LAB 6 report

实验原理

实验内容

- 实现所有非系统 RV32-I 指令
- 实现2bits 感知机局部和全局历史分支预测，正确处理竞争，并接入流水线

模块设计

补全指令集

补全指令集主要修改的部分有MEM模块、alu模块、control模块等

MEM

首先是由于指令数变多，所以地址变为[10:2]，由于store和load有不同长度，对load指令的操作就是，以lb为例，读出一个字之后依据读的地址，取不同地方的八位，然后进行符号扩展即可，然后lbu也同理，只是最后进行无符号扩展。对store指令，也是以sb为例，由于我有要存的数据的地址，我就先把这个地址的一个字给读出来，然后将要store的byte的内容和读出来的内容拼接一下，相当于把这个byte的内容给替换掉，然后再把这个字给存进去，这样就实现了不同长度的访存

```
    wire [31:0] im_addr_rom;
    assign im_addr_rom = (im_addr - 32'h00003000);
    // TODO
    inst_memory inst_memory(.a(im_addr_rom[10:2]), .spo(im_dout));


    wire [2:0]func;
    assign func = inst[14:12];
    wire mem_read;
    assign mem_read = (inst[6:0] == 7'h03) ? 1 : 0;

    reg [31:0] dm_actual_din;
    wire [31:0] dm_dout;

    always @(*) begin       //store
        if(dm_we) begin
            case(func)
                3'b000:         //byte
                    case(dm_addr[1:0])
                        2'b00:  dm_actual_din = {dm_dout[31:8], dm_din[7:0]};
//低八位
                        2'b01:  dm_actual_din = {dm_dout[31:16], dm_din[7:0],
dm_dout[7:0]};
                        2'b10:  dm_actual_din = {dm_dout[31:24], dm_din[7:0],
dm_dout[15:0]};
                        default:dm_actual_din = {dm_din[7:0], dm_dout[23:0]};
                    endcase
                3'b001:         //half word
                    case(dm_addr[1])
```

```
                        1'b0:   dm_actual_din = {dm_dout[31:16], dm_din[15:0]};
  //低十六位
                        default:dm_actual_din = {dm_din[15:0], dm_dout[15:0]};
                    endcase
                3'b010:     dm_actual_din = dm_din;         //sw
                default:    dm_actual_din = dm_dout;
            endcase
        end
        else
            dm_actual_din = dm_dout;
    end

    always @(*) begin       //load
        if(mem_read) begin
            case(func)
                3'b000:     //byte
                    case(dm_addr[1:0])
                        2'b00:  dm_actual_dout = {{24{dm_dout[7]}},
dm_dout[7:0]};
                        2'b01:  dm_actual_dout = {{24{dm_dout[15]}},
dm_dout[15:8]};
                        2'b10:  dm_actual_dout = {{24{dm_dout[23]}},
dm_dout[23:16]};
                        default:dm_actual_dout = {{24{dm_dout[31]}},
dm_dout[31:24]};
                    endcase
                3'b001:     //half word
                    case(dm_addr[1])
                        1'b0:   dm_actual_dout = {{16{dm_dout[15]}},
dm_dout[15:0]};
                        default:dm_actual_dout = {{16{dm_dout[31]}},
dm_dout[31:16]};
                    endcase
                3'b010:     dm_actual_dout = dm_dout;
                3'b100:
                    case(dm_addr[1:0])
                        2'b00:  dm_actual_dout = {24'b0, dm_dout[7:0]};
                        2'b01:  dm_actual_dout = {24'b0, dm_dout[15:8]};
                        2'b10:  dm_actual_dout = {24'b0, dm_dout[23:16]};
                        default:dm_actual_dout = {24'b0, dm_dout[31:24]};
                    endcase
                3'b101:
                    case(dm_addr[1])
                        1'b0:   dm_actual_dout = {16'b0, dm_dout[15:0]};
                        default:dm_actual_dout = {16'b0, dm_dout[31:16]};
                    endcase
                default:    dm_actual_dout = dm_dout;

            endcase
        end
        else
            dm_actual_dout = dm_dout;
    end
```

```
    data_memory data_memory(.a(dm_addr[9:2]), .d(dm_actual_din),
.dpra(mem_check_addr[7:0]), .spo(dm_dout), .dpo(mem_check_data), .we(dm_we),
.clk(clk));
endmodule
```

control

只需要修改br_type和alu_ctrl，根据不同指令来判断即可

```
    always @(*) begin
        if(inst[6:0] == 7'h63) begin
            case(inst[14:12])
                3'b000:    br_type = 3'b001;        //beq
                3'b100:    br_type = 3'b010;        //blt
                3'b001:    br_type = 3'b011;        //bne
                3'b101:    br_type = 3'b100;        //bge
                3'b110:    br_type = 3'b101;        //bltu
                3'b111:    br_type = 3'b110;        //bgeu
                default:   br_type = 3'b000;
            endcase
        end
        else
            br_type = 0;

    end


    always @(*) begin
        if(((inst[6:0] == 7'h13) && (inst[14:12] == 3'b000)) || ((inst[6:0] ==
7'h33) && (inst[31:25] == 0) && (inst[14:12] == 3'b000)) || (inst[6:0] == 7'h03)
|| (inst[6:0] == 7'h23)
        || (inst[6:0] == 7'h67) || (inst[6:0] == 7'h6f) || (inst[6:0] == 7'h37)
|| (inst[6:0] == 7'h17) || (inst[6:0] == 7'h63))    //all     +
            alu_ctrl = 4'b0000;
        else if((inst[6:0] == 7'h33) && (inst[14:12] == 3'b000) && (inst[31:25]
== 7'h20)) //sub
            alu_ctrl = 4'b0001;
        else if((inst[6:0] == 7'h33) && (inst[14:12] == 3'b001)) //sll
            alu_ctrl = 4'b1001;
        else if((inst[6:0] == 7'h33) && (inst[14:12] == 3'b111)) //and
            alu_ctrl = 4'b0101;
        else if((inst[6:0] == 7'h13) && (inst[14:12] == 3'b111)) //andi
            alu_ctrl = 4'b0101;
        else if((inst[6:0] == 7'h13) && (inst[14:12] == 3'b001)) //slli
            alu_ctrl = 4'b1001;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b010)) || ((inst[6:0]
== 7'h13) && (inst[14:12] == 3'b010))) //slt, slti
            alu_ctrl = 4'b0100;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b011)) || ((inst[6:0]
== 7'h13) && (inst[14:12] == 3'b011))) //sltu, sltiu
            alu_ctrl = 4'b0011;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b100)) || ((inst[6:0]
== 7'h13) && (inst[14:12] == 3'b100))) //xor, xori
```

```
            alu_ctrl = 4'b0111;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b101) && (inst[31:25]
== 7'b0)) || ((inst[6:0] == 7'h13) && (inst[14:12] == 3'b101) && (inst[31:25] ==
7'b0))) //srl, srli
            alu_ctrl = 4'b1000;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b101) && (inst[31:25]
== 7'b0100000)) || ((inst[6:0] == 7'h13) && (inst[14:12] == 3'b101) &&
(inst[31:25] == 7'b0100000))) //sra, srai
            alu_ctrl = 4'b1010;
        else if(((inst[6:0] == 7'h33) && (inst[14:12] == 3'b110)) || ((inst[6:0]
== 7'h13) && (inst[14:12] == 3'b110))) //or, ori
            alu_ctrl = 4'b0110;
        else
            alu_ctrl = 4'b1111;
end
endmodule
```

branch

根据新增的br type来选择是否跳转

```
module Branch(
    input [31:0] ra0,
    input [31:0] ra1,
    input [2:0] br_type,
    output reg br,

    input forecast_win
);
        always @(*) begin
                case(br_type)
                    3'b001:  br = (ra0 == ra1) ? 1 : 0;     //beq
                    3'b010:  br = ($signed(ra0) < $signed(ra1)) ? 1 : 0;
 //blt
                    3'b011:  br = (ra0 != ra1) ? 1 : 0;       //bne
                    3'b100:  br = ($signed(ra0) >= $signed(ra1)) ? 1 : 0;
    //bge
                    3'b101:  br = (ra0 < ra1) ? 1 : 0;      //bltu
                    3'b110:  br = (ra0 >= ra1) ? 1 : 0;     //bgeu
                    default: br = 0;
                endcase
        end
endmodule
```

分支预测

cache

首先需要建立一个16个块的cache，在取到要跳转的pc时直接读出要跳转的地址，并且将他赋给nextpc。在读取跳转指令时将we信号传给cache，如果没找到对应的tag，就把新的地址和跳转的地址存到cache中，然后对应valid赋1。当传入的pc有对应的tag并且局部预测和全球预测竞争的结果是要跳转时，然后就传出信号，把cache里的data赋给nextpc的

```
    reg [15:0] eq,eqw;
    reg[3:0] cnt = 15;

    wire w;
    assign w=we_if & ~(|(eqw & valid));        //when valid and the address is
already there, do not write
    always @(posedge clk) begin
        if(w)
            if(cnt == 15)
                cnt = 0;
            else
                cnt = cnt + 1;
        else
            ;
    end
    always @(posedge clk) begin
        if(w)
        begin
            valid[cnt] <= 1;
            tag[cnt] <= memAddr[31:2];
            cacheData[cnt] <= memData;
        end
    end

    always @(*) begin
        for(i=0; i<16; i=i+1)
        begin
            eq[i]=(address[31:2]==tag[i]);
            eqw[i]=(memAddr[31:2]==tag[i]);
        end
    end
```

局部预测BHT与PHT

对于BHT，我的处理是每个存入cache的pc都有一个对应的2bit移位寄存器，也就是BHR，初始值都设为00，当这条指令存入cache后，如果下一次对应的pc hit，并且跳转，就把1从右边存入寄存器，00变为01，否则就把0从右存入寄存器。然后这个2bit寄存器有四个值，所以对应四个2bit counter，也就是跳转计数器，这些计数器初始化都为01，弱不跳转，然后这条指令如果hit且跳转，就更新BHR和BHR的值对应的2bit counter，这个2bit counter就+1，否则就-1

然后还有一个地方就是在if阶段判断是否hit，在ex阶段才能知道这条指令是否真的跳转，也就是在ex阶段才知道预测是否成功，所以要把hit的结果用两个寄存器延迟到ex阶段，然后根据是否预测成功来更新BHR和PHT

还有一个处理是，由于一个pc对应一个BHR，一个BHR对应四个counter，所以counter有16(cache块的个数，也就是BHR的个数)* 4(一个BHR对应四个counter) * 2(counter的位宽) 相当于一个三维数组，三维数组在verilog里面我不确定能不能表示，所以我就当成一个二维数组来处理了，就是当成一个16*8的二维数组，把4 * 2 合并起来，然后对这个位宽为8的寄存器两位两位来操作，下面的case就是处理这种情况的

```
    wire [15:0] hitn;
    assign hitn = eq & valid;

    wire [3:0] sel;
    reg [3:0] sel_ex, sel_id;
    E16_4encoder E16_4encoder(.a(hitn), .b(sel));   //16_4编码器

    always @(posedge clk) begin
        sel_ex <= sel_id;
        sel_id <= sel;
    end

    reg [15:0] hitn_id, hitn_ex;

    always @(posedge clk) begin
        hitn_ex <= hitn_id;
        hitn_id <= hitn;
    end
    //BHT

    always @(posedge clk) begin
        if(|hitn_ex) begin   // hit, but don't know branch or not
            if(inIfBranch)
                BHR[sel_ex] <= (BHR[sel_ex] << 1) + 1;
            else
                BHR[sel_ex] <= (BHR[sel_ex] << 1);
        end
        else
            ;
    end
    wire [7:0] counter_one_dimension;
    assign counter_one_dimension = counter[sel_ex];

    always @(posedge clk) begin //PHT
        case(BHR[sel_ex])
            2'b00: begin
                if(inIfBranch) begin
                    if(counter_one_dimension[1:0] == 2'b11)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] + 1'b1;
                end
                else begin
                    if(counter_one_dimension[1:0] == 2'b00)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] - 1'b1;
                end
            end
```

```
            2'b01:begin
                if(inIfBranch) begin
                    if(counter_one_dimension[3:2] == 2'b11)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] + 3'b100;
                end
                else begin
                    if(counter_one_dimension[3:2] == 2'b00)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] - 3'b100;
                end
            end

            2'b10:begin
                if(inIfBranch) begin
                    if(counter_one_dimension[5:4] == 2'b11)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] + 5'b10000;
                end
                else begin
                    if(counter_one_dimension[5:4] == 2'b00)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] - 5'b10000;
                end
            end

            2'b11:begin
                if(inIfBranch) begin
                    if(counter_one_dimension[7:6] == 2'b11)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] + 7'b1000000;
                end
                else begin
                    if(counter_one_dimension[7:6] == 2'b00)
                        counter[sel_ex] <= counter[sel_ex];
                    else
                        counter[sel_ex] <= counter[sel_ex] - 7'b1000000;
                end
            end
            default:
                ;

        endcase
    end
```

全局历史GHR和PHT

全局历史和局部历史非常相似，只是若干BHR组成的BHT表变成了一个GHR，GHR我设置成了一个4bit移位寄存器，记录着全局的跳转历史，若上一条跳转指令跳转成功则把1从右存到寄存器里面，否则存0。 4bit寄存器有16个值，所以对应了16个全局counter。counter的处理方法跟上面的局部历史处理方法一致，然后同样也需要把hit的信息用两个寄存器传到ex阶段

```
//GHR

    reg [3:0] GHR;
    initial GHR = 4'b0101;

    always @(posedge clk) begin
        if(|hitn_ex) begin
            if(inIfBranch)  //jump
                GHR <= {GHR[2:0], 1'b1};
            else
                GHR <= {GHR[2:0], 1'b0};

        end
        else
            ;
    end

//GHR_PHT
    //reg [31:0] GHR_counter[7:0];     // 2 bit counter * 16

    wire [31:0] GHR_counter_one_dimension;
    assign GHR_counter_one_dimension = GHR_counter[sel_ex];


    always @(posedge clk) begin //PHT
        case(GHR)
            4'b0000: begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[1:0] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] + 1'b1;
                end
                else begin
                    if(GHR_counter_one_dimension[1:0] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] - 1'b1;
                end
            end
            4'b0001:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[3:2] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] + 3'b100;
                end
```

```
                else begin
                    if(GHR_counter_one_dimension[3:2] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] - 3'b100;
                end
            end

            4'b0010:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[5:4] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] + 5'b10000;
                end
                else begin
                    if(GHR_counter_one_dimension[5:4] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] - 5'b10000;
                end
            end

            4'b0011:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[7:6] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] + 7'b1000000;
                end
                else begin
                    if(GHR_counter_one_dimension[7:6] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] - 7'b1000000;
                end
            end

            4'b0100:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[9:8] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
9'b100000000;
                end
                else begin
                    if(GHR_counter_one_dimension[9:8] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
9'b100000000;
                end
            end
```

```
            4'b0101:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[11:10] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
11'b10000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[11:10] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
11'b10000000000;
                end
            end

            4'b0110:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[13:12] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
13'b1000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[13:12] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
13'b1000000000000;
                end
            end

            4'b0111:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[15:14] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
15'b100000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[15:14] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
15'b100000000000000;
                end
            end

            4'b1000:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[17:16] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
```

```
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
17'b10000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[17:16] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
17'b10000000000000000;
                end
            end

            4'b1001:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[19:18] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
19'b1000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[19:18] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
19'b1000000000000000000;
                end
            end

            4'b1010:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[21:20] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
21'b100000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[21:20] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
21'b100000000000000000000;
                end
            end

            4'b1011:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[23:22] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
23'b10000000000000000000000;
                end
```

```
                else begin
                    if(GHR_counter_one_dimension[23:22] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
23'b10000000000000000000000;
                end
            end

            4'b1100:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[25:24] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
25'b1000000000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[25:24] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
25'b1000000000000000000000000;
                end
            end

            4'b1101:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[27:26] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
27'b100000000000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[27:26] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
27'b100000000000000000000000000;
                end
            end

            4'b1110:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[29:28] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
29'b10000000000000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[29:28] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
```

```
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
29'b10000000000000000000000000000;
                end
            end

            4'b1111:begin
                if(inIfBranch) begin
                    if(GHR_counter_one_dimension[31:30] == 2'b11)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] +
31'b100000000000000000000000000000;
                end
                else begin
                    if(GHR_counter_one_dimension[31:30] == 2'b00)
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex];
                    else
                        GHR_counter[sel_ex] <= GHR_counter[sel_ex] -
31'b100000000000000000000000000000;
                end
            end
            default:
                ;

        endcase
    end

//BHR
    wire taken;
    //assign taken = (counter[sel] == 2'b10) | (counter[sel] == 2'b11);
    wire [7:0]counter_sel;
    assign counter_sel = counter[sel];
    assign taken = ((BHR[sel] == 2'b00) & counter_sel[1]) | ((BHR[sel] == 2'b01)
& counter_sel[3])
                 | ((BHR[sel] == 2'b10) & counter_sel[5]) | ((BHR[sel] == 2'b11)
& counter_sel[7]);
    assign hit = (|hitn) & taken;
    assign data = cacheData[sel];


//    GHR

    wire GHR_taken;
    wire [31:0] GHR_counter_sel;
    assign GHR_counter_sel = GHR_counter[sel];
    assign GHR_taken = ((GHR == 4'b0000) & GHR_counter_sel[1]) | ((GHR ==
4'b0001) & GHR_counter_sel[3]) | ((GHR == 4'b0010) & GHR_counter_sel[5])
                     | ((GHR == 4'b0011) & GHR_counter_sel[7]) | ((GHR ==
4'b0100) & GHR_counter_sel[9]) | ((GHR == 4'b0101) & GHR_counter_sel[11])
                     | ((GHR == 4'b0110) & GHR_counter_sel[13]) | ((GHR ==
4'b0111) & GHR_counter_sel[15]) | ((GHR == 4'b1000) & GHR_counter_sel[17])
                     | ((GHR == 4'b1001) & GHR_counter_sel[19]) | ((GHR ==
4'b1010) & GHR_counter_sel[21]) | ((GHR == 4'b1011) & GHR_counter_sel[23])
                     | ((GHR == 4'b1100) & GHR_counter_sel[25]) | ((GHR ==
4'b1101) & GHR_counter_sel[27]) | ((GHR == 4'b1110) & GHR_counter_sel[29])
```

```
                    | ((GHR == 4'b1111) & GHR_counter_sel[31]);
    wire GHR_hit;
    assign GHR_hit = (|hitn) & GHR_taken;
    wire [31:0] GHR_data;
    assign GHR_data = cacheData[sel];
    reg GHR_hit_id, GHR_hit_ex;

    always @(posedge clk) begin
        GHR_hit_ex <= GHR_hit_id;
        GHR_hit_id <= GHR_hit;
    end
```

全局预测与局部预测的竞争

同样设置一个2bit计数器，一开始初始化为01，弱选择局部跳转。真实的跳转结果根据这个2bit的计数器来决定，如果是00，01 则依据局部预测的结果来判断是否跳转，反之则依据全局预测来跳转。上一条指令如果是局部预测且预测成功，则计数器-1，更倾向于局部预测，反之则+1；上一条指令如果是全局预测的结果且预测成功，则计数器+1，反之-1

```
// compete

    reg [1:0] global;
    initial global = 2'b01; //weakly local

    always @(posedge clk) begin
        if((|hitn_ex) & inIfBranch)
            global <= (global == 2'b00) ? global : global - 1;
        else if(GHR_hit_ex & inIfBranch)
            global <= (global == 2'b11) ? global : global + 1;
        else
            global <= global;
    end

    assign real_hit = (global[1] == 0) ? hit : GHR_hit;
```

冒险的修改（预测错误）

hazard模块中的控制冒险需要修改，当预测认为跳转但结果是不跳转，预测失败时，就需要冲刷if id ex，然后重新选择pc

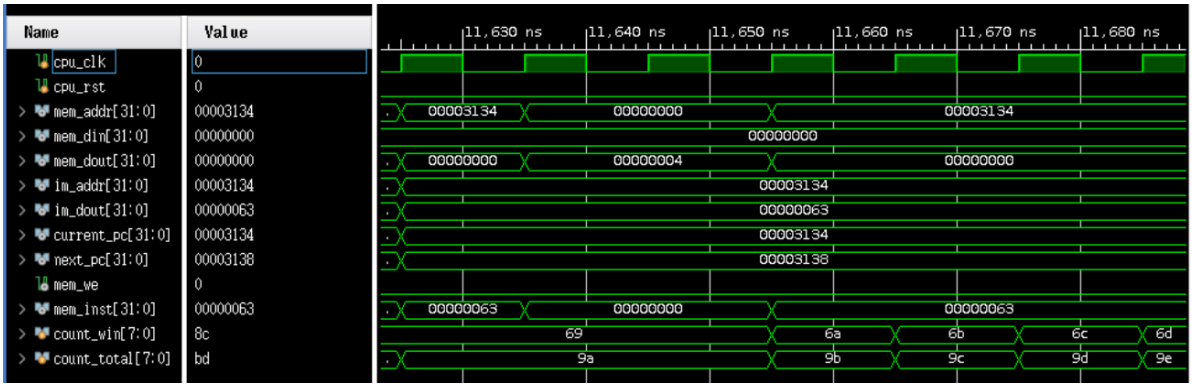
```
    //control hazard
    always @(*) begin
        if(hit_ex == 1 && inIfBranch != 1) begin    //forecast fail
            flush_id = 1;
            flush_ex = 1;
            flush_if = 1;
        end
        else if(pc_sel_ex != 2'b0 && (inIfBranch & hit_ex) == 0) begin
            flush_id = 1;
            flush_ex = 1;
            flush_if = 1;       //??
```

```
        end
        else begin
            flush_id = 0;
            flush_ex = 0;
            flush_if = 0;
        end
    end
```

而对pc的重新选择用一个选择器即可，当cache命中且预测跳转时，pc选择cache里的数据直接跳转，当cache命中，预测跳转，但实际不跳转时，需要选择将当前ex段的pc 传回来，ex的pc+4即为跳转指令的下一条，冲刷流水线并从这个pc+4开始重启流水线

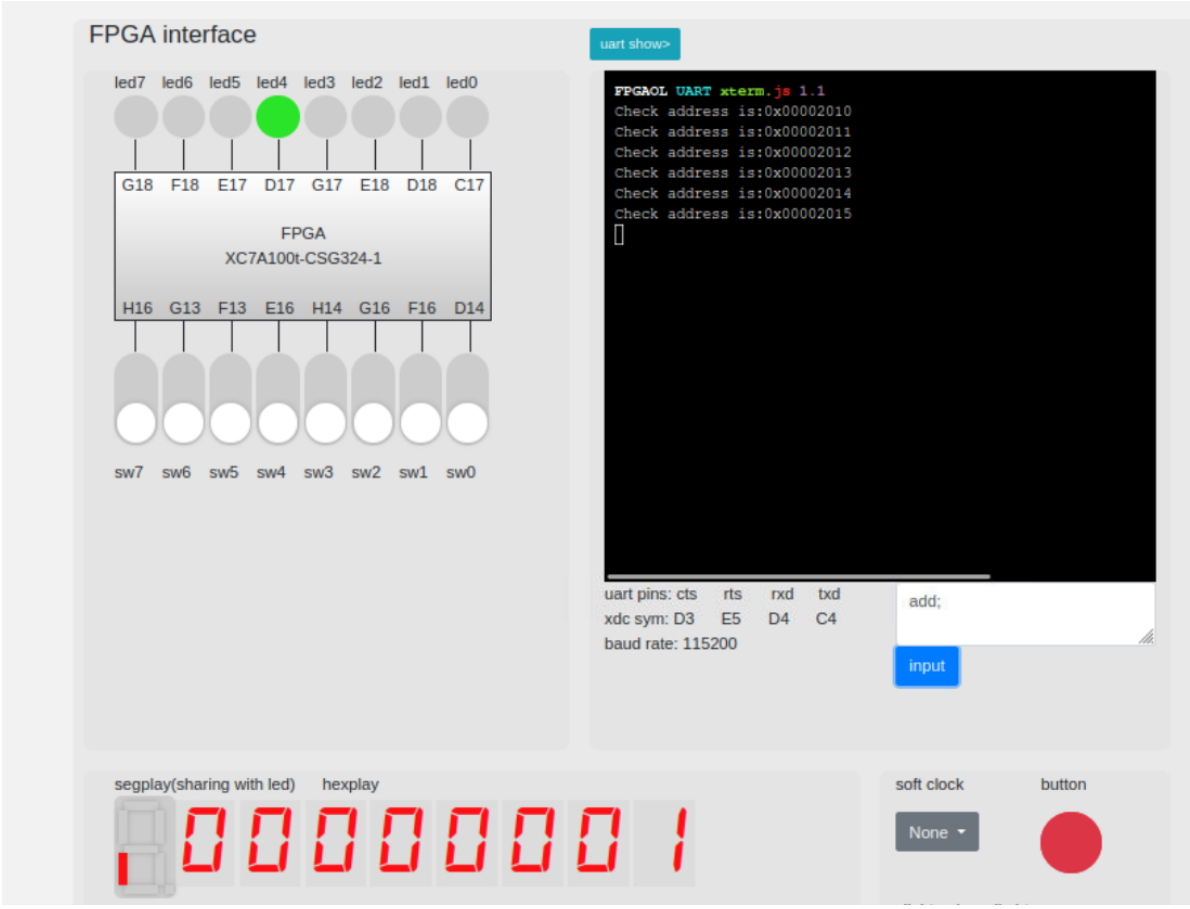
```
    assign sel_two_bit = {forecast_fail, hit_if};
    MUX4 real_pc(.src0(pc_next), .src1(pc_jump), .src2(pc_cur_ex + 3'b100),
.src3(32'b0), .sel(sel_two_bit), .res(real_pc_next));
```

预测结果



运行矩阵乘法代码，代码的最后一行是无条件跳转到自己，所以可以看到pc与之保持在0x3134，所以从一直成功预测的前面计算准确率，其中count_win是预测正确的计数， count_total是总共跳转的计数，0x69/0x9a = 68%，是符合预期的

上板结果



矩阵乘法的结果存在0x0040到0x0054，且全是1，上板用ck2 10检查结果，六个1，结果正确，符合预期