浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：汪辉

学　　院：计算机科学与技术学院

系：计算机系

专　　业：计算机科学与技术

学　　号：3190105609

指导教师：陈文智

2021 年 12 月 21 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：**Dynamically Scheduled Pipelines using Scoreboarding**

学生姓名：汪辉 专业：计算机科学与技术 学号：3190105609

同组学生姓名：王嘉豪 指导老师：陈文智

实验地点：曹西 301 实验日期：2021 年 12 月 21 日

一、 实验目的和要求

- 理解支持多周期指令的流水线设计方法
- 理解 scoreboard 动态调度算法的设计原则
- 掌握支持多周期计算的流水线的设计方法
- 掌握支持 scoreboard 动态调度算法的流水线设计
- 理解掌握 scoreboard 算法的验证原则

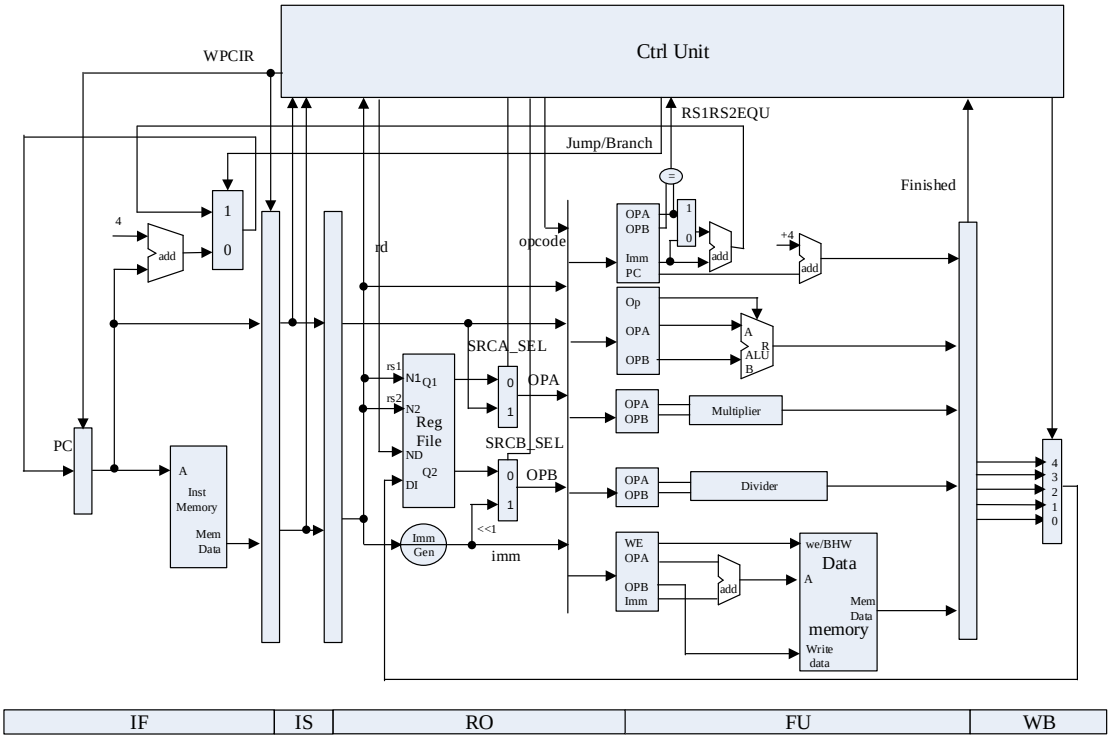
二、 实验内容和原理

1. 实验内容：

- **Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.**
- **Design of a scoreboard and integrate it to CPU.**
- **Verify the Pipelined CPU with program and observe the execution of program.**

2. 实验原理：

2.1 Pipelined CPU



多周期的运算模块在实验五中设计完成，实验六中任务主要为在 ctrlunit 中实现 scoreboard 的动态调度。其中，为了满足调度的选择，需要实现和管理下列结构：Functional Unit Status、Register Result Status。

2.2 Functional Unit Status

FUS 是 5 个 32 位数据的数组，

```
reg[31:0] FUS[1:5]; // array of 32bit_registers
```

存储了当前 JUMP、ALU、MEM、MUL、DIV 五个模块的状态。其中 32 位各个位的含义在宏中定义，参考下图：

Functional unit status			dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk

实验工程中的32位末位是一个代表unit的当前指令是否完成的位FU_DONE。

2.3 Register Result Status

RRS 存储着当前 32 个寄存器的数据来源，若是可立即获得的则 RRS 为 0，

否则为对应的功能单元，

```
// records which FU will write corresponding reg at WB
reg[2:0] RRS[0:31];
```

数组的每一项有 3 位，根据宏的内容，只用到了 6 个数据。

```
// function unit
`define FU_BLANK    3'd0
`define FU_ALU      3'd1
`define FU_MEM      3'd2
`define FU_MUL      3'd3
`define FU_DIV      3'd4
`define FU_JUMP     3'd5
```

三、 实验过程和数据记录

实验六补充完成 CtrlUnit.v 中的各部分信号处理和组合逻辑，根据 scoreboard 的算法步骤和功能，依次进行补充即可。

3.1 normal stall

normal_stall 是全流水线 stall，目的是等待特定单元完成计算以排除结构竞争和 WAW 冒险。

```
// normal stall: structural hazard or WAW
assign normal_stall = {
    // structural hazard
    FUS[`FU_ALU][`BUSY]&use_ALU |
    FUS[`FU_MEM][`BUSY]&use_MEM |
    FUS[`FU_MUL][`BUSY]&use_MUL |
    FUS[`FU_DIV][`BUSY]&use_DIV |
    FUS[`FU_JUMP][`BUSY]&use_JUMP |
    // WAW
    ( RRS[dst] != `FU_BLANK )
};          //fill sth. here
```

3.2 ensure WAR

WAR 冒险的处理有一些复杂，实验代码中，各个单元的 WAR 信号用于判断是否存在 WAR 冒险，在使用各个 WAR 时需要注意当 WAR 为 1 时表示不存在 WAR 冒险可以正常写，由于这和第一反应不同，在实现这个信号处理时更需特

别注意，比如信号中的与和或的关系。

```
// ensure WAR:
// If an FU hasn't read a register value (RO), don't write to it.
wire ALU_WAR = (
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_MEM][`SRC1_H:`SRC1_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_MEM][`RDY1] ) ) &    //fil.
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_MEM][`SRC2_H:`SRC2_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_MEM][`RDY2] ) ) &    //fil.
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_MUL][`SRC1_H:`SRC1_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_MUL][`RDY1] ) ) &    //fil.
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_MUL][`SRC2_H:`SRC2_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_MUL][`RDY2] ) ) &    //fil.
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_DIV][`SRC1_H:`SRC1_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_DIV][`RDY1] ) ) &    //fil.
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_DIV][`SRC2_H:`SRC2_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_DIV][`RDY2] ) ) &    //fil.
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_JUMP][`SRC1_H:`SRC1_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_JUMP][`RDY1] ) ) &    //f.
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] | ( FUS[`FU_JUMP][`SRC2_H:`SRC2_L] == FUS[`FU_ALU][`DST_H:`DST_L] & ~FUS[`FU_JUMP][`RDY2] ) )      //f.
);

wire MEM_WAR = (
```

3.3 Issue

满足发射条件时，进行 FUS 的初始化，并填写相应的 RRS：

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

3.3 Read Operand

Issue 之后等待操作需要的参数，当其他功能单元完成 src 的写后，rdy 位会被置 1，只有当两个源操作数的 rdy 都为 1 时，寄存器数据才被正常读取，并且为了确保不产生对后续指令的持续影响，RO 阶段只有 1 个周期，当读取出数据后，立即将 rdy 立即置 0。

```
    // RO
      if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2]) begin
                // JUMP
                FUS[`FU_JUMP][`RDY1] <= 1'b0;
                FUS[`FU_JUMP][`RDY2] <= 1'b0;
      end
```

3.4 Execute

按照一定先后顺序依次完成 5 个功能单元的运算，需要注意的是，在每个周期内，功能单元最多有一个完成，实验中设计的顺序如下：JUMP、ALU、MEM、MUL、DIV。

```
// EX
FUS[`FU_JUMP][`FU_DONE] <= JUMP_done;
FUS[`FU_ALU][`FU_DONE] <=  ( FUS[`FU_JUMP][`FU_DONE] & FUS[`FU_ALU][`FU_DONE] )
                         | ( FUS[`FU_ALU][`FU_DONE] & ~ALU_WAR )
                         | ALU_done;
FUS[`FU_MEM][`FU_DONE] <= ( ( FUS[`FU_JUMP][`FU_DONE]|FUS[`FU_ALU][`FU_DONE] ) & FUS[`FU_MEM][`FU_DONE] )
                         | ( FUS[`FU_MEM][`FU_DONE] & ~MEM_WAR )
                         | MEM_done;
FUS[`FU_MUL][`FU_DONE] <= ( ( FUS[`FU_JUMP][`FU_DONE]|FUS[`FU_ALU][`FU_DONE]|FUS[`FU_MEM][`FU_DONE] ) & FUS[`FU_MUL][`FU_DONE] )
                         | ( FUS[`FU_MUL][`FU_DONE] & ~MUL_WAR )
                         | MUL_done;
FUS[`FU_DIV][`FU_DONE] <= ( ( FUS[`FU_JUMP][`FU_DONE]|FUS[`FU_ALU][`FU_DONE]|FUS[`FU_MEM][`FU_DONE]|FUS[`FU_MUL][`FU_DONE] ) & FUS[`FU_DIV][`FU_DONE] )
                         | ( FUS[`FU_DIV][`FU_DONE] & ~DIV_WAR )
                         | DIV_done;  //fill sth. here
```

3.5 Write Back

写回数据后，之前被阻塞 RO 的指令需要正常执行，所以需要把 rdy 置 1，这样下一周期数据即可被正常读取。以 JUMP 单元为例：

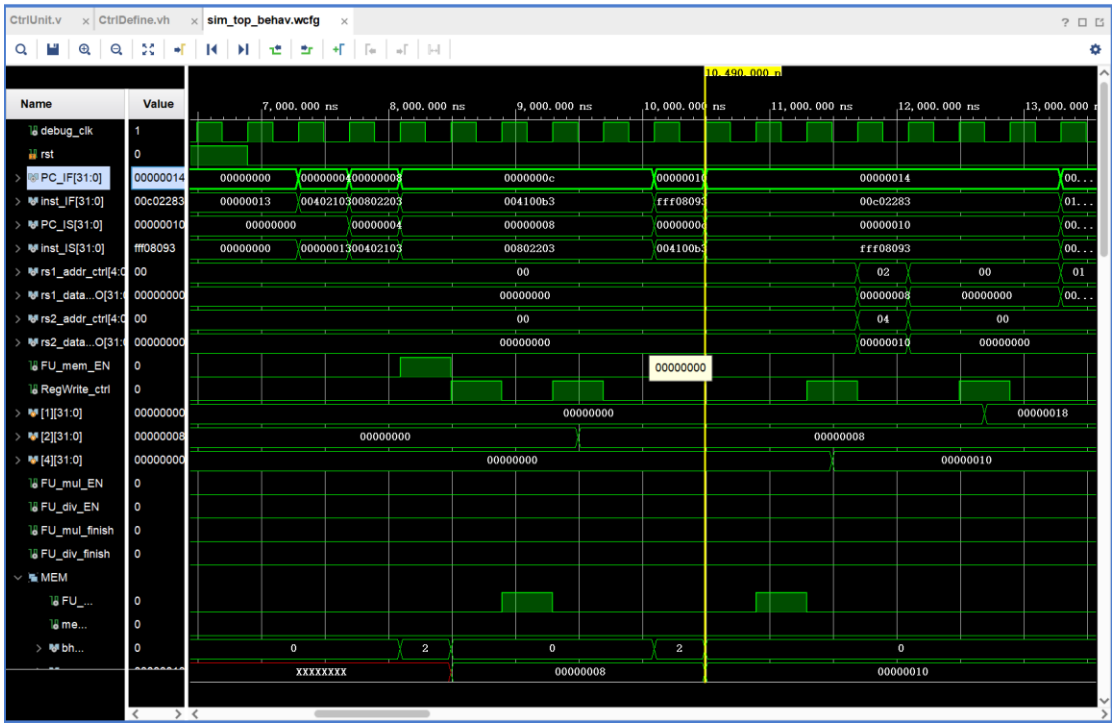
```
          /* WB */
          // JUMP
if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
      RRS[FUS[`FU_JUMP][`DST_H:`DST_L]] <=`FU_BLANK ;
      FUS[`FU_JUMP] <= 32'b0 ;
   // ensure RAW
             if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_JUMP)
          FUS[`FU_ALU][`RDY1] <= 1'b1 ; //fill sth. here
             if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_JUMP)
          FUS[`FU_MEM][`RDY1] <= 1'b1 ; //fill sth. here
             if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_JUMP)
          FUS[`FU_MUL][`RDY1] <= 1'b1 ; //fill sth. here
             if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_JUMP)
          FUS[`FU_DIV][`RDY1] <= 1'b1 ; //fill sth. here

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

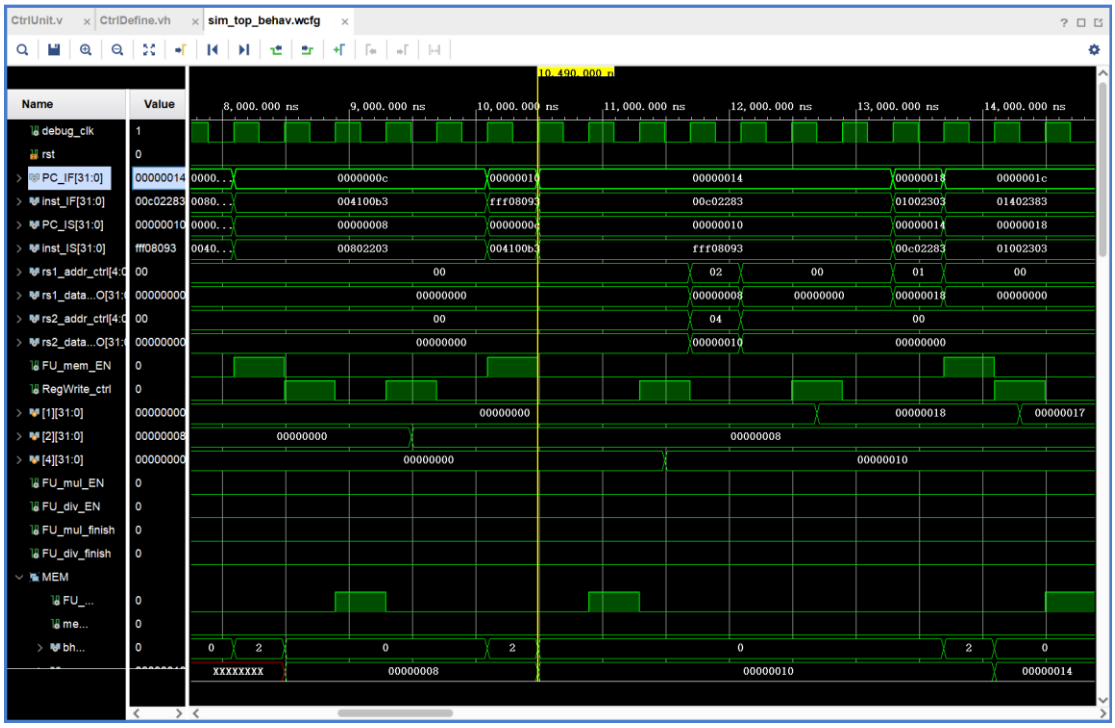
四、　实验结果分析

1. 仿真结果：

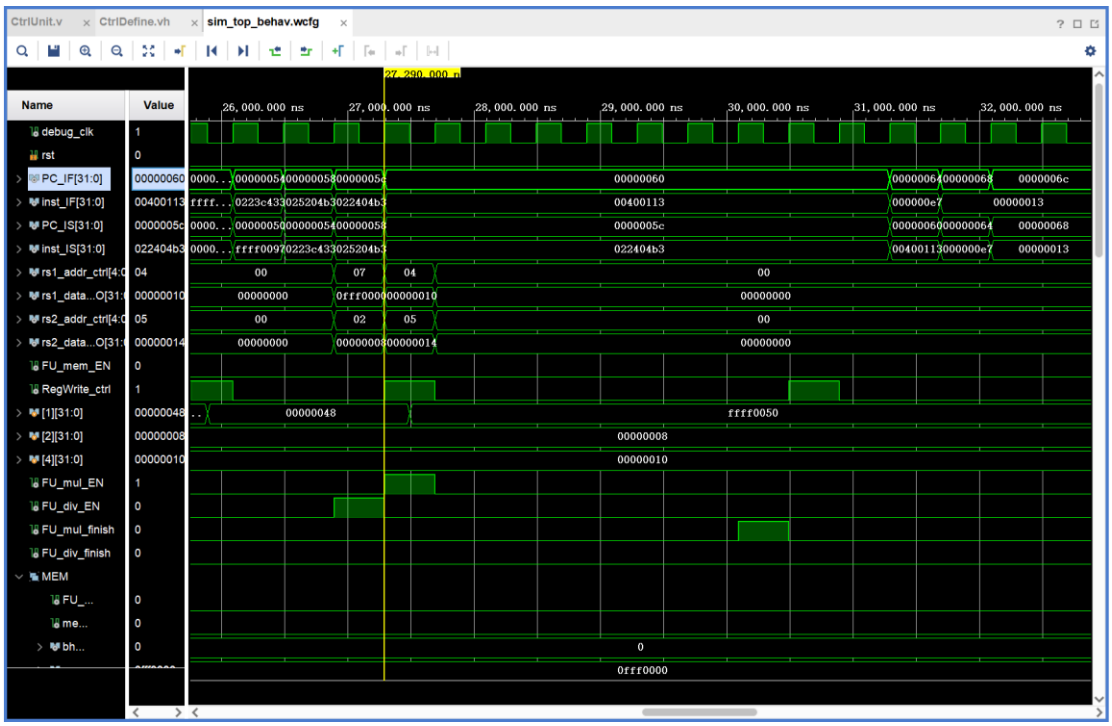
Structural Hazard：Functional Unit 被占用时指令不能发射，比如汇编代码中的连续两条 lw 指令，可以从仿真中看到第二条 load 指令被阻塞了 5 个周期。



WAW：当前面的指令还没有完成目的寄存器的写回时，下一条相同目的寄存器的指令也不能发射，下图中指令 10 处连续的第二条写 x1 的指令被阻塞。

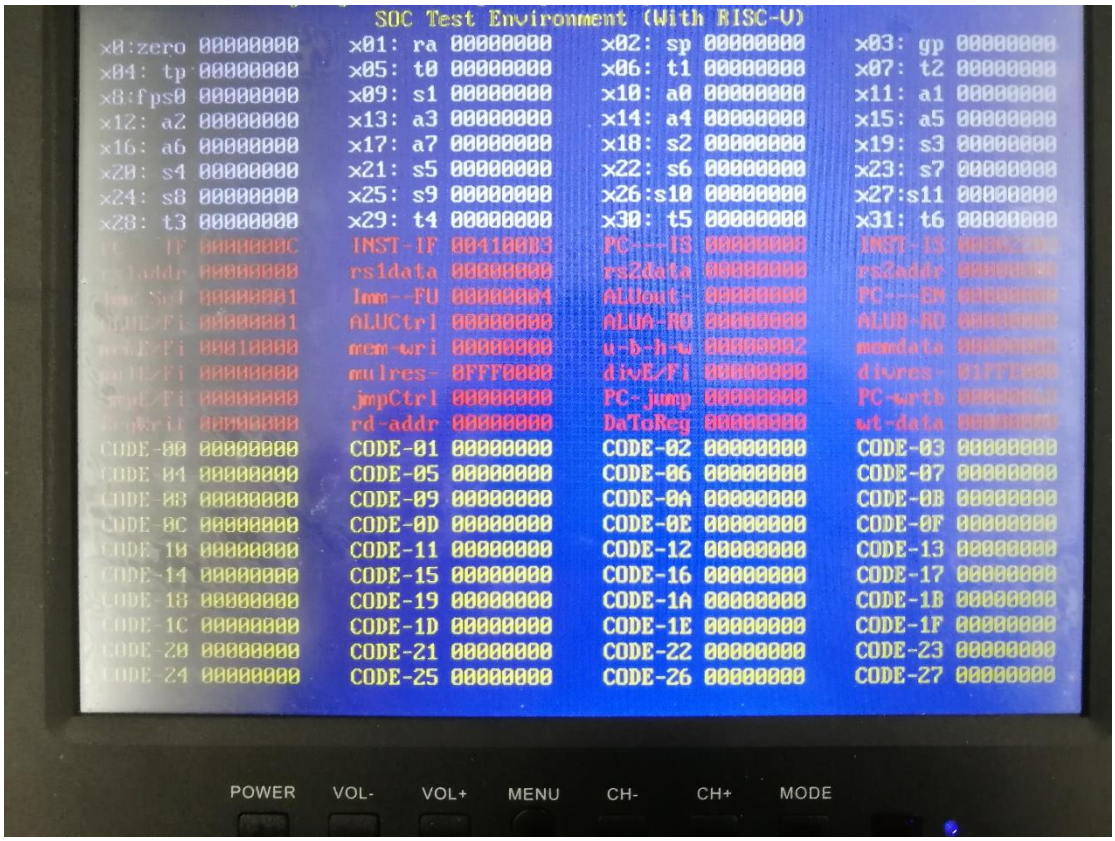


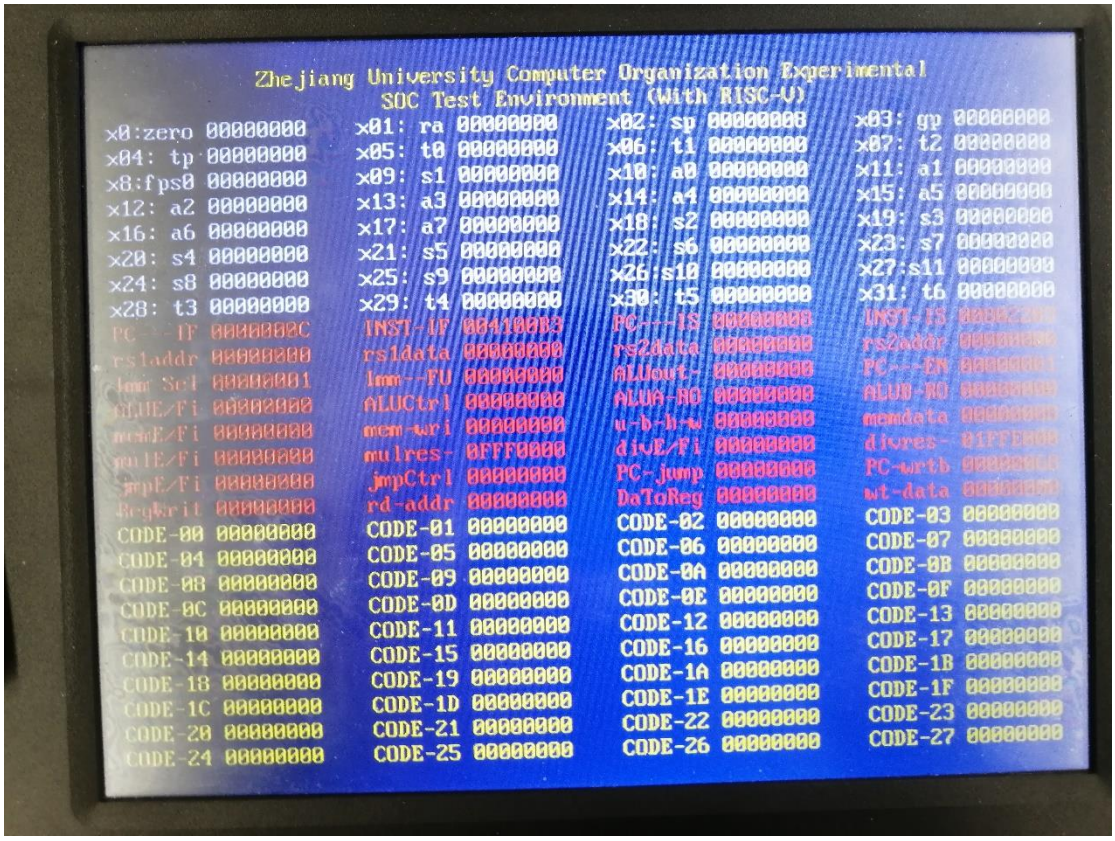
RAW：指令 5C 处发生 RAW 冒险，指令被阻塞。同时由于阻塞时长，源操作数已经顺利读取，下一条指令即使存在 WAR 冒险也可以依次正常运行。



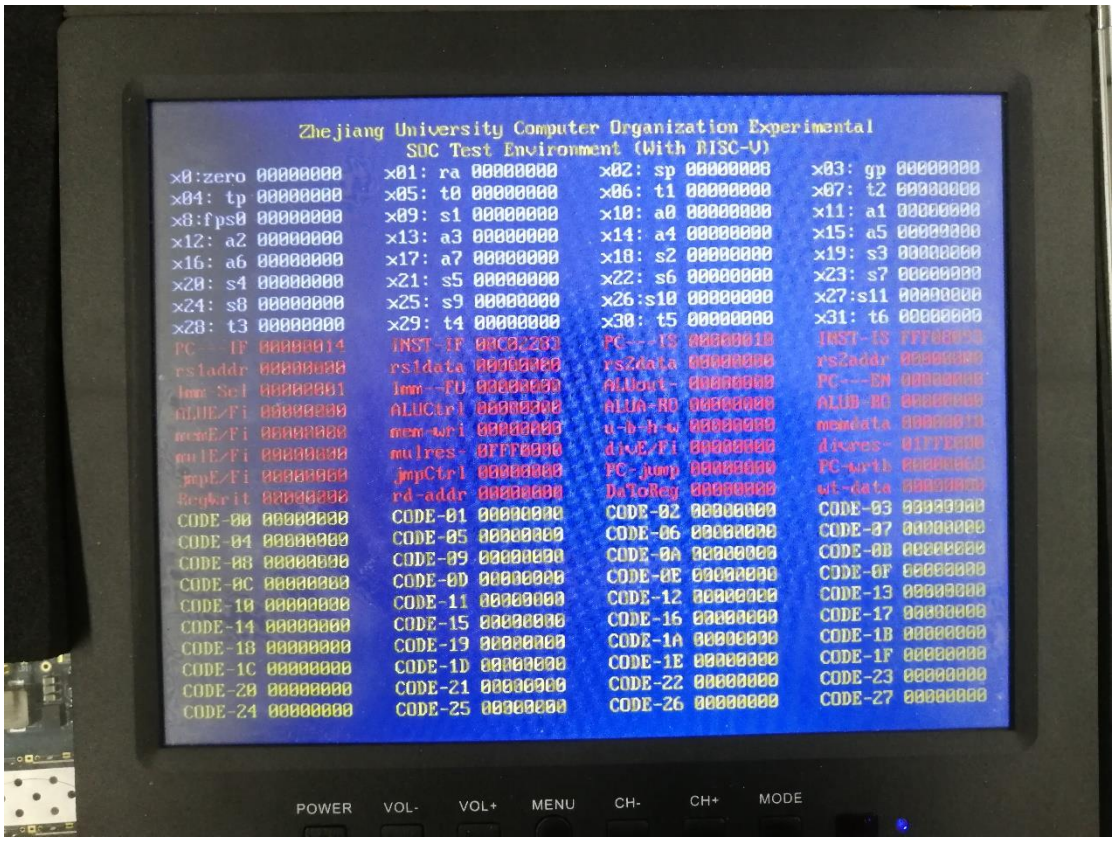
2．实验箱结果：

第二条 lw 指令前后：

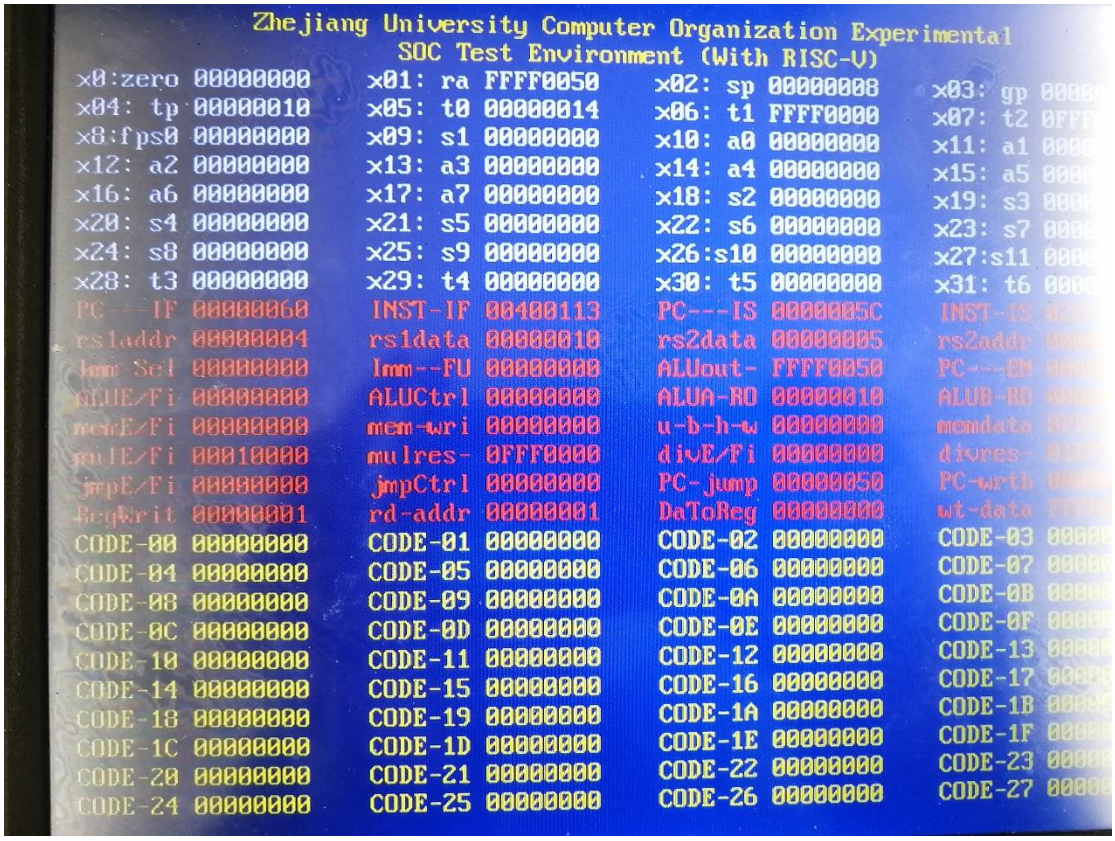


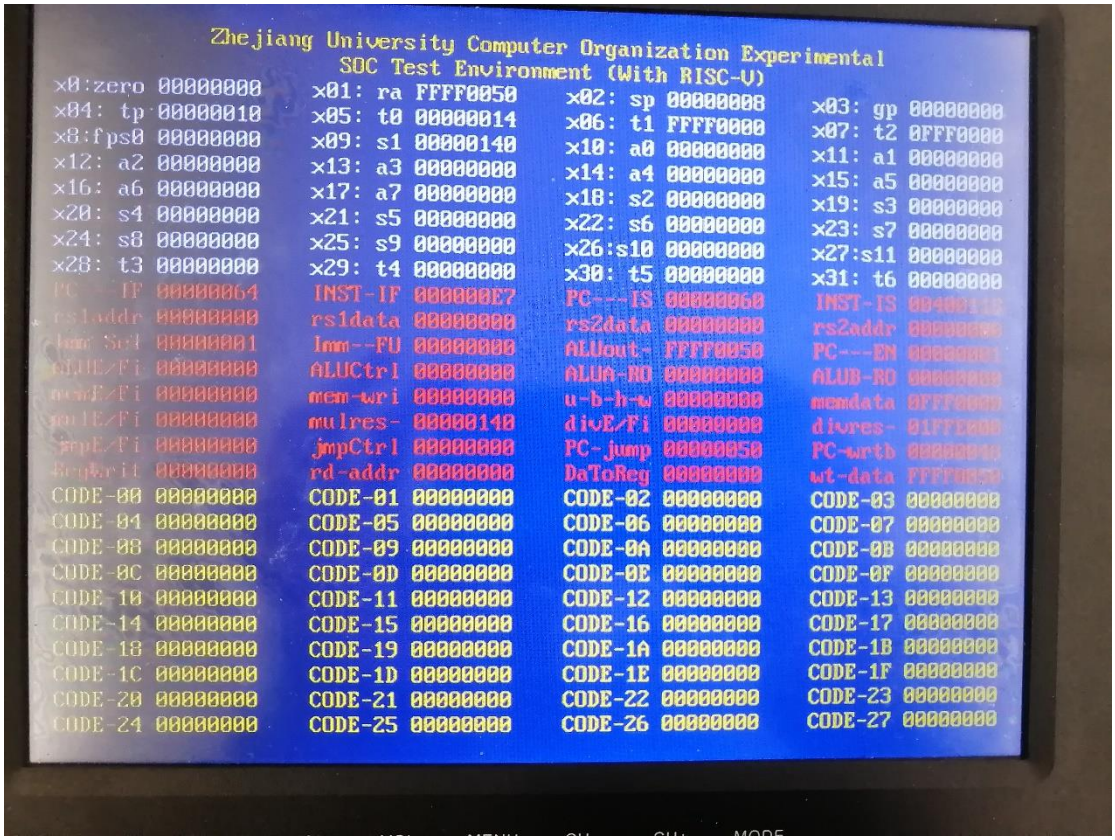

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000000 x02: sp 00000000 x03: gp 00000000
x04: tp 00000000 x05: t0 00000000 x06: t1 00000000 x07: t2 00000000
x8:fps0 00000000 x09: s1 00000000 x10: a0 00000000 x11: a1 00000000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000

WAW 指令运行时；


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra 00000000 x02: sp 00000000 x03: gp 00000000
x04: tp 00000000 x05: t0 00000000 x06: t1 00000000 x07: t2 00000000
x8:fps0 00000000 x09: s1 00000000 x10: a0 00000000 x11: a1 00000000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000
POWER VOL- VOL+ MENU CH- CH+ MODE

WAR 处：


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra FFFF0050 x02: sp 00000008 x03: gp 0000
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 0FFF
x8:fps0 00000000 x09: s1 00000000 x10: a0 00000000 x11: a1 0000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 0000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 0000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 0000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 0000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 0000
PC---IF 00000060 INST-IF 00400113 PC---IS 0000005C INST-IS
rs1addr 00000004 rs1data 00000010 rs2data 00000005 rs2addr
Imm Sel 00000000 Imm--FU 00000000 ALUout- FFFF0050 PC---EN
ALUE/Fi 00000000 ALUCtrl 00000000 ALUA-RO 00000010 ALUB-RO
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000 memdata
mulE/Fi 00010000 mulres- 0FFF0000 divE/Fi 00000000 divres-
jmpE/Fi 00000000 jmpCtrl 00000000 PC-jump 00000050 PC-wrtb
RegWrit 00000001 rd-addr 00000001 DaToReg 00000000 wt-data
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 0000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 0000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 0000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 0000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 0000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 0000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 0000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 0000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 0000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 0000


Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000 x01: ra FFFF0050 x02: sp 00000008 x03: gp 00000000
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 0FFF0000
x8:fps0 00000000 x09: s1 00000140 x10: a0 00000000 x11: a1 00000000
x12: a2 00000000 x13: a3 00000000 x14: a4 00000000 x15: a5 00000000
x16: a6 00000000 x17: a7 00000000 x18: s2 00000000 x19: s3 00000000
x20: s4 00000000 x21: s5 00000000 x22: s6 00000000 x23: s7 00000000
x24: s8 00000000 x25: s9 00000000 x26:s10 00000000 x27:s11 00000000
x28: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000
PC---IF 00000064 INST-IF 000000E7 PC---IS 00000060 INST-IS
rs1addr 00000000 rs1data 00000000 rs2data 00000000 rs2addr
Imm Sel 00000001 Imm--FU 00000000 ALUout- FFFF0050 PC---EN
ALUE/Fi 00000000 ALUCtrl 00000000 ALUA-RO 00000000 ALUB-RO
memE/Fi 00000000 mem-wri 00000000 u-b-h-w 00000000 memdata
mulE/Fi 00000000 mulres- 00000140 divE/Fi 00000000 divres-
jmpE/Fi 00000000 jmpCtrl 00000000 PC-jump 00000050 PC-wrtb
RegWrit 00000000 rd-addr 00000000 DaToReg 00000000 wt-data
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
CODE-04 00000000 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000
CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000
CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000
CODE-10 00000000 CODE-11 00000000 CODE-12 00000000 CODE-13 00000000
CODE-14 00000000 CODE-15 00000000 CODE-16 00000000 CODE-17 00000000
CODE-18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
CODE-1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
CODE-20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
CODE-24 00000000 CODE-25 00000000 CODE-26 00000000 CODE-27 00000000

五、　讨论与心得

本次实验是在实验五完成了支持多周期计算的流水线设计的基础上进行，不同于往几次实验修改一系列单元，scoreboard 只需要设计 ctrlunit，但这也导致了控制单元内容变得非常繁杂，对实验代码的编辑产生了一定的阻碍。

在掌握了 scoreboard 调度算法的基础上，完成此次实验还是相对不难，实验中花费时间调试的地方在于很多的细节。比如，执行阶段，常常需要考虑当前周期执行的到底是哪条指令，而不能通通将所有功能单元视为执行完成。WAR 信号的反向逻辑也给实验带来不小的理解难度。指令的两个源操作数都 ready 后，下一周期必须将其写 0，因此 WAR 判断时通过 rdy 为 0 来判断是否已经读取出寄存器数据。