实验六 综合设计 展示

组员：杨涛、赵子毅

指令补全

实现指令

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu

实现指令（续）

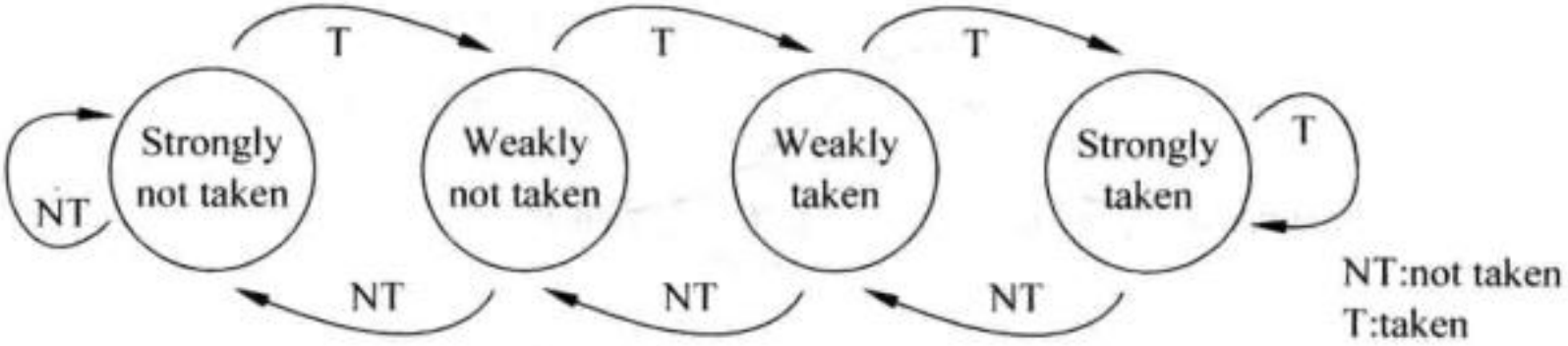
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	101	rd	0010011	I srli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	Rsltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and

分支预测

分支预测

- **1. 饱和计数器 SaturatingCounter**
- **2. 全局历史寄存器 GHR**
- **3. 跳转目标cache（BranchTargetBuffer）**
- **4. 具体逻辑**
- **5. 测试数据**

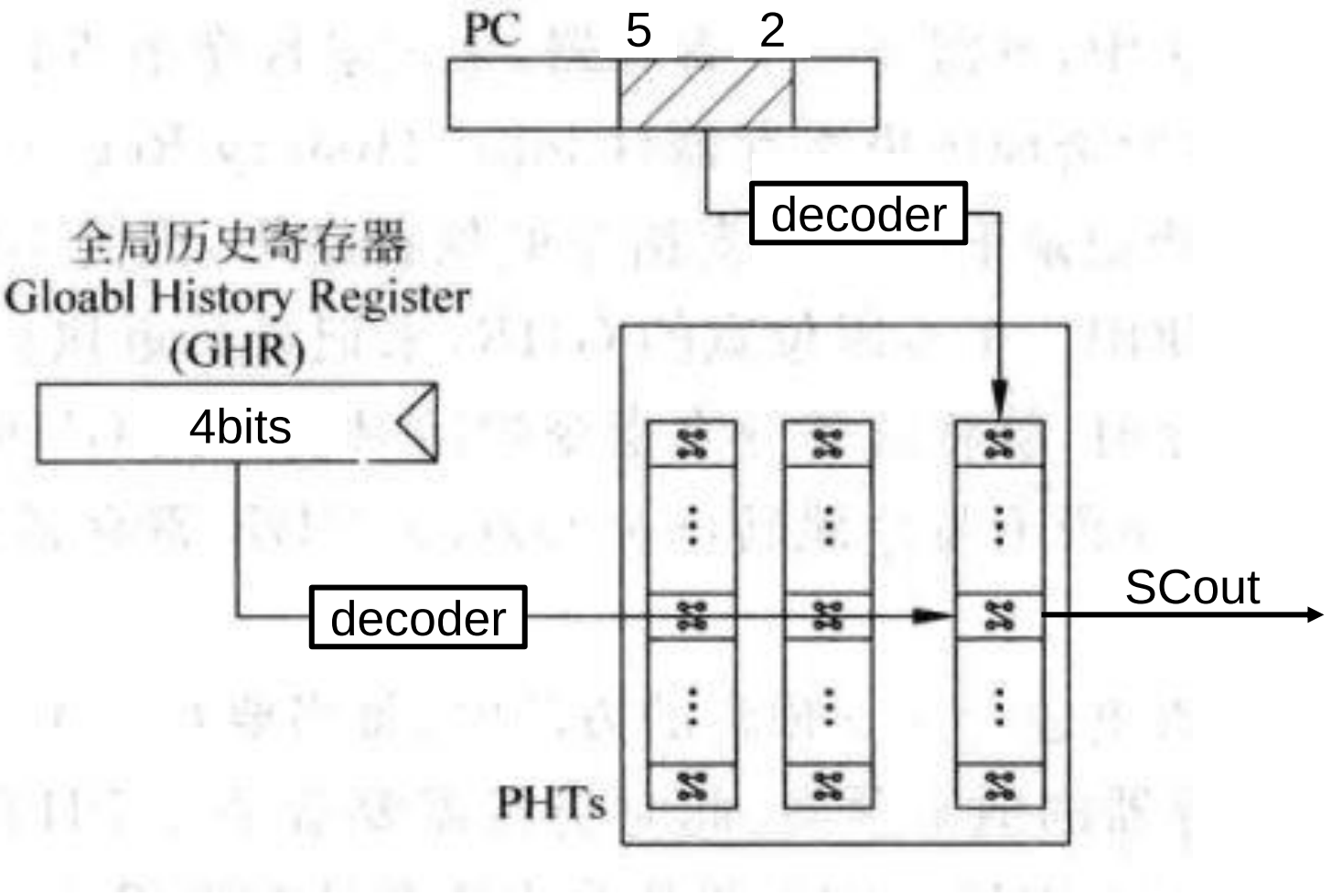
饱和计数器 SaturatingCounter



饱和计数器 SaturatingCounter

```
reg [1:0] counter;
always @(posedge clk or negedge rstn) begin
    if(~rstn) counter<=1;//弱不跳转
    else
    if(we)
        if(inIfBranch)
        begin
            if(counter==2'b11) counter<=counter;
            else counter<=counter+1;
        end
        else
        begin
            if(counter==2'b00) counter<=counter;
            else counter<=counter-1;
        end
end
assign outIfBranch = counter [1];
```

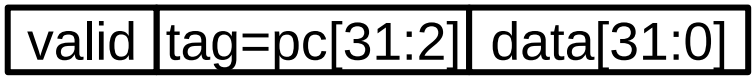
全局历史寄存器 GHR


PC
5
2
decoder
全局历史寄存器
Gloabl History Register
(GHR)
4bits
decoder
SCout
PHTs

跳转目标cache（BranchTargetBuffer）

- 为了在ID段可以读出跳转目标的指令
- 采用全相连、调换策略为FIFO
- 每个单元格式如下：

  有效位、跳转目标地址、地址对应内存data

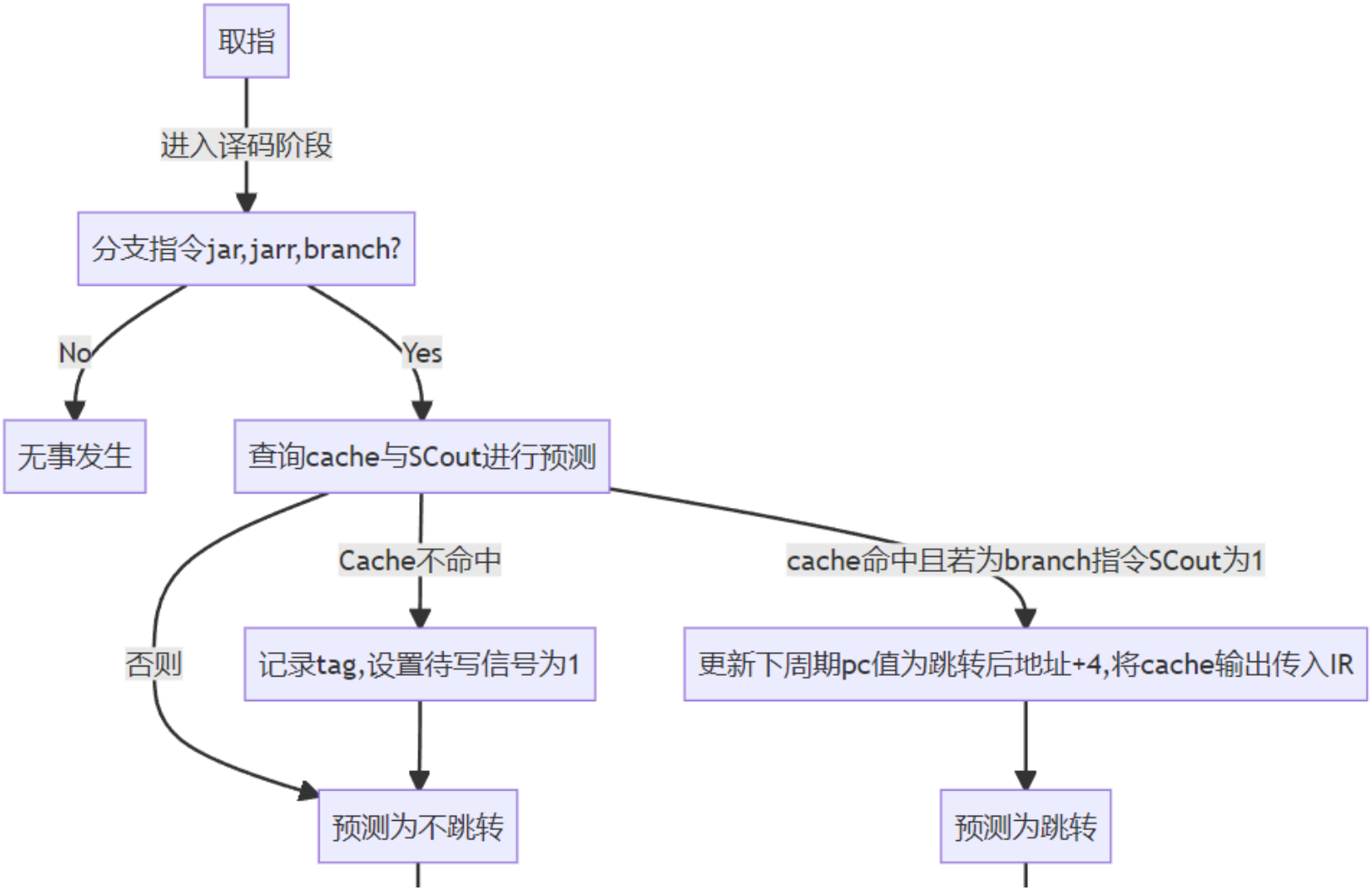


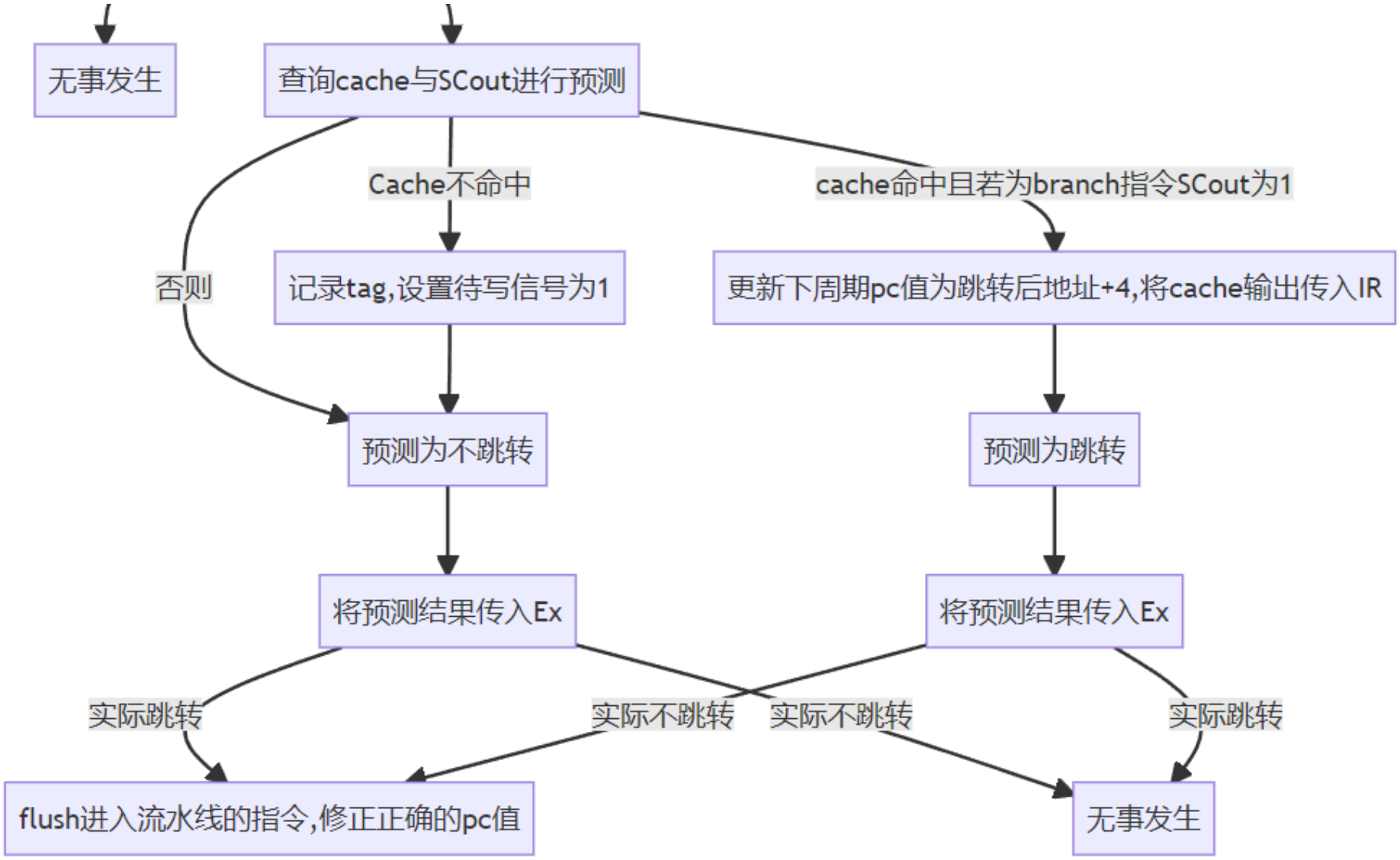
- 模块写入：当输入memWE有效时，判断cache内是否已存在该tag，若无，将memAdd和memData写入cache

```
//全相连，FIFO，共有8个缓存地址

parameter WIDTH=8;
parameter WIDTH_CNT = 3;
reg [WIDTH-1:0] valid=0;
reg [29:0] tag[WIDTH-1:0];//用[31:2]来确定tag
reg [31:0] cacheData[WIDTH-1:0];//对应的地址

reg[WIDTH_CNT-1:0] cnt=0;

wire w;
assign w=memWE&~(|(eqW&valid));
always @(posedge clk) begin
    if(w)
    if(cnt==WIDTH-1) cnt=0;
    else cnt=cnt+1;
end
always @(posedge clk) begin
    if(w)
    begin
        valid[cnt]<=1;
        tag[cnt]<=memAdd[31:2];
        cacheData[cnt]<=memData;
    end
end
```

```
37
38  reg[WIDTH-1:0] eq,eqW;
39  integer i;
40  always @(*) begin
41      for(i=0; i<WIDTH; i=i+1)
42      begin
43      eq[i]=(address[31:2]==tag[i]);
44      eqW[i]=(memAdd[31:2]==tag[i]);
45      end
46  end
47
48  wire [WIDTH-1:0] sig;
49  assign sig = eq & valid;
50  assign hit = |sig;
51  wire [WIDTH_CNT-1:0] sel;
52  encoder_16bits encoder_16bits({0,sig},sel);
53  assign data=cacheData[sel];
54
55  endmodule //InsCache
```

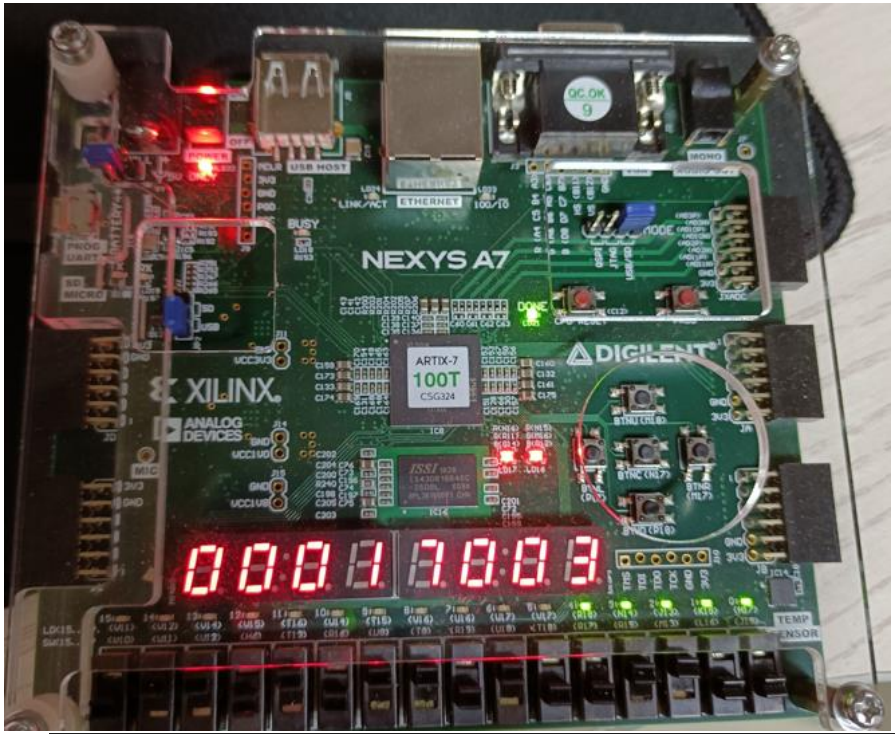
具体分支逻辑设置

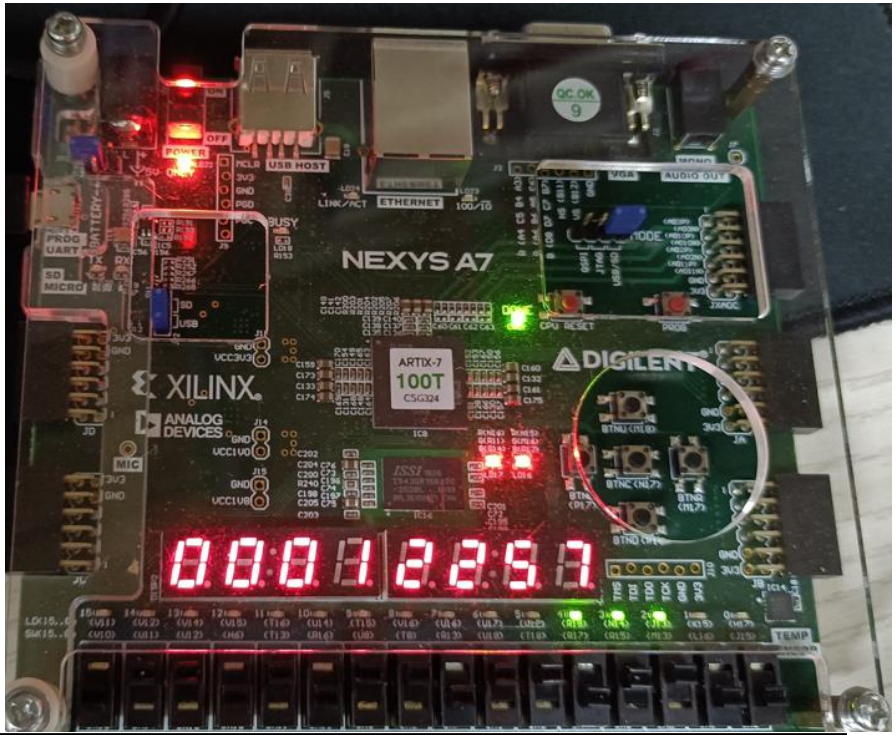

取指
进入译码阶段
分支指令jar,jarr,branch?
No
Yes
无事发生
查询cache与SCout进行预测
Cache不命中
cache命中且若为branch指令SCout为1
否则
记录tag,设置待写信号为1
更新下周期pc值为跳转后地址+4,将cache输出传入IR
预测为不跳转
预测为跳转


无事发生
查询cache与SCout进行预测
Cache不命中
cache命中且若为branch指令SCout为1
否则
记录tag,设置待写信号为1
更新下周期pc值为跳转后地址+4,将cache输出传入IR
预测为不跳转
预测为跳转
将预测结果传入Ex
将预测结果传入Ex
实际跳转
实际不跳转
实际不跳转
实际跳转
flush进入流水线的指令,修正正确的pc值
无事发生

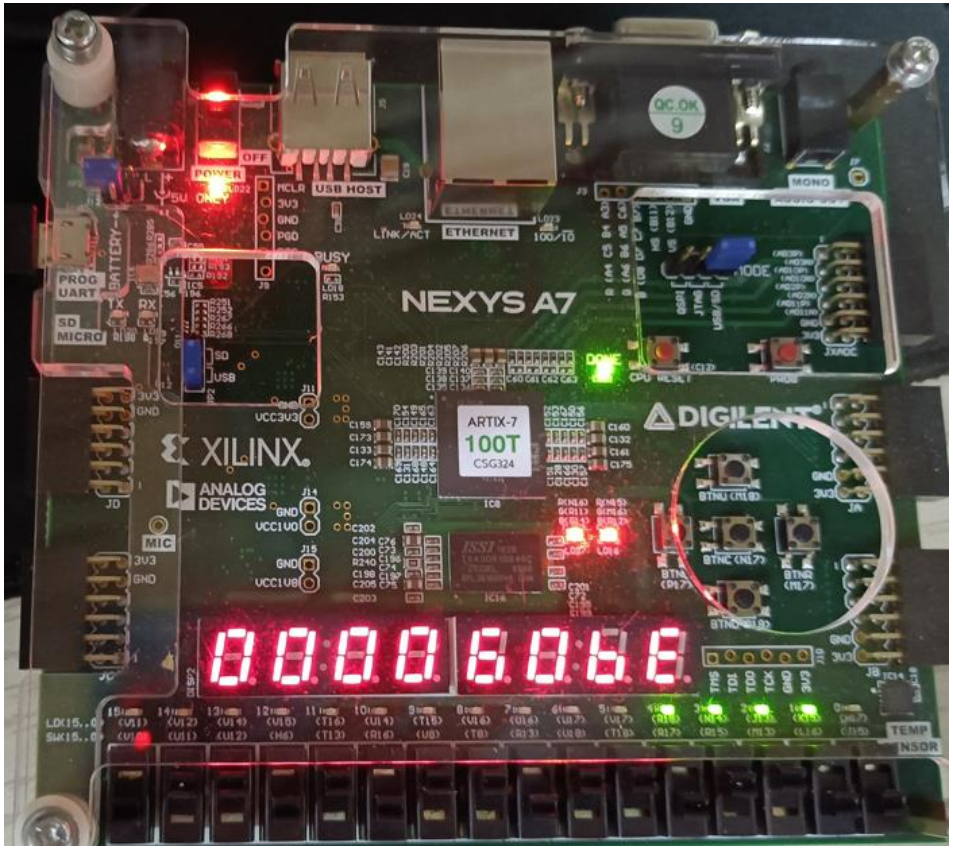
测试数据

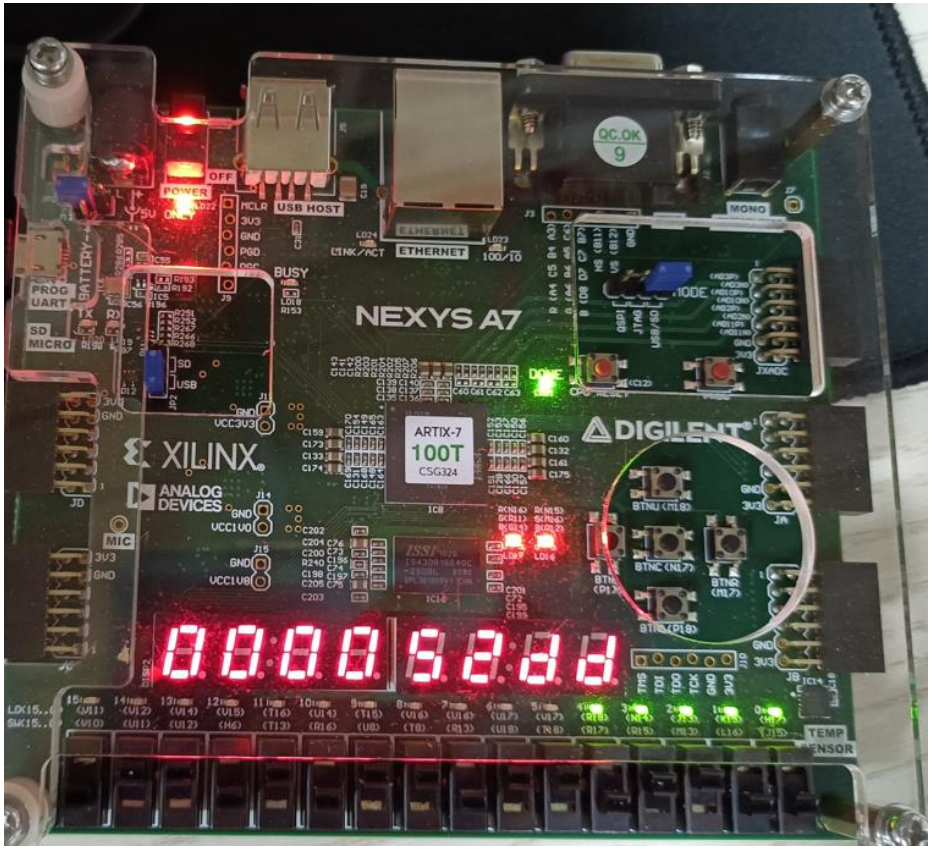
- 以256个数据的冒泡排序为测试程序
- 左\右分别为无（不跳转）\有分支预测所用周期数
- 可计算得性能约是原来的1.26倍





- 左边是跳转指令（jar、jarr、branch）运行总数
- 右边是预测命中（预测＝最终跳转）的总数





- 随机生成多组数据，测试命中率

预测次数	预测成功次数	命中率
0x60be	0x52dd	$\frac{21,213}{24,766} = 0.8565$
0x60be	0x528f	$\frac{21,135}{24,766} = 0.8534$
0x60be	0x5284	$\frac{21,124}{24,766} = 0.8529$
0x60be	0x5086	$\frac{20,614}{24,766} = 0.8324$
0x60be	0x5485	$\frac{21,647}{24,766} = 0.8741$
平均命中率		0.8543

- 排序程序最内层

```
LOOP2:
        blt t2,t1,LOOP2FIN          #跳转概率基本等于1
        lw t4,4(t1)
        bge t3,t4,skip              #跳转概率约0.5
        mv t5,t3
        mv t3,t4
        mv t4,t5
        skip:
        sw t4,0(t1)
        addi t1,t1,4
        j LOOP2                     #跳转概率等于1
LOOP2FIN:
```

- **理论命中率为66.6%+33.3%*0.5=83.25%**

数据Cache

数据cache

- 端口定义：
- 相比ip核新增读请求信号以及rstn信号

```
input [7:0] Address,     //cpu给的地址
input [31:0] WriteData,         //写的数据
input [7:0] DebugAddr,      //chk用 仅可读
input clk,
input wr_req,//写请求信号
input rd_req,
input rstn,
output wire [31:0] ReadData,        //从cache中取出的数据
output wire [31:0] DebugData,
//debug
output wire hit_m
```

数据cache

- localparam way_cnt = 2;//组相连度
- //cache与主存之间使用全写法和不按写分配法
- //使用块大小为一字（一字四字节）两组 共计8块的直接映射cache 总容量8*2*1=16字
- //FIFO策略
- //ADDR: 2:0为index 7:3为tag
- reg [31:0] cache[7:0][way_cnt - 1:0];
- reg valid[7:0][way_cnt - 1:0];//标记有效位
- reg [4:0] tag[7:0][way_cnt - 1:0];//标签位

- assign index = Address[2:0];
- assign tag_in = Address[7:3];

数据cache

- Hit与寻道way_addr
- wire [way_cnt-1 : 0] way_addr;//记录数据来自哪个通道
- wire hit;
- assign hit = (valid[index][0] && tag[index][0] == tag_in) | (valid[index][1] && tag[index][1] == tag_in);
- assign way_addr = (valid[index][0] && tag[index][0] == tag_in) ? 0 : 1;

数据cache

- FIFO策略
- reg FIFO[7:0][way_cnt:0];
- 1. 使用FIFO对每个数据块计数 0表示初始状态 数字从1到way_cnt 越小表示越后进来
- 2. 替换时：
  - 查找有无FIFO == 0的块 有则直接存到该块
  - 若没有 寻找FIFO == way_cnt的块 作为新数据将使用的块
  - 最后修改FIFO的值

数据cache

- 全写法与不按写分配法
- 1．写入：写命中时同时写cache和主存
  - ip核直接连接输入信号 不需要操作
  - cache[index][way_addr] <= WriteData;

2．读取：命中读cache 不命中读主存同时换入

- assign ReadData = (hit == 1) ? cache[index][way_addr] : ip_readdata;
- 不命中：
- cache[index][outway] <= ip_readdata;
- valid[index][outway] <= 1'b1;
- tag[index][outway] <= tag_in;

The End