华中科技大学

2021

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：	CS1807
学　　号：	U201814682
姓　　名：	李世铭
电　　话：	13125077740
邮　　件：	1074822720@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	
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 $v0==50 暂停，等待按键
24	SYSCALL	系统调用	else 数码管显示$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	SLTIU	小于立即数置 1(无符号)	
30	LHU	加载半字(无符号)	
31	BLTZ	小于 0 转移	

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硬布线控制的单周期 CPU，将系统分成指令存储器 IM、控制器、寄存器文件 RegFile、运算器 ALU 和数据存储器 DM 几个模块。系统运行时，每个时钟周期，根据 PC 寄存器中的地址从指令存储器中取出一条指令，硬布线控制器根据指令生成数据通路的控制信号和运算器功能选择信号 ALU_OP，寄存器文件的输出作为 ALU 的操作数，ALU 的运算结果送到数据存储器或者寄存器文件，根据控制信号和地址转移逻辑得到下一条指令的地址，在下一个时钟周期到来时送到 PC 寄存器中进入下一个指令周期。由于指令存储器和数据存储器分离，所以避免了单周期CPU在一个时钟周期内的访存冲突。总体结构图如图 2.1 所示。

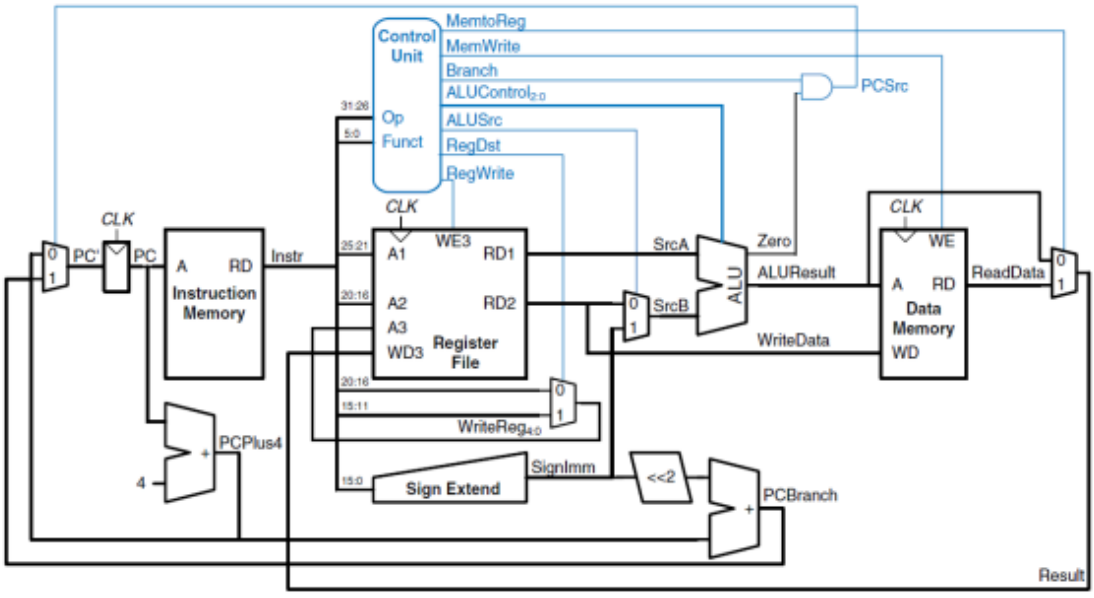

图 2.1 总体结构图

2.1.1 主要功能部

1. 程序计数器 PC

程序计数器储存指令的地址，根据控制信号和地址转移逻辑得到的下一条指令地址作为 PC 的输入，PC 的时钟端需要接时钟信号，PC 的使能端也需要接控制信号，

当有暂停信号时 PC 忽略时钟输入，暂停程序运行，收到 Go 时需要接收下一条指令地址，从暂停中恢复，PC 的输出端输出当前指令地址。输入输出引脚功能描述如表 2.1 所示。

表 2.1 程序计数器引脚与功能描述

引脚	输入/输出	位宽	功能描述
CLK	输入	1	时钟信号
EN	输入	1	使能端，为 0 时忽略时钟信号
PCin	输入	32	下一条指令的地址
PCout	输出	32	当前指令的地址

2. 指令存储器 IM

指令存储器用于存储程序，由于程序运行中不会修改指令，所以使用只读 ROM 作为指令存储器，MIPS 32 地址总线时按字节编址，一条指令大小是 4 个字节，所以对输入端的 PC 的低 2 位进行屏蔽，使得取指令工作能正常进行。因为测试集程序规模都不是很大，所以可以 ROM 的地址位宽可以只用 10 位，相应地，输入端的 PC 只取 2-11 位。输入输出引脚功能描述如表 2.2 所示。

表 2.2 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	指令地址
IR	输出	32	PC 地址处的指令

3. 运算器

运算器有两个 32 位的输入端 X 和 Y，X 和 Y 进行除位移以外的运算，并将结果输出到输出端，位移运算由位移量 shamt 和输入 X 完成，根据运算器操作码 ALU_OP 决定具体进行什么运算，此外如果 X 和 Y 相等，运算器还会输出一个相等信号，运算器输入输出引脚功能描述如表 2.3 所示。

表 2.3 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，决定运算器的运算方法，具体功能见下
shamt	输入	5	位移运算时的位移量
Result	输出	32	ALU 运算结果
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

运算器功能表，如表 2.4 所示。

表 2.4 运算器功能表

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

寄存器堆中有 MIPS 的 32 个寄存器，每个寄存器都是 32 位，有 2 个读寄存器选择端 R1#、R2#和 1 个写寄存器选择端 W，选择端位宽都为 5，2 个 32 位输出端 R1、R2 输出 R1#、R2#指向的 2 个寄存器数据，并且当时钟信号 CLK 到来时，如果写使能端 WE 为 1，将输入的 32 位数据 Din 写到 W 指向的寄存器中。寄存器堆的输入输出引脚功能描述如表 2.5 所示。

表 2.5 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	时钟信号
R1#	输入	5	读寄存器 1 选择信号
R2#	输入	5	读寄存器 2 选择信号
W	输入	5	写寄存器选择信号
WE	输入	1	写使能信号
Din	输入	32	待写入的数据
R1	输出	32	读寄存器 1 的数据
R2	输出	32	读寄存器 2 的数据

5. □据存储器 DM

数据存储器在程序运行中既会读又会写，使用随机存储器 RAM 作为数据存储器，数据存储器有 32 位的数据输入输出端 Din、Dout，地址端 A,与指令存储器同样使用 10 位的地址位宽，写使能 str 和时钟信号 CLK，而读端随时可读，加载使能始终置 1。数据寄存器的输入输出引脚功能描述如

表 2.6 数据存储器引脚与功能描述

引脚	输入/输出	位宽	功能描述
CLK	输入	1	时钟信号
Din	输入	32	输入端的数据
A	输入	10	读写数据的地址
str	输入	1	写使能信号

引脚	输入/输出	位宽	功能描述
Dout	输出	32	输出的数据

2.1.2 数据通路的设计

对于电路中的主要功能部件，忽略控制信号，记录数据通路中的数据信号，得到指令系统数据通路框架表，如表 2.7 所示。

表 2.7 指令系统数据通路框架

指令	PC	IM	RF				ALU				DM	
			R1#	R2#	W#	Din	A	B	shamt	OP	Addr	Din
SLL	PC+4	PC		rt	rd	ALUout		R2	IR[10:6]	AluOP		
SRA	PC+4	PC		rt	rd	ALUout		R2	IR[10:6]	AluOP		
SRL	PC+4	PC		rt	rd	ALUout		R2	IR[10:6]	AluOP		
ADD	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
ADDU	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
SUB	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
AND	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
OR	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
NOR	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
SLT	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
SLTU	PC+4	PC	rs	rt	rd	ALUout	R1	R2		AluOP		
JR	R1	PC	rs									
SYSCALL	PC+4	PC	2	4								
J	PC[31:28]||IR [25:0]||00	PC										
JAL	PC[31:28]|| IR	PC			31	PC+4						

指令	PC	IM	RF				ALU				DM	
			R1#	R2#	W#	Din	A	B	shamt	OP	Addr	Din
BEQ	R[rs]=R[rt]? PC+sign_extend(IMM||00): PC+4	PC	rs	rt			R1	R2				
BNE	R[rs]!=R[rt]? PC+sign_extend(IMM||00): PC+4	PC	rs	rt			R1	R2				
ADDI	PC+4	PC	rs		rt	ALUout	R1	sign_extend(IMM)		AluOP		
ANDI	PC+4	PC	rs		rt	ALUout	R1	sign_extend(IMM)		AluOP		
ADDIU	PC+4	PC	rs		rt	ALUout	R1	zero_extend(IMM)		AluOP		
SLTI	PC+4	PC	rs		rt	ALUout	R1	sign_extend(IMM)		AluOP		
ORI	PC+4	PC	rs		rt	ALUout	R1	sign_extend(IMM)		AluOP		
LW	PC+4	PC	rs		rt	DMout	R1	sign_extend(IMM)		AluOP	ALUout	
SW	PC+4	PC	rs	rt			R1	sign_extend(IMM)		AluOP	ALUout	R2
SLLV	PC+4	PC	rs	rt	rd	ALUout		R2	R1[4:0]	AluOP		
SLTIU	PC+4	PC	rs		rt	ALUout	R1	sign_extend(IMM)		AluOP		
LHU	PC+4	PC	rs		rt	zero_extend(ALUout[1]?	R1	sign_extend(IMM)		AluOP	ALUout	

指令	PC	IM	RF				ALU				DM	
			R1#	R2#	W#	Din	A	B	shamt	OP	Addr	Din
						DMout[31:16]: DMout[15:0])						
BLTZ	R1<0?PC+sign_extend(IMM||00): PC+4	PC	rs									

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 2.8。

表 2.8 主控制器控制信号的作用说明

控制信号	取值	说明
SignedExt	0	对指令字中的立即数 IMM 进行 0 扩展
	1	对指令字中的立即数 IMM 进行符号扩展
JR	0	不进行 JR 跳转
	1	进行 JR 跳转，将寄存器文件的 R1 送到 PC
JMP	0	不进行 JMP 跳转
	1	进行 JMP 跳转，将 PC[31:28]||IR[25:0]||00 送到 PC
Beq	0	不进行 BEQ 跳转
	1	若 ALUeuqal 为 1 进行 BEQ 跳转，将 PC+sign_extend(IMM||00)送到 PC
Bne	0	不进行 BNE 跳转
	1	若 ALUeuqal 为 0 进行 BNE 跳转，将 PC+sign_extend(IMM||00)送到 PC
MemToReg	0	将其他数据送入寄存器文件的 Din，默认为 ALUout
	1	将数据存储器的输出 DMout 送入寄存器文件的 Din
MemWrite	0	数据存储器 DM 写使能为 0，不写入数据，可以读数据
	1	数据存储器 DM 写使能置 1，向 DM 写入数据
AluOP	0000-1100	具体功能见表 2.4

控制信号	取值	说明
AluSrcB	0	将寄存器文件的 R2 送入 ALU 的输入 Y
	1	将扩展后的 IMM 送入 ALU 的输入 Y
RegWrite	0	寄存器文件的写使能为 0，不写入数据
	1	将寄存器文件的写使能置 1，将 Din 的数据写入 W#号寄存器
JAL	0	不进行 JAL 跳转
	1	进行 JAL 跳转，将 PC+sign_extend(IMM||00)送到 PC，将 PC+4 送到 31 号寄存器
RegDst	0	将 rt 送到 W#
	1	将 rd 送到 W#
Syscall	0	不进行系统调用，R1#和 R2#的数据使用默认的 rs 和 rt
	1	进行系统调用，将 2 送到 R1#，将 4 送到 R2#，并比较 R1 和 50，若相等暂停并等待按键，若不相等显示 R2 的数据
SLLV	0	将默认的数据送到 shamt
	1	将 R1[4:0]送到 ALU 的 shamt
LHU	0	将默认的数据送到寄存器堆的 Din
	1	将 DM 输出的根据 ALUout[1]选择半字 0 扩展后的数据送到寄存器堆的 Din
BLTZ	0	不进行 BLTZ 分支跳转
	1	若 R1 小于 0 进行 BLTZ 跳转，将 PC+sign_extend(IMM||00)送到 PC

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 2.9 所示。

表 2.9 主控制器控制信号框架

指令	ALU_OP	MemtoReg	MemWrite	ALU_SRC	RegWrite	SYSCALL	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL	SLLV	LHU	BLTZ
SLL	0				1			1								
SRA	1				1			1								

SRL	2				1			1								
ADD	5				1			1								
ADDU	5				1			1								
SUB	6				1			1								
AND	7				1			1								
OR	8				1			1								
NOR	10				1			1								
SLT	11				1			1								
SLTU	12				1			1								
JR	X										1					
SYSCALL	X					1										
J	X											1				
JAL	X				1								1			
BEQ	X						1		1							
BNE	X						1			1						
ADDI	5			1	1		1									
ANDI	7			1	1											
ADDIU	5			1	1		1									
SLTI	11			1	1		1									
ORI	8			1	1											
LW	5	1		1	1		1									
SW	5		1	1			1									
SLLV	0				1			1						1		

SLTIU	12			1	1		1									
LHU	5	1		1	1		1								1	
BLTZ	X						1									1

2.2 流水 CPU 设计

2.2.1 总体设计

单周期 CPU 的时钟频率取决于数据通路中的最长路径，所以单周期 CPU 的执行效率很低，可以利用流水线技术，将一个任务划分成多个子过程，多个任务的不同阶段的子过程同时在不同部件上执行，缩短数据通路的关键路径，提高 CPU 的时钟频率从而提高执行效率。

MIPS 指令系统的指令执行过程可以分成五个阶段：取指令（IF）、指令译码（ID）、指令执行（EX）、访存（MEM）和写回（WB），利用流水接口部件可以将指令划分到不同阶段执行，MIPS 指令流水线逻辑架构如图 2.2 所示。

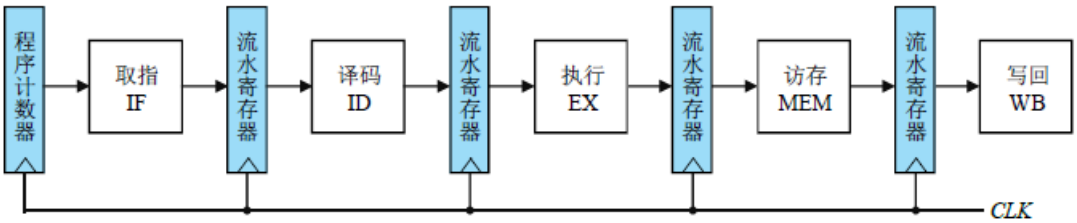

图 2.2 MIPS 指令流水线逻辑架构图

2.2.2 流水接口部件设计

流水接口部件的作用是将上个阶段应该送到下个阶段的数据和控制信号锁存一个周期，以便上个阶段的运行结果能够在下一个时钟周期给下一个阶段使用。流水接口部件使用寄存器实现，接上时钟信号，每个寄存器存储一个控制信号或者数据，根据一个控制信号和数据的位宽选择不同位宽的寄存器。

2.2.3 理想流水线设计

理想流水线不涉及控制冲突和数据冲突，将单周期 CPU 的各个部件按功能划分到流水线的不同阶段中，得到理想流水线的设计。

取指令（IF）：程序计数器 PC，指令存储器 IM。

指令译码（ID）：指令解析，立即数位扩展，控制器，寄存器文件。

指令执行（EX）：运算器 ALU。

访存（MEM）：数据存储器 DM。

写回（WB）：将数据写回寄存器文件。

2.3 气泡流水线设计

2.3.1 总体设计

理想流水线无法控制冲突和数据冲突。控制冲突是指流水线 CPU 遇到分支指令时，指令执行到 EX 端才能确定是否分支跳转和要跳转的目标地址，而顺序执行的后两个指令已经进入流水线的 IF 端和 ID 端，造成控制冲突。数据冲突对流水线 CPU 有影响的时先写后读冲突，如果需要写寄存器的指令 I1 还没有到达 WB 端真正写回寄存器堆，需要读同一个寄存器的指令 I2 已经进入 ID 读出寄存器的旧值发生数据冲突。发生冲突时，根据冲突类型将部分流水接口部件的数据清空（称为插入气泡），并将流水线阻塞直到冲突指令执行完成来解决冲突的流水线称为气泡流水线。

2.3.2 控制冲突处理

将分支跳转的控制信号的目标地址统一放在 EX 段送回 IF 段，控制冲突的检测比较容易，当有任何一个分支跳转的控制信号传回 IF 段时，可以判定发生了分支跳转。发生控制冲突时，需要将已经入流水线的顺序指令清空，清空 IF/ID 和 ID/EX 两个流水接口部件的寄存器。

2.3.3 数据冲突处理

需要设计数据相关性检测组合逻辑，检测 ID 段与 EX、MEM 段指令的数据相关，如果 ID 段的指令的读寄存器是 EX 或 MEM 段指令要写入的寄存器那么判定为数据相关。发生数据相关时，利用使能端将 PC 和 IF/ID 流水接口部件锁存并将 ID/EX 流水接口部件的寄存器清空。

2.3.4 流水接口部件改造

在气泡流水线中，流水接口部件需要增加功能，收到清空信号时，需要在下个时钟周期将寄存器清空，收到阻塞信号时，将寄存器使能端置 0，忽略时钟输入，达到阻塞数据的作用。

2.4 重定向流水线设计

2.4.1 总体设计

气泡流水线虽然可以解决冲突问题，但是需要插入大量的气泡，浪费很多时钟周期，严重影响流水线 CPU 的性能。在气泡流水线的基础上利用数据重定向技术减少数据冲突中时钟周期的浪费。发生数据冲突时，ID 段读寄存器的数据是错误的，正确的数据已经在 EX/MEM 或者 MEM/WB 流水接口部件的寄存器中，利用数据旁路，在下一个时钟周期将 EX 段的 R1 和 R2 替换为正确的数据可以实现重定向，重定向示意图如图 2.3 所示。

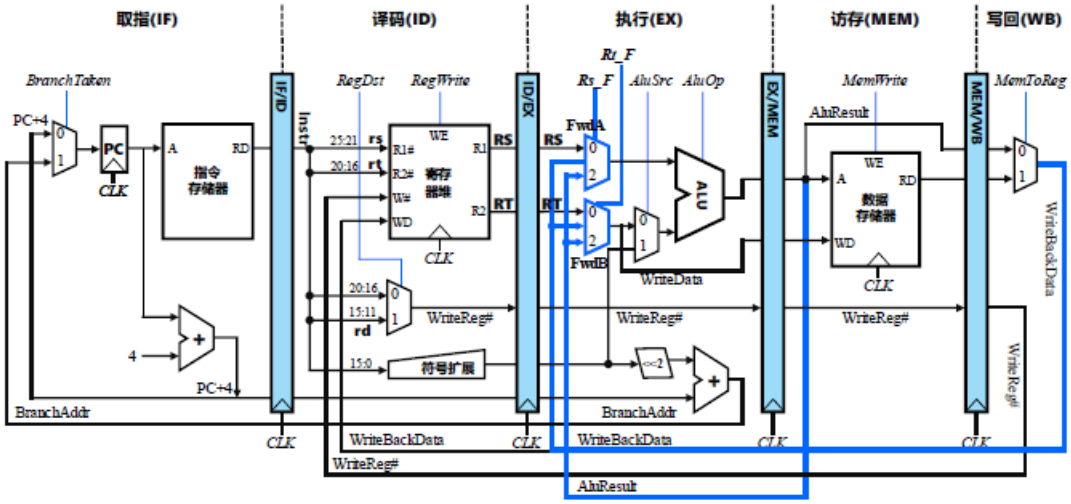

图 2.3 重定向流水线示意图

需要注意的是，若相邻两条指令数据相关，且前一条是访存指令（称为 Load-Use 相关），不能采用重定向，如图 2.4 所示，加入旁路后，EX 段的关键路径变成 MEM 的访存时延加上 ALU 的计算时延，反而使得执行效率大大降低，对于 Load-Use 相关采用和气泡流水线相同的方法，在 ID/EX 流水接口部件插入一个气泡。

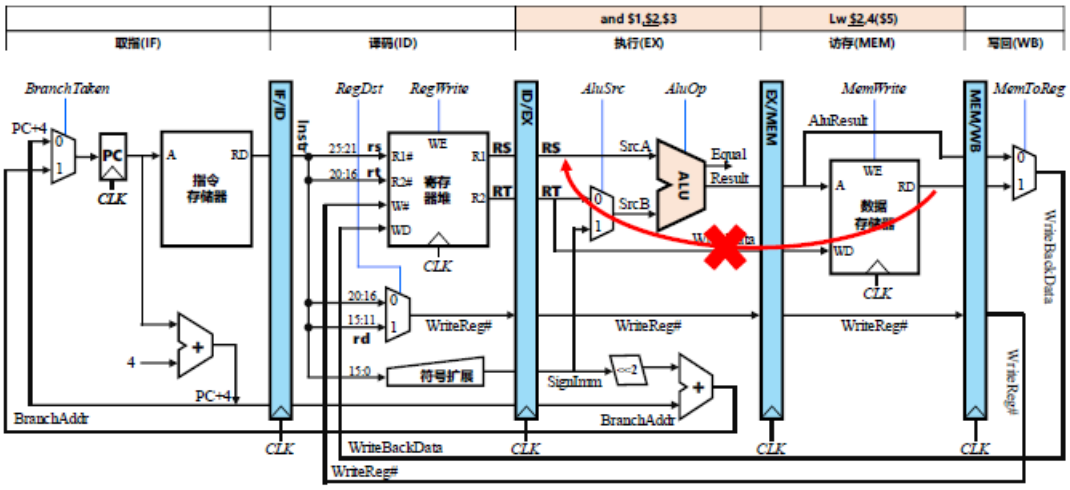

图 2.4 Load-Use 相关示意图

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）

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。Pause 为停机信号，取反后接在寄存器使能端，当需要进行停机时，Pause 控制信号为 1，经过非门之后为 0，使寄存器阻塞，点击 Go 后，使使能端重新置 1，程序恢复运行。如图 3.1 所示。

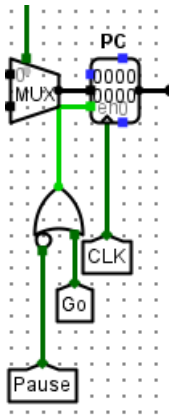

图 3.1 程序计数器（PC）

2) 指令存储器（IM）

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。

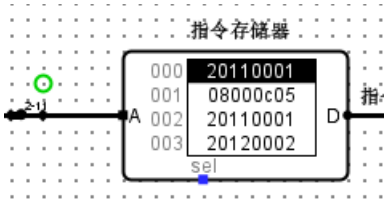

图 3.2 指令存储器（IM）

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，指令系统数据通路表如表 2.7 所示。

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，数据通路如图 3.3 所示。

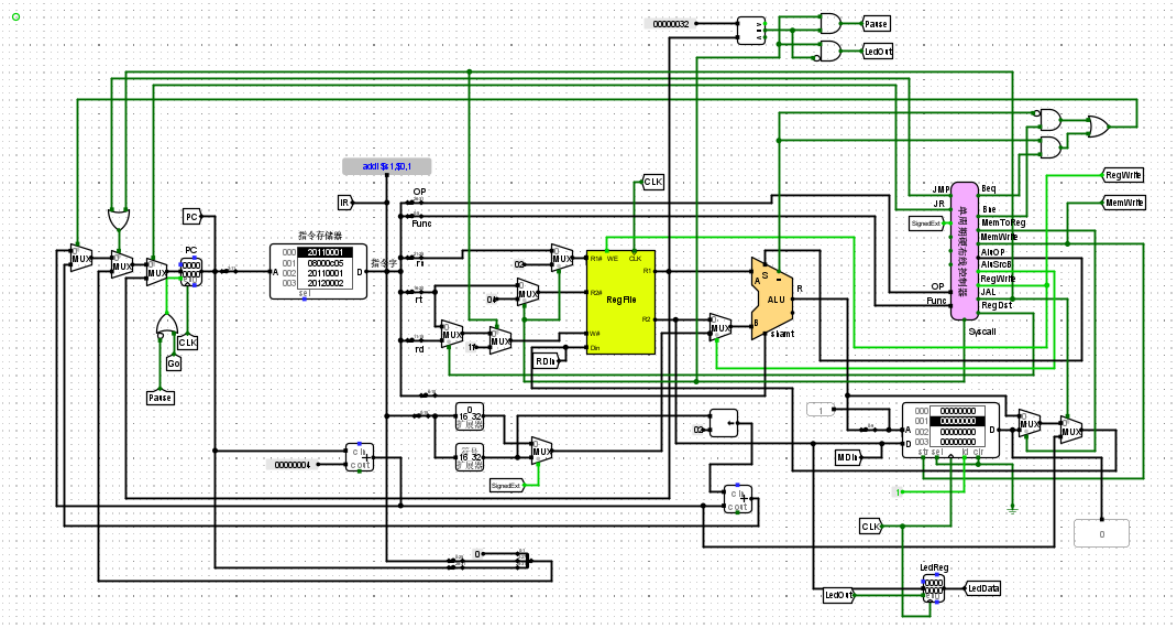

图 3.3 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

根据表 2.9 并查阅指令的 OpCode 和 FUNCT，填写硬布线控制器自动生成表，如图 3.4 所示，得到自动生成的控制信号表达式，在 logisim 中利用组合逻辑电路分析，填写表达式得到运算控制器自动生成电路和控制信号自动生成电路，利用两个电路得到硬布线控制器组合逻辑电路，如图 3.5 所示。

#	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	SLLV	LHU	BLTZ
1	SLL	0	0	0				1			1								
2	SRA	0	3	1				1			1								
3	SRL	0	2	2				1			1								
4	ADD	0	32	5				1			1								
5	ADDU	0	33	5				1			1								
6	SUB	0	34	6				1			1								
7	AND	0	36	7				1			1								
8	OR	0	37	8				1			1								
9	NOR	0	39	10				1			1								
10	SLT	0	42	11				1			1								
11	SLTU	0	43	12				1			1								
12	JR	0	8	X										1					
13	SYSCALL	0	12	X					1										
14	J	2	X	X											1				
15	JAL	3	X	X				1								1			
16	BEQ	4	X	X						1		1							
17	BNE	5	X	X						1			1						
18	ADDI	8	X	5			1	1		1									
19	ANDI	12	X	7			1	1											
20	ADDIU	9	X	5			1	1		1									
21	SLTI	10	X	11			1	1		1									
22	ORI	13	X	8			1	1											
23	LW	35	X	5	1		1	1		1									
24	SW	43	X	5		1	1			1									
25	SLLV	0	4	0				1			1						1		
26	SLTIU	11	X	12			1	1		1									
27	LHU	37	X	5	1		1	1		1								1	
28	BLTZ	1	X	X						1									1

图 3.4 硬布线控制器自动生成表

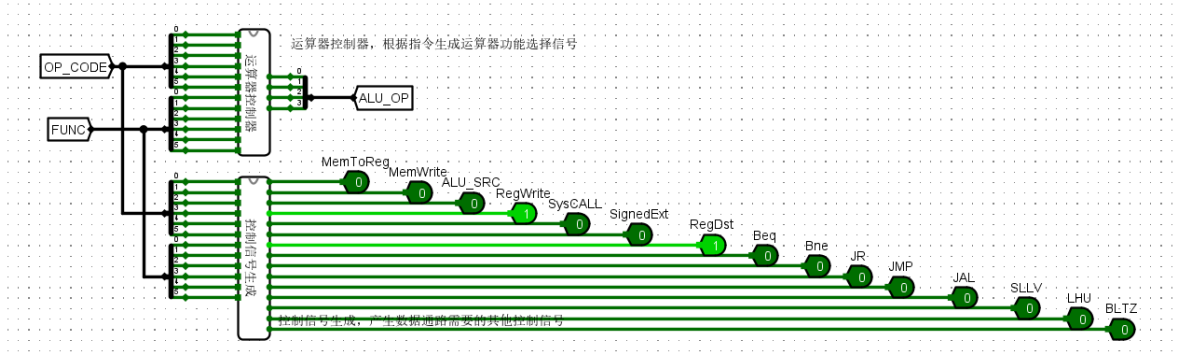

图 3.5 硬布线控制器逻辑电路

3.2 流水 CPU 实现

3.2.1 流水接口部件实现

以 IF/ID 流水接口部件为例，如图 3.6 所示，所有输入信号都用对应位宽的寄存器进行输出，时钟上升沿触发，使能端接信号 en，清零信号接多路选择器的选择端，正常情况下选择输入信号，清零信号到来时选择 0，实现同步清零。

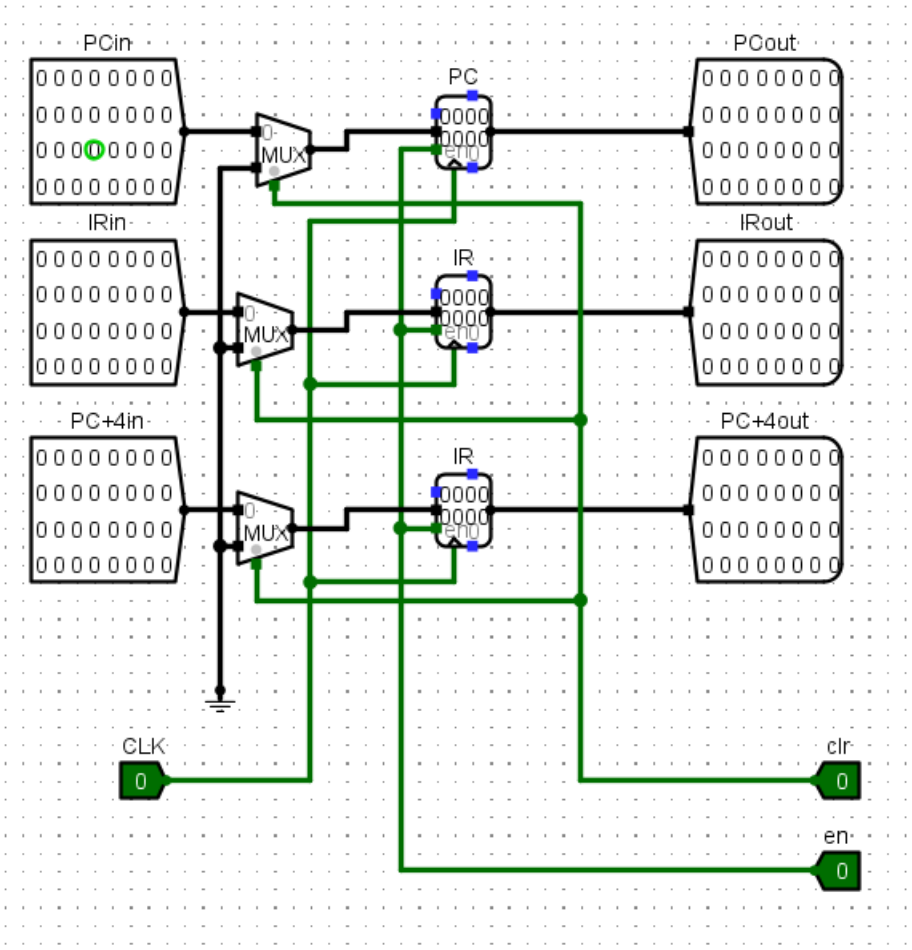

图 3.6 流水接口部件实现

3.2.2 理想流水线实现

IF 段，PC 的输出送 IM 的输入，取出的指令通过 IF/ID 流水接口部件传输到译码段，PC+4 的值也向后传输。

ID 段，利用分线器解析出指令各字段并分别送到控制器和立即数扩展器，操作

控制器输出的控制信号有部分就作用在本段，其他不需要用到的信号送往下一段，寄存器输出值、立即数扩展器的输出也向后传输。

EX 段，找到并使用本段需要用到的控制信号，剩下的控制信号继续向后传送。ALU 的运算结果、寄存器输出和 PC+4 向后传输。

MEM 段，寄存器的输出 R2 作为 DM 的输入，ALU 运算结果作为访存地址，写使能信号送到 DM 的使能端，根据控制信号 MemToReg 和 JAL 选择将访存输出、ALU 运算结果还是 PC+sign_extend(IMM||00)作为写回寄存器的值继续向后传送。

WB 段，将写入寄存器的值和写使能控制信号送回 ID 段、PC+4 的值均送回到 PC。

理想流水线数据通路如图 3.7 所示。

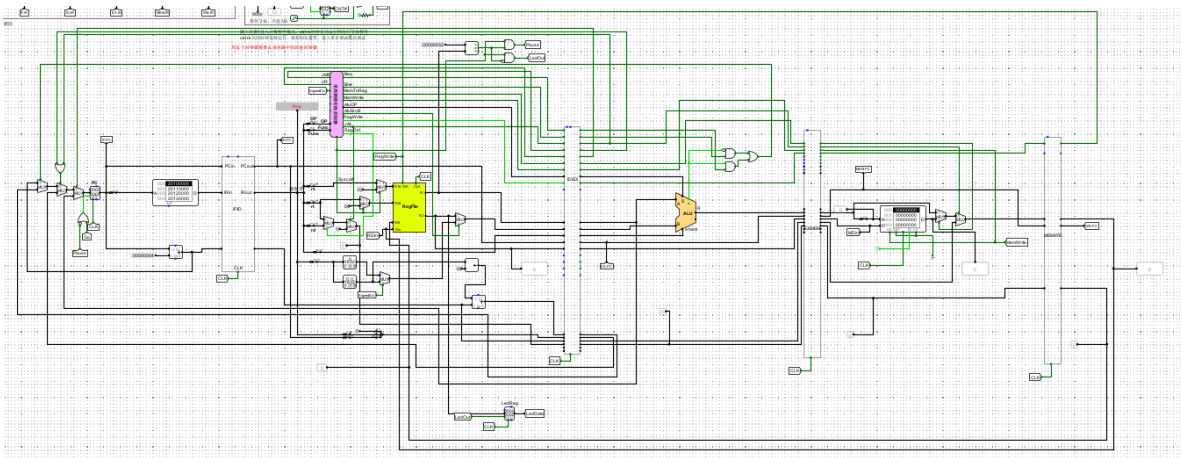

图 3.7 理想流水线数据通路

3.3 气泡流水线实现

3.3.1 控制冲突处理实现

如图 3.8 所示，分支跳转时，地址转移逻辑的多路选择器选择段控制信号会有一个值为 1，使用一个或门可以构成简单的分支跳转检测电路，输出与 IF/ID 和 ID/EX 流水接口部件清空端相连，发生分支跳转时插入俩个气泡。

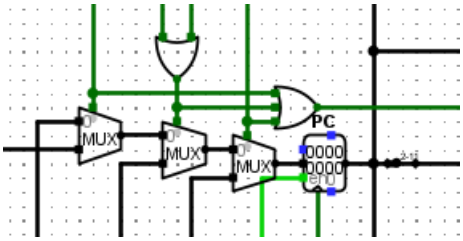

图 3.8 控制冲突处理

3.3.2 数据冲突处理实现

数据相关检测逻辑如下：

DataHazzard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#) +

RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#) +

RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#) +

RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)

#rs、rt 分别表示指令字中的 rs、rt 字段，分别对应指令字中的 25~21、20~16 位

RsUsed、RtUsed 分别表示 ID 段指令需要读 rs、rt 字段对应的寄存器

EX.RegWrite 表示 EX 段的寄存器堆写使能控制信号 RegWrite，锁存在 ID/EX 流水寄存器中

#MEM.WriteReg#表示 MEM 段的写寄存器编号 WriteReg#，锁存在 EX/MEM 流水寄存器中

RsUsed、RtUsed 需要使用 ID 段的 OP 和 FUNCT 识别指令并找到指令需要使用的寄存器文件的哪个输出，可以利用设计硬布线控制器的思想，填写自动生成表，而将控制信号改为 RsUsed 和 RtUsed 两个信号，如图 3.9 所示，得到自动生成的源寄存器查询电路。

#	指令	OpCode (十进制)	FUNCT (十进制)	RsUsed	RtUsed	Mem
1	SLL	0	0		1	
2	SRA	0	3		1	
3	SRL	0	2		1	
4	ADD	0	32	1	1	
5	ADDU	0	33	1	1	
6	SUB	0	34	1	1	
7	AND	0	36	1	1	
8	OR	0	37	1	1	
9	NOR	0	39	1	1	
10	SLT	0	42	1	1	
11	SLTU	0	43	1	1	
12	JR	0	8	1		
13	SYSCALL	0	12	1	1	
14	J	2	X			
15	JAL	3	X			
16	BEQ	4	X	1	1	
17	BNE	5	X	1	1	
18	ADDI	8	X	1		
19	ANDI	12	X	1		
20	ADDIU	9	X	1		
21	SLTI	10	X	1		
22	ORI	13	X	1		
23	LW	35	X	1		
24	SW	43	X	1	1	
25	SLLV	0	4	1	1	
26	SLTIU	11	X	1		
27	LHU	37	X	1		
28	BLTZ	1	X	1		

图 3.9 源寄存器查询自动生成

利用源寄存器查询电路，送入数据相关检测逻辑需要的流水线各个部分的数据和控制信号，根据数据相关检测逻辑得到数据相关检测电路，如图 3.10 所示。

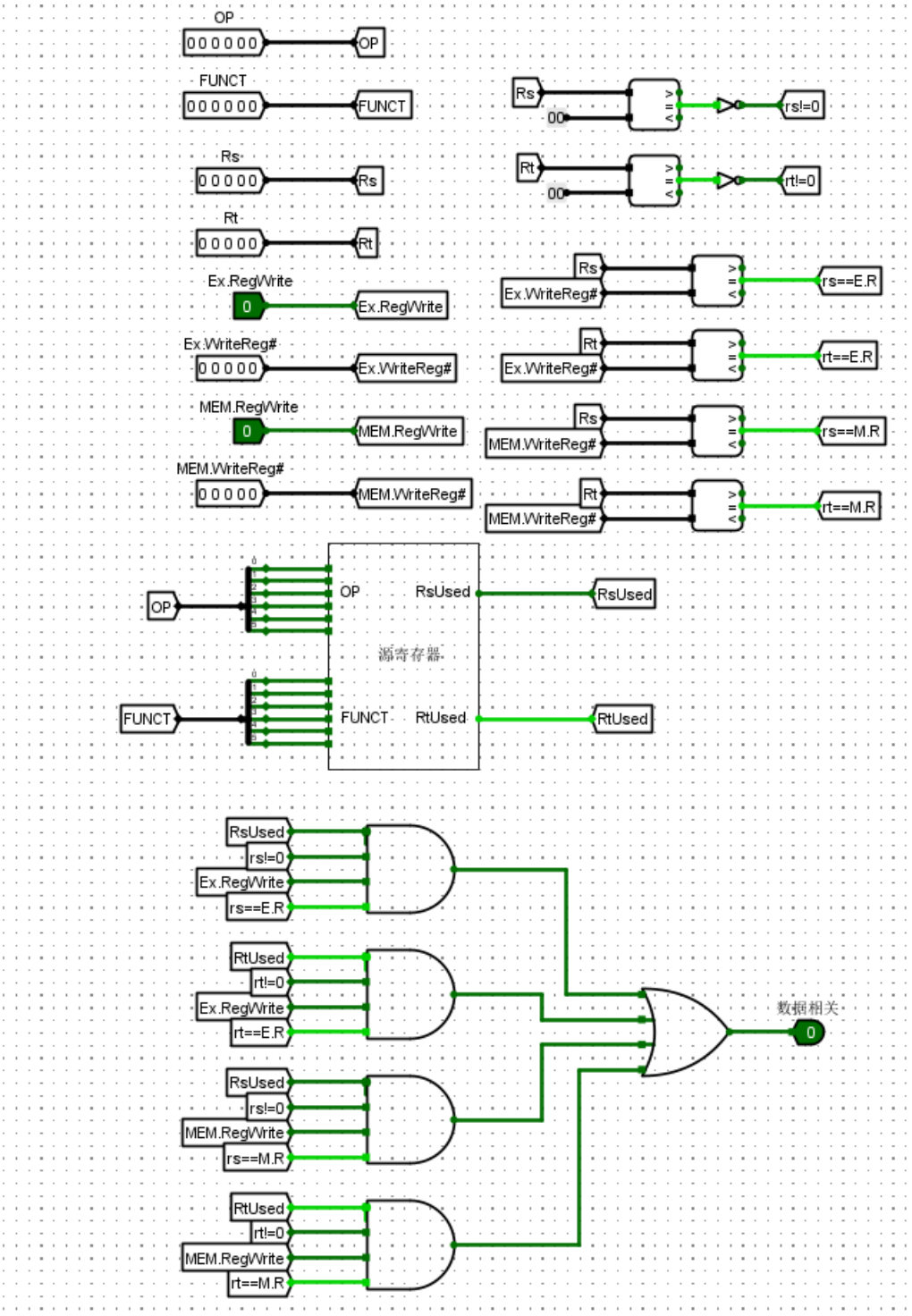

图 3.10 数据相关检测电路

检测到数据相关时，将 ID/EX 流水接口部件清空并将 IF/ID 和 PC 锁存，数据相关取非后与 Pause 取非后向与得到~stall 信号，~stall 与 PC 和 IF/ID 的使能端相连，使系统暂停时或者数据相关时都能将 IF/ID 和 PC 锁存，另外，系统暂停时其他的流水接口部件也要锁存，将 Pause 取非后得到 en，en 与其他流水接口部件使能端相连，完成数据冲突处理，如图 3.11 所示。

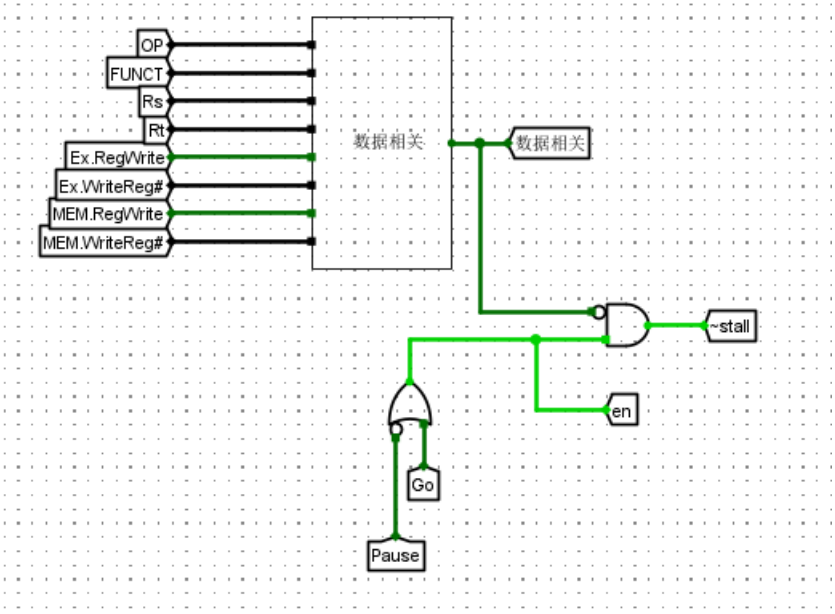

图 3.11 数据冲突处理电路

3.4 重定向流水线实现

在数据相关检测电路的基础上进行改造，完成重定向检测电路的设计。首先需要检测出 Load-Use 相关，Load-Use 逻辑表达式如下：

LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#) + RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)

注意单周期 CPU 实现中为了简化电路，只实现了 MemWrite 写信号，没有实现 MemRead 信号，但由于该信号和 MemToReg 信号是同步的，所以可以用 MemToReg 信号代替 MemRead 信号

然后生成两个重定向选择信号 RsFoward 和 RtFoward，分别用来选择进入 EX 段的 R1 和 R2 使用的是不进行重定向的 ID/EX 流水接口部件中的值、EX/MEM 流水接口部件的值或是 MEM/WB 流水接口部件中的值，RsFoward 赋值逻辑如下：

IF (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))

RsFoward = 2 # ID 段与 EX 段数据相关

else IF (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))

RsFoward = 1 # ID 段与 MEM 段数据相关

else RsFoward = 0 # 无数据相关

RtFoward 赋值逻辑如下：

IF (RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#))

RtFoward = 2 # ID 段与 EX 段数据相关

else IF (RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#))

RtFoward = 1 # ID 段与 MEM 段数据相关

else RtFoward = 0 # 无数据相关

在数据相关检测电路的基础上进行改造，完成重定向检测电路的搭建，如图 3.12 所示。

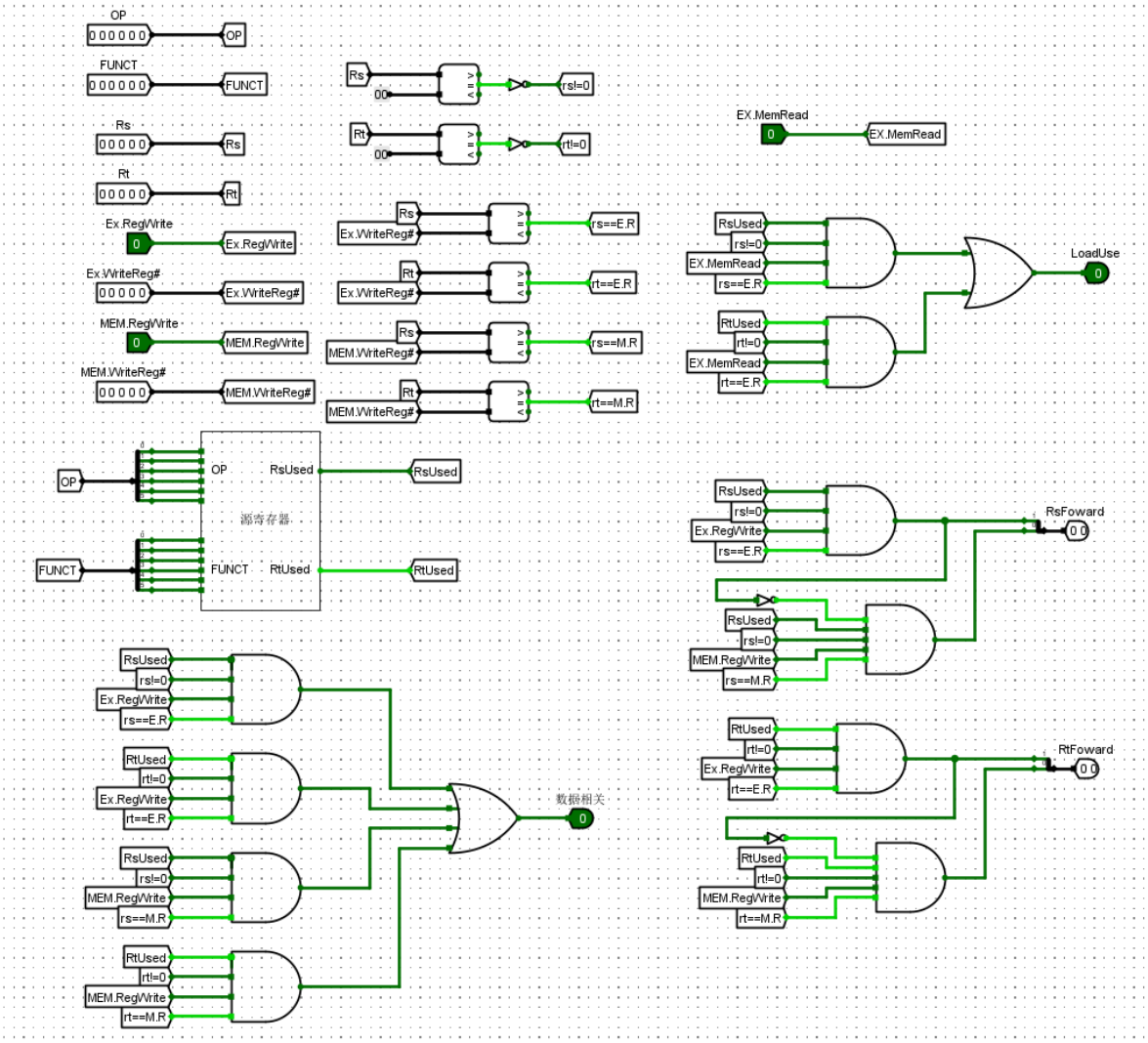

图 3.12 重定向检测电路

RtFoward 和 RsFoward 在 ID 段产生，因为在 EX 段进行重定向数据的选择，所以 RtFoward 和 RsFoward 同样需要使用 ID/EX 段流水接口部件进入下一个阶段使用。改造气泡流水线，进入 EX 段的 R1 和 R2 利用多路选择器替换为 ID/EX 流水接口部件中的值、EX/MEM 流水接口部件的值或是 MEM/WB 流水接口部件中的值，选择端

口的控制信号分别为 RtFoward 和 RSFoward，搭建重定向电路如图 3.13 所示。

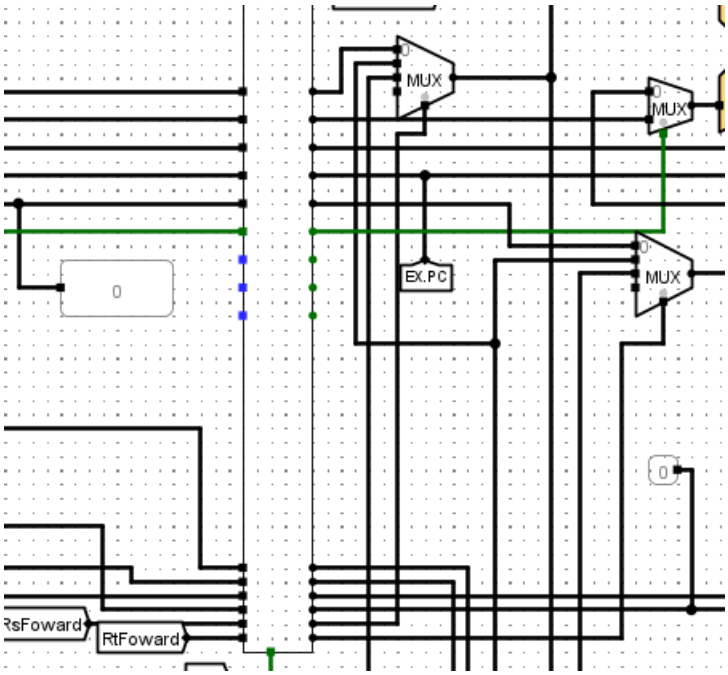

图 3.13 重定向电路设计

重定向处理电路在气泡流水线数据冲突处理电路的基础上进行改造，将数据冲突检测部件替换为重定向检测部件，Load-Use 相关采用和在气泡流水线数据冲突相同的处理方法，RtFoward 和 RSFoward 用于重定向数据的选择，重定向处理电路如图 3.14 所示。

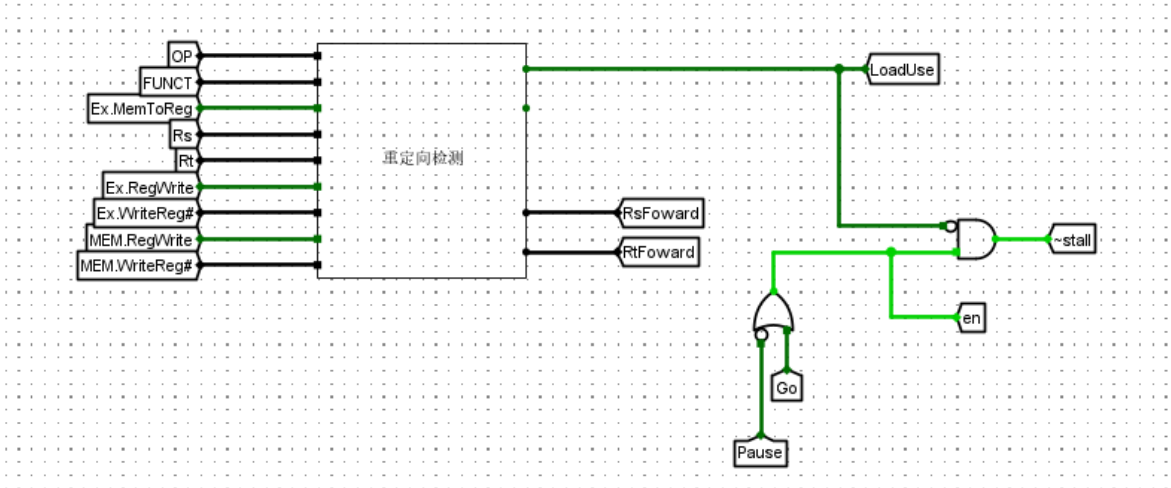

图 3.14 重定向处理电路

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 benchmark 标准程序进行测试，观察运行过程和结果。

4.1.1 单周期 CPU

运行完 benchmark 程序后运行结果如图 4.1 所示，符合预期。

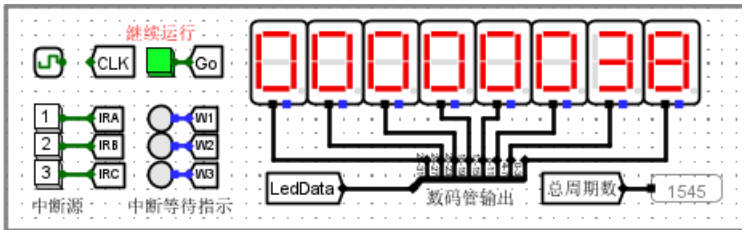

图 4.1 单周期 CPU 运行结果

4.1.2 理想流水线

用理想流水线测试程序进行测试，理想流水线测试程序中不含分支跳转指令，而且避免了数据冲突，运行结果如所示，总周期数为 21，数据存储器内容如所示，前四个字的数据为 0，1，2，3。

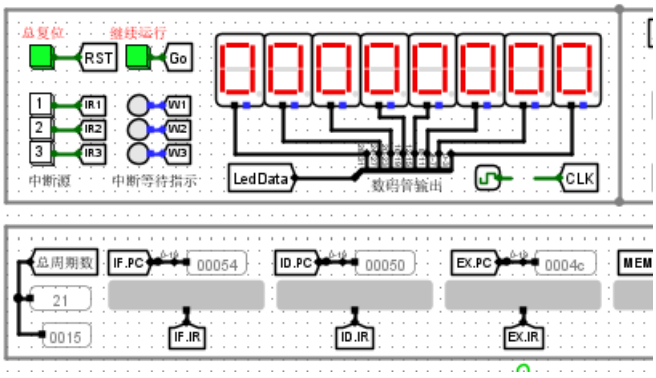

图 4.2 理想流水线运行结果

```
000 00000000000000010000000200000003 000
010 00000000000000000000000000000000 000
```

图 4.3 数据存储器中内容

4.1.3 气泡流水线

用 benchmark 测试程序进行测试，运行完毕后运行结果如图 4.4 所示。

气泡流水线运行标准测试程序后，总周期数 3623，与单周期 cpu 的 1545 相比增加了一倍多。虽然每个周期的时间比单周期要短，但是有大量时钟周期 cpu 在空转，插入了多达 1447 个气泡，性能不好。

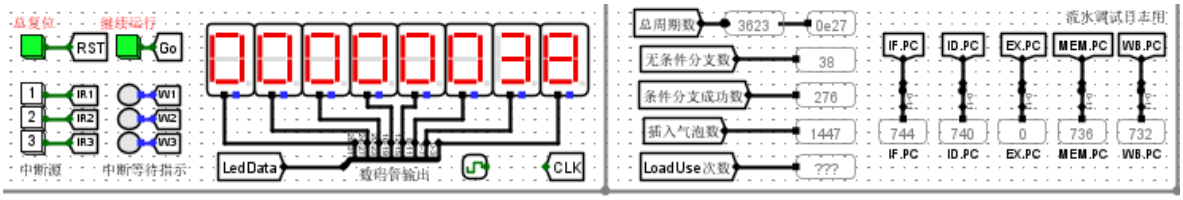

图 4.4 气泡流水线运行结果

4.1.4 重定向流水线

用 benchmark 测试程序进行测试，运行完毕后运行结果如图 4.5 所示。

重定向流水线运行标准测试程序后，总周期数 2302，插入气泡数位 748 与气泡流水线相比插入气泡数大大减少，性能明显提升。

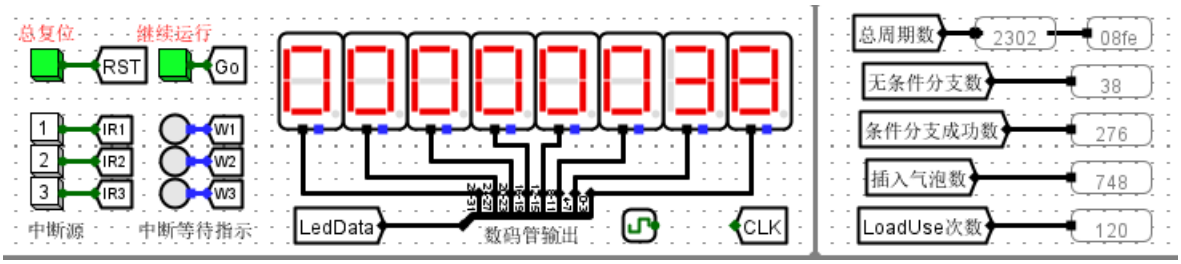

图 4.5 重定向流水线运行结果

在支持差异化指令的重定向流水线上运行增加了差异化指令的 benchmark_ccmb 测试程序，运行完基础指令后，运行结果如图 4.6，符合预期。

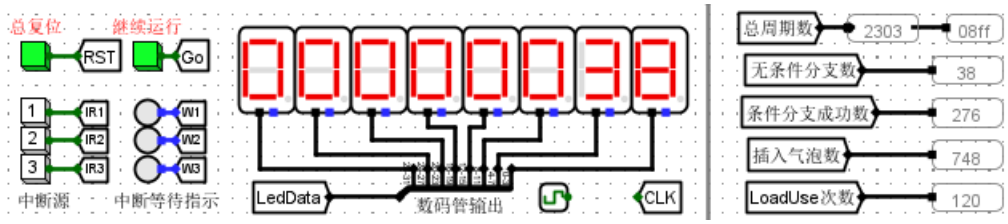

图 4.6 基础指令运行结果

点击 Go，继续运行差异化指令，输出符合预期，截取中间的部分输出，如图 4.7

至图 4.10 所示，结果符合预期。

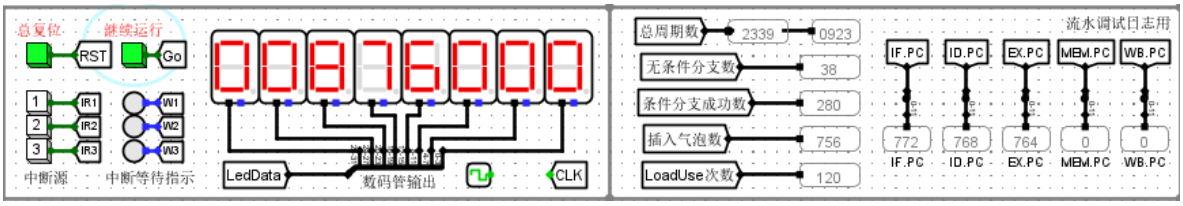

图 4.7 SLLV 指令测试

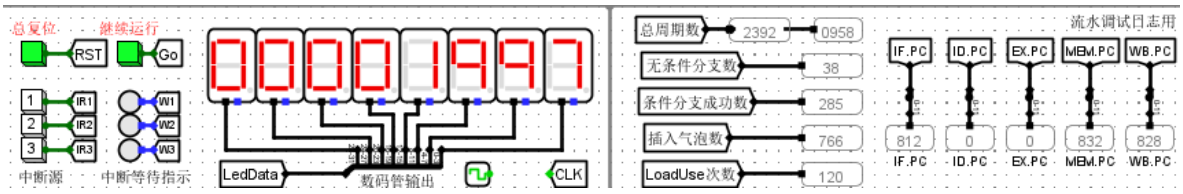

图 4.8 SLTIU 指令测试

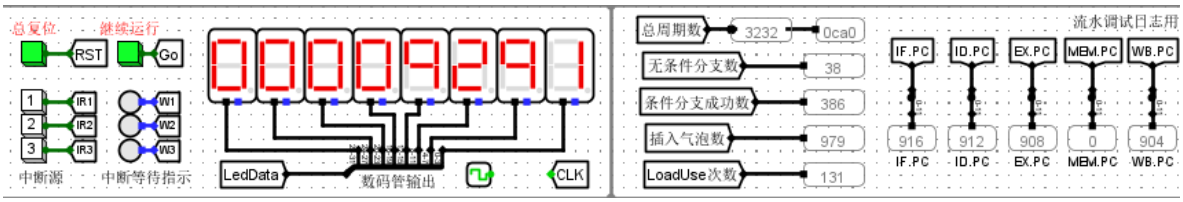

图 4.9 LHU 指令测试

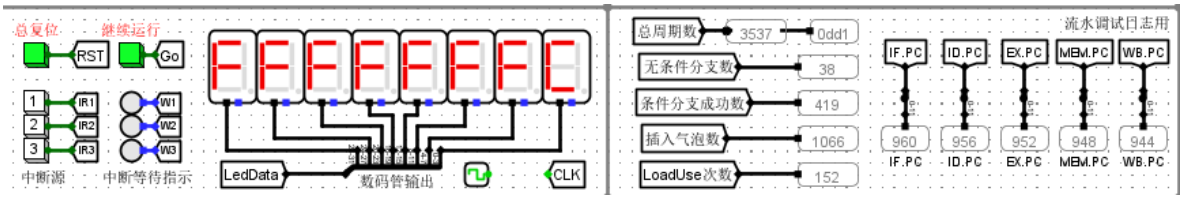

图 4.10 BLTZ 指令测试

4.2 主要故障与调试

4.2.1 流水接口部件故障

气泡流水线： 插入气泡错误问题。

故障现象：测试气泡流水线出现错误，重新单步执行，查看时空检测图，发现流水线比预期提前一个周期插入气泡，如图 4.11 所示。

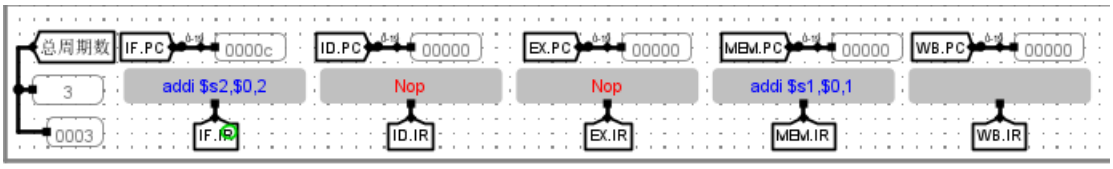

图 4.11 故障图

原因分析：分支指令还在 EX 段时就提前插入气泡，查看流水接口部件电路，以 IF/ID 为例如所图 4.12 示，直接使用寄存器的清空端插入气泡，这会导致清空不受时钟的影响，在分支指令进入 EX 段时就会产生气泡。

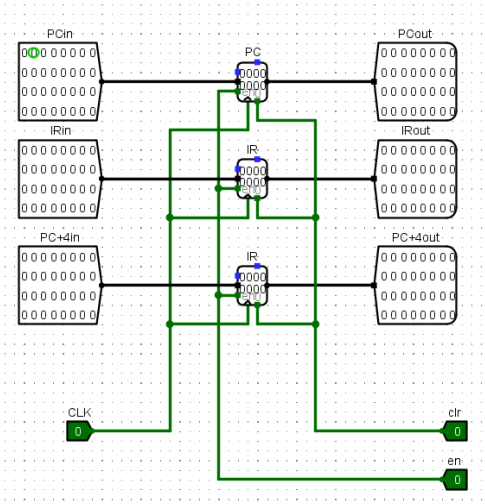

图 4.12 原 IF/ID 流水接口部件

解决方案：预期行为是分支指令进入 EX 段的下一个时钟周期插入信号，所以对流水接口部件进行改造，如图 4.13 所示，利用多路选择器，当收到插入气泡的信号时，用常量 0 作为输入，在下一个时钟周期插入气泡。

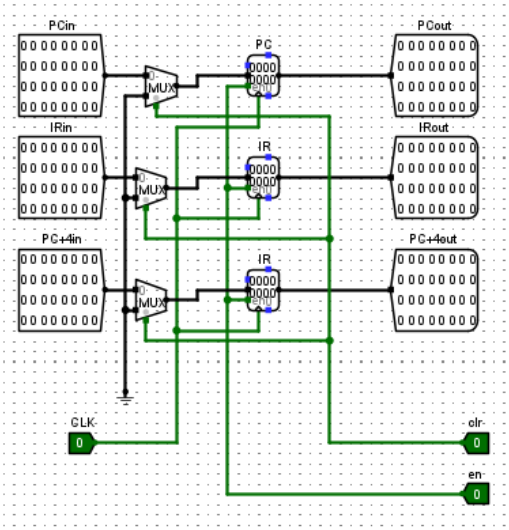

图 4.13 改进后的 IF/ID 流水接口部件

4.3 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习上学期所学的组成原理知识，了解控制信号和数据通路的基本概念，查阅 MIPS 指令手册查看各指令的功能和数据流向
第二天	根据指令手册填写硬布线控制器自动生成表，搭建硬布线控制器，基本完成数据通路的搭建
第三天	educoder 平台提交测试，根据结果调试 BUG，并最终通过测试
第四天	搭建流水接口部件，将单周期 CPU 按阶段划分并使用流水接口部件完成理想流水线设计
第五天	调试 BUG，修改流水接口部件，并通过测试
第六天	改进流水接口部件以适应锁存和清空功能，填写源寄存器使用情况自动生成表
第七天	利用源寄存器查询电路，根据数据相关逻辑完成数据相关检测电路搭建，对理想流水线改造，加入数据冲突和结构冲突处理
第八天	调试气泡流水线并通过测试，完成重定向检测逻辑电路设计
第九天	利用重定向检测电路并在气泡流水线基础上加入旁路，搭建起重定向流水线
第十天	调试重定向流水线并通过测试

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 logisim 单周期 CPU 的设计，对单周期 CPU 进行功能分段并设计流水接口部件，得到了理想流水线；在理想流水线的基础上增加了数据冲突和控制冲突的处理并改造流水接口部件，实现了气泡流水线；在气泡流水线的基础上使用重定向技术，实现了重定向流水线；。

2） 实现了对 CCMB 差异化指令的支持。

3） 对 CPU 的性能优化方法和有了一定的了解，对流水线技术有了更深入的认识，纠正了许多自己 CPU 知识上的错误，对上学期所学的计算机组成原理有了更深入的认识。

5.2 课设心得

我从本次课程设计中学到了很多，虽然最后没能实现更多可选模块的功能，团队也因为个人实验进度不一致也没能完成团队演示系统设计，但我仍记着第一天的老师在实验指导时说的“奋斗一两周，造块 CPU”，很充实的做了两周的实验，看着最后搭建起的重定向流水线 CPU 通过测试的画面，很有成就感，在这个过程中我也有很多收获。

刚看到实验任务书时，我的组原知识已经淡忘了许多，根据无从下手。我先从上学期的课程 PPT 入手，渐渐熟悉起指令的执行周期，数据通路这些概念，然后阅读 MIPS 指令手册，找到需要实现的指令的功能和数据流向，开始填写硬布线控制器自动生成表，稳扎稳打的开始 CPU 设计。在 CPU 设计过程中，比较难的地方在于对流水线 CPU 的冒险的处理，虽然老师提供了参考资料，但理解起来还是要花些时间，实现起来就更加复杂了，其中我感觉比较巧妙的一点在于可以利用硬布线控制器自动生成表来实现源寄存器查询的组合逻辑电路。

关于建议，在这次课程设计中，我感觉比较遗憾的一点在于没有实现团队演示系统的设计，所以我觉着可以先不进行分组，完成个人实验任务后的同学再互相组队，

这样团队就可以集中精力完成团队任务了。

我认为计算机组成原理时我大学里上过的课中非常优秀的一门课，几位老师的教学水平都很高，而且不论是课件和实验都是经过精心设计的。最后，非常感谢队友的帮助和老师的指导，我从中学到了很多。

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] 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.

[5] 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.

[6] 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.

[7] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字：李世铭