Lab4实验报告

PB18111757 陈金宝

实验过程

BTB

在BTB中维持一个BUFFER。用于存预测的PC。同时维持一个STAT数组。值为1或0。值为1时代表进行分支预测，否则不预测。默认的TAG_ADDR_LEN为32-6=26。

```
localparam TAG_ADDR_LEN = 32 - SET_ADDR_LEN;
localparam SET_SIZE     = 1 << SET_ADDR_LEN;
reg [TAG_ADDR_LEN-1:0] btb_tags   [SET_SIZE];
reg [31:0]             btb_pred   [SET_SIZE];
reg                    btb_stat   [SET_SIZE];
```

当tag匹配，且对应stat为1时则进行预测

```
assign PC_SEL = btb_hit&&IFstat;

assign IFstat = btb_stat[pcf_set];
assign btb_hit = (btb_tags[pcf_set] == pcf_tag);
```

在EX段判断预测是否预测成功。不成功则要flush错误装载的指令。预测失败分为:预测taken但没taken；不预测taken但taken。

```
assign btb_prefail = EXstat && (!BranchE);
assign btb_fill = (!EXstat) && BranchE;
assign btb_flush = btb_prefail | btb_fill;
```

在EX段进行更新对应寄存器的值。若预测taken但没taken则将对应stat设为0。若不预测taken但taken。则将对应的指令和BranchTarget装入对应寄存器中。

```
always @(negedge clk or posedge rst)begin
    if(rst)begin
        for(integer i = 0;i < SET_SIZE;i++)begin
            btb_tags[i] <= 0;
            btb_pred[i] <= 0;
            btb_stat[i] <= 0;
        end
    end
    else if(!StallE)begin
        if(btb_prefail)begin
            btb_stat[pce_set] <= 0;
        end else begin
            if(btb_fill)begin
                btb_tags[pce_set] <= pce_tag;
                btb_pred[pce_set] <= BranchTarget;
                btb_stat[pce_set] <= 1;
            end
```

```
        end
    end
end
```

BHT

将BTB接入BHT。在BHT中维持一个2-BIT的状态位。当状态为10或11时BHT预测taken。当BTB,BHT两者均预测taken才预测taken。

```
BTB #(SET_ADDR_LEN)BTB1(
    .clk(clk),
    .rst(rst),
    .PCF(PCF),
    .BranchE(BranchE),
    .BranchTypeE(BranchTypeE),
    .BranchTarget(BranchTarget),
    .StallD(StallD),
    .StallE(StallE),
    .FlushD(FlushD),
    .FlushE(FlushE),
    .PC_PRE(b_pcpre),
    .PC_SEL(b_pcsel),
    .btb_flush(b_flush),
    .btb_prefail(b_prefail),
    .btb_fill(b_fill),
    .PCE(PCE)
);
reg [TAG_ADDR_LEN-1:0] bht_tags   [SET_SIZE];
reg [1:0]              bht_stat   [SET_SIZE];
```

当均预测taken时才预测taken

```
assign PC_SEL = bht_hit && b_pcsel && IFstat[1];

assign pre_taken = PC_SEL;
assign IFstat = bht_stat[pcf_set];
assign bht_hit = (bht_tags[pcf_set] == pcf_tag);
```

在EX段根据stat的旧值和是否Branch来更新stat。

```
always@(*)begin
    if(BranchE)begin
        init_stat <= 2'b01;
        if(EXstat[1])begin
            next_stat <= 2'b11;
        end else begin
            if(EXstat[0])begin
                next_stat <= 2'b10;
            end else begin
                next_stat <= 2'b01;
            end
        end
    end else begin
        init_stat <= 2'b00;
        if(EXstat[1])begin
```

```
            next_stat <= EXstat << 1;
        end else begin
            next_stat <= 2'b00;
        end
    end
end

always @(negedge clk or posedge rst)begin
    if(rst)begin
        for(integer i = 0;i < SET_SIZE;i++)begin
            bht_tags[i] <= 0;
            bht_stat[i] <= 0;
        end
    end
    else if(!StallE)begin
        if(EXhit)begin
            bht_stat[pce_set] <= next_stat;
        end else begin
            if(|BranchTypeE)begin
                bht_tags[pce_set] <= pce_tag;
                bht_stat[pce_set] <= init_stat;
            end
        end
    end
end
```

预测失败次数统计

在RV32CORE中，根据BranchTypeE和BTH_FLUSH来进统计。可得到总的branch指令数和预测失败的次数。

```
reg [31:0] total_br_cnt;
reg [31:0] err_br_cnt;
always @(negedge CPU_CLK or posedge CPU_RST) begin
    if(CPU_RST)begin
        total_br_cnt <= 0;
    end else begin
        if(|BranchTypeE)begin
            total_br_cnt <= total_br_cnt + 1;
        end
    end
end

always @(negedge CPU_CLK or posedge CPU_RST) begin
    if(CPU_RST)begin
        err_br_cnt <= 0;
    end else begin
        if(BTB_FLUSH)begin
            err_br_cnt <= err_br_cnt + 1;
        end
    end
end
```

结果分析

测试时不使用cache。BTB/BHT中SET_ADDR_LEN为6（即buffer大小为64）。

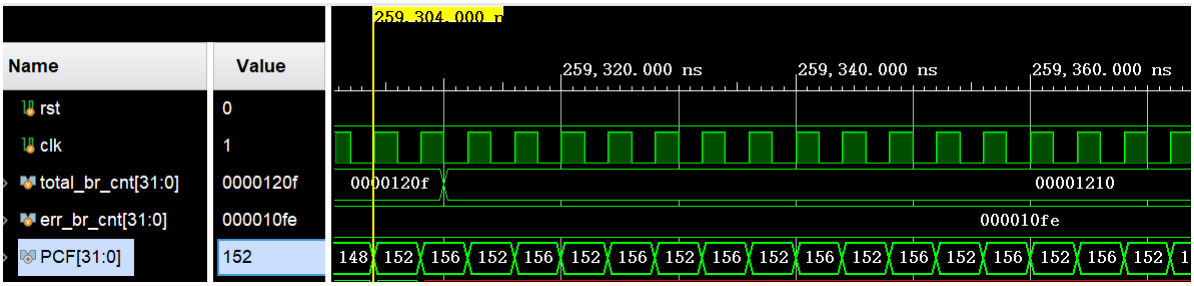
分别使用BHT.S,BTB.S,QUICKSORT.S,MATMUL.S进行测试

预测失败时需要FLUSH掉ID和EX。所以会带来2周期的penalty。分支成功时不需要FLUSH，没有这2周期的penalty。

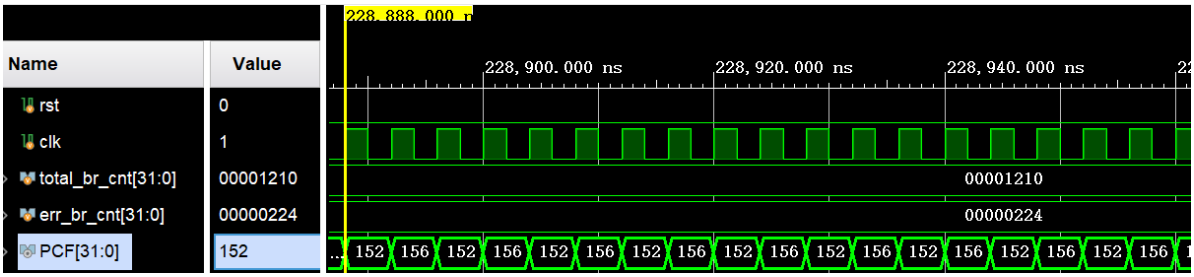
得到测试结果如下：

- 16×16矩阵乘法:

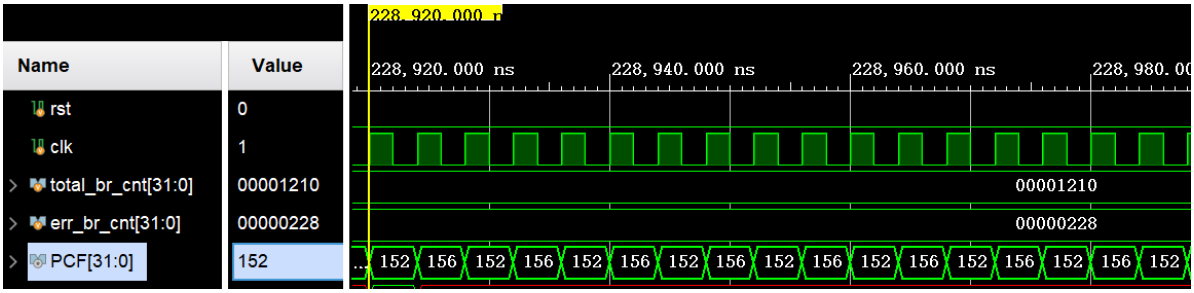
不进行分支预测:



只有BTB:



BHT:



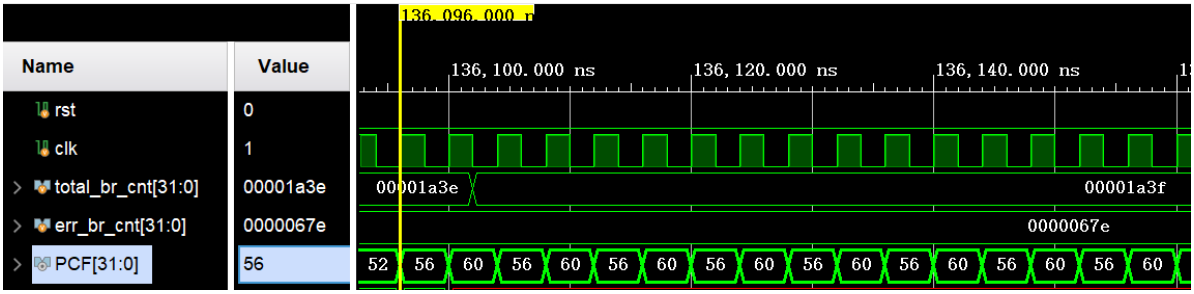
将上述结果制成表格如下:

策略	运行时间(ns)	与不预测时间差(ns)	Branch指令数	预测成功数	预测错误率(%)
不预测(即全部预测不taken)	259,304	0	4624	274	94.07
BTB	228,888	-30,416	4624	4076	11.85
BHT	228,920	-30,384	4624	4072	11.94

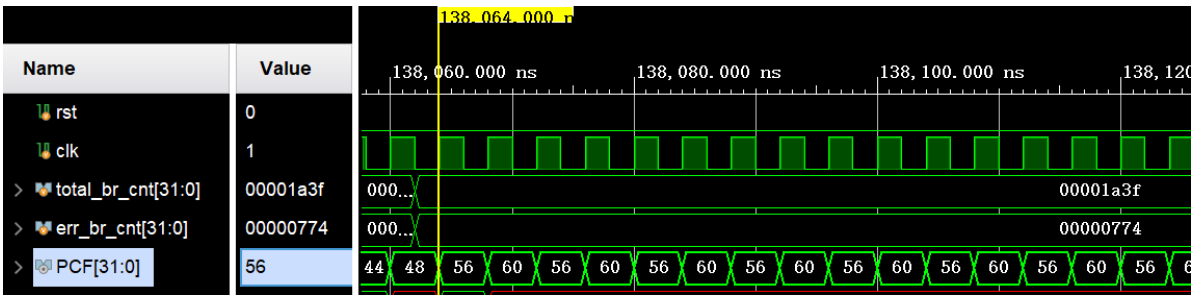
由上述表格可知，使用分支预测时，明显优于不进行分支预测，错误率下降明显。且此时BHT和BTB预测的准确率基本相当。BTB略优于BHT。

- 256个数快排:

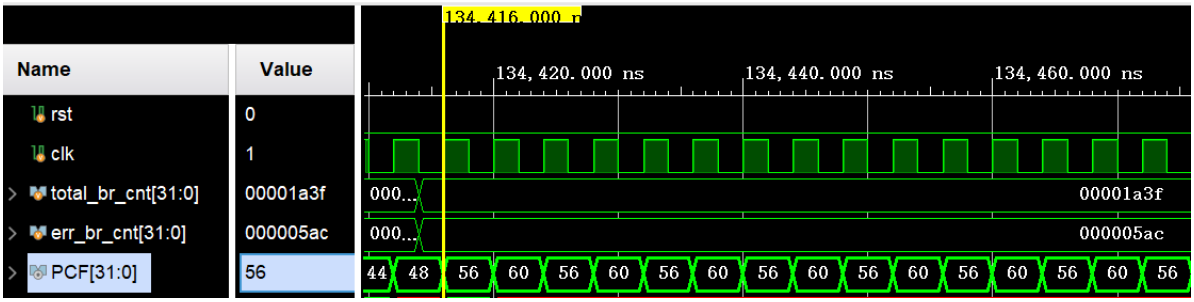
不进行分支预测:



BTB:



BHT:



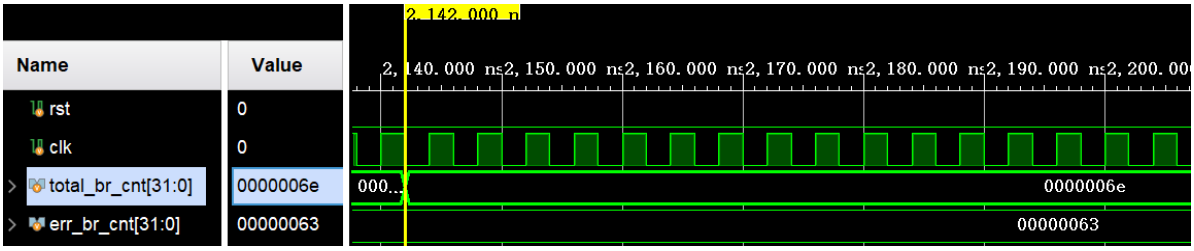
将上述结果制成表格如下:

策略	运行时间(ns)	与不预测时间差(ns)	Branch指令数	预测成功数	预测错误率(%)
不预测(即全部预测不taken)	136,096	0	6719	5057	24.74
BTB	138,064	1,968	6719	4811	28.40
BHT	134,416	-1,680	6719	5267	21.61

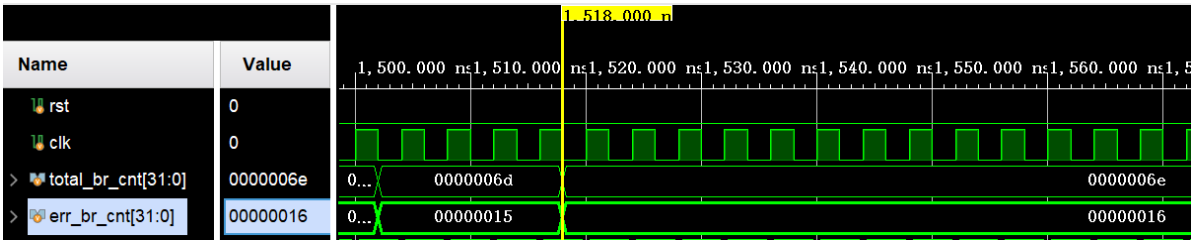
由上述表格可知。此时BTB略差于不预测。BHT优于不预测（和随机生成的256个数的分布有关）。此时BHT是较优的策略

- BHT.S:

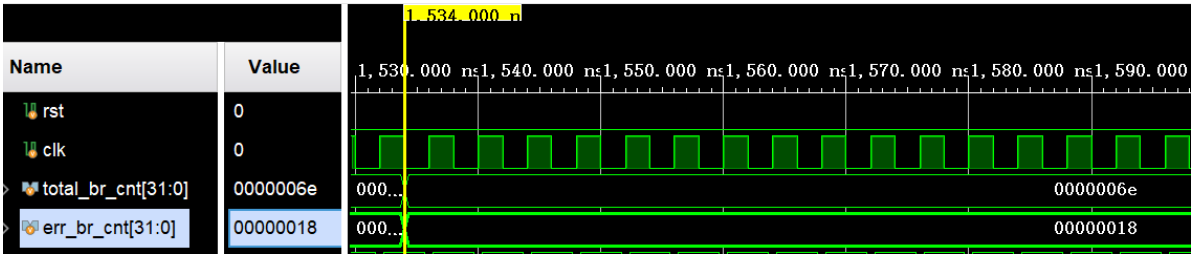
不进行分支预测:



BTB:



BHT:



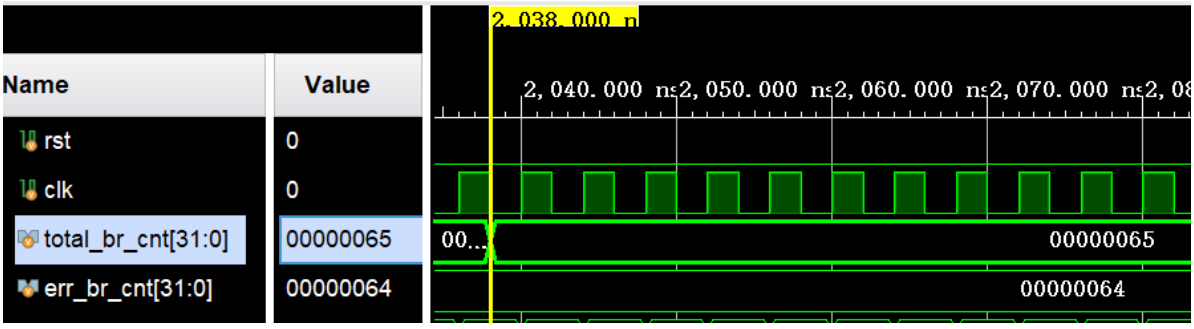
将上述结果制成表格如下:

策略	运行时间(ns)	与不预测时间差(ns)	Branch指令数	预测成功数	预测错误率(%)
不预测(即全部预测不taken)	2,142	0	110	11	90
BTB	1,518	-624	110	88	20
BHT	1,534	-608	110	86	21.82

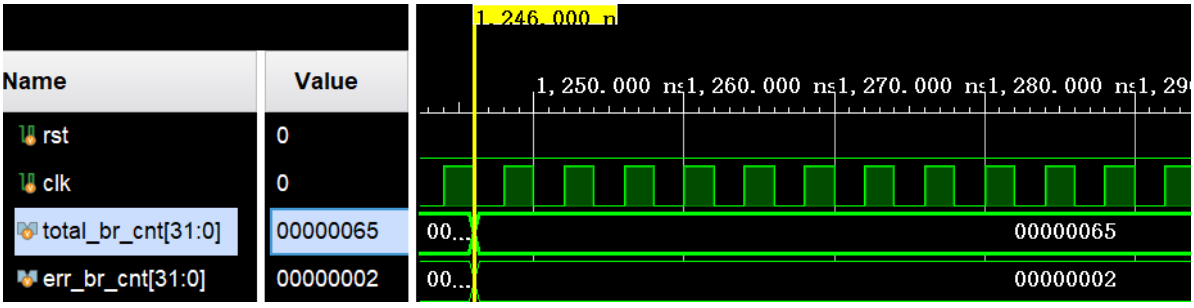
由上述表格可知。预测远优于不预测。BHT与BTB相当。

- BTB.S:

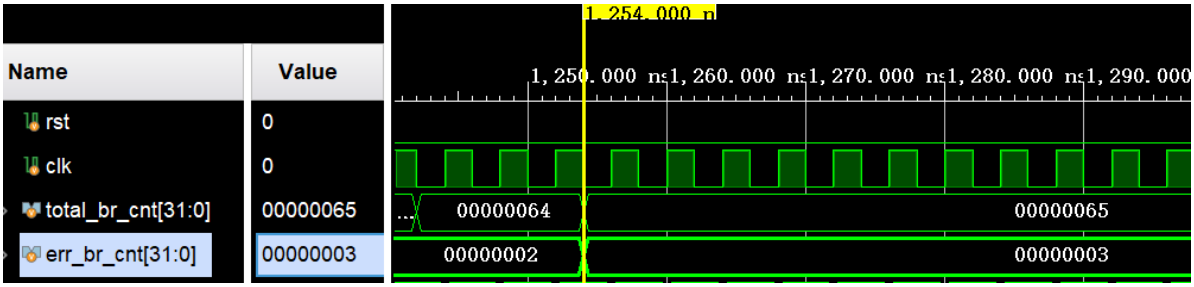
不进行分支预测:



BTB:



BHT:



将上述结果制成表格如下:

策略	运行时间(ns)	与不预测时间差(ns)	Branch指令数	预测成功数	预测错误率(%)
不预测(即全部预测不taken)	2,038	0	101	1	99.01
BTB	1,246	-792	101	99	1.98
BHT	1,254	-784	101	98	2.97

由上述表格可知。预测远优于不预测。BHT与BTB相当。

综上：在进行快排时，BHT较优，其余情况BTB较优，但BHT与之差距不大。

实验总结

熟悉了BTB和BHT，理解了分支预测的原理。分析了不同预测策略在不同场景的表现。

附注:

附件中包含的代码说明
`BTB.sv`： BTB源代码
`HazardUnit.v`： 修改过的HazardUnit源代码
`NPC_Generator.v`： 修改过的NPC_generator源代码
`PRED.sv`:BHT源代码
`RV32Core.v`:修改过的RV32Core源代码
本次实验只改动了上述代码