转移指令的添加

转移指令有哪些?

BEQ、BNE、BGEZ、BGTZ、BLEZ、BLTZ、BLTZAL、BGEZAL、J、JAL、JR、JALR

添加转移指令的要点

1. 有两个要素：一是决定是否跳转、二是跳转时的跳转目标
2. 指令分为两种：B开头的分支指令和J开头的跳转指令，B开头的分支指令要进行判断，J开头的跳转指令一定跳转

跳转目标的计算方式

1. 转移延迟槽指令PC加上转移指令中的偏移
   由于延迟槽的存在，当转移发生之前，延迟槽中的指令已经被执行。因此需要在延迟槽指令PC的基础上加上转移值，由此获取转移到的地址。
2. 转移延迟槽指令PC高位与转移指令中的偏移拼接
   对于J型指令来说，指令中的目标地址字段只有26位。这26位的偏移量将被左移两位（因为在MIPS中所有指令都是4字节对齐的），然后与当前PC值的最高四位拼接在一起形成新的32位的目标地址。之所以使用当前PC值的最高四位，是因为MIPS假设代码段不会跨越不同的内存区域，所以可以利用当前PC的高几位来确定新地址的基本位置。只要PC值的最高几位保持不变，新计算出的目标地址通常仍然位于同一代码段内。
3. 从通用寄存器获取
   这是指有条件或无条件的寄存器跳转指令(如jr和jalr)。例如，jr $ra会将PC设置为寄存器$ra中的值，常用于过程调用之后返回到调用点。

如何实现

先上代码：

```
module compare(
    input wire [31:0] a,b,      // 输入ab
    input wire [5:0] op,funct,  // 操作码、功能码
    input wire [4:0] rt,        // Register Target 目标寄存器索引 5位是因为刚好有32个通用寄存器
    output wire y
    );

    // a[31]是最高位，检查最高位，最高位为0意味着这个数是一个非负数

    // BGTZ，a是非负数且不为0，所以a大于0
    // BLEZ，同理，a为负数且不为0，所以a小于0

    // REGIMM_INST = 6'b000001
    // rt[0]   用于区分是BGEZ还是BLTZ
    // rt[0] = 1表示是BGEZ，a[31]表示a是非负数
    // rt[0] = 0表示是BLTZ，a[31]表示a是负数

    // R型指令的操作细节由funct给出
    // JR   Jump Register          跳转后不返回
    // JALR Jump and Link Register 跳转后会保存返回地址，执行完毕后可跳转回来，多用于调用
    assign y = ((op == `BEQ) && (a == b)) ||
               ((op == `BNE) && (a != b)) ||
               ((op == `BGTZ) && (a[31] == 0) && (a != 32'b0))   ||
               ((op == `BLEZ) && ((a[31] == 1) || (a == 32'b0))) ||
               ((op == `REGIMM_INST) && rt[0] && (a[31] == 0))   ||
               ((op == `REGIMM_INST) && ~rt[0] && (a[31] == 1))   ||
               ((op == `R_TYPE) && (funct == `JR || funct == `JALR));

endmodule
```

如何理解？

先给要实现的代码分个类：

1. BEQ和BNE

   这两个比较简单，就不说了。

2. BGEZ、BGTZ、BLEZ、BLTZ

   功能定义和BEQ还有BNE比较类似，唯一多了一个就是a[31]的判断，a[31]是用于区分a是非负数和负数的关键点，当判断完a的正负后，判断a是否为0即可。

3. J、JAL

   这两个是J型指令，J用于无条件跳转到指定的目标地址，JAL用于无条件跳转到指定的目标地址，并将返回地址保存到寄存器 $ra（即寄存器31）。

   由于它们只提供26位地址字段，所以实际跳转范围是有限的，并且依赖于当前PC值的一部分来确定完整的32位地址。

   就是上文提到的第二种计算方式。

4. BGEZAL、BLTZAL

   这两个指令就是BGEZ+JAL，BLTZ+JAL，逻辑上等同于BGEZ的实现，无论分支是否跳转，都会将该分支对应延迟槽指令之后的指令的PC值保存至第31号通用寄存器中，就是JAL。因此是BGEZ和JAL的结合。

   BLTZAL同理。

5. JR、JALR

   这两个是R型指令而不是J型指令，因为它们使用寄存器而不是立即数作为目标地址。它们允许根据寄存器中的值进行跳转，提供了更大的灵活性，比如用于实现函数返回或间接跳转。

   JR的跳转目标为寄存器rs中的值。

JALR的跳转目标也为寄存器rs中的值，但是会将该分支对应延迟槽指令之后的指令的PC值保存至寄存器rd中。