华中科技大学

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：	物联网工程
班　　级：	IoT1501
学　　号：	U2014881
姓　　名：	赵浩东
电　　话：	15927259552
邮　　件：	Haydenz0011@outlook.com
完成日期：	2018-04-07 周六下午

计算机科学与技术学院

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	SLLV	可变逻辑左移	
29	SRLV	可变逻辑右移	
30	LH	加载半字	
31	BGEZ	大于等于 0 时跳转	

2 总体方案设计

2.1 单周期 CPU 设计

本次采用的方案是硬布线控制，首先构造主要功能部件，包括 ALU、寄存器组、控制器等，其中最重要的是控制信号的综合，然后构建数据通路，连接各部件。

单周期 CPU 基本沿用上学期组成原理实验的单周期 CPU，在此基础上作了如下修改：

1. 添加了 4 条扩展指令的支持
2. 封装了部分器件，即 RegisterFile, ALU, ALU_in, NPC，便于课设后期改流水线

总体结构图如图 2–1 所示。

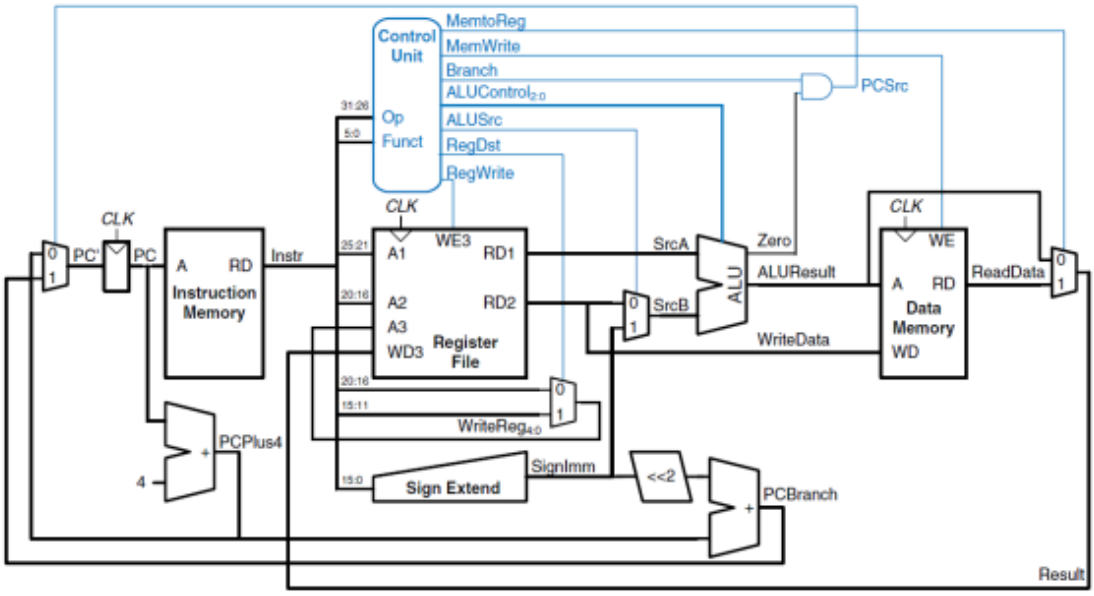


图 2－1 总体结构图

2.1.1 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 2.1。

表 2.1 主控制器控制信号的作用说明

控制信号	取值	说明
ALU_sel [4:0]	00000 -11111	指令对应 ALU 的操作输入 ALUOp
PC-sel	00-11	选择 PC 的下一条指令的地址来源
rw-sel	0/1	0 表示写入寄存器编号为 R2 1 表示写入寄存器编号为指令的第 11-15 位
RF_WE		1 表示寄存器可写入数据 0 表示寄存器不可写入数据
MemtoReg		0 表示无数据需要从内存写入寄存器 1 表示有数据需要从内存写入寄存器
DM-WE		0 表示无数据需要写入到内存 1 表示有数据需要写入到内存
Beq		1 表示当前指令为 beq
bne		1 表示当前指令为 bne
X-sel		1 表示 ALU 的 X 来自寄存器 B，0 表示来自 A
Y-sel		1 表示 ALU 的 Y 来自寄存器立即数，0 表示来 B
shift		1 表示当前指令为移位指令
jal		1 表示当前指令为 jal，寄存器 2 输入及写寄存器输入为第 31 号寄存器，寄存器写入数据为内存输出数据
Syscall		1 表示当前指令为系统调用指令，ALU 的 Y 操作数输入为 10，寄存器组 1 输入选择为 2 号寄存器，寄存器组 2 输入选择为第 4 号寄存器，ALU 比较结果若相同，则数码管显示寄存器 2 输出的内容，否则使时钟信号无效
bgez		1 表示当前指令为 BGEZ
Lh		1 表示当前指令为 LH
slv		1 表示当前指令为 SLLV 或 SRLV
Imm-gates		1 表示当前指令为 ANDI 或 ORI

2.2 中断机制设计

2.2.1 总体设计

设计一个 CR0 协处理器用于处理中断，发出的中断信号直接送交 CR0 处理，CR0 接受数据输入和输出，PC 输入和输出，以实现 MFC0、MTC0 两条指令和 PC 的存取，同时 MTC0、MFC0、ERET 三条指令由 CP0 负责解析，控制器只生成一个判断是否是这三条指令之一的控制信号并输入。为了简化设计，直接将中断程序存放在指令存储器中，将其地址固化在电路中。

2.2.2 硬件设计

输入 CR0 的中断信号通过优先编码器实现硬件响应优先级，CR0 内部有三个寄存器，一个是 EPC，一个可供 MTC0/MF0 指令修改，另一个用于存放当前相应的中断编号。另外，CR0 内部会解析指令来判断是 MTC0/MFC0/ERET 并作出正确处理。

主电路中，直接将中断程序的地址固化。通过多路选择器来进行中断程序的选择。

2.2.3 软件设计

编写的中断程序在一开始应该使用 MTC0 指令将 CR0 中的 1 号寄存器置为 1 以关中断，然后将需要保护的寄存器值压栈再执行中断程序，执行完毕后恢复压入栈中的寄存器的值，然后开中断。

2.3 流水 CPU 设计

2.3.1 总体设计

采用五段流水线设计，分为取指、译码、执行、访存、写回五个阶段，各阶段通过接口部件传递相应数据，包括 PC、指令、控制信号、寄存器读取的值、运算器结果等。

2.3.2 流水接口部件设计

接口部件用于连接两个阶段，接口部件内部存放需要传递的数据对应的寄存器，每个接口部件的输入为传递数据以及 enable 信号、clear 信号和时钟信号，输出为传

递的数据。

2.3.3 理想流水线设计

在单周期 CPU 上做修改，将原来的单周期 CPU 拆分为五个阶段，用四个接口部件连接各个阶段，传递每个阶段或后面的阶段需要的数据，另外，传递给各个部件的控制信号也要注意修改为对应阶段传递过来的控制信号。

2.4 气泡式流水线设计

理想流水线无法解决冲突的问题，所以需要设计一个解决冲突的单元（Hazzard Unit），Hazzard Unit 接受来自 ID 段的 Rs、Rt，EX 阶段和 MEM 阶段的 RW，如果相同且寄存器写入使能端 rw-we 为 1 则表示产生数据冲突，插入空指令（气泡）。如果发生了跳转，即控制冲突，则需清空误读的指令。最后，除寄存器以外的器件均采用时钟上升沿触发，避免结构冲突。

2.5 数据转发流水线设计

气泡流水线通过延缓 ID 段取操作数动作的方式解决数据冲突问题，但大量气泡的插入会严重影响流水性的性能。如果先不考虑 ID 段所取的操作数是否正确，而是等到 EX 段实际需要使用这些操作数时再考虑正确性问题，直接将正确数据从其所在位置重定向到 EX 段合适的位置，这样就可以避免插入气泡引起的流水线性能下降。

修改数据通路和 Hazzard Unit，Hazzard Unit 接受 EX 阶段的 MemtoReg 信号输入，如果该信号为 1，并且产生了 EX 阶段的数据相关冲突，则说明产生了 Load-Use 相关，需要访存，时间过长，不能重定向，则需要插入一个气泡。

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) 运算器 ALU

① Logism 实现：

基于上个学期的实验，ALU 采用内置的功能单元实现，通过多路选择器由 ALU_sel 选择输出结果。为了简化电路，删除了溢出检测功能；为了加快 CPU 运行速度，将自主设计的加法器改为 logism 内置加法器。如图 3－1 所示。

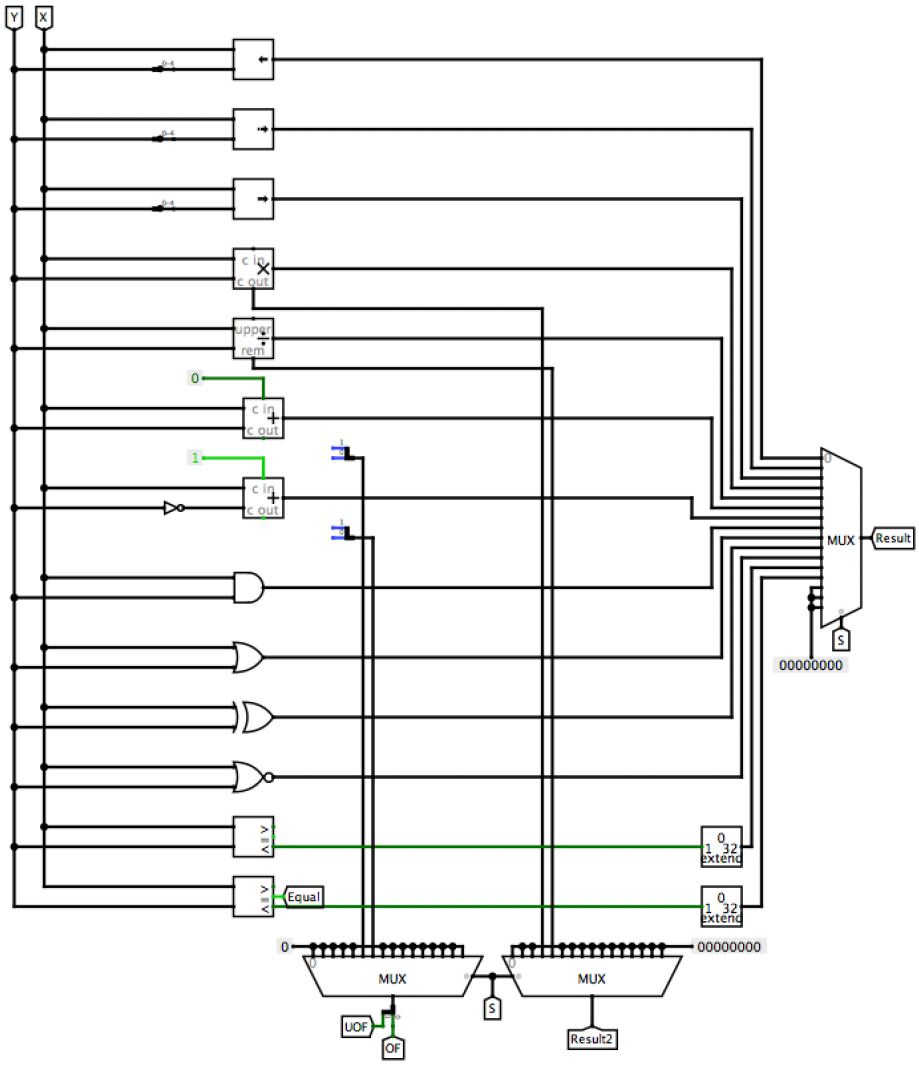


图 3－1 运算器（ALU）

② FPGA 实现：

运算器 ALU 的 Verilog 代码如下：

```
module ALU (
    a, b, S,
    result, result2, equal
);
    input [31:0] a, b;
    input [3:0] S;
    output reg [31:0] result, result2;
    output reg equal;
    reg [63:0] temp;
  initial begin
    result<=0;
    result2<=0;
  end
    always @(*)
    begin
        equal <= (a==b);
      case(S)
          4'b0000:
            result <= a << b;
          4'b0001:
            result <= a >>> b;
          4'b0010:
            result <= a >> b;
          4'b0011:begin
            temp = a*b;
            result <= temp[31:0];
            result2 <= temp[63:32];
          end
```

```
                4'b0100:begin
                    result <= a/b;
                    result2 <= a%b;
                end
                4'b0101:
                     result <= a + b;
                4'b0110:
                     result <= a - b;
                4'b0111:
                     result <= a & b;
                4'b1000:
                    result <= a | b;
                4'b1001:
                    result <= a ^ b;
                4'b1010:
                    result <= ~(a | b);
                4'b1011:
                    result <= (a < b) ? 1 : 0;
                4'b1100:
                    result <= ($unsigned(a) < $unsigned(b)) ? 1 : 0;
            endcase
        end
    endmodule
```

2) Syscall_implementation

① Logism 实现：

若 Rs 指向的内容为 a，则输出 Halt 为 1，使 PC 暂停，当 Rs 指向的内容为 a，且 syscall 为 1 时，将 Rt 指向的内容输出至 syscallout。如图 3－2 所示。

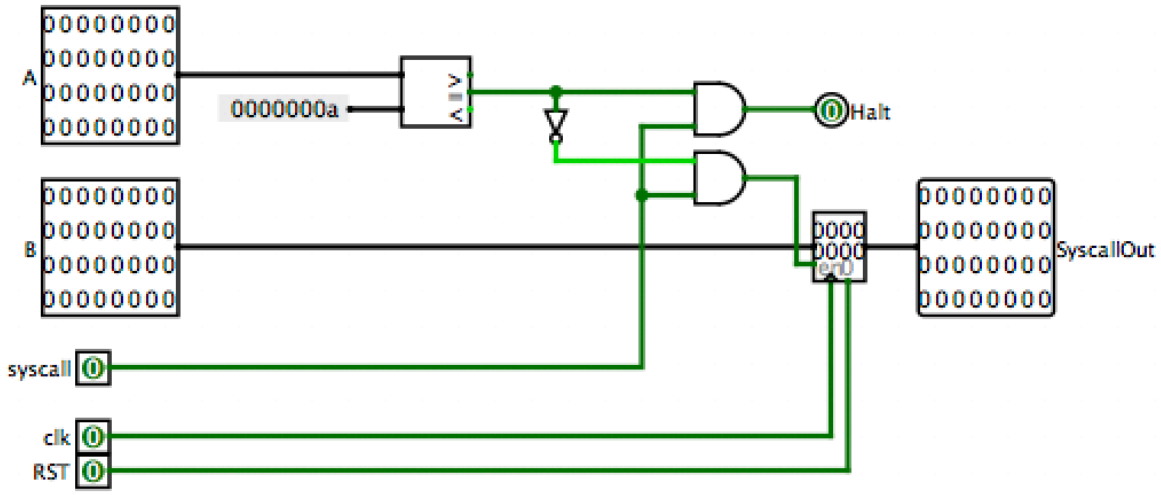


图 3-2 sycall_implementation

② FPGA 实现：

Syscall_implementation 的 Verilog 代码如下：

```
module syscall_implementation(
    A, B, syscall, clk, RST,
    Halt, SyscallOut
);
    input [31:0] A, B;
    input syscall, clk, RST;
    output Halt;
    output reg [31:0]SyscallOut;
    assign Halt = (A == 32'h000a) ? syscall : 0;
    initial
    SyscallOut<=0;
    always @(posedge clk or posedge RST)begin
        if(RST)
            SyscallOut<=0;
        else if((A != 32'h000a) && syscall)
                SyscallOut <= B;
    end
    endmodule
```

3.2 中断机制实现

3.2.1 多级中断

（1）中断信号的产生。

利用实验指导书上的中断信号电路进行修改，实现了 3 级中断信号产生电路，如图 3–3 所示。

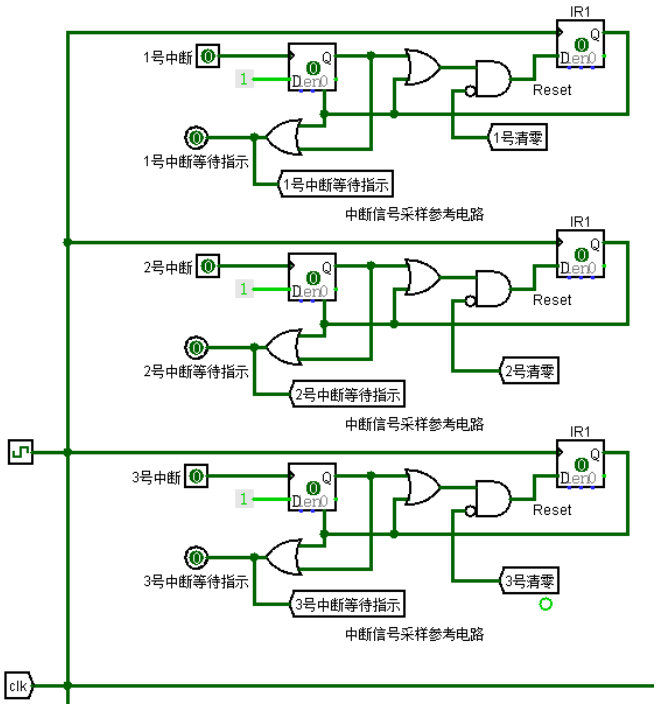


图 3－3

（2）中断屏蔽字的使用

图 3－4 中有一个中断屏蔽字寄存器用于寄存中断屏蔽字，在多级中断中程序需要根据中断屏蔽字判断中断是否可以进行响应，中断屏蔽字和等待指示灯相与后得到了 IR（即是否有需要响应的中断）。

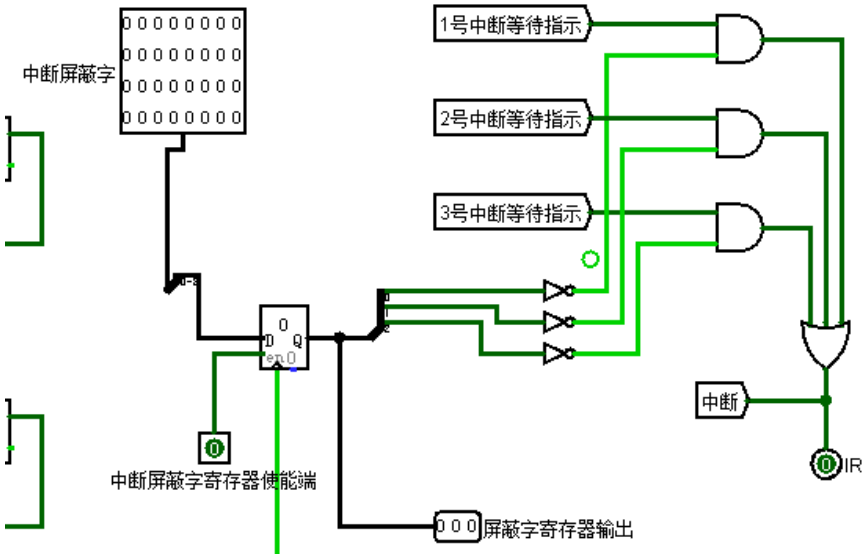


图 3－4 中断屏蔽字

（3）清零信号

图 3－5 表示的清零逻辑的实现，由于中断优先级为 3>2>1,因此 xy 总是显示的是最当前优先级最高的中断，且 eret 所返回的中断也是中断优先级最高的中断。

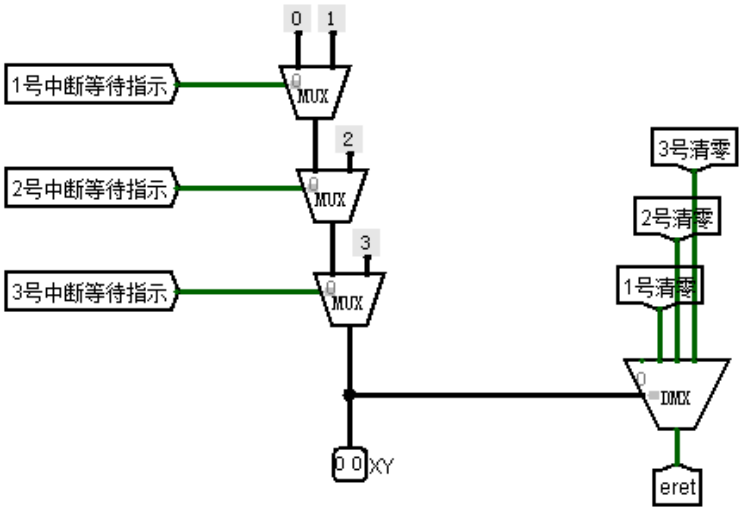


图 3－5 清零信号

（4）IE—开关中断

图 3－6 表示的是 IE 的实现，其中当 IE&IR、eret、中断开关中任意一个为 1 时，一个时钟上升沿来到时，IE 的值就会翻转，以此实现开关中断的功能。其中 IE&IR 代表此时可以进入中断，这时需要关中断，于是 IE 翻转为 0；eret 为中断返回指令，同样需要在关中断状态下开启中断，因此这是 IE 翻转；中断开关由 break 指令控制，用于保护现场后和恢复现场前的开关中断。其中，IE&IR 属于硬件实现，后两种属于软

件实现。

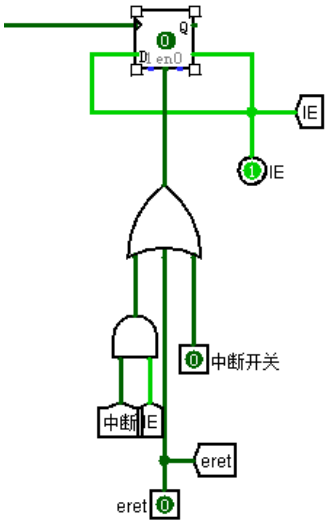


图 3－6 IE 实现

（5）更改控制器

更改控制器实现，使其完成 eret,mtc0,mfc0,break 等指令所需要的控制信号。其中，mtc0,mfc0 所实现的是将寄存器 EPC,中断屏蔽字寄存器与内存之间的内容交互。寄存器的编号由自己设定，在我的程序中，0 号代表 EPC，1 号代表中断屏蔽字寄存器。

（6）保护现场

下面表示的是保护现场的代码段。其中 sw 将需要保护的数据保存到内存中以便后面的使用，同时增加 sp 以维护堆栈指针；mfc0 和 sw 搭配用于保存 0 号以及 1 号我自己定义的寄存器即 EPC 和中断屏蔽字寄存器；同时利用 mtc0 指令将数字 3 保存到中断屏蔽字寄存器中以设置新的中断屏蔽字，因为这段代码为 2 号中断中截取的，因此新的中断屏蔽字为 3，即 011，因此在执行中断服务程序的过程中，只能响应三号中断。一号与二号中断均被屏蔽。；break 利用软件方式打开中断，接下来便是执行二号中断服务程序的内容。sw $s0,0($sp)

```
addi $sp,$sp,4
sw $s1,0($sp)
addi $sp,$sp,4
sw $s2,0($sp)
addi $sp,$sp,4
sw $s3,0($sp)
```

```
addi $sp,$sp,4
sw $s4,0($sp)
addi $sp,$sp,4
sw $s5,0($sp)
addi $sp,$sp,4
sw $s6,0($sp)
addi $sp,$sp,4
sw $s7,0($sp)
addi $sp,$sp,4
sw $v0,0($sp)
addi $sp,$sp,4
sw $a0,0($sp)
addi $sp,$sp,4

mfc0 $s2,$0
sw $s2,0($sp)
mfc0 $s2,$1
sw $s2,4($sp)
addi $sp,$sp,8

addi $s2,$zero,3
mtc0 $s2,$1
break
```

（7）恢复现场

下面表示的是恢复现场。Break 用于执行完中断服务程序后关闭中断用于恢复现场；lw 和 mtc0 搭配用于还原 EPC 和中断屏蔽字寄存器的内容；lw 用于恢复寄存器组的内容；eret 有两个功能：开启中断以及将 EPC 的值赋给 PC 寄存器，继续从断点处执行程序。

```
break
addi $sp,$sp,-4
```

```
lw $s2,0($sp)
mtc0 $s2,$1
addi $sp,$sp,-4
lw $s2,0($sp)
mtc0 $s2,$0

addi $sp,$sp,-4
lw $a0,0($sp)
addi $sp,$sp,-4
lw $v0,0($sp)
addi $sp,$sp,-4
lw $s7,0($sp)
addi $sp,$sp,-4
lw $s6,0($sp)
addi $sp,$sp,-4
lw $s5,0($sp)
addi $sp,$sp,-4
lw $s4,0($sp)
addi $sp,$sp,-4
lw $s3,0($sp)
addi $sp,$sp,-4
lw $s2,0($sp)
addi $sp,$sp,-4
lw $s1,0($sp)
addi $sp,$sp,-4
lw $s0,0($sp)
eret
```

（8）电路图实现

图 3－7 表示的是多级中断 PC 地址选取的逻辑电路部分，根据中断信号以及使能端利用控制器和中断判断的信号，通过多路选择器进行 PC 的选择。

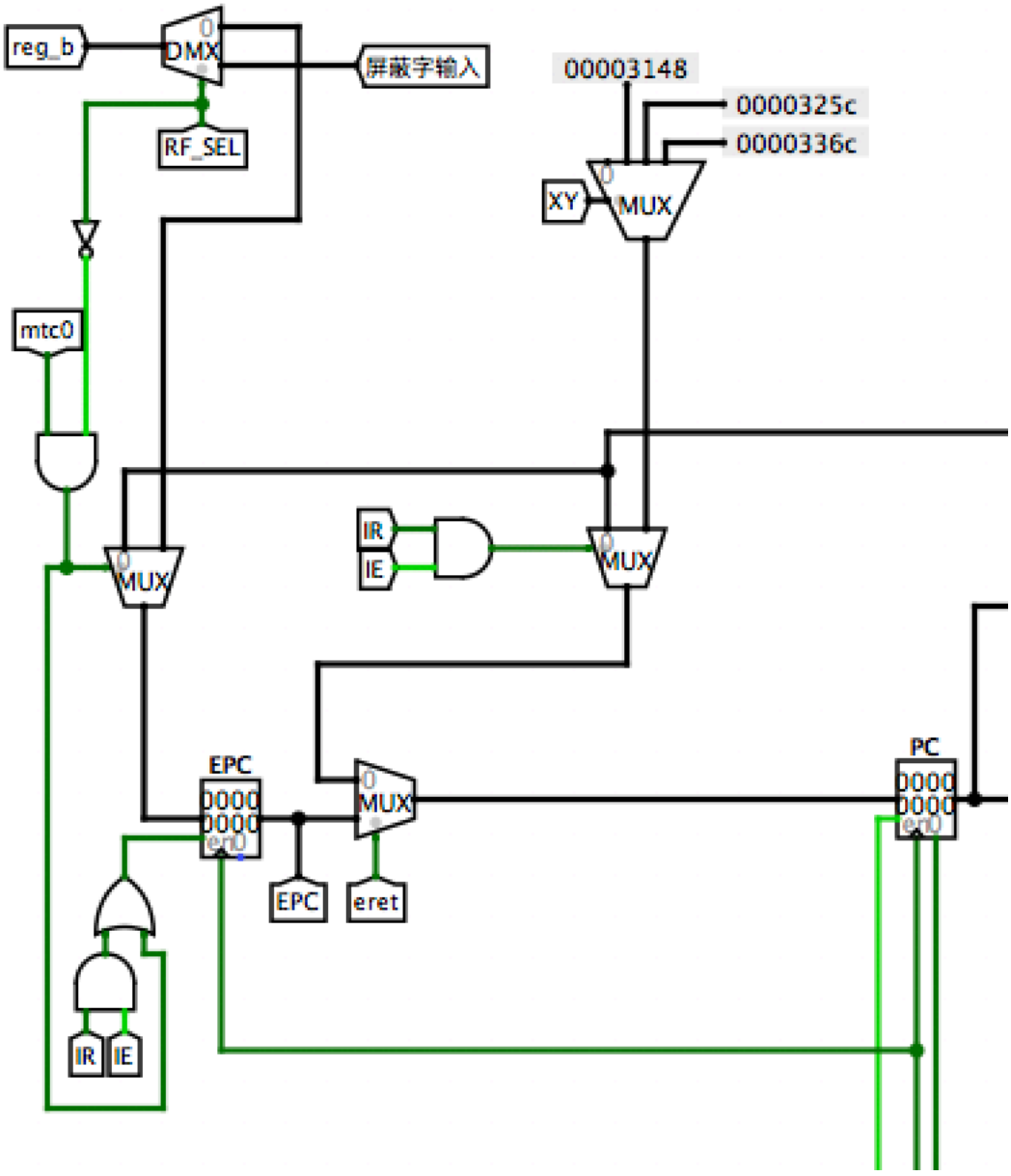


图 3－7 电路图实现

3.2.2 流水单级中断

图 3－8 表示的是重定向流水线沿用单级中断的逻辑电路，只不过流水线单极中断在发生中断时，ID 和 IF 段的指令都会被清零，因此 EPC 需要保护哪条指令的地址至关重要，不能简单的保存 ID 段的 pc 因为跳转指令执行后 ID 和 IF 都会被清空，此时 pc 为 0，如果保存进了 EPC 会导致中断返回时出现错误，因此我们需要对这种情况进行判断，电路图如下：

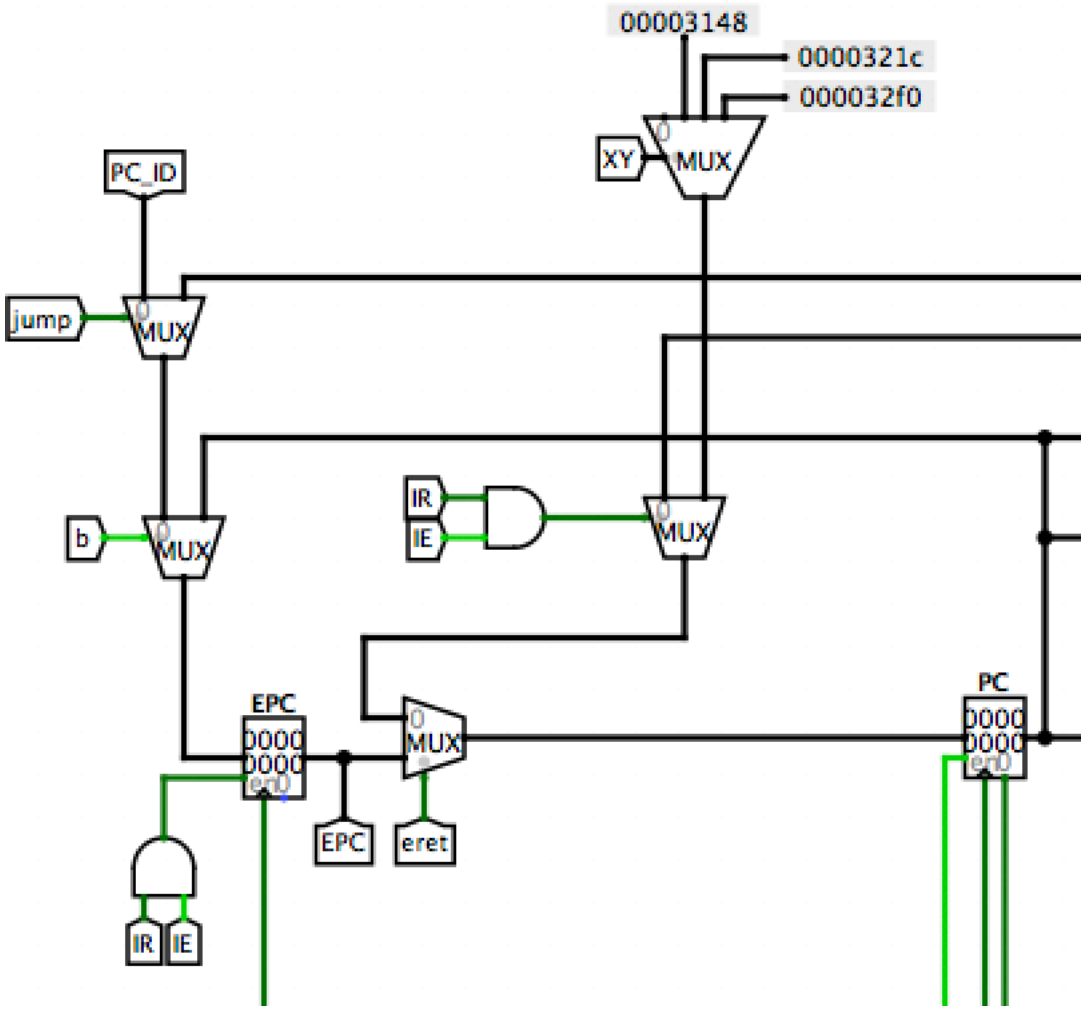


图 3-8 流水中断

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

将需要输入输出的信号接到对应寄存器的输入输出，同时将时钟信号、使能信号、清零信号接到每个寄存器即可。

ID/EX 接口部件还需要传递读取的寄存器值（Reg1、Reg2），写寄存器编号（RW），指令中的立即数（Imm）以及相应控制信号；

EX/MEM 接口部件还需要传递 ALU 结果（ALUResult），寄存器值（Reg2），ALU 的 X 操作数（ALUX），写寄存器编号（RW）以及相应控制信号；

MEM/WB 接口部件还需要传递地址（Addr）和数据（Data），写寄存器编号（RW）

以及相应控制信号。流水接口举例如下：

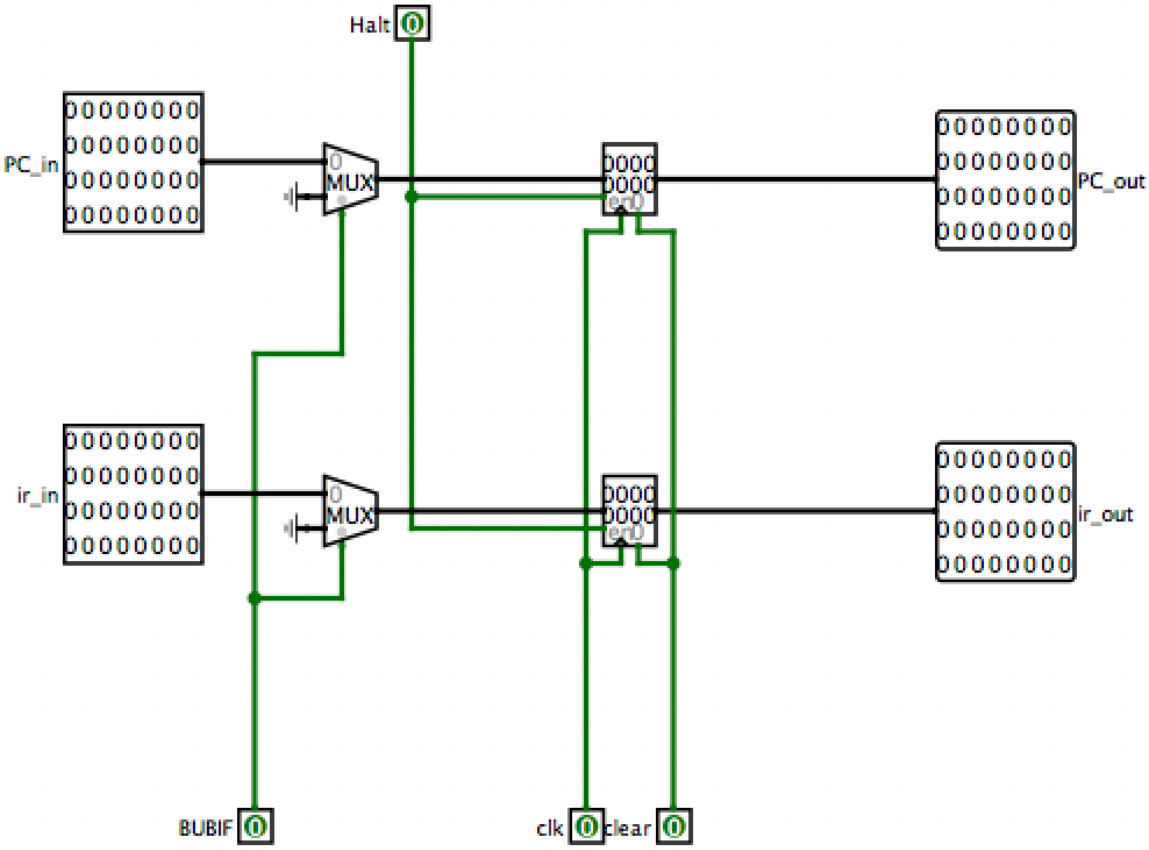


图 3－9 IF/ID 流水接口

3.3.2 理想流水线实现

在单周期 CPU 上做修改，将原来的单周期 CPU 拆分为五个阶段，用四个接口部件连接各个阶段，传递每个阶段或后面的阶段需要的数据，另外，传递给各个部件的控制信号也要注意修改为对应阶段传递过来的控制信号。如图 3－10 所示：

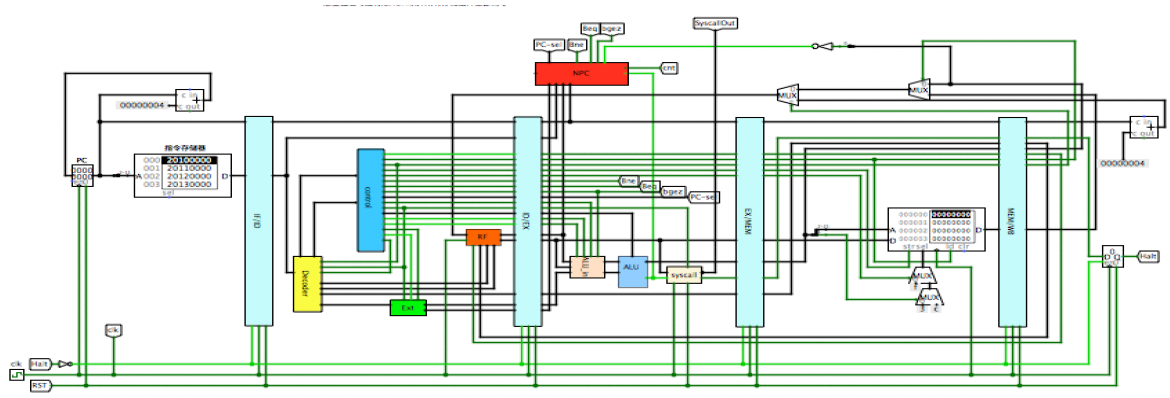


图 3－10 理想流水线

3.4 气泡式流水线实现

Hazzard Unit 若检测到 ID 段源操作数和 EX 段或 MEM 段目的操作数相同则说明发生了数据冲突，这时需要锁住 PC 寄存器以及 IF/ID 接口。如果发生了分支则需要同时插入两个气泡。如图 3－11 所示：

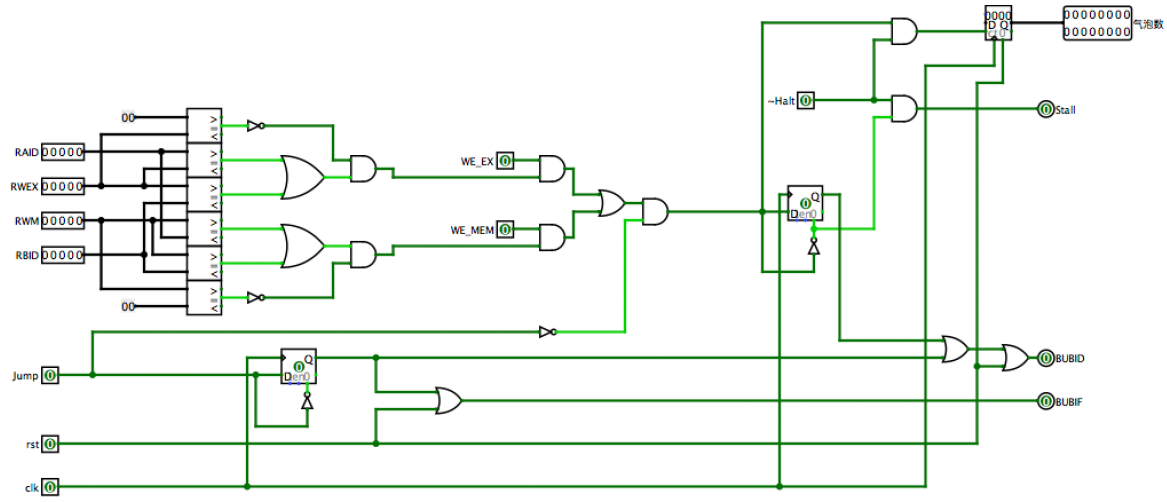


图 3－11 检测数据冲突

3.5 数据转发流水线实现

当产生冲突时，直接重新定位数据，故主电路数据通路部分需要修改，如图 3－12 所示：

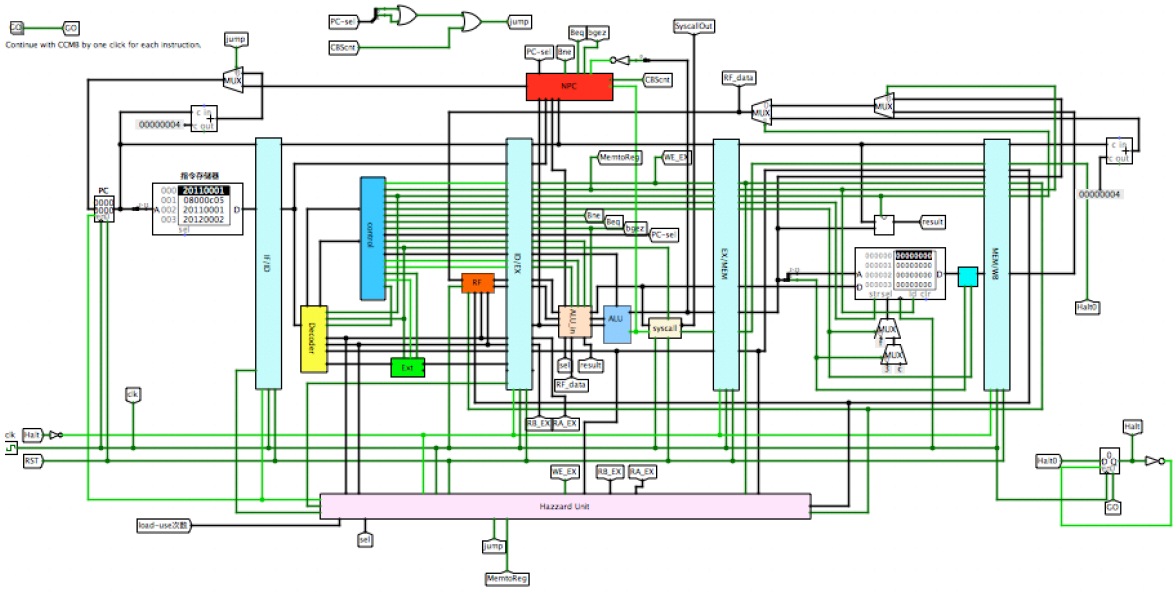


图 3－12 重定向流水线

对于 Hazzard Unit，仅当为 load-use 时才插入气泡，故逻辑修改如图 3－13 所示：

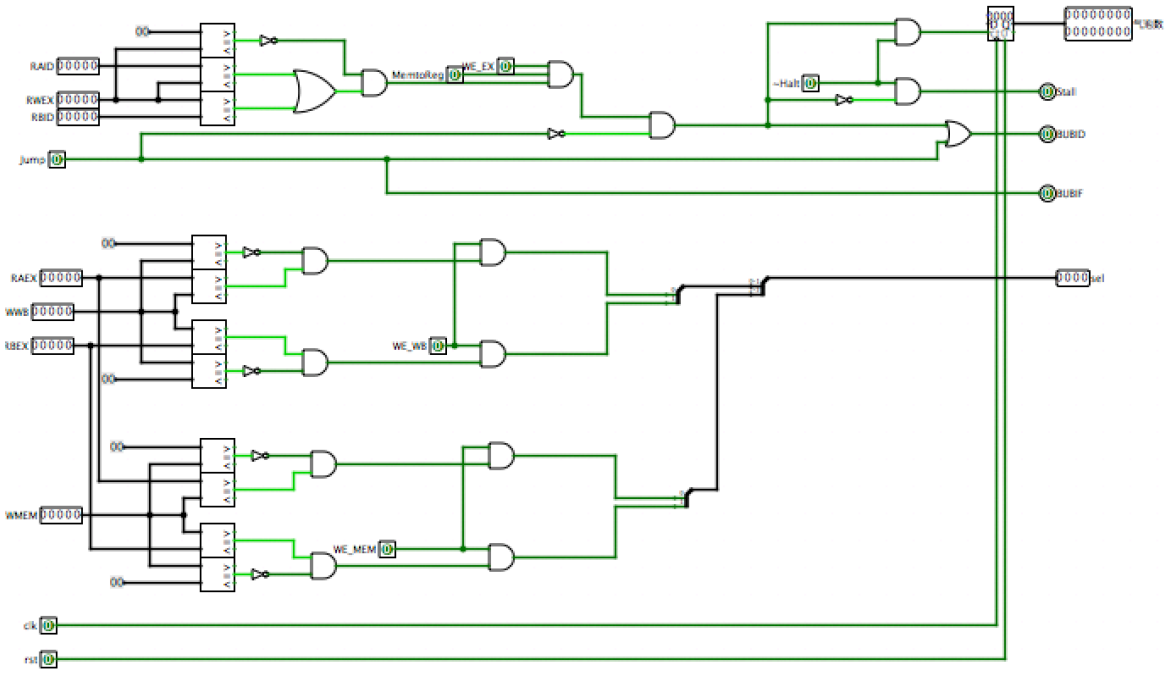


图 3－13 重定向冲突判断

4 实验过程与调试

4.1 测试用例和功能测试

4.1.1 理想流水测试

测试程序如下所示：

```
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
addi $s1,$zero, 0
addi $s2,$zero, 0
addi $s3,$zero, 0
syscall
```

执行测试程序后，可得出在 21 个周期后程序停止，RAM 中的数据也被正确存入。如图 4－1 所示：

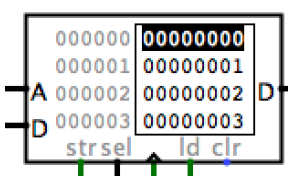


图 4－1 理想流水测试

4.1.2 重定向流水测试

重定向周期数=单周期数+流水冲满时间+J 指令*误取深度+条件分支成功次数*误取深度+LOAD-USE 气泡数。因此重定向跑 benchmark 的正确周期数应为 1546+4+38*2+276*2+120=2298。运行结果如所示：

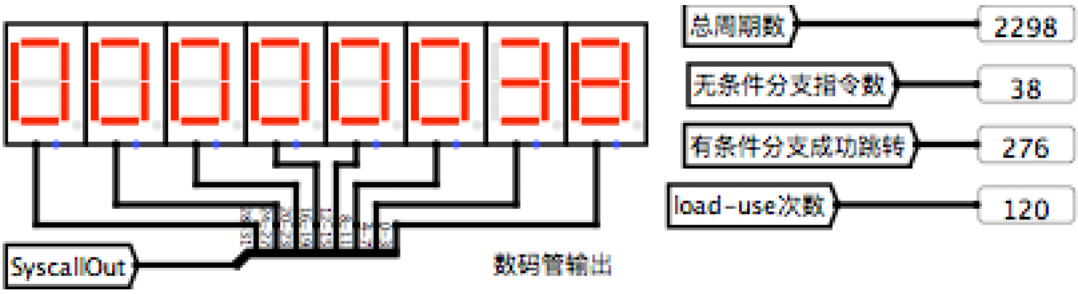
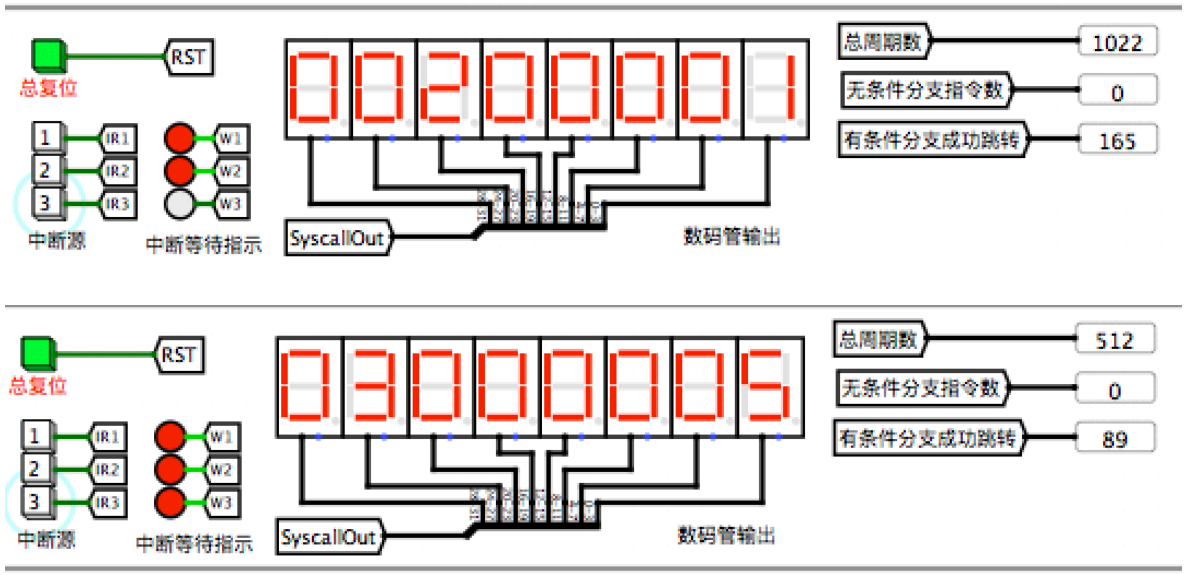


图 4－2 重定向流水测试

4.1.3 多级中断测试

主程序为从 200H 开始的倒数，中断程序 1、2、3 分别为显示 1、2、3 的跑马灯程序，循环次数为 6。中断响应优先级为 1<2<3，中断处理优先级也为 1<2<3。

测试时依次按下 2、1、3 号按钮，首先 2 号程序被 3 号中断，3 号程序执行完后继续执行 2 号，最后执行 1 号程序。结果正确。



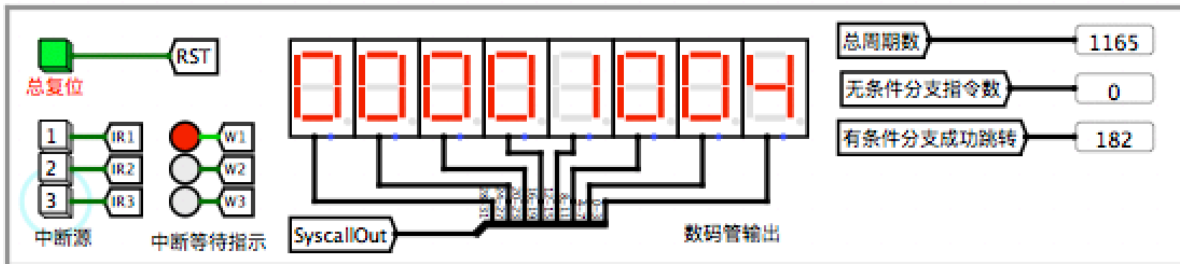


图 4－3 多级中断测试

4.1.4 流水单级中断测试

主程序与中断程序实现效果同上，当依次按下 1、2、3 号按钮时，首先进入 1 号中断程序，程序执行完后执行优先级更高的 3 号程序，最后执行 2 号程序。结果正确。

4.2 性能分析

若 ALU 使用 logisim 自带的加法器，可以发现仿真运行时的频率得到显著提升，从 60Hz 左右提升到 200 多 Hz。

4.3 主要故障与调试

4.3.1 理想流水线周期数错误

理想流水线： 周期数错误

故障现象：周期数不为 21

原因分析：停机指令需要再多经过一个寄存器。

解决方案：从 WB 阶段出来的 Halt 在加上一个寄存器。

4.3.2 气泡流水线错误

气泡流水无法正确运行 benchmark。

故障现象：benchmark 运行途中进入死循环

原因分析：冲突未能完全解决，经过不断排查，发现是寄存器 RF 未设置成下降沿触发，导致结构冲突未能解决。

解决方案：将 RF 更改为下降沿触发。

4.3.3 重定向流水错误

重定向流水无法正确运行 benchmark。

故障现象：运行 benchmark 时进入死循环。

原因分析：指令无法继续执行，可能因为数据通路未能连接完善。经排查发现流水接口内部存在线未能连接的情况

解决方案：将出错模块的内部线连好。

4.4 实验进度

表 4.1 课程设计进度表

时间	进度
第一天	小组完成分工，搭建好了环境（VSCode+iVerilog+Scansion）,完成了 ALU 模块的 verilog 代码编写。
第二天	完成了 ALU、syscall 模块的编写以及各个模块的连接工作
第三天	上板持续失败，debuging
第四天	完成单周期上板，开始着手理想流水线，决定先将单周期电路的各个模块封装重新设计一遍方便未来布线。
第五天	理想流水基本完成，但测试程序跑出来的周期数有问题，经过排查后成功解决，开始着手气泡流水
第六天	气泡流水 debuging，最终发现是 PC 的时钟触发沿错误导致结构冲突无法解决。气泡流水完成。
第七天	按要求在气泡流水中加上 CCMB 的单条执行功能，验收完毕后开始着手重定向。
第八天	重定向电路出线一个低级错误：封装内部连线不完整。最终解决后开始着手中断，准备直接做多级中断，较为困难
第九天	仍在继续多级中断，软硬结合的特点是这个实验的难点
第十天	最终多级中断完成，流水中断较为简单，完成

5 设计总结与心得

5.1 课设总结

本次课程设计完成了以下工作：

1） 通过团队合作将单周期 CPU 在 FPGA 开发板上实现，加深了对 verilog 语言的掌握。

2） 成功设计了五段流水线，其中包括理想流水，气泡流水和重定向流水，完成了多级中断和流水单级中断，加深了对 CPU 工作机制的理解。

3） 提高了使用 logisim 设计电路的能力，以及 verilog 编程能力。

4） 提升了团队合作的能力，对于团队合作开发项目有了一定的认识。

5） 加深了对计算机体系与结构的认知，锻炼了逻辑思维能力和动手能力。

5.2 课设心得

这次的课程设计可谓是大学以来最难的一次，两个星期连续不断地实验让我学到了了很多也成长了很多。

刚开始着手课设的时候有点不知所措，在组长的带领下完成了对 CPU 的 verilog 代码编写，而我只完成了这其中的一小部分；在编写将各个模块连接的代码前，我和另一个小组成员对数据通路的线进行命名，命名方法很不友好，导致后来对编写的速度造成了影响。

在设计理想流水时，我先将之前的单周期 CPU 进行了一定的修改，模块化更加规范和美观，方便了后续流水设计时的拆分。而在到了重定向/气泡阶段，这里因为思路比较清晰，所以直接上手做冒险检测单元，这里遇到了很多小问题，包括有线路不小心连接错误或者未连上、触发器触发方式调整等，花了比较多的时间 debug，但是整个过程思路很清晰，所以虽然时间花的有点长但是可以说还是很顺利的。最后由于时间问题只完成到了中断，也算是比较可惜。但是经过了两个星期的磨炼，我对于 CPU 的结构和机制有了一个更加高层次的理解。

持续两周的课设不可谓不辛苦，每天从早写到晚，就连吃饭洗澡都在想课设的问题。不过我也从一开始的叫苦连连到后面变成乐在其中，毕竟这可以说是一门质量很

高的课程设计，我从中实实在在地学到了很多。而当能从一门课设中真正学到知识的时候，我相信大家都是快乐的。关于课设的建议，我觉得可以精简关于报告的部分，实验报告过于繁琐而且性价比低。

衷心祝愿老师们能将这门课办的越来越好！

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: