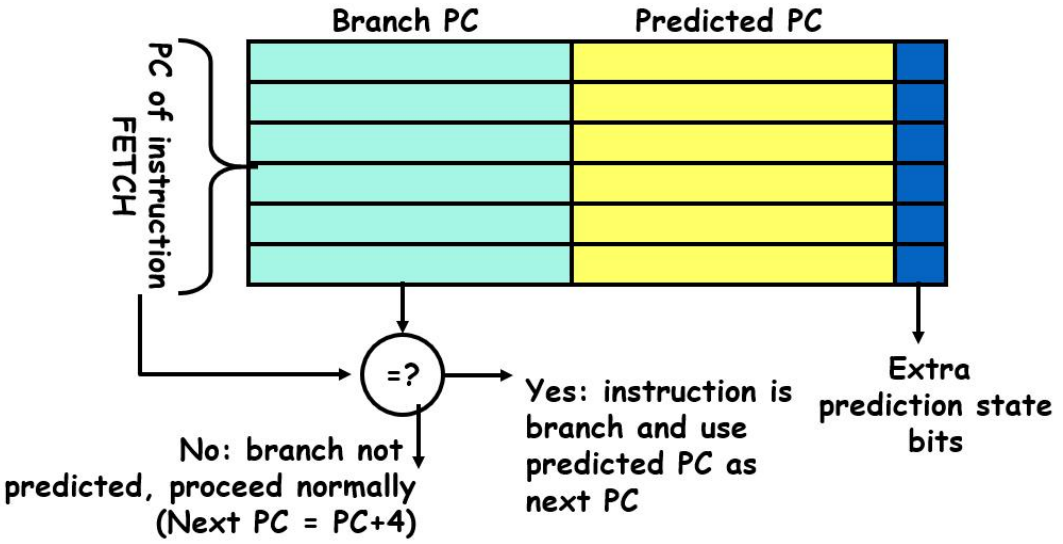
分支预测设计指导

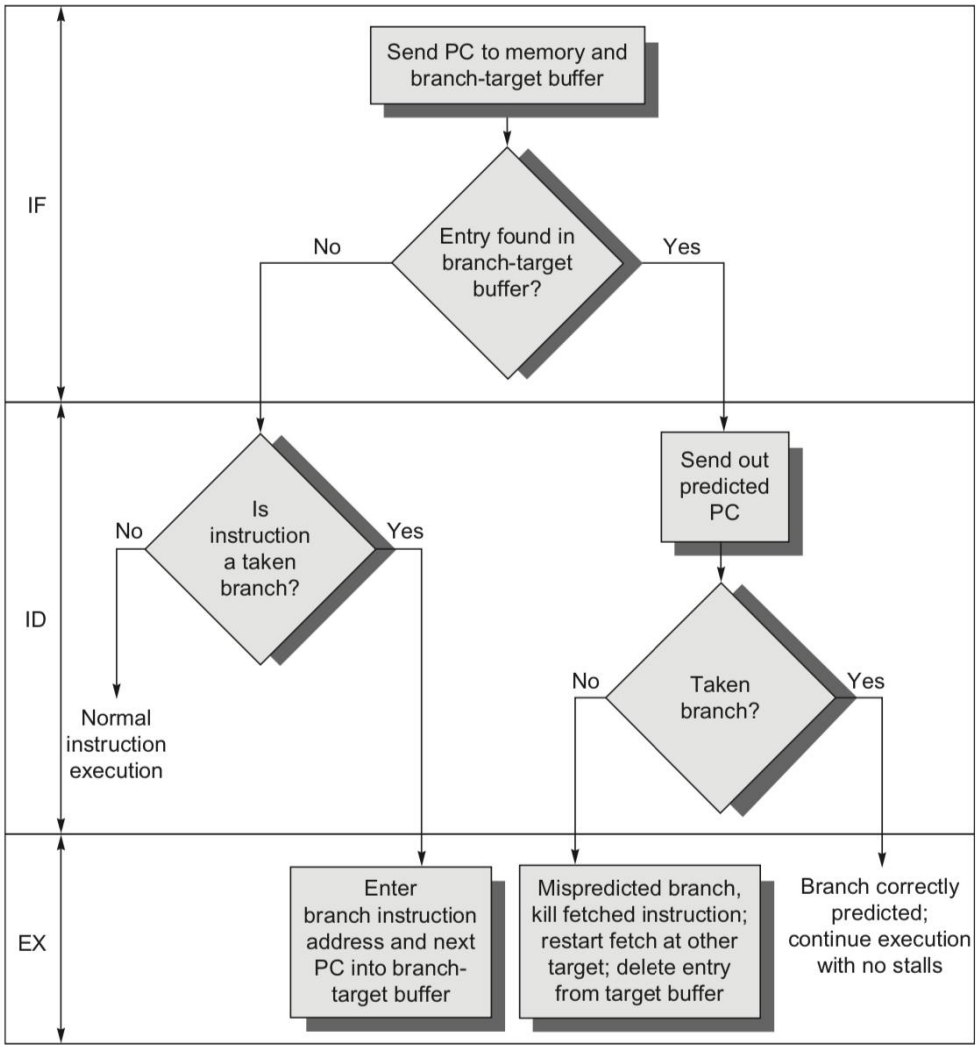
BTB 实现

BTB（Branch Target Buffer）是动态分支预测的一种基本方法。它使用一个 Buffer，里面记录了历史指令跳转信息。对于每一条跳转的 Branch 指令，它都将其写入 buffer，记录其跳转的地址，并有一个标志位标记最近一次执行是否跳转。这样如果有一条在 Buffer 里的跳转指令将执行时，可以根据 buffer 记录的历史跳转信息，预测下一条要执行的指令地址，预测正确的话可以减小分支开销。BTB 只使用了 1-bit 的历史信息，也可以视作 1-bit BHT。



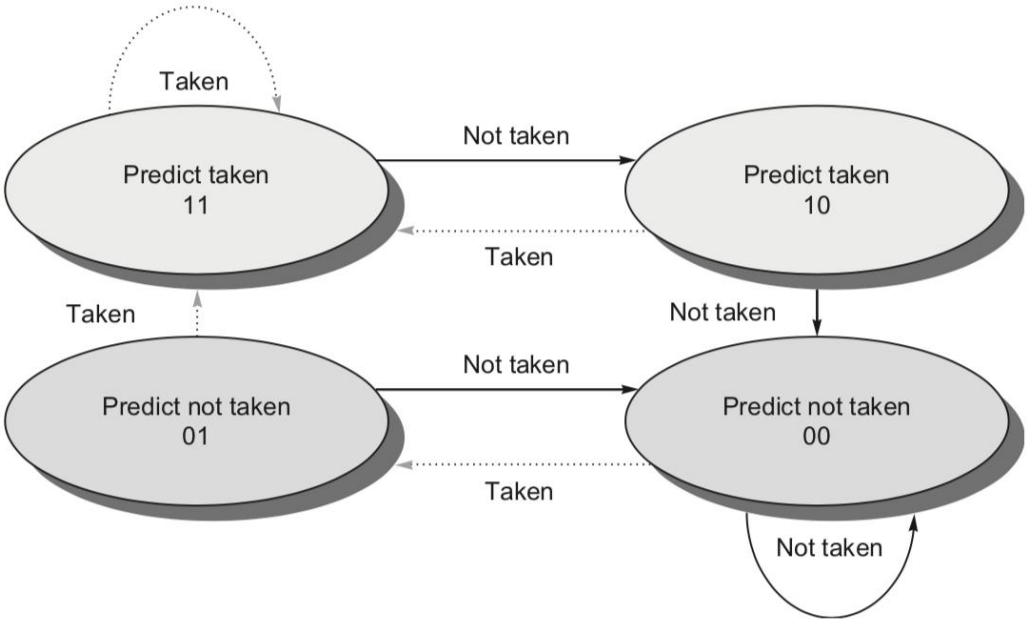
在我们之前实现的 lab2 RV32I Core 中，下一条 PC 地址是 PC + 4。在添加了 BTB 之后，对于 IF 阶段产生的 PC，在 BTB Buffer 里检查是否有对应项，如果有的话，根据其历史跳转记录，确定是否选择 predicted PC 作为下一 PC。如果当前 PC 不在 BTB 表里，但在 EX 段发现是一条需要跳转的 Branch 指令，则在 EX 阶段更新 BTB 表。另外，如果 PC 在 BTB 表中，在 EX 阶段发现预测的跳转失败，也需要更新 BTB 表，并 flush 错误装载的指令。

状态机如下图。



BHT 实现

BHT（Branch History Table）是动态分支预测的另一种基本策略。类似 BTB，它也维护了一个 N * 2 的 cache 作为 buffer。其中，N 是 BHT 表的项数（**一般取 4096 项**），根据 PC 的低位查找 BHT 表，每个项都维护了一个独立的 2-bit 状态机。如下图：



BHT 和 BTB 一样，在 IF 阶段对当前 PC 预测其是否跳转。相较于 BTB，BHT 的预测准确度更高，在 IF 阶段，首先判断当前 PC 在 BTB 表中是否跳转，如果跳转，再到 BHT 表中寻找其是否跳转。只有两者都预测跳转时，才预测当前指令跳转，并将 BTB 表中的预测跳转地址作为下一条指令的 PC 地址。特别的，如果 BHT 表预测跳转，BTB 表预测不跳转，或者 BHT 表预测不跳转，BTB 表预测跳转，都不预测当前指令跳转。

在 EX 阶段，BHT 表根据实际的跳转结果，更新 2-bit 的状态机，BTB 表则在冲突时更新。

实验报告

1. 在实验报告中补全下表：

Branch History Table (BHT)

BTB	BHT	REAL	NPC_PRED	flush	NPC_REAL	BTB update
Y	Y	Y	BUF	N		N
Y	Y	N		Y	PC_EX+4	
Y	N	Y	PC_IF+4			
Y	N	N				
N	Y	Y				
N	Y	N				
N	N	Y				
N	N	N				

1. 上表前三列是输入，其余是输出。
2. BTB 表示 BTB 的 buffer 是否命中；BHT 表示当前指令地址对应 BHT 中的状态是否是 predict taken 状态；REAL 表示当前分支指令是否真正跳转。其中 BTB 和 BHT 是否命中信号在 IF 阶段产生，随流水线段寄存器传递到 EX 阶段；REAL 信号在 EX 阶段产生。
3. NPC_PRED 表示预测下一条指令地址，BUF 表示从 BTB 中取出的地址；flush 表示刷新流水线；NPC_REAL 表示 EX 阶段正确判断出的 NPC。
4. BHT 根据状态机更新；BTB 在 cache 冲突时更新。
5. 动态分支预测根据 BHT 的是否命中来确定（因为更精确）；但是如果 BTB 没命中，BHT 命中，那么 NPC_PRED 选择 PC_IF+4。

2. **我们提供了 btb.s、bht.s、QuickSort.s、MatMul.s 四个测试样例，分别执行这四个测试样例，并在实验报告中分析以下内容：**
   * 分析分支收益和分支代价
   * 统计未使用分支预测和使用分支预测的总周期数及差值
   * 统计分支指令数目、动态分支预测正确次数和错误次数
   * 对比不同策略并分析以上几点的关系

注意：由于 QuickSort.s 和 MatMul.s 执行完成后会陷入死循环，测试终止时间记为第一次执行到死循环的 jal 指令为止。