浙江大学

本科实验报告

课程名称：	计算机体系结构
姓　名：	夏尤楷
学　院：	计算机科学与技术学院
系：	计算机科学与技术
专　业：	计算机科学与技术
学　号：	3210104331
指导教师：	陈文智

2024 年 1 月 10 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：　用计分牌实现动态规划流水线　

学生姓名：夏尤楷 专业：计算机科学与技术 学号：3210104331

同组学生姓名：　来思锐　指导老师：　陈文智　

实验地点：　曹西 301　实验日期：　2023　年 12 月 12 日

一、　实验目的和要求

1. 理解支持多周期指令的流水线原理。
2. 理解带计分牌的动态规划原理。
3. 掌握支持多周期指令的流水线的设计方法。
4. 掌握用计分牌的动态规划流水线的设计方法。
5. 掌握用计分牌的动态规划流水线的验证方法。

二、　实验内容和原理

内容：

1. 重新设计支持多周期指令的带 IF/IS/RO/FU/WB 阶段的流水线。
2. 设计一个计分板并将其加入 CPU。
3. 用程序验证流水线 CPU 并观察程序执行。

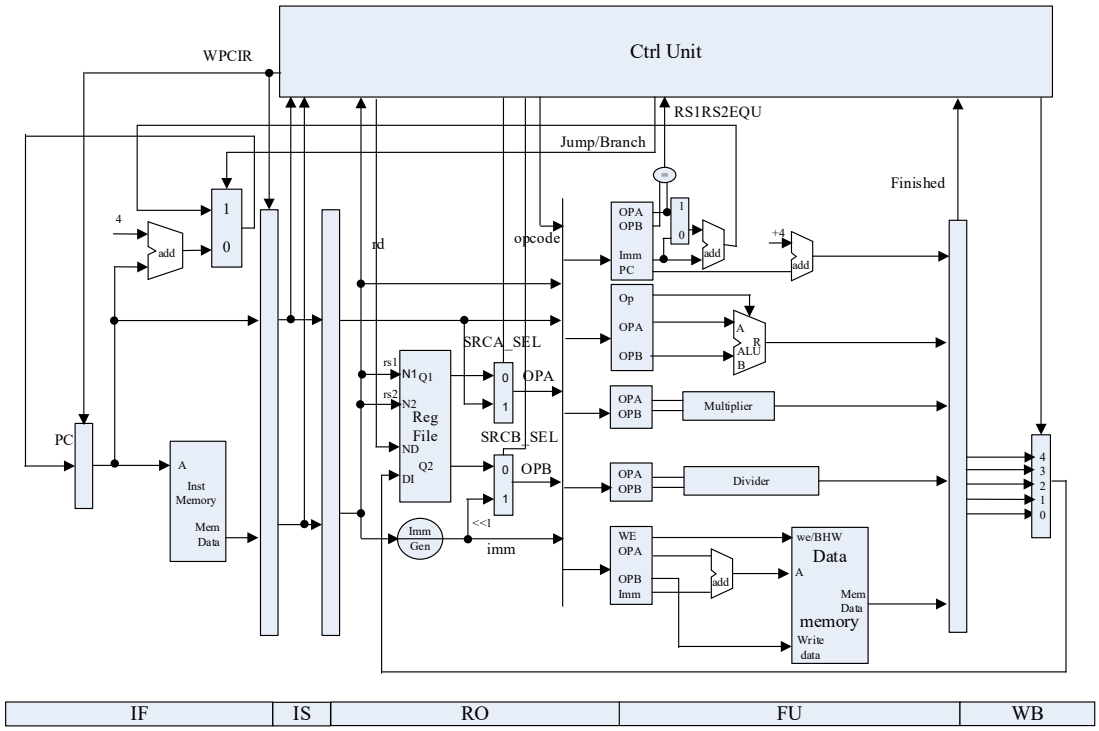
原理：

各阶段需要等待计分牌相关变量满足某些条件后，才能执行该阶段，并设置某些计分牌变量，以避免和处理各种冲突（结构冲突、WAW、WAR、RAW），保证流水线的正确运行。相关条件如下表：

Status	Wait until	Bookkeeping
Issue (Structural and WAW)	!busy[FU] && !result[RD]	busy[FU] = 1, op[FU] = OP, dst[FU] = RD, src1[FU] = RS1, src2[FU] = RS2, read1[FU] = result[RS1], read2[FU] = result[RS2], wait1[FU] = !read1[FU], wait2[FU] = !read2[FU], result[RD] = FU, done[FU] = 0

Read operands (RAW)	wait1[FU] && wait2[FU]	wait1[FU] = 0, wait2[FU] = 0
Function unit	Function unit completes its work	done[FU] = 1
Write back (WAR)	∀f, !(src1[f] == dst[FU] && wait1[FU] || src2[f] == dst[FU] && wait2[FU]) or similarly ∀f, (src1[f] != dst[FU] || !wait1[FU]) && (src2[f] != dst[FU] || !wait2[FU])	∀f, if read1[f] == FU then wait1[f] = 1, if read2[f] == FU then wait2[f] = 1. result[dst[FU]] = 0, busy[FU] = 0

整个 CPU 的结构基本如下图：



三、 实验过程和数据记录

本次实验只需修改 CtrlUnit.v 文件。

1. IS 阶段

normal_stall 就是控制 IS 阶段不发生结构冲突和 WAW 的变量，当它为 1 时，IS_en 会被置零，使得 IS 阶段停顿。所以完善 normal_stall 的赋值如下：

```
assign normal_stall = (use_FU == 3'b0 ? 1'b0 : FUS[use_FU][`BUSY]) |
                      (RRS[dst] != `FU_BLANK);          //fill sth.
here
```

根据表格，补充设计计分牌变量的代码如下：

```
// 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;
            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
```

2. RO 阶段

根据表格，完善代码如下：

```
// 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
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;                        //fill sth. here.
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;                        //fill sth. here.
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;                        //fill sth. here.
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;                        //fill sth. here.
end
```

3. FU 阶段

功能模块计算完成后，产生的完成信号（1’b1）从功能模块传至该模块中。这个信号只会持续一个周期，在这个信号的驱动下，功能模块的计算结果被写回。但由于各种原因，功能模块的计算结果可能并没有办法在这个周期内被写回，而是要到之后的某个周期才会被写回，因此要保持相应的计算完成信号。因此，该功能模块的保存信号的寄存器应当设为：当其中的值为 0 时，用外来的信号值更新该寄存器内值；当其中值为 1’b1 时，则停止更新，直到该功能模块再次执行完 IS 阶段后，这个寄存器内的值才会在初始化时更新为 0。代码如下：

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

4. WB 阶段

为每个功能模块分配一个 WAR 控制信号，该信号为 1 时，功能模块的 WB 阶段才能执行。根据表格，这些信号的赋值代码如下：

```
// 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])  & //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
);

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

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

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

    wire JUMP_WAR = (
    (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_ALU][`RDY1]) & //fill sth. here
    (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_ALU][`RDY2]) & //fill sth. here
    (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MEM][`RDY1]) & //fill sth. here
    (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MEM][`RDY2]) & //fill sth. here
    (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MUL][`RDY1]) & //fill sth. here
    (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MUL][`RDY2]) & //fill sth. here
```

```
    (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_DIV][`RDY1]) & //fill sth. here
    (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_DIV][`RDY2])    //fill sth. here
);
```

设置某些计分牌变量的代码如下（其中处理了 RAW 冲突）：

```
    // WB
    // JUMP
    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; //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
    // ALU
    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_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][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_JUMP][`RDY1] <= 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;
        if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_JUMP][`RDY2] <= 1'b1;
    end //fill sth. here
    // MEM
    else 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_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][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_JUMP][`RDY1] <= 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;
        if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_JUMP][`RDY2] <= 1'b1;
    end //fill sth. here
    // MUL
    else 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_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][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_JUMP][`RDY1] <= 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;
```

```
    if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_JUMP][`RDY2] <= 1'b1;
end //fill sth. here
// DIV
else 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_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][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_JUMP][`RDY1] <= 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;
    if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_JUMP][`RDY2] <= 1'b1;
end //fill sth. here
```

四、 实验结果分析

本实验运行的代码如下表：

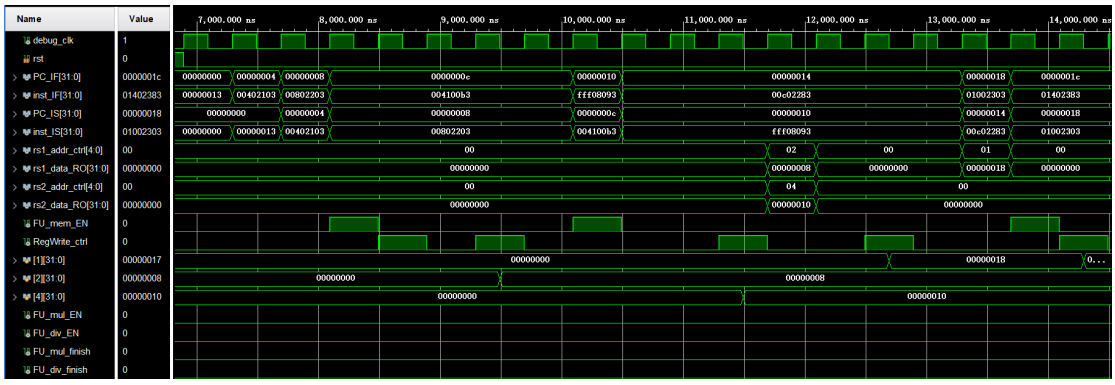
NO.	Instruction	Addr.	Label	ASM	Comment
0	00000013	0	__start:	addi x0, x0, 0	Str.(in 2nd round with 60)
1	00402103	4		lw x2, 4(x0)	
2	00802203	8		lw x4, 8(x0)	Structural Haz.(0x4)
3	004100b3	C		add x1, x2, x4	
4	fff08093	10		addi x1, x1, -1	Struct.,WAW(with 0xC)
5	00c02283	14		lw x5, 12(x0)	
6	01002303	18		lw x6, 16(x0)	
7	01402383	1C		lw x7, 20(x0)	Done at the same time Write respectively
8	402200b3	20		sub x1, x4, x2	
9	ffd50093	24		addi x1, x10,-3	
10	00520c63	28		beq x4,x5,label0	
11	00420a63	2C		beq x4,x4,label0	
12	00000013	30		addi x0, x0, 0	not executed
13	00000013	34		addi x0, x0, 0	not executed
14	00000013	38		addi x0, x0, 0	not executed
15	00000013	3C		addi x0, x0, 0	not executed
16	000040b7	40	label0:	lui x1, 4	
17	00c000ef	44		jal x1, 12	WAW(with 0x40)
18	00000013	48		addi x0, x0, 0	not executed
19	00000013	4C		addi x0, x0, 0	not executed
20	ffff0097	50		auipc x1, 0xffff0	
21	0223c433	54		div x8, x7, x2	
22	025204b3	58		mul x9, x4, x5	
23	022404b3	5C		mul x9, x8, x2	Struct.,WAW(58);RAW(54)
24	00400113	60		addi x2, x0, 4	WAR(with 0x5C)
25	000000e7	64		jalr x1, 0(x0)	
26	00000013	68		addi x0, x0, 0	not executed

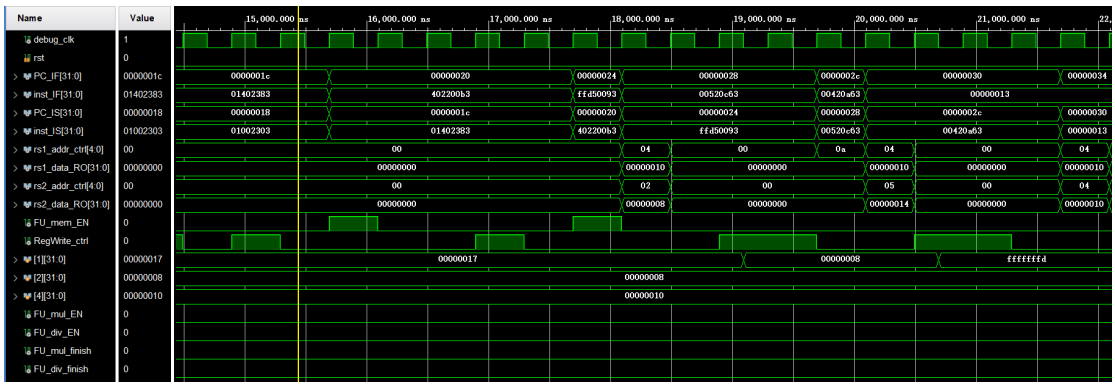
27	00000013	6C		addi x0, x0, 0	not executed

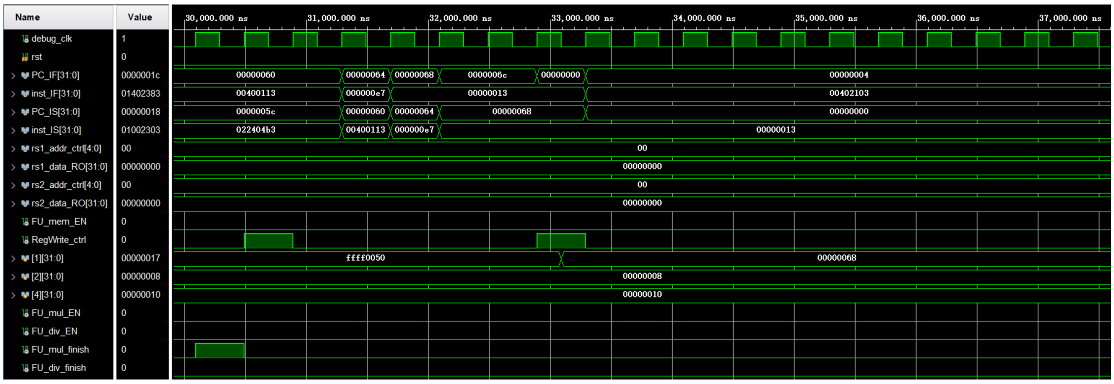
本次实验使用的数据如下表：

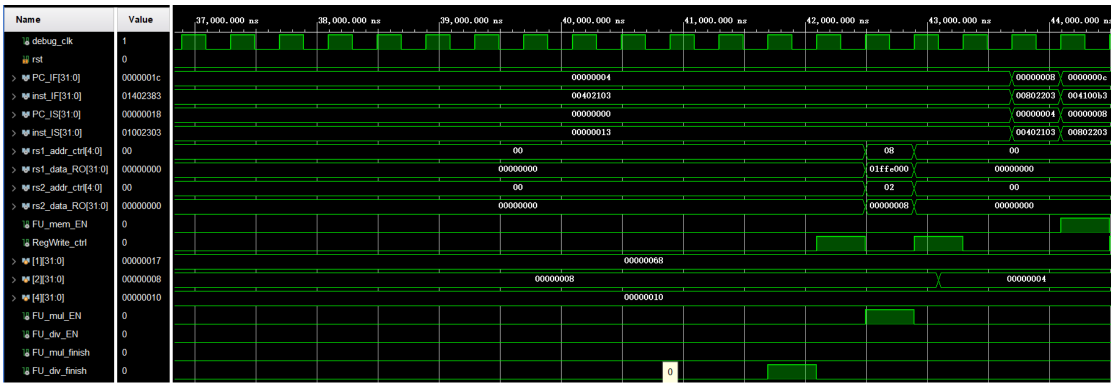
NO.	Data	Addr.	NO.	Instruction	Addr.
0	000080BF	0	16	00000000	40
1	00000008	4	17	00000000	44
2	00000010	8	18	00000000	48
3	00000014	C	19	00000000	4C
4	FFFF0000	10	20	A3000000	50
5	0FFF0000	14	21	27000000	54
6	FF000F0F	18	22	79000000	58
7	F0F0F0F0	1C	23	15100000	5C
8	00000000	20	24	00000000	60
9	00000000	24	25	00000000	64
10	00000000	28	26	00000000	68
11	00000000	2C	27	00000000	6C
12	00000000	30	28	00000000	70
13	00000000	34	29	00000000	74
14	00000000	38	30	00000000	78
15	00000000	3C	31	00000000	7C

在 vivado 上运行仿真，结果图如下：









下面对仿真结果中的处理各种冲突的行为举例进行解释：

（1）结构冲突：6900ns~10100ns：流水线依序发射指令 32’h00000013（addi x0, x0, 0）、32’h00402103（lw x2, 4(x0)），分别占用了 ALU 和 MEM 功能单元（占用 MEM 单元后的一个时钟周期内 FU_mem_EN 变为 1’b1）。而对于下一条指令 32’h00802203（lw x4, 8(x0)），由于 MEM 功能单元已被占用，所以只能停机等待。8500ns~8900ns 时，RegWrite_ctrl 变为 1，在其中的时钟下降沿写回的寄存器是 x0，写回的数据是 ALU 功能单元的结果，之后释放 ALU 功能单元；9300ns~9700ns 时，RegWrite_ctrl 变为 1，在其中的时钟下降沿写回的寄存器是 x2，写回的数据是 MEM 功能单元的结果（仿真图内可见寄存器[2]的值变

为了地址 32’h4 处的值 32’h00000008），之后释放 ALU 功能单元，再过一个时钟周期，发射指令 lw x4, 8(x0)进入并占用 MEM 功能单元（之后的一个时钟周期内 FU_mem_EN 变为 1’b1）；

（2）RAW 冲突：10100ns~13300ns：先发射指令 32’h004100b3（add x1, x2, x4）进入并占用 ALU 功能单元，故下一条指令 32’fff08093（addi x1, x1, -1）不可发射。Add x1, x2, x4 和上一条指令 lw x4, 8(x0)存在 RAW 冲突。故须等待上一条指令写入寄存器后，add x1, x2, x4 才能读取操作数并执行。写入发生在 11300ns~11700ns RegWrite_ctrl = 1 期间的时钟下降沿（仿真图内可见寄存器[2]的值变为了地址 32’h8 处的值 32’h000000010），之后 add x1, x2, x4 才读取操作数并执行（从 x2, x4 中读取，故 rs1_addr_ctrl == 2，rs2_addr_ctrl == 4，读取到的数分别为 rs1_data_RO == 32’h00000008 和 rs2_data_RO == 32’h00000010，正确）。在 12500ns~12900ns 时，RegWrite_ctrl 变为 1，在其中的时钟下降沿写入的寄存器是 x1，写回的数据是 ALU 功能单元的结果（仿真图内可见寄存器[1]的值变为运算结果 32’h00000018，正确）。

（3）同时写入：17700ns~19700ns：在开始的两个时钟周期内，依序发射指令 32’h01402383（lw x7, 20(x0)）和 32’h402200b3（sub x1, x4, x2）分别进入并占用 MEM 和 ALU 功能模块（进入 MEM 功能模块之后的一个时钟周期内 FU_mem_EN 变为 1）。由于 lw 指令在等待 3 个周期后输出结果，而 sub 指令在等待 2 个周期后输出结果，所以它们刚好同时输出要写回寄存器的结果。于是我们看到 RegWrite == 1 的情况持续了两个时钟周期（18900ns~19700ns），在其中的第一个时钟下降沿，写回的寄存器的是 x1，写回的数据是 ALU 功能单元的结果（仿真图内可见寄存器[1]的值变为运算结果了 32’h0000008，正确）；在其中的第二个时钟下降沿，写回的寄存器是 x7，写回的数据是 MEM 功能单元的结果。

（4）WAW 冲突：23300ns~26100ns：在一开始的时钟周期内发射了指令 32’h000040b7（lui x1, 4），接下去的指令 32’h00c000ef（jal x1, 12）因为与上一条指令要写回的寄存器相同，均为 x1，所以在上一条指令写回之前，它们存在 WAW 冲突。此时 lui x1, 4 先占用寄存器 x1，在 24100ns~24500ns 时 RegWrite_ctrl == 1 时，在其中的时钟下降沿往 x1 写入 lui 指令的结果（仿真图内可见寄存器

[1]的值变成了 32’h00004000，这个结果是正确的），然后释放 x1，于是再过一个周期，指令 jal x1, 12 才被发射出去，这条指令占用寄存器 x1。在 25700ns~26100ns 时 RegWrite_ctrl == 1 时，在其中的时钟下降沿往 x1 写入 jal 指令的结果（仿真图内可见寄存器[1]的值变成了 32’h000000a48，这个结果是正确的）。

（5）WAR 冲突：31300ns~：在开始的两个时钟周期内，依序发射的指令为 32’h022404b3（mul x9, x8, x2）和 32’h00400113（addi x2, x0, 4）。可见，addi 要写回的寄存器恰是 mul 要读取的寄存器 x2。addi 指令在发射后的两个时钟周期后就已经产生了要写回 x2 的结果，但此时，由于 mul 前面有一条指令 div x8, x7, x2，mul 指令在等待 div 将结果写回 x8，这就造成了 mul 和 addi 之间的 WAR 冲突。到 42100ns~42500ns 时 RegWrite_ctrl == 1，div 在其中的时钟下降沿将数据写回 x8，然后在接下去的一个时钟周期内，mul 才读取操作数并执行（从 x8, x2 中读取，故 rs1_addr_ctrl == 8，rs2_addr_ctrl == 2，读取到的数分别为 rs1_data_RO == 32’h01ffe000 和 rs2_data_RO == 32’h00000008，正确），然后再在下一个时钟周期内，RegWrite_ctrl == 1，在其中的时钟下降沿被写回寄存器的正是 addi 指令的结果，写回的寄存器为 x2（仿真图内可见寄存器[2]的值变成了 32’h00000004，这个结果是正确的）。

物理验证已经验收过了，这里不再赘述。

五、 讨论与心得

这次实验让我对带计分牌的多周期流水线 CPU 的运行原理有了更深刻的认识。