***2018***



**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： | CS1503 |
| 学 号： | U201514559 |
| 姓 名： | 周铭昊 |
| 电 话： | 15802740273 |
| 邮 件： | [630212894@qq.com](mailto:630212894@qq.com) |
| 完成日期： | 2018-03-10 周五下午 |

目 录

[1 课程设计概述 3](#_Toc474706961)

[1.1 课设目的 3](#_Toc474706962)

[1.2 设计任务 3](#_Toc474706963)

[1.3 设计要求 3](#_Toc474706964)

[1.4 技术指标 4](#_Toc474706965)

[2 总体方案设计 6](#_Toc474706966)

[2.1 单周期CPU设计 6](#_Toc474706967)

[2.2 中断机制设计 9](#_Toc474706968)

[2.3 流水CPU设计 10](#_Toc474706969)

[2.4 气泡式流水线设计 11](#_Toc474706970)

[2.5 数据转发流水线设计 11](#_Toc474706971)

[3 详细设计与实现 12](#_Toc474706973)

[3.1 单周期CPU 实现 12](#_Toc474706974)

[3.2 中断机制实现 19](#_Toc474706975)

[3.3 流水CPU实现 21](#_Toc474706976)

[3.4 气泡式流水线实现 22](#_Toc474706977)

[3.5 数据转发流水线实现 23](#_Toc474706978)

[4 实验过程与调试 25](#_Toc474706980)

[4.1 测试用例和功能测试 25](#_Toc474706981)

[4.2 性能分析 28](#_Toc474706983)

[4.3 主要故障与调试 28](#_Toc474706984)

[4.4 实验进度 29](#_Toc474706985)

[5 设计总结与心得 31](#_Toc474706986)

[5.1 课设总结 31](#_Toc474706987)

[5.2 课设心得 31](#_Toc474706988)

[参考文献 34](#_Toc474706989)

# 课程设计概述

## 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。

## 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。

## 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。

## 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 |
| 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 $v0==10 halt(停机指令)  else 数码管显示$a0值 |
| 24 | SYSCALL | 系统调用 |
| 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 | SH | 存储半字 |  |
| 31 | BLEZ | 小于等于0转移 |  |

# 总体方案设计

## 单周期CPU设计

本次我们采用的方案是首先设计数据通路，把CPU 中有数据联系的部件连接起来，根据不同的指令选择连接不同的数据来源，再通过控制信号来标识指令的功能，每经过一个时钟周期就执行一条指令。在实施过程中，分为多个模块进行，参照已经设计完成的logisim电路图来分工写对应的verilog代码，然后进行FPGA仿真及上板测试。

总体结构图如图 2.1所示。



图 . 总体结构图

### 主要功能部件

我主要完成了分频器和寄存器模块的实现，具体设计思路如下。

#### 分频器

主要思想是每当经过一段设定的时间后，就把信号翻转一次，从而得到对应频率的时钟信号，通过计数来实现设定时间的功能。

#### 运算器

根据ALU\_OP的不同选择不同的功能对两个32位操作数进行相应操作。算术逻辑运算单元ALU的引脚与功能描述如表 2.1所示，在本次CPU实验中输出引脚只用到Result和Equal。

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 |

#### 寄存器堆RF

采用32个寄存器，通过5位地址线来选择，每个寄存器中保存的值为32bit。

### 数据通路的设计

画出主要功能输入来源表格框架，如表 2.2所示, 得到输入来源后，连接电路中的数据通路，若一个端口有两个以上的数据输入，则添加若干个多路选择器，如此便可以实现特定指令传输特定数据的功能，多路选择器的控制端连接的信号即为控制器所需要的控制信号。

表 2.2指令系统数据通路框架

| 指令 | PC | IM | RF | | | | ALU | | | DM | | EXT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

### 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所示。

表 2.3主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| ALUOP | 0000~1100 | 运算器的功能选择，共用到11种功能 |
| RegDst | 0 | 写入寄存器的地址W#来源于指令的[20:16]位 |
| 1 | 写入寄存器的地址W#来源于指令的[15:11]位 |
| Alusrc | 0 | ALU的第二个操作数B来自于寄存器堆RF.R2 |
| 1 | ALU的第二个操作数B来自于立即数扩展EXT.s |
| Regwrite | 0 | 不用写入寄存器 |
| 1 | 需要写入寄存器 |
| MemToReg | 0 | 写入寄存器的值来源于运算器ALU |
| 1 | 写入寄存器的值来源于数据存储器DM |
| Memwrite | 0 | 不用写入数据存储器 |
| 1 | 需要写入数据存储器 |
| Extop | 00 | 需要扩展的值来源于指令的[15:0]位，且为有符号扩展 |
| 01 | 需要扩展的值来源于指令的[15:0]位，且为无符号扩展 |
| 10 | 需要扩展的值来源于指令的[10:6]位，且为无符号扩展 |
| blez | 0 | 当前指令不为blez |
| 1 | 当前指令为blez |
| beq | 0 | 当前指令不为beq |
| 1 | 当前指令为beq |
| bne | 0 | 当前指令不为bne |
| 1 | 当前指令为bne |
| j | 0 | 当前指令不为j |
| 1 | 当前指令为j |
| jal | 0 | 当前指令不为jal |
| 1 | 当前指令为jal |
| jr | 0 | 当前指令不为jr |
| 1 | 当前指令为jr |
| Syscall | 0 | 当前指令不为Syscall |
| 1 | 当前指令为Syscall |
| sh | 0 | 当前指令不为sh |
| 1 | 当前指令为sh |

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 | ALUop | RegDst | ALUsrc | Reg write | Mem ToReg | Memwrite | Ext op | blez | beq | bne | j | jal | jr | Syscall | sh |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## 中断机制设计

### 总体设计

添加三级中断IR1、IR2、IR3的按钮，等级依次增加，按下中断按钮后，产生中断请求信号IR1\_S、IR2\_S、IR3\_S，如果无法及时响应，对应的等待指示灯亮。进入中断后先保护现场，把中断程序用到的寄存器压栈，同时保存断点的PC寄存器值，然后执行中断程序，中断程序运行结束后回恢复现场，返回到中断前的状态，执行中断前的下一条指令，三个中断程序依次为1、2、3的跑马灯程序。添加指令mfc0开中断，mtc0关中断，eret中断返回信号。

### 硬件设计

使用中断请求信号的产生电路，当响应信号IG出现后，表示系统已经响应了该中断，使对应中断的等待信号W和中断请求信号IR\_S同步清零。EPC寄存器用于保存断点；中断请求寄存器IR用于保存正在执行的中断程序对应的编号；通过多路选择器选择3种不同的中断服务程序地址，用中断号作为选择信号。最后要实现高等级中断打断低等级中断的效果，再添加两个信号w111表示中断1被2或3打断，w222表示中断2被3打断。IE控制开中断和关中断。

### 软件设计

分别设计1、2、3的跑马灯汇编程序，并获取这三个中断服务程序的入口地址， 把入口地址写到多路选择器的输入端。每个中断服务程序的开头和结尾都要加上保护现场（压栈）和回复现场（出栈）的代码，而且在保存现场和恢复现场时，不能接受中断，因此需要进行开关中断的控制。

## 流水CPU设计

### 总体设计

要实现经典的五段流水线，把一条指令的执行过程分为取指令IF、译码ID、执行EX、访存MEM、写回WB五个阶段，每一阶段需要的时间大致相等，在每个阶段后面加一个流水接口部件（实际上是一个锁存器，用于锁存上一个阶段加工数据或处理结果）。流水线执行过程可能会遇到数据冲突或控制冲突的现象，需要插入气泡或者重定向来解决。

### 流水接口部件设计

流水接口就是一个简单的锁存器，每经过一个时钟周期就把输入的值更新到输出接口。由于需要暂停流水线的运行，所以流水接口部件还需要一个锁信号，当锁信号为1时才进行赋值，锁信号为0时输出的值不改变。

### 理想流水线设计

理想流水线不必考虑数据冲突和控制冲突等特殊情况，只需要把各个部件封装完成，信号通过流水接口依次传递即可，如果一个信号在后面的阶段需要使用，那么就需要从前一个流水部件传递过来，后面用不到的信号便不用传递。

## 气泡式流水线设计

数据冲突指后续指令要用到前面指令的操作结果，而这个结果尚未产生或尚未送到指定的位置，从而造成后续指令无法继续执行的状况。当检测到这些指令出现并且确实出现了数据冲突时（如果两条指令访问的是不同的寄存器便不会造成冲突），插入气泡，若仍存在数据冲突，则继续插入气泡直到不冲突。在插入气泡时应当使PC暂停，不再执行下一条指令。

结构冲突可以通过修改寄存器触发方式，改为先写后读即可。

控制冲突指执行阶段EX执行了一条跳转分支指令，那么在这条指令之后的指令都要清空，即把误取指令在IF/ID和ID/EX两个流水接口处清空。

## 数据转发流水线设计

重定向流水线与气泡流水线的控制冲突解决方式相同，但是处理数据冲突的方式不同，当发生数据冲突时，将正确的数据从对应位置重定向到运算器ALU的操作数端。但是如果两条相邻指令存在数据相关，且前一条指令时访存指令时，不能使用重定向的方式处理，成为load-use情况，出现load-use时需要插入气泡解决。

# 详细设计与实现

## 单周期CPU 实现

### 主要功能部件实现

1. 分频器
2. Logism实现：

分频器不需要Logisim实现，是FPGA特有内容。

1. FPGA实现：

分频器的Verilog代码如下：

always @(posedge clk\_in) begin

if(choose) begin

if (cnt>=30)//actually 2500000 times

begin

clk\_out <= ~clk\_out;

cnt <= 0;

end

else

begin

cnt <= cnt+1;

end

end

else begin

if (cnt>=2500000)//actually 2500000 times

begin

clk\_out <= ~clk\_out;

cnt <= 0;

end

else

begin

cnt <= cnt+1;

end

end

end

1. 寄存器堆（RF）
2. Logism实现：

使用CS3401库中的32位寄存器组，5位输入地址，每一个寄存器都是32位，共有32个寄存器。

如图 3.1所示。

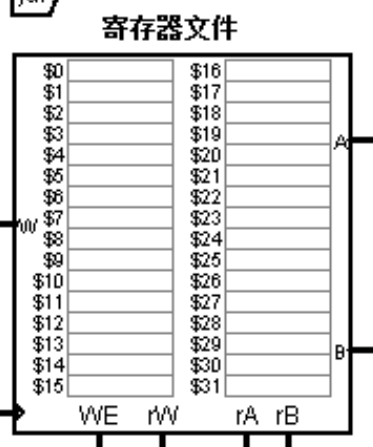


图 3.1寄存器堆（RF）

1. FPGA实现：

直接使用Vivado中自带的32位reg作为单个寄存器，声明32个寄存器，选择地址位宽为5位， 两个输出端分别连接到对应地址的寄存器上，每次遇到上升沿时，若写使能且写入寄存器不是0号寄存器，则把输入进来的32位数据写入到对应地址。

寄存器堆RF的Verilog代码如下：

assign A = register[rA];

assign B = register[rB];

always @(posedge clk)

begin

if ((rW != 0) && (WE == 1))

begin

register[rW] <= W;

end

end

### 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 .指令系统数据通路表

| 指令 | PC | IM | RF | | | | ALU | | | DM | | EXT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |  |
| 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。如图 3.2所示

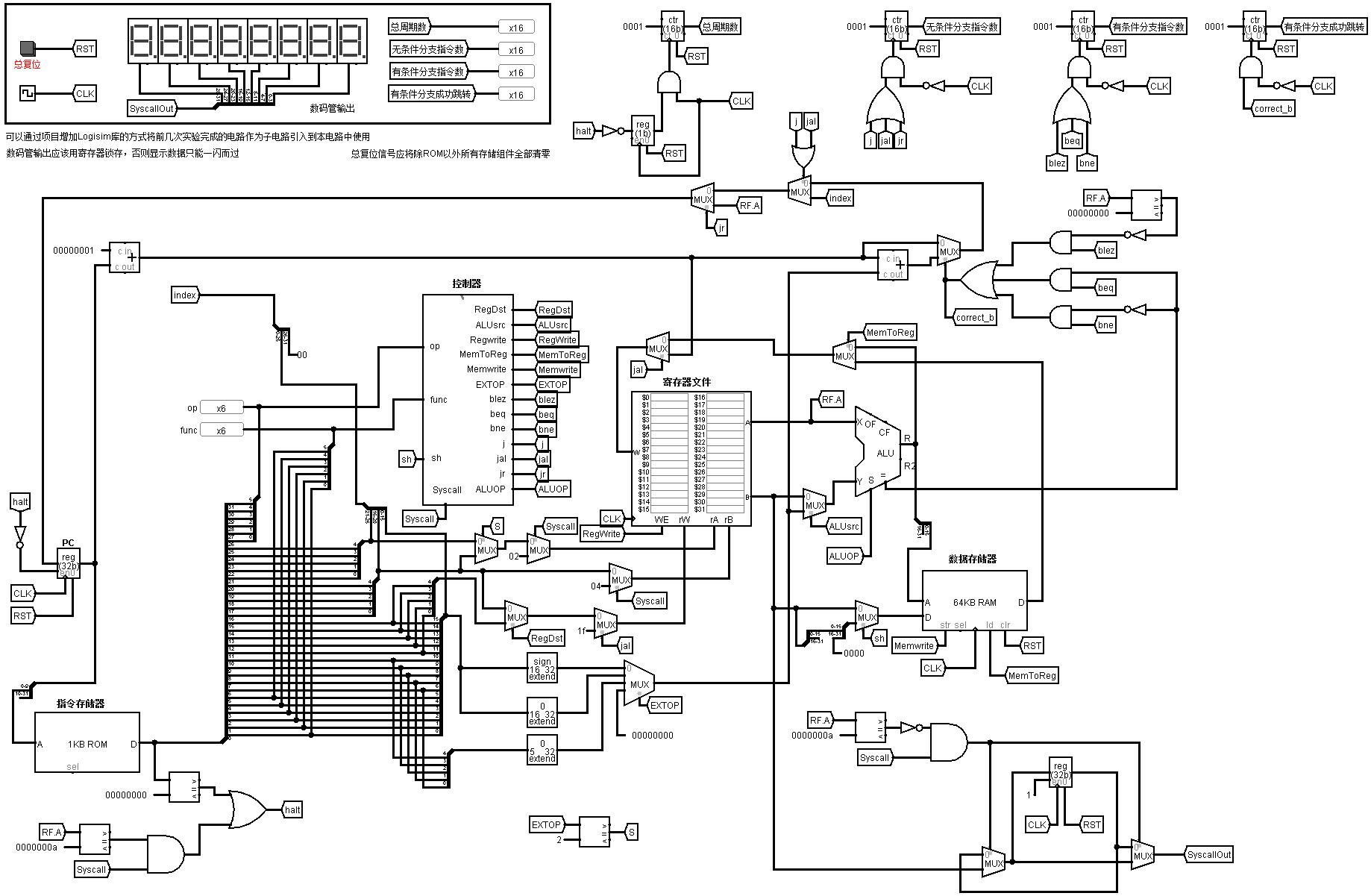


图 .单周期CPU数据通路（Logism）

在Vivado中使用Verilog语言搭建的数据通路的原理图如图 3.3所示。

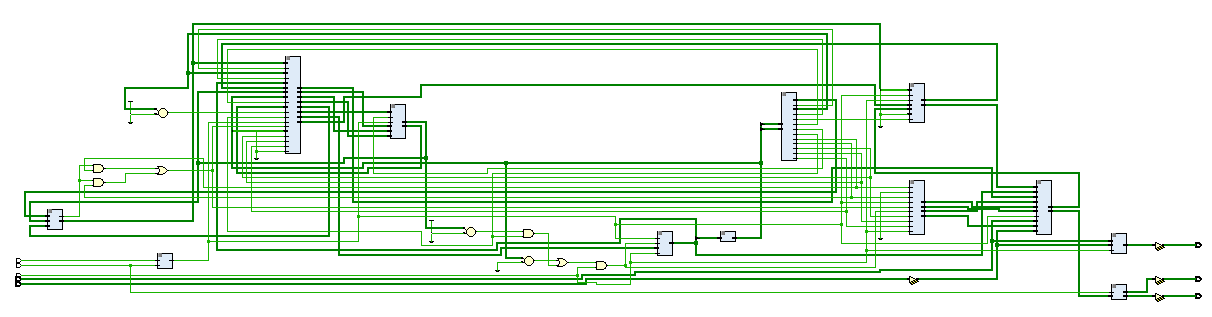


图 .单周期CPU数据通路（FPGA）

### 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 | ALUop | RegDst | ALUsrc | Reg write | Mem ToReg | Memwrite | Ext op | blez | beq | bne | j | jal | jr | Syscall | sh |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| add | 0101 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| addi | 0101 | 0 | 1 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| addiu | 0101 | 0 | 1 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| addu | 0101 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| and | 0111 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| andi | 0111 | 0 | 1 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sll | 0000 | 1 | 1 | 1 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sra | 0001 | 1 | 1 | 1 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| srl | 0010 | 1 | 1 | 1 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sub | 0110 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| or | 1000 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ori | 1000 | 0 | 1 | 1 | 0 | 0 | 01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| nor | 1010 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| lw | 0101 | 0 | 1 | 1 | 1 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sw | 0101 | 0 | 1 | 0 | 0 | 1 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| beq | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| bne | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| slt | 1011 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| slti | 1011 | 0 | 1 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sltu | 1100 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| j | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| jal | 1101 | 0 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| jr | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| syscall | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| XOR | 1001 | 1 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SLTIU | 1100 | 0 | 1 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SH | 0101 | 0 | 1 | 0 | 0 | 1 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| BLEZ | 1101 | 0 | 0 | 0 | 0 | 0 | 00 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Logisim上add操作部分控制电路如图 3.4所示。

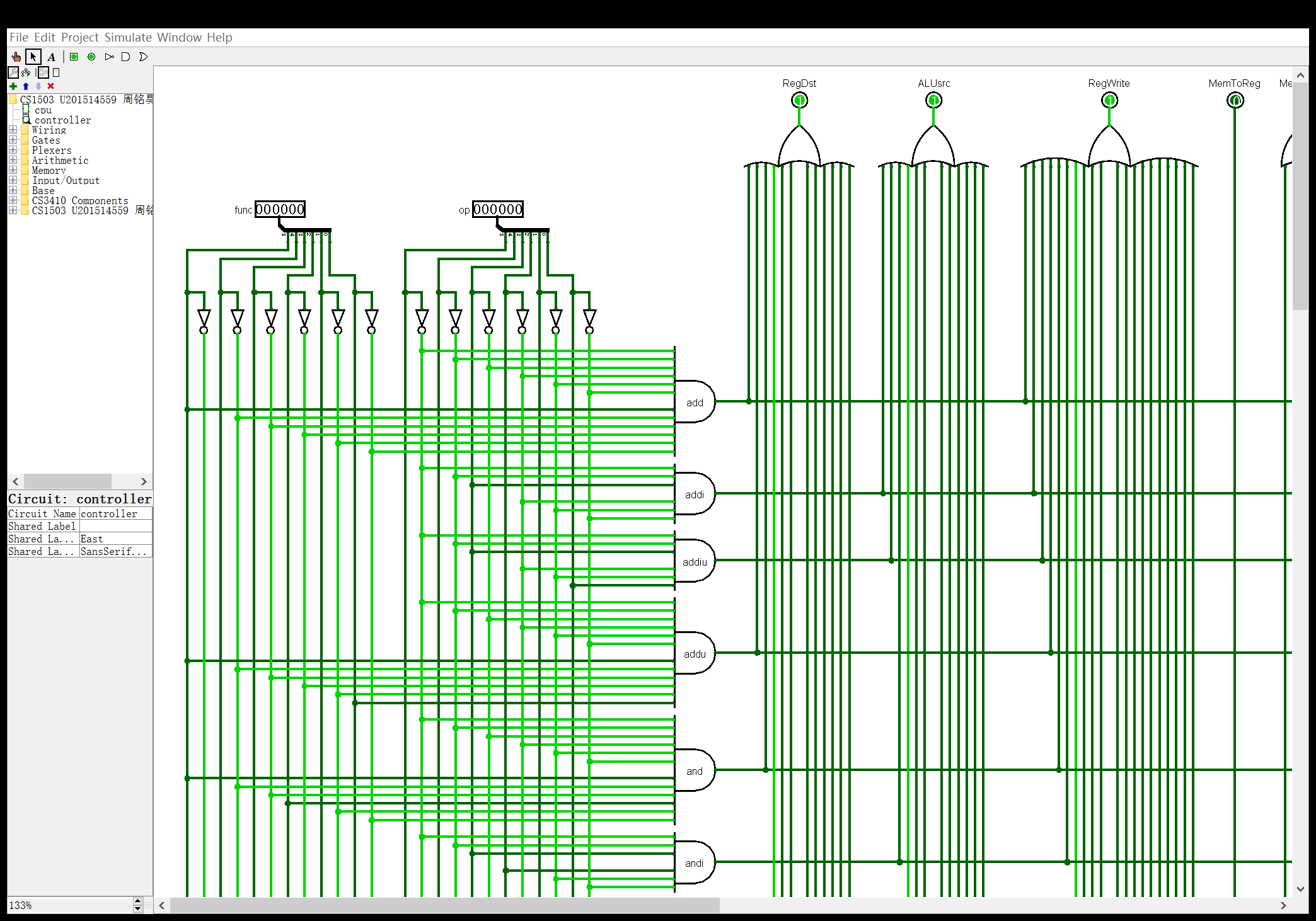


图 . add操作部分控制电路

1. FPGA实现

根据在Logism实现中得到的各条指令对应的控制信号，Verilog代码过于冗长，故只取对于指令add对应的控制信号生成代码举例如下：

6'b100000:begin//add

jr <=0;

jal <=0;

j<=0;

bne<=0;

beq <=0;

Memwrite <=0;

MemToReg<=0;

ALUsrc <=0;

Syscall <=0;

RegDst <= 1;

Regwrite <= 1;

EXTOP <= 2'b00;

ALUOP <= 4'b0101;

end

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 .主控制器原理图

## 中断机制实现

### 硬件过程

指当接收到中断请求信号后，判断当前是否能响应这个中断，如果可以响应则发出中断响应信号IG并且进入中断周期，首先暂停PC寄存器并保存断点，即把PC的值写入到EPC中，然后进行中断识别，把当前响应的中断写入中断寄存器IR中，再根据中断号把中断服务程序的入口地址写入到PC中。到此完成了中断周期的硬件准备工作，一共需要两个周期。

要实现多级中断，只有当前中断请求信号的优先级高于正在运行的中断优先级时，才发出IG响应信号，否则处于等待状态，直到中断结束，再次判断是否响应。

### 中断服务程序（软件过程）

首先保护现场，由于中断服务程序中用到寄存器$s1、$a0、$v0，因此将它们分别入栈，随后因为有多级中断，所以EPC可能需要保存不止一个值，因此EPC也需要入栈，但是由于EPC不在寄存器堆RF中，不能直接使用sw指令，因此这里采用所有指令都没有用到的30号寄存器来代替EPC寄存器，当出现sw $30, 0($sp)指令时，实际上不是30号寄存器，而是在电路中替换为EPC入栈，保护现场过程完成后系统才可以接受新中断，因此在入栈完成后开中断，代码如下：

addi $sp, $sp, -4 #入栈

sw $s1, 0($sp)

addi $sp, $sp, -4 #入栈

sw $a0, 0($sp)

addi $sp, $sp, -4 #入栈

sw $v0, 0($sp)

addi $sp, $sp, -4 #EPC入栈

sw $30, 0($sp)

mfc0 $0,$0#开中断

恢复现场与保护现场思路类似，由于恢复现场过程不能被打断，所以首先进行关中断，然后按照入栈的反顺序依次出栈，出栈完成后开中断，在最后加上中断返回信号，代码如下：

mtc0 $0,$0#关中断

#恢复现场

lw $30, 0($sp) #EPC出栈

addi $sp, $sp, 4 #实际上不是30号寄存器，当检测到lw且$30时，替换为EPC入栈

lw $v0, 0($sp) #出栈

addi $sp, $sp, 4

lw $a0, 0($sp) #出栈

addi $sp, $sp, 4

lw $s1, 0($sp) #出栈

addi $sp, $sp, 4

mfc0 $0,$0#开中断中断返回

eret#中断返回

## 流水CPU实现

### 流水接口部件实现

流水部件共有IF/ID、ID/EX、EX/MEM、MEM/WB四个，每个流水部件的结构类似，都是实现锁存器的作用。区别在于接口的多少，其中接口最多的是ID/EX流水接口部件，接口最少的是IF/ID流水部件，IF/ID的实现电路如图 3.6所示。

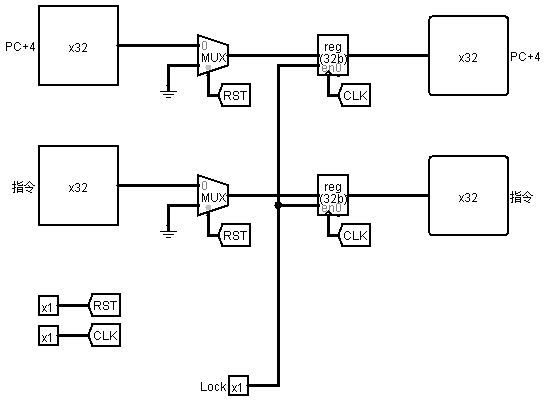


图 . IF/ID流水部件原理图

### 理想流水线实现

理想流水线实现过程中，首先把所有的部件封装一下，然后把封装好的模块按照功能放在五段流水线中的相应位置。IF取指令阶段包括PC寄存器和指令存储器，ID译码阶段包括控制器、译码器、位扩展、寄存器堆，EX执行阶段包括运算器、NPC、判断停机部件，MEM阶段包括数据存储器，WB阶段没有具体模块，由几个多路选择器构成。链接完成后进行测试，原理图如图 3.7所示。

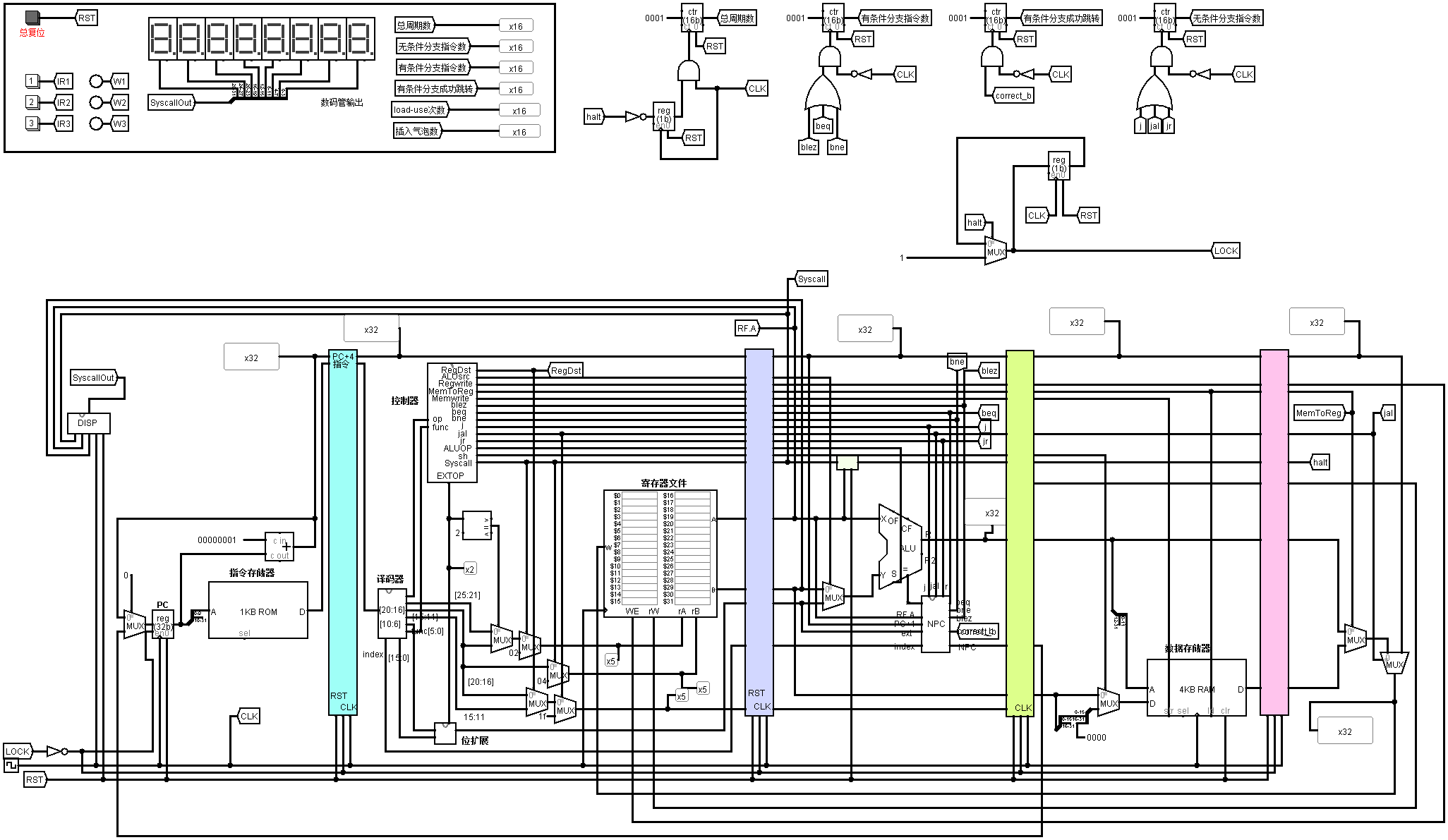


图 . 理想流水线logisim电路图

## 气泡式流水线实现

在理想流水线的基础上，添加气泡判断逻辑，当一条指令在MEM阶段或EX阶段的Rd与在ID阶段的Rs相同，并且写使能的话，则读取的数据在之前修改过，判定为Rs发生数据冲突。当一条指令在MEM阶段或EX阶段的Rd与在ID阶段的Rt相同，并且写使能的话，判定为Rt发生数据冲突。当Rs发生数据冲突且当前ID段的指令不是j或jal（即该指令用到了Rs）时，判定为发生数据冲突。当Rt发生数据冲突且当前ID段的指令确实用到了Rt时，也判定为发生数据冲突。

成功判定发生数据冲突后，stall标志变为1，把PC寄存器和流水部件IF/ID锁住，不再取下一条指令执行，在ID/EX阶段插入气泡，直到数据冲突被消除，stall标志变为0.

然后解决控制冲突，当出现无条件跳转指令或者分支指令成功跳转时，j\_bub标志变为1，分别在IF/ID、ID/EX两个流水部件上插一个气泡即可把误区指令清除，从而消除冲突。判定冲突部分的原理图如图 3.8所示。

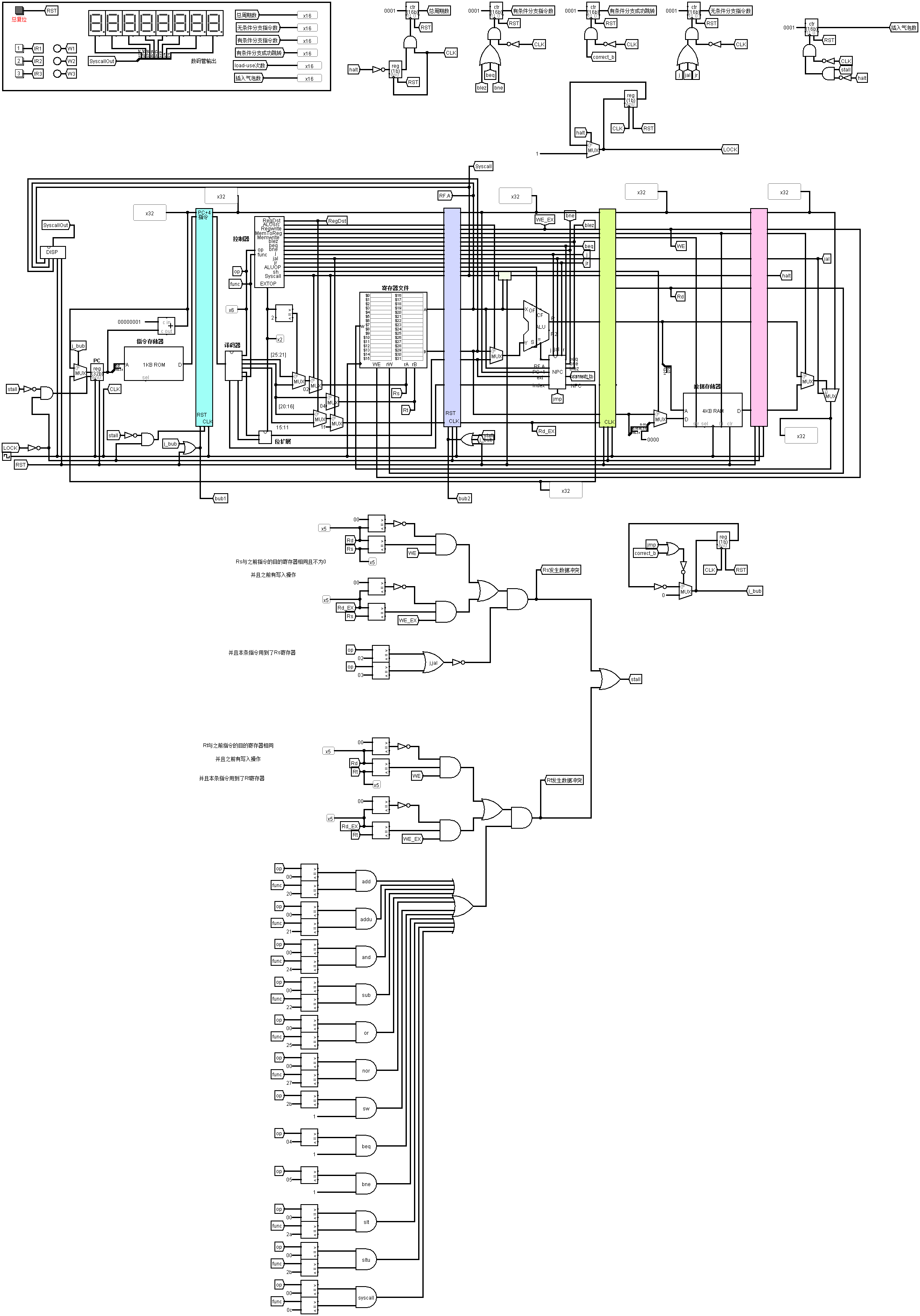


图 .气泡流水线冲突判断logisim电路图

## 数据转发流水线实现

重定向流水线的控制冲突解决方案与气泡流水线一致，都是通过标志位j\_bub来进行控制。在发生数据冲突时，如果前一条指令的memtoreg或memwrite为1时，代表是仿存指令，再发生冲突即load-use，出现load-use时load-use标志变为1，把PC寄存器和流水部件IF/ID锁住，不再取下一条指令执行，在ID/EX阶段插入气泡，直到load-use被消除，load-use标志变为0。重定向流水线cpu电路图如图 3.9所示

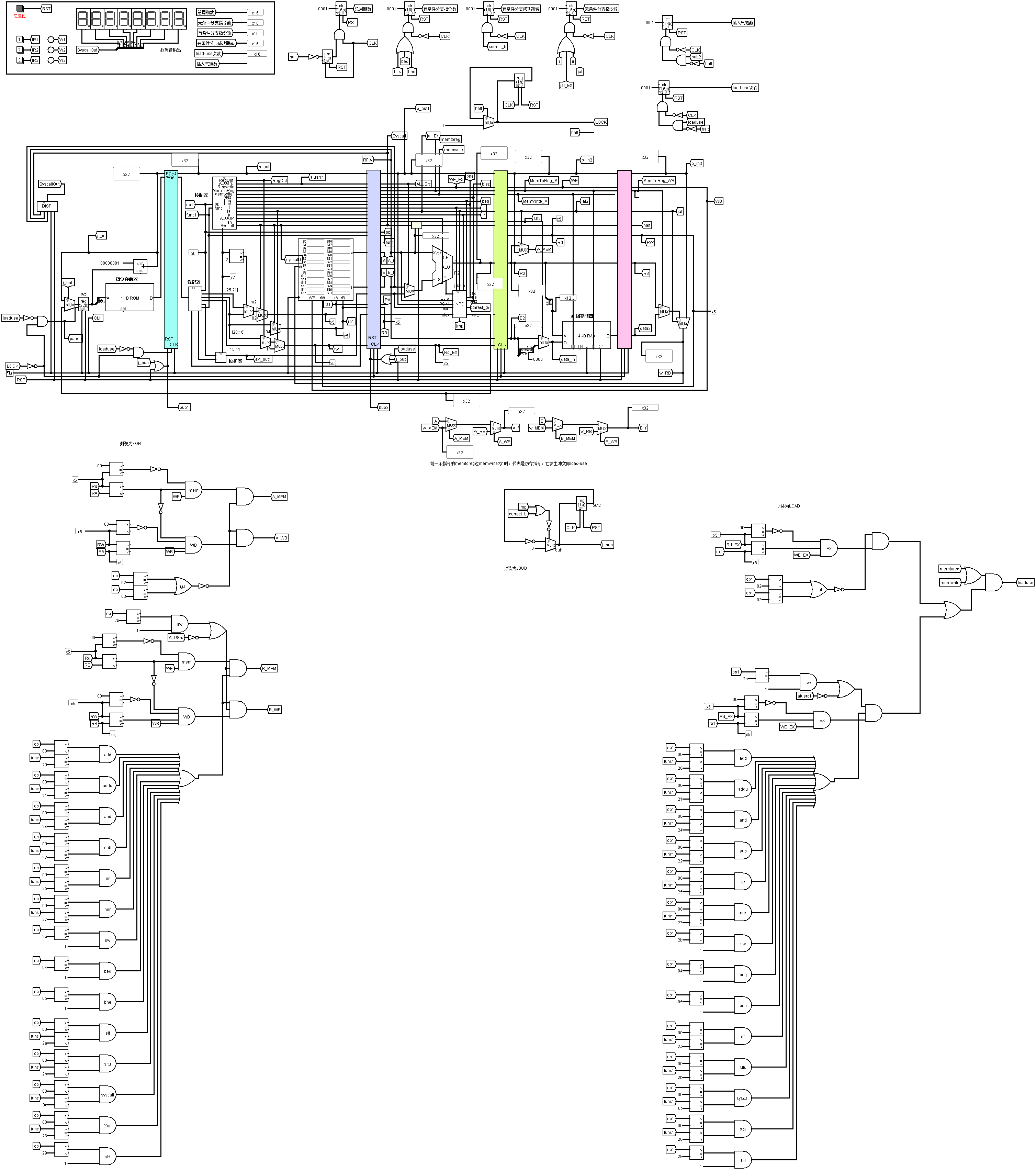


图 .重定向流水线logisim电路图

# 实验过程与调试

## 测试用例和功能测试

分别测试单周期CPU、理想流水线、气泡流水线、重定向流水线、多级中断、流水线FPGA及CCMB测试。

### 单周期CPU测试

首先编写test代码，choose=1代表选择频率为高频进行仿真，代码如下

module i7\_test();

reg clk,RST;

reg [2:0]pro\_reset;

reg [11:0]in\_addr;

reg choose;

wire [15:0]leds;

wire [7:0]SEG;

wire [7:0]AN;

initial begin

clk = 0;

RST = 0;

pro\_reset = 0;

in\_addr = 12'b0;

choose = 1;

//下为iverilog+gtkwave调试所需的语句

$monitor("At time %t, ocnt = %d", $time, clk);

$dumpfile("counter\_test.vcd");

$dumpvars(0, i7test);

end

always #5 clk = ~clk;

i7\_6700k i7test(clk,RST,pro\_reset,in\_addr,choose,leds,SEG,AN);

endmodule

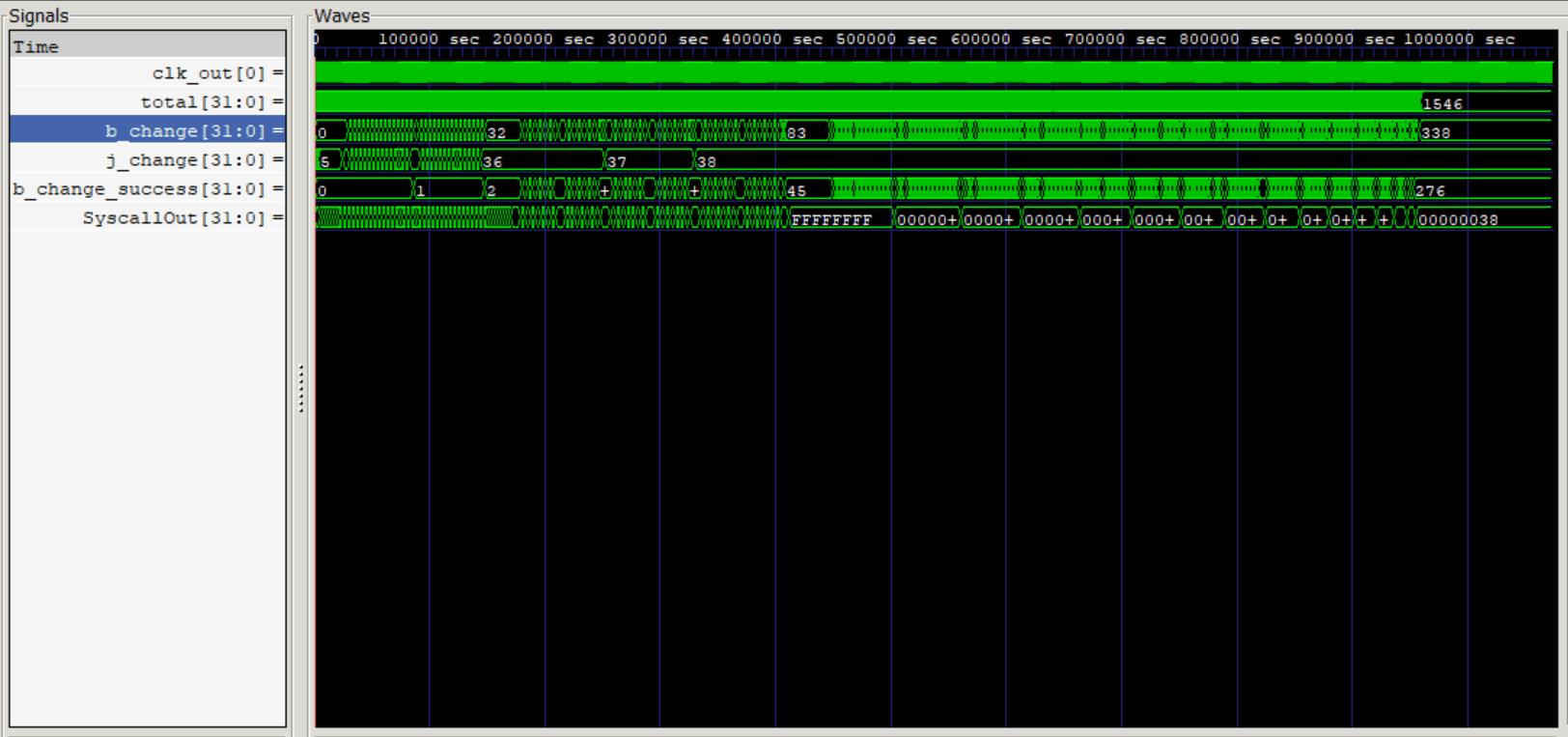
使用iverilog和gtkwave进行仿真，单周期CPU仿真结果如

图 4.1所示，运行结束后总周期为1546，无条件跳转指令38条，条件分支指令338条，条件分支指令成功跳转276条，结果正确。

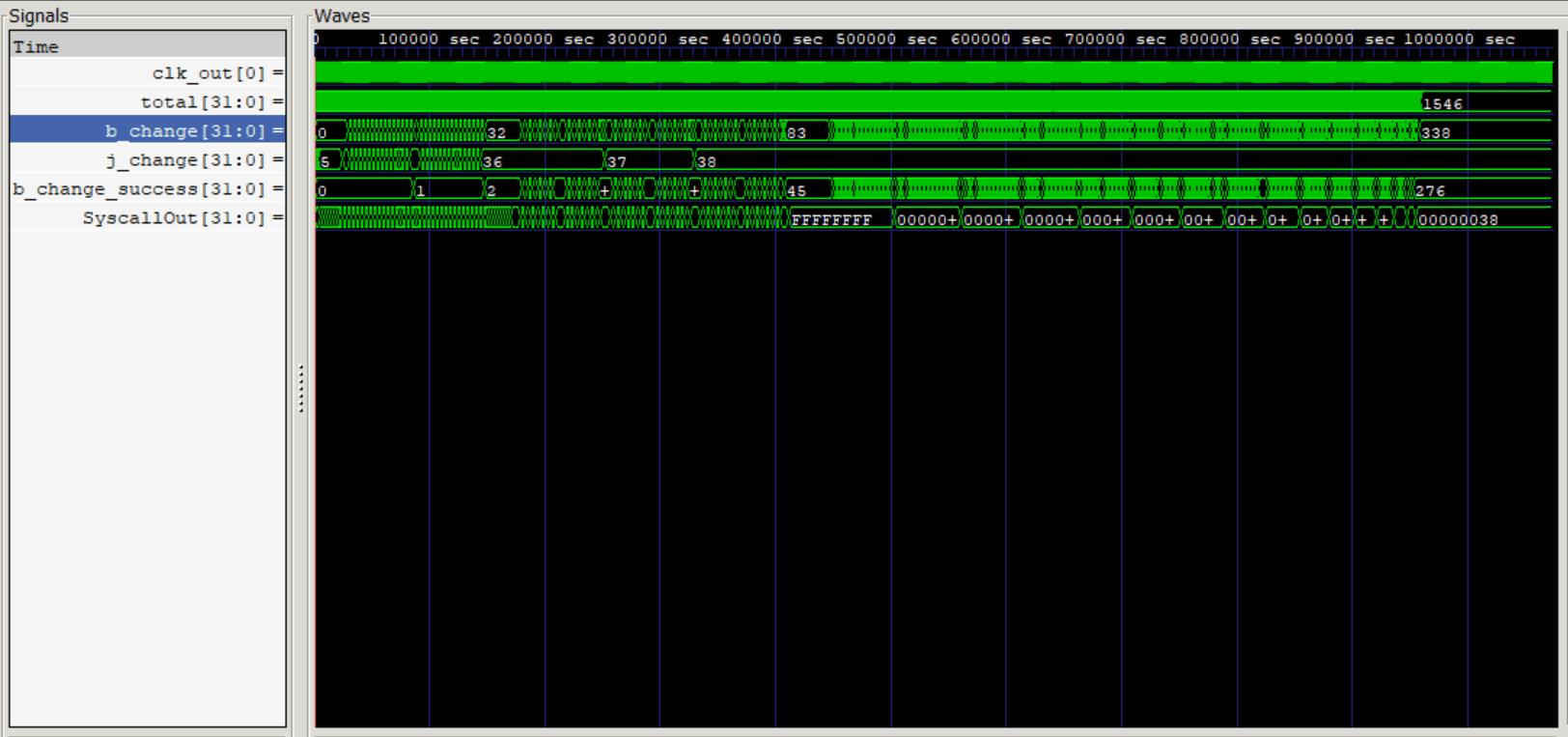


图 .单周期CPU仿真结果

### 理想流水线测试

把老师提供的测试程序加载到指令存储器中，运行一次测试程序，总周期数应该为21，运行结果如图 4.2所示，测试正确。

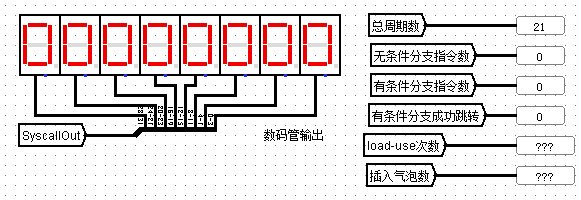


图 .理想流水线测试结果

### 气泡流水线测试

把benchmark加载到指令存储器中，运行一次测试程序，结果总周期数应该为3624，数据冲突插入气泡数为1447个，j指令数38条，条件分支指令338条，条件分支指令成功跳转276条，满足1546+4+38\*2+276\*2+1447=3624， 运行结果如图 4.3所示，测试正确。

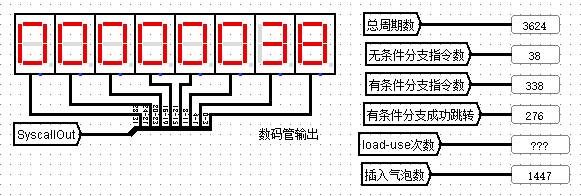


图 .气泡流水线测试结果

### 重定向流水线测试

把benchmark加载到指令存储器中，运行一次测试程序，结果总周期数应该为2298，load-use次数为120次，j指令数38条，条件分支指令338条，条件分支指令成功跳转276条，满足1546+4+38\*2+276\*2+120=2298， 运行结果如图 4.4所示，测试正确。

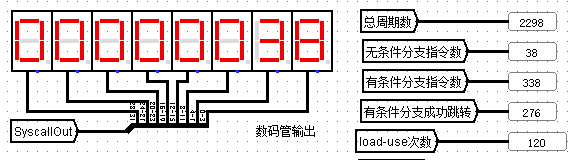


图 .重定向流水线测试结果

### 多级中断测试

把添加了中断程序后的benchmark加载到指令存储器中，依次按下213三个中断首先进入2中断，执行2的跑马灯程序，1进入等待状态，再按下3后打断2的跑马灯程序，进入3的跑马灯程序，结束后返回继续运行2的跑马灯，最后运行1的跑马灯，运行结果中间截图如图 4.5所示，经老师检查测试正确。

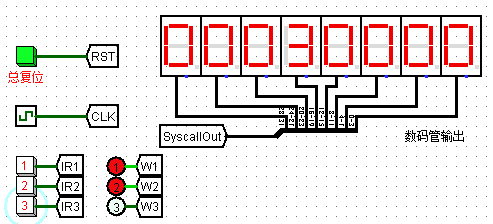


图 .多级中断测试中间截图

### 流水线FPGA测试

运行一次测试程序，结果总周期数应该为2298，load-use次数为120次，j指令数38条，条件分支指令338条，条件分支指令成功跳转276条，满足1546+4+38\*2+276\*2+120=2298，运行结果如图 4.6所示，测试结果正确。

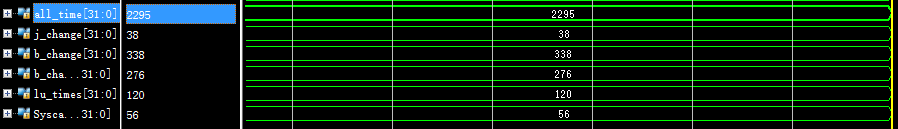


图 .流水线FPGA仿真结果

## 性能分析

分析不同方案时钟周期数差异： 气泡流水线的周期数多于单周期，但运行速率更快，因为等待运行的时间很少，取完一条指令之后可以马上进行下一条指令的执行，而不必等这条指令执行完成 。重定向流水线可以显著减少周期数，因此比气泡流水线的性能更优。

## 主要故障与调试

### 单周期FPGA故障

单周期FPGA：分频器错误。

**故障现象：**仿真结果正确，上板结果与标准结果不一致。

**原因分析：**在分频器模块中用到计数器来记录时间，每过一个周期计数器+1，但要求计数器的位数足够。如果计数器只有3位，那么永远不能达到250000使其翻转。

**解决方案：**把计数器的位数修改为32位。

### 气泡流水线故障

气泡流水线：气泡数的统计结果错误。

**故障现象：**气泡数和总周期数的统计结果比标准值明显偏高。

**原因分析：**在出现数据冲突时，不仅仅要判断MEM和EX段写入的数据与当前ID段要读取的数据出现冲突，而且要弄清楚这些数据冲突的确会造成错误，因为有一些指令是没有用到发生冲突的寄存器的，那么就不需要插入气泡来排除冲突。

**解决方案：**在判断数据冲突时，加入一个逻辑，判断这条指令是否真正用到了发生冲突的寄存器：Rs寄存器除了j和jal指令没有用到，其余指令都用到了。用到Rt寄存器的指令有：add、addu、and、sub、or、nor、sw、beq、bne、slt、sltu、syscall。

### 中断程序故障

多级嵌套中断：213中断顺序按下后，无法正确回到2号跑马灯程序。

**故障现象：**先按下2号中断按钮，进入2号中断服务程序，再按下1号按钮，不会打断2号中断，1号中断进入等待状态，再按下3号按钮，打断2号中断，当3号中断服务程序结束后，无法正确返回到2号中断，错误。

**原因分析：**当3号中断完成之后，当前中断清0，由于2号中断已经响应过，因此2号中断请求信号也清0了，相当于处于开中断的程序响应了1号中断，所以没有办法回到2号中断分析，必然导致程序出错。

**解决方案：**加一个标志3break2来表示2号中断执行过程中被3号中断打断，当3break2标志为1时，不响应1号的中断请求，这样就解决了**。**

## 实验进度

表 . 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，完成分工的寄存器模块和分频器模块。 |
| 第二天 | 把大家写好的模块合并起来进行仿真查错。 |
| 第三天 | 完成了CPU上板检查测试。 |
| 第四天 | 把单周期CPU的各种部件封装起来，设计流水接口，完成了理想流水线的检查测试。 |
| 第五天 | 在理想流水线的基础上进行气泡流水线的设计，完成气泡流水线logisim并检查测试。 |
| 第六天 | 在气泡流水线的基础上修改数据冲突的处理逻辑，完成重定向流水线的检查测试。 |
| 第七天 | 用verilog写出重定向流水线的代码，并进行仿真测试。仿真成功后上板测试。 |
| 第八天 | 完成单周期CPU上的单单级中断。 |
| 第九天 | 在单级中断的基础上完成多级嵌套中断。 |
| 第十天 | 在最终版本添加CCMB支持，完成CCMB检查和测试。 |

# 设计总结与心得

## 课设总结

本次课设实现经典五段流水线的CPU设计和测试。我主要作了如下几点工作：

1. 完成方案总结：设计了理想、气泡、重定向流水线的具体电路，设计了流水接口的结构，设计了中断的硬件部分和软件部分。
2. 功能总结：实现了单级中断和多级嵌套中断，实现了气泡流水功能，实现了重定向流水功能，实现了数据冲突和控制冲突的判定，实现了load-use情况的判定和统计。
3. 完成了一个支持多级中断的单周期CPU，完成了一个支持重定向的五段流水线CPU，完成了重定向CPU上板。

## 课设心得

本次课程设计历时2周，有效工作时间在9天左右，差5分通关是一大遗憾，在做气泡流水线时我的进度还算领先，周末休息两天就跟不上同学的脚步了。让我意识到竞争是随时随地的，学习时应当紧绷神经，不然就会被别人超越。

回顾整个课程设计的过程，团队沟通还算比较愉快高效，是全班第二个完成团队任务检查的组，虽然团队后期大家基本都在自己做自己的，但是在合作的时候大家都很尽心尽力，按时开例会，坐在一起讨论问题，那段时间学到了很多调试的技巧，iverilog也是那时学会用的，所以后期进行重定向上板时调试起来更加轻松方便。

单周期CPU上板完成之后，团队任务告一段落，考验个人能力的时候到了。首先是理想流水线的设计，主要工作量其实都在封装之前的单周期CPU上，剩余的流水接口其实比较简单，只要通读一遍老师提供的原理图就能明白流水线的原理，懂了流水线的运作原理后，具体实现起来就比较轻松。这里要感谢写实验指导文档的老师，写的很好很详细，让我们用最少的时间学会实验内容。

理想流水线并不能算是一个完整的CPU，因为会遇到很多并不理想的情况。要解决这些冲突，首先设计的是气泡流水线，这里遇到的一个障碍就是气泡数调试了很久都没对，一开始是因为把数据冲突的气泡和结构冲突的气泡都进行了计数，所以气泡数高的离谱，后来改成只计算数据冲突的气泡后，还是比标准要稍高一些，虽然此时已经满足公式要求了，总周期数可以对得上号。但是我还是不明白这些多的气泡从何而来，最后在吃饭时灵光一闪，原来是自己没有判断有一些指令并没有用到具体的寄存器，如果这些指令的机器码恰好表现出数据冲突，那就会加入一些无效的气泡，下午立马赶到实验室进行修改，然后检查完成。随后便是周末，休息了两天。

回来之后发现同学进度都比较快，于是我马上开始画重定向电路，在气泡流水线的基础上略作修改即可，检查完成后进入重定向上板的过程，有了之前单周期CPU的上板经验，这次重定向上板显得轻车熟路，前面整个团队用了3天才实现单周期FPGA，自己只用了一天多一点就完成了重定向的上板并检查，这就是经验的力量。

重定向上板完成后进入比较困难的部分—中断，这一部分尽管老师的文档写得很详细，我还是看了很久，边做才边明白其中的原理，主要就是软硬件部分协同比较难理解，在明白了各个部件的功能之后，做起来就很快了。要对中断的过程比较熟悉，响应了一个中断后，先是保护现场，主要保护的是PC的值，保护完现场之后才开中断，然后进入中断服务程序，在这些程序中再次进行现场保护，这次主要保护的是寄存器中的值，单级中断并不需要保存EPC的值，因为一次只有一个中断进行。中断服务程序完成后先进行关中断，因为回复现场的过程不能被打断，然后恢复现场，再次开中断，最后中断返回，完成单级中断。

多级中断中需要修改响应中断的逻辑，根据当前运行的优先级来进行响应，这里有一个问题自己没能测试到，就是213按下后，不能回到2号中断，等到给老师测试的时候才意识到这个问题，就是一个没有考虑到的漏洞，修改起来倒是不难，添加一个标志即可，但是却很难发现。让我意识到测试用例的重要性，老师们应该都比较清楚什么样的测试比较能测试出完整的功能，这就需要自己多加学习了。

最后一天星期五添加CCMB支持并且进行最终版本检查，并没有做什么实质性的工作，所以有效工作时间基本上就是之前的9天。本次课程设计的主要不足就是到了后期，同学之间的交流不像第一个实验那样频繁，大家都在自己做自己的，导致队员之间的差异较大。

我建议以后在进行个人的实验时，可以保持开例会的传统，大家交流一下自己的进度，自己遇到的问题等等，这样可以保证一下大家的进度不会相差太远。课程设计的内容我感觉如果周末不加班要想通关就比较困难了，但是难度也适中吧，要想通关也不能设置的太过容易。

还有就是有关分组的建议，我建议自由分组，这样比较熟悉的同学在同一组，大家讨论起来也比较热烈，如果不是很熟的话，大家的交流可能会少一点。

最后总结一下，本次课程设计是一次难忘的体验，让我对系统结构有了更进一步的了解，提升了团队协作的能力。感谢各位老师的细心指导以及这份非常详尽的实验文档，祝越来越好。

# 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:** |