Lab4 实验报告

PB20000072 王铖潇

实验目的

1. 实现BTB（Branch Target Buffer）和BHT（Branch History Table）两种动态分支预测器
2. 体会动态分支预测对流水线性能的影响

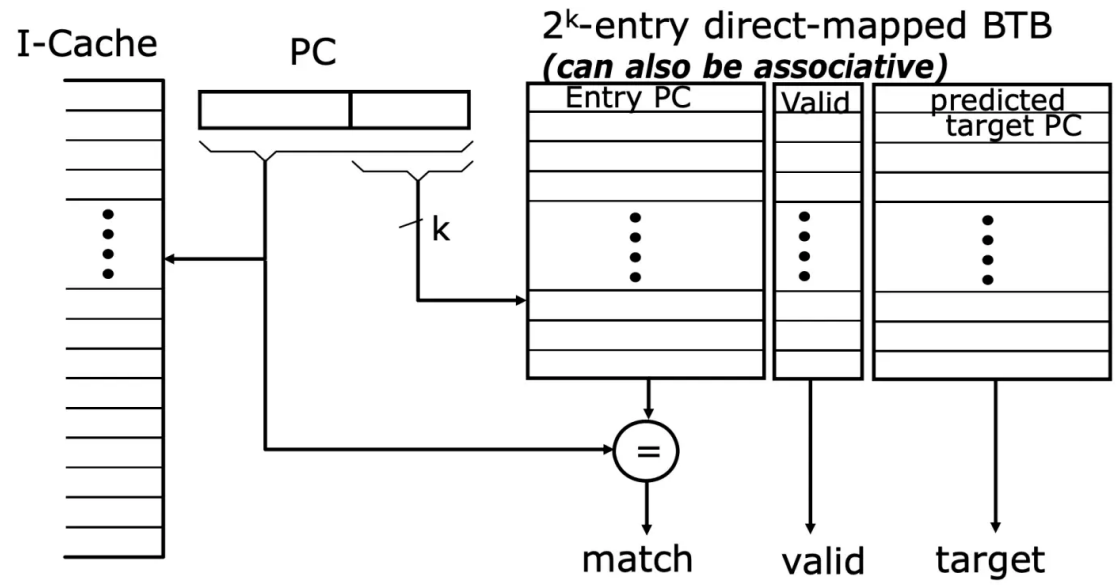
实验内容

1. 在Lab3阶段二的RV32I Core基础上，实现BTB
2. 在BTB的基础上，实现BHT
3. 分析比较性能并撰写实验报告

实验过程

BTB的实现

参考的图片如下：



按照需要查询的PC的低位`addr_search`进行索引：

```
assign {tag_search, addr_search} = PC_search;
```

如果索引到的条目有效（`valid[addr_search]==1`）并且存储的`tag`和查询的PC的`tag`相等，说明BTB表中有对应表项。

`BTB_br`表示是否需要预测跳转：

```
assign BTB_br = (valid[addr_search]==1) && (tag_search == pc_tag[addr_search])
&& (state[addr_search] == 1);
```

`PC_new`是需要更新的PC：

```
assign {tag_new, addr_new} = PC_new
```

对BTB表的更新发生在EX阶段。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：

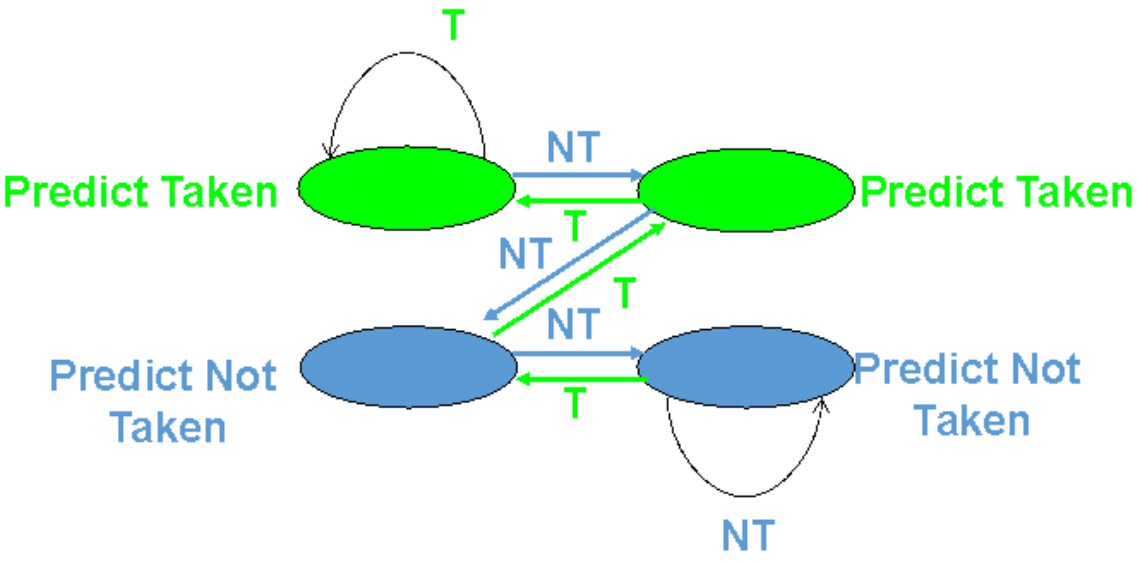
```
state[addr_new] <= 1;
valid[addr_new] <= 1;
pc_pred[addr_new] <= tag_new;
pc_tag[addr_new] <= predict_new;
```

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：

```
state[addr_new] <= 0;
valid[addr_new] <= 1;
pc_pred[addr_new] <= tag_new;
pc_tag[addr_new] <= predict_new;
```

BHT的实现

与上图几乎同理，只不过是`state`变成了2位，需要维护一个下图的状态机。



由于可以借助BTB，所以不需要存储图中的`Entry PC`，`valid`和`predicted target PC`，只需要在BTB表中查询，并且修改更新`state`的过程。

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`state`：

```
if(state[addr_new] != 2'b11)
                        state[addr_new] <= state[addr_new] + 1;
                    else
                        state[addr_new] <= state[addr_new];
```

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`state`：

```
if(state[addr_new] != 0)
                        state[addr_new] <= state[addr_new] - 1;
                    else
                        state[addr_new] <= state[addr_new];
```

维护一个Buffer

有时在IF段预测的指令在EX段才发现出错，这种情况下需要恢复，所以需要一个Buffer，记录前几个时钟周期的PC和预测过的`branch`（是否跳转）。

```
always @(posedge clk or posedge rst) begin
        if(rst) begin
            for(integer i = 0; i < 2; i++) begin
                PC_Buffer[i] <= 0;
                branch_Buffer[i] <= 0;
            end
        end
        else begin
            PC_Buffer[1] <= PC_in;
            branch_Buffer[1] <= branch_in;
            PC_Buffer[0] <= PC_Buffer[1];
            branch_Buffer[0] <= branch_Buffer[1];
        end
    end
```

```
assign branch_out = branch_Buffer[0];
    assign PC_out = PC_Buffer[0];
```

`branch_out`是当前位于EX段的指令在2个时钟周期前被预测的是否跳转，`PC_out`是当前位于EX段的指令的真实的下一条指令地址。

分支预测

`branch_in`：在IF段，BTB或者BHT预测出的下一条指令是否跳转；

`PC_pred_in`：在IF段，BTB或者BHT预测出的下一条指令跳转地址；

`br_target`：EX段，跳转指令的目标地址。

`PC_out`：EX段，真实的下一条指令地址。

```
always @(*) begin
    if(~is_br_type) begin // 不是跳转指令
        NPC = branch_in ?  PC_pred_in: PC_4; // NPC是IF段的那一条指令如果预测是进行跳转，那么就预测为它预测跳转地址；否则是IF段指令的PC+4
        branch_prediction_miss = 0;
```

```
    end
    else begin // 是跳转指令
        if(branch == branch_out) begin // 预测成功
            NPC = branch_in ?  PC_pred_in: PC_4; // NPC是IF段的那一条指令如果预测是进行跳转，那么就预测为它预测跳转地址；否则是IF段指令的PC+4
            branch_prediction_miss = 0;
        end
        else begin
            // 预测失败
            NPC = branch ? br_target : PC_out; // 真实情况如果是进行跳转，那么下一条指令是跳转的目标地址，否则是真实的下一条指令地址
            branch_prediction_miss = 1; // miss信号置1，在Harzard模块中需要flush ID和EX段。
        end
    end
end
```

实验分析

Branch History Table

BTB	BHT	REAL	NPC-PRED	flush	NPC-REAL	BTB-UPADTE
Y	Y	Y	BUF	N	BUF	N
Y	Y	N	BUF	Y	PC_EX+4	N
Y	N	Y	PC_IF+4	Y	BUF	N
Y	N	N	PC_IF+4	N	PC_EX+4	N
N	Y	Y	PC_IF+4	Y	Br_target	Y
N	Y	N	PC_IF+4	N	PC_EX+4	N
N	N	Y	PC_IF+4	Y	Br_target	Y
N	N	N	PC_IF+4	N	PC_EX+4	N

结果分析

1. 分支收益和分支代价

   预测成功时，分支收益是2个时钟周期；预测失败时，分支收益是0个时钟周期。

   分支代价是2个时钟周期。

2. 未使用分支预测和使用分支预测的总周期数及差值

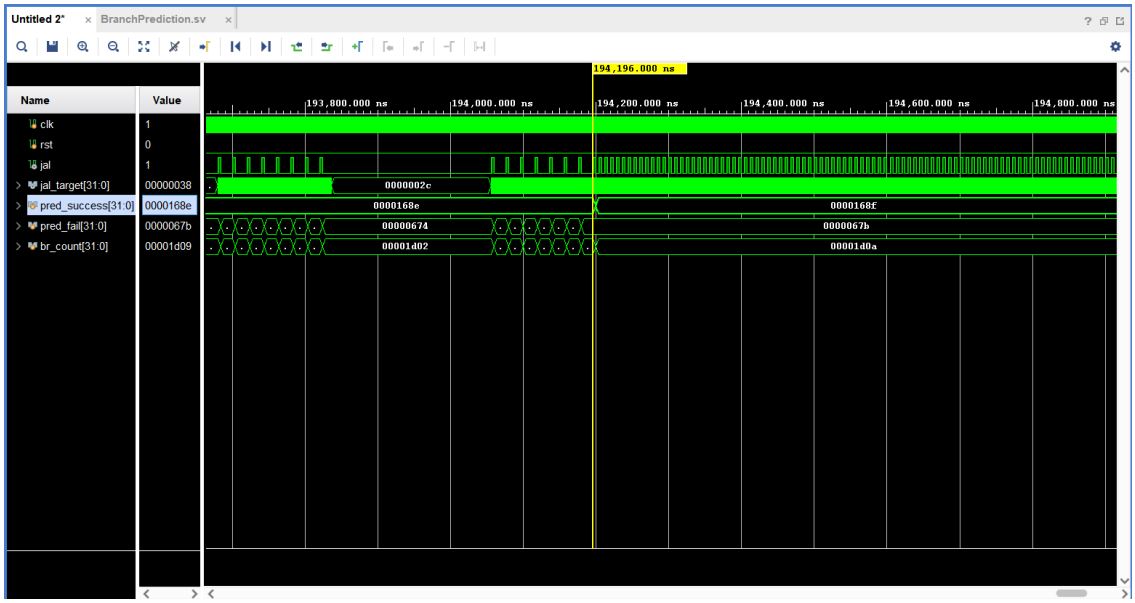
	矩阵乘法	快速排序	BTB	BHT
BTB	336003	34547	325	386
BHT	335725	33538	321	365
无预测	338301	35909	515	535
BTB与无预测的差值	2298	1362	190	149
BHT与无预测的差值	2576	2371	194	170

3. 分支指令数目、动态分支预测正确次数和错误次数

BTB：

	矩阵乘法	快速排序	BTB	BHT
预测正确	4106	5774	99	87
预测错误	518	1659	2	23
分支指令数目	4624	7433	101	110

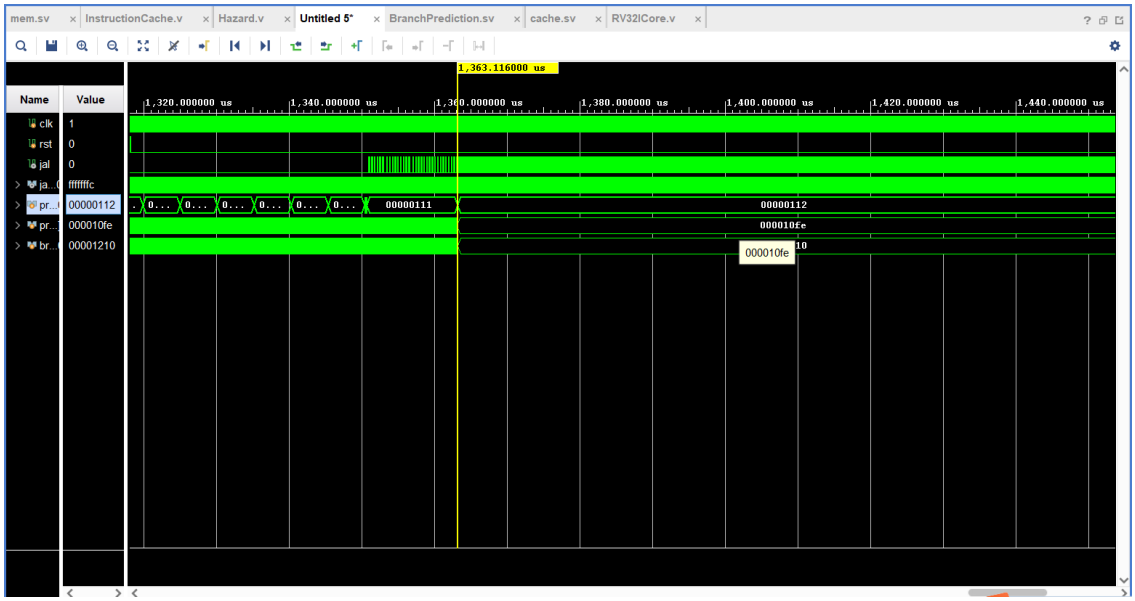
快速排序的仿真结果如图：



BHT：

	矩阵乘法	快速排序	BTB	BHT
正确	4350	5801	99	96
失败	274	1632	2	14
分支指令数目	4624	7433	101	110

矩阵乘法的仿真结果如图：



4. 对比不同策略并分析以上几点的关系
    - 分支代价和分支收益与具体的策略无关，与CPU的设计有关。
    - 使用BHT的预测结果普遍比BTB好（分支预测的成功数更多，运行的周期数更短）。

实验总结

本次实验思路较为清晰，完成了BTB和BHT分支预测的设计，加深了对分支预测的理解。但由于没有给出代码框架，刚开始的时候实现起来有些困难。