华中科技大学

2020

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：	CS1704
学　　号：	U201714636
姓　　名：	王天阳
电　　话：	18327631232
邮　　件：	976271408@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 $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$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	扩展指令：SUBU	无符号减	
29	扩展指令：SLTIU	小于立即数置 1（无符号）	
30	扩展指令：SB	存储字节	
31	扩展指令：BLTZ	小于 0 转移	

2 总体方案设计

2.1 单周期 CPU 设计

将单周期 CPU 分成几个模块：指令存储器(Instruction Memory)、硬布线控制器（Control Unit）、寄存器文件（Register File）、运算器（ALU）、数据存储器（Data Memory）、地址转移逻辑等。在一个时钟周期内，完成取指令 IF、指令译码 ID、指令执行 EXE、存储器访问 MEM、结果写回 WB 五个阶段的处理。由于取指令和执行指令阶段都涉及访存，所以只能将指令存储器和数据存储器分开（哈佛结构），避免引起冲突。在 24 单周期 CPU 的基础上添加 4 条扩展指令分别是：SUBU、SLTIU、SB、BLTZ。

总体结构图如图 2. 1 所示。

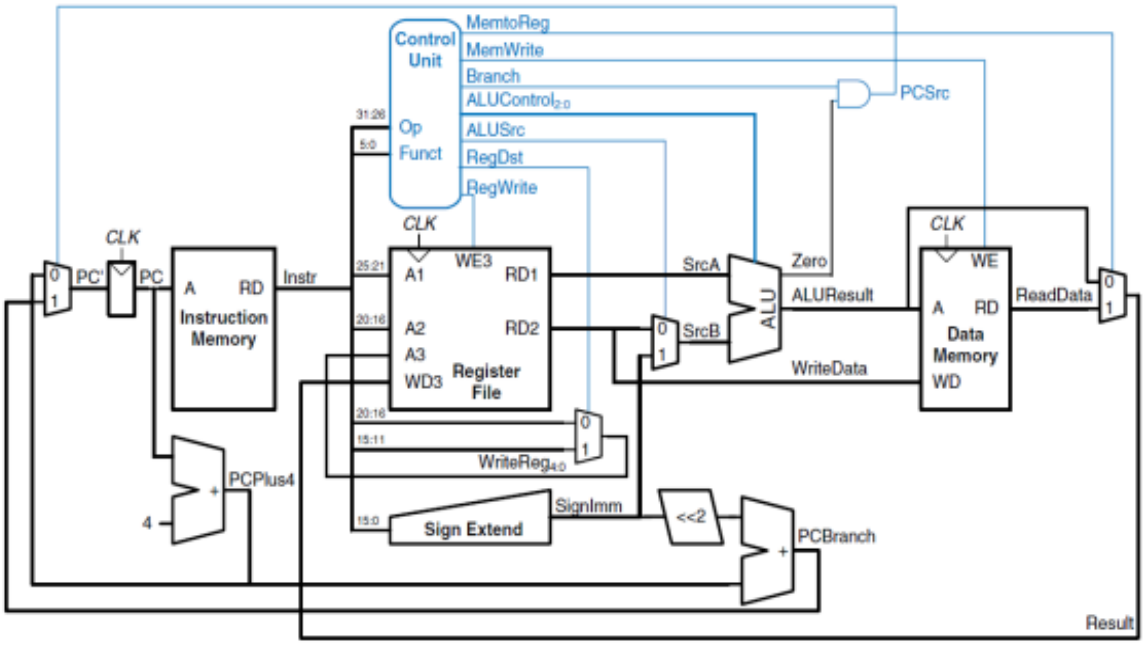


图 2. 1 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

PC 记录指令地址，送入指令存储器（IM），输入输出引脚描述如表 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
RST	输入	1	清零
EN	输入	1	使能端，为 0 时忽略时钟输入
PC_IN	输出	32	地址转移逻辑输入到 PC 寄存器中的值
PC_OUT	输出	32	PC 输出端的值

2. 指令存储器 IM

指令用 ROM 存储，数据位宽设置为 32 位，指令存储器以字寻址，取 PC 的第 2 到 11 位输入。输入输出引脚描述如表 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	程序地址
IR	输出	32	指令

3. 运算器

根据操作码对两个 32 位的操作数进行对应运算并输出 32 位结果和一位表示两操作数是否相等。输入输出引脚描述如表 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
A	输入	32	操作数 A
B	输入	32	操作数 B
ALU_OP	输入	4	运算器功能码，决定运算方式
Shamt	输入	5	移位操作数
Result	输出	32	ALU 运算结果
Equal	输出	1	两操作数是否相等

4. 寄存器堆 RF

寄存器堆包括 MIPS 里 32 个 32 位寄存器，读写数据均为 32 位，每次可读两个寄存器 R1,R2, 可写一个寄存器 W，写使能信号控制写数据，输入输出引脚描述如表 2. 4 所示。

表 2. 4 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
R1#	输入	5	读寄存器 1
R2#	输入	5	读寄存器 2
W#	输入	5	写寄存器
WE	输入	1	写使能
Din	输入	32	写入数据
A	输出	32	读出数据 1
B	输出	32	读出数据 2

5. 数据存储器

用 RAM 实现，通过两位的访问模式 sel 决定采用字节、半字、字访问，写使能信号控制读写，取 2 到 11 位作为读写地址，输入输出引脚描述如表 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
RST	输入	1	清零
sel	输入	2	访问模式
Str	输入	1	写使能信号
Din	输入	32	写入的数据
Addr	输入	10	读写的字节地址
Dout	输出	32	读出的数据

2.1.2 数据通路的设计

表 2. 624+4 条指令系统数据通路框架

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Add	Din
SLL	PC+1	P		rt	rd	ALU	RF.D2	shamt	0		
SRA	PC+1	P			rd	ALU	RF.D2	shamt	1		
SRL	PC+1	P			rd	ALU	RF.D2	shamt	2		
ADD	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	5		
ADDU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	5		
SUB	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	6		
AND	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	7		
OR	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	8		
NOR	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
SLT	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
SLTU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
JR	RF.D1	P	rs						X		
SYSCA	PC+1	P	v0(2)	a0(4)	a0(4)		RF.D1	10	X		
J	(PC&0XF000)	P							X		
JAL	(PC&0XF000)	P			31	PC+1			X		
BEQ	PC+1+offset	P	rs	rt			RF.D1	RF.D2	X		
BNE	PC+1+offset	P	rs	rt			RF.D1	RF.D2	X		
ADDI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	5		
ANDI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	7		
ADDIU	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	5		
SLTI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	1		
ORI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	8		
LW	PC+1	P	rs		rt	DM.Dout	RF.D1	IMM[15:0]	5	ALU	
SW	PC+1	P	rs	rt			RF.D1	IMM[15:0]	5	ALU	RF.D
SUBU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	6		
SLTIU	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	1		
SB	PC+1+offset	P	rs	rt			RF.D1	IMM[15:0]	5	ALU	RF.D
BLTZ	PC+1+offset	P	rs	rt(0)			RF.D1	RF.D2	5		

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. 7。

表 2. 7 主控制器控制信号的作用说明

控制信号	取值	说明
Syscall	1	函数调用，写寄存器的编号选择 31 号
	0	写寄存器编号选择指令解析器解析出的 rd、rt
RegDst	1	写寄存器编号为 rt
	0	写寄存器编号位 rd
Jal	1	函数调用，写寄存器的编号选择 31 号
	0	写寄存器编号选择指令解析器解析出的 rd、rt
RegWrite	1	写寄存器使能信号，允许将寄存器的输入加载到输出
	0	不允许写寄存器，寄存器输出值不变
Aluop	0000-1100	运算器规格
MemWrite	1	写内存控制信号
	0	不允许写内存
MemToReg	1	寄存器写入数据来自存储器
	0	不允许从存储器写入数据
Bne	1	Bne 指令译码信号
	0	非 Bne 指令
Beq	1	Beq 指令译码信号
	0	非 Beq 指令
JMP	1	无条件跳转，npc 取 26 位立即数
	0	不跳转，npc 取其他值
JR	1	寄存器跳转指令译码信号
	0	不允许寄存器跳转

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。该控制信号表框架如表 2.8 所示，为空表示取值位 0 或者取任意值。

表 2. 8 主控制器控制信号框架

指令	ALU_OP	Mem to Reg	Mem Write	ALU_SRC	Reg Write	SYS CALL	Signd Ext	Reg Dst	BEQ	BNE	JR	JMP	JAL	SB	BLTZ
SLL				1			1								
SRA				1			1								
SRL				1			1								
ADD				1			1								
ADDU				1			1								
SUB				1			1								
AND				1			1								
OR				1			1								
NOR				1			1								
SLT				1			1								
SLTU				1			1								
JR										1	1				
SYSCAL					1										
J											1				
JAL				1							1	1			
BEQ						1		1							
BNE						1			1						
ADDI			1	1		1									
ANDI			1	1											
ADDIU			1	1		1									
SLTI			1	1		1									
ORI			1	1											
LW	1		1	1											1
SW		1	1												
SUBU				1			1								
SLTIU			1	1		1									
SB			1	1		1							1		
BLTZ						1								1	

2.2 中断机制设计

2.2.1 总体设计

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。为了在中断执行后能够正确返回主程序，必须将跳转前的状态保存。各级中断在执行过程中需改变中断优先级，故需要若干中断屏蔽寄存器。优先级高的中断可以打断优先级低的中断，当有优先级更高的终端的时候转去执行优先级更高的中断。每次优先级更高的中断打断优先级低的中断时需将当前的 PC 存放到一个特殊的寄存器的值压栈处理。某些操作不允许外部事件中断，需要添加中断使能寄存器 IE。多级中断对 EPC、IE 和中断屏蔽寄存器的操作需要增加 ERET、MFC0 和 MTC0 指令并更新控制器和数据通路。

2.2.2 硬件设计

（1） 中断屏蔽寄存器

3 个中断需要 3 个上升沿触发的 D 触发器实现 3 个中断屏蔽寄存器，中断程序执行前写 1，寄存器输出取反和中断请求信号相与。

（2） 中断使能寄存器

允许中断时输出 1，屏蔽所有中断时输出 0，通过 D 触发器来实现。

（3） EPC 寄存器

使用上升沿触发的 32 位寄存器实现，中断时，使能端置 1，中断中，使能端为 0。

（4） 控制器更新

单级中断新增 ERET 指令。多级中断新增 ERET、MFC0 和 MTC0 指令。

2.3 流水 CPU 设计

2.3.1 总体设计

由于单周期 CPU 设计实现简单，所有指令执行时间均是一个相同的周期，速度取决于最慢的指令，效率低，性能差，流水 CPU 取而代之。

流水 CPU 建立在单周期 CPU 基础上，将指令执行过程细分为取指令 IF、指令译码 ID、指令执行 EX、访存 MEM、写回 WB 五个阶段，图 2. 2 所示。

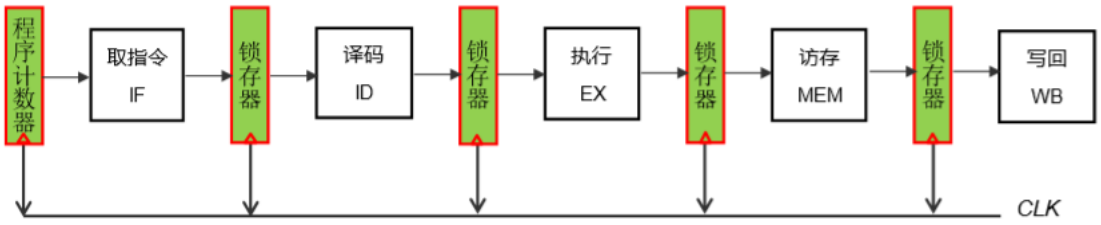


图 2. 2 指令流水线逻辑架构

2.3.2 流水接口部件设计

使每个阶段后都需要增加一个锁存器，用于锁存本段处理完成的数据或结果，以保证该阶段的执行结果给下一个阶段使用。程序计数器、所有锁存器采用共用时钟同步，用寄存器组存放需要的数据和控制信号，将清零端设为常量 0。

2.3.3 理想流水线设计

指令执行的五个阶段分别对应单周期 CPU 中的不同部件，取指令（IF）阶段对应程序计数器 PC、NPC 下址逻辑、指令存储器 IM、，译码（ID）阶段对应控制器逻辑、取操作数逻辑、寄存器堆等，执行（EX）阶段对应运算器 ALU，访存（MEM）对应数据存储器 DM，写回（WB）阶段对应寄存器堆即寄存器写入控制模块。不考虑冲突冒险。

2.4 气泡式流水线设计

理想流水线无法解决数据读写冲突，通过增加数据相关检测逻辑，改造流水接口部件，增加插入气泡逻辑解决数据冲突（写后读 RAW、读后写 WAR、写后写 WAW）；流水线遇到分支指令或其它改变 PC 值得指令时，分支指令后续载入流水线相邻指令可能不能进入执行阶段，产生控制冲突，增加分支冲突处理逻辑解决控制冲突。

2.5 重定向流水线设计

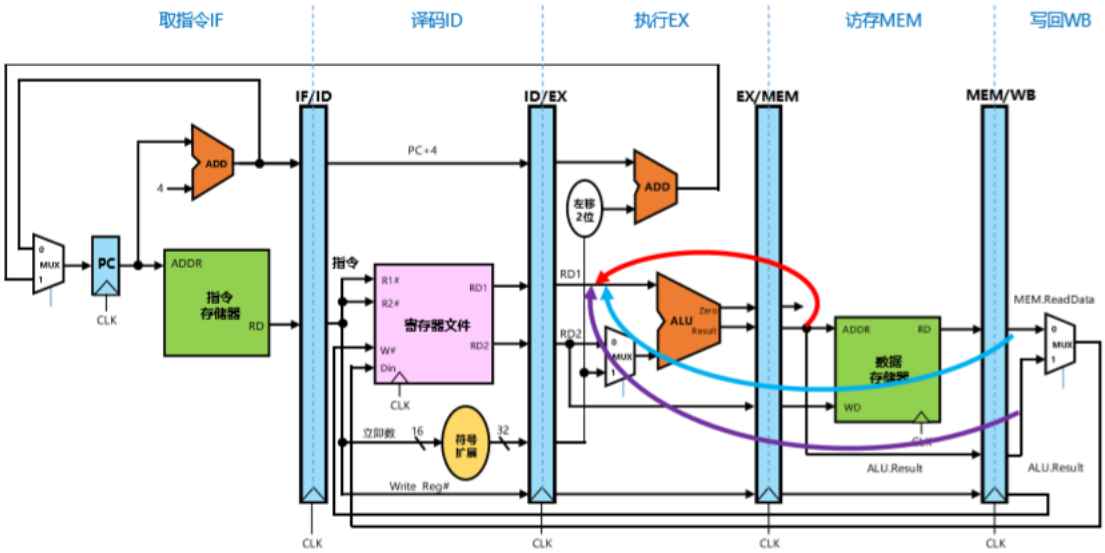


图 2. 3 数据重定向示意图

大量气泡的插入会严重影响指令流水线的性能。可以先不考虑 ID 段所取得操作数是否正确，而是等到实际需要使用这些操作数时再考虑正确性问题，图 2. 3 所示，EX 段得指令可能与 MEM 段、WB 段两条指令存在数据相关，此时 EX 段取得的操作数应该是错误的数据，正确的数据分别存放在 EX/MEM 以及 MEM/WB 流水接口部件中，还未写到寄存器中，此时可以直接将正确数据从其所在位置重定向到 EX 段合适的位置即可。改造数据相关检测逻辑，增加 Load-use 数据相关检测逻辑，构造 EX 段重定向逻辑。

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，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。EN 为寄存器使能信号，如图 3.1 所示。

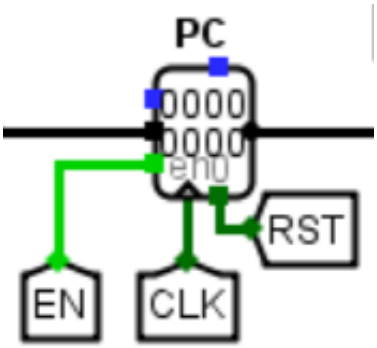


图 3. 1 程序计数器（PC）

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。

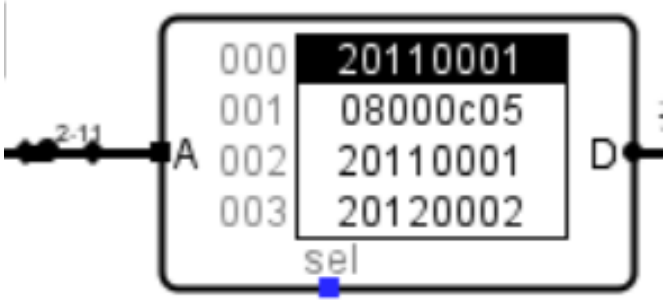


图 3. 2 指令存储器（IM）

3) 寄存器文件（RegFile）

Logism 实现：

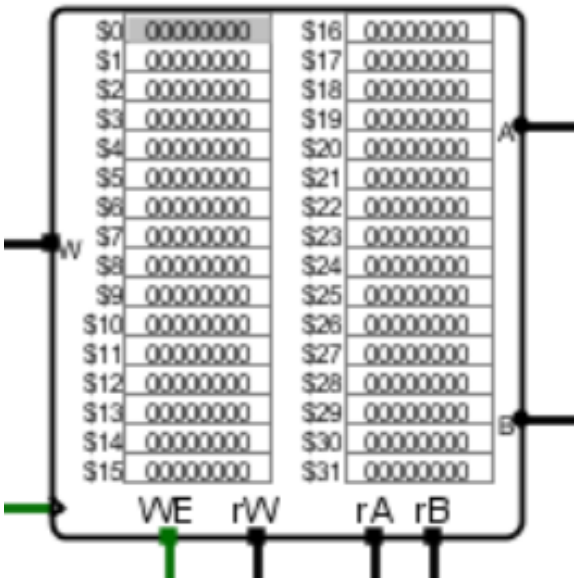


图 3. 3 寄存器文件（RegFile）

4) 数据存储器（DM）

Logism 实现：使用 4 个数据位宽为 8 的随机存储器 RAM 实现，从 32 位地址中截取 2-11 位作为 DM 的地址，如图 3. 4 所示。

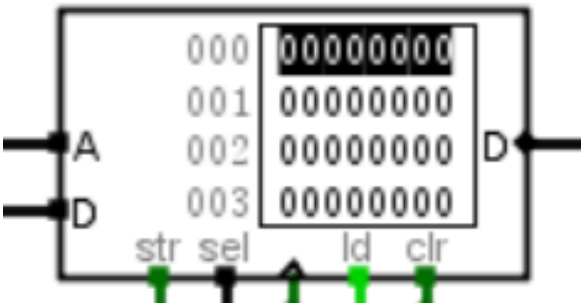


图 3. 4 数据存储器（DM）

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，数据通路表如下表 3. 1 所示。

表 3. 1 24+4 条指令系统数据通路

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Add	Din
SLL	PC+1	P		rt	rd	ALU	RF.D2	shamt	0		
SRA	PC+1	P			rd	ALU	RF.D2	shamt	1		
SRL	PC+1	P			rd	ALU	RF.D2	shamt	2		
ADD	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	5		
ADDU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	5		
SUB	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	6		
AND	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	7		
OR	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	8		
NOR	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
SLT	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
SLTU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	1		
JR	RF.D1	P	rs						X		
SYSCA	PC+1	P	v0(2)	a0(4)	a0(4)		RF.D1	10	X		
J	(PC&0XF000)	P							X		
JAL	(PC&0XF000)	P			31	PC+1			X		
BEQ	PC+1+offset	P	rs	rt			RF.D1	RF.D2	X		
BNE	PC+1+offset	P	rs	rt			RF.D1	RF.D2	X		
ADDI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	5		
ANDI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	7		
ADDIU	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	5		
SLTI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	1		
ORI	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	8		
LW	PC+1	P	rs		rt	DM.Dout	RF.D1	IMM[15:0]	5	ALU	
SW	PC+1	P	rs	rt			RF.D1	IMM[15:0]	5	ALU	RF.D
SUBU	PC+1	P	rs	rt	rd	ALU	RF.D1	RF.D2	6		
SLTIU	PC+1	P	rs		rt	ALU	RF.D1	IMM[15:0]	1		
SB	PC+1+offset	P	rs	rt			RF.D1	IMM[15:0]	5	ALU	RF.D
BLTZ	PC+1+offset	P	rs	rt(0)			RF.D1	RF.D2	5		

根据列出的数据通路表，进行多指令数据通路的合并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 3.5 所示。

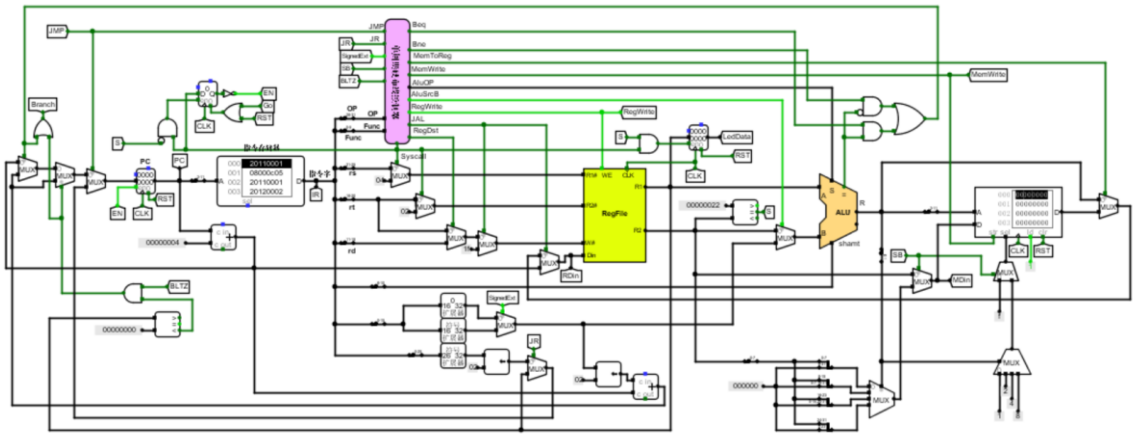


图 3. 5 单周期 CPU 数据通路（Logism）

3.1.3 控制器的实现

主控制器包括 24 条基本指令和 4 条扩展指令，根据指令 op 和 func 字段判断指令类型，对照主控制器控制信号表 3.2 填写控制信号的逻辑表达式和生成表表来生成表达式，将表达式逐一输入 logisim 自动生成主控制器。

表 3. 2 主控制器控制信号

指令	ALU_OP	Mem to Reg	Mem Write	ALU_SRC	Reg Write	SYS CALL	Signd Ext	Reg Dst	BEQ	BNE	JR	JMP	JAL	SB	BLTZ
SLL				1			1								
SRA				1			1								
SRL				1			1								
ADD				1			1								
ADDU				1			1								
SUB				1			1								
AND				1			1								
OR				1			1								
NOR				1			1								
SLT				1			1								
SLTU				1			1								
JR										1	1				
SYSCAL					1										
J											1				
JAL				1							1	1			
BEQ						1		1							

指令	ALU_OP	Mem to Reg	Mem Write	ALU_SRC	Reg Write	SYS CALL	Signd Ext	Reg Dst	BEQ	BNE	JR	JMP	JAL	SB	BLTZ
BNE						1			1						
ADDI			1	1		1									
ANDI			1	1											
ADDIU			1	1		1									
SLTI			1	1		1									
ORI			1	1											
LW	1		1	1											1
SW		1	1												
SUBU				1			1								
SLTIU			1	1		1									
SB			1	1		1							1		
BLTZ						1								1	

3.2 中断机制实现

硬件实现

（1） 中断信号的产生

使用参考电路中的中断按键产生电路，如图 3.6 所示，用两个 D 触发器实现。

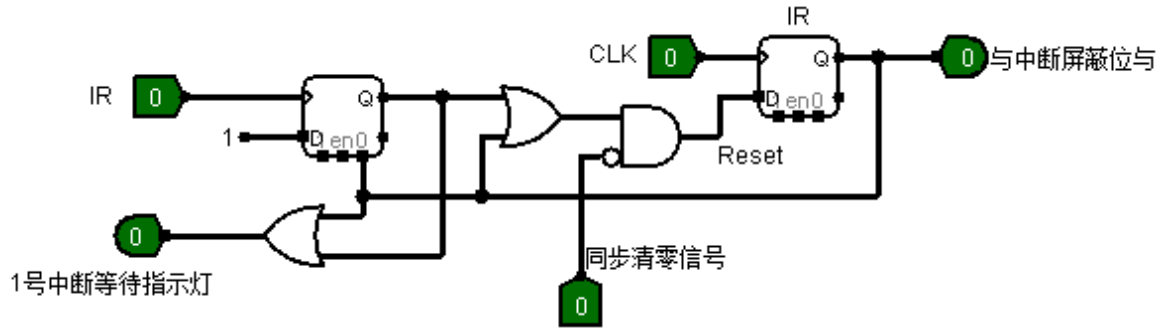


图 3. 6 中断信号产生电路

（2） 中断入口地址选择

中断请求信号接优先编码器，输出编码代表当前优先级最高的中断，用中断编号选出的地址几位中断程序入口地址。

单级中断入口地址选择如图 3.7 所示：

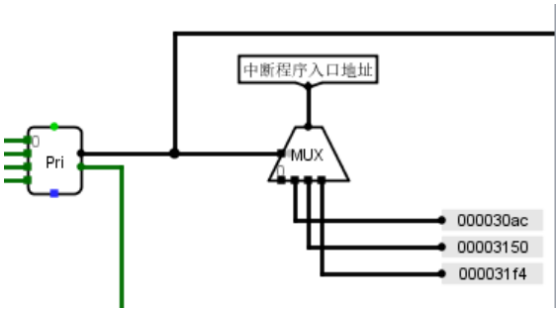


图 3. 7 单级中断入口地址选择

多级中断入口地址选择如图 3.8 所示：

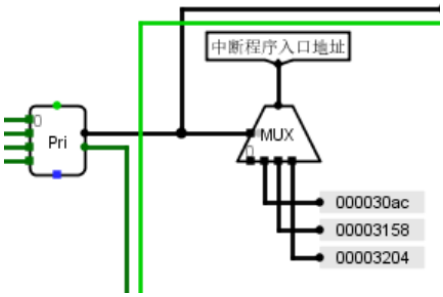


图 3. 8 多级中断入口地址选择

（3） 中断信号清零

单级中断信号清零如图 3.9 所示：

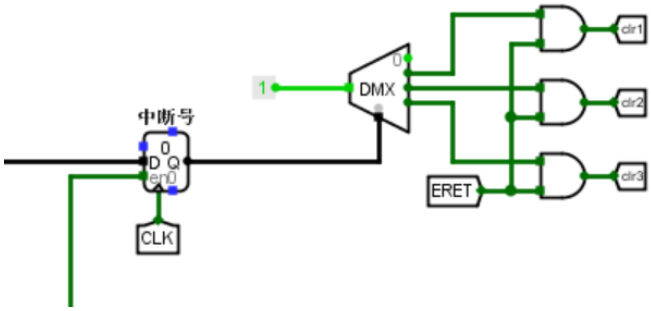


图 3. 9 单级中断信号清零

多级中断信号清零如图 3.10 所示：

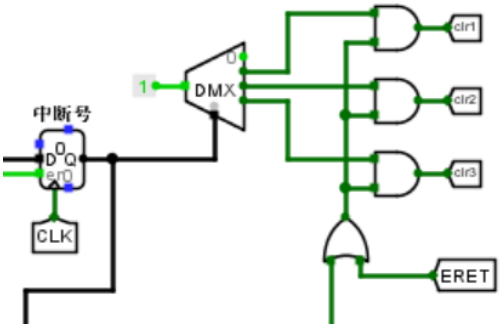


图 3. 10 多级中断信号清零

（4） CP0 和 IE 寄存器

单级中断：

CP0:

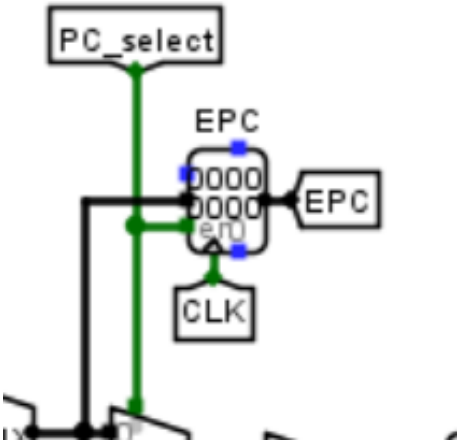


图 3. 11 单级中断 CP0 寄存器

IE:

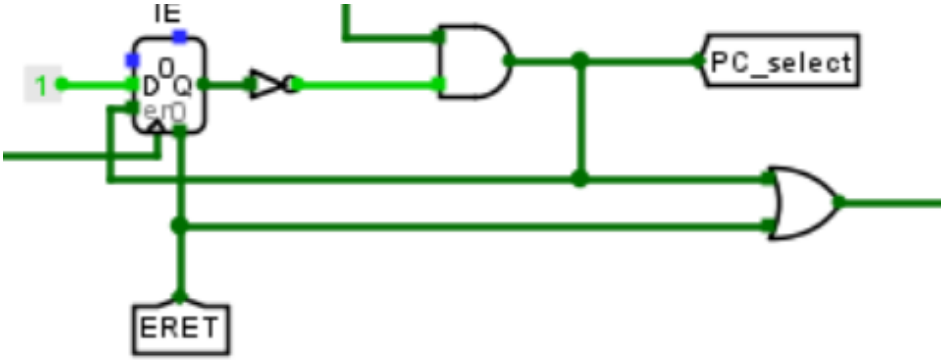


图 3. 12 单级中断 IE 寄存器

多级中断：

CP0：

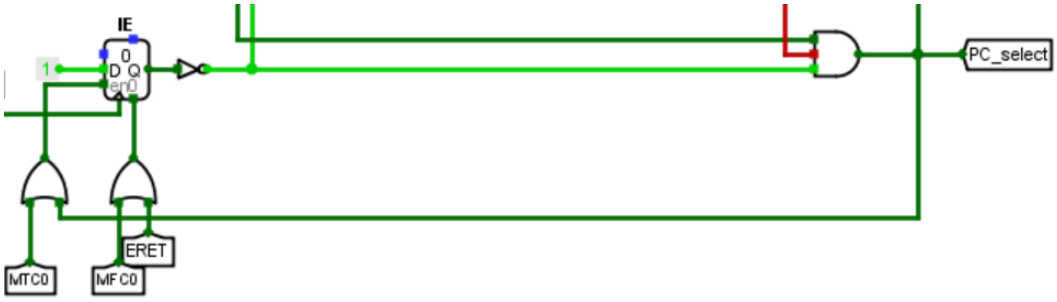


图 3. 13 多级中断 CP0

IE：

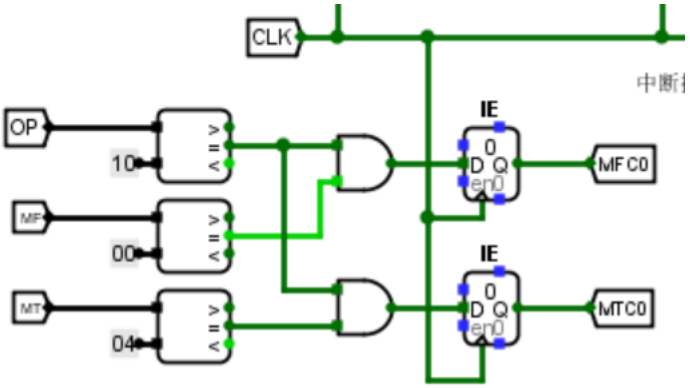


图 3. 14 多级中断 IE

（5） 中断控制信号

由于单周期 CPU 不支持中断，故需要在其基础上增加三条中断指令 ERET、MFC0 和 MTC0,单级中断只需要 ERET 指令，多级中断需要添加 MFC0 和 MTC0 指令。三条指令的 OP 都是 0x10，但是 rs 段不同，新增 ExtendedSignal 电路，是 ERET 指令的控制信号，如图 3. 15 所示。

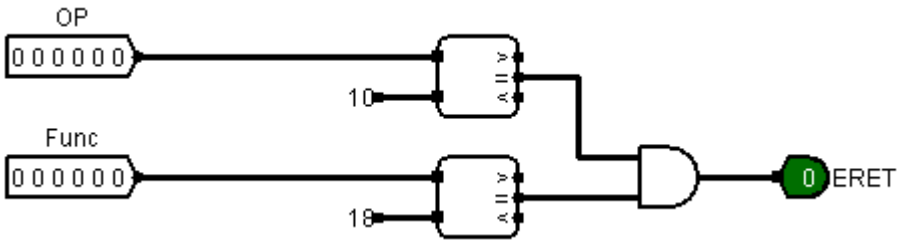
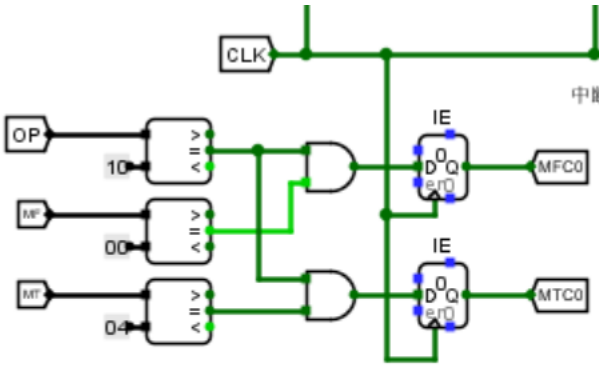


图 3. 15 ERET 指令的控制信号

MFC0 和 MTC0 指令的控制信号如下图 3.15 所示。



3.15 MFC0 和 MTC0 指令的控制信号

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水 CPU 将指令执行过程细分为取指令 IF、指令译码 ID、指令执行 EX、访存 MEM、写回 WB 五个阶段，每个阶段后都需要增加一个锁存器，用于锁存本段处理完成的数据或结果，以保证该阶段的执行结果给下一个阶段使用。程序计数器、所有锁存器采用共用时钟同步，用寄存器组存放需要的数据和控制信号，将清零端设为常量 0，所有输入信号都用对应位宽的寄存器输出，按时钟上升沿触发，使能端低电平输入高电平暂停。

（1） IF-ID 阶段

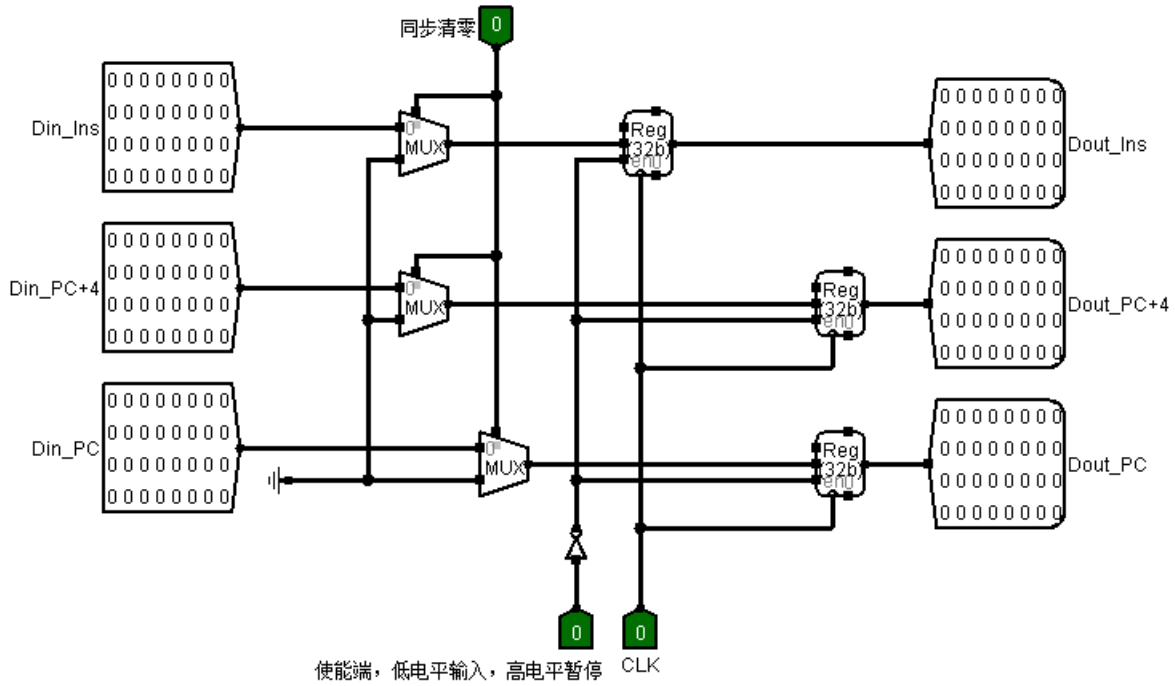


图 3. 16 IF-ID 接口部件

（2） ID-EX 阶段

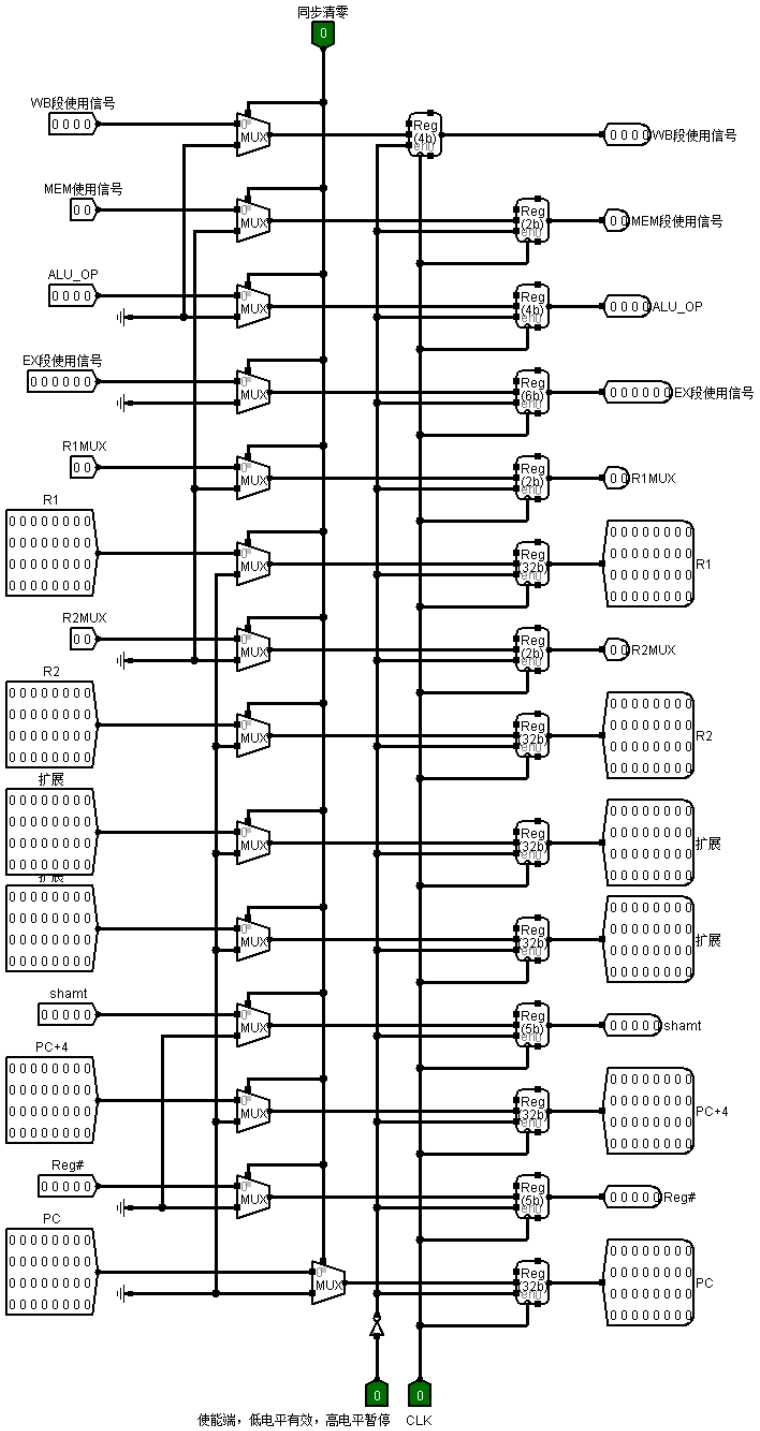


图 3. 17 ID-EX 结构部件

（3） EX-MEM 阶段

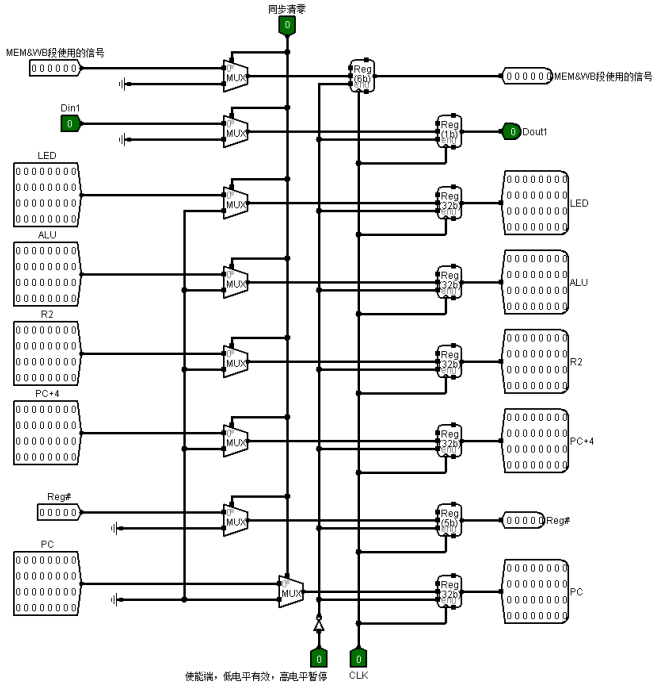


图 3. 18 EX/MEM 接口

（4） MEM-WB 阶段

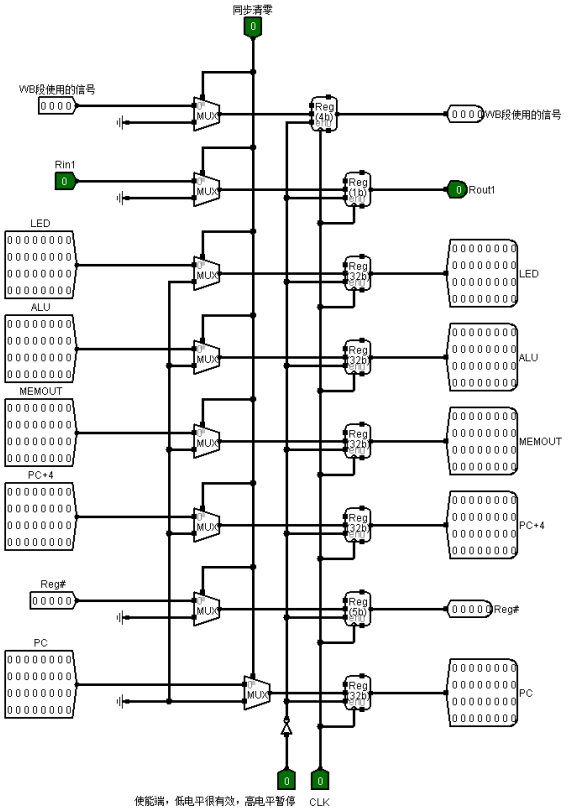


图 3. 19 MEM/WB 接口

3.3.2 理想流水线实现

删去单周期上多余的数据通路，将 IF-ID、ID-EX、EX-MEM、MEM-WB 加入，重新连接数据通路。理想流水线的数据通路如图 3.20 所示。

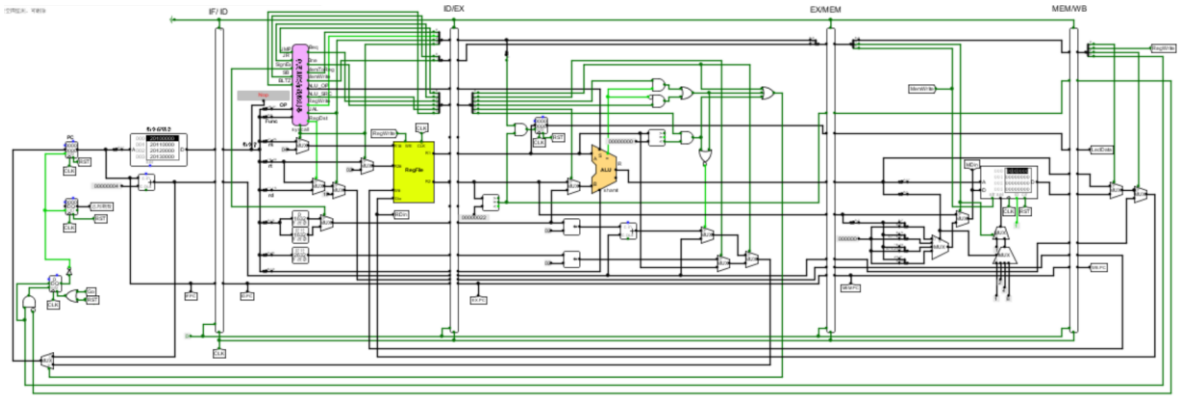


图 3. 20 理想流水线数据通路

3.4 气泡式流水线实现

在理想流水线的基础上增加数据冲突检测，通过硬件插入气泡的方式消除数据冲突；增加控制冲突处理逻辑正确处理分支指令引起的控制冲突。通过填写硬布线控制器表达式自动生成表生成表达式，输入 logisim 生成控制寄存器控制信号。

数据相关检测如图 3.21 所示。

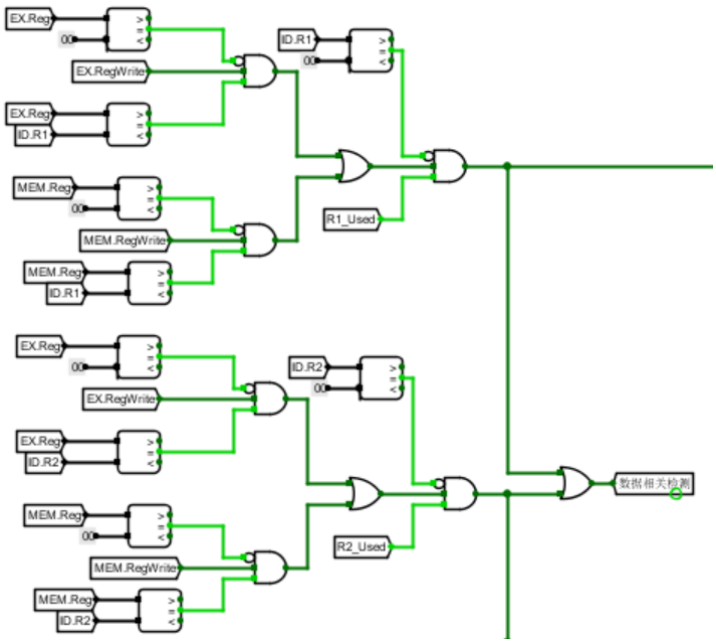


图 3. 21 气泡生成电路

3.5 重定向流水线实现

改造气泡流水线，增加重定向数据通路，增加 Load-Use 数据相关检测逻辑。Load-Use 相关处理电路如图 3.22 所示，重定向控制信号如图 3.23 所示。

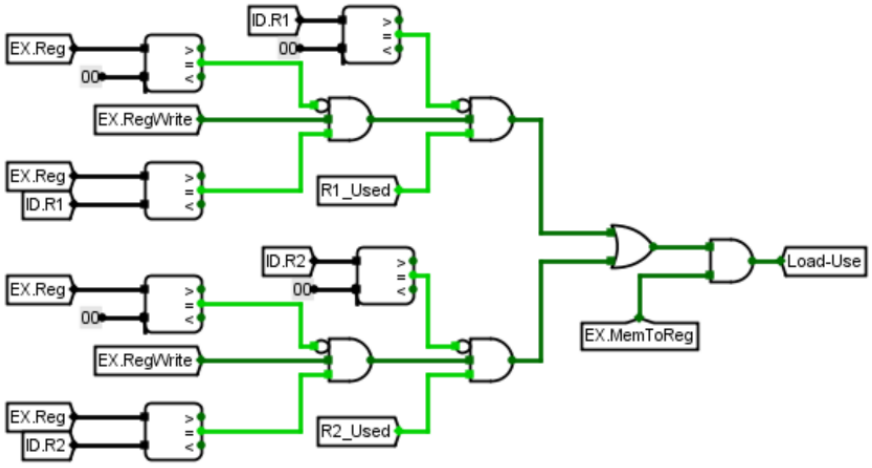


图 3. 22 Load-Use 相关处理电路

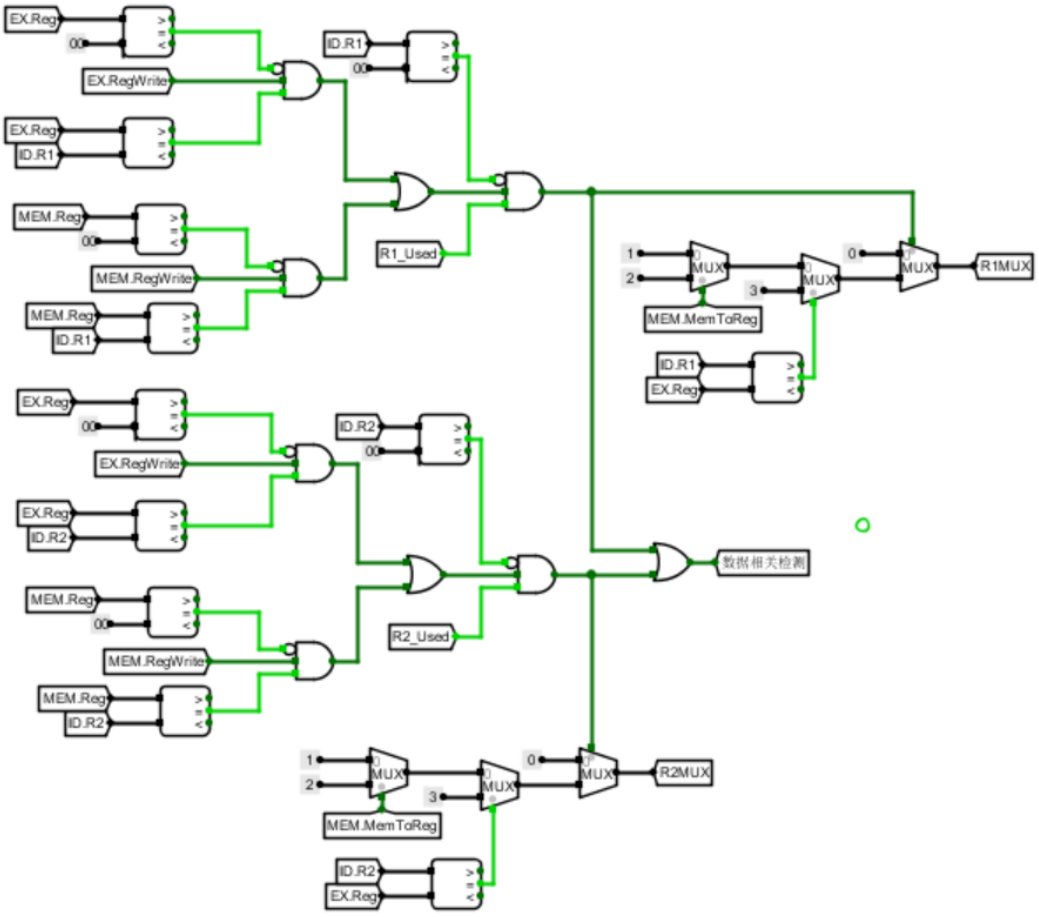


图 3. 23 重定向控制信号生成电路

4 实验过程与调试

4.1 测试用例和功能测试

分别对各个电路进行测试，各个电路使用对应的测试程序，并且测试自己的扩展指令：SUBU、SLTIU、SB、BLTZ。

4.1.1 单周期 CPU

加载 benchmark.hex 标准程序测试功能，各个周期数如图 4.1 所示。

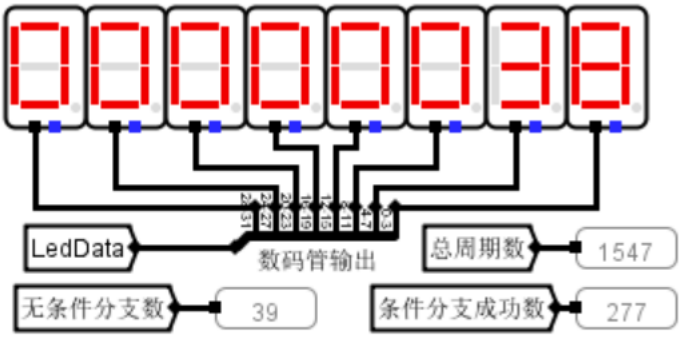


图 4. 1 单周期 logismi 测试周期数

4 条扩展指令通过 MARS4_5.jar 汇编，然后利用菜单中的 Dump 功能将代码段和数据段导出，采用十六进制文本的方式导出到某个文本文件，然后再第一行加入“v2.0 raw”，载入 logisim 进行测试。

（1） SUBU 无符号减

依次输出 0x00000010、0x0000000F、0x0000000E…0x00000001、0x00000000，无法动态演示过程，只能截取中间结果，测试情况如图 4.2 所示。

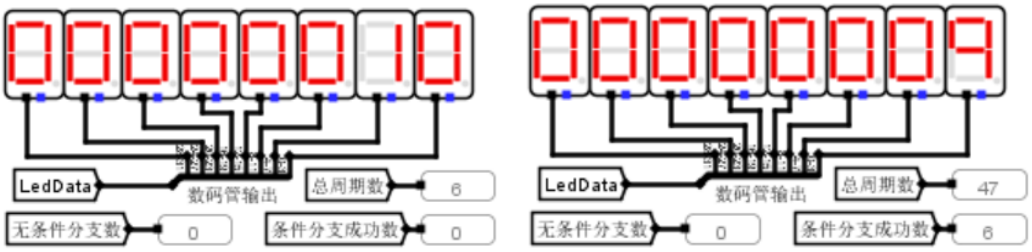


图 4. 2 SUBU 指令测试

测试结果正确。

（2） SLTIU 小于立即数置 1（无符号）

依次输出 0x00001996、0x00001995、0x00001994…0x0000194a、0x00001949，无法动态演示过程，只能截取中间结果，测试情况如图 4.3 所示。

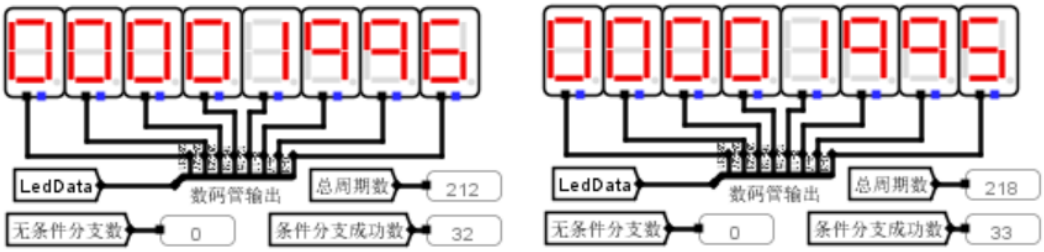


图 4. 3 SLTIU 指令测试

测试结果正确。

（3） SB 存储字节

依次输出 0x00000000、0x0000001、0x0000002…,无法动态演示过程，只能截取中间结果，测试情况如图 4.4 所示。

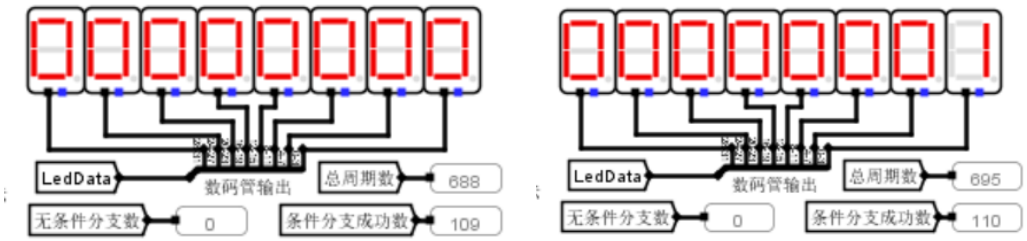


图 4. 4 SB 指令测试

测试结果正确。

（4） BLTZ 小于 0 转移

初始为 0xfffffff1、0xfffffff2、0xfffffff3…0xfffffffe、0xffffffff, 无法动态演示过程，只能截取中间结果，测试情况如图 4.5 所示。

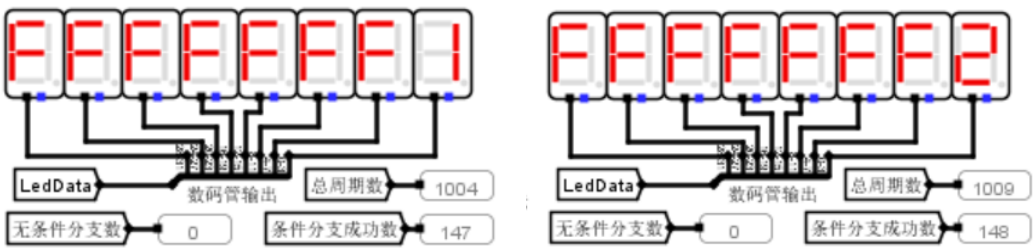


图 4. 5 BLTZ 指令测试

测试结果正确。

4.1.2 单级中断

正常进入主程序如图 4. 6 所示。

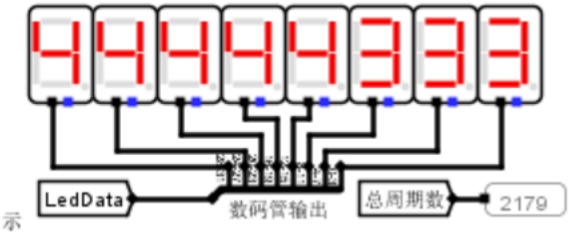


图 4. 6 进入主程序

（1）点击 1 号中断的时候进入 1 号中断子程序，当 1 号中断子程序执行完后返回主程序如图 4. 7 所示。

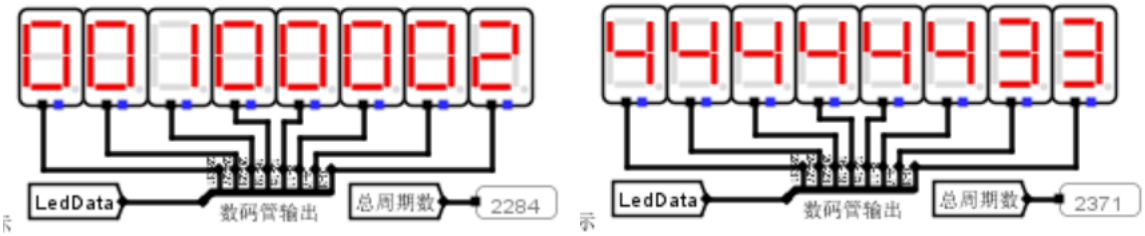


图 4. 7 1 号中断程序进入和返回

（2）点击 2 号中断的时候进入 2 号中断子程序，当 2 号中断子程序执行完后返回主程序如图 4. 8 所示。

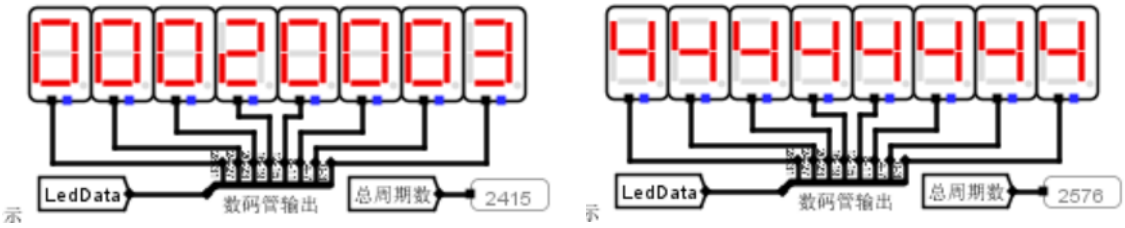


图 4. 8 2 号中断程序进入和返回

（3）点击 3 号中断的时候进入 3 号中断子程序，当 3 号中断子程序执行完后返回主程序如图 4. 9 所示。

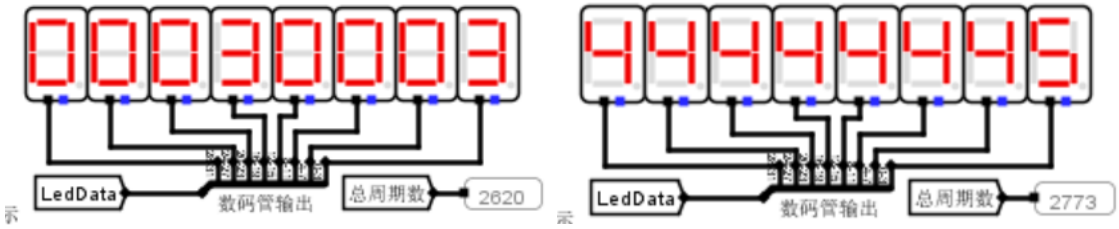


图 4. 9 3 号中断程序进入和返回

测试结果正确。

4.1.3 多级中断

多级中断一次可以输入多个中断号，根据优先级高低选择执行。测试时先正常执行，然后按下中断按钮的顺序为 1、2、3、1，当执行完 1、2、3、1 号程序后返回主程序，测试结果如下。

正常进入主程序如图 4. 10 所示。

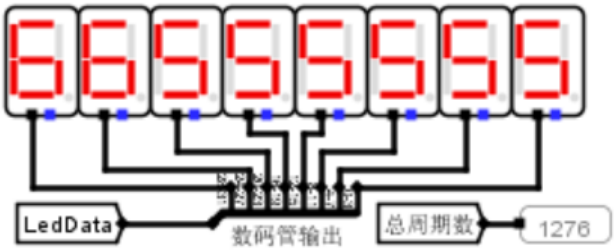


图 4. 10 进入主程序

点击 1、2、3 号中断，进入 3 号中断子程序，如图 4. 11 所示。

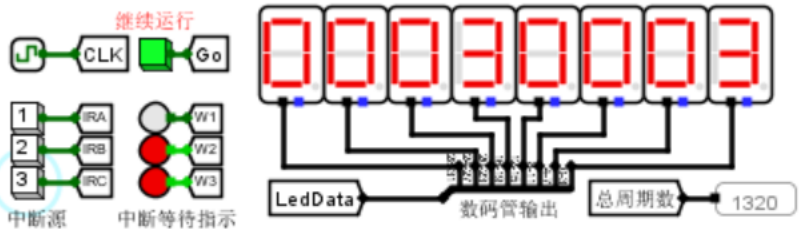


图 4. 11 多级中断 3 号中断

执行完 3 号中断子程序后进入 2 号中断子程序，如图 4. 12 所示。

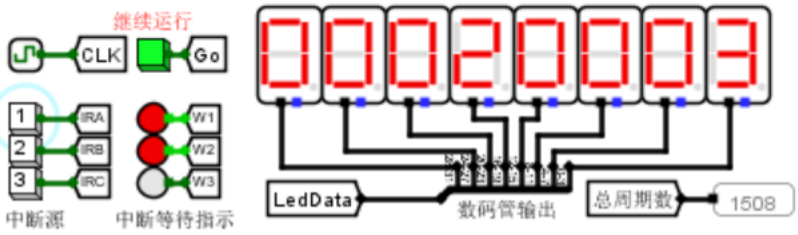


图 4. 12 多级中断 2 号中断

执行完 2 号中断子程序后进入 1 号中断子程序，如图 4. 13 所示。

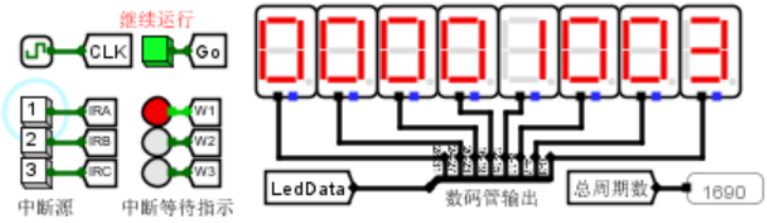


图 4. 13 多级中断 1 号中断

执行完 1 号中断子程序后返回主程序，如图 4. 14 所示。

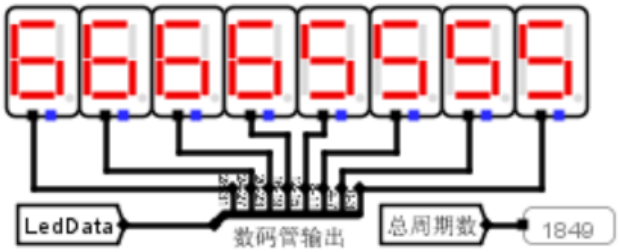


图 4. 14 返回主程序

4.1.4 理想流水线

加载理想流水线.hex 标准程序测试功能，各个周期数如图 4. 15 所示。

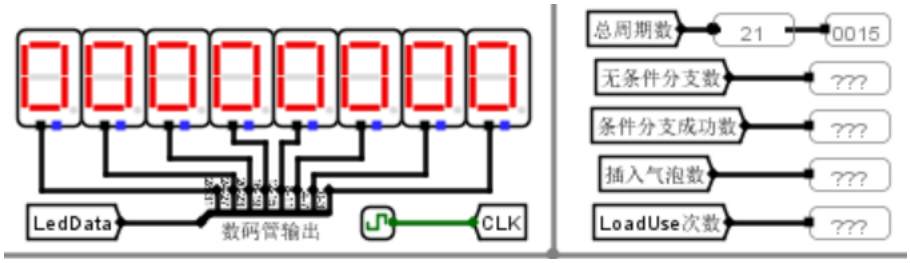


图 4. 15 理想流水线测试

运行完毕后，查看数据存储器中的内容，如图 4. 16 所示。

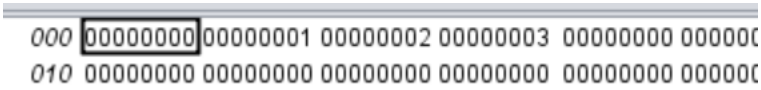


图 4. 16 数据存储器内容

测试结果正确。

4.1.5 气泡流水线

加载 benchmark.hex 标准程序测试功能，各个周期数如图 4. 17 所示。

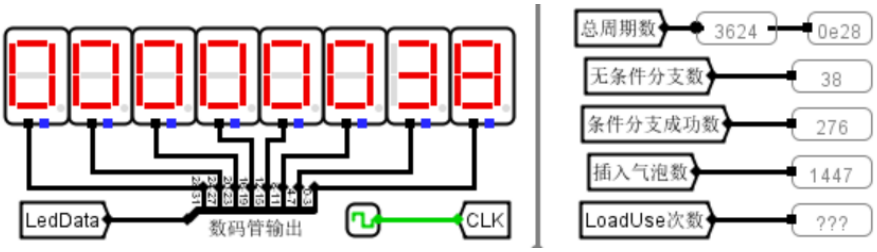


图 4. 17 气泡流水线测试

测试结果正确。

4.1.6 重定向流水线

加载 benchmark.hex 标准程序测试功能，各个周期数如图 4. 18 所示。

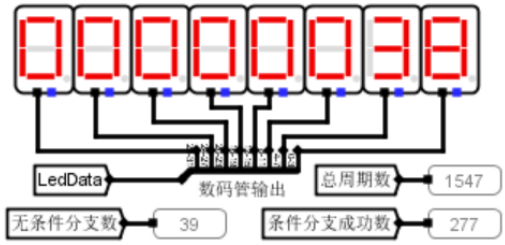


图 4. 18 重定向流水线测试

测试结果正确。

4.2 性能分析

（1） 理想流水线不考虑气泡和冲突冒险，故总周期最少；

（2） 气泡流水线与单周期 CPU 相比总周期数增加了一倍多，虽然每个周期的时间比单周期要短，但大量时钟周期 CPU 在空转，插入了大量的气泡，性能不好。

（3） 总周期数介于单周期 CPU 和气泡流水线之间，性能相较于旗袍流水线得到了极大的提升。

4.3 主要故障与调试

4.3.1 中断故障

没有对中断信号产生装置进行清零操作。

故障现象： 进入中断后，中断信号一直存在，无法实现多级中断。

原因分析： 在中断信号产生电路中只设计了中断信号的产生电路，没有设计中断信号的清零电路。

解决方案：在中断信号产生电路中设计一个中断信号清零电路，当中断执行完毕需要对中断执行清零操作，退出中断。

多级中断，高优先级中断无法优先执行故障。

故障现象： 依次点击 1、2、3 号中断按钮，3 号中断无法优先执行

原因分析： 中断请求信号接错。

解决方案： 将中断请求信号正确接入优先编码器，输出编码代表当前优先级最高的中断，正确将各中断的入口地址用常数的形式接到多路选择器的输入。

4.3.2 总周期数故障

理想流水线 CPU：benchmark 程序运行总周期数与预期不符。

故障现象： 运行 benchmark 标准测试程序后，总周期数与预期结果不符。

原因分析： 数据通路中的指令逻辑不对或者某些信号没有正确缓存。

解决方案： 对照表达式生成表，按照理想流水线数据通路逐条指令走一遍，找出有误的指令逻辑，修改对应的数据通路；检查接口部件，找出没有正确缓存的信号，修改接口部件逻辑。

4.3.3 Mars 打不开故障

无法打开 Mars

故障现象： 下载组原课设资料发布包后，配置环境，打不开 Mars。

原因分析： 可能与本地的 Java 工具 JDK 版本冲突，导致无法打开 Mars。

解决方案： 将扩展指令 SUBU.asm、SLTIU.asm、SB.asm、BLTZ.asm 写入一个.asm 文件，发给可以打开 Mars 的同学帮忙生成.hex 文件。

4.4 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	复习组成原理相关只是，制定计划。
第二天	根据相关资料，设计单周期 CPU 的数据通路。
第三天	完成单周期 CPU 的数据通路。完成扩展指令的设计和实现
第四天	填写控制器自动生成表。
第五天	测试调试，通过单周期 CPU 的 educoder 关卡。
第六天	设计实现接口实现，完成理想流水线。
第七天	查阅资料，设计气泡逻辑，实现气泡流水线。
第八天	查阅资料，设计重定向逻辑，实现重定向流水线。
第九天	完成单级中断，通过 educoder。
第十天	完成多级嵌套中断，通过 educoder

5 设计总结与心得

5.1 课设总结

本次课设总结如下：

1） 单周期 CPU 实现了 24 条基本指令+4 条扩展指令分别为 SUBU（无符号减）、SLTIU（小于立即数置 1（无符号））、SB（存储字节）、BLTZ（小于 0 转移）。在其基础上加入中断逻辑，实现了单级中断和多级嵌套中断。对单周期 CPU 指令执行阶段分成五个阶段：取指令 IF、指令译码 ID、指令执行 EX、访存 MEM、写回 WB，正确设计实现接口部件，得到理想流水线。在理想流水线基础上添加数据相关检测和控制冲突处理逻辑实现气泡流水线。对气泡流水线进行优化，增加 Load-Use 逻辑，实现重定向流水线。

2） 理想流水线不考虑气泡和冒险冲突，运行最快，执行的总周期数最少；气泡流水线实现流水执行，但是插入了大量的气泡，性能差，重定向流水线增加 Load-Use 逻辑，大大提升性能；多级中断支持高优先级打断低优先级中断。

5.2 课设心得

本课设所需设计的内容总体说很多，任务量很大，具有挑战性十几个日夜的不懈努力才终于完成了整个课程设计，虽然没能完成所有的任务，但还是历练到了自己。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课设建立在组原实验基础上，由于上学期的组原课程学习浅薄，课程设计刚刚开始的时候举步维艰，好在老师同学悉心解答疑问，课设得以向前推进。首先是设计单周期 CPU，该任务和上学期实验课的任务类似。但是在设计实现 4 条扩展指令时，我将弄错了控制信号，导致进度延缓，好在及时发现了问题所在。

流水线的设计需要设计实现接口部件，插入气泡、重定向技术处理冲突冒险；中断部分内容很多，高优先级中断可以打断低优先级中断，通过不断查阅资料，最终实现了多级嵌套中断，此刻尤为感到自我学习能力的必要性。

随着课设进行到后面，突然发现前面有个地方出了问题，导致后面做的全部不能用，得重新来一遍，实在得不偿失，由此凸显版本备份的重要性。

然而对于本次课程设计，我还有一些小小的建议和改进。有的同学可能实力强劲，天资过人，很容易完成课设要求，几天就完成了所有的任务，但是也有很多同学基础薄弱，课设进度缓慢，到最后也无法完成所有的任务，由此造成团队建设难以进行，所以希望老师分组的时候应该是随机组队，而不是仍由同学自己挑人组队，否则这样的结果就是“圣益圣，愚益愚”。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我坚信组成原理课程设计是我大学生涯中无比难忘的回忆。

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] 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.

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：王天阳