



## 1. 实验目的和要求

---

### Purpose

- Understand the principle of pipelines that support multicycle operations.
- Understand the principle of Dynamic Scheduling With a Scoreboard.
- Master the design methods of pipelines that support multicycle operations.
- Master the design methods of Dynamically Scheduled Pipelines using Scoreboarding.
- Master verification methods of Dynamically Scheduled Pipelines using Scoreboarding.

### Task

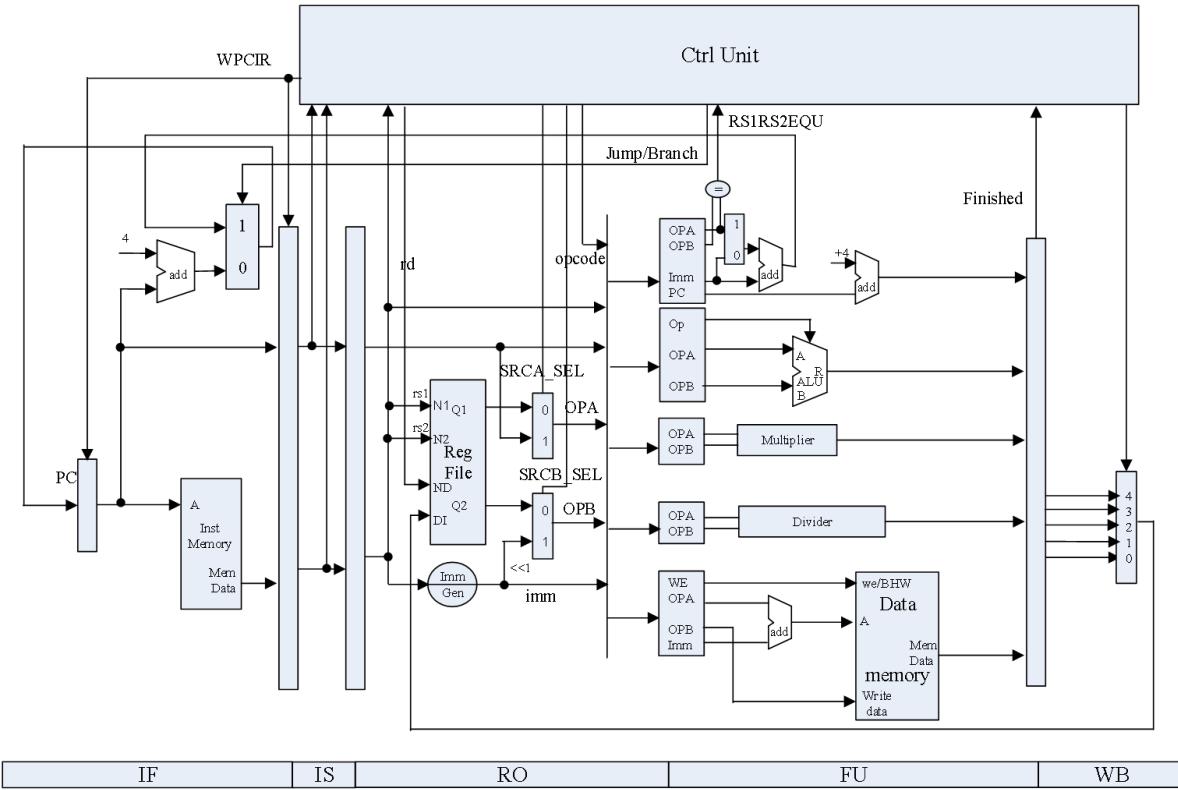
- Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.
- Design of a scoreboard and integrate it to CPU.
- Verify the Pipelined CPU with program and observe the execution of program.

## 2. 实验内容和原理

---

### Pipeline

The whole system is divided into 5 stages: Instruction Fetch, Instruction Issue, Read Operands, Function Unit and Writeback. Comparing to the previous pipeline in lab 5, this one allows more parallelism, that is, out-of-order execution. Scoreboarding technique is adopted here. Background knowledge is introduced in class, we can refer to the slides for help.



In this experiment, we should implement the control unit in the pipeline, which controls the functionality of the pipeline.

### 3. 实验过程和数据记录

There is only one module to be filled out.

#### CtrlUnit.v

The first signal `normal_stall` indicates whether there is a structural hazard or WAW. If the function unit to be used is busy now, there is a structural hazard, which is judged by the first four clauses. The last clause judges if the destination register will be written by the previous instructions currently in other function units.

```
// normal_stall: structural hazard or WAW
assign normal_stall = (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) |
    (|RRS[dst] & dst != 0);           //fill sth. here
```

We judge a WAR. The analysis of all funciton units is similar, so we only take ALU as an example. If ALU is going to write  $x_0$ , there is no WAR for sure. Otherwise, if the destination register of ALU is the same as the source register of previous instructions, which reside in other function unit, and the value of such register hasn't been read by other funciton units, there is a WAR.

Here, the transition of `RDY` bit is noteworthy. In the beginning, set to 0 if the source register is not ready to be read. Then, if the source register is ready to be read but hasn't been read, `RDY` is set to 1. Finally, if the ready value is read, `RDY` is set to 0 again. To prevent WAR, we need to ensure that an FU has read the value, so we need to judge whether `RDY` is 0 here. Three states share only one bit here will not cause any problem since `ALU_WAR` will be used together with `DONE` bit in the following instructions.

```
// 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][`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] )
) || |FUS[`FU_ALU][`DST_H:`DST_L];
```

The following block maintains all kinds of information for the pipeline. If stage RO is enabled, we should record the information of the current instruction in RRS, FUS and IMM.

In stage RO, if both source registers are ready, values will be read in another block, and we should set `RDY` bits to 0 here, which indicate that the value has been read.

In stage FU, each function unit will be working if it is assigned an instruction with all operands. Once the execution of a function unit is done, a signal will be sent back to control unit. Here, if such signal is received, we should set `DONE` bit to 1, which indicates that the execution of one function unit is done. However, if such signal is 0, the `DONE` bit should remain unchanged.

In stage WB, the last stage of an instruction, the information recorded in FUS and RRS should be cleared. Also, `RDY` bits of the following instructions should be set to 1 for the pipeline to continue working.

```
// maintain the table
always @ (posedge clk or posedge rst) begin
  if (rst) begin
    // reset the scoreboard
    for (i = 0; i < 32; i = i + 1) begin
      RRS[i] <= 3'b0;
    end

    for (i = 1; i <= 5; i = i + 1) begin
      FUS[i] <= 32'b0;
      IMM[i] <= 32'b0;
    end
  end
end
```

```

else begin
    // IS issue
    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] <= rdyl;
        FUS[use_FU][`RDY2] <= rdy2;

        IMM[use_FU] <= imm;
        PCR[use_FU] <= PC;
    end

    // RO read operands
    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
//fill sth. here.

        // 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
//fill sth. here.

        // 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
//fill sth. here.

        // 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
//fill sth. here.

        // DIV
        FUS[`FU_DIV][`RDY1] <= 1'b0;
        FUS[`FU_DIV][`RDY2] <= 1'b0;
    end

    // EX execute
    FUS[`FU_ALU][`FU_DONE] <= ALU_done ? 1'b1 : FUS[`FU_ALU][`FU_DONE];
//fill sth. here
    FUS[`FU_MEM][`FU_DONE] <= MEM_done ? 1'b1 : FUS[`FU_MEM][`FU_DONE];
    FUS[`FU_MUL][`FU_DONE] <= MUL_done ? 1'b1 : FUS[`FU_MUL][`FU_DONE];
    FUS[`FU_DIV][`FU_DONE] <= DIV_done ? 1'b1 : FUS[`FU_DIV][`FU_DONE];

```

```

FUS[`FU_JUMP][`FU_DONE] <= JUMP_done ? 1'b1 : FUS[`FU_JUMP][`FU_DONE];

// WB write back
if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
    // clear everything
    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;

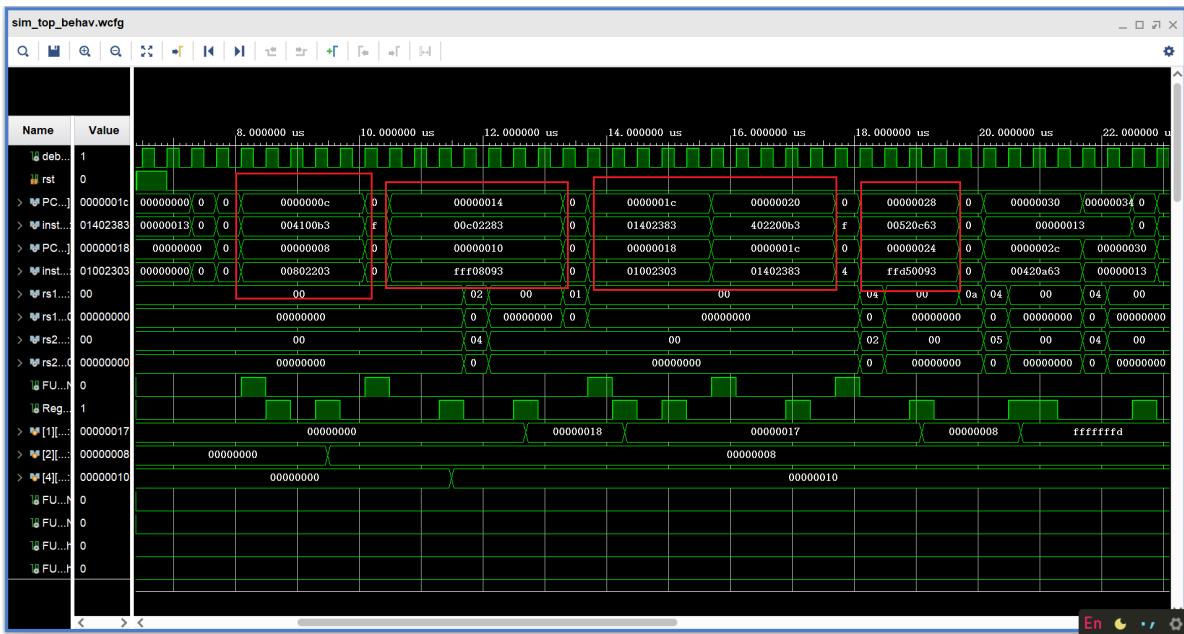
    if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_ALU][`RDY2] <=
1'b1;
    if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MEM][`RDY2] <=
1'b1;
    if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MUL][`RDY2] <=
1'b1;
    if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_DIV][`RDY2] <=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

```

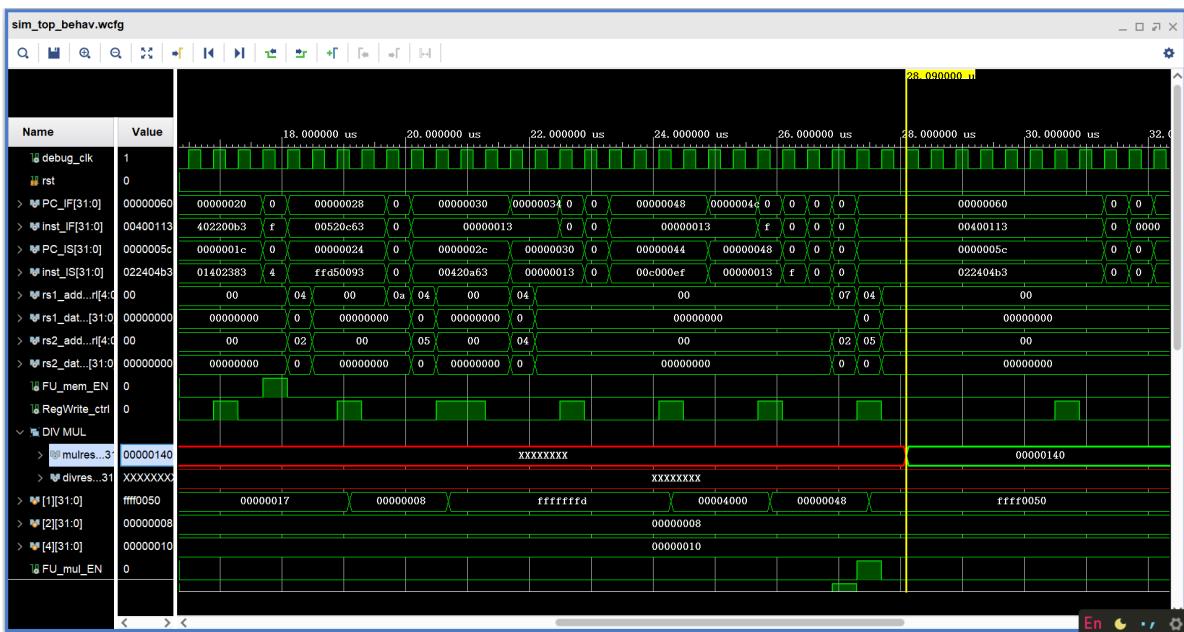
## 4. 实验结果分析

### Simulation

As the simulation shows, whenever there is a hazard or WAW, the pipeline stops issuing. Check the value of each signal, we can find the values are correct.



We can see although the mul instruction comes following the div instruction, the result of multiplication comes out first. This is because there is no dependency and they can be both issued without stalling. Multiplication requires less cycles so the result of multiplication comes out first.



The other part of the result is the same as that given in the guidance, so is omitted here.

## Program Device

When the first load instruction is issued, we can see that the pipeline stalls because of a structural hazard. From the value of signals, we can infer that FU memory is generating immediate.

```

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
x08: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- 00000140  divE/Fi 00000000  divres- FFFFFFFF
jmpE/Fi 00000000  jmpCtrl 00000000  PC-jump 00000000  PC-wrtb 00000068
RegWrit 00000001  rd-addr 00000000  DaToReg 00000000  wt-data 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-10 00000000  CODE-0B 00000000
CODE-0C 00000000  CODE-11 00000000  CODE-12 00000000  CODE-0F 00000000
CODE-10 00000000  CODE-13 00000000  CODE-14 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

```

Loading continues. From the value of signals, we can infer that FU memory is reading from memory.

```

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
x08: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- 00000140  divE/Fi 00000000  divres- FFFFFFFF
jmpE/Fi 00000000  jmpCtrl 00000000  PC-jump 00000000  PC-wrtb 00000068
RegWrit 00000001  rd-addr 00000000  DaToReg 00000000  wt-data 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-10 00000000  CODE-0B 00000000
CODE-0C 00000000  CODE-11 00000000  CODE-12 00000000  CODE-0F 00000000
CODE-10 00000000  CODE-13 00000000  CODE-14 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

```

Loading continues. From the value of signals, we can infer that FU memory is writing value to destination register. We can see the value read out at `memdata` is 0x08, which is correct.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x8: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 00410003  PC---IS 00000000  INST-IS 000002203
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- 00000140  divE/Fi 00000000  divres- FFFFFFFF
jmpE/Fi 00000000  jmpCtrl 00000000  PC-jump 00000000  PC-wrb 00000000
ReqWrit 00000001  rd-addr 00000002  DaToReg 00000001  wt-data 00000000
CODE-00 00000000  not in us
CODE-04 00000000  not in us
CODE-08 00000000  not in us
CODE-0C 00000000  not in us
CODE-10 00000000  not in us
CODE-14 00000000  CODE-15 00000000  CODE-16 00000000  CODE-03 00000000
CODE-18 00000000  CODE-19 00000000  CODE-1A 00000000  CODE-07 00000000
CODE-1C 00000000  CODE-1D 00000000  CODE-1E 00000000  CODE-0B 00000000
CODE-20 00000000  CODE-21 00000000  CODE-22 00000000  CODE-0F 00000000
CODE-24 00000000  CODE-25 00000000  CODE-26 00000000  CODE-13 00000000
                                         not in uss CODE-17 00000000
                                         not in uss CODE-1B 00000000
                                         not in uss CODE-1F 00000000
                                         not in uss CODE-23 00000000
                                         not in uss CODE-27 00000000

```

Then, at the beginning of the mul instruction, we can see the result is cleared.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x8:zero 00000000  x01: ra FFFF0050  x02: sp 00000000  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 00000060  INST-IF 00400113  PC---IS 00000005C  INST-IS 02240483
rs1addr 00000004  rs1data 00000010  rs2data 00000005  rs2Addr 00000014
Imm Sel 00000000  Imm--FU 00000000  ALUout- FFFF0050  PC---EN 00000000
ALUE/Fi 00000000  ALUCtrl 00000000  ALUA-RO 00000010  ALUB-RO 00000014
memE/Fi 00000000  mem-wri 00000000  u-b-h-w 00000000  memdata 0FFF0000
mulE/Fi 00001000  mulres- 00000000  divE/Fi 00000001  divres- 00000000
jmpE/Fi 00000000  jmpCtrl 00000000  PC-jump 00000050  PC-wrb 00000048
ReqWrit 00000001  rd-addr 00000001  DaToReg 00000000  wt-data FFFF0050
CODE-00 00000000  not in us
CODE-04 00000000  not in us
CODE-08 00000000  not in us
CODE-0C 00000000  not in us
CODE-10 00000000  not in us
CODE-14 00000000  CODE-15 00000000  CODE-16 00000000  CODE-03 00000000
CODE-18 00000000  CODE-19 00000000  CODE-1A 00000000  CODE-07 00000000
CODE-1C 00000000  CODE-1D 00000000  CODE-1E 00000000  CODE-0B 00000000
CODE-20 00000000  CODE-21 00000000  CODE-22 00000000  CODE-0F 00000000
CODE-24 00000000  CODE-25 00000000  CODE-26 00000000  CODE-13 00000000
                                         not in uss CODE-17 00000000
                                         not in uss CODE-1B 00000000
                                         not in uss CODE-1F 00000000
                                         not in uss CODE-23 00000000
                                         not in uss CODE-27 00000000

```

Multiplication continues for several cycles and we can see the result at `mulres` is 0x140, which is consistent with the simulation result.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-U)

x0:zero 00000000 x01: ra FFFF0050 x02: sp 00000008 x03: gp 00000000
x04: tp 00000010 x05: t0 00000014 x06: t1 FFFF0000 x07: t2 00000000
x08: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 022401D3
rs1addr 00000000 rs1data 00000000 rs2data 00000000 rs2Addr 00000000
Imm Sel 00000000 Imm-FU 00000000 ALUout- FFFF0050 PC---EM 00000000
ALUE/Fi 00000000 ALUCtrl 00000000 ALUA-RO 00000000 ALUB-RO 00000000
memE/Fi 00000000 mem-uri 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 00000050 PC-wrtb 00000048
Beqleit 00000000 rd-addr 00000000 DaToReg 00000000 wt-data FFFF0050
CODE-00 00000000 CODE-01 00000000 CODE-02 00000000 CODE-03 00000000
04 00000000 CODE-04 00000000 CODE-05 00000000 CODE-06 00000000
08 00000000 CODE-07 00000000 CODE-08 00000000 CODE-09 00000000
0C 00000000 CODE-10 00000000 CODE-11 00000000 CODE-12 00000000
10 00000000 CODE-13 00000000 CODE-14 00000000 CODE-15 00000000
14 00000000 CODE-16 00000000 CODE-17 00000000 CODE-18 00000000
18 00000000 CODE-19 00000000 CODE-1A 00000000 CODE-1B 00000000
1C 00000000 CODE-1D 00000000 CODE-1E 00000000 CODE-1F 00000000
20 00000000 CODE-21 00000000 CODE-22 00000000 CODE-23 00000000
24 00000000 CODE-24 00000000 CODE-25 00000000 CODE-26 00000000
CODE-27 00000000
not in us

```

## 5. 讨论与心得

This experiment is a relatively difficult one. The organization of the code is not that clear as other experiments. Also, compared to the scoreboard introduced in class, this pipeline introduce a `DONE` bit, which works slightly differently from `BUSY` bit.

As for me, the three states of `RDY` bit causes me some trouble. When I was coding for WAR, I didn't figure out that and I used

```
FUS[FU_MEM][RDY1]
```

rather than

```
!FUS[FU_MEM][RDY1]
```

This makes the pipeline works abnormally. After figuring out the transition of `RDY`, as explained above, I finally corrected this error.