浙江大学

计算机体系结构实验报告

课程名称：	计算机体系结构
姓　　名：	周楠
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
学　　号：	3220102535
指导教师：	常瑞

2024 年 11 月 27 日

浙江大学实验报告

课程名称：	计算机体系结构	实验类型：	综合		
实验项目名称：	实验 5 - 使用 Scoreboard 的乱序流水线设计				
学生姓名：	周楠	专业	计算机科学与技术	学号：	3220102535
实验地点：	玉泉实验室	实验时间：	2024.10.22	指导教师：	常瑞

一. 操作方法与实验步骤

1.1 生成一个乘法器和除法器 IP 核

1. 首先，从左侧导航栏打开”IP Catalog”。
2. 乘法器
   (a) 搜索”multiplier” 并从”Math Functions” 中选择”Multiplier”。
   (b) 自定义”Basic” 如下：
   (c) 自定义”Output and Control” 如下：
3. 除法器
   (a) 搜索”divider” 并从”Math Functions” 中选择”Divider Generator”。
   (b) 自定义”Channel Settings” 如下：
   (c) 自定义 Channel Settings

1.2 Scoreboard 算法

1. normal_stall 控制信号：是流水线中的一个阻塞 (stall) 信号，用于处理结构冒险 (Structural Hazard) 和写后写 (Write After Write, WAW) 冒险

```
1    assign normal_stall = (use_FU != `FU_BLANK && FUS[use_FU][
        `BUSY]) | (|dst && |RRS[dst]);
```

- (use_FU != FU_BLANK && FUS[use_FU][BUSY]) 部分：检查当前指令需要使用的功能单元是否被占用，如果被占用则阻塞。对应的是结构冒险。

- (|dst && |RRS[dst]) 部分：|RRS[dst] 检查目标寄存器的保留站 (Register Reservation Station) 是否已经被占用。也就是检查是否存在 WAW 情况，也就是是否存在两条指令同时写入同一个寄存器。如果存在则阻塞。

2. ensure WAR：检查功能单元能否写入目标寄存器。

   如果存在 WAR 情况，也就是说当前指令前面还未执行完的指令的源操作数和当前指令的目标寄存器相同，并且源操作数处于 yes 的 ready 状态，还没有读取源操作数的值，此时功能单元不能写入目标寄存器，需要等待前面的指令执行完毕。

   要想 ALU 能够写入目标寄存器，需要满足以下条件：

   - 除了 ALU 之外的其他功能单元的源操作数和目标寄存器不同
   - 或者源操作数的 ready 状态为 no。

   以下是 ALU 的 WAR 检查代码：

```
wire ALU_WAR = (
    // allow to write
    // !(FUS[`FU_MEM][`SRC1_H:`SRC1_L] == FUS[`FU_ALU][`DST_H:
       `DST_L] && FUS[`FU_MEM][`RDY1]) == 1 & //fill sth. here
    (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
);
```

3. maintain the table

- IS 阶段：如果 RO_en 控制信号为 1，表示当前指令可以被 issue，需要 read operands，更新 FUS 和 RRS。

```
    // IS
    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; //fill sth. here.
        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;
        FUS[use_FU][`RDY2] <= rdy2;
        FUS[use_FU][`FU_DONE] <= 1'b0;
        IMM[use_FU] <= imm;
        PCR[use_FU] <= PC;
    end
```

- RO 阶段：如果某个功能单元的源寄存器的状态都为 ready，表示源寄存器的值已经准备好，能够读取但还没开始读取。此时我们需要读取源寄存器的值，将源寄存器的状态都设置为 no，表示已经读取，更新源寄存器的来源都是 FU_BLANK。

  以下是 ALU 部分的 RO 阶段代码：

```
    else if (FUS[`FU_ALU][`RDY1] & FUS[`FU_ALU][`RDY2])
        begin //fill sth. here.
        // ALU
        FUS[`FU_ALU][`RDY1] <= 1'b0;
        FUS[`FU_ALU][`RDY2] <= 1'b0;
        FUS[`FU_ALU][`FU1_H:`FU1_L] <= 3'b0;
        FUS[`FU_ALU][`FU2_H:`FU2_L] <= 3'b0;
    end
```

- EX 阶段根据外部传入的 done 控制信号，判断功能单元是否已经完成计算，如果完成则将 FU_DONE 设置为 1。

```
1    if (~FUS[`FU_ALU][`FU_DONE])
2        FUS[`FU_ALU][`FU_DONE] <= ALU_done;
3    if (~FUS[`FU_MEM][`FU_DONE])
4        FUS[`FU_MEM][`FU_DONE] <= MEM_done; //fill sth. here
5    if (~FUS[`FU_MUL][`FU_DONE])
6        FUS[`FU_MUL][`FU_DONE] <= MUL_done;
7    if (~FUS[`FU_DIV][`FU_DONE])
8        FUS[`FU_DIV][`FU_DONE] <= DIV_done;
9    if (~FUS[`FU_JUMP][`FU_DONE])
10       FUS[`FU_JUMP][`FU_DONE] <= JUMP_done;
```

- WB 阶段

  执行 WB 的前提条件：

  - 当前功能单元的 FU_DONE 为 1, 表示当前功能单元已经完成计算，只剩下写回操作。
  - 不存在 WAR 情况，也就是当前功能单元能够安全写入目标寄存器。

  更新 FUS 和 RRS 的状态：

  - 执行完 WB 后，需要将当前功能单元的 busy 状态设置为 0，表示当前功能单元空闲。
  - 将目标寄存器的保留站的状态设置为 FU_BLANK，表示目标寄存器的值已经写入, 并且不再被占用。
  - 如果其他功能单元的源操作数依赖于当前功能单元的目标寄存器，需要将这些功能单元的源操作数的 ready 状态设置为 yes，表示能够读取源操作数的值。

以下是 ALU 部分的 WB 阶段代码：

```
    else if (FUS[`FU_ALU][`FU_DONE] & ALU_WAR) begin
        FUS[`FU_ALU] <= 32'b0;
        RRS[FUS[`FU_ALU][`DST_H:`DST_L]] <= 3'b0;

        // ensure RAW
        if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_ALU)
            FUS[`FU_JUMP][`RDY1] <= 1'b1; //fill sth. here
        if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_ALU)
            FUS[`FU_MEM][`RDY1] <= 1'b1; //fill sth. here
        if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_ALU)
            FUS[`FU_MUL][`RDY1] <= 1'b1; //fill sth. here
        if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_ALU)
            FUS[`FU_DIV][`RDY1] <= 1'b1; //fill sth. here

        if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_ALU)
            FUS[`FU_JUMP][`RDY2] <= 1'b1; //fill sth. here
        if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_ALU)
            FUS[`FU_MEM][`RDY2] <= 1'b1; //fill sth. here
        if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_ALU)
            FUS[`FU_MUL][`RDY2] <= 1'b1; //fill sth. here
        if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_ALU)
            FUS[`FU_DIV][`RDY2] <= 1'b1; //fill sth. here
    end
```

二. 实验结果分析

2.1 指出仿真波形中乱序发生的位置

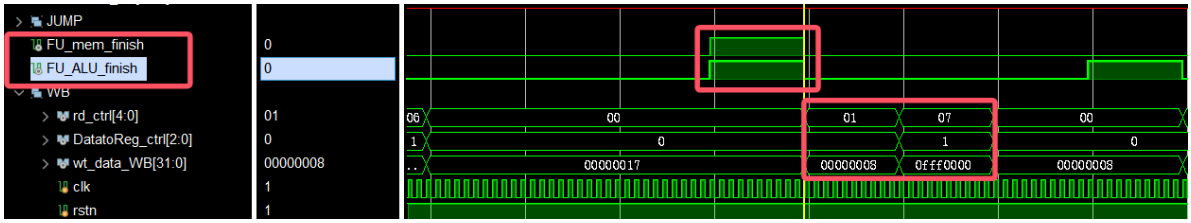

图 1: 乱序发生的位置

PC = 0x1C 对应的指令是 01402383，lw x7, 20(x0)

PC = 0x20 对应的指令是 402200B3，sub x1, x4, x2

按照顺序执行的顺序应该是 lw x7, 20(x0) -> sub x1, x4, x2，也就是说先完成 load 操作，对 x7 进行修改，然后再进行 sub 操作，对 x1 进行修改。但是在乱序执行的情况下，sub x1, x4, x2 这条指令会在 lw x7, 20(x0) 之前执行完成，也就是说 x1 的值在 x7 之前被修改.

观察上图：两条指令对应的功能单元 finish 的时间相同，但是因为每一个周期只能写入一个寄存器，所以 load 指令和 sub 指令的写入寄存器时间存在先后关系，观察 rd_ctrl 信号，可以发现先写入 x1，再写入 x7，对应的在此位置发射顺序与完成顺序不一致，因此乱序发生在这里。