Lab4 分支预测实验报告

徐煜森 PB16110173

一. 实验目标

掌握使用 BTB 和 BHT 进行分支预测的方法。

二. 实验环境和工具

环境：Windows 10
工具：Vivado 2017

三. 实验内容和过程

1. BTB

```
10      parameter BRANCH_ADDR_LEN = 4;
11      parameter BRANCH_SIZE = 1<<BRANCH_ADDR_LEN;
12      reg [31:0] BranchPC [BRANCH_SIZE-1:0];
13      reg [31:0] PredictPC [BRANCH_SIZE-1:0];
14      reg valid [BRANCH_SIZE-1:0];
15
16      wire [BRANCH_ADDR_LEN-1:0] pc_addr;
17      wire [BRANCH_ADDR_LEN-1:0] new_pc_addr;
18      wire [BRANCH_ADDR_LEN-1:0] result_pc_addr;
19      assign result_pc_addr=PCE[BRANCH_ADDR_LEN+1:2];
20      assign pc_addr=PCF[BRANCH_ADDR_LEN+1:2];
21      assign new_pc_addr=PCD[BRANCH_ADDR_LEN+1:2];
```

定义 BranchPC 记录分支指令的地址，PredictPC 记录分支指令跳转的地址，valid 记录该指令上次是否跳转，BTB 总大小为 BRANCH_SIZE。

pc_addr、new_pc_addr、result_pc_addr 分别为 PCF、PCD、PCE 的第 BRANCH_ADDR_LEN+1 位到第二位，因为正常对齐取指时 PC 的第 1 位和第 0 位都是 0，因此无需记录。其中 pc_addr 用于在 IF 阶段寻址 BTB 做出分支预测；new_pc_addr 用于在 ID 阶段寻址 BTB 添加新的分支指令；result_pc_addr 用于在 EX 阶段寻址 BTB 更新 valid 位。

```
24      // IF hit or miss
25      always @(*) begin
26          if (PCF==BranchPC[pc_addr]) begin
27              if (valid[pc_addr]==1) begin
28                  NPCPred<=PredictPC[pc_addr];
29                  preBranchF<=1;
30              end
31              else begin
32                  NPCPred<=PCF+4;
33                  preBranchF<=0;
34              end
35          end
36          else begin
37              NPCPred<=PCF+4;
38              preBranchF<=0;
39          end
40      end
```

在 IF 阶段，若 PCF 值与 BTB 中相应表项相同且该 valid 位为 1，表示该指令为分支指令且上次跳转，因此预测其跳转并读出相应跳转地址。否则预测其不跳转，取下一条指令地址为 PCF+4。

```
43      // insert new record
44      integer i;
45      always@(posedge clk or posedge rst) begin
46          if(rst) begin
47              for(i=0;i<BRANCH_SIZE; i=i+1) begin
48                  BranchPC[i] <= 0;
49                  PredictPC[i] <= 0;
50                  valid[i] <= 0;
51              end
52          end
53          else begin
54          if (BranchTypeD!=`NOBRANCH && BranchPC[new_pc_addr] != PCD) begin
55              BranchPC[new_pc_addr] <= PCD;
56              PredictPC[new_pc_addr] <= BranchTargetD;
57              valid[new_pc_addr]<=1;
58          end
59          end
60      end
```

在 ID 段，如果译码发现指令是分支指令，且其不在 BTB 中，则将其加入 BTB，初始 valid 置为 1。

值得注意的是，此处更新是时序逻辑电路，BranchPC、PredictPC 和 valid 都被期望综合成寄存器，因此使用 always@ (posedge clk)。若使用 always@(*)则会将这段电路综合成组合逻辑，与期望不符，实验结果也会产生难以预料的错误。

```
62      // update
63      always@(posedge clk) begin
64          if (BranchPC[result_pc_addr]==PCE) begin
65              valid[result_pc_addr] <= BranchE;
66          end
67      end
```

在 EX 段，若当前指令存在于 BTB 相应表项中，则记录其本次分支结果 BranchE。值得注意的是，此处更新也是时序逻辑电路，原因与上文中 ID 段的相同。

2. BHT

BHT 主体部分与 BTB 类似，在本报告中仅叙述其不同之处。
此时 valid 寄存器组应为 2 位，表示每一表项的 BHT 状态机。

```
24      // IF hit or miss
25      always @(*) begin
26          if (PCF==BranchPC[pc_addr]) begin
27              if (valid[pc_addr]==2'b11 || valid[pc_addr]==2'b10) begin
28                  NPCPred<=PredictPC[pc_addr];
29                  preBranchF<=1;
30              end
31              else begin
32                  NPCPred<=PCF+4;
33                  preBranchF<=0;
34              end
35          end
36          else begin
37              NPCPred<=PCF+4;
38              preBranchF<=0;
39          end
40      end
```

在 IF 段，若 valid 位为 11 或 10 均预测其分支成功。

```
43      integer i;
44      always@(posedge clk or posedge rst) begin
45          if(rst) begin
46              for(i=0;i<BRANCH_SIZE; i=i+1) begin
47                  BranchPC[i] <= 0;
48                  PredictPC[i] <= 0;
49                  valid[i] <= 2'b0;
50              end
51          end
52          else begin
53          if (BranchTypeD!=`NOBRANCH && BranchPC[new_pc_addr] != PCD) begin
54              BranchPC[new_pc_addr] <= PCD;
55              PredictPC[new_pc_addr] <= BranchTargetD;
56              valid[new_pc_addr]<= 2'b11;
57          end
58          end
59      end
```

在 ID 段，插入新表项时 valid 置为 11。

```
61      // update
62      always@(posedge clk) begin
63          if (BranchPC[result_pc_addr]==PCE) begin
64              if (BranchE ==1 && valid[result_pc_addr]!=2'b11) begin
65                  valid[result_pc_addr] <= valid[result_pc_addr] + 2'b01;
66              end
67              if (BranchE == 0 && valid[result_pc_addr]!=2'b00) begin
68                  valid[result_pc_addr] <= valid[result_pc_addr] - 2'b01;
69              end
70          end
71      end
```

在 EX 段，更新 valid 时不仅依据本次分支结果 BranchE，也与 valid 现在的状态相关。此时构成一个分支预测的 2 位状态机。

3. 其他修改模块

NPC_Generator：

```
always @(*)
begin
    if(JalrE)
        PC_In <= JalrTarget;
    else if(BranchE == 1 && preBranchE == 0)
        PC_In <= BranchTarget;
    else if(BranchE == 0 && preBranchE == 1)
        PC_In <= PCE + 4;
    else if(JalD)
        PC_In <= JalTarget;
    else
        PC_In <= NPCPred;
end
```

修改部分：若为分支指令，预测错误分两种情况：预测不跳转实际跳转；预测跳转实际不跳转。PC_In 应与实际情况符合，实际跳转则设为 BranchTarget，实际不跳转则设为 PCE+4。若既不是 J 指令也不是预测错的分支指令，则 PC_In 设为来自 BTB 或 BHT 的预测 PC 值。

注：对于非 J 非分支的指令，NPCPred 始终为 PCF+4，具体操作可见 BTB 和 BHT 的实现。

IDSegReg：

增加输入 preBranchF 和输出 reg preBranchD，用于记录当前指令预测跳转与否。

EXSegReg：

增加输入 preBranchD 和输出 reg preBranchE，用于记录当前指令预测跳转与否。

HarzardUnit：

```
else if((BranchE != preBranchE) | JalrE)
    {StallF, FlushF, StallD, FlushD, StallE, FlushE, StallM, FlushM, StallW, FlushW} <= 10'b0001010000;
```

改变分支指令时 FlushD 和 FlushE 的条件，此时为预测失败时 FlushD、FlushE，预测成功时不进行 Flush。

RV32Core：

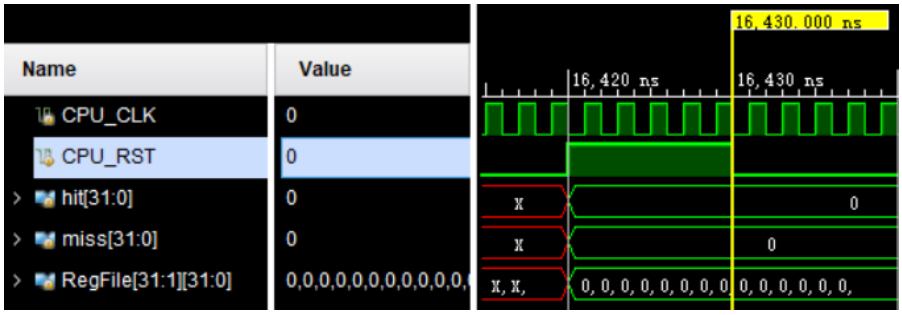
除了 BTB、BHT 和以上修改模块的连线外，增加统计分支预测成功和失败的寄存器。

```
160     reg [31:0] hit;
161     reg [31:0] miss;
162     always@(posedge CPU_CLK or posedge CPU_RST) begin
163         if(CPU_RST) begin
164             hit <= 32'b0;
165             miss <= 32'b0;
166         end
167         else begin
168             if(BranchTypeE != `NOBRANCH) begin
169                 if(BranchE == preBranchE) hit <= hit + 1;
170                 else miss <= miss + 1;
171             end
172         end
173     end
```

在指令执行到 EX 时判断是否为分支指令和预测是否成功，注意此处统计时应该先判断 EX 段的指令是否是分支指令。

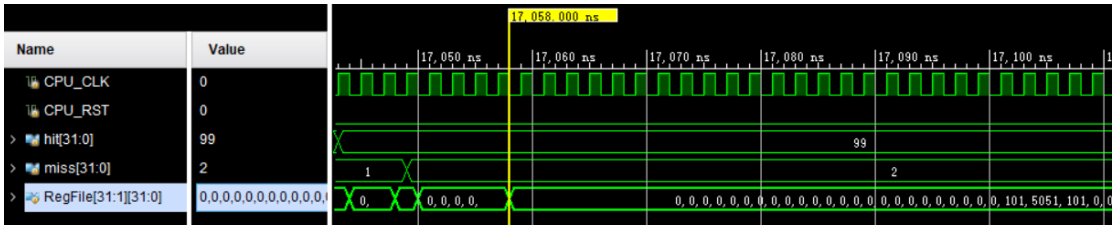
四． 实验结果

本节展示 BTB 和 BHT 两种分支预测分别测试 BTB.S 和 BHT.S 的实验结果。
一个周期 2ns。
对于所有仿真，CPU_RST 结束的时间都是 16430ns



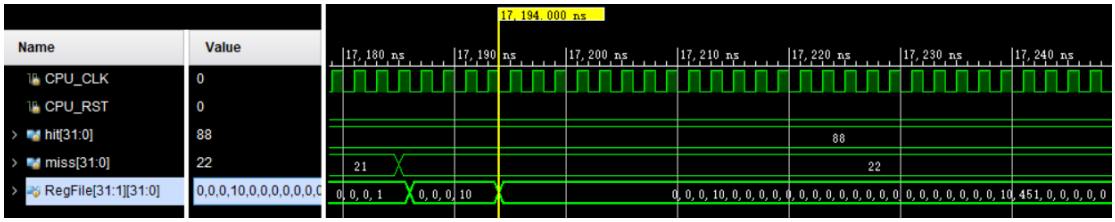
1. BTB

1.1 BTB.S



17058ns 结束，总周期数(17058-16430)/2 = 314。
分支预测 Hit 99 次， Miss 2 次。

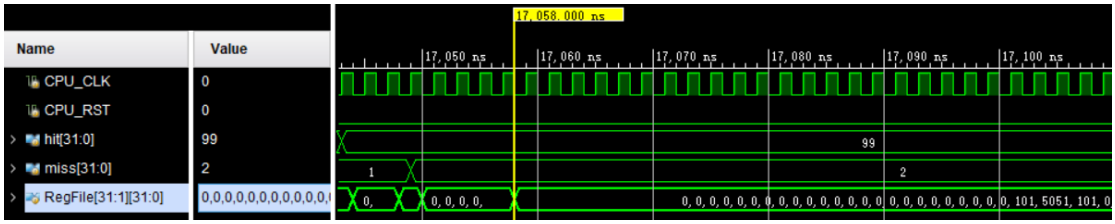
1.2 **BHT.S**



17194ns 结束，总周期数(17194-16430) / 2 = 382。
分支预测 Hit 88 次， Miss 22 次。

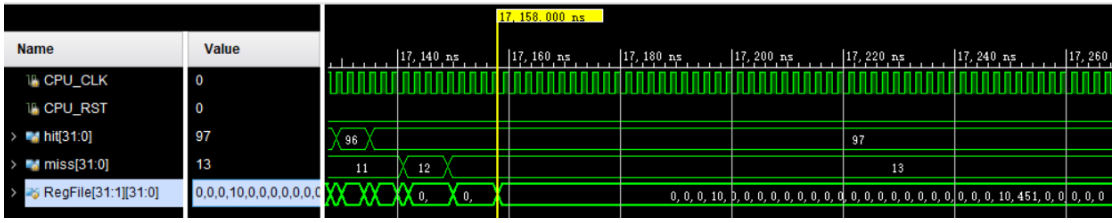
2. BHT

2.1 **BTB.S**



17058ns 结束，总周期数(17058-16430)/2 = 314。
分支预测 Hit 99 次， Miss 2 次。

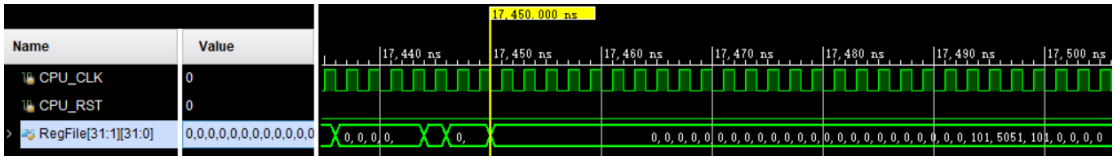
2.2 **BHT.S**



17158ns，总周期数(17158-16430)/2 = 364。
分支预测 Hit 97 次， Miss 13 次。

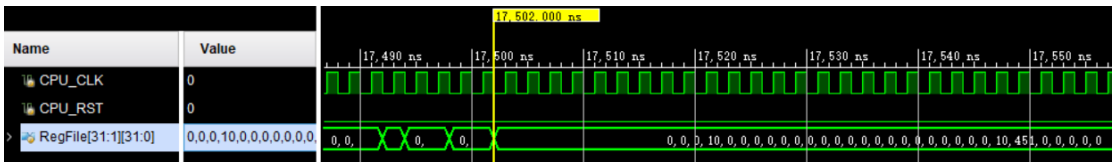
3. 无分支预测 CPU

3.1 **BTB.S**



17450ns 结束，总周期数(17450-16430)/2 = 510。

3.2 BHT.S



17502ns 结束，总周期数(17502-16430)/2 = 536。

五． 实验分析

1. BTB.S

总共 307 条指令，其中 101 条分支指令。

表一 BTB.S 实验结果

	Hit 数	Miss 数	总周期	与无分支预测周期数差值	加速比	CPI
无分支预测	-	-	510	-	-	1.6612
BTB	99	2	314	196	1.6242	1.0228
BHT	99	2	314	196	1.6242	1.0228

2. BHT.S

总共 335 条指令，其中 110 条分支指令。

表二 BHT.S 实验结果

	Hit 数	Miss 数	总周期	与无分支预测周期数差值	加速比	CPI
无分支预测	-	-	536	-	-	1.600
BTB	88	22	382	154	1.4031	1.1403
BHT	97	13	364	172	1.4725	1.0866

3. 对比分析

分支收益：预测成功时减少流水线因为控制相关而导致的 Flush，每预测成功一条分支指令可以节约两个周期。

分支代价：本次实验中预测失败时消耗周期数与无预测相同，均为两个周期，可以理解为在仿真中预测失败相较于无预测情况无额外代价。但在实际情况中，预测失败后消耗的周期数往往比无预测情况要多，此时需要评估分支预测带来的收益是否大于代价。

对比表一表二可以发现，对于 BTB.S，BTB 和 BHT 的效果一样，因为该测试

程序中仅有一个一层循环，BHT 的 2 位状态机预测发挥不出效果。BTB 与 BHT 均为进入第一次循环时有一次预测失败，最后一次循环退出时有一次预测失败。

对于 BHT.S，该测试程序中有两重循环，此时可以体现出 BHT 中 2 位状态机的效果，BHT 比 BTB 少 miss 9 次。

六. 实验总结

总结本次实验遇到的问题：

对于希望将 reg 综合成寄存器的电路，应写为时序逻辑电路，如 always@(posedge clk)。若写为 always@(*)，则会综合成组合逻辑电路。对于 BranchPC、PredictPC、valid、hit、miss 这些寄存器而言，都应该使用时序逻辑更新，若使用组合逻辑则会产生无法预料的错误结果。

七. 改进意见

暂无，感谢助教一学期的帮助。