

Lab6 Report

实验原理

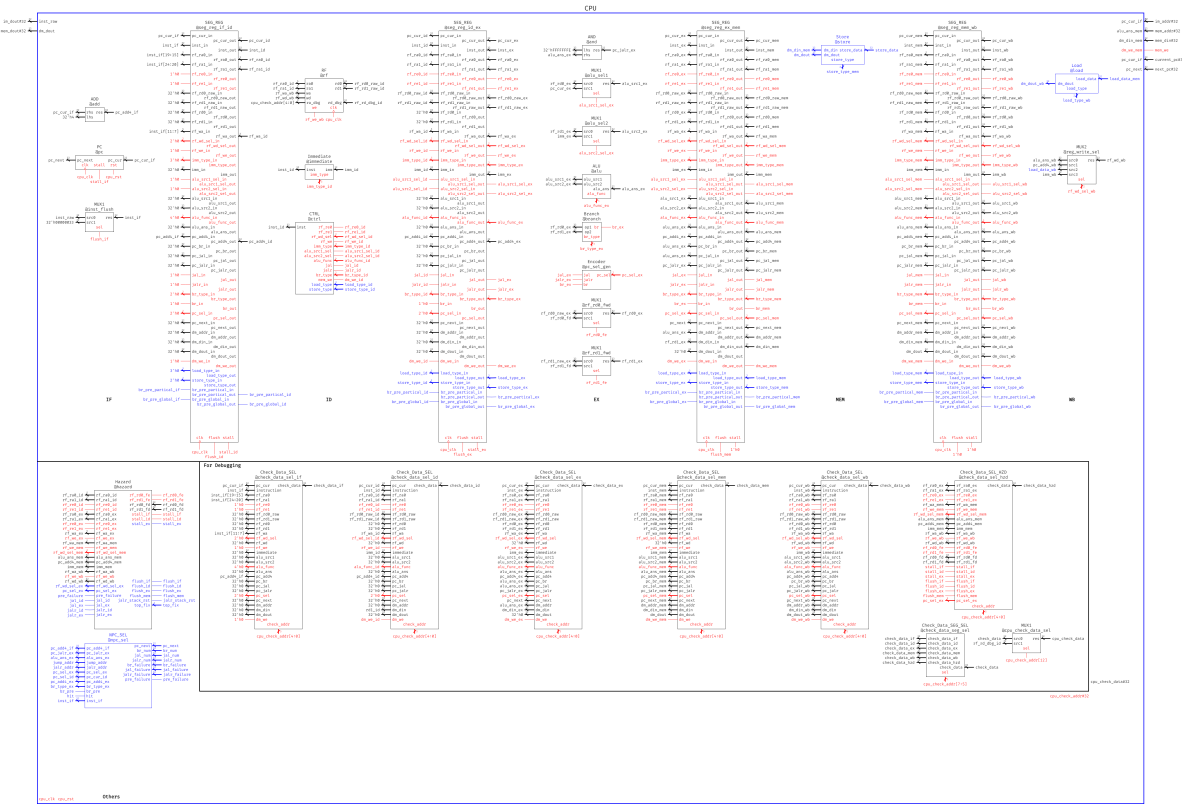
实验内容

- 1. 必做：RV32-I 指令子集扩展，在流水线CPU中实现所有非系统 RV32-I 指令(共 27 条)
- 2. 选做：2bits 感知机局部历史分支预测；2bits 感知机全局历史分支预测

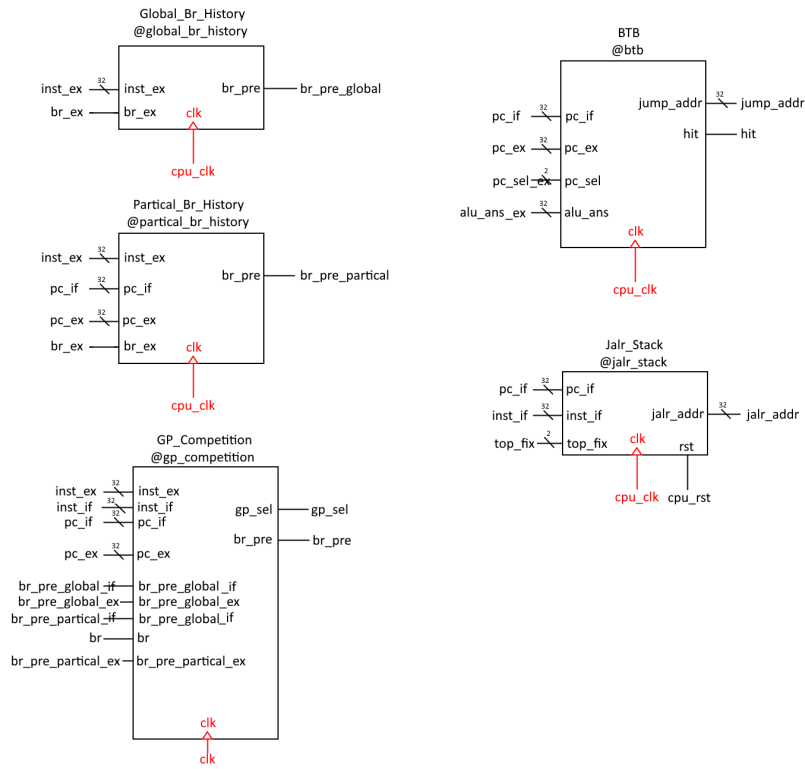
设计流程

数据通路

除分支预测模块之外的数据通路与Lab5的必做部分并无太多不同，图中蓝色部分为改动的部分，包含 L-Type S-Type 指令的实现和分支预测的部分实现



分支预测关键模块的数据通路



关键模块设计

CTRL模块

发送不同指令对应的不同控制信号，具体含义在其他模块中解释

```

1  module CTRL (
2      input [31:0] inst,
3      output reg jal,
4      output reg jalr,
5      output reg [2:0] br_type,
6      output reg alu_src1_sel,
7      output reg alu_src2_sel,
8      output reg [3:0] alu_func,
9      output reg mem_we,
10     output reg [2:0] imm_type,
11     output reg rf_re0,
12     output reg rf_re1,
13     output reg [1:0] rf_wd_sel,
14     output reg rf_we,
15     output reg [2:0] load_type,
16     output reg [1:0] store_type
17 );
18     always @(*) begin
19         jal=0; jalr=0; br_type=0;
20         alu_src1_sel=0; alu_src2_sel=0;
21         alu_func=0; mem_we=0; imm_type=0; load_type=0; store_type=0;
22         rf_re0=0; rf_re1=0; rf_wd_sel=0; rf_we=0;
23         case (inst[6:0])
24             7'b0010011:begin
25                 alu_src1_sel=0;
26                 alu_src2_sel=1;
27                 imm_type=1;
28                 rf_wd_sel=0;

```

```

29         rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
30         rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
31         case (inst[14:12])
32             3'b000: alu_func=4'b0000;    //addi
33             3'b001: alu_func=4'b1001;    //slli
34             3'b010: alu_func=4'b0100;    //slti
35             3'b011: alu_func=4'b0011;    //sltiu
36             3'b100: alu_func=4'b0111;    //xori
37             3'b101:begin
38                 case (inst[31:25])
39                     7'b0000000: alu_func=4'b1000;    //srli
40                     7'b0100000: alu_func=4'b1010;    //srai
41                 endcase
42             end
43             3'b110: alu_func=4'b0110;    //ori
44             3'b111: alu_func=4'b0101;    //andi
45         endcase
46     end
47     7'b0110011:begin
48         alu_src1_sel=0;
49         alu_src2_sel=0;
50         rf_wd_sel=0;
51         rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
52         rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
53         rf_re1= (inst[24:20]==5'b00000) ? 0 : 1;
54         case (inst[14:12])
55             3'b000:begin
56                 case (inst[31:25])
57                     7'b0000000: alu_func=4'b0000;    //add
58                     7'b0100000: alu_func=4'b0001;    //sub
59                 endcase
60             end
61             3'b001: alu_func=4'b1001;    //sll
62             3'b010: alu_func=4'b0100;    //slt
63             3'b011: alu_func=4'b0011;    //sltu
64             3'b100: alu_func=4'b0111;    //xor
65             3'b101:begin
66                 case (inst[31:25])
67                     7'b0000000: alu_func=4'b1000;    //srli
68                     7'b0100000: alu_func=4'b1010;    //srai
69                 endcase
70             end
71             3'b110: alu_func=4'b0110;    //or
72             3'b111: alu_func=4'b0101;    //and
73         endcase
74     endcase
75 end
76 7'b0110111:begin
77     alu_src2_sel=1;
78     imm_type=4;
79     rf_wd_sel=3;
80     rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
81     alu_func=4'b0000;    //lui
82 end
83 7'b0010111:begin

```

```

84         alu_src1_sel=1;
85         alu_src2_sel=1;
86         imm_type=4;
87         rf_wd_sel=0;
88         rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
89         alu_func=4'b0000;    //auipc
90     end
91     7'b1101111:begin
92         jal=1;
93         alu_src1_sel=1;
94         alu_src2_sel=1;
95         imm_type=5;
96         rf_wd_sel=1;
97         rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
98         alu_func=4'b0000;    //jal
99     end
100    7'b1100111:begin
101        jalr=1;
102        alu_src1_sel=0;
103        alu_src2_sel=1;
104        imm_type=1;
105        rf_wd_sel=1;
106        rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
107        rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
108        alu_func=4'b0000;    //jalr
109    end
110    7'b1100011:begin
111        alu_src1_sel=1;
112        alu_src2_sel=1;
113        imm_type=3;
114        alu_func=4'b0000;
115        rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
116        rf_re1= (inst[24:20]==5'b00000) ? 0 : 1;
117        case (inst[14:12])
118            3'b000: br_type=1;    //beq
119            3'b001: br_type=2;    //bne
120            3'b100: br_type=3;    //blt
121            3'b101: br_type=4;    //bge
122            3'b110: br_type=5;    //bltu
123            3'b111: br_type=6;    //bgeu
124        endcase
125    end
126    7'b0000011:begin
127        alu_src1_sel=0;
128        alu_src2_sel=1;
129        imm_type=1;
130        alu_func=4'b0000;
131        rf_wd_sel=2;
132        rf_we= (inst[11:7]==5'b00000) ? 0 : 1;
133        rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
134        case (inst[14:12])
135            3'b000: load_type=1;    //lb
136            3'b001: load_type=2;    //lh
137            3'b010: load_type=3;    //lw
138            3'b100: load_type=4;    //lbu

```

```

139         3'b101: load_type=5;    //1hu
140     endcase
141 end
142 7'b0100011:begin
143     alu_src1_sel=0;
144     alu_src2_sel=1;
145     imm_type=2;
146     alu_func=4'b0000;
147     rf_re0= (inst[19:15]==5'b00000) ? 0 : 1;
148     rf_re1= (inst[24:20]==5'b00000) ? 0 : 1;
149     mem_we=1;
150     case (inst[14:12])
151         3'b000: store_type=1;    //sb
152         3'b001: store_type=2;    //sh
153         3'b010: store_type=3;    //sw
154     endcase
155 end
156 endcase
157 end
158 endmodule

```

Load模块

1b 1h 指令使用符号位扩展，1bu 1hu 指令使用 0 扩展

```

1  module Load (
2      input [31:0] dm_dout,
3      input [2:0] load_type,
4      output reg [31:0] load_data
5  );
6      always @(*) begin
7          case (load_type)
8              3'h1: load_data={{24{dm_dout[31]}}, dm_dout[31:24]};    //1b
9              3'h2: load_data={{16{dm_dout[31]}}, dm_dout[31:16]};    //1h
10             3'h3: load_data=dm_dout;    //1w
11             3'h4: load_data={{24'h0, dm_dout[31:24]};    //1bu
12             3'h5: load_data={{16'h0, dm_dout[31:16]};    //1hu
13             default: load_data=32'h0;
14         endcase
15     end
16 endmodule

```

Store模块

遇到 sb sh 指令时，同步将数据存储器中对应位置的数据读取出来，通过位拼接，将要写入的值和原值拼接，然后写入数据存储器中，如此不需要改变数据存储器的结构

```

1  module Store (
2      input [31:0] dm_din,
3      input [31:0] dm_dout,
4      input [1:0] store_type,
5      output reg [31:0] store_data
6  );
7      always @(*) begin
8          case (store_type)

```

```

9         2'h1: store_data={dm_din[7:0], dm_dout[23:0]}; //sb
10        2'h2: store_data={dm_din[15:0], dm_dout[15:0]}; //sh
11        2'h3: store_data=dm_din; //sw
12        default: store_data=0;
13    endcase
14 end
15 endmodule

```

其他指令大多只需要改变ALU模块模式信号，或Branch模块模式信号等等，不再赘述

2bits 感知机局部历史分支预测

接入流水线的模块为 `Partical_Br_History`。在该模块中，`BHT` 模块通过 `pc_if` 的哈希，找到对应的局部历史寄存器；`PHT_Partical` 模块通过刚刚取得的局部历史，找到对应的 2bits 寄存器（感知机），读取此时的感知机状态，并根据此输出是否跳转的预测信号

除此之外，当 `ex` 阶段的指令为分支指令时，`BHT` 模块通过 `pc_ex` 的哈希，找到对应的局部历史寄存器；`PHT_Partical` 模块通过刚刚取得的局部历史，找到对应的 2bits 寄存器（感知机），并根据 `ex` 阶段计算出的 `br` 信号修改感知机状态值。

当 `if` `ex` 阶段的指令为同一条分支指令时，通过写优先读出 `ex` 阶段写入的值

```

1  module BHT (
2      input clk,
3      input [31:0] pc_if,
4      input [31:0] pc_ex,
5      input br_inst, //ex阶段是否是br指令
6      input br,
7      output [3:0] br_history_if,
8      output [3:0] br_history_ex
9  );
10     reg [3:0] bht [0:63];
11     wire [5:0] pc_if_hash, pc_ex_hash;
12     integer i=0;
13
14     assign pc_if_hash=pc_if[15:10]^pc_if[7:2];
15     assign pc_ex_hash=pc_ex[15:10]^pc_ex[7:2];
16     assign br_history_if= (br_inst && pc_if==pc_ex) ? {bht[pc_ex_hash]
17 [2:0],br} : bht[pc_if_hash]; //写优先
18     assign br_history_ex= (br_inst) ? {bht[pc_ex_hash][2:0],br} :
19 bht[pc_ex_hash]; //写优先
20
21     initial begin
22         for (i=0; i<64; i=i+1) begin
23             bht[i]=4'h0; //所有BHR初值为零
24         end
25     end
26
27     always @(posedge clk) begin
28         if(br_inst)
29             bht[pc_ex_hash]<={bht[pc_ex_hash][2:0],br}; //移位寄存器
30     end
31 endmodule
32
33 module PHT_Partical (

```

```

32     input clk,
33     input [31:0] pc_if,
34     input [31:0] pc_ex,
35     input br_inst, //是否是br指令
36     input br,
37     input [3:0] br_history_if,
38     input [3:0] br_history_ex,
39     output reg br_pre
40 );
41     reg [1:0] pht [0:63][0:15];
42     wire [5:0] pc_if_hash, pc_ex_hash;
43     integer i=0, j=0;
44
45     assign pc_if_hash=pc_if[15:10]^pc_if[7:2];
46     assign pc_ex_hash=pc_ex[15:10]^pc_ex[7:2];
47
48     always @(*) begin
49         if(br_inst && pc_if==pc_ex)begin //写优先
50             case (pht[pc_ex_hash][br_history_ex])
51                 2'b00: br_pre=0;
52                 2'b01: br_pre= (br) ? 1 : 0;
53                 2'b10: br_pre= (br) ? 1 : 0;
54                 2'b11: br_pre=1;
55             endcase
56         end
57         else begin
58             case (pht[pc_if_hash][br_history_if])
59                 2'b00: br_pre=0;
60                 2'b01: br_pre=0;
61                 2'b10: br_pre=1;
62                 2'b11: br_pre=1;
63             endcase
64         end
65     end
66
67     initial begin
68         for (i=0; i<64; i=i+1) begin
69             for (j=0; j<16; j=j+1) begin
70                 pht[i][j]=2'b01; //弱不跳转
71             end
72         end
73     end
74
75     always @(posedge clk) begin
76         if (br_inst) begin
77             case (pht[pc_ex_hash][br_history_ex])
78                 2'b00: pht[pc_ex_hash][br_history_ex]<= (br) ? 2'b01 :
2'b00;
79                 2'b01: pht[pc_ex_hash][br_history_ex]<= (br) ? 2'b10 :
2'b00;
80                 2'b10: pht[pc_ex_hash][br_history_ex]<= (br) ? 2'b11 :
2'b01;
81                 2'b11: pht[pc_ex_hash][br_history_ex]<= (br) ? 2'b11 :
2'b10;
82             endcase

```

```

83         end
84     end
85 endmodule
86
87 module Partical_Br_History (
88     input clk,
89     input [31:0] pc_if,
90     input [31:0] pc_ex,
91     input [31:0] inst_ex,
92     input br_ex,
93     output br_pre
94 );
95     wire br_inst;
96     wire [3:0] br_history_if;
97     wire [3:0] br_history_ex;
98     assign br_inst= (inst_ex[6:0]==7'b1100011) ? 1 : 0;
99
100     BHT bht(
101         .clk(clk),
102         .pc_if(pc_if),
103         .pc_ex(pc_ex),
104         .br_inst(br_inst),
105         .br(br_ex),
106         .br_history_if(br_history_if),
107         .br_history_ex(br_history_ex)
108     );
109
110     PHT_Partical pht_partical(
111         .clk(clk),
112         .pc_if(pc_if),
113         .pc_ex(pc_ex),
114         .br_inst(br_inst),
115         .br(br_ex),
116         .br_history_if(br_history_if),
117         .br_history_ex(br_history_ex),
118         .br_pre(br_pre)
119     );
120 endmodule

```

2bits 感知机全局历史分支预测

与局部历史分支预测原理类似。接入流水线的模块为 `Global_Br_History`。在该模块中，`GHR` 模块存储当前的全局分支指令跳转历史；`PHT_Global` 模块通过刚刚取得的全局历史，找到对应的 2bits 寄存器（感知机），读取此时的感知机状态，并根据此输出是否跳转的预测信号

除此之外，当 `ex` 阶段的指令为分支指令时，`GHR` 模块修改当前的全局分支指令跳转历史；`PHT_Global` 模块通过刚刚取得的全局历史，找到对应的 2bits 寄存器（感知机），并根据 `ex` 阶段计算出的 `br` 信号修改感知机状态值。

当 `if` `ex` 阶段的指令为同一条分支指令时，通过写优先读出 `ex` 阶段写入的值

```

1 module GHR (
2     input clk,
3     input br_inst, //ex阶段是否是br指令
4     input br,

```



```

5     output [7:0] br_history
6 );
7     reg [7:0] ghr;
8
9     assign br_history= (br_inst) ? {ghr[6:0],br} : ghr; //写优先
10
11    initial begin
12        ghr=8'h0;
13    end
14
15    always @(posedge clk) begin
16        if(br_inst)
17            ghr<={ghr[6:0],br}; //移位寄存器
18    end
19 endmodule
20
21 module PHT_Global (
22     input clk,
23     input br_inst, //是否是br指令
24     input br,
25     input [7:0] br_history,
26     output reg br_pre
27 );
28     reg [1:0] pht [0:255];
29     integer i=0;
30
31     always @(*) begin
32         if(br_inst)begin //写优先
33             case (pht[br_history])
34                 2'b00: br_pre=0;
35                 2'b01: br_pre= (br) ? 1 : 0;
36                 2'b10: br_pre= (br) ? 1 : 0;
37                 2'b11: br_pre=1;
38             endcase
39         end
40         else begin
41             case (pht[br_history])
42                 2'b00: br_pre=0;
43                 2'b01: br_pre=0;
44                 2'b10: br_pre=1;
45                 2'b11: br_pre=1;
46             endcase
47         end
48     end
49
50    initial begin
51        for (i=0; i<256; i=i+1) begin
52            pht[i]=2'b01; //弱不跳转
53        end
54    end
55
56    always @(posedge clk) begin
57        if (br_inst) begin
58            case (pht[br_history])
59                2'b00: pht[br_history]<= (br) ? 2'b01 : 2'b00; //强不跳转

```

```

60         2'b01: pht[br_history] <= (br) ? 2'b10 : 2'b00; //弱不跳转
61         2'b10: pht[br_history] <= (br) ? 2'b11 : 2'b01; //弱跳转
62         2'b11: pht[br_history] <= (br) ? 2'b11 : 2'b10; //强跳转
63     endcase
64 end
65 end
66 endmodule
67
68 module Global_Br_History (
69     input clk,
70     input [31:0] inst_ex,
71     input br_ex,
72     output br_pre
73 );
74     wire br_inst;
75     wire [7:0] br_history;
76     assign br_inst = (inst_ex[6:0] == 7'b1100011) ? 1 : 0;
77
78     GHR ghr(
79         .clk(clk),
80         .br_inst(br_inst),
81         .br(br_ex),
82         .br_history(br_history)
83     );
84
85     PHT_Global pht_global(
86         .clk(clk),
87         .br_inst(br_inst),
88         .br(br_ex),
89         .br_history(br_history),
90         .br_pre(br_pre)
91     );
92 endmodule

```

全局历史预测与局部历史预测的竞争

在该模块中，根据 `pc_if` 的哈希找到对应的 2bits 寄存器（感知机），根据其中的值，决定选择全局历史预测还是局部历史预测

除此之外，当 `ex` 阶段的指令为分支指令时，该模块根据 `pc_ex` 的哈希找到对应的 2bits 寄存器（感知机），并根据 `ex` 阶段计算出的 `br` 信号修改感知机状态值。当全局预测正确时，修改 2bits 寄存器向全局方向转化；当全局预测错误时，修改 2bits 寄存器向局部方向转化

```

1  module GP_Competition (
2      input clk,
3      input [31:0] pc_if,
4      input [31:0] pc_ex,
5      input [31:0] inst_ex,
6      input [31:0] inst_if,
7      input br_pre_global_if,
8      input br_pre_partical_if,
9      input br_pre_global_ex,
10     input br_pre_partical_ex,
11     input br,
12     output reg gp_sel, //使用全局(1)/分支(0)历史预测

```

```

13     output br_pre
14 );
15     wire br_inst, gp_sel_ex;
16     wire [5:0] pc_if_hash, pc_ex_hash;
17     reg [1:0] gpht[0:63];
18     integer i=0;
19
20     assign br_inst= (inst_ex[6:0]==7'b1100011) ? 1 : 0;
21     assign pc_if_hash=pc_if[15:10]^pc_if[7:2];
22     assign pc_ex_hash=pc_ex[15:10]^pc_ex[7:2];
23     assign gp_sel_ex= (br==br_pre_global_ex) ? 1 : 0;    //全局历史预测优先，全局
    正确时认为是全局
24     assign br_pre= (inst_if[6:0]!=7'b1100011) ? 0 :
25         (gp_sel) ? br_pre_global_if : br_pre_partical_if;
26
27     always @(*) begin
28         if(br_inst && pc_if==pc_ex)begin    //写优先
29             case (gpht[pc_ex_hash])
30                 2'b00: gp_sel=0;
31                 2'b01: gp_sel= (gp_sel_ex) ? 1 : 0;
32                 2'b10: gp_sel= (gp_sel_ex) ? 1 : 0;
33                 2'b11: gp_sel=1;
34             endcase
35         end
36         case (gpht[pc_if_hash])
37             2'b00: gp_sel=0;
38             2'b01: gp_sel=0;
39             2'b10: gp_sel=1;
40             2'b11: gp_sel=1;
41         endcase
42     end
43
44     initial begin
45         for (i=0; i<256; i=i+1) begin
46             gpht[i]=2'h0;    //所有GBHR初值为零
47         end
48     end
49
50     always @(posedge clk) begin
51         if(br_inst)
52             case (gpht[pc_ex_hash])
53                 2'b00: gpht[pc_ex_hash]<= (gp_sel_ex) ? 2'b01 : 2'b00; //强局
    部
54                 2'b01: gpht[pc_ex_hash]<= (gp_sel_ex) ? 2'b10 : 2'b00; //弱局
    部
55                 2'b10: gpht[pc_ex_hash]<= (gp_sel_ex) ? 2'b11 : 2'b01; //弱全
    局
56                 2'b11: gpht[pc_ex_hash]<= (gp_sel_ex) ? 2'b11 : 2'b10; //强全
    局
57             endcase
58         end
59
60 endmodule

```

跳转地址预测

在该模块中，根据 `pc_if` 的哈希找到对应的 `jat` 寄存器，若对应的 `tag` 寄存器中的值当前 `pc_if` 的[5:2]位相等时，认为命中，使用对应 `jat` 寄存器中的值为预测的地址

除此之外，当 `ex` 阶段的指令为分支指令或跳转指令时，该模块根据 `pc_ex` 的哈希找到对应的 `jat` 寄存器，并根据 `ex` 阶段计算出的 `alu_ans` 信号修改对应的 `jat` 寄存器的值和对应的 `tag` 寄存器的值

```
1  module BTB (
2      input clk,
3      input [31:0] pc_if,
4      input [31:0] pc_ex,
5      input [1:0] pc_sel,
6      input [31:0] alu_ans,
7      output reg hit,
8      output reg [31:0] jump_addr
9  );
10     wire [5:0] pc_if_hash, pc_ex_hash;
11     wire [3:0] tag_if, tag_ex;
12     reg [3:0] tag [0:63];
13     reg [31:0] jat [0:63]; //jump addr table
14     integer i=0;
15
16     assign tag_if=pc_if[5:2];
17     assign tag_ex=pc_ex[5:2];
18     assign pc_if_hash=pc_if[15:10]^pc_if[7:2];
19     assign pc_ex_hash=pc_ex[15:10]^pc_ex[7:2];
20
21     initial begin
22         for(i=0; i<64; i=i+1)begin
23             tag[i]=4'h0;
24             jat[i]=32'h0;
25         end
26     end
27
28     always @(*) begin
29         if(pc_if==pc_ex)begin
30             hit=1;
31             jump_addr=alu_ans;
32         end
33         else begin
34             if(tag_if==tag[pc_if_hash])begin
35                 hit=1;
36                 jump_addr=jat[pc_if_hash];
37             end
38             else begin
39                 hit=0;
40                 jump_addr=0;
41             end
42         end
43     end
44
45     always @(posedge clk) begin
46         if(pc_sel==2 || pc_sel==3)begin //jal, br
47             if(tag[pc_ex_hash]!=tag_ex)begin
```

```

48         jat[pc_ex_hash] <= alu_ans;
49         tag[pc_ex_hash] <= tag_ex;
50     end
51 end
52 end
53 endmodule

```

jalr 指令地址栈

当 if 阶段遇到 jal 指令时，存储当前 pc_if + 4 到栈中

当 if 阶段遇到 jalr 指令时，出栈，并将出栈的值作为预测的跳转地址

```

1  module Jalr_Stack (
2      input clk,
3      input rst,
4      input [31:0] pc_if,
5      input [31:0] inst_if,
6      input [1:0] top_fix,
7      output reg [31:0] jalr_addr
8  );
9      reg [31:0] jas[0:63]; //jalr addr stack
10     reg [5:0] top;
11
12     integer i=0;
13
14     initial begin
15         top=1;
16         for(i=0; i<64; i=i+1)begin
17             jas[i]=0;
18         end
19     end
20
21     always @(*) begin
22         jalr_addr=jas[top-1];
23     end
24
25     always @(posedge clk) begin
26         if(rst)begin
27             top<=1;
28         end
29         else begin
30             top<=top+top_fix;
31             if(inst_if[6:0]==7'b1100111)begin //jalr, 出栈
32                 top<=top+top_fix-1;
33             end
34             if(inst_if[6:0]==7'b1101111)begin //jal, 入栈
35                 jas[top]=pc_if+4;
36                 top<=top+top_fix+1;
37             end
38         end
39     end
40 endmodule

```

Hazard模块的修改

预测失败时，插入气泡，冲刷 id ex 阶段

若被冲刷的指令中包含 jal jalr 指令，则需要将被改变的 jalr 栈指针修改回去，修改的值即为

top_fix

```
1  if(pre_failure)begin
2      case (pc_sel_ex)
3          2'h0: begin //ex阶段不应该跳转，但跳转了，
4              top_fix=jalr_id+jalr_ex-jal_id-jal_ex;
5              flush_id=1;
6              flush_ex=1;
7          end
8          2'h1: begin //jalr预测失败，说明jalr不是用于函数调用的返回，为了安全清空jalr
栈
9              jalr_stack_rst=1;
10             flush_id=1;
11             flush_ex=1;
12         end
13         2'h2: begin //br
14             top_fix=jalr_id+jalr_ex-jal_id-jal_ex;
15             flush_id=1;
16             flush_ex=1;
17         end
18         2'h3: begin//jal
19             top_fix=jalr_id+jalr_ex;
20             flush_id=1;
21             flush_ex=1;
22         end
23     endcase
24 end
```

NPC_SEL模块的修改

根据 ex 阶段的结果来分类，同时统计分支跳转指令的次数和预测错误次数

```
1  module NPC_SEL (
2      input clk,
3      input [31:0] pc_add4_if,
4      input [31:0] pc_jalr_ex,
5      input [31:0] alu_ans_ex,
6      input [31:0] jump_addr,
7      input [31:0] jalr_addr,
8      input [1:0] pc_sel_ex,
9      input [31:0] pc_cur_id,
10     input [31:0] pc_add4_ex,
11     input [2:0] br_type_ex,
12     input br_pre,
13     input hit,
14     input [31:0] inst_if,
15     output reg [31:0] pc_next,
16     output reg pre_failure, //当预测失败时为1
17     output reg [31:0] br_num,
18     output reg [31:0] jal_num,
```

```

19     output reg [31:0] jalr_num,
20     output reg [31:0] br_fail_num, //预测失败的次数
21     output reg [31:0] jal_fail_num,
22     output reg [31:0] jalr_fail_num
23 );
24     initial begin
25         br_num=0; jal_num=0; jalr_num=0;
26         br_fail_num=0; jal_fail_num=0; jalr_fail_num=0;
27     end
28
29     always @(*) begin
30         pc_next=pc_add4_if;
31         pre_failure=0;
32         case (pc_sel_ex)
33             2'h0: begin //ex阶段是不应该跳转，根据if阶段预测pc_next
34                 if(pc_cur_id!=pc_add4_ex && br_type_ex!=0)begin //ex阶段是br
指令，且本不应该跳转
35                     pre_failure=1; //预测失败，插入气泡
36                     pc_next=pc_add4_ex;
37                 end
38             else begin
39                 case (inst_if[6:0])
40                     7'b1101111: begin //jal
41                         if(hit)
42                             pc_next=jump_addr;
43                     end
44                     7'b1100111: begin //jalr
45                         pc_next=jalr_addr;
46                     end
47                     7'b1100011: begin //br
48                         if(br_pre)
49                             if(hit)
50                                 pc_next=jump_addr;
51                     end
52                 endcase
53             end
54
55         end
56         2'h1: begin //ex阶段为jalr指令，jalr指令"在99%的情况下"一定被正确预
测，但在助教的测试程序中不行
57             if(pc_cur_id!=pc_jalr_ex)begin
58                 pre_failure=1; //预测失败，插入气泡
59                 pc_next=pc_jalr_ex;
60             end
61         else begin
62             case (inst_if[6:0])
63                 7'b1101111: begin //jal
64                     if(hit)
65                         pc_next=jump_addr;
66                 end
67                 7'b1100111: begin //jalr
68                     pc_next=jalr_addr;
69                 end
70                 7'b1100011: begin //br
71                     if(br_pre)

```

```

72             if(hit)
73                 pc_next=jump_addr;
74             end
75         endcase
76     end
77 end
78 2'h2: begin //ex阶段为br指令，需要检测br指令ex阶段预测是否正确，若不正
确则pc_next=alu_ans_ex
79     if(pc_cur_id!=alu_ans_ex)begin
80         pre_failure=1; //预测失败，插入气泡
81         pc_next=alu_ans_ex;
82     end
83     else begin //ex阶段预测成功，根据if阶段预测pc_next
84         case (inst_if[6:0])
85             7'b1101111: begin //jal
86                 if(hit)
87                     pc_next=jump_addr;
88                 end
89                 7'b1100111: begin //jalr
90                     pc_next=jalr_addr;
91                 end
92                 7'b1100011: begin //br
93                     if(br_pre)
94                         if(hit)
95                             pc_next=jump_addr;
96                         end
97                     endcase
98                 end
99             end
100 2'h3: begin //ex阶段为jal指令，需要检测jal指令ex阶段预测是否正确，若不
正确则pc_next=pc_jalr_ex
101     if(pc_cur_id!=pc_jalr_ex)begin
102         pre_failure=1; //预测失败，插入气泡
103         pc_next=pc_jalr_ex;
104     end
105     else begin //ex阶段预测成功，根据if阶段预测pc_next
106         case (inst_if[6:0])
107             7'b1101111: begin //jal
108                 if(hit)
109                     pc_next=jump_addr;
110                 end
111                 7'b1100111: begin //jalr
112                     pc_next=jalr_addr;
113                 end
114                 7'b1100011: begin //br
115                     if(br_pre)
116                         if(hit)
117                             pc_next=jump_addr;
118                         end
119                     endcase
120                 end
121             end
122         endcase
123     end
124

```



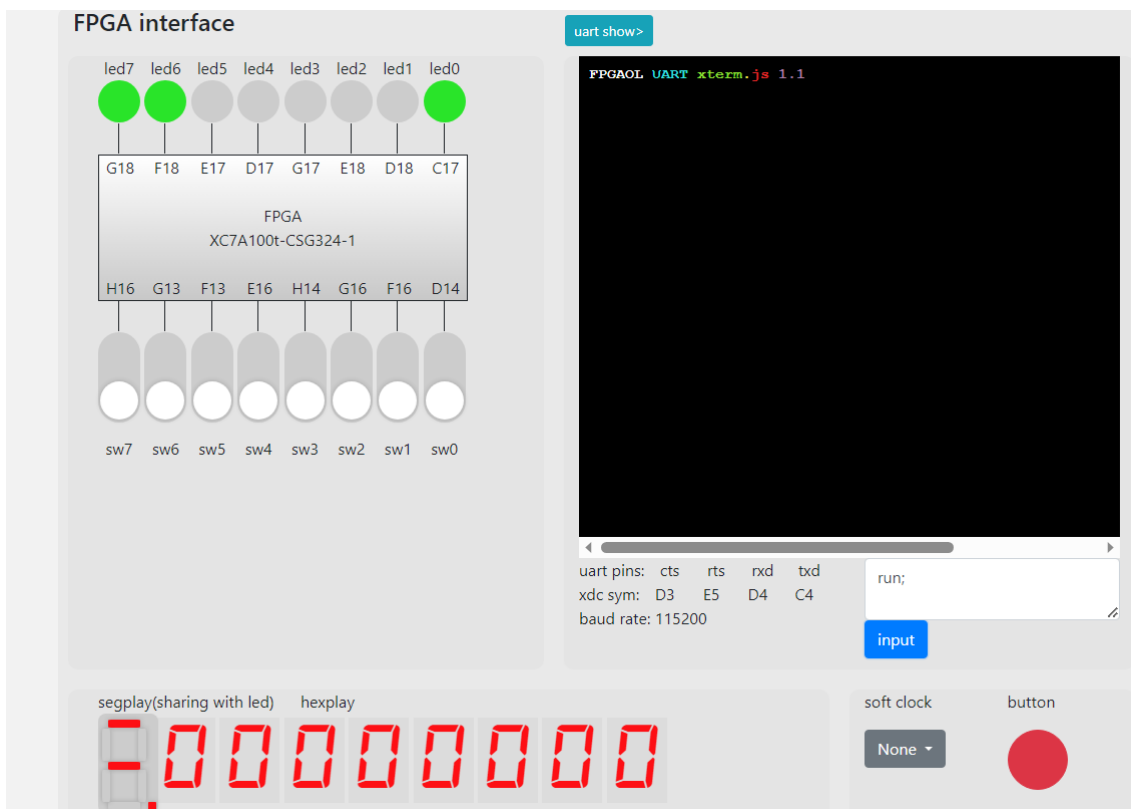
```

125     always @(posedge clk) begin
126         case (pc_sel_ex)
127             2'h0: begin //ex阶段是不应该跳转，根据if阶段预测pc_next
128                 if(pc_cur_id!=pc_add4_ex && br_type_ex!=0)begin //ex阶段是br
指令，且本不应该跳转
129                     br_num<=br_num+1;
130                     br_fail_num<=br_fail_num+1;
131                 end
132                 else if(br_type_ex!=0)
133                     br_num<=br_num+1;
134                 end
135             2'h1: begin //ex阶段为jalr指令，jalr指令"在99%的情况下"一定被正确预
测，但在助教的全指令测试程序中不行
136                 if(pc_cur_id!=pc_jalr_ex)begin
137                     jalr_num<=jalr_num+1;
138                     jalr_fail_num<=jalr_fail_num+1;
139                 end
140                 else
141                     jalr_num<=jalr_num+1;
142                 end
143             2'h2: begin //ex阶段为br指令，需要检测br指令ex阶段预测是否正确，若不正
确则pc_next=alu_ans_ex
144                 if(pc_cur_id!=alu_ans_ex)begin
145                     br_num<=br_num+1;
146                     br_fail_num<=br_fail_num+1;
147                 end
148                 else
149                     br_num<=br_num+1;
150                 end
151             2'h3: begin //ex阶段为jal指令，需要检测jal指令ex阶段预测是否正确，若不
正确则pc_next=pc_jalr_ex
152                 if(pc_cur_id!=pc_jalr_ex)begin
153                     jal_num<=jal_num+1;
154                     jal_fail_num<=jal_fail_num+1;
155                 end
156                 else
157                     jal_num<=jal_num+1;
158                 end
159             endcase
160         end
161     endmodule

```

实验结果

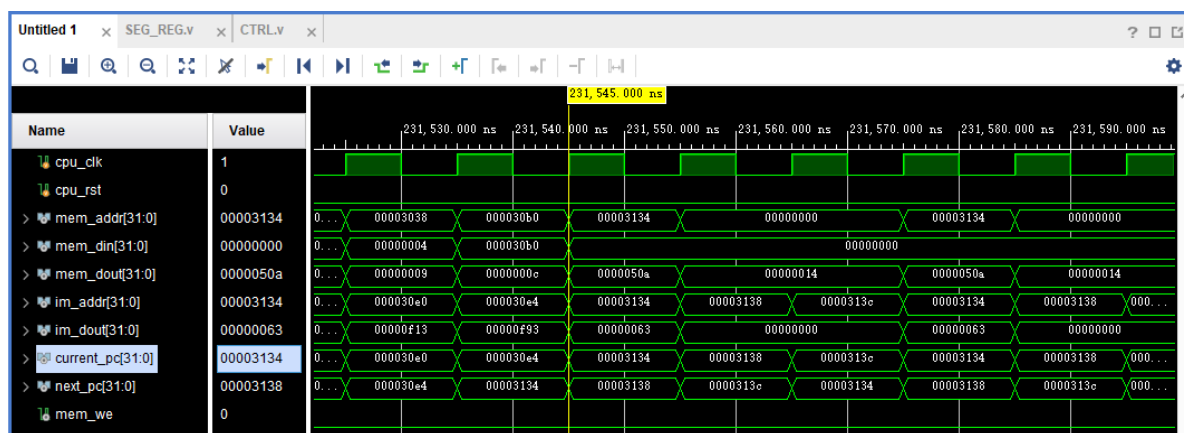
全指令测试



矩阵乘法（两个矩阵大小分别为4*5，5*6，其中数值按顺序为1~20和30~1）

无分支预测的流水线处理器（Lab5）

周期数：23154



仅局部历史分支预测的流水线处理器

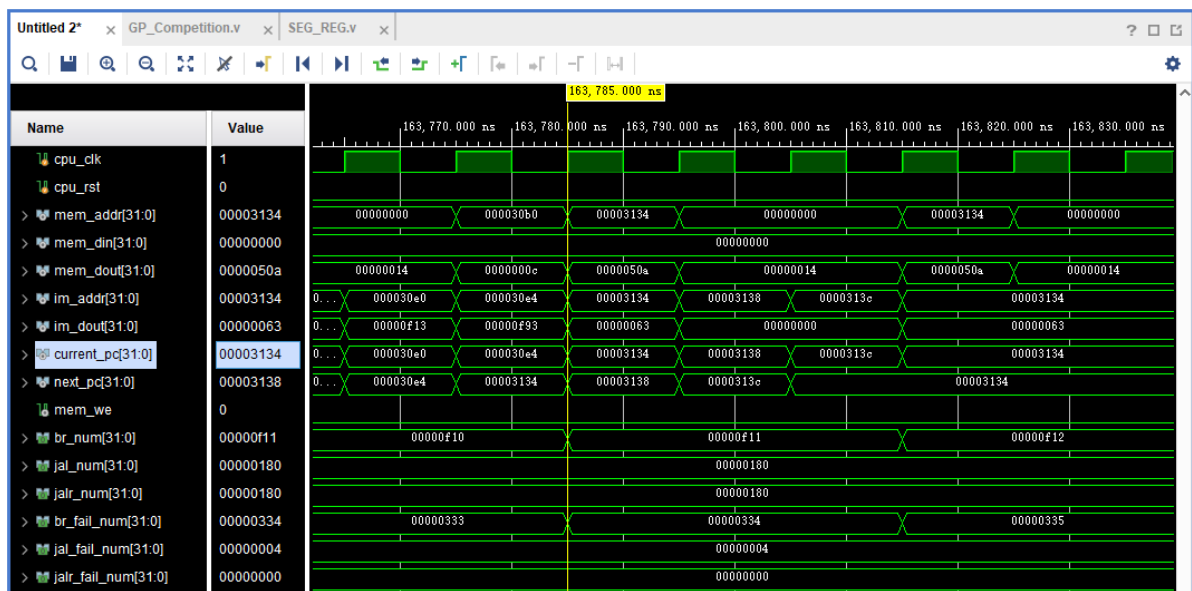
周期数：16378

分支指令数：3'bf10 (3856)

分支指令预测失败数：3'b333 (819)

分支预测成功率：78.76%

比无分支预测的处理器快：41.37%



仅全局历史分支预测的流水线处理器

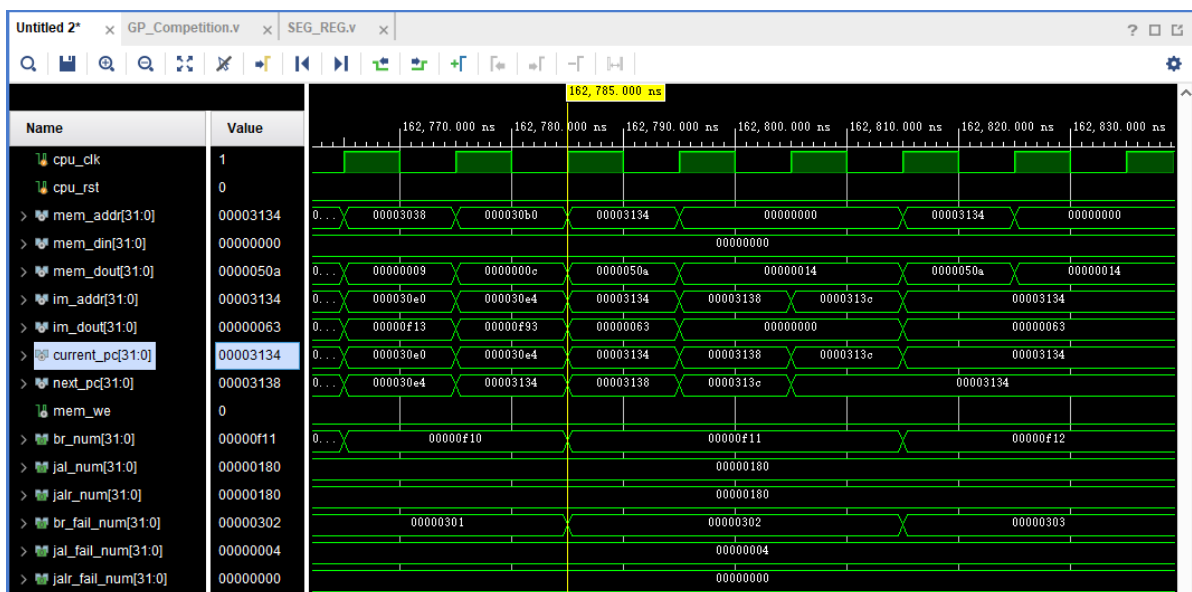
周期数：16278

分支指令数：3'bf10 (3856)

分支指令预测失败数：3'b302 (770)

分支预测成功率：80.03%

比无分支预测的处理器快：42.24%



包含全局历史、局部历史分支预测的流水线处理器

周期数：15636

分支指令数：3'bf10 (3856)

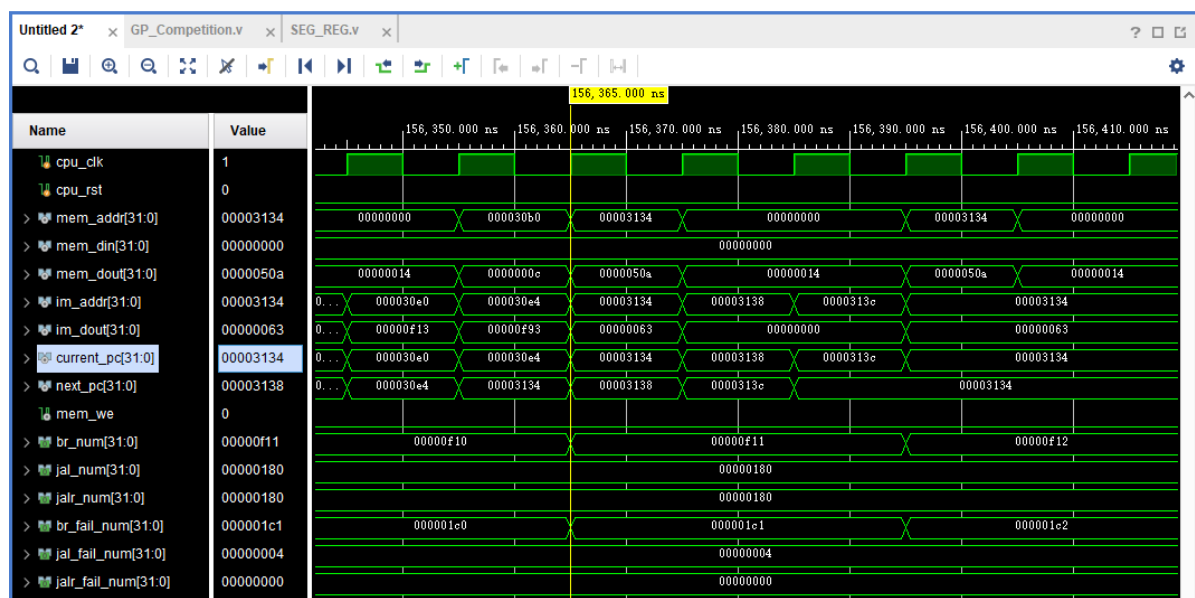
分支指令预测失败数：3'b1c0 (448)

分支预测成功率：88.38%

比无分支预测的处理器快：48.08%

比仅局部历史分支预测的处理器快：4.745%

比仅全局历史分支预测的处理器快：4.106%



体会

分支预测的各种冲突非常多，现代处理器这么快真是人类科技的结晶。期待早日超到10GHz