No Control Hazard CPU

# 设计流程

控制相关性，是主要针对于跳转指令出现的多执行一条指令而需要解决的问题，就跳转指令而言主要分为两类，第一类为j、jal和jr，这一类指令的跳转地址在译码阶段给出，另一类为beq和bne，这一类指令的跳转地址在执行阶段给出，分别对这两类指令进行分析。

1）j、jal和jr指令

以j指令为例，如图所示，由于这类指令的跳转地址在译码阶段给出，所以不可避免的会流入一条指令，MIPS规定跳转指令都可以有一个时钟周期的延迟槽，这意味着，它允许流入一条指令，如果希望利用延迟槽的话，可以在延迟槽内插入一条不相关指令即可，所以这类指令不存在控制相关性问题。



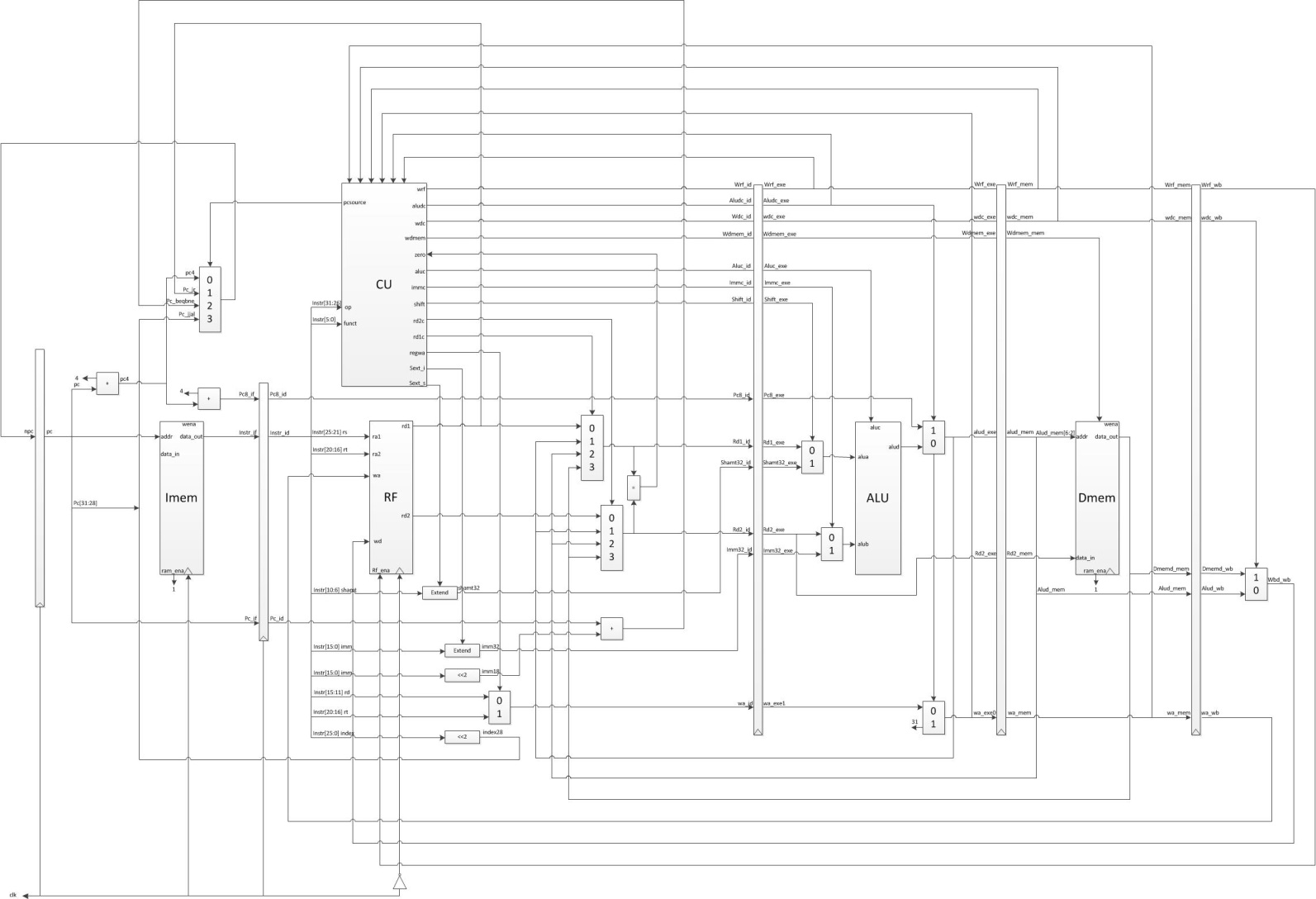
2）beq和bne指令

以beq指令为例，如图1所示，由于这类指令的跳转地址在执行阶段给出，所以不可避免的会流入两条指令，而MIPS规定跳转指令只有一个时钟周期的延迟槽，这就因多流入一条不应该执行的指令而造成控制相关性。对于控制相关性的处理方式为，将在执行阶段处理的跳转控制信号的运算在译码阶段提前做一次，只需要简单的给出跳转控制信号即可，这样可以在译码阶段就得到相应的跳转地址，从而将控制流入指令数量为MIPS允许的一条（即一个延迟槽），这就解决了控制相关性问题，如图2所示。

图1 未处理的beq和bne指令



图2 经过处理的beq和bne指令



# 测试数据：

如图1所示，对于具有数据相关性的汇编指令，如下代码所示，在准pipeline CPU中由于控制相关性，会造成beq或者bne指令多流入一条指令（而j，jal，jr则不会造成这个问题如下图所示2#寄存器修改的指令没有流入），如下图1的4#寄存器所示，理论上不会对这个寄存器进行写操作，而实际上由于控制相关性问题，多流入了一条对4#寄存器修改的指令。当处理了控制相关性之后，跑相同的汇编指令，得到的结果如下图2所示，3#和4#寄存器的值和预期的一致。

j target1

add $0, $0, $0

addi $2,$0, 1

target1:

addi $3, $0, 1

beq $1, $3, target2

add $0, $0, $0

addi $4,$0, 2

target2:

addi $5, $0, 2

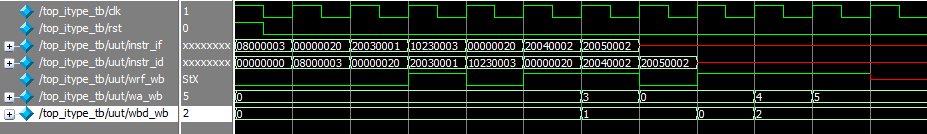


图1没有处理beq和bne控制相关性

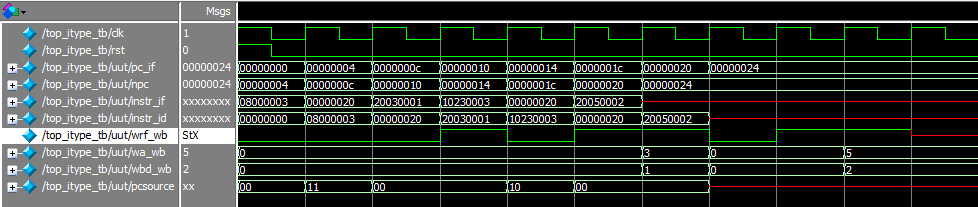


图2处理beq和bne控制相关性

# 注意事项：

修改数据通路图，pc\_beqbne由ID级产生，而不是原来的IF级，为的是beq和bne指令相加的pc其实是beq和bne这条指令的pc，而不是流水线当前的pc值