**《计算机体系结构》实验报告**

|  |
| --- |
| 一、实验目的   1. 熟悉并掌握--基于局部历史的分支预测。 2. 学习并理解--基于全局历史的分支预测。 3. 学习并理解--竞争的分支预测。 4. 了解其他分支预测内容。 5. 理解分支预测的意义。 |
| 二、实验项目内容   1. （最低要求）实现基于局部历史的分支指令方向预测。PS：要求在报告中给出指令计数器（PC）多路选择器的真值表。 2. （较高要求）实现基于全局历史的分支指令方向预测。PS：要求在预测时更新GHR 3. （最高要求）实现竞争的分支指令方向预测。 4. （学有余力）实现直接跳转指令的地址预测 5. （学有余力）实现间接跳转指令的地址预测 |
| 三、实验过程或算法（源程序）  0，实现分支预测在数据通路中的逻辑  设计思路：在取指令阶段实现动态分支预测，在指令译码阶段实现动态分支预测结果的判断，若预测结果错误，纠正PC的值，在指令执行阶段实现饱和计数器和表单的更新。  具体实现是基于计算机组成原理实验的MIPS五级流水CPU实现的，利用了其提供的数据前推和流水线暂停以及指令延迟槽机制，在此基础上，对datapath（数据链路）部分进行修改，添加动态分支预测模块。动态分支预测模块的具体实现与datapath是解耦合的，相互之间只通过相同的接口访问。  改进后的datapath的改进的部分如下图所示。  在取值阶段通过pc和instrF进行分支预测，输出pcsrc预测和jump预测信号，连接PC前的选择器，并生成F-D级的触发器控制信号。  在译码阶段，通过pcsrc预测、jump预测信号、计组实验实现的提前分支模块的pcsrc信号和jump信号判断预测是否正确，并生成纠错码jiucuoCode，其为一个三位宽度的信号，最高位表示是否发生错误，后两位表示具体错误情况。    具体verilog代码见mycpu.v文件  模块接口如下所示   1. //动态分支预测模块 2. module control\_dongtai ( 3. //预测部分 4. input wire [31:0] pc, 5. input wire[31:0] instrF, 6. output reg pcsrcFyuce, 7. output reg jumpFyuce, 8. //验证部分 9. input wire pcsrcD, 10. input wire pcsrcDyuce, 11. input wire jumpD, 12. input wire jumpDyuce, 13. output wire[2:0] jiucuo, 14. //更新表单部分 15. input wire[31:0] pcE 16. );   部分添加的datapath代码：      通过实现一个很基础的分支策略验证datapath：实现一个遇到分支指令就预测跳转的算法，并通过仿真验证datapath是否被打通。      至此，数据链路已被打通，可以实现具体的动态分支预测算法。  1，实现基于局部历史的分支指令方向预测  基于局部历史的分支预测算法的实现：  设计思路：  对于要判断的指令，首先通过hash算法将PC值映射为一个k位的位串A，然后以此位串A为地址，从BHR中寻找出以此为地址的元素，这个元素表示此分支指令对应的跳转与否的历史，也就是前n次跳转情况组成的串B，以此串为地址，可以从PHT中获取此地址对应的饱和计数器，然后根据这个饱和计数器的值判断是否跳转。  关于更新阶段，本实现是在执行阶段对BHR和PHT进行更新的，根据实际的跳转情况，更新BHR和PHT：首先通过PC对应的BHR找出对应的PHT中的饱和计数器，更新饱和计数器的状态，然后将此PC对应的BHR的值左移一位，然后填充正确的跳转情况。    本实验中，采用PC的[6:2]位（共5位）作为hash，然后通过这5位索引查找BHR,BHR共32个2位寄存器，对应4个饱和计数器。  具体实现代码如下：   1. BHR部分（32个2位寄存器） 2. module bhr\_32( 3. input wire clk, 4. input wire we3, 5. input wire[4:0] ra1,wa3, 6. input wire[1:0] wd3, 7. output wire[1:0] rd1 8. ); 9. reg [1:0] rf[31:0]; 10. always @(posedge clk) begin 11. **if**(we3) begin 12. rf[wa3] <= wd3; 13. end 14. end 15. assign rd1 = (ra1 != 0) ? rf[ra1] : 0; 16. endmodule 17. PHT部分（4个饱和计数器） 18. module pht\_4( 19. input wire clk, 20. input wire we3, 21. input wire[1:0] ra1,wa3, 22. input wire wd3, 23. output wire[1:0] rd1 24. ); 25. reg [1:0] rf[3:0]; 26. always @(posedge clk) begin 27. **if**(we3) begin 28. **case**({rf[wa3],wd3}) 29. 3'b000:rf[wa3]<=2'b00; 30. 3'b001:rf[wa3]<=2'b01; 31. 3'b010:rf[wa3]<=2'b00; 32. 3'b011:rf[wa3]<=2'b11; 33. 3'b100:rf[wa3]<=2'b01; 34. 3'b101:rf[wa3]<=2'b11; 35. 3'b110:rf[wa3]<=2'b10; 36. 3'b111:rf[wa3]<=2'b11; 37. **default**:rf[wa3]<=rf[wa3]; 38. endcase 39. end 40. end 41. assign rd1 = (ra1 != 0) ? rf[ra1] : 0; 42. endmodule     wd3接口为此分支真正的跳转情况，case部分实现了根据计数器当前状况和当前分支真实的跳转情况更新计数器的状态，状态转化图见上图。   1. 分支预测链路部分 2. //动态分支预测模块 3. module control\_dongtai ( 4. input wire clk, 5. //预测部分 6. input wire [31:0] pc, 7. input wire[31:0] instrF, 8. output reg pcsrcFyuce, 9. output reg jumpFyuce, 10. //验证部分 11. input wire pcsrcD, 12. input wire pcsrcDyuce, 13. input wire jumpD, 14. input wire jumpDyuce, 15. output reg[2:0] jiucuo, 16. //更新表单部分 17. input wire[31:0] pcE 18. ); 19. //预测 20. wire [4:0] suoyinF,suoyinE; 21. wire [1:0] bhrF,bhrD,bhrE; 22. wire [1:0] jishuqi,jishuqiD,jishuqiE; 23. wire pcsrcE; 24. assign suoyinF = pc[6:2]; 26. //连接bhr 27. bhr\_32 myBHR( 28. .clk(clk), 29. .we3(1'b1),      //写使能 30. .ra1(suoyinF),//要读的地址 suoyin为地址 31. .wa3(suoyinE),      //要写的地址 32. .wd3({bhrE[0:0],pcsrcE}),      //写入导入数据 33. .rd1(bhrF)       //读出的数据 34. ); 35. //根据bhr(局部历史寄存器)找到对应的饱和计数器 36. pht\_4 myPHT( 37. .clk(clk), 38. .we3(1'b1),      //写使能 39. .ra1(bhrF),//要读的地址 bhr为地址 40. .wa3(bhrE),      //要写的地址 41. .wd3(pcsrcE),      //写入导入数据 42. .rd1(jishuqi)       //读出的数据 43. ); 44. //根据计数器判断是否跳转 45. always @(\*) begin 46. //跳转指令一定执行 47. **case**(instrF[31:26]) 48. 6'b000010:jumpFyuce<=1'b1; 49. **default**:jumpFyuce<=1'b0; 50. endcase 51. **case**(jishuqi) 52. 2'b00:begin 53. pcsrcFyuce<=1'b0; 54. end 55. 2'b01:begin 56. pcsrcFyuce<=1'b0; 57. end 58. 2'b10:begin 59. pcsrcFyuce<=1'b1; 60. end 61. 2'b11:begin 62. pcsrcFyuce<=1'b1; 63. end 64. **default**:begin 65. pcsrcFyuce<=1'b0; 66. end 67. endcase 68. end 70. //验证 71. always @(\*)begin 72. **case**({pcsrcD,pcsrcDyuce}) 73. 2'b00:jiucuo<=3'b000; 74. 2'b01:jiucuo<=3'b100; 75. 2'b10:jiucuo<=3'b101; 76. 2'b11:jiucuo<=3'b000; 77. **default**:jiucuo<=3'b000; 78. endcase 79. end 80. //更新表单 81. //加入流水线 82. flopr #(2) flopenrc\_400(clk,1'b0,bhrF,bhrD); 83. flopr #(2) flopenrc\_401(clk,1'b0,bhrD,bhrE); 84. flopr #(1) flopenrc\_402(clk,1'b0,pcsrcD,pcsrcE); 85. assign suoyinE = pcE[6:2];  88. endmodule   2，实现基于全局历史的分支指令方向预测（在预测阶段更新GHR）  设计思路：  判断一个分支指令，首先通过hash得到pc一组索引，然后和GHR做拼接得到PHT的地址，然后通过此地址对应的饱和计数器的状态判断是否进行分支跳转。在此阶段，更新GHR。（通过左移一位并在最低为补充预测结果），后面通过提交阶段恢复法实现GHR的维护。    对于表单的更新，此算法涉及GHR表和饱和计数器的更新，首先，存在一个GHR的副本，称为Retired GHR，此Retired GHR在预测阶段不会更新。在更新阶段，根据Retired GHR的值和pc的hash结果以及真实跳转情况，更新饱和计数器，然后更新Retired GHR（通过左移一位并在最低为补充真实结果），当在判断判断预测错误时，将此Retired GHR的值赋值给GHR。（提交阶段修复法）  具体实现代码如下：  1，PHT部分（由256个饱和计数器组成）状态转移图同练习1   1. module pht\_256( 2. input wire clk, 3. input wire we3, 4. input wire[7:0] ra1,wa3, 5. input wire wd3, 6. output wire[1:0] rd1 7. ); 8. reg [1:0] rf[255:0]; 9. always @(posedge clk) begin 10. **if**(we3) begin 11. **case**({rf[wa3],wd3}) 12. 3'b000:rf[wa3]<=2'b00; 13. 3'b001:rf[wa3]<=2'b01; 14. 3'b010:rf[wa3]<=2'b00; 15. 3'b011:rf[wa3]<=2'b11; 16. 3'b100:rf[wa3]<=2'b01; 17. 3'b101:rf[wa3]<=2'b11; 18. 3'b110:rf[wa3]<=2'b10; 19. 3'b111:rf[wa3]<=2'b11; 20. **default**:rf[wa3]<=rf[wa3]; 21. endcase 22. end 23. end 24. assign rd1 = (ra1 != 0) ? rf[ra1] : 0; 25. endmodule   2，GHR部分（由3位寄存器组成）   1. module ghr\_3( 2. input wire clk, 3. input wire we3,we4,re1,//能端 4. input wire wd3,//wd3用于加入预测结果 5. input wire[2:0] wd4,//wd4用于retiredGHR覆盖 6. output wire[2:0] rd1 7. ); 8. reg [2:0] rf; 9. always @(posedge clk) begin 10. **if**(we3) begin 11. rf <= {rf[1:0],wd3}; 12. end 13. **if**(we4) begin 14. rf <= wd4; 15. end 16. end 17. assign rd1 = (re1 != 0) ? rf : 0; 18. endmodule   3，基于全局历史的分支预测链路部分   1. //动态分支预测模块 2. module control\_dongtai ( 3. input wire clk, 4. //预测部分 5. input wire [31:0] pc, 6. input wire[31:0] instrF, 7. output reg pcsrcFyuce, 8. output reg jumpFyuce, 9. //验证部分 10. input wire pcsrcD, 11. input wire pcsrcDyuce, 12. input wire jumpD, 13. input wire jumpDyuce, 14. output reg[2:0] jiucuo, 15. //更新表单部分 16. input wire[31:0] pcE 17. ); 19. wire [7:0] suoyinF,suoyinD,suoyinE; 20. wire [2:0] ghrF; 21. wire jishuqi; 22. wire [2:0] ghr\_read; 23. wire pcsrcEyuce; 25. //全局历史寄存器，两个写入一个读出端 26. ghr\_3 myGHR( 27. .clk(clk), 28. .we3(1'b1), 29. .we4(pcsrcEyuce ^ pcsrcE),//实现retired写回的能端，为1表示需要替换GHR 30. //E阶段判断是否需要替换GHR 31. .re1(1'b1), 32. .wd3(pcsrcFyuce|jumpFyuce),//预测结果写入 33. .wd4(ghr\_read),//替换 34. .rd1(ghrF) 35. ); 36. //拼接得到索引 37. assign suoyinF = {pc[6:2],ghrF}; 38. //查找pht 39. pht\_256 myPHT2( 40. .clk(clk), 41. .we3(1'b1),      //写使能 42. .ra1(suoyinF),//要读的地址 43. .wa3(suoyinE),      //要写的地址 44. .wd3(pcsrcE),      //写入导入数据 45. .rd1(jishuqi)       //读出的数据 46. ); 47. //预测分支结果 48. always @(\*)begin 49. **case**(instrF[31:26]) 50. 6'b000010:jumpFyuce<=1'b1; 51. **default**:jumpFyuce<=1'b0; 52. endcase 53. **case**(jishuqi) 54. 2'b00:begin 55. pcsrcFyuce<=1'b0; 56. end 57. 2'b01:begin 58. pcsrcFyuce<=1'b0; 59. end 60. 2'b10:begin 61. pcsrcFyuce<=1'b1; 62. end 63. 2'b11:begin 64. pcsrcFyuce<=1'b1; 65. end 66. **default**:begin 67. pcsrcFyuce<=1'b0; 68. end 69. endcase 70. end  73. //验证 74. always @(\*)begin 75. **case**({pcsrcD,pcsrcDyuce}) 76. 2'b00:jiucuo<=3'b000; 77. 2'b01:jiucuo<=3'b100; 78. 2'b10:jiucuo<=3'b101; 79. 2'b11:jiucuo<=3'b000; 80. **default**:jiucuo<=3'b000; 81. endcase 82. end  85. //更新阶段判断是否修改 86. //retiredGHR只需要一个写入端we3和wd3，we4能端为0，表示不使用 87. ghr\_3 retiredGHR( 88. .clk(clk), 89. .we3(1'b1), 90. .we4(1'b0), 91. .re1(1'b1),//读端 92. .wd3(pcsrcE),//E阶段开始写数据 93. .wd4(3'b000), 94. .rd1(ghr\_read)//E阶段写入后读数据 95. ); 97. //更新变量 98. //加入流水线 99. flopr #(1) flopenrc\_404(clk,1'b0,pcsrcD,pcsrcE); 100. flopr #(8) flopenrc\_406(clk,1'b0,suoyinF,suoyinD); 101. flopr #(8) flopenrc\_407(clk,1'b0,suoyinD,suoyinE);  104. endmodule   3，实现竞争的分支指令方向预测  设计思路：  对于预测阶段，首先生成PC的hash，本次实验采用切片，PC经过hash产生PC[6:2] 5位索引，然后分别作局部历史分支预测和全局历史分支预测，然后通过与全局历史GHR拼接后的索引查找CPHT，然后更新GHR（使用提交阶段恢复法实现GHR的维护），CPHT用于存放饱和计数器，此饱和计数器用于判断使用全局历史分支预测或局部历史分支预测。  对于更新阶段，通过Retired GHR机制维护GHR的正常，并更新BHR和PHT1，HT2，CPHT。  具体实现代码如下：  1，竞争机制下的动态分支预测数据链路部分   1. //动态分支预测模块 2. module control\_dongtai ( 3. input wire clk, 4. //预测部分 5. input wire [31:0] pc, 6. input wire[31:0] instrF, 7. output reg pcsrcFyuce, 8. output reg jumpFyuce, 9. //验证部分 10. input wire pcsrcD, 11. input wire pcsrcDyuce, 12. input wire jumpD, 13. input wire jumpDyuce, 14. output reg[2:0] jiucuo, 15. //更新表单部分 16. input wire[31:0] pcE 17. ); 18. wire [7:0] suoyinF,suoyinE; 19. wire pcsrcFyuceQuanjv,pcsrcFyuceJvbu; 20. wire jumpFyuceQuanjv,jumpFyuceJvbu; 21. wire[2:0] jiucuoCodeQuanjv,jiucuoCodeJvbu; 22. wire pcsrcDyuceQuanjv,pcsrcDyuceJvbu; 23. wire jumpDyuceQuanjv,jumpDyuceJvbu; 24. wire pcsrcEyuceQuanjv,pcsrcEyuceJvbu; 25. wire jumpEyuceQuanjv,jumpEyuceJvbu; 26. wire pcsrcE; 28. //连接全局历史预测模块 29. quanjv myquanjv( 30. .clk(clk), 31. .pc(pc), 32. .instrF(instrF), 33. .pcsrcFyuce(pcsrcFyuceQuanjv),//输出 34. .jumpFyuce(jumpFyuceQuanjv),//输出 35. .pcsrcD(pcsrcD), 36. .pcsrcDyuce(pcsrcDyuceQuanjv), 37. .jumpD(jumpD), 38. .jumpDyuce(jumpDyuceQuanjv), 39. .jiucuo(jiucuoCodeQuanjv),//输出 40. .pcE(pcE), 41. //相比于前两个练习的新增部分 42. .suoyinF(suoyinF), 43. .suoyinE(suoyinE) 44. ); 46. //索引通过与hash和与GHR拼接得到8位索引，在全局预测模块中得到 47. //CPHT部分 48. cpht\_256 cpht( 49. .clk(clk), 50. .we3(1'b1),         //写使能 51. .ra1(suoyinF),      //要读的地址 52. .wa3(suoyinE),      //要写的地址 53. //写入CPHT的数据，高位表示局部分支预测是否错误，低位表示全局分支预测是否错误 54. //E阶段正式写入表（clk与流水线CPU的clk相同，都为posedge） 55. .wd3({pcsrcEyuceJvbu^pcsrcE,pcsrcEyuceQuanjv^pcsrcE}), 56. .rd1(jishuqi)       //读出的数据 57. ); 58. reg swtichF; 59. wire swtichD,swtichE;//选择信号，为0使用局部历史，为1使用全局历史 60. always @(\*) begin 61. **case**(jishuqi) 62. 2'b00:swtichF<=1'b1; 63. 2'b01:swtichF<=1'b1; 64. 2'b10:swtichF<=1'b0; 65. 2'b11:swtichF<=1'b0; 66. **default**:swtichF<=1'b0; 67. endcase 68. end 70. //连接局部历史预测模块 71. jvbu myjvbu( 72. .clk(clk), 73. .pc(pc), 74. .instrF(instrF), 75. .pcsrcFyuce(pcsrcFyuceJvbu),//输出 76. .jumpFyuce(jumpFyuceJvbu),//输出 77. .pcsrcD(pcsrcD), 78. .pcsrcDyuce(pcsrcDyuceJvbu), 79. .jumpD(jumpD), 80. .jumpDyuce(jumpDyuceJvbu), 81. .jiucuo(jiucuoCodeJvbu),//输出 82. .pcE(pcE) 83. ); 84. //根据swtich选择输出预测信号 85. always @(\*) begin 86. **case**(swtichF) 87. 1'b0:begin 88. pcsrcFyuce<=pcsrcFyuceJvbu; 89. jumpFyuce<=jumpFyuceJvbu; 90. end 91. 1'b1:begin 92. pcsrcFyuce<=pcsrcFyuceQuanjv; 93. jumpFyuce<=jumpFyuceQuanjv; 94. end 95. **default**:begin 96. pcsrcFyuce<=pcsrcFyuceJvbu; 97. jumpFyuce<=jumpFyuceJvbu; 98. end 99. endcase 100. end 101. //处理错误情况（输出错误码） 102. //swtich信号加入流水线 103. flopr #(1) flopenrc\_408(clk,1'b0,swtichF,swtichD); 104. flopr #(1) flopenrc\_409(clk,1'b0,swtichD,swtichE); 105. //预测信号加入流水线 106. flopr #(1) flopenrc\_410(clk,1'b0,pcsrcFyuceJvbu,pcsrcDyuceJvbu); 107. flopr #(1) flopenrc\_411(clk,1'b0,pcsrcFyuceQuanjv,pcsrcDyuceQuanjv); 108. flopr #(1) flopenrc\_412(clk,1'b0,jumpFyuceJvbu,jumpDyuceJvbu); 109. flopr #(1) flopenrc\_413(clk,1'b0,jumpFyuceQuanjv,jumpDyuceQuanjv); 110. flopr #(1) flopenrc\_414(clk,1'b0,pcsrcDyuceJvbu,pcsrcEyuceJvbu); 111. flopr #(1) flopenrc\_415(clk,1'b0,pcsrcDyuceQuanjv,pcsrcEyuceQuanjv); 112. flopr #(1) flopenrc\_416(clk,1'b0,jumpDyuceJvbu,jumpEyuceJvbu); 113. flopr #(1) flopenrc\_417(clk,1'b0,jumpDyuceQuanjv,jumpEyuceQuanjv); 114. flopr #(1) flopenrc\_418(clk,1'b0,pcsrcD,pcsrcE); 116. //处理预测错误 117. always @(\*)begin 118. **case**(swtichD) 119. 1'b0:jiucuo<=jiucuoCodeJvbu; 120. 1'b1:jiucuo<=jiucuoCodeQuanjv; 121. **default**:jiucuo<=jiucuoCodeJvbu; 122. endcase 123. end  126. endmodule   2，局部竞争分支预测模块   1. module jvbu( 2. input wire clk, 3. //预测部分 4. input wire [31:0] pc, 5. input wire[31:0] instrF, 6. output reg pcsrcFyuce, 7. output reg jumpFyuce, 8. //验证部分 9. input wire pcsrcD, 10. input wire pcsrcDyuce, 11. input wire jumpD, 12. input wire jumpDyuce, 13. output reg[2:0] jiucuo, 14. //更新表单部分 15. input wire[31:0] pcE 16. ); 17. //预测 18. wire [4:0] suoyinF,suoyinE; 19. wire [1:0] bhrF,bhrD,bhrE; 20. wire [1:0] jishuqi,jishuqiD,jishuqiE; 21. wire pcsrcE; 22. assign suoyinF = pc[6:2]; 24. //连接bhr 25. bhr\_32 myBHR( 26. .clk(clk), 27. .we3(1'b1),      //写使能 28. .ra1(suoyinF),//要读的地址 suoyin为地址 29. .wa3(suoyinE),      //要写的地址 30. .wd3({bhrE[0:0],pcsrcE}),      //写入导入数据 31. .rd1(bhrF)       //读出的数据 32. ); 33. //根据bhr(局部历史寄存器)找到对应的饱和计数器 34. pht\_4 myPHT( 35. .clk(clk), 36. .we3(1'b1),      //写使能 37. .ra1(bhrF),//要读的地址 bhr为地址 38. .wa3(bhrE),      //要写的地址 39. .wd3(pcsrcE),      //写入导入数据 40. .rd1(jishuqi)       //读出的数据 41. ); 42. //根据计数器判断是否跳转 43. always @(\*) begin 44. //跳转指令一定执行 45. **case**(instrF[31:26]) 46. 6'b000010:jumpFyuce<=1'b1; 47. **default**:jumpFyuce<=1'b0; 48. endcase 49. **case**(jishuqi) 50. 2'b00:begin 51. pcsrcFyuce<=1'b0; 52. end 53. 2'b01:begin 54. pcsrcFyuce<=1'b0; 55. end 56. 2'b10:begin 57. pcsrcFyuce<=1'b1; 58. end 59. 2'b11:begin 60. pcsrcFyuce<=1'b1; 61. end 62. **default**:begin 63. pcsrcFyuce<=1'b0; 64. end 65. endcase 66. end 68. //验证 69. always @(\*)begin 70. **case**({pcsrcD,pcsrcDyuce}) 71. 2'b00:jiucuo<=3'b000; 72. 2'b01:jiucuo<=3'b100; 73. 2'b10:jiucuo<=3'b101; 74. 2'b11:jiucuo<=3'b000; 75. **default**:jiucuo<=3'b000; 76. endcase 77. end 78. //更新表单 79. //加入流水线 80. flopr #(2) flopenrc\_400(clk,1'b0,bhrF,bhrD); 81. flopr #(2) flopenrc\_401(clk,1'b0,bhrD,bhrE); 82. flopr #(1) flopenrc\_402(clk,1'b0,pcsrcD,pcsrcE); 83. assign suoyinE = pcE[6:2]; 84. endmodule   3，全局竞争分支预测模块   1. module quanjv( 2. input wire clk, 3. //预测部分 4. input wire [31:0] pc, 5. input wire[31:0] instrF, 6. output reg pcsrcFyuce, 7. output reg jumpFyuce, 8. //验证部分 9. input wire pcsrcD, 10. input wire pcsrcDyuce, 11. input wire jumpD, 12. input wire jumpDyuce, 13. output reg[2:0] jiucuo, 14. //更新表单部分 15. input wire[31:0] pcE, 16. //新增部分 17. output wire[7:0] suoyinF, 18. output wire[7:0] suoyinE 19. ); 21. wire [7:0] suoyinD; 22. wire [2:0] ghrF; 23. wire jishuqi; 24. wire [2:0] ghr\_read; 25. wire pcsrcEyuce; 27. //全局历史寄存器，两个写入一个读出端 28. ghr\_3 myGHR( 29. .clk(clk), 30. .we3(1'b1), 31. .we4(pcsrcEyuce ^ pcsrcE),//实现retired写回的能端，为1表示需要替换GHR 32. //E阶段判断是否需要替换GHR 33. .re1(1'b1), 34. .wd3(pcsrcFyuce|jumpFyuce),//预测结果写入 35. .wd4(ghr\_read),//替换 36. .rd1(ghrF) 37. ); 38. //拼接得到索引 39. assign suoyinF = {pc[6:2],ghrF}; 40. //查找pht 41. pht\_256 myPHT2( 42. .clk(clk), 43. .we3(1'b1),      //写使能 44. .ra1(suoyinF),//要读的地址 45. .wa3(suoyinE),      //要写的地址 46. .wd3(pcsrcE),      //写入导入数据 47. .rd1(jishuqi)       //读出的数据 48. ); 49. //预测分支结果 50. always @(\*)begin 51. **case**(instrF[31:26]) 52. 6'b000010:jumpFyuce<=1'b1; 53. **default**:jumpFyuce<=1'b0; 54. endcase 55. **case**(jishuqi) 56. 2'b00:begin 57. pcsrcFyuce<=1'b0; 58. end 59. 2'b01:begin 60. pcsrcFyuce<=1'b0; 61. end 62. 2'b10:begin 63. pcsrcFyuce<=1'b1; 64. end 65. 2'b11:begin 66. pcsrcFyuce<=1'b1; 67. end 68. **default**:begin 69. pcsrcFyuce<=1'b0; 70. end 71. endcase 72. end  75. //验证 76. always @(\*)begin 77. **case**({pcsrcD,pcsrcDyuce}) 78. 2'b00:jiucuo<=3'b000; 79. 2'b01:jiucuo<=3'b100; 80. 2'b10:jiucuo<=3'b101; 81. 2'b11:jiucuo<=3'b000; 82. **default**:jiucuo<=3'b000; 83. endcase 84. end  87. //更新阶段判断是否修改 88. //retiredGHR只需要一个写入端we3和wd3，we4能端为0，表示不使用 89. ghr\_3 retiredGHR( 90. .clk(clk), 91. .we3(1'b1), 92. .we4(1'b0), 93. .re1(1'b1),//读端 94. .wd3(pcsrcE),//更新阶段写数据,E阶段开始写数据 95. .wd4(3'b000), 96. .rd1(ghr\_read)//E阶段写入后读数据 97. ); 99. //更新变量 100. //加入流水线 101. flopr #(1) flopenrc\_404(clk,1'b0,pcsrcD,pcsrcE); 102. flopr #(8) flopenrc\_406(clk,1'b0,suoyinF,suoyinD); 103. flopr #(8) flopenrc\_407(clk,1'b0,suoyinD,suoyinE);  106. endmodule   4，CPHT部分（本实现采用8位索引，即256个饱和计数器）   1. module cpht\_256( 2. input wire clk, 3. input wire we3, 4. input wire[7:0] ra1,wa3, 5. input wire [1:0] wd3, 6. output wire[1:0] rd1 7. ); 8. reg [1:0] rf[255:0]; 9. always @(posedge clk) begin 10. **if**(we3) begin 11. **case**({rf[wa3],wd3}) 12. 4'b0000:rf[wa3]<=rf[wa3]; 13. 4'b0001:rf[wa3]<=2'b01; 14. 4'b0010:rf[wa3]<=2'b00; 15. 4'b0011:rf[wa3]<=rf[wa3]; 16. 4'b0100:rf[wa3]<=rf[wa3]; 17. 4'b0101:rf[wa3]<=2'b10; 18. 4'b0110:rf[wa3]<=2'b00; 19. 4'b0111:rf[wa3]<=rf[wa3]; 20. 4'b1000:rf[wa3]<=rf[wa3]; 21. 4'b1001:rf[wa3]<=2'b11; 22. 4'b1010:rf[wa3]<=2'b01; 23. 4'b1011:rf[wa3]<=rf[wa3]; 24. 4'b1100:rf[wa3]<=rf[wa3]; 25. 4'b1101:rf[wa3]<=2'b11; 26. 4'b1110:rf[wa3]<=2'b10; 27. 4'b1111:rf[wa3]<=rf[wa3]; 28. **default**:rf[wa3]<=rf[wa3]; 29. endcase 30. end 31. end 32. assign rd1 = (ra1 != 0) ? rf[ra1] : 0; 33. endmodule   CPHT的饱和计数器状态转换图如下：    5，对于局部历史分支预测和全局历史分支预测模块内部的BHR、GHR和PHT部分与练习1和练习2的相同，详细代码见练习1和练习2  4，完整代码见文档附件myCPU.v文件（全部代码）。 |
| 四，程序运行结果  1，指令计数器（PC）多路选择器的真值表      2，局部分支预测的仿真结果    3，全局分支预测的仿真结果    4，基于竞争的分支预测的仿真结果 |

备注：