华中科技大学

2022

计算机组成原理　　课程设计报告

题　　目：	5 段流水 CPU 设计
专　　业：	计算机科学与技术
班　　级：	CSXJ1901
学　　号：	U201911111
姓　　名：	刘庆远
电　　话：	18322563137
邮　　件：	2359772709@qq.com

目 录

1 课程设计概述

1.1 课设目的

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

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

1.2 设计任务

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

1.3 设计要求

（1） 根据课程设计指导书的要求，制定出设计方案；

（2） 分析指令系统格式，指令系统功能。

（3） 根据指令系统构建基本功能部件，主要数据通路。

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

（5） 设计出实现指令功能的硬布线控制器；

（6） 调试、数据分析、验收检查；

（7） 课程设计报告和总结。

1.4 技术指标

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10）支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

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

（13）能运行教师提供的标准测试程序，并自动统计执行周期数

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

表 1.1 指令集

#	指令助记符	简单功能描述	备注
1	ADD	加法	指令格式参考 MIPS32 指令集，最终功能以 MARS 模拟器为准。
2	ADDI	立即数加	
3	ADDIU	无符号立即数加	
4	ADDU	无符号数加	
5	AND	与	
6	ANDI	立即数与	
7	SLL	逻辑左移	
8	SRA	算数右移	
9	SRL	逻辑右移	
10	SUb	减	
11	OR	或	
12	ORI	立即数或	

#	指令助记符	简单功能描述	备注
13	NOR	或非	
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	SLLV	逻辑可变左移	
29	XORI	异或立即数	
30	LHU	加载半字（无符号）	
31	BGEZ	大于等于 0 转移	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 设计，本次采用的方案是硬布线控制方案，并且在本次实验中，指令与数据得到分开存储。在指令周期内，控制器根据读入的指令给出相应控制信号，基于这些控制信号，其余功能部件实现不同的任务，从而实现指令，如对存储器的存储、对寄存器组的访问等。在实施的过程中，全部采用 Logisim 仿真实现。总体结构图如图 2.1 所示。

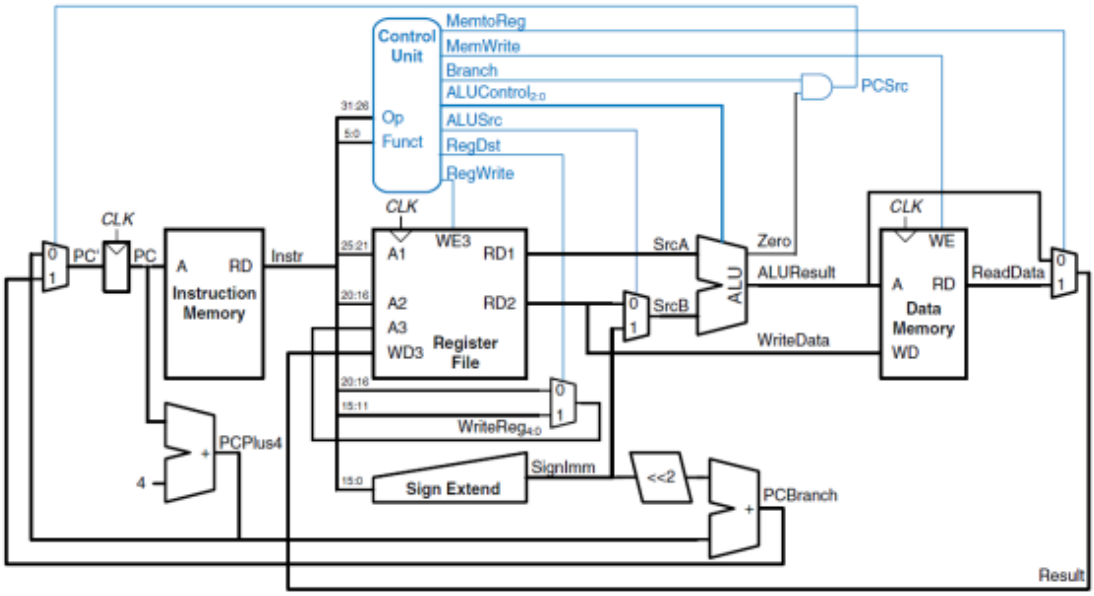

图 2.1 总体结构图

2.1.1 主要功能部件

主要功能部件有运算器、程序计数器 PC、寄存器堆 RF、指令存储器 IM 与数据存储器 DM 以及符号扩展模块及数据选择器等。具体设计思路如下：

1. 程序计数器 PC

主要存储并给出下一条指令在指令存储器中的位置，输入端口有：下一条指令在 IM 中地址、控制 PC 是否跳转到输入的使能信号、时钟信号以及复位信号；输出为

下一条指令对应指令存储器的指令地址。同时设置为上升沿触发。

2. 指令存储器 IM

指令存储器负责存储程序的指令,经由 PC 给出地址之后,读取出来对应的指令。。输入端口有：指令的存储地址；输出端口有：二进制 MIPS 指令。因为一个指令占有四个字节，所有为了取整指令，需要将地址末 2 位去掉。取 2-11 位作为地址。

3. 运算器

运算器为实验提供实现好的部件，引脚描述如下：

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

表 2.2 ALUOP 和功能描述

ALU_OP	运算功能
0000	Result = X << Y 逻辑左移 （Y 取低五位） Result2=0
0001	Result = X >>>Y 算术右移 （Y 取低五位） Result2=0
0010	Result = X >> Y 逻辑右移 （Y 取低五位） Result2=0
0011	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 无符

ALU_OP	运算功能
	号乘法
0100	Result = X/Y; Result2 = X%Y 无符号除法
0101	Result = X + Y (Set OF/UOF)
0110	Result = X - Y (Set OF/UOF)
0111	Result = X & Y 按位与
1000	Result = X | Y 按位或
1001	Result = X⊕Y 按位异或
1010	Result = ~(X |Y) 按位或非
1011	Result = (X < Y) ? 1 : 0 符号比较
1100	Result = (X < Y) ? 1 : 0 无符号比较

4. 寄存器堆 RF

寄存器堆采用 MIPS 自带的寄存器实现，包含 32 个通用寄存器，该寄存器堆的输入输出引脚以及相应功能的描述如表 2.3 所示。

表 2.3 寄存器堆引脚与功能描述

引脚	位宽
R1#	所读寄存器 1 的编号
R2#	所读寄存器 2 的编号
W#	写寄存器信号
Din	写寄存器的值
WE	写使能
CLK	时钟信号
R1	寄存器 R1 里面的值
R2	寄存器 R 里面的值

5. 控制器

采用硬布线控制，通过解析出的指令给出不同的控制信号，根据输入的指令通过组合逻辑生成相应的控制信号即可

2.1.2 数据通路的设计

表 2.4 指令系统数据通路框架

指令	PC	RF				ALU			DM	
		R1#	R2#	W#	Din	A	B	OP	Addr	Din
RR 型	PC+4	Rs	Rt	Rd	Alu	R1	R2	Op		
R1 型	PC+4	Rs		Rt	Alu	R1	Imm	Op		
分支指令	$[PC+4]_{\{31:28\}}\|\|\text{imm}\|\|00$									
跳转指令	$[PC+4]_{\{31:28\}}\|\|\text{intr}_{\text{index}\|\|00\|\|R1}$									
Store 型	PC+4	Rs	Rt			R1	Imm		Alu	R2
Load 型	PC+4	Rs	Rt	Rt	Data	R1	Imm	Op	Alu	
SysCall 型	PC+4	2#	4#			R1	Imm			

2.1.3 控制器的设计

控制器设计主要还是要去对照指令的功能，依照功能去选择对应的信号，然后填写 excel 表格自动生成表达式，自动生成电路。

表 2.5 主控制器控制信号的作用说明

控制信号	产生条件	信号说明
RegWrite	寄存器写回信号	寄存器写使能
MemWrite	sw 指令　未单独设置 MemRead 信号	写内存控制信号
AluOP	R 型指令根据 Func 选择	运算器操作控制符（4 位）
MemToReg	lw 指令	寄存器写入数据来自存储器
RegDst	R 型指令	写入寄存器编号 rt/rd 选择
AluSrcB	lw 指令，sw 指令，立即数运算类指令	运算器 B 输入选择

控制信号	产生条件	信号说明
SignedExt	ADDI、ADDIU、SLTI、LW、SW	立即数符号扩展
JR	JR 指令	寄存器跳转指令译码信号
JAL	JAL 指令 ，选择寄存器写回编号，写回值	JAL 指令译码信号
JMP	J、JAL、JR 指令，选择无条件分支地址	无条件分支控制信号

图 2.1 主控制器控制信号框架

指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL
SLL	0	0	0				1			1					
SRA	0	3	1				1			1					
SRL	0	2	2				1			1					
ADD	0	32	5				1			1					
ADDU	0	33	5				1			1					
SUB	0	34	6				1			1					
AND	0	36	7				1			1					
OR	0	37	8				1			1					
NOR	0	39	10				1			1					
SLT	0	42	11				1			1					
SLTU	0	43	12				1			1					
JR	0	8	X							1			1	1	
SYSCALL	0	12	X					1							
J	2	X	X											1	
JAL	3	X	X				1							1	1
BEQ	4	X	X								1				
BNE	5	X	X									1			
ADDI	8	X	5			1	1		1						
ANDI	12	X	7			1	1								
ADDIU	9	X	5			1	1		1						
SLTI	10	X	11			1	1		1						
ORI	13	X	8			1	1								
LW	35	X	5	1		1	1		1						
SW	43	X	5		1	1			1						
SLLV	0	4	0				1			1					
XORI	14	X	9			1	1								
LHU	37	X	5	1		1	1		1						
BGEZ	1	X	5						1						

2.2 中断机制设计

2.2.1 总体设计

中断分为外部中断和内部中断，在本次实验中只考虑可屏蔽的外部中断。对于有单级中断的实现，考虑到需要在执行中断后退回程序原位置继续，这里未使用 EPC 来保存当前 PC 值，在判断遇到中断后将当前中断保存在 EPC 中，同时在完成中断后读

取 EPC 的值继续原程序执行。

此外，对于流水中断，还需要考虑到其执行的位置。同时，对于多级中断的实现，在实现 PC 存储的基础上，还需要设置机制来判断终端的优先级，优先级低的不能打断优先级高的中断的执行。在退出多级中断时，也需要判断继续执行一条被打断的低级中断还是开始执行一个已经在排队的低级中断。所以，这里的的 PC 保存要采用堆栈的形式。

中断响应的执行流程如图 2.2 所示。

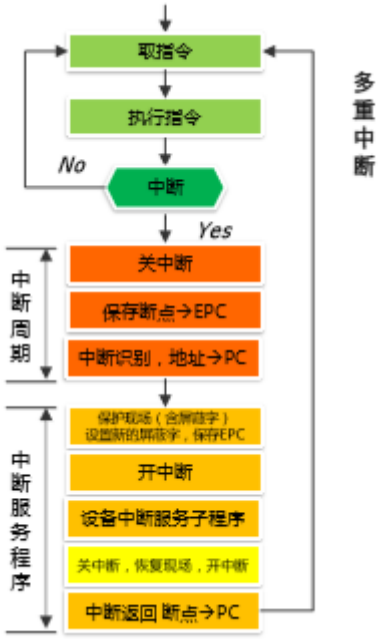

图 2.2 多级中断响应流程

2.2.2 硬件设计

在实现单级中断时，首先是考虑中断的优先级以及 PC 的存储，对于优先级的判断，可以使用优先编码器选择出最先执行的中断号。同时设置寄存器 EPC 来保存中断发生时的 PC 值。同时，需要注意 EPC 寄存器中的值只有当中断发生时才需要更新。在 PC 输入时，也要依据中断发生信号和 ERET 指令，来确保正确的 EPC 复位地址。

在实现多级中断时，需要实现 COP0 寄存器的模拟，在此即为实现 EPC 寄存器和 IE 寄存器的保护，同时用 4 个 D 触发器实现中断屏蔽字的保护，此时进入中断

需要保护的现场包括 PC 以及中断屏蔽字，同时不再关中断，因为高优先级的中断能够打断低优先级中断服务程序的执行，这样就能够实现多级中断的保存。

2.3 流水 CPU 设计

2.3.1 总体设计

在本次实验中，核心是将原有的理想 CPU 处理指令阶段分开独立，以便于指令流水线的工作，来提高运行效率。这里将指令处理分为五个阶段：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）。

对于理想流水线而言，其实就是将原先的 CPU 分区处理了，在每个区之间（IF，ID，EX，MEM，WB）使用寄存器来保存需要传入下一个阶段的信息。

值得注意的是 EX 阶段如果发生跳转指令，需要额外设计机制来传入 IF 阶段的 PC 寄存器，同时在对隧道设置角标时，注意重名。

2.3.2 流水接口部件设计

首先阐述流水线接口应该完成的工作：当时钟信号更新时，将上一阶段的控制或者指令信息传入下一阶段，同时需要保存当前阶段的各类信息。同时，在遇到使能端控制时要将信息锁在寄存器里面。

大体由寄存器实现，当时钟信号为高电平时，将现阶段所有的数据输入端读入数据并将其缓存在寄存器中，数据输出端的值与寄存器保持一致；使能为低电平则不进行操作。若清零端为高电平则进行同步清零；清零端为低电平则不进行清零操作。

2.3.3 理想流水线设计

完成接口设计之后，剩下要做的就是根据教材将单周期 CPU 进行分块，每一个阶段所有的信息都要从其左边的接口获取。

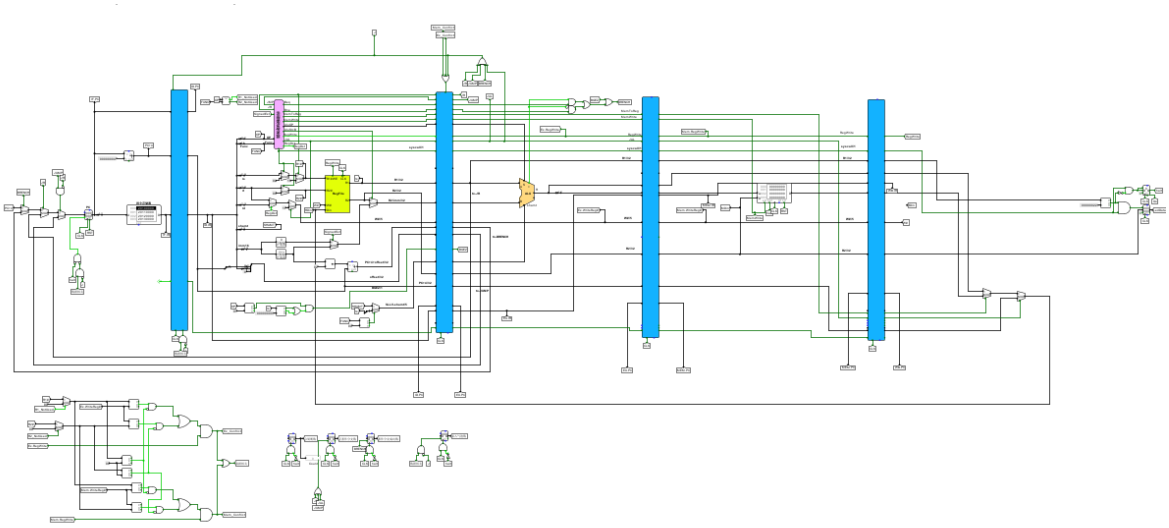

图 2.3 理想流水线实现

2.4 气泡式流水线设计

气泡流水线主要解决的问题是：

（1） 在理想流水线中关于 PC 寄存器接受跳转指令地址与 PC 自动加 4 冲突的问题。由于流水线 CPU 将指令处理划分为 5 各阶段，但在指令解析后去除跳转地址后，往往已经进入都三阶段 EX 阶段，这时 ID 阶段的指令却是 PC+4 的指令。

（2） 在读取数据时，也会因为流水线的构造，导致读取的数据再上一个阶段被修改过，也需要在本次实验中改进。

综上，需要做出以下修改：

① 在 PC 寄存器的使能端口加上“数据相关”信号的判断逻辑以保存发生数据相关时的 PC 状态；

② 同时，在接口处添加置 0 操作，在检测到发生数据冲突时生产气泡。

③ 增加数据相关模块，通过建立“源寄存器使用情况”子模块和“数据相关检测”子模块来检测数据冲突是否存在，生成“数据相关”信号。

2.5 数据转发流水线实现

数据重定向的含义为：将数据可能的输入路径全部汇总，然后根据指令的执行情况判断从哪条路径上取得数据。但是如果发生数据冲突（Load-Use 相关），仍需要加入气泡来回避错误。

数据重定向流水线在本次实验中基于气泡流水线实现。添加 forward 模块以及 load-use 模块，如图 2 . 4 所示。模块内容将在下文介绍。

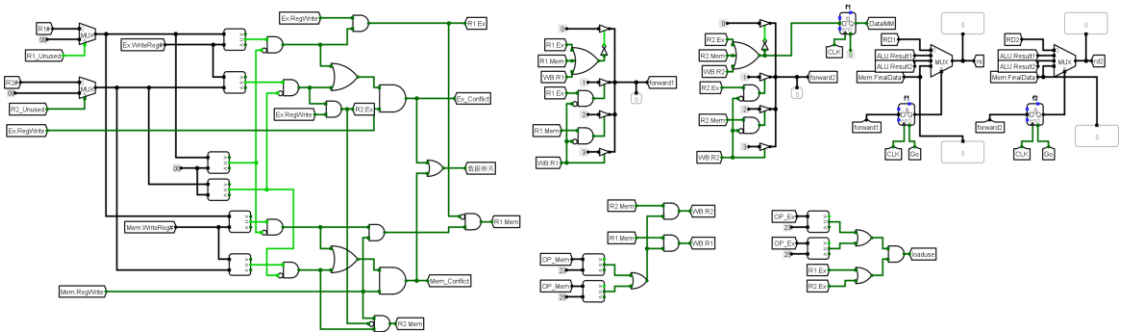

图 2.4　forward 模块与 load-use 模块

2.6 动态分支预测机制

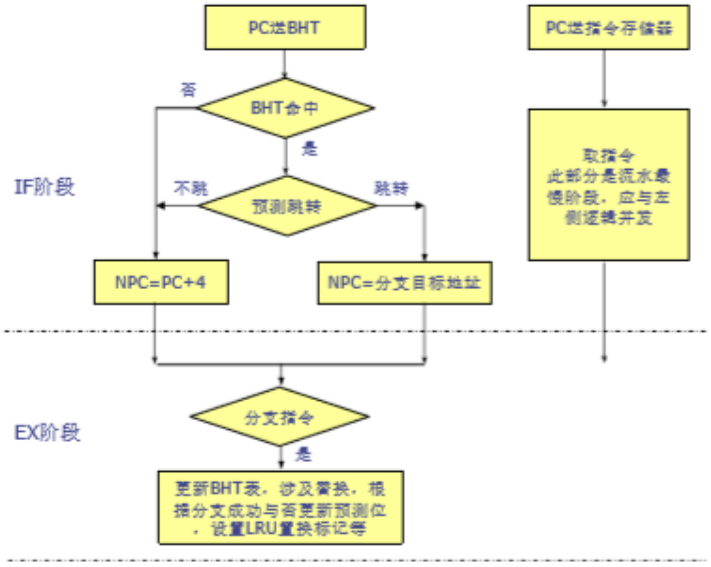

图 2.5 动态分支预测流程图

在气泡流水线实验中，我们解决了数据冲突的问题，在重定向流水线在一定程度上减少了插入气泡数，但二者均无法解决在跳转指令生效前由于预取深度产生的周期浪费，于是我们通过分支预测的方式尝试对跳转指令之后的指令进行预测，从而达到尝试预取正确指令、减少周期浪费的目的。为实现对跳转指令下一条指令的方式，使用构造 cache 的方式实现 BHT 这一硬件，用其对分支指令进行预测。

2.6.1 BHT 设计

分支预测分支历史表 (Branch History Table，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）、valid 位。本实验采用双预测位来进行 BHT 状态转换。

在具体实现中，需要注意两点，首先比较核心的是采用的替换策略，这里我选用的是 LRU 替换策略。类似于 cache。同时还应设置空行判断，如果有空行，应优先存入空行中。在 IF 阶段通过状态判断输出预测结果，在 EX 段根据实际执行结果通过硬件电路更新状态表中的状态。

2.6.2 总体设计

动态分支的设计，是基于重定向流水线的基础上的，特别之处就是在 IF 阶段与 EX 阶段关于 BHT 的操作，在这里我们在 IF 阶段 PC 取地址时要先去 BHT 寻找对应跳转地址。未命中，则动态分支预测流水线的表现与未添加分支预测时的流水线一致；使用了 BHT 但预测失败，这里需要在 EX 段中对预测结果进行判断，结果就是和普通流水线处理一样，添加气泡，重新从正确的地方开始；使用了 BHT 且预测成功，此时流水线能够顺利向下推进而无需插入气泡，避免了由于预取深度造成的周期浪费

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，触发方式为上升沿触发，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，取反后接在寄存器的使能端。同时，PC 的选择也会受到不同指令的影响，在这里添加了选择器进行实现。

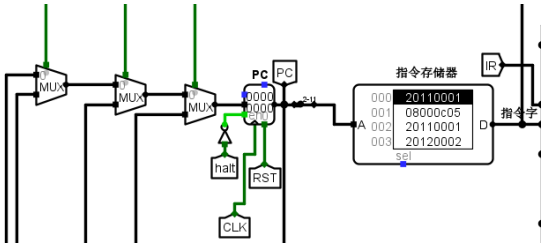

图 3.1 程序计数器（PC）实现

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 3.1 所示。

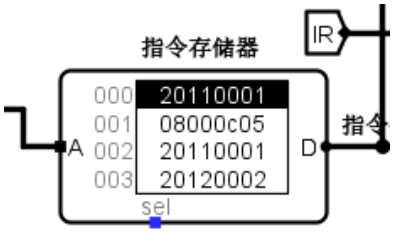

图 3.1 指令存储器（IM）

3）数据存储器（DM）

与指令存储器 IM 一样,DM这里使用了一个RAM实现。设置该地址位宽为10 位，数据位宽为 32 位。因为计算得出的地址为 32 位，而 RAM 地址线宽度有限，仅为

10 位，故将 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。在完成 LB 指令时，需要将数据进行划分并且进行符号扩展。如图 3.3 所示。

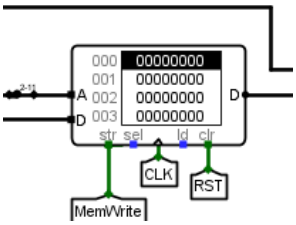

图 3.3 数据存储器（DM）实现

4）寄存器堆（RF）

本次实验使用的是 cs3410.jar 文件中预先实现好的 MIPS 寄存器组。

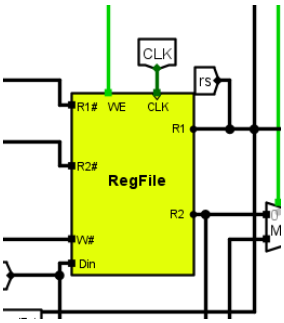

图 3.4 寄存器堆（RF）实现

3.1.2 数据通路的实现

将以上 PC 寄存器，寄存器堆，数据存储器等部件，依照指令功能连接在一起。如下 3.5 图所示

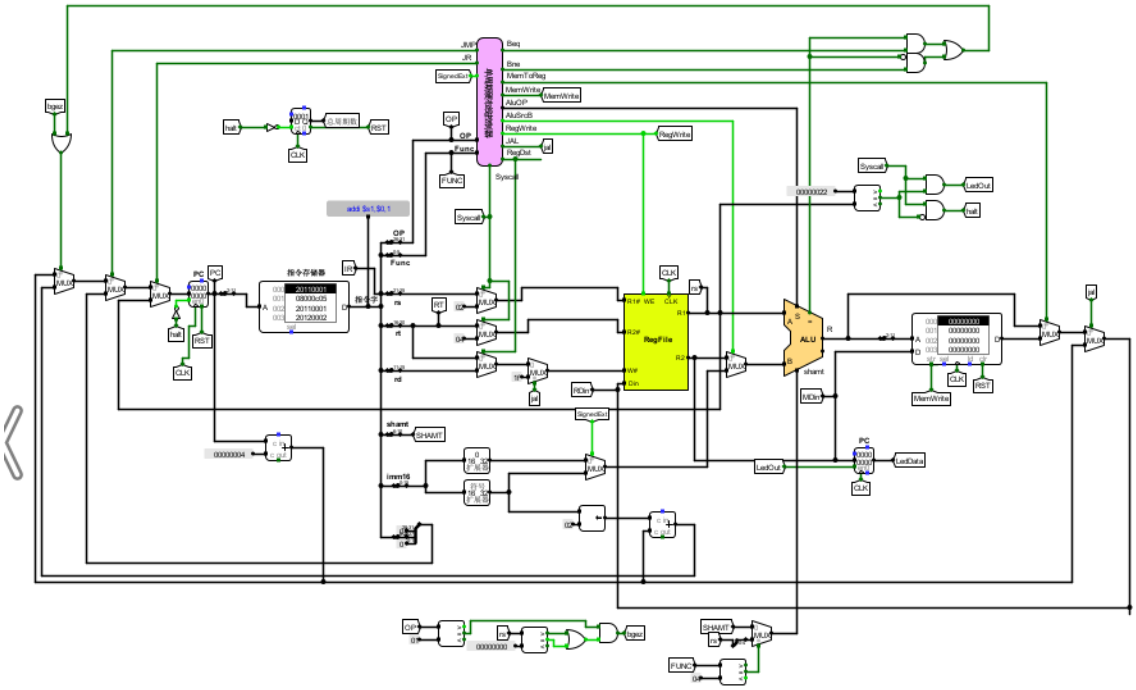

图 3.5 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

填写 excel 表格，使用 Excel 的预先定义好的函数得到相应电路图的规范表达式，使用 Logisim 的自动生成电路功能完成控制器的实现，最终实现的控制器如图 3.6 所示。

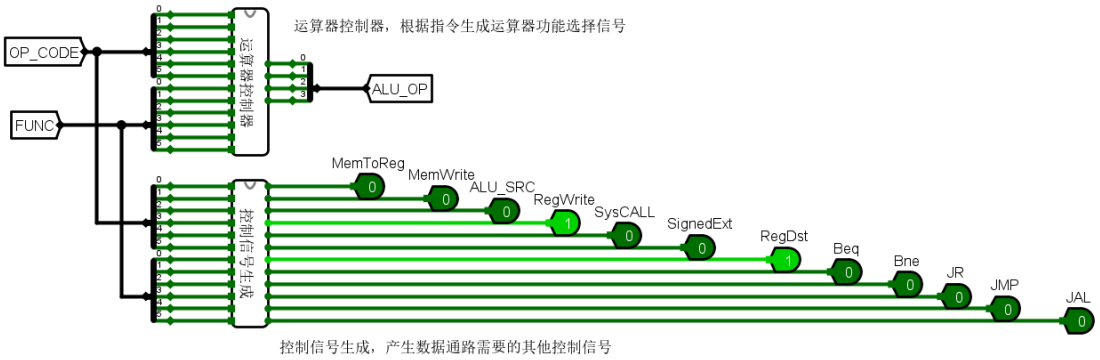

图 3.6 控制器实现

指令	OpCode (十进制)	FUNCT (十进制)	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL
SLL	0	0	0				1			1					
SRA	0	3	1				1			1					
SRL	0	2	2				1			1					
ADD	0	32	5				1			1					
ADDU	0	33	5				1			1					
SUB	0	34	6				1			1					
AND	0	36	7				1			1					
OR	0	37	8				1			1					
NOR	0	39	10				1			1					
SLT	0	42	11				1			1					
SLTU	0	43	12				1			1					
JR	0	8	X							1			1	1	
SYSCALL	0	12	X					1							
J	2	X	X											1	
JAL	3	X	X				1							1	1
BEQ	4	X	X								1				
BNE	5	X	X									1			
ADDI	8	X	5			1	1		1						
ANDI	12	X	7			1	1								
ADDIU	9	X	5			1	1		1						
SLTI	10	X	11			1	1		1						
ORI	13	X	8			1	1								
LW	35	X	5	1		1	1		1						
SW	43	X	5		1	1			1						
SLLV	0	4	0				1			1					
XORI	14	X	9			1	1								
LHU	37	X	5	1		1	1		1						
BGEZ	1	X	5						1						

图 3.7 指令与控制信号关系

3.2 中断机制实现

3.2.1 单级中断

（1）中断的产生与地址位选择

如下图 3.8 所示，首先根据通过隧道将测试中断开关输入，之后采用优先编码器，来判断中断之间的优先级，选择级别高的输出，然后根据选择器选择对应的中断处理地址。

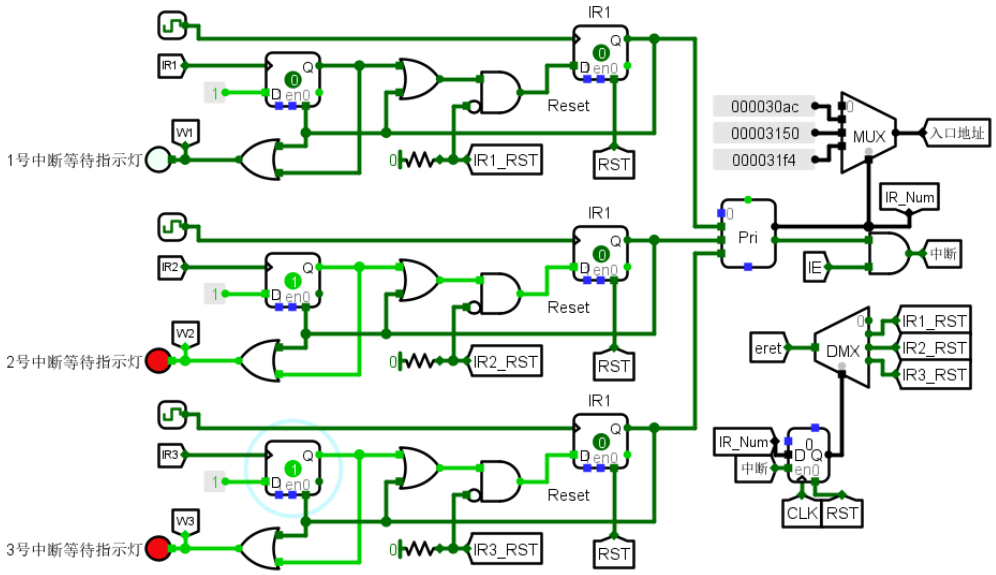

图 3.8 中断处理 1

（2）PC 的保存与恢复

实现如下图，在中断发生时，存入现在的 PC,而后根据 eret 指令输出下一个 PC 地址。

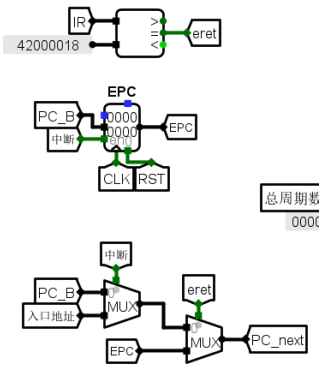

图 3.9 PC 保存与恢复

（3）IE 实现

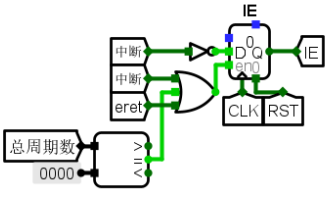

图 3.10 IE 实现

（4）eret 信号生成

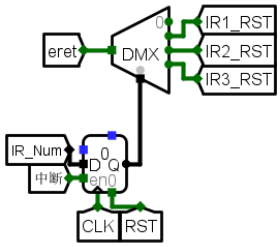

图 3.11 eret 信号生成

（5）总体结构

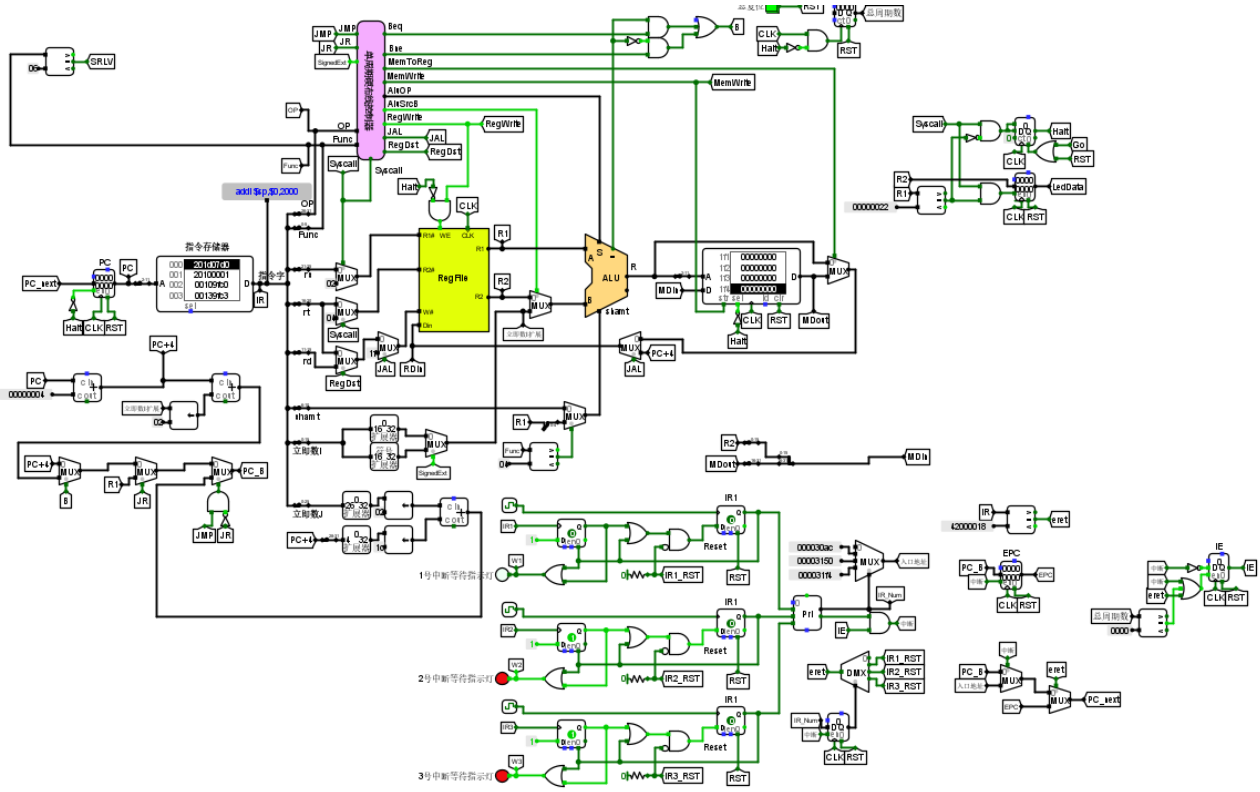

图 3.10 单级中断实现

3.2.2 多级中断

1）多级中断的检测和识别

在单级中断的基础上，增加了新的中断号 IR_Num 和当前运行中断号 IR_Num_Now 的优先级比较电路，主程序的优先级最低。

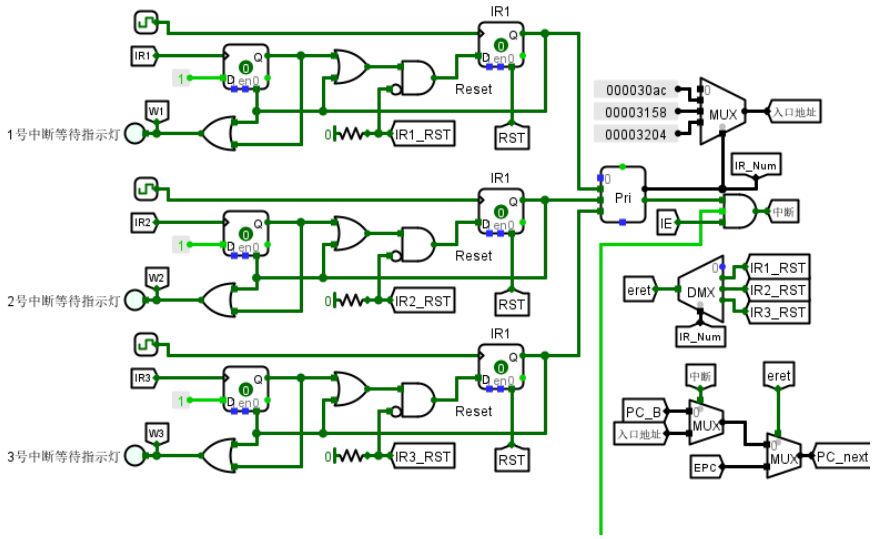

图 3.11 多级中断的检测和识别

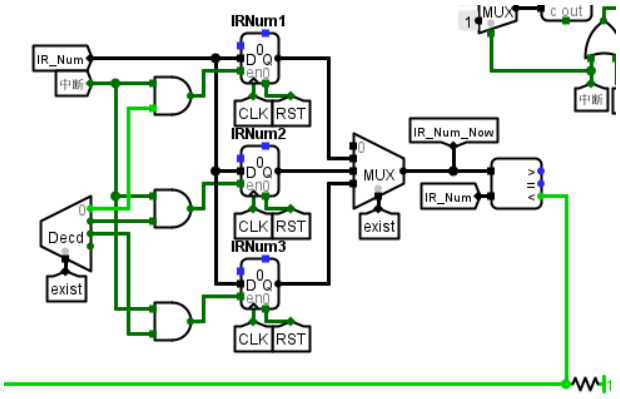

图 3.12 多级中断的检测和识别

2）EPC 赋值与返回结构

这里在返回地址时，不仅需要考虑中断的发生，还需要考虑嵌套中断是否结束。

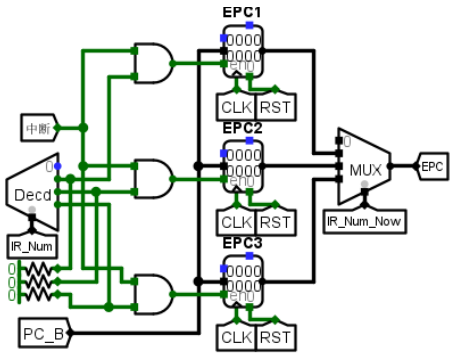

图 3.13 PC 返回后

3）总体设计

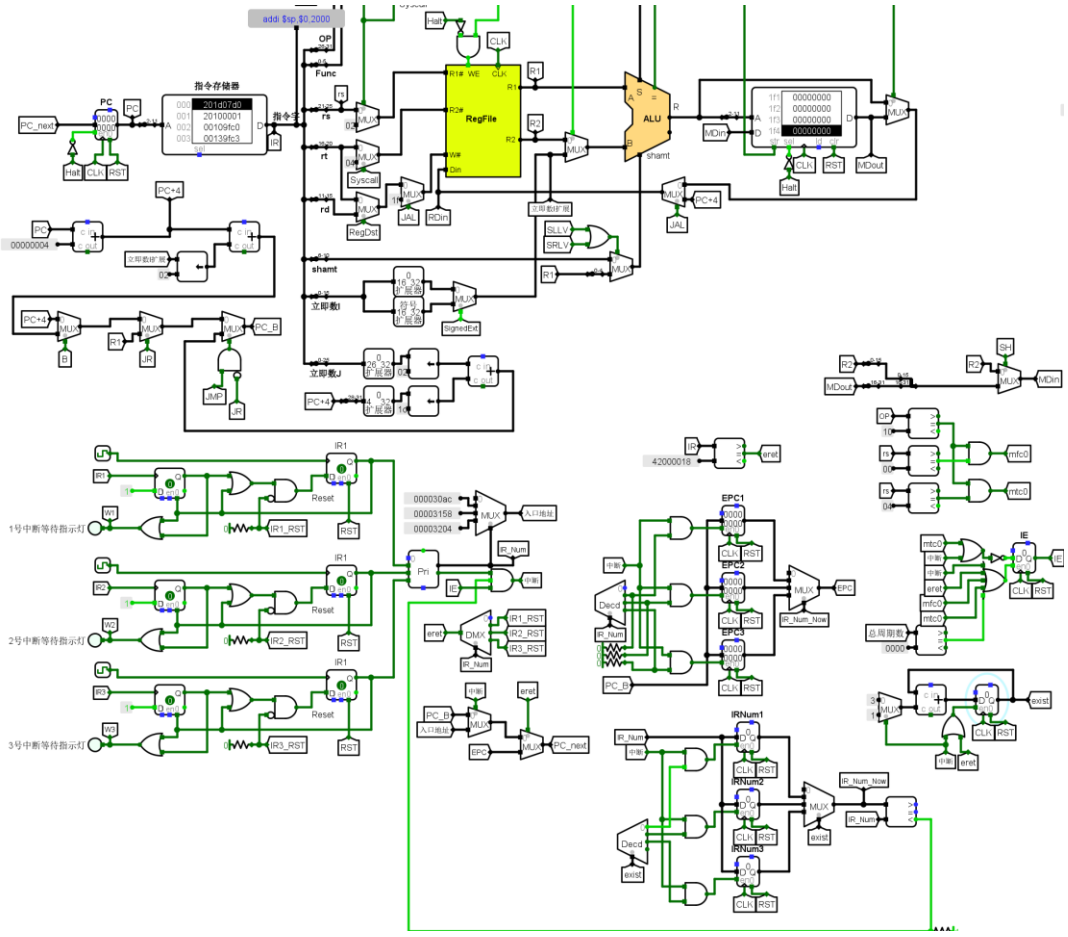

图 3.14 多级终端总体设计

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

接口部件的作用就是在下一个阶段开始之前保存上一阶段的信息，故选择寄存器完成此任务。同时寄存器设置上升沿触发，以及同步清零以实现气泡。具体设计如下图 3.15。

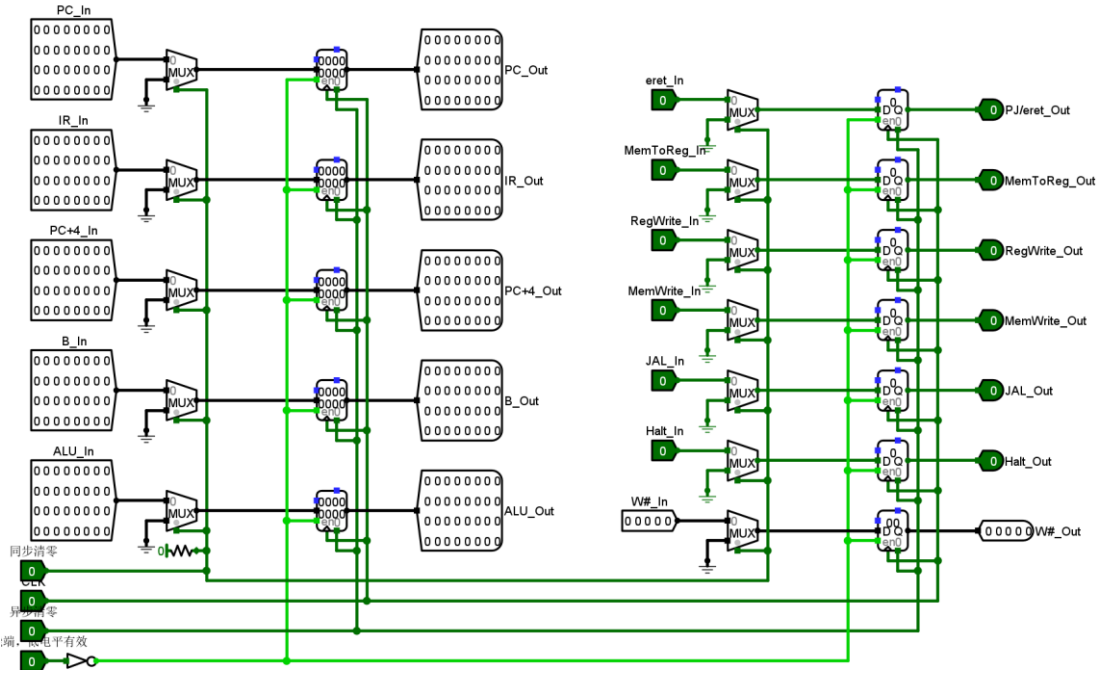

图 3.15 接口部件实现

3.3.2 理想流水线实现

因为理想流水线不考虑数据冲突以及跳转指令，所以仅需将原先的理想单周期 CPU 分块即可。因为我电路图画的比较大，这里就展示部分。如图 3.16.

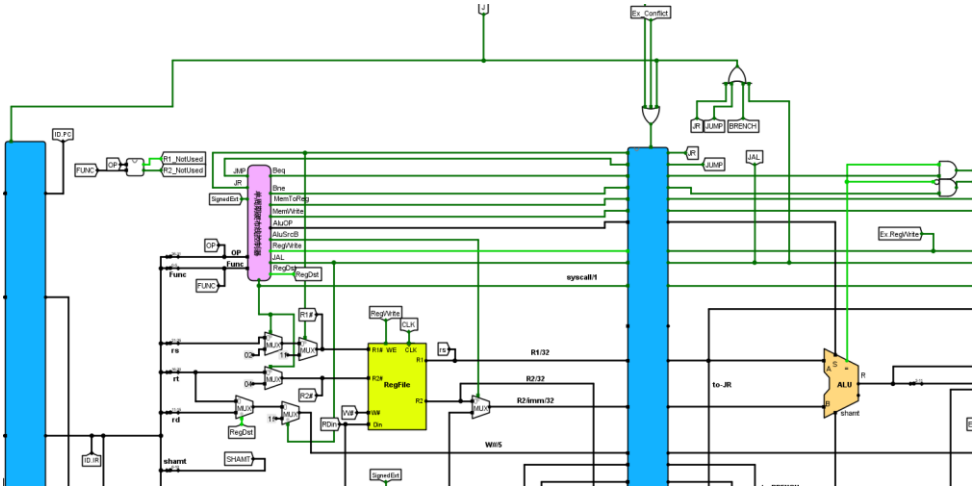

图 3.16 理想流水线部分展示

3.4 气泡式流水线实现

气泡流水线相较于理想流水线解决了数据冲突以及跳转问题，在理想流水线的基

础上，首先需要建立冲突检测机制，如图 3.17。观测 ID 阶段寄存器的读，是否和 EX 与 MEM 阶段的写寄存器发生冲突。如果冲突则接口部件置 0，加入气泡。

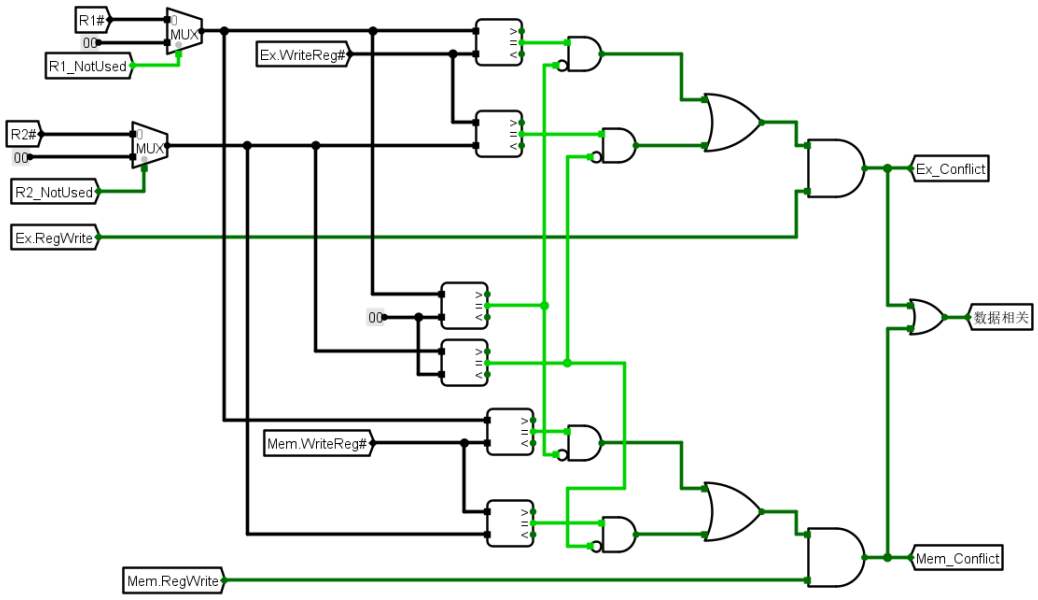

图 3.17 气泡流水线冲突检测

第二点是跳转指令检测，因为理想流水线 PC 每阶段+4，没考虑跳转指令情况。这里我在 EX 阶段检测了跳转指令的发生，若发生则 IF，ID 阶段生成气泡，同时将跳转 PC 输入 PC 寄存器。

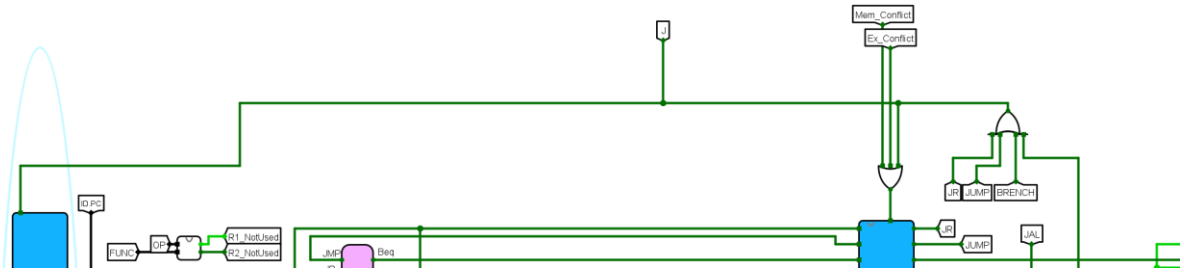

图 3.18 气泡流水线跳转处理

3.5 数据转发流水线实现

数据重定向流水线主要加入数据重定向路径处理以及 load-use 数据冲突检测。

首先是数据重定向机制，这里首先构造 forward 模块，根据对比 EX，MEM，WB 阶段的寄存器使用情况进行冲突检测，实现如下：

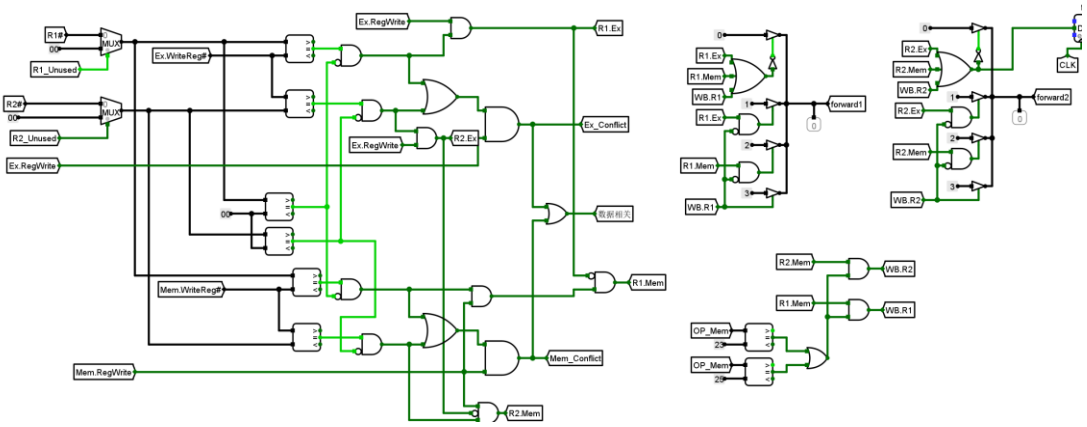

图 3.19　forward 模块

根据冲突检测出的 forward 信号进行数据的重定向，如下图：

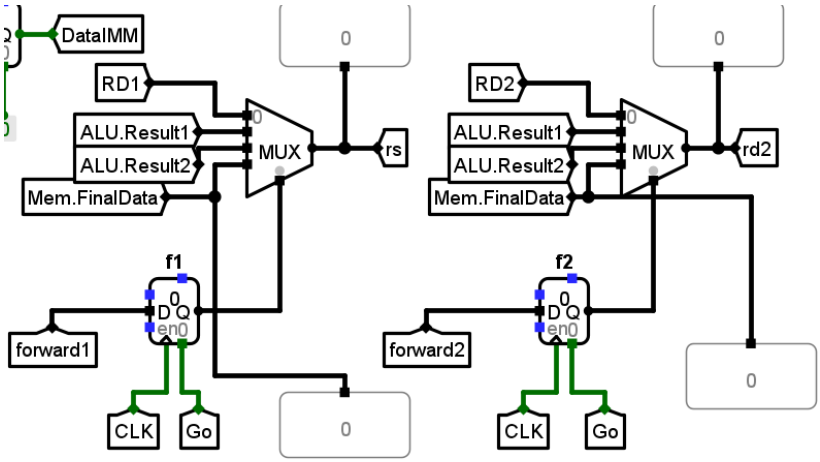

图 3.20　重定向

这些重定向的数据会被输送至 EX 阶段的 ALU 中，如果没有冲突，则正常选择。若发生冲突，则多路选择器会根据 forward 信号选择正确的寄存器值。ALU 输入的结构如下：

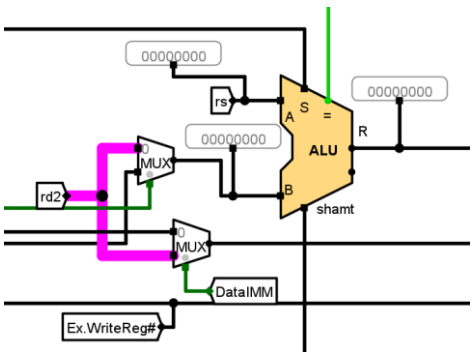

图 3.21　ALU 输入被替换，而非与接口部件直接相连

最后，load-use 冲突检测实现如下：

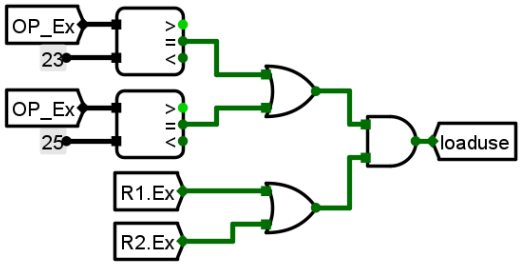

图 3.22　load-use 检测

3.6 动态分支预测机制实现

3.6.1 BHT 信息存储结构

在进行动态分支预测时，首先需要考虑 cache 结构，这里我们保存历史分支预测地址以及目标分支地址，同时，和 LRU 策略一样，我们需要考虑有效位 valid、标签位 Tag、淘汰计数 C。

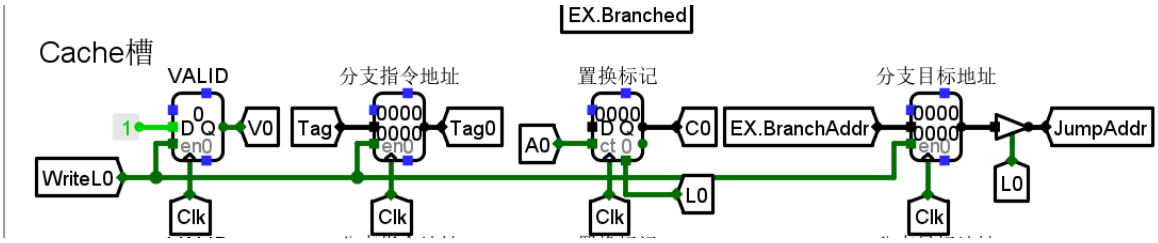

图 3.23　cache 槽

3.6.2 BHT 中 IF / EX 交互逻辑的更新

这里为了统一，我都封装在了 BHT 里，对于 EX 跳转成功的判断如下：这里就是根据 EX 阶段的 PC 值，来对 cache 中现有的预测地址进行判断，如果成功，则不变；如果失败，则进入替换策略如图 3.25.

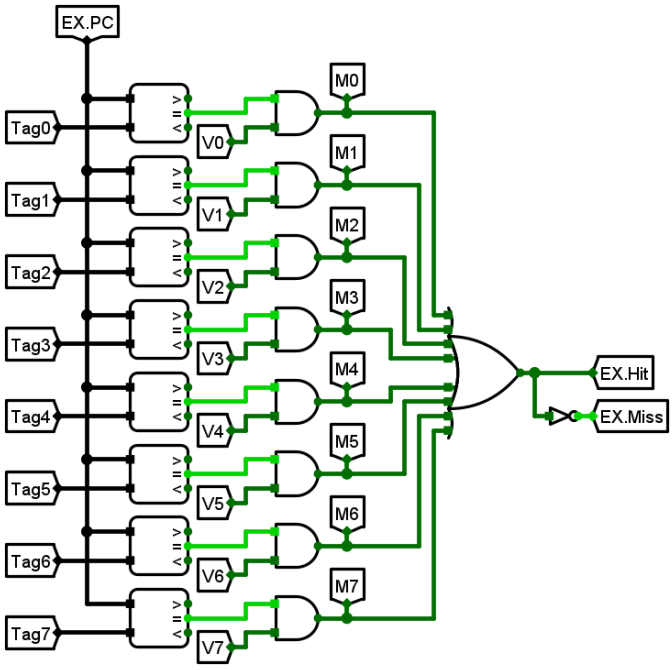

图 3.24　EX 检验跳转逻辑

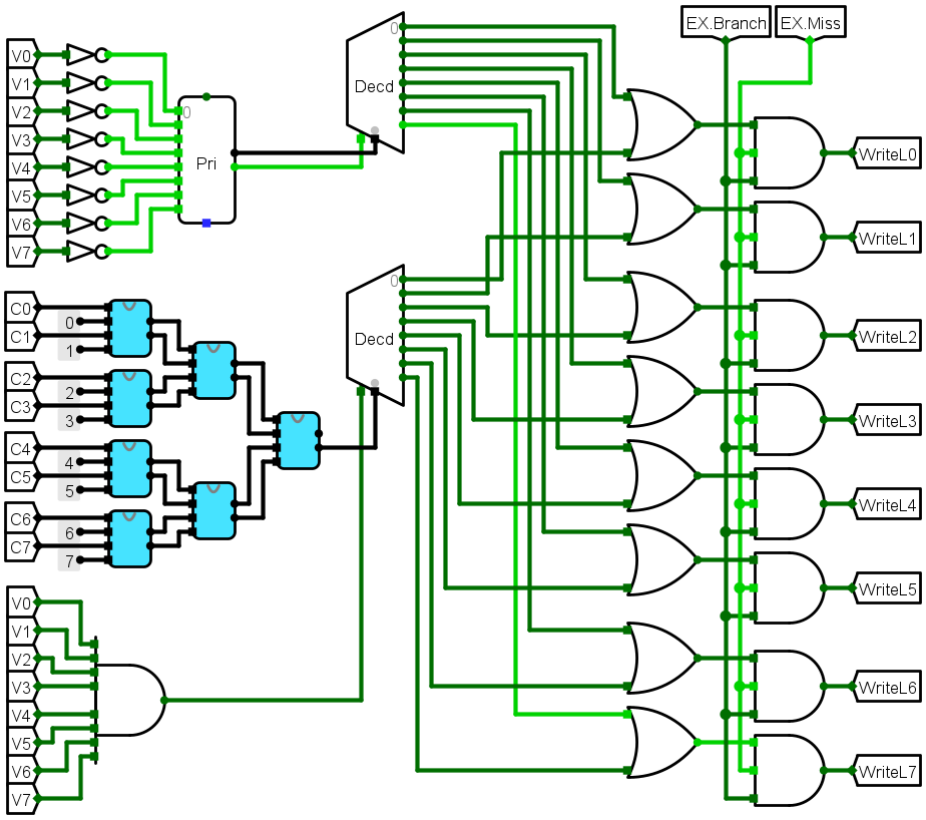

图 3.25 跳转失败替换策略

3.6.3 预测跳转

当命中 BHT 时需要根据命中的转态进行是否跳转的预测，逻辑比较简单，就是状态位的高位，而在出现跳转指令时更新目标地址时同时也要更新状态，根据是否真的跳转，此处状态设置为“3”和“1”。生成的新状态具体作为哪个存储结构的输入端要根据淘汰算法算出的编号利用解码器进行选择。

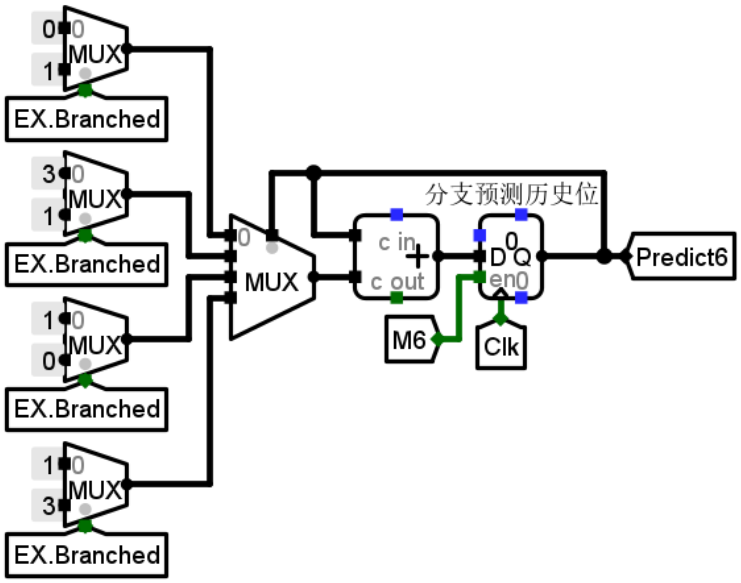

图 3.26 分支预测历史位

4 团队任务

4.1 总体介绍

我们的团队任务是做一个具有交互功能的视频播放器，如下：

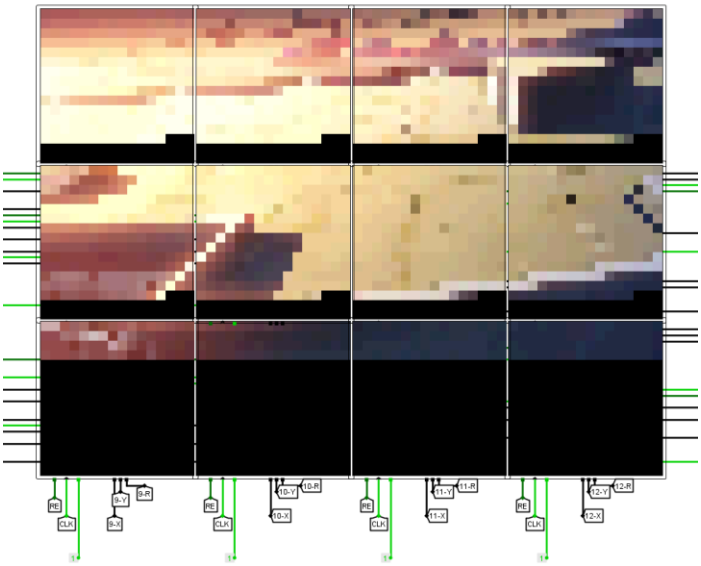

图 4.1 视频播放器

总体流程：

0.逐帧解析视频，按时间顺序存储每一个像素点的信息至 hex 文件

1.汇编指令顺序读取 ROM 指令，也就是视频每一个像素地址

2.从 ROM 中读取像素信息，包括 X，Y，RGB 值

3.输送至显示屏显示

4.视频播放辅助功能：快进，后退等

5.通过键盘对显示屏进行操控

（详细信息可见我们的 B 站视频。）

4.2 负责部分

我主要负责汇编代码的编写以及硬件部分的实现。

首先，汇编代码较为简单，因为视频已经事先逐帧解析过了，并装入了 hex 文件里，我所要做的就是将其按地址位 0 开始读取信息即可，因此，汇编代码如下：

```
addi $s2,$zero, 0
start:
lw $s1, 0($s2)
addi $s2, $s2, 4
j start
```

图 4.2 汇编代码

有了简单的读取指令，就需要设置相应的硬件实现，我们这里采用的是基于单周期 CPU 来实现视频播放器。但因为 logism 本身频率上不去，考虑到速度原因，我们将视频分割为 12 快，用十二个显示器拼接起来并行显示，这样，显示时间（显示器刷新时间）就缩短了 12 倍。其中一个部分实现如下图：

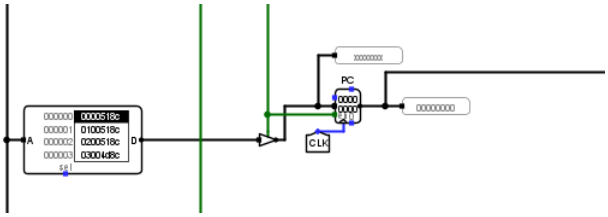

图 4.4 加载视频信息

同时，因为基于单周期而非流水线 CPU 实现，还需要考虑 ROM 读取与 MemToReg 信号同步问题，这里我通过三态门与 PC 寄存器解决。

最后，我还添加了一些辅助功能。比如快进与后退，通过对 S1 寄存器，也就是地址指针修改实现。如下：

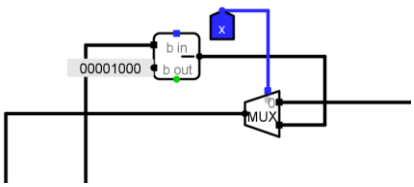

图 4.5 视频播放后退实现

5 实验过程与调试

5.1 主要故障与调试

5.1.1 地址位故障

ROM 中读取的 32 位地址错误

故障现象： 代码输出为乱码，指令不能正常执行

原因分析： 指令为 32 位，占 4 字节，所以地址选择中应该舍弃后两位，也就是取指令时取整。同时由于 logism 自身 ROM 限制，地址位最高位 10 位。所以在取指令时应取 2-11 位。

解决方案：如下图 4.1，添加位选择即可.

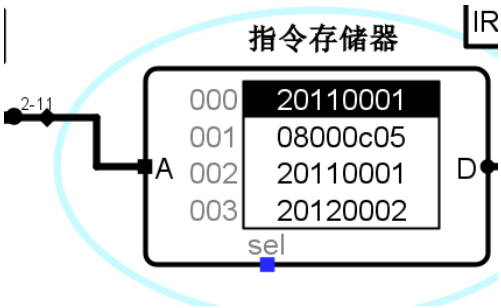

图 5.1 取指令地址限制在 2-11 位

5.1.2 BHT 淘汰策略故障

BHT 中指令存储 cache 更新，错误的在 IF 段直接更新。应该与 EX 段 PC 对比后进行。

故障现象： 不易被发现，但会有异常多的气泡

原因分析： 在程序运行时，不能一直去更新 BHT 中的记录，因为很多并非涉及到跳转指令，如果每次都更新 BHT 的话，淘汰信号更新太快，正确的地址不能得到保存。

解决方案：更新淘汰信号机制，具体为使用两个寄存器，一个以是否命中为使能，一个以跳转指令发生为使能，使得真正需要装入块时更新。如下图：

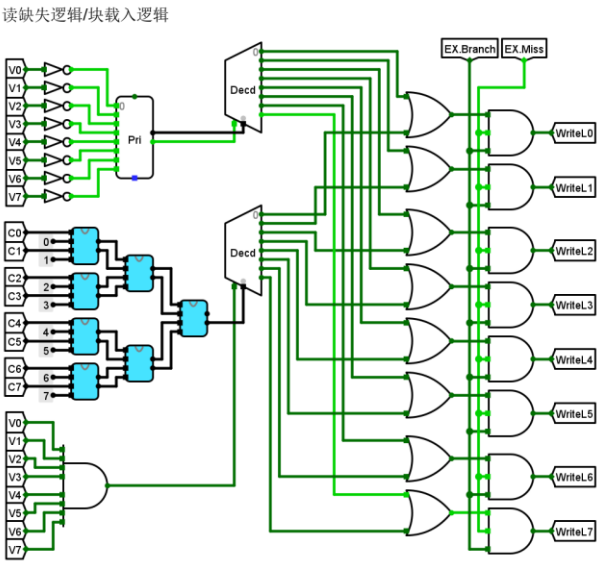

图 5.2 淘汰机制

5.1.3 气泡流水线跳转不同步

跳转失败，依旧每次 PC+4

故障现象： 跳转指令不能执行，依旧每次读取 PC+4 地址的指令

原因分析： 进行单步调试，发现每次遇到跳转指令可以正常解析，但是跳转信号总是在 ID 阶段向后传，而跳转地址放在了 EX 阶段去计算。导致跳转信号与地址的传送不同步。在地址传送回来之后，跳转信号没有了，故 PC 每次+4。

解决方案：检查流水线所有涉及选择的部件，将其选择信号与备选的值都放在一个阶段取计算.。更改如下。

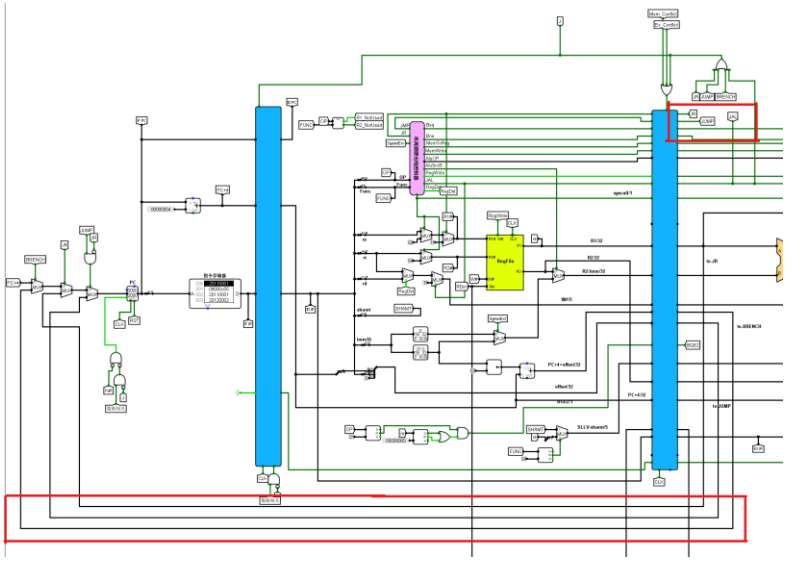

图 5.3 跳转信号与地址同步传送

5.1.4 团队任务地址读取故障

团队任务做视频播放器，读取视频 RGB 等数据时地址出现错乱

故障现象： 地址不按顺序读取，是乱序的

原因分析： 结合团队任务的汇编代码，应该是每次 PC+4 后读取，代码没有问题。在 CPU 单独调试，发现第一次读取时，由指令控制器先解析 MemtoReg 控制信号，而地址信息需要经过 Alu 运算器，导致出现不同步，核心原因还是因为使用单周期 CPU。

解决方案：我使用三态门加 PC 寄存器来解决此问题，实现如下：其中绿线为 MemToReg 信号

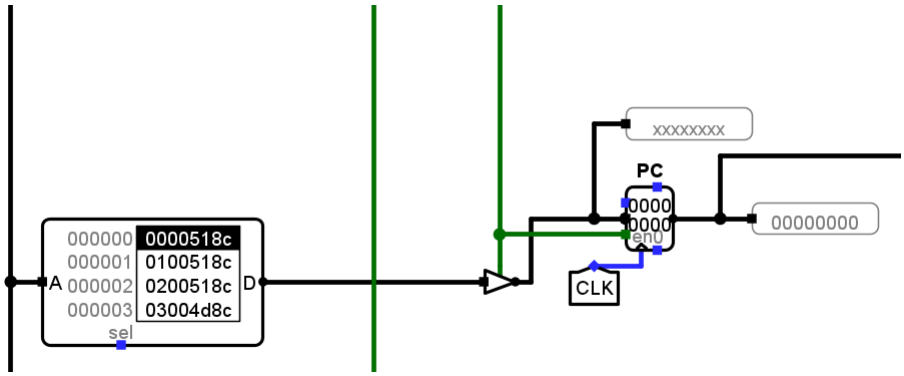

图 5.4 地址读取

5.2 实验进度

表 5.1 课程设计进度表

时间	进度
第一天	复习组成原理 CPU 相关理论知识，阅读课设任务书，阅读 MIPS 指令手册，并列出 CPU 各部件的数据通路表，并完成数据通路的基本构建。
第二天	完成单周期 CPU 的控制信号表，使用 Logisim 搭建控制器，实现了单周期 CPU 并且通过了测试。完成部分 Logism 单周期 CPU 故障报告。
第三天	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。

时间	进度
第六天	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。
第九天	完成冒险处理中的数据冲突和分支处理。

6 设计总结与心得

6.1 课设总结

本次实验主要的工作与收获：

（1）通过单周期 CPU 的制作了解了 MIPS 指令与硬件交互，同时遇到了指令信号控制器生成，地址处理等问题，通过同学与老师的沟通都解决了。

（2）通过理想流水线的搭建，理解了 CPU 的一种高效结构，通过将功能分区，提高 CPU 对指令的运行效率。同时也遇到了如跳转信号因为流水线而不同步的 bug，通过解决这些问题，使我对实验的理解更深入了。

（3）通过对气泡流水线的处理，解答了我心中原先对于理想流水线的数据冲突与指令跳转的 bug 的疑惑。

（4）通过对重定向流水线的搭建，我学习到了数据重定向技术，掌握了更进一步的流水线处理方式。

（5）解决了 CPU 对于单级中断与多级中断的处理，通过这一实验，我了解到了硬件层面上的 PC 存储栈的构建，以及中断优先级的判断与使用。

（6）最后的动态分支预测流水线，我实现了分支地址与预测位的存储，以及 LRU 策略的实际应用，还解决了判断预测成功失败等信号关于阶段的 bug。掌握了真正贴近高效的 CPU 设计思想。

6.2 课设心得

我认为本次课设算是大学中课程设计里面最难的几项之一了。唯一值得慰藉的是这项课设的参考资料还算丰富，如网上关于流水线，单周期的资料，以及课本，慕课等、通过学习是可以从根本上掌握每一种设计思路的原理的。完成本次实验后，我认为最值得回味的便是满满的成就感了。

在这次实验中，我不仅了解了 MIPS 指令的构造及其汇编代码的使用，同时，更是结合到了硬件层面，这是我对 CPU 的工作流程有了一个更深的理解，我认为这种理解是每一个计算机专业学生所必备的，虽说可能未来我们并不会去深入研究这方面的工作。但它确实整个计算机的基础。同时，了解 CPU 与指令的交互也有助于我们

构造上层代码，比如在本次实验中的动态分支预测我就了解到了关于数据冲突，跳转等在硬件层面所面临的约束，这种约束在 CPU 上无法得到最高效的解决，若想编写出更优秀的代码，我们可以在上层写代码时，就注意此类问题。

在团队任务上，我也更进一步的了解到了 CPU 的伟大之处，只有真正通过编写汇编代码，然后实现在自己运行的 CPU 上，才能最深刻地体会到本次课设的意义以及 CPU 本身的强大。我们实现了一个基于单周期 CPU 的视频播放器，可以并行播放视频，同时具有用户交互界面。我在团队任务中负责硬件的实现与汇编代码的编写。虽说团队任务不是很深入，就是在原有的 CPU 基础上进行了修改，但我认为对于我自身的理解也有很大的促进作用。

本次课程设计我认为存在很有必要，但同时我也希望老师学长们多提供写参考资料，一开始我在做动态分支与测试，真的是从 0 开始，所有资料都要从网上找。希望组员课设参考一下 OS 课设做一个 Gitee 的参考文档，由有能力的学生去不断完善，分享他们的经验与思路，可以额外给他们加分。这样理解较浅的同学可以更容易地去学会课设设计，能力强的同学也避免了卷就能加分。

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：刘庆远