浙江大学

本科实验报告

课程名称：	计算机体系结构
姓　　名：	张天逸
学　　院：	竺可桢学院
专　　业：	计算机科学与技术
学　　号：	3220106424
指导教师：	姜晓红

2024 年 12 月 22 日

浙江大学实验报告

课程名称：计算机体系结构　　实验类型：综合

实验项目名称：Dynamically Scheduled Pipelines using Scoreboarding/Tomasulo

学生姓名：张天逸　专业：计算机科学与技术　学号：3220106424

同组学生姓名：朱家骏　　指导老师：姜晓红

一、 实验目的和要求

1. 理解支持多周期操作的流水线原理、设计方法和验证方法
2. 理解带有 Scoreboard/Tomasulo 的动态调度原理
3. 掌握支持多周期指令的流水线的设计方法
4. 掌握带有 Scoreboard/Tomasulo 的动态规划流水线的设计方法
5. 掌握带有 Scoreboard/Tomasulo 的动态规划流水线的验证方法

二、 实验内容和原理

实验内容：

1. 像往常一样新建项目，导入 design source
2. 导入 multiplier 和 divider 的 IP 核
3. 根据代码的提示，完成 CtrlUnit 和 FU_jump 中的代码编写
4. 自行编写 RISCV 代码和 RAM 内存文件完成调试

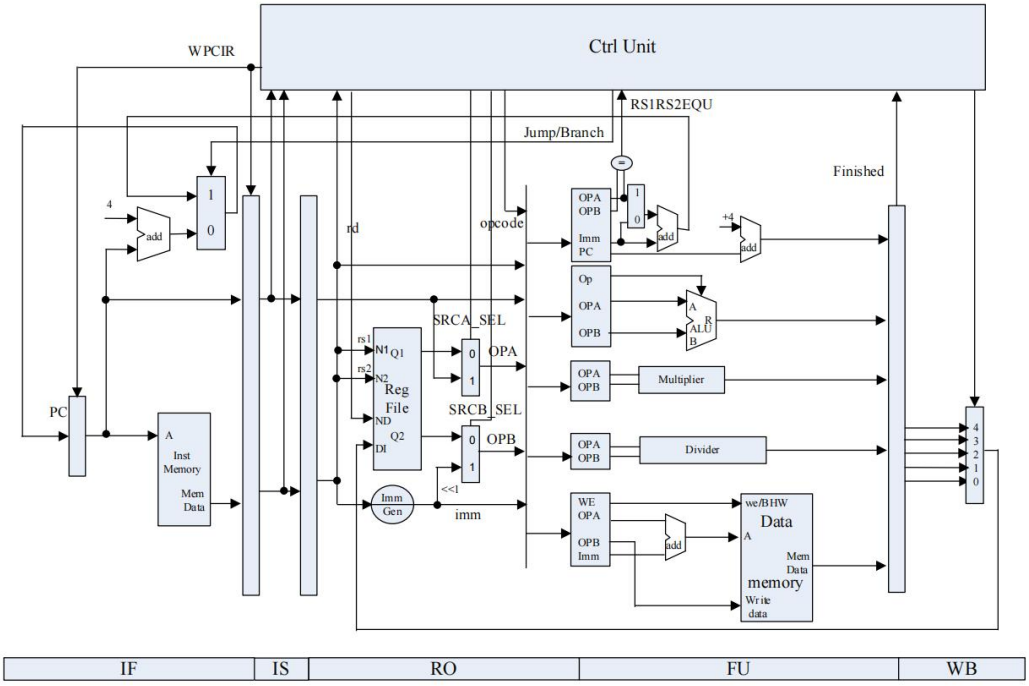
实验原理：

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及结果分析

1. 补充 CtrlUnit.v

(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。

```
164     // normal stall: structural hazard or WAW
165     assign normal_stall = (use_ALU & FUS[`FU_ALU][`BUSY]) |
166                           (use_MEM & FUS[`FU_MEM][`BUSY]) |
167                           (use_MUL & FUS[`FU_MUL][`BUSY]) |
168                           (use_DIV & FUS[`FU_DIV][`BUSY]) |
169                           (use_JUMP& FUS[`FU_JUMP][`BUSY])|
170                           (RRS[dst] != 0);
```

根据设计原理，补充设计计分牌变量的代码。

```
349             // IS
350             if (RO_en) begin
351                 // not busy, no WAW, write info to FUS and RRS
352                 // Issue指令时候的处理逻辑
353                 if (|dst) RRS[dst] <= use_FU;
354                 FUS[use_FU][`BUSY] <= 1'b1;
355                 FUS[use_FU][`SRC1_H:`SRC1_L] <= src1;
356                 FUS[use_FU][`SRC2_H:`SRC2_L] <= src2;
357                 FUS[use_FU][`DST_H:`DST_L] <= dst;
358                 FUS[use_FU][`OP_H:`OP_L] <= op;
359                 FUS[use_FU][`FU1_H:`FU1_L] <= fu1;
360                 FUS[use_FU][`FU2_H:`FU2_L] <= fu2;
361                 FUS[use_FU][`RDY1] <= rdy1;
362                 FUS[use_FU][`RDY2] <= rdy2;
363                 FUS[use_FU][`FU_DONE] <= 1'b0;
364 
365                 IMM[use_FU] <= imm;
366                 PCR[use_FU] <= PC;
367             end
```

(2) RO 阶段

根据设计原理，完善 RO 逻辑代码。

```
369         // RO阶段检测每个FU是否已经完成取操作数，以及对应的scoreboard更新操作
370         if (FUS[`FU_JUMP][`RDY1] & FUS[`FU_JUMP][`RDY2]) begin
371             // JUMP
372             FUS[`FU_JUMP][`RDY1] <= 1'b0;
373             FUS[`FU_JUMP][`RDY2] <= 1'b0;
374         end
375         else if (FUS[`FU_ALU][`RDY1] & FUS[`FU_ALU][`RDY2]) begin
376             // ALU
377             FUS[`FU_ALU][`RDY1] <= 1'b0;
378             FUS[`FU_ALU][`RDY2] <= 1'b0;
379         end
380         else if (FUS[`FU_MEM][`RDY1] & FUS[`FU_MEM][`RDY2]) begin
381             // MEM
382             FUS[`FU_MEM][`RDY1] <= 1'b0;
383             FUS[`FU_MEM][`RDY2] <= 1'b0;
384         end
385         else if (FUS[`FU_MUL][`RDY1] & FUS[`FU_MUL][`RDY2]) begin
386             // MUL
387             FUS[`FU_MUL][`RDY1] <= 1'b0;
388             FUS[`FU_MUL][`RDY2] <= 1'b0;
389         end
390         else if (FUS[`FU_DIV][`RDY1] & FUS[`FU_DIV][`RDY2]) begin
391             // DIV
392             FUS[`FU_DIV][`RDY1] <= 1'b0;
393             FUS[`FU_DIV][`RDY2] <= 1'b0;
394         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。

```
396             // EX
397             FUS[`FU_ALU][`FU_DONE] <= ALU_done ? 1'b1 : FUS[`FU_ALU][`FU_DONE];
398             FUS[`FU_MEM][`FU_DONE] <= MEM_done ? 1'b1 : FUS[`FU_MEM][`FU_DONE];
399             FUS[`FU_MUL][`FU_DONE] <= MUL_done ? 1'b1 : FUS[`FU_MUL][`FU_DONE];
400             FUS[`FU_DIV][`FU_DONE] <= DIV_done ? 1'b1 : FUS[`FU_DIV][`FU_DONE];
401             FUS[`FU_JUMP][`FU_DONE]<= JUMP_done ? 1'b1 : FUS[`FU_JUMP][`FU_DONE];
```

(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。

```
278     wire ALU_WAR = (
279         (FUS[`FU_MEM][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_MEM]
            [`RDY1])  &
280         (FUS[`FU_MEM][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_MEM]
            [`RDY2])  &
281         (FUS[`FU_MUL][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_MUL]
            [`RDY1])  &
282         (FUS[`FU_MUL][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_MUL]
            [`RDY2])  &
283         (FUS[`FU_DIV][`SRC1_H:`SRC1_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_DIV]
            [`RDY1])  &
284         (FUS[`FU_DIV][`SRC2_H:`SRC2_L]  != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_DIV]
            [`RDY2])  &
285         (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_JUMP]
            [`RDY1])  &
286         (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_ALU][`DST_H:`DST_L] | !FUS[`FU_JUMP]
            [`RDY2])
287     );
```

```
289     wire MEM_WAR = (
290         (FUS[`FU_ALU][`SRC1_H:`SRC1_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY1]) &
291         (FUS[`FU_ALU][`SRC2_H:`SRC2_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY2]) &
292         (FUS[`FU_MUL][`SRC1_H:`SRC1_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY1]) &
293         (FUS[`FU_MUL][`SRC2_H:`SRC2_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY2]) &
294         (FUS[`FU_DIV][`SRC1_H:`SRC1_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY1]) &
295         (FUS[`FU_DIV][`SRC2_H:`SRC2_L]  != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY2]) &
296         (FUS[`FU_JUMP][`SRC1_H:`SRC1_L] != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY1]) &
297         (FUS[`FU_JUMP][`SRC2_H:`SRC2_L] != FUS[`FU_MEM][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY2])
298     );
```

```
300     wire MUL_WAR = (
301         (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY1]) &
302         (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY2]) &
303         (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY1]) &
304         (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY2]) &
305         (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY1]) &
306         (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY2]) &
307         (FUS[`FU_JUMP][`SRC1_H:`SRC1_L]!= FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY1]) &
308         (FUS[`FU_JUMP][`SRC2_H:`SRC2_L]!= FUS[`FU_MUL][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY2])
309     );
```

```
311     wire DIV_WAR = (
312         (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY1]) &
313         (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY2]) &
314         (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY1]) &
315         (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY2]) &
316         (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY1]) &
317         (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY2]) &
318         (FUS[`FU_JUMP][`SRC1_H:`SRC1_L]!= FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY1]) &
319         (FUS[`FU_JUMP][`SRC2_H:`SRC2_L]!= FUS[`FU_DIV][`DST_H:`DST_L] | !FUS[`FU_JUMP]
        [`RDY2])
320     );
```

```
322     wire JUMP_WAR = (
323         (FUS[`FU_ALU][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY1]) &
324         (FUS[`FU_ALU][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_ALU]
        [`RDY2]) &
325         (FUS[`FU_MEM][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY1]) &
326         (FUS[`FU_MEM][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MEM]
        [`RDY2]) &
327         (FUS[`FU_MUL][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY1]) &
328         (FUS[`FU_MUL][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_MUL]
        [`RDY2]) &
329         (FUS[`FU_DIV][`SRC1_H:`SRC1_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY1]) &
330         (FUS[`FU_DIV][`SRC2_H:`SRC2_L] != FUS[`FU_JUMP][`DST_H:`DST_L] | !FUS[`FU_DIV]
        [`RDY2])
331     );
```

设置计分牌变量代码逻辑，并在其中处理 RAW 冲突。

```
404             // WB
405             if (FUS[`FU_JUMP][`FU_DONE] & JUMP_WAR) begin
406                 FUS[`FU_JUMP] <= 0;
407                 RRS[FUS[`FU_JUMP][`DST_H:`DST_L]] <= 3'b0;
408 
409                 // ensure RAW
410                 if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_ALU][`RDY1] <= 1'b1;
411                 if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_MEM][`RDY1] <= 1'b1;
412                 if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_MUL][`RDY1] <= 1'b1;
413                 if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_JUMP) FUS[`FU_DIV][`RDY1] <= 1'b1;
414 
415                 if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_ALU][`RDY2] <= 1'b1;
416                 if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MEM][`RDY2] <= 1'b1;
417                 if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_MUL][`RDY2] <= 1'b1;
418                 if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_JUMP) FUS[`FU_DIV][`RDY2] <= 1'b1;
419             end
```

```
420             // ALU
421             else if (FUS[`FU_ALU][`FU_DONE] & ALU_WAR) begin
422                 FUS[`FU_ALU] <= 32'b0;
423                 RRS[FUS[`FU_ALU][`DST_H:`DST_L]] <= 3'b0;
424 
425                 // ensure RAW
426                 if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_JUMP][`RDY1] <= 1'b1;
427                 if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_MEM][`RDY1] <= 1'b1;
428                 if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_MUL][`RDY1] <= 1'b1;
429                 if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_ALU) FUS[`FU_DIV][`RDY1] <= 1'b1;
430 
431                 if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_JUMP][`RDY2] <= 1'b1;
432                 if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_MEM][`RDY2] <= 1'b1;
433                 if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_MUL][`RDY2] <= 1'b1;
434                 if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_ALU) FUS[`FU_DIV][`RDY2] <= 1'b1;
435             end
```

```
436             // MEM
437             else if (FUS[`FU_MEM][`FU_DONE] & MEM_WAR) begin
438                 FUS[`FU_MEM] <= 32'b0;
439                 RRS[FUS[`FU_MEM][`DST_H:`DST_L]] <= 3'b0;
440 
441                 // ensure RAW
442                 if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_ALU][`RDY1] <= 1'b1;
443                 if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_JUMP][`RDY1] <= 1'b1;
444                 if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_MUL][`RDY1] <= 1'b1;
445                 if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_MEM) FUS[`FU_DIV][`RDY1] <= 1'b1;
446 
447                 if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_MEM) FUS[`FU_ALU][`RDY2] <= 1'b1;
448                 if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_MEM) FUS[`FU_JUMP][`RDY2] <= 1'b1;
449                 if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_MEM) FUS[`FU_MUL][`RDY2] <= 1'b1;
450                 if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_MEM) FUS[`FU_DIV][`RDY2] <= 1'b1;
451             end
```

```
452             // MUL
453             else if (FUS[`FU_MUL][`FU_DONE] & MUL_WAR) begin
454                 FUS[`FU_MUL] <= 32'b0;
455                 RRS[FUS[`FU_MUL][`DST_H:`DST_L]] <= 3'b0;
456 
457                 // ensure RAW
458                 if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_ALU][`RDY1] <= 1'b1;
459                 if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_MEM][`RDY1] <= 1'b1;
460                 if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_JUMP][`RDY1] <= 1'b1;
461                 if (FUS[`FU_DIV][`FU1_H:`FU1_L] == `FU_MUL) FUS[`FU_DIV][`RDY1] <= 1'b1;
462 
463                 if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_MUL) FUS[`FU_ALU][`RDY2] <= 1'b1;
464                 if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_MUL) FUS[`FU_MEM][`RDY2] <= 1'b1;
465                 if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_MUL) FUS[`FU_JUMP][`RDY2] <= 1'b1;
466                 if (FUS[`FU_DIV][`FU2_H:`FU2_L] == `FU_MUL) FUS[`FU_DIV][`RDY2] <= 1'b1;
467             end
```

```
468             // DIV
469             else if (FUS[`FU_DIV][`FU_DONE] & DIV_WAR) begin
470                 FUS[`FU_DIV] <= 32'b0;
471                 RRS[FUS[`FU_DIV][`DST_H:`DST_L]] <= 3'b0;
472 
473                 // ensure RAW
474                 if (FUS[`FU_ALU][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_ALU][`RDY1] <= 1'b1;
475                 if (FUS[`FU_MEM][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_MEM][`RDY1] <= 1'b1;
476                 if (FUS[`FU_MUL][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_MUL][`RDY1] <= 1'b1;
477                 if (FUS[`FU_JUMP][`FU1_H:`FU1_L] == `FU_DIV) FUS[`FU_JUMP][`RDY1] <= 1'b1;
478 
479                 if (FUS[`FU_ALU][`FU2_H:`FU2_L] == `FU_DIV) FUS[`FU_ALU][`RDY2] <= 1'b1;
480                 if (FUS[`FU_MEM][`FU2_H:`FU2_L] == `FU_DIV) FUS[`FU_MEM][`RDY2] <= 1'b1;
481                 if (FUS[`FU_MUL][`FU2_H:`FU2_L] == `FU_DIV) FUS[`FU_MUL][`RDY2] <= 1'b1;
482                 if (FUS[`FU_JUMP][`FU2_H:`FU2_L] == `FU_DIV) FUS[`FU_JUMP][`RDY2] <= 1'b1;
483             end
```

补充 WB 的处理逻辑，检测何时写入，以及写入的位置。在实验结果中，我们的仿真结果再每次写入后都会比标准结果延迟一个周期，分析原因为 WB 写入时与之后的代码开始执行的操作并不同时开始，因此会停顿一个周期，但不影响实验结果。

```
579     // WB
580     // WB的处理逻辑，检测何时写入，以及写入的位置
581     always @ (*) begin
582         write_sel = 0;
583         reg_write = 0;
584         rd_ctrl = 0;
585 
586         if (JUMP_WAR & FUS[`FU_JUMP][`FU_DONE]) begin
587             write_sel = 3'd4;
588             reg_write = 1'b1;
589             rd_ctrl = FUS[`FU_JUMP][`DST_H:`DST_L];
590         end
591         else if (ALU_WAR & FUS[`FU_ALU][`FU_DONE]) begin
592             write_sel = 3'd0;
593             reg_write = 1'b1;
594             rd_ctrl = FUS[`FU_ALU][`DST_H:`DST_L];
595         end
596         else if (MEM_WAR & FUS[`FU_MEM][`FU_DONE]) begin
597             write_sel = 3'd1;
598             reg_write = 1'b1;
599             rd_ctrl = FUS[`FU_MEM][`DST_H:`DST_L];
600         end
601         else if (MUL_WAR & FUS[`FU_MUL][`FU_DONE]) begin
602             write_sel = 3'd2;
603             reg_write = 1'b1;
604             rd_ctrl = FUS[`FU_MUL][`DST_H:`DST_L];
605         end
606         else if (DIV_WAR & FUS[`FU_DIV][`FU_DONE]) begin
607             write_sel = 3'd3;
608             reg_write = 1'b1;
609             rd_ctrl = FUS[`FU_DIV][`DST_H:`DST_L];
610         end
611     end
```

2. 补充 FU_jump.v

将 finish 信号初始置为 1。

```
13      assign finish = state == 1'b1;
```

在模块触发（EN 为 1 且 state 为 0）时，进行寄存器赋值并将 state 切换为 1，否则均将 state 置为 0。

```
22      always@(posedge clk) begin
23          if(EN & ~state) begin // state == 0
24              JALR_reg <= JALR;
25              cmp_ctrl_reg <= cmp_ctrl;
26              rs1_data_reg <= rs1_data;
27              rs2_data_reg <= rs2_data;
28              imm_reg <= imm;
29              PC_reg <= PC;
30              state <= 1;
31          end
32          else state <= 0;
33      end
```

可以看到，jump 的判断逻辑中有三个触发条件，当满足任意一个时即触发该信号。

```
41      assign is_jump = cmp_ctrl_reg[0] | cmp_res | JALR;
```

3. 实验结果

本次实验运行的代码与使用的数据为框架中自带文件，在仿真网站上运行仿真，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：流水线依序发射指令 32’h00190093、32’h00402103，分别占用了 ALU 和 MEM 功能单元。而对于下一条指令 32’h00802203，由于 MEM 功能单元已被占用，所以只能停机等待。RegWrite_ctrl 变为 1，在其中的时钟下降沿写回的数据是 ALU 功能单元的结果，之后释放 ALU 功能单元；RegWrite_ctrl 变为 1，在其中的时钟下降沿写回的数据是 MEM 功能单元的结果，之后释放 ALU 功能单元，再过一个时钟周期，发射指令进入并占用 MEM

单元。

2. RAW 冲突：先发射指令 32’h004100b3 进入并占用 ALU 功能单元，故下一条指令 32’hfff08193 不可发射。32’h004100b3 与上一条指令 32’h004100b3 存在 RAW 冲突。故须等待上一条指令写入寄存器后，下一条指令才能读取操作数并执行。写入发生在 RegWrite_ctrl = 1 期间的时钟下降沿，之后下一条指令才读取操作数并执行。之后 RegWrite_ctrl 变为 1，在其中的时钟下降沿写回的数据是 ALU 功能单元的结果。

3. WAW 冲突：在第一个时钟周期内发射了指令 32’h00004537，接下去的指令 32’h014005ef 因为与上一条指令要写回的寄存器相同，所以在上一条指令写回之前，它们存在 WAW 冲突。RegWrite_ctrl == 1 时，在其中的时钟下降沿往寄存器写入 lui 指令的结果然后释放寄存器，再过一个周期后，指令 32’h014005ef 才被发射出去，这条指令占用相同的寄存器。之后 RegWrite_ctrl == 1 时，在其中的时钟下降沿往 x1 写入 jal 指令的结果。

4. WAR 冲突：在开始的两个时钟周期内，依序发射的指令为 32’h022687b3 和 32’h00400113。可见，addi 要写回的寄存器恰是 mul 要读取的寄存器。addi 指令在发射后的两个时钟周期后就已经产生了要写回的结果，但此时由于 mul 前面有一条 div 指令，mul 指令在等待 div 将结果写回寄存器，这就造成了 mul 和 addi 之间的 WAR 冲突。div 在其中的时钟下降沿将数据写回，然后在接下去的一个时钟周期内，mul 才读取操作数并执行。然后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，存入的寄存器也正确。

5. 同时写入：在开始的两个时钟周期内，依序发射指令 32’h01402383 和 32’h40220433 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 RegWrite == 1 的情况持续了两个时钟周期，在其中的第一个时钟下降沿写回的数据是 ALU 功能单元的结果；在其中的第二个时钟下降沿写回的数据是 MEM 功能单元的结果。

四、 讨论与心得

在此次计算机体系结构的实验中，我们深入探究了动态调度流水线的设计，实现了 Scoreboard 算法的应用，完成了同时处理结构冲突和多种数据依赖（如 RAW、WAR、WAW）的系统功能。

实验过程中，主要工作为编写 CtrlUnit 和 FU_jump 模块的代码。而实现 RO 阶段逻辑时，我们必须精确控制各个功能单元的状态转换，确保它们在适当的时间点进行计算结果的写回。然而，由于对某些寄存器行为的理解不足，我们在早期版本的代码中出现了多次错误，导致仿真结果不符合预期。为了攻克这一难题，我们花费大量时间调试代码，并对原框架进行了适当调整。经过不懈努力，我们成功实现了预期的功能。这段经历让我认识到，细致入微的工作态度和耐心是解决问题的关键。

在最后的结果中，可以看到每次写回后流水线的时序逻辑都会比标准结果慢一个周期。经过分析后，我们认为是 WB 的写入逻辑有差别导致写回的结尾与下一条的进入并没有同时进行，但逻辑依然正确。总的来说，我们成功完成了此次实验的要求，有较大收获。