华中科技大学

2019

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：	IT1601
学　　号：	U201614898
姓　　名：	潘翔
电　　话：	18086010495
邮　　件：	hover@hust.edu.cn

目 录

1 课程设计概述.....3

1.1 课设目的.....3
1.2 设计任务.....3
1.3 设计要求.....3
1.4 技术指标.....4

2 总体方案设计.....6

2.1 单周期 CPU 设计.....6
2.2 中断机制设计.....13
2.3 流水 CPU 设计.....14
2.4 气泡式流水线设计.....16
2.5 重定向流水线设计.....17
2.6 动态分支预测机制.....18

3 详细设计与实现.....20

3.1 单周期 CPU 实现.....20
3.2 中断机制实现.....27
3.3 流水 CPU 实现.....30
3.4 气泡式流水线实现.....31
3.5 重定向流水线实现.....32
3.6 动态分支预测机制实现.....35

4 实验过程与调试.....38

4.1 测试用例和功能测试.....38
4.2 性能分析.....41
4.3 主要故障与调试.....43
4.4 实验进度.....45

5 设计总结与心得.....47

5.1 课设总结.....47

5.2 课设心得.....47

参考文献.....49

1 课程设计概述

1.1 课设目的

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

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

1.2 设计任务

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

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

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

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10） 支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

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

（13） 能运行教师提供的标准测试程序，并自动统计执行周期数

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

表 1.1 指令集

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

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

2 总体方案设计

2.1 单周期 CPU 设计

在单周期的 CPU 设计中，我们采用的方案是微程序控制，且主、控存分开的方案，即采用微程序控制方式，实现主存储器（MM）和微程序控制存储器（CM）不共用一个存储器的方式完成方案的设计。同时在实施的过程中，采用部分电路用 FPGA 方式下载、部分电路用硬件搭建的方式完成。实现过程采用工程化设计思路，整体设计数据通路，然后进行模块细化。

总体结构图如图 2.1 所示。

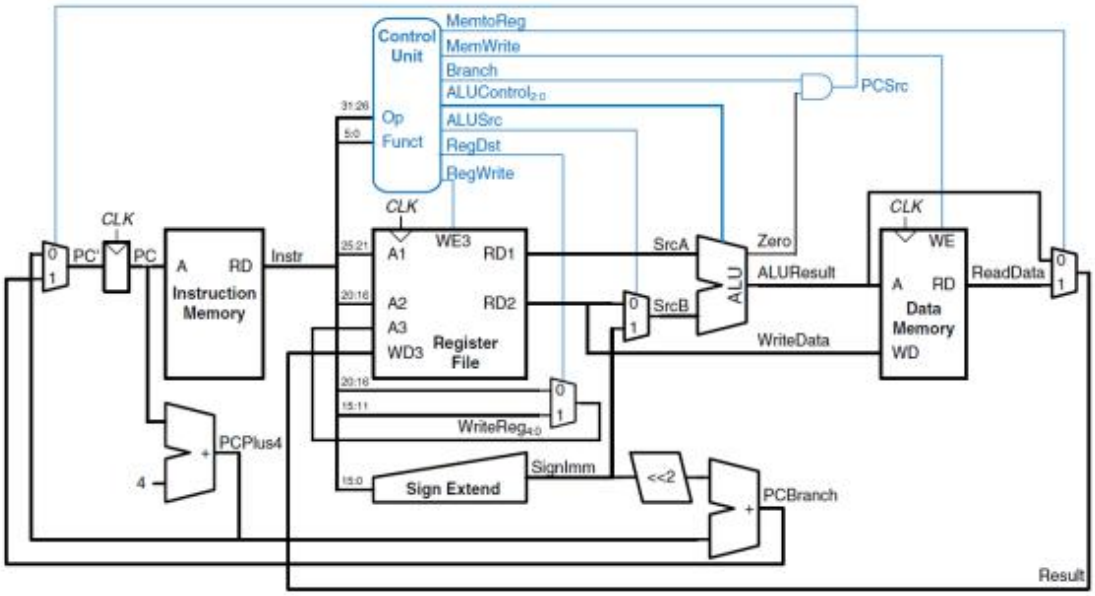

图 2.1 总体结构图

2.1.1 主要功能部件

主要用到的功能部件有：程序计数器 PC、指令存储器 IM、运算器、寄存器堆 RF、数据存储器 DM 以及符号扩展模块及数据选择器等。下面对主要功能部件具体介绍：

1. 程序计数器 PC

采用时钟上升沿触发的寄存器来作为程序计数器 PC，此处未使用 counter 而采用寄存器是因为可能出现跳转指令，方便修改其值。实现以下功能：

1) 计数：时钟上升时，将输入端锁存的 32 位 PC 值传输到输出端，实现每个时钟周期 PC 值的自动更新。
2) 使能：当使能端为 0 时忽略时钟信号，锁存输出值，使能端连接停机信号来实现运行到目标指令时的停机功能。
3) 复位：连接 RST 手动复位信号方便调试

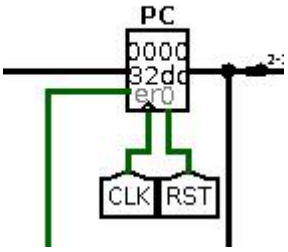

图 2.2 程序计数器 PC 结构图

2. 指令存储器 IM

指令存储器 IM 采用Logisim 默认 ROM，实现下述功能：

1) 指令装载：CPU 运行之前将对应的指令集手动加载到该 ROM 中
2) 指令取出：取出 Addr 地址的指令输出到 Dout
3) 使能：由 PC 端的锁存进行使能实现

图 2.3 指令存储器 IM 结构图

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
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

在实现过程中，需要使用对应于 CS341ALU 的相应 ALUOP，列表如下

表 2.2 ALU_OP 功能对应表

ALU_OP	十进制	运算功能
0000	0	Result = X << Y 逻辑左移 （Y 取低五位） Result2=0
0001	1	Result = X >>>Y 算术右移 （Y 取低五位） Result2=0
0010	2	Result = X >> Y 逻辑右移 （Y 取低五位） Result2=0
0011	3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 无符号乘法
0100	4	Result = X/Y; Result2 = X%Y 无符号除法
0101	5	Result = X + Y (Set OF/UOF)
0110	6	Result = X - Y (Set OF/UOF)
0111	7	Result = X & Y 按位与
1000	8	Result = X | Y 按位或
1001	9	Result = X⊕Y 按位异或
1010	10	Result = ~(X |Y) 按位或非
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

4. 寄存器堆 RF

使用已经实现的二次封装 MIPS Regfile 作为寄存器堆 RF 的实现，为双路的寄存器。

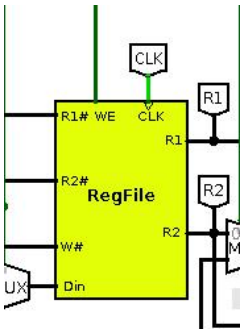

图 2.4 寄存器堆 RF 结构图

2.1.2 数据通路的设计

每条的指令各个模块的相连情况如 表 2.3 指令系统数据通路框架 所示。其中 EXT 表示根据 IM 进行不同扩展得到的数据结果或 PC 值。各条指令对应的表格项表示与该模块连接的模块或引脚名。

为了便于说明，采用下面命名规则：

驼峰+下划线命名：

1) 器件内部首字母大写其相连，如：AluOP
2) 输入输出采用 in,Out，Result 进行区分如 Alu_Result
3) 不同流水段采用 流水段.接口命名 如 DM.PC

以下章节采用相同命名规范，不再赘述

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
预先载入 IM 中	与 IM,DM 相连，解析后作为地址	PC 输出相连	Rs	Rt	Rd	DM_Out	R1	R2	ALUop	ALU_Result	

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2.4。

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	X	X∈[0000,0010] 表示运算功能
MemtoReg	0	选择 ALU 计算的结果存入到寄存器文件中
	1	选择从 DM 读取的数据存入到寄存器文件中
MemWrite	0	不对 DM 进行写入操作
	1	将 DM 部件 Din 端的数据写入到 DM 中
ALU_SRCB	0	选择 R2 作为 ALU 中 B 端数据的输入
	1	选择扩展后的立即数作为 ALU 中 B 端数据的输入
RegWrite	0	不对寄存器文件进行写操作
	1	将 Din 端的数据写入到寄存器文件里
SYSCALL	0	表示当前指令不是 syscall 指令
	1	表示当前指令是 syscall 指令
SignedExt	0	表示对立即数进行 0 扩展
	1	表示对立即数进行符号扩展
RegDst	0	表示选择 rt 作为 R2#
	1	表示选择 rd 作为 R2#
BEQ	0	表示当前指令不是 beq 指令
	1	表示当前指令是 beq 指令
BNE	0	表示当前指令不是 bne 指令
	1	表示当前指令是 bne 指令
JR	0	表示当前指令不是 jr 指令
	1	表示当前指令是 jr 指令

控制信号	取值	说明
JMP	0	表示当前指令不是无条件跳转指令
	1	表示当前指令是无条件跳转指令
SRLV	0	表示当前指令不是 SRLV 指令
	1	表示当前指令是 SRLV 指令
XORI	0	表示当前指令不是 XORI 指令
	1	表示当前指令是 XORI 指令
SH	0	表示当前指令不是 SH 指令
	1	表示当前指令是 SH 指令
BLEZ	0	表示当前指令不是 BLEZ 指令
	1	表示当前指令是 BLEZ 指令

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 **Error! Bookmark not defined.**所示。

表 2.5 主控制器控制信号框架

指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	R1_Used	R2_Used	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	Signed Ext	RegDst	BEQ	BNE	JR	JMP	JAL
SLL	0	0	0	1	1				1			1					
SRA	0	3	1	1	1				1			1					
SRL	0	2	2	1	1				1			1					
ADD	0	32	5	1	1				1			1					
ADDU	0	33	5	1	1				1			1					
SUB	0	34	6	1	1				1			1					
AND	0	36	7	1	1				1			1					
OR	0	37	8	1	1				1			1					
NOR	0	39	10	1	1				1			1					

SLT	0	42	11	1	1				1			1					
SLTU	0	43	12	1	1				1			1					
JR	0	8	X	1											1	1	
SYSCALL	0	12	X	1	1					1							
J	2	X	X													1	
JAL	3	X	X						1							1	1
BEQ	4	X	X	1	1								1				
BNE	5	X	X	1	1									1			
ADDI	8	X	5	1				1	1		1						
ANDI	12	X	7	1				1	1								
ADDIU	9	X	5	1				1	1		1						
SLTI	10	X	11	1				1	1		1						
ORI	13	X	8	1				1	1								
LW	35	X	5	1		1		1	1		1						
SW	43	X	5	1	1		1	1			1						
SRLV	0	6	2	1	1				1			1					
XORI	14	X	9	1	1			1	1								
SH	41	X	5		1		1	1			1						
BLEZ	6	X	11	1													

2.2 中断机制设计

2.2.1 总体设计

设计过程中，采用循序渐进的方式，先实现单级中断，然后在其上进行改装实现多级中断，最后加入流水线，便于每一部分的调试。

实现的中断为外中断，且存在优先级，其结构图如 图 2.5 中断响应结构图 所示。

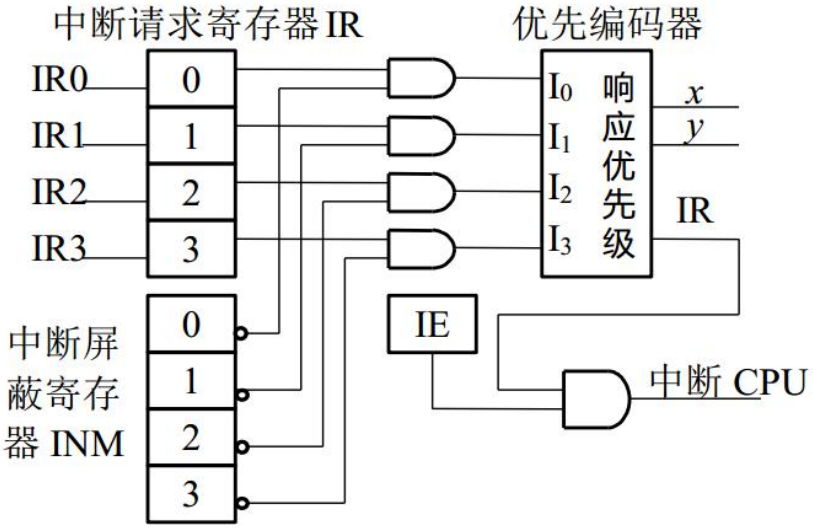

图 2.5 中断响应结构图

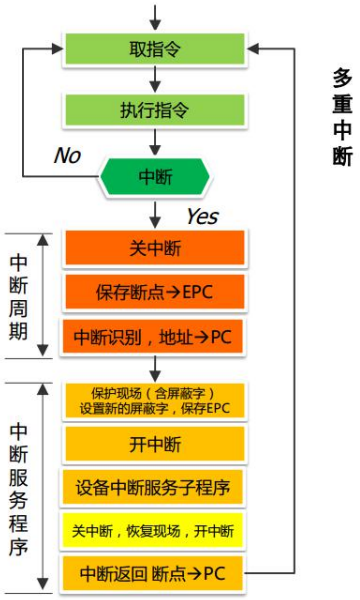

图 2.6 多级中断原理图

2.2.2 硬件设计

1) 保存断点：

中断程序即将执行的下一条指令的地址，也就是当前 PC 的值

2) 中断识别并跳转：

根据中断源找到中断，服务程序入口地址，修改 PC 值转到中断服务程序

2.2.3 软件设计

3) 利用堆栈保护现场

4) 执行中断服务程序

5) 恢复现场，完成中断返回前的准备工作

6) 中断返回

2.3 流水 CPU 设计

2.3.1 总体设计

MIPS 指令执行过程分为 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）。在其中插入流水接口部件完成相应信号量的锁存和传递，从而形成一定程度的隔离，使不同阶段能够处理不同的指令。

在处理流水误取上，预留原件 reset，便于后面对相应的流水进一步优化。

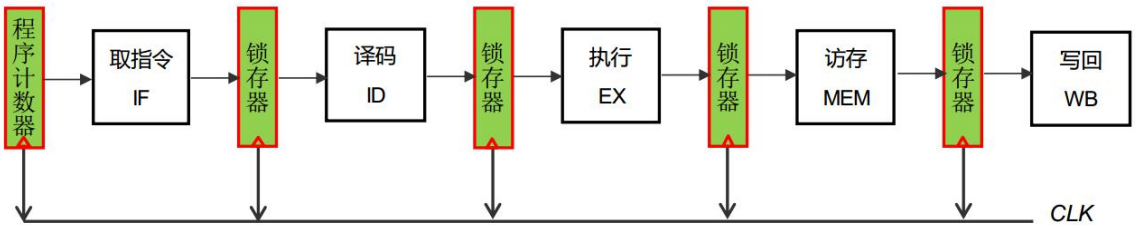

图 2.7 流水接口逻辑设计图

2.3.2 流水接口部件设计

采用流水信号逐步向后传递的设计，对于所需要的信号逐步向后，同时取出本阶段所需要的信号。

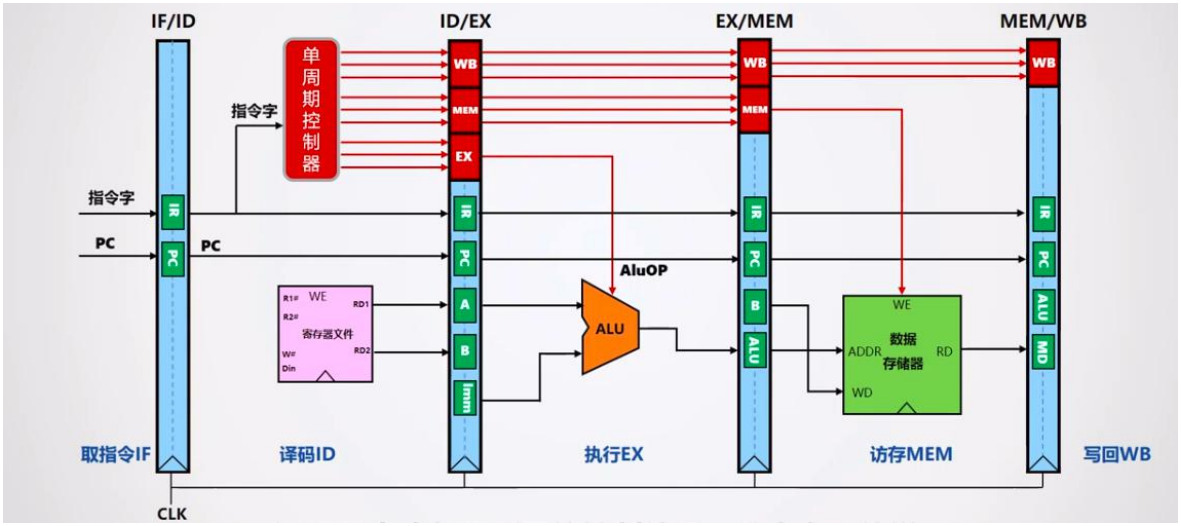

图 2.8 流水接口部件设计图

2.3.3 理想流水线设计

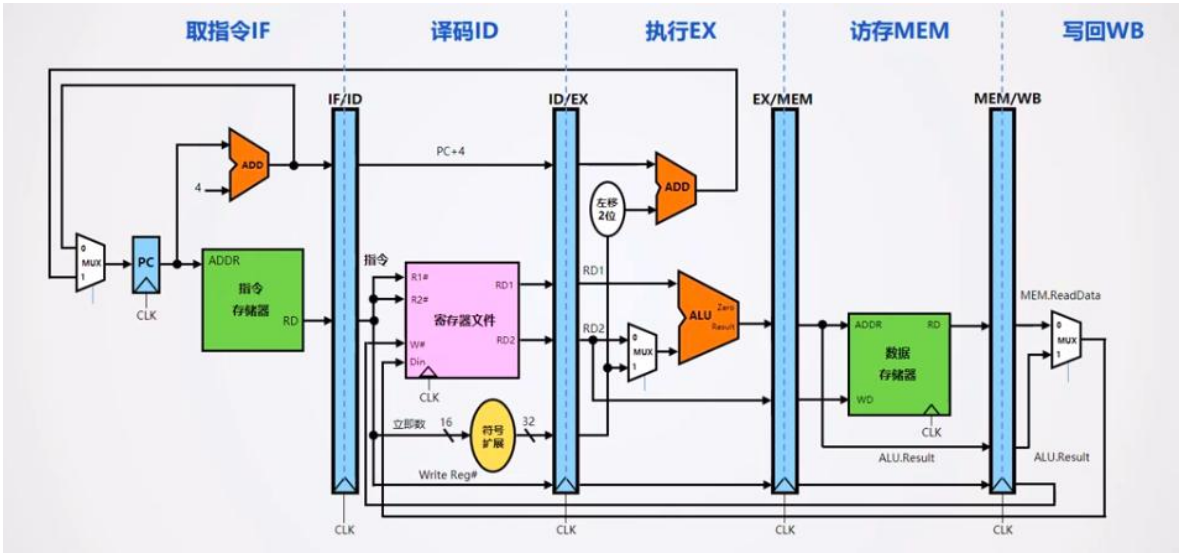

图 2.9 理想流水整体设计图

2.4 气泡式流水线设计

理想流水线假设指令间相互隔离且不存在相关性，此处对于实际应用场景，假设不成立，从而需要逐步处理各种相关。

气泡式流水线需要处理分支相关和数据相关。分支相关包括条件分支跳转和无条件分支跳转，此处将跳转指令放置于 EX 段执行，在 EX 阶段再进行跳转，并清除 IF 和 ID 阶段读入的错误指令，即插入两个气泡。

1) 数据相关检测

由于 ID 段需要取操作数，所以数据相关检测逻辑应设置在 ID 段。通过 ID 段、EX 段和 MEM 段的对应数据检测是否出现数据相关的情况。

2) 数据冲突处理

ID 段与 WB 段的数据相关可以通过寄存器先写后读的方式解决，具体方式是寄存器文件写入数据时采用下跳沿触发，由于寄存器的读出是组合逻辑， 数据写入后立刻可获得寄存器新值， 所以在下跳沿成功写入后，上跳沿触发流水线进行指令传送时，送入 ID/EX 的操作数已经是最新的寄存器值。

3) 插入气泡逻辑

出现数据相关时将在 EX 段插入气泡， IF、 ID 段暂停。ID 段与 EX 段存在数据相关，假设相关检测逻辑已经检测到数据相关， 应产生 stall（暂停） 控制信号， 阻塞 PC 以及 IF/ID 部件， 暂停 IF、 ID 段的工作以延缓取操作数的执行（可以通过控制程序计数器 PC 和 IF/ID 流水接口部件的使能端实现,这样时钟到来时锁存器的值不会改变，可能需要增加新的引脚），另外下一个时钟上跳沿到来时，需要向 EX 段插入一个气泡（空指令，MIPS 中空指令是全零的机器码，功能是 0 号寄存器左移零位送入 0 号寄存器），以避免 ID 段的指令向后传送。

2.5 重定向流水线设计

气泡流水线通过插入气泡来解决数据冲突，大量的气泡插入会影响性能。重定向流水线的原理是检测到数据发生冲突时，不考虑在 ID 取得的值是否正确，继续流水，在 EX 段需要使用这个值的时候，从前面阶段流水中获取正确的值进行重定向，从而避免插入气泡，提升了性能。

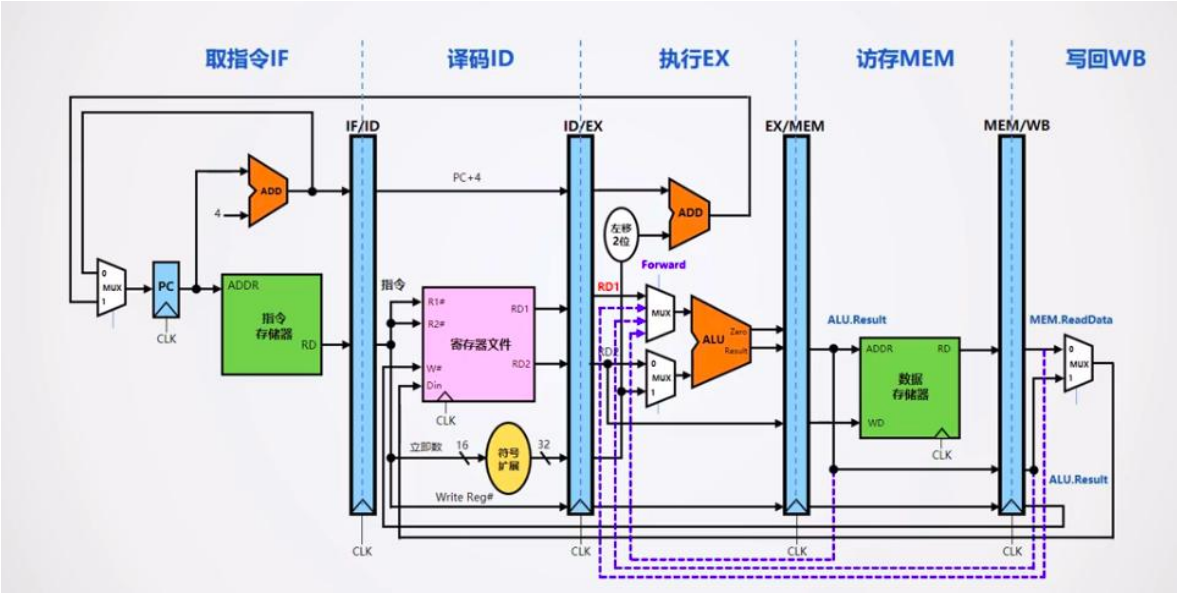

图 2.10　重定向流水整体设计图

1) 检测数据相关

检测到数据相关的时候，判断是否为 Load-Use 相关

a) 如果无数据相关，流水正常进行

b) 如果有数据相关，但不是 Load-Use 相关，在 EX 进行 Forward 信号进行选择

c) 如果有数据相关，且是 Load-Use 相关

2) 生成数据选择信号 Forward，进行数据通路的选择。

EX 段数据来源：MEM 阶段、WB 阶段。

MEM 段数据来源：ALU 的结果、立即数扩展、访存得到的结果(Load-Use 相关)。

WB 阶段转发的数据来源：ALU 的运算结果、访存结果、PC+4 和立即数扩展。在 MEM 阶段经过选择，在 WB 阶段经信号 MemTorReg 选择后的结果作为 WB 段转发的数据。

2.6 动态分支预测机制

采用重定向机制后，指令流水线中数据相关基本不需要插入气泡就可解决，只有少数 Load-Use 冲突需要插入一个气泡解决冲突问题，指令流水线性能得到较好的提升。此时指令流水线中结构冲突（分支冒险）对流水线性能影响很大，分支指令会使指令流水线暂停， 因为分支使得 IF 段以及 ID 段执行的指令（误取指令）被清空， 这部分流水线性能损失称为分支延迟。指令误取深度越长，分支延迟越大。为减少分支延迟，应在流水线中尽早判断出分支是否成功跳转，并尽早计算出分支目标地址，比如将分支指令放在 ID 段完成，这样误取深度就是 1，分支延迟也只有一个时钟周期。

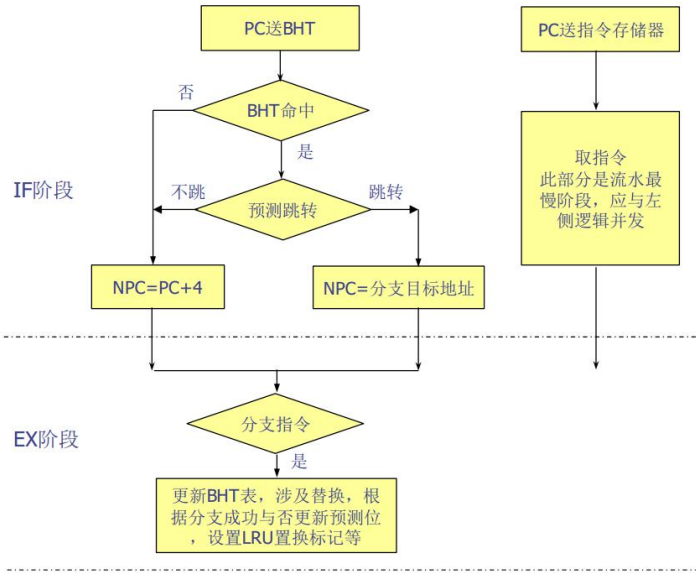

图 2. 11 动态分支预测流程图

2.6.1 BHT 设计

分支预测分支历史表 (Branch History Table，BHT)，BHT 表中存放分支指令的地址， 分支目标地址，历史跳转信息描述位（分支预测历史位），其中 valid 位表示对应表项是否有效，如果不设置 valid 位， 无法区分哪些表项是有效数据，比如初始化时表格全零，也会被当做是 0 号地址是分支指令，分支目标地址是 0。

采用双预测位进行 BHT 的状态转换，如 图 2.12 动态分支预测状态图 所示

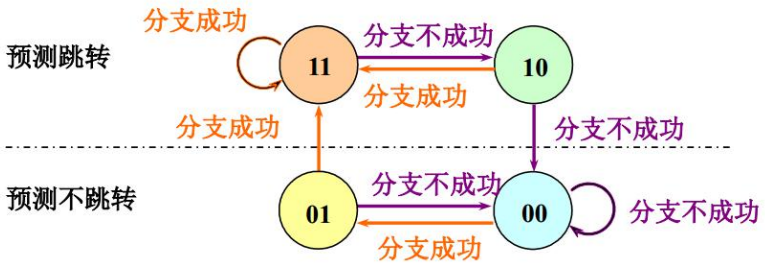

图 2.12 动态分支预测状态图

2.6.2 预测统计

加入预测统计模块，进行预测相关的状态统计，从而分析相关跳转状态。

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。

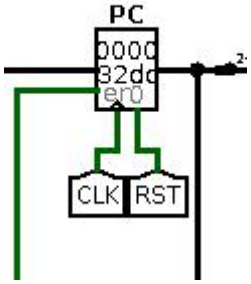

图 3.1 程序计数器（PC）

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

```
module pc(clk,rst,enable,pcout,pc,pc4);
    input clk,rst,enable;
    input [31:0] pcout;
    output reg[31:0] pc;
    output [31:0] pc4;
    assign pc4 = pc + 4;
    initial begin
        pc <= 0;
```

```
        end
        always@(posedge clk) begin
            if(rst) begin
                pc <= 0;
            end
            else begin
                if(0==enable)begin
                    pc <= pc;
                end
                else begin
                    pc <= pcout;
                end
            end
        end
    endmodule
```

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。

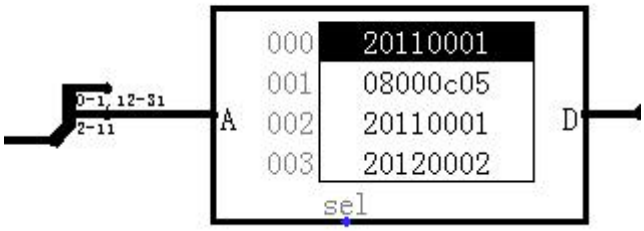

图 3.2 指令存储器（IM）

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 **Error! Bookmark not defined.**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：

```
module IM(pc,instru);
    input [31:0] pc;
    output [31:0] instru;
    reg [31:0] memory [0:1023];
    wire [9:0] pc1;
    initial begin

$readmemh("/home/hover/Desktop/Labs/HUST_ComputerOrginazation_CourseDesign/Verilog_Test/ccmb_.hex",memory);
    end
    assign pc1[9:0] = pc[11:2];
    assign instru = memory[pc1][31:0];
endmodule
```

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。同时为了方便加载指令，直接读取本地文件进行载入。

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			
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，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。

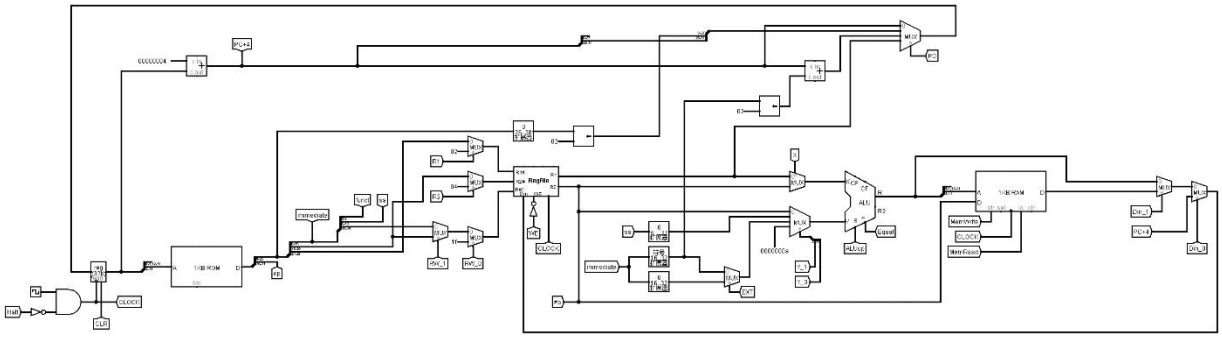

图 3.3 单周期 CPU 数据通路（Logism）

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3.4 所示。

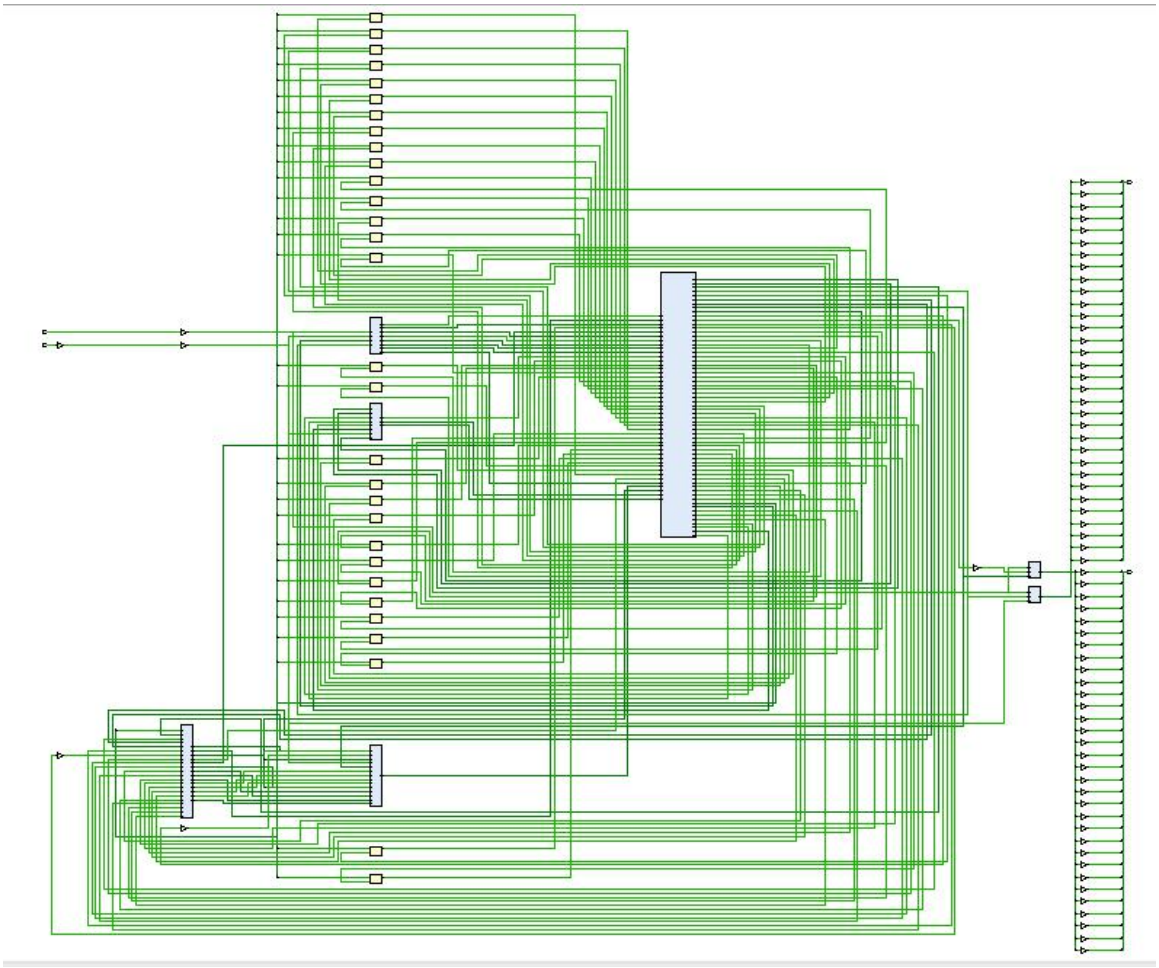

图 3.4 单周期 CPU 数据通路（FPGA）

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、SYSCALL 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。

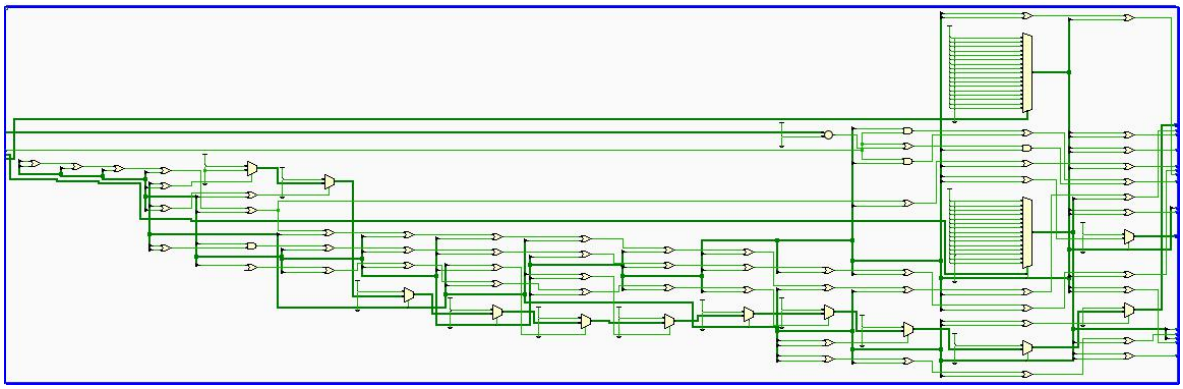

图 3.5 主控制器原理图

3.2 中断机制实现

3.2.1 单级中断实现

1) 中断按键信号产生

采用提供的中断按键信号产生电路进行封装

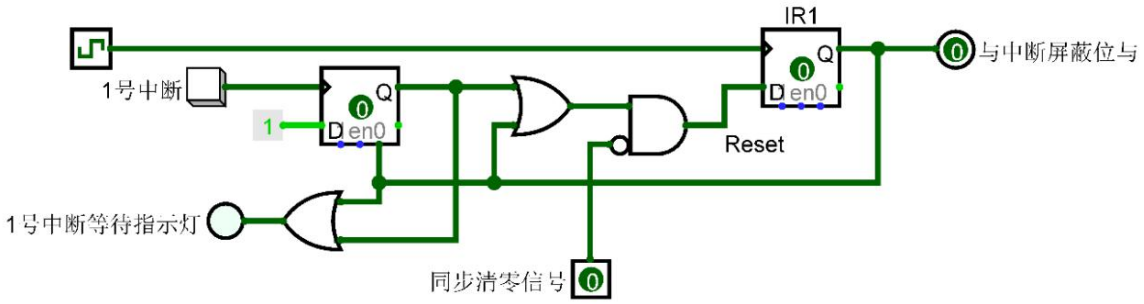

图 3.6 中断按键信号电路图

2) 中断识别逻辑

实现中断响应优先级，使用硬向量实现中断向量，此处存出中断程序入口地址，如果发生中断，根据优先编码器进行选择，然后进行下一个中断程序入口地址的载入。

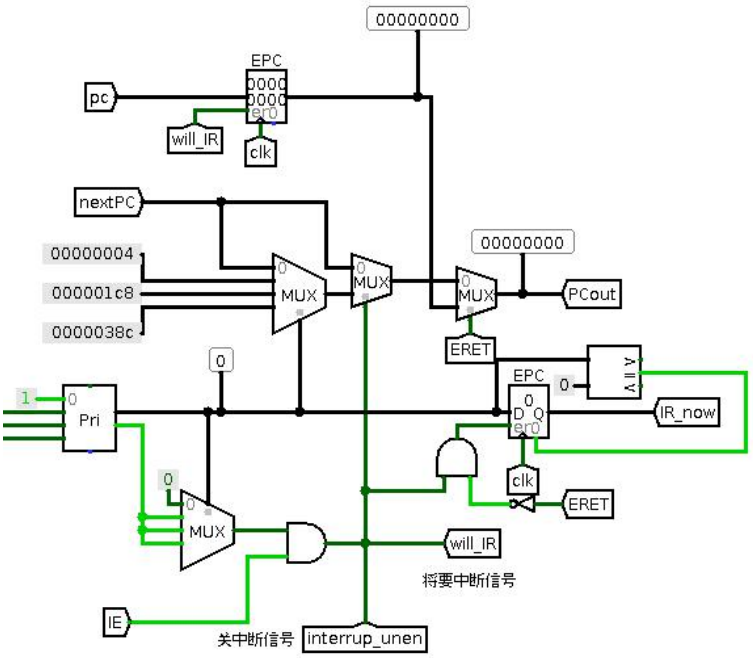

图 3.7 中断识别逻辑电路图

3) 存储区间规划

4) 实现 CP0 寄存器组中与中断相关的寄存器， 包括中断使能寄存器 IE， 异常程序计数器 EPC， 同时增加开关中断的硬件实现逻辑。

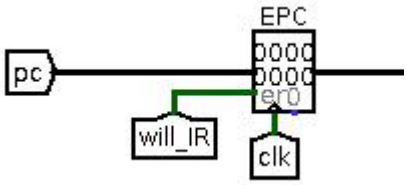

图 3.8 异常程序计数器 EPC 电路图

5) 设计中断隐指令数据通路， 包括保存断点至 EPC、中断识别，中断服务程序入口地址送 PC 等，中断隐指令部分可能需要多个时钟周期，这个过程中 CPU 应不能执行指令。

6) 增加中断返回指令 eret 的支持， 需要重新设计控制器，注意 eret 指令应该同时开中断。

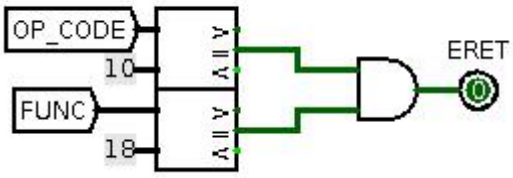

图 3.9 ERET 指令电路图

7) 编写中断服务程序， 在走马灯演示程序的基础上编写 3 个中断源对应的 3 个中断服务子程序，并合理放置在指令存储器中相应的位置。

3.2.2 多级中断实现

1) 中断开关

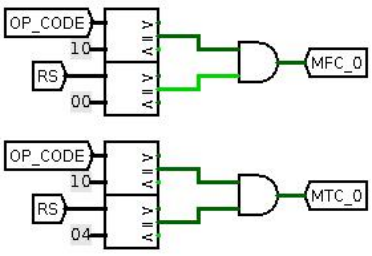

图 3.10 MFC0 和 MTC0 信号产生电路图

1) 中断计数器

寄存器 IE 保存中断嵌套次数进行中断计数，初始值为 0。当中断到来时候加 1，中断过程中维持此值，中断完成时减 1.

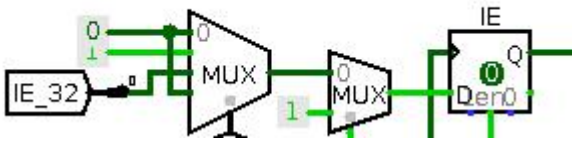

图 3.11 中断计数器电路图

2) 中断屏蔽字

根据不同的中断信号产生不同的中断屏蔽字

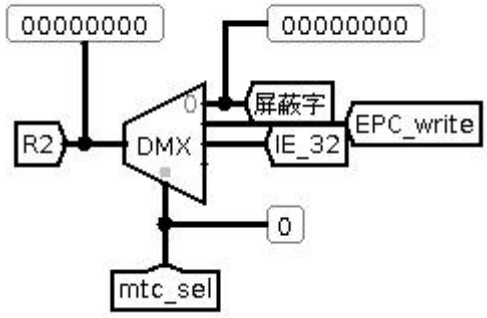

图 3.12 中断屏蔽字产生电路图

根据中断屏蔽字进行中断状态屏蔽

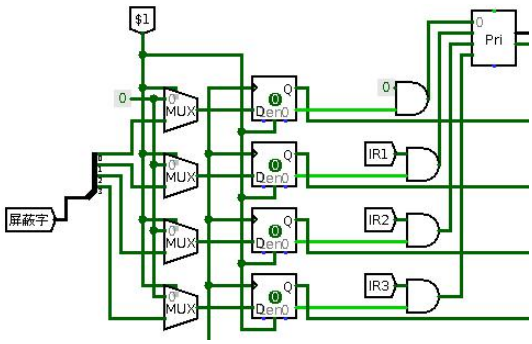

图 3.13 中断屏蔽字屏蔽电路图

3.2.3 流水中断实现

将 ID 段 ERET 信号输入单机中断进行判断，对下一步 PC 进行选择

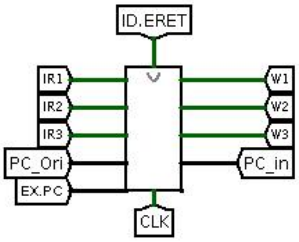

图 3. 14 流水中断电路模块图

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口为寄存器文件，为了方便布线，没有对与不同的传输信号产生单独的接口，而是使用统一的接口，出来后再进行分线器进行相应的操作。取出需要的信号，同时进一步传输，从而提高了模块的复用性。

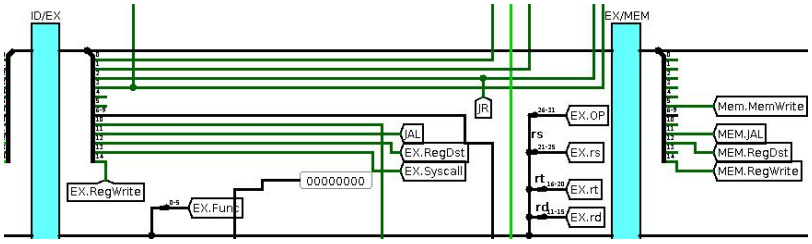

图 3. 15 流水中断电路模块图

流水接口内部采用统一布局，左端为输入，中间为寄存器和时钟及使能信号，右端为输出。

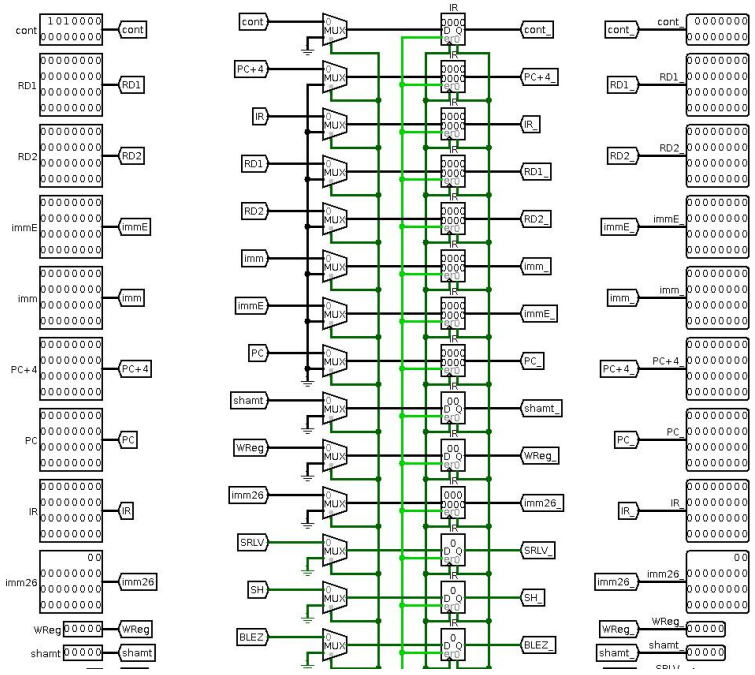

图 3.16 流水接口内部实现图

3.3.2 理想流水线实现

将上述流水接口加入单周期数据通路即生成理想流水线。

3.4 气泡式流水线实现

1) 整体实现

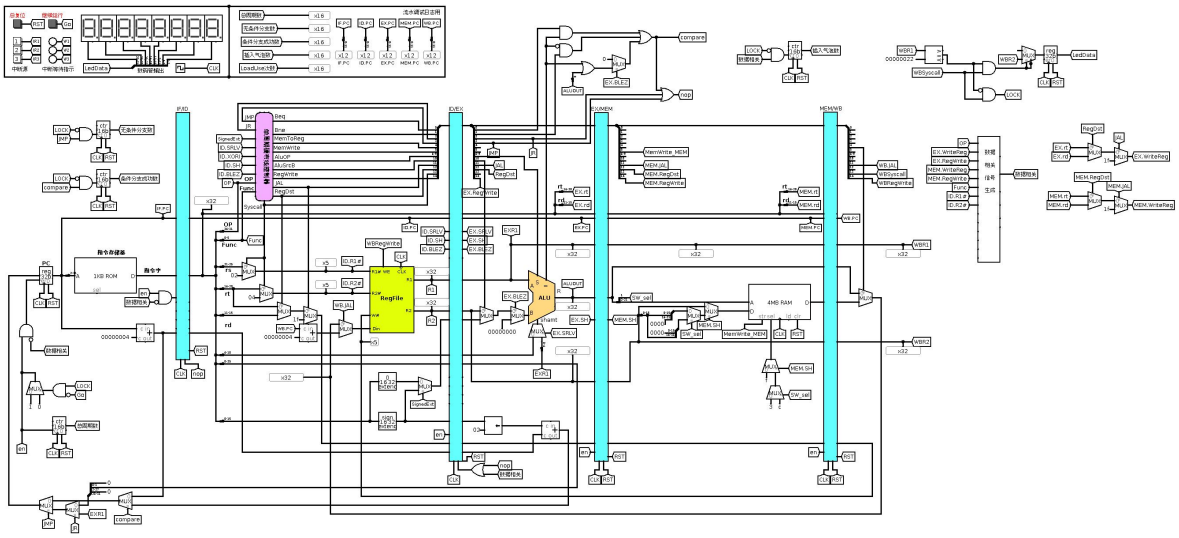

图 3.17 气泡流水线整体实现图

2) 数据相关检测

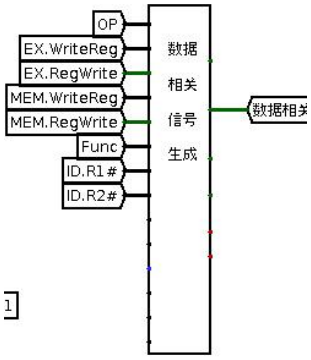

图 3.18 数据相关检测模块电路图

利用不同信号的对于 R1 和 R2 的使用情况进行相关使用信号的生成

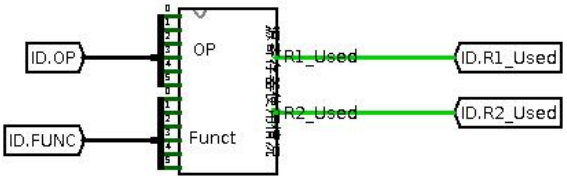

图 3.19 源寄存器使用模块电路图

根据不同的数据相关性生成不同的数据相关信号，便于之后处理

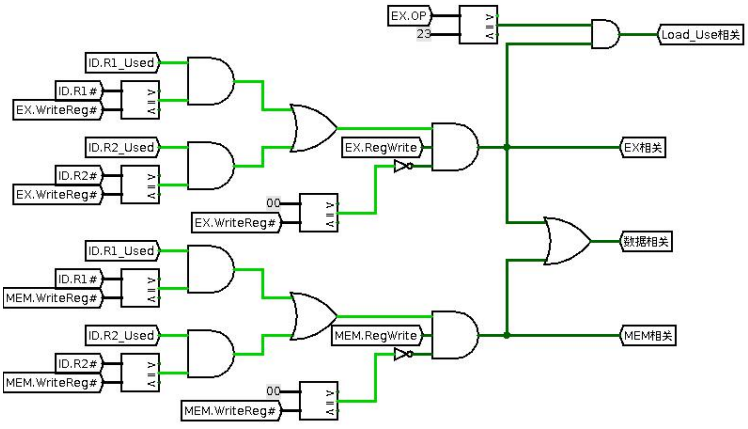

图 3.20 源寄存器使用模块电路图

3.5 重定向流水线实现

① Logisim 实现

重定向流水线主要是数据通路和 Forward 信号产生部分。

1) 重定向数据通路

利用 Forward 信号进行数据来源的选择

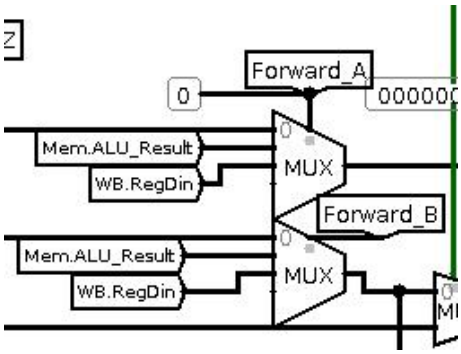

图 3.21 重定向数据通路图

2) Forward 信号产生

过滤掉 0 号寄存器后，根据 select 信号产生不同的 Forward 信号。判断 EX 段的 R1 和 R2 的寄存器编号是否与 MEM 段或 WB 段的写回的寄存器编号相等，如果相等就置 Forward 为 1 或 2，否则 Forward 为 0。

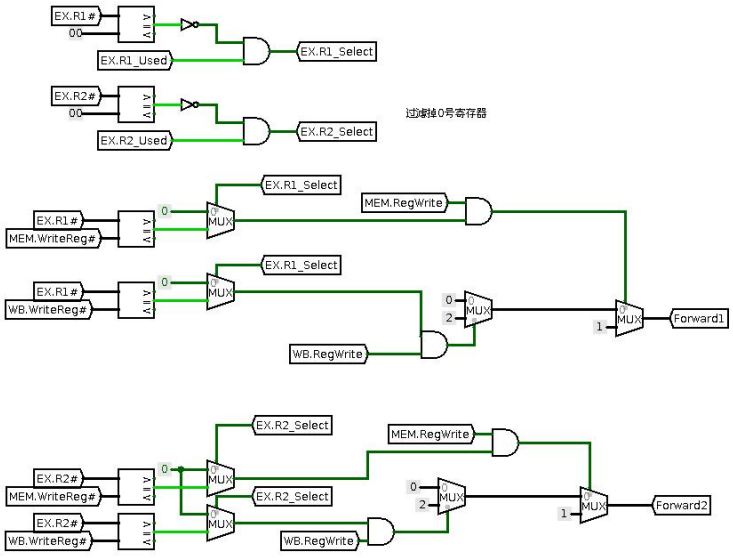

图 3.22 Forward 信号产生电路图

② FPGA 实现

在之前单周期的模块基础上，添加重定向模块对数据通路进行相应的封装，同时对 ALU、IM 和 DM 等进行复用，更改部分流水接口。

重定向模块：

```
module redirect(ra_r,rb_r,regWrite4,regWrite5,regDin4,jal4,ra,rb,rw4,rw5,sel);
    input ra_r,rb_r,regWrite4,regWrite5;
    input regDin4,jal4;
```

```
    input [4:0]ra,rb,rw4,rw5;
    output [3:0]sel;
    assign sel[0] = ra_r & (rw4==ra) & (rw4!=0) & (~regDin4)
                    & (~jal4) & regWrite4;
    assign sel[1] = rb_r & (rw4==rb) & (rw4!=0) & (~regDin4)
                    & (~jal4) & regWrite4;
    assign sel[2] = ra_r & (rw5==ra) & regWrite5;
    assign sel[3] = rb_r & (rw5==rb) & regWrite5;
endmodule
```

由于 Verilog 实现的时候，对于数据选择可以使用 case 进行实现，从而避免了大量的数据选择器操作，其自动优化为逻辑门电路。

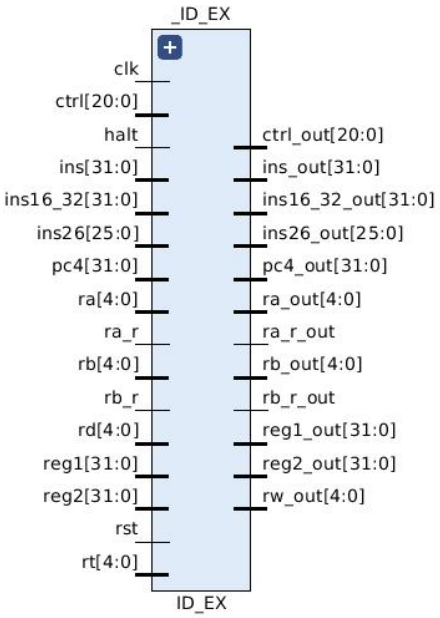

图 3. 23 重定向流水接口封装图

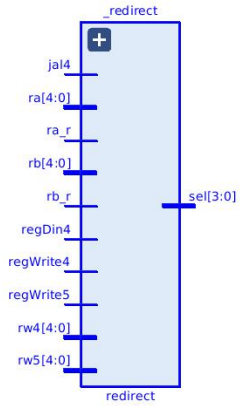

图 3.24 重定向模块封装图

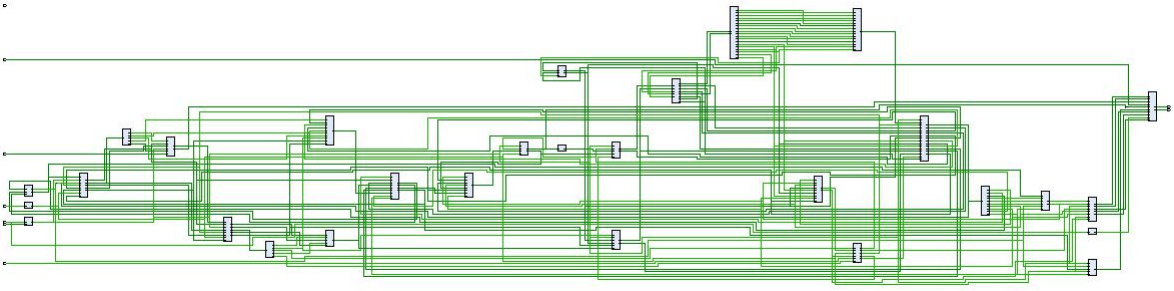

图 3.25 重定向顶层电路图

3.6 动态分支预测机制实现

1) 设计双位预测状态机组合电路，组合逻辑即可，BHT 表中会使用该模块。输入位原预测位，实际跳转情况，输出为新的状态位。

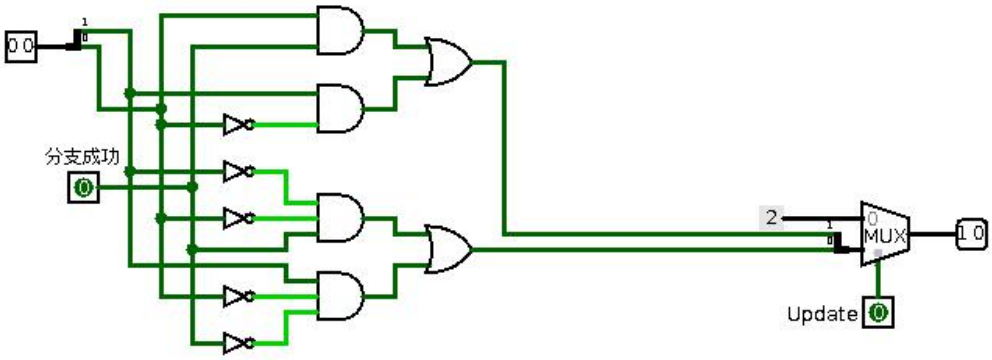

图 3.26 双位预测状态机图

2) 设计 BHT 表，设计全相联并发查找机制，也就是增加多路并发比较比较机制，根据比较的结果获取分支地址以及分支预测位。

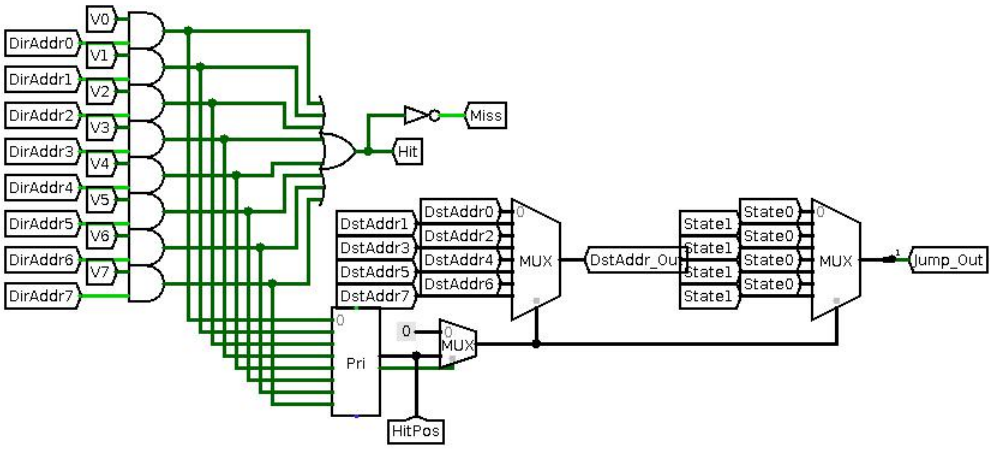

图 3.27 全相联比较电路图

3) 为 BHT 表增加写入逻辑，能够将一条新的分支指令信息加入 BHT 表，当有空位时能将表项内容添加到空位中，注意 BHT 的写入是在分支指令执行时实施的。

4) 实现 BHT 表的 LRU 置换算法。

类似于 cache 的内容相关查找机制，利用 PC 作为查询关键字，目标地址和预测状态机作为存储数据，进行 cache 行的复制，完成整个电路。

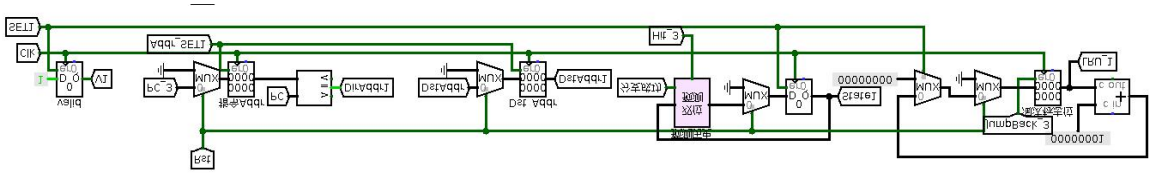

图 3.28 分支预测 cache 实现图

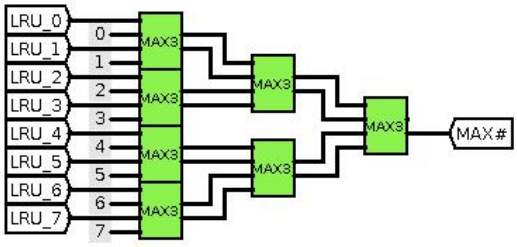

图 3.29 LRU 位置产生电路图

5) 整体架构

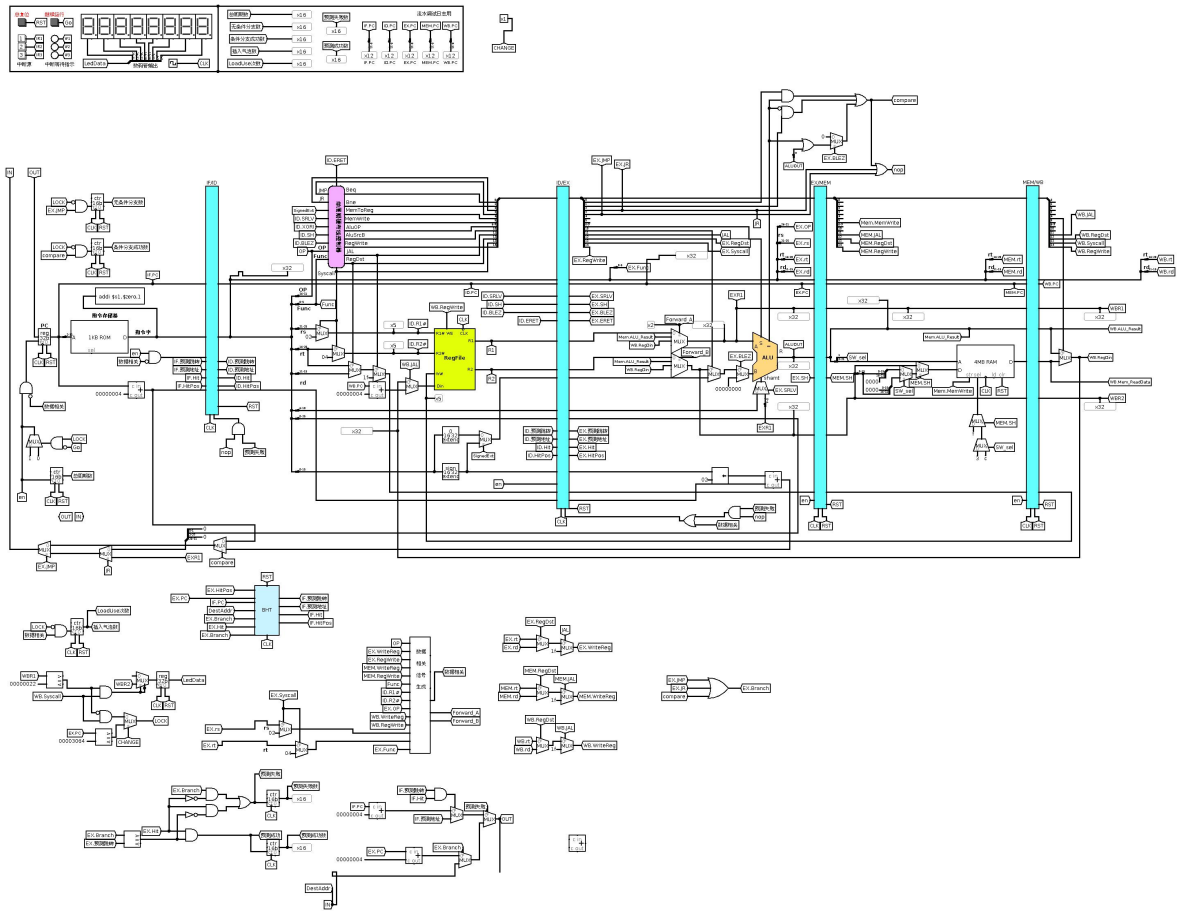

图 3. 30 分支预测整体电路图

4 实验过程与调试

4.1 测试用例和功能测试

对实验各部分进行逐项测试，由于未保存上板图片，故以实验现象描述。

4.1.1 单周期上板测试

① Logisim 测试

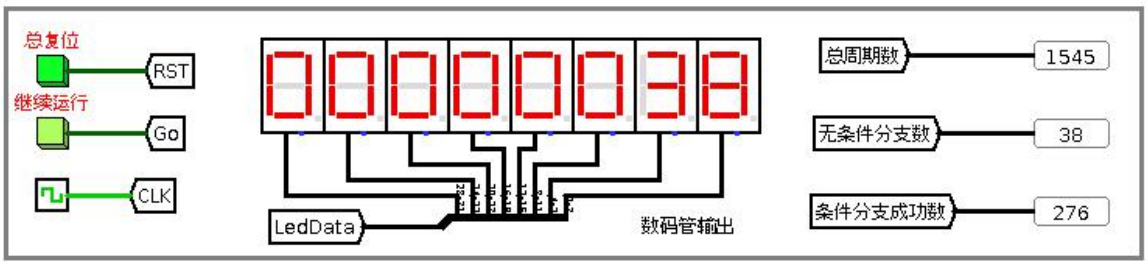

图 4.1 单周期 Logisim 测试结果图

00000 0000000e 0000000d 0000000c 0000000b 0000000a 00000009 00000008 00000007 00000006 00000005 00000004 00000003 00000002 00000001 00000000 ffffffff

图 4.2 单周期 Logisim 测试 DM 图

② FPGA 仿真测试

在实验过程中，对于每一部分进行仿真测试，最后进行综合仿真测试，进行行为仿真，输出变量波形进行调试。

```
`timescale 1ns / 1ps
module simu_top( );
    reg rst,clk;
    reg [3:0]mem_addr;
    reg [2:0] dis_mode;
    reg swi_halt;
    reg swi_freq;
     wire [7:0] seg;
    wire [7:0] an;
```

```
        TopLayer _top(
            .rst(rst),.CLK(clk),.mem_addr(mem_addr),.dis_mode(dis_mode),
            .swi_halt(swi_halt),.swi_freq(swi_freq),.seg(seg),.an(an)
            );
        initial begin
            clk <=0;
            dis_mode = 0;
            swi_freq = 0;
            mem_addr = 0;
            rst = 0;
        end
        always begin
            15 clk <= ~clk;
        end
    endmodule
```

③ FPGA 上板测试

上板所得结果与 Logisim 相同。

4.1.2 理想流水线测试

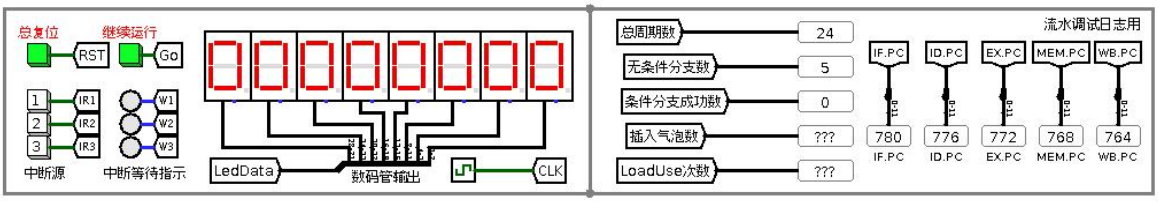

图 4.3 理想流水线测试 Logisim 测试结果图

4.1.3 气泡流水线测试

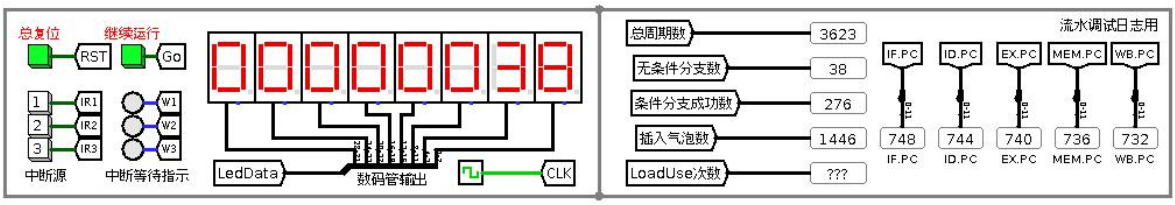

图 4.4 气泡流水线 Logisim 测试结果图

4.1.4 重定向流水线测试

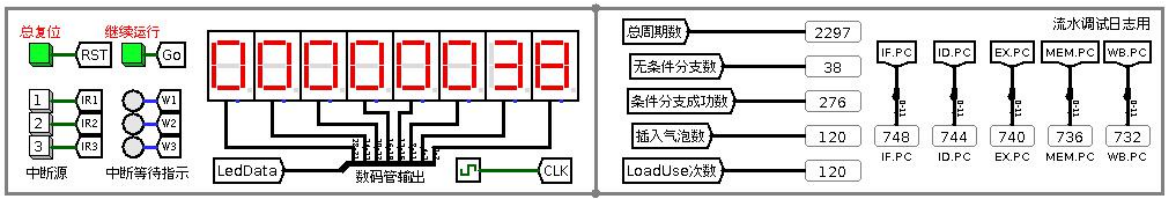

图 4.5 重定向流水线 Logisim 测试结果图

4.1.5 单级中断测试

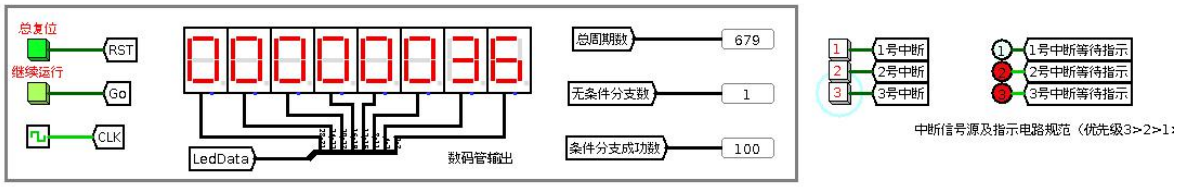

图 4.6 单级中断测试结果图 1

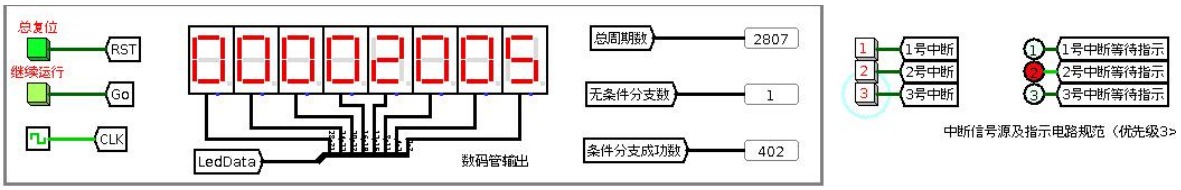

图 4.7 单级中断测试结果图 2

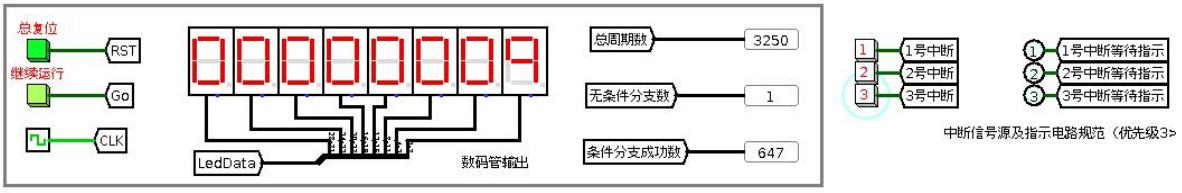

图 4.8 单级中断测试结果图 3

4.1.6 多级中断测试

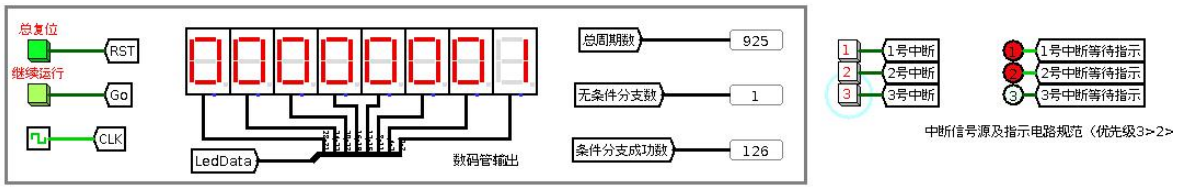

图 4.9 多级中断抢占结果图

4.1.7 流水中断测试

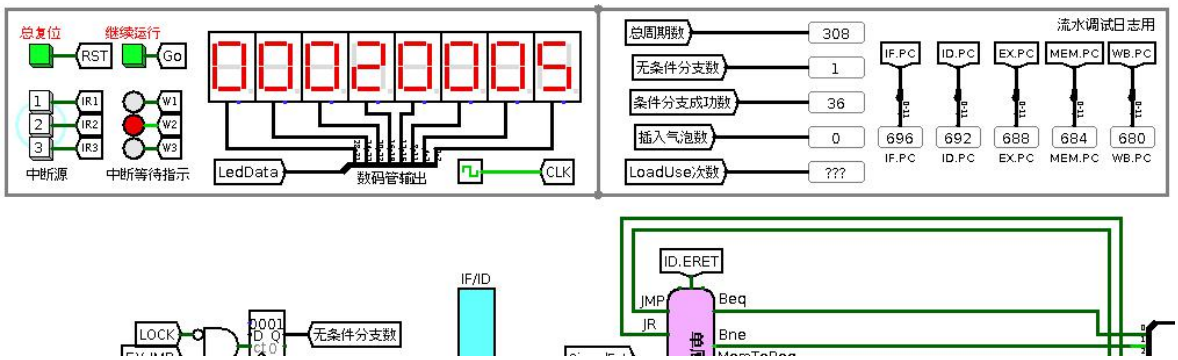

图 4.10 流水中断测试图

4.2 性能分析

分析不同方案时钟周期数差异：

单周期周期数为 1545，和最终的分支预测流水线进行比较，周期数稳定，不随指令改变，虽然周期数较少，但每一个周期较长，为后续流水线一个周期的若干倍。整体上单周期性能最差。由 图 4.11 单周期电路统计图 和 图 4.12 分支预测电路统计图可以看到，分支预测流水线的门数是单周期的两倍以上。

理想流水线时理想中的，对于数据冲突不进行处理，一旦发生冲突，最终就会产生错误，故不进行性能考量。

气泡流水线的误取深度为 2，发生错误即产生 2 个时空槽的浪费，在 ID/EX 进行处理可能会有不同。

重定向流水线，采用重定向机制解决大部分数据冲突，在给定指令下，重定向流水线的周期比气泡流水线大概少了三分之一，由于重定向的机制，关键路径有一定增长，总体频率上限有一定减低，但是整体上仍然性能提高。

分支预测机制，更多的依赖于数据集，其性能不稳定，当总体上仍有提高。

Logisim: ◇单周期MIPS Statistics

Component	Library	Simple	Unique	Recurs...
TOTAL (with subcircuits)		180	1150	1150
TOTAL (without project's su...		177	1144	1144
◇运算控制器	IT1601_U2016...	0	1	1
◇控制信号生成	IT1601_U2016...	0	1	1
◆硬布线控制器	IT1601_U2016...	1	1	1
◆MIPS Regifile	IT1601_U2016...	1	1	1
◆MIPS ALU	IT1601_U2016...	1	1	1
XOR Gate	Gates	0	1	1
Tunnel	Wiring	61	94	94
Subtractor	Arithmetic	0	1	1
Splitter	Wiring	17	26	26
Shifter	Arithmetic	1	4	4
ROM	Memory	1	1	1
Register File	CS3410 Compo...	0	1	1
Register	Memory	2	2	2
Probe	Wiring	3	3	3
Pin	Wiring	0	95	95
OR Gate	Gates	3	14	14
NOT Gate	Gates	0	673	673
NOR Gate	Gates	0	1	1
Multiplier	Arithmetic	0	1	1
Multiplexer	Plexers	20	22	22
MIPS RAM	CS3410 Compo...	1	1	1
LED	Input/Output	3	3	3
Label	Base	24	31	31
Hex Digit Display	Input/Output	8	8	8
Ground	Wiring	0	2	2
Divider	Arithmetic	0	1	1
Counter	Memory	3	3	3
Constant	Wiring	17	25	25
Comparator	Arithmetic	1	9	9
Clock	Wiring	1	1	1
CCMB_Conrtoller	IT1601_U2016...	0	1	1
Button	Input/Output	2	2	2
Bit Extender	Wiring	2	2	2
AND Gate	Gates	5	114	114
Adder	Arithmetic	2	3	3

图 4.11 单周期电路统计图

Logisim: ◇分支预测 Statistics

Component	Library	Simple	Unique	Recurs...
TOTAL (with subcircuits)		488	2881	3464
TOTAL (without project's su...		479	2852	3435
◇运算控制器	IT1601_U2016...	0	1	1
◇源寄存器使用情况	IT1601_U2016...	0	2	2
◇数据相关检测	IT1601_U2016...	1	1	1
◇控制信号生成	IT1601_U2016...	0	1	1
◇双位预测状态机	IT1601_U2016...	0	8	8
◇MEM/WB	IT1601_U2016...	1	1	1
◇MAX_3	IT1601_U2016...	0	7	7
◇IF/ID	IT1601_U2016...	1	1	1
◇ID/EX	IT1601_U2016...	1	1	1
◇EX/MEM	IT1601_U2016...	1	1	1
◇BHT	IT1601_U2016...	1	1	1
◆硬布线控制器	IT1601_U2016...	1	1	1
◆MIPS Regifile	IT1601_U2016...	1	1	1
◆MIPS ALU	IT1601_U2016...	1	1	1
XOR Gate	Gates	0	1	1
Tunnel	Wiring	256	774	774
Subtractor	Arithmetic	0	1	1
Splitter	Wiring	35	51	65
Shifter	Arithmetic	1	4	4
ROM	Memory	1	1	1
Register File	CS3410 Compo...	0	1	1
Register	Memory	2	84	84
Probe	Wiring	29	29	29
Priority Encoder	Plexers	0	2	2
Pin	Wiring	1	253	331
OR Gate	Gates	6	26	48
NOT Gate	Gates	2	1019	1379
NOR Gate	Gates	0	1	1
Multiplier	Arithmetic	0	1	1
Multiplexer	Plexers	34	134	153
MIPS RAM	CS3410 Compo...	1	1	1
Mips Probe	CS2018 Compo...	1	1	1
LED	Input/Output	3	3	3
Label	Base	34	43	49
Hex Digit Display	Input/Output	8	8	8
Ground	Wiring	0	59	59
Divider	Arithmetic	0	1	1
Decoder	Plexers	0	1	1
Counter	Memory	6	6	6
Constant	Wiring	26	82	89
Comparator	Arithmetic	3	33	39
Clock	Wiring	1	1	1
CCMB_Conrtoller	IT1601_U2016...	0	1	1
Button	Input/Output	5	5	5
Bit Extender	Wiring	2	2	2
AND Gate	Gates	16	209	280
Adder	Arithmetic	6	15	15

图 4.12 分支预测电路统计图

4.3 主要故障与调试

4.3.1 分支预测无法正确跳转故障

故障现象：分支预测无法正确跳转，且大部分数据预测失败。

原因分析：将 PC 和 PC+4 弄混，传输的信号为 PC 需要进行+4 后作为输入信号

解决方案：更正预测跳转模块

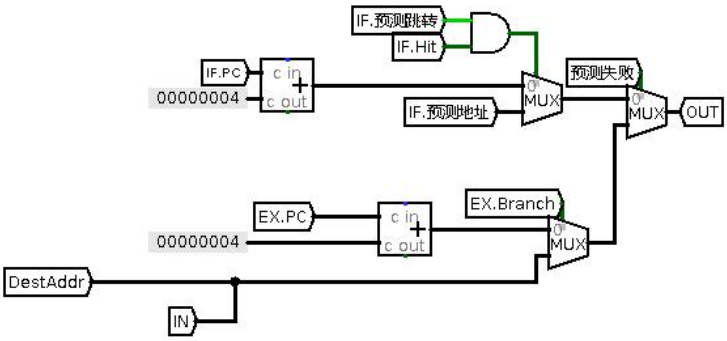

图 4. 13 更正后分支预测跳转模块图

4.3.2 数据相关信号红线故障

故障现象：如图所示，输出出现红线操作。

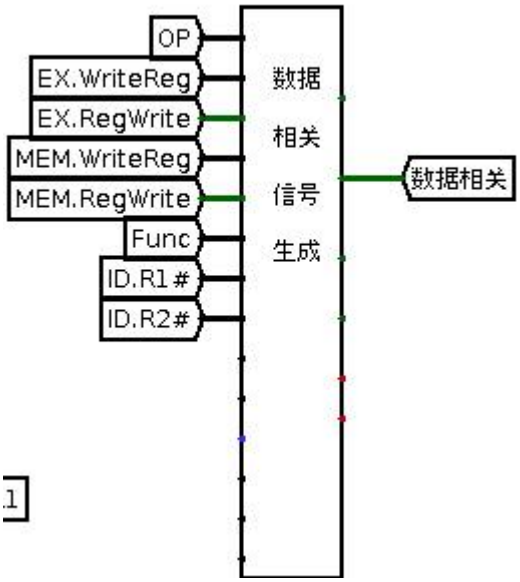

图 4. 14 数据相关模块故障图

原因分析：

由于共用同一个模块，不同的电路使用的引脚不同，造成输入引脚悬空，从而造成输出红线

解决方案：不同电路切换的时候，使用 ctrl+R 进行电路重置，对于悬空引脚，注意不要将输出引出，否则造成电路错误。

4.3.3 流水线状态故障

故障现象：

流水线整体现实正确，但流水线周期数不对。

53	72	0	0	872	868	880	0	0	872	868	1				
54	76	72	0	0	872	884	880	0	0	872	1	1			
55	80	76	72	0	0	888	884	880	0	0	1	1	1		
56	84	80	76	72	0	892	888	884	880	0	1	1	1	1	
57	88	84	80	76	72	896	892	888	884	880	1	1	1	1	1
58	92	88	84	80	76	900	896	892	888	884	1	1	1	1	1
59	96	92	88	84	80	904	900	896	892	888	1	1	1	1	1
60	100	96	92	88	84	908	904	900	896	892	1	1	1	1	1
61	104	100	96	92	88	912	908	904	900	896	1	1	1	1	1
62	108	104	100	96	92	916	912	908	904	900	1	1	1	1	1
63	112	108	104	100	96	920	916	912	908	904	1	1	1	1	1
64	84	0	0	104	100	924	920	916	912	908	1	1	1	1	1
65	88	84	0	0	104	928	924	920	916	912	1	1	1	1	1

图 4. 15 流水线时空图对比错误图

原因分析：

追踪到当前指令，逐步调试，跳转指令阶段标签错误，造成跳转指令不工作

解决方案：

更改至正确的标签

4.4 实验进度

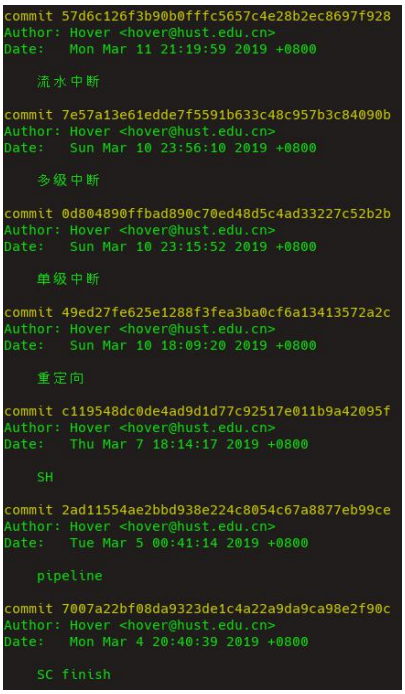

图 4.16 Git Log 实验过程纪录图

实验过程中采用 Git 进行版本管理，并标注进度，上图为 log 图。

表 4.1 课程设计进度表

时间	进度
第一天	完成单周期 Logisim 的绘制和小组相关分工
第二天	完成 Verilog 单周期部分模块的编写和相关测试工作
第三天	进行 Verilog 单周期整体模块的仿真和上板测试
第四天	完成理想流水线的 Logisim 绘制，调试通过理想流水线
第五天	完成气泡流水线的 Logisim 绘制，调试通过气泡流水线
第六天	完成重定向流水线的 Logisim 绘制，调试通过重定向流水线
第七天	更改单周期 Verilog 代码相应接口，增加流水接口
第八天	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。
第九天	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。
第十天	完成数据重定向的 Verilog 代码并成功烧入 FPGA 板内。成功实现动态分支预

时间	进度
	测，预测成功率显著提高，并成功将代码烧入 FPGA 板内。

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) 工作流程规范化：设计过程中，一定要先想好设计目标，从工程化的角度进行考虑，从而不会在设计的过程中走弯路。
2) FPGA:复习了 Verilog 相关知识，并使用其开发较大工程，进一步强化了硬件开发能力。
3) 性能优化：通过不断处理数据相关问题，CPU 相应性能逐步提升，从而加深理解了 CPU 的工作机制及其优化方向。
4) 中断机制：实现 OS 中断的底层支持，更好的理解了中断原理。

5.2 课设心得

课程设计过程由于有过流水线相应的课程基础，所以相对来说会轻松一些，实现过程中，也参阅了许多资料，体会到作为一个架构开发人员所经历的。

在寒假中使用 Logisim 制作 24+4 的单周期 CPU，由于在上学期的实验中进行了一定的尝试，对于 J 型和 I 型指令有了一定的基础，实验完成较快，且工程化的设计思路对于指令条数实际上并无脑力上的差异。

对于单周期上板，由于 Verilog 已久未使用，故而参阅了《自己动手写 CPU》和《CPU 自制入门》两书，对其中前面 Verilog 精要介绍进行复习，从而能够较快的捡起 Verilog 知识，且当时 Verilog 上课时所完成的数据通路文件还在，也起到了一定的作用。

实现过程中，采用分块测试仿真的方法，因为 Veriolog 代码好写，DEBUG 难，为了不留下各种隐藏炸弹，对于每一块进行仿真和测试，此时较为麻烦的是对于引脚，有些不需要使用的引脚 wire 声明，但是仿真需要 Input 和 Output 暴露出来，从而造成一定的反复修改，做法是另取一份出来，改完作为 Test_Version，而不更改原有的工程文件。

流水线部分，参阅老师 MOOC 讲解，基本只需将抽象的逻辑通路具体化，其选择信号可以用较为软件的方式产生，先手写出逻辑表达式，然后取出相应的信号进行逻辑运算，输出判断信号。测试样例部分，有时候会弄混不同的指令测试文件，此处可以将 Logisim_Test 和 Verilog_Test 两个文件分开放置，有文件头的差异，从而避免 FPGA 编译很长时间而拷入错误的指令文件。

重定向流水部分，需要区分不同段的信号，此时标准的命名就很重要，不仅仅是在封装电路图外面，在电路封装里面，曾经因为错误的命名导致标签连接错误，玄学 BUG 好久，统一采用规范命名，能节省很多时间。

由于 Logisim 的仿真问题，导致出现单周期速度最快，重定向反而较慢的物理错觉，且频率切换不明显，时间仅仅在周期数上有体现。

对于中断部分，由于中断开关的设置问题，在某个时间中断不响应的情况，采用单步调试最终得以解决。

由于 Logisim 的某些特性，标签由于小的差异而不显示连接，较远的标签不好找等问题，造成了很多不必要的时间浪费，出现电路改着改着不工作的情况，及时保留历史版本，随手 GIT 是一个好习惯。总的来说，课程设计学到了很多知识，同时将 OS 和组成原理的知识串联起来，且工程化锻炼明显，受益良多。

关于课设的建议：

1) 相对来说，Logisim 使用明显易操作，Logisim 为类似标签语言，受学长 Pipeline-Generator 的启发，在工作的早期曾经想实现 Logisim 生成 Verilog 代码的工具，GitHub 上有 Developing Project。完全生成可能起不到硬件开发的相关锻炼，但是如果能够用 Verilog 实现类似于 Logisim 和 VHDL 的嵌套，应该是不错的体验。
2) 分组机制：整体工作上，小组讨论居多，对于代码的共同开发可能较少，尽管使用 GitHub 作为小组代码共享，仍然感觉合作力度不够。
3) 时间安排：时间相对来说很紧，对于很多设计上由于时间压力没有做到很优化，导致代码风格和模块化设计并不是很好，如果能够提早发布任务，和阅读材料(比如《自己动手写 CPU》)应该能够留下更完美的 CPU 作品，作为本科很棒的一个纪念。

最后在这里也感谢在实验过程中提供帮助的老师和同学，小组合作的队友和相关资料的提供者。

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] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4] 谭志虎，秦磊华，胡迪青.计算机组成原理实践教程.北京：清华大学出版社，2018.

[5] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

[6] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

[7] 雷思磊.自己动手写 CPU.电子工业出版社，2014 年.

[8] [日] 水头一寿,[日] 米泽辽,[日] 藤田裕士.CPU 自制入门.人民邮电出版社，2014 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：