浙江大学

计算机体系结构实验报告

课程名称：	计算机体系结构
姓　　名：	周楠
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
学　　号：	3220102535
指导教师：	常瑞

2024 年 11 月 5 日

浙江大学实验报告

课程名称：	计算机体系结构	实验类型：	综合		
实验项目名称：	动态分支预测				
学生姓名：	周楠	专业	计算机科学与技术	学号：	3220102535
实验地点：	玉泉实验室	实验时间：	2024.11.06	指导教师：	常瑞

一. 操作方法与实验步骤

1.1 PC_Table 模块

1. branch-history table(BHT)，又名 branch-prediction buffer，它是一小块包含了跳转地址和历史跳转信息的 buffer。我们在遇到跳转指令的时候，通过对比之前保存在 buffer 中的跳转地址和相应的跳转信息来决定当前这条跳转指令是否应该发生跳转。

2. branch-target buffer(BTB)，也叫 branch-target cache，用来保存预测的分支跳转目标地址。

3. PC_Table 模块是 BHT 和 BTB 的结合，它保存了当前 PC 地址对应的 BHT 和 BTB 信息。也就是当前 PC 地址对应的跳转地址和跳转信息。

    - 创建一个有 256 个 entry 的 PC_Table, 每一个 entry 有 34 位，前 2 位表示历史跳转信息，后 32 位表示跳转目标地址。
    - 在 IF 阶段，PC_Table 模块根据当前 PC 地址，在 PC_Table 中查找相应的 entry，获取跳转信息和跳转目标地址。
    - 在 ID 阶段，PC_Table 模块会根据当前分支指令是否真的发生跳转，更新 BHT 中的历史跳转信息（状态机），更新 BTB 中的跳转目标地址。

1.1.1 查询 PC_Table

```
module PC_Table (
    input wire rst,
    input wire clk,
    input wire [31:0] PC_IF,
    input wire stall,
    input wire [31:0] PC_ID,
    input wire [31:0] PC_branch,
    input wire branch_new,

    output wire [31:0] PC_result,
    output wire predict_pc_branch
);

// 34位，2位status，32位PC，
// 256个条目，对应256条指令，对应PC的范围是0-1024
reg [33:0] PC_Table [0:255];

// 读取PC_Table
wire [33:0] entry;
assign entry = PC_Table[PC_IF[9:2]];
assign predict_pc_branch = entry[33];
assign PC_result = entry[31:0];
```

PC_IF 表示 IF 阶段的指令地址，利用 PC_IF[9:2] 来索引 PC_Table，找到对应的 entry。预测是否跳转的依据是 entry[33]，如果为 1，则预测跳转，否则预测不跳转。所以 predict_pc_branch 的值为 entry[33]。预测的跳转地址为 entry[31:0]。

1.1.2 更新 PC_Table

```
// 更新PC_Table
integer i;
wire [33:0] entry_new;
assign entry_new = PC_Table[PC_ID[9:2]];

always @(posedge clk or posedge rst) begin
    if (rst) begin
        for (i = 0; i < 256; i = i + 1) begin
            PC_Table[i] <= 34'b0;
        end
    end
    else if (~stall) begin
        if (branch_new) begin
            if(entry_new[33:32] == 2'b00)
                PC_Table[PC_ID[9:2]] <= {2'b01, PC_branch[31:0]};
            else
                PC_Table[PC_ID[9:2]] <= {2'b11, PC_branch[31:0]};
        end
        else begin
            if(entry_new[33:32] == 2'b11)
                PC_Table[PC_ID[9:2]] <= {2'b10, entry_new[31:0]};
            else
                PC_Table[PC_ID[9:2]] <= {2'b00, entry_new[31:0]};
        end
    end
end
```

- branch_new 为 1 表示当前分支指令是真实发生跳转。因此，如果 entry_new[33:32] 为 00，则更新为 01，否则更新为 11。同时更新 BTB 中的跳转目标地址，也就是说更新 entry_new[31:0] = PC_branch[31:0]。
- branch_new 为 0 表示当前分支指令没有发生跳转。因此，如果 entry_new[33:32] 为 11，则更新为 10，否则更新为 00。由于没有实际发生跳转，所以不需要更新 BTB 中的跳转目标地址。

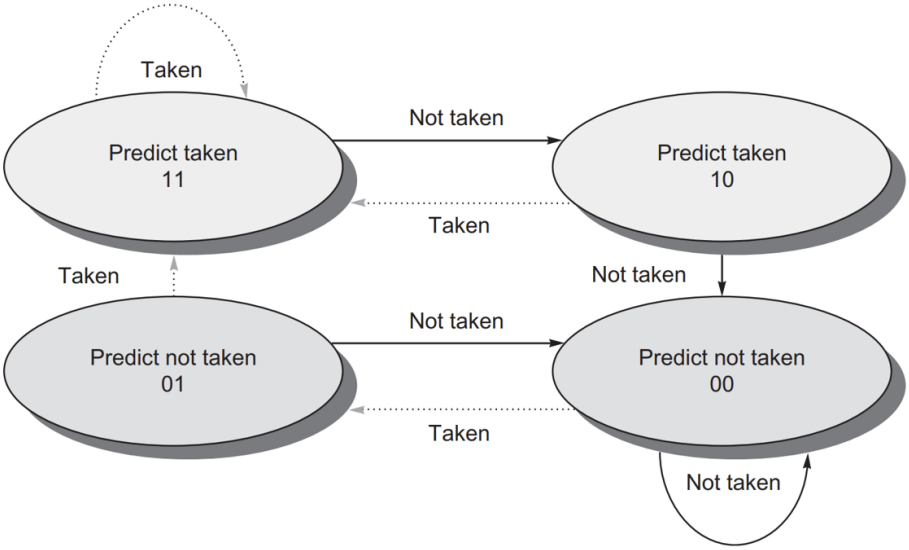

图 1: 有限状态机

1.2 Predict_branch_ctrl 模块

1. 计算 refetch 信号：

   - 预测结果与实际分支结果不符

     当 Branch != prev_predict_branch 时，表示预测结果与实际分支结果不符，需要重新获取正确的 PC 值。因此 refetch = 1。

   - 预测跳转且实际跳转，但目标地址不同

     当 Branch & prev_predict_branch & (jump_PC_ID != prev_predict_PC) 时，表示预测跳转且实际跳转，但目标地址不同，需要重新获取正确的 PC 值。因此 refetch = 1。

2. 计算 stall 信号:

   - load-use 冒险

     当 reg_FD_stall 为 1 且 PC_EN_IF 为 0 时，表示此时处于 load_use 冲突，需要暂停 IF 阶段的 PC 更新。这种情况下 ID 和 IF 阶段的指令会推迟到下一周期执行。所以我们需要在本周期 stall，停止 PC_Table 的更新

   - 预测错误

     当 predict_fail 为 1 时，表示预测失败，需要重新获取正确的 PC 值。预测错误时，下一周期的 ID 阶段指令需要被 flush 掉，防止执行。因此我们需要在下一周期 stall，停止 PC_Table 的更新。

3. 计算 predict_fail 信号：

   此处的 predict_fail 信号用于在预测错误时用于 flush ID 阶段的指令，不更新 PC_Table。

   前一个周期，ID 阶段计算得到 branch 信号和 prev_predict_branch 信号，两者不相等，发生冲突，此时令 predict_fail = 1。从而下一个周期发生时，stall 信号为 1，停止 PC_Table 的更新。

4. 计算 next_PC_IF 信号：

   当 refetch 为 1 时，表示预测错误，需要重新获取正确的 PC 值。此时 next_PC_IF = Correct_PC。

   当 refetch 为 0 时，表示预测正确，此时根据是否预测跳转，选择 next_PC_IF = Predict_PC 或 PC_IF + 4。

```
module Predict_branch_ctrl(
    // 系统信号
    input wire rst, // 复位信号
    input wire clk, // 时钟信号

    // 控制信号
    input wire PC_EN_IF, // IF阶段PC使能
    input wire reg_FD_stall, // FD寄存器暂停信号

    // PC和分支信息
    input wire [31:0] PC_ID, // ID阶段PC
    input wire [31:0] PC_IF, // IF阶段PC
    input wire [31:0] jump_PC_ID, // ID阶段跳转目标地址
    input wire Branch, // 分支指令标志

    // 输出信号
    output wire refetch, // 分支预测冲突控制信号
    output wire [31:0] next_PC_IF // 预测的PC
);

reg prev_predict_branch; // 上一次的分支预测结果
reg [31:0] prev_predict_PC; // 上一次的预测PC
```

```
23 reg predict_fail; // 预测失败标志
24
25
26 wire Predict_PC_Branch; // 预测是否分支跳转
27 wire [31:0] Predict_PC; // 预测的下一条PC
28 wire [31:0] Correct_PC; // 发生冲突时的正确PC
29
30 PC_Table PC_Table(
31     .clk(clk),
32     .rst(rst),
33     .PC_IF(PC_IF),
34
35     .PC_result(Predict_PC),
36     .predict_pc_branch(Predict_PC_Branch),
37
38     .stall((reg_FD_stall & ~PC_EN_IF) | predict_fail),
39     .PC_ID(PC_ID),
40     .PC_branch(jump_PC_ID),
41     .branch_new(Branch)
42 );
43
44 // 计算冲突时的正确PC值
45 assign Correct_PC = (Branch) ? jump_PC_ID : PC_ID + 4;
46
47 // 1. 预测结果与实际分支结果不符
48 // 2. 预测跳转且实际跳转，但目标地址不同
49 assign refetch = (Branch != prev_predict_branch) || (Branch &
       prev_predict_branch & (jump_PC_ID != prev_predict_PC));
50
51 // 计算下一条指令的
52 assign next_PC_IF = (refetch) ? Correct_PC : (Predict_PC_Branch) ?
       Predict_PC : PC_IF + 4;
53
54 always @(posedge clk or posedge rst) begin
55     if(rst) begin
```

```
56          // 复位时清空所有状态
57          prev_predict_branch <= 1'b0;
58          prev_predict_PC <= 32'd0;
59          predict_fail <= 1'b0;
60      end
61      else begin
62          if(reg_FD_stall & ~PC_EN_IF) begin
63              // 流水线暂停时保持状态不变
64              prev_predict_branch <= prev_predict_branch;
65              prev_predict_PC <= prev_predict_PC;
66              predict_fail <= predict_fail;
67
68          end
69          else if(refetch) begin
70              // 发生预测冲突时，清空预测状态并标记预测失败
71              prev_predict_branch <= 0;
72              prev_predict_PC <= 0;
73              predict_fail <= 1;
74          end
75          else begin
76              // 正常情况下更新预测状态
77              prev_predict_branch <= Predict_PC_Branch;
78              prev_predict_PC <= Predict_PC;
79              predict_fail <= 0;
80          end
81      end
82  end
83  endmodule
```

二. 思考题

1. 请在报告里展示四种仿真波形：分支预测跳转但实际不跳转，分支预测不跳转但实际跳转，分支预测跳转且实际跳转，分支预测不跳转且实际不跳转。

1. 分支预测跳转但实际不跳转

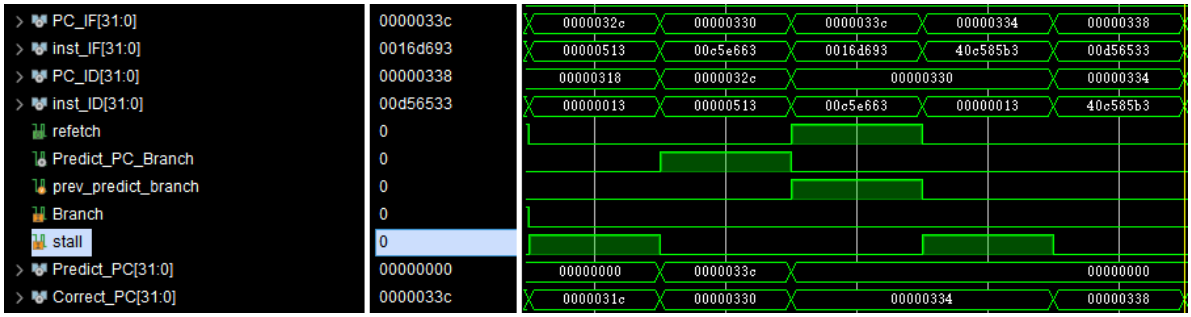

图 2: 分支预测跳转但实际不跳转

- PC_IF=0x00000330 处的指令，由于 predict_pc_branch=1, 所以下一周期的 PC_IF 会获取跳转的地址也就是 0x0000033c。
- 但是当 0x00000330 处的指令执行到 ID 阶段时，会发现此时并不需要跳转，refetch 信号置为 1，获取正确的 PC 值为 0x00000334。
- 下一周期开始时，令 predict_fail=1，从而 stall 信号置为 1，停止 PC_Table 的更新。

2. 分支预测不跳转但实际跳转

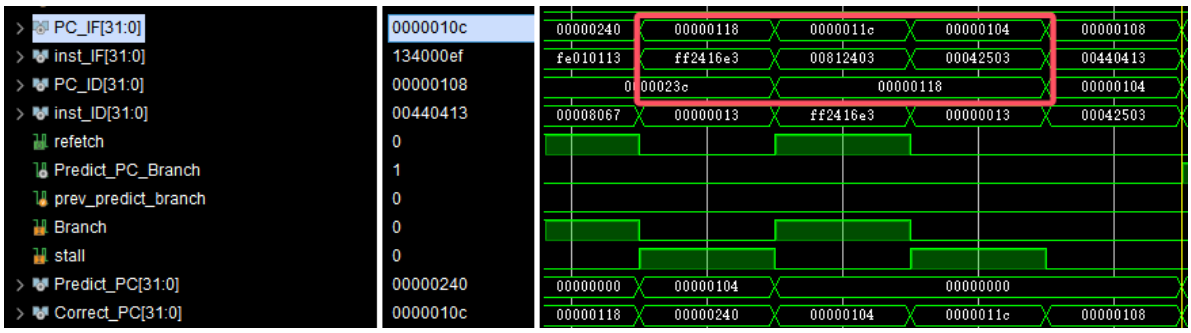

图 3: 分支预测不跳转但实际跳转

80000118: ff2416e3 bne s0,s2,80000104 <display+0x20>

- PC_IF = 000000118 处的指令，由于 predict_pc_branch=0, 所以下一周期的 PC_IF 会获取 PC_IF+4 的值也就是 0x00000011c。
- 但是当 0x000000118 处的指令执行到 ID 阶段时，会发现此时需要跳转，refetch 信号置为 1，获取正确的 PC 值为 0x000000104。
- 下一周期开始时，令 predict_fail=1，从而 stall 信号置为 1，暂停 PC_Table 的更新。

3. 分支预测跳转且实际跳转

图 4: 分支预测跳转且实际跳转

- PC_IF=0x0000010c 处的指令，由于 predict_pc_branch=1, 所以下一周期的 PC_IF 会获取跳转的地址也就是 0x00000240。
- 当 0x0000010c 处的指令执行到 ID 阶段时，会发现此时确实需要跳转，refetch 信号置为 0，不需要重新获取 PC 值。

4. 分支预测不跳转且实际不跳转

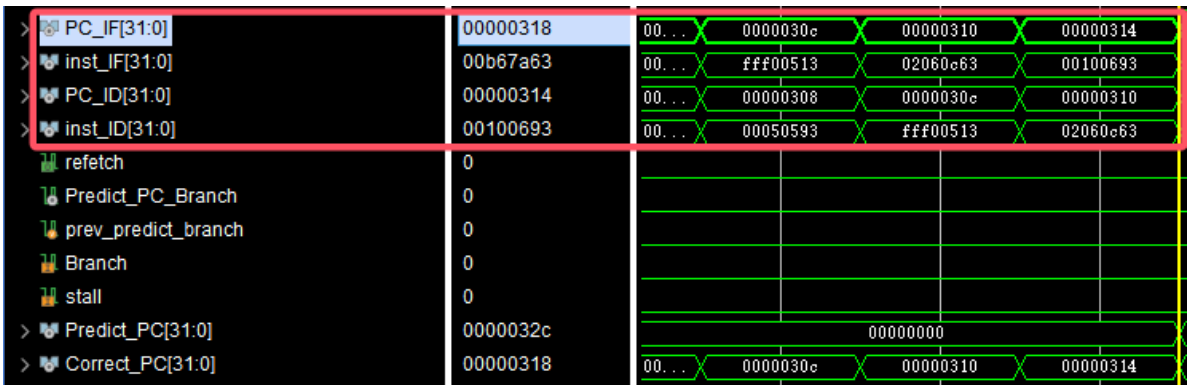

图 5: 分支预测不跳转且实际不跳转

80000310: 02060c63 beq a2,zero,80000348 <___udivsi3+0x44>

- PC_IF=0x00000310 处的指令，由于 predict_pc_branch=0, 所以下一周期的 PC_IF 会获取 PC_IF+4 的值也就是 0x00000314。
- 当 0x00000310 处的指令执行到 ID 阶段时，会发现此时确实不需要跳转，refetch 信号置为 0，不需要重新获取 PC 值。

2. 相比于静态分支预测，说明动态分支预测的优点以及缺点

- 优点

  1. **更高的准确性**

     动态分支预测根据程序的历史执行情况调整预测策略，因此能够动态适应不同的程序行为。通过利用分支的历史信息，动态预测器可以识别出程序中的模式，比如“常常跳转”或“总是顺序执行”的情况，从而提高预测的准确性。

  2. **减少控制冒险**（Control Hazard）

     在流水线中，分支指令可能导致控制冒险（Control Hazard），即当分支预测错误时，流水线中的指令需要被撤销并重新加载。通过精确的动态分支预测，可以减少这种控制冒险，从而提高处理器的吞吐量和性能。

- 缺点

  1. **硬件开销大**

     动态分支预测器需要额外的硬件资源来存储历史信息、管理预测表和进行模式匹配, 会增加硬件的复杂性和功耗。此外，动态预测器需要频繁地更新其状态，这也增加了额外的开销。

  2. **复杂性增加**

     动态分支预测器的设计较为复杂，需要考虑如何高效地存储和访问历史信息、如何更新预测状态以及如何选择合适的预测算法。在一些设计中，还需要平衡准确性和硬件资源的使用，这增加了硬件设计的复杂性。