Lab 4 report

PB21020485 吴敌

实验内容

- 实现动态分支预测器，分别采用BTB1bit和BTB+BHT2bit进行分支预测
- 统计分支预测对流水线性能的影响，并做定量分析

实验设计

- BTB：

  BTB采用1bit预测，和直接映射cache类似

```
    always @ (posedge clk)
    begin
        if (flushF)
        begin
            for (i=0; i<BTB_SET; i=i+1)
            begin
                BTB_TAG[i] <= 0;
                BTB_VALID[i] <= 0;
                BTB_TARGET[i] <= 0;
                BTB_HIS[i] <= 0;
            end
        end
        else if (is_br_type)
        begin
            BTB_TAG[PC_EX[BTB_SET_WIDTH+1: 2]] <= PC_EX[31: BTB_SET_WIDTH+2];//
字对齐,同时不是TAG_WIDTH
            BTB_HIS[PC_EX[BTB_SET_WIDTH+1: 2]] <= br;
            BTB_TARGET[PC_EX[BTB_SET_WIDTH+1: 2]] <= br_target;
            BTB_VALID[PC_EX[BTB_SET_WIDTH+1: 2]] <= 1;
        end
    end
```

  表项都会在**EX阶段计算出结果**时更新，`HIS`会写入这条跳转指令的跳转情况，从而预测下一条指令。

  在**IF**阶段时，BTB会首先根据tag来确定是否命中，如果命中，那么就会根据历史表进行预测，并给出对应的跳转地址。如不命中，那么会给出PC+4的值。

- BHT：

BTB命中采用跳转的地址

在这里我采用了一个表来记录所有的状态机，其判断使用高位：

```
assign BHT_hit = BHT_BIT[PC_IF[BHT_SET_WIDTH+1: 2]][1];

always @(posedge clk)
begin
    if (flushF)
    begin
        for (i = 0; i < BHT_SET; i=i+1)
        begin
            BHT_BIT[i] <= 2'b01;
        end
    end
    else if (is_br_type)
    begin
        if (br)
        begin
            if (BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] == 2'b11)
            begin
                BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] <= 2'b11;
            end
            else BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] <=
BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] + 1;
        end
        else
        begin
            if (BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] == 2'b00)
                BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] <= 2'b00;
            else BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] <=
BHT_BIT[PC_EX[BHT_SET_WIDTH+1: 2]] - 1;
        end
    end
end
```

除此以外，我们**复位时将所有状态复位为01**，这样预测器可以迅速学习。

BHT同样会在**EX阶段**时更新对应表项。

- 加入流水线：

```
always @(posedge clk)
begin
    if (flushF)
    begin
        br_num <= 0;
        pre_succ_num <= 0;
```

```
        end
        else if (is_br_type & !bubbleE)//EX阻塞会多算
        begin
            br_num <= br_num + 1;
            if (!br_fail)
                pre_succ_num <= pre_succ_num + 1;
        end
    end
```

我将分支预测器给出的预测地址传入**EX段**，再和真实结果一起反馈回分支预测器，判断是否预测成功，生成 `br_fail` 信号。这个信号和br信号的作用十分类似，所以在 `Hazard` 中使用相同位置进行flush。

Branch History Table

BTB	BHT	REAL	NPC_PRED	flush	NPC_REAL	BTB_update
Y	Y	Y	BUF	N	BUF（BTB预测跳且真跳）	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

分支收益和分支代价

- BTB 1bit：
  - 收益：记录下跳转指令的跳转地址，以根据最后一次的跳转历史来进行预测而不Flush，对于循环的分支指令的预测非常有效
  - 代价：加大资源使用；同时存储器都是异步读，时序不好
- BTB+BHT 2bit：
  - 收益：减少flush，同时在多层循环中改进了内层循环在1bit预测时的“抖动”
  - 代价：资源使用量更大，而且所有相同低位地址共用BHT容易冲突

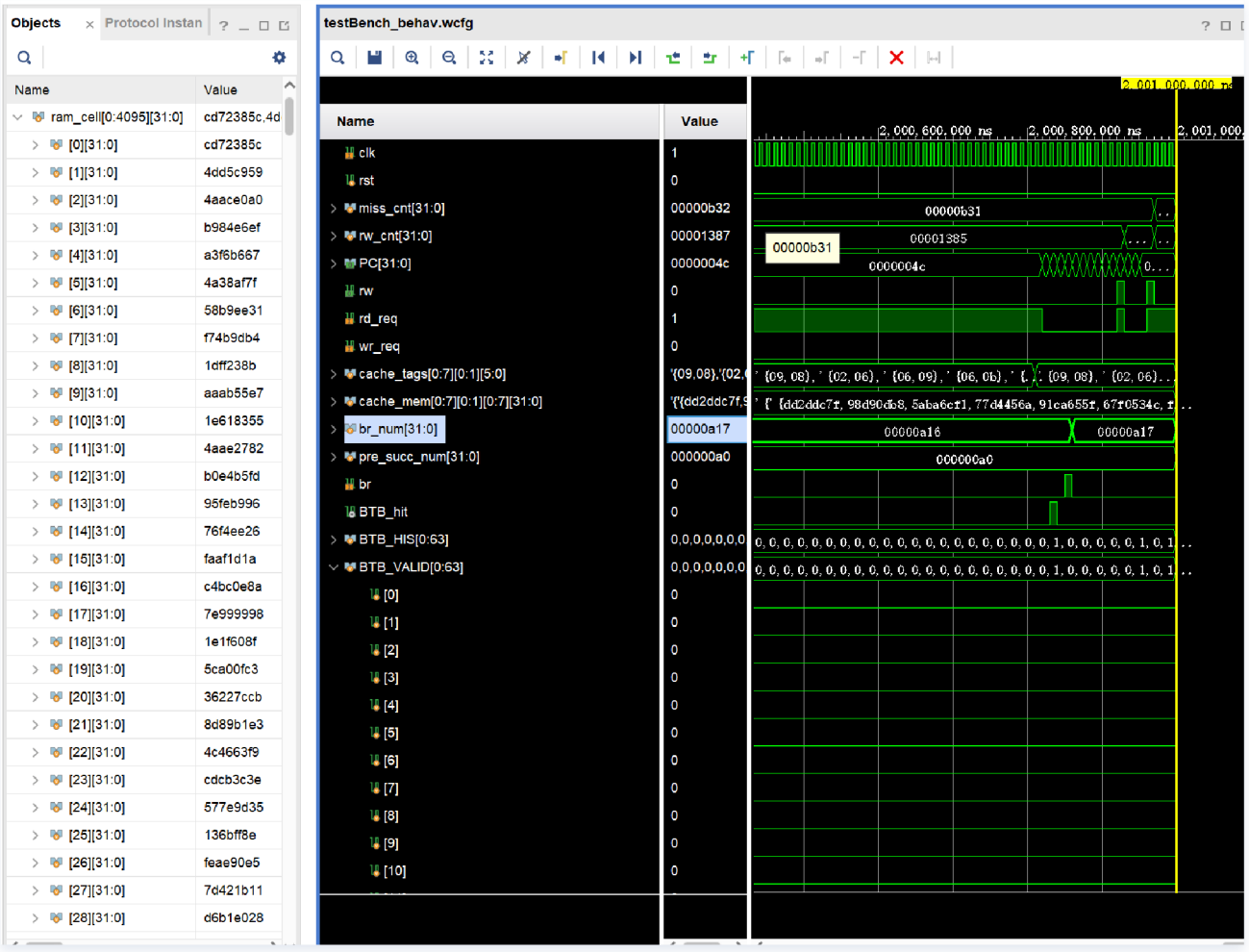
性能测试实验数据记录

QuickSort

预测方法	总周期数	总执行分支指令数	分支预测正确次数	分支预测正确率
不使用分支预测	54634	7205	5585	0.775
使用BTB预测	55104	7205	5349	0.742
使用BHT预测	53624	7205	6089	0.845

MatMul

首先是正确性：



```
// dst matrix C
ram_cell[       0] = 32'h0;  // 32'hcd72385c;
ram_cell[       1] = 32'h0;  // 32'h4dd5c959;
ram_cell[       2] = 32'h0;  // 32'h4aace0a0;
ram_cell[       3] = 32'h0;  // 32'hb984e6ef;
ram_cell[       4] = 32'h0;  // 32'ha3f6b667;
ram_cell[       5] = 32'h0;  // 32'h4a38af7f;
ram_cell[       6] = 32'h0;  // 32'h58b9ee31;
ram_cell[       7] = 32'h0;  // 32'hf74b9db4;
ram_cell[       8] = 32'h0;  // 32'h1dff238b;
ram_cell[       9] = 32'h0;  // 32'haaab55e7;
ram_cell[      10] = 32'h0;  // 32'h1e618355;
ram_cell[      11] = 32'h0;  // 32'h4aae2782;
ram_cell[      12] = 32'h0;  // 32'hb0e4b5fd;
ram_cell[      13] = 32'h0;  // 32'h95feb996;
ram_cell[      14] = 32'h0;  // 32'h76f4ee26;
ram_cell[      15] = 32'h0;  // 32'hfaaf1d1a;
ram_cell[      16] = 32'h0;  // 32'hc4bc0e8a;
ram_cell[      17] = 32'h0;  // 32'h7e999998;
ram_cell[      18] = 32'h0;  // 32'h1e1f608f;
ram_cell[      19] = 32'h0;  // 32'h5ca00fc3;
ram_cell[      20] = 32'h0;  // 32'h36227ccb;
ram_cell[      21] = 32'h0;  // 32'h8d89b1e3;
ram_cell[      22] = 32'h0;  // 32'h4c4663f9;
ram_cell[      23] = 32'h0;  // 32'hcdcb3c3e;
ram_cell[      24] = 32'h0;  // 32'h577e9d35;
ram_cell[      25] = 32'h0;  // 32'h136bff8e;
ram_cell[      26] = 32'h0;  // 32'hfeae90e5;
ram_cell[      27] = 32'h0;  // 32'h7d421b11;
ram_cell[      28] = 32'h0;  // 32'hd6b1e028;
ram_cell[      29] = 32'h0;  // 32'heeaec5fa;
ram_cell[      30] = 32'h0;  // 32'hd8ff552c;
ram_cell[      31] = 32'h0;  // 32'hb7b96484;
ram_cell[      32] = 32'h0;  // 32'h9a628de7;
ram_cell[      33] = 32'h0;  // 32'h9ab5c0b2;
ram_cell[      34] = 32'h0;  // 32'heaf92a23;
```

预测方法	总周期数	总执行分支指令数	分支预测正确次数	分支预测正确率
不使用分支预测	340779	4624	274	0.059
使用BTB预测	333173	4624	4076	0.881
使用BHT预测	332633	4624	4346	0.940

btb

预测方法	总周期数	总执行分支指令数	分支预测正确次数	分支预测正确率
不使用分支预测	510	101	1	0.010
使用BTB预测	312	101	99	0.980
使用BHT预测	312	101	99	0.980

bht

预测方法	总周期数	总执行分支指令数	分支预测正确次数	分支预测正确率
不使用分支预测	536	110	11	0.100
使用BTB预测	380	110	88	0.800
使用BHT预测	362	110	97	0.882

实验分析

1. 快速排序有循环嵌套，而且其中Partition找位置的分支变化非常频繁，因此1bit的分支预测效果会比较差（比不预测更差）。但2bit预测器减少抖动，多重循环和交替跳转的问题，因此获得比较好的命中效果。
2. 矩阵乘法中有较大循环，因此有1bit分支预测强于没有预测。但同时，由于矩阵乘法依然存在多重，故2bit预测器的效果也好于1bit预测器。
3. btb测试使用单层循环。在这种情况下不使用分支预测错误非常多，而使用1bit或2bit的效果是一致的，且错误只出现在第一次跳转和最后一次不跳转上。
4. bht测试，使用1bit时内层循环就会的上一次结束到下一次开始的两次分支预测一定会失败，而2bit预测器在这种多重循环就解决了这个问题。
5. 综上所述：造成2bit和1bit预测器性能区别的主要体现在了多重循环和交替跳转上，而1bit预测和不预测的性能区别主要取决于程序中的循环大小。

实验难点

本次实验在调试过程中遇到的困难有：

1. 没有深刻理解好BHT中对BTB的使用，一开始以为同时跳转就跳转，这样的结果很不好。后来找到原因在于BTB提供地址，所以只需要命中而不需要1bit BTB同时预测跳转。

实验总结

- 全面认识两种分支预测策略，并结合具体测试分析不同策略的优劣和性能影响
- 熟悉了两种分支预测器的设计方法，加深了对其的理解