计算机体系结构实验报告

LAB04

题目: 动态分支预测

姓名: 魏钊

学号: PB18111699

实验目的

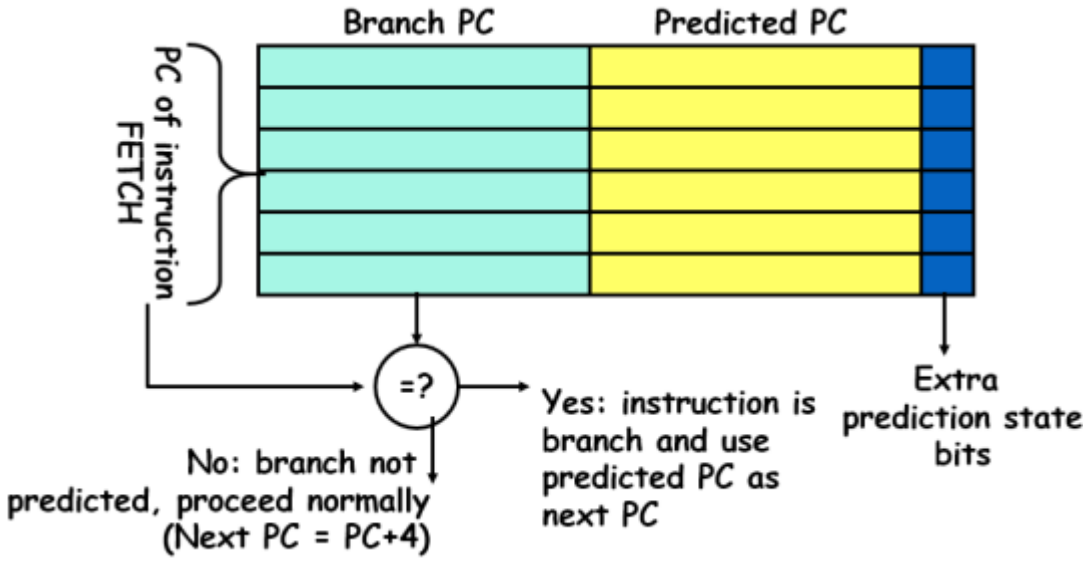
- 实现 BTB（Branch Target Buffer）和 BHT（Branch History Table）两种动态分支预测器
- 体会动态分支预测对流水线性能的影响

实验平台

Vivado

实验设计与过程

一、 BTB



BTB 模块中的 buffer 结构如上图，与 Cache 类似。第一部分 BranchPC 是缓存的地址(tag)列表，采用直接映射策略。 第二部分 PredictedPC 是预测的跳转地址。第三部分为 1 位宽，表示该项是否有效。

输入的 PC 低位寻址到对于表项，对 tag 部分进行比较，如果相等， 并且该项是有效的， 则认为命中， 然后使用 Predicted PC 作为输出， 表示根据历史信息，这是一个分支指令并且预测跳转。

BTB 模块介于 IF 和 ID 段寄存器之间，对于 IF 段寄存器输出的 PCF，输入到 BTB 中， BTB 查找 Buffer，如果命中，则输出地址向 NPC 模块输入， NPC 模块选择预测 PC 作为下一个 PC，如果不命中，直接使用 PCF+4 作为下一个 PC，同时，命中的状态位（1 表示进行了预测且预测跳转， 0 表示不预测） 和预测 PC 也随流

水段流到 EX 段。

等到当前指令到达了 EX 段， 就可以确定当前指令是否跳转，如果实际跳转但之前未预测跳转或者实际不跳转但之前预测跳转，都需要清空 ID 和 EX 段的寄存器，NPC 模块重新生成下一个 PC。 在实际跳转但之前未预测跳转的情况下，需要把当前 PC 和跳转的 PC 更新写入 BTB 的 buffer 中。 在实际不跳转但之前预测跳转的情况下，需要把 BTB 的 buffer 中的对应项的有效位置为 0。

```
module BTB #( //Branch Target Buffer，采用直接映射
    parameter  BUFFER_ADDR_LEN = 12//决定了Buffer有多大
)(
    input  clk, rst,

    input [31:0] rd_PC,                 //输入PC
    output reg rd_predicted,                //对外输出的信号，表示rd_PC是跳转指令，此时rd_predicted_PC是有效数据
    output reg [31:0] rd_predicted_PC,  //从buffer中得到的预测PC
    input wr_req,                       //写请求信号
    input [31:0] wr_PC,                 //要写入的分支PC
    input [31:0] wr_predicted_PC,       //要写入的预测PC
    input wr_predicted_state_bit        //要写入的预测状态位
);

localparam TAG_ADDR_LEN = 32 - BUFFER_ADDR_LEN - 2; //计算tag的数据位宽
localparam BUFFER_SIZE = 1 << BUFFER_ADDR_LEN;      //计算buffer的大小

reg [TAG_ADDR_LEN - 1 : 0] PCTag           [0 : BUFFER_SIZE - 1];//BUFFER_SIZE个分支PC的TAG
reg [             31 : 0] PredictPC        [0 : BUFFER_SIZE - 1];//BUFFER_SIZE个预测PC
reg                       PredictStateBit  [0 : BUFFER_SIZE - 1];//BUFFER_SIZE个预测状态位

wire [BUFFER_ADDR_LEN - 1 : 0] rd_buffer_addr;//将输入地址拆分成3个部分
wire [   TAG_ADDR_LEN - 1 : 0] rd_tag_addr;
wire [             2 - 1 : 0] rd_word_addr; //PC是4的倍数，末2位总为0

wire [BUFFER_ADDR_LEN - 1 : 0] wr_buffer_addr;//将输入地址拆分成3个部分
wire [   TAG_ADDR_LEN - 1 : 0] wr_tag_addr;
wire [             2 - 1 : 0] wr_word_addr; //PC是4的倍数，末2位总为0

assign {rd_tag_addr, rd_buffer_addr, rd_word_addr} = rd_PC; //拆分 32bit rd_PC
assign {wr_tag_addr, wr_buffer_addr, wr_word_addr} = wr_PC; //拆分 32bit wr_PC

always @ (*) begin //判断输入的 PC 是否在 buffer 中命中
    if(PCTag[rd_buffer_addr] == rd_tag_addr && PredictStateBit[rd_buffer_addr])//如果tag与输入地址中的tag部分相等
        rd_predicted = 1'b1;
    else
        rd_predicted = 1'b0;
    rd_predicted_PC = PredictPC[rd_buffer_addr];
end

always @ (posedge clk or posedge rst) begin//写入buffer
    if(rst) begin
        for(integer i = 0; i < BUFFER_SIZE; i = i + 1) begin
            PCTag[i] = 0;
            PredictPC[i] = 0;
            PredictStateBit[i] = 1'b0;
        end
        rd_predicted = 1'b0;
        rd_predicted_PC = 0;
    end else begin
        if(wr_req) begin
            PCTag[wr_buffer_addr] <= wr_tag_addr;
            PredictPC[wr_buffer_addr] <= wr_predicted_PC;
            PredictStateBit[wr_buffer_addr] <= wr_predicted_state_bit;
        end
    end
end

endmodule
```

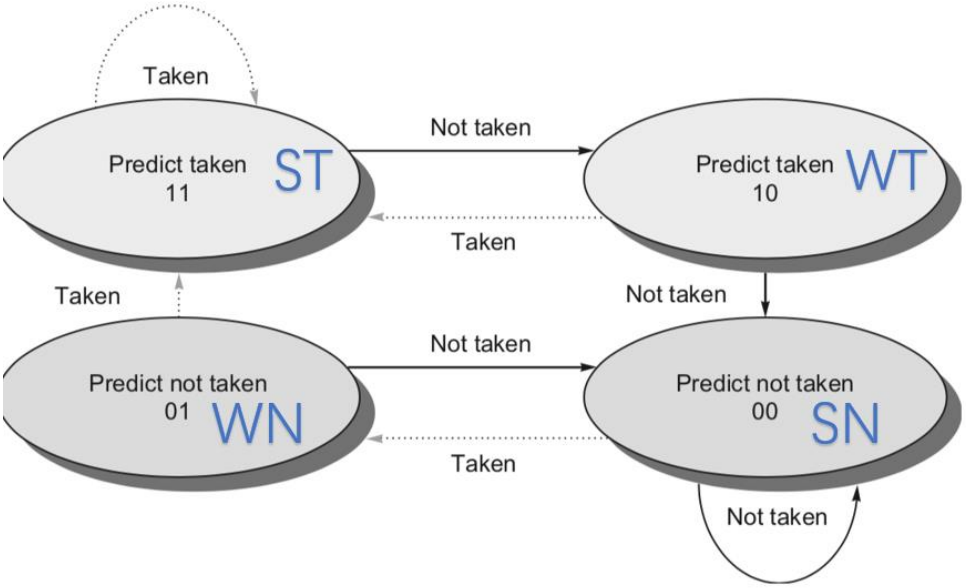
二、 BHT

分支策略采用 BHT 时， 分支 PC 和预测的 PC 仍然保存在 BTB 模块的 buffer 中，但 BTB 的 buffer 不再决定是否预测跳转， BTB 模块的命中与否只决定是否进行预测（或者说当前指令是否为分支指令）。预测的状态位（0 表示不预测， 1 表示预测） 和预测的 PC 也随流水线流到 EX 段。 BTB 的更新方法同上（不再使用有效位，只更新地址）。

BHT 模块则进行预测是否跳转。 维护一个表，每个表项为 2bit 的状态，查表同样采用直接映射，但不再维护和比较 tag。 BHT 模块对于输入的 PC，查到对应表项后，如果 2bit 的值为 0 或 1，则预测跳转，如果为 2 或 3，则预测不跳转，预测结果（0 表示不跳转， 1 表示跳转） 也随流水线流到 EX 段。

等到当前指令到达了 EX 段，就可以确定当前指令是否实际跳转。根据之前传过来的是否预测的状态位（BTB hit）、是否预测跳转状态位(BHT taken)为以及当前的实际跳转结果，我们可以决定是否清空流水线的 ID 和 EX 段进行重新取指令、是否对 BTB 进行更新（见下面的真值表），以及 BHT 如何更新（见下面的状态转换图）。

BHT 的每一项本质是一个 2bit 的状态机， 每次执行分支指令都要更新。 如果 EX 段实际跳转， 则对应表项状态按 0->1->2->3->...->3 更新，如果 EX 段实际不跳转， 则状态按 3->2->1->0->...->0 更新。



BTB hit	BHT taken	REAL	NPC_PRED	flush	NPC_REAL	BTB update
Y	Y	Y	BTB_BUF	N	BRANCH_PC (= BTB_BUF = NPC_PRED)	N
Y	Y	N	BTB_BUF	Y	PC_EX + 4 (!= NPC_PRED)	N
Y	N	Y	PC_IF+4	Y	BRANCH_PC (= BTB_BUF != NPC_PRED)	N
Y	N	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N
N	Y	Y	PC_IF+4	Y	BRANCH_PC (!= NPC_PRED)	Y
N	Y	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N
N	N	Y	PC_IF+4	Y	BRANCH_PC (!= NPC_PRED)	Y
N	N	N	PC_IF+4	N	PC_IF + 4 (= NPC_PRED)	N

```
23  module BHT #( //Branch History Table，采用直接映射
24      parameter  TABLE_ADDR_LEN = 12//决定了Table有多大，此处应该与BTB中的BUFFER_ADDR_LEN一致
25  )(
26      input  clk, rst,
27      input [31:0] rd_PC,                 //输入PC
28      output reg rd_predicted_taken,      //对外输出的信号，表示预测rd_PC跳转
29      input wr_req,                       //写请求信号
30      input [31:0] wr_PC,                 //要更新的分支PC
31      input [31:0] wr_taken               //要更新的分支PC实际是否跳转
32  );
33
34  localparam TABLE_SIZE = 1 << TABLE_ADDR_LEN;        //计算buffer的大小
35
36  reg [1 : 0] Table [0 : TABLE_SIZE - 1];//TABLE_SIZE个分支PC的状态
37
38  wire [TABLE_ADDR_LEN - 1 : 0] rd_table_addr;
39  wire [TABLE_ADDR_LEN - 1 : 0] wr_table_addr;
40
41
42  assign rd_table_addr = rd_PC[TABLE_ADDR_LEN + 1 : 2]; //取PC低为表地址，跳过末2位
43  assign wr_table_addr = wr_PC[TABLE_ADDR_LEN + 1 : 2]; //取PC低为表地址，跳过末2位
44
45  always @ (*) begin //状态0/1预测不跳转，2/3预测跳转
46      rd_predicted_taken = Table[rd_table_addr] >= 2'b10;
47  end
48
49  always @ (posedge clk or posedge rst) begin//写入buffer
50      if(rst) begin
51          for(integer i = 0; i < TABLE_SIZE; i = i + 1) begin
52              Table[i] = 2'b00;
53          end
54          rd_predicted_taken = 2'b00;
55      end else begin
56          if(wr_req) begin//更新PC对应表项的状态，如果实际taken:0->1->2->3->...->3，如果实际not tal
57              if(wr_taken) begin
58                  if(Table[wr_table_addr] != 2'b11)
59                      Table[wr_table_addr] <= Table[wr_table_addr] + 2'b01;
60                  else
61                      Table[wr_table_addr] <= Table[wr_table_addr];
62              end else begin
63                  if(Table[wr_table_addr] != 2'b00)
64                      Table[wr_table_addr] <= Table[wr_table_addr] - 2'b01;
65                  else
66                      Table[wr_table_addr] <= Table[wr_table_addr];
67              end
68          end
69      end
70  end
71
72  endmodule
73
```

实验结果

- BTB.S
    - 分支收益和分支代价：预测准确收益 2 个 cycle，预测错误 2 个 Cycle
    - 未使用分支预测的总周期数：(1024 - 8) / 2 = 508 cycle = 307 + 100 * 2 + 1
    - 使用分支预测的总周期数：(632 - 8) / 2 = 312 cycle = 307 + 2 * 2 + 1
    - 周期差值(未分支预测的总周期数-使用分支预测的总周期数)：196 cycle
    - 分支指令数：101 条
    - 动态分支预测正确次数和错误次数： 多命中 98 次，节省 196 个 cycle，加速比为 1.628
        - 跳转指令 101 条，正确 1 次，错误 100 次
        - 跳转指令 101 条，正确 99 次，错误 2 次
- BHT.S
    - 分支收益和分支代价：预测准确收益 2 个 cycle，预测错误 2 个 Cycle
    - 未使用分支预测的总周期数：(1074 - 8) / 2 = 533 cycle = 335 + 99 * 2
    - 使用 BTB 分支预测总数：(774-8) / 2 = 383
    - 周期差值(未分支预测的总周期数-使用 BTB 分支预测的总周期数)：150 cycle
    - 使用 BTB&BHT 分支预测的总周期数：(730 - 8) / 2 = 361 cycle = 335 + (11 + 2) * 2
    - 周期差值(未分支预测的总周期数-使用 BTB&BHT 分支预测的总周期数)：172 cycle
    - 分支指令数： 110 条
    - BTB 动态分支预测正确次数，错误次数：正确 86 次，错误 24 次，加速比为 1.414
    - BTB&BHT 动态分支预测正确次数和错误次数： 多命中 8 * 9 + 7 + 7 = 86 次，节省 172 个 cycle，加速比为 1.476

        - B: 跳转指令 110 条，正确 11 次，错误 99 次
        - A: 跳转指令 110 条，正确 9 * 9 + 8 + 8 = 97 次，错误 13 次

- Quicksort.S(256)
    - 分支收益和分支代价：预测准确收益 2 个 cycle，预测错误 2 个 Cycle
    - 未使用分支预测的总周期数：(136700 - 8) / 2 = 68346
    - 使用 BTB 分支预测总周期数：(138400 - 8) / 2 = 69196
    - 周期差值(未分支预测的总周期数-使用 BTB 分支预测的总周期数)：-850 cycle
    - 使用 BTB&BHT 分支预测的总周期数：(135572 - 8) / 2 = 67782
    - 周期差值(未分支预测的总周期数-使用 BTB&BHT 分支预测的总周期数)：564 cycle
    - 分支指令数： 16178
    - 使用 BTB 动态分支预测正确次数和错误次数：正确 14350 次，错误 1828，加速比为 0.993
    - BTB&BHT 动态分支预测正确次数和错误次数： 正确 15057 次，错误 1121 次，加速比为 1.012

- MatMul.S(16 * 16)
    - 分支收益和分支代价：预测准确收益 2 个 cycle，预测错误 2 个 Cycle
    - 未使用分支预测的总周期数：(709228 - 8) / 2 = 354610
    - 使用 BTB 分支预测的总周期数：(693640 - 8) / 2 = 346816
    - 使用 BTB&BHT 分支预测的总周期数：(692696 - 8) / 2 = 346344
    - 周期差值(未分支预测的总周期数-使用 BTB 分支预测的总周期数)：7794
    - 周期差值(未分支预测的总周期数-使用 BTB&BHT 分支预测的总周期数)：8266 cycle

- 分支指令数： 4624
- 动态分支预测 BTB 正确次数和错误次数：正确 4076 次，错误 548 次，加速比为 1.022
- 动态分支预测 BTB&BHT 正确次数和错误次数：正确 4312 次，错误 312 次，加速比为 1.024

对比不同策略并分析以上几点关系

- 分支收益和分支代价：流水段决定
- 未使用分支预测的总周期数：指令数 + 错误预测数 * 预测错误惩罚
- 使用分支预测总周期数：指令数 + 错误预测数 * 预测错误惩罚
- 两者差值：(未使用分支预测的错误预测数 - 使用分支预测的错误预测数) * 预测错误惩罚
- 未使用分支预测的错误预测数：跳转指令 - 循环个数(最后一条)
- BTB：使用分支预测的错误预测数 = 循环个数 * 2 (启动与退出)
- BHT(从 01 启动而不是 00)：使用分支预测的错误预测数 = 循环个数(最后一条) + 相同循环种数(启动)

总结 通过数据分析对比，可以看出 BHT 和 BTB 相结合的策略更好，它的预测错误次数少，总周期少，加速比大，分支预测的效果更好。在某些情况下，由于预测失误较多，可能得到使用分支预测的周期多于未使用分支预测周期的情况。