实验 2 — 暂停与 Forwarding

姓名：汤尧 学号：3200106252 学院：计算机学院

课程名称：计算机系统II 同组学生姓名：无

实验时间：周四 3，4，5 节课 实验地点：紫金港机房 指导老师：卢立、申文博

一、 实验目的和要求

1. 深入理解流水线冒险，掌握利用 stall 处理流水线冒险，保证流水功能正确性的基本技术
2. 完善流水线的基本功能，实现 Forwarding 机制
3. 尝试思考改进的方案

二、 实验内容和原理

2.1 实验内容

1-1 在 lab1-2 的基础上加入 stall 机制。
1-2 在 lab2-1 的基础上加入 Forwarding 机制。

2.2 设计模块

2.2.1stall 实现分析

以数据冲突为例进行分析，常见的数据相关有 WAW, WAR, RAW ，由于我们的指令都是顺序执行的，这里只考虑 RAW 一种情况即可。当指令流入 ID 阶段时，我们对指令进行译码，根据译码的结果读取对应寄存器的值，此时如果发现待读取的寄存器发生了 RAW 冲突，就要触发 stall 机制进行处理。

我们不妨假设 ID 段要读取的寄存器设与 EX 段将要写入的寄存器发生了冲突。（与 MEM, WB 段冲突的情况请同学们自行考虑）那么从逻辑上我们的 stall 机制应当使得 ID 段的指令被锁住， 直到寄存器的值可以被正确读取前都不应当向后流入 EX 段，否则读取的寄存器值错误，会影响指令执行的正确性。例如相邻的两条指令为 addi x1, x0, 1; add x2, x1, x0 第二条指令执行时

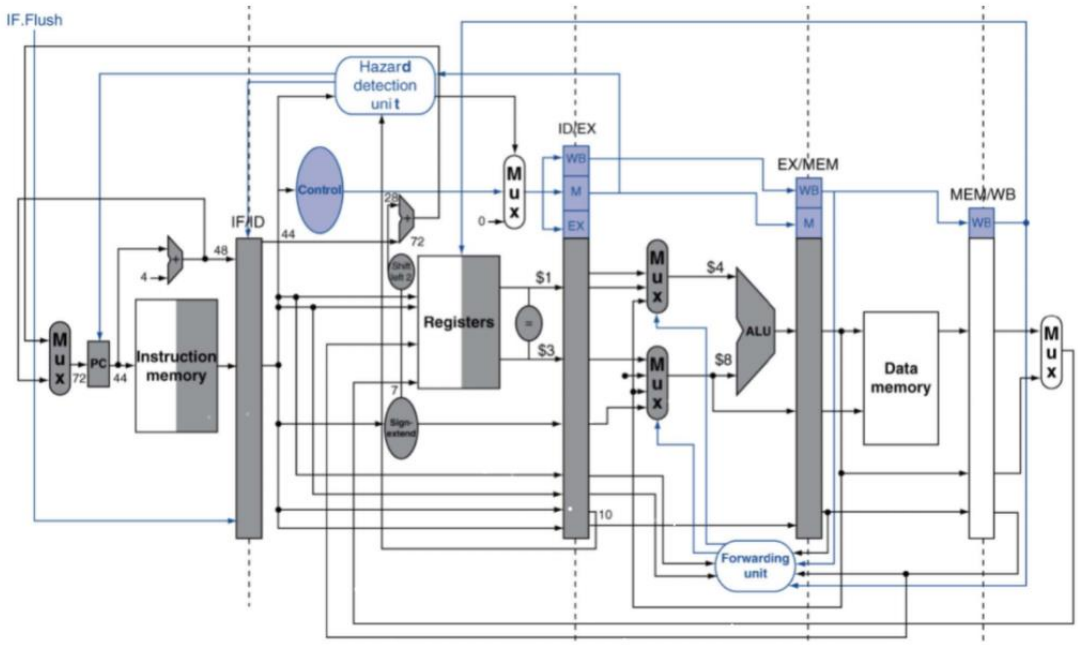
x1 寄存器的值应该为 1，而在流水线中上一条指令还并没有执行完，如果不添加 stall 机制，读取到的值为第一条指令尚未执行时 x1 寄存器的值。

也就是说接下来的一个周期，该指令仍然锁住在 ID 阶段，直到与之冲突的 EX 阶段指令流入到 WB 段完成写回，使得该指令可以读取到正确的寄存器的值时，该指令才可以流入到 EX 段。那么我们以 ID 阶段来看，向前看 IF, ID 两阶段都应当锁住当前的指令和状态，等待若干周期后继续执行，向后看 EX, MEM, WB 阶段应当继续执行段内的指令，正常的让指令流入下一阶。

再考虑分支跳转指令的情况，当执行到分支跳转指令时，由于可能发生跳转，在 pc 置位前已经流入 IF 段的指令可能并不是我们将要执行的指令， stall 机制需要考虑流水线内指令的锁存和刷新两方面的问题，对于分支跳转指令，根据同学们实现的流水线不同，也需要 stall 不同的周期数。

2.2.2 forwarding 介绍

在流水线的运行过程中，数据冲突是拖累流水线效率的一个重要的因素。在顺序单发射处理器中，数据冒险只需要考虑 RAW 一种情况，本质上就是由读写同一个寄存器所导致的，所以数据冒险本身的检测只需对流水线中相应流水段中指令的寄存器编号进行比较，即可对数据冒险的发生进行判断。



实现 Forwarding 的基本功能需要在流水线的基础上添加冲突检测逻辑和相应的数据通路，这样就能够在 WB 阶段之前，将最新的寄存器的数据回传给后续的指令，减少 stall 的周期数。

三、 主要仪器设备

- HDL：Verilog、SystemVerilog
- IDE：Vivado
- 开发板：NEXYS A7

四、 操作方法与实验步骤

4.1 操作方法

完成 Stall 机制和 Forwarding 机制设计，进行仿真测试检验 CPU 基本功能。

4.2 实验步骤

1 使用开发工具建立工程，推荐在 Vivado 2019.2 以上版本完成。

2 实现 stall 机制和 Fowarding 机制。

3 进行仿真测试，以检验 CPU 基本功能。

4 进行上板测试，以检验 CPU 设计规范，上板测试调试工具相关内容请参考 lab0。

五、 实验结果与分析

5.1 主要代码片段

Stall 模块

需要判断以下几点：

1、 是否存在 hazard 冲突。

2、 是否为 id 指令，因为只有 ld 指令的 hazard 冲突通过前递无法解决

3、 是否会因为跳转预测失败。在实际代码中，如果 ld 指令在跳转指令后，跳转指令需要在 mem 阶段写回，则先满足跳转指令，因为跳转指令如果执行那么 ld 指令将会被 flush 清空。（这里未判断指令是否跳转，实际应当判断 bne 和 beq 的执行不成功问题）并且插入 bubble 会干扰跳转指令下一条 pc 的地址。

```
reg [31:0] bubble;
wire hazard, pc_valid, stall_valid;

assign bubble_stop = (bubble==32'b0)?1'b0:1'b1;

assign pc_valid = (mem_pc_src==2'b00)? 1'b1:1'b0;
assign hazard = (id_read_reg_1==ex_write_register)||(id_read_reg_2==ex_write_register);
assign  stall_valid = (pc_valid && (ex_is_ld && hazard))? 1'b1:1'b0;

always @(posedge clk) begin
    if(stall_valid)  bubble <= 1;
    else if(bubble) bubble <= bubble - 1;
    else bubble<= 32'b0;
end
```

Forwarding 模块

需要满足以下几点：

1、产生数据冲突，即 mem/wb 段要写入的寄存器和 ex 段需要使用的寄存器（id 阶段读取的寄存器）相同。

2、wb/mem 阶段是否需要为插入 bubble 后的无效指令，若为 bubble 则不执行。

```
    // 2'b00 表示ALU的第一个操作数来自寄存器, 2'b01 表示数据来自mem冒险
    // 2'b10表示数据来自mem的alures即ex阶段冒险, 2'b11
    wire mem_forA, wb_forA;
    wire mem_forB, wb_forB;
    wire wb_valid,ex_valid, mem_valid;

    assign wb_valid = ((!wb_bubble_clear) && wb_reg_write) ? 1'b1:1'b0;
    assign ex_valid = (ex_reg1_read && (!ex_bubble_clear)) ? 1'b1:1'b0;
    assign mem_valid = ((!mem_bubble_clear) && mem_reg_write) ? 1'b1:1'b0;

    assign mem_forA = (ex_valid && mem_valid && ex_read_reg_1 == mem_write_reg)? 1'b1:1'b0;
    assign wb_forA = ( ex_valid && wb_valid && (ex_read_reg_1 == wb_write_reg))? 1'b1:1'b0;
    assign forwardA = mem_forA ? 2'b10:(wb_forA ? 2'b01 : 2'b00);

    assign mem_forB = (ex_valid && mem_valid && ex_read_reg_2 == mem_write_reg)? 1'b1:1'b0;
    assign wb_forB = (ex_valid && wb_valid && ex_read_reg_2 == wb_write_reg)? 1'b1:1'b0;
    assign forwardB = mem_forB ? 2'b10:( wb_forB ? 2'b01: 2'b00);

```

SCPU 中 forwarding 选择器模块

```
253  mux2 forwardMuxA(
254      // 2'b00 表示ALU的第一个操作数来自寄存器, 2'b01 表示数据来自mem冒险
255      // 2'b10表示数据来自mem的alures即ex阶段冒险, 2'b11
256      .s(forwardA),
257      .I0(ex_read_data_1),//
258      .I1(wb_write_data),
259      .I2(mem_ALUres),
260      .I3(wb_read_data),
261      .o(aluMux1)
262  );
263  mux2 forwardMuxB(
264      .s(forwardB),
265      .I0(ex_read_data_2),//alures
266      .I1(wb_write_data),
267      .I2(mem_ALUres),
268      .I3(wb_read_data),
269      .o(aluMux2)
270  );
271     assign  ex_mux1_out = (ex_alu_src)? ex_extend : aluMux2;
```

PCmux 中如果跳转则发送清空 ex，mem 寄存器中信号值。

```
42      always @(*)
43      case(s)
44      2'b00: begin o <= I0;
45          if(clear!=0) clear=clear-1;
46          else clear <= 2'b00;
47      end
48      2'b01: begin o <= I1; clear <= 2'b10; end
49      2'b10: begin o <= I2; clear <= 2'b10; end
50      2'b11: begin
51           if(b_byte ^ zero == 1) begin o <= I3; clear <= 2'b10; end
52           else o <= I0;
53          end
```

状态寄存器收到 bubble 信号时什么都不做，收到 clear 信号时清空信号。

```
87      else begin
88        if(bubble_stop) begin
89                  // id_pc <= 32'b0;
90         end
91         else if(jmp_clear ) begin
92            ex_pc = id_pc;
93            ex_extend = 32'b0;
94            ex_read_data_1 <= 32'b0;
95            ex_read_data_2 <= 32'b0;
96            ex_write_register <= 32'b0;
97            ex_pc_src <= 2'b00;
98            ex_reg_write = 1'b0;
```

跳转模块

pc 控制模块，遇到跳转指令时先预测不跳转，pc 顺序读之后的指令。如果跳转指令成立，需要跳转，则发送清空指令，将后续信号全部清空。

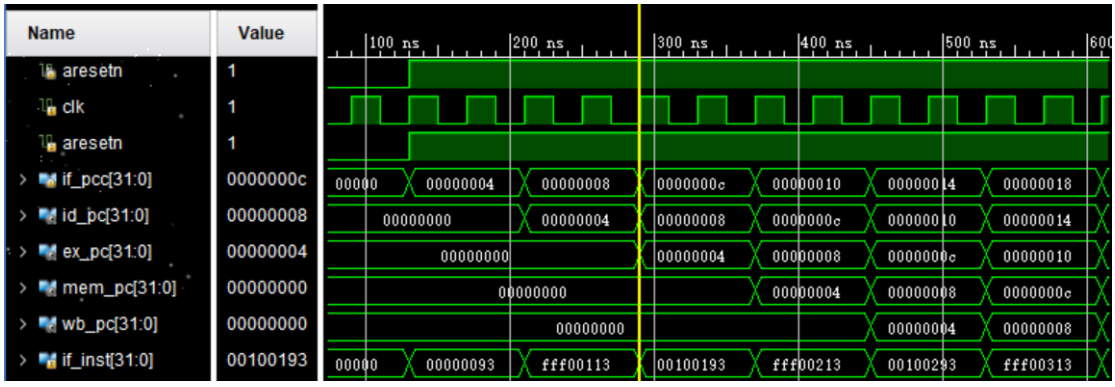
```
36  // 2'b00 表示pc的数据来自pc+4, 2'b01 表示数据来自JALR跳转地址,
37  // 2'b10表示数据来自JAL跳转地址(包括branch).
38  //branch 跳转根据条件决定,2'b11表示bne，看不为0就跳
39   assign clear_out = (clear==2'b10)?1'b1:1'b0;
40      always @(*)
41      case(s)
42      2'b00: begin o <= I0;
43          if(clear!=0) clear=clear-1;
44          else clear <= 2'b00;
45      end
46      2'b01: begin o <= I1; clear <= 2'b10; end
47      2'b10: begin o <= I2; clear <= 2'b10; end
48      2'b11: begin
49           if(b_byte ^ zero == 1) begin o <= I3; clear <= 2'b10; end
50           else o <= I0;
51          end
52      endcase
53
54  endmodule
```

Register 模块采用 double dump，上升沿写入下降沿读出，这样就不需要解决 wb 冲突。

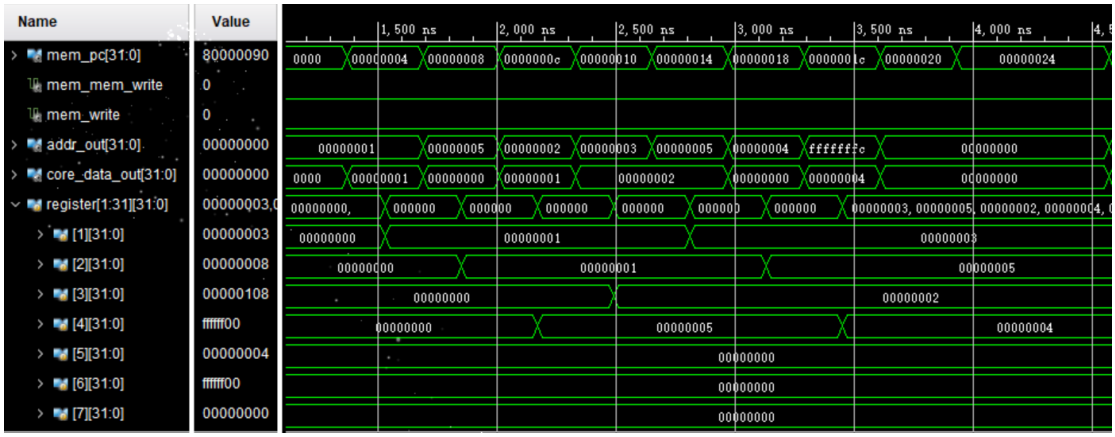
```
36  assign read_data_1 = (read_addr_1 == 0) ? 0 : register[read_addr_1]; // read  :
37  assign read_data_2 = (read_addr_2 == 0) ? 0 : register[read_addr_2]; // read
38
39  always @(negedge clk or posedge rst) begin //(negedge clk or posedge rst)
40  if (rst == 1) for (i = 1; i < 32; i = i + 1) register[i] <= 0; // reset
41  else if (we == 1 && write_addr != 0) register[write_addr] <= write_data;
42  end
43
```

5.2 仿真模拟

不同模块的 pc 随时钟周期依次传递。



使用测试代码，寄存器变化和预想结果一致。



六、 思考题

1-1 Stall 暂停

1. 请你对数据冲突情况进行分析归纳，试着将他们分类列出。
   a) 上一条指令要修改寄存器，下一条指令要读取寄存器，下一条指令读取时上一条指令还未修改成功。
   b) 上一条指令需要从内存中读取数据放入寄存器，下一条指令需要读取该寄存器的值，此时上一条指令还未读出。
2. 如果 EX, MEM, WB 段中不止一个段的写寄存器与 ID 段的读寄存器发生了冲突，该如何处理？

forward 是有优先级的，先就近判断 mem 段是否需要写寄存器寄存器，再判断 wb 段。因为有可能在 wb 段写回的寄存器在 mem 段继续被改写。

1-2 Forwarding 前递

1. 在引入 Forwarding 机制后，是否意味着 stall 机制就不再需要了？为什么？

不是。数据前递不能解决的冲突：load 指令，必须通过 stall 机制插入一个 bubble，在 mem 阶段后半周期读出之后，进入 wb 阶段，通过数据前递，将要写回的值给 ex 阶段。

2. 你认为 Forwarding 机制在实际的电路设计中是否存在一定的弊端？如果存在，请给出你的理由。

不存在。