浙江大學

本科实验报告

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

2023 年　　1 月　　11 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：　Lab6：Dynamically Scheduled Pipelines using Scoreboarding

学生姓名：　　　　　　专业：计算机科学与技术　　　　学号：

同组学生姓名：　　　　　　　　　　　　　　　　指导老师：卜凯

实验地点：　曹西 301　　　　　　　　　　实验日期：2023 年 1 月 11 日

1 Tasks and requirements

1.1 Tasks

The main tasks of Lab-6 are:

1. Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
2. Design of a scoreboard and integrate it to CPU.

1.2 Requirements

This experiment would be based on SWORD development board, with xc7k325tffg676-2L FPGA.

We are given a Verilog project complementing a CPU core and other components for debugging on board; this CPU should be able to deal with integer and float point operations. Most of the parts have been finished. There are still one unit we need to complete, it is:

CtrlUnit.v. This module is the control module of the whole CPU, and will be designed to support scoreboarding.

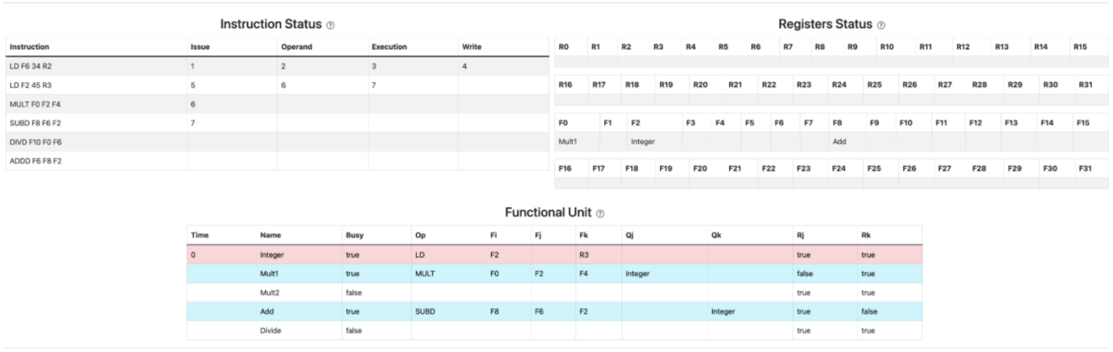
When this work is finished, we shall verify the simulation results of a preset program.

2 Contents and principles

All of the following principles are based on RISC-V pipelined float-point CPU.

2.1 Scoreboarding

Scoreboarding is a technique to support out-of-order execution of multicycle CPU. Scoreboarding uses 3 tables to indicate the current status of function units:



Instruction Status

Instruction	Issue	Operand	Execution	Write
LD F6 34 R2	1	2	3	4
LD F2 45 R3	5	6	7	
MULT F0 F2 F4	6			
SUBD F8 F6 F2	7			
DIVD F10 F0 F6				
ADDD F6 F8 F2				

Registers Status

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15

R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Mult1		Integer						Add							

F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28	F29	F30	F31

Functional Unit

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
0	Integer	true	LD	F2		R3			true	true
	Mult1	true	MULT	F0	F2	F4	Integer		false	true
	Mult2	false							true	true
	Add	true	SUBD	F8	F6	F2		Integer	true	false
	Divide	false							true	true

The first table logs the running status of the **instructions**, especially holding the information about the stages which every instruction has finished; The second table logs the status of the **registers**, especially which function unit will write that register; The third table logs the information of the **function units**, about which unit is dealing with which instruction, and the table contain such entries:

Busy	– Indicate whether the unit is busy or not.
Op	– Operation to perform in the unit (e.g., addr or subtract).
Fi	– Destination register.
Fj, Fk	– Source-register numbers.
Qj, Qk	– Functional units producing source registers Fj, Fk
Rj, Rk	– Flags indicating when Fj, Fk are ready and not yet read. Set to No after operands.
+ FU_DONE	

The **FU_DONE** signal is set to indicate the first table, giving out the information that a function unit has finished an instruction.

From these tables, we could detect WAR and WAW hazards in out-of-order execution, that information will be passed to ID stage.

In order to use and write the information properly, we also divide ID stage into 2 stages: **IS(Issue)** and **RO(Read Operands)**. **IS** stage decide whether an instruction will be issued by the current scoreboard, and **RO** stage will write information to the scoreboard.

3 Steps and data records

3.1 Completed Verilog source files

For clarity, we will omit some pre-set code segments, and only analysis the code blocks that we need to fill in.

3.1.1 CtrlUnit.v

In this file, we are requested to finish the parts that maintains the table 2 and 3 of scoreboarding. First, we will detect structural hazards and **WAW** hazards.

```
assign normal_stall = ( use_ALU & FUS[`FU_ALU][`BUSY]) |
                      ( use_MEM & FUS[`FU_MEM][`BUSY]) |
                      ( use_MUL & FUS[`FU_MUL][`BUSY]) |
                      ( use_DIV & FUS[`FU_DIV][`BUSY]) |
                      (use_JUMP & FUS[`FU_JUMP][`BUSY])|
                      ((R_valid | I_valid | L_valid | LUI | AUIPC | JAL | JALR) & RRS[rd]);
```

Then we use the table information to avoid **WAR** hazards. For each function unit we guarantee there are no other units which will use the destination register of current instruction as a source, or both waiting to be written first. As **WAW** hazards have been detected above, we can only check the **R_i** field of that register.

```
wire ALU_WAR = (
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY1])   &
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY2])   &
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY1])   &
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY2])   &
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY1])   &
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L]   != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY2])    &
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY1])    &
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY2])
);
wire MEM_WAR = (
    (FUS[`FU_ALU][`SRC1_H:`SRC1_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY1]) &
    (FUS[`FU_ALU][`SRC2_H:`SRC2_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY2]) &
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY1]) &
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY2]) &
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY1]) &
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L]    != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY2]) &
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY1]) &
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_MEM][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY2])
);

wire MUL_WAR = (
    (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY1]) &
    (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY2]) &
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY1]) &
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY2]) &
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY1]) &
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY2]) &
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L]!= FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY1]) &
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L]!= FUS[`FU_MUL][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY2])
);

wire DIV_WAR = (
    (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY1]) &
    (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY2]) &
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY1]) &
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY2]) &
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY1]) &
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY2]) &
    (FUS[`FU_JUMP][`SRC1_H:`SRC1_L]!= FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY1]) &
    (FUS[`FU_JUMP][`SRC2_H:`SRC2_L]!= FUS[`FU_DIV][`DST_H:`DST_L]
        | !FUS[`FU_JUMP][`RDY2])
);

wire JUMP_WAR = (
    (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY1]) &
    (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_ALU][`RDY2]) &
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY1]) &
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_MEM][`RDY2]) &
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY1]) &
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_MUL][`RDY2]) &
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY1]) &
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L]
        | !FUS[`FU_DIV][`RDY2])
);
```

If any instruction successfully entered **RO** stage, we update the table. First update table 3:

```
FUS[use_FU][`SRC1_H:`SRC1_L] <= src1;
FUS[use_FU][`SRC2_H:`SRC2_L] <= src2;
FUS[use_FU][`DST_H:`DST_L] <= dst;
FUS[use_FU][`FU1_H:`FU1_L] <= fu1;
FUS[use_FU][`FU2_H:`FU2_L] <= fu2;
FUS[use_FU][`OP_H:`OP_L] <= op;
FUS[use_FU][`RDY1] <= rdy1;
```

```
FUS[use_FU][`RDY2] <= rdy2;
if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2]) begin
    // JUMP
    FUS[`FU_JUMP][`RDY1] <= 1'b0;
    FUS[`FU_JUMP][`RDY2] <= 1'b0;
end
else if (FUS[`FU_ALU][`RDY1] & FUS[`FU_ALU][`RDY2]) begin
    // ALU
    FUS[`FU_ALU][`RDY1] <= 1'b0;
    FUS[`FU_ALU][`RDY2] <= 1'b0;
end
else if (FUS[`FU_MEM][`RDY1] & FUS[`FU_MEM][`RDY2]) begin
    // MEM
    FUS[`FU_MEM][`RDY1] <= 1'b0;
    FUS[`FU_MEM][`RDY2] <= 1'b0;
end
else if (FUS[`FU_MUL][`RDY1] & FUS[`FU_MUL][`RDY2]) begin
    // MUL
    FUS[`FU_MUL][`RDY1] <= 1'b0;
    FUS[`FU_MUL][`RDY2] <= 1'b0;
end
else if (FUS[`FU_DIV][`RDY1] & FUS[`FU_DIV][`RDY2]) begin
    // DIV
    FUS[`FU_DIV][`RDY1] <= 1'b0;
    FUS[`FU_DIV][`RDY2] <= 1'b0;
end

// EX
FUS[`FU_ALU][`FU_DONE] <= FUS[`FU_ALU][`FU_DONE] | ALU_done;
FUS[`FU_MEM][`FU_DONE] <= FUS[`FU_MEM][`FU_DONE] | MEM_done;
```

```
FUS[`FU_MUL][`FU_DONE] <= FUS[`FU_MUL][`FU_DONE] | MUL_done;
FUS[`FU_DIV][`FU_DONE] <= FUS[`FU_DIV][`FU_DONE] | DIV_done;
FUS[`FU_JUMP][`FU_DONE] <= FUS[`FU_JUMP][`FU_DONE] | JUMP_done;
```

At **WB** stage (when an instruction is done), we clear the table entry affected by this instruction from table 2 and 3, and set **R_i** fields of other entries.

```
// 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)
    FUS[`FU_ALU][`RDY1]<=1'b1;
  If (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_JUMP)
    FUS[`FU_MEM][`RDY1]<=1'b1;
  If (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_JUMP)
    FUS[`FU_MUL][`RDY1]<=1'b1;
  If (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_JUMP)
    FUS[`FU_DIV][`RDY1]<=1'b1;

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

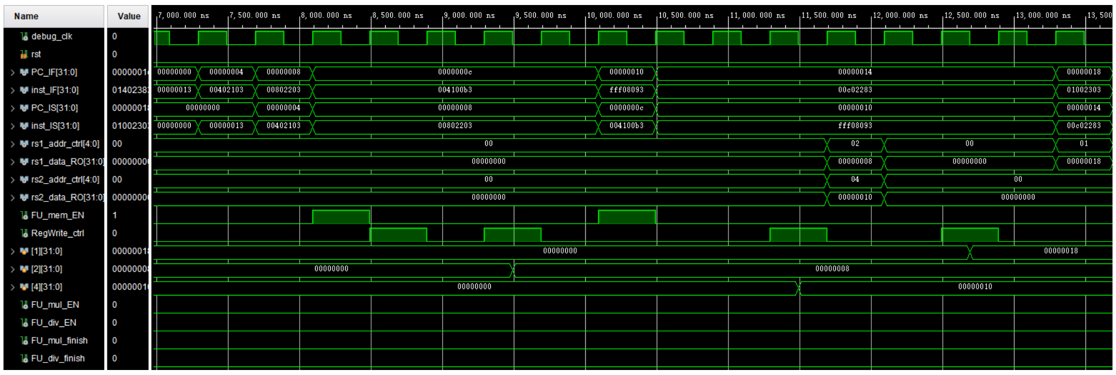
```
if (FUS[`FU_ALU][`FU_DONE] & ALU_WAR) begin
    FUS[`FU_ALU] <= 32'b0;
    RRS[FUS[`FU_ALU][`DST_H:`DST_L]] <= 3'b0;

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

    if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_ALU)
        FUS[`FU_JUMP][`RDY2]<=1'b1;
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_ALU)
        FUS[`FU_MEM][`RDY2]<=1'b1;
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_ALU)
        FUS[`FU_MUL][`RDY2]<=1'b1;
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_ALU)
        FUS[`FU_DIV][`RDY2]<=1'b1;
end
// MEM
if (FUS[`FU_MEM][`FU_DONE] & MEM_WAR) begin
    FUS[`FU_MEM] <= 32'b0;
    RRS[FUS[`FU_MEM][`DST_H:`DST_L]] <= 3'b0;

// ensure RAW
    if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_MEM)
```

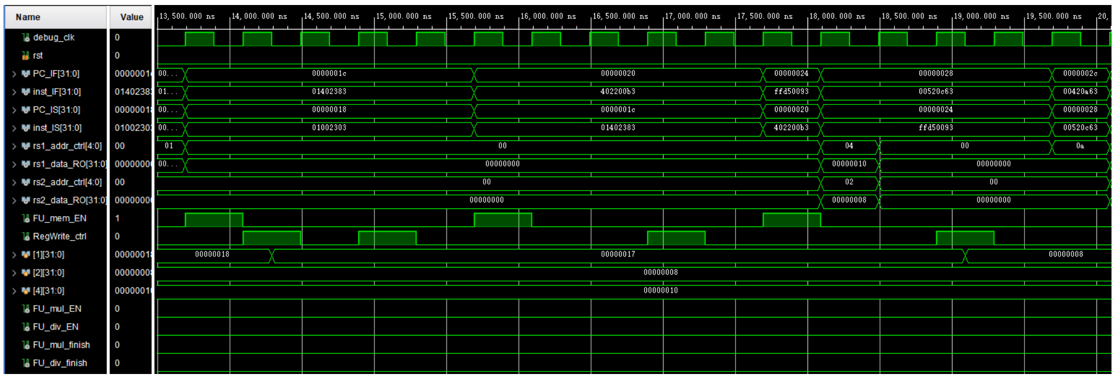
```
        FUS[`FU_JUMP][`RDY1]<=1'b1;
    if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_MEM)
        FUS[`FU_ALU][`RDY1]<=1'b1;
    if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_MEM)
        FUS[`FU_MUL][`RDY1]<=1'b1;
    if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_MEM)
        FUS[`FU_DIV][`RDY1]<=1'b1;

    if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_MEM)
        FUS[`FU_JUMP][`RDY2]<=1'b1;
    if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_MEM)
        FUS[`FU_ALU][`RDY2]<=1'b1;
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_MEM)
        FUS[`FU_MUL][`RDY2]<=1'b1;
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_MEM)
        FUS[`FU_DIV][`RDY2]<=1'b1;
end
// MUL
if (FUS[`FU_MUL][`FU_DONE] & MUL_WAR) begin
    FUS[`FU_MUL] <= 32'b0;
    RRS[FUS[`FU_MUL][`DST_H:`DST_L]] <= 3'b0;

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

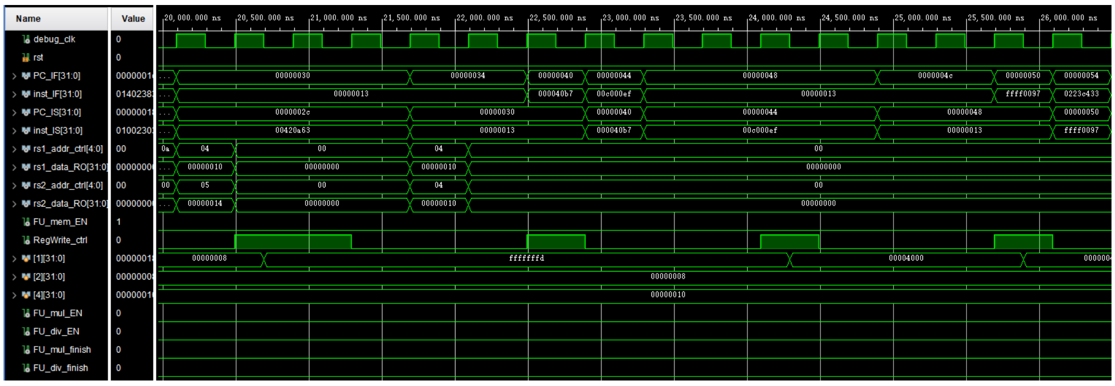
```
        FUS[`FU_DIV][`RDY1]<=1'b1;

    if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_MUL)
        FUS[`FU_JUMP][`RDY2]<=1'b1;
    if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_MUL)
        FUS[`FU_ALU][`RDY2]<=1'b1;
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_MUL)
        FUS[`FU_MEM][`RDY2]<=1'b1;
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_MUL)
        FUS[`FU_DIV][`RDY2]<=1'b1;
end
// DIV
if (FUS[`FU_DIV][`FU_DONE] & DIV_WAR) begin
    FUS[`FU_DIV] <= 32'b0;
    RRS[FUS[`FU_DIV][`DST_H:`DST_L]] <= 3'b0;

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

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

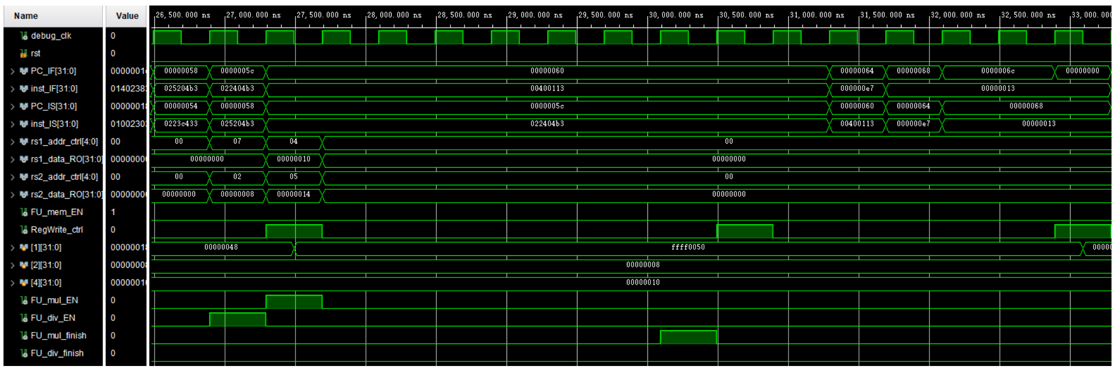
3.2 Implementation results

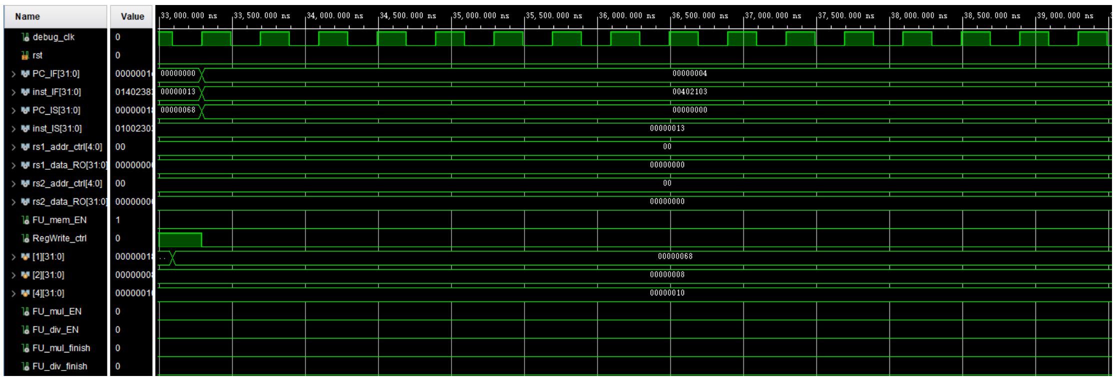
Here we record all our behavioral simulation results. Most will not be used for analysis.

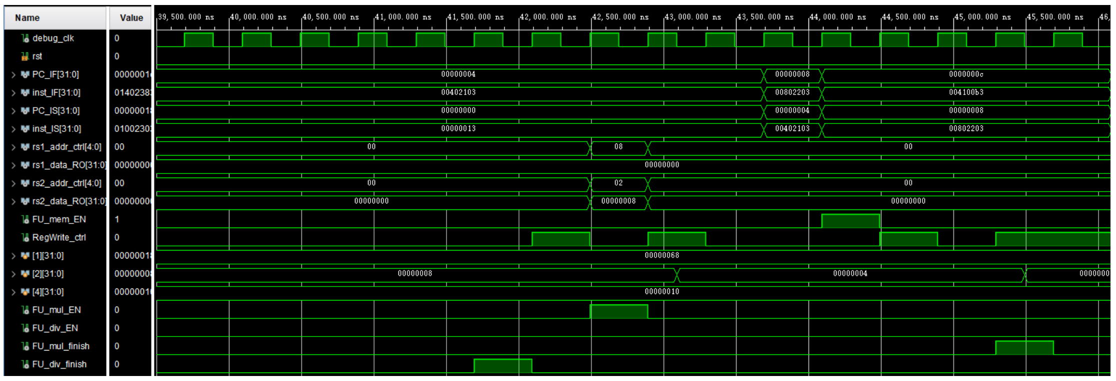







Name
Value
debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
[1][31:0]
[2][31:0]
[4][31:0]
FU_mul_EN
FU_div_EN
FU_mul_finish
FU_div_finish

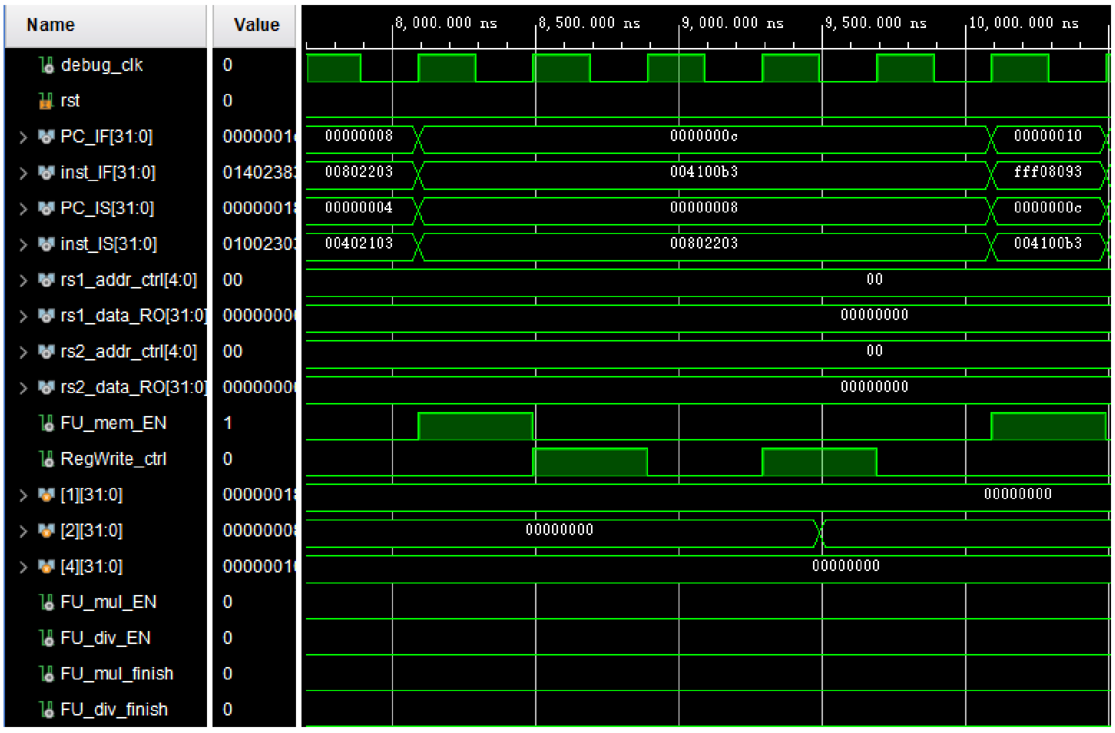

Name
Value
debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
[1][31:0]
[2][31:0]
[4][31:0]
FU_mul_EN
FU_div_EN
FU_mul_finish
FU_div_finish


Name
Value
debug_clk
rst
PC_IF[31:0]
inst_IF[31:0]
PC_IS[31:0]
inst_IS[31:0]
rs1_addr_ctrl[4:0]
rs1_data_RO[31:0]
rs2_addr_ctrl[4:0]
rs2_data_RO[31:0]
FU_mem_EN
RegWrite_ctrl
[1][31:0]
[2][31:0]
[4][31:0]
FU_mul_EN
FU_div_EN
FU_mul_finish
FU_div_finish

4 Analysis of the results

Here we only discuss some typical results.

4.1 Structural hazard avoidance



Consider the result in 7500ns to 10500ns. The mentioned instructions are:

PC: 0x00000004, Inst: 0x00402103, **lw x2, 4(x0)**

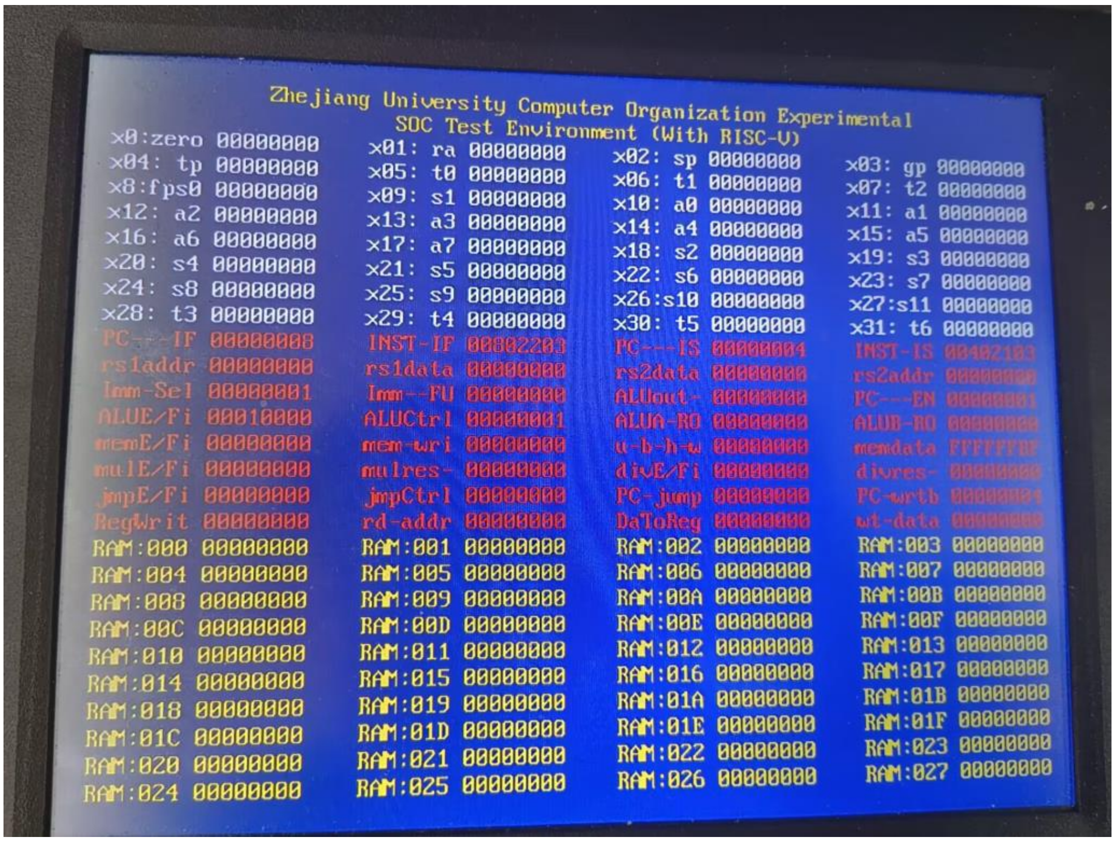
PC: 0x00000008, Inst: 0x00802203, **lw x4, 8(x0)**

PC: 0x0000000c, Inst: 0x004100b3, **add x1, x2, x4**

As there are only one **FU_MEM** function unit, there produced a **structural hazard**. We could see, after the function during 9300-9700ns, the register was successfully written by the first instruction. Then the second instruction is issued during 9700ns-10100ns.

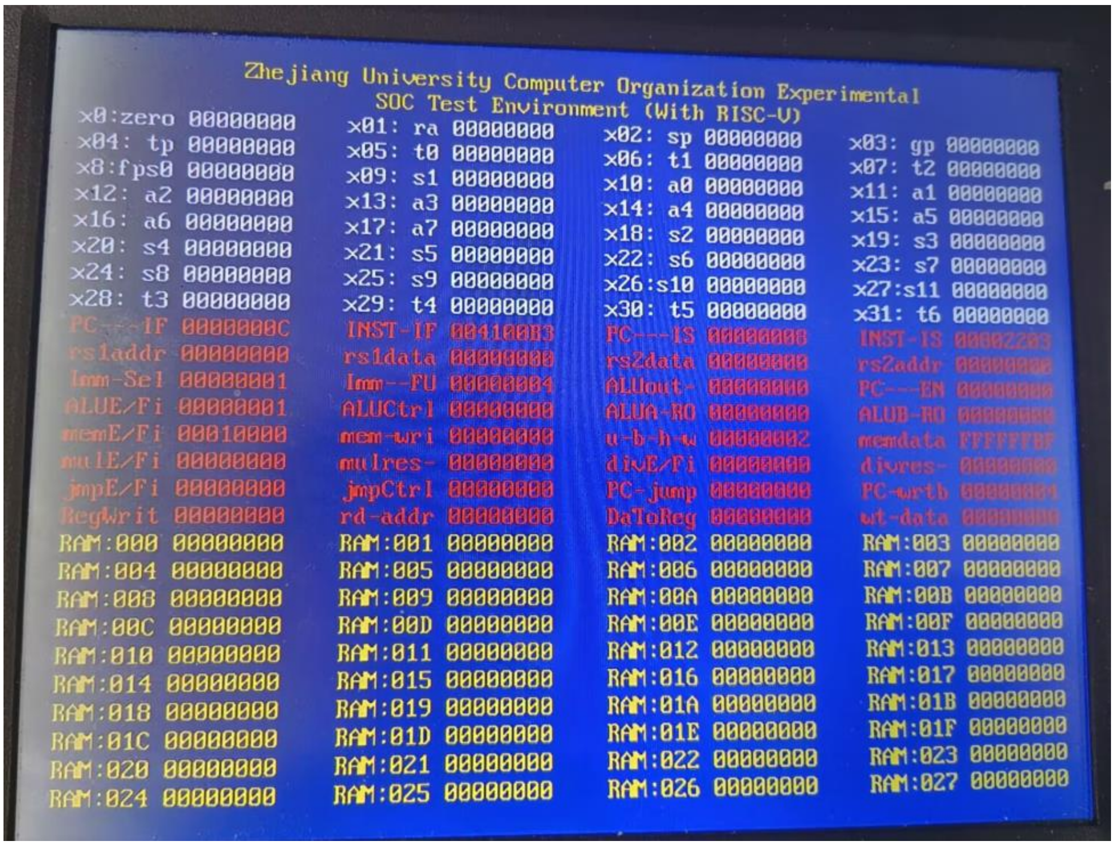
The third has no dependency so it's issued in 10100-10500ns as predicted.

Following are the on-board results of this situation:



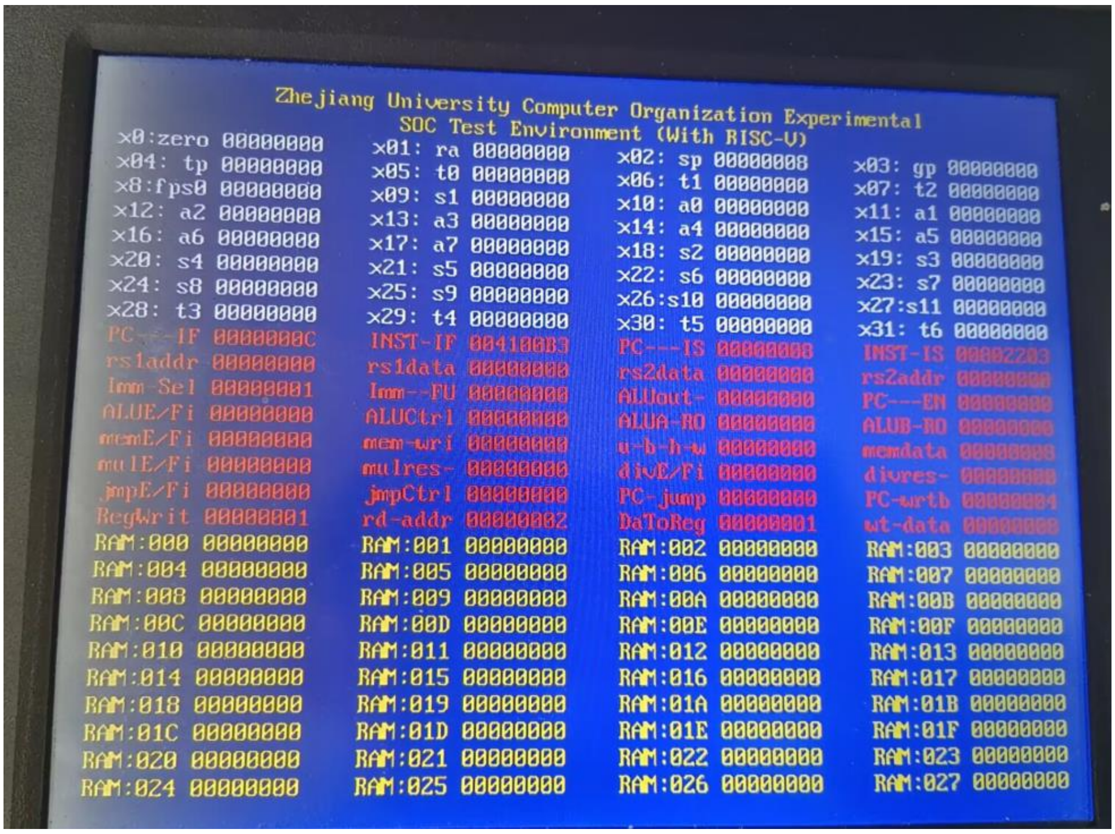
```
Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000000   x02: sp 00000000   x03: gp 00000000
x04: tp 00000000   x05: t0 00000000   x06: t1 00000000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000000   x10: a0 00000000   x11: 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
PC---IF 00000008   INST-IF 00802203   PC---IS 00000004   INST-IS 00402103
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000000   ALUout- 00000000   PC---EN 00000001
ALUE/Fi 00010000   ALUCtrl 00000001   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata FFFFFFBF
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- 00000000
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb 00000004
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data 00000000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

Before issuing the first **lw**.



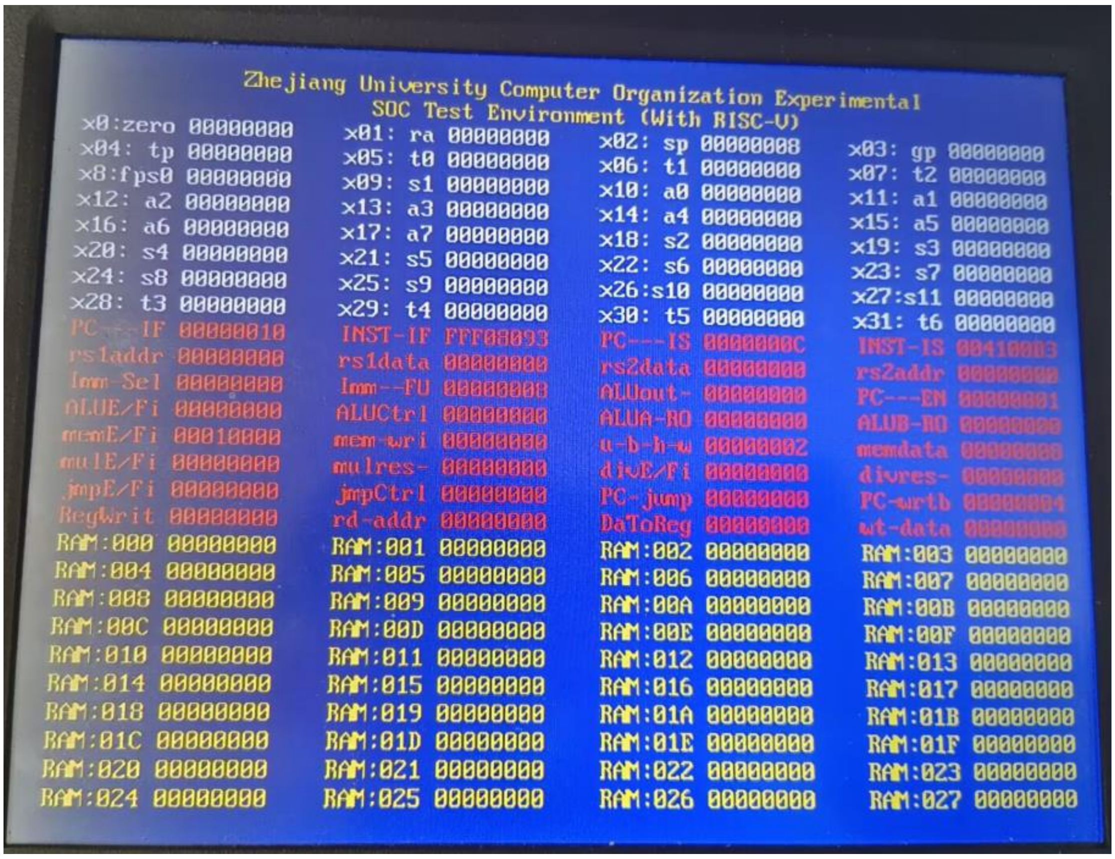
```
Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000000   x02: sp 00000000   x03: gp 00000000
x04: tp 00000000   x05: t0 00000000   x06: t1 00000000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000000   x10: a0 00000000   x11: 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
PC---IF 0000000C   INST-IF 004100B3   PC---IS 00000008   INST-IS 00802203
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000004   ALUout- 00000000   PC---EN 00000000
ALUE/Fi 00000001   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00010000   mem-wri 00000000   u-b-h-w 00000002   memdata FFFFFFBF
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- 00000000
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb 00000004
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data 00000000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

Issued, **mem_EN** is set to 1.



```
        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
PC---IF 0000000C   INST-IF 004100B3   PC---IS 00000008   INST-IS 00002203
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000000   ALUout- 00000000   PC---EN 00000000
ALUE/Fi 00000000   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 00000000
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- 00000000
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb 00000004
RegWrit 00000001   rd-addr 00000002   DaToReg 00000001   wt-data 00000000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

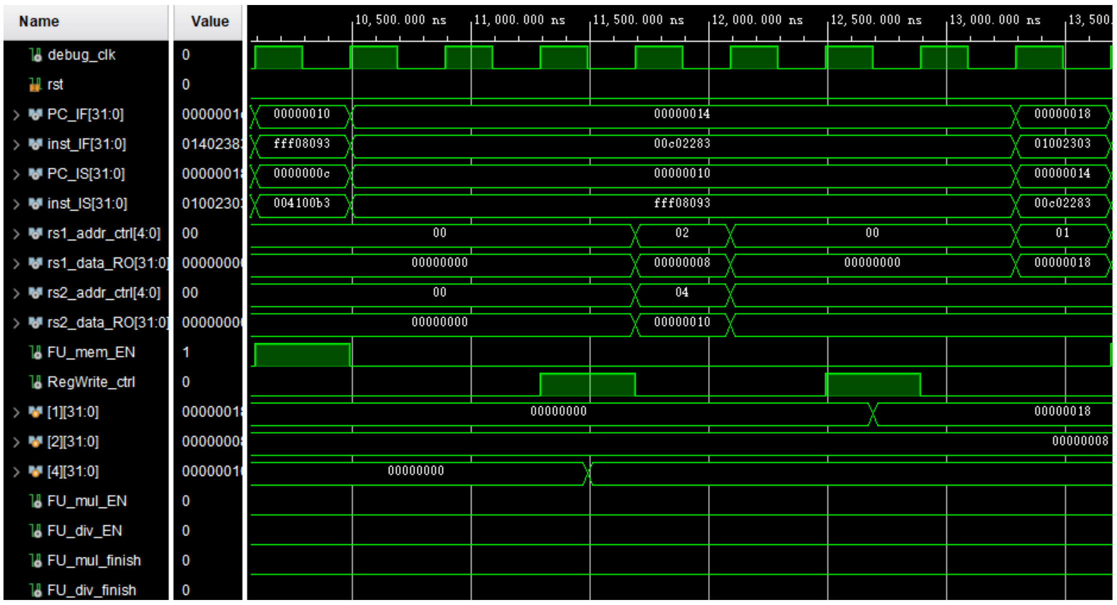
After 3 cycles, the instruction finished, and **RegWrite** is set.



```
        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
PC---IF 00000010   INST-IF FFF08093   PC---IS 0000000C   INST-IS 00410083
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000000   Imm--FU 00000008   ALUout- 00000000   PC---EN 00000001
ALUE/Fi 00000000   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00010000   mem-wri 00000000   u-b-h-w 00000002   memdata 00000000
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- 00000000
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb 00000004
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data 00000000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

After another 1 cycle the next instruction was issued normally.

4.2 WAW hazard avoidance



Consider the result in 10100ns to 13700ns. The mentioned instructions are:

PC: 0x0000000c, Inst: 0x004100b3, **add x1, x2, x4**

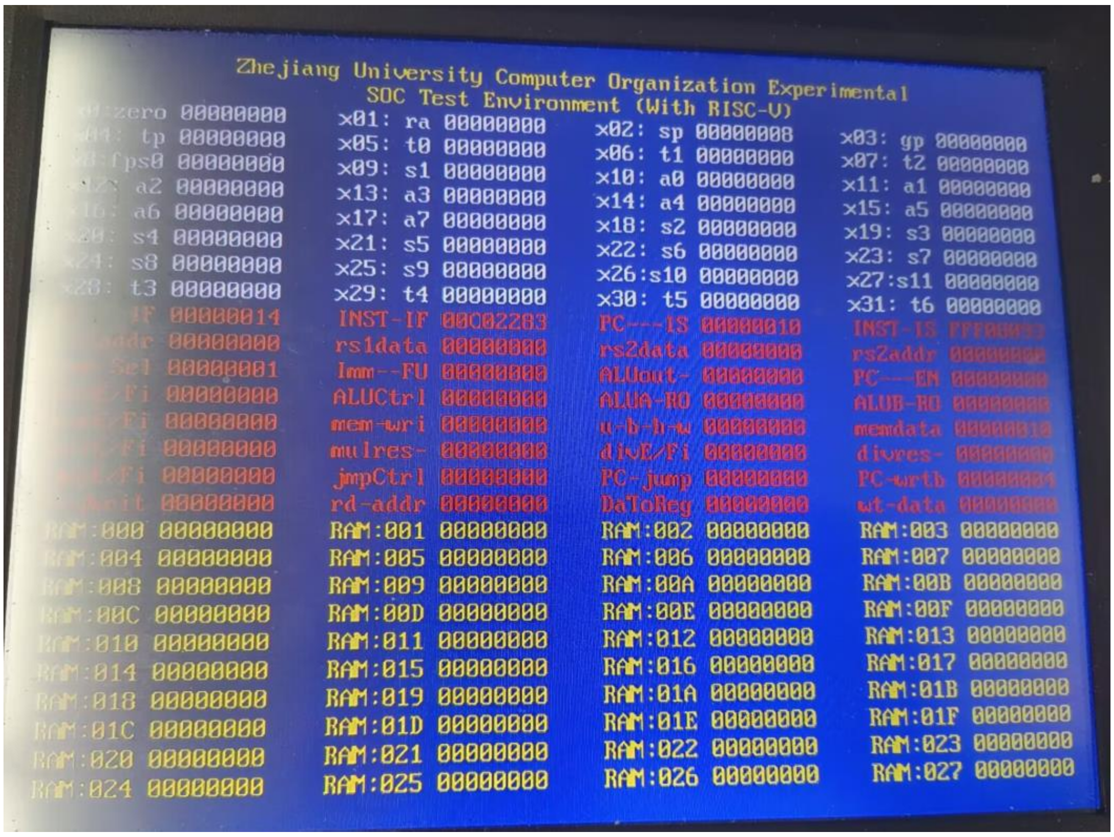
PC: 0x00000010, Inst: 0xfff08093, **addi x1, x1, -1**

PC: 0x00000014, Inst: 0x00c02283, **lw x5, 12(x0)**

We could see the second instruction has a **data dependency** on the first instruction, so it has to wait to be issued after the first instruction is finished. We see the first instruction finished in 12500-12900ns, and in the next cycle the second is issued. (However, this example may be not so clear, as it is also a structural hazard.)

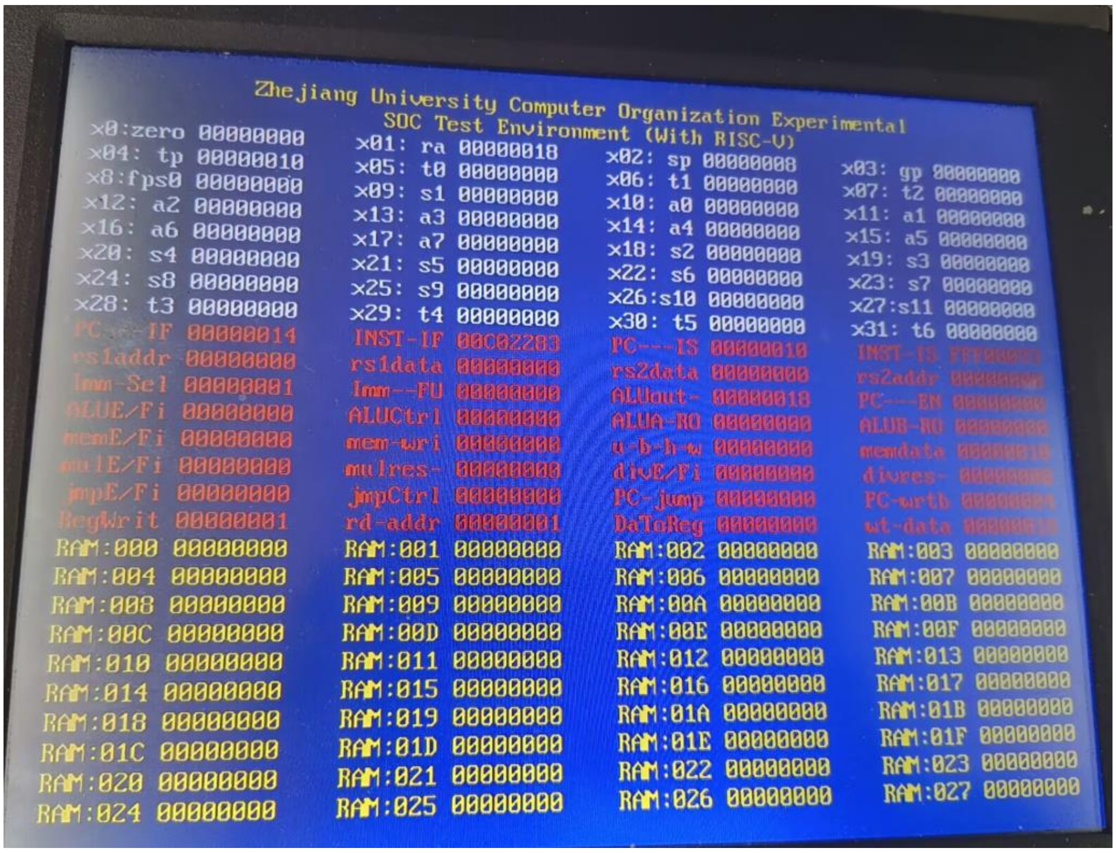
The third has no dependency so it’s issued in the next cycle as predicted.

Following are the on-board results:



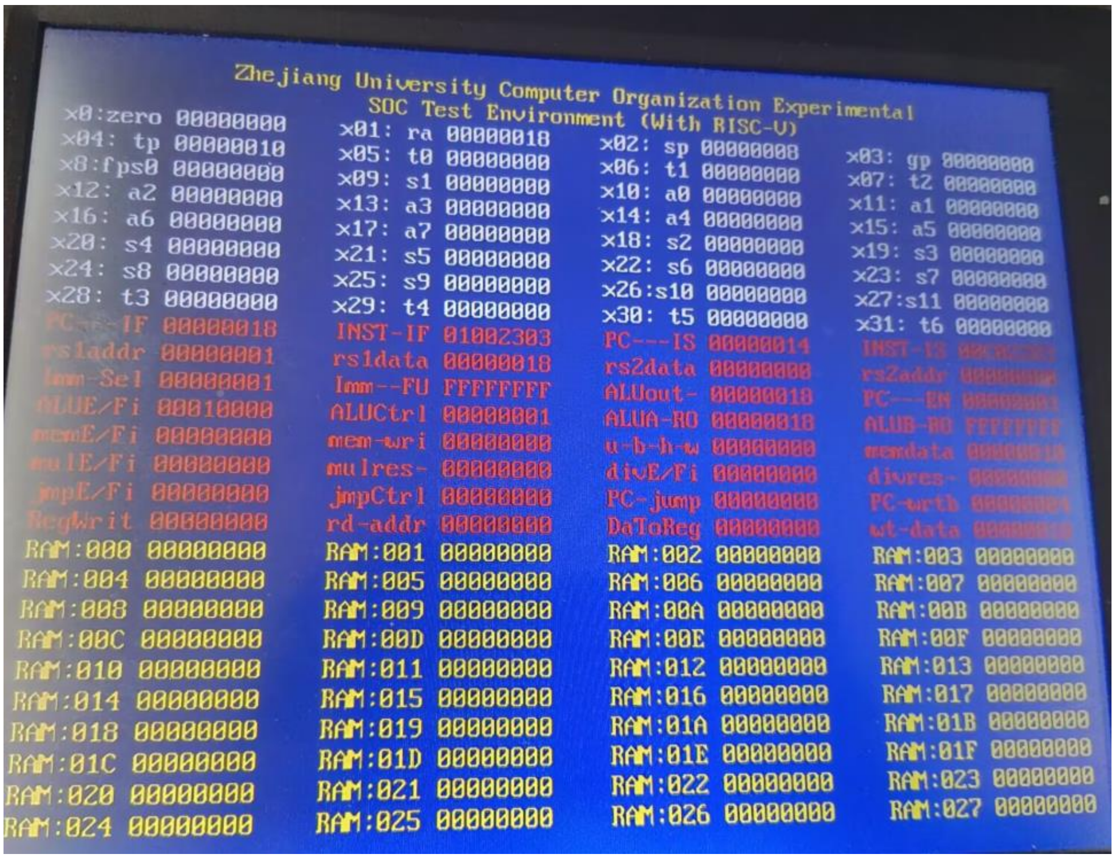
```
              Zhejiang University Computer Organization Experimental
                      SOC Test Environment (With RISC-V)
x0:zero  00000000   x01: ra  00000000   x02: sp  00000008   x03: gp  00000000
x04: tp  00000000   x05: t0  00000000   x06: t1  00000000   x07: t2  00000000
x8:fps0  00000000   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
PC---IF  00000014   INST-IF  00C02283   PC---IS  00000010   INST-IS  FFF00093
rs1addr  00000000   rs1data  00000000   rs2data  00000000   rs2addr  00000000
Imm-Sel  00000001   Imm--FU  00000000   ALUout-  00000000   PC---EN  00000000
ALUE/Fi  00000000   ALUCtrl  00000000   ALUA-RO  00000000   ALUB-RO  00000000
memE/Fi  00000000   mem-wri  00000000   u-b-h-w  00000000   memdata  00000010
mulE/Fi  00000000   mulres-  00000000   divE/Fi  00000000   divres-  00000000
jmpE/Fi  00000000   jmpCtrl  00000000   PC-jump  00000000   PC-wrtb  00000004
RegWrit  00000000   rd-addr  00000000   DaToReg  00000000   wt-data  00000000
RAM:000  00000000   RAM:001  00000000   RAM:002  00000000   RAM:003  00000000
RAM:004  00000000   RAM:005  00000000   RAM:006  00000000   RAM:007  00000000
RAM:008  00000000   RAM:009  00000000   RAM:00A  00000000   RAM:00B  00000000
RAM:00C  00000000   RAM:00D  00000000   RAM:00E  00000000   RAM:00F  00000000
RAM:010  00000000   RAM:011  00000000   RAM:012  00000000   RAM:013  00000000
RAM:014  00000000   RAM:015  00000000   RAM:016  00000000   RAM:017  00000000
RAM:018  00000000   RAM:019  00000000   RAM:01A  00000000   RAM:01B  00000000
RAM:01C  00000000   RAM:01D  00000000   RAM:01E  00000000   RAM:01F  00000000
RAM:020  00000000   RAM:021  00000000   RAM:022  00000000   RAM:023  00000000
RAM:024  00000000   RAM:025  00000000   RAM:026  00000000   RAM:027  00000000
```

Add instruction issued.



```
              Zhejiang University Computer Organization Experimental
                      SOC Test Environment (With RISC-V)
x0:zero  00000000   x01: ra  00000018   x02: sp  00000008   x03: gp  00000000
x04: tp  00000010   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
PC---IF  00000014   INST-IF  00C02283   PC---IS  00000010   INST-IS  FFF00093
rs1addr  00000000   rs1data  00000000   rs2data  00000000   rs2addr  00000000
Imm-Sel  00000001   Imm--FU  00000000   ALUout-  00000018   PC---EN  00000000
ALUE/Fi  00000000   ALUCtrl  00000000   ALUA-RO  00000000   ALUB-RO  00000000
memE/Fi  00000000   mem-wri  00000000   u-b-h-w  00000000   memdata  00000010
mulE/Fi  00000000   mulres-  00000000   divE/Fi  00000000   divres-  00000000
jmpE/Fi  00000000   jmpCtrl  00000000   PC-jump  00000000   PC-wrtb  00000004
RegWrit  00000001   rd-addr  00000001   DaToReg  00000000   wt-data  00000018
RAM:000  00000000   RAM:001  00000000   RAM:002  00000000   RAM:003  00000000
RAM:004  00000000   RAM:005  00000000   RAM:006  00000000   RAM:007  00000000
RAM:008  00000000   RAM:009  00000000   RAM:00A  00000000   RAM:00B  00000000
RAM:00C  00000000   RAM:00D  00000000   RAM:00E  00000000   RAM:00F  00000000
RAM:010  00000000   RAM:011  00000000   RAM:012  00000000   RAM:013  00000000
RAM:014  00000000   RAM:015  00000000   RAM:016  00000000   RAM:017  00000000
RAM:018  00000000   RAM:019  00000000   RAM:01A  00000000   RAM:01B  00000000
RAM:01C  00000000   RAM:01D  00000000   RAM:01E  00000000   RAM:01F  00000000
RAM:020  00000000   RAM:021  00000000   RAM:022  00000000   RAM:023  00000000
RAM:024  00000000   RAM:025  00000000   RAM:026  00000000   RAM:027  00000000
```

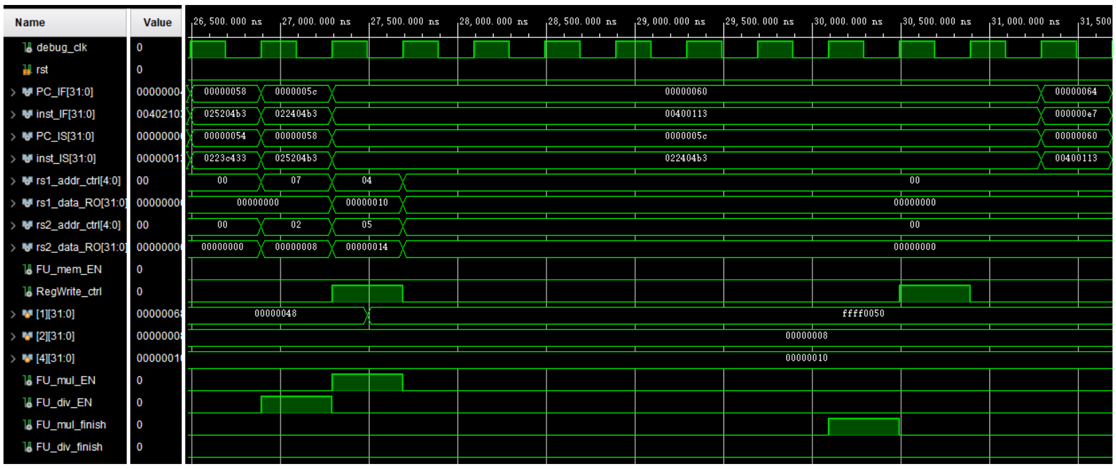
Its **RegWrite** cycle.



```
        Zhejiang University Computer Organization Experimental
              SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000018   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   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
PC---IF 00000018   INST-IF 01002303   PC---IS 00000014   INST-IS [illegible]
rs1addr 00000001   rs1data 00000018   rs2data 00000000   rs2addr [illegible]
Imm-Sel 00000001   Imm--FU FFFFFFFF   ALUout- 00000018   PC---EN [illegible]
ALUE/Fi 00010000   ALUCtrl 00000001   ALUA-RO 00000018   ALUB-RO FFFFFFFF
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata [illegible]
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- [illegible]
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb [illegible]
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data [illegible]
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

The data-dependent instruction was issued after the first finished.

4.3WAR hazard avoidance



Consider the result in 26500ns to 31700ns. The mentioned instructions are:

PC: 0x00000054, Inst: 0x0223c433, **div x8, x7, x2**

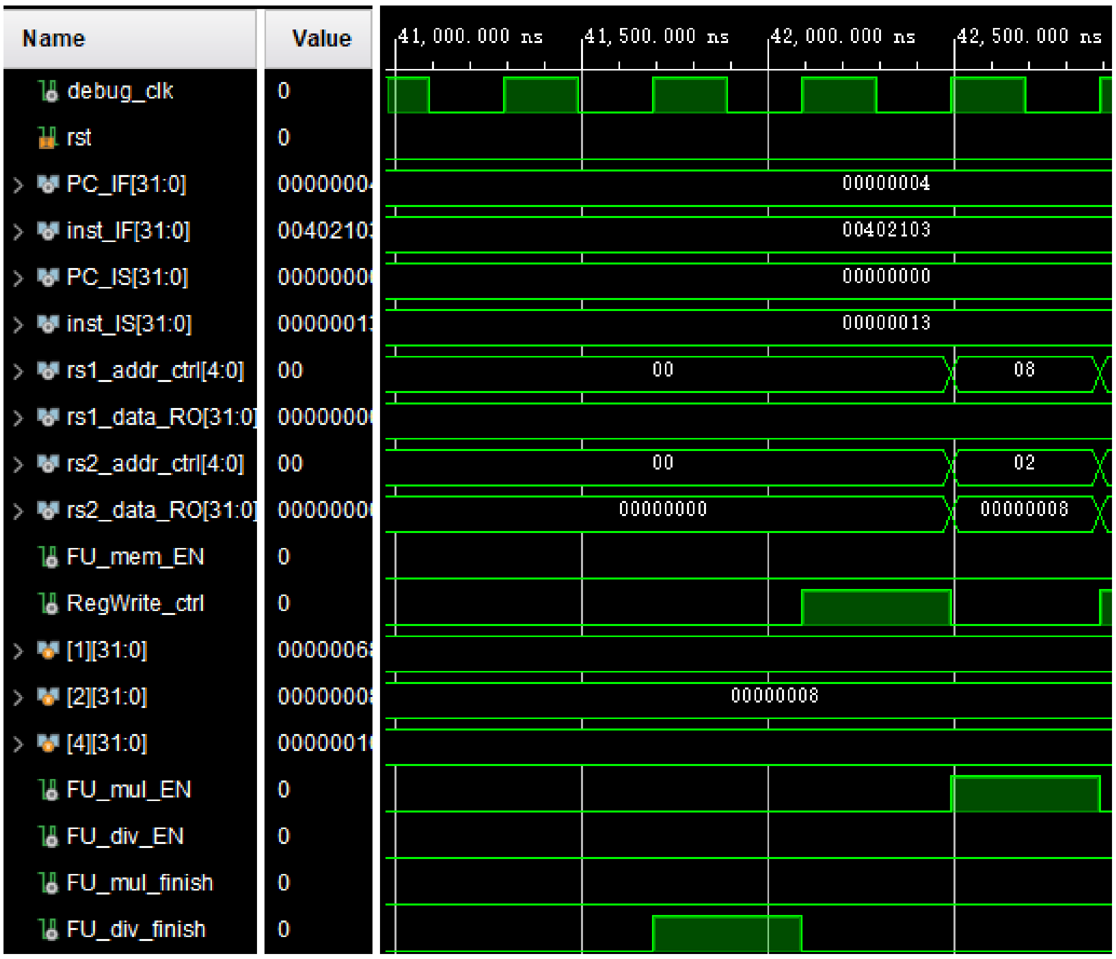
PC: 0x00000058, Inst: 0x025204b3, **mul x9, x4, x5**

PC: 0x0000005c, Inst: 0x022404b3, **mul x9, x8, x2**

PC: 0x00000060, Inst: 0x00400113, **addi x2, x0, x4**

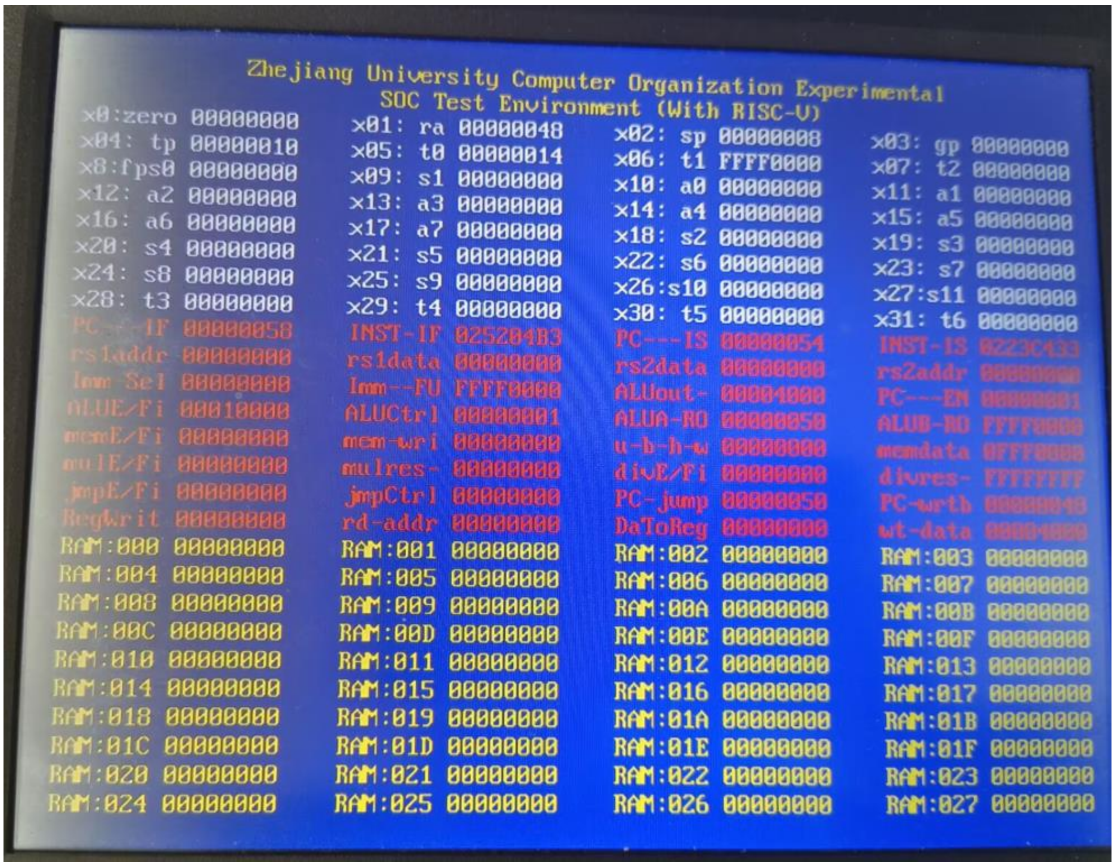
We could see the fourth instruction may produce a **WAR** hazard with the first instruction, so it has to wait to be dealt after the first instruction is finished.

However, the first instruction is a divide function, so it will cost very many cycles… In fact, it was finished in 41700-42100ns. Before it was finished, the two multiply operations have long been finished, and it is truly **out-of-order execution**.


Name
Value
41,000.000 ns
41,500.000 ns
42,000.000 ns
42,500.000 ns
debug_clk 0
rst 0
PC_IF[31:0] 0000000 00000004
inst_IF[31:0] 0040210 00402103
PC_IS[31:0] 0000000 00000000
inst_IS[31:0] 0000001 00000013
rs1_addr_ctrl[4:0] 00 00 08
rs1_data_RO[31:0] 0000000
rs2_addr_ctrl[4:0] 00 00 02
rs2_data_RO[31:0] 0000000 00000000 00000008
FU_mem_EN 0
RegWrite_ctrl 0
[1][31:0] 0000006
[2][31:0] 0000000 00000008
[4][31:0] 0000001
FU_mul_EN 0
FU_div_EN 0
FU_mul_finish 0
FU_div_finish 0

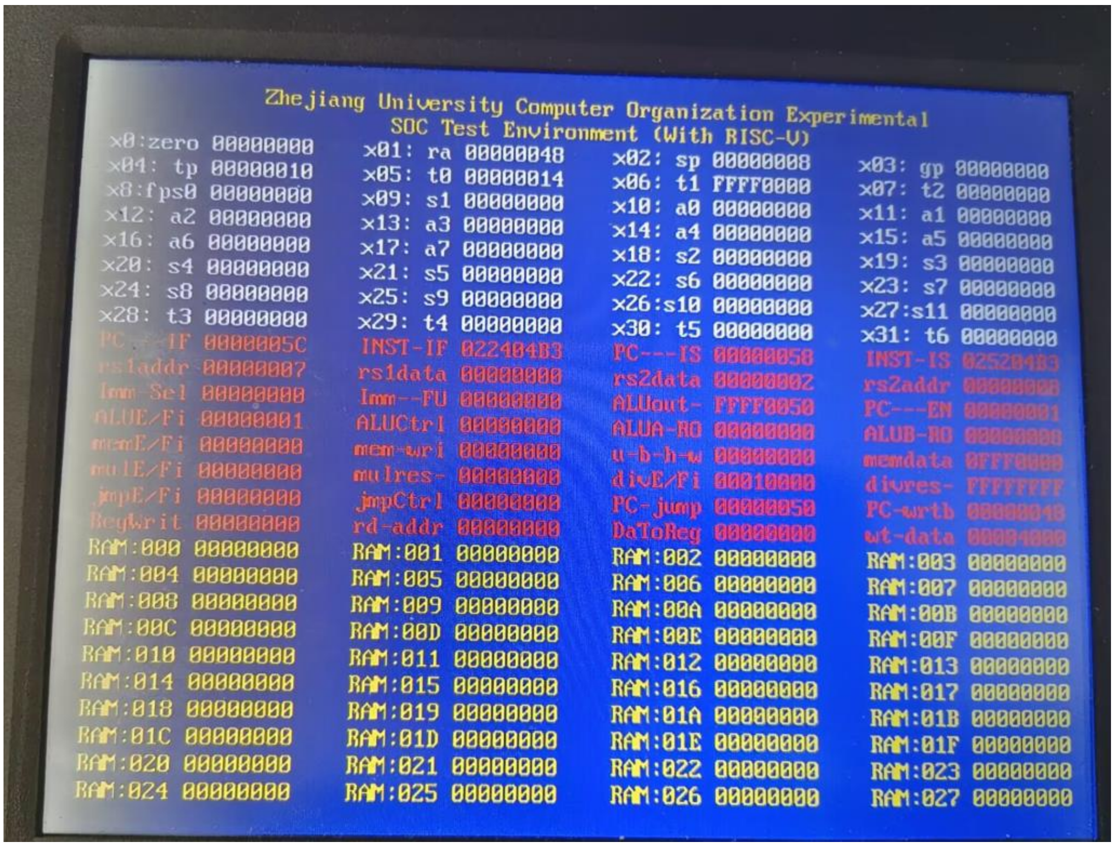
After that division finished, the ALU instruction finished in 42500-42900ns. A **WAR** hazard is avoided by enforcing order.

Following are the on-board results:



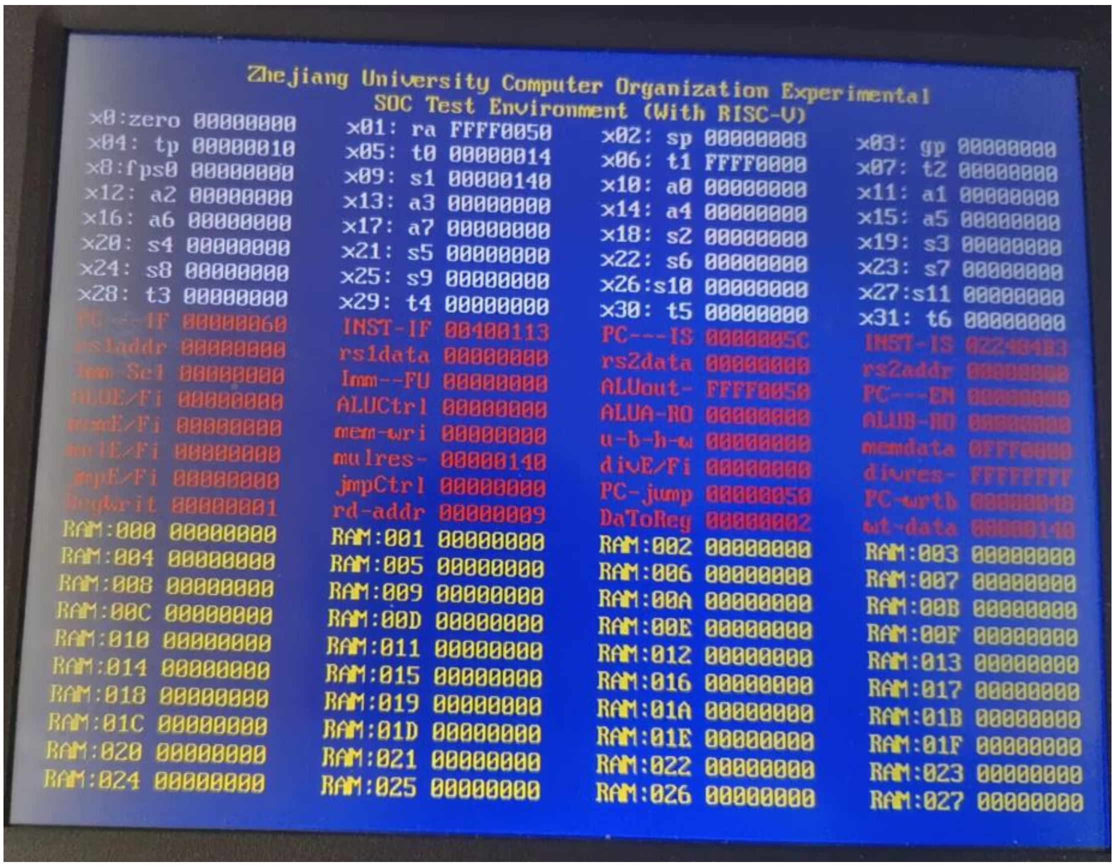
```
            Zhejiang University Computer Organization Experimental
                   SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000048   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000000   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
PC---IF 00000058   INST-IF 025204B3   PC---IS 00000054   INST-IS 0223C433
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000000   Imm--FU FFFF0000   ALUout- 00004000   PC---EN 00000001
ALUE/Fi 00010000   ALUCtrl 00000001   ALUA-RO 00000050   ALUB-RO FFFF0000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00000000   divres- FFFFFFFF
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000050   PC-wrtb 00000048
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data 00004000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

Issuing the **div** instruction.



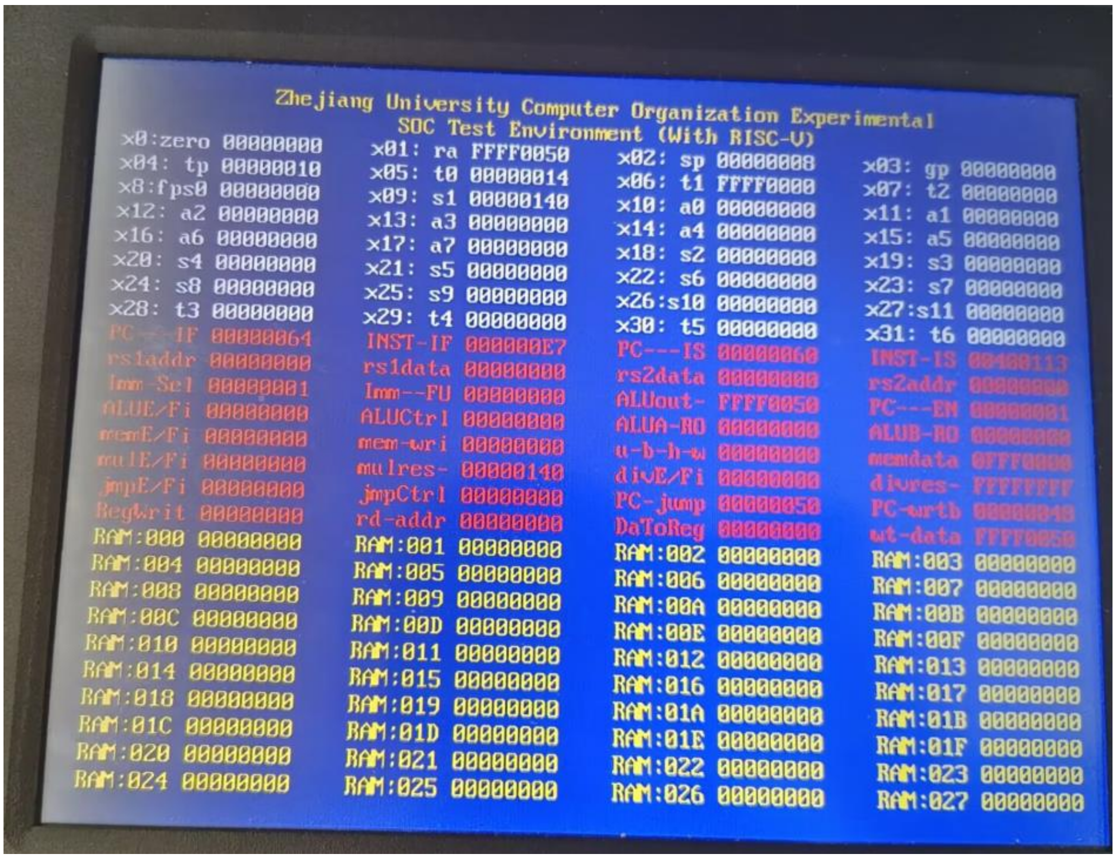
```
            Zhejiang University Computer Organization Experimental
                   SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000048   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000000   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
PC---IF 0000005C   INST-IF 022404B3   PC---IS 00000058   INST-IS 025204B3
rs1addr 00000007   rs1data 00000000   rs2data 00000002   rs2addr 00000000
Imm-Sel 00000000   Imm--FU 00000000   ALUout- FFFF0050   PC---EN 00000001
ALUE/Fi 00000001   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000000   divE/Fi 00010000   divres- FFFFFFFF
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000050   PC-wrtb 00000048
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data 00004000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

Issuing the first **mul** instruction.



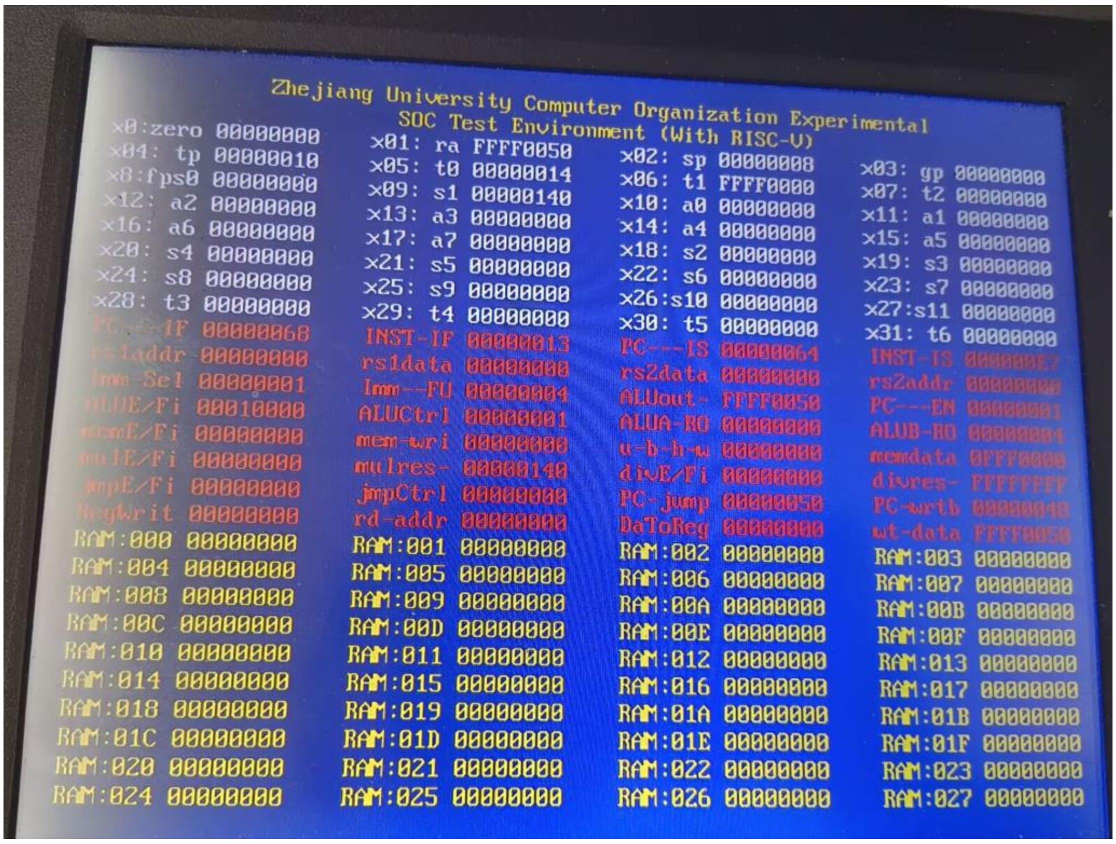
```
         Zhejiang University Computer Organization Experimental
                  SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra FFFF0050   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000140   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 00000060   INST-IF 00400113   PC---IS 0000005C   INST-IS 022404B3
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000000   Imm--FU 00000000   ALUout- FFFF0050   PC---EN 00000000
ALUE/Fi 00000000   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000140   divE/Fi 00000000   divres- FFFFFFFF
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000050   PC-wrtb 00000048
RegWrit 00000001   rd-addr 00000009   DaToReg 00000002   wt-data 00000140
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

The first **mul** writes the critical register.



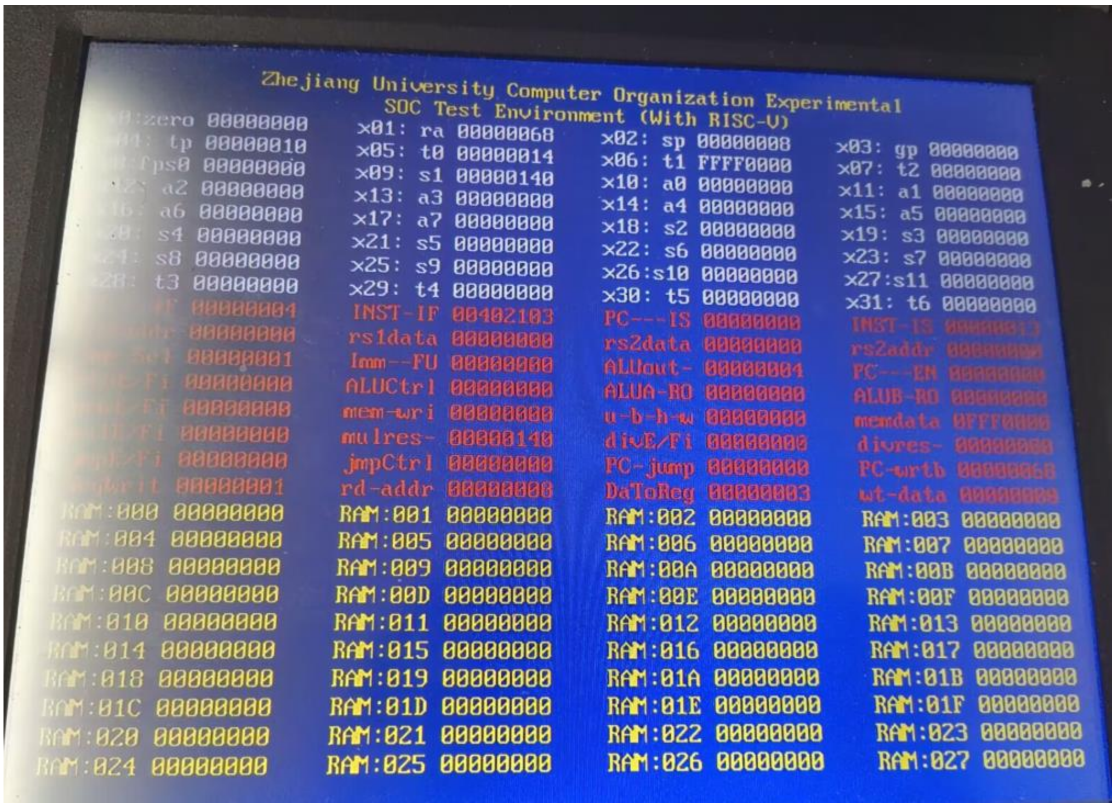
```
         Zhejiang University Computer Organization Experimental
                  SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra FFFF0050   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
x8:fps0 00000000   x09: s1 00000140   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 00400113
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000000   ALUout- FFFF0050   PC---EN 00000001
ALUE/Fi 00000000   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000140   divE/Fi 00000000   divres- FFFFFFFF
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000050   PC-wrtb 00000048
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data FFFF0050
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

After 1 cycle the second **mul** issued.



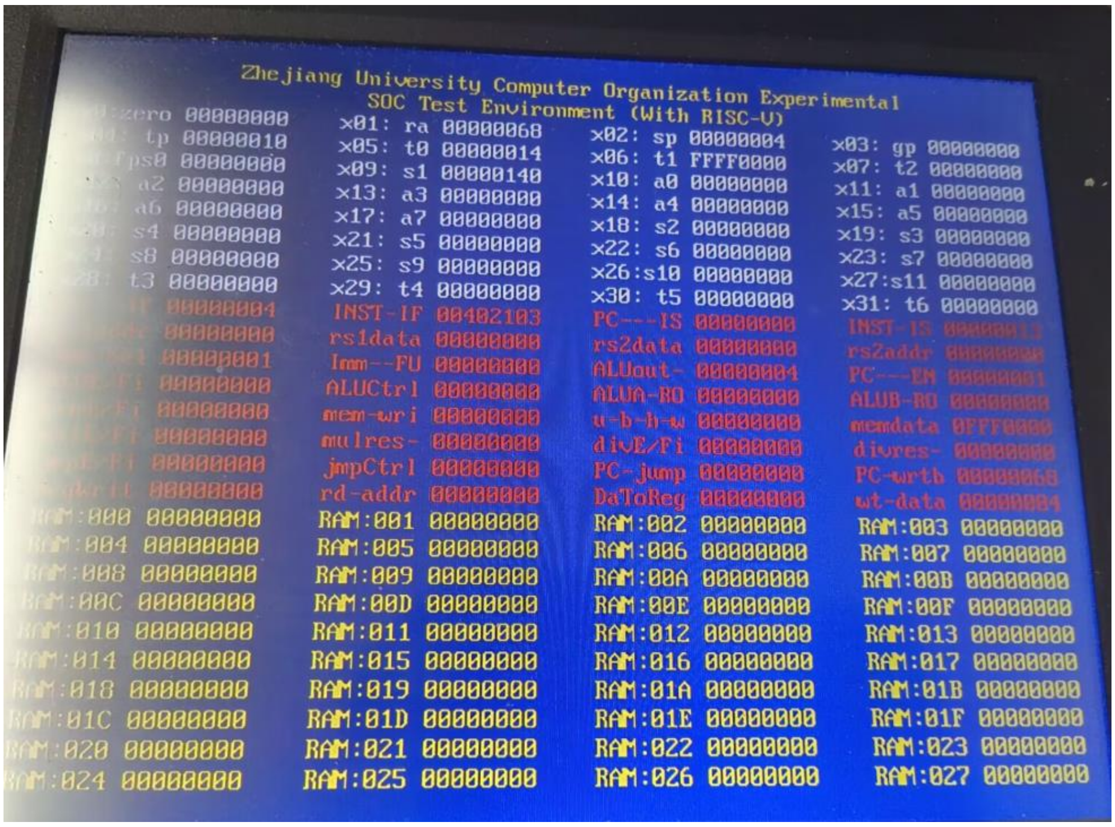
```
                 Zhejiang University Computer Organization Experimental
                         SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra FFFF0050   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
x8:fps0 00000000   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 00000068   INST-IF 00000013   PC---IS 00000064   INST-IS 000000E7
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000004   ALUout- FFFF0050   PC---EN 00000001
ALUE/Fi 00010000   ALUCtrl 00000001   ALUA-RO 00000000   ALUB-RO 00000004
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000140   divE/Fi 00000000   divres- FFFFFFFF
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000050   PC-wrtb 00000048
RegWrit 00000000   rd-addr 00000000   DaToReg 00000000   wt-data FFFF0050
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

As arithmetic unit was not occupied, the **addi** instruction is issued.



```
                 Zhejiang University Computer Organization Experimental
                         SOC Test Environment (With RISC-V)
x0:zero 00000000   x01: ra 00000068   x02: sp 00000008   x03: gp 00000000
x04: tp 00000010   x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
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 00000004   INST-IF 00402103   PC---IS 00000000   INST-IS 00000013
rs1addr 00000000   rs1data 00000000   rs2data 00000000   rs2addr 00000000
Imm-Sel 00000001   Imm--FU 00000000   ALUout- 00000004   PC---EN 00000000
ALUE/Fi 00000000   ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO 00000000
memE/Fi 00000000   mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
mulE/Fi 00000000   mulres- 00000140   divE/Fi 00000000   divres- 00000000
jmpE/Fi 00000000   jmpCtrl 00000000   PC-jump 00000000   PC-wrtb 00000068
RegWrit 00000001   rd-addr 00000000   DaToReg 00000003   wt-data 00000000
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

However, after very many cycles, the **div** instruction finally finished.



```
            Zhejiang University Computer Organization Experimental
                     SOC Test Environment (With RISC-V)
[illegible]:zero 00000000   x01: ra 00000068   x02: sp 00000004   x03: gp 00000000
[illegible]: tp 00000010    x05: t0 00000014   x06: t1 FFFF0000   x07: t2 00000000
[illegible]fps0 00000000    x09: s1 00000140   x10: a0 00000000   x11: a1 00000000
[illegible] a2 00000000     x13: a3 00000000   x14: a4 00000000   x15: a5 00000000
[illegible] a6 00000000     x17: a7 00000000   x18: s2 00000000   x19: s3 00000000
[illegible] s4 00000000     x21: s5 00000000   x22: s6 00000000   x23: s7 00000000
[illegible] s8 00000000     x25: s9 00000000   x26:s10 00000000   x27:s11 00000000
[illegible] t3 00000000     x29: t4 00000000   x30: t5 00000000   x31: t6 00000000
[illegible]IF 00000004      INST-IF 00402103   PC---IS 00000000   INST-IS [illegible]
[illegible] 00000000        rs1data 00000000   rs2data 00000000   rs2addr [illegible]
[illegible] 00000001        Imm--FU 00000000   ALUout- 00000004   PC---EM [illegible]
[illegible] 00000000        ALUCtrl 00000000   ALUA-RO 00000000   ALUB-RO [illegible]
[illegible] 00000000        mem-wri 00000000   u-b-h-w 00000000   memdata 0FFF0000
[illegible] 00000000        mulres- 00000000   divE/Fi 00000000   divres- [illegible]
[illegible] 00000000        jmpCtrl 00000000   PC-jump 00000000   PC-wrtb [illegible]
[illegible] 00000000        rd-addr 00000000   DaToReg 00000000   wt-data 00000004
RAM:000 00000000   RAM:001 00000000   RAM:002 00000000   RAM:003 00000000
RAM:004 00000000   RAM:005 00000000   RAM:006 00000000   RAM:007 00000000
RAM:008 00000000   RAM:009 00000000   RAM:00A 00000000   RAM:00B 00000000
RAM:00C 00000000   RAM:00D 00000000   RAM:00E 00000000   RAM:00F 00000000
RAM:010 00000000   RAM:011 00000000   RAM:012 00000000   RAM:013 00000000
RAM:014 00000000   RAM:015 00000000   RAM:016 00000000   RAM:017 00000000
RAM:018 00000000   RAM:019 00000000   RAM:01A 00000000   RAM:01B 00000000
RAM:01C 00000000   RAM:01D 00000000   RAM:01E 00000000   RAM:01F 00000000
RAM:020 00000000   RAM:021 00000000   RAM:022 00000000   RAM:023 00000000
RAM:024 00000000   RAM:025 00000000   RAM:026 00000000   RAM:027 00000000
```

The **addi** instruction began to execute after that, and legally write **x2**.

5 Discussion and Conclusion

5.1 Problems

5.1.1 Problems concerning the Experiment Guides

Problem 1. Not Represented WAW Hazards

In fact, the **WAW** hazard is not represented in the given program.

All **WAW** hazards given in the program seemly is with a **structural hazard**, so we have to use a not so perfect example to explain how to avoid WAW hazard.

This may need to be updated afterwards.

5.2 Achievements and conclusion

In this experiment, I implemented a float-point CPU supporting multicycle operations and scoreboarding. I also relearned the knowledge of scoreboarding theory and its architectural support. Overall, the experiment is successful and educational.