当包含流水线技术的处理器处理分支指令时就会遇到一个问题，根据判定条件的真/假的不同，有可能会产生转跳，而这会打断流水线中指令的处理，因为处理器无法确定该指令的下一条指令，直到分支执行完毕。流水线越长，处理器等待的时间便越长，因为它必须等待分支指令处理完毕，才能确定下一条进入流水线的指令。

分支预测技术便是为解决这一问题而出现的。

分支预测技术包含编译时进行的静态分支预测和硬件在执行时进行的动态分支预测。

静态分支预测:

最简单的静态分支预测方法就是任选一条分支。**1）认为branch一定会token；2）认为branch一定不会token；**这样平均命中率为**50%**。更精确的办法是根据原先运行的结果进行统计从而尝试预测分支是否会跳转。

任何一种分支预测策略的效果都取决于该策略本身的精确度和条件分支的频率。

SPARC与MIPS的最早实现（作为第一代商用RISC体系结构处理器）使用单方向静态分支预测：总是预测条件跳转不发生，因此总是顺序取下一条指令投机执行。仅当条件跳转指令被求值确实发生了跳转，则非顺序的代码地址被加载执行。两种CPU都是在解码阶段评价分支指令，取指令占用1个周期。因此分支目标需要两个周期（即经过了去指令、解码两个周期）才能确定。两种处理器都会在分支指令进入流水线的执行阶段时，插入一个分支延迟间隙。分支指令完成流水线的执行阶段，就已经能确定是否跳转，这时就可以决定是后续的顺序出现的指令被继续执行还是跳转到的新指令进入流水。

更复杂的静态预测假定向后分支将会发生，向前的分支不会发生。向后分支，是指跳转到的新地址比当前地址要低。这有助于配合经常出现的程序的循环控制结构。

一些处理器允许分支预测提示出现在代码中。Intel Pentium 4就是如此。但这一特征在Intel此后的处理器中不再支持。

静态预测也用于某些处理器分支动态预测没有任何可用信息时的一个最后的办法。Motorola MPC7450

动态分支预测:

动态分支预测是近来的处理器已经尝试采用的的技术。最简单的动态分支预测策略是分支预测缓冲区（Branch Prediction Buff)或分支历史表(branch history table)。

BHT——Branch History Table，顾名思义，这是记录分支历史信息的表格，用于判定一条分支指令是否token；这儿记录的是跳转信息，简单点的，可以用1bit位记录，例如1表示跳转，0表示不跳转，而这个表格的索引是指令PC值；考虑在32位系统中，如果要记录完整32位的branch history，则需要4Gbit的存储器，这超出了系统提供的硬件支持能力；所以一般就用指令的后12位作为BHT表格的索引，这样用4Kbit的

一个表格，就可以记录branch history了。当然，通过大伙的不懈努力和分析，发现在BHT中用1bit位记录分支是否跳转还不够准确，用2bit位记录就非常好了，而用3bit或者更多位记录，效果与2bit类似。所以在BHT中，一般就用2bit位记录分支是否跳转：例如11和10表示这条分支会跳转；01和00表示分支不会跳转。这个2bit计数器大伙叫做饱和计数器。

BTB——用于记录一条分支指令的跳转地址，由于这儿存储的是指令地址，例如32位地址，因此，这个表格就不能做到存储BHT那样多的内容了，如果也支持4K条指令，则需要128Kbit的存储空间，这几乎可以赶上一个L1Cache的容量了，所以BTB一般很小，就32项或者64项。由于这个BTB容量小，并且其用于是记录分支指令的跳转地址，因此，如果这条指令不跳转，即其下一条指令就是PC+4，则不会在BTB中记录的。

基于BTB和BHT的分支预测就很简单了：

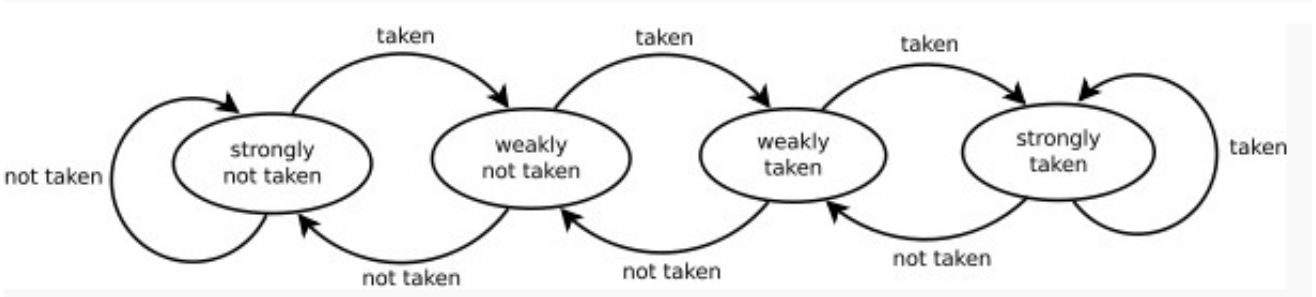
1）在取指阶段利用PC寻址BTB，如果命中，则说明这是一条跳转指令，利用从BTB中获取到的地址去取icache；

2）由于BTB中保存的内容不够多，因此BHT的准确率更高，这个时候索引BHT表格，如果发现BHT也跳转，则说明这条指令预测是跳转的；如果BHT不跳转，则说明不跳转，这个时候就取消BTB中的指令地址，重新PC+4去取icache；

一些典型预测器介绍：

饱和计数

饱和计数其（saturating counter）或者称双模态预测其（bimodal predictor）是一种有4个状态的状态机：



2位饱和计数器

- 强不选择Strongly not taken
- 弱不选择Weakly not taken
- 弱选择Weakly taken
- 强选择Strongly taken

当一个分支命令被求值，对应的状态机被修改。分支不采纳，则向“强不选择”方向降低状态值；如果分支被

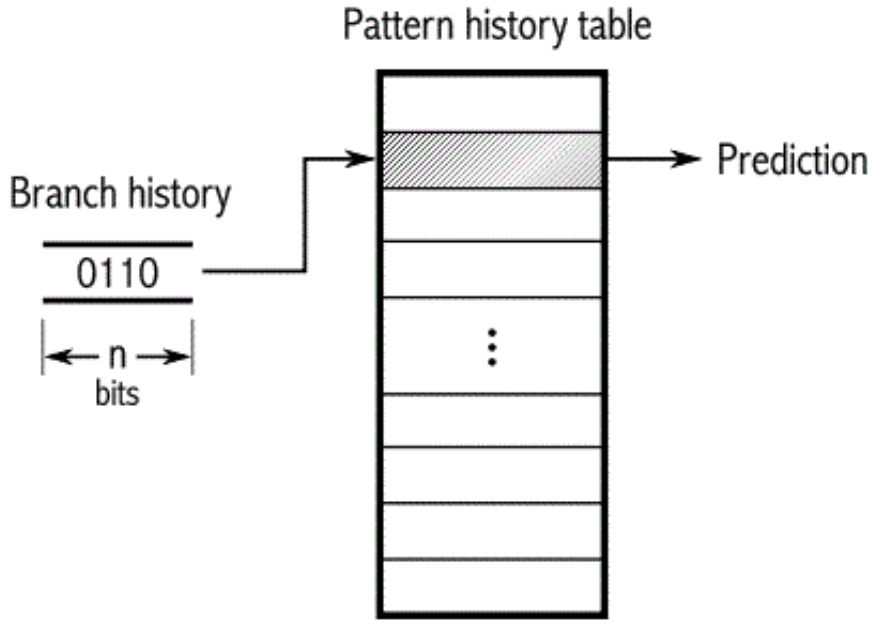
采纳，则向“强选择”方向提高状态值。这种方法的优点是，该条件分支指令必须连续选择某条分支两次，才能从强状态翻转，从而改变了预测的分支。

最初的不具有MMX的Intel Pentium处理器使用了这种饱和计数器。虽然实现不够完美。

在SPEC'89 benchmark测评中, 饱和预测达到了93.5%正确率，如果每条条件分支指令都映射了自己的计数器。

预测器表使用条件分支指令的地址作为索引。因此处理器可以在分支指令解码前就给它分配一个预测器。

两级自适应预测器：



两级自适应预测器。pattern history table中的每个条目是上文的2位饱和计数器。

对于一条分支指令，如果每2次执行发生一次条件跳转，或者其它的规则发生模式，那么用上文提到的饱和计数器就很难预测了。如图所示，一种二级自适应预测器可以记住过去n次执行该指令时的分支情况的历史，可能的2^n种历史模式的每一种都有1个专用的饱和计数器，用来表示如果刚刚过去的n次执行历史是此种情况，那么根据这个饱和计数器应该预测为跳转还是不跳转。

例如，n = 2。这意味着过去的2次分支情况被保存在一个2位的移位寄存器中。因此可能有4种不能的分支历史情况：00, 01, 10, 11。其中0表示未发生跳转，1表示发生了分支跳转。现在，设计一个模式历史表（pattern history table），有4个条目，对应于2^n= 4种可能的分支历史情况。4中历史情况的每一种都在模式历史表对应于一个2位饱和计数器。分支历史寄存器用于选择哪个饱和计数器供现在使用。如果分支历史寄存器是00，那么选择第一个饱和计数器；如果分支历史寄存器是11，那么选择第4个饱和计数器。

假定，例如条件跳转每隔2次执行就发生一次，即分支情况的历史串行是001001001...。在这种情况下，00对应的饱和计数器将是状态“强选择”（strongly taken），表明在两个0之后必然是出现一个1。01对

应的饱和计数器将是状态“强不选择”（strongly not taken），表示在01之后必然是出现一个0。这也同样适用于10状态。而11状态从未使用，因为不可能出现连续两个1。

2级自适应预测器的一般规则是n位分支历史寄存器，可以预测在所有n周期以内出现的所有的重复串行的模式。

2级自适应预测器的优点是能快速学会预测任意的重复模式。此方法1991年被提出。 已经变得非常流行。以此为基础很多变种方法被用于现代微处理。

本地分支预测：

本地分支预测（local branch predictor）对于每个条件跳转指令都有专用的分支历史情况缓冲区；模式历史表可以是专用的，也可以是所有条件分支指令共用。

Intel Pentium MMX, Pentium II, Pentium III使用本地分支预测器，记录4位的历史情况，每条条件跳转指令使用专用的本地模式历史表，当然是包含2^4= 16个条目。

对SPEC'89 benchmark测评，非常大的本地预测器的正确率达到97.1%。

全局分支预测：

全局分支预测器（global branch predictor）并不为每条条件跳转指令保持专用的历史记录。相反，它保持一份所有条件跳转指令的共用的历史记录。优点是能识别出不同的跳转指令之间的相关性。缺点是历史记录被不相关的不同的条件跳转指令的执行情况稀释了（diluted）；甚至历史记录没有一位是来自同一个分支指令，如果有太多的不同的分支指令。

这种方法只有在历史缓冲区足够长，才能发挥出性能。但是模式历史表的条目数是历史缓冲区位数的指数量级。因此只能是在所有的条件跳转指令间共享这个大的模式历史表。

AMD的CPU，以及Intel的Pentium M, Core，Core 2使用了此种方法。

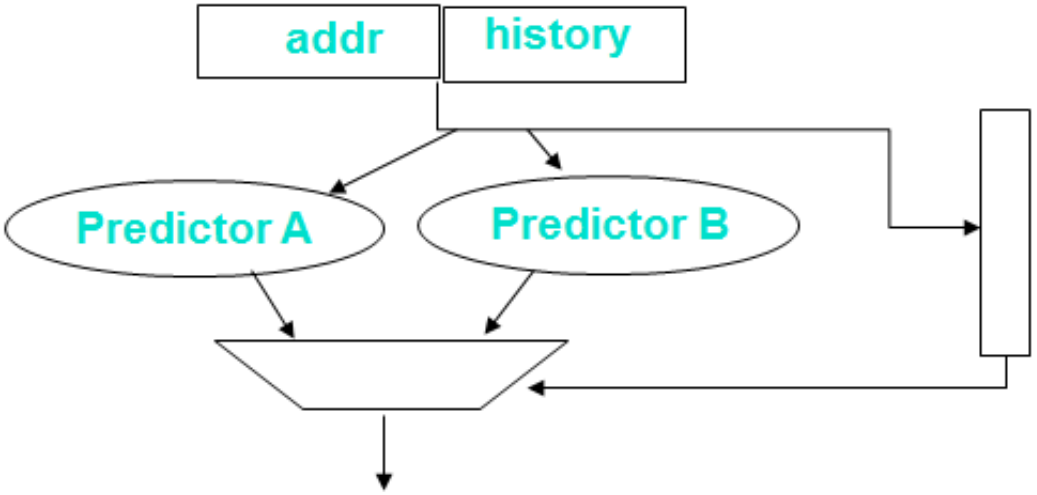
SPEC'89 benchmark评测，非常大的gshare预测器达到了96.6%正确率，略低于本地分支预测。

融合分支预测：

融合分支预测器（alloyed branch predictor）组合了本地与全局预测原理，把本地与全局的分支历史情况连接（concatenating）起来。VIA Nano处理器可能采用此方法。

Tournament预测器：整体局部自适应预测器

Tournament预测器通过使用多个预测器-----其中一个基于全局信息，另一个基于局部信息，通过一个选择器将二者结合，而将这种方法又向前推进了一步。



Tournament预测器的先进性在于，它可以为特定的转移选择正确的预测器，而这一点对于定点基准测试程序来说十分关键。Alpha的Tournament预测器由局部转移地址索引的4K个2bit计数器，在全局预测器和局部预测器间进行选择。

全局预测器共有4K个入口，由最近执行的12个转移进行索引：其中每个入口为一个标准2bit预测器。局部预测器由两层组成。上层是由1024个10bit入口组成的局部历史表；每一个10bit入口对应这个转移最近10次的执行情况。

局部历史表中被选中的入口用来对一个表进行索引，这个表由1K个入口组成，每个入口为3bit饱和计数器，该计数器用来进行局部预测。这种组合共使用了4K*2 +4K*2+1K*10 + 1K*3=29K bit。错误率在1.6%~4%之间。

Agree预测器：

Agree预测器是一种两级自适应全局预测器，但是附加了一个bias饱和计数器，用来记录历次预期的准确度。最终输出结果是全局预测与bias饱和计数器的异或。Intel Pentium 4使用了此种方法，但此后的Intel处理器放弃了此种方法。

循环预测器：

程序循环的控制部分所对应的条件跳转指令最好用专门的循环控制器来预测分支。将要重复N次的循环的底部的条件跳转指令，前N-1次选择跳转，只有一次不跳转而是顺序执行。条件跳转指令有很多次选择了一条分支，只有一次选择另一分支，这种行为将被作为循环行为被检测。这种条件跳转指令可以用简单的计数器很容易地检测出来。循环预测器是一种混合预测器，其中元预测器判断该条件跳转指令是否具有循

环行为。许多微处理器现在都有循环预测器。

间接跳转预测：

间接跳转指令（indirect jump）是指操作数不是要转去的下一条指令地址（这是直接跳转），而是一个存储位置（寄存器或者内存），这个存储位置的内容才是要转去的下一条指令地址。例如`je EAX`，表示在ZF标志寄存器为1情况下，跳转到寄存器EAX所保存的内存地址开始的代码快。

间接跳转指令可以选择多于两条分支的执行路线。Intel与AMD的更新的处理器使用两级自适应预测器能预测间接跳转。间接跳转指令在历史缓冲区贡献了超过1位的表示。没有这种预测机制的处理器，就简单地预测间接跳转指令是否会如同上次一样选择同一目标。

函数返回的预测：

许多处理器有专门用于表示函数调用返回地址的return stack buffer。

overriding分支预测：

快速与精确，是分支预测需要平衡的两个性能指标。有时就设置两个分支预测器。第一个是简单且快速。第二个是更慢、更复杂、表更大、可以覆盖第一个预测器作出的预测。

Alpha 21264与Alpha EV8处理器使用一个快速的、单周期的下一行（next line）预测器处理分支目标，提供一个简单快速的分支预测。两个微处理器都有更复杂的、两个周期的分支预测其，可以覆盖下一行预测器的不准确的结果。

Intel Core i7有两个分支目标预测器（Branch target predictor），可能有两个或更多分支预测器。

神经分支预测器：

1999年提出的神经分支预测器（neural branch predictor）。突出优点是能够利用很长的历史记录，仅导致了资源的线性增长。而传统预测器的资源需要量与历史长度是指数增长关系。这种方法主要缺点是高延迟。

神经分支预测器的准确度非常突出。（参见Intel's "Championship Branch Prediction Competition" ）。Intel在IA-64的模拟器 (2003)中实现了这一方法。

其它总结性文献可参考论文：《处理器分支预测研究的历史和现状》———冯子军 肖俊华 章隆兵