Lab4

实验目的

在已有的 RISCVcore 上实现

1. BTB 分支预测
2. 在 BTB 的基础上实现 BHT 预测

实验环境与工具

使用 windows10 下的 verilog 仿真工具

实验内容与过程

BTB 主要代码如下。

1. 当输入的 PC 在 BTB 表中且对应的预测位为 1，则预测这次为跳转，并输出跳转的 PC。
2. 当预测结果与跳转结果不一致的时候更新 BTB 表

```
wire update_req = BTB_E ^ Branch_E;
```

```
always @ (*) begin
    if(PCTag[Branch_buffer_addr] == Branch_tag_addr && PredictStateBit[Branch_buffer_addr])
        Branch_predicted <= 1'b1;
    else
        Branch_predicted <= 1'b0;
    Branch_predicted_PC <= PredictPC[Branch_buffer_addr];
end
always @ (posedge clk or posedge rst) begin
    if(rst) begin
        for(integer i = 0; i < BUFFER_SIZE; i = i + 1) begin
            PCTag[i] <= 0;
            PredictPC[i] <= 0;
            PredictStateBit[i] <= 1'b0;
        end
        Branch_predicted <= 1'b0;
        Branch_predicted_PC <= 0;
        //Branch_taken = 1;
        //Branch_predicted <= 1'b0;
    end
    else begin//更新
        if(update_req) begin
            PCTag[ex_buffer_addr] <= ex_tag_addr;
            PredictPC[ex_buffer_addr] <= ex_predicted_PC;
            PredictStateBit[ex_buffer_addr] <= Branch_E;
        end
    end
end
```

Hazard 中，若是预测结果与实际跳转结果不一致则清空 ID 与 EX

```
end
else if((br^(Branch_takenE && BTB_E)) | jalr)
    {flushF, bubbleF, flushD, bubbleD, flushE, bubbleE, flushM, bubbleM, flushW, bubbleW} <= 10'b0010100000;
```

NPC 中，

1. 若是实际跳转但之前预测不跳转，则选择实际跳转的 PC。
2. 若是实际不跳转但是预测跳转，则从 EX 段对应指令的下一条开始执行。
3. 若是 BTB 预测跳转则选择预测的 PC

```
end
else if (br==1 && !(BTB_E==1 && Branch_takenE == 1)) begin
    NPC <= br_target;
end
else if (br==0 && (BTB_E==1 && Branch_takenE == 1) )
    NPC <= PC_EX;
else if (jal==1) begin
    NPC <= jal_target;
end
else if(BTB_F==1 && Branch_takenF==1)
    NPC <= BTB_PCF;
else begin
```

BHT 的实现参照下表

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	Br_target	N
Y	N	N	PC_IF+4	N	PC_EX+4	Y
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

代码如下，状态 0.1 表示不跳转，状态 2，3 表示跳转

```
//BHT
reg [1 : 0] state [BUFFER_SIZE-1:0];
always @ (*) begin
    Branch_taken <= (state[Branch_buffer_addr] >= 2'b10);
end
always @ (posedge clk or posedge rst) begin
    if(rst) begin
        for(integer i = 0; i < BUFFER_SIZE; i = i + 1) begin
            state[i] <= 2'b00;
        end
        Branch_taken <= 0;
    end else begin
        if(BHT_update) begin
            if(Branch_E) begin
                if(state[ex_buffer_addr] != 2'b11)
                    state[ex_buffer_addr] <= state[ex_buffer_addr] + 2'b01;
                else
                    state[ex_buffer_addr] <= state[ex_buffer_addr];
            end else begin
                if(state[ex_buffer_addr] != 2'b00)
                    state[ex_buffer_addr] <= state[ex_buffer_addr] - 2'b01;
                else
                    state[ex_buffer_addr] <= state[ex_buffer_addr];
            end
        end
    end
end
```

实验结果

BHT 的实现参照下表

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	Br_target	N
Y	N	N	PC_IF+4	N	PC_EX+4	Y
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

我所提交的代码为 BHT，要实现 BTB 只需注释 BHT 部分并设 Branch_taken 永远为 1。
不预测即为预测不跳转。只需设 Branch_predict 永远为 0 即可
实验截图在最后

BTB 结果统计如下

策略	跳转指令数	预测成功次数	预测失败次数	用时/ns	总周期数	周期数与静态的差值
BTB	101	99	2	1256	313	196
BHT	101	98	3	1264	315	198
不预测	101	1	100	2040	509	--

BHT 结果统计如下

策略	跳转指令数	预测成功次数	预测失败次数	用时/ns	总周期数	周期数与静态的差值
BTB	110	88	22	1528	381	154
BHT	110	95	15	1476	367	168
不预测	110	11	99	2144	535	--

矩阵乘法结果，开始时间为 12ns，忽略不计

策略	跳转指令数	预测成功次数	预测失败次数	用时/ns	总周期数	周期数与静态的差值
BTB	4623	4072	545	1176264	294065	7546
BHT	4623	4076	547	1176232	294057	7554
不预测	4623	273	4350	1206448	301611	--

快排结果，开始时间为 12ns，忽略不计

策略	跳转指令数	预测成功次数	预测失败次数	用时/ns	总周期数	周期数与静态的差值
BTB	6614	4728	1886	161338	40333	-956
BHT	6614	5495	1119	155202	38799	578
不预测	6614	5206	1409	157514	39377	--

实验分析

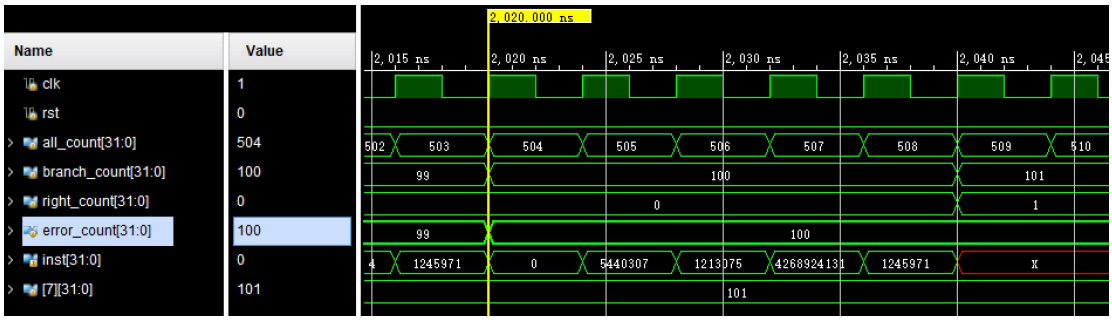
1. 若预测跳转且实际跳转，则比不预测要少两个周期，因为不需要清空 ID，EX，分支收益为 2cycle
2. 若预测跳转且实际未跳转，则比不预测要多两个周期，因为需要清空 ID，EX，开销为 2cycle
3. 从整体上看，三种策略的效率为 BHT>BTB>不预测
4. btb.S 中 BHT 策略不如 BTB 策略是因为其中只有一个循环
5. bht.S 中 BHT 策略优于 BTB 策略是因为其中有两层循环
6. 快排中效率 BHT>不预测>BTB 是因为快排的跳转收实际数据的影响很大，没有明确规律，预测很容易失败。
7. 矩阵乘法中效率 BHT>BTB>不预测 是因为矩阵乘法有循环，较有规律，预测效果较好

实验小结

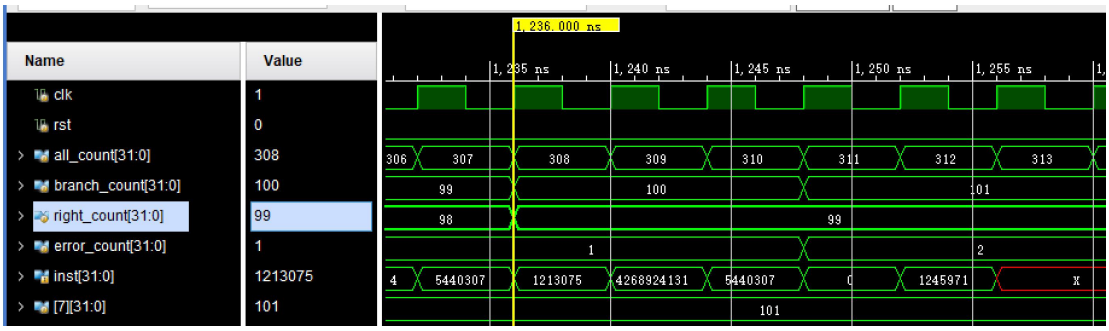
通过本次实验我初步掌握了 BTB 与 BHT 预测策略的原理，并了解了这两个策略在实际运用中的效率情况。

实验截图

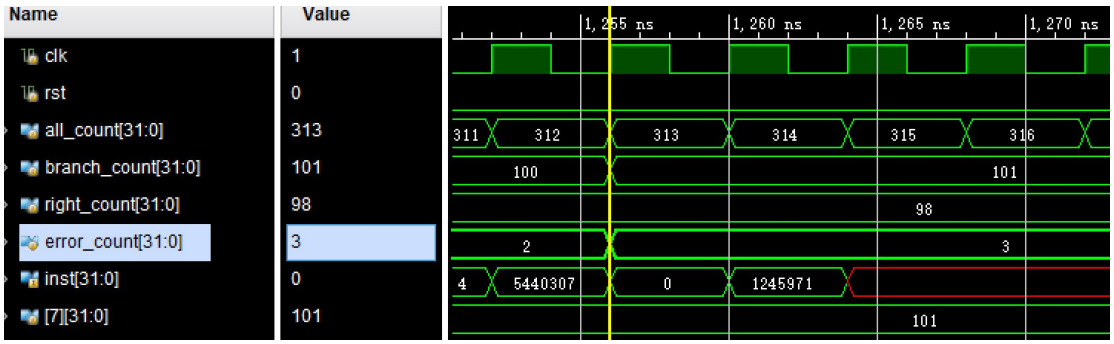
btb.S 不预测



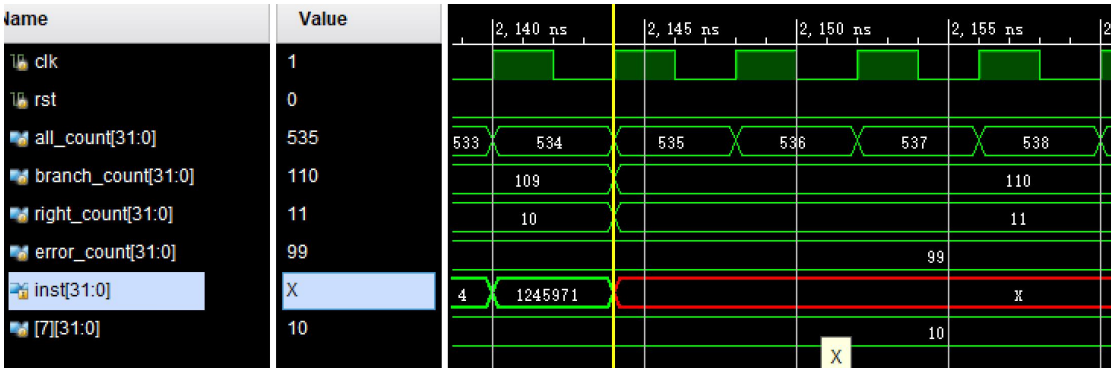
btb.S BTB 策略



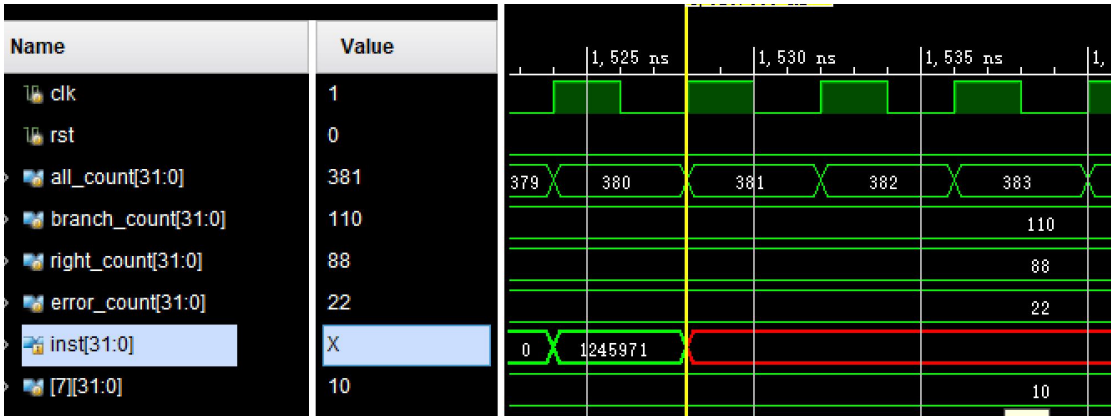
btb.S BHT 策略



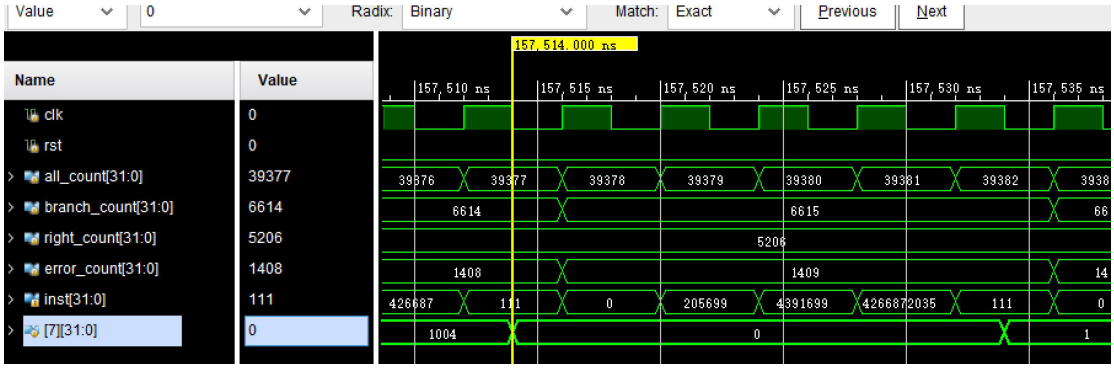
bht.S 不预测



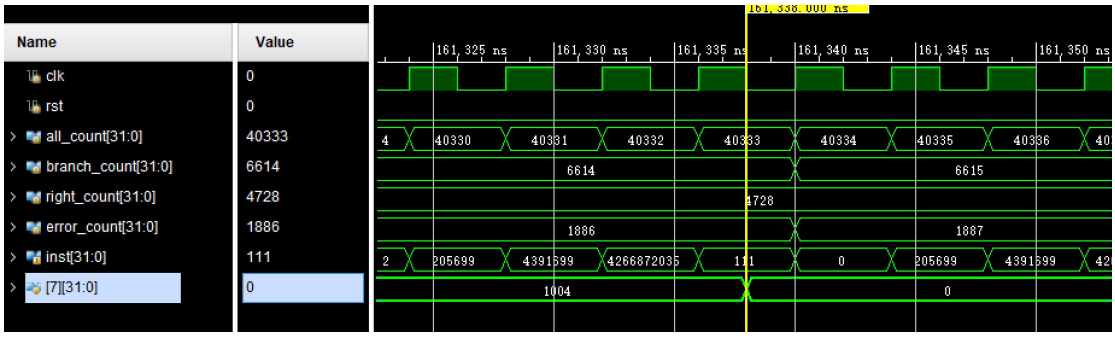
Bht.S BTB 策略



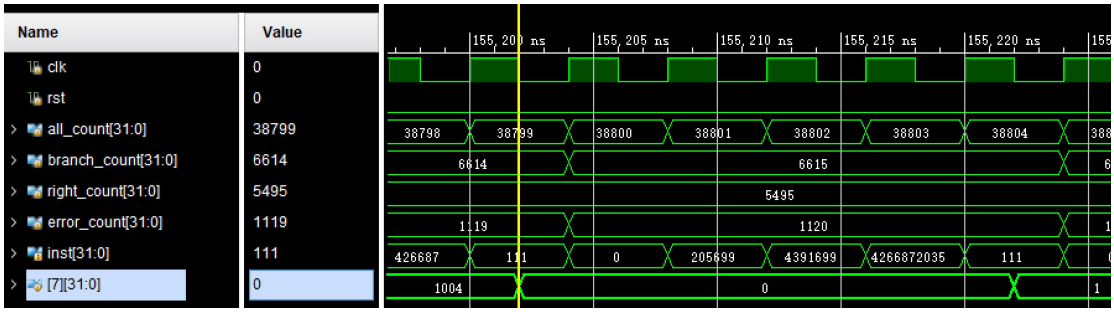
快排 不预测



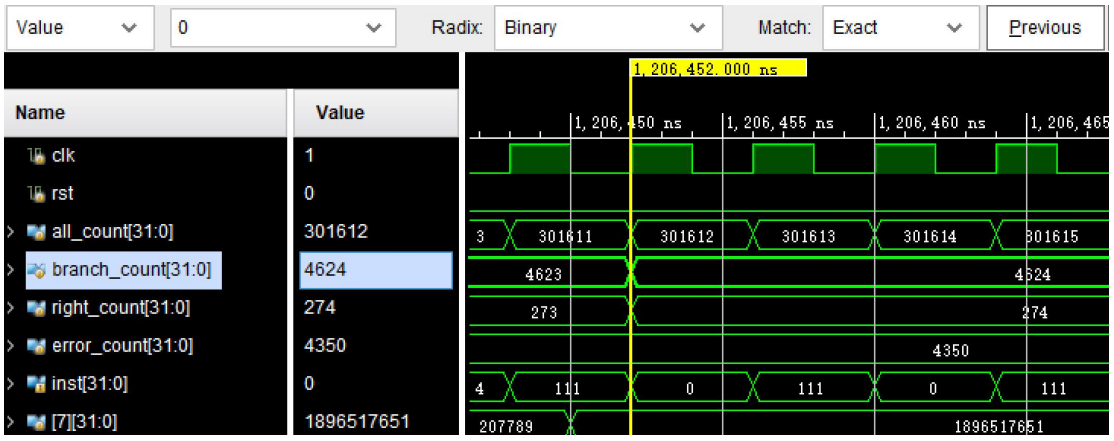
快速排序 BTB 策略



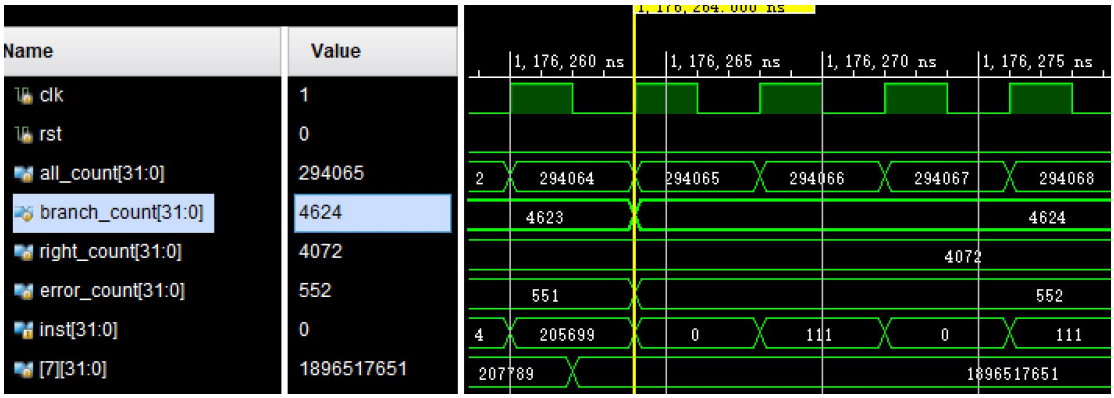
快排 BHT 策略



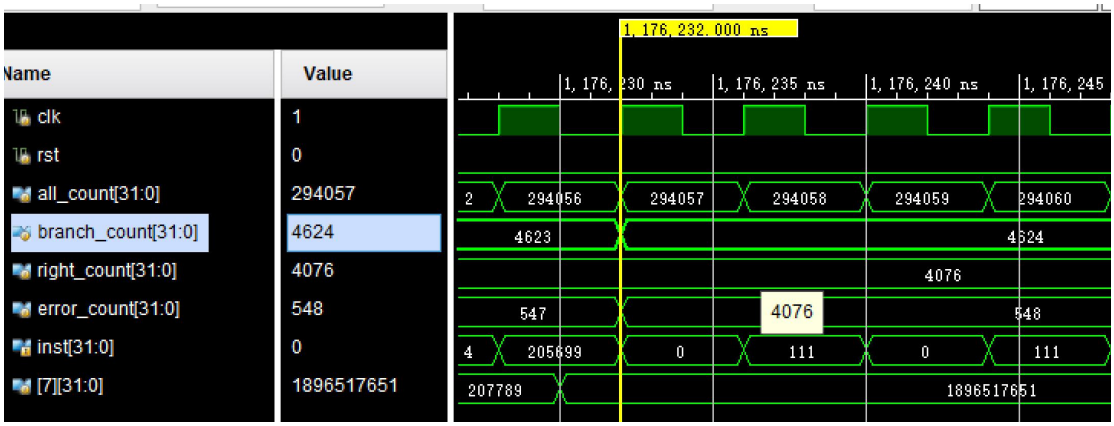
矩阵 不预测



矩阵 BTB 策略



矩阵 BHT 策略


Name
Value
clk
1
rst
0
all_count[31:0]
294057
branch_count[31:0]
4624
right_count[31:0]
4076
error_count[31:0]
548
inst[31:0]
0
[7][31:0]
1896517651
1, 176, 232.000 ns
1, 176, 230 ns
1, 176, 235 ns
1, 176, 240 ns
1, 176, 245
2
294056
294057
294058
294059
294060
4623
4624
4076
547
4076
548
4
205699
0
111
0
111
207789
1896517651