2.3.6 动态分支预测技术

1.动态分支预测原理

采用重定向机制后，指令流水线中数据相关基本不需要插入气泡就可解决，只有少数 Load-Use 冲突需要插入一个气泡，流水线性能得到较好的提升。此时流水线中的结构冲突对流水线性能影响最大，分支指令会使指令流水线暂停，如果在 EX 段执行分支会使得 IF、ID 段中的误取指令被清空，这部分性能损失称为**分支延迟**。指令预取深度越长，分支延迟越大。为减少分支延迟，应尽可能提前执行分支指令，比如将分支指令放在 ID 段完成，这样指令预取深度为 1，分支延迟为一个时钟周期。

进一步降低分支延迟的主要方法有**静态分支预测**与**动态分支预测**两种。静态分支预测主要是基于编译器的编译信息对分支指令进行预测，预测信息是静态的不能改变，与分支的实际执行情况无关，通常是采用一些简单的策略进行预测或处理，具体如下：

预测分支失败：这是缺省逻辑，与无分支预测的指令流水方案相同。

预测分支成功：这个逻辑效果和第一种应该差不了太多。

延迟分支：由编译器将一条有用的指令或空指令放在分支指令后，作为分支指令的延迟槽，不论分支指令是否跳转，都要按顺序执行延迟槽指令，如果分支指令放在 ID 段执行，延迟槽技术可以完全消除分支延迟，但这需要编译器进行指令调度，取决于编译器能否将程序中的有效指令放入延迟槽且不影响程序功能，对编译器有较高的要求。

动态分支预测依据分支指令的分支跳转历史，不断的对预测策略进行更新，具有较高的预测准确率，存放分支跳转历史统计信息的逻辑必须用硬件实现，现代处理器中均支持动态分支预测算法。分支行为之所以可以预测是因为程序中分支指令的分支局部性，比如 while 循环生成汇编代码第一条判断 while 表达式条件的分支指令只有在最后一次循环时进行跳转，其他时间全部是不跳转；而 do while 循环正好相反，大多时候跳转，一次不跳转；For 循环生成的分支指令也有类似的局部性行为，动态分支预测正是利用了分支指令的分支局部性进行预测。

最简单的动态分支预测策略是分支预测分支历史表 (Branch History Table，BHT)，也称为分支预测缓冲器（Branch Prediction Buffer)，BHT 表用于存放分支指令的历史分支统计信息，每个表项主要包括 valid 位、分支指令地址，分支目标地址，历史跳转信息描述位（预测状态位）、置换标记五项，如表 2.3 所示，其中 valid 位用于标记当前表表项是否有效。BHT 表本质上是一个 cache，通常以全相联的方式进行组织，表项为 8 或者 16 项。BHT 表体积较小，存放的表项都是经常访问的分支指令，通常是程序中循环体的对应的分支指令。

每一条分支指令执行时，会将分支指令地址、分支目标地址、是否跳转共三项信息送 BHT 表，BHT 表以分支指令地址为关键字，在 BHT 表内进行全相联并发比较，如果数据缺失，表

示当前分支指令相关信息不在 BHT 表中，需要将该分支指令的相关信息载入 BHT 表中，并设置合适的分支预测历史位初值，以方便后续预测。注意载入过程中可能涉及到淘汰策略。如果有相符的表项，数据命中，表明当前分支指令历史分支信息已存放在 BHT 表中，这时需要根据本次分支是否跳转的信息调整对应表项的分支预测历史位，以提升预测准确率，并且处理 cache LRU 置换标记信息即可。

表 2.3　分支历史表 BHT 格式

#	Valid	分支指令地址	分支目标地址	分支预测历史位	置换标记
0	1（高电平有效）	xxxx	xxxx	11 预测跳转	xxxx
1	1	xxxx	xxxx	10 预测跳转	xxxx
2	1	xxxx	xxxx	01 预测不跳转	xxxx
3	1	xxxx	xxxx	00 预测不跳转	xxxx
4	0（无效）	xxxx	xxxx	xxxx	xxxx
5	0	xxxx	xxxx	xxxx	xxxx
6	0	xxxx	xxxx	xxxx	xxxx
7	0	xxxx	xxxx	xxxx	xxxx

分支预测历史位本质上是当前分支指令历史跳转情况的统计信息，是进行分支预测的依据，最早分支预测历史位仅采用一位数据表示，为 1 表示预测跳转，为 0 表示预测不跳转。科学研究表明，双预测位可在较低的成本下实现很高的预测准确率，所以现在普遍采用双位预测，一个典型的双位预测位状态转换图如图 2.29 所示。当状态位为 00、01 时预测不跳转，为 10、11 时预测跳转。当前分支指令是否进行跳转将会决定状态的变迁。注意，预测位初始值的设置也很重要，对于无条件分支指令，初始值如果是 00，则预测失败次数是 2 次，实际设计时应适当动态调整预测位初始值。

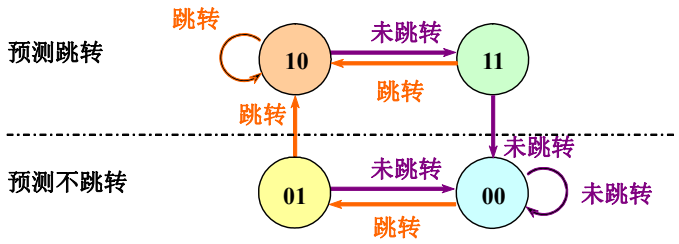

图 2.29 双预测位状态转换图

BHT 表会放在取指令 IF 段，利用 PC 的值作为关键字进行全相联比较，此过程应与指令存储器取指令操作并发。BHT 表命中表示当前指令是分支指令，可以根据 BHT 表中当前指令的历史预测位决定下条指令的地址是 PC+4 还是 BHT 表中的分支目标地址，注意这个分支目标地址不能在 IF 段取指令后计算，而是 BHT 表中的 BHT 表项提供的。

如果 BHT 表缺失，表明当前指令可能不是分支指令或者是不经常使用的分支指令，则按照 PC+4 取下条指令。BHT 表的引入使得 IF 段指令并未取出的情况下进行分支预测，由于双

位预测的高准确率，可以消除指令流水线中的大多数的分支延迟。

注意分支指令在 EX 段执行时，如果 IF 段预测正确，指令流水线不会停顿，如果预测失败，则分支指令在实际执行时还是应该清空预取的指令，并重新修正 PC 地址取出正确的指令。

2.动态分支预测硬件实现

动态分支预测逻辑必须用硬件实现，内部是一个全相联的 cache 结构，其主要输入输出引脚及功能说明如表 2.4 所示，PC 为 IF 段程序寄存器 PC 的输出值，IF 段利用 PC 值在 BHT 表中进行全相联比较，一旦数据命中，根据对应表项中的分支预测历史位的值输出 PredictJump 控制位，预测历史位为 10、11 时 PredictJump=1，表示程序要跳转执行，同时 BHT 表还要输出该指令的分支目标地址。如未命中，当前指令无法预测下条指令的地址，PredictJump=0，程序顺序执行。PredictJump 最终用于选择顺序地址 PC+4 与分支目标地址中的一路送 PC 寄存器输入端，这一部分逻辑由 ID 段执行，不会改写 BHT 表中的内容，是组合逻辑电路。

表中带 EX 前缀的四个输入都是流水线 EX 段反馈回来的数据和操作控制信号，是 BHT 表的写入逻辑，是时序组合逻辑，由 EX 段负责。当 EX 段执行分支指令，也就是 EX.Branch=1 时，BHT 表会根据 EX.PC 全相联查找，如数据缺失要将当前分支指令的信息载入 BHT 表，如果 BHT 表已满，还需要进行淘汰置换；如果数据命中，则只需要根据 EX 段指令实际分支跳转情况 EX.Branched 的值依照预测位状态机更新分支预测历史位的值。

表 2.4　分支历史表 BHT 逻辑引脚说明

#	Valid	I/O 类型	位宽	功能说明
1	CLK	输入	1	时钟控制信号，BHT 表载入新表项或更新预测位需要时钟配合
2	PC	输入	32	程序计数器地址，是在 BHT 表中全相联查找的关键字
3	EX.Branch	输入	1	EX 段分支指令译码信号，1 为分支指令，分支指令才会更新 BHT
4	EX.Branched	输入	1	EX 段分支指令是否跳转，1 为跳转，0 为不跳转
5	EX.PC	输入	32	EX 段指令对应的 PC 地址
6	EX.BranchAddr	输入	32	EX 段分支指令的分支目标地址
7	PredictJump	**输出**	1	预测跳转位，为 1 表示预测跳转，向右通过流水接口依次传输给 EX 段
8	JumpAddr	**输出**	32	预测跳转为为 1 时输出 ID 段跳转指令的分支目标地址

图 2.30 为动态分支预测 BHT 逻辑的具体实现，注意图中指令存储器取指令和 BHT 分支预测逻辑是并发工作的，只需要知道指令的 PC 地址即可进行分支预测，BHT 表的引入并不会增加 IF 段的关键延迟。图中粗线为多位宽的数据，细线为操作控制信号。当 EX 段从段首的 ID/EX 接口接收到从 IF 段传送来的预测跳转信号 PredictJump 时，当实际跳转和预测跳转不符时就会生成预测失败 EX.PredictErr 信号，该信号反馈会 IF 段控制 PC 输入端多路选择器重新取指令，具体取那一个地址的指令由 EX.Branched 选择，当 EX.Branched 为 1 时应该选择从 EX.BranchAddr 处取指令，否则从 EX.PC+4 处取指令。

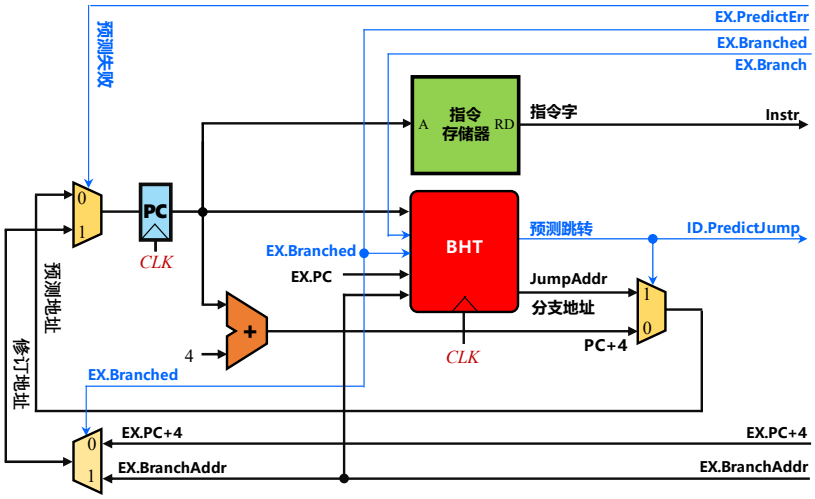

图 2.30　动态分支预测 BHT 逻辑实现

将图 2.30 的硬件逻辑加入 MIPS 指令流水线中就得到图 2.31 所示的支持动态分支预测的五段 MIPS 流水线，注意新的流水线数据通路中 IF 段增加了一个 PredictJump 信号需要通过流水接口部件逐级传递到 EX 段，并最终与分支信号 Branched 进行比较，不相等时表示预测失败。由于 EX 阶段只有检测到分支预测错误才会清空流水线中的误取指令，所以这里将预测错误信号 PredictErr 接入相关处理逻辑中原 Branched 引脚即可复用原电路，其他逻辑并没有任何变化。