華中科技大學

2022

硬件综合训练　　课程设计报告

题　　目：	5 段流水 CPU 设计
专　　业：	计算机科学与技术
班　　级：	CS2004
学　　号：	U202015409
姓　　名：	李学森
电　　话：	18998023220
邮　　件：	2575067251@qq.com

目 录

1 课程设计概述

1.1 课设目的

本课程力图以“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”。硬件综合训练课程是完成该计算机组成原理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 RISC-V 指令集（指令集任选），具体见表 1.1；

（2） 在 CCAB 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，指令编号详见公文包中的 任务分配；

（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	指令类型	简单功能描述	备注
1	ADD	R	加法	指令格式与功能 请参考 RISC-V32 指令集英文手 册，或参考 RARS 模拟器
2	ADDI	I	立即数加	
3	AND	R	与	
4	ANDI	I	立即数与	
5	SLLI	I	逻辑左移	
6	SRAI	I	算数右移	
7	SRLI	I	逻辑右移	
8	SUB	R	减	
9	OR	R	或	
10	ORI	I	立即数或	
11	XORI	I	或非/立即数异或	

#	指令助记符	指令类型	简单功能描述	备注
12	LW	I	加载字	
13	SW	S	存字	
14	BEQ	B	相等跳转	
15	BNE	B	不相等跳转	
16	SLT	R	小于置数	
17	SLTI	I	小于立即数置数	
18	SLTU	R	小于无符号数置数	
19	JAL	J	转移并链接	
20	JALR	I	转移到指定寄存器	
21	ECALL	I	系统调用	if ($a7==34) LED 输出$a0 的值 else 等待 Go 按键暂停 4 注意显示逻辑需要考虑如何锁存过去的数据，否则数据一闪而过。
22	CSRRSI	I	访问 CSR 寄存器	中断相关，可简化为开中断
23	CSRRCI	I	访问 CSR 寄存器	中断相关，可简化为关中断
24	URET	I	中断返回	清中断，mEPC 送 PC，开中断
25	SRL	R	逻辑右移	
26	SLTIU	I	无符号小于立即数置数	
27	SH	S	存半字	
28	BGE	B	大于等于跳转	

2 总体方案设计

2.1 单周期 CPU 设计

在单周期 CPU 通路设计中，我们采用硬布线控制器控制指令运行。在该单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，任何需要被多次使用的资源(如加法器等)都需要设置多个，否则就会发生资源冲突，而取指令和执行指令阶段均需要使用存储器，所以在该单周期 CPU 的设计中采用了指令存储器和数据存储器相分离的结构（哈佛结构）。

该单周期 CPU 的设计工作均在 Logisim 平台完成，其中的重要框架（如指令存储器，数据存储器等）沿用课设资料包中提供的框架。

总体结构图如图 2.1 所示。

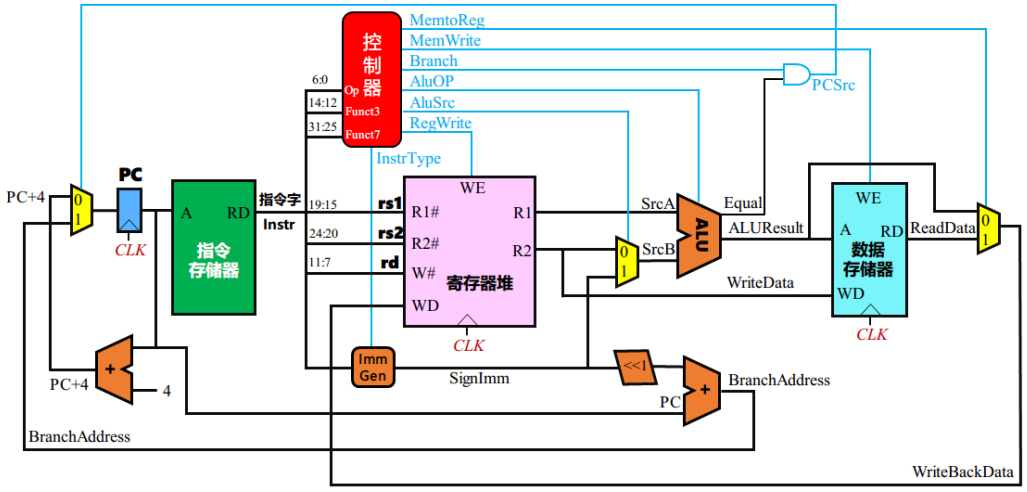


图 2.1 总体结构图

2.1.1 主要功能部件

运算器部分，具体设计思路如下：

1. 程序计数器 PC

程序计数器 PC 设计思想是分支式的。PC 中存储着当前指令的下一指令所处的地址。因为该单周期 CPU 为 32 位 CPU，所以每一条指令都为 32 位，占据 4 个字节，

若不出现分支跳转的情况，下一指令的地址应该为 PC+4，否则应当按照跳转信号选择运算器计算出的跳转地址。此外 PC 寄存器还需要确保在停机时，无视时钟输入，输出不发生改变。

2. 指令存储器 IM

指令存储器 IM 的设计思想是存取式，在 Logisim 平台中，程序是通过 16 进制数据镜像加载到存储器 IM 中，每当获取一个 PC 值时，IM 以 PC 的 2~11 位作为地址，输出该地址处连续的 4 个字节数据作为指令。

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
>=	输出	1	'>=' =(x>=y)?1:0, 对所有操作有效
<	输出	1	'<' =(x<y)?1:0, 对所有操作有效
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

表 2.2 寄存器堆 RF 引脚与功能描述

引脚	输入/输出	位宽	功能
R1#	输入	5	读取寄存器 1 的编号
R2#	输入	5	读取寄存器 2 的编号
W#	输入	5	写入目的寄存器的编号
Din	输入	32	写入目的寄存器的数据

引脚	输入/输出	位宽	功能
WE	输入	1	写入使能信号
CLK	输入	1	时钟信号
R1	输出	32	读取寄存器 1 中的数据
R2	输出	32	读取寄存器 2 中的数据

2.1.2 数据通路的设计

表 2.3 指令系统数据通路框架

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
R 型	PC+4	/	rs1	rs2	rd	ALUresult	R1	R2	Aluop	/	/
I 型	PC+4	imm[11:0]	rs1	/	rd	ALUresult	R1	imm12	Aluop	/	/
B 型	PC+2*imm[12:1]	imm[12|10:5]	rs1	rs2	/	/	R1	R2	Aluop	/	/
jal	PC+imm[21:30, 12:19]	imm[21:30, 12:19]	/	/	rd	PC+4	/	/	/	/	/
jalr	[rs1]+imm12	imm[11:0]	rs1	/	rd	PC+4	R1	imm12	Aluop	/	/
lw	PC+4	imm[11:0]	rs1	/	/	/	R1	imm12	Aluop	ALUresult	/
sw	PC+4	imm[11:0]	rs1	rs2	/	/	R1	imm12	Aluop	ALUresult	R2
ecall	PC+4	a0	a7	/	/	/	/	/	/	/	/
sltiu	PC+4	imm[11:0]	rs1	/	rd	1/0	R1	imm12	Aluop	/	/
sh	PC+4	imm[11:0]	rs1	Rs2	/	/	/	/	/	[rs1]+ imm12	rs2 [15:0]

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 2.4 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	运算器操作控制数
MemToReg	0/1	寄存器写入数据
MemWrite	0/1	内存写入控制信号
Alu_Src	0/1	运算器 B 选择输入信号
RegWrite	0/1	寄存器写入使能
ecall	0/1	ecall 指令译码信号
S_Type	0/1	S 型指令译码信号
BEQ	0/1	beq 指令译码信号
BNE	0/1	bne 指令译码信号
Jal	0/1	jal 指令译码信号
jalr	0/1	jalr 指令译码信号
SRL	0/1	srl 指令译码信号（可选）
SLTIU	0/1	sltiu 指令译码信号
SH	0/1	Sh 指令译码信号
BGE	0/1	bge 指令译码信号
rs1_used	0/1	寄存器 rs1 使用信号
rs2_used	0/1	寄存器 rs2 使用信号
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.所示。

表 2.5 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	SRL	SLTIU	SH	BGE	rs1_used	rs2_used	CSRRSI	CSRRCI
1	add	0	0	c	5				1											1	1		
2	sub	32	0	c	6				1											1	1		
3	and	0	7	c	7				1											1	1		
4	or	0	6	c	8				1											1	1		
5	slt	0	2	c	11				1											1	1		
6	sltu	0	3	c	12				1											1	1		
7	addi		0	4	5			1	1											1			
8	andi		7	4	7			1	1											1			
9	ori		6	4	8			1	1											1			
10	xori		4	4	9			1	1											1			
11	slti		2	4	11			1	1											1			
12	slli	0	1	4	0			1	1											1			
13	srli	0	5	4	2			1	1											1			
14	srai	32	5	4	1			1	1											1			
15	lw		2	0	5	1		1	1											1			
16	sw		2	8	5		1	1			1									1	1		
17	ecall	0	0	1c						1													
18	beq		0	18	6							1								1	1		
19	bne		1	18	6								1							1	1		
20	jal			1b					1					1									
21	jalr		0	19	5			1	1						1					1			
22	CSRRSI		6	1c																1		1	
23	CSRRCI		7	1c																1			1
24	URET	2	0	1c																			
25	SRL	0	5	c	2				1							1				1	1		
26	SLTIU		3	4	12			1	1								1			1			
27	SH		1	8			1	1			1							1		1	1		
28	BGE		5	18	11														1	1	1		

2.2 中断机制设计

2.2.1 总体设计

在本次实验中，我分别完成了单级中断和多级中断两种中断的设计。两者都是在单周期 CPU 上改良而来，且最多可以相应 3 种中断。每一个中断对应一个中断号，CPU 根据中断号选择对应的中断响应程序（具体的中断服务程序入口地址需要通过汇编工具查看），在跳转到对应中断相应程序前保存对应指令地址，以便在执行完中断响应程序后恢复现场。

在实现单级中断时，首先按照中断信号采样参考电路对中断信号进行采样，根据中断信号选择对应的中断响应程序，并将其地址作为下一条指令的地址，同时将当前指令保存到 mEPC 寄存器。然后关中断，进入中断响应程序。执行中断响应程序后根据 URET 指令清除中断产生的信号，开中断，同时返回中断之前指令的地址。

在实现多级嵌套中断时，需要根据中断的优先级判断，能否在当前中断的执行过程当中打断中断，并进入新的中断响应程序，同时利用堆栈保存多个中断，多个 mEPC 保存不同层中断的返回地址。

为了实现多级中断，需要添加 3 个中断屏蔽寄存器动态地改变中断的优先级；为了实现开关中断的功能呢，需要添加中断使能寄存器 IE；为了实现 CSR 寄存器访问指令 CSRRCI 和 CSRRSI，以及中断返回指令 URET，需要对硬布线控制器以及数据通路进行修改。

2.2.2 硬件设计

三个中断信号采样电路仿照资料包中给出的参考电路进行设计，在中断响应的过程中，将中断请求锁存，当执行 URET 指令时，退出当前中断，最后清除中断信号。

在设计多级中断时，需要判断当前执行的中断与现在触发的中断的优先级，还要判断几个同时产生的中断的优先级，选择高优先级中断优先执行。需要用优先编码器实现高优先级中断先行执行，还要利用堆栈保存不同的中断，以保证执行高优先级中断后能够执行下一优先级的中断响应程序。

在中断响应程序执行之前需要保存现场，将当前指令地址（即现场）输入 mEPC 中保存，中断响应程序执行完后，从 mEPC 中获取指令地址，恢复现场，继续执行原来的指令。

添加中断使能寄存器 IE 来保存开关中断的状态。

修改硬布线控制器以及数据通路使其能够适应 CSR 寄存器访问指令 CSRRCI 和 CSRRSI，以及中断返回指令 URET。

2.2.3 软件设计

单级中断与单周期 CPU 相比添加了返回信号 URET 用于退出中断响应程序，多级嵌套中断与单级中断相比多了 CSR 寄存器访问指令 CSRRCI 和 CSRRSI 用于开关中断。因此需要修改真值表，增添以上三个控制信号，获得对应的逻辑表达式，并更新硬布线控制器。

2.3 流水 CPU 设计

2.3.1 总体设计

在单周期流水线 CPU 中，RISC-V 指令的执行过程被分为 5 个阶段，即取指令阶段 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。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，流水线应向后续段传递数据信息，控制信息，向前段传递反馈信息，后续部件对数据的加工处理依赖于前阶段传递过来的信息。即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，同时指令产生的各种控制信号能够跟随指令进入不同的阶段。IF 阶段根据 PC 获取指令寄存器中的指令；ID 阶段将指令按照对应格式分别由单周期

硬布线控制器生产控制信号，以及读取寄存器操作数等；EX 阶段利用 ALU 进行运算，计算分支跳转地址以及判断是否符合停机条件；MEM 阶段在数据存储器中按指令执行存取操作；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，并依据信号执行停机操作。

2.3.2 流水接口部件设计

流水接口部件用于五个指令流水阶段的连接，锁存前一阶段的信号并传输到下一阶段。当使能端处于高电平状态时，每一个时钟周期都会将前一个阶段的信号或者数据（如控制信号，PC，IR 等）从输入端中读入并缓存到寄存器里，同时输出端把寄存器中的信号或者数据发送给下一阶段。若清零端等处于高电平转态时，将寄存器中的数据进行同步清零。

2.3.3 理想流水线设计

理想流水线是在单周期 CPU 基础上，将每个阶段进行分割，并加入流水接口部件，使得数据通路转变成流水线的形式。在设计理想流水线时无须考虑数据冲突等问题，因此只需要保证理想流水线中信号在每一个阶段正确传输即可，不需要添加冲突处理部件。

2.4 气泡式流水线设计

气泡式流水线与理想流水线相比，需要考虑实际运行过程中出现的冲突问题。解决数据冲突的策略是插入气泡延缓取指令，保证不会出现访存冲突；当出现分支跳转指令时，会出现分支冲突，其解决策略是清空分支指令到达 EX 段时，IF 和 ID 段的指令，使得分支指令执行完后能过直接执行跳转之后的指令。气泡流水线处理数据冲突和执行分支指令的时空图如图 2.2 和 2.3 所示。气泡流水线顶层视图如图 2.4 所示。

clks	IF	ID	EX	MEM	WB
3	sub $1, $2, $3	add $6, $5, $7	lw $5, 4($1)		
4	sub $1, $2, $3	add $6, $5, $7		lw $5, 4($1)	
5	sub $1, $2, $3	add $6, $5, $7			lw $5, 4($1)
6	or $7, $6, $7	sub $1, $2, $3	add $6, $5, $7		
7	and $9, $7, $6	or $7, $6, $7	sub $1, $2, $3	add $6, $5, $7	
8	and $9, $7, $6	or $7, $6, $7		sub $1, $2, $3	add $6, $5, $7
9	Next Instr	and $9, $7, $6	or $7, $6, $7		sub $1, $2, $3
10	Next Instr	and $9, $7, $6		or $7, $6, $7	
11	Next Instr	and $9, $7, $6			or $7, $6, $7

图 2.2 插入气泡解绝数据冲突的流水时空图

clks	IF	ID	EX	MEM	WB
1	beq		EX段执行分支		
2	add	beq			
3	addi	add	beq		
4	bne			beq	
5	lw	bne			beq
6	sw	lw	bne		
7	New			bne	

图 2.3 EX 段执行分支指令解决分支冲突的流水时空图

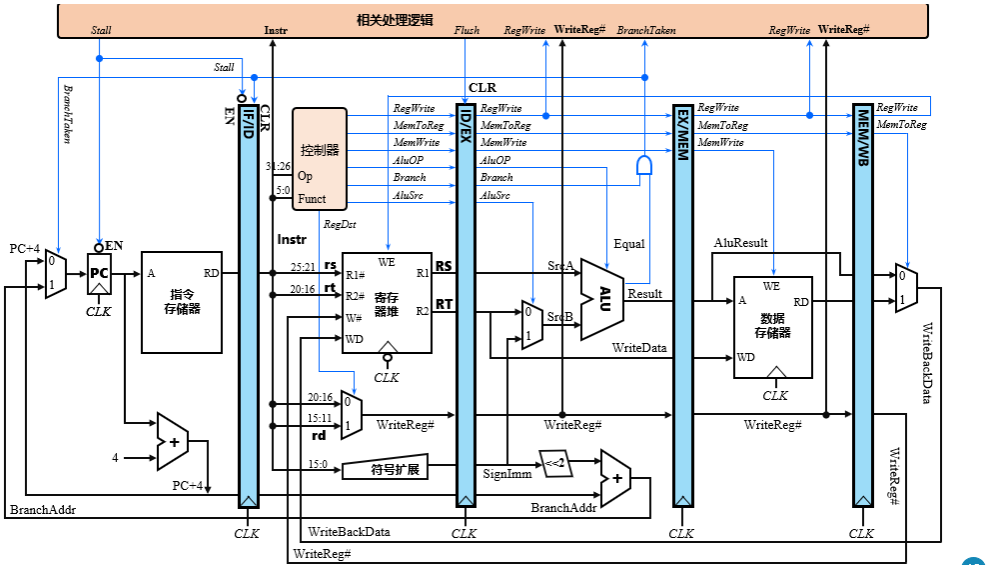


图 2.4 气泡流水线顶层视图

2.5 数据转发流水线设计

重定向（数据转发）流水线先不考虑 ID 段所取的寄存器操作数是否正确，等到指令实际使用这些寄存器操作数时再考虑其正确性的问题。其将 MEM 段的 ALU 运算结果和 WB 段的写回数据传入 EX 段，通过重定向处理逻辑判断前后指令是否存在数据相关，从而在 EX 段选择正确的操作数进入 ALU 运算。重定向流水线顶层视图如图 2.5 所示。但是当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 Load-Use 信号产生时，暂停 IF、ID 段的指令并向 EX 段中插入一个气泡以消除这

种相关性。

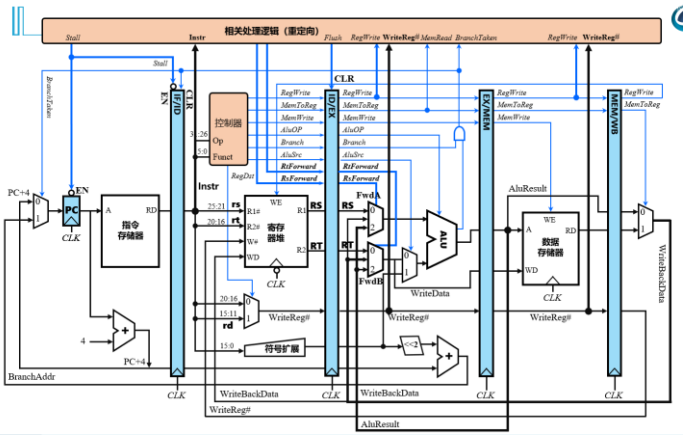


图 2.5 重定向流水线顶层视图

2.6 动态分支预测机制

采用重定向机制后，指令流水线中的控制冲突对流水线性能影响最大。基于加快经常性事件的原理，应尽可能提前执行分支指令以减少分支指令引起的分支延迟损失。我们采用了最简单的动态分支预测策略-分支预测缓冲器，其用于存放分支指令的分支跳转历史统计信息。每一条分支指令执行时都要将其输入到 BTB 表内，并修改表内数据，提高预测准确率。分支预测需要和指令存储器取指令操作并发进行，预测失败的时候仍然需要清空误取指令修正 PC 以取出正确的指令。根据 BTB 预测的结果选取预测地址是分支地址还是下一条指令的地址，然后和修订的地址进行多路选择，若是预测错误则选择修订地址，清除误取指令，否则按照预测地址进行取指。动态分支预测流水线的顶层视图如图 2.6 所示。

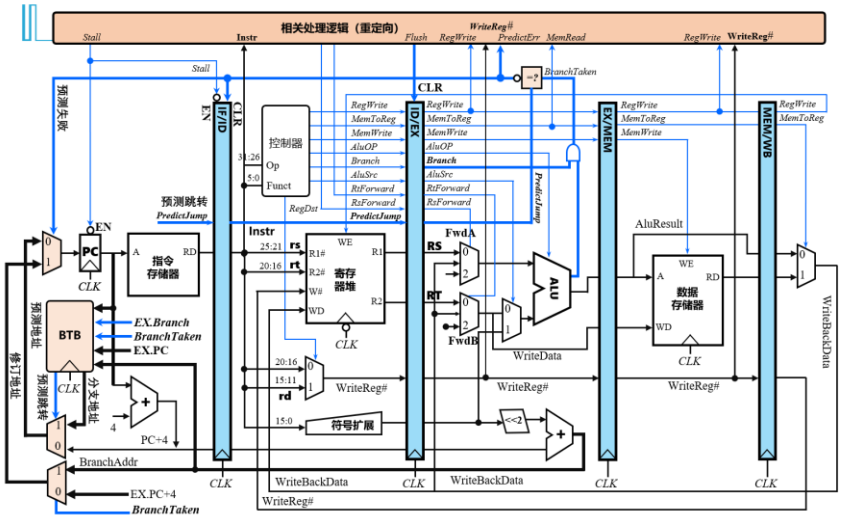


图 2.6 动态分支预测流水线顶层视图

采用 BTB 后流水线的运行状况如图 2.7 所示。

指令在 BTB 中?	预测情况	下条指令地址	实际情况	预测情况	流水停顿周期
命中	预测跳转	分支目标地址	跳转	预测成功	0
命中	预测跳转	分支目标地址	未跳转	预测失败	2
命中	预测不跳转	顺序地址	跳转	预测失败	2
命中	预测不跳转	顺序地址	未跳转	预测成功	0
缺失		顺序地址	跳转		2
缺失		顺序地址	未跳转		0

图 2.7 BTB 流水线运行状况

2.7 流水中断机制

流水线单级中断，由于不需要实现嵌套中断。因此是在重定向流水线的基础之上，结合单周期 CPU 单级中断中实现的。其数据通路的顶层视图，基本上与重定向流水线一模一样，唯一的区别在于添加了单级中断的中断处理机制。若中断时遇到气泡，则应该进行判断，选择气泡前指令的地址作为返回地址。

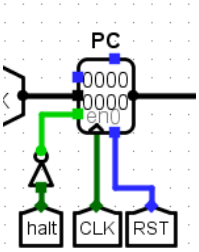
3 详细设计与实现

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**错误!未找到引用源。**所示。



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 3.1 所示。

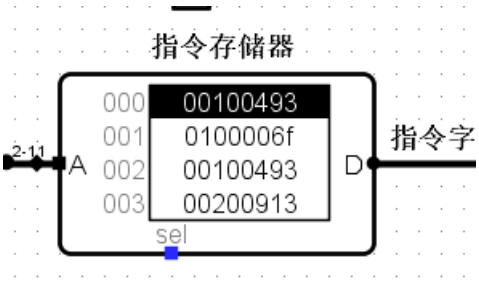


图 3.1 指令存储器（IM）

3）运算器（ALU）

根据 ALUop 控制信号对输入的数据 A 和 B 采用不同的运算方式，输出结果以及

标志位信号。其电路封装如图 3.3 所示。

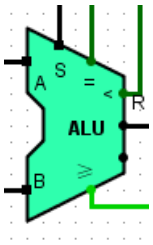


图 3.3 运算器（ALU）

4）寄存器堆（RF）

存储 CPU 内部的所有寄存器，其封装如图 3.4 所示。

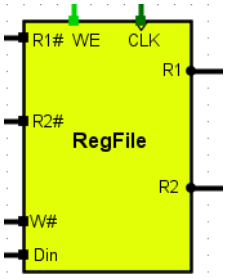


图 3.4 寄存器堆(RF)

5）数据存储器（DM）

数据存储器使用 CS3410 库中的 RAM 实现如图 3.5，地址位宽为 10 位（取 ALU 运算结果的第 2~11 位），数据位宽为 32 位。写入的数据根据 ALUresult 的第 1 位判断，若为 0 则取 R2，否则取 R2<<1。片选信号根据指令以及 ALUresult 的数据进行确定，如图 3.6 所示电路。

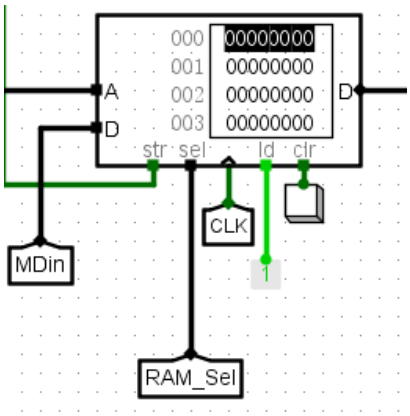


图 3.5 数据存储器（DM）

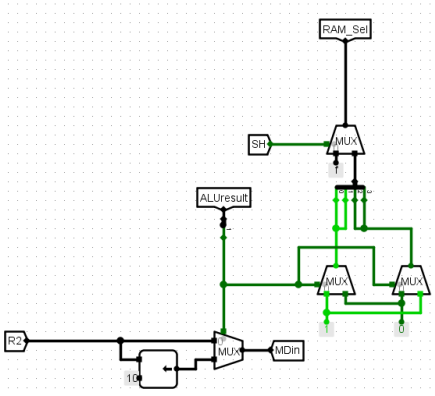


图 3.6 片选信号电路

3.1.2 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现方法为，对于每一条指令，将其改写成 RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据 RTL 功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 3.1 所示。

表 3.1 指令系统数据通路表

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ADD	PC+4	PC	rs	rt	rd	alu	r1	r2	5	alu	Mdin
ADDI	PC+4	PC	rs	rt	rd	alu	r1	立即数	5	alu	Mdin
AND	PC+4	PC	rs	rt	rd	alu	r1	r2	7	alu	Mdin
ANDI	PC+4	PC	rs	rt	rd	alu	r1	立即数	7	alu	Mdin
SLTI	PC+4	PC	rs	rt	rd	alu	r1	立即数	11	alu	Mdin
SRAI	PC+4	PC	rs	rt	rd	alu	r1	立即数	1	alu	Mdin
SRLI	PC+4	PC	rs	rt	rd	alu	r1	立即数	2	alu	Mdin
SUB	PC+4	PC	rs	rt	rd	alu	r1	r2	6	alu	Mdin
OR	PC+4	PC	rs	rt	rd	alu	r1	r2	8	alu	Mdin
ORI	PC+4	PC	rs	rt	rd	alu	r1	立即数	7	alu	Mdin
XORI	PC+4	PC	rs	rt	rd	alu	r1	立即数	9	alu	Mdin
LW	PC+4	PC	rs	rt	rd	Dout	r1	立即数	5	alu	Mdin
SW	PC+4	PC	rs	rt	rd	alu	r1	立即数	5	alu	Mdin
BEQ	PC+4	PC	rs	rt	rd		r1	r2		alu	Mdin
BNE	PC+4	PC	rs	rt	rd		r1	r2		alu	Mdin
JAL	PC+4	PC	rs	rt	rd	PC+4	r1	r2		alu	Mdin
JALR	PC+4	PC	rs	rt	rd	PC+4	r1	立即数	5	alu	Mdin

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ECALL	PC+4	PC	a7	a0	rd		r1	r2		alu	Mdin
CSRRSI	PC+4	PC	rs	rt	rd		r1	r2		alu	Mdin
CSRRCI	PC+4	PC	rs	rt	rd		r1	r2		alu	Mdin
URET	PC+4	PC	rs	rt	rd		r1	r2		alu	Mdin
SRL	PC+4	PC	rs	rt	rd	alu	r1	r2	2	alu	Mdin
SLTIU	PC+4	PC	rs	rt	rd	0/1	r1	立即数	12	alu	Mdin
SH	PC+4	PC	rs	rt	rd	alu	r1	立即数		alu	Mdin
BGE	PC+4	PC	rs	rt	rd		r1	r2	11	alu	Mdin

对于所有的 Mdin=(alu[1]==0)?r2:(r2<<1);

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，表，将各个主要功能部件进行连接，根据数据通路合并表的最终结果，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建。

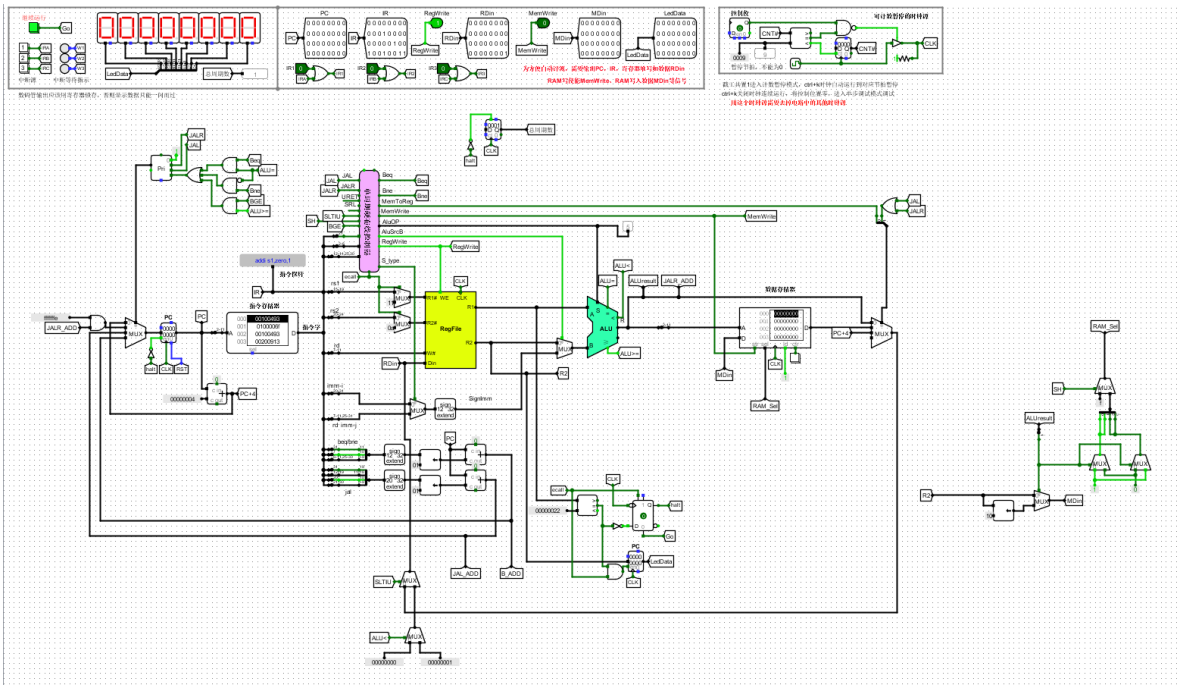


图 3.7 单周期 CPU 数据通路

3.1.3 控制器的实现

根据总体方案设计中控制器的设计一小节的相关内容，在 Logism 上完成主控制器具体实现。

对照表 3.2 所示。

表 3.2 主控制器控制信号

#	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	SRL	SLTIU	SH	BGE	rs1_used	rs2_used	CSRRSI	CSRRCI
1	add	0	0	c	5				1											1	1		
2	sub	32	0	c	6				1											1	1		
3	and	0	7	c	7				1											1	1		
4	or	0	6	c	8				1											1	1		
5	slt	0	2	c	11				1											1	1		
6	sltu	0	3	c	12				1											1	1		
7	addi		0	4	5			1	1											1			
8	andi		7	4	7			1	1											1			
9	ori		6	4	8			1	1											1			
10	xori		4	4	9			1	1											1			
11	slti		2	4	11			1	1											1			
12	slli	0	1	4	0			1	1											1			
13	srli	0	5	4	2			1	1											1			
14	srai	32	5	4	1			1	1											1			
15	lw		2	0	5	1		1	1											1			
16	sw		2	8	5		1	1			1									1	1		
17	ecall	0	0	1c						1													
18	beq		0	18	6							1								1	1		
19	bne		1	18	6								1							1	1		
20	jal			1b					1					1									
21	jalr		0	19	5			1	1						1					1			
22	CSRRSI		6	1c																1		1	
23	CSRRCI		7	1c																1			1
24	URET	2	0	1c																			
25	SRL	0	5	c	2				1							1				1	1		
26	SLTIU		3	4	12			1	1								1			1			
27	SH		1	8			1	1			1							1		1	1		
28	BGE		5	18	11														1	1	1		

根据主控制信号表，使用 Logisim 平台的分析组合逻辑电路功能，分别生成了运算控制器电路和控制信号电路，并将其组合成硬布线控制器，如图 3.8 所示，其中的三个输入信号分别是 IR21 用于区分 ecall 和 uret 指令，Funct 和 OP_CODE 用于确定运算控制器功能选择信号和其他输出控制信号。

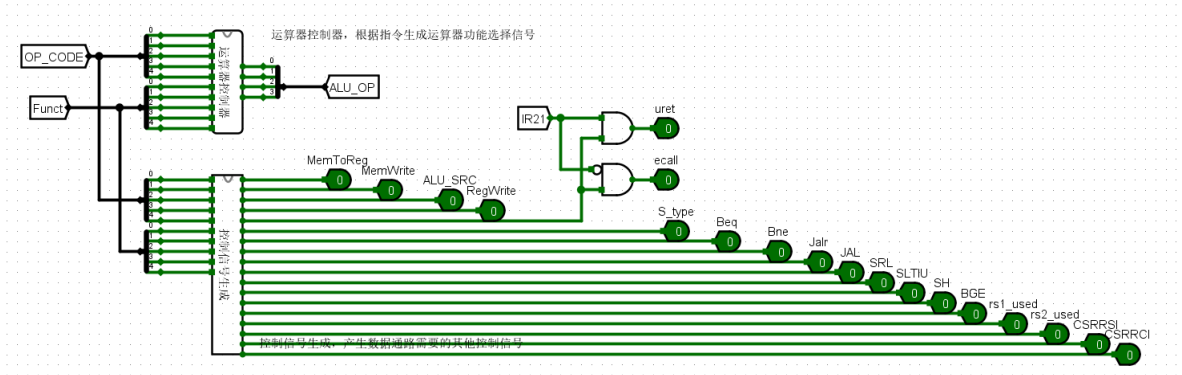


图 3.8 单周期硬布线控制器

3.2 中断机制实现

3.2.1 单周期单级中断

增加中断按键信号采样电路参考给出的模板电路，IR（1~3）就是中断请求寄存

器，输出与中断屏蔽位进行逻辑与后送中断优先编码器生成中断号与中断请求信号，同步清零信号用于清除中断请求信号，这个采样电路对三个中断源是否产生中断请求进行检测和锁定。电路如图 3.9 所示。

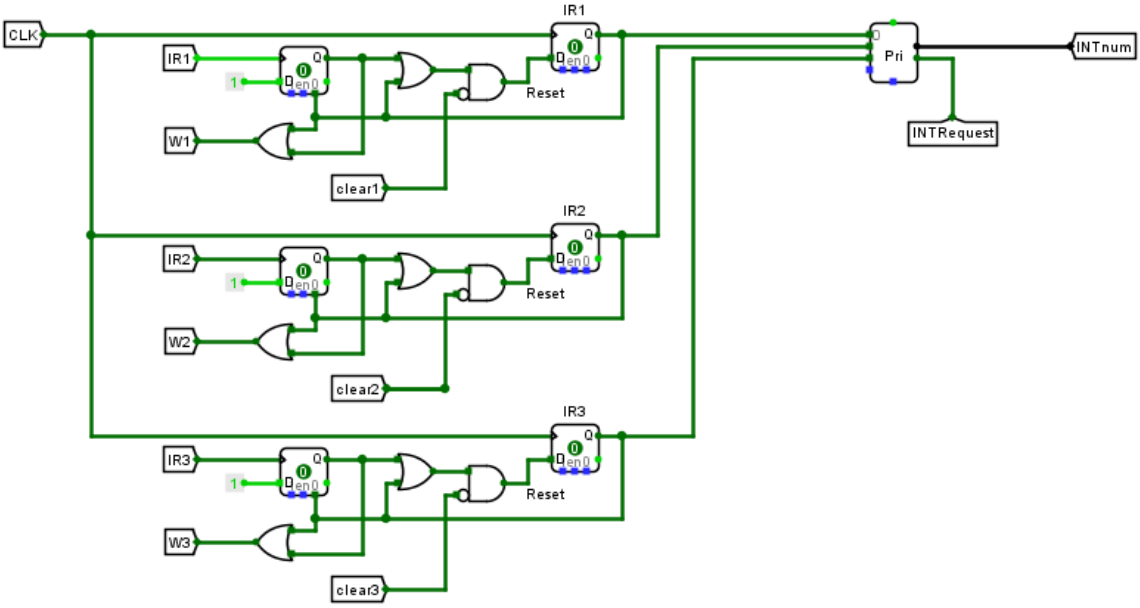


图 3.9 单级中断信号采样与优先级编码电路

由中断优先编码器获得当前应该处理的中断号后，将该中断号所对应的中断服务程序的首地址送入 INT_PC 中，如图 3.10 所示。

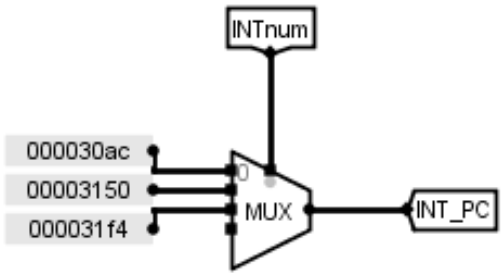


图 3.10 中断地址保存电路

通过中断请求与中断使能信号进行与运算，获得当前是否应当执行中断的 INT 信号，然后获取中断号后，将其锁存。当中断程序执行结束后通过 URET 指令实现中断请求信号的清除。中断信号获取、锁存以及清除电路如图 3.11 所示。

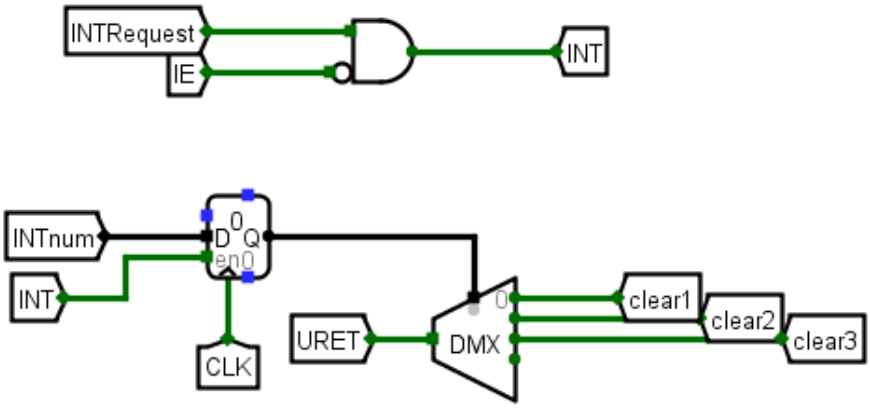


图 3.11 中断信号获取、锁存以及清除电路

中断结束后，现场的返回电路如图 3.12 所示。

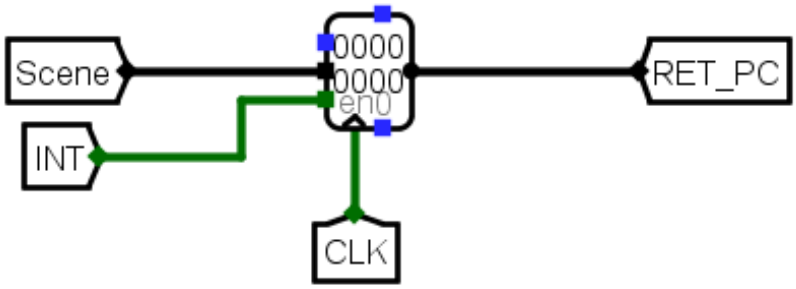


图 3.12 现场返回电路

正常情况下 IE 应该始终保持开中断状态，进入中断服务后应当处于关中断状态，中断服务程序执行完毕后又进入开中断状态。因此 IE 状态电路如图 3.13 所示。

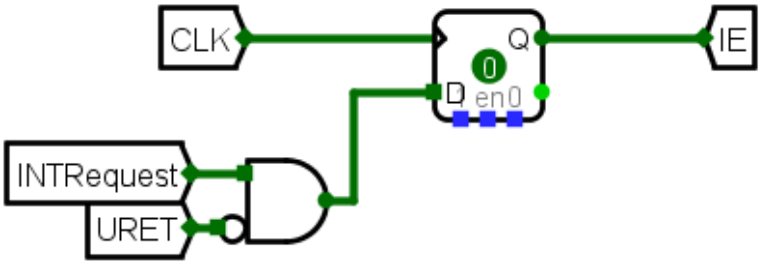


图 3.13 IE 状态电路

在单周期 CPU 数据通路上，为了能够执行中断，还应该添加两个多路选择器，用于保护现场进入中断，退出中断恢复现场。如图 3.14 所示。

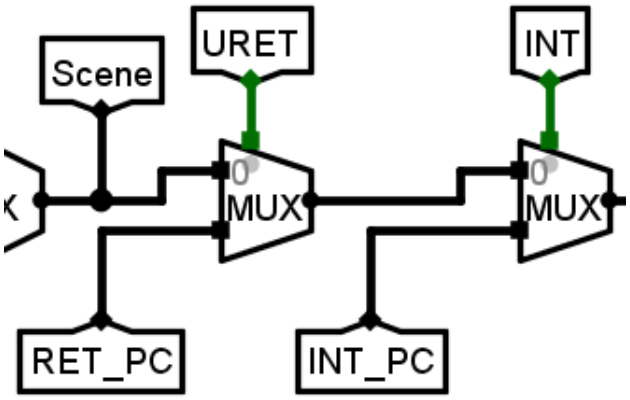


图 3.14 数据通路中断相关电路

单周期单级中断 CPU 的电路如图 3.15 所示。

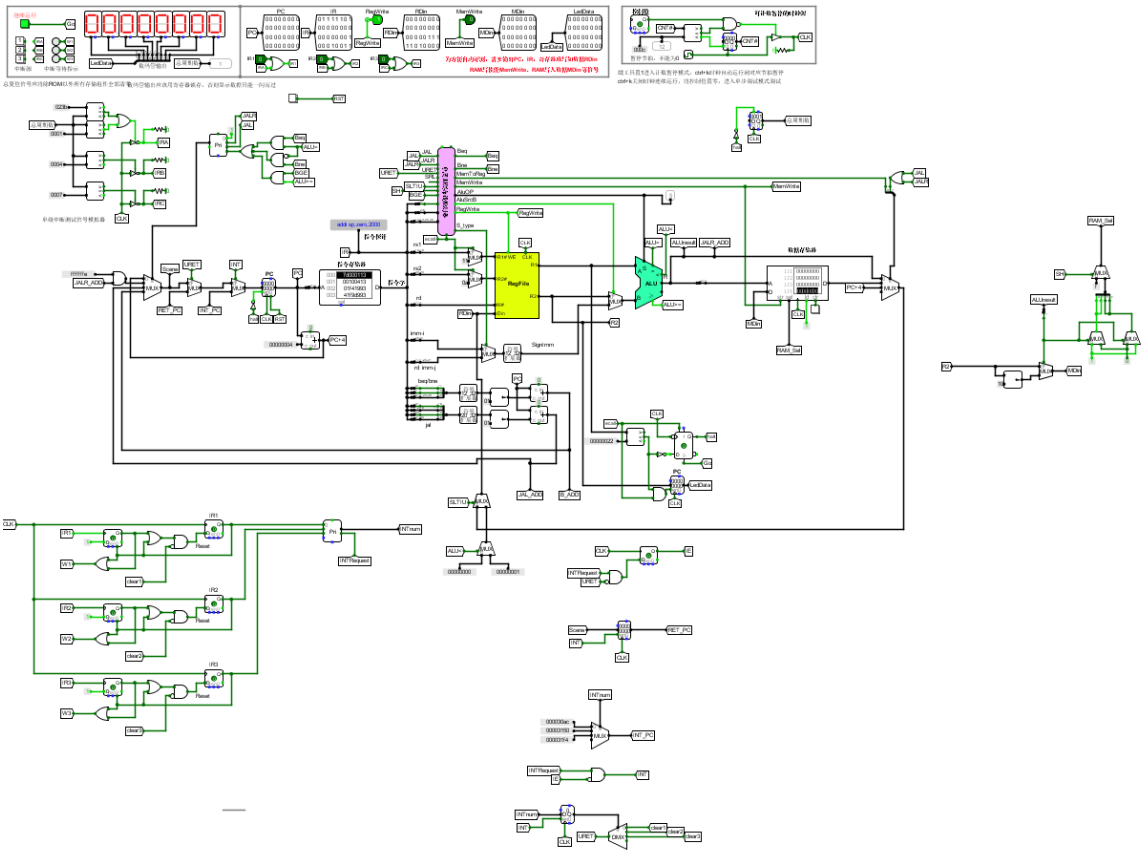


图 3.15 单周期单级中断电路

3.2.2 单周期多级中断

中断信号采样电路与单级中断相同，中断请求寄存器的输出与中断屏蔽位进行逻

辑与后送中断优先编码器（位置与单级中断不同）生成中断号与中断请求信号。如图 3.16 所示。

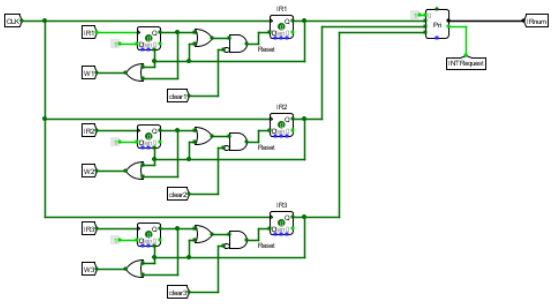


图 3.16 多级中断信号采样与优先级编码电路

由中断优先编码器获得当前应该处理的中断号后，将该中断号所对应的中断服务程序的首地址或者现场地址送入 INT_PC 中。获得中断请求后，与中断开关信号 IE 以及中断插入信号 in 做与运算获得中断信号 INT。如图 3.17 所示。

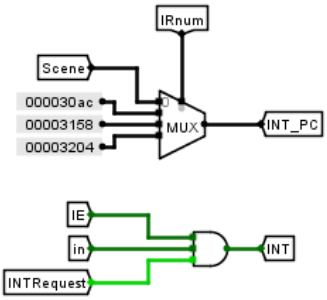


图 3.17 多级中断中断地址和信号产生电路

多级中断中可能会出现中断嵌套的情况，需要根据中断优先级进行判断，并通过堆栈来存储优先级排序后的中断号，因为只有三个不同的中断信号，所以在这里使用三个寄存器来实现硬件堆栈。如图 3.18 所示。

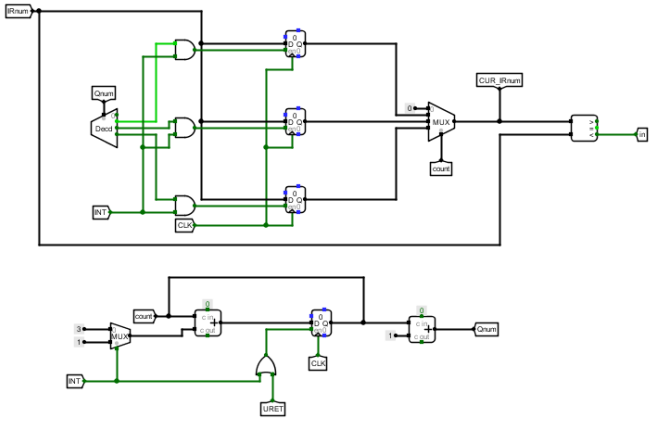


图 3.18 多级中断硬件堆栈电路

与单级中断相比，因为多级中断添加了对 CSRRSI 和 CSRRCI 两个指令的支持，故开关中断信号 IE 的产生电路也需要进行修改。如图 3.19 所示。

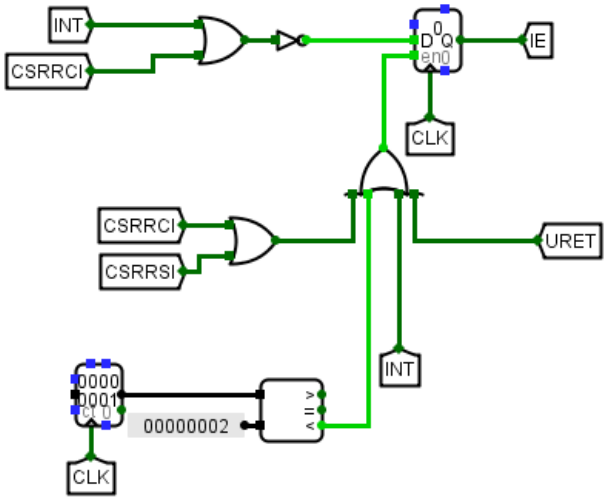


图 3.19 多级中断开关信号 IE 产生电路

中断服务程序结束之后，由于可能会出现三个中断（即三个返回地址），故需要用三个 mEPC 寄存器存储不同中断的返回地址。每次中断结束之后同样要对中断信号进行清除。如图 3.20 所示。

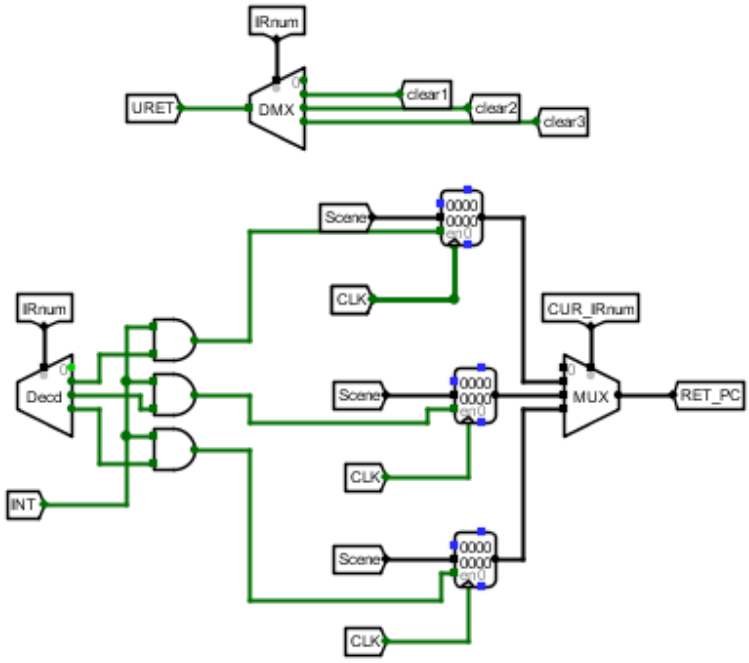


图 3.20 中断信号清除、多级中断返回地址保存电路

单周期多级中断整体电路如图 3.21 所示。

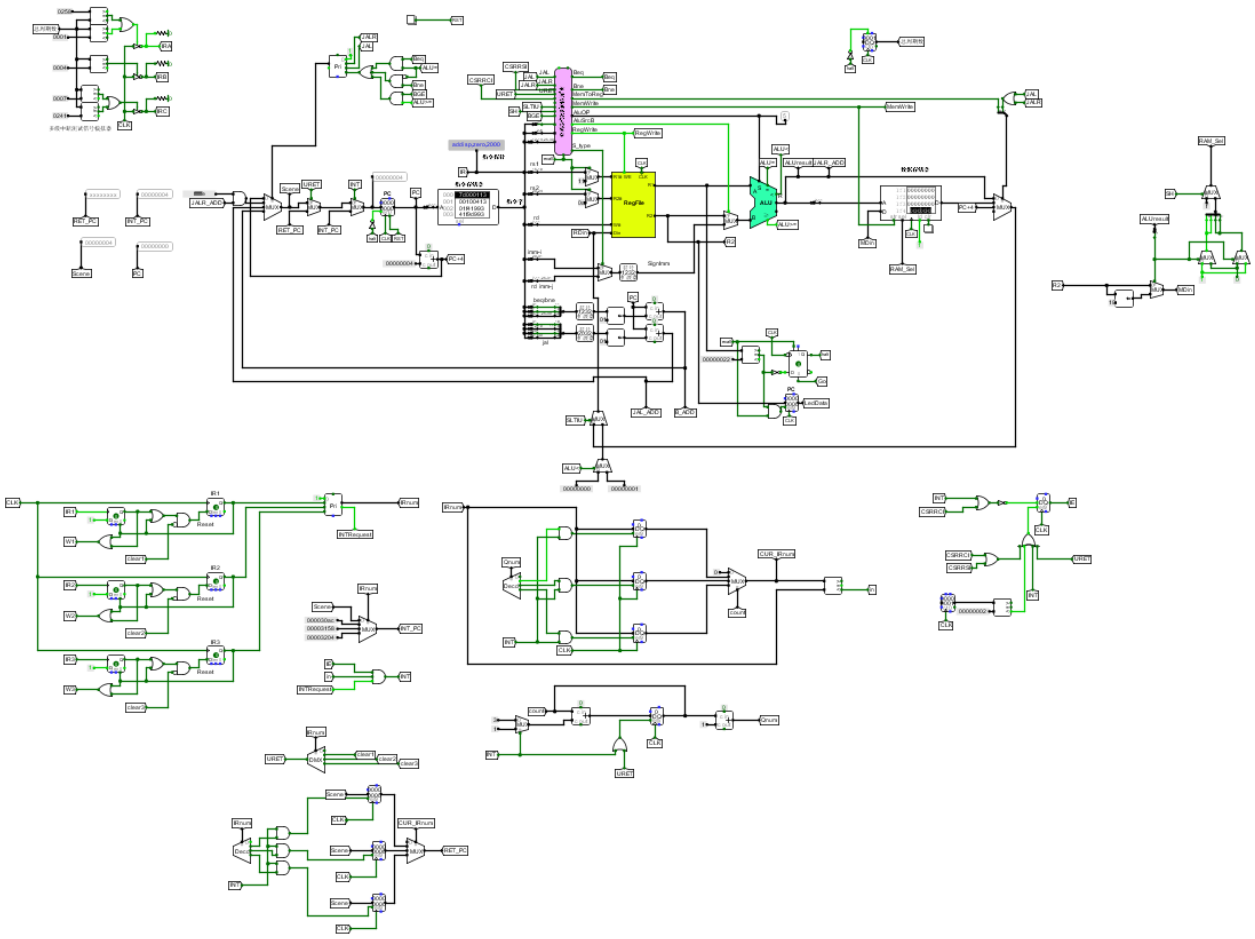


图 3. 21 单周期多级中断电路

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口部件是流水 CPU 中将数据通路上相邻的操作段连接起来的部件。其内部采用寄存器对从上一个阶段获得的信号进行锁存。所有的寄存器采用上升沿触发，这样就能保存上一个段的数据，供下一个流水段使用达到信息流水传递，使不同段执行不同的指令。四个流水部件的内部结构如图 3.22，3.23，3.24，3.25 所示。

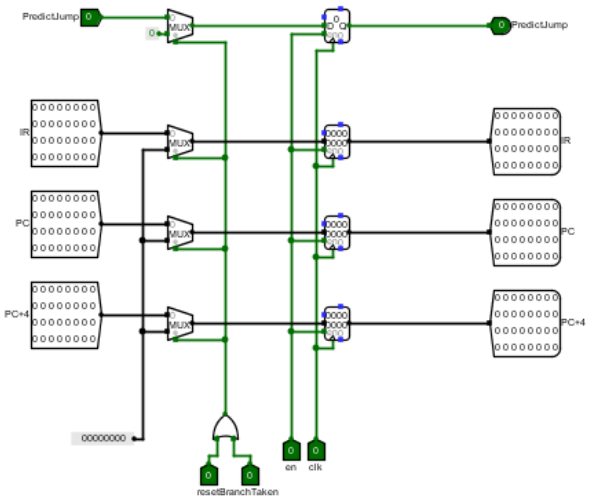


图 3.22 IF/ID 接口部件

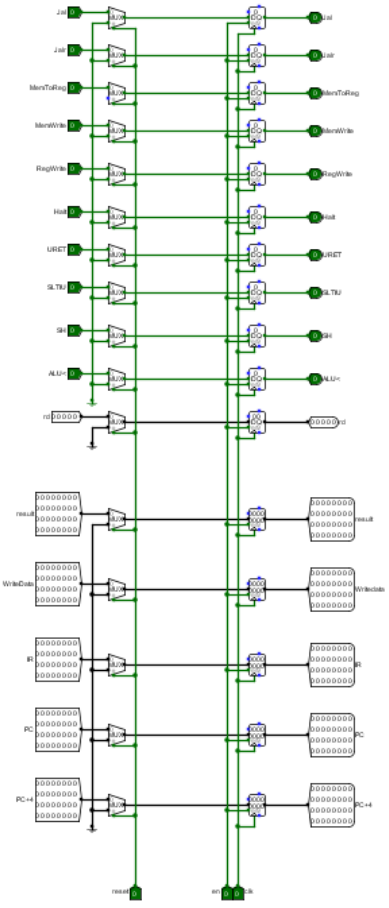


图 3.23 EX/MEM 接口部件

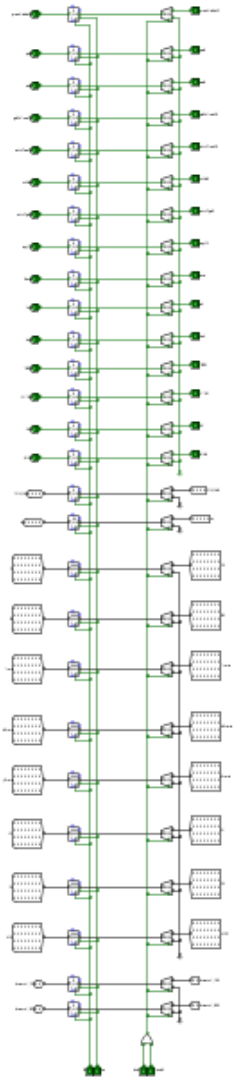


图 3.24 ID/EX 接口部件

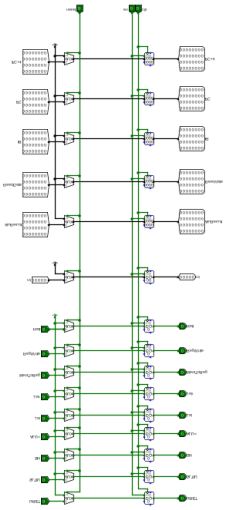


图 3.25 MEM/WB 接口部件

其中改变使能端 en 的信号以忽略时钟输入来实现暂停流水线操作，改变 reset 的信号可以实现器件内部锁存信号清零的功能，而 BranchTaken 和 Flush 信号是为后续插入气泡等操作预留接口。

3.3.2 理想流水线实现

理想流水线的实现无需考虑分支跳转的结构冲突问题和相邻指令之间的数据冲突问题。实现理想流水线的时候只需要保证下一阶段需要的信息能够准确从上一个阶段传递下来即可。因此在实现理想流水线的时候只需要在单周期 CPU 的基础上，将其准确的划分为五个操作段，段与段之间通过流水接口部件进行连接，并且将部件使用的信号替换成当前操作段的信号即可，理想流水线 CPU 如图 3.26 所示。

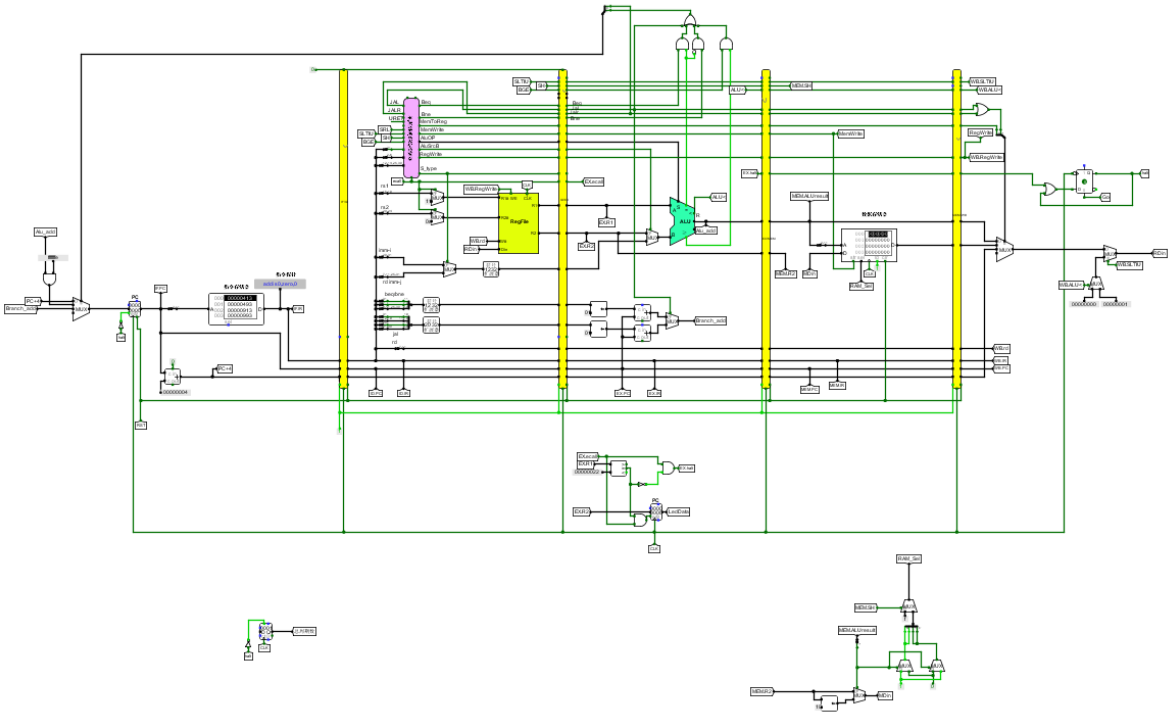


图 3.26 理想流水线 CPU

与单周期 CPU 区别的地方是 ecall 指令的执行问题。当 ecall 指令传递到 EX 段的时候，若满足条件应该要使 CPU 停机，后续指令暂停执行，但是 MEM 与 WB 段的指令需要继续执行，因此停机指令 halt 会通过数据通路继续传递到 WB 段，然后再生成 PC 计数器上 halt 指令，以达到停机的目的。

3.4 气泡式流水线实现

气泡流水线是在理想流水线的基础之上，通过在 ID 段插入气泡延迟数据进入 EX 段解决数据冲突问题，在 EX 段出现分支指令时清空 IF/ID 段的指令以解决结构冲突问题。检测数据冲突时，我们通过在真值表中添加控制信号 rs1_used 和 rs2_used 来判断指令是否使用寄存器，之后再检查 EX 和 MEM 段的寄存器堆写入控制信号 RegWrite 是否为 1，且使用的编号 rd 是否与源寄存器编号相同，若相同则说明出现数据冲突。数据冲突检测电路如图 3.27 所示。检测出数据冲突后，要在 ID/EX 段同步清零插入气泡，同时暂停 PC 计数器与 IF/ID 段的流水部件的信号传递。

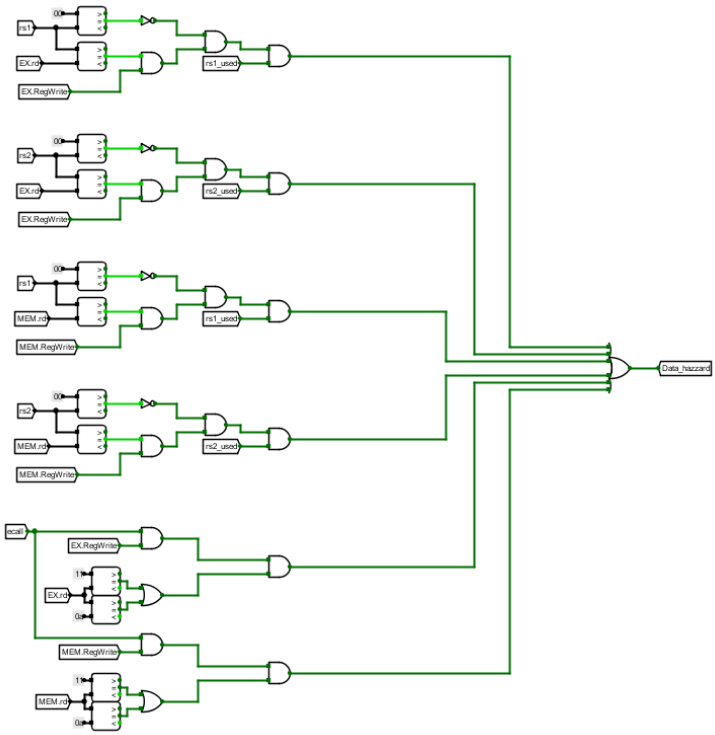


图 3.27 数据冲突检测电路

在处理结构冲突时，若在 EX 段出现分支跳转指令，当条件分支指令满足跳转条件或者执行无条件跳转（BranchTaken）的时候要在 IF/ID 和 ID/EX 段插入气泡。气泡流水判断插入气泡机制如图 3.28 所示。

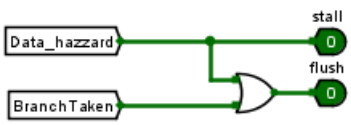


图 3.28 气泡流水线判断插入气泡机制

气泡流水线 CPU 如图 3.29 所示。

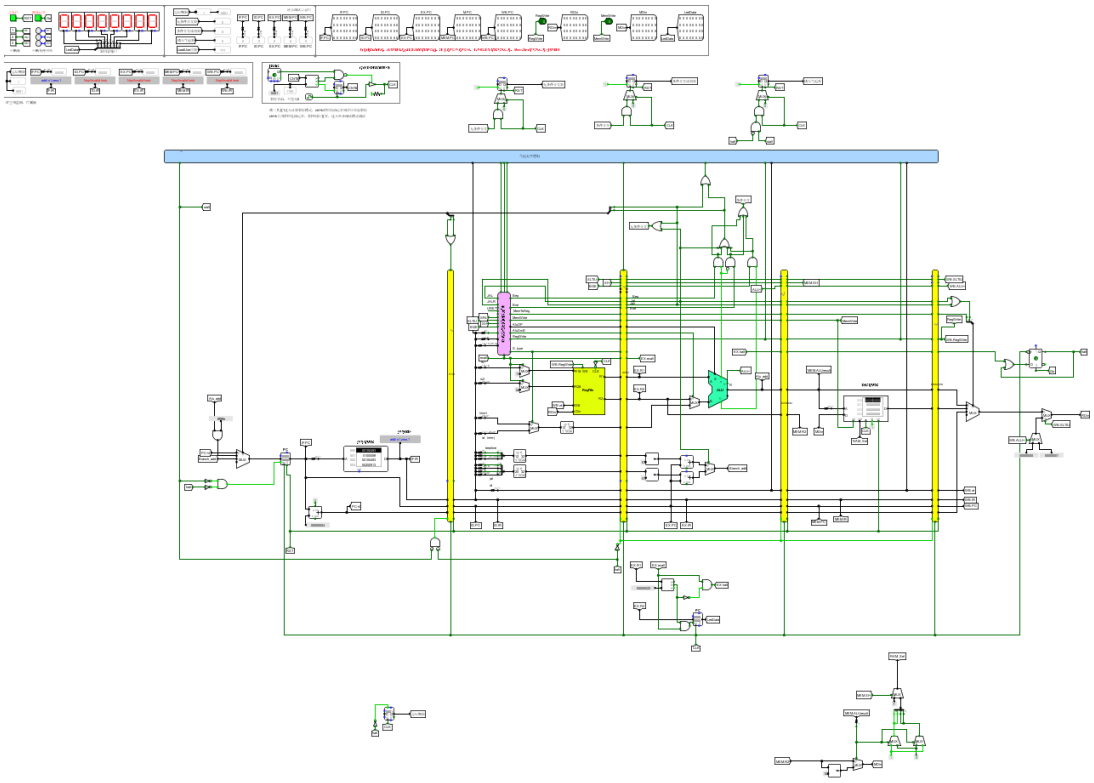


图 3. 29 气泡流水线

3.5 数据转发流水线实现

重定向流水线对于 Load-Use 类型的指令仍然使用插入气泡的方式消除数据冲突，通过检测 MemToReg 信号以及指令的 rs1 与 rs2 是否和 EX 段的 rd 相同来判断是否发生 Load-Use 型冲突，若是，则在 ID/EX 段插入气泡。对于其他类型的数据冲突，则分别检测 EX 段和 MEM 段寄存器是否和 ID 段发生冲突，生成多路选择信号用于 ALU 输入端的选择，当 0 表示没发生数据冲突，1 表示 ID 和 EX 段冲突，2 表示 ID 和 MEM 段数据冲突，将 EX 和 MEM 中可能发生冲突的数据重定向回多路选择器输入端，实现 ALU 输入端的数据修正。对于分支冲突，解决方法与气泡流水线一致。重定向流水线的逻辑处理电路如图 3.30 所示，重定向流水线 CPU 如图 3.31 所示。

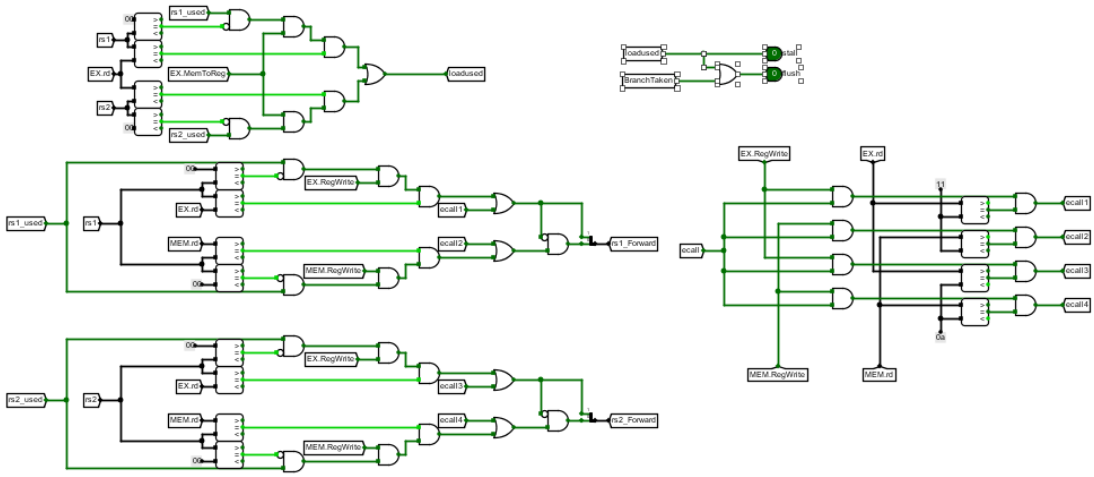


图 3.30 重定向流水线逻辑处理电路

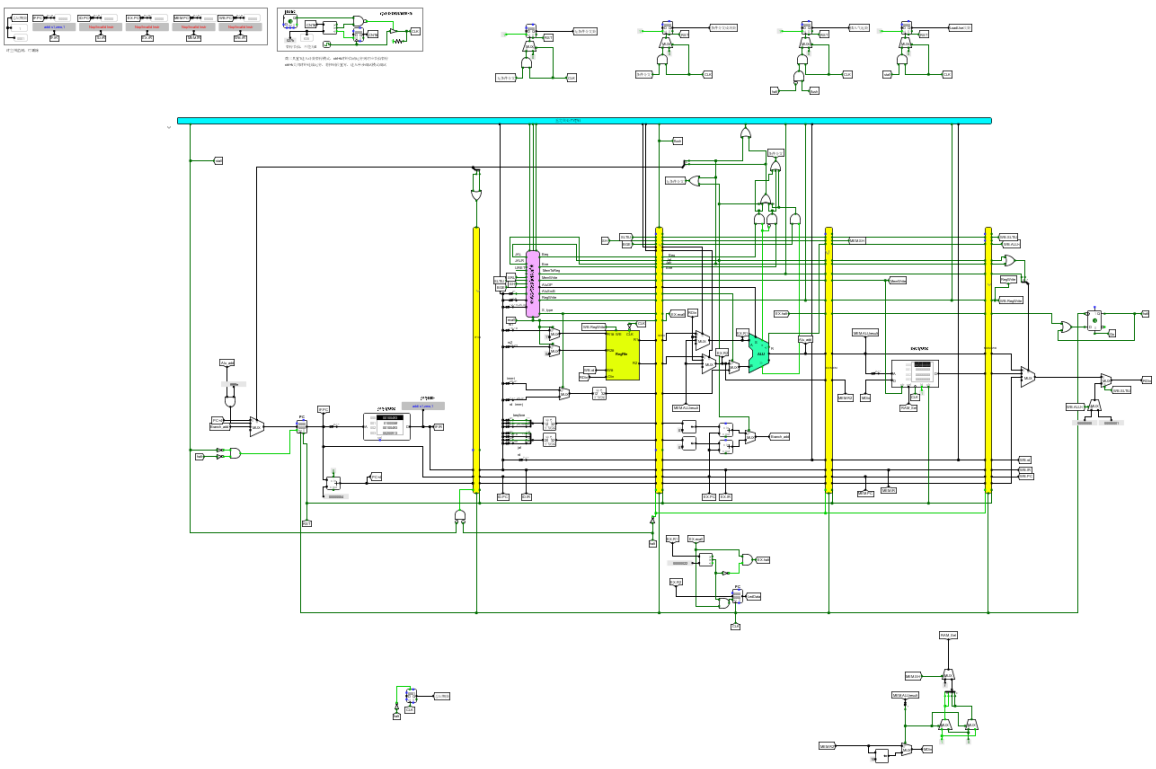


图 3.31 重定向流水线

3.6 动态分支预测机制实现

动态分支预测通过 BTB 表来储存历史的分支跳转信息，对遇到的分支跳转指令进行地址预测，若是预测成功选择预测地址输入 PC，预测失败选择修正地址分支输入 PC。BTB 表类似一个全相联的 cache 结构。每次分支指令执行的时候会把指令地址，目标地址是否跳转的信息送入 BTB 表，BTB 进行全相联并发比较，数据缺失则

载入分支数据，淘汰采取 LRU 算法实现，数据命中则只需要根据实际跳转情况的值更新分支预测历史位的值，以提高预测准确率。BTB 的具体实现如图 3.32 所示。

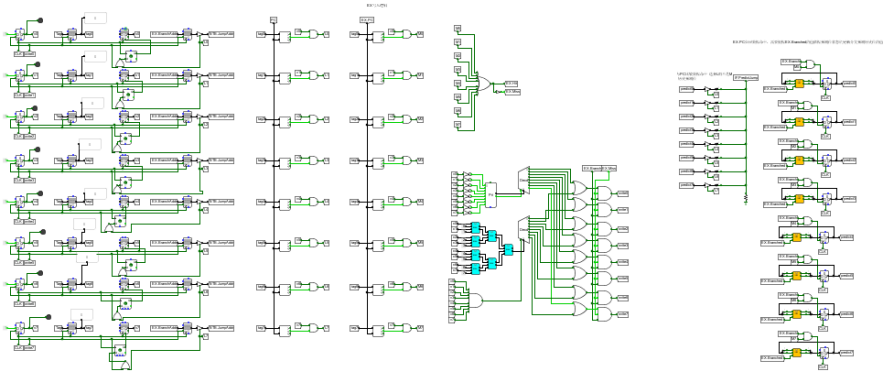


图 3.32　BTB 表具体实现

分支预测采用双位状态预测，根据课本上的状态转换图构建有限状态机 FSM，如图 3.33 所示。

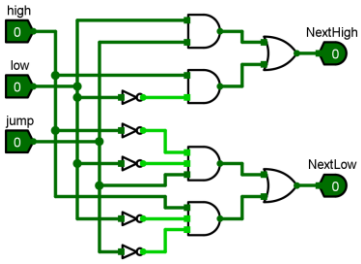


图 3.33　双位状态预测 FSM 具体实现

在主电路中应当添加分支预测所需要的控制信号，分支跳转成功信号。通过 BTB 生成预测地址，并通过预测判别信号，判断选择预测地址还是修正地址。若预测失败，则采用原始的重定向方式插入气泡，保证指令正确运行。相关电路如图 3.34 所示。

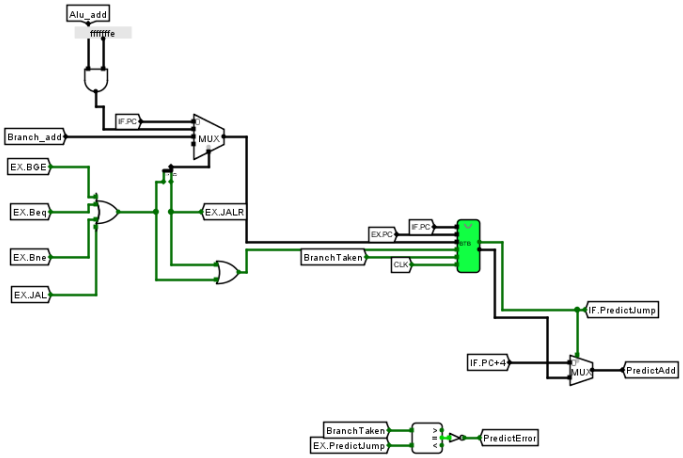


图 3.34 预测地址相关电路

动态分支预测如图 3.35 所示。

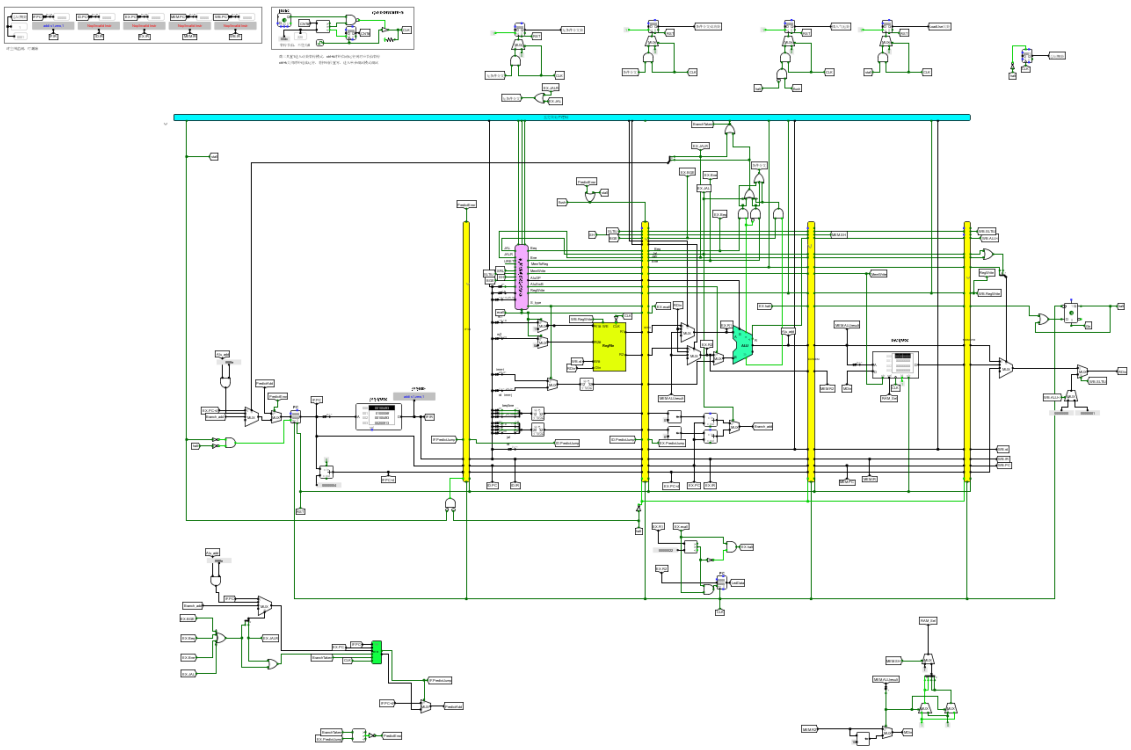


图 3. 35 动态分支预测流水线

3.7 流水中断机制实现

流水中断是在重定向流水的基础上加入了单周期CPU单级中断的部件改造而来。特别要注意的是，中断相关的部件信号应该由全局信号改为流水信号。且为防止在插入气泡时出现中断，导致返回地址出现问题，应当对同时出现在流水线中的分支跳转指令次数以及 Load-Use 指令进行计数，并由此获得正确的返回地址。，相关电路如图 3.36 所示。

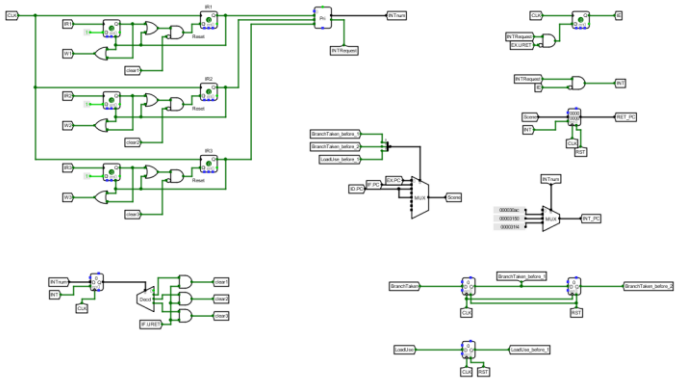


图 3. 36 流水中断相关电路

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 CPU 的 21 条公共指令和 CCAB 个性化指令

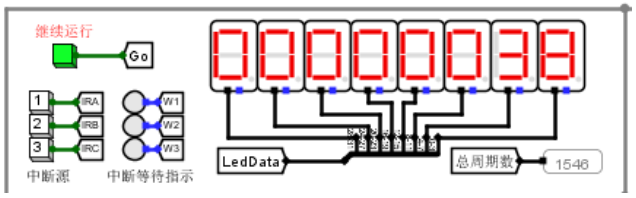


图 4.1 单周期 21 条公共指令运行结果

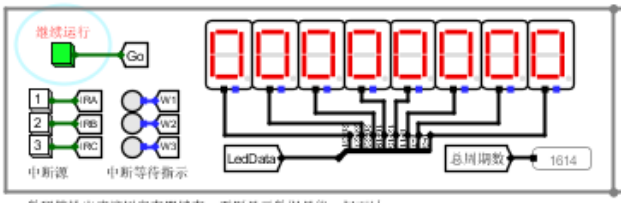


图 4.2 单周期 SRL 指令运行结果

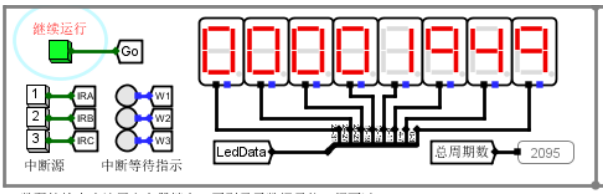


图 4.3 单周期 SLTIU 指令运行结果

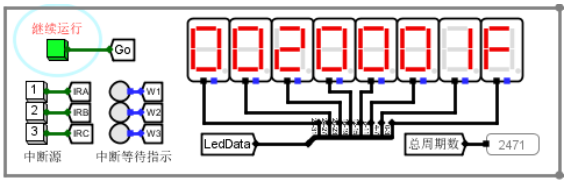


图 4.4 单周期 SH 指令运行结果

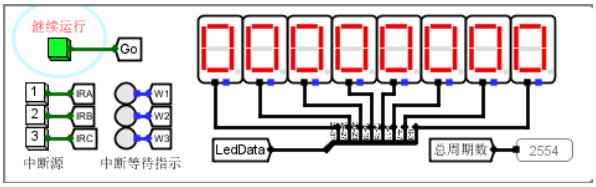


图 4.5 单周期 BGE 指令运行结果

4.1.2 理想流水线测试

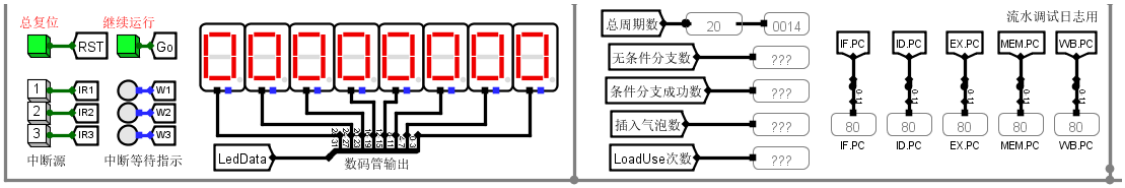


图 4.6 理想流水线运行结果

4.1.3 气泡流水线测试

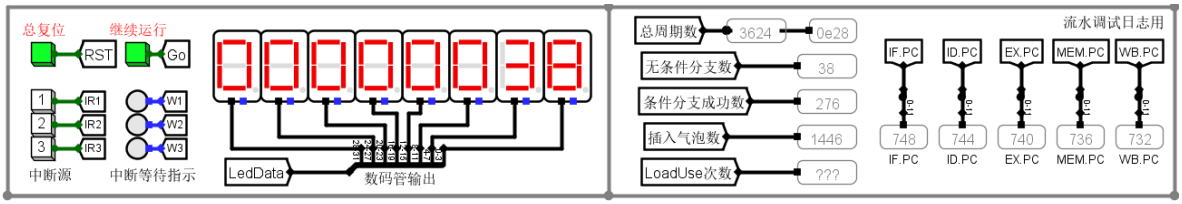


图 4.7 气泡流水线 21 条公共指令运行结果

4.1.4 重定向流水线测试

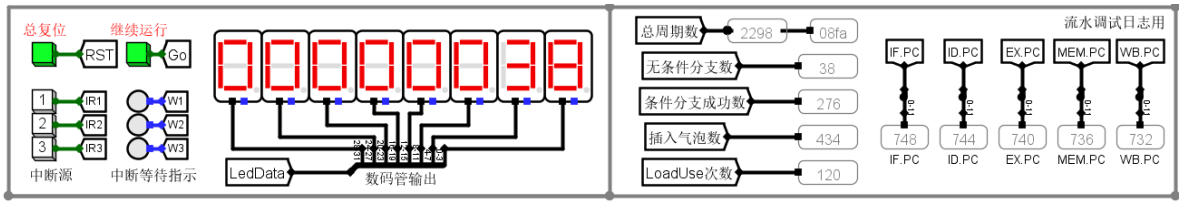


图 4.8 重定向流水线 21 条公共指令运行结果

4.1.5 单周期单级中断

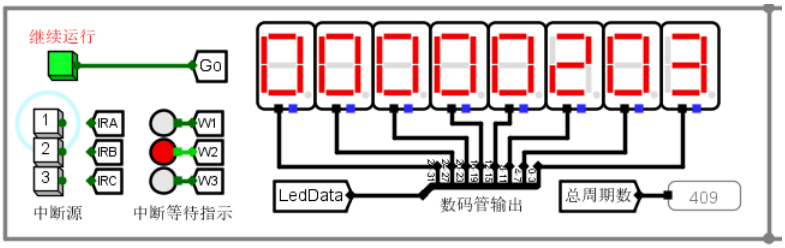


图 4.9 单周期单级中断运行结果

4.1.6 单周期多级中断

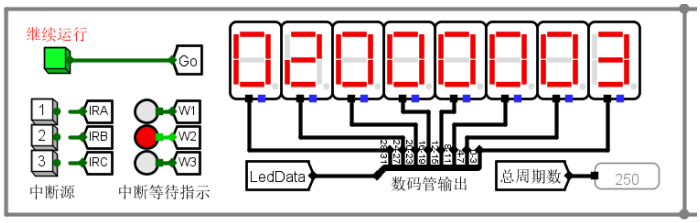


图 4.10 单周期多级中断运行结果

4.1.7 流水线单级中断

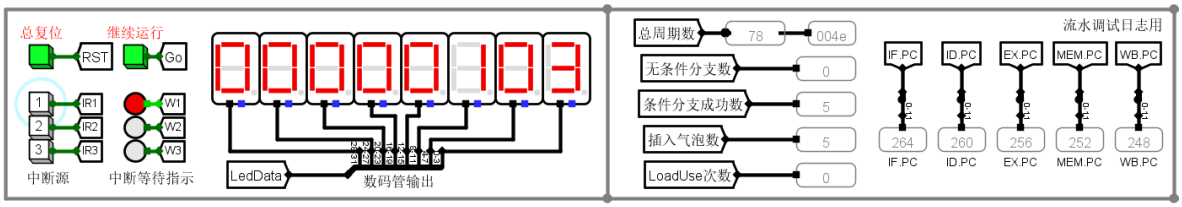


图 4.11 流水线单级中断运行结果

4.1.8 动态分支预测

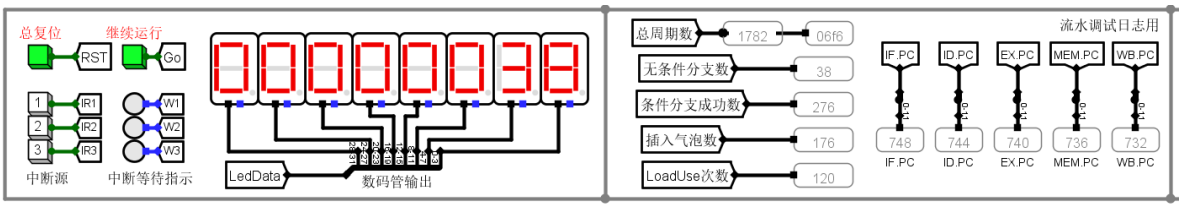


图 4.12 分支预测运行结果

4.2 性能分析

由 21 条公共指令写成的 risc-v-benchmark.asm 程序作为基准测试程序对单周期 CPU，气泡流水线 CPU，重定向流水线 CPU 以及分支预测流水线 CPU 进行测试。得到的测试结果如表 4.1 所示。

表 4.1 基准测试程序 CPU 运行周期

CPU 名称	运行周期
单周期	1545

CPU 名称	运行周期
气泡流水线	3624
重定向流水线	2298
动态分支预测	1782

由表 4.1 可以看出单周期 CPU 的运行周期数是四者里面最少的，但是实际上单周期 CPU 的每个周期的运行时间要比另外三个流水线 CPU 的运行时间长很多，故事实上单周期 CPU 运行基准测试程序的用时是最多的，故性能是最差。而在三个流水线 CPU 中，气泡流水线运行周期数最多，因为气泡流水线在解决数据冲突时都采用插入气泡的方式，而大量的气泡插入会影响 CPU 的性能，故气泡流水线是流水线 CPU 中性能最差的。而重定向流水线次之，因为重定向流水线是在气泡流水线的基础上改进而来，在处理非 Load-Use 指令时，通过将数据重定向给 ALU 的输入端避免了气泡的插入，只有 Load-Use 型指令才选择插入气泡，故插入气泡的数量大大减少，因此性能相比气泡流水线大大提升。动态分支预测 CPU 是所有流水线 CPU 中性能最好的，因为动态分支预测是在重定向流水线的基础上进行优化，通过 BTB 表学习分支跳转指令进行分支跳转地址预测，若预测成功，则可以较少跳转指令执行周期，因此性能最好。

4.3 主要故障与调试

4.3.1 ecall 指令 D 触发器故障

故障现象：单周期 CPU 测试时，本地 logisim 可以正确得到 LED 数据且周期数也是正确的，但是 educoder 上的最后一个周期报错。

原因分析： D 触发器的采用上升沿触发，与寄存器的触发方式相同，导致最后一个周期内数据的传递发生偏差

解决方案： 将 D 触发器改为下降沿触发。

4.3.2 SH 指令故障

故障现象：运行 SH 指令测试程序后，当 LED 显示 20 之后，后面的显示全部错误。

原因分析：直接使用了课设资料保重的数据存储器，在执行 SH 指令时会地址选

择出现缩字的情况，导致数据存储地址错误。

解决方案：改用 CS3410 库中的 MIPS RAM 实现数据存储器读取功能，按照 alu 计算的结果来设置片选信号与 MDin 信号。

4.3.3 气泡流水线停机故障

故障现象：气泡流水线运行完所有程序后无法正确停机，一直运行空指令。

原因分析：在 EX 段正确生成停机信号 halt 后，该信号没有能够正确地传递到 WB 段。

解决方案：定位 ecall 指令出现在 EX 段的周期数，并逐步查看 halt 信号向后传递的过程，排查错误，发现是 EX/MEM 段的流水接口部件中 halt 信号处的多路选择器没有接上选择端口，导致信号传递失败，因此将选择端口接上即完成了停机信号 halt 的传递。

4.3.4 流水线中断故障

故障现象：流水线中断点击中断按钮后，有时会出现无法正确返回中断地址的情况。

原因分析：执行中断时，恰好 EX 段遇上气泡的插入，此时若直接将 EX.PC 的地址直接作为返回地址，便会出现直接跳转回第一条指令处的情况。

解决方案： 添加部件，记录重定向流水线 CPU 中同时执行的指令中分支跳转成功以及 Load-Use 的次数，并作为选择信号，用来选择 EX.PC，IF.PC，ID.PC 三个中的一个作为正确的返回地址存入 mPEC 中。

4.3.5 分支预测失败故障

故障现象： 分支预测的周期数与基础的重定向流水线周期数相比基本没有发生改变，无法达到 1800 以内。

原因分析： 分支预测的地址基本全部失败，导致分支预测流水线按照正常的重定向流水线方式执行指令。

解决方案： 排查分支预测相关部件的情况，发现 BTB 表内部的 MAX 部件的端口封装错误，导致实际上进行比较的值与其编号不对应，因此将该部件的封装进行修改。

4.4 实验进度

表 4.2 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 相关理论知识，阅读课设任务书，阅读 RISC-V 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	
第三天	完成单周期 CPU 的控制信号真值表，通过 Logisim 平台的分析组合逻辑电路的功能完成硬布线控制器的生成以及特殊通路的构建，通过测试。
第四天	对照指令表，查看 CCAB 指令的功能，修改单周期 CPU 的控制信号真值表，并修改硬布线控制器，对三条 CCAB 指令构建特殊通路。
第五天	
第六天	学习流水线相关知识，在单周期 CPU 的基础上，将其改造成理想流水线，并通过测试。
第七天	学习气泡流水线相关知识，完成气泡流水线的部件，并以理想流水线作为基础，搭建出了气泡流水线的总体框架。查找气泡流水线中出现的 bug，并进行修改，通过测试。
第八天	
第九天	学习重定向流水线相关知识，完成重定向流水线的部件，并以理想流水线作为基础，完成重定向流水线，并通过测试。
第十天	学习单周期中断相关的知识，完成单周期单级中断电路的搭建，通过测试。
第十一天	学习单周期中断相关的知识，完成单周期多级中断电路的搭建，通过测试。
第十二天	学习流水线中断知识，完成流水线单级中断电路的搭建，通过检测。
第十三天	学习动态分支预测的知识，完成 BTB 表部件的搭建，并构建出动态分支预测流水线，通过 debug，达到预定指标。
第十四天	

5 设计总结与心得

5.1 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1） 设计了单周期 RISC-V CPU 的数据通路，实现了 21 条公共指令以及 4 条 CCAB 指令的正确运行，完成了程序测试。

2） 设计了理想流水线，气泡流水线以及重定向流水线相关部件，实现了正确的气泡插入以及数据重定向使得 21 条公共指令和 4 条 CCAB 指令在流水线上正确运行，完成了程序测试。

3） 设计了单周期单级中断，单周期多级中断以及流水线中断相关部件，实现了中断进入以及中断返回功能，完成了程序测试。

4） 设计了 BTB 表等动态分支预测相关电路，实现了通过分支跳转地址预测减少气泡插入以提高流水线 CPU 性能，完成了预定指标。

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以及中秋节假期的辛苦加班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。

课设的第一关是利用 Logisim 设计一个 RISC-V 的单周期 CPU，为此我先重新复习了有关单周期 CPU 数据通路的相关知识，同时也通过 PPT 以及指令集手册学习了 RISC-V 指令集。之后参考 PPT 与课本上给出的数据通路顶层视图，一步一步地将单周期 CPU 的框架搭建出来，之后通过 educoder 平台测试，修改其中的 bug。

完成单周期 CPU 之后，我就着手完成三个流水线 CPU 的构建。我同样也是先复习了流水线相关知识，搭建出了理想流水线的电路之后，在其基础之上再通过添加部件的方式完成气泡流水线以及重定向流水线的搭建。在修改 bug 的过程中，遇到一个比较大的问题就是，知道错误信息发生的周期之后，经常需要回退到错误点之前的好

几个周期，才能确定错误的原因并进行修改，而且如果不注意信号的变化，就可能要重新进行一次回溯，经常会出现构建电路一小时，修改 bug 几小时的情况。在这过程中，我深刻体会到在修改电路的过程中，记录信号变化过程的重要性以及从底层电路排查问题的困难。

完成了几个 CPU 的搭建之后，我就开始复习中断相关的知识，为几个 CPU 添加中断相关的部件。单周期单级中断以及多级中断相对来说问题不是很大，但是在实现流水线单级中断的时候，一开始我之间按照单周期单级中断的方式修改电路，但是一直会出现 bug，后来在同学指导下，发现当插入气泡时响应中断会导致中断返回地址为 0，因此根据这一情况设计了电路，根据气泡的插入确定返回地址，最后成功解决问题。这让我明白了闭门造车是不行的，当遇到问题时只会埋头苦干，很有可能在错误的道路上越走越远，相反，在这种情况下，我们更应该与同学多沟通交流，发现问题所在。

个人任务最后一关是动态分支预测，这关的难度非常大，其中最关键的就是 BTB 表这一个部件的构造。为了完成这一部件，我不仅将课本上相关内容都看了一遍，还复习了上学期 cache 实验相关的内容。这一关让我体会到了即使复习的重要性。

我们组的做的是基于 RISC-V 单周期 CPU 的走迷宫软件，我在组内负责硬件电路的搭建工作。这个任务其实不算很难，主要就是利用 32 个 32 位数据来显示一个 32×32 的点阵，通过六个中断服务程序实现输入，同时配合相应的判断报警电路即可。在这个过程中，遇到比较大的问题是软件部分与硬件部分沟通不够，我以我的单周期 CPU 为基础开发相应电路，而负责软件的同学没有告知我一些要用到的 CCAB 指令，导致在最后测试时出现了比如拖影等问题，最后经过一同调试才发现是几条 CCAB 指令没有实现，添加上去之后问题就解决了，这告诉我们团队内要定时交流，不能单干。

最后，我想向老师提出以下建议：

（1） 为 logisim 软件提供状态回退功能，使电路可以通过这个功能回退到任意一个周期，而不是要重新跑一次，便于我们调试。

（2） 为流水中断和动态分支预测两个关卡提供 educoder 测试，方便我们寻找程序中的 bug。

（3） 对于比较复杂的关卡，可以分割成几个部件的单独关卡，降低难度梯度。

6 团队任务

6.1 团队任务设计

6.1.1 项目介绍

我们小组根据往届学长的各种团队任务方向作为参考灵感，最终选择在 logisim 上实现一个走迷宫的小游戏，该迷宫采用 32*32 规模的 LED 点阵元件作为游戏的显示屏幕，游戏目的为操纵可移动的小光点从右上角走到左下角终点处。CPU 运行后首先显示初始化界面，然后设计两种难度的关卡进行选择，根据选择后载入地图，然后鼠标点击键盘逻辑处，通过输入 WASD（记得调整为英文输入语法，否则可能无法识别键盘输入）来操纵小光点的移动，当下一步操作会撞到亮起的墙壁后会发出警告，禁止该无效操作，当走到左下角后屏幕输出 SUCCESS 字符，并响起庆祝音乐。（建议在 128Hz 下游玩）

6.1.2 项目的设计方案

项目总体分为软件和硬件两个方面进行设计。

软件方面主要是迷宫游戏的汇编程序的编写然后将其利用 RARS 软件转换为 Hex 文件输入 CPU。程序的逻辑如下：首先初始化的时候设定某些固定的点阵亮起作为初始化界面，然后进入空操作的循环等待用户进行关卡选择，每个关卡选择都是一个中断程序，能够打断空操作死循环进入相应的终端服务程序，重新更改界面，载入相应难度的迷宫地图，然后同样进入空指令 nop 的循环，依靠上下左右移动四个中断程序来实现光点位置的移动，在每个移动程序的末尾都有一个当前位置是否为终点的判断，保证一旦抵达终点就能跳转到游戏通关部分程序段，修改点阵信息显示 SUCCESS 字样，并且触发音乐播放功能。

硬件方面主要依靠单人任务中实现的单周期 CPU 作为游戏的主控 CPU。在该 CPU 上增添对游戏的相关硬件支持。

首先是关卡选择和上下左右移动六个中断逻辑的实现，需要更改原本设定好的三个中断逻辑为程序经过 rars 编译后的六个中断服务程序的入口作为断点。

还有显示逻辑的硬件支持，需要设计 32 个 32 位的寄存器来控制对应的 LED 点阵的亮灭，实现的时候要设计两个 32 位寄存器电路一个用于备份，保证将地图和操

作光点数据写入内存的同时也要备份一份以便于能在同一时钟周期内全部读取，利用ecall 指令的调用来更新地图信息，将备份寄存器的数据更新到点阵链接的寄存器组即可。

6.2 团队任务实现

6.2.1 团队分工

主要工作分配如下：

陈鹏翰主要负责软件部分汇编代码的编写。

李学森主要负责硬件电路的绘制。

陈铭铭负责协调软硬件的协同问题，根据实际情况调整软件或者硬件方面的设计，进行相关测试进行 debug，进行最终的展示视频和 ppt 制作。

6.2.2 开发平台

主要利用 Visual Studio 来完成汇编程序的编写，利用 RARS 来编译 Risc-V 汇编程序并进行 hex 文件的转换，硬件开发仍是基于 Logisim 平台。

6.2.3 开发中遇到的问题

首先根据测试给的那些汇编程序作为参考，编写走迷宫的汇编程序，其中上下左右键移动的程序逻辑比较复杂，要和关卡选择中断互锁，判断是否抵达终点，是否撞墙超出边界等。在程序的实践中发现当向左碰到墙壁后无法再进行移动，后续通过程序中逻辑的检查发现了向左撞墙程序段相比其他方向撞墙程序少了处理程序。在编写汇编程序的时候发现还需要额外扩展指令的存在，例如 xor 指令不存在移动光点会产生拖影，sll 指令不存在无法移动光点， LUI 指令用于加载地图，根据这些在软件中需要的指令我们又在 CPU 中增添了对该指令的支持。

对于报警电路的实现刚开始有点没有思绪，后来想到利用 ecall 指令来实现，当需要报警时置 a7 为 100，即可触发报警电路。但是在高频下声音太短了，后来又想到用一个延时的电路让报警声音延长 8 倍，最后报警的效果很好。

最终实现迷宫游戏玩家交互电路图如图 6.1 所示，具体功能见 PPT 和视频。

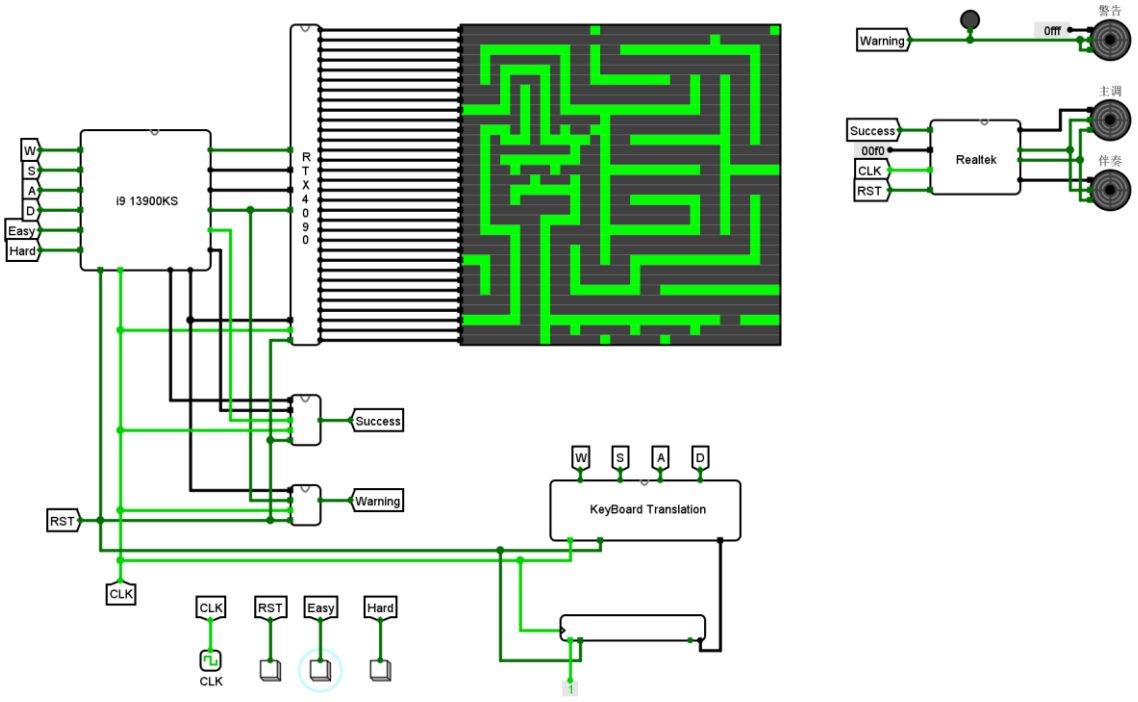


图 6.1 迷宫游戏玩家电路示意图

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：李学森