目　录

1　课程设计概述 4

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

2　总体方案设计 6

2.1　单周期 CPU 设计 6
2.2　中断机制设计 11
2.3　流水 CPU 设计 12
2.4　气泡式流水线设计 14
2.5　数据转发流水线设计 14

3　详细设计与实现 15

3.1　单周期 CPU 实现 15
3.2　中断机制实现 26
3.3　流水 CPU 实现 27
3.4　气泡式流水线实现 34
3.5　数据转发流水线实现 36

4　实验过程与调试 37

4.1　测试用例和功能测试 37
4.2　可自行安排章节 46
4.3　性能分析 46
4.4　主要故障与调试 46
4.5　实验进度 49

5　设计总结与心得 50

5.1　课设总结 50

批注 [T1]: 注意目录的格式，系统自动生成的格式和这个有差异，生成目录后按这个格式排版，不要三级目录。

修改章节后再这里右键点击更新域更新目录

5.2 课设心得..... 50
参考文献..... 52

1　课程设计概述

批注 [T2]: 章节形式，注意每章必须新起一页，具体方法是在上一章尾部增加一个分页符

1.1 课设目的

批注 [T3]: 二级标题，具体可以可以用格式刷复制

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	

批注 [TZH4]: 所有图标均需交叉引用

批注 [TZH5]: 须插入题注，注意在表格的上方，编号形式为 X.X，表格格式参照这个吧，这个还画的比较漂亮。此表只是个例子，主要说明在报告中的表格依照此种格式，而不是说这个部分必须用表格。

注意创建的编号格式 N.X

N 为章，X 本章表的序号，如表 2.1 表示是第二章的第一个表。表的标号及名称要比正文小一个字号。

#	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	XOR	异或	
29	SLTIU	小于立即数置 1(无符号)	
30	SB	存储字节	
31	BLEZ	小于等于 0 转移	

2　总体方案设计

批注 [U6]: 给出方案概要设计，注意这章是设计，不是最终实现，请不要将最终的连线图，代码等放在这一章

2.1 单周期 CPU 设计

本次我们采用的方案是工程化程序控制，且主、控存分开的方案，即采用微程序控制方式，实现主存储器（MM）和微程序控制存储器（CM）不共用一个存储器的方式完成方案的设计。同时在实施的过程中，采用部分电路用 FPGA 方式下载、部分电路用硬件搭建的方式完成……等等…。

总体结构图如图 2.1 所示。

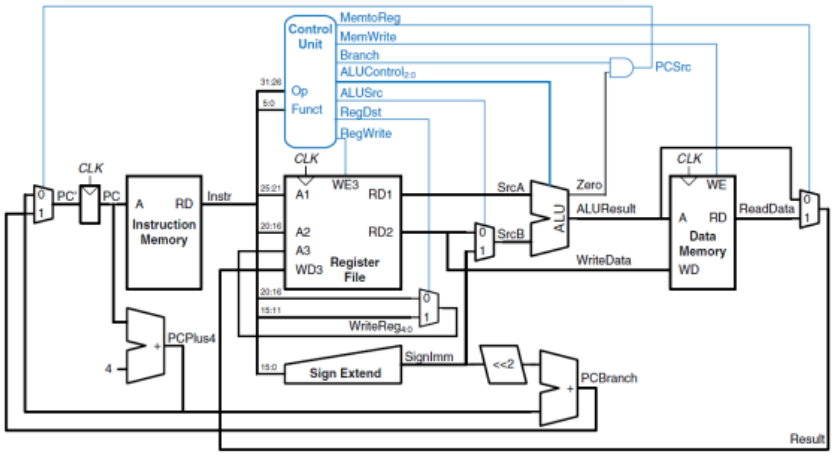

图 2.1　总体结构图

批注 [T7]: 图注，用插入题注的方式插入，插入后可用交叉引用引用，注意题注，交叉引用是 word 很重要的概念，希望大家认真找点资料看看这个功能，这是 office 排版的高级技巧，希望大家掌握，如果插入题注没有对应的图题注，只有 figure 等，可以自己创建，注意创建的编号格式 N.X

N 为章，X 本章表的序号，如图 3.3 表示是第三章的第 3 个图。图的标号及图名称比正文小一个字号。

批注 [T8]: 所有图都必须有图名

2.1.1 主要功能部件

运算器部分，具体设计思路如下

1.　　程序计数器 PC

思想是创建一个寄存器 pc，在寄存器 pc 里面存放 pc 的值，从总控里获取下一步的 pc 设为几的信号再具体操作。

2. 指令存储器 IM

输入是 pc，输出是响应的指令，存储器采用的是 256 个 32 位的字数组。

3. 运算器

表 2.1 算术逻辑运算单元引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

2.1.2 数据通路的设计

表 2.2 指令系统数据通路框架

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
Add	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x5			
Addi	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0x5			
Addiu	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0x5			
Addu	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x5			
And	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x7			
Andi	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x7			

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
Sll	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x0			
Sra	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x1			
Srl	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x2			
Sub	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x6			
Or	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x8			
Ori	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x8			
Nor	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0xa			
Lw	PC+1	PC	Rs		Rt	DM.out	RF.D1	s-ext	0x5	Alu		
Sw	PC+1	PC	Rs	Rt			RF.D1	s-ext	0x5	alu	RF.D2	
Beq	Ins16-32	PC	Rs	Rt			RF.D1	RF.D2				
Bne	Ins16-32	PC	Rs	Rt			RF.D1	RF.D2				
Slt	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0xb			
Slti	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0xb			
sltu	PC+1	PC	rs	Rt	Rd	ALU	RF.D1	RF.D2	0xc			
J	Ins26	PC										
Jal	Ins26	PC			31	PC+4						
Jr	RF.D1	PC	Rs									
Syscall	PC+1	PC	V0(2)	A0(4)			RF.D1	10				
Xor	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x9			
Sltiu	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0xb			
Sb	PC+1	PC	Rs		Rt	DM.out	RF.D1	Un-ext	0x5	alu		
blez	Ins16-32	PC	Rs	rt			RF.D1	RF.D2	0xb			
合并	4输入	PC	2输入	2输入	3输入	3输入	2输入	5输入		Alu	RF.D2	

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 2.3。

表 2. 3 主控制器控制信号的作用说明

控制信号	取值	说明
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
RegDst	1	写寄存器编号为 rt
	0	写寄存器编号为 rd
Syscall	1	读寄存器编号 1 为 2，读寄存器编号 2 为 4
	0	读寄存器编号 1 为 rs，读寄存器编号 2 为 rt
Jump	1	无条件跳转，npc 取 26 位立即数
	0	不跳转，npc 取其他值
Branch	1	有条件跳转成功，npc 选择 16 位立即数
	0	Npc 为其他值
Return	1	Jr 指令，函数调用返回，npc 为 31 号寄存器的值
	0	Npc 为其他值
Regdin	1	写入寄存器的值为数据存储器的输出
	0	写入寄存器的值为 alu 的输出
Shift	1	移位标志，立即数扩展器输出 5 位移位值的 32 位扩展结果
	0	立即数扩展器输出 16 位立即数的扩展结果
Unsigned	1	无符号扩展标记，16 位立即数做无符号扩展
	0	16 位立即数做有符号扩展
regW	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
aluB	1	Alu 操作数 Y 的选择，选择立即数扩展的结果
	0	选择寄存器文件的第二个输出

控制信号	取值	说明
DMWrite	1	数据存储器写使能信号，允许向 DM 写数据
	0	从 DM 读数据
Mode	00	对 DM 进行字访问
	01	对 DM 进行字节访问
	10	对 DM 进行半字访问
Aluop	0000-110	功能过多，详见表 2.6

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 2.4 所示。

表 2.4 主控制器控制信号框架

指令	b	j	regdst	regdin	alub	ALUop	sft	unsign	regw
Add									1
Addi			1		1				1
Addiu			1		1				1
Addu									1
And									1
Andi			1		1			1	1
Sll					1		1		1
Sra					1		1		1
Srl					1		1		1
Sub									1
Or									1
Ori			1		1				1
Nor									1
Lw			1	1	1				1
Sw			1		1				1
Beq	1								

指令	b	j	regdst	regdin	alub	ALUop	sft	unsign	regw
Bne	1								
Slt									1
Slti			1		1				1
sltu									1
J		1							
Jal		1							1
Jr									
Syscall									
Xori			1		1			1	1
Sltiu			1		1				1
Lbu			1	1	1			1	1
blez	1								

2.2 中断机制设计

2.2.1 总体设计

每一个中断源发出中断请求的时间是随机的，而中断要满足一定的条件才能被响应，也就是说有的中断可能需要经过较长的时间才能被响应。因此，当中断源的中断事件发生时，先将请求信号保存在中断请求触发器中。对应于每个中断源有一个中断请求触发器，多个中断请求触发器组合在一起，就构造了中断请求寄存器，其内容称为中断字或中断码。cpu 进行中断处理时，根据中断字确定中断源，然后转入相应的服务程序。

2.2.2 硬件设计

中断源的中断事件发生时，将中断信号传给 interrupt，先将请求信号保存在 D 触发器中，其后把中断处理程序的地址和 pc 值也传入，然后将 D 触发器中的信号传给另一个 D 触发器和 epc 寄存器的使能端，因此随着将 pc 值存到寄存器 epc 中，同时更新另一个 D 触发器的内容，并返回一个信号异步重置起初的 D 触发器状态为 0.，

并将信号传入多路选择器中，当多路选择器为 1 时输出的中断处理程序的地址，否则将输出 pc 值。当中断结束时，将取出寄存器 epc 中的 pc 值，进行输出。以此保证当中断结束时，cpu 返回原来的地址，继续执行中断发生前的指令。

2.2.3 软件设计

由于完成的是单级的中断，因此根本没有用到 MFC0 和 MTC0 指令，在多级中断中这两个指令起比较重要的作用，但是在单级中断中并没有使用到，因为单级中断只用到一个寄存器因此不用选择输入的寄存器，所以并没有使用这两条指令。为了从中断返回用到了 ERET 指令，并书写中断处理程序。

2.3 流水 CPU 设计

2.3.1 总体设计

一条指令的执行过程可以分为 5 个时钟周期，分别是取指令周期 IF，指令译码/读存储器周期 ID，执行/有效地址计算周期 EX，存储器访问/分支完成周期 MEM 和写回周期 WB，在这 5 个周期的实现方案中，分支指令和需要 4 个周期，其他指令需要 5 个周期才能完成，为了追求高性能，可以把分支指令的执行提前到 ID 周期完成，这样的话分支指令只需要 2 个周期就能完成。因此需要增加一个专门用于计算转移目标地址的加法器。

把上述的 5 个周期每一个周期作为一个流水段，并在各段上加上锁存器，就构成 5 段流水线，也就是加上流水寄存器。5 段流水线的大致结构如图 2.2 所示：

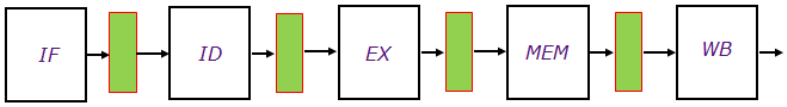

图 2.2 经典的 5 段流水线

在流水线方式下，要保证不会再同一时钟周期要求同一段功能段做两件不同的工作，其次，为了避免 IF 段的访存与 MEM 段的访存发生冲突，必须采用分离的指令存储器和数据存储器。ID 段要对通用寄存器组进行读操作，而 WB 段要对通用寄存器组进行写操作，为了解决对同一通用寄存器的访问冲突，要把写操作安排在时钟周期的前半拍完成，把读操作安排在时钟周期的后半拍完成。为了做到每一个周期启动一条指令，

必须在每个周期都进行 PC+的操作。这要在 IF 段完成，因此需要设置一个专门的加法器，另外，分支指令也要修改 PC 的值，它到 MEM 段才能进行修改。5 段流水线的时空图如图 3.4 所示：

表格 2.1 5 段流水线时空图

时钟 周期	1	2	3	4	5	6	7	8
指令 k	IF	ID	EX	MEM	WB			
指令 k+1		IF	ID	EX	MEM	WB		
指令 k+2			IF	ID	EX	MEM	WB	
指令 k+3				IF	ID	EX	MEM	WB

2.3.2 流水接口部件设计

将流水线分为 5 断流水线，分别是 IF，ID，EX，MEM，WB，在各个段之间设置缓冲接口部件，构建各阶段之间的接口部件，接口的定义将会简化并且不在同一段使用的信号不重名，流水线应向后续段传递数据信息，控制信息，反馈信息，后续部件对数据的加工处理依赖于前阶段传递过来的信息。

供参考的接口定义如下：

IF→IF/ID

数据 IR

ID→ID/EX

数据 IR WriteReg# A B

控制 ALUop、IMM、ALUSrc、 Beq，Bne，jmp

MemReq MemWrite RegWrite MemRead

EX→EX/MEM

数据 IR WriteReg# AluResult Rt

控制 MemReq MemWrite RegWrite MemRead

MEM→MEM/WB

数据 WriteReg# WriteBackData

控制 RegWrite

PC，IR 需要全部传递，这样便于流水线的调试。

2.3.3 理想流水线设计

基于已经搭建好的单周期 CPU，在此基础上搭建理想流水线，理想流水线的特点是所有对象均通过同样的阶段，不同阶段无共享资源，各段的传输延迟一致，进入流水线的对象不受其他阶段的影响，指令处理的相关性大，各段之间的接口应该传递待加工的数据，控制数据传递的控制信号和反馈信号，否则后端无法对数据进行处理。

2.4 气泡式流水线设计

创建数据冲突和结构冲突检测模块，检测出冲突时在 if/ex 段给寄存器输入 0，并锁住 id/if 段从而产生生气泡。

2.5 数据转发流水线设计

在 ex 段设计数据冲突检测，如果遇到数据冲突就从 mem 或者 wb 段取回数据进行运算。

3 详细设计与实现

批注 [U9]: 撰写具体的实现细节，流程图，原理图，引脚连接等等，微程序实现，微指令实现，控存安排等等细节，上一章主要讲方案，这章描述最终实现细节。

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

① Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Halt 为停机信号，将此控制信号通过非门取反之后和时钟相与，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，与时钟信号相与，屏蔽时钟信号，使整个电路停机。如图 3.1 所示。

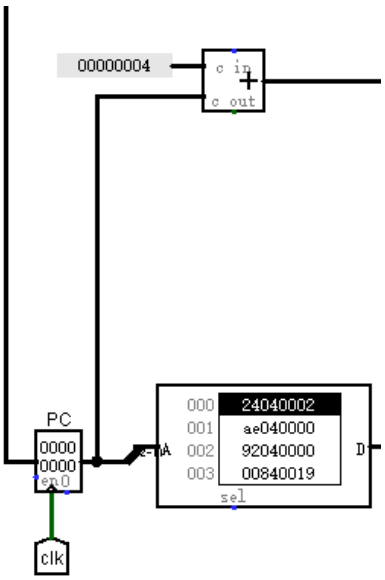

图 3.1 程序计数器（PC）

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

批注 [TZH10]: 代码格式，请按下面的来，灰色底纹

```
module pc(
clk,rst,halt,pcout,pc,pc4
```

```
    );
    input clk,rst,halt;
    input [31:0] pcout;
    output reg[31:0] pc;
    output [31:0] pc4;
    assign pc4 = pc + 4;
    initial begin
        pc <= 0;
    end
    always@(posedge clk) begin
        if(0==rst) begin
            pc <= 0;
        end
        else begin
            if(halt) begin
                pc <= pcout;
            end
            //低电平停机
            else begin
                pc <= pc;
            end
        end
    end
endmodule
```

指令存储器（IM）

① Logism 实现：

使用一个只读存储器 ROM 实现指令存储器（IM）。设置该只读存储器的地址位宽为 10 位，数据位宽为 32 位。因为 PC 中存储的指令地址有 32 位，而 ROM 地址线宽度有限，仅为 10 位，故将 32 位指令地址高位部分和字节偏移部分直接屏蔽，使用分线器只取 32 位指令地址的 2-11 位作为指令存储器的输入地址。如图 3.2 所示。

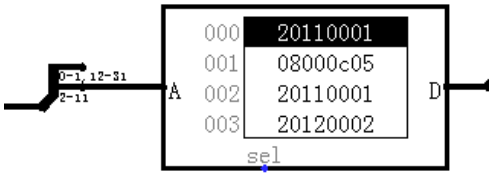

图 3.2 指令存储器（IM）

② FPGA 实现：

直接使用 Vivado 中自带的 ROM 作为指令存储器，其设置如**错误!未找到引用源。**所示。选择 ROM 的数据位宽为 32 位，因为该 ROM 的地址位宽为 10 位，所以选择 ROM 的大小选择为 1024。

指令存储器 IM 的 Verilog 代码如下：

```
module IM(pc,data);
    input [31:0] pc;
    output [31:0] data;

    reg [31:0] memory [0:255];
    wire [7:0] pc1;

    initial begin
        $readmemh("D:/benchmark.hex",memory);
    end

    assign pc1[7:0]=pc[9:2];
    assign data[31:0]=memory[pc1][31:0];
endmodule
```

直接调用之前设置的 ROM 作为指令存储器，输入为指令地址的 2-11 位，输出为该指令。

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+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x5			
Addi	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0x5			
Addiu	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0x5			
Addu	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x5			
And	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x7			
Andi	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x7			
Sll	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x0			
Sra	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x1			
Srl	PC+1	PC		Rt	Rd	ALU	RF.D2	Shamt	0x2			
Sub	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x6			
Or	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0x8			
Ori	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x8			
Nor	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0xa			
Lw	PC+1	PC	Rs		Rt	DM.out	RF.D1	s-ext	0x5	Alu		
Sw	PC+1	PC	Rs	Rt			RF.D1	s-ext	0x5	alu	RF.D2	

指令	PC	IM	RF				ALU			DM		Tube
			R1#	R2#	W#	Din	A	B	OP	Addr	Din	
Beq	Ins16-32	PC	Rs	Rt			RF.D1	RF.D2				
Bne	Ins16-32	PC	Rs	Rt			RF.D1	RF.D2				
Slt	PC+1	PC	Rs	Rt	Rd	ALU	RF.D1	RF.D2	0xb			
Slti	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0xb			
sltu	PC+1	PC	rs	Rt	Rd	ALU	RF.D1	RF.D2	0xc			
J	Ins26	PC										
Jal	Ins26	PC			31	PC+4						
Jr	RF.D1	PC	Rs									
Syscall	PC+1	PC	V0(2)	A0(4)			RF.D1	10				
Xori	PC+1	PC	Rs		Rt	ALU	RF.D1	Un-ext	0x9			
Sltiu	PC+1	PC	Rs		Rt	ALU	RF.D1	s-ext	0xb			
Lbu	PC+1	PC	Rs		Rt	DM.out	RF.D1	Un-ext	0x5	alu		
blez	Ins16-32	PC	Rs	rt			RF.D1	RF.D2	0xb			
合并	4 输入	PC	2 输入	2 输入	3 输入	3 输入	2 输入	5 输入		Alu	RF.D2	

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，表，将各个主要功能部件进行连接，根据数据通路合并表的最终结果，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建。

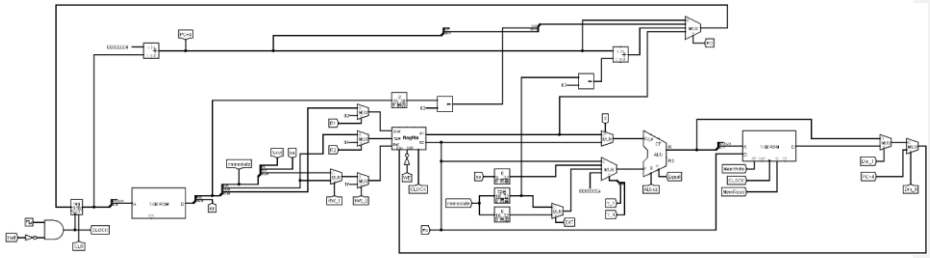

图 3.3 单周期 CPU 数据通路（Logism）

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

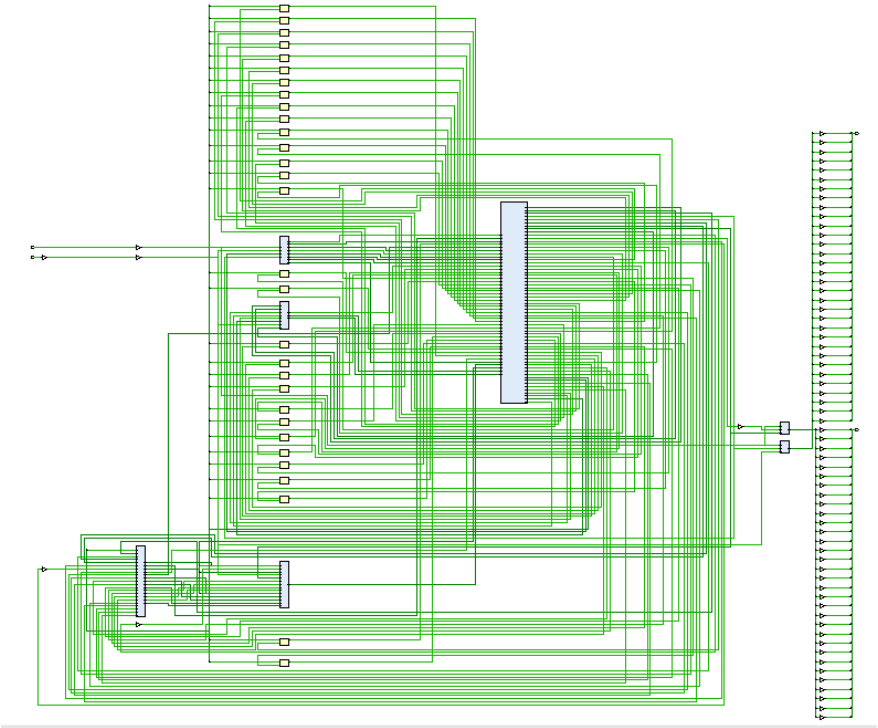

图 3.4 单周期 CPU 数据通路（FPGA）

3.1.3 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，分别在 Logism 和 Vivado 上进行主控制器、Branch 控制器、SYSCALL 控制器的具体实现。

主控制器

对照表 3.2 所示。

表 3.2 主控制器控制信号

指令	b	j	regdst	regdin	alub	ALUop	sft	unsign	regw
Add									1
Addi			1		1				1
Addiu			1		1				1
Addu									1
And									1
Andi			1		1			1	1
Sll					1		1		1
Sra					1		1		1
Srl					1		1		1
Sub									1
Or									1
Ori			1		1				1
Nor									1
Lw			1	1	1				1
Sw			1		1				1
Beq	1								
Bne	1								
Slt									1
Slti			1		1				1
sltu									1

指令	b	j	regdst	regdin	alub	ALUop	sft	unsign	regw
J		1							
Jal		1							1
Jr									
Syscall									
Xori			1		1			1	1
Sltiu			1		1				1
Lbu			1	1	1			1	1
blez	1								

部生成。

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：

```
module controller(
    op, func, alur,equal,
    jump,branch,ret,RegDst1,jal,RegDin1,
    shift,ALUOP,syscall,RegWrite,sw,
    mode,bat,AluB1
    );
    input [5:0] op; //op 字段
    input [5:0] func;//func 字段
    input [31:0] alur;   //alu 的运算结果
    input equal;            //alu 是否相等的判断输出
    output   jump;      //无条件分支
    output   branch;   //有条件分支成功
    output   bat; //有条件分支尝试
    output   ret; //return  返回
```

```
output    RegDst1; //
output    jal; //函数调用 jal
output    RegDin1; //
output    shift;     //位移
output    [3:0]ALUOP;     //alu 的操作码
output    syscall; //停机
output    RegWrite;    //写寄存器
output    sw;          //DMWrite  写数据存储器
output    [3:0]mode;       //数据存储器访问模式
output    AluB1;    //alu 的 b 输出的选择信号
wire r;
wire addi;
wire addiu;
wire andi;
wire ori;
wire slti;
wire xori;
wire sltiu;
wire lw;
wire lbu;
wire beq;
wire bne;
wire blez;
wire j;
wire add;
wire addu;
wire And;
wire sll;
wire sra;
wire srl;
```

```
wire sub;
wire Or;
wire Nor;
wire slt;
wire sltu;
wire tmpsys,tmpbr;
assign r = (op==6'h00);          //8
assign addi = (op==6'h08);
assign addiu = (op==6'h09);
assign andi = (op==6'h0c);
assign ori = (op==6'h0d);
assign slti = (op==6'h0a);
assign xori = (op==6'h0e);
assign sltiu = (op==6'h0b);
assign lw = (op==6'h23);
assign sw = (op==6'h2b);
assign lbu = (op==6'h24);
assign beq = (op==6'h04);
assign bne = (op==6'h05);
assign blez = (op==6'h06);
assign j = (op==6'h02);
assign jal = (op==6'h03);
assign add = (func==6'h20)& r;
assign addu = (func==6'h21)& r;
assign And = (func==6'h24)& r;
assign sll = (func==6'h00)& r;
assign sra = (func==6'h03)& r;
assign srl = (func==6'h02)& r;
assign sub = (func==6'h22)& r;
assign Or = (func==6'h25)& r;
```

```
assign Nor = (func==6'h27)& r;
assign slt = (func==6'h2a)& r;
assign sltu = (func==6'h2b)& r;
assign ret = (func==6'h08)& r;
assign syscall = (func==6'h0c)& r;
assign tmpsys = r & (~(ret | syscall));
assign tmpbr = equal | (alur==32'h00000001);

assign jump = j|jal;
assign bat = beq|bne|blez;
assign RegDst1 = addi|addiu|andi|ori|slti|xori|sltiu|lw|lbu;
assign RegDin1 = lw|lbu;
assign AluB1 = addi|addiu|andi|ori|slti|xori|sll|sltiu|lw|sra|sw|lbu|srl;
assign RegWrite = tmpsys|addi|addiu|andi|ori|slti|xori|sltiu|jal|lw|lbu;
assign branch = beq & equal | (~equal)& bne | tmpbr & blez;
assign shift = sll | sra | srl;

assign mode = lbu ? 4'b0011:
      (sw | lw)? 4'b1111: 4'b0000;
assign ALUOP = sll ? 4'h00:
                     sra ? 4'h01:
                     srl ? 4'h02:
                     (add|addi|addiu|addu|lw|lbu|sw)? 4'h05:
                     sub? 4'h06:
                     (And|andi)? 4'h07:
                     (Or|ori)? 4'h08:
                     xori? 4'h09:
                     Nor? 4'h0a:
                     (slt|slti|blez)? 4'h0b:
                     (sltu|sltiu)? 4'h0c:4'h00;//最后一种情况？？
```

endmodule

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。在 Vivado 中使用 Verilog 语言构成的主控制器原理图如图 3.5 所示。

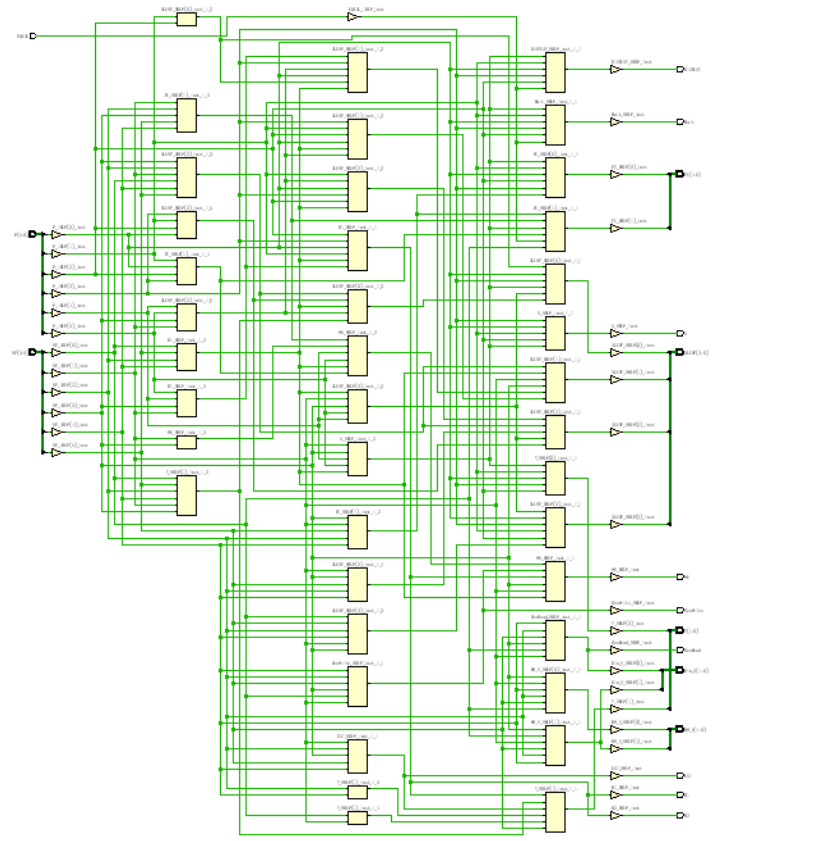

图 3.5 主控制器原理图

3.2 中断机制实现

3.2.1 原理

中断源的中断事件发生时，将中断信号传给 interrupt，先将请求信号保存在一号 D 触发器中，其后把中断处理程序的地址给 addr，还有把 pc 值给 pc_in，然后将一号 D 触发器中的信号传给二号 D 触发器和 epc 寄存器的使能端，因此随着时钟 clk 的更

新把 pc 值存到寄存器 epc 中，同时二号 D 触发器随着 clk 更新触发器的内容，并返回一个信号异步重置一号触发器状态为 0.，其次将信号传入多路选择器中，当多路选择器为 1 是 pc_out 输出的中断处理程序的地址，否则将输出传入的 pc 值。当中断结束时，将取出寄存器 epc 中的 pc 值，进行输出。以此保证当中断结束时，cpu 返回原来的地址，继续执行中断发生前的指令。结构如图 3.10 所示：

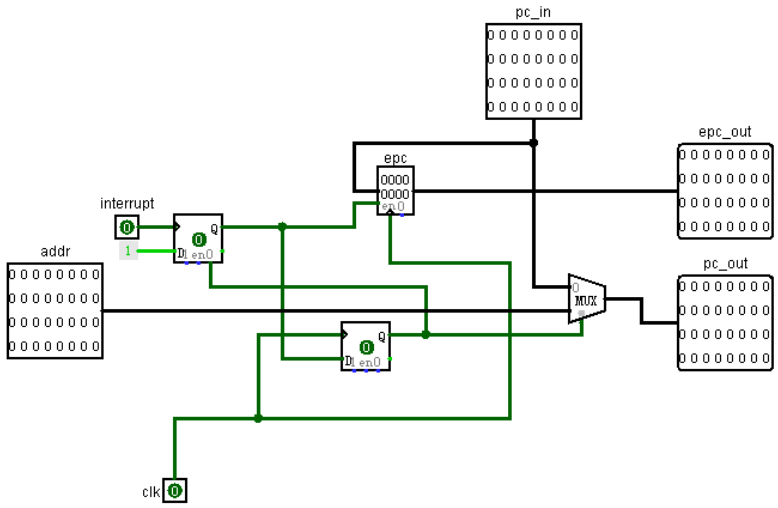

图 3.6 单级中断

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

1. 分别设计 5 段中的四个流水线寄存器模块，名称为：IF/ID，ID/EX，EX/MEM，MEM/WB。

（1） IF/ID 中包含两个数据位宽为 32 位的寄存器，pc_in 端，rom_in 端分别连接输入和 pc_out 端，rom_out 端分别连接输出，数据位宽均为 32 位，使能端连接时钟 clk，清零端连接清零 0，触发方式为上升沿。

IF/ID 模块示意图如图 3.11 所示：

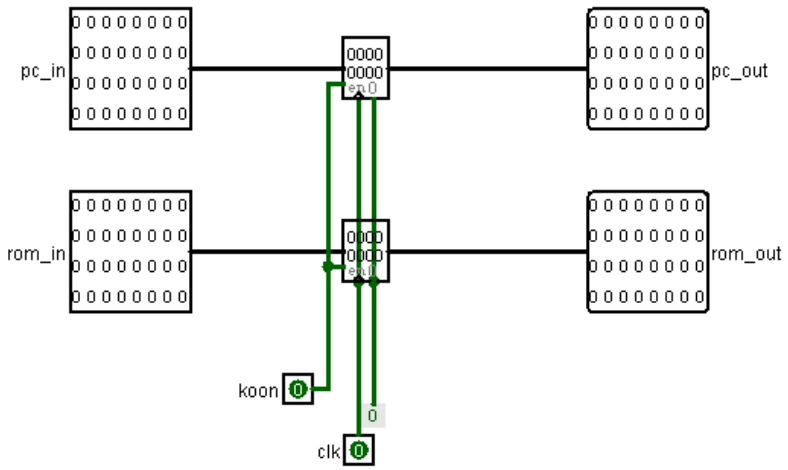

图 3.7 IF/ID 寄存器模块示意图

（2） ID/EX 段包含数据位宽为 1 位的 12 个输入输出端的寄存器（bgzt，memwrite，branch,，memtoreg，jal，bne，memread，alusrc，jump，beq，lbu，jr，sys，mflo，multu，shift，regwrite，eret_is），6 个数据位宽为 32 的寄存器（ j_in，pc_in，R1，R2，zero，shamt），1 个数据位宽为 4 的寄存器（funct），1 个数据位宽为 5 的寄存器（rt /rd）用于写回，使能端连接时钟 clk，清零端连接清零 0，触发方式为上升沿。

ID/EX 模块示意图如图 3.12 和图 3.13，图 3.14 所示：

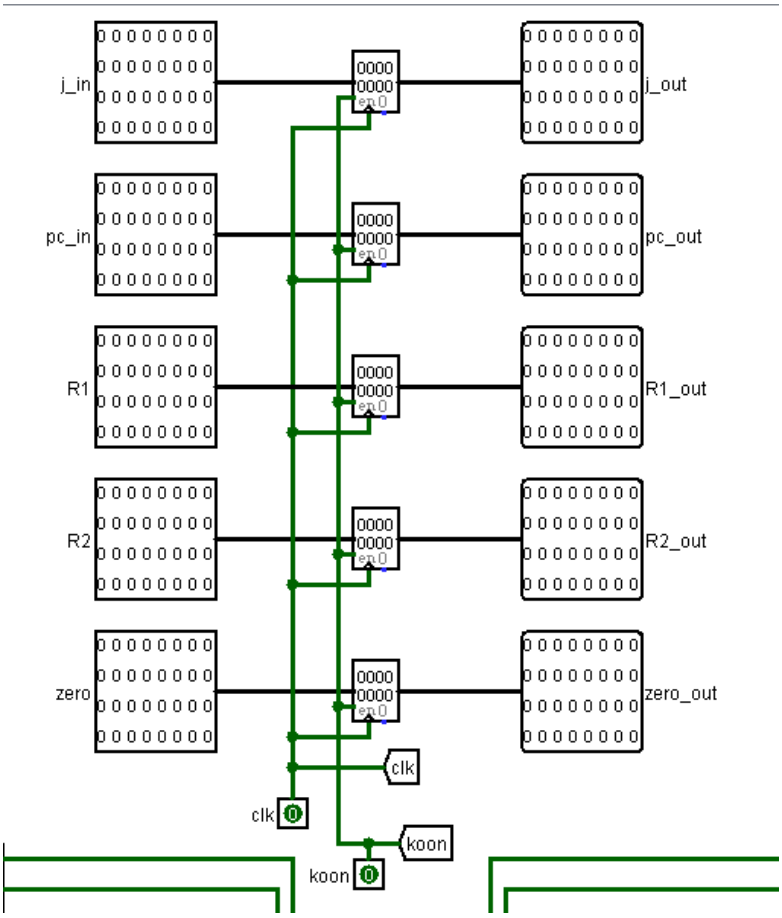

图 3.8 ID/EX 寄存器模块示意图

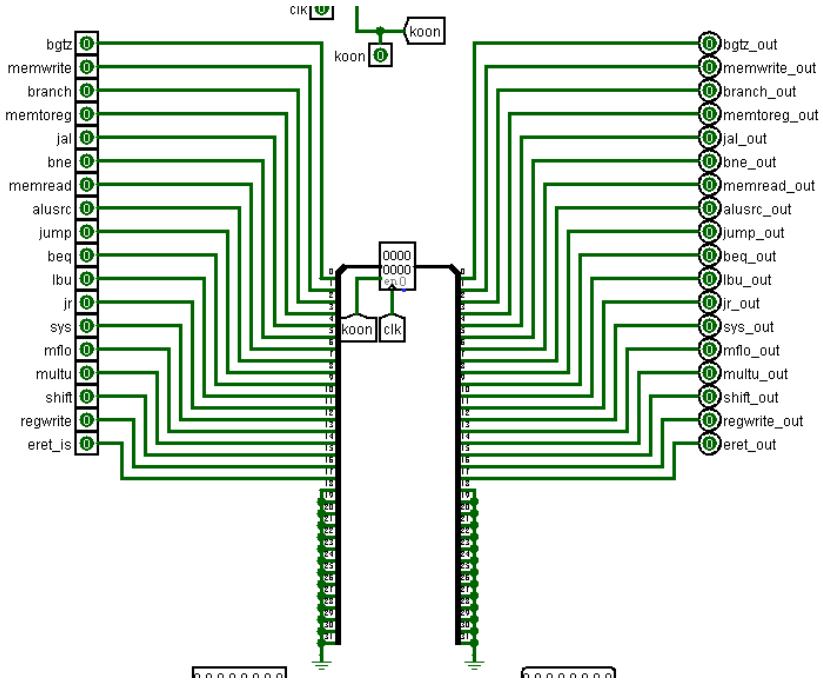

图 3.9 ID/EX 寄存器模块示意图

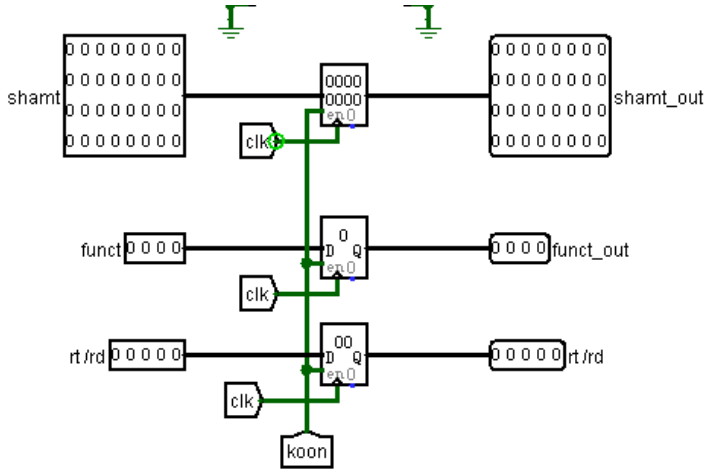

图 3.10 ID/EX 寄存器模块示意图

（3） EX/MEM 段包含数据位宽为 1 位的 11 个输入输出端的寄存器（memwrite_out，memtoreg_out，jal_out，memread_out，lbu_out，mflo_out，multu_out，regwrite_out，eret_out，jump_out，koon_out ）1 个数据位宽为 4 的寄存器（写回地址），3 个数据位宽为 32 位的寄存器（pc_in，R_in，R2_in），使能端连接时钟 clk，清零端连接清零 0，触发方式为上升沿。EX/MEM 模块示意图如图 3.15 和图 3.16 所示：

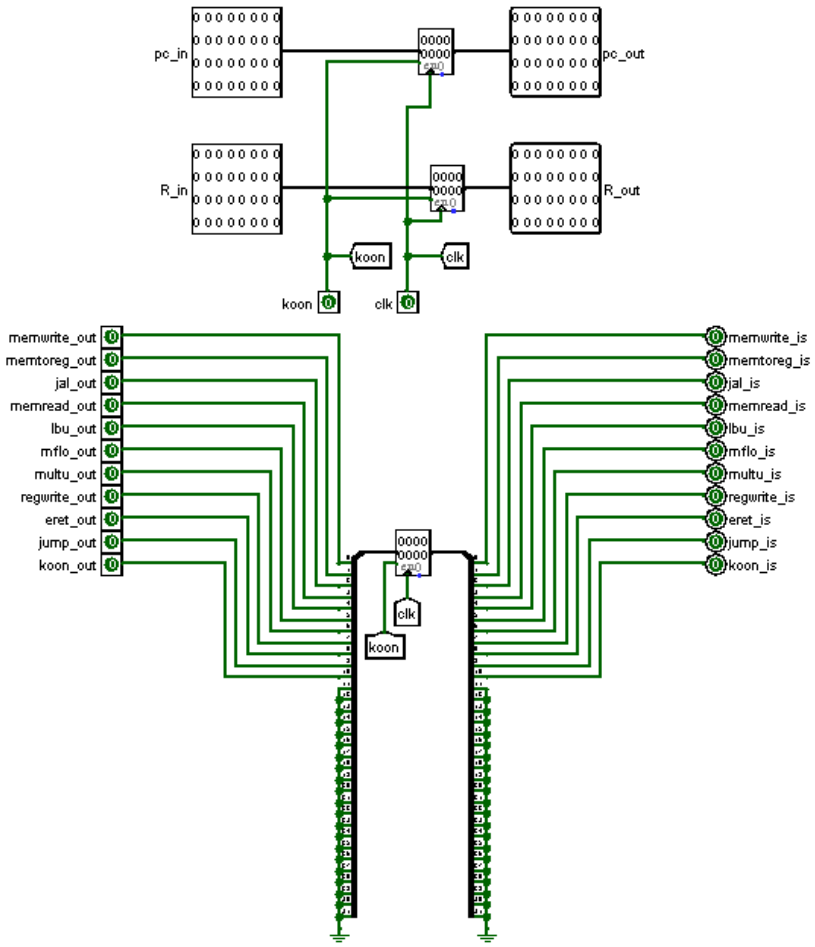

图 3.11 EX/MEM 寄存器模块示意图

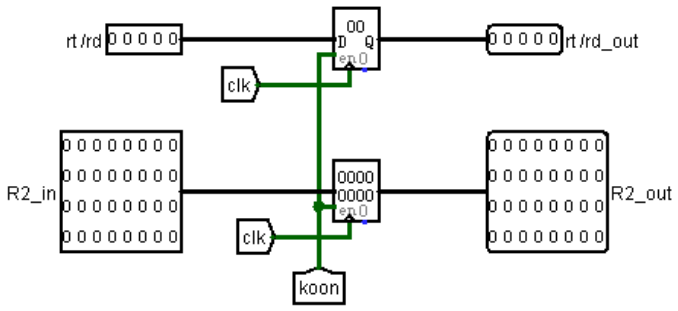

图 3.12　EX/MEM 寄存器模块示意图

（4）　MEM/WB 段包数据位宽为 1 位的 5 个输入输出端的寄存器（jal_is，mflo_is，multu_is，regwrite_is，koon_is ）1 个数据位宽为 4 的寄存器（写回地址），2 个数据位宽为 32 位的寄存器（pc_in，D_in），使能端连接时钟 clk，清零端连接清零 0，触发方式为上升沿。EX/MEM 模块示

MEM/WB 模块示意图如图 3.17 所示：

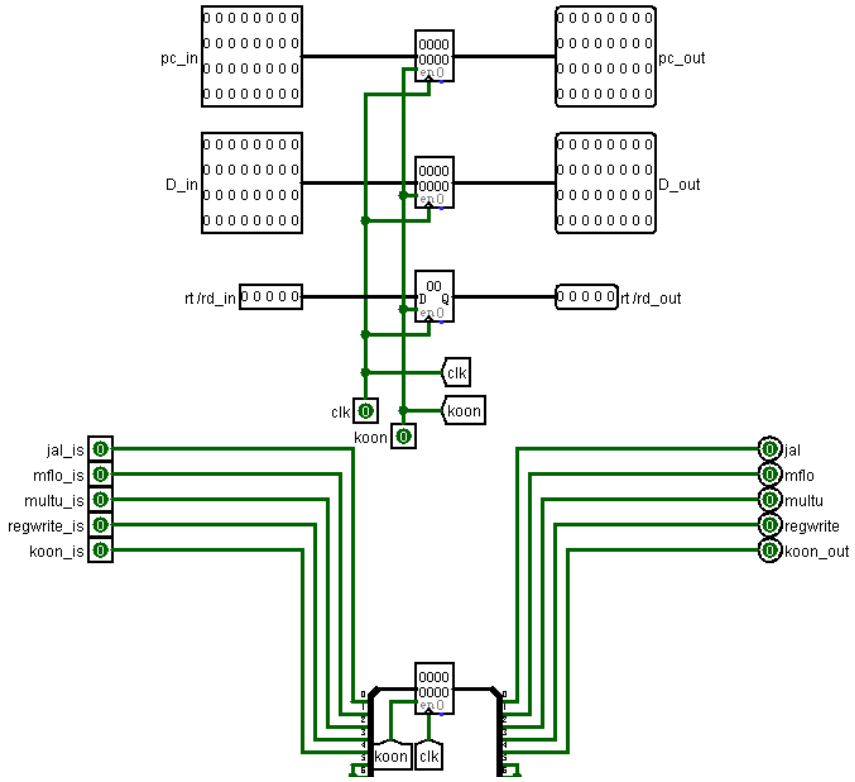

图 3.13　MEM/WB 寄存器模块示意图

3.3.2　理想流水线实现

基于已经搭建好的单周期 CPU，在此基础上搭建理想流水线，理想流水线的特点是所有对象均通过同样的阶段，不同阶段无共享资源，各段的传输延迟一致，进入流水线的对象不受其他阶段的影响，指令处理的相关性大，各段之间的接口应该传递待加工的数据，控制数据传递的控制信号和反馈信号，否则后端无法对数据进行处理。

将流水线分为 5 断流水线，分别是 IF，ID，EX，MEM，WB，在各个段之间设置缓冲接口部件，构建各阶段之间的接口部件，接口的定义将会简化并且不在同一段使用的信号不重名，流水线应向后续段传递数据信息，控制信息，反馈信息，后续部件对数据的加工处理依赖于前阶段传递过来的信息，因而设计出的理想流水线的示意图见图 3.18 和 3.19 所示：

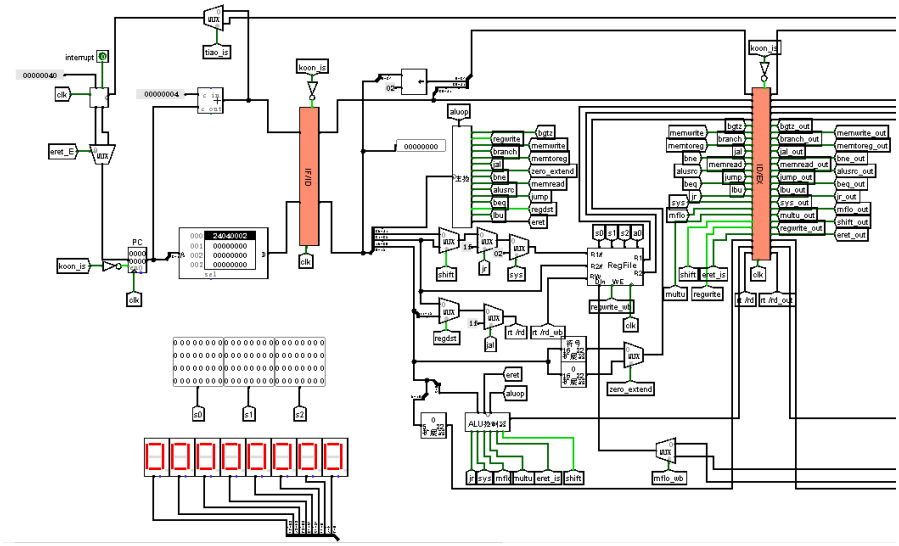

图 3.14 理想流水线

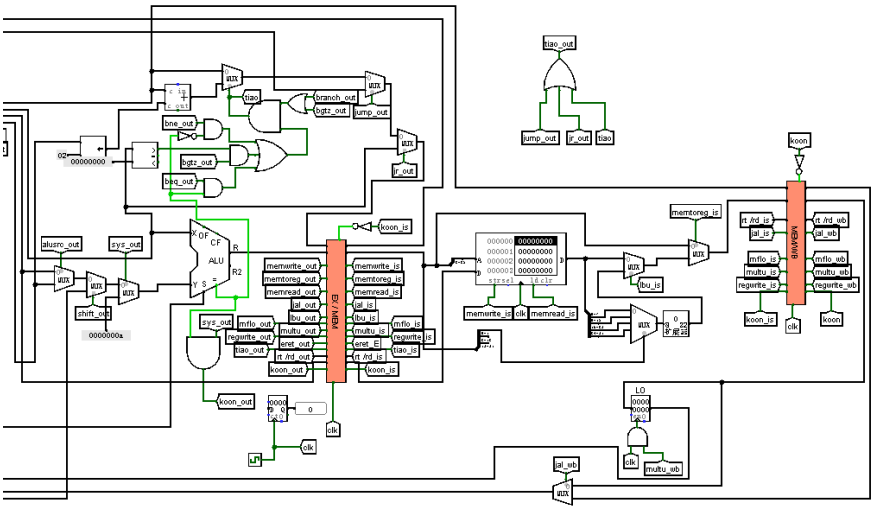

图 3.15 理想流水线

3.4 气泡式流水线实现

创建数据冲突和结构冲突检测模块，检测出冲突时在 if/ex 段给寄存器输入 0，并

锁住 id/if 段从而产生生气泡。如下为气泡检测装置结构图：

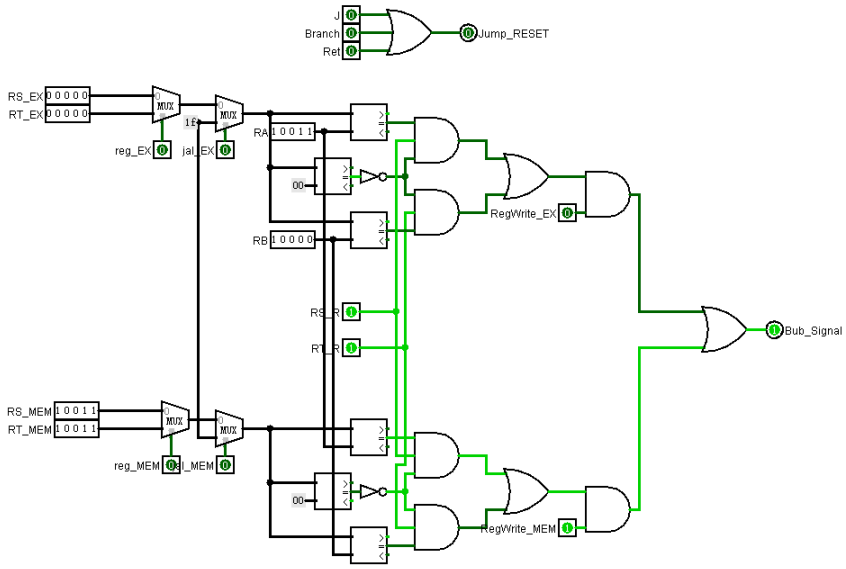

图 3.20 气泡检测装置

如下为总结构图：

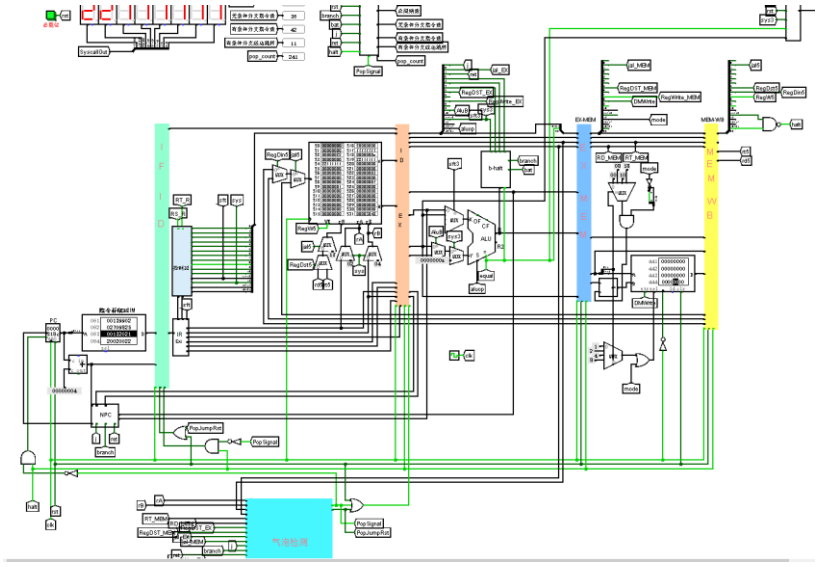

图 3.21 气泡流水 cpu 结构图

3.5 数据转发流水线实现

在 ex 段设计数据冲突检测，如果遇到数据冲突就从 mem 或者 wb 段取回数据进行运算。如下为数据冲突检测装置结构图：

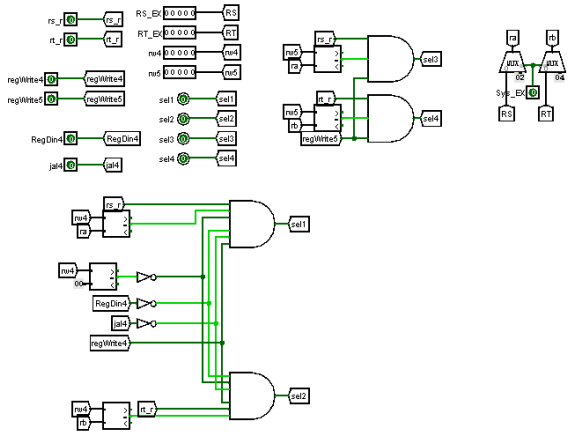

图 3. 20 数据冲突检测装置

如下为总结构图：

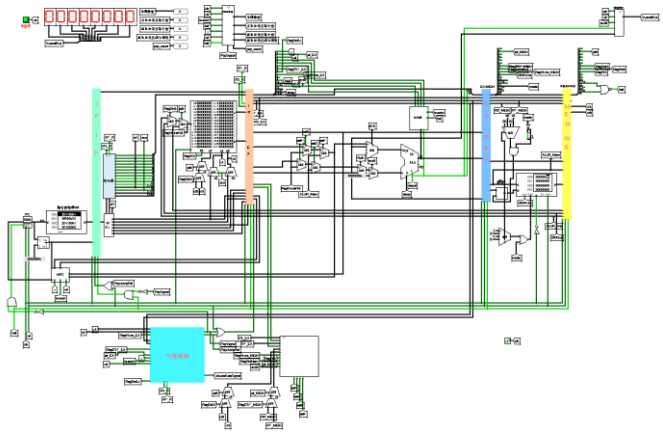

图 3. 21 气泡流水 cpu 结构图

4　实验过程与调试

4.1 测试用例和功能测试

4.1.1 测试用例 1

原有的 24 条指令的基础再填加 7 条新指令，从而进行指令的扩展。因此需要对新增加的指令进行指令的测试，此时 RAM 加载 benchmark 测试程序，观察结果。

```
.text
begin:
li      $a0, 2
sw      $a0,0($s0)
lbu     $a0,0($s0)
multu   $a0, $a0
mflo    $a0
bgtz    $a0, begin
```

图 4.1 benchmark 测试程序

（1）ROM 加载 LBU 指令的测试，这条指令的作用是加载字节(无符号)。

加载 LBU 指令执行前后的测试结果见如图 4.2 和图 4.3 所示：

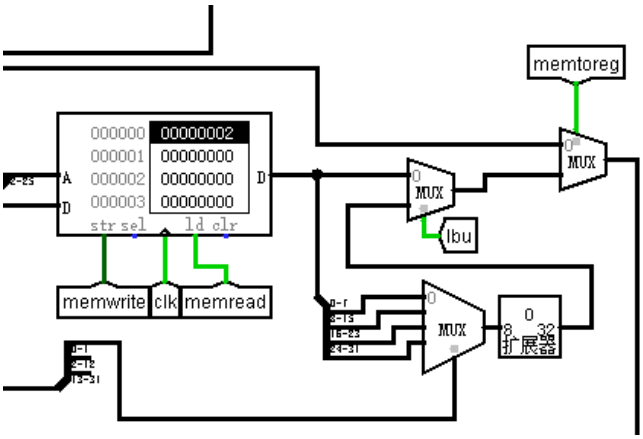

图 4.2 执行指令 LBU 前

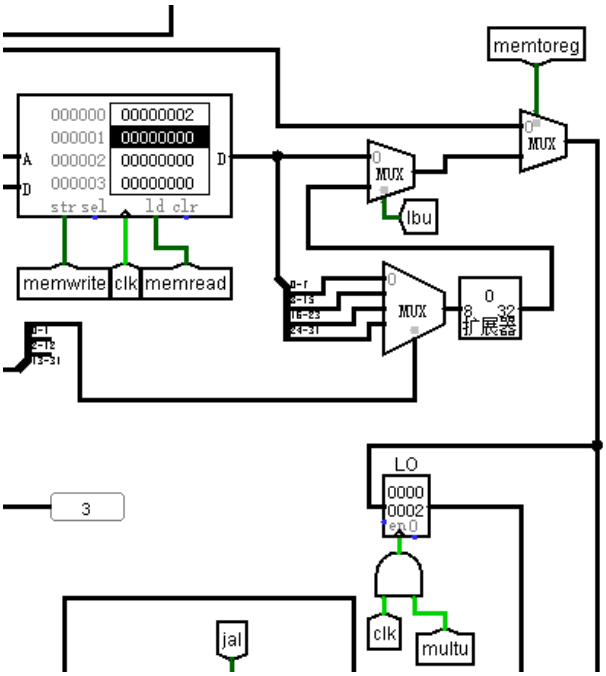

图 4.3 执行指令 LBU 后

（2）ROM 加载 MULTU 指令的测试，这条指令的作用是乘无符号。

加载 MULTU 指令测试后显示如图 4.4 所示：

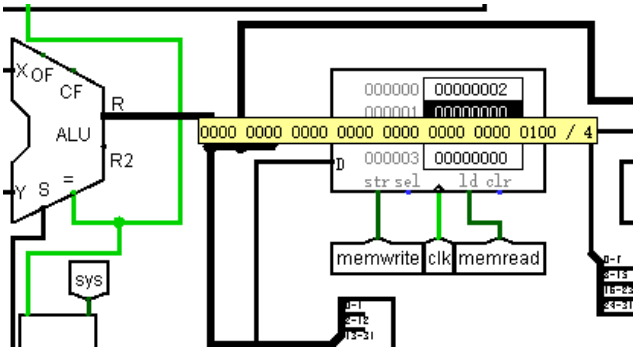

图 4.4 指令 MULTU

（3）ROM 加载 MFLO 指令的测试，这条指令的作用是读 LO 寄存器。

加载 MFLO 指令测试后显示如图 4.5 和 4.6 所示：

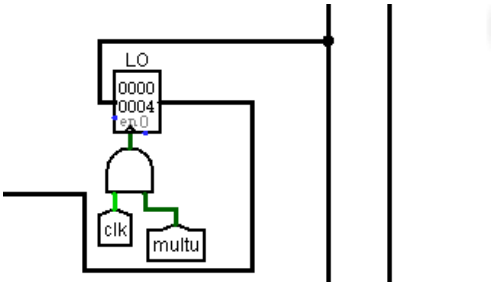

图 4.5 指令 MFLO

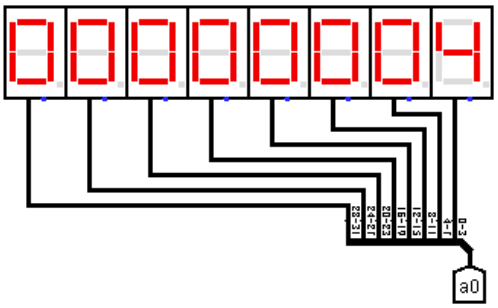

图 4.6 执行指令 MFLO

（4）ROM 加载 BGTZ 指令的测试，这条指令的作用是大于 0 转移。

加载 BGTZ 指令测试后显示如图 4.7 所示：

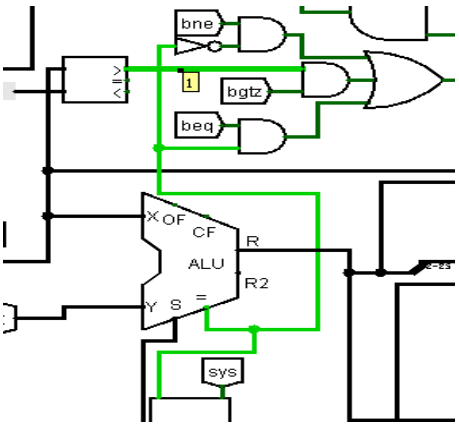

图 4.7 指令 BGTZ

4.1.2 单周期 cpu 上班测试

跑标准 benchmark 结果如下

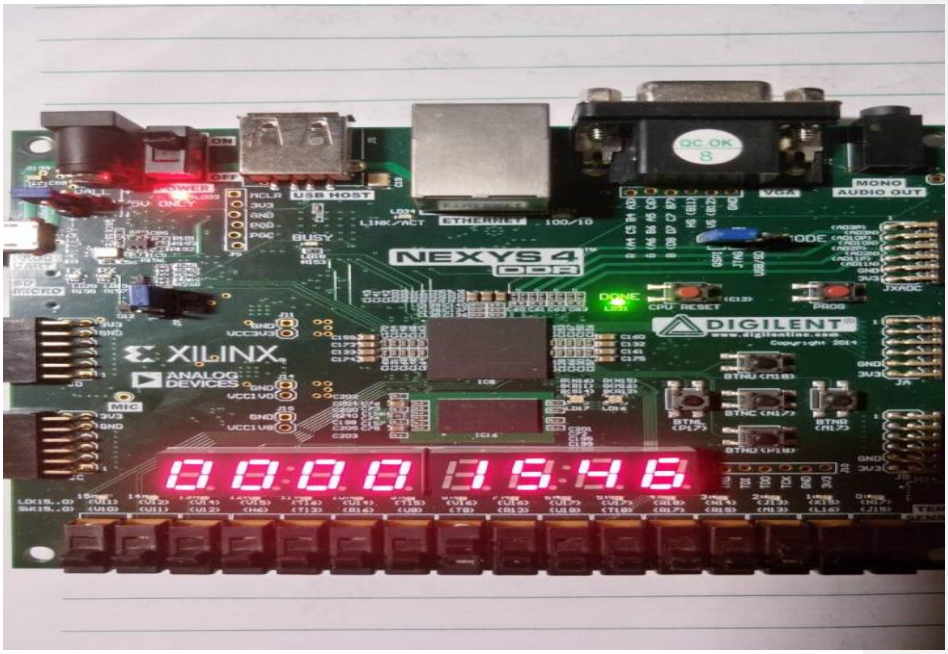

图 4.8 总周期数

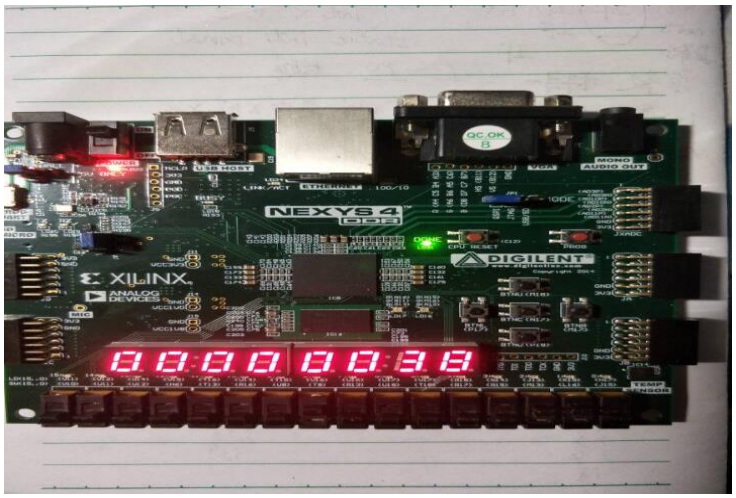

图 4.9 无条件转移数

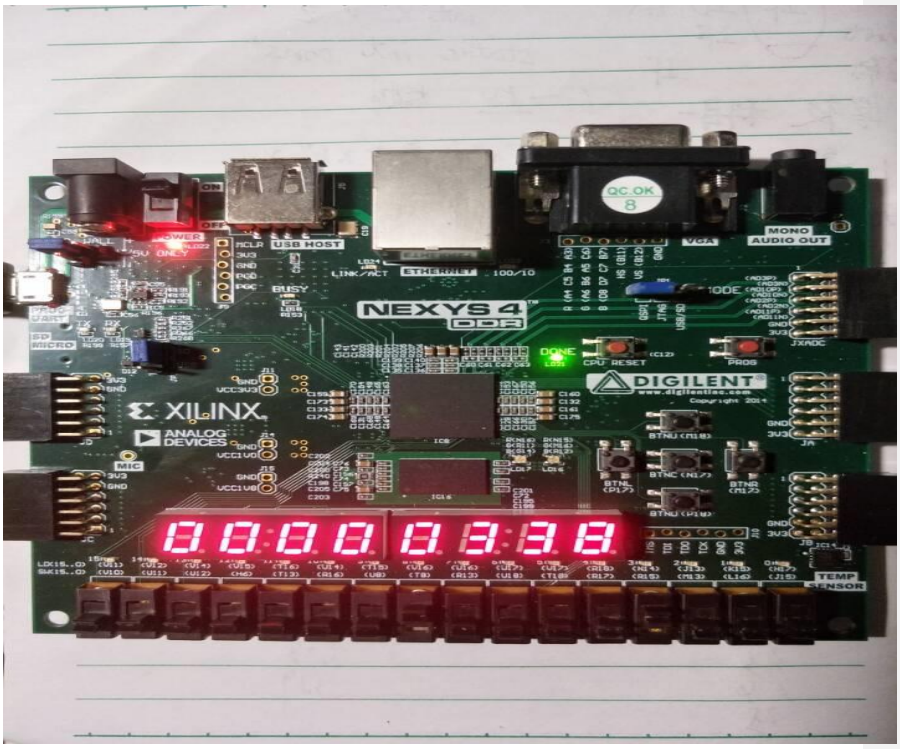

图 4.10 有条件转移数

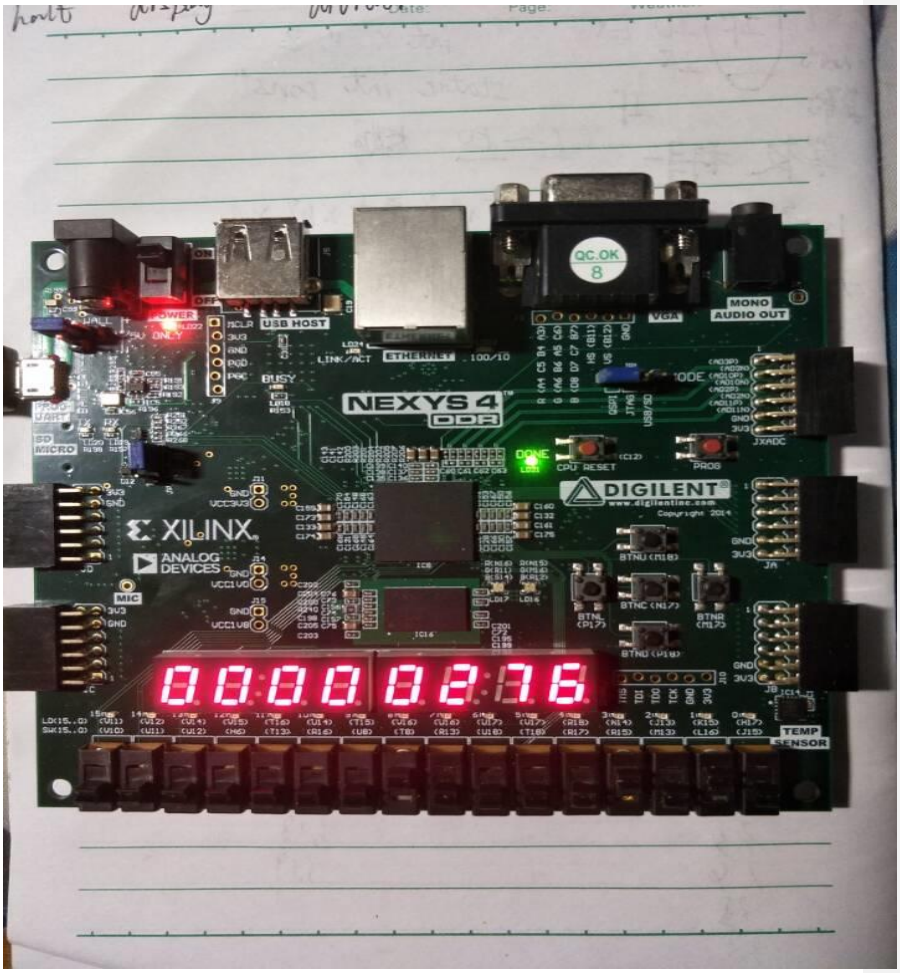

图 4.11 有条件转移成功

4.1.3 测试用例 2

每一个中断源发出中断请求的时间是随机的，而中断要满足一定的条件才能被响应，也就是说有的中断可能需要经过较长的时间才能被响应。因此利用中断的特点，我们书写中断测试程序，此测试程序将进行累加和，当发生中断时程序将会从 9 减到 0 并回到发生中断的地址，继续进行累加和，为了证明结果没有出错因此将在 RAM 中加载 interrupt 测试程序，观察结果。

```
# if correct, display will show
# main: from 0x0 to 0xffff
# interrupt: from 0x9 to 0x0
# interrupt addr: 0x10 (need mul 4 in PC)

addi    $s1, $zero, 0x0
addi    $s2, $zero, 0xffff

MAIN_LOOP:
addi    $s1, $s1, 1

add     $a0, $0, $s1        # display $s1
addi    $v0, $0, 34         # display hex
syscall

bne     $s1, $s2, MAIN_LOOP

addi    $v0, $zero, 10      # system call for exit
syscall

nop
nop
nop
nop
nop

INTERRUPT:
sw      $s1,0
sw      $s2,4
addi    $s1, $zero, 0x9
```

```
addi      $s2, $zero, 0x0
INTERRUPT_LOOP:
addi      $s1, $s1, -1

add       $a0, $0, $s1          # display $s1
addi      $v0, $0, 34           # display hex
syscall

bne       $s1, $s2, INTERRUPT_LOOP
lw        $s1,0
lw        $s2,4
eret
```

注：因运行过程动态显示，因此不在这儿给出截图。

4.1.4 测试用例 3

理想流水线的特点是所有对象均通过同样的阶段，不同阶段无共享资源，各段的传输延迟一致。为了检测 4 条拓展指令和理想流水线，在 RAM 加载理想流水线测试程序。测试结果见图 4.8：

```
#理想流水线测试
li      $a0, 2
nop
nop
nop
nop
multu   $a0, $a0
nop
nop
nop
nop
sw        $a0,0($s0)
```

```
nop
nop
nop
nop
lbu         $a0,0($s0)
nop
nop
nop
nop
mflo    $a0
nop
nop
nop
nop
bgtz    $a0, begin
nop
nop
nop
nop
begin:
addi $s0,$zero, 0
addi $s1,$zero, 0
addi $s2,$zero, 0
addi $s3,$zero, 0
ori $s0,$s0, 0
ori $s1,$s1, 1
ori $s2,$s2, 2
ori $s3,$s3, 3
sw $s0, 0($s0)
sw $s1, 4($s0)
```

```
sw $s2, 8($s0)
sw $s3, 12($s0)
addi $v0,$zero,10          # system call for exit
addi $s1,$zero, 0            #消除相关性
addi $s2,$zero, 0
addi $s3,$zero, 0
syscall                    # we are out of here.
```

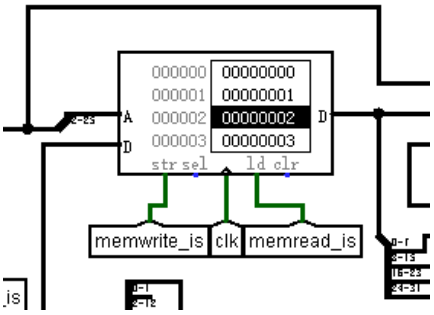

图 4.8 理想流水线测试结果

4.2 可自行安排章节

4.3 性能分析

分析不同方案时钟周期数差异。

4.4 主要故障与调试

4.4.1 指令拓展的故障

理想流水线： 接口处数据传输问题。

故障现象：指令的扩展没有达到预期的效果，乘无符号指令无法执行。

原因分析：MULTU 指令是属于 R 型指令的，在因此进行 R 指令的操作是 MULTU 指令由 op 值为零，而且 funct 值的最高位为零及最低两位选择的输入段为空，因此将无法执行乘无符号指令。

解决方案：改成后的电路图见如图 4.9 所示，只要添加 funct 值的最低两位选择的输入段为零，这样 shift 的输出也为零，因此进行 MULTU 指令的操作。

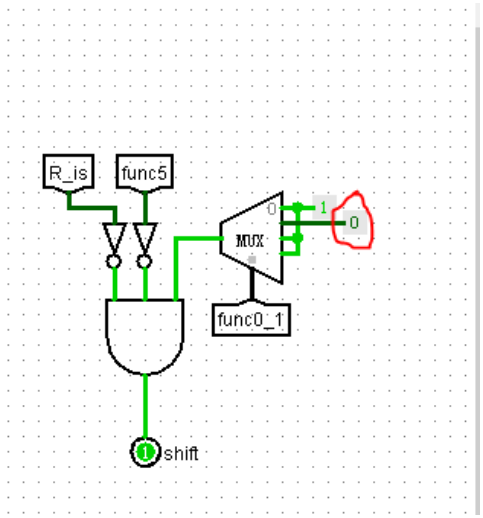

图 4.9 修改后的电路图

4.4.2 指令拓展的故障

理想流水线： 接口处数据传输问题。

故障现象：指令的扩展没有达到预期的效果，读 LO 寄存器时出现 LO 的值随着时钟的改变而被更改，因而读出的值是错误的。

原因分析：MIPS 提供了两个 32 位乘商寄存器 Hi 和 Lo，执行乘法时，Hi 和 Lo 分别存放 64 位乘积的高 32 位和底 32 位，执行除法时，Hi 和 Lo 分别存放余数和商，因此这个两个寄存器的用途在乘除指令的运行段，所以不能用 clk 更新 LO 寄存器的值，否则就会出现 LO 的值不断改变的结果。

解决方案：为了避免 LO 寄存器的值被改变，我们就从另一个角度考虑问题，如果在更新寄存器的值时我们不只考虑 clk 而且还考虑在给定某一个特定的指令时才更新寄存器的值，那么给定 MULTU 指令时，相应的输出信号 multu 控制是否把值读入 LO 寄存器，这样就避免了寄存器的值不断被改变的情况了，改后的电路见图 4.10

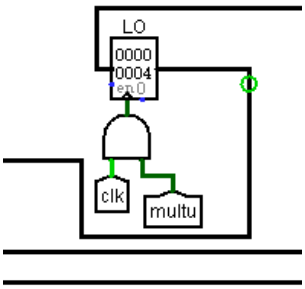

图 4.10 修改后的电路图

4.4.3 中断的故障

中断发生地址书写出现错误。

故障现象：无法实现中断机制，当发生中断时无法跳到中断处理程序段。

原因分析：因为时间限制，我只完成了单极中断，单极中断无论是硬件设计好事软件设计都比多级中断简单很多，因此也没有出现很严重的问题，但是容易出错的一块是中断的发生地址输入错误，因为无法判别是否发生中断，因此就出现无法发生中断的情况。

解决方案：添加正确的地址。如图 4.11 所示：

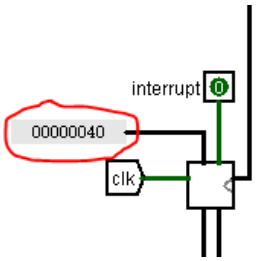

图 4.11 修改后的电路图

4.4.4 理想流水线的故障

执行所有的指令后，在指令的停止控制出现的问题。

故障现象：无法实现理想流水线，ID/EX 寄存器模块出现信号无法通过的状态。

原因分析：因为时钟和指令进行或运算，原本进行这一项是为了使指令停止而添加的电路图，但是也因为这个原因才导致在理想流水线中指令的传递出现了问题，当执行指令 sys 是，过 ID/EX 寄存器时因下降沿而无法传递信号。如图 4.13 所示：

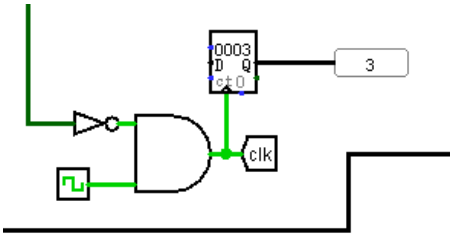

图 4.12

解决方案：指令的停止更改为另一种方法，而不在采用通过或门停止的方法，从而解决了信号无法传递的问题。

4.5 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	查阅相关资料，完成部分指令的拓展。
第二天	完成单周期 cpu 模块的编码
第三天	调试成功并上板但在周期 CPU
第四天	查看理想流水线资料并整理模块
第五天	完成理想流水线。
第六天	查看气泡流水线资料并整理模块
第七天	完成气泡流水线
第八天	查看重定向流水线资料
第九天	完成重定向流水线

5　设计总结与心得

5.1 课设总结

本次实验主要做了以下几点工作：

1） 完成在原有的 24 条指令的基础上添加 7 条新指令，从而进行指令的扩展。并且完善单周期 CPU，解决原来存在但是不明显的错误。

2） 完成单级中断，从而了解中断的实现机制，后续有尝试完成多级中断，但是没有得到理想的结果，因此最后实验只是采用了单级中断。

3） 完成理想流水线，开始做流水线时并没有领会到流水线的在 CPU 中所起的作用，但是在最后书写测试程序的时候才领会流水线重要作用，并使之前所拓展的指令能在流水线中执行，完成理想流水线的测试。

5.2 课设心得

本次课程设计是在原来的实验课中的第三个实验的基础上完成的。由于在完成实验三的时候没有投入太多精力也没有认真的完成实验，因此当初的偷懒引来的后果是课程设计的初步阶段就被别人慢了很多，因为在原来的基础上扩展指令实际上也不算是很难设计的，但是由于寒假假期的时间足以让我对实验三的内容忘得一干二净，所以我想完成课设必须从最初的实验三内容拾起才行，我得知道当初为何设计那些信号与隧道，而且电路原本就存在并且没有被发现的小故障等等，说实话也为当初为什么不认真做实验后悔了一段时间。因此我用了三天时间完成了指令的拓展和测试。

第二阶段的任务是如何完成中断，在这段任务的设计中，我很感谢我们团队里的队友，因为从我们拿到任务书后，大家都忙着设计个人指令拓展部分，所以以为所谓的团队只是说说而已，但是事实证明并非如此，在团队里熊志强同学扮演着队长的角色，在我们都在做指令拓展的时候，他和肖涵就开始研究中断，并且最后搞定了，但是他们完成这部分思路总结时并没有马上忙着做自己下部分的设计，而是召集队友一起开会，然后一起分享他们总结的思路，并且耐心的解说，对其他队友提出的异议也做出相应的解释，当时觉得原来队友是这样的，不是相互抄袭而是相互扶持，帮助彼此。但是由于个人能力的不足，最后也没能完成多级中断，队友们也在多级中断完成

过程中帮了很多忙，但是最终因为时间原因放弃了多级中断的设计，而改为单级中断才完成这部分的设计。

在总体设计中我的速度和进度都比其他人慢很多，所以在本次课程设计中我只设计了指令的拓展，单级中断和理想流水线。其实得承认因为能力上的不足才导致我的实验进展变慢，因为在出现错误并要调试的时候，我很难找到出现错误的地方。因此很多时间浪费在了查找电路的错误当中。有些错误的发生很奇怪，我将近花了一天的时间去解决问题，并且试着询问三四个同学查找错误的地方，但是一无所获。最后还好在另一个同学的帮助下找到错误，而且这个出现问题的地方是在连流水线前没有出现过，就是指令的停止出现了问题，明明流水线接口部件没有问题，可是信号怎么都没传过去，最后发现是 clk 出现了问题，只是之前的单周期 CPU 没有被发现，就这样实验进度就只能到理想流水线，这次实验让我收获不少，如果有时间我会试着再继续闯一下关。

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] 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.

[5] 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: