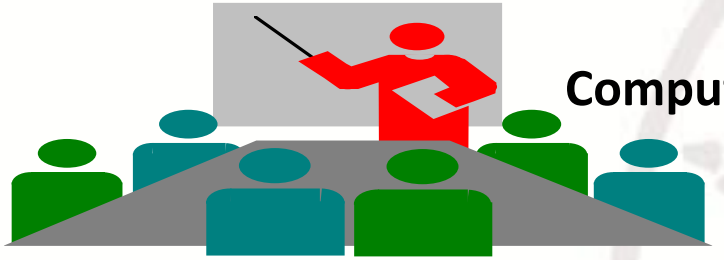


Computer Organization & Design
实验与课程设计

实验十二

无流水级内锁之流水线优化

--流水线数据相关硬件优化：Forwarding

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

1. **深入理解流水CPU结构**
   - 只能运行固定状态机事件的寄存器传输结构
   - 相关性极大限制了流水线性能发挥
2. **深入理解流水优化性能的本质**
   - 流水技术降低了单条指令的执行性能
   - 流水线改善提高了指令执行的吞吐率
3. **深入理解相关性数据与冒险竞争的本质**
   - 寄存器数据相关性与流水级冒险竞争
4. **硬件重定向数据相关消除寄存器变量的冒险竞争**

实验环境

实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统

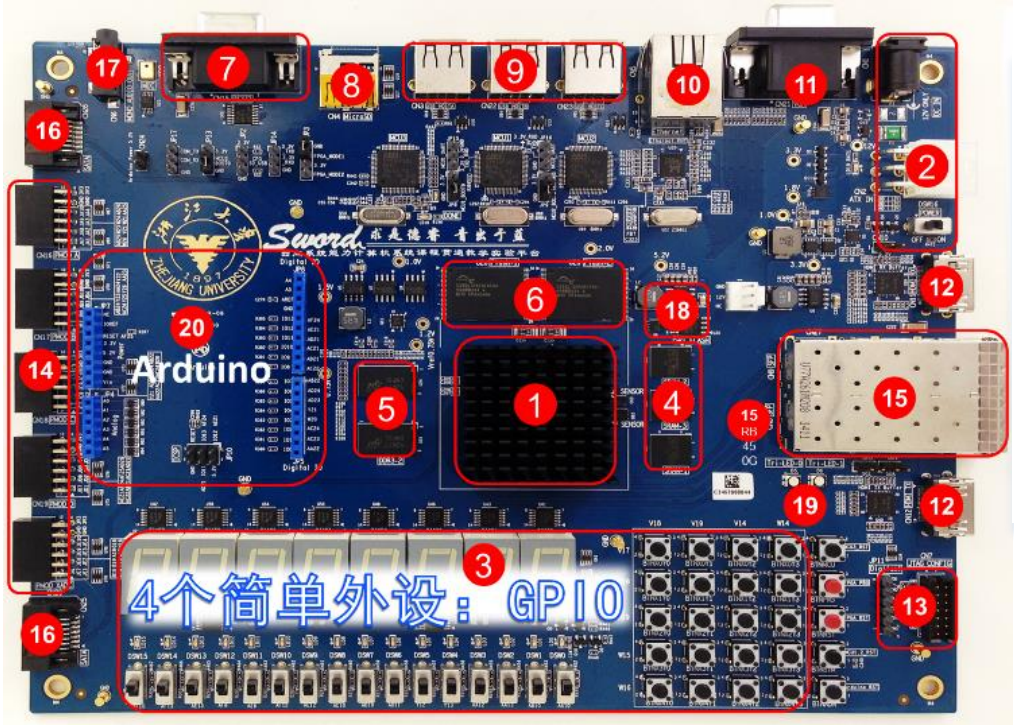
2. 计算机软硬件课程贯通教学实验系统(Sword)

3. 3. Vivado 2017.4或Xilinx ISE14.4及以上开发工具

材料

无

计算机软硬件课程贯通教学实验系统



贯通教学实验平台主要参数

▼ **核心芯片**

Xilinx Kintex™-7系列的XC7K160/325资源：

162,240个，Slice：25350，片内存储：11.7Mb

▼ **存储体系** 支持32位存储层次体系结构

6MB SRAM静态存储器：支持32Data，16位TAG

512M BDDR3动态存储：支持32Data

32MB NOR Flash存储 ：支持32位Data

▼ **基本接口** 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管

▼ **标准接口** 支持基本计算机系统实现

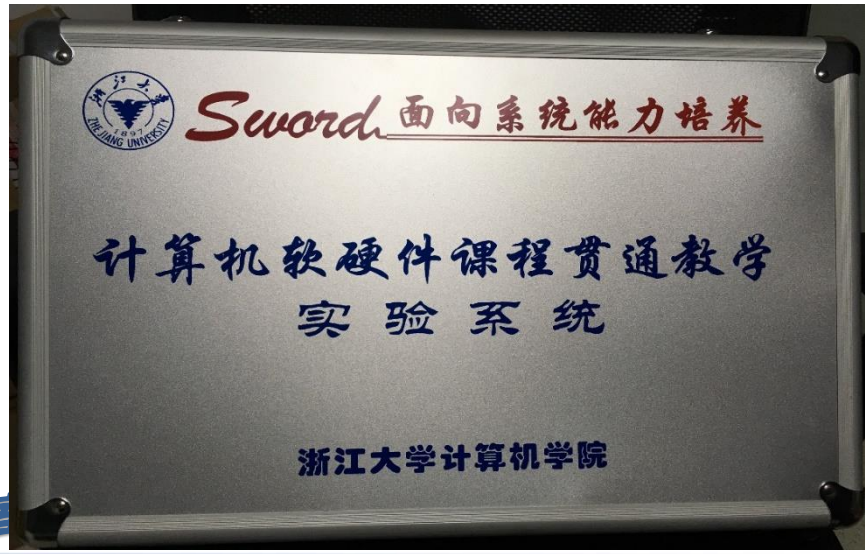
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **通讯接口** 支持数据传输、调试和网络

UART接口、 10M/100M/1000M以太网、 SFP光纤接口

▼ **扩展接口** 支持外存、多媒体和个性化设备

MicroSD(TF）、PMOD、 HDMI、Arduino



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务

1. 数据通路Forward(直接定向反馈)通路设计

- EXE级增加ALU运算Forwarding/Bypass
- ID级增加条件判断Forwarding/Bypass(不经锁存器通路)

2. 基于Forward通路检测控制电路设计

- 增加直接通路判断一：EXE级检测冒险并控制直接定向通路
- 增加直接通路判断二：ID级检测Branch Condition code冒险并控制

3. 修改数据通路相关性检测

- Forward后仍需阻塞流水线设计(Stall)：
  - Load相关指令(Branch条件比较Forward经锁存器) 阻塞等待1 clk

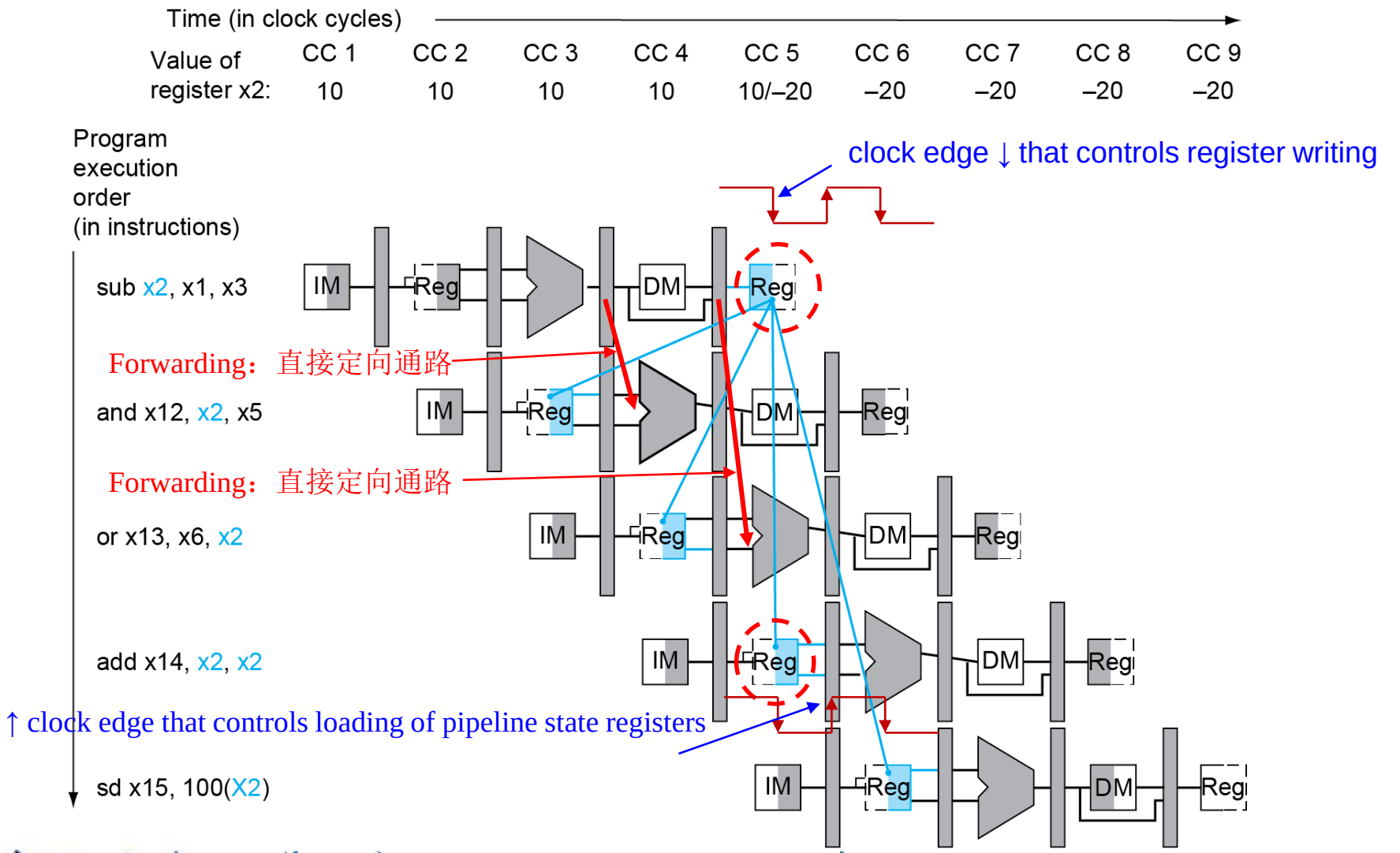
4. 测试not taken预测流水线冲刷清除

- 设计测试程序遍历测试所有可能的相关性
- 此项需要完备性测试

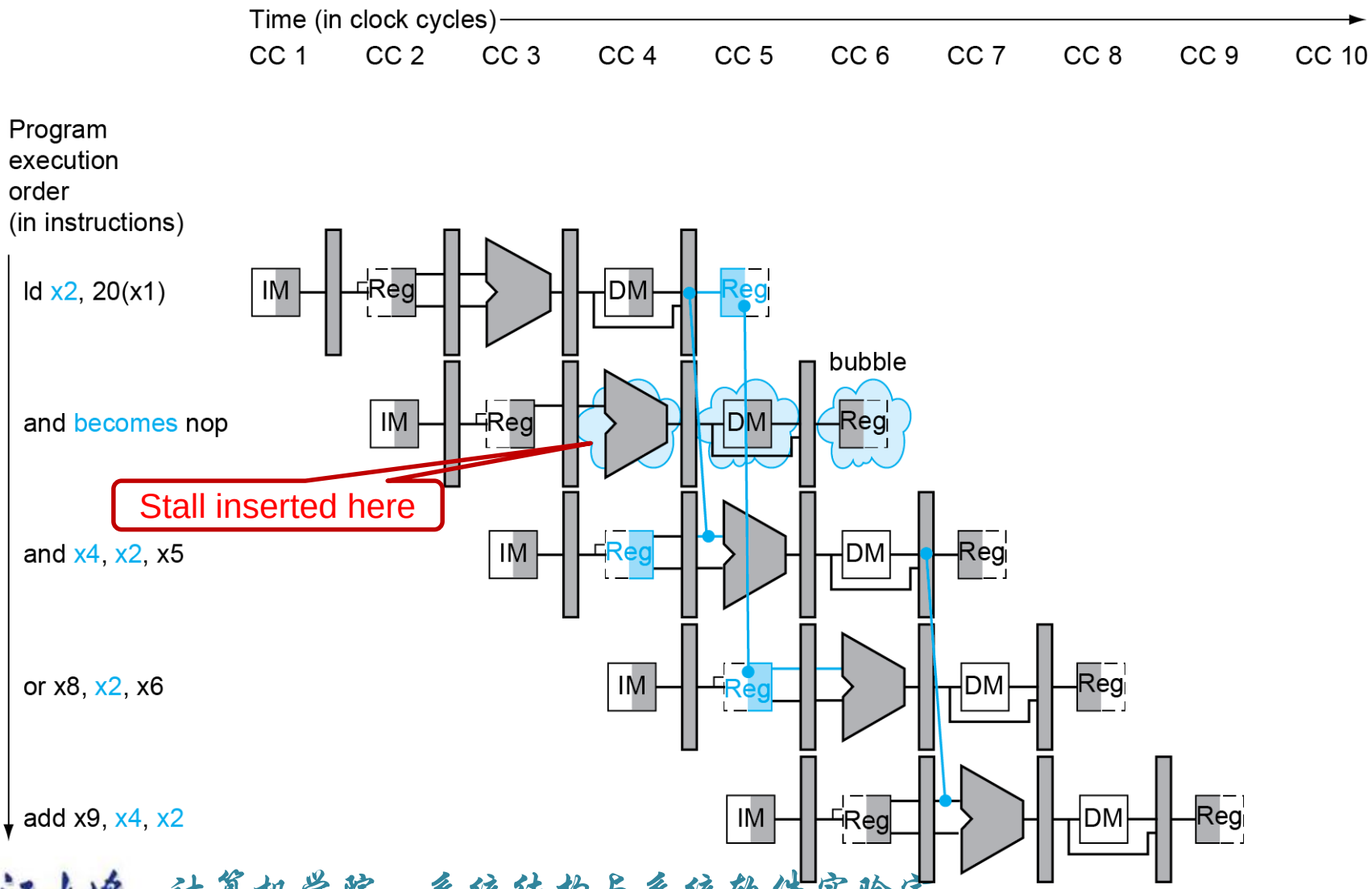
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

Dependencies & Forwarding

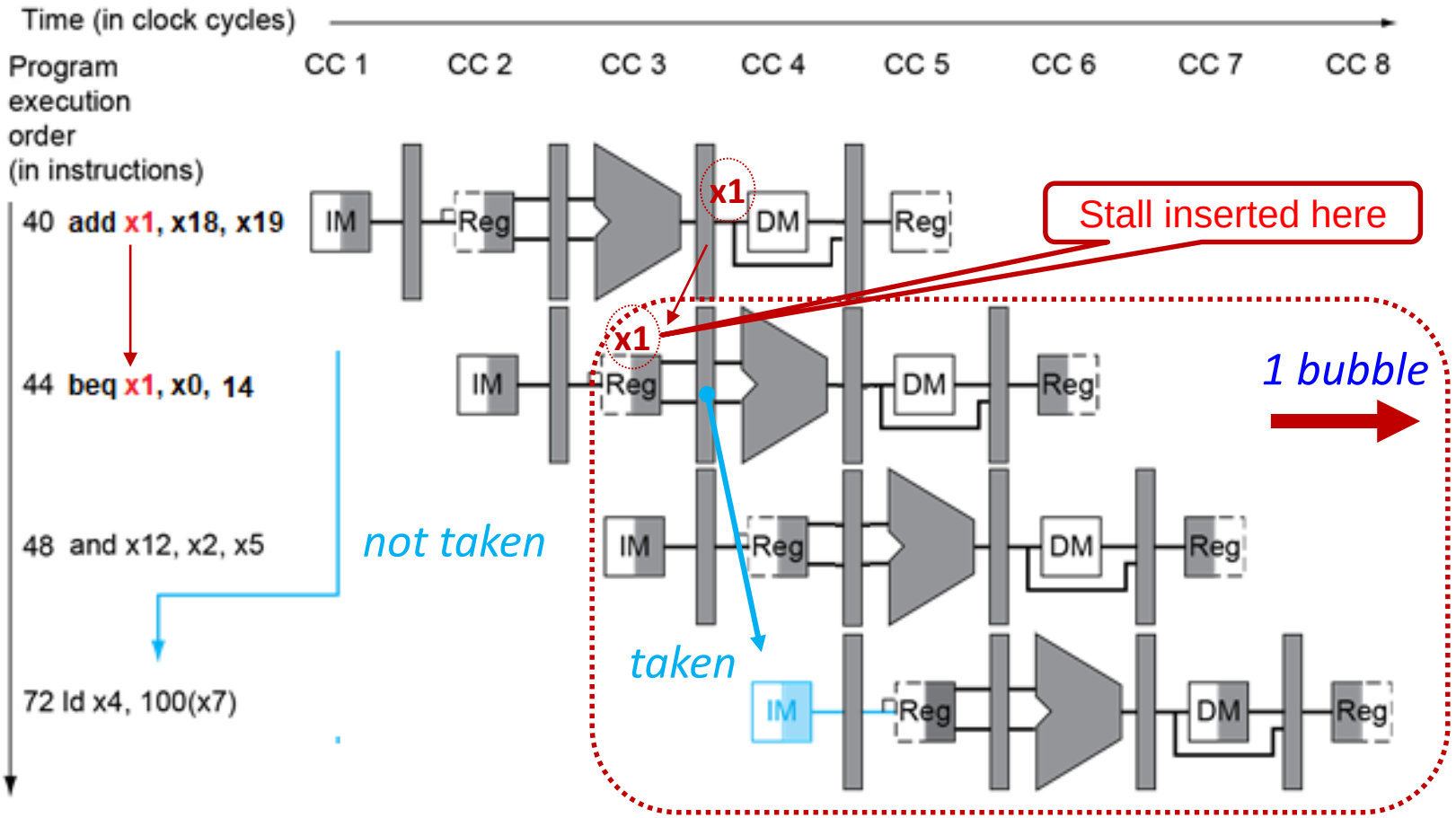


Load-Use Data Hazard



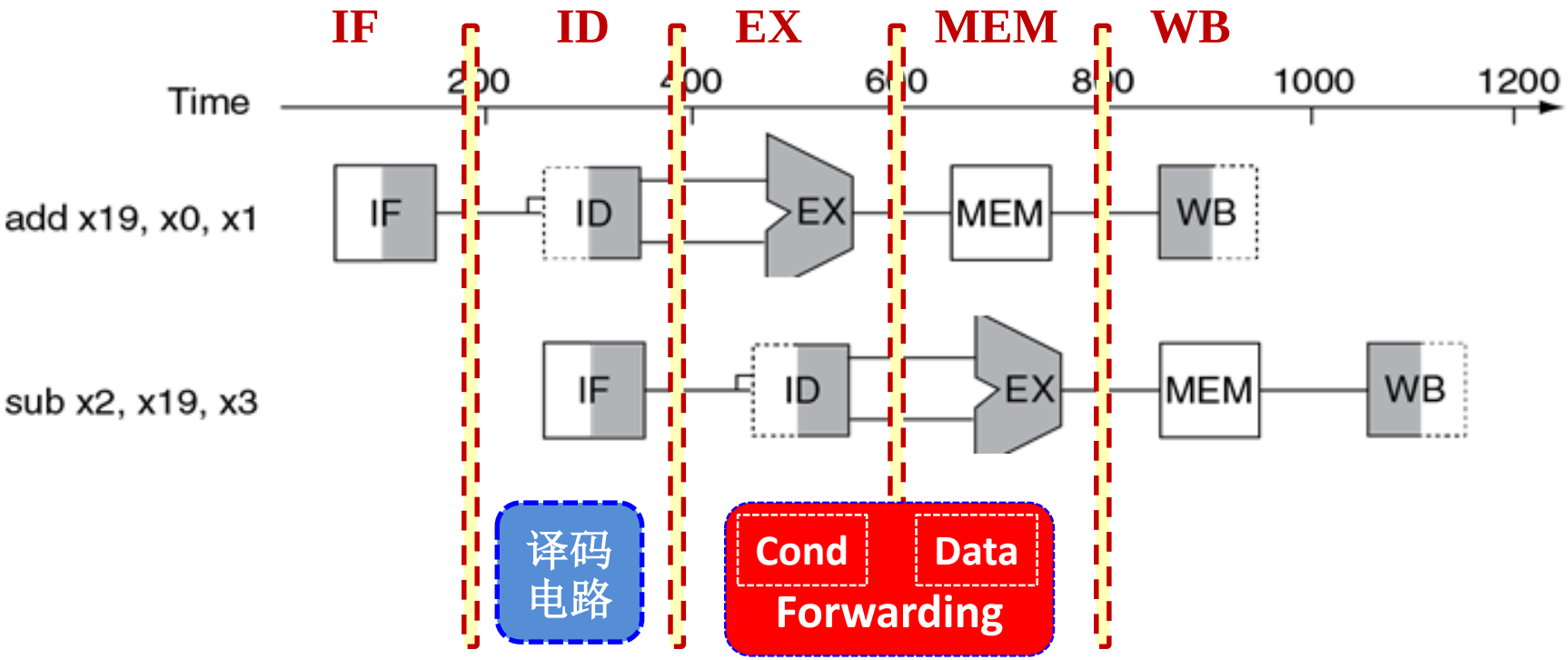
Branch Condition Code Data Hazard

Dependencies with Condition Code



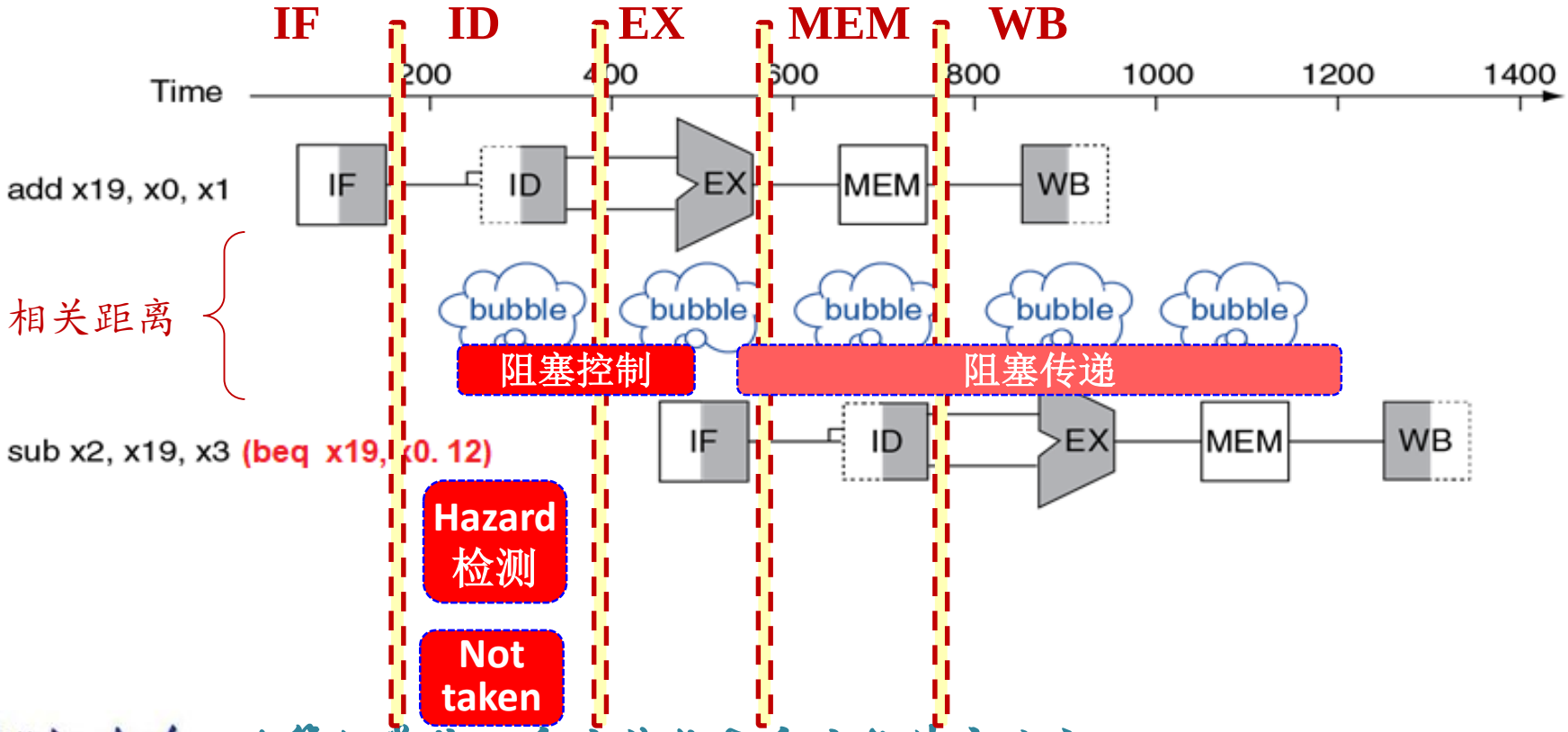
相关性变量重定向传输通路

- 定向通路本质：及时检测，及时选择
  - 普通指令EXE级检测数据相关性：判断相关数据通路
  - Branch 指令在ID级检测条件码相关性：判断相关数据通路



Detecion：Load or Condition Code

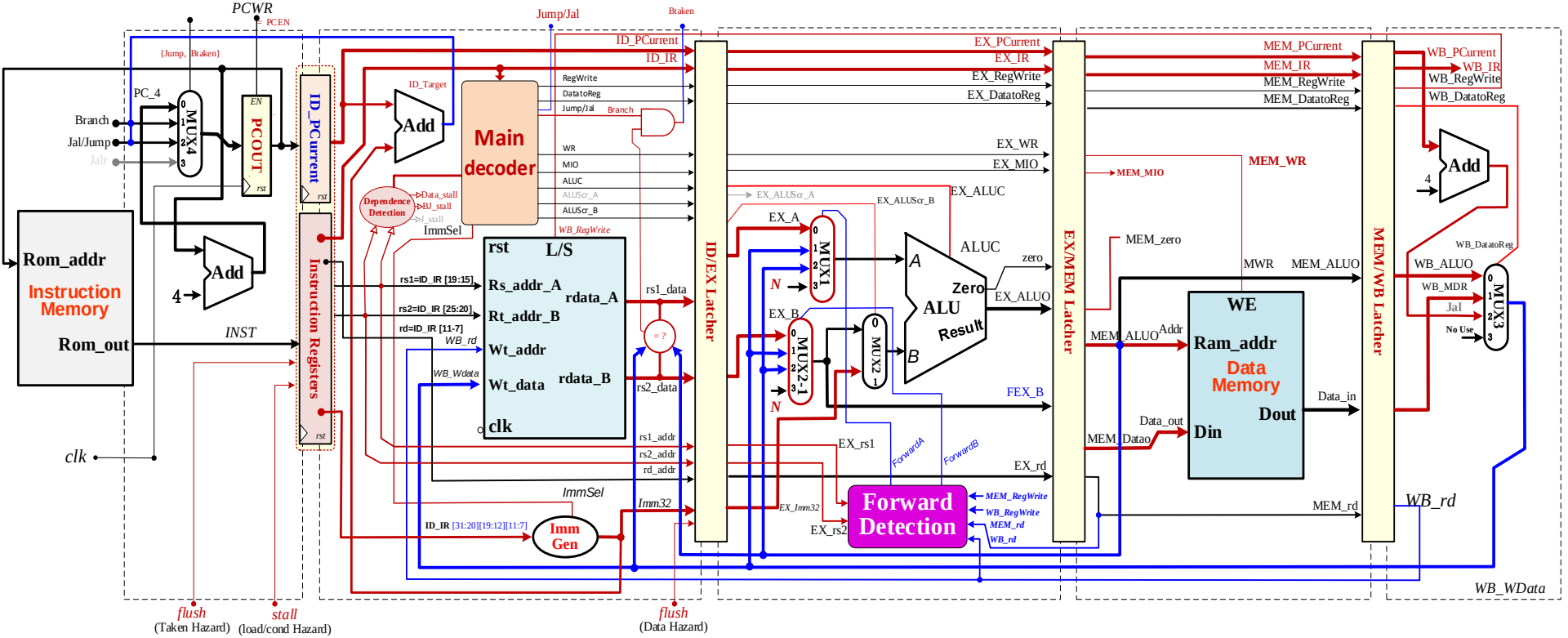
- **Still need to stall pipeline after Forwarding**
  - 尽早检测，清除方便
  - 当前指令ID级检测Load相关性和条件码相关



无流水级内锁数据通路： Forwarding

- 基于实验11增加Forwarding通路
  - EX级增加： Data Forwarding
  - ID级增加： Branch not taken Forwarding

以9条指令为例

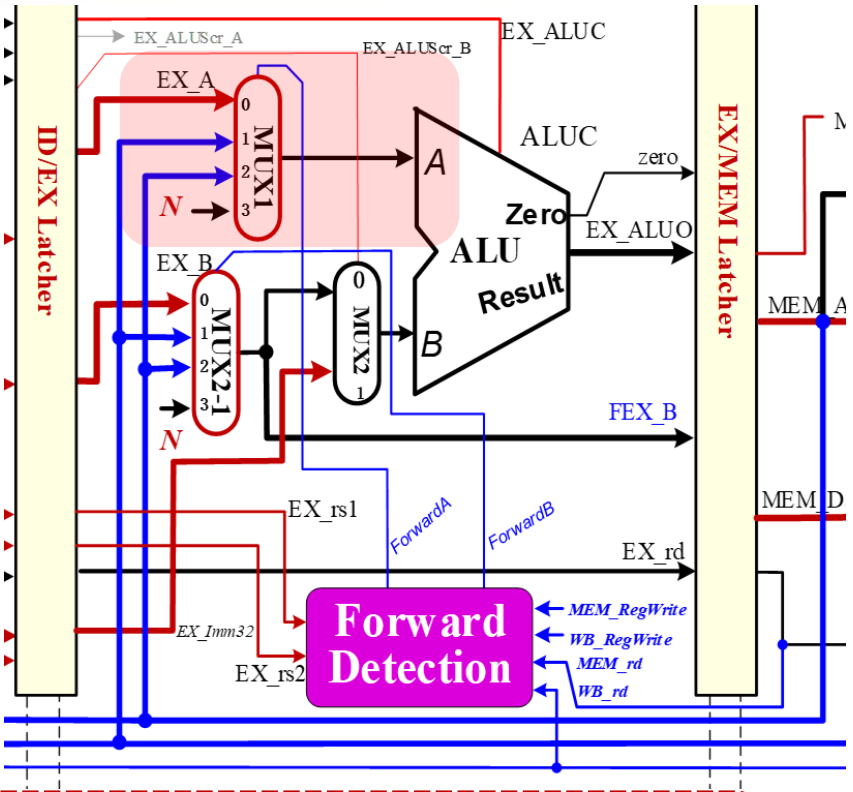


Forwarding：ALU A输入端

- 增加或扩展ALU A输入端通路
  - MUX1

```
//ALUA Forwarding with Data Harzard
    always @* begin                    //ALU A通道输入数据
        case(ForwardA)                 // MUX1
            2'b00: ALUA = EX_A;        //无竞争 直接选择rs1寄存器读出
            2'b01: ALUA = Wt_data;     //竞争2: 选择直接通路WB级Wt_data
            2'b10: ALUA = MEM_ALUO;    //竞争1: 选择直接通路MEM级MEM_ALUO
            2'b11: ALUA = EX_A;        //No Use
        endcase
    end


  //Forwarding Unit 在EX级检测
  reg[1:0]ForwardA, ForwardB;
  //reg EX_rs1_addr,   EX_rs2_addr;
      always @* begin
    if(MEM_RegWrite && MEM_rd !=0 &&          //前续EX级指令rd存在写：锁存于MEM级
       EX_rs1 != 0 && EX_rs1 == MEM_rd)       //当前指令与前续指令1相关rs1== MEM_rd
       ForwardA = 10;                         //选择EX级输出锁存通路
    else if(WB_RegWrite && WB_rd !=0 &&       //前续MEM级指令rd存在写：锁存于WB级
            EX_rs1 != 0 && (EX_rs1 == WB_rd))  //当前指令与前续指令2相关rs1== WB_rd
            ForwardA = 01;                    //选择MEM级输出锁存通路
          else  ForwardA = 2'b00;
```



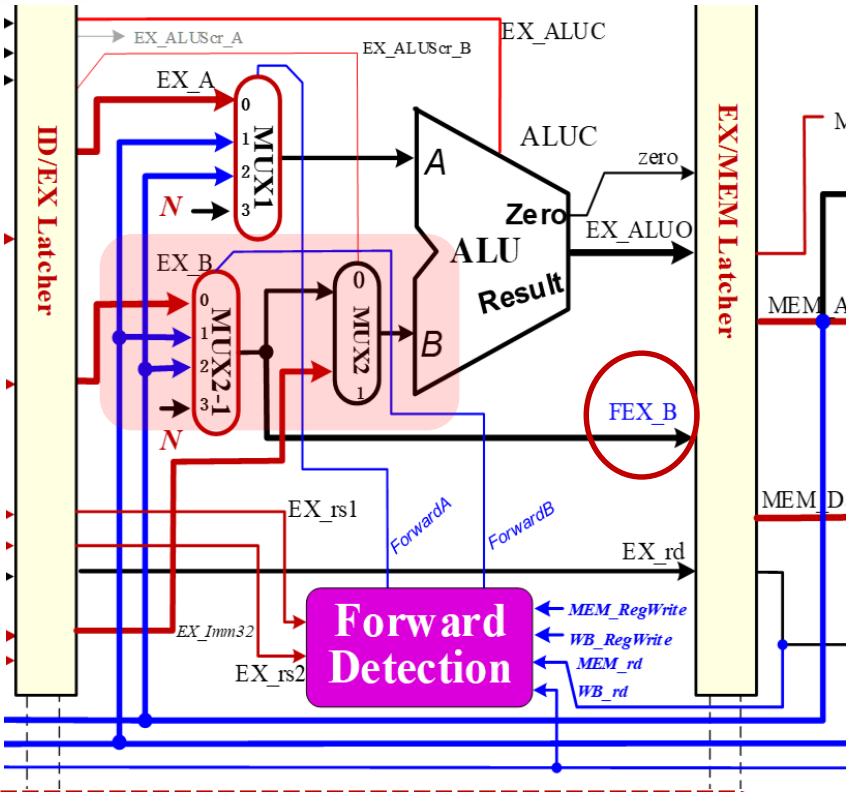
靠近优先

Forwarding：ALU B输入端

- 增加ALU B输入端通路
  - 保持MUX2：选择FEX_B或Imm32
  - 增加MUX2-1：Forwarding选择通路

```
always @* begin                    //ALU B通道输入数据
    case(ForwardB)                 // MUX2-1 直接rs2寄存器读出
        2'b00: FEX_B = EX_B;       //无竞争 直接选择rs2寄存器读出
        2'b01: FEX_B = Wt_data;    //竞争2: 选择直接通路WB级Wt_data
        2'b10: FEX_B = MEM_ALUO;   //竞争1: 选择直接通路MEM级MEM_ALUO
        2'b11: FEX_B = EX_B;       //No Use
    endcase
end                //保持MUX2，选择最终ALU B通道输入数据:
assign ALUB = EX_ALUSrc_B ? EX_Imm32 : FEX_B;
```

MUX2




```
always @* begin
        -----------
if(MEM_RegWrite && MEM_rd !=0 &&          //前续EX级指令rd存在写：锁存于MEM级
   EX_rs2 != 0 && EX_rs2 == MEM_rd)       //当前指令与前续指令1相关rs2== MEM_rd
   ForwardB = 10;                         //选择EX级输出锁存通路
else if(WB_RegWrite && WB_rd !=0 &&       //前续MEM级指令rd存在写：锁存于WB级
        EX_rs2 != 0 && (EX_rs2 == WB_rd))  //当前指令与前续指令2相关rs2== WB_rd
        ForwardB = 01;                    //选择MEM级输出锁存通路
     else  ForwardB = 2'b00;
end
```

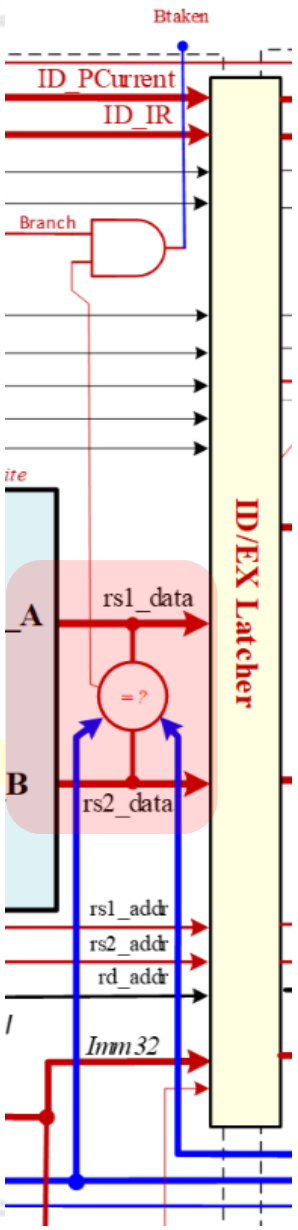
靠近优先

Forwarding： Condition Code Hazard

- 条件判断在ID级，无法共享EXE级通路
  - 条件码相关性与Load类同，
  - Forwarding通路需独立重建

```
reg[1:0]Forwardrs1, Forwardrs2;
  always @* begin
    if(Branch && EX_RegWrite && EX_rd !=0 &&          //前续EX级指令rd存在写：正在执行ALU
      rs1_addr != 0 && rs1_addr == EX_rd)             //当前ID级指令与前续EX级指令相关rs1== EX_rd
          Forwardrs1 = 10;          靠近优先          //选择EX级输出，非锁存
     else if(Branch && MEM_RegWrite && MEM_rd !=0 &&  //前续MEM级指令rd存在写：锁存于MEM级
          rs1_addr != 0 && (rs1_addr == MEM_rd))      //当前ID级指令与前续MEM指令相关rs1== MEM_rd
          Forwardrs1 = 01;                            //选择MEM级输出，非锁存
        else  Forwardrs1 = 2'b00;

    if(Branch && EX_RegWrite && EX_rd !=0 &&          //前续EX级指令rd存在写：正在执行ALU
      rs2_addr != 0 && rs2_addr == EX_rd)             //当前ID级指令与前续EX级指令相关rs2== EX_rd
          Forwardrs2 = 10;          靠近优先          //选择EX级输出，非锁存
     else if(Branch && MEM_RegWrite && MEM_rd !=0 &&  //前续MEM级指令rd存在写：锁存于MEM级
          rs2_addr != 0 && (rs2_addr == MEM_rd))      //当前ID级指令与前续MEM指令相关rs2== MEM_rd
          Forwardrs2 = 01;                            //选择MEM级输出，非锁存
        else  Forwardrs2 = 2'b00;
  end
```



Decision making：Branch通路

- 在ID级判断条件，决定是否转移
  - 若条件码Btaken=1，预测出错需要冲刷(flush)
  - 同时修改PC： PCNEXT = ID_Target

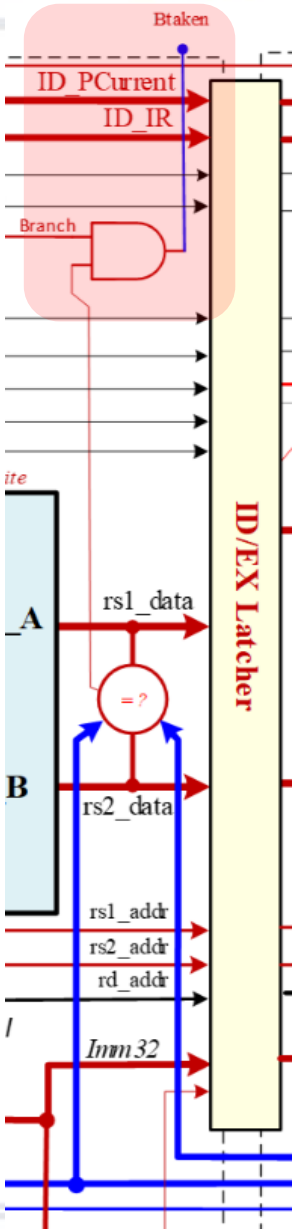
```
assign PCSource = {Jump, Btaken};
// Forwarding Branch with Data Harzard
reg Btaken;
always @*begin
    case({Forwardrs1,Forwardrs2})
        4'b0000: Btaken =  Branch && rs1_data == rs2_data;
        4'b0010: Btaken =  Branch && rs1_data == EX_ALUO;
        4'b0001: Btaken =  Branch && rs1_data == MEM_ALUO;

        4'b1000: Btaken =  Branch && EX_ALUO  == rs2_data;
        4'b1010: Btaken =  Branch && EX_ALUO  == EX_ALUO;
        4'b1001: Btaken =  Branch && EX_ALUO  == MEM_ALUO;

        4'b0100: Btaken =  Branch && MEM_ALUO == rs2_data;
        4'b0110: Btaken =  Branch && MEM_ALUO  == EX_ALUO;
        4'b0101: Btaken =  Branch && MEM_ALUO == MEM_ALUO;
         default: Btaken =  Branch && rs1_data == rs2_data;
    endcase
end
```

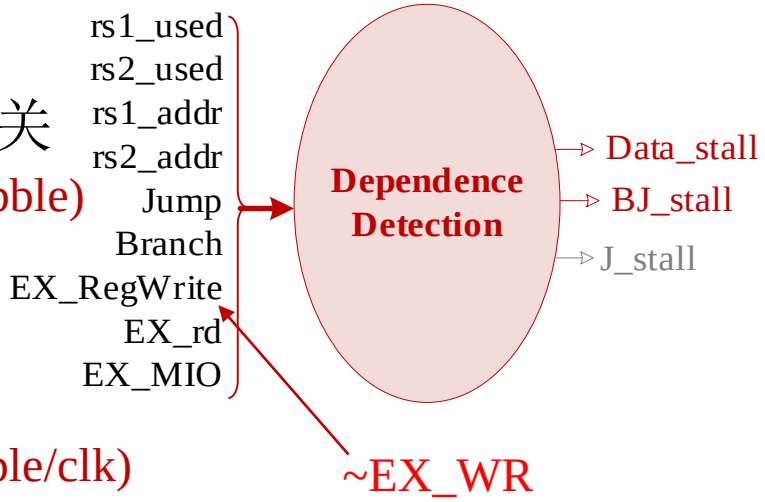
是否完备请测试

靠近优先



Still need to stall： Dependence Detection

- 数据相关
  - ID级rs1、rs2与EX级Load指令rd相关
    - Data Stall = 1：阻塞流水线1clk(Bubble)
- 控制相关
  - ID级指令译码为Control类指令
    - 转移预测错误则冲刷流水线
      - BJ Stall = 1：清除 ？条指令 (bubble/clk)



Data Stall

```
//Load相关性检测
reg HarzardLoad;
    always @* begin                                    //ID级判断与Load指令竞争
    HarzardLoad = 0;
        if(~EX_WR  && EX_MIO && EX_rd !=0 &&           //前续EX级Load指令rd存在写且不为0
          ((rs1_addr !=0 && rs1_addr == EX_rd) ||      //当前ID级rs1与前续EX级rd相关
           (rs2_addr !=0 && rs2_addr == EX_rd)))begin  //当前ID级rs2与前续EX级rd相关
           HarzardLoad = 1;
        end
    end
```

同实验11：Branch Flush

```
//Hazards Detection by Branch Dependence Hazard
    wire BJ_stall = Btaken || Jump;                    //当前ID级为转移指令
```

流水控制器译码参考：方案不唯一

```
reg [1:0] ALUop;
wire[3:0] Fun;

    assign ALE = ~clk;
    assign PCEN = 1;

    always @* begin
        ALUSrc_A  =0;
        ALUSrc_B  =0;
        DatatoReg =0;
        RegWrite  =0;
        Branch    =0;
        Jump      =0;
        WR        =0;
        CPU_MIO   =0;
        ALUop     =2'b10;
        rs1_used  = 0;
        rs2_used  = 0;

      case(OPcode)
        5'b01100: begin ALUop=2'b10;RegWrite=1;                ALUSrc_B=0;Branch=0;Jump=0;DatatoReg=2'b00;      //ALU(R)
                        rs1_used = 1; rs2_used = 1;                                                          end
        5'b00000: begin ALUop=2'b00;RegWrite=1;ImmSel=00;ALUSrc_B=1;Branch=0;Jump=0;DatatoReg=2'b01;      //load
                        rs1_used = 1;                                                WR=0;CPU_MIO = 1; end
        5'b01000: begin ALUop=2'b00;RegWrite=0;ImmSel=01;ALUSrc_B=1;Branch=0;Jump=0;WR=1;CPU_MIO = 1;     //store
                        rs1_used = 1; rs2_used = 1;                                                          end
        5'b11000: begin ALUop=2'b01;RegWrite=0;ImmSel=10;ALUSrc_B=0;Branch=1;Jump=0;                      //beq
                        rs1_used = 1; rs2_used = 1;                                                          end
        5'b11011: begin              RegWrite=1;ImmSel=11;                          Jump=1;DatatoReg=2'b10; end //jump
        5'b00100: begin ALUop=2'b11;RegWrite=1;ImmSel=00;ALUSrc_B=1;Branch=0;Jump=0;DatatoReg=2'b00;      //ALU(I)
                        rs1_used = 1;                                                                        end
        default:         ALUop=2'b00;
      endcase
    end
```

```
assign Fun = {Fun3,Fun7};
always @* begin
    case(ALUop)
        2'b00: ALUC=3'b010;                 //load/store
        2'b01: ALUC =3'b110;                //sub: beq
        2'b10:
            case(Fun)
                4'b0000: ALUC =3'b010;      //add
                4'b0001: ALUC =3'b110;      //sub
                4'b1110: ALUC =3'b000;      //and
                4'b1100: ALUC =3'b001;      //or
                4'b0100: ALUC =3'b111;      //slt
                4'b1010: ALUC =3'b101;      //srl
                4'b1000: ALUC =3'b011;      //xor
                default: ALUC =3'bx;
            endcase
        2'b11:
            case(Fun3)
                3'b000: ALUC =3'b010;       //addi
                3'b111: ALUC =3'b000;       //andi
                3'b110: ALUC =3'b001;       //ori
                3'b010: ALUC =3'b111;       //slti
                3'b101: ALUC =3'b101;       //srli
                3'b100: ALUC =3'b011;       //xori
                default:   ALUC =3'bx;
            endcase
        default:   ALUC =3'bx;
    endcase
end
endmodule
```

仍然不需修改

实验六或七插入源操作数使用标志

功能测试程序

- Exp07的功能测试程度手工调度
  - 本实验仅建立流水结构没有处理竞争
- 请将实验7测试程序手工插入nop
  - 数据相关插入2个nop
  - 转移指令后手入3个nop

详细参考pdf：

实验10：硬件stall去掉数据相关nop

本实验硬件直接定向并插入必要气泡，选用单周期功能测试程序：

软件插入 nop(32'h00000013) = addi zero, zero, 0

数据相关硬件插入气泡(32'h00000000) = flush zero

控制相关硬件插入气泡(32'h00002003) = lw zero, 0(zero

实验四测试程序优化片段

```
loop2:
    lw a1, 0x0(s1)          # 
    nop
    nop
    add a1, a1, a1
    nop
    nop
    add a1, a1, a1
    nop
    nop
    sw a1, 0x0(s1)          # x11输出到GPIO端口F0000000，计数器通道counter_set=00端口不变、LED=SW:  {GPIOf0
    lw a1, 0x0(s1)          # 再读GPIO端口F0000000状态
    nop
    nop
    and s8, a1, t0          # 取最高位=out0，屏蔽其余位送x14
    add s6, s6, t1          # 程序计数延时
    nop
    nop
    #beq    s8, t0, C_init  # 若硬件计数启用：C0=0，Counter通道0溢出，转计数器初始化，修改7段码显示
    beq s6, zero, C_init    # 程序计数x22=0，转计数器初始化，修改7段码显示：C_init
    nop
    nop
    nop
```

Data 2 nop

Branch 3 nop

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

设计工程：OExp10-RCSTE-RV32IPNTBF

Not Taken Branch Forwarding

◎ 基于实验11数据通路增加Forwarding通路

- 新增Forwarding通路
  - ⊙ ALU运算指令Forwarding通路
  - ⊙ Branch指令条件比较Forwarding通路

◎ 设计Forwarding Unit

- EX级ALU竞争检测通路控制： ForwardA/ ForwardB
- ID级条件码竞争检测通路控制： Forwardrs1, Forwardrs2

◎ 修改实验11阻塞流水线功能

- 修改实验11的Data_Stall控制为Load Stall
- 修改实验11冲刷流水线功能

设计要点

注：本实验在Exp11基础上实现。相当于用Forward策略优化替换Stall，工作量相当于实验10、实验11之和。

- **设计Forward检测控制**
  - 注意存在两处Forward检测控制
    - EXE级ALU寄存器操作数运算
      - EX级锁存传递用于Store写入存储器的不是EX_B，而是FEX_B
    - ID级Branch条件比较运算
- **修改流水线阻塞和冲刷清除电路**
  - Forward后仍然存在相关性冒险竞争
    - Load指令相关仍然存在冒险竞争需要阻塞流水线1clk
    - Branch条件比较时Forward通道不经过所在级流水锁存器
      - 与前续1指令相关时：条件判断需要等EX级ALU运行结果，延时会长
      - 若经过锁存器则需要与Load一样阻塞1clk
  - 冲刷流水线电路(Flush)：同实验11
- **流水级命名(实现方法一用)**
  - IFNF_Stage、IDNF_Stage、EXENF_Stage、MEMNS_Stage、WB_Stage

设计要点：功能测试代码. coe

- **32位指令存储器：**
  - SWORD实验平台 ROM用Distributed Memory
- **ROM初始化文件(RISCV-DEMO9.coe)**
  - 功能测试程序，其功能与实验四-六完全相同
    - 相关性非常强，用于测试流水竞争非常适用

```
memory_initialization_radix=16;
memory_initialization_vector=
0200006F,00000033,00000033,00000033,00000033,00000033,00000033,00000033,00C02283,00502333,
006303B3,00638E33,00738733,01CE02B3,005282B3,01C28EB3,01DE8F33,01EF0F33,01CF0433,01EF0F33,
01EF0F33,01DF0FB3,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF04B3,01E4E633,00948933,012902B3,005282B3,
406006B3,00C4A223,0004A583,00B585B3,00B585B3,00B4A023,006A8AB3,01592023,01402B03,0004A583,
00B585B3,00B585B3,00B4A023,0004A583,0055FC33,006B0B33,040B0E63,0004A583,00E70BB3,017B8CB3,
019B8BB3,0175FC33,000C0C63,037C0463,00E70BB3,037C0663,01592023,FB9FF06F,00D78463,0080006F,
00D687B3,00F92023,FA5FF06F,0609AA83,01592023,F99FF06F,0209AA83,01592023,F8DFF06F,01402B03,
00F787B3,0067E7B3,00E989B3,0089F9B3,006A8AB3,00DA8463,00C0006F,00E00AB3,006A8AB3,0004A583,
00B58C33,018C0C33,0184A023,00C4A223,F6DFF06F;
```

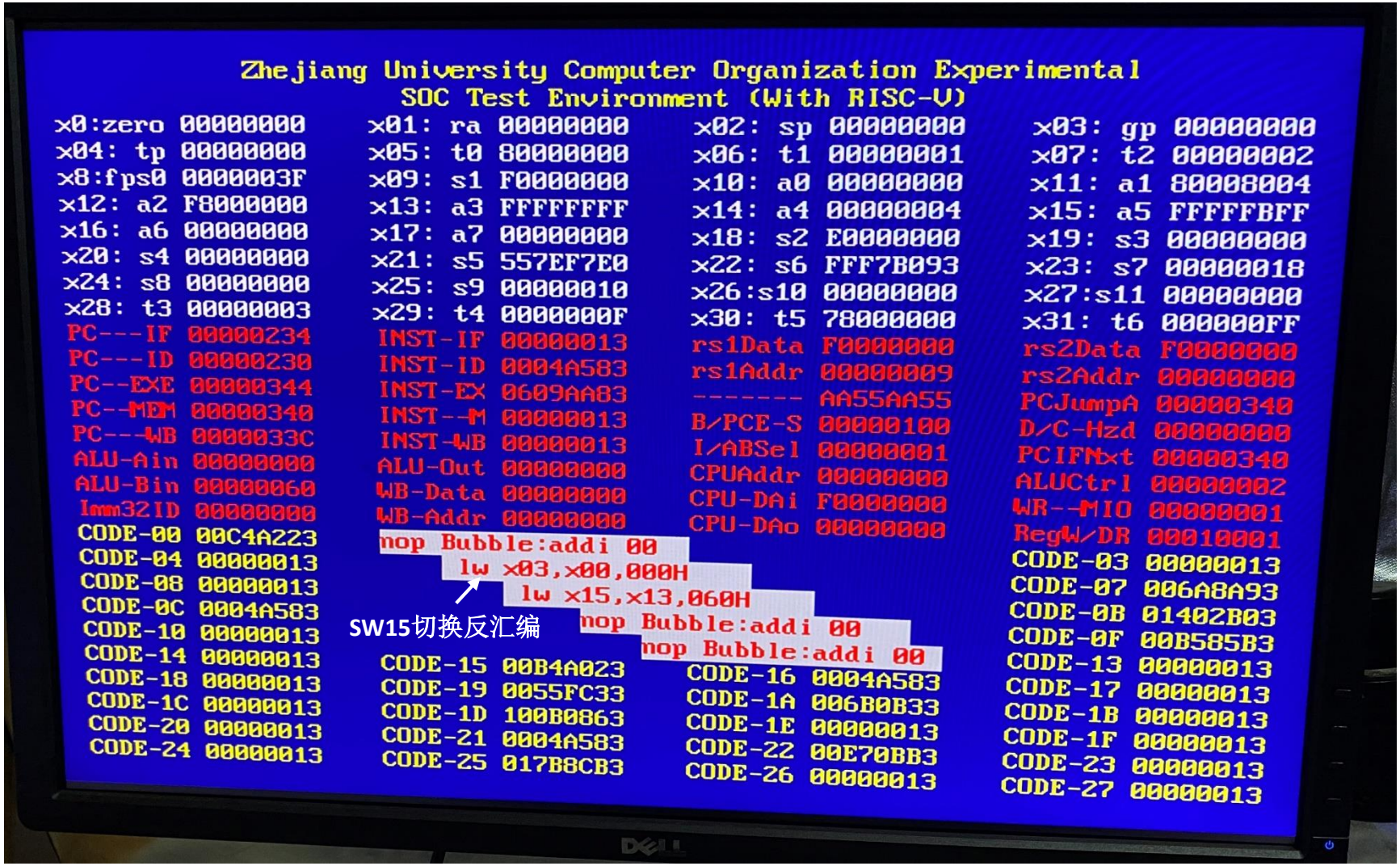
设计要点：数据存储器模块测试

- **32位数据存储器模块**
  - 7段码显示器的地址是E0000000/FFFFFFE0
  - LED显示地址是F0000000/FFFFFF00
  - 请设计存储器模块测试程序
    - 测试结果在7段显示器上指示
- **RAM初始化数据同OExp05/06**

```
memory_initialization_radix=16;
memory_initialization_vector=
f0000000, 000002AB, 80000000, 0000003F, 00000001, FFF70000, 0000FFFF, 80000000,
00000000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777,
88888888, 99999999, aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, eeeeeeee, ffffffff,
557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9,
FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF,
03bdf020, 03def820, 08002300;
```

RAM初始化数据。红色数据为七段LED图形

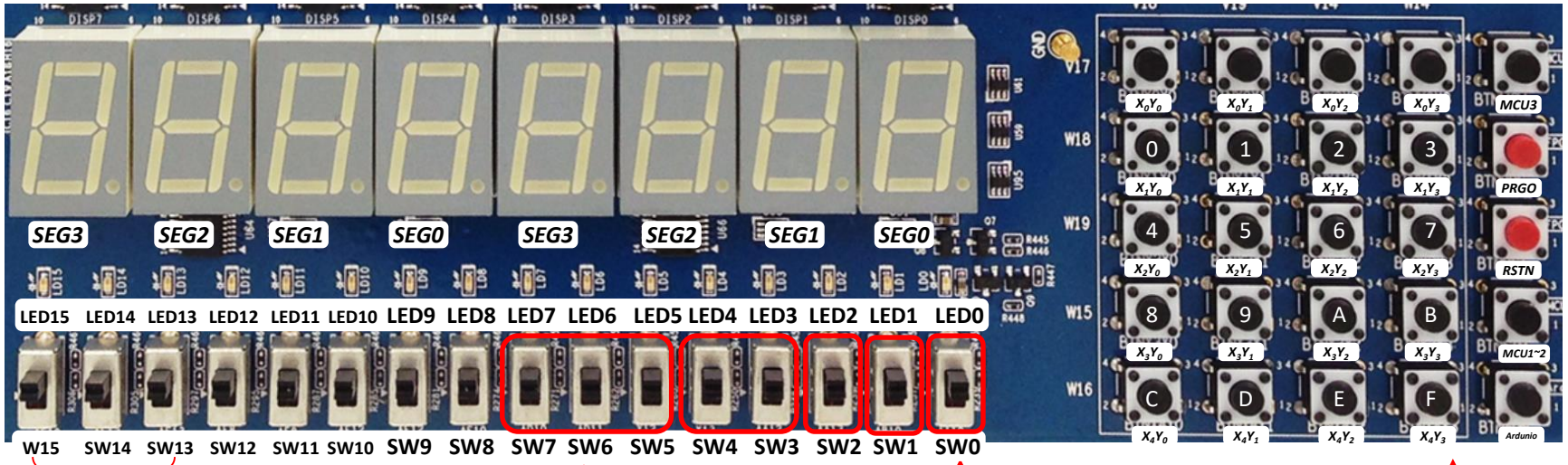
VGA_TESTP增加功能



物理验证

- 使用**RISCV-DEMO9**测数据通路功能
  - DEMO单步调试数据通路
    - 建议增加手动单步功能方便相关性调试
    - 保留原有自动单步，在Clkdiv接入Pulse信号(Arraykeys)
      - 选用SW8切换或控制
    - 结合SW15切换连续5条指令的反汇编
  - DEMO接口功能
    - SW[7:5]=000， SW[2]=0(全速运行)
      - SW[4:3]=00， SW[0]=0，点阵显示程序：跑马灯
      - SW[4:3]=00， SW[0]=0，点阵显示程序：矩形变幻
      - SW[4:3]=01， SW[0]=1，内存数据显示程序：0～F
      - SW[4:3]=10， SW[0]=1，当前寄存器s5(x21)+1显示

设计要点：物理验证接口（详细参见实验二）



SW[13]=0选择测试ROM

SW[]13=1选择测试RAM

SW[14]=0/1测试数据翻页

SW[15]=1显示5条反汇编

SW[7:5]=显示通道选择
SW[7:5]=000：CPU程序运行输出
SW[7:5]=001：测试PC字地址
SW[7:5]=010：测试指令字
SW[7:5]=011：测试计数器
SW[7:5]=100：测试RAM地址
SW[7:5]=101：测试CPU数据输出
SW[7:5]=110：测试CPU数据输入

SW[0]=文本图形选择

SW[1]=高低16位选择

没有使用

SW[2]=CPU单步时钟选择

SW[4:3]=00，点阵显示程序：跑马灯
SW[4:3]=00，点阵显示程序：矩形变幻
SW[4:3]=01，内存数据显示程序：0～F
SW[4:3]=10，当前寄存器+1显示(用户扩展保留)

下载验证流水处理器

- **非IP核仿真**
  - 对自己设计的模块做时序仿真(单周期时仿真过的略)
  - 第三方IP核不做仿真(固核无法做仿真)
  - 流水结构不难，但时序紧凑仿真可以减少大量时间
- **SOC物理验证**
  - 下载流文件.bit
  - 验证调试SOC功能
    - 功能不正确时排查错误
  - 定性观测SOC关键信号
    - 本实验只要求定性观测，功能执行正确

思考题

- 扩展下列指令，Forward检测和通路控制有什么不同：

R-Type：	sra, sll, sltu；
I-Type：	addi, andi, ori, xori, lui, slti , srai, slli, sltiu
B-Type：	bne, blt；
UJ-Type：	Jal；
U-Type：	lui；

- 针对Branch指令, 本PPT采用的Forward通路是否合理？
- 针对本PPT的EX级ForwardB通路是否有更好的优化结构？
- Branch指令条件判断**(Btaken)**，本PPT的设计描述是否合理？若存在问题，怎么改进？


END