华中科技大学

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：	CS2003
学　　号：	U202015372
姓　　名：	张宁静
电　　话：	19907148507
邮　　件：	u202015372@hust.edu.cn

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	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	

#	指令助记符	简单功能描述	备注
13	SW	存字	
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 $a7==50 halt(停机指令)
21	ECALL	系统调用	else 数码管显示$a0 值
22	CSRRSI	访问 CSR	中断相关，可简化，选做
23	CSRRCI	访问 CSR	中断相关，可简化，选做
24	URET	中断返回	异常返回，选做
25	SLL	逻辑左移	C-1
26	XOR	异或	C-4
27	LB	读取一个字节写入 rd 中	A-1
28	BLTU	无符号小于条件分支	B-3

2 总体方案设计

2.1 单周期 CPU 设计

本次我们采用的方案是采用指令存储器和数据存储器相分离的结构（哈佛结构）。在这里我们采用简单迭代法实现单周期处理器。 这种方法相对比较直观简单，设计者先完成支持一条固定 R 型指令（如加法指令）的基本数据通路，测试运行通过后再在这个基础上不断增加新的数据通路，支持新的指令，直至所有指令都能正常运行。要支持一条指令的运行必须有取指令逻辑和执行指令的逻辑，所有设计者首先应该完成取指令的数据通路。

需要特别说明的是，在单周期处理器中，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，因此任何需要被多次使用的资源(如加法器)都需要设置多个，否则就会发生资源冲突，这是设计单周期处理器数据通路时必需考虑的重要环节。

总体结构图如图 2.1 所示。

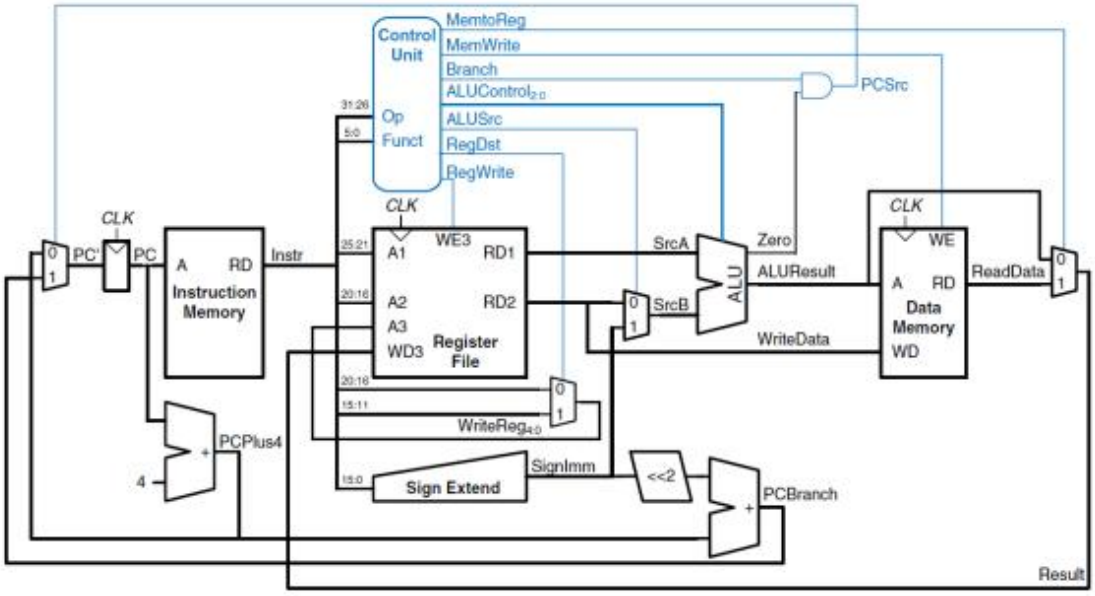

图 2.1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

取指令通路应能取出程序计数器 PC 锁存地址对应的的机器指令，并能在时钟上跳沿到来时实现 PC 自动加 4，从而进入下一条指令的指令周期。这里程序计数器 PC 可以采用寄存器实现，其输出作为指令存储器的地址输入，指令存储器经过一个存储周期后一次性取出 32 位的机器指令，故其数据宽度应为 32。

2. 指令存储器 IM

为简化电路设计，Logisim 中采用 ROM 实现。在顺序执行方式下，每一个时钟周期内 CPU 取指令后将 PC 寄存器的值加 4，形成下一条指令的地址。这里加“4”是因为 RISC-V32 中指令字长均为 4 字节，每条指令在存储器中占用 4 个字节的存储单元，而 PC 中存放的地址是字节地址。为避免资源冲突，这里加法器应该是独立的器件，和 CPU 中的算术逻辑运算单元分开。

表 2.1 指令存储器引脚与功能描述

引脚	输入/输出	位宽	功能描述
PC	输入	10	程序地址
IR	输出	32	待解析指令

3. 运算器 ALU

运算器接受两个 32 位的操作数输入，根据操作码决定对两个数进行何种运算，输出一个 32 位的结果，并输出一位数表示两个操作数是否相等。除此之外还有有符号溢出、无符号溢出等多种输出，由于在这里不需要使用，所以忽略这些输出。输入输出引脚描述如表 2.2。

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

引脚	输入/输出	位宽	功能描述
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 时输出 1，其他操作为零
<	输出	1	在特定 ALU_OP 输入且 X<Y 时为 1，其他为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

寄存器文件 RF 是 RISCV - CPU 中 32 个 32 位寄存器组成的阵列，通常由快速的静态随机读写存储器（SRAM）实现。这种 RAM 具有专门的读端口与写端口，可以多路并发访问不同的寄存器。RF 一次可以读取 R1 和 R2 两个寄存器；一次可写一个寄存器，用 Din 表示，用时钟控制，一位写使能信号用来标识当前周期是否需要写数据；写入和读出的数据都是 32 位的。输入输出引脚描述如 3。

表 2.3 寄存器堆引脚与功能描述

引脚	输入/输出	位宽	功能描述
Clk	输入	1	时钟
W#	输入	5	写寄存器号
R1#	输入	5	读寄存器号 1
R2#	输入	5	读寄存器号 2
WE	输入	1	写使能
Din	输入	32	写入数据
R1	输出	32	读出数据 1
R2	输出	32	读出数据 2

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	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
ADD	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
ADDI	PC+4	PC	rs1		rd	ALU	RF.R1		4			
AND	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
ANDI	PC+4	PC	rs1		rd	ALU	RF.R1		4			
SLL	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
SRA	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
SRL	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
SUB	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
OR	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
ORI	PC+4	PC	rs1		rd	ALU	RF.R1		4			
XORI	PC+4	PC	rs1		rd	ALU	RF.R1		4			
LW	PC+4	PC	rs1		rd		RF.R1		0	ALU		
SW	PC+4	PC	rs1	rs2			RF.R1	RF.R2	8	ALU	RF.R2	
BEQ		PC	rs1	rs2			RF.R1	RF.R2	0x18			
BNE		PC	rs1	rs2			RF.R1	RF.R2	0x18			
SLT	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
SLTI	PC+4	PC	rs1		rd	ALU	RF.R1		4			
SLTU	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
JAL		PC			rd				0x1b			
JALR		PC	rs1	0	rd		RF.R1		0x19			
ECALL	PC+4	PC	0	0	0				0x1c			
CSRRSI		PC	rs1		rd		RF.R1		0x1c			
CSRRCI		PC	rs1		rd		RF.R1		0x1c			
URET		PC	0	2	0				0x1c			

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
SLL	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
XOR	PC+4	PC	rs1	rs2	rd	ALU	RF.R1	RF.R2	c			
LB		PC	rs1		rd		RF.R1		0			
BLTU		PC	rs1	rs2	rd		RF.R1	RF.R2	0x18			

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2.5。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
OP	指令字的2-6位	每种指令对应的 opcode 不同，具体值根据 RISC-V 指令码表确定
Funct	指令字的第12-14+25+30位	funct 字段取值
IR21	指令字第 21 位	URET 和 ECALL 仅凭 funct3 和 funct7 不能区分，要使用指令的第 21 位特判。
ALU_OP	四位二进制数	运算器操作控制符
MemToReg	0/1	写入寄存器的数据来自存储器
MemWrite	0/1	写内存控制信号
ALUSrcB	0/1	运算器第二输入选择
RegWrite	0/1	寄存器写使能控制信号
uret	0/1	中断信号
ecall	0/1	停机信号
S_type	0/1	sw 相关
Beq	0/1	Beq 指令译码信号
Bne	0/1	Bne 指令译码信号

控制信号	取值	说明
Jalr	或 0/1	函数调用，写寄存器的编号选择 31 号
JAL		写寄存器编号选择指令解析器解析出的 rd、rt
LB	0/1	LB 指令提示信号
BLTU	0/1	BLTU 指令提示信号
CSRRSI	0/1	多级中断相关，开中断
CSRRCI	0/1	多级中断相关，关中断
rs1	0/1	用于气泡流水线和重定向流水线的判断
rs2	0/1	

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	SLL	XOR	LB	BLTU	rs1_used	rs2_used
add	0	0	c	5				1											1	1
sub	32	0	c	6				1											1	1
and	0	7	c	7				1											1	1
or	0	6	c	8				1											1	1
slt	0	2	c	11				1											1	1
sltu	0	3	c	12				1											1	1
addi		0	4	5			1	1											1	
andi		7	4	7			1	1											1	
ori		6	4	8			1	1											1	
xori		4	4	9			1	1											1	
slti		2	4	11			1	1											1	
slli	0	1	4	0			1	1											1	
srli	0	5	4	2			1	1											1	
srai	32	5	4	1			1	1											1	
lw		2	0	5	1		1	1											1	
sw		2	8	5		1	1			1									1	1
ecall	0	0	1C						1											
beq		0	18								1								1	1
bne		1	18									1							1	1
jal			1b					1					1							
jalr		0	19					1						1					1	
CSRRSI		6	1c																	
CSRRCI		7	1c																	
URET	2	0	1c																	
SLL	0	1	c	0				1							1				1	1
XOR	0	4	c	9				1								1			1	1
LB		0	0	5	1		1	1									1		1	
BLTU		6	18	12														1	1	1

2.2 中断机制设计

2.2.1 总体设计

本实验要求设计三种中断，单极中断、多级中断还有流水线中断。

每个中断对应一个中断号。根据中断号确定执行哪段中断程序，用类似于中断向量表的方式存放中断程序入口，可以事先获取中断入口地址，以常量形式进行选择。为了在中断执行后能够正常返回主程序，要将跳转前的地址保存下来，就像 jal 指令将返回地址保存在 31 号寄存器中一样。每级中断在执行过程中都需要动态地

改变中断优先级，所以需要若干个中断屏蔽寄存器。

2.2.2 单级中断设计

单级中断采用一个寄存器 EPC 保留返回地址，用一个 D 触发器控制中 2 断使能信号 IE。指令方面只需要增加对 ERET 指令的支持即可。在一条指令的上升沿后，检测到中断请求，立刻执行中断隐指令：将 PC 的值保存到 RET_PC 中，将中断使能 IE 置为 0，并通过该中断号对应的入口地址 INT_PC 取得该中断程序的第一条指令, 那么下一个时钟上升沿就开始执行中断服务程序了。

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

2.2.3 多级中断设计

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

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

2.2.4 流水中断设计

本实验选择在 EX 段处理中断：①MEM、WB 段的 2 条指令要继续执行完；②IF、ID 段的 2 条指令要取消；③EX 段指令如果继续执行完，断点地址应该是该顺序指令地址或分支跳转地址；如果不允许继续执行，断点就是 EX 段指令的 PC；（通常做法是不允许这条指令继续执行）；另外如有指令在 EX 段之前完成，比如无条件分支指令在 ID 段完成，此时逻辑又有区别；④暂停流水线进行中断响

应；（根据具体的实现方法也可以不暂停）；⑤重新启动流水线从中断入口地址处取指令，进入中断服务阶段。

2.3 流水 CPU 设计

2.3.1 总体设计

单周期 RISC-V 处理器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。

所有流水寄存器，程序计数器 PC、寄存器堆、数据存储器均采用统一时钟 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

这里总共增加了四个流水寄存器，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 个流水寄存器细分为五段流水线。对于不同的流水线，需要对这几个流水寄存器进行适当修改，以适应不同流水线功能需要。

2.3.3 理想流水线设计

1）设计流水寄存器

2）重新封装较大尺寸的功能部件

3）重构数据通路

4）增加流水线时空图检测

5）增加统计功能

2.4 气泡式流水线设计

2.4.1 总体设计

理想流水线只能顺序执行，无法解决跳转问题；此外，相邻指令间还存在着读写数据的冲突，读寄存器指令读取的寄存器值还未被及时写回，会导致读取错误。

分支跳转是否成功要等到执行段才能判断出来,此时修改 PC 的值为分支目标地址，取指段和译码段已经取出了两条指令，这两条指令是不应该被执行的，应该清空，使 CPU 空转两个周期，这就是控制冲突的解决方案。

2.4.2 结构冲突处理

多条指令在同一时钟周期都需要使用同一操作部件而引起的冲突称为结构冲突。在本实验中，计算 PC+4、计算分支目标地址、运算器运算都需要使用运算器，访问指令和访问数据都需要使用存储器。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。

2.4.3 控制冲突处理

当流水线遇到分支指令或其他会改变 PC 值的指令时，在分支指令之后载入流水线的相邻指令可能因为分支跳转而不能进入执行阶段，这种冲突成为控制冲突。分支指令是否分支跳转、分支目标地址的计算要等到 EX 段才能确定，而分支指令后续若干条指令已经预取进入流水线，当分支指令成功跳转时，已预取的指令不能继续执行，此时需要清空这些预取指令，同时修改程序计数器 PC 的值，取出分支目标地址处的指令。

2.4.4 数据冲突处理

当前指令要用到先前指令的操作结果，而这个结果尚未产生或尚未到达指定的位置，会导致当前指令无法继续执行，这称为数据冲突。最简单的处理办法是推后执行与其相关的指令，直至目的操作数写入才开始取源操作数，以保证指令和程序执行的正确性。通常可以采用插入“气泡”解决。

2.5 重定向流水线设计

2.5.1 总体设计

进一步改造气泡式流水线，增加重定向机制，增加 Load-Use 数据相关检测机制，使得流水线能在不插入气泡的情况下处理大部分数据相关问题，最终能正确运行单周期测试程序 benchmark.hex。

2.5.2 重定向数据通路

将 EX/MEM 中锁存的运算器运算结果、MEM/WB 中锁存的运算结果、访存数据等重定向到 EX 段，在合适的位置增加多路选择器，增加选择控制信号。

2.5.3 重定向逻辑

首先增加 Load-Use 数据相关检测逻辑，如出现 Load-Use 相关执行原有插入气泡的逻辑进行处理。如出现非 Load-Use 数据相关，则由重定向逻辑直接生成操作数来源选择信号，对进入运算器或存储器的操作数正确来源进行选择。

2.6 动态分支预测机制

2.6.1 总体设计

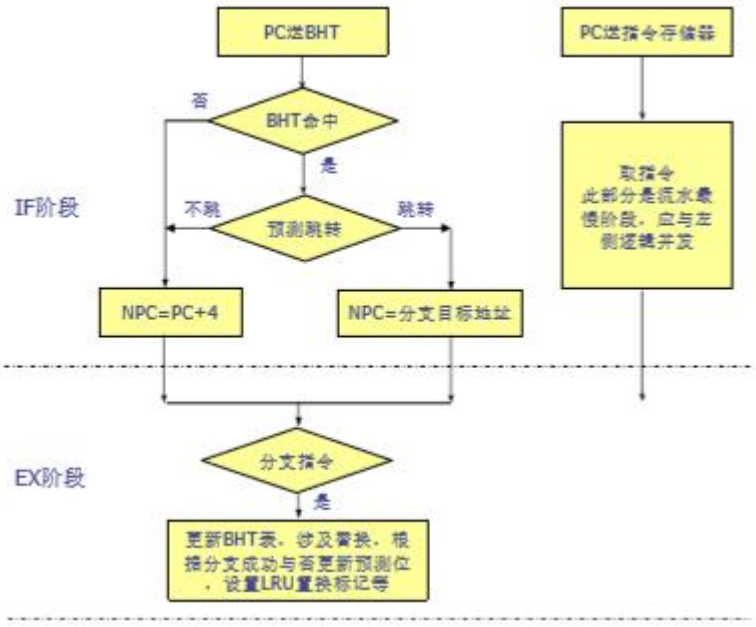

图 2.2 动态分支预测流程

动态分支预测技术的目的是判断预测分支是否成功，尽快找到分支目标地址（或指令），避免控制相关造成流水线停顿。在预测错误时，要作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。动态分支预测流程如图 2.3 所示。使用分支历史表 BHT 记录分支指令最近一次或几次的执行情况（成功或不成功），并据此预测。采用两位预测位的状态转换图如下：

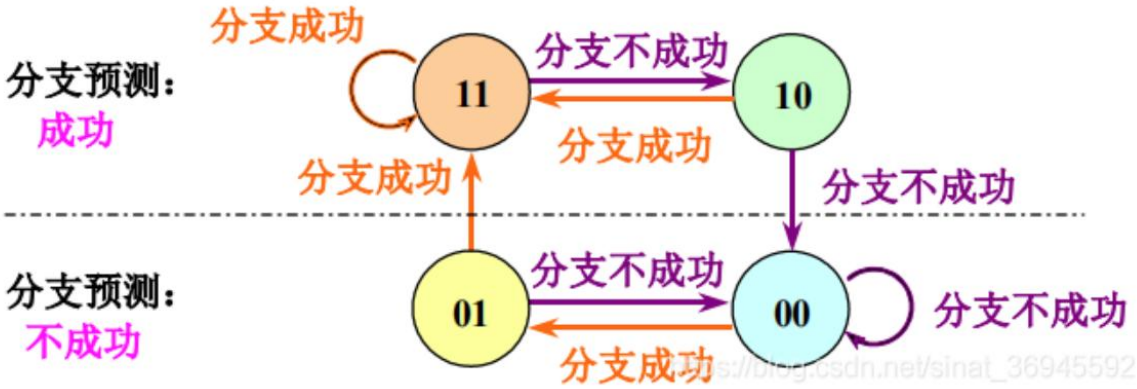

图 2.3 状态转换图

2.6.2 硬件设计

需要设计一个分支目标缓冲器 BTB。目标是将分支的开销将为 0，方法是将分支成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存起来，缓冲区以分支指令的地址作为标识。这个缓冲区就是分支目标缓冲器。

BTB 可以是用专门的硬件实现的一张表格，表格中的每一项通常有两个字段：执行过的成功分支指令的地址和预测的分支目标地址。

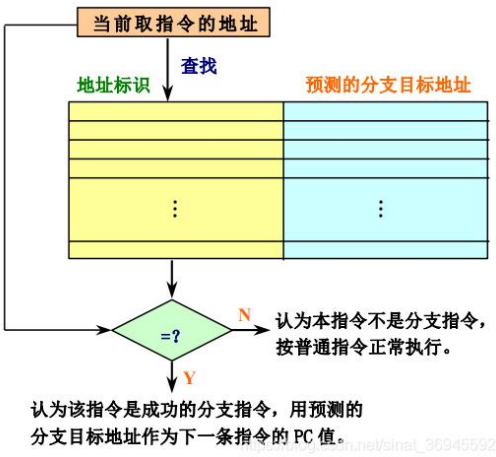

图 2.4 BTB 设计图

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）

① Logism 实现：

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接在寄存器使能端，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，寄存器忽略时钟输入，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

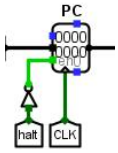

图 3.1 程序计数器（PC）

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

```
always@(negedge clk,posedge clear)
begin
    if(clear)
        pc_out<=0;
    else if(!halt)
        pc_out<=pc_in;
end
```

2) 指令存储器（IM）

① Logism 实现：

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）

② FPGA 实现：

直接使用 Vivado 中自带的 ROM 作为指令存储器，其设置如**错误!未找到引用源。**所示。选择 ROM 的数据位宽为 32 位，因为该 ROM 的地址位宽为 10 位，所以选择 ROM 的大小选择为 1024。

指令存储器 IM 的 Verilog 代码如下：

```
pc pcmeml(im_in[11:2],im_out);
```

直接调用之前设置的 ROM 作为指令存储器，输入为指令地址的 2-11 位，输出为该指令。

3) 寄存器文件（RF）

① Logism 实现：

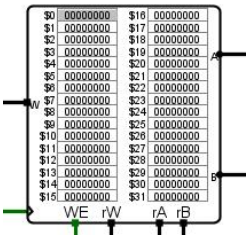

图 3.3 寄存器文件 RF

② FPGA 实现：

```
reg [31:0] regifile[31:0];
integer k;
initial begin
        for(k=0;k<32;k=k+1)
                regifile[k]=0;
```

```
            read_reg1_data <= 0;
            read_reg2_data <= 0;
        end

        always @(posedge clk) begin
            if(write_read_ena && write_reg_addr) begin
                regifile[write_reg_addr] <= data_in;
            end
    end
```

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		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
ADD	PC+4	PC	rs	rt	rd	alu	r1	r2	5			
ADDI	PC+4	PC	rs		rt	alu	r1	立即数	5			
ADDIU	PC+4	PC	rs		rt	alu	r1	立即数	5			
ADDU	PC+4	PC	rs	rt	rd	alu	r1	r2	5			
AND	PC+4	PC	rs	rt	rd	alu	r1	r2	7			
ANDI	PC+4	PC	rs		rt	alu	r1	立即数	7			
SLL	PC+4	PC		rt	rd	alu	r2	立即数	0			
SRA	PC+4	PC		rt	rd	alu	r2	立即数	1			

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
SRL	PC+4	PC		rt	rd	alu	r2	立即数	2			
SUB	PC+4	PC	rs	rt	rd	alu	r1	r2	6			
OR	PC+4	PC	rs	rt	rd	alu	r1	r2	8			
ORI	PC+4	PC	rs		rt	alu	r1	立即数	8			
NOR	PC+4	PC	rs	rt	rd	alu	r1	r2	10			

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据通路表，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，如图 3.4 所示。

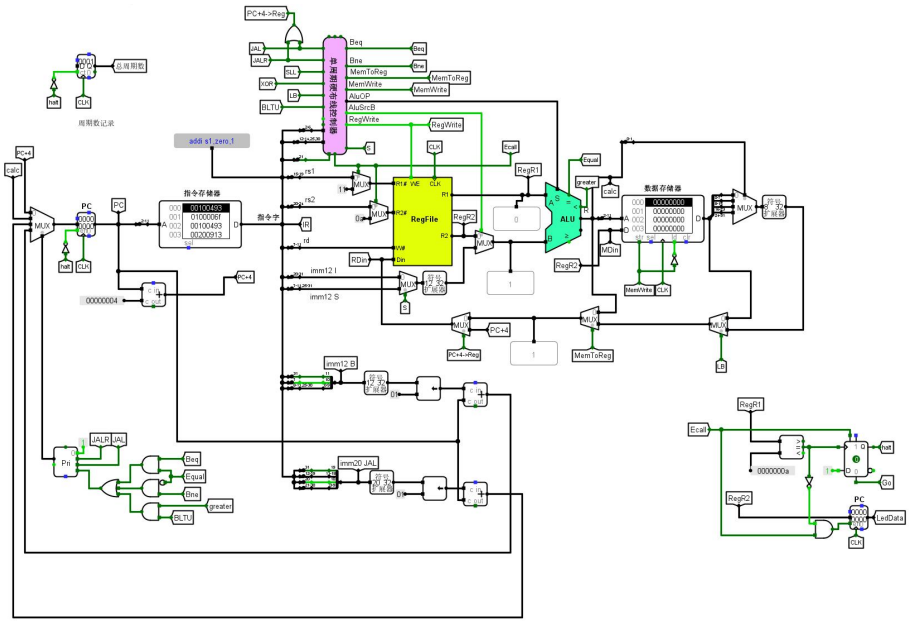

图 3.4 单周期 CPU 数据通路（Logism）

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3.5 所示，代码过长，此处省略。

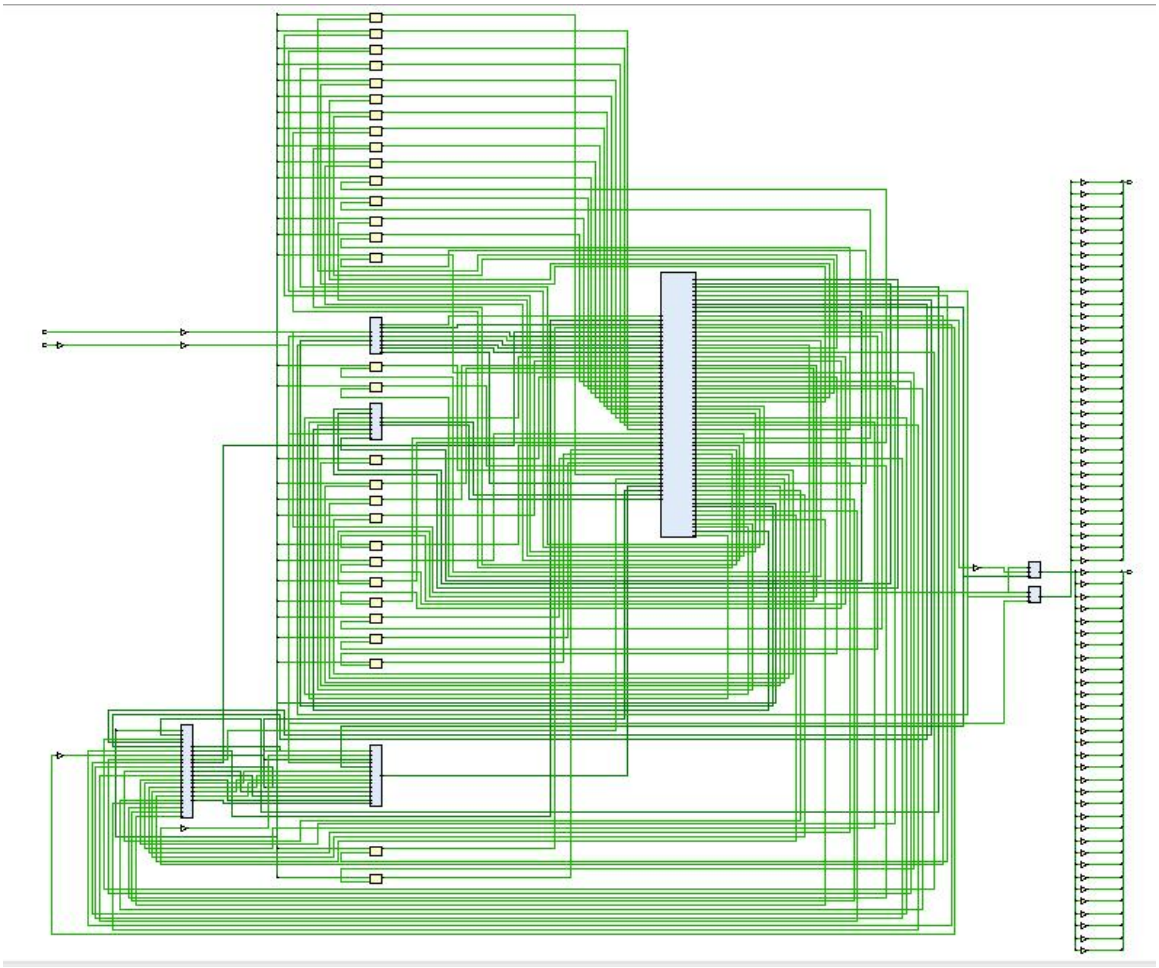

图 3.5 单周期 CPU 数据通路（FPGA）

3.1.3 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，分别在 Logism 和 Vivado 上进行主控制器、Branch 控制器、ECALL 控制器的具体实现。

主控制器对照表 3.2 所示。

表 3.2 主控制器控制信号

指令	R	RW	WE	X	EXT	Y	ALUop	MemWrite	MemRead	Din	Branch	SYSCALL
ADD	00	00	1	0	0	00	0101	0	0	00	00	0
ADDI	00	10	1	0	0	10	0101	0	0	00	00	0
ADDIU	00	10	1	0	0	10	0101	0	0	00	00	0
ADDU	00	00	1	0	0	00	0101	0	0	00	00	0

AND	00	00	1	0	0	00	0111	0	0	00	00	0
ANDI	00	10	1	0	1	10	0111	0	0	00	00	0
SLL	00	00	1	1	0	01	0000	0	0	00	00	0
SRA	00	00	1	1	0	01	0001	0	0	00	00	0
SRL	00	00	1	1	0	01	0010	0	0	00	00	0
SUB	00	00	1	0	0	00	0110	0	0	00	00	0
OR	00	00	1	0	0	00	1000	0	0	00	00	0
ORI	00	10	1	0	1	10	1000	0	0	00	00	0
NOR	00	00	1	0	0	00	1010	0	0	00	00	0
LW	00	10	1	0	0	10	0000	0	1	10	00	0
SW	00	00	0	0	0	10	0000	1	0	00	00	0
BEQ	00	00	0	0	0	00	0000	0	0	00	01	0
BNE	00	00	0	0	0	00	0000	0	0	00	01	0
SLT	00	00	1	0	0	00	1011	0	0	00	00	0
SLTI	00	10	1	0	0	10	1011	0	0	00	00	0
SLTU	00	00	1	0	0	00	1100	0	0	00	00	0
J	00	00	0	0	0	00	0000	0	0	00	10	0
JL	00	01	1	0	0	00	0000	0	0	01	10	0
JR	00	00	0	0	0	00	0000	0	0	00	11	0
SYSCALL	11	00	0	0	0	11	0000	0	0	00	00	1

① FPGA 实现

根据在 Logism 实现中得到的各个一位控制信号的表达式，直接使用数据流建模，使用 assign 分的 Verilog 代码过于冗长，故只取对于控制信号 X 的生成代码举例如下：

```
assign
    X=(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F
    [2]&~F[1]&~F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~
    F[4]&~F[3]&~F[2]&F[1]&F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP
```

```
[0]&~F[5]&~F[4]&~F[3]&~F[2]&F[1]&~F[0]);
```

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。在 Vivado 中使用 Verilog 语言构成的主控制器原理图如图 3.5 所示。

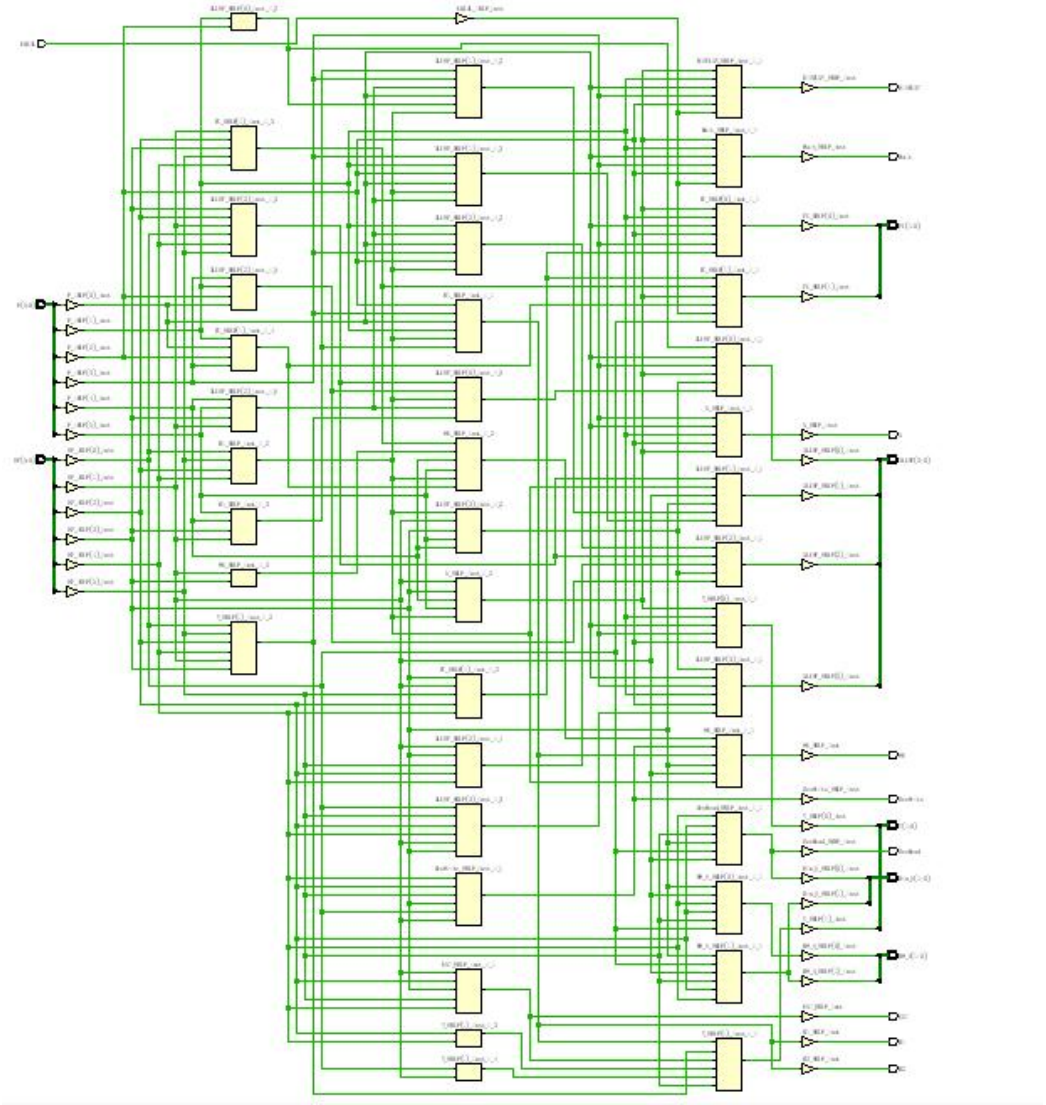

图 3.5 主控制器原理图

3.2 中断机制实现

3.2.1 单周期+单级中断

计算 PC 部件如图 3.6 所示，在原单周期 PC 部件上修改，增设中断 PC 输入。

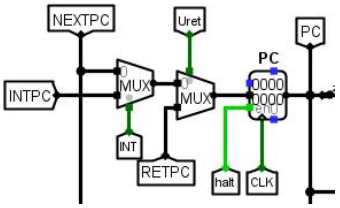

图 3.6 PC 计算部件图

单极中断控制电路如图 3.7 所示，包括中断控制信号生成、中断入口地址选择、等待信号清零等功能实现。

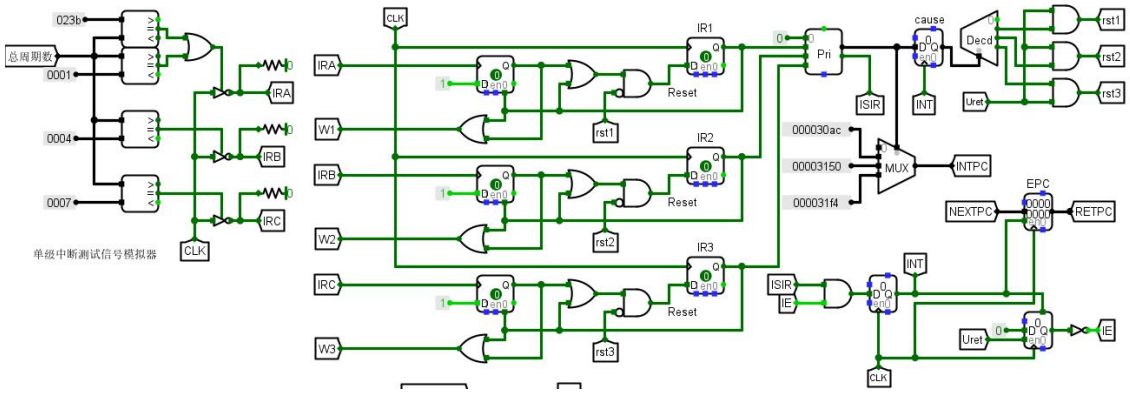

图 3.7 单极中断控制电路

3.2.2 单周期+多级中断

多级中断控制电路如图 3.8 所示，比单级中断多增加 csrrsi、csrrci、csrrw 指令数据通路，用于实现开中断，关中断，访问 mEPC 寄存器。并且相对单级中断，多重中断中 mEPC 寄存器也必须作为现场进行保护，如采用内存堆栈方式保护 mEPC 寄存器，则必须通过 csrrsi 指令读取 mEPC 寄存器的值到通用寄存器后压栈，恢复现场时从堆栈弹出 mEPC 寄存器的值，然后利用 csrrw 指令写入 CSR 中的 mEPC 寄存器；另外多重中断保护现场结束后应开中断，需要执行 csrrsi 指令设置 IE 使能位，恢复现场之前也需要关中断、同样需要执行 csrrci 指令设置 IE 使能位。

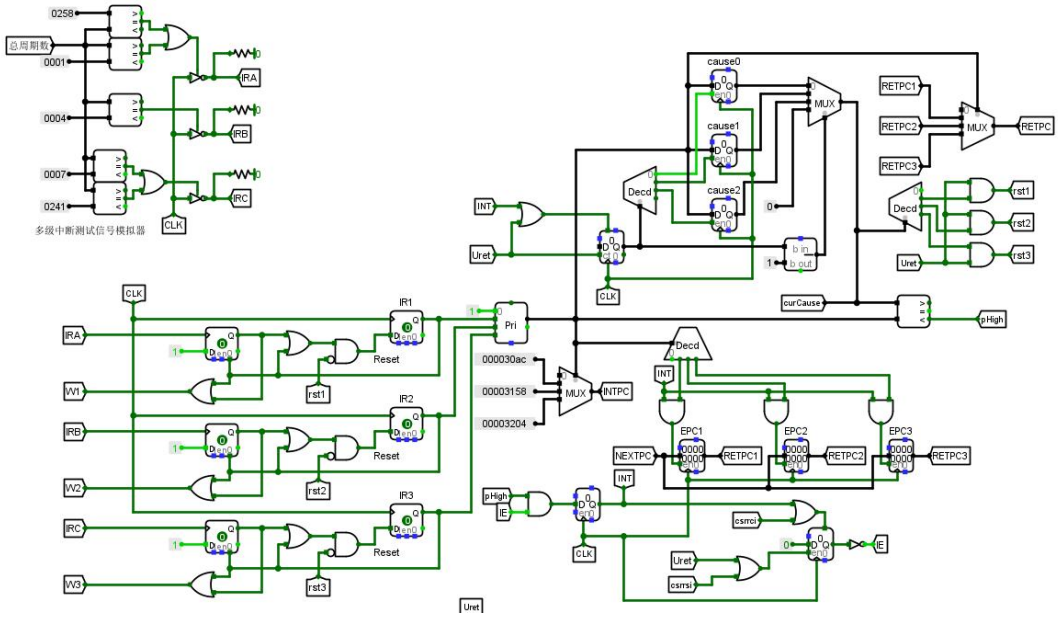

图 3.8 多极中断控制电路

3.2.3 流水线中断

根据在 EX 段来处理中断的设计，流水中断的控制部件如下。

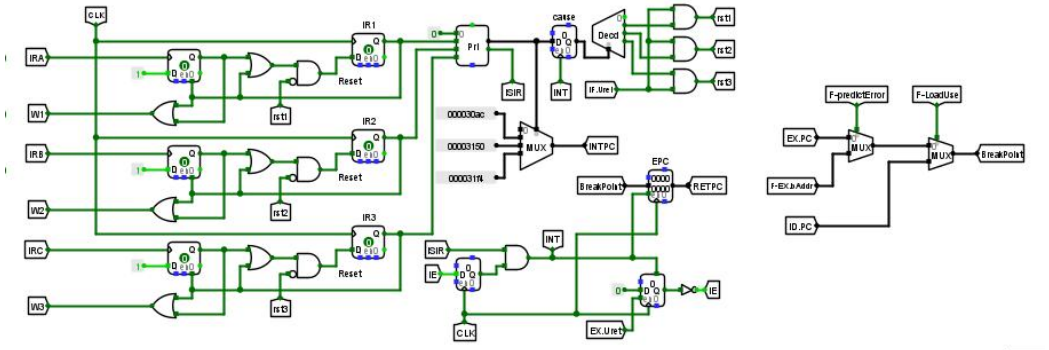

图 3.9 单极中断控制电路

3.3 理想流水 CPU 实现

3.3.1 理想流水接口部件实现

1）理想流水线寄存器

首先设计最为复杂的 ID/EX 流水寄存器，封装设计完成后再复制修改生成 IF/ID、EX/MEM、MEM/WB 流水寄存器。

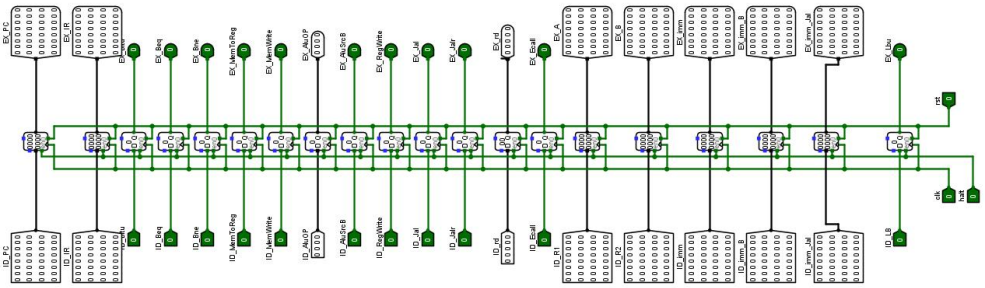

图 3.10 ID/EX 流水寄存器电路

条件分支逻辑设计

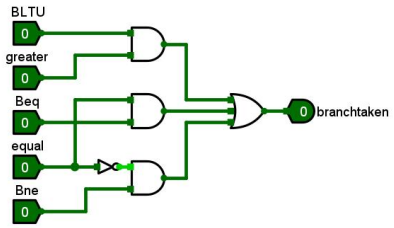

图 3.11 条件分支逻辑设计电路

无条件跳转逻辑设计

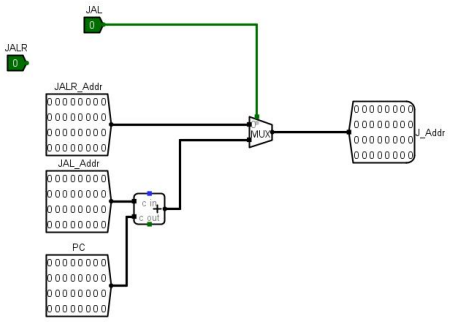

图 3.12 无条件跳转逻辑设计电路

流水线 pc 计算

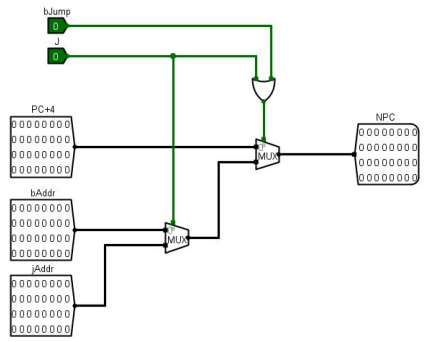

图 3.13 流水线 pc 电路

3.3.2 理想流水线实现

将单周期 CPU 的数据通路全部删除，重新进行电路布局，将各段功能部件与流水寄存器进行连接，将各流水功能段重要监控点如 PC 值、指令字、控制信号按需连接到引脚区的隧道标签。

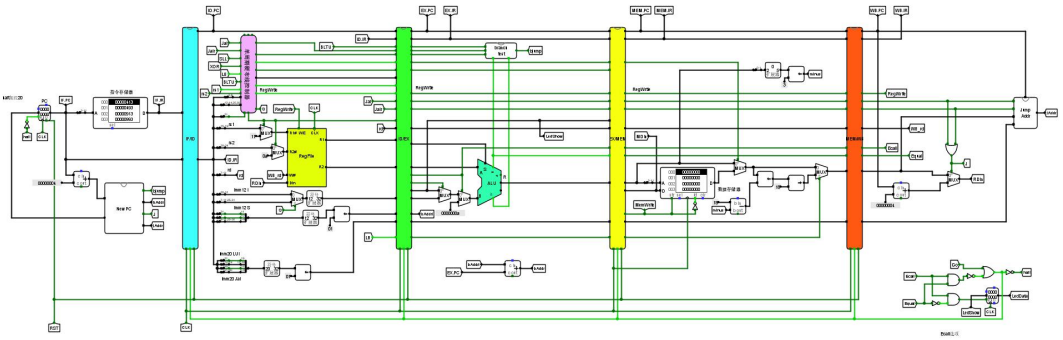

图 3.14 理想流水线数据通路

统计功能实现

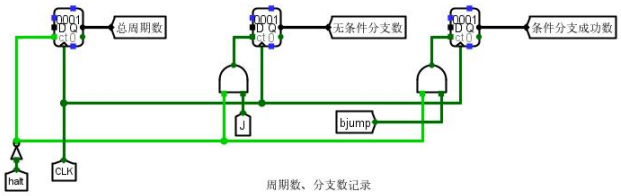

图 3.15 统计功能电路

3.4 气泡式流水线实现

3.4.1 气泡流水线接口部件实现

气泡 IR 扩展器

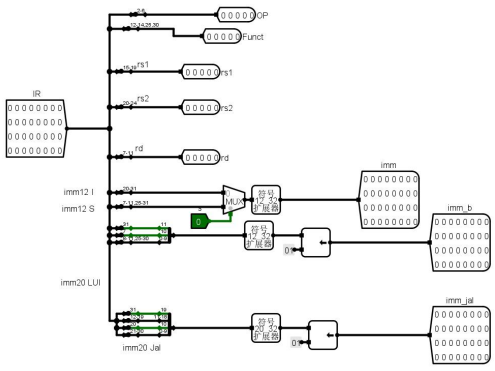

图 3.16 气泡 IR 扩展器电路

气泡流水线的流水寄存器 IF/ID、ID/EX、EX/MEM、MEM/WB 类似理想流水线。

气泡发生器部件如图 3.17 所示。

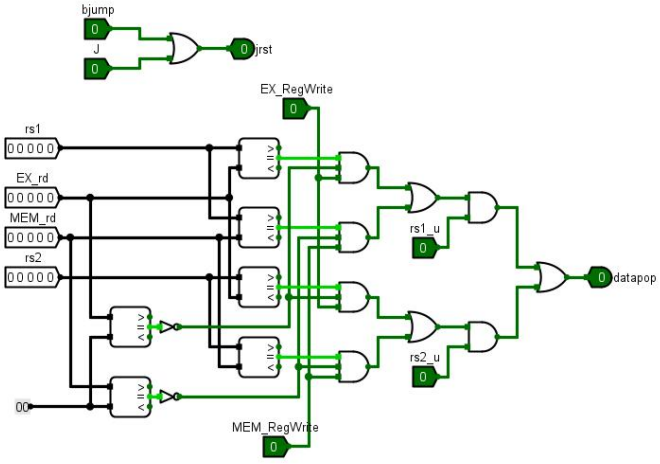

图 3.17 气泡发生器电路

3.5 重定向流水线实现

3.5.1 重定向流水线接口部件实现

重定向逻辑部件如图 3.18 所示。

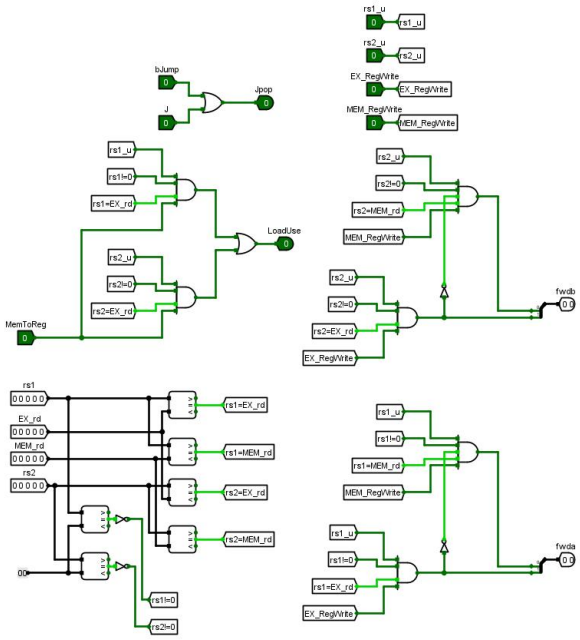

图 3.18 重定向逻辑设计电路

本实验将重定向逻辑放在 EX 段，将气泡流水线 ID/EX 部件改造后得重定向流

水线 ID/EX，如图 3.19 所示。

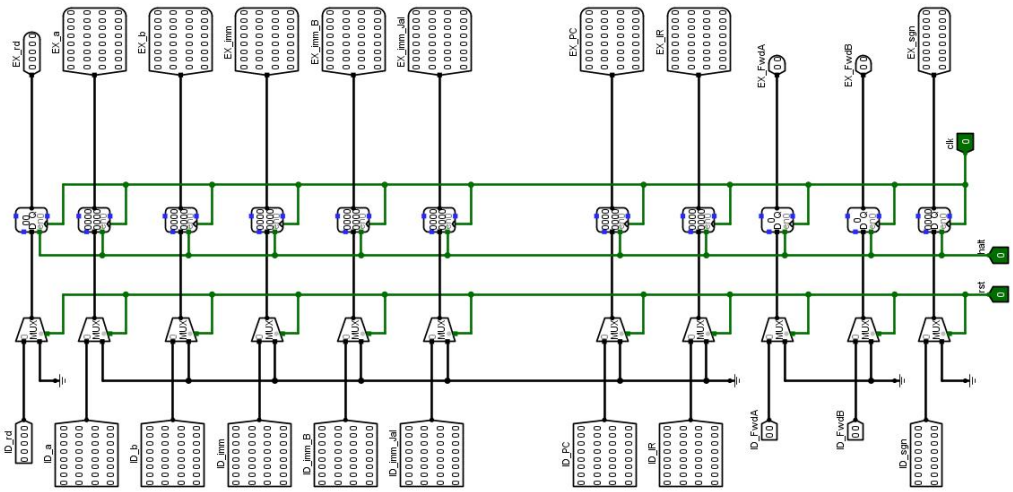

图 3.19 重定向 ID/EX 电路

3.6 动态分支预测机制实现

3.6.1 动态分支预测接口部件实现

当分支指令到达译码段（ID）时，根据从 BHT 读出的信息进行分支预测 。若预测正确，就继续处理后续的指令，流水线没有断流。否则，就要作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。BTB 根据运行状况，修改 BHT 的状态。BTB 部件如图 3.20 所示。

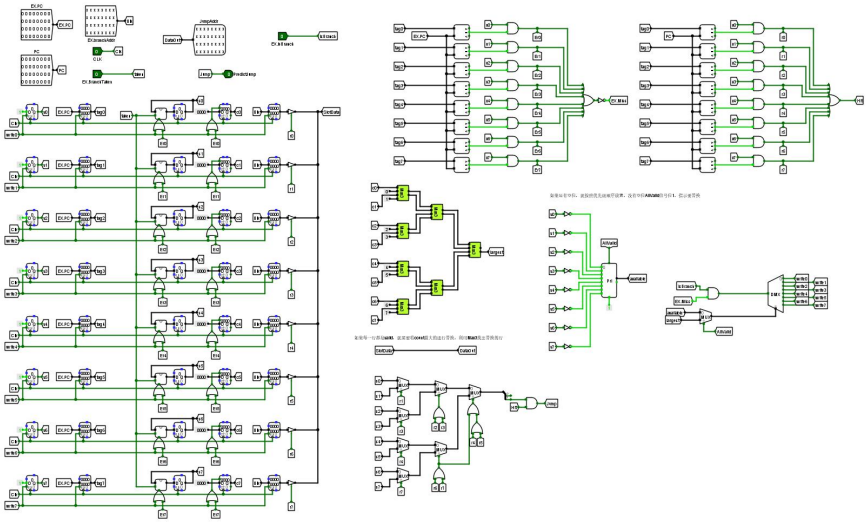

图 3.20 BTB 部件电路

动态分支预测 PC 计算部件电路如图 3.21 所示。

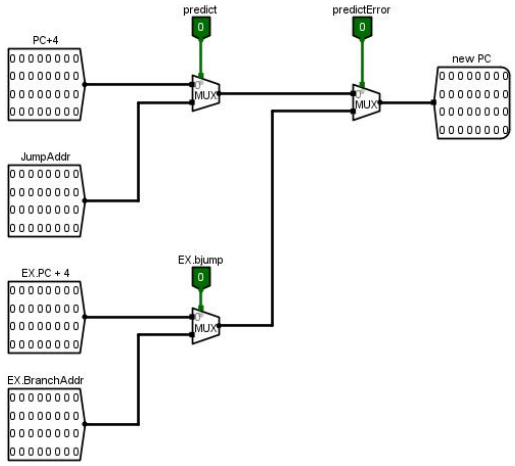

图 3.21 动态分钟预测 PC 计算部件电路

3.6.2 动态分支预测硬件实现

将上述部件（PC 计算部件、BTB 部件）增加在重定向流水线上可以得到：

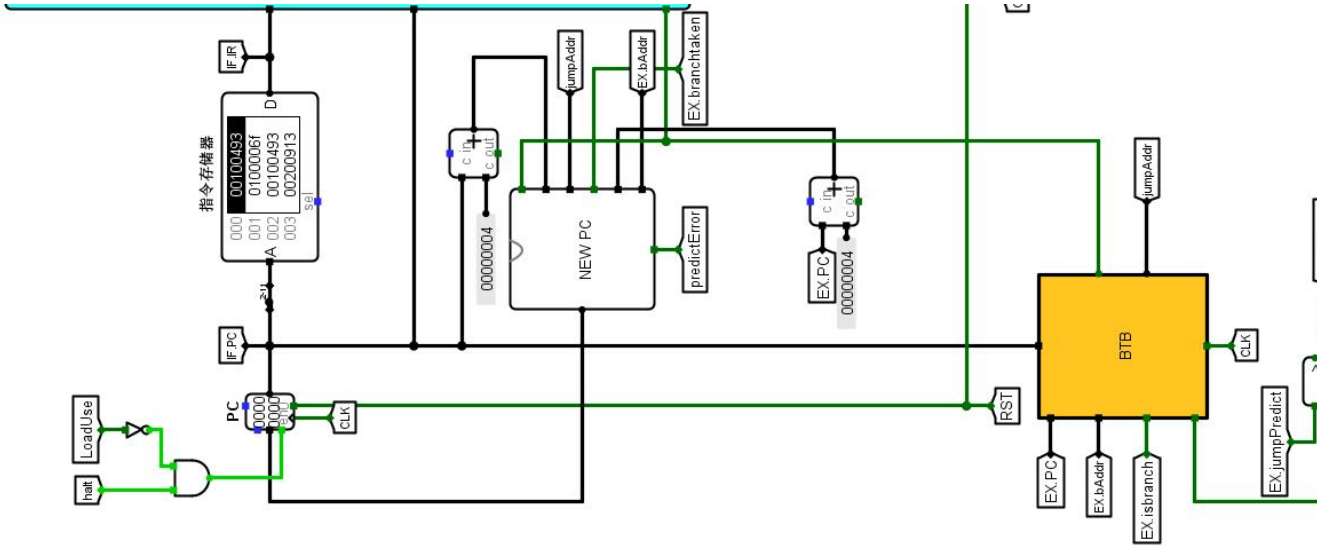

图 3.22 动态分支预测硬件实现

4 实验过程与调试

4.1 测试用例和功能测试

对各个任务分别单独测试，除多级中断要执行中断服务程序外，其他任务都用 benchmark 标准程序进行测试。在 benchmark 程序的最后还添加了四条扩展指令的测试程序，每执行一条暂停一次，手动按键 go 继续执行。

4.1.1 单周期 cpu

① Logisim 测试

用 benchmark_ccab 标准程序在 logisim 上进行测试，测试结果均符合预期。

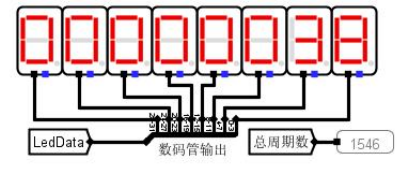

图 4.1 单周期 logisim 测试

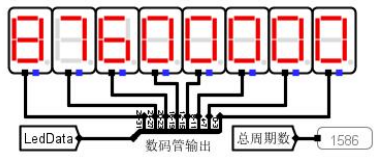

图 4.2 sll 指令测试

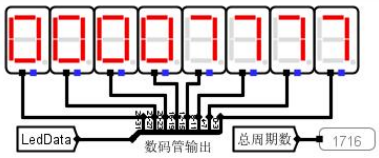

图 4.3 xor 指令测试

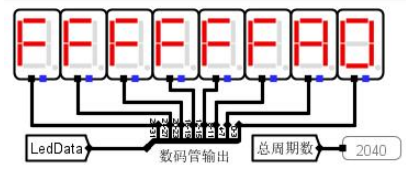

图 4.4 lb 指令测试

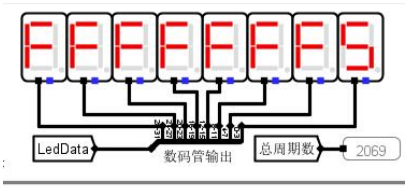

图 4.5 bltu 指令测试

② FPGA 测试

在开发板上运行标准测试程序，结果如图 4.6 所示，ccab 结果亦与图 4.2-5 相同，均符合预期。

图 4.6 单周期 cpu 上板结果

4.1.2 单级中断

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

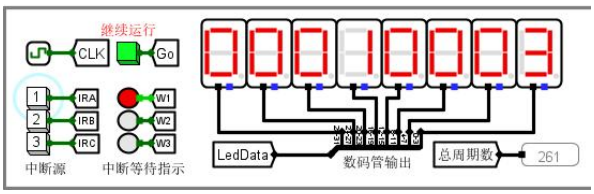

图 4.7 单级中断测试结果

4.1.3 多级中断

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

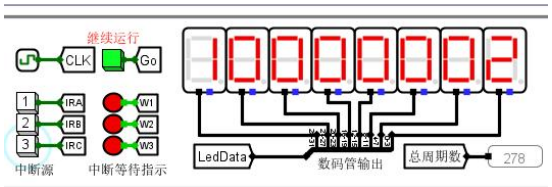

图 4.8 多级中断测试结果

4.1.4 流水线中断

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

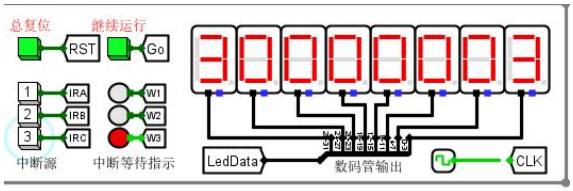

图 4.9 流水线中断测试结果

4.2 流水线测试

4.2.1 理想流水线

加载“理想流水线测试.hex”，运行参数如下所示，总周期数为 21，符合预期。

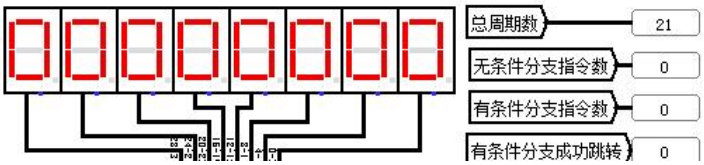

图 4.10 理想流水线测试结果

4.2.2 气泡流水线

加载“benchmark+ccab.hex”，运行结果如下，符合预期。

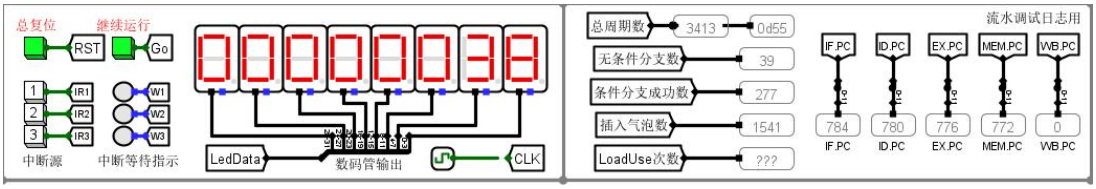

图 4.11 气泡流水线测试结果

4.2.3 重定向流水线

加载“benchmark+ccab.hex”，运行结果如下，符合预期。

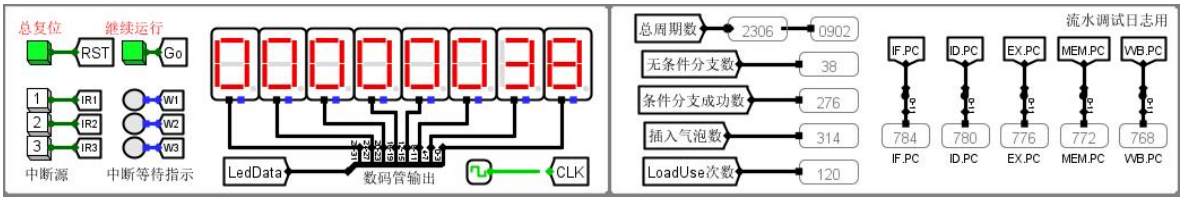

图 4.12 重定向流水线测试结果

4.2.4 动态分支预测

加载“benchmark+ccab.hex”，运行结果如下。

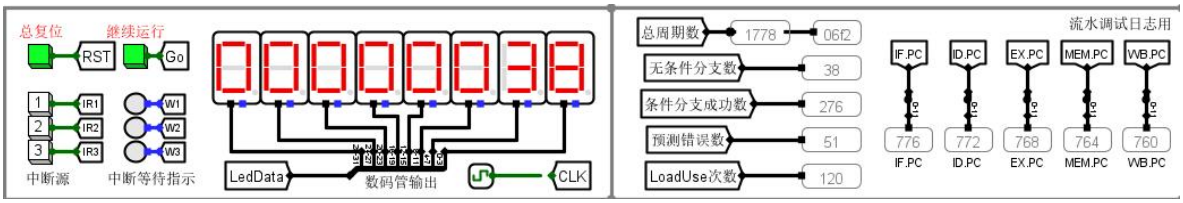

图 4.13 动态分支预测测试结果

可见，总周期为 1778，与基本的重定向流水的 2306 相比，性能提升不少。

4.3 性能分析

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

表 4.1 各方案基准测试周期表

方案	CPU24+4	气泡流水线	重定向流水线	动态分支预测
周期数	1546	3413	2306	1778

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

气泡流水线因为数据相关和分支相关插入了大量的气泡，所以它的周期数是最多的；重定向流水线在气泡流水线的基础上改进了对数据相关的处理，用旁路技术代替插入气泡，一定程度提高了流水的性能，但是分支相关和 Load_Use 相关还是要插入气泡；在重定向流水线的基础上加入动态分支预测技术，进一步优化了对分支相关的处理，使得流水周期减少了 528 个时钟周期，流水性能得到很明显的提升。

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

4.4 主要故障与调试

4.4.1 ccab 指令执行故障

故障现象：执行 bltu 指令时报错或不能正确输出。

原因分析：如图 4.14，ALU 输出除了 Result 部分还有“等于”和“大于等于”的输出被使用，而 bltu 指令使用的输出实际上是“小于”，当“小于”输出 1 时说明执行 bltu 指令。

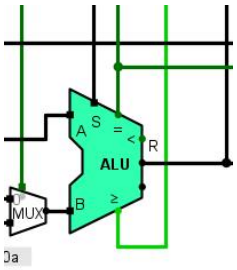

图 4.14 ALU 输出使用

解决方案：将“大于等于”改为使用“小于”输出。

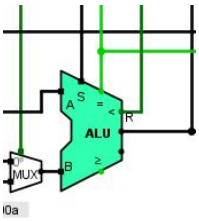

图 4.15 ALU 输出修改

4.4.2 ROM 故障

故障现象：在流水线执行过程中的输出不符合预期或者停不下来。

原因分析：benchmark+ccab.hex 里面的代码顺序不对以及没有正确写入暂停程序代码。

解决方案：严格按照注释的提示填入 ccab 的测试代码，并在适当的地方写入暂停代码。

4.5 实验进度

表 4.2 课程设计进度表

时间	进度

时间	进度
第 1-2 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 3-4 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 4-5 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。使用 Verilog 实现了部分单周期 CPU 的重要部件，并通过仿真检查。
第 5-6 天	继续使用 Verilog 进行实现单周期 CPU 的工作，完成了所有部件的编写、控制器的编写，以及所有部件以及控制器的仿真测试，正在进行数据通路的拼接。
第 7-8 天	使用 Verilog 完成单周期 CPU 数据通路的连接，并且通过仿真测试。使用 Verilog 完成时钟分频以及七段数码管的代码编写，正在调试。
第 9-10 天	完成 CPU 电路的功能仿真和时序仿真，并成功将生成 bit 流烧入 FPGA 板内实现预计功能。
第 11 天	复习关于指令流水线的知识点，完成理想流水线的 verilog 代码，正在调试。
第 12 天	调试成功理想流水线 verilog 代码，并成功将 bit 流烧至 FPGA 板中。完成冒险处理中的数据冲突处理和分支处理代码编写，正在调试。
第 13 天	完成冒险处理中的数据冲突和分支处理，并成功烧入 FPGA 板内。完成数据重定向的 Verilog 代码的编写，正在调试。
第 14 天	检查各项任务的最终成功是否符合预期。

5 设计总结与心得

5.1 课设总结

本次课程设计主要作了如下几点工作：

1） 完成了 Risc-v 的 24+4 条基本指令的硬件实现。

2） 对单周期 cpu 进行了合理的划分，得到了理想流水线；对理想流水线加入冲突处理逻辑，实现了气泡流水线；对气泡流水线进行优化，实现了重定向流水线；对重定向流水线加入 bht，实现了动态分支预测。

3） 完成了 logisim 到 Verilog 的转换，在 FPGA 上实现了单周期 cpu。

5.2 课设心得

花了两个多星期才完成的硬件综合课程设计让我收获良多。

由于起步比别人稍晚一两天的我开始有些急躁，急于去按照课件上建议的任务完成的时限完成每个任务，但越急躁越难以成事，所以一开始我的节奏是很慢的。后来逐渐理解还掌握了这些任务之间的一些联系和调 bug 的方法后，渐渐形成了一种，每天能完成一个任务的绝大部分，留下的一些 bug 只有在第二天才能完全排除的工作模式。掌握了自己的工作节奏后按时完成了任务。

在第一次检查没有排上队的情况下，本来只完成了 logisim 部分的我决定在第二次检查之前挑战自己完成上板的工作，期间遇到了很多困难，也得到了同学和老师的帮助。最终额外完成了上板的任务。

然而对于本次课程设计，我还有一些小小的建议和改进。希望 educoder 上的任务在前期可以分细一点，有助于让学生测试和检验自己的电路是否正确。

最后在这里也感谢老师们对于我在本次课程设计中无数问题的耐心解答，也感谢同学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：张宁静