基于 BPB 和 BTB 组合的分支预测

童敢 黄家腾 赵琛然

摘要：本次实验我们小组在上一次实现的流水线基础上，基于 BPB 和 BTB 的组合，实现了分支预测，此前由分支和跳转必然带来一个周期的 STALL，在引入 2-bit 的分支预测之后可以以较高的准确度预测分支跳转，在 IF 阶段根据 BTB 中的地址直接跳转，大大提高效率。

一、设计实现

（一）设计思路

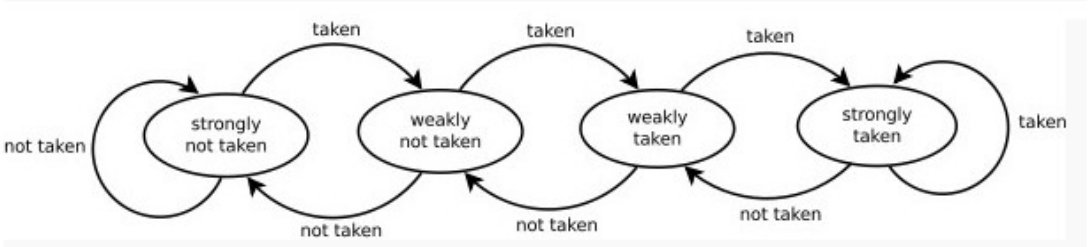
在之前的设计中，无条件分支和条件分支都放在了 ID 阶段，对于无条件分支 JAL 和 JALR 而言，一定会浪费一个周期，对于发生分支的跳转指令而言，也会产出一个周期的 STALL，如果程序中有大量的循环，则效率很低。

现在的处理器都使用 BPB 或 BTB 来防止出现这种状况，对于 RISC-V 而言，官方文档明确表明延迟槽不应该存在，因为现在的分支预测技术已经十分成熟，由误预测带来的流水线 Flush 已经很少了。

考虑到之前的设计，决定综合 BPB 和 BTB 两者的优势，同时在 IF 阶段引入两者，BPB 负责预测分支的方向，BTB 负责提供预测为跳转时的目标地址的提供，在 IF 阶段首先检测指令是否为分支/跳转指令，如果是则进行预测，预测结果为跳转则更新 PC 为 BTB 中的地址。

（二）BPB 结构

BTB 内置在 pc_reg 模块中，为 32 个二位的寄存器，当处理器初始化时，BTB 中的所有位置为 0，需要预测指令时，根据指令地址的后 5 位索引分支预测器的值，对于两位的分支预测有以下几种情况：



1. 00

表示 Strongly_Not_Taken，此时预测为不跳转，如果该条指令跳转了，则将此值更新为 01。

2. 01

表示 Weakly_Not_Taken，此时预测为不跳转，如果该条指令跳转了，则将此值更新为 11，如果该条指令没有跳转，将此值更新为 00。

3. 11

表示 Weakly_Taken，此时预测为跳转，如果该条指令跳转了，则将此值更新为 10，如果没有跳转，将此值更新为 01。（考虑每次只更新一位所以此处将 11 设置为 Weakly_Taken 而不是 10）

4. 10

表示 Strongly_Taken，此时预测为跳转，如果该条指令没有跳转，则将此值更新为 11。

总的来说，对于一个两位的分支预测器而言，其本质就是一个状态机，根据指令是否成功分支更新状态，根据前人的经验，这种方式可以保持很高的预测准确度。

（三）BTB 结构

实现完指令是否分支的预测之后，可以结合 BTB，在预测为跳转时提供 BTB 中的值更新 PC。BTB 为 32 个 32 位寄存器，同样由指令地址的低 5 位进行索引，每个寄存器存放跳转的地址。

当 ID 阶段返回上一条指令结果为成功跳转之后，根据目的地址和该指令地址的低 5 位更新 BTB 值，之后在预测为跳转时，取 BTB 值更新 PC 值。

（四）取值逻辑

加入 BTB 和 BPB 之后，取值逻辑如下：

1. 如果 IF 阶段不 STALL，则判断上一步预测是否正确：
2. 如果上一步预测正确，则判断本条指令是否为分支指令，如果是分支指令且根据 BPB 预测为跳转，则更新 PC 值为 BTB 中的对应的地址；否则直接将 PC 更新为 PC+4。
3. 如果上一步预测不正确，说明需要恢复正确的指令地址，如果被误预测的结果为分支成功，说明该跳转没有跳转，则更新 PC 值为跳转的目标地址；如果是分支指令但是没有成功跳转，则说明不需要跳转但是跳转了，则更新 PC 值为被误预测的指令的地址+4；其他情况更新 PC 为 PC+4 即可。

最终版本的代码实现文件路径为：

05.BranchPrediction\BranchPrediction.srcs\sources_1\new

二、运行测试程序

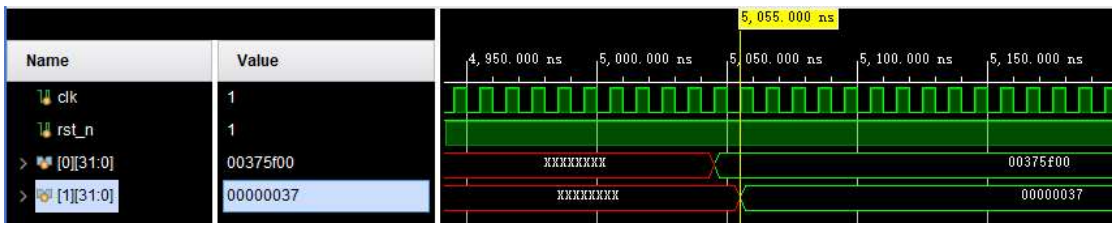
（一）测试程序

由于未提供测试程序，所以依然使用上节课提供的测试程序，该程序中有循环、判断、函数调用等，存在很多分支指令，可以用来测试本次实现。

设置为 n 的值为 10，分别测试引入分支预测和不引入分支预测两种情况下得到结果时使用的时钟周期数即可知道是否有带来效率上的提升。

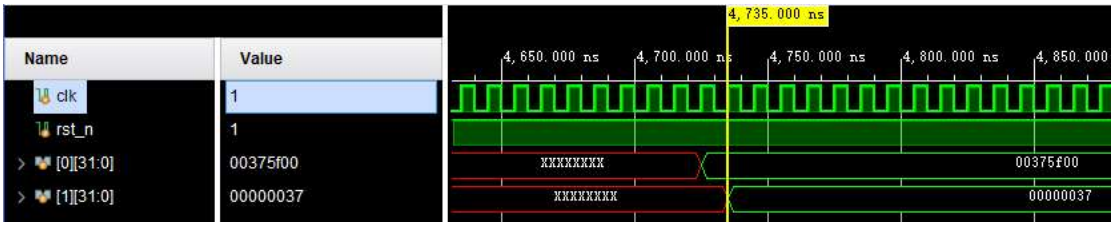
（二）运行测试程序

1. 未引入分支预测时



共消耗了 5055ns，约 505 个时钟周期。

2. 引入分支预测后



共消耗了 4735ns，约 473 个时钟周期。

和未引入分支预测相比，共节约了约 32 个时钟周期，且结果正确有效。因此结论是引入分支预测后，分支效率大大提高，当分支指令数量很大时，这种优势会更明显。