计算机体系结构 - Exp3 Report

一、实验目的和要求

实验目的

- 了解分支预测原理
- 实现以 BHT 和 BTB 为基础的动态分支预测

实验要求

- 在 lab2 的基础上实现用 BTB 和 BHT 做动态分支预测
- 通过仿真测试和上板验证
- 验收要求指出使用了 BTB 和 BHT 的跳转指令位置，展示 PC 的变化

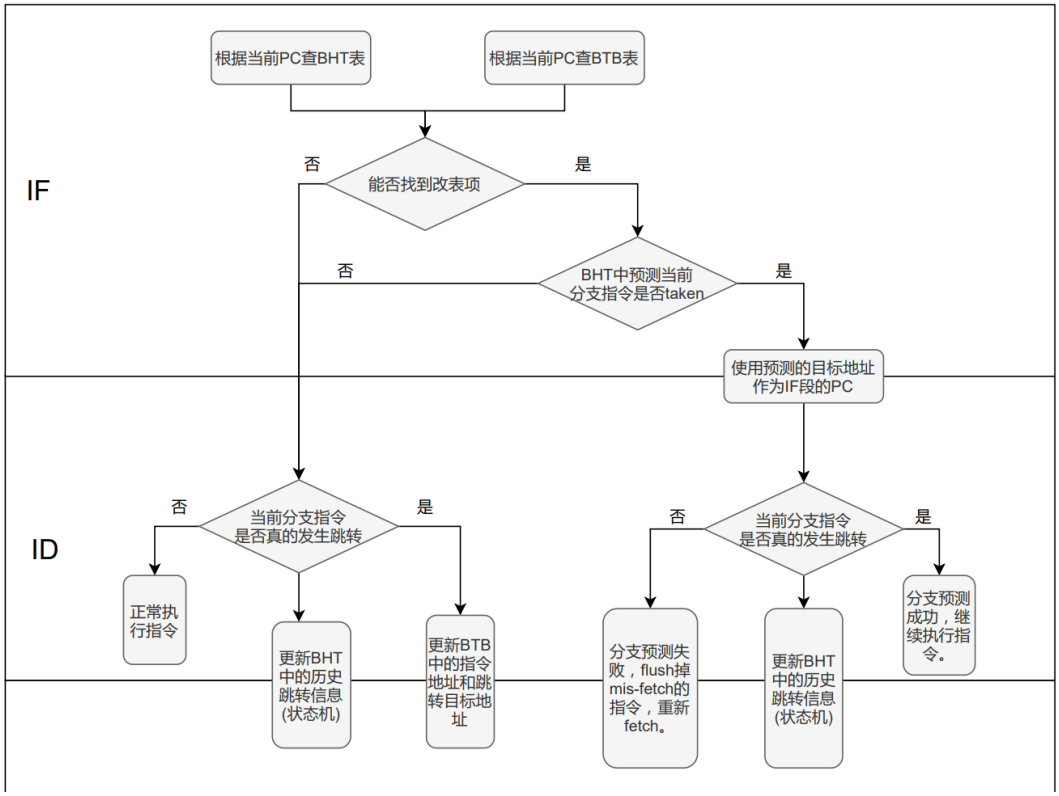
二、实验内容和原理

实验简介

实验任务

- 在 lab2 的基础上，在 5 段流水线内增加 BTB 和 BHT
- 在给定的 SOC 中，加入自己的 CPU，通过仿真测试和上板验证

分支预测流程图



实验原理

RV32core修改

预测指令下，我们主要对IF和ID阶段的pc转移进行修改。首先由于我们需要refetch指令，因此需要得到重新读取的指令地址，另外还需根据PC_4_IF和jump_PC得到下一条IF指令地址。在fetch的情况下选择next_pc_ID，否则选择next_pc_IF。其中由于我们记录的指令pc值的高六位，因此需要加入两位后缀0。判断是否命中BTB，如果未命中则按照正常处理，如果命中则处理BTB中的语句。最后由refetch信号进行标记，并送入redirectPC进行中断检测。代码修改如下：

```
MUX2T1_32
 mux_IF(.I0(next_pc_IF),.I1(next_pc_ID),.s(refetch),.o(next_PC_IF));
MUX2T1_32
 mux_IF_normal(.I0(PC_ID+4),.I1(jump_PC_ID),.s(Branch_ctrl),.o(next_pc_ID));
MUX2T1_32
 mux_IF_predict(.I0(PC_4_IF),.I1({22'b0,pc_to_take,2'b0}),.s(taken),.o(next_pc_I
F));
MUX2T1_32
 redirectPC(.I0(next_PC_IF),.I1(PC_redirect_exp),.s(redirect_mux_exp),.o(final_P
C_IF));
```

此外，我们需要在前递模块中也引入refetch信号，保持一致性。

```
HazardDetectionUnit
hazard_unit(.clk(debug_clk),.Branch_ID(refetch),.rs1use_ID(rs1use_ctrl),

.rs2use_ID(rs2use_ctrl),.hazard_optype_ID(hazard_optype_ctrl),.rd_EXE(rd_EXE),

.rd_MEM(rd_MEM),.rs1_ID(inst_ID[19:15]),.rs2_ID(inst_ID[24:20]),.rs2_EXE(rs2_EXE
),
```

分支预测

输入信号

```
module Branch_Prediction(
          input clk,
          input rst,

          // from IF
          input [7:0] PC_Branch,
          output taken,
          output [7:0] PC_to_take,

          // from ID
          input Branch_ID,
          input J,
          input [7:0] PC_to_branch,
          output refetch
        );
```

IF部分，引入的信号是PC_Branch，将PC的9-2位引入作为index。由于PC中总是4位为一份，因此0、1位恒为0。

ID部分，引入的信号为Branch_ID，来源为CtrlUnit模块的Branch是否跳转的信号，用来判断是否进行分支预测。引入的信号J信号是所有SB,JAR,JALR信号，只要有可能发生跳转即可引入。引入的PC_to_Branch是经过ALU处理过的跳转的目的地址PC的9-2位，用作BTB表储存的参数。

变量信号

```
reg taken_IF, taken_ID, refetch_, is_refetch_ed;
reg [7:0] PC_to_take_, PC_to_take_ID;

reg [9:0] BHT[0:63];

reg [9:0] BTB[0:63];

wire [5:0] index_IF;
wire [1:0] tag_IF;
assign index_IF = PC_Branch[5:0];

assign tag_IF = PC_Branch[7:6];

reg [5:0] index_ID;
reg [1:0] tag_ID;

reg BHT_Hit_IF, BHT_Hit_ID, BTB_Hit_IF, BTB_Hit_ID;
```

除去必要的信号，还有4位的64位BHT寄存器，其中3-2位用来存放tag进行比对。BTB表则是10位的64位寄存器，低8位存放重定向PC，高二位存放tag。

refetch判断

接下来分为两部分书写，第一部分仅做读操作，读取BHT和BTB的值，并判断当前指令是否需要refetch。注意由于是读操作，所以不需要控制是上升或者下降沿。

在此过程中，也需要判断是否能够读取当前指令的BHT和BTB，如果能够读取则赋hit为1。

关于refetch如何赋值，分为三种情况。如果之前做过refetch，后续使用is_refetch_ed进行恢复。此外，当同时BHT,BTB命中才考虑使用BTB表进行寻址替换，同时，要进行替换也需要满足Branch_ID==1成立，即首先要是需要跳转的SB指令。此外，如果BTB，BLT其中一个不满足，不需要跳转的时候也置1，参考之前的图，refetch变量置1表示不使用BLT,BTB进行正常跳转。Refetch置0表示不跳转或者使用BLT,BTB跳转

```
if (BHT[index_IF][3:2] == tag_IF)//index和tag都一样，认为是hit
begin
    BHT_Hit_IF <= 1'b1;
    taken_IF <= BHT[index_IF][1];//两位表示状态机，高位表示这次是否进行跳转
end
else
begin
    BHT_Hit_IF <= 1'b0;
    taken_IF <= 1'b0;
end

if (BTB[index_IF][9:8] == tag_IF)//BTB命中，取剩下的位数作为地址参考。
begin
    BTB_Hit_IF <= 1'b1;
    PC_to_take_ <= BTB[index_IF][7:0];
end
else
begin
    BTB_Hit_IF <= 1'b0;
    PC_to_take_ <= 8'b0;
end
```

```
if (is_refetch_ed || ((taken_ID & BTB_Hit_ID) && Branch_ID && PC_to_branch ==
PC_to_take_ID)
|| ((~(taken_ID & BTB_Hit_ID)) && (~Branch_ID)))//
//刚做过refetch;使用BTB跳转;不跳转;BTB未命中下跳转这四个情况下选择BTB，否则直接传值
begin
    refetch_ <= 1'b0;
end
else
begin
    refetch_ <= 1'b1;
end
```

BHT、BTB判断

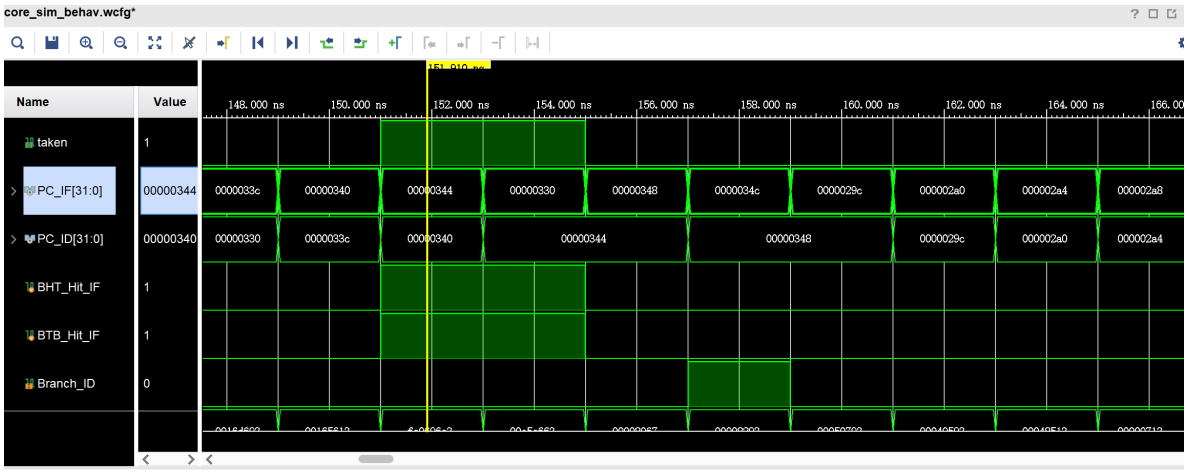
首先我们需要记录IF阶段的BHT、BTBhit的情况，跳转的地址等等，需要将他们传入ID阶段.根据ID阶段的跳转情况和跳转地址，对BHT和BTB进行修改。只有在跳转的情况下，才BTB写入数据。另外如果第一次遇到某条跳转指令，如果跳转则直接写入BHT，如果没有跳转，需要确定此条为跳转指令的时候才进行写入。

```
  begin
    if (rst)
      begin
        taken_ID <= 1'b0;
        is_refetch_ed <= 1'b0;
        BHT_Hit_ID <= 1'b0;
        BTB_Hit_ID <= 1'b0;
        index_ID <= 0;
        tag_ID <= 0;
        for (i = 0; i < 64; i = i + 1)//重置BTB,BHT表
          begin
            BHT[i] <= 0;
            BTB[i] <= 0;
          end
      end
    else
      begin
        taken_ID <= taken_IF;//顺序传递
        index_ID <= index_IF;
        tag_ID <= tag_IF;
        BHT_Hit_ID <= BHT_Hit_IF;
        BTB_Hit_ID <= BTB_Hit_IF;
        PC_to_take_ID <= PC_to_take_;
        is_refetch_ed <= refetch_;

        if (Branch_ID)//确实进行跳转
          begin
            if (BHT_Hit_ID)//处理BHT表
              begin
                if (BHT[index_ID][0] == 1'b0)//10改11，00改01
                  begin
                    BHT[index_ID][0] <= 1'b1;
                  end
                else
                  begin
                    BHT[index_ID][1] <= 1'b1;
                  end
```

```
                end
              else//未命中，表明应该没有存过对应的tag，进行存储
                begin
                  BHT[index_ID] <= {tag_ID, 2'b10};
                end

              BTB[index_ID] <= {tag_ID, PC_to_branch};//发生了跳转的SB指令，存BTB表，
将tag和接下来要跳转的PC存进去。
              //注意！！！分支不taken也不需要进行记录
            end
          else//没有进行跳转
            begin
              if (BHT_Hit_ID)//命中
                begin
                  if (BHT[index_ID][0] == 1'b0)
                    begin
                      BHT[index_ID][1] <= 1'b0;
                    end
                  else
                    begin
                      BHT[index_ID][0] <= 1'b0;
                    end
                end
              else if (J)//没有命中，则标记tag（未跳转的SB,J,JALR）
                begin
                  BHT[index_ID] <= {tag_ID, 2'b01};
                end
            end
        end
    end
```

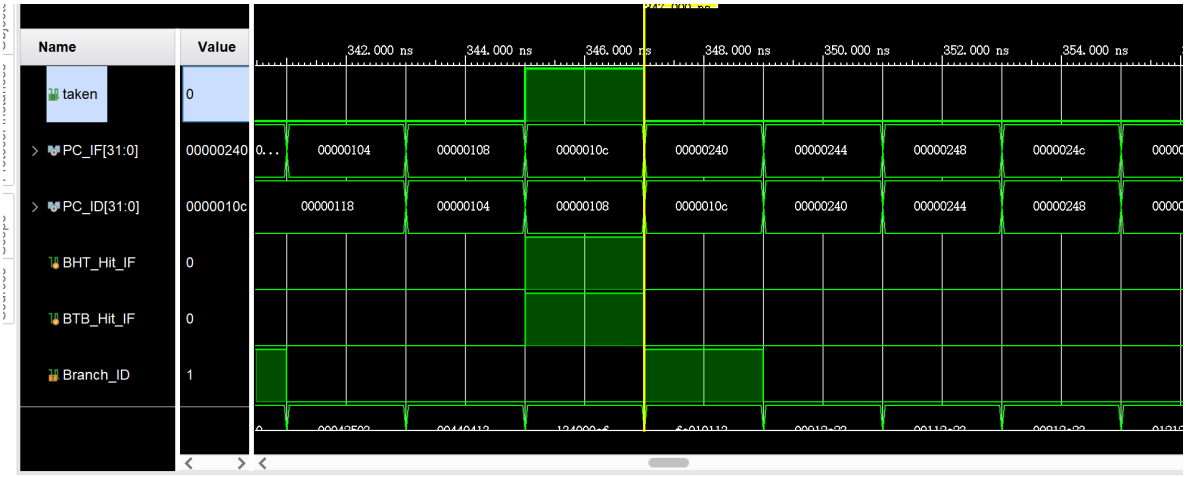
三、实验过程和数据记录

仿真验证



这一部分先看344，taken为1，双Hit，因此下一个PC使用表中的跳转PC为330。

但是对ID阶段的344进行判断，发现不应该跳转，因此再次跳转回344的下一拍PC348，并修改相应的BHT和BTB。



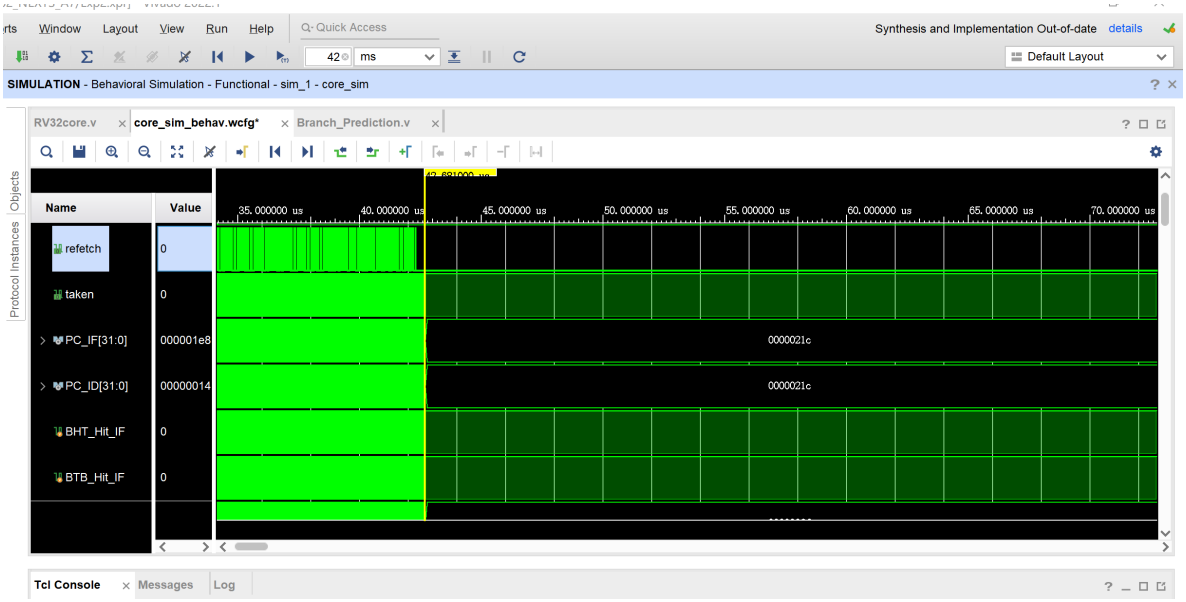
这一部分中，10c显示taken，于是使用跳转，到240，并且10c再次判断时也正确，继续执行

排序算法仿真结果

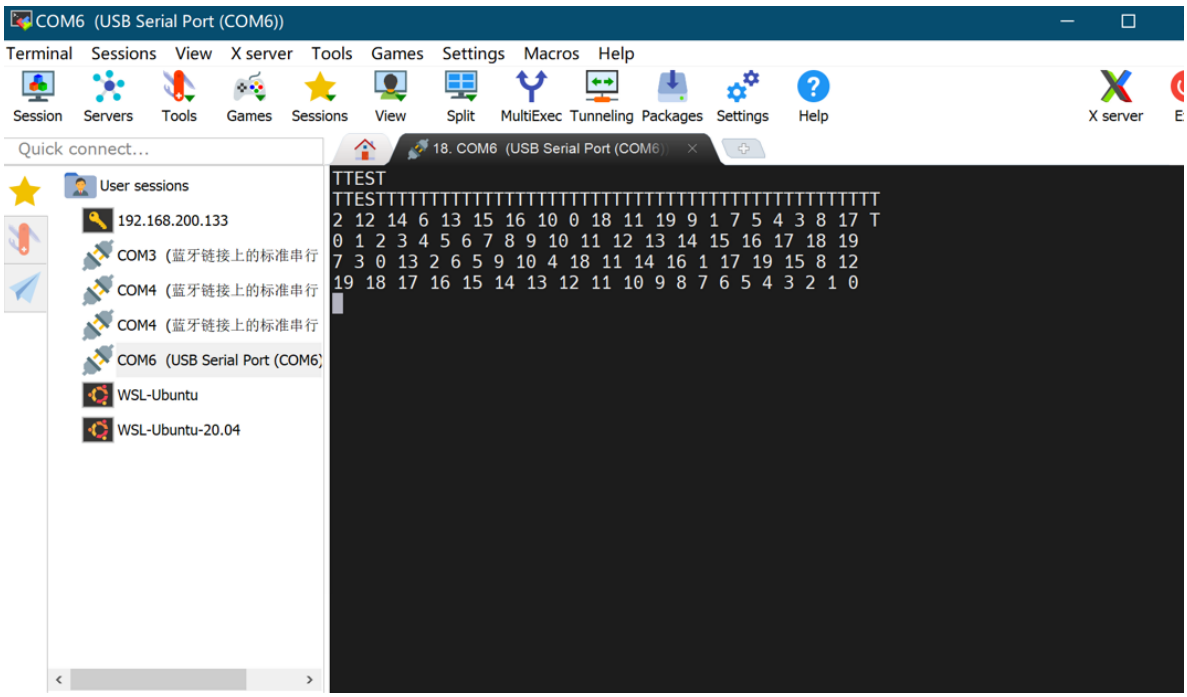
首先我们运行50微秒，可以在TCL console中得到最终的输出结果，符合排序的结果，第二行为从小到大，第四行为从大到小

```
Time resolution is 1 ps
2 12 run all
14 6 13 15 16 10 0 18 11 19 9 1 7 5 4 3 8 17
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
7 3 0 13 2 6 5 9 10 4 18 11 14 16 1 17 19 15 8 12
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

我们得到运行完全部程序的总时间约为42600ns，对比lab1程序运行同样的结果，明显减少时间



上板结果



四.实验心得

心得：这次的实验主要做的是分支的预测实验，去实现BHT和BTB相结合的动态分支预测技术。设计的时候要注意的是一些逻辑设置，比如BHT是一个每次都要更新的表，而BTB是一个只有taken的时候才进行更新的表格，所以设计的时候后者是进行一个逻辑判断而前者需要一直都有。另外，BHT由于经常需要使用，所以对于快速查找也是一个比较要思考的，在这里我使用了组相联映射的方式，进行了分组分块，以此来加快我们的查找速度，这一点我觉得十分困难，实现的时候也是思考了很久才想明白了其中的逻辑，才一点点写出来的。而这次实验对于小板子也加入了电脑显示的环节，十分友好，后面的实验应该会更加方便了。

思考题：

1.在报告里分析分支预测成功和预测失败时的相关波形。

具体可以见仿真分析。

2.在正确实现 BTB 和 BHT 的情况下，有没有可能会出现 BHT 预测分支发生跳转，也就 是 branch taken，但是 BTB 中查不到目标跳转地址，为什么?

不可能出现，因为预测分为两种情况：

若确实发生了跳转，则BTB会存跳转地址和tag，BHT也会存tag。之后taken肯定会去BTB位置。

```
begin
    BHT[index_ID] <= {tag_ID, 2'b10};
end
BTB[index_ID] <= {tag_ID, PC_to_branch};
```

若未发生跳转，此时只会记录BHT表下的tag，不会记录BTB表。但是下一次若做进入同样PC，若仍然不跳转，则不会taken成立。若发生跳转，此时状态机进入11，存储BTB地址。但是由于本次不taken，所以也不会出现问题。

```
if (J)
begin
    BHT[index_ID] <= {tag_ID, 2'b01};
end
```

3.前面介绍的 BHT 和 BTB 都是基于内容检索，即通过将当前 PC 和表中存储的 PC 比 较来确定分支信息存储于哪一表项。这种设计很像一个全相联的 cache，硬件逻辑实际 上会比较复杂，那么能否参考直

根据cache的设计思路，比如设计直接映射的BHT/BTB，规定对应的index下对应的地址。如果访问对应的index，将下次跳转的地址存入BTB表，这样做可以直接查询到对应的index下有没有记录地址。若采用组相联映射也同理，将对应index下不同tag的地址存在一个模块，根据最新跳转的地址替换掉老的跳转地址。