

课程名称:	计算机体系结构
姓名:	杨吉祥
学院:	计算机科学与技术学院
系:	竺可桢学院图灵班
专业:	计算机科学与技术
学号:	3230106222
指导教师:	常瑞

一、设计思路

• 如果存在结构或 WAW 危险性,则指令发射会停滞,直到这些危险性消除为止。判断结构冲突就是看当前指令使用的功能单元是否为空且是否正在被使用,判断WAW就是看该指令要写入的目的寄存器是否正在被其他功能单元写入。两个有一个发生就需要停滞指令发射。

```
// normal stall: structural hazard or WAW
   assign normal_stall = (use_FU != `FU_BLANK & FUS[use_FU][`BUSY]) | RRS[dst]
!= 3'b0;   //fill sth. here
```

• 该部分确保功能部件不会写入目的寄存器当之前指令还没读取该寄存器。所以如果其他功能部件的源寄存器不等于该功能部件的目的寄存器的话,就肯定不会发生WAR,如果等于的话,就要确保其他功能部件的源寄存器已经读取完毕。其他功能部件的WAR逻辑与此类似。

```
// ensure WAR:
   // An FU should not write its results to dst if
   // a previous inst hasn't read that register.
   // *_WAR should be driven high when the FU is allowed to write.
   // i.e. there is no WAR or when WAR clears
   wire ALU_WAR = (
       (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_MEM][`RDY1]) & //fill sth. here
       (FUS[`FU_MEM][`SRC2_H: `SRC2_L] != FUS[`FU_ALU][`DST_H: `DST_L] |
!FUS[`FU_MEM][`RDY2]) & //fill sth. here
       (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_MUL][`RDY1]) & //fill sth. here
       (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_MUL][`RDY2]) & //fill sth. here
       (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_DIV][`RDY1]) & //fill sth. here
       (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_DIV][`RDY2]) & //fill sth. here
       (FUS[`FU_JUMP][`SRC1_H: `SRC1_L] != FUS[`FU_ALU][`DST_H: `DST_L] |
!FUS[`FU_JUMP][`RDY1]) & //fill sth. here
       (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] |
!FUS[`FU_JUMP][`RDY2]) //fill sth. here
   );
```

• 该部分对 FUS 和 RRS 寄存器进行一个写入操作,将对应字段的值写入即可。

```
if (RO_en) begin
    // not busy, no waw, write info to FUS and RRS
    if (|dst) RRS[dst] <= use_FU;
    FUS[use_FU][`BUSY] <= 1'b1;
    FUS[use_FU][`OP_H:`OP_L] <= op;
    FUS[use_FU][`DST_H:`DST_L] <= dst;
    FUS[use_FU][`SRC1_H:`SRC1_L] <= src1;
    FUS[use_FU][`SRC2_H:`SRC2_L] <= src2;
    FUS[use_FU][`FU1_H:`FU1_L] <= fu1;
    FUS[use_FU][`FU2_H:`FU2_L] <= fu2;
    FUS[use_FU][`RDY1] <= rdy1;</pre>
```

```
FUS[use_FU][`RDY2] <= rdy2;
FUS[use_FU][`FU_DONE] <= 1'b0;//fill sth. here.

IMM[use_FU] <= imm;
PCR[use_FU] <= PC;
end</pre>
```

• 该部分解决RAW危险,当功能部件的两个源寄存器的值都已经准备好了之后,才进行读操作,并且将该功能部件的准备信号以及操作数来源清零。其他功能部件的读取逻辑与此类似。

• EX部分为功能单元的执行,执行完成后,将对应功能部件的完成信号置为1。

```
// EX
             if(ALU_done)begin
                  FUS[`FU_ALU][`FU_DONE] <= 1'b1;</pre>
             end
             if(MEM_done)begin
                  FUS[`FU_MEM][`FU_DONE] <= 1'b1;</pre>
             end
             if(MUL_done)begin
                  FUS[`FU_MUL][`FU_DONE] <= 1'b1;</pre>
             end
             if(DIV_done)begin
                  FUS[`FU_DIV][`FU_DONE] <= 1'b1;</pre>
             end
             if(JUMP_done)begin
                  FUS[`FU_JUMP][`FU_DONE] <= 1'b1;</pre>
             end
             //fill sth. here
```

WB部分为写回部分,当功能部件已经执行完毕且不会发生WAR时进行写回。由于是乱序执行,为了避免同一时间WB阶段对多条指令进行写回操作,所以不同功能器件之间的写回用 if else 进行避免。
 为了避免RAW危险的发生,所以如果有其他功能部件的源寄存器为该功能部件的话,需要将其他功能部件的准备信号置为1,操作数来源置为0。

```
// WB

if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
    FUS[`FU_JUMP] <= 32'b0;

RRS[FUS[`FU_JUMP][`DST_H:`DST_L]] <= 3'b0;

// ensure RAW

if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_JUMP) begin
    FUS[`FU_ALU][`RDY1] <= 1'b1;
    FUS[`FU_ALU][`FU1_H:`FU1_L] <= 3'b0;</pre>
```

```
end
    if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_JUMP) begin
        FUS[`FU_MEM][`RDY1] <= 1'b1;</pre>
        FUS[`FU_MEM][`FU1_H: `FU1_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_JUMP) begin
        FUS[`FU_MUL][`RDY1] <= 1'b1;</pre>
        FUS[`FU_MUL][`FU1_H: `FU1_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_DIV][`FU1_H: `FU1_L] == `FU_JUMP) begin
        FUS[`FU_DIV][`RDY1] <= 1'b1;</pre>
        FUS[`FU_DIV][`FU1_H: `FU1_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_JUMP) begin
        FUS[`FU_ALU][`RDY2] <= 1'b1;</pre>
        FUS[`FU_ALU][`FU2_H: FU2_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_JUMP) begin
        FUS[`FU_MEM][`RDY2] <= 1'b1;</pre>
        FUS[`FU_MEM][`FU2_H: FU2_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_JUMP) begin
        FUS[`FU_MUL][`RDY2] <= 1'b1;</pre>
        FUS[`FU_MUL][`FU2_H: FU2_L] <= 3'b0;</pre>
    end
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_JUMP) begin
        FUS[`FU_DIV][`RDY2] <= 1'b1;</pre>
        FUS[`FU_DIV][`FU2_H: FU2_L] <= 3'b0;</pre>
    end
end//fill sth. here
```

二、思考题

1.

<u>__start:</u>

```
addi x0, x0, 0

lw x2, 4(x0) # x2 = 0x8, clk_cnt = 0x7

lw x4, 8(x0) # x4 = 0x10, clk_cnt = 0xc

add x1, x2, x4 # x1 = 0x18, clk_cnt = 0xf

addi x1, x1, -1 # x1 = 0x17, clk_cnt = 0x13

lw x5, 12(x0) # x5 = 0x14, clk_cnt = 0x15

lw x6, 16(x0) # x6 = 0xffff0000, clk_cnt = 0x1A

lw x7, 20(x0) # x7 = 0x0fff0000, clk_cnt = 0x20

sub x1,x4,x2 # x1 = 0x8, clk_cnt = 0x1f

addi x1,x10,-3 # x1 = 0xfffffffd, clk_cnt = 0x23

beq x4,x5,label0 # not jump
```

1⊌ debug_clk	0			9,000	. 000 15		20,000,000	/ 5	10, 50	0.000 /15			, s		V. VVV. 15		12,000,000	i e e e e e
₩ rst	0																	
> W PC_IF[31:0]	0000000c	00000020 \ 00000024 \ 0000			00028		0000002e	0002e		00000030		00000034		00000040	00000044 00000048		0048	
∨ ™ WB																		
> W rd_ctrl[4:0]	00	00				01	07	0	0	01					10			
> W DatatoReg_ctrl[2:0]	0	0					1		-	4			. 0		4	1) 0		
> wt_data_WB[31:0]	00000000	00000017			00000008	01110000	0000	0008 / fffffffd		0000002c	fffffffd		00000030		ffffffd			
¼ clk	0				innannní		<u> </u>	1111111111		(111111111		innanna	(1111111111	(0000000	1111111111	d a a a a a a a a a a a a a a a a a a a		
¼ rstn	1																	
> Winst_IF[31:0]	004100b3	402200bS ffd50093 00520e63						00420a63			0000	0013	00004067			00c000ef	00000	0013
> W PC_IS[31:0]	80000000	0000001c 00000020 000000			0024		00000028	0000002c				00000030			00000040	0000044		
> Winst_IS[31:0]	00802203	01402383	01402383 402200b3 ffd5			0093		00520663	00420a63				00000013			00004067	00c000ef	
> W rs1_addr_ctrl[4:0]	00	0	00 / 04 /			00		0a	04		00		04			00		
> W rs1_data_RO[31:0]	00000000	00000000 00000010			00000000			00000010 00000000			00000010			00000000				
> W rs2_addr_ctrl[4:0]	00	00 02				00			05		00	00				00		
> W rs2_data_RO[31:0]	00000000	0000	0000	00000008	00000000			00000014		00000000		00000010	Χ		00000000			
% FU_mem_EN	0																	
la RegWrite_ctrl	0																	
> 😻 [1][31:0]	00000000	00000017					00000008					fffffffd						
> 😻 [2][31:0]	00000000	0000008																
> ▶ [4][31:0]	00000000	0000010																
18 FU_mul_EN	0																	
18 FU_div_EN	0																	
18 FU_mul_finish	0																	
% FU_div_finish	0																	
> MEM																		
> MI STALL																		
11.50 51 15	1																	

可以看到0x1C对应的指令为 1w x7, 20(x0),0x20对应的指令为 sub x1,x4,x2,正常应该x7的写入发生在x1的写入前面,但是可以看到仿真图中x1写入0x8发生在x7写入0x0fff0000前面,说明此处发生乱序执行。

2.

本学期的体系实验基本就结束了,在此感谢老师助教们的辛苦付出。一开始做体系实验还能用到上学期计组的实验,觉得挺轻松,结果到后面发现直接去填空反而更加轻松,工作量与上学期的计组相比少了不是一点,真的太感谢老师助教们了。我感觉现在体系实验这样子就挺好的,没有什么建议。