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：	CS1804
学　　号：	U201814755
姓　　名：	彭子晨
电　　话：	15258242073
邮　　件：	1289513580@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

（1） 支持表 1-1 前 27 条基本 32 位 MIPS 指令；

（2） 支持教师指定的 4 条扩展指令；

（3） 支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（4） 支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（5） 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。

（6） 能运行教师提供的标准测试程序，并自动统计执行周期数

（7） 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	SRAV	算术可变右移	
29	LUI	立即数加载至高位	
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 本次我们采用的方案是硬布线控制，且实现指令和数据分开存储的方式完成方案的设计。在一个周期内，控制器根据读入的指令给出相应控制信号，同时其余各功能部件根据得到的控制信号完成相应功能的实现，如对存储器的存储、对寄存器组的访问等。在实施过程中，全部采用 Logisim 仿真实现。

总体结构图如图 2-1 所示。

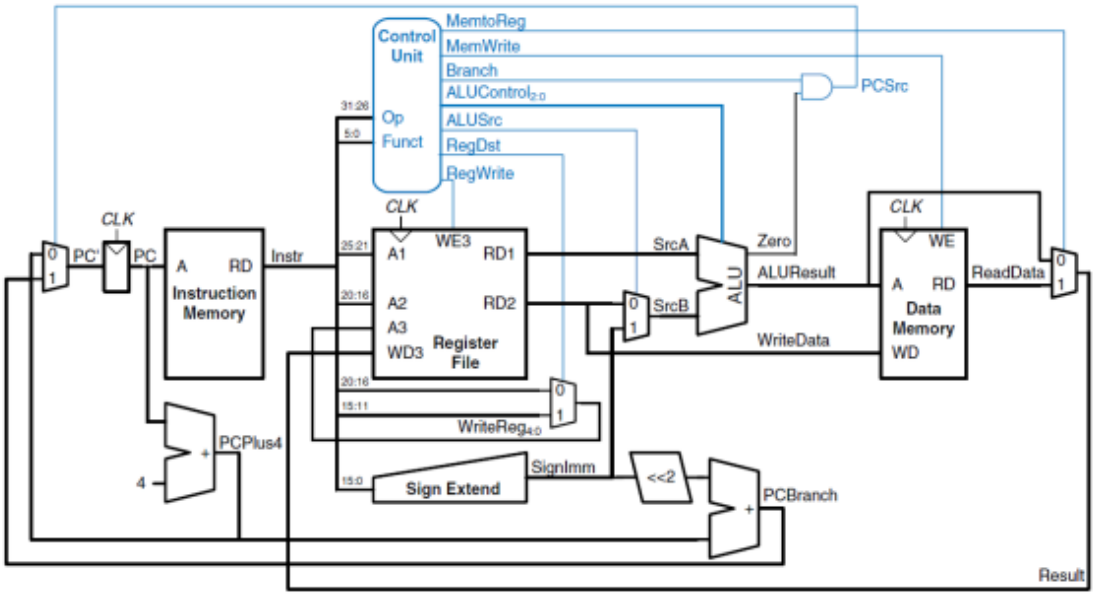

图 2-1 总体结构图

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。具体设计思路如下：

1. 程序计数器 PC

程序计数器 PC 用于存储下一条指令在指令存储器中的地址。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、时钟信号和复位信号；

输出为取值阶段的指令地址，在 logsim 中使用时钟上升沿触发的寄存器。

2. 指令存储器 IM

用于存储二进制形式的MIPS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。在 logisim 的具体实现中，使用 ROM 器件即可，注意要进行指令的装载，即右键将对应的指令集加载到改 ROM 中。指令的加载直接手动实现即可。在取指令时根据 PC 寄存器输出端口的值来进行指令的读取，一条指令占 4 个字节，所以需要将读取的 PC 的值去掉末两位再取第 2-11 位作为地址进行读取。

3. 运算器 ALU

运算器的引脚与功能描述如表 2.1 所示。相应的 ALUOP 和对应具体的功能如表 2.2 所示。

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 操作码与运算功能的对应关系表

ALUOP	运算功能
0	Result = X << Y 逻辑左移 （Y 取低五位）Result2=0
1	Result = X >>>Y 算术右移 （Y 取低五位）Result2=0
2	Result = X >> Y 逻辑右移 （Y 取低五位）Result2=0

3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 无符号乘法
4	Result = X/Y; Result2 = X%Y 无符号除法
5	Result = X + Y (Set OF/UOF)
6	Result = X - Y (Set OF/UOF)
7	Result = X & Y 按位与
8	Result = X | Y 按位或
9	Result = X⊕Y 按位异或
10	Result = ~(X |Y) 按位或非
11	Result = (X < Y) ? 1 : 0 符号比较
12	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	功能描述
R1#	读寄存器 1 的编号
R2#	读寄存器 2 的编号
W#	写寄存器的编号
Din	写入的数据
WE	写使能
CLK	时钟信号
R1	[R1]号寄存器中的数据
R2	[R2]号寄存器中的数据

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

根据 MIPS 文档逐个分析 28 条指令，得到如下表 2.4 所示的 28 条指令数据通路分析表：

表 2-4 28 条指令对应数据通路分析表

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
SLL	PC+4	PC	rs	rt	rd	ALU		R2	0		
SRA	PC+4	PC	rs	rt	rd	ALU		R2	1		
SRL	PC+4	PC	rs	rt	rd	ALU		R2	2		
ADD	PC+4	PC	rs	rt	rd	ALU	R1	R2	5		
ADDU	PC+4	PC	rs	rt	rd	ALU	R1	R2	5		
SUB	PC+4	PC	rs	rt	rd	ALU	R1	R2	6		
AND	PC+4	PC	rs	rt	rd	ALU	R1	R2	7		
OR	PC+4	PC	rs	rt	rd	ALU	R1	R2	8		
NOR	PC+4	PC	rs	rt	rd	ALU	R1	R2	10		
SLT	PC+4	PC	rs	rt	rd	ALU	R1	R2	11		
SLTU	PC+4	PC	rs	rt	rd	ALU	R1	R2	12		
JR	R1	PC	rs								
SYSCALL	PC+4	PC	2	4							
J	(PC+4)[31:28]||IMM26||00	PC									
JAL	(PC+4)[31:28]|IMM26||0	PC			31	PC+4					
BEQ	PC+4/PC+4+IMM<<2	PC	rs	rt			R1	R2			
BNE	PC+4/PC+4+IMM<<2	PC	rs	rt			R1	R2			
ADDI	PC+4	PC	rs		rt	ALU	R1	IMM	5		

指令	PC	IM	RF				ALU			DM	
			R1#	R2#	W#	Din	A	B	OP	Addr	Din
ANDI	PC+4	PC	rs		rt	ALU	R1	IMM	7		
ADDIU	PC+4	PC	rs		rt	ALU	R1	IMM	5		
SLTI	PC+4	PC	rs		rt	ALU	R1	IMM	11		
ORI	PC+4	PC	rs		rt	ALU	R1	IMM	8		
LW	PC+4	PC	rs		rt	Dout	R1	IMM	5	ALU	
SW	PC+4	PC	rs	rt			R1	IMM	5	ALU	R2
SRAV	PC+4	PC	rs	rt	rd	ALU	R1	R2	1		
LUI	PC+4	PC			rt	ALU	R1	IMM	15		
LHU	PC+4	PC	rs		rt	Dout	R1	IMM	5	ALU	
BGEZ	PC+4/PC+4+IM M<<2	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 2.5 所示。

表 2-5 主控制器控制信号的作用说明

控制信号	说明	取值(何时为 1)
RegWrite	寄存器写使能	寄存器写回信号
MemWrite	写内存控制信号	sw 指令，未单独设置 MemRead 信号
AluOP	运算器操作控制符（4 位）	R 型指令根据 Func 选择
MemToReg	寄存器写入数据来自存储器	lw 指令
RegDst	写入寄存器编号 rt/rd 选择	R 型指令
AluSrcB	运算器 B 输入选择	lw 指令，sw 指令，立即数运算类指令
SignedExt	立即数符号扩展	ADDI、ADDIU、SLTI 指令
JR	寄存器跳转指令译码信号	JR 指令
JAL	JAL 指令译码信号	JAL 指令 ，选择寄存器写回编号，写回值

JMP	无条件分支控制信号	J、JAL、JR 指令，选择无条件分支地址
Beq	Beq 指令译码信号	Beq 指令，用于有条件分支控制
Bne	Bne 指令译码信号	Bne 指令，用于有条件分支控制
Syscall	Syscall 指令译码信号	根据$V0 寄存器的值，决定是停机还是输出
LB	LB 指令译码信号	LB 指令
BGEZ	BGEZ 指令译码信号	BGEZ 指令，用于有条件分支控制

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

图 2-2 控制信号表

2.2 中断机制设计

2.2.1 总体设计

为单周期 MIPS 增加单级中断机制，可支持 1、2、3 共 3 个按键中断事件，中断优先级 1＜2＜3,CPU 执行中断服务程序时不能被其他中断请求中断。

按照事件发生的顺序，单级中断过程包括：

①中断源发出中断请求;

②判断当前处理机是否允许中断和该中断源是否被屏蔽;

③优先权排队;

④处理机执行完当前指令或当前指令无法执行完，则立即停止当前程序，保护断点地址和处理机当前状态，转入相应的中断服务程序;

⑤执行中断服务程序;

⑥恢复被保护的状态，执行“中断返回”指令回到被中断的程序或转入其他程序。

其中前四项操作由硬件完成，后两项由软件完成。

2.2.2 硬件设计

要为单周期 MIPS 增加单级中断机制，需要增加以下功能部件/中断信号：

1. 中断按键电路：接收中断请求。通过三个中断按键电路，按下 3 个按键分别表示接收 3 个中断事件；

2. 中断使能寄存器 IE 和关中断信号 ERET：判断当前是否处于中断过程中，以及是否接收中断信号；

3. 优先级判断电路：按照优先级依次处理中断信号；

4. 保护中断现场：处理机执行完当前指令或当前指令无法执行完，则立即停止当前程序，保护断点地址和处理机当前状态，转入相应的中断服务程序;

5. 中断信号清零；在处理完对应中断指令后应该对其进行清零操作。

2.2.3 软件设计

1. 中断按键电路

中断按键电路通过 IR 按键输入，使用 D 触发器保存输入状态，输出 INT1、INT2、INT3 信号表示当前处于的中断程序。直到中断程序处理完成后会通过 Clear 信号进行清零操作。

2. 中断使能寄存器 IE 和关中断信号 ERET

ERET 信号可以通过对输入指令 IR 进行比较得到，IE 信号可以通过判断当前处于中断状态且未关中断的逻辑通过一个 D 触发器得到。

优先级判断的电路主要是通过一个优先编码器实现，从而跳转到对应的中断程序

位置，处理完毕后清零当前中断请求。

4. 中断地址保存

通过寄存器保存中断前的 PC 地址到 EPC，并通过 ERET 信号判断当前是否为关中断，如果不是则向 PC 寄存器读入对应中断程序地址 INTADDR；如果是则恢复断点，向 PC 寄存器读入之前保存的 EPC。

2.3 流水 CPU 设计

2.3.1 总体设计

将单周期 CPU 修改为流水线 CPU。MIPS32 指令系统其指令的执行过程 细分为五个阶段，每个阶段由对应的功能部件完成，分别是：

- 取指令(IF)：负责从指令存储器取出指令；
- 指令译码(ID)：操作控制器对指令字进行译码，同时从寄存器堆取操作数；
- 指令执行(EX)：执行运算操作或计算地址；
- 访存(MEM)：对存储器进行读写操作；
- 写回(WB)：将指令执行结果写回寄存器堆。

MIPS 指令流水线在每个执行阶段的后面都需要增加一个流水寄存器，用于锁存本段处理完成的所有数据或结果，以保证本段的执行结果能在下一个时钟周期给下一个阶段使用，整体逻辑架构如图 2-3 所示。

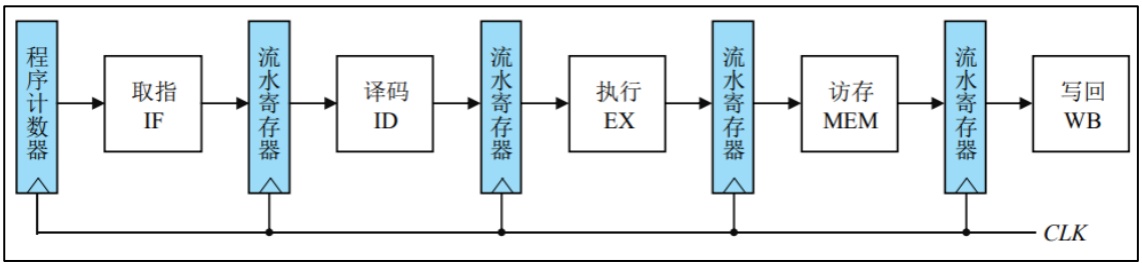

图 2-3 MIPS 指令流水线逻辑架构

2.3.2 流水接口部件设计

流水接口部件的设计基本上由输入信号、寄存器、输出信号构成，整体部件内的寄存器统一由时钟信号 CLK 和清零信号 CLR 以及使能信号 en 控制。

2.3.3 理想流水线设计

在单周期硬布线 CPU 的实现基础上，根据视频中的实验原理图将电路图分成五段, 并在段间插入流水接口部件，每个部件内实现需要寄存信号的寄存器。数据通路的结构大致相似，只需要正确进行分段即可。

2.4 气泡式流水线设计

在流水线设计中会存在各种结构冲突，如计算 PC+4、计算分支目标地址、运 算器运算都需要使用运算器，访问指令和访问数据都需要使用存储器。气泡流水线设计的解决方案是通过插入气泡延缓取指令的方式解决访存冲突。

当发生数据相关时给 ID/EX 流水寄存器一个同步清空信号 Flush；而要暂停 IF、ID 段指令执行，只需要控制寄存器使能端保证程序计数器 PC 的和 IF/ID 流水寄存器的值不变，因此只要将数据相关检测逻辑生成的数据相关信号 DataHazzard 作为暂停信号 Stall 取反后送对应的使能端即可。

2.5 重定向流水线设计

重定向流水线的设计思路是先不考虑 ID 段所取的寄存器操作数是否正确，而是等到指令实际使用这些寄存器操作数时再考虑正确性问题。

采用重定向机制后，需要相关处理逻辑检测出 Load-Use 相关；其他数据相关都以无阻塞的方式解决，相关处理逻辑只需要在 ID 段生成两个重定向选择信号 RsFoward、RtFoward 传输给 ID/EX 流水寄存器即可。当发生 Load-Used 相关时，需要得到相关处理逻辑阻塞信号 Stall、清空信号 Flush 暂停执行。

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，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。

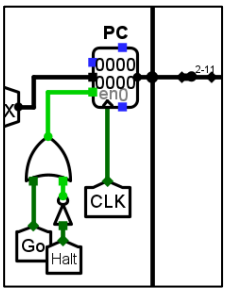

图 3-1 程序计数器（PC）

2) 指令存储器（IM）

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。

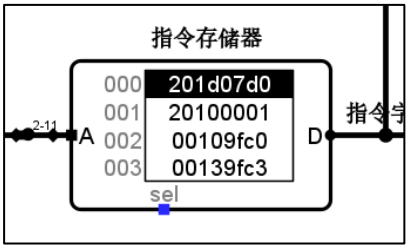

图 3-2 指令存储器（IM）

3) 运算器（ALU）

在 24 条指令的电路中使用老师提供的 ALU 即可。在 28 条指令中，为了实现 LUI 指令新增了一个 ALU_OP 为 15 时的计算，所以使用自建的 ALU_28 条指令。具体电

路如图 3-3，图 3-4 所示。

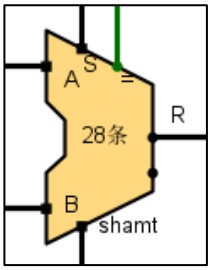

图 3-3 数据通路中的 ALU（28 条指令）

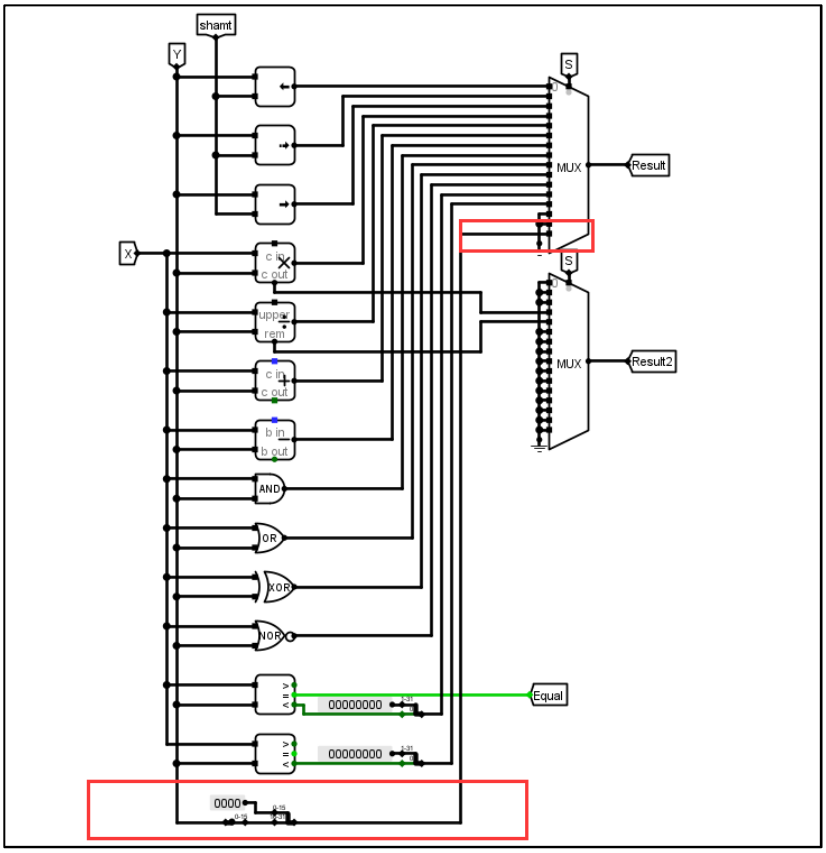

图 3-4 增加 LUI 指令的 ALU 实现

4) 寄存器堆（RF）

使用老师提供的 RF 即可，如图 3-5 所示。

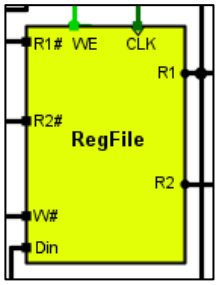

图 3-5 寄存器 RegFile

5) 数据存储器

使用 logisim 库中的 RAM 即可，如图 3-6 所示。

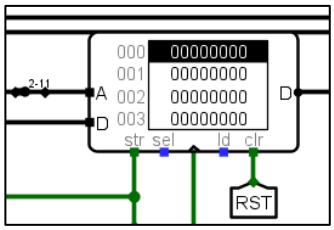

图 3-6 数据存储器

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			

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
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。

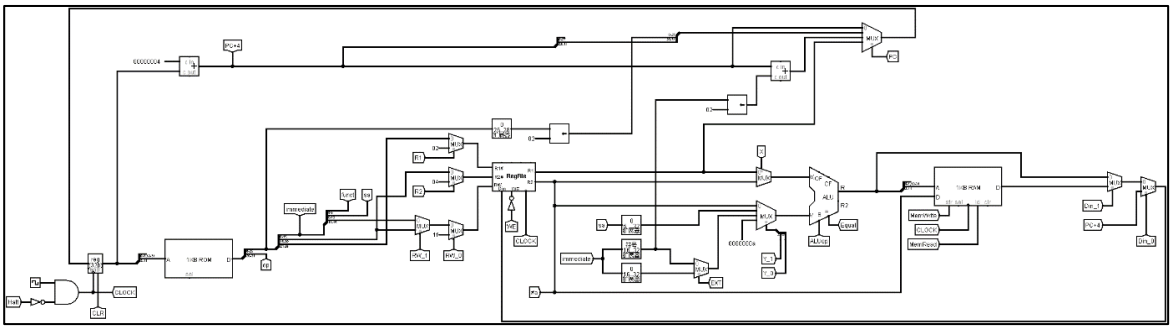

图 3-7 单周期 CPU 数据通路（Logism）

在 Vivado 中使用 Verilog 语言搭建的数据通路的原理图如图 3-8 所示。

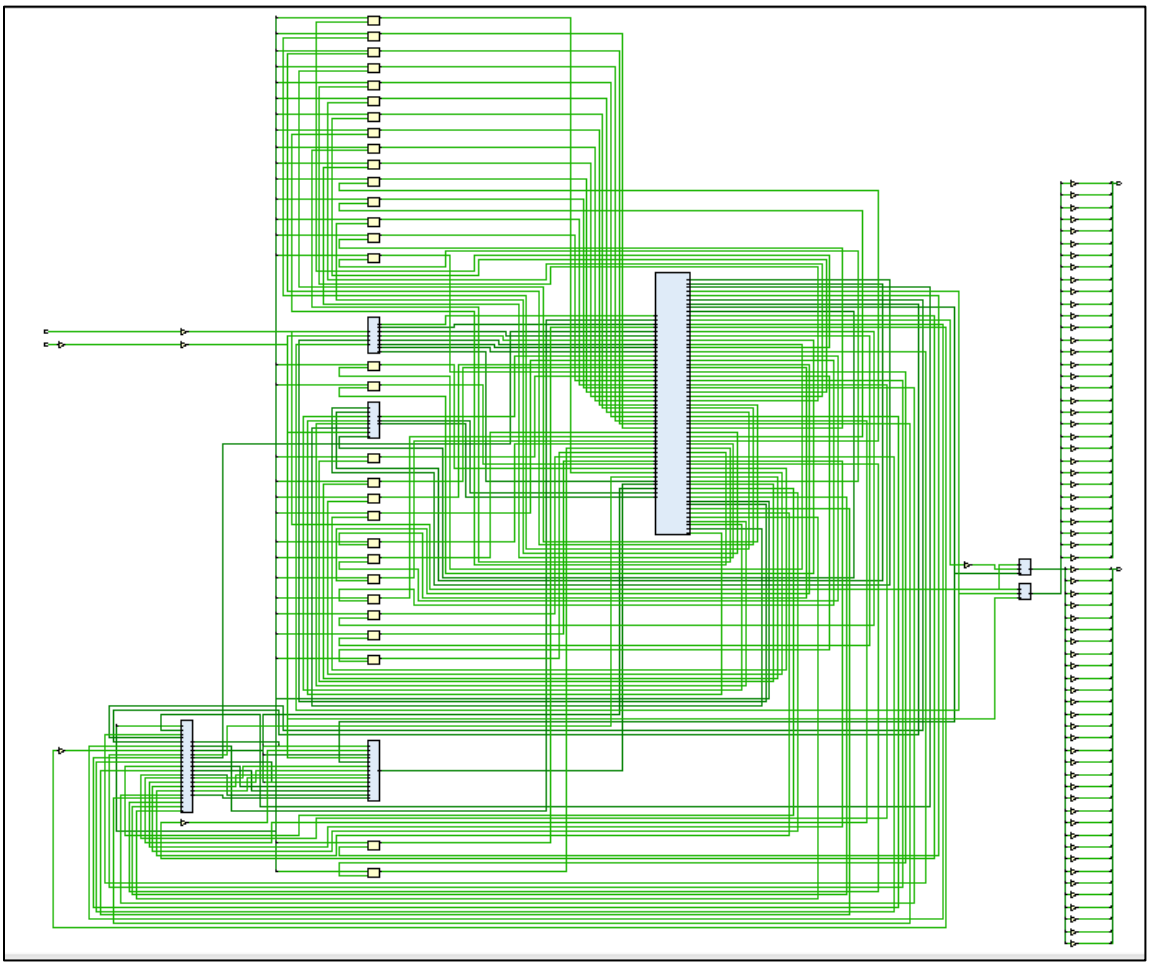

图 3-8 单周期 CPU 数据通路（FPGA）

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 所示控制信号表进行生成。

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 3-9 所示。

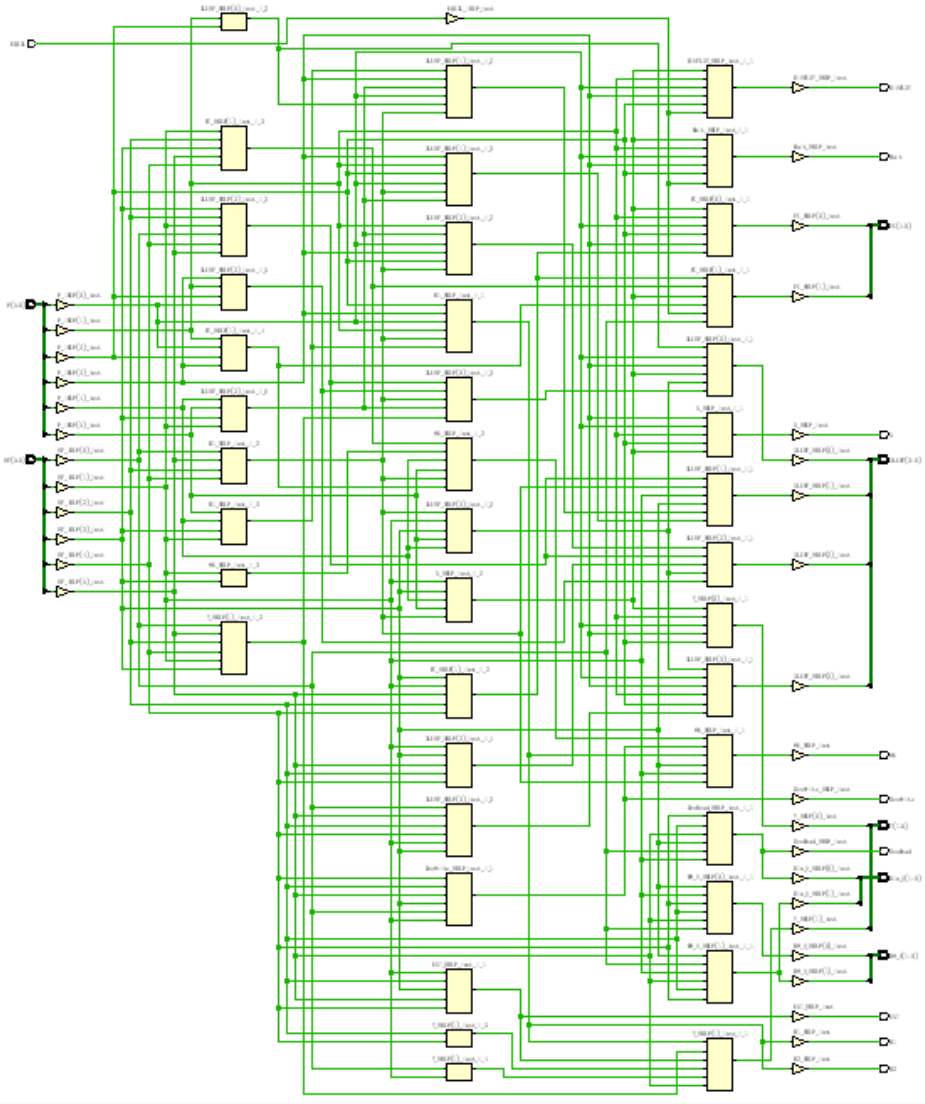

图 3-9 主控制器原理图

3.2 中断机制实现

3.2.1 主要部件的实现

1. 中断按键电路

中断按键电路通过 IR 按键输入，使用 D 触发器保存输入状态，输出 INT1、INT2、INT3 信号表示当前处于的中断程序。直到中断程序处理完成后会通过 Clear 信号进行清零操作。

具体电路设计如图 3-10 所示。

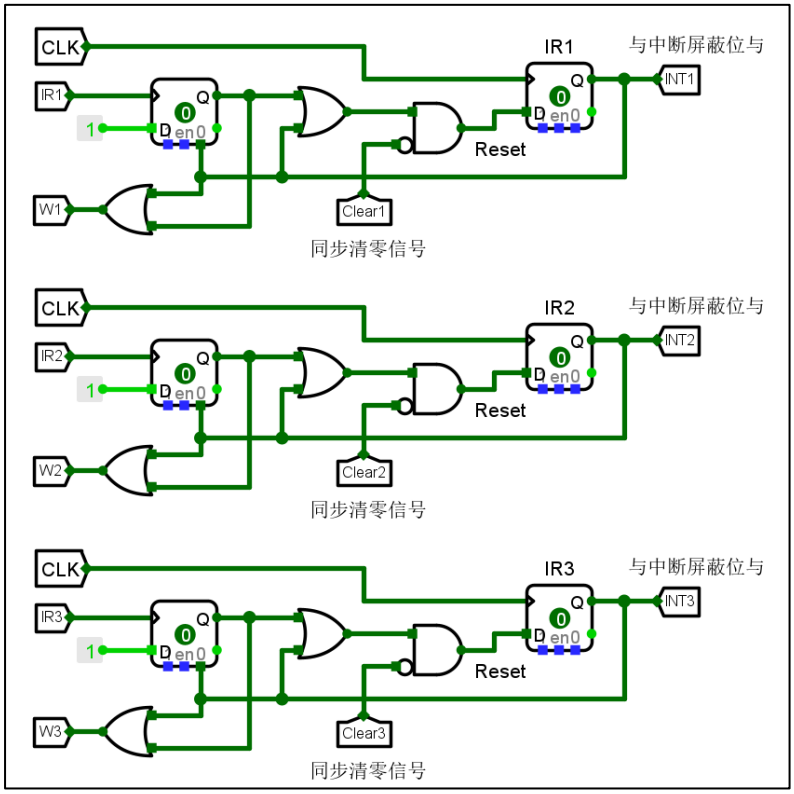

图 3-10 中断按键电路

2. 中断使能寄存器 IE 和关中断信号 ERET

ERET 信号可以通过对输入指令 IR 进行比较得到，IE 信号可以通过判断当前处于中断状态且未关中断的逻辑通过一个 D 触发器得到。电路如图 3-11，图 3-12 所示。

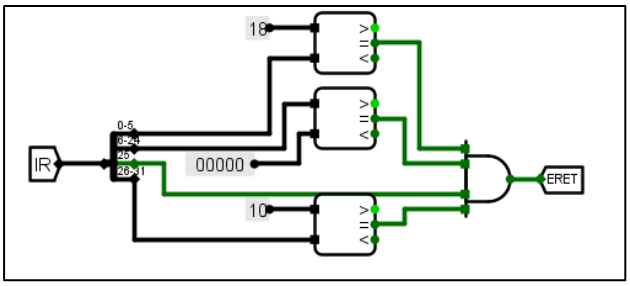

图 3-11 ERET 信号判断

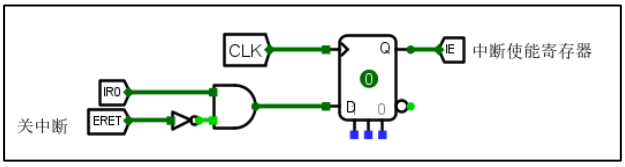

图 3-12 IE 信号

3. 优先级判断电路和中断信号清零

优先级判断的电路主要是通过一个优先编码器实现，从而跳转到对应的中断程序位置，处理完毕后清零当前中断请求。该电路如图 3-13 所示。

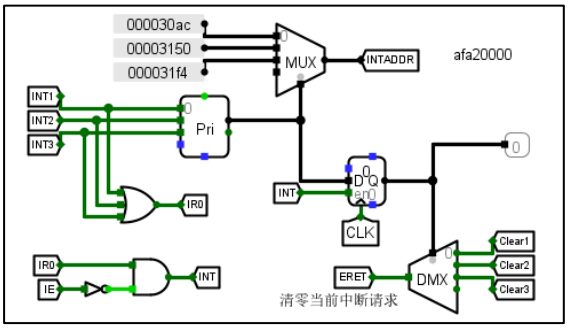

图 3-13 优先级判断和中断信号清零

4. 中断地址保存

通过寄存器保存中断前的 PC 地址到 EPC，并通过 ERET 信号判断当前是否为关中断，如果不是则向 PC 寄存器读入对应中断程序地址 INTADDR；如果是则恢复断点，向 PC 寄存器读入之前保存的 EPC。如图 3-14，图 3-15 所示。

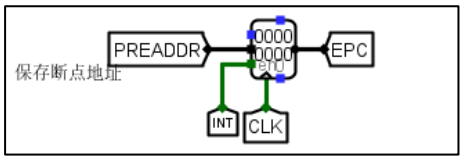

图 3-14 断点地址保存

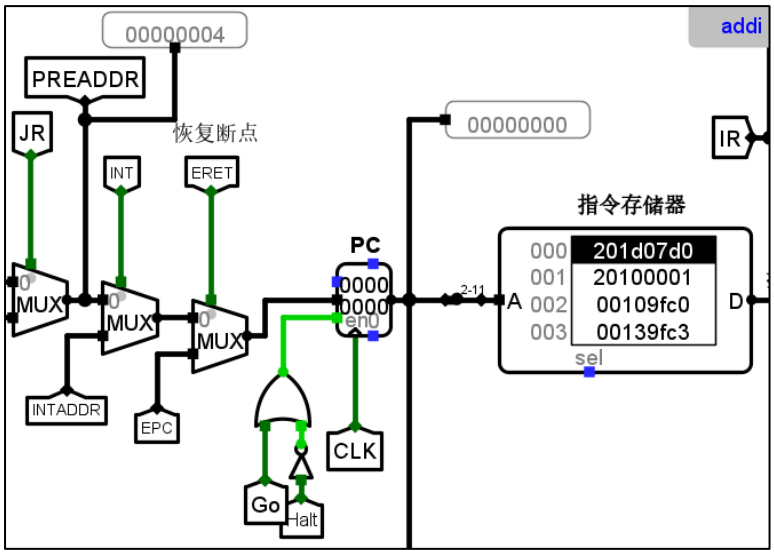

图 3-15 进入中断程序和断点恢复

3.2.2 整体数据通路

MIPS CPU 单级中断实现后完整的数据通路电路图如图 3-16 所示。

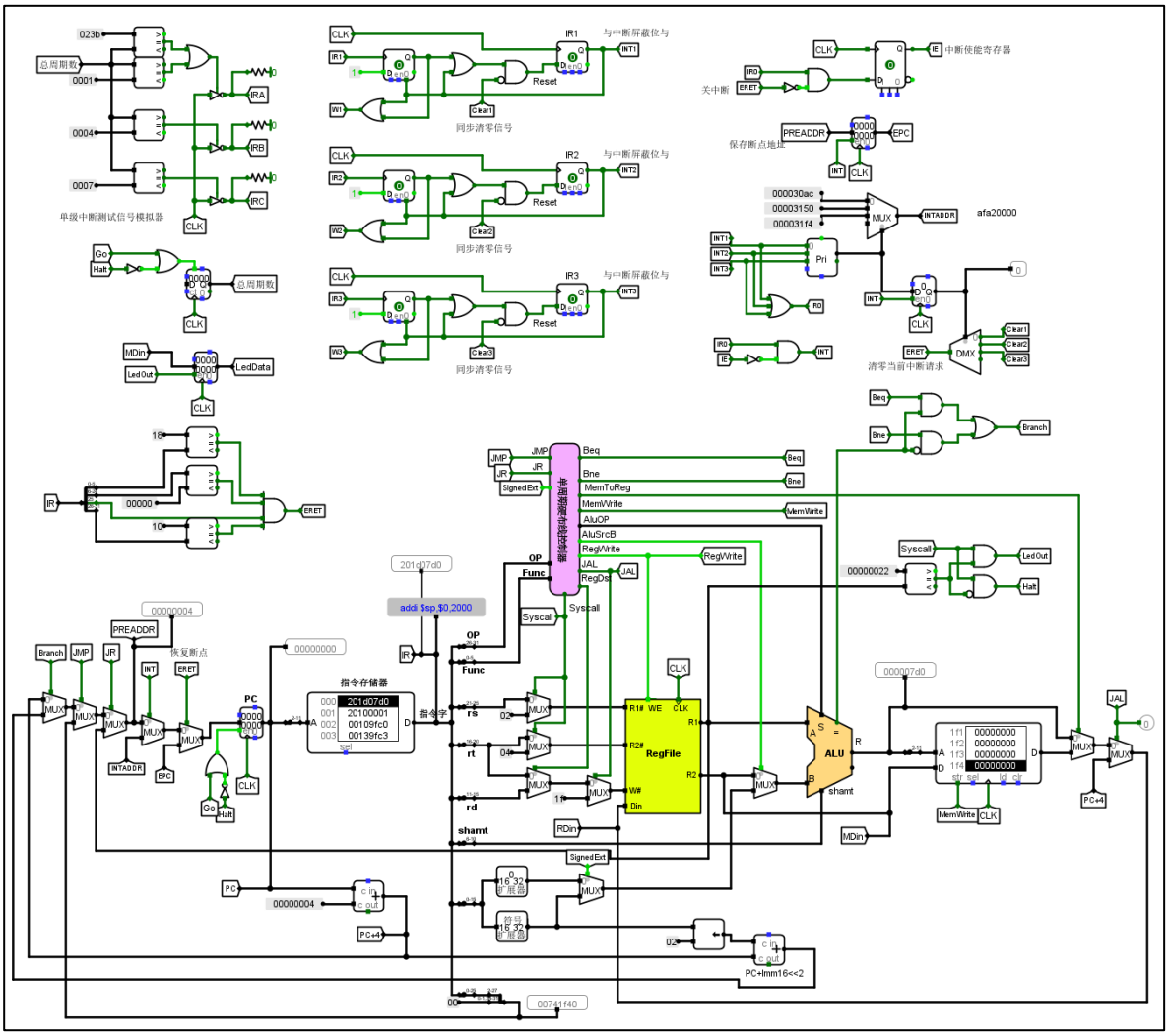

图 3-16 MIPS CPU 单级中断数据通路

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

从理想流水线，到气泡流水线再到重定向流水线，流水接口部件中的寄存器都是不断增加、不断更新的。四个流水线接口部件都是相同的形式，包括输入信号、寄存器、输出信号，以及控制整个接口部件的使能信号和同步清零信号。为节省篇幅，以下只展示 ID/EX 流水接口部件的实现截图。

ID/EX 段的接口部件如图 3-17 所示。

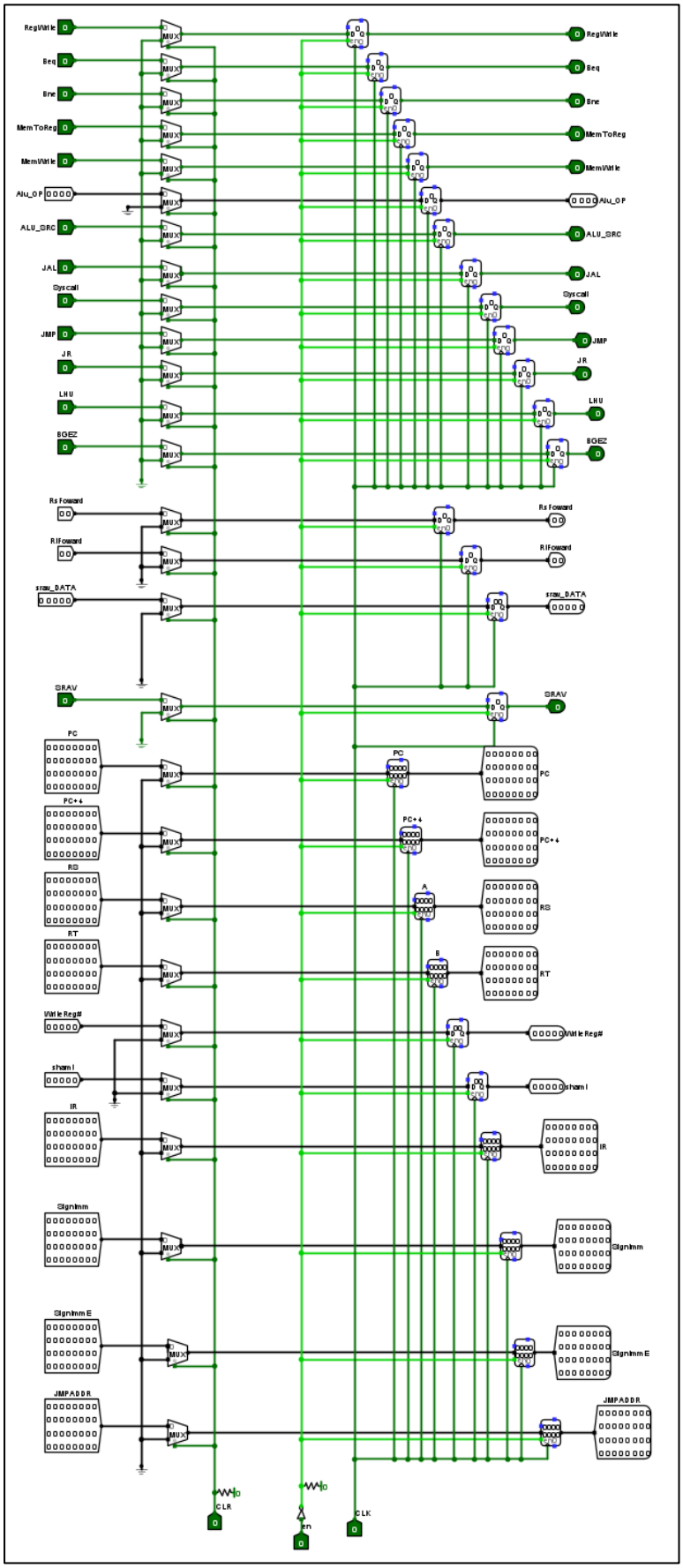
RegWrite
Beq
Bne
MemToReg
MemWrite
Alu_OP
ALU_SRC
JAL
Syscall
JMP
JR
LHU
BGEZ
Rs Foward
Rt Foward
srav_DATA
SRAV
PC
PC+4
RS
RT
Write Reg#
shamt
IR
SignImm
SignImmE
JMPADDR
MUX
CLR
CLK
en

图 3-17 ID/EX

3.3.2 理想流水线实现

在单周期硬布线 CPU 的实现基础上，根据视频中的实验原理图将电路图分成五段，并在段间插入流水接口部件，每个部件内实现需要寄存信号的寄存器。数据通路的结构大致相似，只需要正确进行分段即可。理想流水线数据通路如下图 3-18 所示。

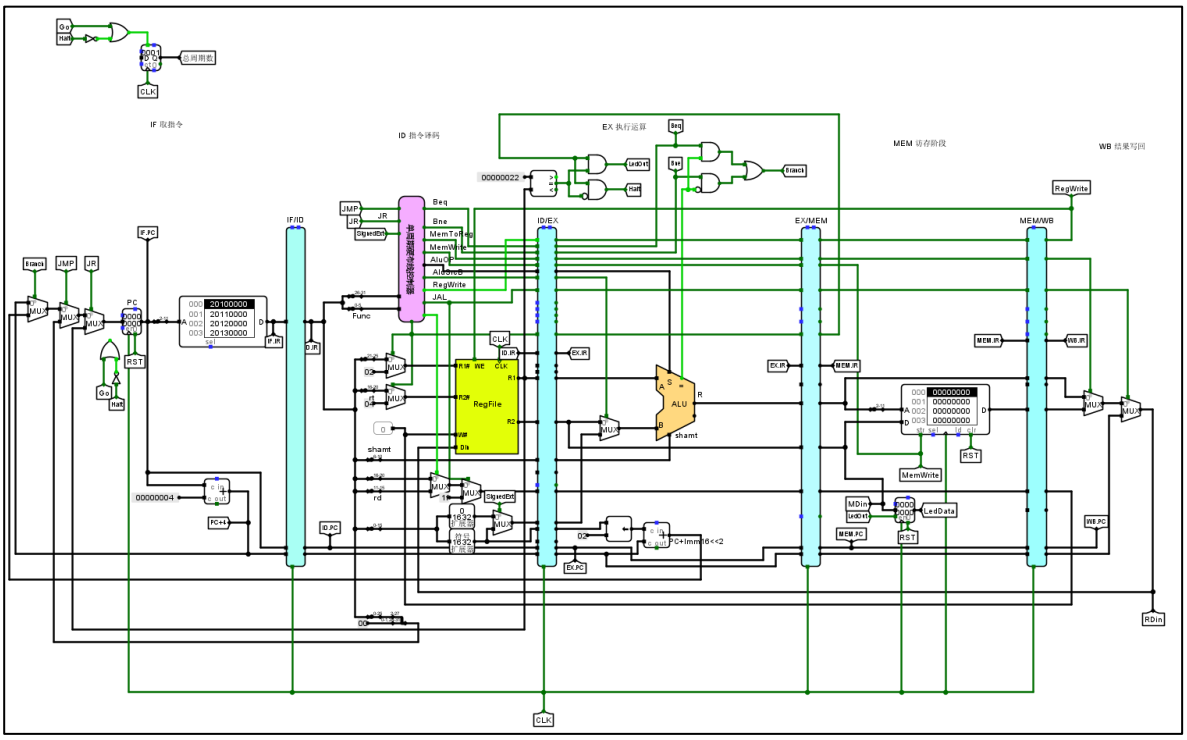

图 3-18 理想流水线数据通路

3.4 气泡式流水线实现

首先流水线中必须增加硬件逻辑实现 ID 段与 EX、MEM 段指令的数据相关性检测，MIPS 指令包括 0~2 个源操作数，分别是 rs、rt 字段对应的寄存器，其中 0 号寄存器恒零，不需要考虑相关性。要想确认 ID 段指令使用的源寄存器是否在前两条指令中写入，只需要检查 EX、MEM 段的寄存器堆写入控制信号 RegWrite 是否为 1，且写寄存器编号 WriteReg#是否和源寄存器编号相同即可。

有了数据相关检测逻辑，只需考虑如何暂停 IF、ID 段指令的执行以及如何插入气泡的问题，可以在发生数据相关时给 ID/EX 流水寄存器一个同步清空信号 Flush；而要暂停 IF、ID 段指令执行，只须保证程序计数器 PC 的和 IF/ID 流水寄存器的值不变即可，要做到这一点，需要控制寄存器使能端，当使能端为 1 时，寄存器正常工作，为 0 时则忽略时钟输入，寄存器值保持不变。只要将数据相关检测逻辑生成的数据相

关信号 DataHazzard 作为暂停信号 Stall 取反后送对应的使能端即可。

具体操作步骤主要是下面两步：

1. 在 excel 表格中增加 RsUsed/RtUsed 列，填写各指令对 Rs,Rt 的使用情况并生成电路，输入和控制信号生成一样，输出为 RtUsed/RsUsed 信号，如图 3-19 所示。

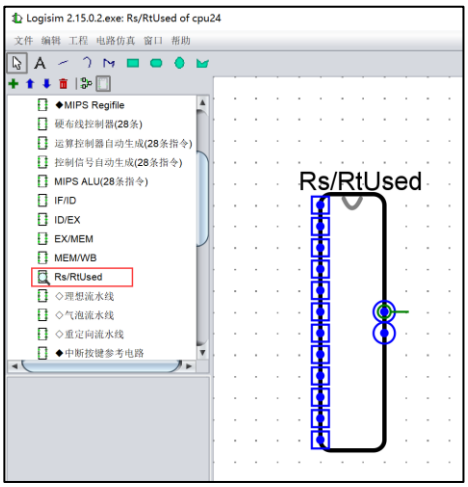

图 3-19 Rs 和 Rt 使用情况

2. 在实现了数据相关检测电路后，在理想流水线电路的基础上，利用产生的数据冲突信号控制气泡的插入和流水线的暂停，即可成功实现所需的功能。具体连线和逻辑参照书中的内容，最终气泡流水线的数据通路如下图 3-20 所示：

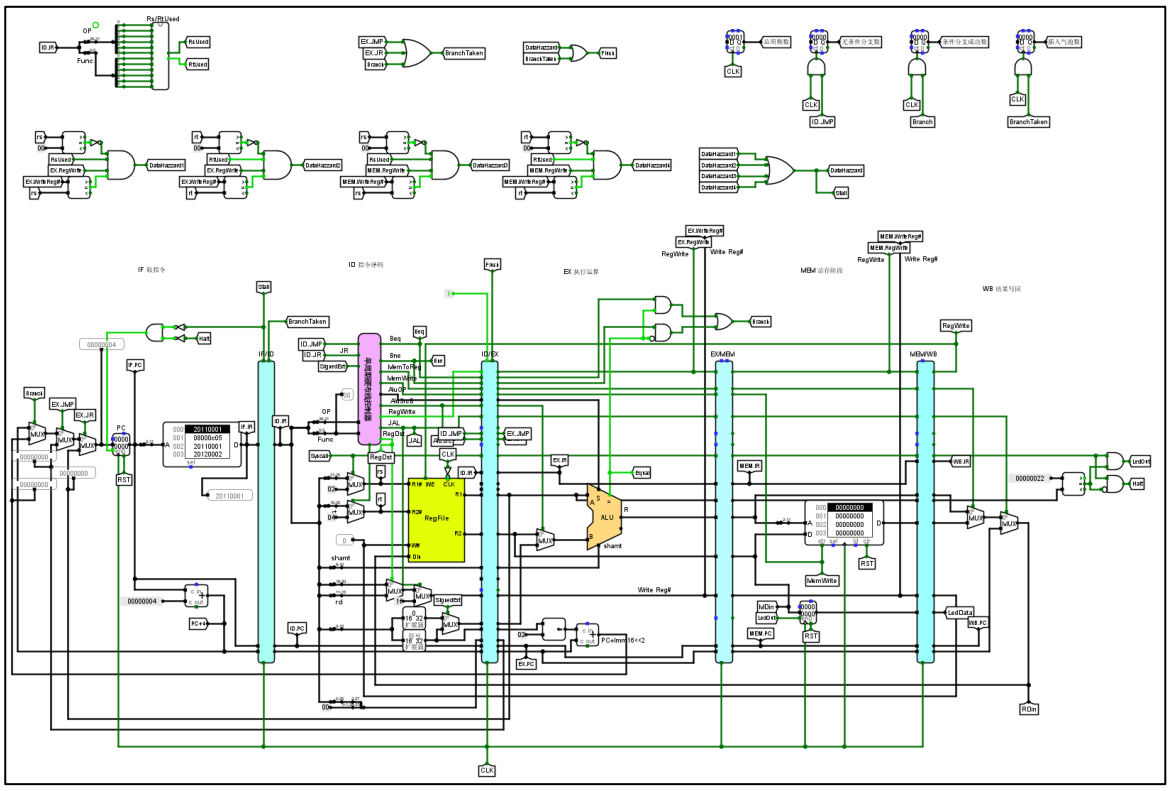

图 3-20 气泡流水线数据通路

3.5 重定向流水线实现

采用重定向机制后，由于重定向中 Load-Use 数据相关仍然需要通过插入气泡方式进行消除，所以相关处理逻辑应该能检测出 Load-Use 相关；其他数据相关都可以采用重定向方式以无阻塞的方式解决，相关处理逻辑只需要在 ID 段生成两个重定向选择信号 RsFoward、RtFoward 传输给 ID/EX 流水寄存器即可。

当发生 Load-Used 相关时，需要暂停 IF、ID 段指令执行、并在 EX 段插入气泡，需要控制 PC 使能端 EN、IF/ID 使能端 EN、ID/EX 清零端 CLR，而 EX 段执行分支指令时会清空 ID 段、EX 段中的误取指令，会使用 IF/ID 清零端 CLR、ID/EX 清零端 CLR。综合两部分逻辑，可以得到相关处理逻辑阻塞信号 Stall、清空信号 Flush。

利用已经实现的重定向相关检测电路，基于之前实现的气泡流水线，参照总体设计中的思路，为逻辑电路图增加重定向相关功能，即可完成重定向流水线的总体数据通路。具体连线和逻辑参照书中的内容。总体数据通路的实现如下图 3-21 所示：

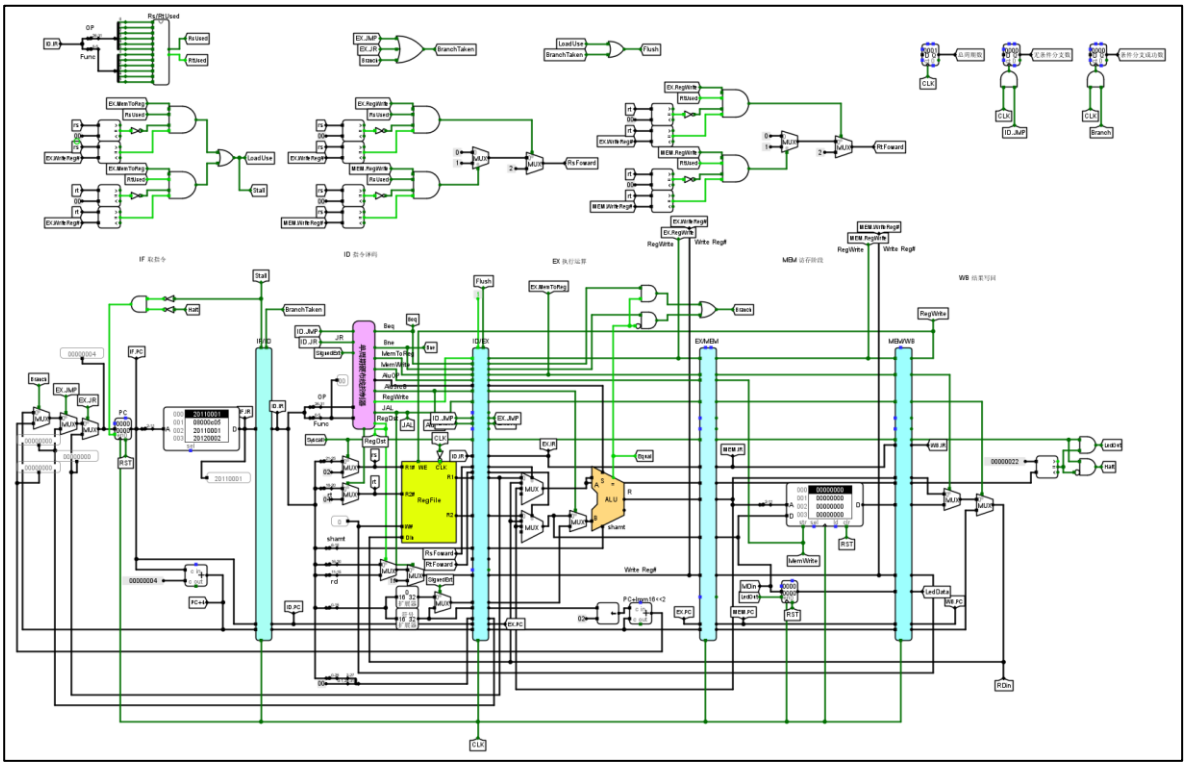

图 3-21 重定向流水线数据通路

4 实验过程与调试

4.1 测试用例和功能测试

由于单周期和流水线 CPU 测试已在 educoder 上验证通过，ccmb 也在老师的检查下通过了为节省篇幅，这里只放了两个测试用例。

4.1.1 单周期 MIPS CPU 测试

此测试中使用 benchmark.hex 进行测试，观察 LED 的变化情况，与程序预期现象一致. 也通过了 educoder 的测试。结果如下图 4-1 所示:

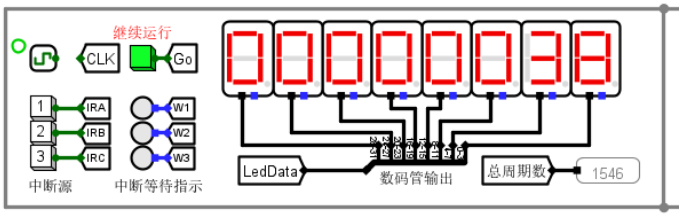

图 4-1 单周期 MIPS CPU 测试结果

打开数据存储器，观察数据的排列方式，发现已经成功按降序排序，正确实现功能。测试截图如下图 4-2 所示：

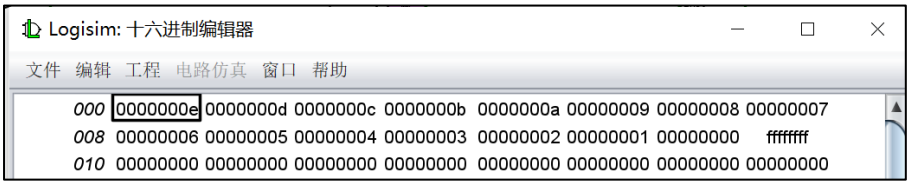

图 4-2 单周期 MIPS CPU 测试排序结果

4.1.2 单级中断测试

在同时有多个中断信号时，会优先处理优先级最高的一个，且处理过程中不会被打断。如下图 4-3 所示，是中断处理程序 3 正在运行的时候的截图。

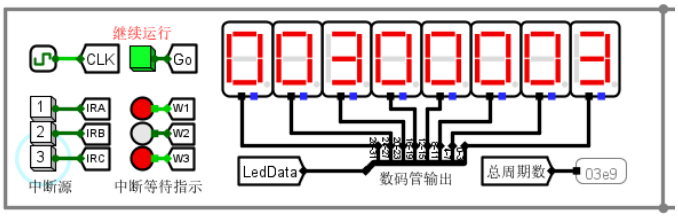

图 4-3 单级中断测试

4.2 性能分析

对于不同实现方式的 CPU 的实现 benchmark 得到的总周期数有差异。可以观察到，单周期 CPU 总周期数 1545,气泡流水线总周期数为 3623，重定向流水线总周期数为 2297。把 CPU 的频率提升到更高. 单周期 CPU 虽然周期数少，但由于它非常长的数据通路，门延迟导致单个周期的耗时很长，所以最终性能上，气泡流水线和重定向流水线都要比单周期 CPU 强很多。

而重定向流水线与气泡流水线相比，因为采用重定向计数，导致插入的气泡数大大减少，因此大大降低 CPU 流水线运行的停顿，在气泡流水线的基础上额外获得了一定的性能提升。

4.3 主要故障与调试

4.3.1 连线位置故障

理想流水线： 接口处数据传输问题。

故障现象：MemW、RegW、Mdin 输出错误。

原因分析：这三个信号连接的位置都不对，应该连在 MEM 访存段，而不是前面。

解决方案：将三个信号重新连到对应的位置。Mdin 的正确连法如图 4-4 所示。

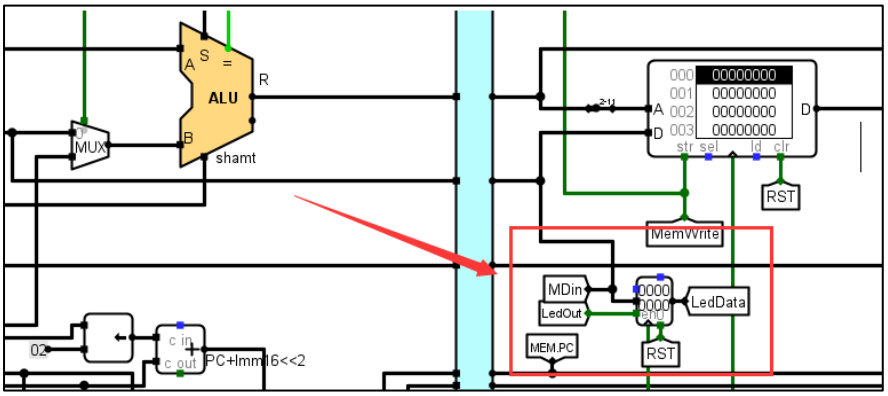

图 4-4 MDin 的正确位置

4.3.2 Syscall 问题

气泡流水线：RtUsed/RtUsed 生成错误。

故障现象： IF/ID/EX 和 PC 都显示错误。如图 4-5 所示。

图 4-5 程序故障图

原因分析：使用 ctrl+T 在 logisim 中单步执行直到出错的 CLK 位置，发现是遇到 syscall 指令没有插入气泡，如图 4-6 所示。 仔细检查后发现是 excel 内的 SYSCALL 这一行填表错误。

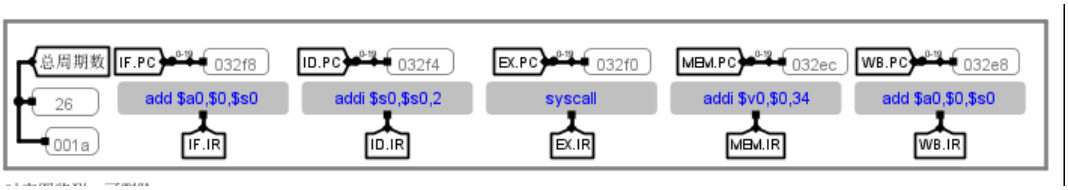

图 4-6 故障时间点

解决方案：重新填写 excel 表格，生成新的 RtUsed/RtUsed 电路。正确如图 4-7 所示。

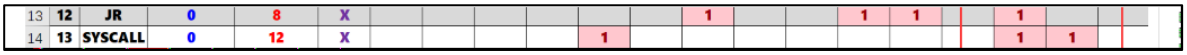

图 4-7 excel 填写

4.3.3 SRAV 指令

重定向流水线：SRAV 指令。

故障现象：使用流水线单独测试 SRAV 指令，LedData 只会显示 8,7,6,5,4,…

原因分析：通过 MARS 单步执行对比发现寄存器的值没有问题，是 ledout 的位置有误，导致在数据出现的时候没有及时储存到寄存器进行 LED 显示。调试过程中和寄存器内数据对比如图 4-8 所示。

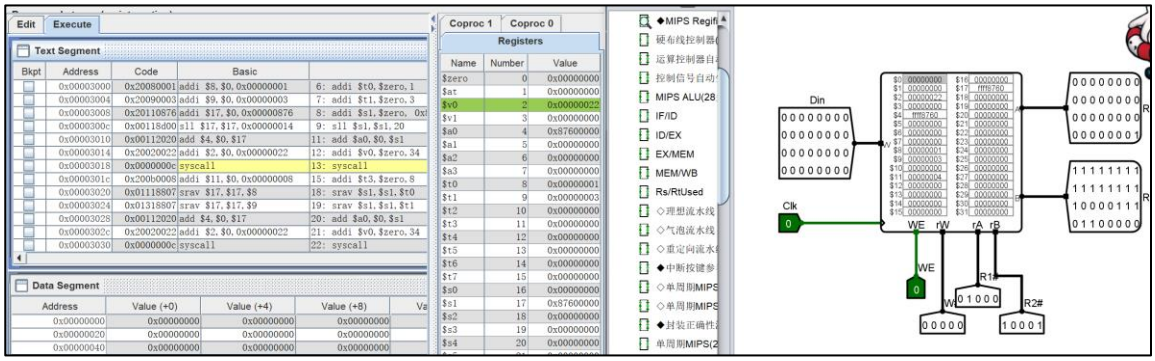

图 4-8 MARS 单步调试

解决方案：重新连接 LedData 的位置到写回段，同时把 LedOut 信号移到 EX 段。

正确的图如图 4-9 所示。

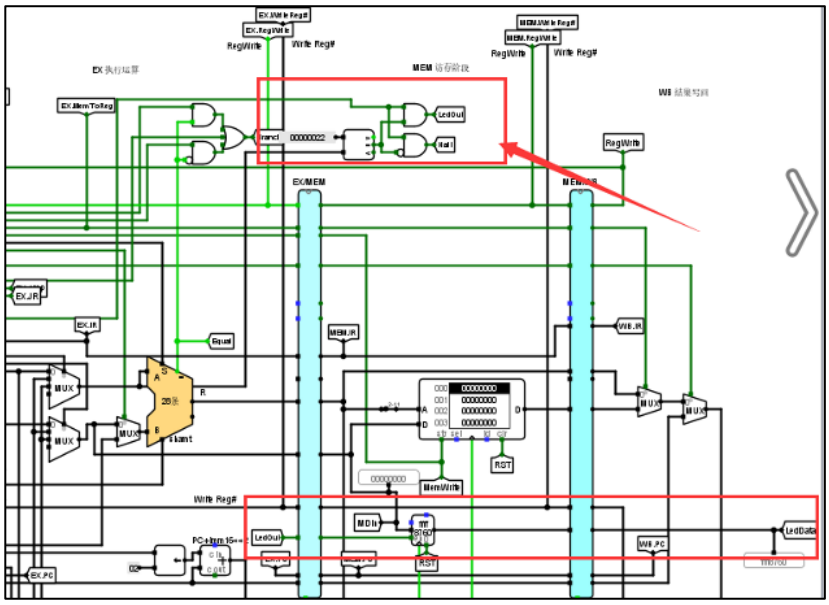

图 4-9 LedData 更改

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 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。使用 Verilog 实现了部分单周期 CPU 的重要部件，并通过仿真检查。
第四天	完成 Logism 单周期 CPU + 单级中断。
第五天	根据 MOOC 视频学习流水线相关，并完成理想流水线的大致数据通路，开始进行总体调试。
第六天	理想流水线的 logisim 已经调试通过，通过了 educoder 的在线检查。开始进行后面气泡流水线的工作。
第七天	根据老师发下来的流水线课件，参考书中的电路图和逻辑完成了气泡流水线的数据通路，并通过了 logisim 测试。
第八天	基于气泡流水线，参考书中的电路图和逻辑完成了重定流水线的数据通路，并

时间	进度
	通过了 logisim 测试。
第九天	为单周期 MIPS CPU 加入四条个人指令，很快成功。并尝试给流水线增加四条个人指令。
第十天	选择直接给重定向加入四条个人指令，成功给重定向流水线加入了四条个人指令，ccmb 调试通过。成功进行了检查。

5 团队任务

5.1 团队成员及分工

表 5-1 团队成员及分工

团队成员	职能	分工
彭子晨	组长	电路连线、代码撰写、讲解视频录制
闻佳佳	组员	视频取模、讲解视频制作
任海娇	组员	电路连线、演示视频制作
刘虹	组员	视频取模、演示 PPT 制作

5.2 项目概述

使用 28 条指令的单周期 CPU 设计实现了一个 32*32 大小、支持两色的视频演示系统。演示的视频选择的是 BadApple。

5.3 项目介绍

小组任务总共花费了 4 天左右完成。具体工作时间分配如表 5-2 所示。

表 5-2 团队任务工作进度

第 1 天	小组讨论团队任务内容，确定制作视频播放器。
第 2 天	完成视频的取模和 16 进制文件的转换。
第 3 天	完成 logisim 连线和汇编代码编写。
第 4 天	完成了成果视频、演示视频的录制，放到了 b 站上。

项目亮点：1. 使用 32*32 的 LED 点阵实现了图片的显示；

2. 使用自己编写的汇编代码实现了视频的逐帧显示；

5.4 项目实施

1.视频取模：使用 PotPlayer 对视频取帧，设置每秒截图时间进行截图，保存到统一的文件夹下，共计取到 2189 帧；用 PS 对截图进行裁剪压缩，并设置宽：高=1:1。由于是 32*32 点阵，故设置 32*32 像素，然后对图片使用批量处理；最后用图片取模工具将处理后的图片按顺序转换成能被单片机读取的十六进制数。

2.数据处理：在 nodepad++中使用正则表达式去除 0x/,/注释等文本，只留下 16 进制数；撰写一个 C++程序读取文件并重新输出成新文件，使其每 8 个字符一换行；用任意文本编辑器在新文件首行加入“v2.0 raw”，使其可以被 logisim 载入；

3.连线和代码编写：使用 32*32 的 LED 点阵显示视频，由于要逐行渲染，所以设计使用了 32 个寄存器存储每行的数据；使用 MemToReg 作为显示的时钟信号，这表示每次循环中使用 lw 取 32 位的模值；使用计数器，每次 MemToReg 信号触发就会使计数值加 1，通过译码器后作为当前要渲染的行的时钟信号，同时通过计数器可知道当前为第 x 行，数据存储器的输出端 Show 是获得的图片 x 行模值，触发对应的第 x 个寄存器，实现图片的渲染和显示；

4.调试及反馈：调试电路查看输出图像是否有误，进行反复调试；

5.PPT 和视频制作：制作 PPT、录制讲解视频、项目展示视频并上传到 b 站。

5.5 项目分析

5.5.1 问题及解决

1. 代码问题：在 MARS 中撰写汇编代码，使用 addi 增加大于 32768 的数时，会被自动分解为 lui,ori,add 三条指令，如图 5-1 所示。所以需要实现了 LUI 指令的 CPU 才能成功运行该代码。

Text Segment

Bkpt	Address	Code	Basic	Sou
	0x00003000	0x3c010001	lui $1, 0x00000001	3: addi $s1, $0, 100000 #s1记录循环次数
	0x00003004	0x342186a0	ori $1, $1, 0x000086a0	
	0x00003008	0x00018820	add $17, $0, $1	
	0x0000300c	0x20120000	addi $18, $0, 0x00000000	4: addi $s2, $0, 0 #s2记录当前图片位置
	0x00003010	0x8e440000	lw $4, 0x00000000($18)	6: lw $a0, 0($s2) #获取图片数据
	0x00003014	0x20020022	addi $2, $0, 0x00000022	7: addi $v0, $0, 34
	0x00003018	0x0000000c	syscall	8: syscall #输出当前值
	0x0000301c	0x22520004	addi $18, $18, 0x00000004	9: addi $s2, $s2, 4 #下一张图片
	0x00003020	0x2231ffff	addi $17, $17, 0xffffffff	10: addi $s1, $s1, -1
	0x00003024	0x1620fffa	bne $17, $0, 0xfffffffa	11: bne $s1, $0, bne_branch #测试指令
	0x00003028	0x2002000a	addi $2, $0, 0x0000000a	13: addi $v0, $zero, 10 #停机指令
	0x0000302c	0x0000000c	syscall	14: syscall #系统调用

图 5-1 代码转换

2. 显示延迟问题：最开始连完线后成功显示了视频图像，但是图像首行是从第 2 行开始的，第 0 行和第 1 行是底部两行的显示。最后发现是由于从数据存储器读取图片的时候有 1 个寄存器导致的延迟；在写入 S0~S31 时又有 1 个寄存器导致的延迟，两个时钟延迟导致图像从第 2 行开始正常显示。最后把显示器的数据输入改成了 S2~S31,S0,S1 后，视频正常显示。显示器的输入端电路如图 5-2 所示。

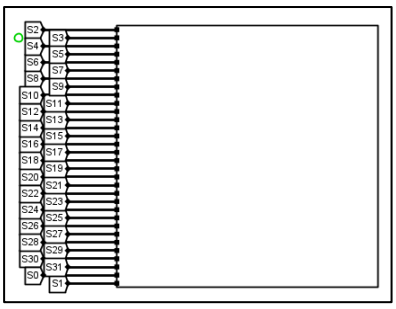

图 5-2 显示器输入端

5.5.2 优化及拓展：

1. 可以导出有颜色的视频，通过分析每个像素点的 RGB 值，实现彩色视频展示；

2. 使用 32 个寄存器和计数器的设计会导致图片是逐行渲染的，由于电脑性能的不足，logisim 最高运行频率在 200hz 左右，所以其实渲染速度很慢，最后的演示视频进行了后期加速。可以通过增加数据存储器的方式进行并行渲染，增加渲染速度。

5.6 成果展示

最终显示部分电路图如图 5-3 所示。

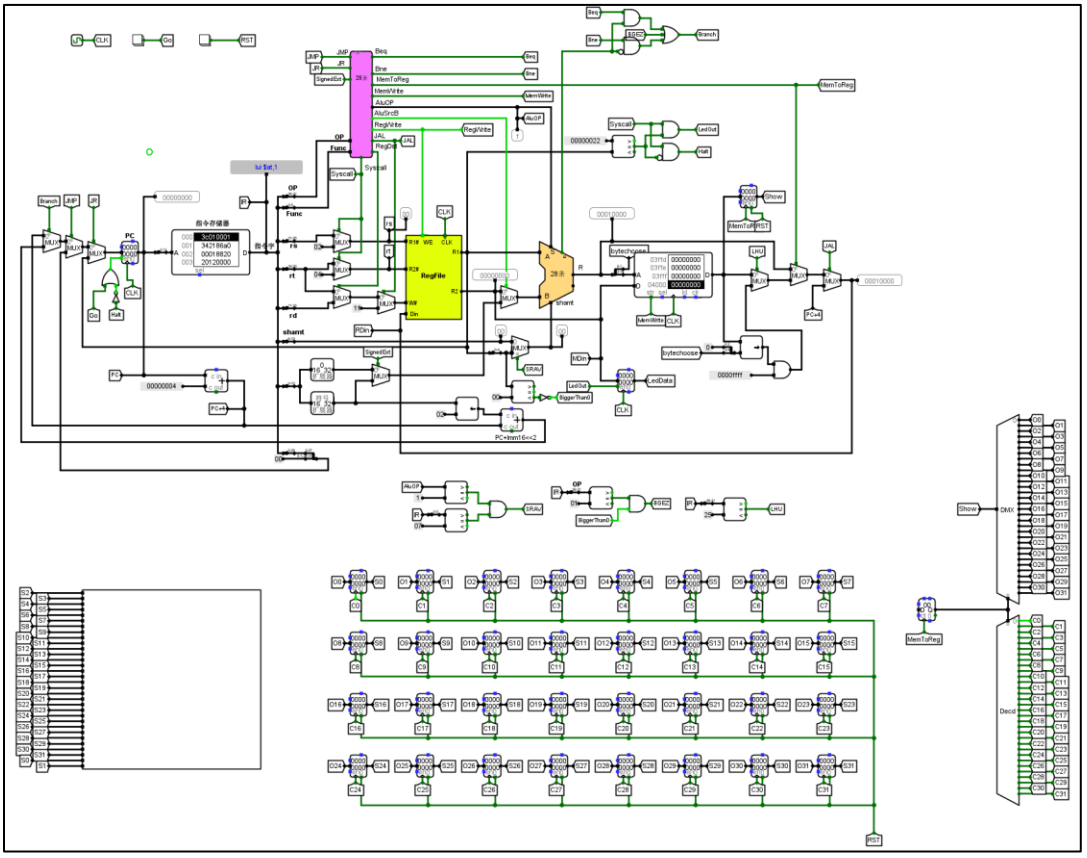

图 5-3 视频显示器数据通路

6 设计总结与心得

6.1 课设总结

在这次为期两周的课设中。我总共做了如下几点工作：

1） 实现了单周期 MIPS CPU，可以成功运行 24 条基础指令和 4 条个人指令；

2） 实现了单周期 CPU 加入单级中断功能；

3） 实现了理想流水线、气泡流水线、重定向流水线的逻辑电路；

4） 实现了重定向流水线中加入 4 条个人指令的功能；

5） 设计了团队任务内容；

6） 完成了团队任务中 logisim 连线和代码编写的部分；

7） 实现了使用 CPU 进行自定义的成果演示；

6.2 课设心得

本次课程设计长达两周，加上一些额外的时间以及团队任务的实施，总共耗时三周左右。我从这次课程设计中学到了很多，首先是 CPU 的整体设计和实现，让我重新理解了 CPU 的构造和实现原理。然后是流水线的理解和实现，遇到数据冲突的解决方法——气泡和重定向两种方式的实现，让我对 CPU 的指令系统的优化有了更深的了解。老师还为每个人分配了 4 条个人指令，保证了我们彻底理解指令的运行和实施过程，在反复调试和观察显示问题的时候对整个实验有更深入的了解。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计按照分组的机制，我认为团队任务过于自由化会导致组内同学的分歧变大。希望加权更高的同学倾向于花更多时间做一个更难的项目，决定考研/工作的同学却更希望做一个简单的、不花太多时间的展示，从而导致任务的确定和职能的分配会有一些争端。所以我建议老师们可以规定展示的课题，提供几个难度差不多的课题供队伍选择。

最后在这里也感谢三位老师对于我在本次课程设计中遇到问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 谭志虎，秦磊华，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021 年.

[4] 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：彭子晨