Lab4 实验报告

PB18071463 朱映 2021.6.6

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

使用的硬件：xc7a100tcsg324-1

- **实验工具**：Vivado，MacBook Pro 13，Windows 10，VS CODE
- **实验方式**：Vivado自带的波形仿真

实验内容

阶段一：在Lab3阶段二的RV32I Core基础上，实现BTB

阶段二：在阶段一的基础上实现BHT

编写模块

1. BTB

   BTB类似于直接映射的 Cache，用当前 PC 作为 tag 和索引，存储目标 PC 地址。每次遇到未记录的分支指令，就将其记录；遇到记录的分支指令但是没有跳转，就设置有效位为 0 从而将其删去。

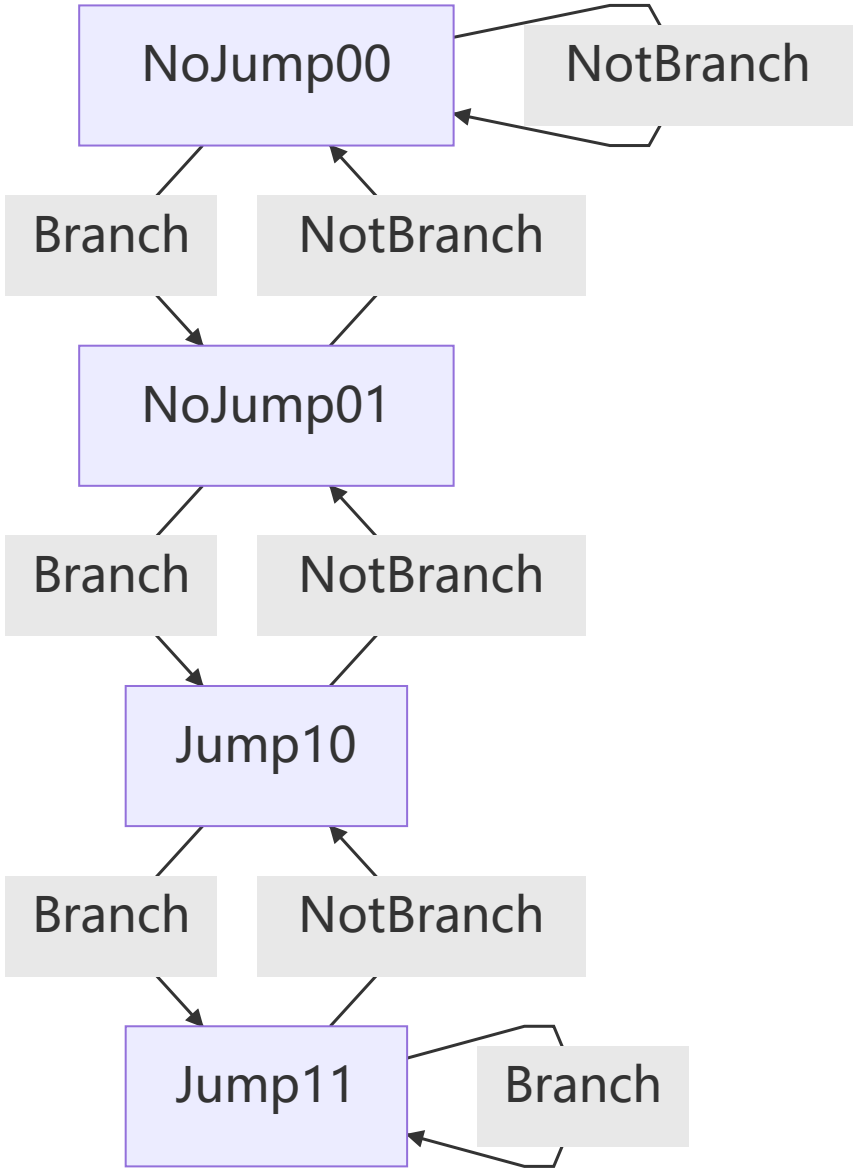
   核心代码如下。

```
always@(*) begin    // BTB hit信号和读出缓存
        if (!rst && BTBvalid[PC_ADDR] && BTBTAG[PC_ADDR] == PC[31:ADDR_LEN])
begin
            BTB_hit <= 1;
            BTB_NPC <= BTBNPC[PC_ADDR];
        end
        else begin
            BTB_hit <= 0;
            BTB_NPC <= 0;
        end
    end

reg BTB_Del,BTB_Wr;
    always@(*) begin
        if (rst) begin
            BTB_Wr <= 0;
            BTB_Del <= 0;
        end
```

```
        else begin
            if (!BTB_hit_EX && BrInstr_EX && Branch) BTB_Wr <= 1;
            else BTB_Wr <= 0;
            if (BTB_hit_EX && BrInstr_EX && Branch) BTB_Del <= 1;
            else BTB_Del <= 0;
        end
    end

    integer j;
    always@(posedge clk) begin
        if (!StallF) begin
            if (rst) begin       // reset
                for (j = 0;j < BUFFER_SIZE;j = j + 1) begin
                    BTBTAG[j] <= 0;
                    BTBNPC[j] <= 0;
                    BTBvalid[j] <= 0;
                end
            end
            else begin      // 更新
                if (BTB_Wr && Branch) begin
                    BTBTAG[BranchPC_ADDR] <= PCE[31:ADDR_LEN];
                    BTBNPC[BranchPC_ADDR] <= BranchTarget;
                    BTBvalid[BranchPC_ADDR] <= 1;
                end
                else if (BTB_Del) begin    // 删除
                    if (BTBTAG[BranchPC_ADDR] == PCE[31:ADDR_LEN]) begin
                        BTBvalid[BranchPC_ADDR] <= 0;
                    end
                end
            end
        end
    end
```

2. BHT

   BHT比BTB更简单，它为每一条分支指令维护一个状态机，状态机如下。如果判断跳转，则从对应的 BTB 中寻找跳转目标。BTB 此时不需要删除功能，因此时它的功能主要作为一个存储 PC 的缓存。BTB 命中后将BHT的信号与之想与，如果两者都判断跳转才跳转，否则不跳转。



核心代码如下。

```
// State Machine
always @ (posedge clk) begin
if ( !rst && !StallF && BrInstr_EX) begin
       if (StateBuf[BranchPC_ADDR]!=2'b11 || StateBuf[BranchPC_ADDR]!=2'b00
)
            StateBuf[BranchPC_ADDR] <= Branch? StateBuf[BranchPC_ADDR]+1 :
StateBuf[BranchPC_ADDR]-1;
       else if (StateBuf[BranchPC_ADDR]==2'b11)
            StateBuf[BranchPC_ADDR] <= Branch? StateBuf[BranchPC_ADDR] :
StateBuf[BranchPC_ADDR]-1;
       else StateBuf[BranchPC_ADDR] <= Branch? StateBuf[BranchPC_ADDR]+1 :
StateBuf[BranchPC_ADDR];
       end
end
// BHT signal
always @ (*) begin
    if (!StallF) begin
        if (!rst && StateBuf[PC_ADDR][1]) BHT_jump <= 1;
        else BHT_jump <= 0;
```

```
    end
end
```

实验步骤

1. **分析分支收益和分支代价**

   没有分支预测：1 branch 代价 2 cycle

   对一个执行 B 次的循环，分支代价如下。

无预测	**2(B-1) 只有最后一次跳出循环无代价**
BTB	**4** 只有第一次和最后一次预测失败
BTB + BHT	**6** 只有第一次、第二次和最后一次预测失败

   显然对于较复杂的程序，有分支预测代价小于无分支预测，而2bit BHT+BTB比单纯的BTB更加灵活，因此代价更小；对于稍微简单的循环，BTB+BHT也只比 BTB 多两个周期。所以总体上使用 BTB + BHT 的效果是最好的。

2. **统计未使用分支预测和使用分支预测的总周期数及差值**

   结果的截图类似下方，为了节省报告篇幅，图片结果全部都放在 SrcCode/images 文件夹中。这里只列出相对应的统计结果。

策略\周期数\运行程序	**btb**	**bht**	**QuickSort**	**MatMul**
无预测	510	536	47112	330885
BTB	401	445	47147	326331
BTB+BHT	409	447	46736	326336
无预测 - BTB	109	91	-35	4554
无预测 - (BTB+BHT)	101	89	376	4549

3. **统计分支指令数目、动态分支预测正确次数和错误次数**

   下方表格每个有三个数据，分别是：分支指令数，预测错误数和预测正确数。具体截图同样在 images 文件夹。

   Total Cycle：记录总周期数

   BranchCount：分支指令数

   Wrong Count：预测但未分支以及未预测但分支次数

   PredictCount：BTB表中有且分支次数

   NotPredictedCount：未预测但分支次数

   预测错误数=Wrong Count-NotPredictedCount

   预测正确数=PredictCount - 预测错误数

策略\数据\运行程序	btb	bht	QuickSort	MatMul
BTB	101/1/50	110/6/49	6815/2580/847	4624/137/2174
BTB+BHT	101/0/50	110/0/44	6815/2499/863	4624/138/2173

4. **对比不同策略并分析以上几点的关系**
    1. 从运行周期数来看，程序复杂情况下，带有分支预测的CPU要比不带有分支预测的CPU更快，且 BTB+BHT的程序运行周期比只有BTB的要少。 在程序较简单时，更复杂的预测体现不出优势。
    2. 从预测正确率来看，BTB+BHT 效果总体上比 BTB 更好。对于快速排序这种依赖于数据的测试，更复杂的预测机制体现出了更强的应变能力，不论数据如何变化其都有一定的加速效果，证明了 BHT 状态机的作用。
    3. 由于一开始都默认分支不跳转，所以每次遇到跳转都需要进行stall，非常耗时。通过进行分支预测，在预测正确时流水线可以不停止地向前运转，因此预测正确次数越多，预测错误次数越少，程序用的时钟周期数越少，可见对分支进行预测通过减少分支代价来给系统性能带来优化，并且在复杂场景中效果更好。

收获

花费时间：思考逻辑并打草稿1h，编程1h，debug 约1天（主要是思考计数器逻辑）。

收获：

1. 对于 Verilog 和 system Verilog有了更加深入的了解。
2. 对于分支预测和CPU的工作有了更深的理解，明白了分支预测在复杂程序运行情况下对性能带来的提升。