计算机体系结构实验报告

LAB04

题目：动态分支预测

姓名：林祥

学号：PB16020923

实验目的

在给定 RISC-V 32I 的 CPU 核心代码上，对于 Branch 类指令实现：

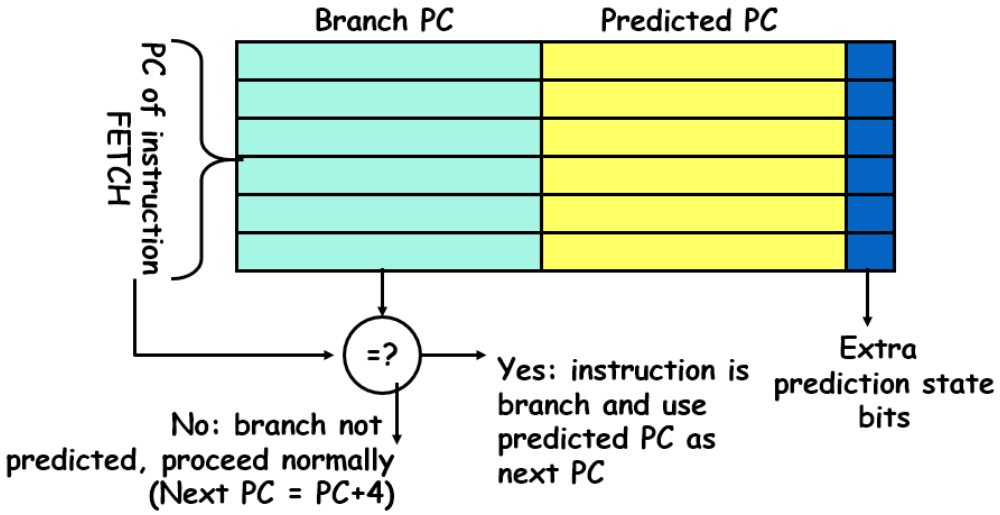
1. 基于 BTB（Branch Target Buffer）的动态分支预测
2. 在 BTB 的基础上实现 2bit BHT（Branch History Table）的动态分支预测

实验平台

EDA 工具为 Vivado 2017.4

实验设计与过程

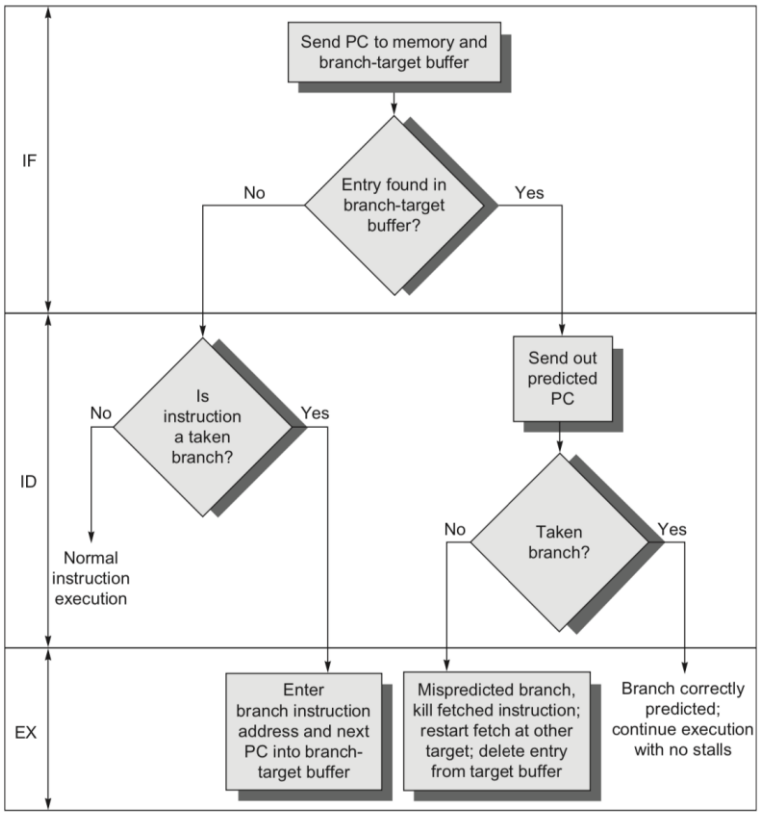
一、 BTB



BTB 模块中的 buffer 结构如上图，与 Cache 类似。第一部分 Branch PC 是缓存的地址(tag)列表，采用直接映射策略。第二部分 Predicted PC 是预测的跳转地址。第三部分为 1 位宽，表示该项是否有效。

输入的 PC 低位寻址到对于表项，对 tag 部分进行比较，如果相

等，并且该项是有效的，则认为命中，然后使用 Predicted PC 作为输出，表示根据历史信息，这是一个分支指令并且预测跳转。



BTB 工作流程如上图。

BTB 模块介于 IF 和 ID 段寄存器之间，对于 IF 段寄存器输出的 PCF，输入到 BTB 中，BTB 查找 Buffer，如果命中，则输出地址向 NPC 模块输入，NPC 模块选择预测 PC 作为下一个 PC，如果不命中，直接使用 PCF+4 作为下一个 PC，同时，命中的状态位（1 表示进行了预测且预测跳转，0 表示不预测）和预测 PC 也随流水段流到 EX 段。

等到当前指令到达了 EX 段，就可以确定当前指令是否跳转，如果实际跳转但之前未预测跳转或者实际不跳转但之前预测跳转，都需要清空 ID 和 EX 段的寄存器，NPC 模块重新生成下一个 PC。在实际跳转但之前未预测跳转的情况下，需要把当前 PC 和跳转的 PC 更新写入

BTB 的 buffer 中。在实际不跳转但之前预测跳转的情况下，需要把 BTB 的 buffer 中的对应项的有效位置为 0。

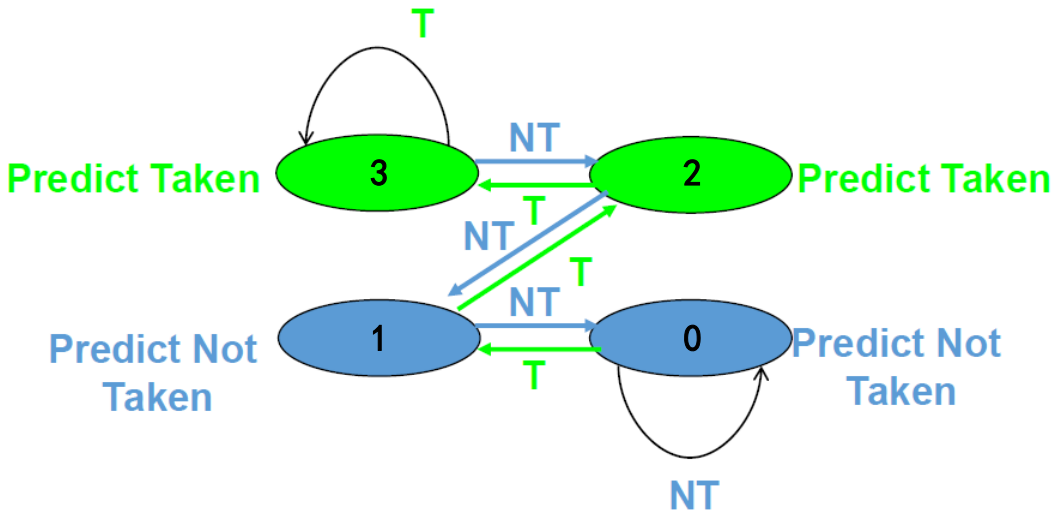
二、 BHT

分支策略采用 BHT 时，分支 PC 和预测的 PC 仍然保存在 BTB 模块的 buffer 中，但 BTB 的 buffer 不再决定是否预测跳转，BTB 模块的命中与否只决定是否进行预测（或者说当前指令是否为分支指令）。预测的状态位（0 表示不预测，1 表示预测）和预测的 PC 也随流水线流到 EX 段。BTB 的更新方法同上（不再使用有效位，只更新地址）。

BHT 模块则进行预测是否跳转。维护一个表，每个表项为 2bit 的状态，查表同样采用直接映射，但不再维护和比较 tag。BHT 模块对于输入的 PC，查到对应表项后，如果 2bit 的值为 0 或 1，则预测跳转，如果为 2 或 3，则预测不跳转，预测结果（0 表示不跳转，1 表示跳转）也随流水线流到 EX 段。

等到当前指令到达了 EX 段，就可以确定当前指令是否实际跳转。根据之前传过来的是否预测的状态位（BTB hit）、是否预测跳转状态位(BHT taken)为以及当前的实际跳转结果，我们可以决定是否清空流水线的 ID 和 EX 段进行重新取指令、是否对 BTB 进行更新（见下面的真值表），以及 BHT 如何更新（见下面的状态转换图）。

BHT 的每一项本质是一个 2bit 的状态机，每次执行分支指令都要更新。如果 EX 段实际跳转，则对应表项状态按 0->1->2->3->...->3 更新，如果 EX 段实际不跳转，则状态按 3->2->1->0->...->0 更新。



真值表如下，左边 3 列为 8 种情况组合输入，右边为输出

BTB hit	BHT taken	REAL	NPC_PRED	flush	NPC_REAL	BTB update
Y	Y	Y	BTB_BUF	N	BRANCH_PC (= BTB_BUF = NPC_PRED)	N
Y	Y	N	BTB_BUF	Y	PC_EX + 4 (!= NPC_PRED)	N
Y	N	Y	PC_IF+4	Y	BRANCH_PC (= BTB_BUF != NPC_PRED)	N
Y	N	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N
N	Y	Y	PC_IF+4	Y	BRANCH_PC (!= NPC_PRED)	Y
N	Y	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N
N	N	Y	PC_IF+4	Y	BRANCH_PC (!= NPC_PRED)	Y
N	N	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N

其中，

- BTB hit 为 Y 表示在 BTB 根据当前 PC 的低位在 Buffer 中命中(Tag 部分匹配）了，说明可以进行预测（历史信息表明这是分支指令）
- BHT taken 为 Y 表示 BHT 根据 PC 低位直接映射到 Table 中的一项，该项的 2bit 状态机预测该跳转
- REAL 为 Y 表示该分支指令实际发生了跳转
- NPC_PRED 表示下一个 PC 的预测值（或来源）
- NPC_REAL 表示下一个 PC 的实际值
- Flush 为 Y 表示对流水线 ID 和 EX 段进行清空
- BTB update 为 Y 表示需要对 BTB 进行更新

实验结果

注：测试使用的 BTB 的 buffer 大小为 32 项（buffer 地址长为 5 位）

表 1

运行时间（起始时间 16431000ps）		
	BTB.S/ps	BHT.S/ps
BTB	17059000	17195000
BHT	17063000	17167000
静态	17451000	17503000

表 2

运行周期（每周期 2000ps）			与静态分支预测的差值	
	BTB.S	BHT.S	BTB.S	BHT.S
BTB	314	382	196	154
BHT	316	368	194	168
静态	510	536	-	-

表 3

总指令数		
测试文件	BTB.S	BHT.S
指令数	307	335

表 4

分支指令数		
测试文件	BTB.S	BHT.S
指令数	101	110

表 5

正确次数		
预测策略\测试文件	BTB.S	BHT.S
BTB	99	88
BHT	98	95
静态	1	11

表 6

错误次数		
预测策略\测试文件	BTB.S	BHT.S
BTB	2	22
BHT	3	15
静态	100	99

回答问题

1. 对于一条分支指令，当动态分支预测命中，实际命中时，比不用分支预测少用几个周期？

   答：2 个周期，因为无需清空流水线的 ID 和 EX 段重新取指。

2. 对于一条分支指令，动态分支预测失败比非跳转指令多用几个周期？

   答：2 个周期，因为需要清空流水线的 ID 和 EX 段重新取指。

3. 统计未使用分支预测和使用分支预测的总周期数及差值。

   答：见上面的实验结果的表 1、2。

4. 统计分支指令数目、动态分支预测正确次数和错误次数。

   答：见上面的实验结果表 4、5、6。

5. 对比不同策略并分析以上几点的关系

   答：

   （1）总体而言，对于执行两个测试文件使用的周期数和预测正确次数，使用动态分支预测（BTB,BHT）的效果明显好于使用静态分支预测（直接预测不跳转），因为两个测试文件主体都是循环，实际跳转的情况较多，静态预测几乎总是会预测错误。

   （2）BTB.s 这段代码使用 BTB 策略的效果最好，因为它只有一个单层循环，除了首尾两次，分支结果稳定为跳转，使用 BTB 一个状态位进行简单的预测就足够了。BHT.s 这段代码使用 BHT 策略的效果最好，因为它有一个两层循环，内层循环分支结果变化较大，上一次外循环的最后一次内循环不跳转，本次外循环的第一次内循

环跳转，如果只使用 BTB 一个状态位进行预测，内循环就会在每次外循环跳转前后出现 2 次预测错误，而使用 BHT 的 2 个状态位只会有 1 次预测错误。

（3）实际运行时间并不完全取决于预测正确的次数，因为分支指令只是代码的一部分，还有算术逻辑指令在总时间中也占很大一部分。预测正确率提高只能带来有限度的总时间减少（加速）。

6. 计算整体 CPI 和加速比

答：由上面实验结果的表 2 的运行周期数除以表 3 的总指令数即可得 CPI，如下面表 7。由上面实验结果的表 2 的 BTB 和 BHT 策略的运行周期除以静态分支预测策略的运行周期即可得加速比，如下面表 8。

表 7

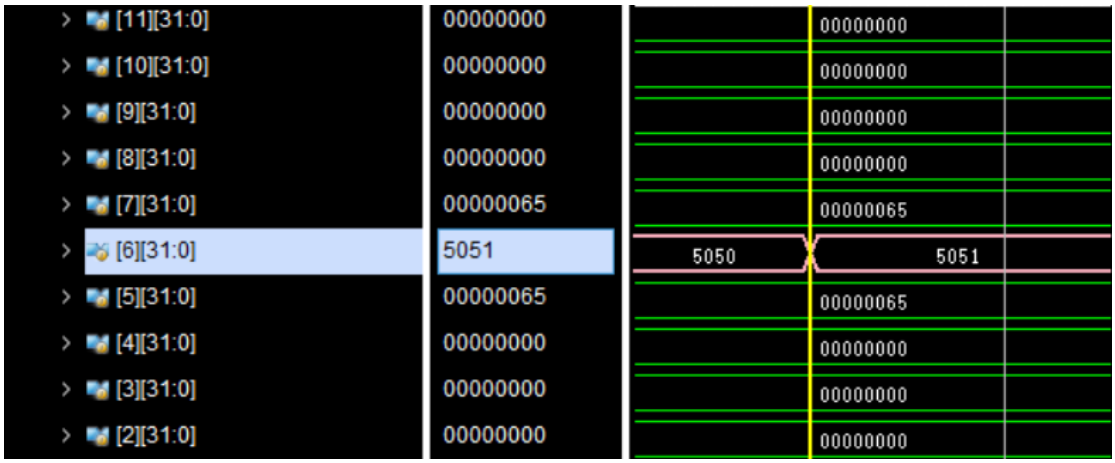
CPI		
	BTB.S	BHT.S
BTB	1.022801	1.140299
BHT	1.029316	1.098507
静态	1.661238	1.600000

表 8

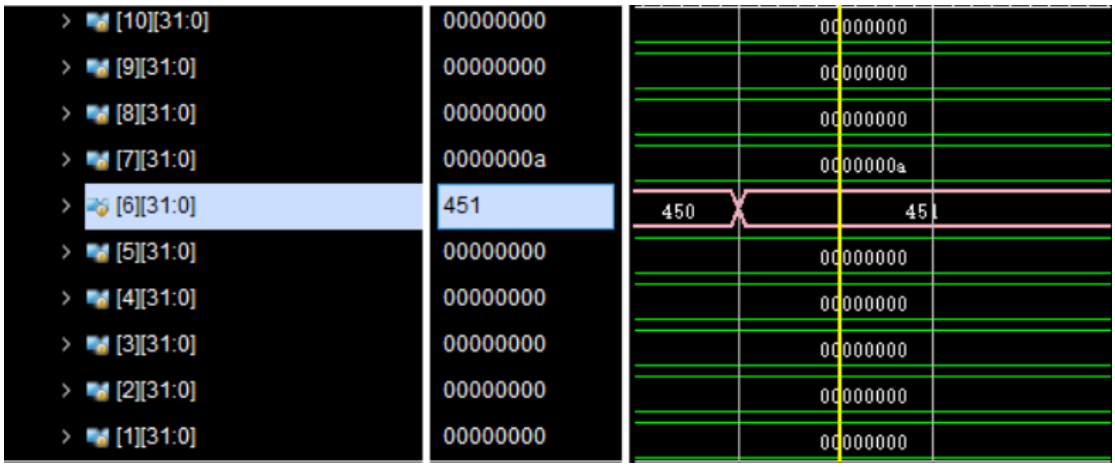
加速比		
	BTB.S	BHT.S
BTB	1.6242038	1.4031414
BHT	1.6139241	1.4565217

附：部分截图

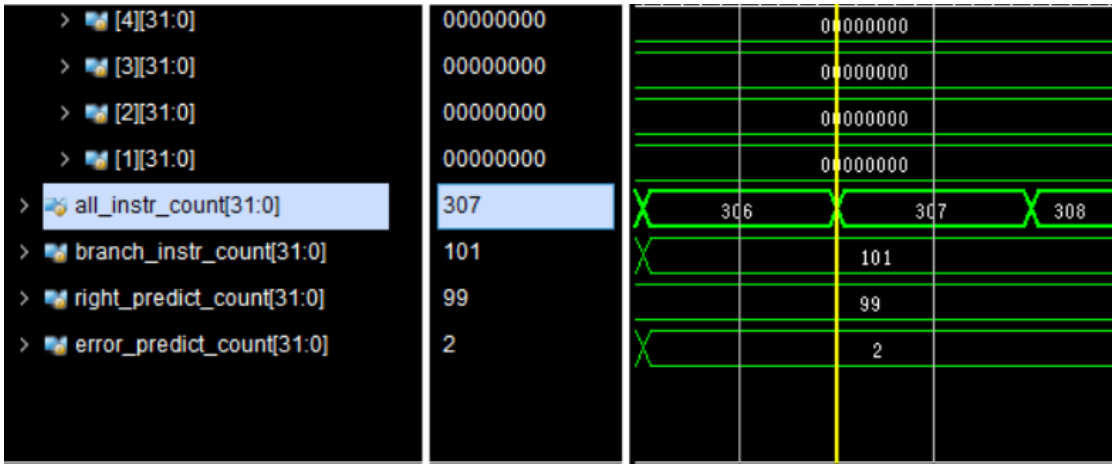
BTB.s 的运行结果



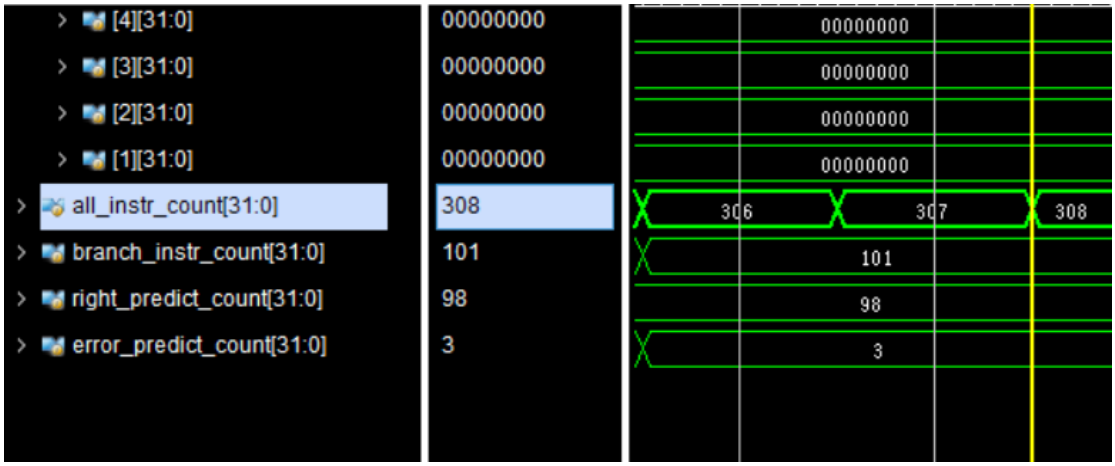
BHT.s 的运行结果



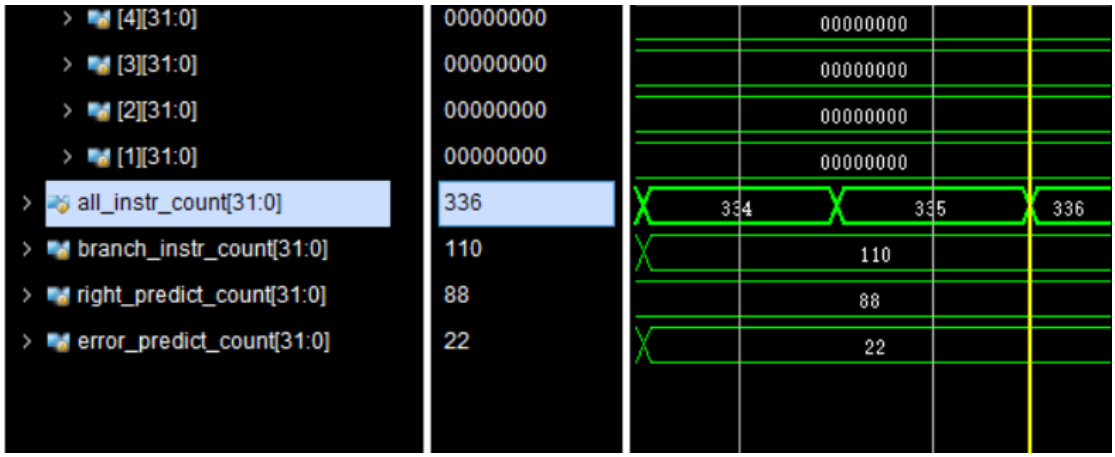
BTB.s 使用 BTB 策略统计结果



BTB.s 使用 BHT 策略统计结果



BHT.s 使用 BTB 策略统计结果



BHT.s 使用 BHT 策略统计结果