr	Din
RI 型指令	PC+4	Rs1		Rd	Alu result	R1	Imm12	OP		
分支指令	PC+4||Imm(8-11,25-30)	Rs1	Rs2			R1	R2	OP		
Jal 指令	PC+Imm(21-30,12-19)			Rd	PC+4					
Jalr 指令	[Rs1]+Imm12	Rs1		Rd	PC+4	R1				
Store 型指令	PC+4	Rs1	Rs2			R1	Imm	OP	Alu result	R2
Load 型指令	PC+4	Rs1		Rd	Data	R1	Imm	OP	Alu result	
Ecall 指令	PC+4	A0	A7							

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.。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
RegWrite	0/1	寄存器写使能
MemWrite	0/1	写内存控制信号
AluOP	0-12	运算器操作控制符
MemToReg	0/1	寄存器写入数据来自存储器
S_Type	0/1	S 型指令译码信号
AluSrcB	0/1	运算器 B 输入选择
JALR	0/1	JALR 指令译码信号
JAL	0/1	JAL 指令译码信号
SB	0/1	SB 指令译码信号
Beq	0/1	Beq 指令译码信号
Bne	0/1	Bne 指令译码信号
Begu	0/1	Begu 指令译码信号

控制信号	取值	说明
Ecall	0/1	ecall 指令译码信号
CSRRSI	0/1	CSRRSI 指令译码信号
CSRRCI	0/1	CSRRCI 指令译码信号

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 2.6 所示。

表 2. 6 主控制器控制信号框架

指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr	SB	BGEU	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	0	1	4	0			1	1										
srli	0	5	4	2			1	1										
srai	32	5	4	1			1	1										
lw		2	0	5	1		1	1										
sw		2	8	5		1	1			1								
ecall	0	0	1c						1									
beq		0	18								1							
bne		1	18									1						
jal			1b					1					1					
jalr		0	19				1	1						1				
CSRRSI		6	1c														1	
CSRRCI		7	1c															1
URET																		
SRA	32	5	c	1				1										
XOR	0	4	c	9				1										
SB		0	8	5		1	1			1					1			
BGEU		7	18													1		

2.2 中断机制设计

2.2.1 总体设计

本实验只考虑可被屏蔽的外部中断。对于外部中断，有硬件支持的中断也有软件支持的中断。首先对于单级中断，不用考虑优先级，由于中断完成后要返回进入中断前的地址并恢复整个 CPU 的状态，所以在进入中断时首先保存 CPU 的状态，将 PC 的状态保存在 EPC 中，这样在完成中断程序退出中断后将 EPC 中的值返回给程序寄存器 PC。实现多级中断时，多级中断存在优先级，故要对不同等级的中断进行处理，需要判断新的中断能否打断旧中断；在退出多级中断程序时也要判断是执行上一次被打断的低级中断还是开始执行在排队的中断。多级中断进入中断前的 PC 应保存在堆栈中。

中断执行流程如图 2.1 所示:

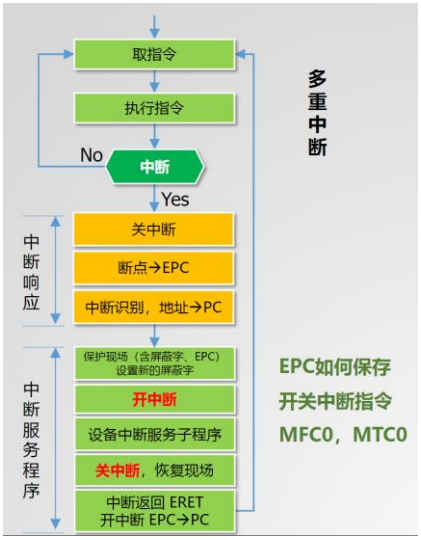

图 2.1 中断执行流程

本次实验最多可响应 3 个中断，每个中断对应一个中断号，根据中断号选择执行的中断服务程序，将每个中断服务程序入口地址作为常量按照优先级组合作为多路选择器的输入来实现，具体的中断服务程序入口地址可以通过的 MARS 仿真器工具手动查看。

在跳转前保存对应指令地址，以便在执行完中断服务程序后恢复现场。通过 3 个中断屏蔽寄存器动态改变优先级达到多级中断的实现。需要添加中断使能寄存器 IE 执行开关中断的功能，高电平有效。实现多级中断时，在测试程序中还有 CSRRCI 和 CSRRSI 指令，以及中断返回指 URET，故控制器和数据通路需要修改。

2.2.2 硬件设计

设计设计分为以下四个部分

1.中断请求生成逻辑:3 个中断源对应三个电路，产生 IR1,IR2,IR3，锁存产生的中断请求，在中断服务后清除中断。

2.中断优先级仲裁:对于接受到的不同的中断信号，通过这个部件判断要执行哪个中断服务程序。

3.中断响应周期:在进入中断服务程序前保存进入中断服务程序前的地址，然后跳转到优先级仲裁判断的下一个要执行的中断服务程序入口地址

4.与中断相关的寄存器:异常程序计数器 EPC 用来保存断点；中断使能寄存器 IE 是开关中断，支持开和关两个操作控制信号。

2.2.3 软件设计

较单级中断，需要新增对指令 CSRRCI 和 CSRRSI 的支持，当控制器检测到这两条指令的时候，给出“CSRRCI”和“CSRRSI”的信号，以便多级中断电路能及时开关中断。保护现场后执行 CSRRSI 开中断，恢复现场之前执行指令 CSRRCI 关中断。

2.3 流水 CPU 设计

2.3.1 总体设计

RISC-V 指令在单周期 CPU 流水线中执行过程分为 5 个阶段：取指令阶段（IF）、译码取数阶段（ID）、 指令执行阶段（EX）、访存阶段（MEM）、写回阶段（WB）。通过流水接口部件完成相应信号量的锁存和传递，使不同阶段能够处理不同的指令，

IF 阶段根据 PC 获取指令寄存器中的指令，向后传递；ID 阶段将输入的指令拆分由单周期硬布线控制器生产控制信号，并读取寄存器操作数等，向后传递；EX 阶段利用 ALU 进行运算，或计算分支地址等；MEM 阶段向数据存储器存数据或读数据；WB 阶段将 ALU 计算结果或从内存读出的数据写入到目的寄存器。

2.3.2 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。流水接口在使能为高电平时，每一个时钟周期从所有的数据输入端读入数据并将其缓存在寄存器中，数据输出端的值与寄存器保持一致。若清零端为高电平则进行同步清零。每一个接口部件都向后传递 PC 和 IR，方便流水线的实现。

2.3.3 理想流水线设计

理想流水线不需要我们考虑数据冲突、结构冲突和分支冲突等等问题，只需要考虑数据通路的实现，再将对应的信号连接到对应的模块端口即可。理想流水线设计不涉及分支指令，因此不需要考虑分支跳转和无条件跳转指令，故只需要在单周期 CPU 设计中按照 CPU 流水线的五个阶段将部件划分清楚然后加入流水接口部件正确传递每一个需要传递的值即可。

2.4 气泡式流水线设计

气泡式流水线通过插入气泡的方式消除指令之间的数据冲突和分支冲突。分支冲突是指分支指令还没来得及跳转，紧跟其后的若干指令就进入了流水线，导致执行出错，气泡式流水线采用插入气泡的方式来消除分支冲突，即在分支跳转发生时，将分支指令后进入流水线的指令清空，一般分支跳转发生在 EX 段，故可在检测到分支跳转后对 IF,ID 段的流水线进行同步清零，如图 2.2。

图 2. 2 分支相关流水线时空图

数据冲突是指流水线后阶段的指令需要读取的寄存器与前面指令需要写入的寄存器相同，但前面的指令还没有把新数据写入寄存器。解决方法是暂停后面指令，让前面的指令继续执行，直到写入新的数据。故在发生数据冲突的两个指令之间插入气泡即可。具体是若是 ID 与 WB 冲突，则将寄存器文件设置为下降沿触发来解决，即在一个时钟周期的前半个周期写，后半个周期读；若是 ID 与 MEM 冲突，则暂停 PC 和 IF 段，在 ID 段插入一个气泡即可；若是 ID 与 EX 冲突，暂停 PC 和 IF 段，在 ID 段插入一个气泡，然后会变为 ID 与 WB 冲突。

2.5 数据转发流水线设计

相对于气泡式流水线，数据转发流水线对于发生数据冲突的两条指令，通过数据重定向旁路直接将发生数据冲突的前一条指令产生的新的数据重定向到后面需要新数据的指令的地方，一般重定向在 ALU 前，如图 2.3。

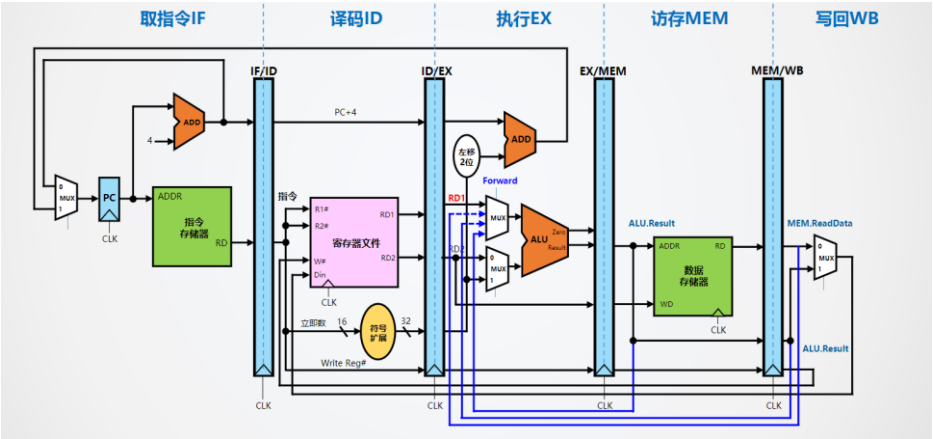

图 2.3 数据转发流水线设计

如果出现相邻两条指令存在数据冲突，且前一条指令是访存指令时（称为 Load-Use 相关），不能采用重定向方式进行处理，仍采用插入气泡来解决。在 LoadUse 信号产生时，暂停 IF、ID 端的行为并向 EX 端中插入一个气泡。

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，经过非门之后为 0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。PC 的输入端针对不同的指令下一条指令地址也不尽相同，用多路选择器完成选择。

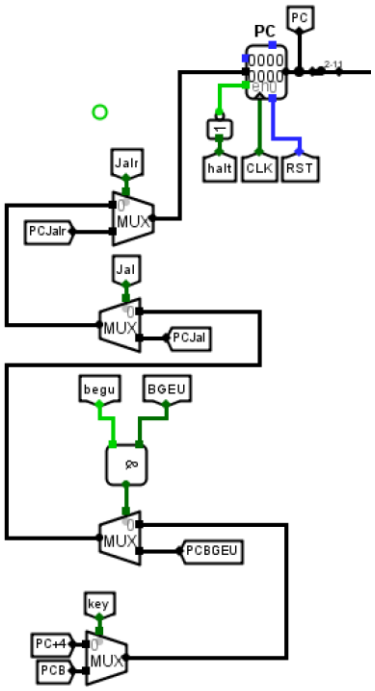

图 3.1 程序计数器（PC）

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。

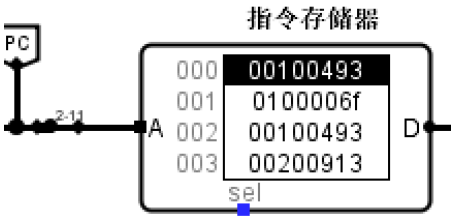

图 3.2 指令存储器（IM）

3) 运算器（ALU）

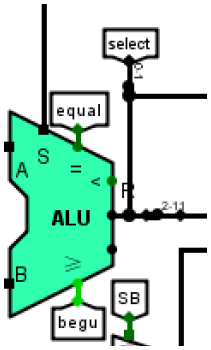

图 3.3 运算器 ALU

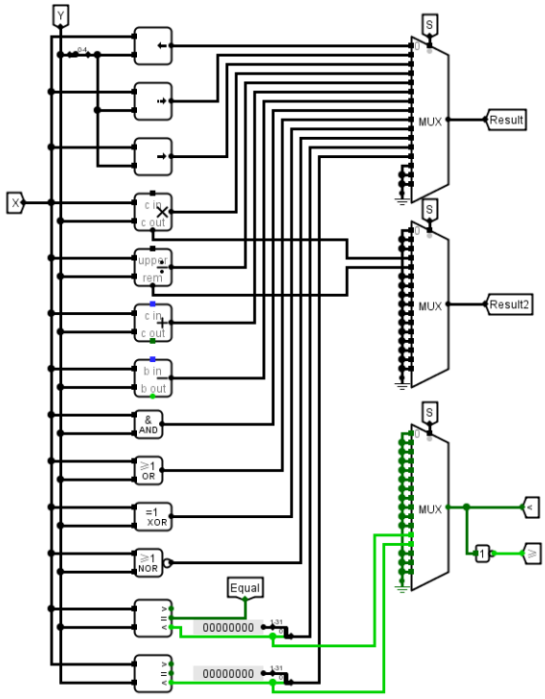

图 3.4 运算器内部构造

运算器支持多种类型的运算，由 AluOP 控制信号选择运算方式，然后输出结果，封装后如图 3.3 所示。在运算器内部，提供了所有的指令运算可能用到的运算功能。具体的内部结构如图 3.4

4) 数据存储器（DM）

DM 使用一个 RAM 实现,该地址位宽为 10 位，数据位宽为 32 位。因为计算得出的地址为 32 位，而 RAM 地址线宽度有限，仅为 10 位，故将 2 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 3.5 所示。

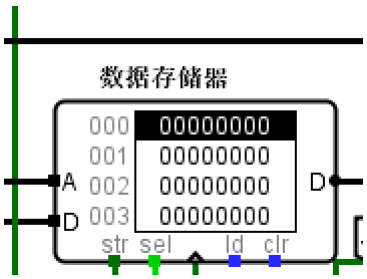

图 3.5 数据存储器

5) 寄存器堆（RF）

寄存器堆是一堆寄存器的结合。内部构造如图 3.6 所示

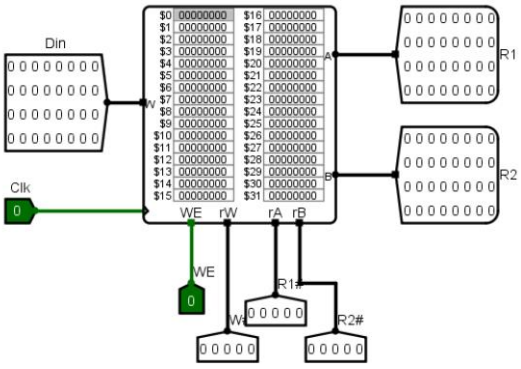

图 3.6 寄存器堆

3.1.2 数据通路的实现

完成主要功能部件的设计和实现后，将各个部件组合连接之后得到如图 3.7 所示的数据通路。

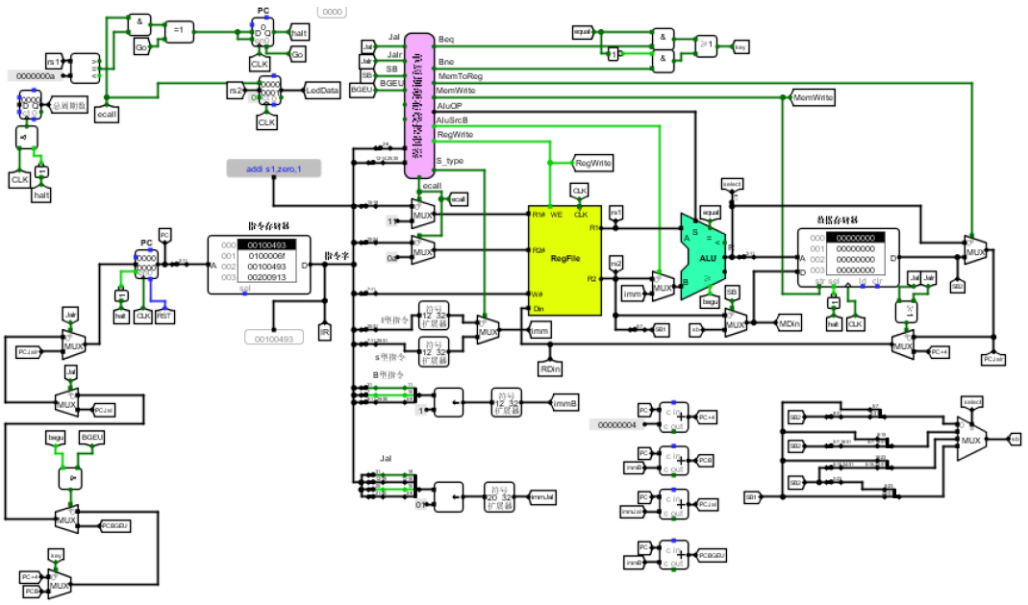

图 3.7 单周期 CPU 数据通路

具体的停机指令 ecall 设计如图 3.8

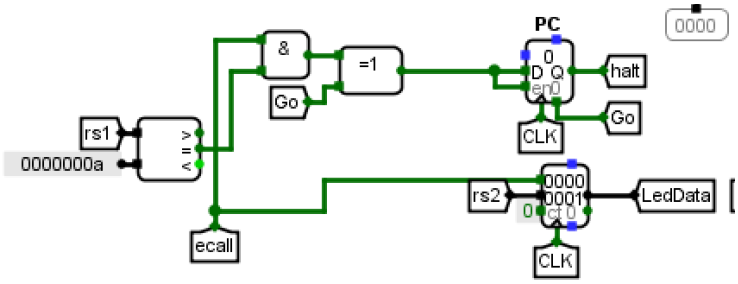

图 3.8 ecall 设计

对于 SB 指令要根据 ALU 计算结果的 0-1 位来选择需要写入的字节的位置，如图 3.9 所示。Select 是 ALU 计算结果的 0-1 位作为多路选择器的选择端。

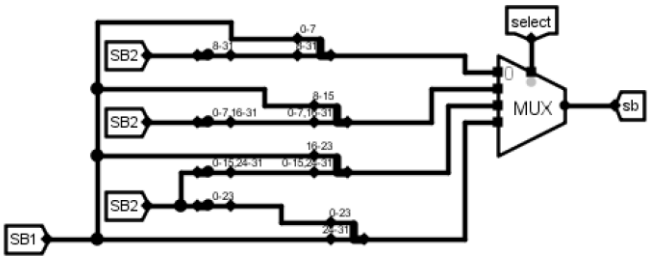

图 3.9 sb 设计

3.1.3 控制器的实现

填写好表 2.6 所示的控制信息表后，使用 Excel 预先定义好的函数得到相应电路

的规范表达式，使用 Logisim 的自动生成电路功能完成控制器的实现，最终单周期硬布线控制器如图 3.10 所示，其中的运算器控制器和控制器信号生成部件的电路太过复杂故没有贴图。

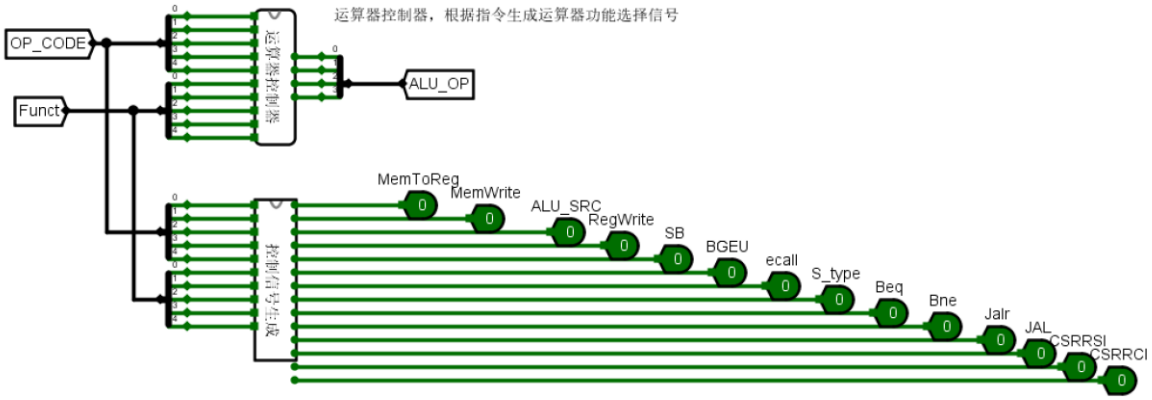

图 3.10 单周期硬布线控制器

3.2 中断机制实现

3.2.1 单周期单级中断

中断信号产生电路的设计实验包中给出的方式，如图 3.11 所示

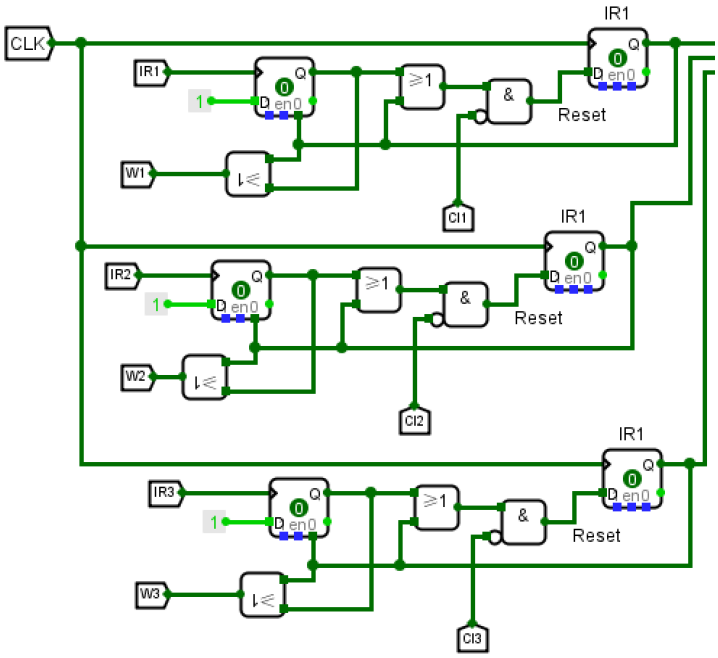

图 3.11 中断信号产生电路

中断信号产生后传输到优先级仲裁以及保存断点的电路，其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 信号如图 3.12 所示

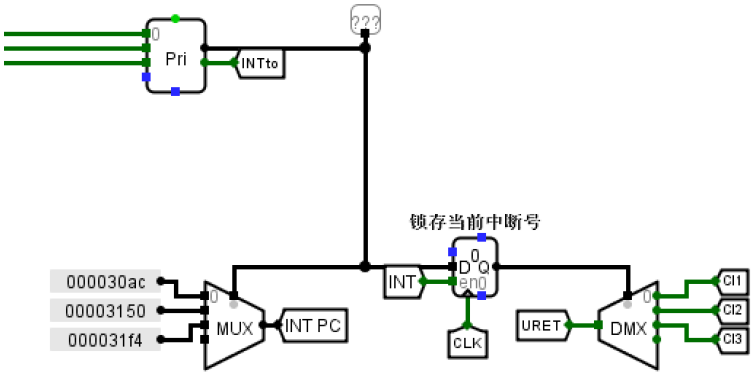

图 3.12 优先级仲裁以及中断号保存电路

然后在执行某个中断程序时关中断，中断返回后开中断，并保存返回地址到 Ret_PC，如图 3.13

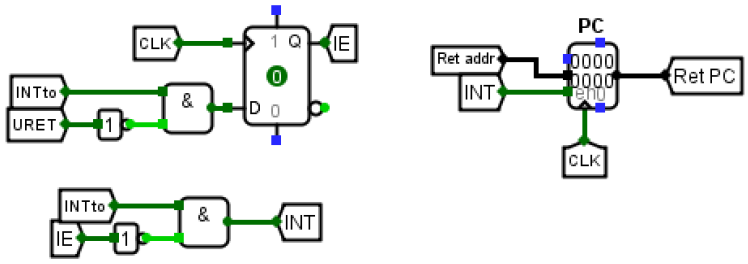

图 3.13 开关中断和返回地址电路

对于 PC 的输入端也需要修改如图 3.14 在原来的输入后再加两个多路选择器，一个是选择中断服务程序入口地址的，一个是关中断以后返回断点的。

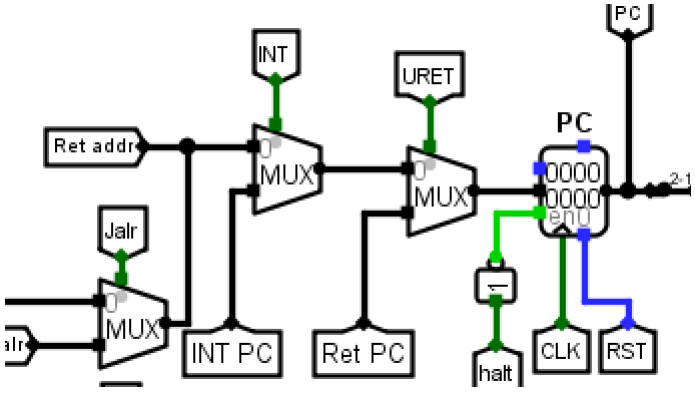

图 3.14 PC 输入端修改

整个单周期单级中断电路如图 3.15 所示

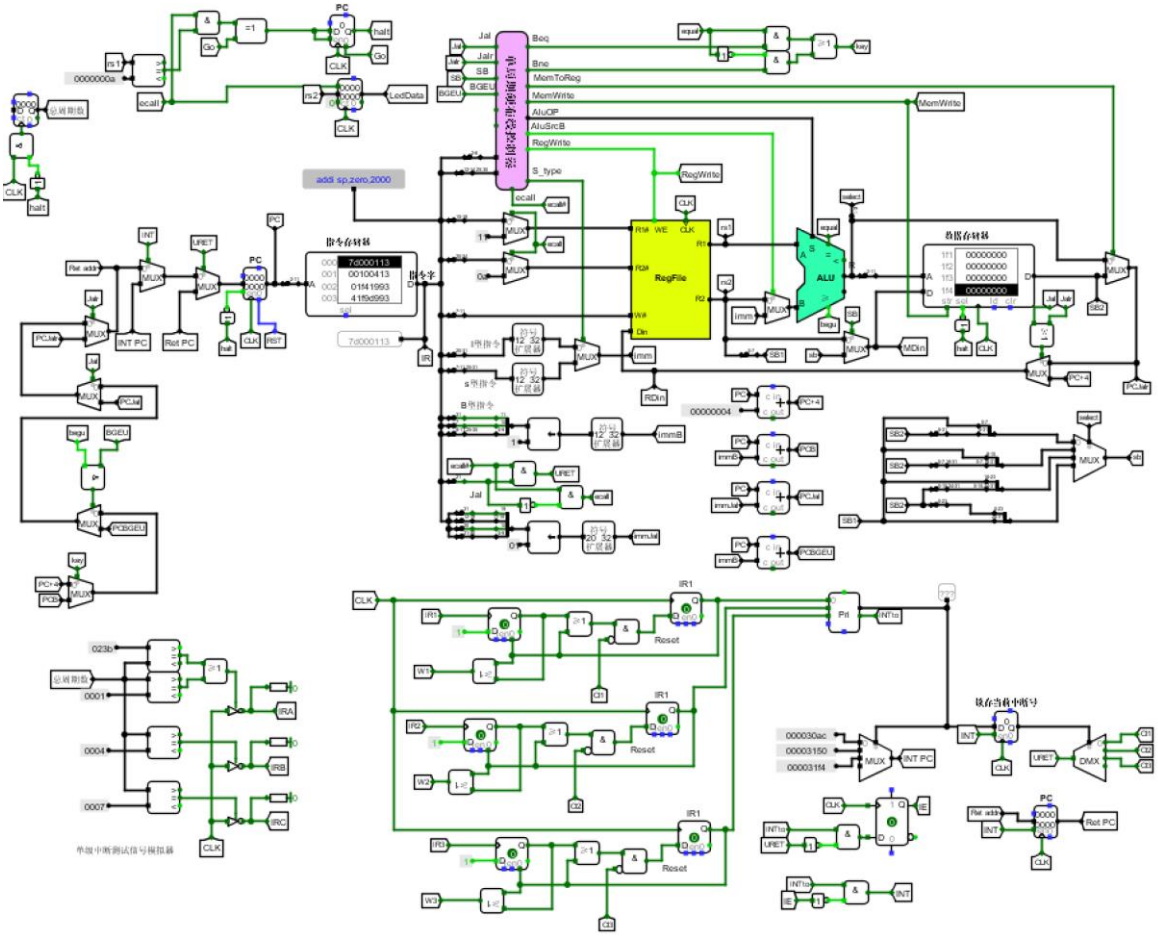

图 3.15 单周期单级中断电路

3.2.2 单周期多级中断

中断信号产生电路的设计与单级中断一致，然后就是多级中断的优先级仲裁以及对于多级嵌套中断要存储不同级别中断的中断号的硬件堆栈，由于最多三次嵌套所以有三个寄存器，如图 3.16.

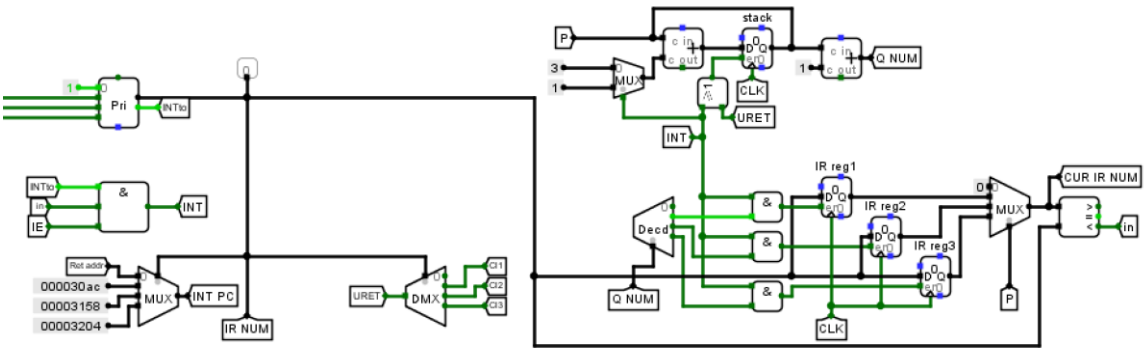

图 3.16 单周多级中断优先级仲裁电路

然后就是开关中断电路需要配合 CSRRCI，CSRRSI 以及 URET 这三个控制信号参与，如图 3.17 所示。

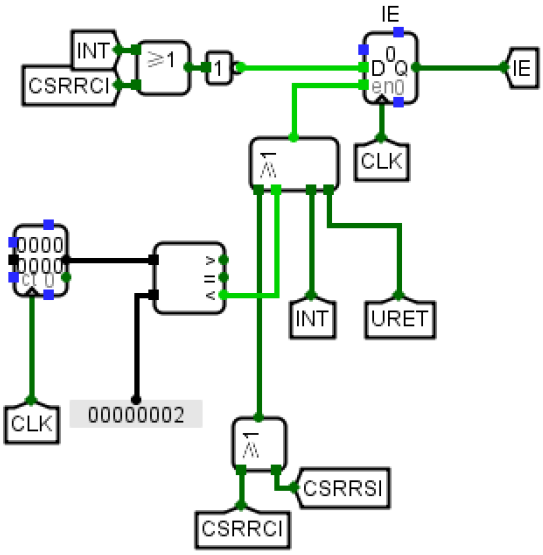

图 3.17 单周多级中断开关中断电路

最后就是实现硬件堆栈用于存储不同级别中断的断点达到完成优先级最高的中断后能回到被打断的优先级低的中断服务程序的断点，如图 3.18 所示。

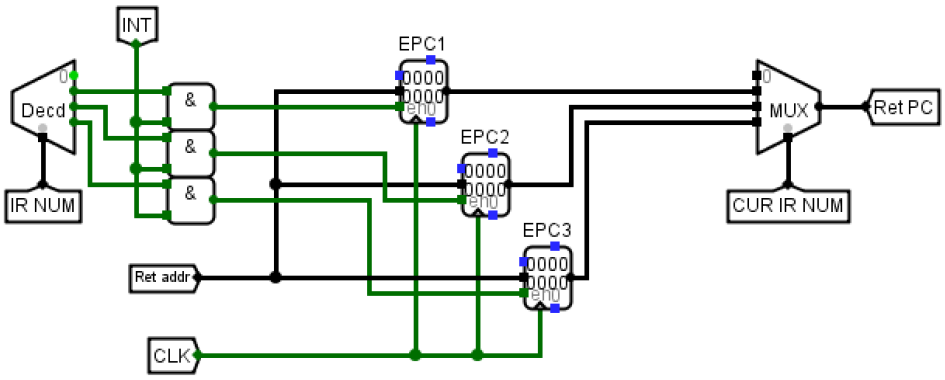

图 3.17 单周多级中断返回断点电路

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口部件将数据通路彼此相邻的段连接起来，内部采用寄存器进行锁存，寄存器均采用上升沿触发，这样就能保存上一个段的数据，供下一个流水段使用达到传

递各种信息的目的。IF/ID 段流水线接口其内部构造如图 3.18 所示。可以同步清零、插入气泡以及暂停流水线。（由于 ID/EX 等段的流水线接口需要传递很多的信息，故电路需要更多的寄存器来保存，所以显得电路很复杂，实际就是图 3.18 电路的复制。

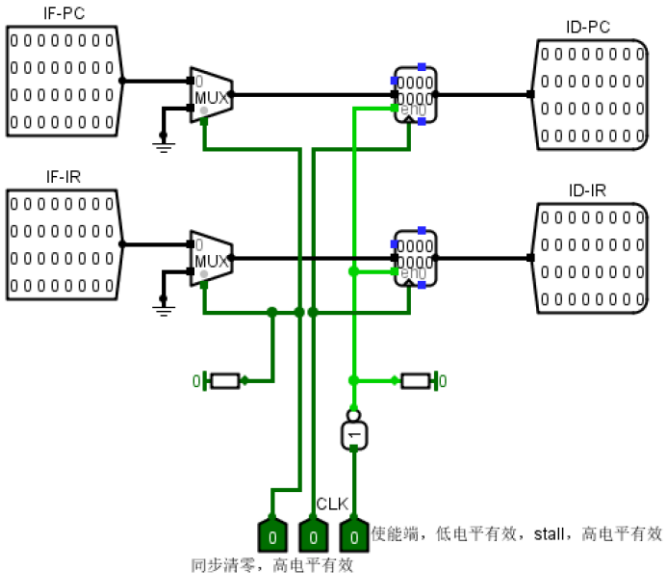

图 3.18 流水接口部件

3.3.2 理想流水线实现

理想流水线不用考虑分支跳转和数据相关等冲突，故只需要将每段流水线接口所需要的信息从上一段流水线接口传递下来即可。具体数据通路如图 3.19 所示。

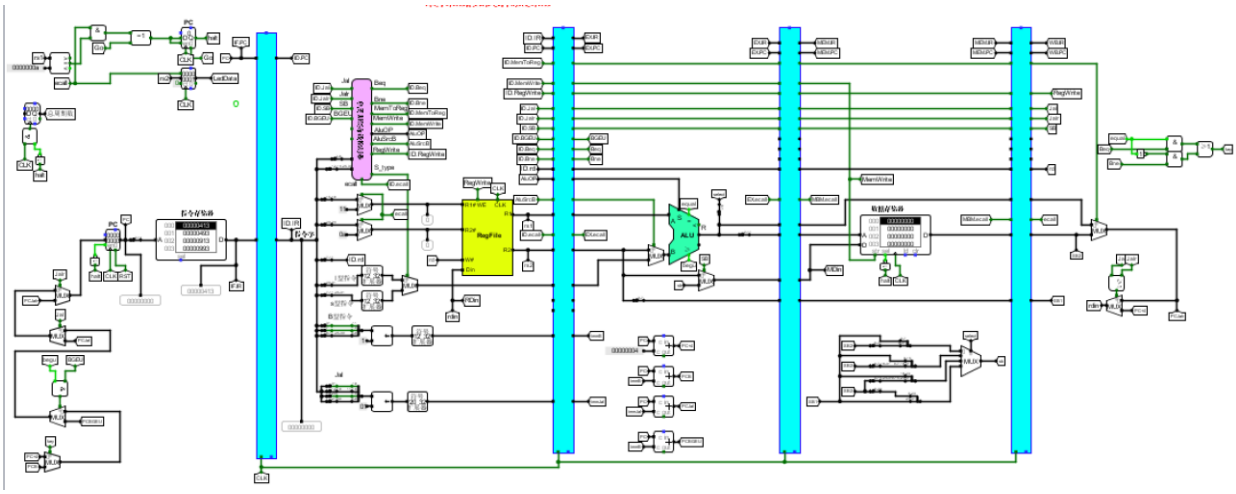

图 3.19 理想流水线实现

3.4 气泡式流水线实现

对于分支冲突检测是在 EX 段，当条件分支指令成功跳转以及发生无条件跳转时在 IF/ID 和 ID/EX 段插入气泡即同步清零。分支冲突检测电路如图 3.20。

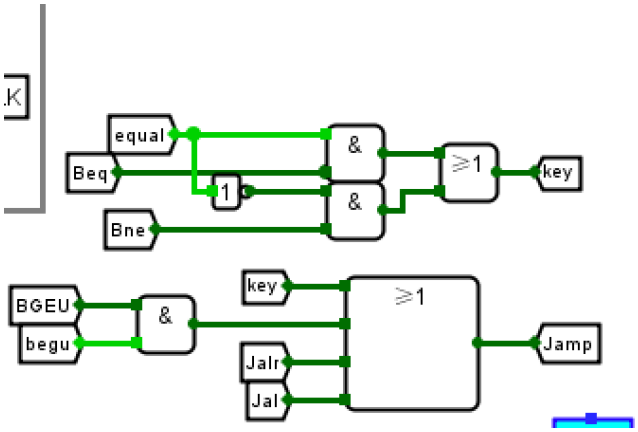

图 3.20 分支冲突检测电路

然后就是数据冲突检测首先需要检测后面指令是否需要读取寄存器，可以根据指令的 OP 和 Funct 字段来判断 ID 段要使用的寄存器，用 RS1_Used,RS1_Used 来表示。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，且写寄存器编好 WriteReg#是否和源寄存器编号相同即可。相关电路如图 3.21 所示

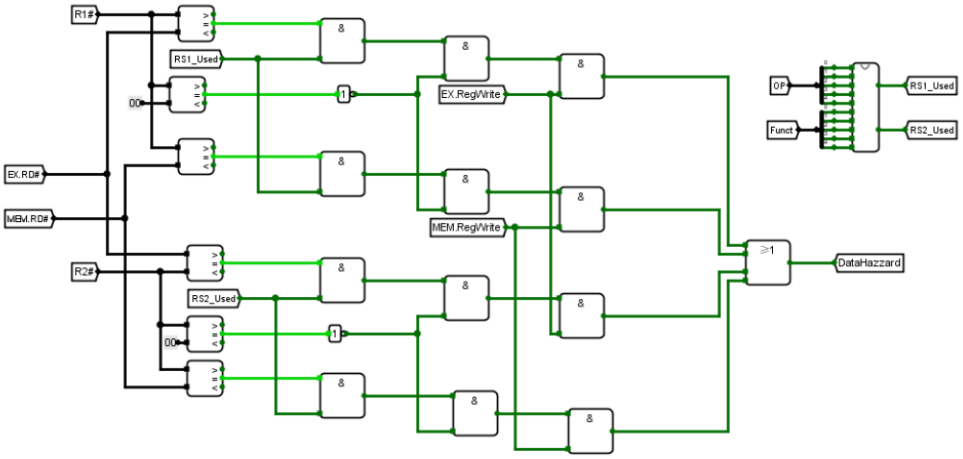

图 3.21 数据冲突检测电路

发生数据冲突时要在 ID/EX 段插入气泡即同步清零，同时暂停 PC 计数器和 IF/ID 段的流水线接口。气泡流水线数据通路如图 3.22 所示

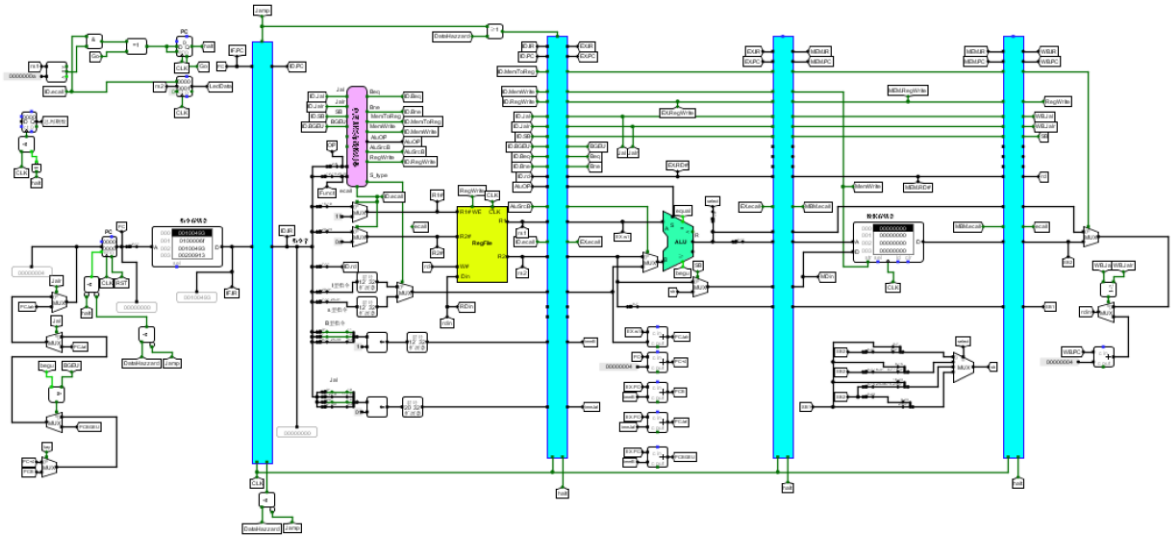

图 3.22 气泡流水线数据通路

3.5 数据转发流水线实现

由于数据转发流水线对于 LW 指令仍然采用插入气泡的方式消除数据冲突，故需要 Load-Use 电路来检测是否发生 LW 指令的数据冲突，发生时，在 ID/EX 段插入气泡，暂停 PC 计数器和 IF/ID 段。电路如图 3.23 所示

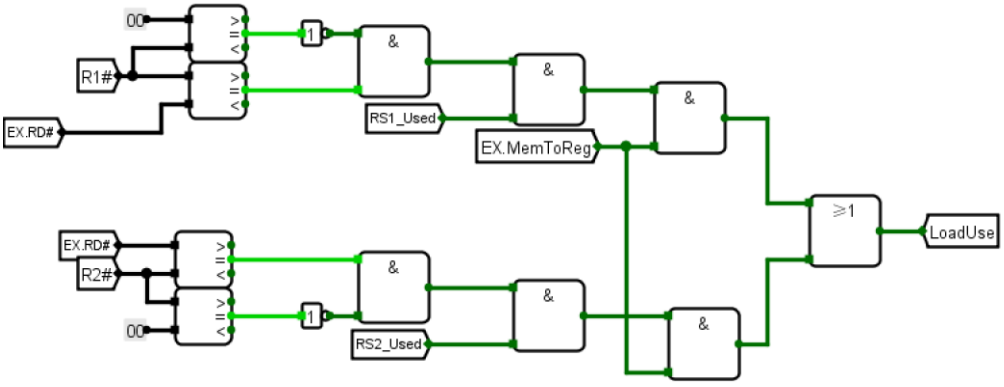

图 3.23 Load-Use 电路

为了能选择正确的数据输入 ALU，故需要在 EX 段产生 FwdRs1 和 FwdRs2 两个信号作为 ALU 输入端两个多路选择器的选择端，当其为 0 时表示没有发生数据冲突，输入为从寄存器中取出的数，为 1 时表示 ID 段与 EX 段数据冲突，故在 EX 段时要从 MEM 段中将数据重定向回来，为 2 时表示 ID 段与 MEM 段数据冲突，故在 EX 段中将 WB 段的数据重定向回来，为 3 时表示即与 EX 也与 MEM 发生数据冲突，只要在 EX 段将 MEM 段数据重定向即可。FwdRs 信号产生电路如图 3.24 所示，多路选择数据通路如图 3.25 所示

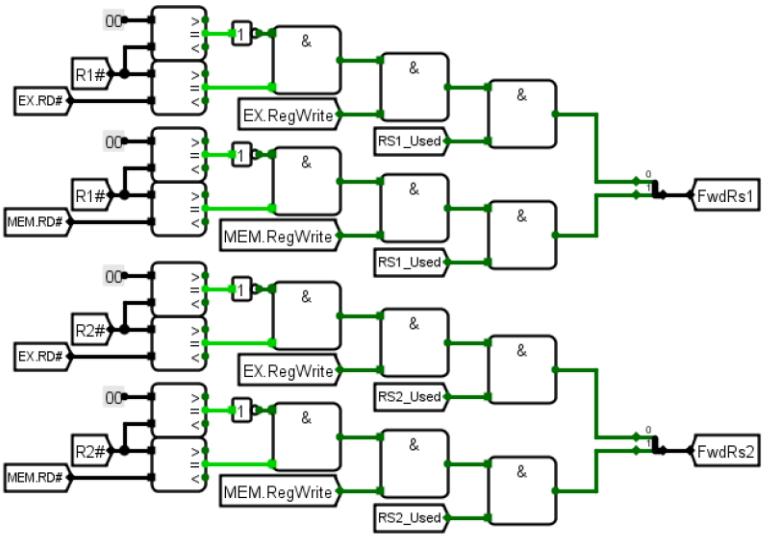

图 3.23 FwdRs 信号产生电路

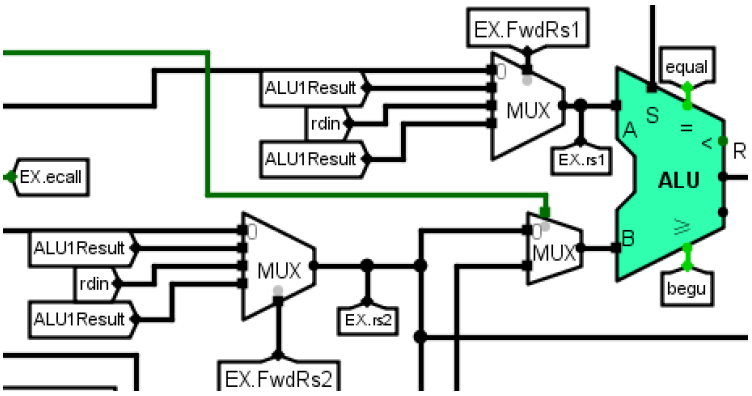

图 3.23 ALU 输入端多路选择电路

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 单周期 RISC-VCPU21+4 条指令

在 risc-v-benchmark_ccab.asm 内预留的 ccab 指令内容位置加入 SRA,XOR,SB,BGEU 这些指令的代码，然后利用汇编工具完成编译生成 risc-v-benchmark_ccab.hex 并在指令寄存器中加载。CPU21+4 运行结果如图 4.1-4.5.

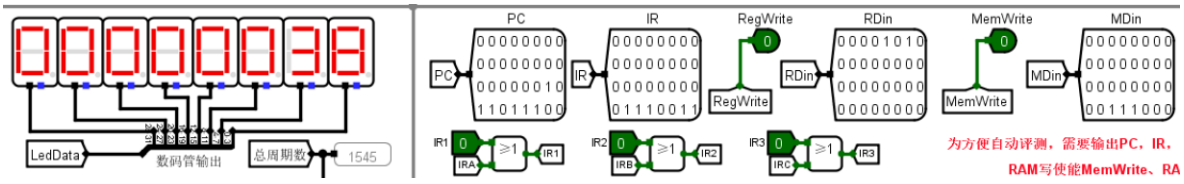

图 4.1 CPU21 条指令运行结果

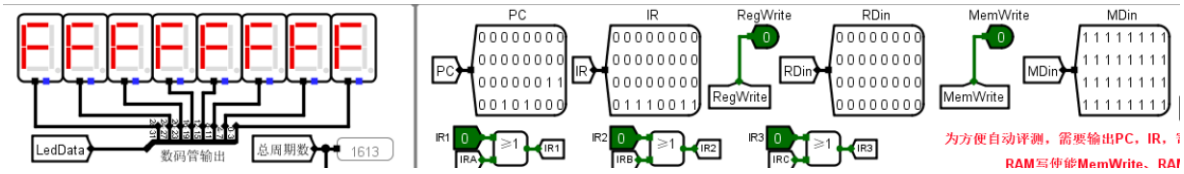

图 4.2 SRA 指令运行结果

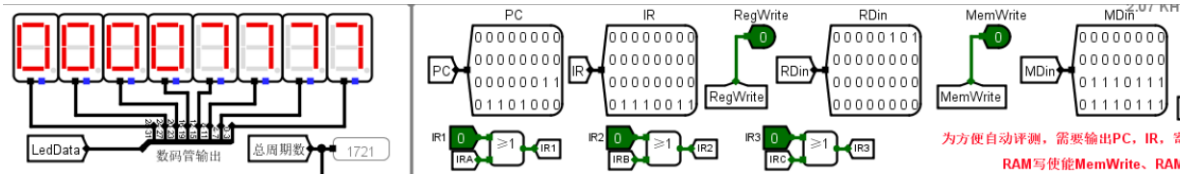

图 4.3 XOR 条指令运行结果

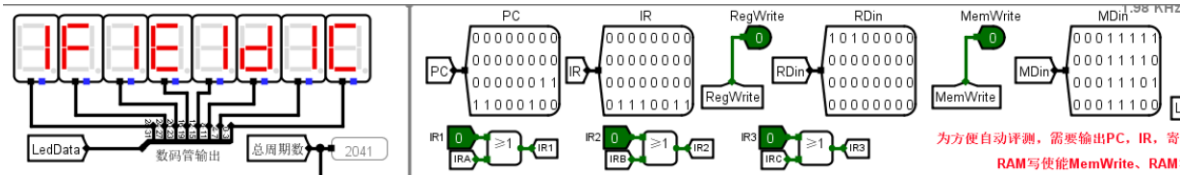

图 4.4 SB 条指令运行结果

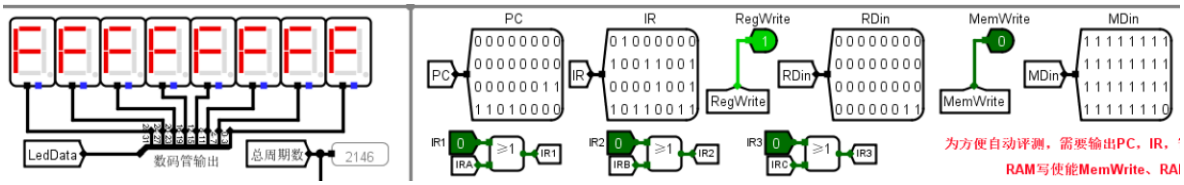

图 4.5 BGEU 条指令运行结果

4.1.2 理想流水线测试

在指令寄存器中加载 risc-v-理想流水线测试.hex，结果如图 4.6 所示

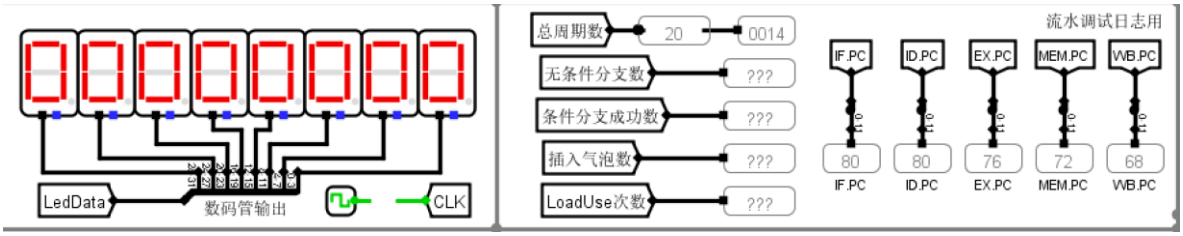

图 4.6 理想流水线运行结果

4.1.3 气泡流水线测试

在指令寄存器中加载 risc-v-benchmark.hex，结果如图 4.7 所示

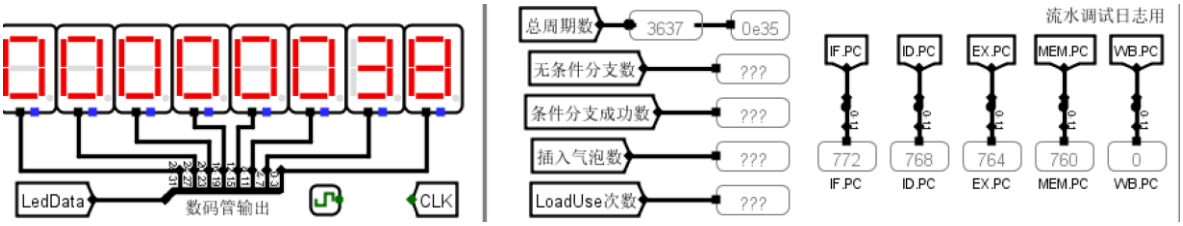

图 4.7 气泡流水线运行结果

4.1.4 重定向流水线测试

在指令寄存器中加载 risc-v-benchmark_ccab.hex，结果如图 4.8-4.12 所示

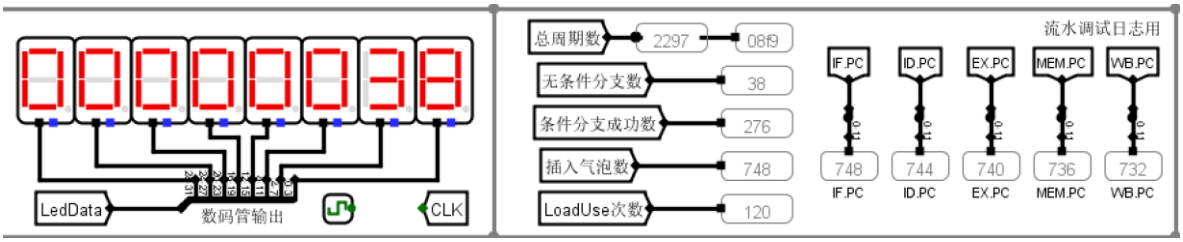

图 4.8 重定向流水线运行结果

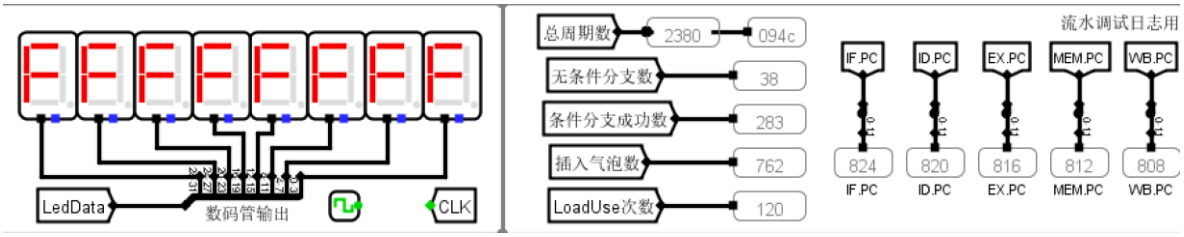

图 4.9 重定向流水线 SRA 运行结果

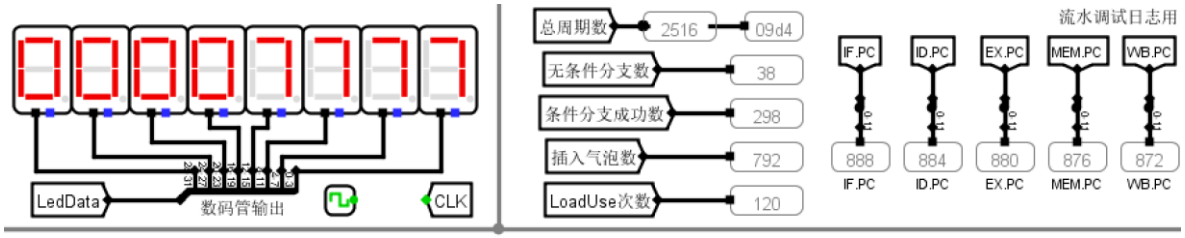

图 4.10 重定向流水线 XOR 运行结果

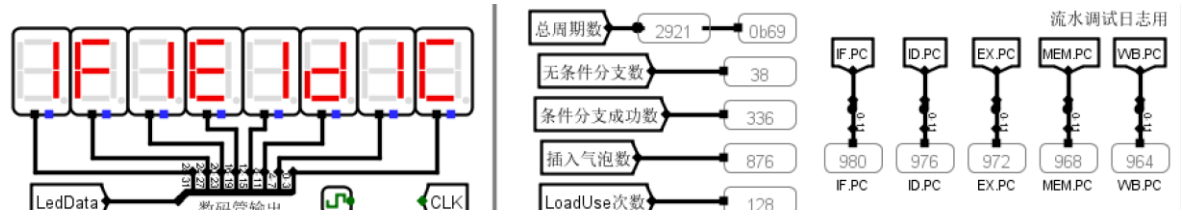

图 4.11 重定向流水线 SB 运行结果

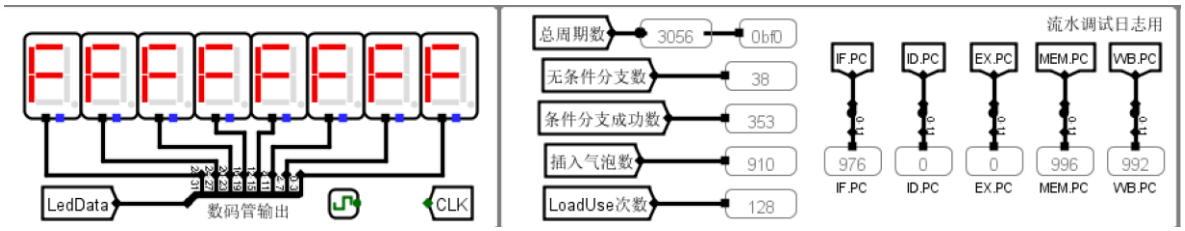

图 4.12 重定向流水线 BGEU 运行结果

4.1.5 单级中断测试

在指令寄存器中加载 risc-v 单级中断测试程序.hex，在程序运行开始后迅速依次按下中断 1，2，3，执行顺序为 1，3，2 运行过程截图如图 4.13 所示

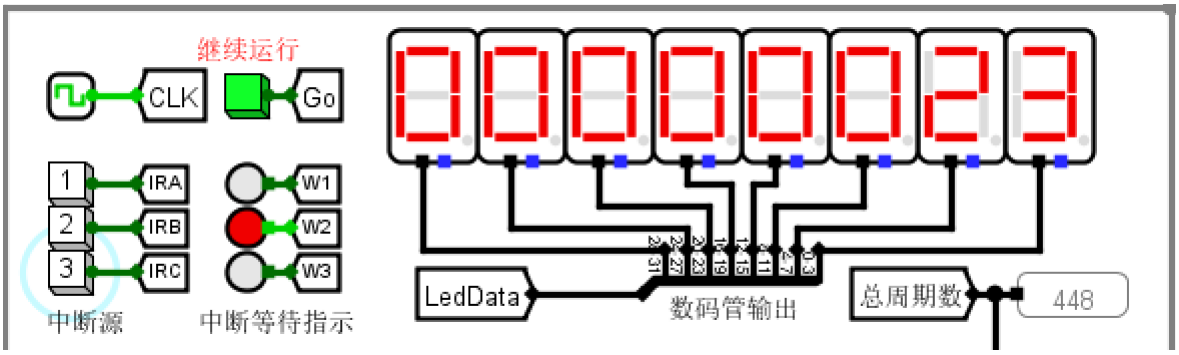

图 4.13 单级中断

4.1.6 多级中断测试

在指令寄存器中加载 risc-v 多级中断测试程序.hex，在程序运行开始后迅速依次按下中断 1，2，3，执行顺序为 3，2，1 运行过程截图如图 4.14 所示

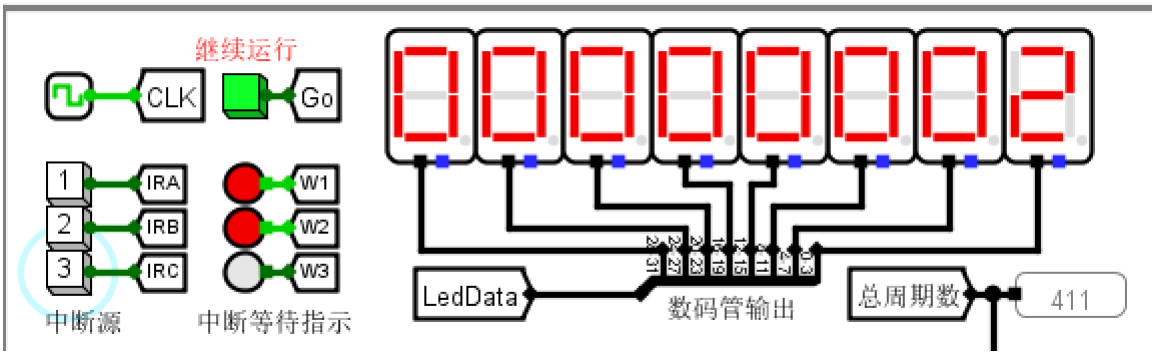

图 4.14 多级中断

4.2 性能分析

使用 risc-v-benchmark_ccab.hex 对单周期，气泡流水线和重定向流水线 RISC-VCPU 分别进行测试，最后测试的周期数分别为 1545、3637、2297.

与基本的单周期方案相比，各流水解决方案的周期数都更多，但周期更短，所以性能会有提升。气泡流水线因为数据相关和分支相关插入了大量的气泡，所以周期数最多；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，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。

4.3 主要故障与调试

4.3.1 单周期 RISCV 故障

故障现象：执行 sb 指令时最后几个 ledData 与预期不一致。

原因分析：sb 指令对于写入的新的单字节的位置固定为替换第一关字节。导致原本指令中要替换第 2，第 3，第 4 个字节变成了第一个字节。

解决方案： 在 ALU 运算结果中截取 0-1 位作为 sb 指令要写入新字节的位置，由 select 信号量表示，作为多路选择器的控制端。该多路选择器的输入段为要写入的新字节以及其他原来没被替换的字节，如图 4.15sb 指令实现所示。

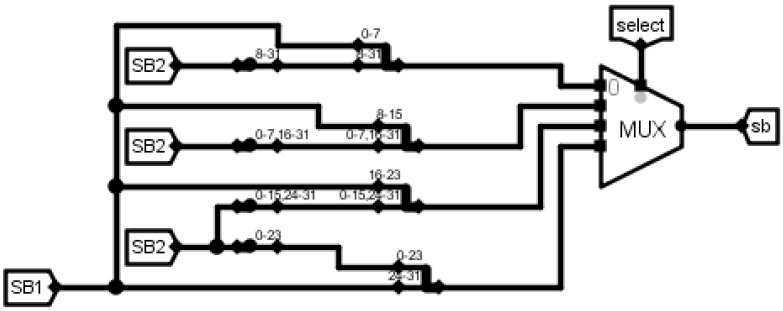

图 4.15 sb 指令实现

4.3.2 重定向暂停故障

故障现象：执行 ecall 指令不在 WB 暂停，即不在预期中的周期数暂停。

原因分析：在执行 ecall 指令时，识别的 ecall 指令是 ID/EX 阶段的，不是 WB 阶段的，而且在 ecall 指令生效时，ID/EX、EX/MEM、MEM/WB 流水线接口部件并未暂停，仍然往后传递相应的信息。

解决方案：将 ecall 指令生效的时间放在 WB 阶段，且在 ID/EX、EX/MEM、MEM/WB 流水线接口部件的使能端添加 halt 即可。

4.3.3 重定向旁路故障

故障现象：当判断发生数据冲突的电路输出 FwdRs1 和 FwdRs2 为 11 时发生重定向错误。

原因分析：FwdRs1 和 FwdRs2 为 11 时表示数据冲突与 EX 和 MEM 段均相关，而在 ALU 前的多路选择器输入端没考虑到 FwdRs1 和 FwdRs2 为 11 的情况。

解决方案：在 ALU 前的多路选择器输入端加入 FwdRs1 和 FwdRs2 为 11 时的输入，和 FwdRs1 和 FwdRs2 为 01 时的输入一样，均是 ALU 的运算结果（在 MEM/WB 段）。如图 4.16 所示。

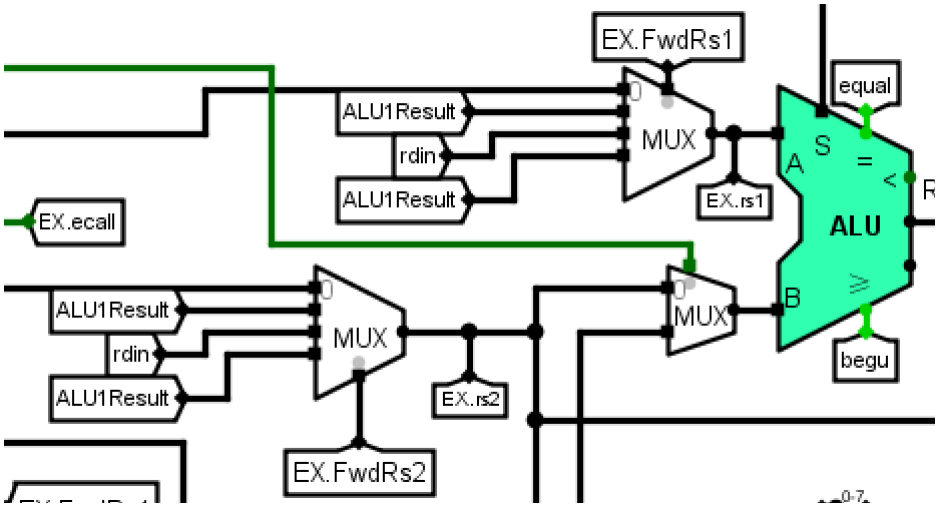

图 4.16 重定向旁路解决

4.3.4 重定向旁路故障

故障现象： 对于含立即数的指令发生数据冲突后重定向出问题。

原因分析： 在含立即数的指令发生数据冲突后，ALU 的 B 端本应输入立即数的时候却输入了重定向后的数据。

解决方案：在选择 ALU 的 B 端应该输入立即数还是寄存器中的数据前完成数据的重定向即可如图 4.17 所示。

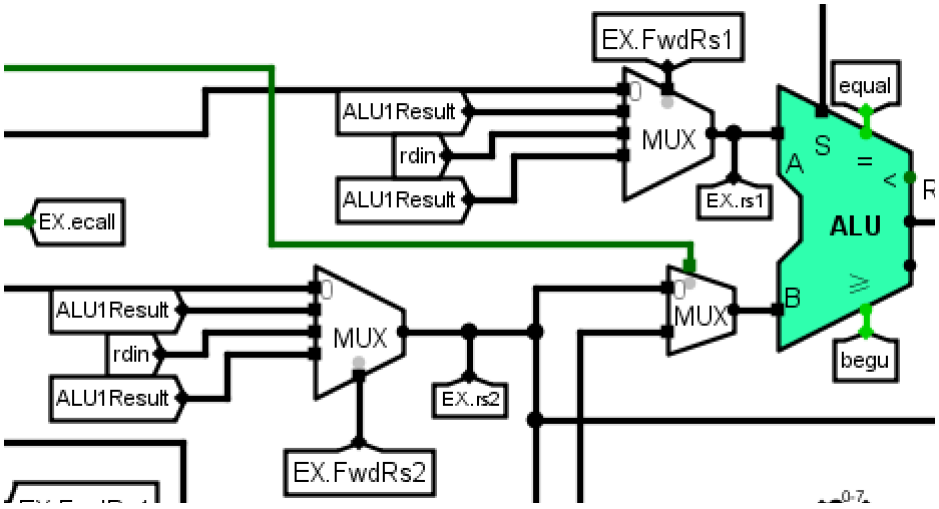

图 4.16 重定向旁路解决

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 Riscv 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	完成 Logism 单周期 CPU 理想流水线，并通过测试
第四天	学习数据相关，分支相关，气泡的知识，并在 logisim 上搭建部分单周期 CPU 气泡流水线并通过测试
第五天	完成单周期 CPU 气泡流水线，学习重定向相关知识并开始搭建框架
第六天	在 Logisim 上完善单周期 CPU 重定向流水线框架
第七天	调试重定向流水线，通过 Educoder 测试
第八天	学习中断相关知识。
第九天	完成单周期 CPU 单级中断
第十天	完成单周期 CPU 多级中断

5 设计总结与心得

5.1 课设总结

计算机组成原理课设完成的任务是对 RISCV CPU 进行由简到繁的设计。主要作了如下几点工作：：

1） 设计了单周期 RISCV CPU，实现了 21+4 条指令的运转。

2） 设计了单周期 RISCV CPU 的理想流水线，实现了流水线接口部件以及流水线的停机分割。

3） 设计了单周期 RISCV CPU 的气泡流水线，实现了判别发生数据相关的电路，实现了插入气泡的方式来解决分支相关和数据相关。

4） 设计了单周期 RISCV CPU 的重定向流水线，实现了判断发生数据相关的电路以及实现了重定向旁路的方式来解决数据相关。

5） 设计了单周期 RISCV CPU 的单级中断，实现了单级中断。

6） 设计了单周期 RISCV CPU 的多级中断，实现了嵌套中断。

5.2 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力、才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，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。

本次课程设计我是从单周期 RISCV CPU—理想流水线—气泡流水线—重定向流水线—单级中断—多级中断的顺序来做的。万事开头难，对于单周期 RISCV CPU 来说一开始由于数据通路的复杂导致我做起来感觉有点吃力，后来理清了数据通路以后做起来就比较顺手，但还是遇到了困难比如 sb 指令的时候，对于 sb 指令的理解不深入一开始竟然不知道 ALU 的结果中的 0-1 位是作为判别写入的位置的。这让我知道了对于 RISCV 指令集中的一些指令我还只是知其然不知其所以然。后面理想流水线除了比较繁琐不上特别难，这锻炼了我的耐心和仔细。对于气泡流水线就是在理想流水线基础上增加判断分支相关和数据相关的电路以及插入气泡的电路，这部分 mooc

和课本讲的都比较详细，仔细学习后完成起来比较一帆风顺。对于重定向流水线我耗时的时间最多，因为在设计前没有仔细考虑到 ecall 指令以及 ALU 输入端多路选择器重定向数据的具体来源，加上 benchmark-ccab 指令集完成测试的周期数比较多，所以每次调试耗时比较久。这让我明白了在做实验前要考虑清楚，同时这也让我掌握了调试大项目的方法，也加深了对 ecall 指令和重定向技术的理解。

本次课程设计我认为很好地锻炼了个人的动手实验能力，但由于疫情原因我成为密切接触者被隔离在酒店 14 天，这 14 天里克服困难完成了整个课程设计，更好地锻炼了我的自学能力和个人查错调试的能力。但也因为隔离无法完成上板实验比较遗憾。对于计算机组成原理的课程设计我认为可以对后面比较困难的技术实现发一些资料（比如动态分支预测）。

最后在这里也感谢各位老师对于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.

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：向隆航