计算机体系结构 Lab4 实验报告

PB20111686 黄瑞轩

1. 实现思路

1.1 BTB 实现思路

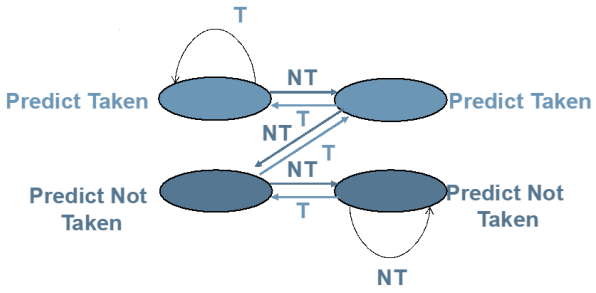
使用一个 Buffer，里面记录历史指令跳转信息。对于每一条跳转的 Branch 指令，它都将其写入 Buffer，记录其跳转的地址，并有一个标志位标记最近一次执行是否跳转。这样如果有一条在 Buffer 里的跳转指令将执行时，根据记录的历史跳转信息，预测下一条要执行的指令地址。否则预测为不跳转，并将信息写入 Buffer 中。

下面是核心代码（篇幅所限，非必要部分如复位已省略）。

```
reg [31:0] tag[entry_num-1:0];     // 分支指令内容
reg [31:0] target[entry_num-1:0];  // 分支目标地址
reg entry_valid[entry_num-1:0];    // 分支是否成功
reg [ENTRY_LEN-1:0] pointer;       // 下一次写入 Buffer 的位置
// 查找表项以输出
always@(*) begin
    PC_predicted <= 0;
    valid <= 1'b0;
    for (integer j = 0; j < entry_num; j = j + 1)
        if ((PC_IF == tag[j]) && entry_valid[j]) begin
            PC_predicted <= target[j];
            valid <= 1'b1;
        end
end
// 查找表项以更新
always@(posedge clk or posedge rst) begin
    if (opcode_EX == br_op) begin
        flag <= 1'b0;
        for (integer i = 0; i < entry_num; i = i + 1)
            if (PC_EX == tag[i]) begin
                target[i] <= br_target;
                entry_valid[i] <= br;
                flag <= 1'b1;
            end
        if (flag == 1'b0) begin
            target[pointer] <= br_target;
            tag[pointer] <= PC_EX;
            entry_valid[pointer] <= br;
            pointer <= pointer + 1;
        end
    end
end
```

1.2 BHT 实现思路

维护一个 N×2 的 Buffer。其中 N 是 BHT 表的项数，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（如图所示）。



下面是核心代码（篇幅所限，非必要部分如复位已省略）。

```
reg [1:0] entry[entry_num-1:0];    // 表项
assign prediction = entry[tag][1]; // 输出

always@(posedge clk or posedge rst) begin
    if (opcode_EX == br_op) begin
        if (br) begin
            if (entry[tag_EX] == ST) begin
                entry[tag_EX] <= ST;
            end
            else begin
                entry[tag_EX] <= entry[tag_EX] + 2'b01;
            end
        end
        else begin
            if (entry[tag_EX] == SN) begin
                entry[tag_EX] <= SN;
            end
            else begin
                entry[tag_EX] <= entry[tag_EX] - 2'b01;
            end
        end
    end
end
```

1.3 如何实现预测

在 IF 阶段对当前 PC 预测其是否跳转，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。

特别地，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。

2. 测试、统计与分析

根据 BTB、BHT 模块的实现，在原来的 CPU 中加入相关的连线，将 BTB、BHT 模块加入 CPU 中。根据实验要求还添加了部分用于统计的寄存器和 always 模块。

使用 btb.s、bht.s、QuickSort.s、MatMul.s 四个程序作为测试样例，并统计未使用分支预测和使用分支预测的总周期数及差值，以及分支指令数目、动态分支预测正确次数和错误次数。测试的相关截图放在本报告的附录部分。

	总周期数（以 ns 计）		分支预测减少周期数	分支指令数目	动态分支预测	
	使用分支预测	不使用分支预测			正确次数	错误次数
btb.s	1266	2050	784	315	314	1
bht.s	1594	2154	560	397	387	10
QuickSort.s	215734	220010	4276	53932	53862	70
MatMul.s	1375834	1404586	28752	343957	343699	258

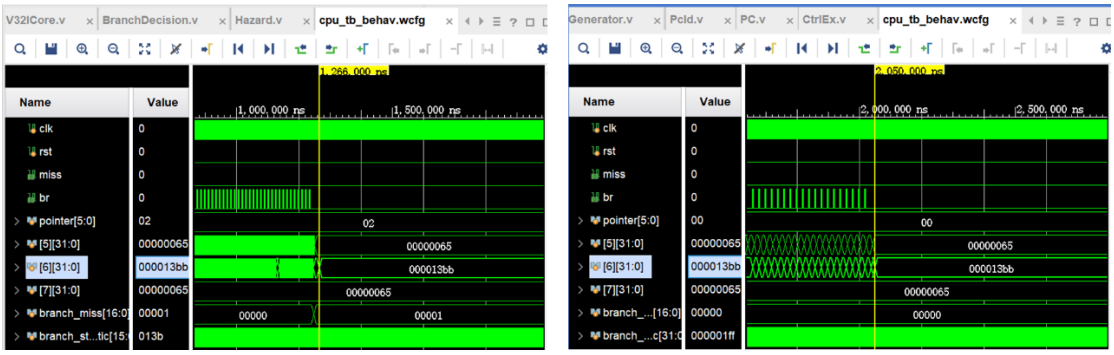
从以上数据可以看出，在使用分支预测的情况下，各程序测例的总周期数都比不使用分支预测的情况下的总周期数少。可以看出，动态分支预测正确的次数很高，错误次数相对较少，这说明分支预测机制的正确率较高，可以有效地提高程序的执行效率。

另外，从数据中可以看出，使用 btb.s 和 bht.s 程序时，分支预测的效果（减少的时钟周期数）不如 QuickSort.s 和 MatMul.s 程序，这是因为前两个程序中的分支指令（频率）较少，分支预测机制的优势没有得到充分发挥。因此，分支预测机制的效果还受到程序结构、分支指令数量等因素的影响。

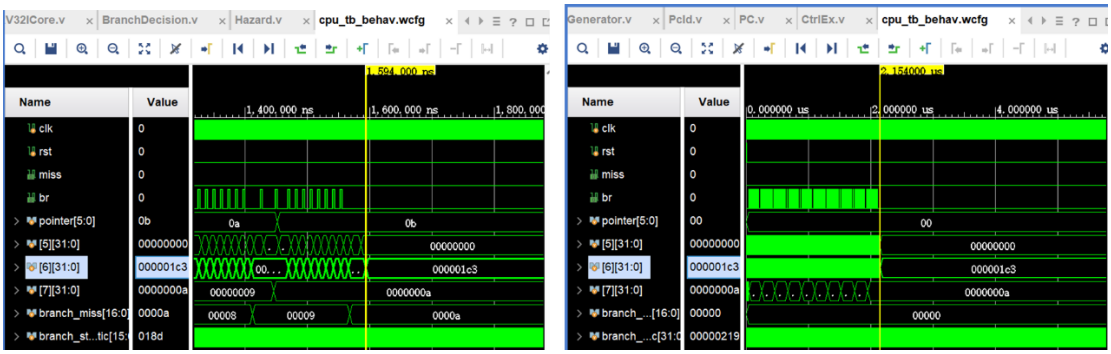
综上，分支预测机制可以显著减少程序的执行周期数，提高程序的执行效率。但其效果还受到程序结构、分支指令数量等因素的影响。

3. 附录

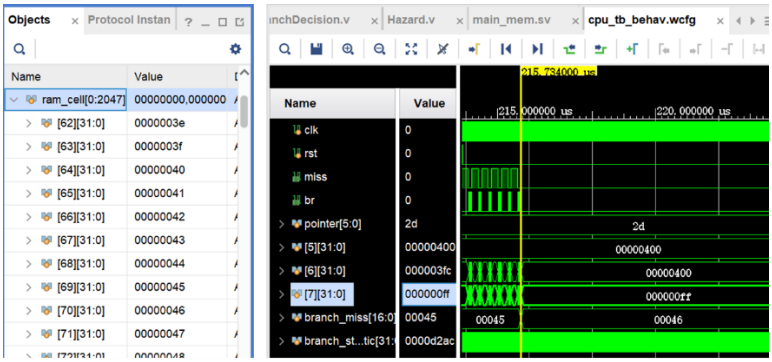
btb.s 测试截图（左边为使用分支预测，右边为不使用分支预测，下同）

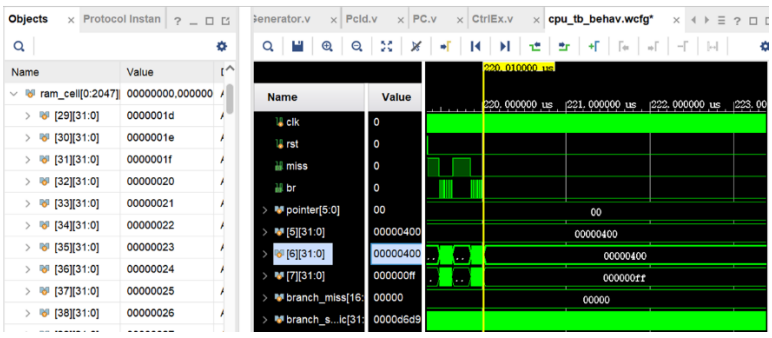


bht.s 测试截图

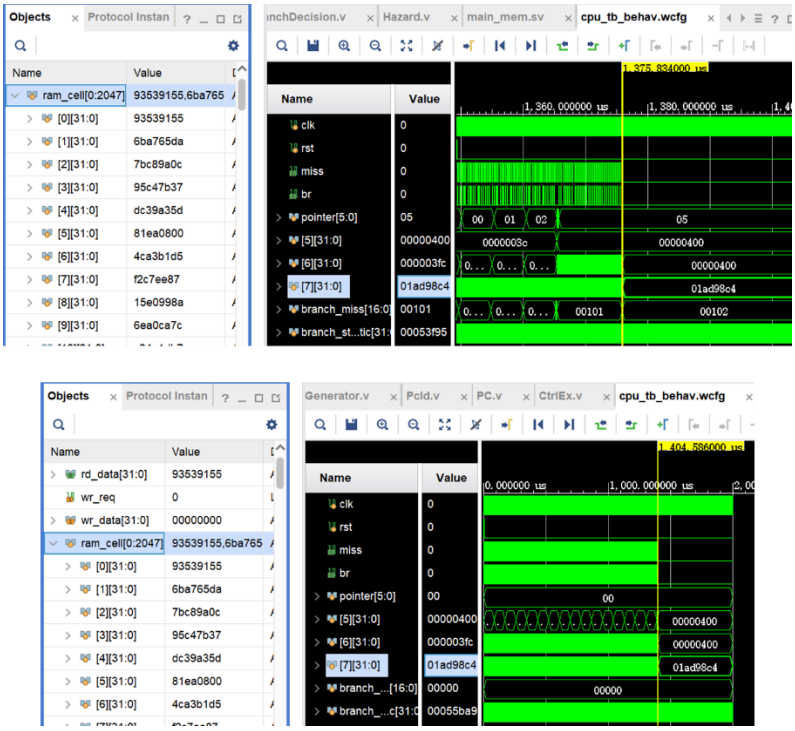


QuickSort.s 测试截图（上方为使用分支预测，下方为不使用分支预测，左侧为主存内容，显示了正确性，下同）





MatMult.s 测试截图



Branch History Table (BHT)

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