哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

课程报告

开课学期：2022 夏季

课程名称：计算机设计与实践

项目名称：基于 miniRV-1 的 SoC 设计

项目类型：综合设计型

课程学时：56 地点：T2210

学生班级：4 班

学生学号：200110420

学生姓名：文欣婉

评阅教师：

报告成绩：

实验与创新实践教育中心制

2022 年 7 月

注：本设计报告中各个部分如果页数不够，请同学们自行扩页。原则上一定要把报告写详细，能说明设计的成果、特色和过程。报告应该详细叙述整体设计，以及设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计的功能描述（含所有实现的指令描述，以及单周期/流水线 CPU 频率）
实现的 18 条指令如下： 【注释：(r)表示寄存器 r 的值，Mem[addr]表示地址为 addr 的存储单元的值，sext(a)表示 a 的符号扩展】 1. add rd, rs1, rs2： 加(rd)←(rs1)+(rs2) 2. sub rd, rs1, rs2: 减(rd)←(rs1)-(rs2) 3. and rd, rd, rs1,rs2: 与(rd)←(rs1)&(rs2) 4. or rd, rs1, rs2: 或(rd)←(rs1)|(rs2) 5. xor rd, rs1, rs2: 异或(rd)←(rs1)^(rs2) 6. sll rd, rs1, rs2: 左移(rd)←(rs1)<<(rs2) 7. srl rd, rs1, rs2: 逻辑右移(rd)←(rs1)>>(rs2) 8. sra rd, rs1, rs2: 算术右移(rd)←($signed(rs1))>>(rs2) 9. addi rd, rs1, imm: 加立即数(rd)←(rs1)+sext(imm) 10. andi rd, rs1, imm: 与立即数(rd)←(rs1)&sext(imm) 11. ori rd, rs1, imm: 或立即数(rd)←(rs1)\|sext(imm) 12. xori rd, rs1, imm: 异或立即数(rd)←(rs1)^sext(imm) 13. slli rd, rs1, shamt: 左移立即数位(rd)←(rs1)<<shamt 14. srli rd, rs1, shamt: 逻辑右移立即数位(rd)←(rs1)>>shamt 15. srai rd, rs1, shamt: 算数右移立即数(rd)← ($signed(rs1))>>shamt 16. lw rd, offset(rs1): 取数(rd)←sext(Mem[(rs1)+sext(offset)][31:0] 17. jalr rd, offset(rs1): 跳转 t←(pc)+4;(pc)←((rs1)+sext(offset))&~1;(rd)←t（将原来的 pc+4 的值写入寄存器 rd，将 pc 设置为(rs1)+sext(offset)，把计算出的地址的最低位设为 0）

18. sw rs2, offset(rs1):
    存数 Mem[(rs1)+sext(offset)]←(rs2)[31:0]
19. beq rs1, rs2, offset:
    相等则跳转 if((rs1)==(rs2))(pc)←(pc)+sext(offset)
20. bne rs1, rs2, offset:
    不等则跳转 if((rs1)≠(rs2))(pc)←(pc)+sext(offset)
21. blt rs1, rs2, offset:
    小于则跳转 if((rs1)<(rs2))(pc)←(pc)+sext(offset)（有符号数）
22. bge rs1, rs2, offset:
    大于等于则跳转 if((rs1)≥(rs2))(pc)←(pc)+sext(offset)（有符号数）
23. lui rd, imm:
    取长立即数(rd)←sext(imm[31:12]<<12)
24. jal rd, offset:
    跳转(rd)←(pc)+4; (pc)←(pc)+sext(offset)

按照指导书,单周期时钟频率使用 25mhz，流水线频率使用 50mhz 皆成功。单周期未做更高频率的尝试，流水线 cpu 频率成功上板最高为 180mhz。

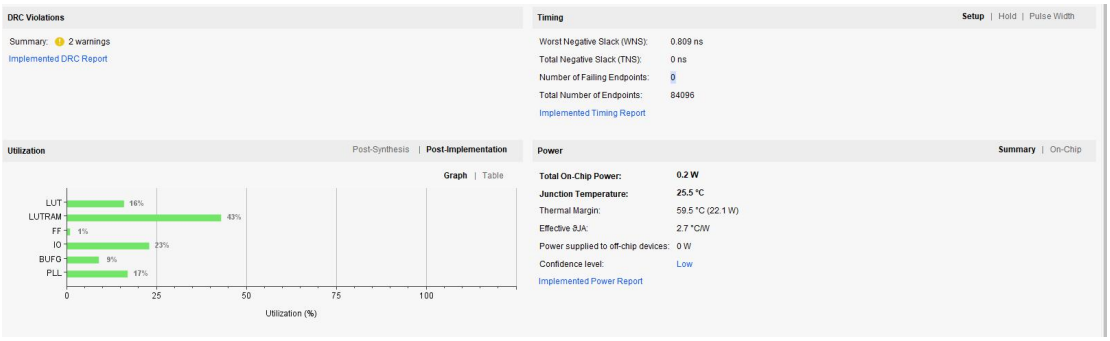
设计的主要特色（除基本要求以外的设计）

流水线 CPU 设计中，对于不涉及访存的数据冒险，RAW 冒险的三种情形均采用前递的方式解决，提高执行效率；对于涉及访存的数据冒险，采用先停顿后续指令一个时钟周期，再前递的方式解决；对于控制冒险，采用静态预测的方式解决，预测分支指令总是不跳转，若在分支指令到达执行阶段时发现预测错误，则将后续已传入错误指令的流水级清空，以在下一个时钟周期传入正确的指令。

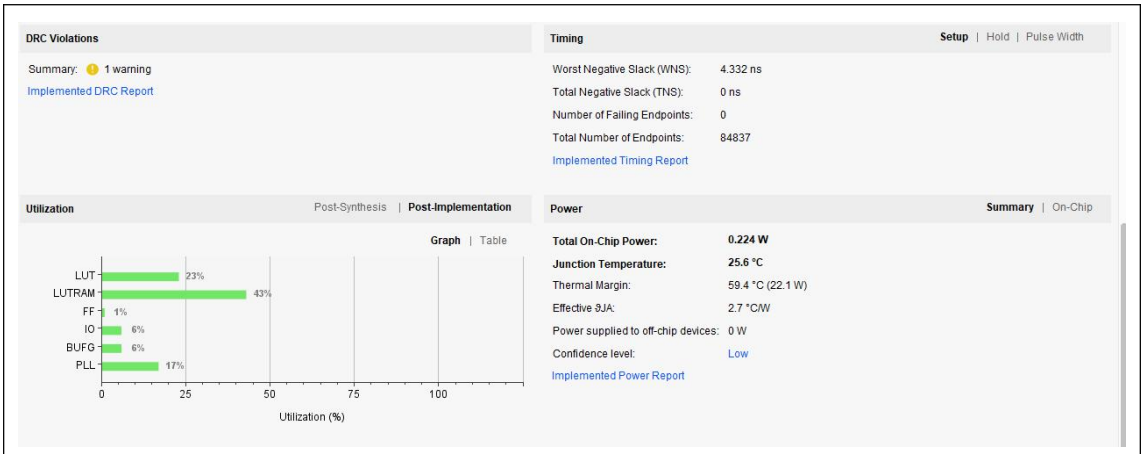
资源使用、功耗数据截图（Post Implementation；含单周期、流水线 2 个截图）

以下是示例，请贴自己的图。

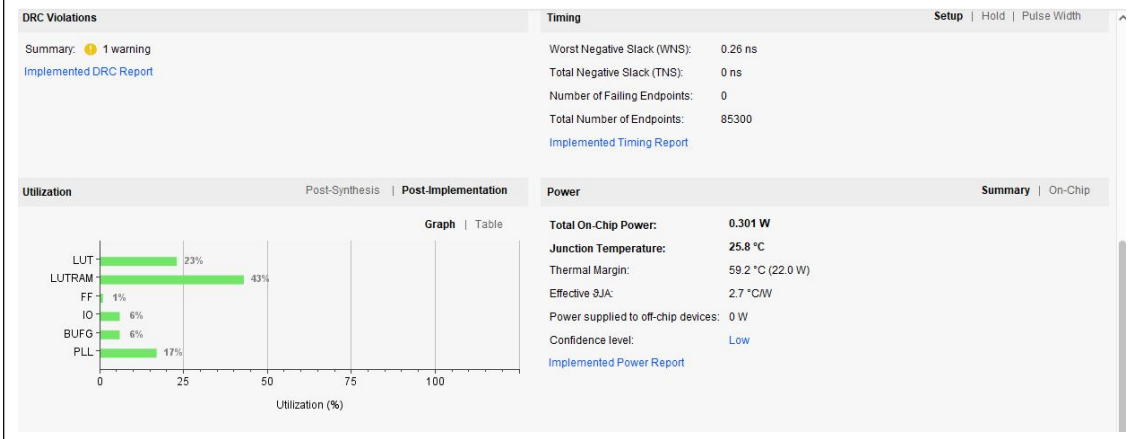
单周期 25mhz



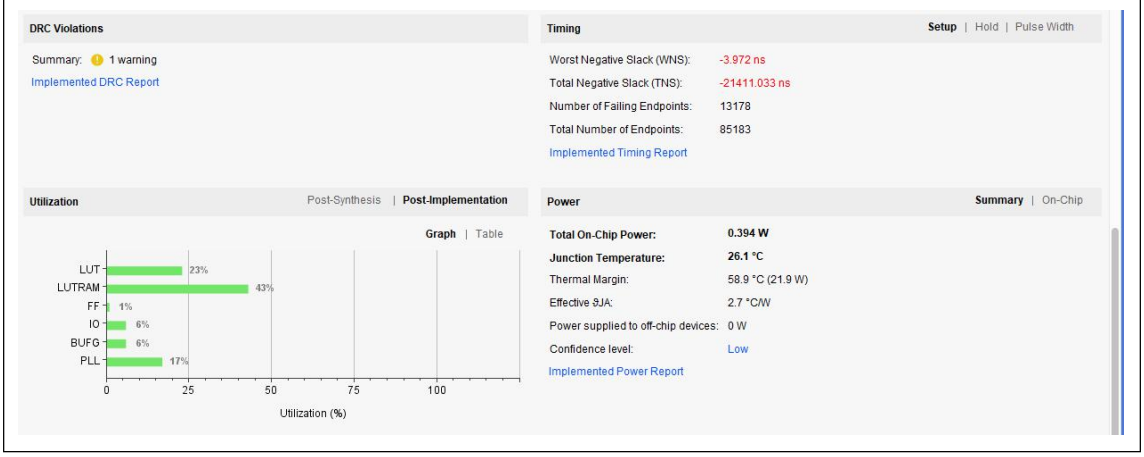
流水线 50mhz



流水线 100mhz



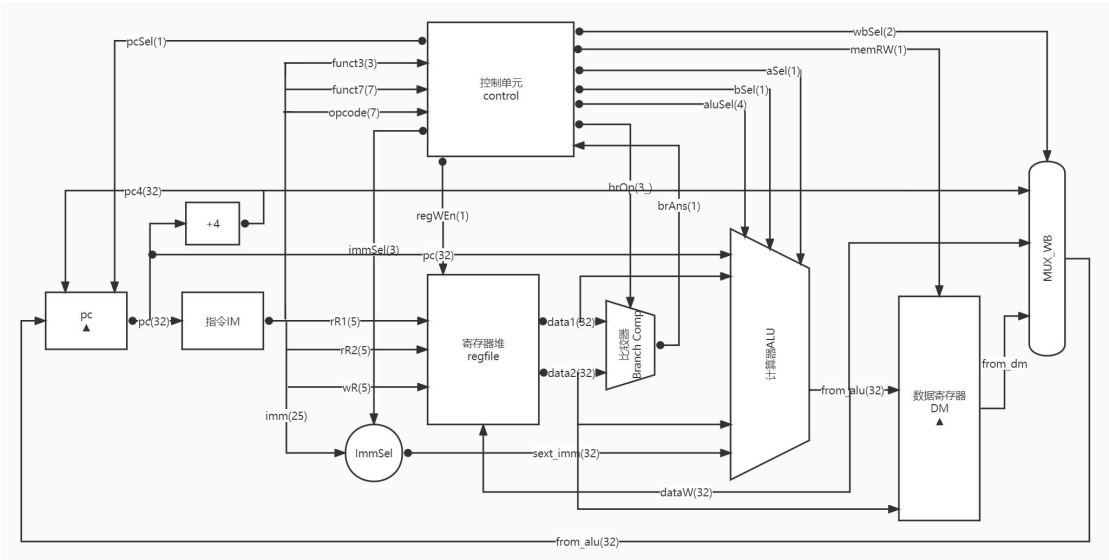
流水线 180mhz



1 单周期 CPU 设计与实现

1.1 单周期 CPU 整体框图

要求：无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，以及说明每个模块的功能含义。



各个模块功能：

pc：更新 pc

指令 IM：指令寄存器

寄存器堆 regfile：从此获取 32 个寄存器存储内容

控制单元 control：给出控制信号

比较器 BranchComp：获取相应比较结果

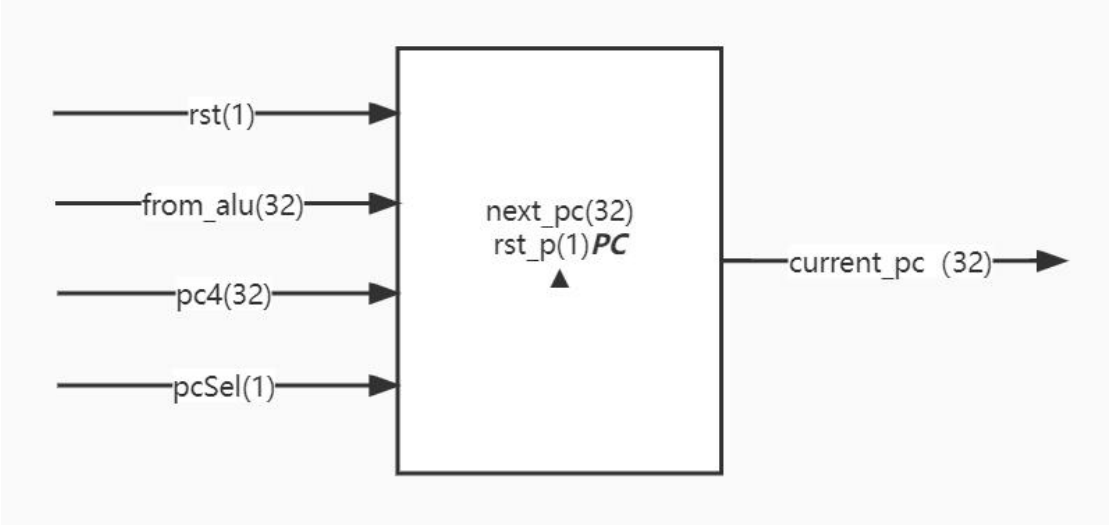
计算器 ALU：获取相应计算结果

数据寄存器 DM：根据所给地址获得内存内容

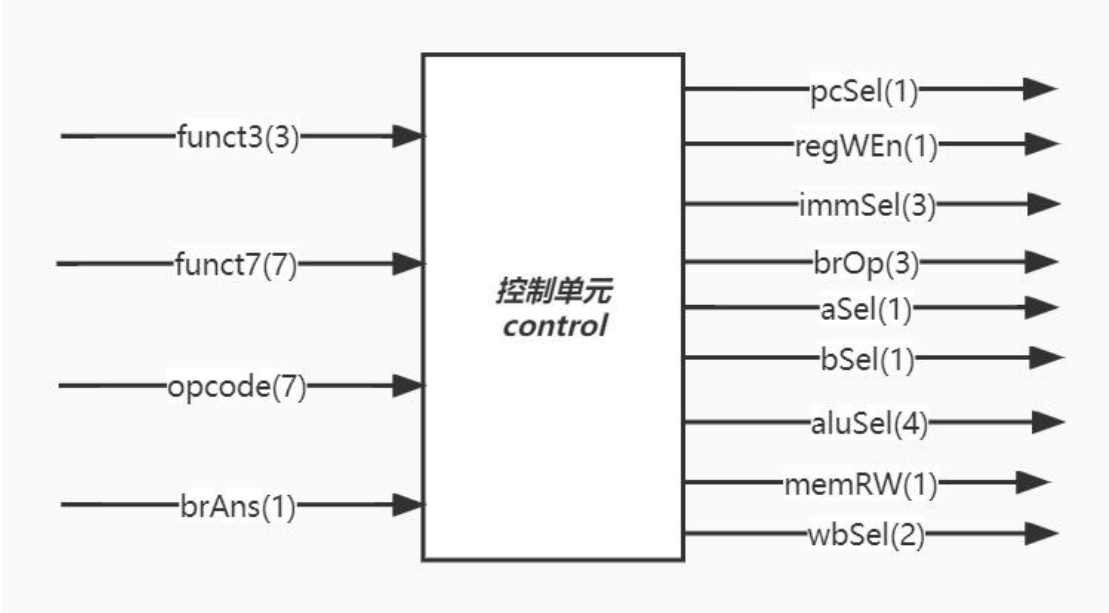
MUX_WB：选择写回数据寄存器的内容

1.2 单周期 CPU 模块详细设计

要求：画出各个模块的详细设计图，包含内部的子模块，以及关键性逻辑；标出子模块接口信号名、各信号线的信号名和位宽，并有详细的解释说明。



next_pc 的赋值采用组合逻辑，当 pcSel 为 0 时表示跳转，被赋为计算出来的地址 from_alu；当 pcSel 为 1 时表示不跳转，被赋为 pc4（恒为 pc+4）。复位信号 rst 无效一个时钟周期后的时钟上升沿信号 rst_p 无效。current_pc 的赋值采用时序逻辑，在每个时钟上升沿，当 rst_p 有效时，被赋值为 0；无效时，被赋值为 next_pc。以实现 pc 的更新。

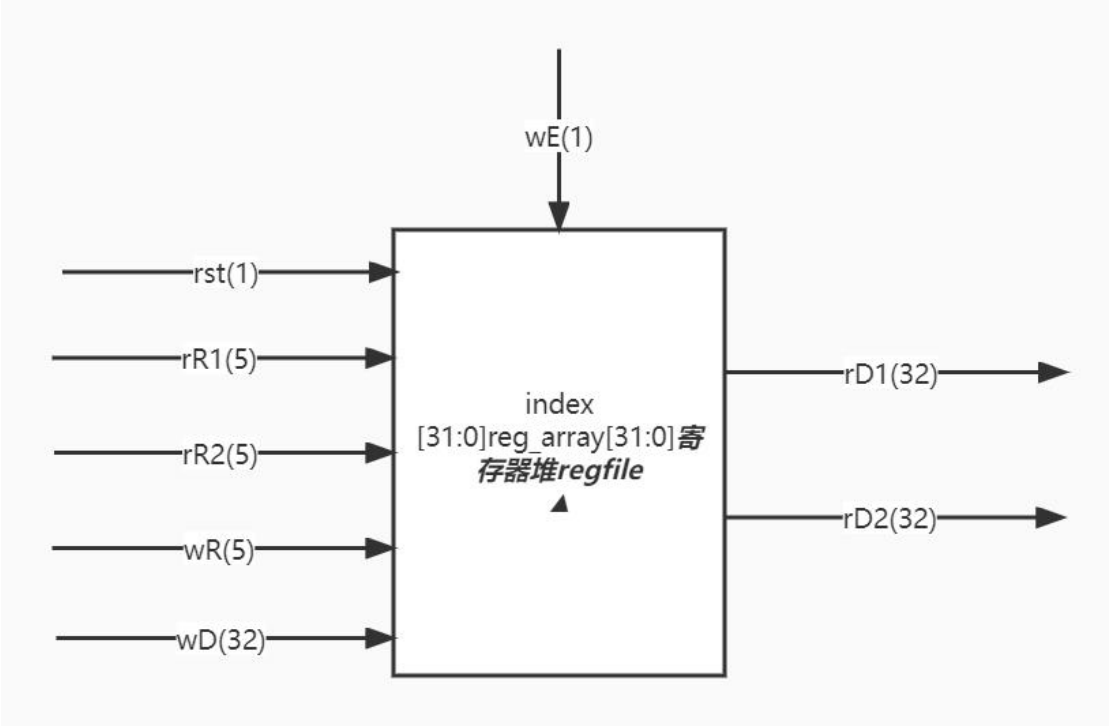


控制信号的赋值均采用组合逻辑。

pcSel：根据 opcode 判断指令类型，当指令为 R 型指令、I 型指令、LW 指令、S 型指令、U 型指令时，指令不跳转，pcSel 设为 1；当指令为 B 型指令时，由比较器结果判断指令跳转与否，pcSel 设为 brAns；当指令为 J 型指令或 jalr 指令，指令跳转，pcSel 设为 0；其余情况默认指令不跳转，pcSel 设为 1。

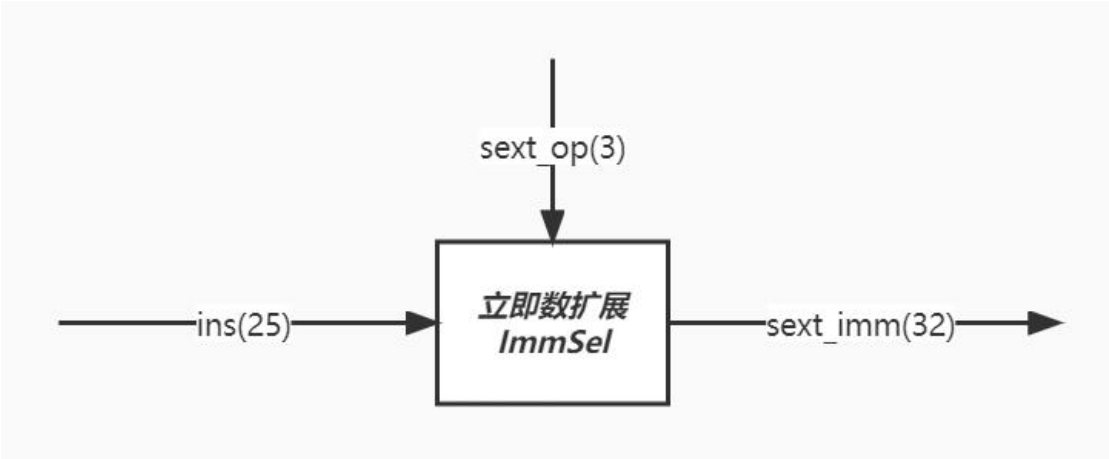
regWEn：当指令为 S 型指令或 B 型指令时，不写寄存器，regWEn 设为 0，否则设为 1。

immSel：根据 opcode 判断指令类型选择如何对立即数进行扩展。当指令为 I 型指令、LW 指令、jalr 指令时，immSel 设为`I_SEXT；当指令为 S 型指令时，immSel 设为`S_SEXT ；当指令为 B 型指令时，immSel 设为`B_SEXT；当指令为 U 型指令时，immSel 设为`U_SEXT；当指令为 J 型指令时，immSel 设为`J_SEXT；其余情况默认 immSel 设为 3'b111。
brOp：当指令类型为 B 型指令时，比较器做何比较由 funct3 决定，直接将 brOp 设为 funct3；其余情况默认 brOp 设为 3'b111。
aSel：根据指令类型判断 ALU 第一个操作数。当指令类型为 B 型指令或 J 型指令时， aSel 设为 0；否则 aSel 设为 1。
bSel：根据指令类型判断 ALU 第二个操作数。当指令类型为 R 型指令时， bSel 设为 0；否则 bSel 设为 1。
aluSel：当指令类型为 R 型指令时，根据 funct3 和 funct7 判断计算器进行的操作类型，分别将 aluSel 设为`SUB、`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令类型为 I 型指令时，根据 funct3 和 funct7 判断计算器进行的操作类型，分别将 aluSel 设为`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令为 LW 指令、jalr 指令、S 型指令、B 型指令、J 型指令时，aluSel 设为`ADD；当指令为 U 型指令时，aluSel 设为`LUI；其余情况默认 aluSel 设为`NOOP。
memRW：根据指令类型判断读写内存。当指令类型为 S 型指令时需要写内存，memRW 设为 1；其余情况默认不写内存，设为 0。
wbSel：根据指令类型选择写回寄存器的内容。当指令类型为 J 型指令、jalr 指令时， wbSel 设为`PC4；当指令为 LW 指令时， wbSel 设为`FROM_DM；其余情况默认 wbSel 设为`FROM_ALU。

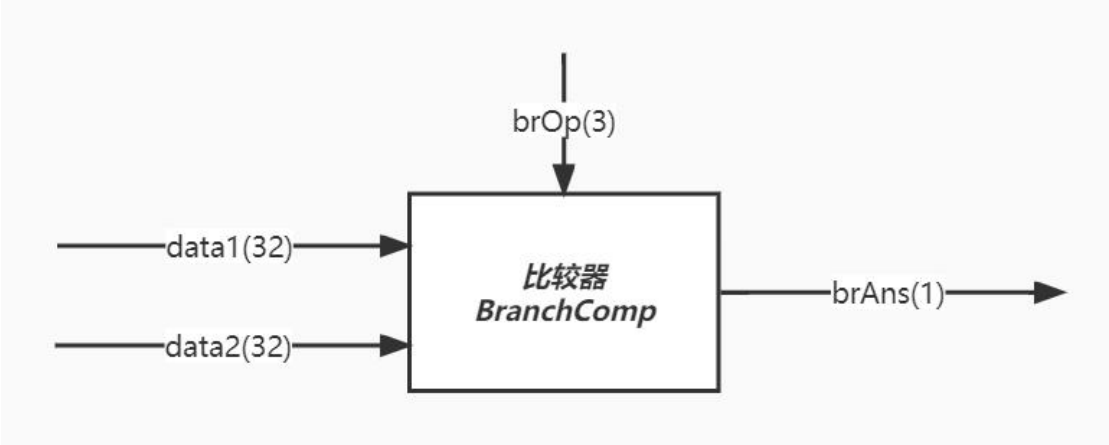


读寄存器采用时序逻辑，每个时钟下降沿来临时，若 rR1 不是 0 号寄存器，将其中存储的内容赋给 rD1，否则将 rD1 设为 0；若 rR2 不是 0 号寄存器，将其中存储的内容赋给 rD2，否则将 rD2 设为 0。

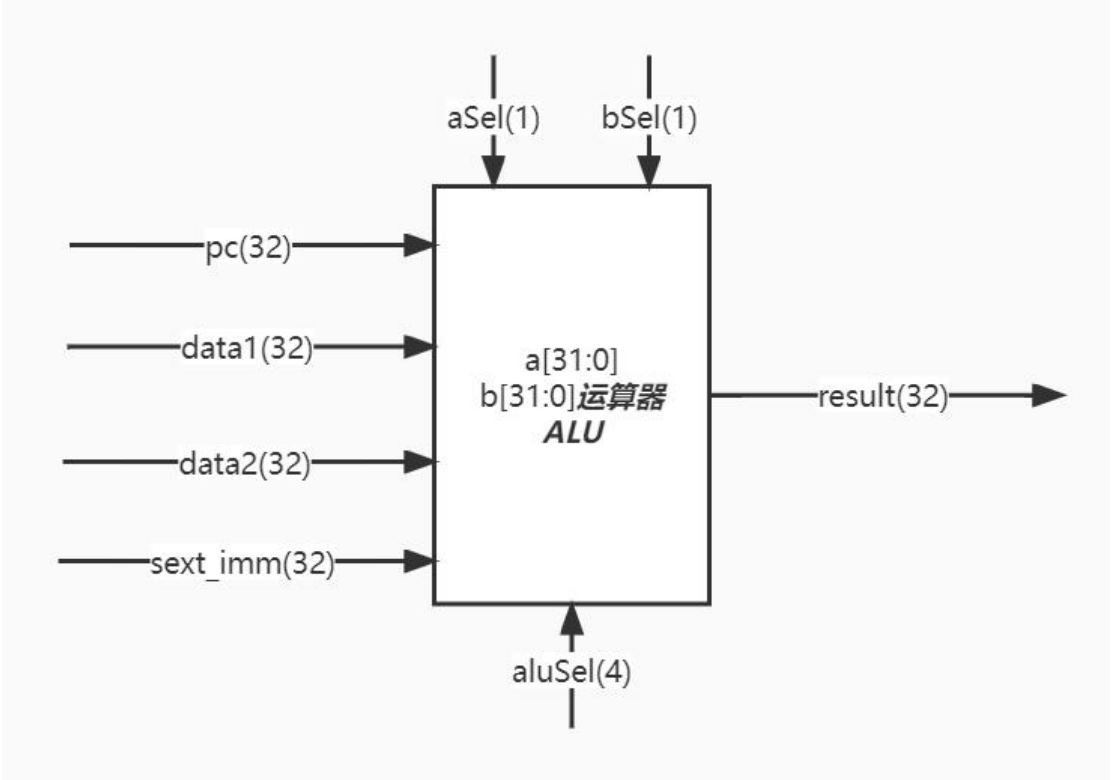
写寄存器采用时序逻辑，每个时钟上升沿来临时，0 号寄存器 reg_array[0]恒设为 0，复位信号有效时，所有寄存器设为 0。不复位时，若写寄存器堆信号有效，即 wE 为 1 时，且目标寄存器不是 0 号寄存器，则将 reg_array[wR]设为 wD。



根据控制信号对 25 位的数 ins 进行相应立即数扩展为 32 位的 sext_imm。当 sext_op 为`I_SEXT 时，进行 I 型立即数扩展；当 sext_op 为`S_SEXT 时，进行 S 型立即数扩展;当 sext_op 为`B_SEXT,进行 B 型立即数扩展;当 sext_op 为`U_SEXT 时，进行 U 型立即数扩展；当 sext_op 为`J_SEXT 时，进行 J 型立即数扩展；其余情况默认设置 sext_imm 为 0。

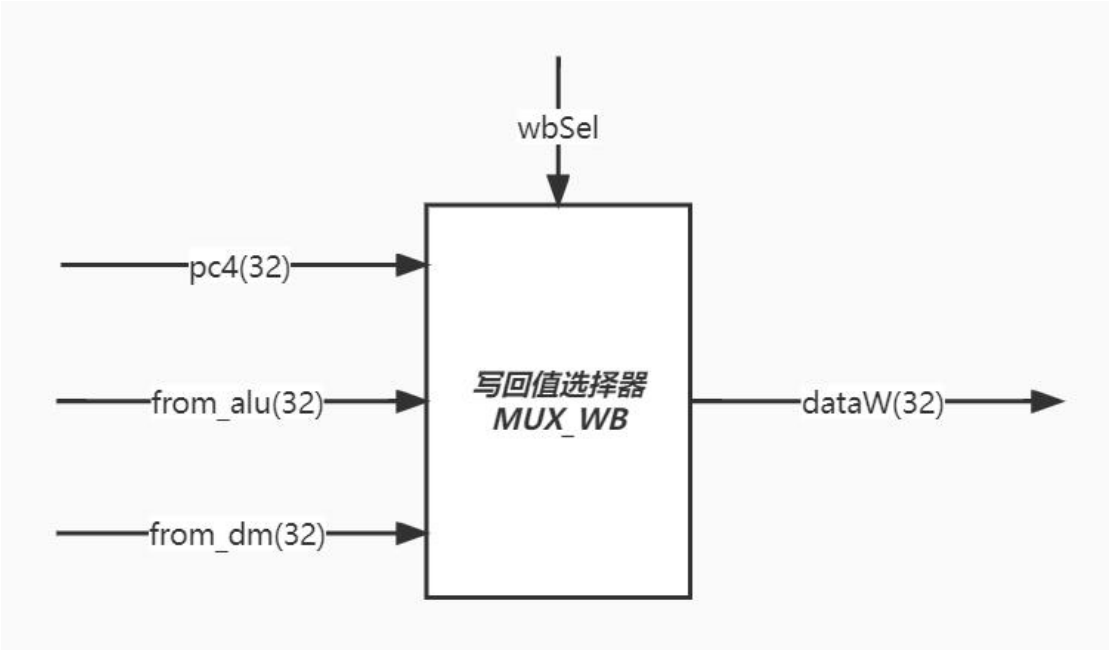


brOp 为`BEQ 时，data1==data2 则将 brAns 设为 0，否则为 1；为`BNE 时，与`BEQ 相反；为`BLT 时，data1<data2 则将 brAns 设为 0，否则为 1；为`BGE 时，与`BLT 相反。

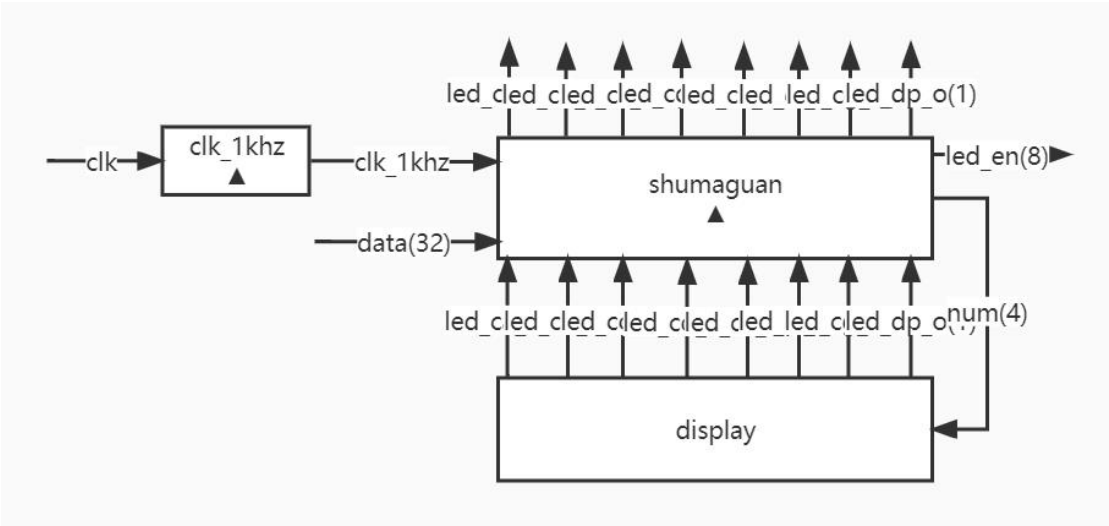


运算器采用组合逻辑。

aSel 为 0 时，ALU 第一个操作数 a 选择 pc，为 1 时选择 reg1 存储的数。bSel 为 0 时，ALU 第二个操作数 b 选择 reg2 存储的数，为 1 时选择扩展后的立即数。当 aluSel 为`ADD 时，ALU 对两操作数进行加法操作（result=a+b）；当为`SUB 时，做减法操作（result=a-b）；当为`AND 时，做与操作（result=a&b）；当为`OR 时，做或操作（result=a|b）；当为`XOR 时，做异或操作（result=a^b）；当为`SLL 时，做左移操作（result=a<<b[4:0]）；当为`SRL 时，做逻辑右移操作（result=a>>b[4:0]）；当为`SRA 时，做算数右移操作（result=($signed(a))>>>b[4:0]）；当为`LUI 时，直接将 b 赋值给 result（result=b）；其他情况默认将 result 直接赋值为 0。



寄存器堆写回值 dataW 的赋值采用组合逻辑。当 wbSel=`PC4 时，选择 pc+4；当 wbSel=`FROM_ALU 时，选择 ALU 的计算结果；当 wbSel=`FROM_DM 时，选择从 DM 中取出的结果。

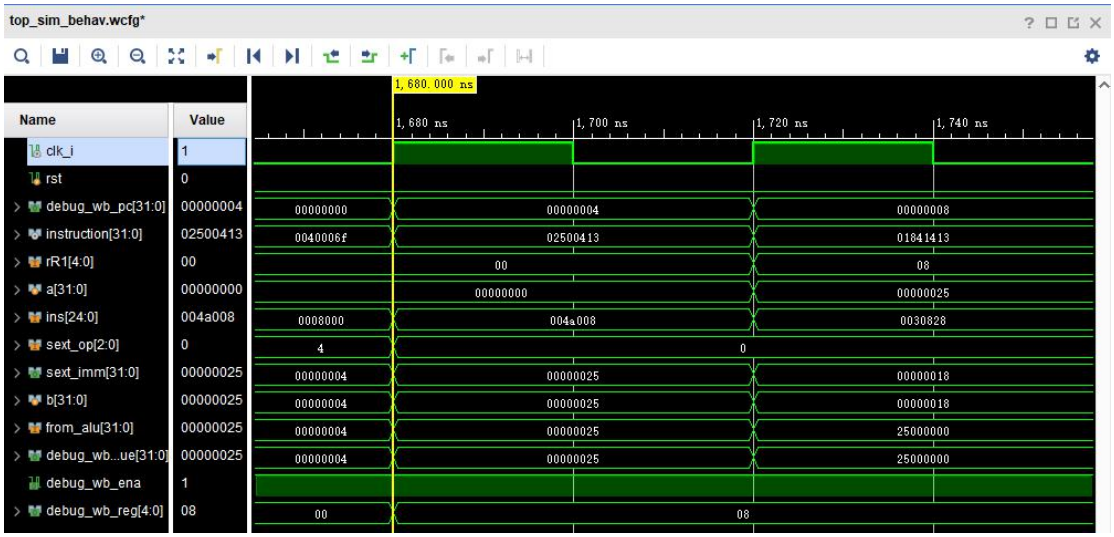


数码管显示逻辑。Top 模块获取数码管所需显示的数字 data 和进行时钟分频得到时钟信号 clk_1khz，传入 shumaguan 模块，由 shumaguan 模块计算出每个分频后时钟周期显示的数字 num 和数码管使能信号。

1.3 单周期 CPU 仿真及结果分析

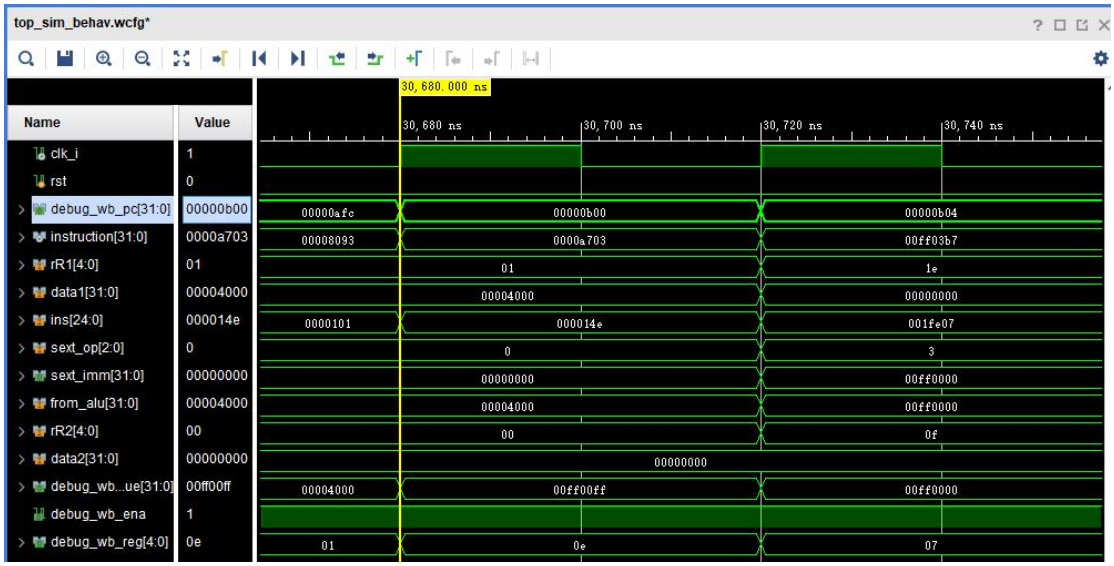
要求：包含逻辑运算、访存、分支跳转三类指令的仿真截图以及波形分析；每类

指令的截图和分析中，至少包含 1 条具体指令；截图需包含信号名和关键信号。



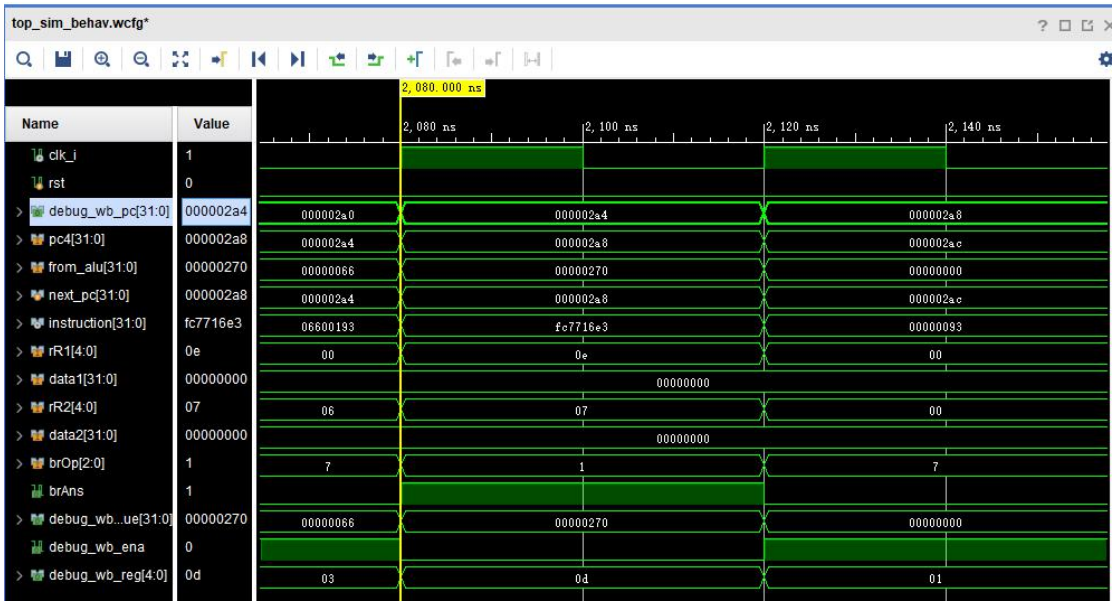
4: 02500413 addi x8,x0,37

在时钟上升沿获得当前 pc 后，从 IM 取出指令 instruction，分析指令。从寄存器堆中取出寄存器 rR1，即 0 号寄存器的内容作为 ALU 的第一个操作数，即 a=0；同时由 sext_op=0，判断将指令后 25 位做 I 型立即数扩展，得到 32 位的立即数 sext_imm 作为 ALU 的第二个操作数，即 b=0x25。两操作数在 ALU 中做加法操作得到结果，即 from_alu=0x25，并将该结果写回 8 号寄存器。



b00: 0000a703 lw x14,0(x1)

在时钟上升沿获得当前 pc 后，从 IM 取出指令 instruction，分析指令。从寄存器堆中取出寄存器 rR1，即 1 号寄存器的内容作为 ALU 的第一个操作数，即 a=0x4000；同时由 sext_op=0，判断将指令后 25 位做 I 型立即数扩展，得到 32 位的立即数 sext_imm 作为 ALU 的第二个操作数，即 b=0。两操作数在 ALU 中做加法操作得到结果，即 from_alu=0x4000。该结果作为内存地址，取出该地址中存储的内容 0xff00ff 写入 14 号寄存器。



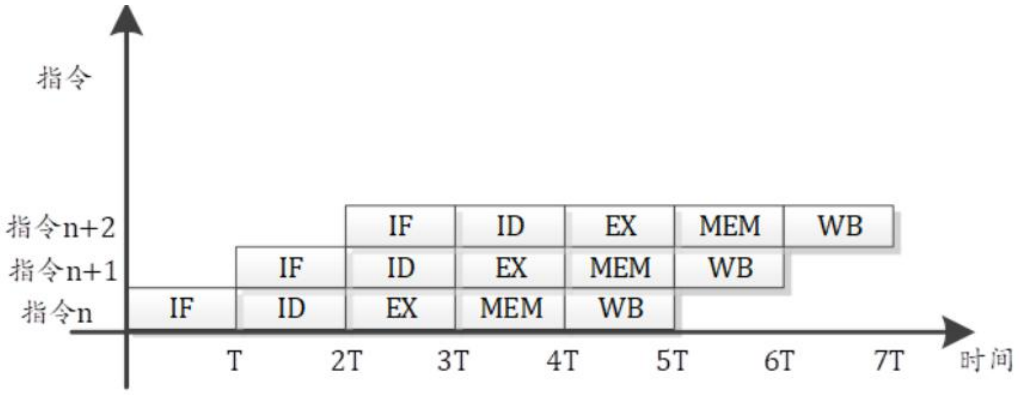
2a4: fc7716e3 bne x14,x7,270 <fail>

在时钟上升沿获得当前 pc 后，从 IM 取出指令 instruction，分析指令。从寄存器堆中取出寄存器 rR1，即 14 号寄存器的内容作为比较器的第一个操作数，即 data1=0；从寄存器堆中取出寄存器 rR2，即 7 号寄存器的内容作为比较器的第一个操作数，即 data2=0；由 brOp=1，判断两操作数是否相等，得判断结果相等，即 brAns=1 ，说明指令顺序执行，不跳转，所以将 pc4 作为下条 pc，即 next_pc=0x2a8，而非 ALU 得到的跳转地址 0x270。

2 流水线 CPU 设计与实现

2.1 流水线的划分

要求：画出流水线如何划分，说明每个流水级具备什么功能、需要完成哪些操作。



经典五级流水结构：

》取指 阶段（Instruction Fetch，IF）：是指将指令从指令存储器中读取出来的过程。

》译码 阶段（Instruction Decode，ID）：是指将取指阶段取出的指令进行翻译的过程。译码阶段将得到指令的操作码、功能码、操作数寄存器号等信息，然后从寄存器堆（Register File）取出操作数，同时产生指令执行所需的控制信号。

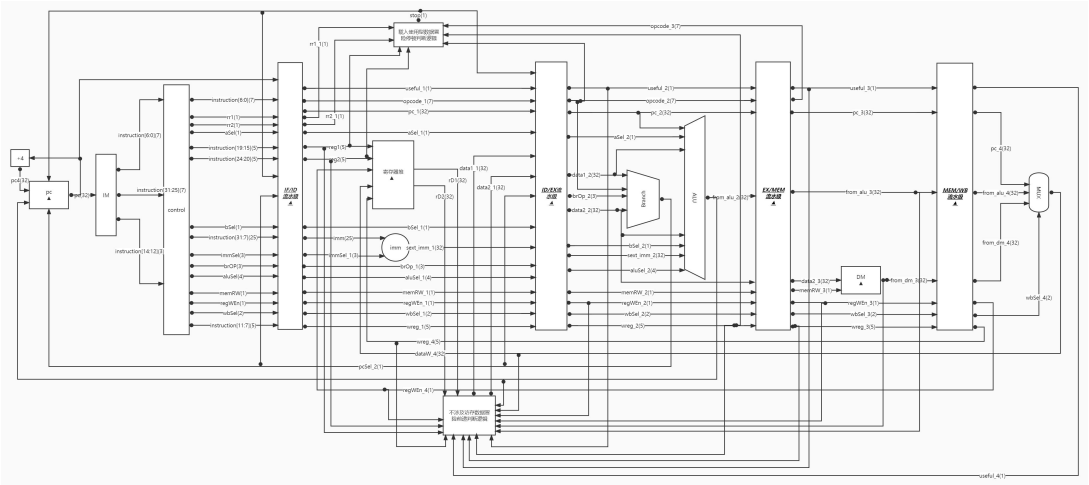
》执行 阶段（instruction EXecute，EX）：是指根据指令的操作码和功能码，对指令操作数进行运算的过程。如果指令是一条加法运算指令，则对操作数进行加法操作；如果是减法运算指令，则进行减法操作。CPU 中的算术逻辑单元（Arithmetic Logical Unit，ALU）是实施具体运算的硬件功能单元，是执行阶段的核心部件。

》访存 阶段（MEMory access，MEM）：是指访存指令从存储器读出数据，或将数据写入存储器的过程。

》写回 阶段（Write Back，WB）：是指将指令执行结果写回到目标寄存器的过程。运算类指令的执行结果是执行阶段的输出，而访存指令的执行结果则是访存阶段从存储器读取出的数据。

2.2 流水线 CPU 整体框图

要求：无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，以及说明每个模块的功能含义。



各个模块功能：

pc：更新 pc

指令 IM：指令寄存器

寄存器堆 regfile：从此获取 32 个寄存器存储内容

控制单元 control：给出控制信号

比较器 BranchComp：获取相应比较结果

计算器 ALU：获取相应计算结果

数据寄存器 DM：根据所给地址获得内存内容

MUX_WB：选择写回数据寄存器的内容

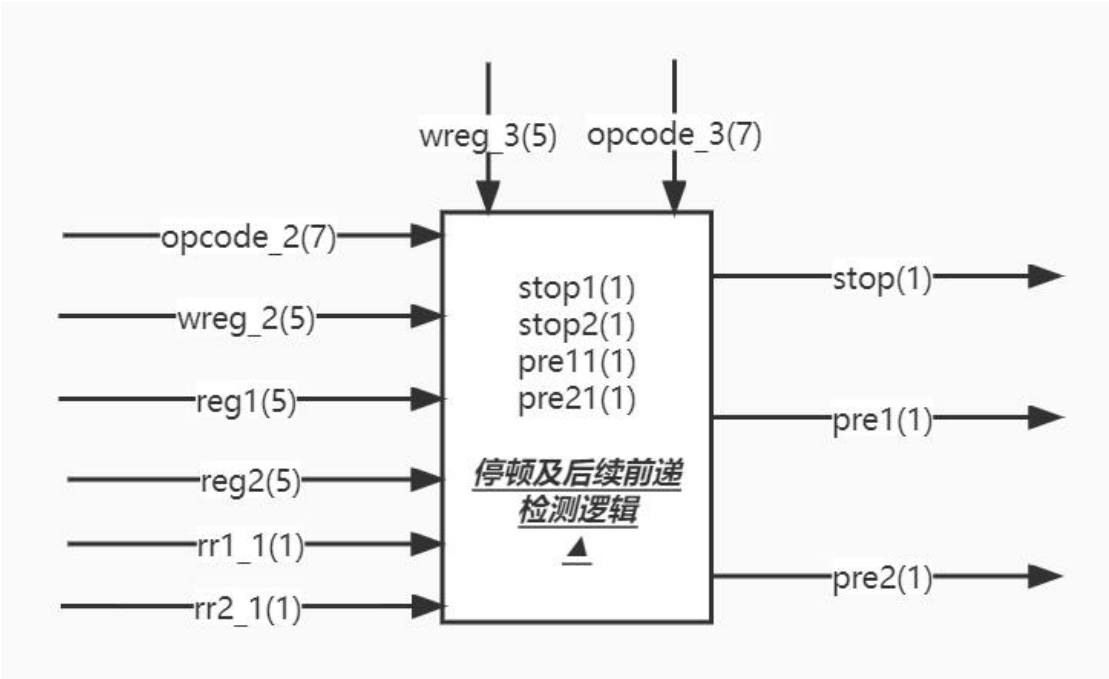
IF/ID 流水级、ID/EX 流水级、EX/MEM 流水级、MEM/WB 流水级：4 个用于暂存指令执行的中间结果的流水线寄存器

停顿判断逻辑：判断载入使用型数据冒险，并通过暂停和前递解决冒险

前递判断逻辑：判断不涉及 LW 指令的数据冒险，并通过前递解决冒险

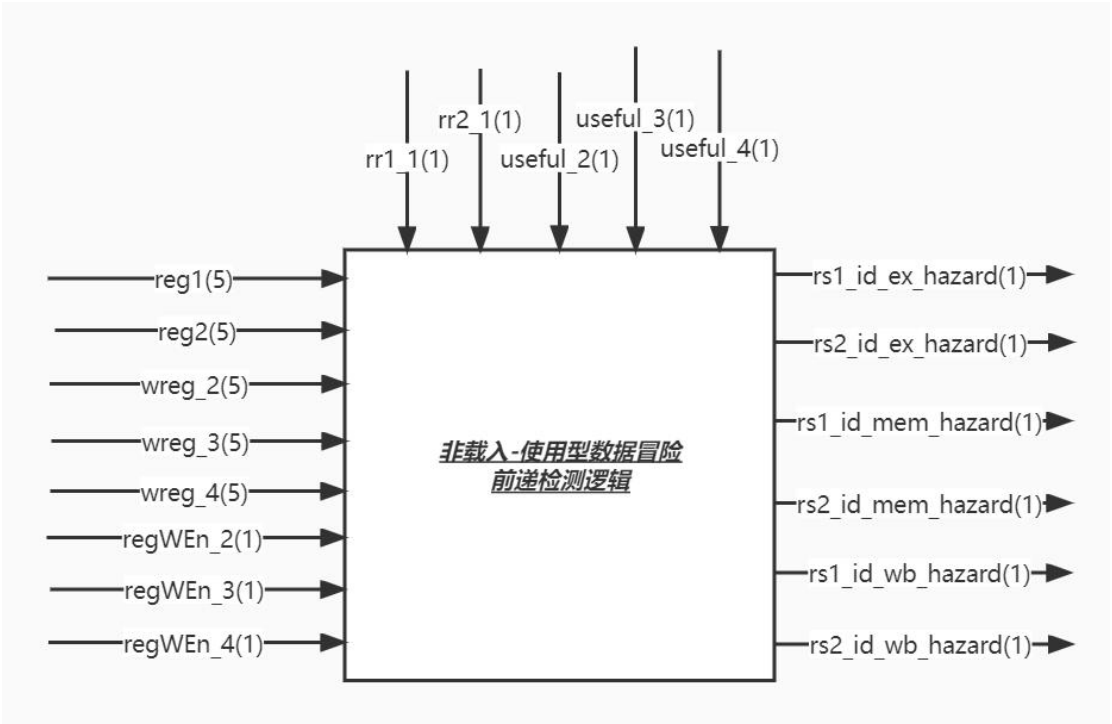
2.3 流水线 CPU 模块详细设计

要求：画出各个模块的详细设计图，包含内部的子模块，以及关键性逻辑；标出子模块接口信号名、各信号线的信号名和位宽，并有详细的解释说明；此外，必须结合模块图，详细说明数据冒险、控制冒险的解决方法。



当 EX 阶段正在执行取数指令(opcode_2==`LW_TYPE)，EX 阶段的写回寄存器与 ID 阶段要读取的寄存器 reg1/reg2 相同(wreg_2==reg1/reg2 且 rr1_1/rr2_1，其中 rr1_1/rr2_1 表示 ID 阶段要读取寄存器 reg1/reg2)时，说明此时 EX 阶段和 ID 阶段的两条指令发生载入-使用型数据冒险，采用组合逻辑，赋 stop1/stop2=1，而停顿信号 stop=stop1|stop2，得知此时得停顿。pre11 和 pre21 的赋值采用时序逻辑，复位信号无效时，在每个时钟上升沿将 pre11<=stop1、pre21<=stop2。停顿一个时钟周期后，利用 pre11 或 pre21 的改变将 stop 变回 0，以实现只停顿一个时钟周期。同时采用组合逻辑，赋停顿后续前递信号 pre1\pre2 为 pre11\pre21，或着当 MEM 阶段正在执行取数指令(opcode_3==`LW_TYPE)，MEM 阶段的写回寄存器与 ID 阶段要读取的寄存器 reg1/reg2 相同(wreg_3==reg1/reg2 且 rr1_1/rr2_1，其中 rr1_1/rr2_1 表示 ID 阶段要读取寄存器 reg1/reg2)时，将 pre1\pre2 赋为 1。停顿操作在部分部件设计中说明。
代码如下：

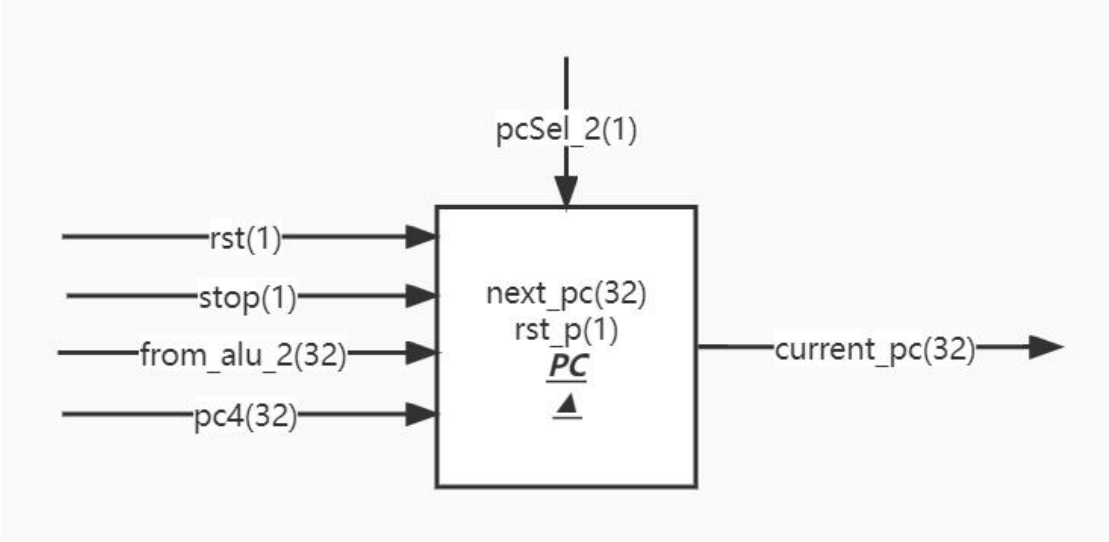
```
// 停顿1个时钟周期
reg pre11;
reg pre21;
wire    stop1   =   (opcode_2 == `LW_TYPE) & (wreg_2 == reg1) & rr1_1 & (!pre11);
wire    stop2   =   (opcode_2 == `LW_TYPE) & (wreg_2 == reg2) & rr2_1 & (!pre21);
wire    stop    =   stop1   | stop2;
always @(posedge clk) begin
    if(rst)         pre11 <= 0;
    else            pre11 <= stop1;
end
wire  pre1    =   pre11 | ((opcode_3 == `LW_TYPE) & (wreg_3 == reg1) & rr1_1);
always @(posedge clk) begin
    if(rst)         pre21 <= 0;
    else            pre21   <= stop2;
end
wire  pre2    =   pre21 | ((opcode_3 == `LW_TYPE) & (wreg_3 == reg2) & rr2_1);
```



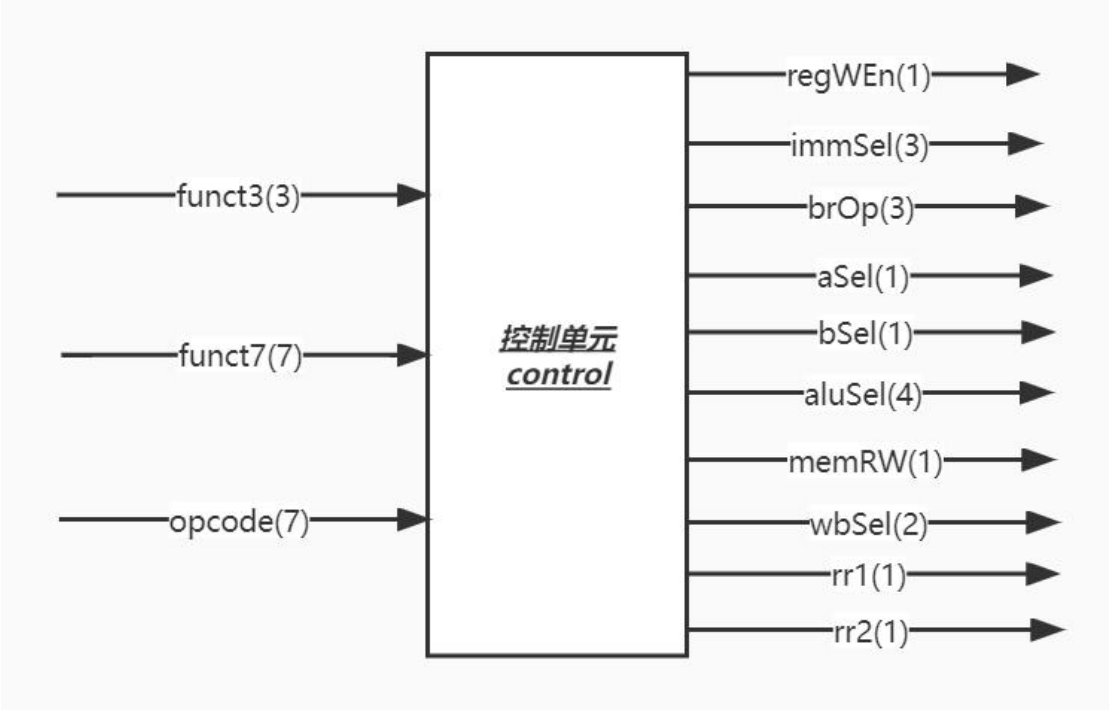
当 ID 阶段正在读寄存器 reg1/reg2（即 rr1_1/rr2_1）时，若 EX 阶段的指令要写寄存器 wreg_2（即 regWEn_2），同时两寄存器是同一个（即 reg1== wreg_2/reg2== wreg_2），则检测到 RAW 情形 A，设 rs1_id_ex_hazard=1/ rs2_id_ex_hazard=1；若 MEM 阶段的指令要写寄存器 wreg_3（即 regWEn_3），同时两寄存器是同一个（即 reg1== wreg_3/reg2== wreg_3），则检测到 RAW 情形 B，设 rs1_id_mem_hazard=1/ rs2_id_mem_hazard=1；若 WB 阶段的指令要写寄存器 wreg_4（即 regWEn_4），同时两寄存器是同一个（即 reg1== wreg_4/reg2== wreg_4），则检测到 RAW 情形 C，设 rs1_id_wb_hazard=1/ rs2_id_wb_hazard=1。都需要前递。

前递的具体操作：

pre1=1：将 MEM 阶段从 DM 中取来的数(from_dm_3)赋给 data1_1
pre2=1：将 MEM 阶段从 DM 中取来的数(from_dm_3)赋给 data2_1
rs1_id_ex_hazard=1：将 EX 阶段 ALU 计算出的结果(from_alu_2)赋给 data1_1
rs2_id_ex_hazard=1：将 EX 阶段 ALU 计算出的结果(from_alu_2)赋给 data2_1
rs1_id_mem_hazard=1：将 MEM 阶段 ALU 计算出的结果(from_alu_3)赋给 data1_1
rs2_id_mem_hazard=1：将 MEM 阶段 ALU 计算出的结果(from_alu_3)赋给 data2_1
rs1_id_wb_hazard=1：将 WB 阶段 ALU 计算出的结果(from_alu_4)赋给 data1_1
rs2_id_wb_hazard=1：将 WB 阶段 ALU 计算出的结果(from_alu_4)赋给 data2_1



next_pc 的赋值采用组合逻辑，当 EX 阶段传来的 pcSel_2 为 0 时表示跳转，被赋为 EX 阶段计算出来的地址 from_alu_2；当 pcSel_2 为 1 时表示不跳转，被赋为 pc4（恒为 pc+4）。复位信号 rst 无效一个时钟周期后的时钟上升沿信号 rst_p 无效。current_pc 的赋值采用时序逻辑，在每个时钟上升沿，当 rst_p 有效时，被赋值为 0；其它当停顿检测逻辑传来的信号有效，即 stop=1 时，保持；其他情况默认赋值为 next_pc。以实现 pc 的更新。



控制信号的赋值均采用组合逻辑。

》regWEn：当指令为 S 型指令或 B 型指令时，不写寄存器，regWEn 设为 0，否则设为 1。

》immSel：根据 opcode 判断指令类型选择如何对立即数进行扩展。当指令为 I 型指令、LW 指令、jalr 指令时，immSel 设为`I_SEXT；当指令为 S 型指令时，immSel 设为`S_SEXT ；当指令为 B 型指令时，immSel 设为`B_SEXT；当指令为 U 型指令时，immSel 设为`U_SEXT；当指令为 J 型指令时，immSel 设为`J_SEXT；其余情况默认 immSel 设为 3'b111。

》brOp：当指令类型为 B 型指令时，比较器做何比较由 funct3 决定，直接将 brOp 设为 funct3；其余情况默认 brOp 设为 3'b111。

》aSel：根据指令类型判断 ALU 第一个操作数。当指令类型为 B 型指令或 J 型指令时， aSel 设为 0；否则 aSel 设为 1。

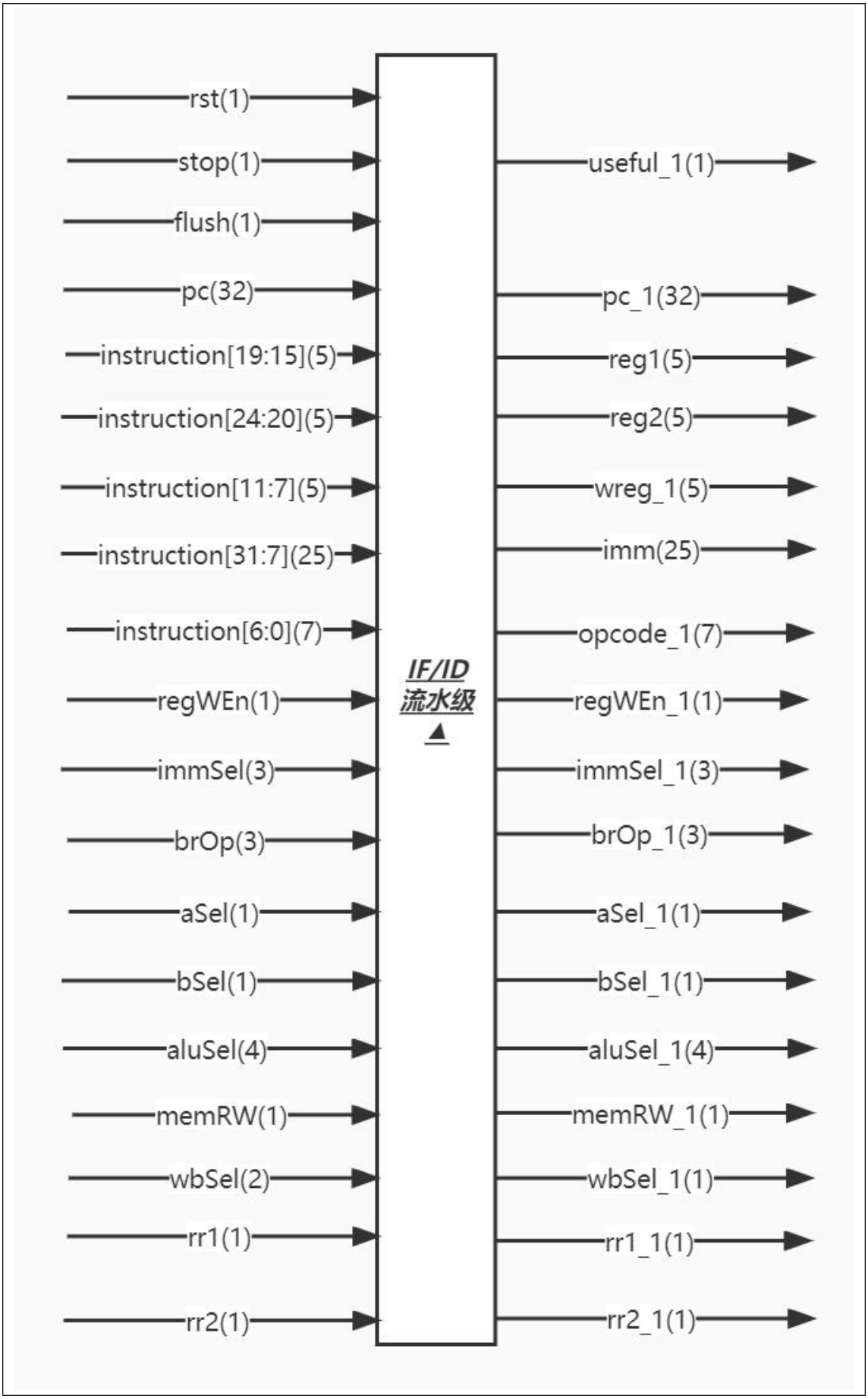
》bSel：根据指令类型判断 ALU 第二个操作数。当指令类型为 R 型指令时， bSel 设为 0；否则 bSel 设为 1。

》aluSel：当指令类型为 R 型指令时，根据 funct3 和 funct7 判断计算器进行的操作类型，分别将 aluSel 设为`SUB、`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令类型为 I 型指令时，根据 funct3 和 funct7 判断计算器进行的操作类型，分别将 aluSel 设为`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令为 LW 指令、jalr 指令、S 型指令、B 型指令、J 型指令时，aluSel 设为`ADD；当指令为 U 型指令时，aluSel 设为`LUI；其余情况默认 aluSel 设为`NOOP。

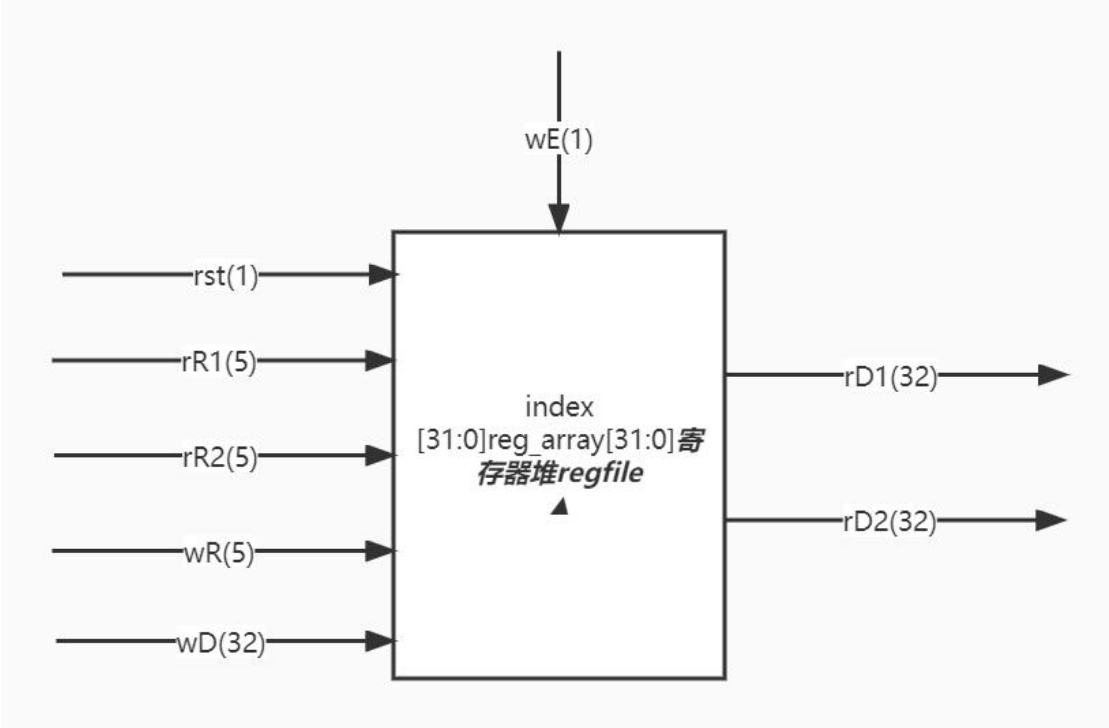
》memRW：根据指令类型判断读写内存。当指令类型为 S 型指令时需要写内存，memRW 设为 1；其余情况默认不写内存，设为 0。

》wbSel：根据指令类型选择写回寄存器的内容。当指令类型为 J 型指令、jalr 指令时， wbSel 设为`PC4；当指令为 LW 指令时， wbSel 设为`FROM_DM；其余情况默认 wbSel 设为`FROM_ALU。

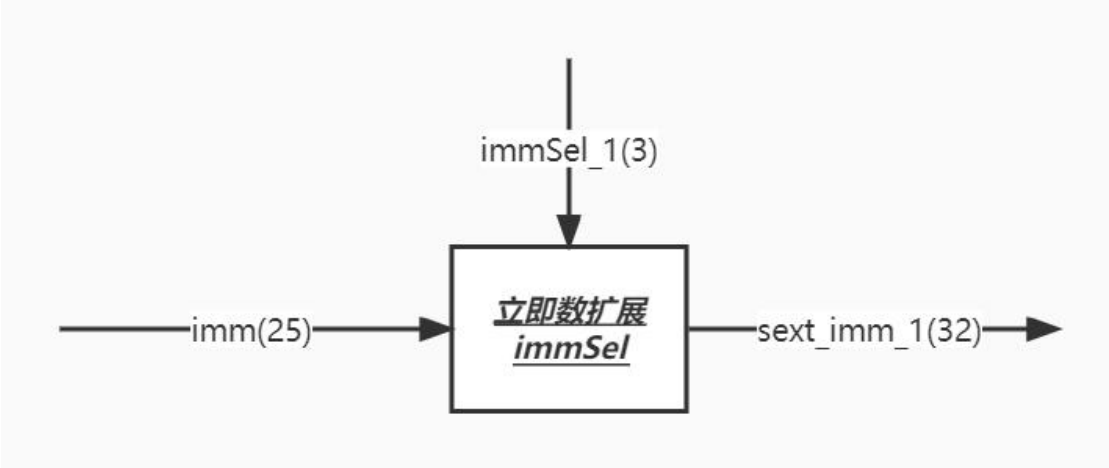
》rr1：根据指令类型判断是否需要读取寄存器 reg1 中的内容。当指令为 R 型指令、I 型指令、LW 型指令、jalr 指令、S 型指令、B 型指令时，rr1 设为 1；当指令为 U 型指令、J 型指令时，rr1 设为 0；其他情况默认设为 0。
》rr2：根据指令类型判断是否需要读取寄存器 reg2 中的内容。当指令为 R 型指令、S 型指令、B 型指令时，rr2 设为 1；当指令为 I 型指令、LW 型指令、jalr 指令、U 型指令、J 型指令时，rr2 设为 0；其他情况默认设为 0。


rst(1)
stop(1)
flush(1)
pc(32)
instruction[19:15](5)
instruction[24:20](5)
instruction[11:7](5)
instruction[31:7](25)
instruction[6:0](7)
regWEn(1)
immSel(3)
brOp(3)
aSel(1)
bSel(1)
aluSel(4)
memRW(1)
wbSel(2)
rr1(1)
rr2(1)
IF/ID
流水级
useful_1(1)
pc_1(32)
reg1(5)
reg2(5)
wreg_1(5)
imm(25)
opcode_1(7)
regWEn_1(1)
immSel_1(3)
brOp_1(3)
aSel_1(1)
bSel_1(1)
aluSel_1(4)
memRW_1(1)
wbSel_1(1)
rr1_1(1)
rr2_1(1)

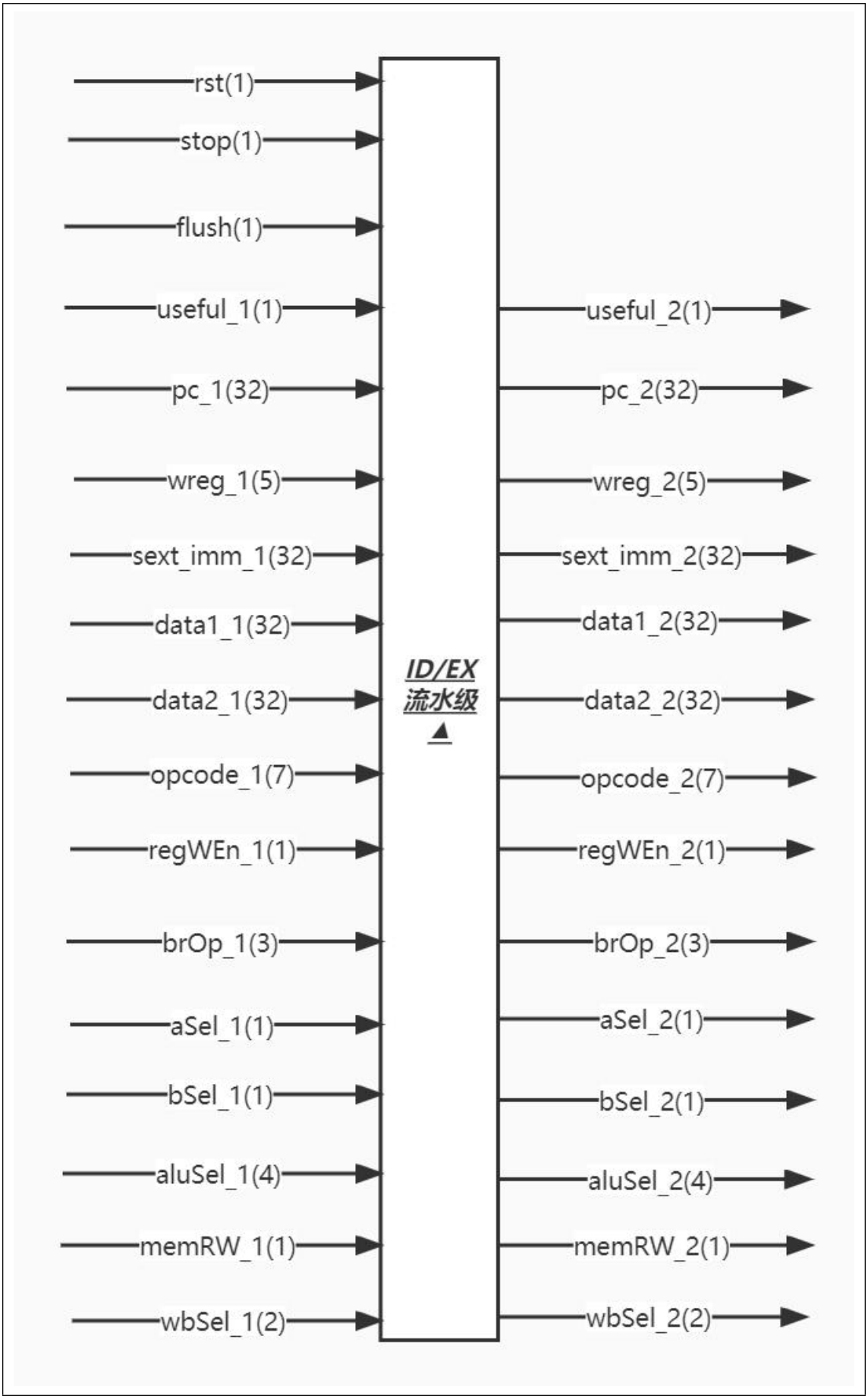
所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：
当复位信号有效，即 rst=1 时，或要清空流水级，即 flush（EX 阶段得到的 pcSel）=0 时，要将相应的寄存器清零，useful_1 设为'b0；pc_1 设为 32'b0；reg1 设为 5'b0；reg2 设为 5'b0；wreg_1 设为 5'b0；imm 设为 25'b0；opcode_1 设为 7'b0；regWEn_1 设为 1；immSel_1 设为'b111；brOp_1 设为'b111；aSel_1 设为 1；bSel_1 设为 1；aluSel_1 设为`NOOP；memRW_1 设为 0；wbSel_1 设为`FROM_ALU；rr1_1 设为 0；rr2_1 设为 0。
当来自停顿判断逻辑的停顿信号有效，即 stop=1 时，所有输出信号保持。
其它情况默认，useful_1 设为'b1；pc_1 设为 pc；reg1 设为 instruction[19:15]；reg2 设为 instruction[24:20]；wreg_1 设为 instruction[11:7]；imm 设为 instruction[31:7]；opcode_1 设为 instruction[6:0]；regWEn_1 设为 regWEn；immSel_1 设为 immSel；brOp_1 设为 brOp；aSel_1 设为 aSel；bSel_1 设为 bSel；aluSel_1 设为 aluSel；memRW_1 设为 memRW；wbSel_1 设为 wbSel；rr1_1 设为 rr1；rr2_1 设为 rr2。



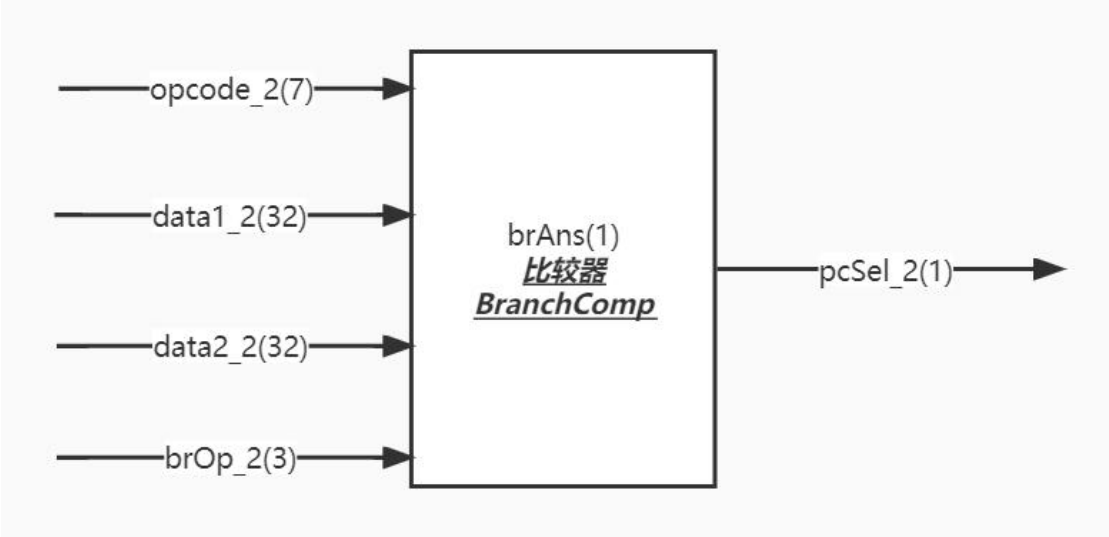
读寄存器采用组合逻辑，若 rR1 不是 0 号寄存器，将其中存储的内容赋给 rD1，否则将 rD1 设为 0；若 rR2 不是 0 号寄存器，将其中存储的内容赋给 rD2，否则将 rD2 设为 0。
写寄存器采用时序逻辑，每个时钟上升沿来临时，0 号寄存器 reg_array[0]恒设为 0，复位信号有效时，所有寄存器设为 0。不复位时，若写寄存器堆信号有效，即 wE 为 1 时，且目标寄存器不是 0 号寄存器，则将 reg_array[wR]设为 wD。



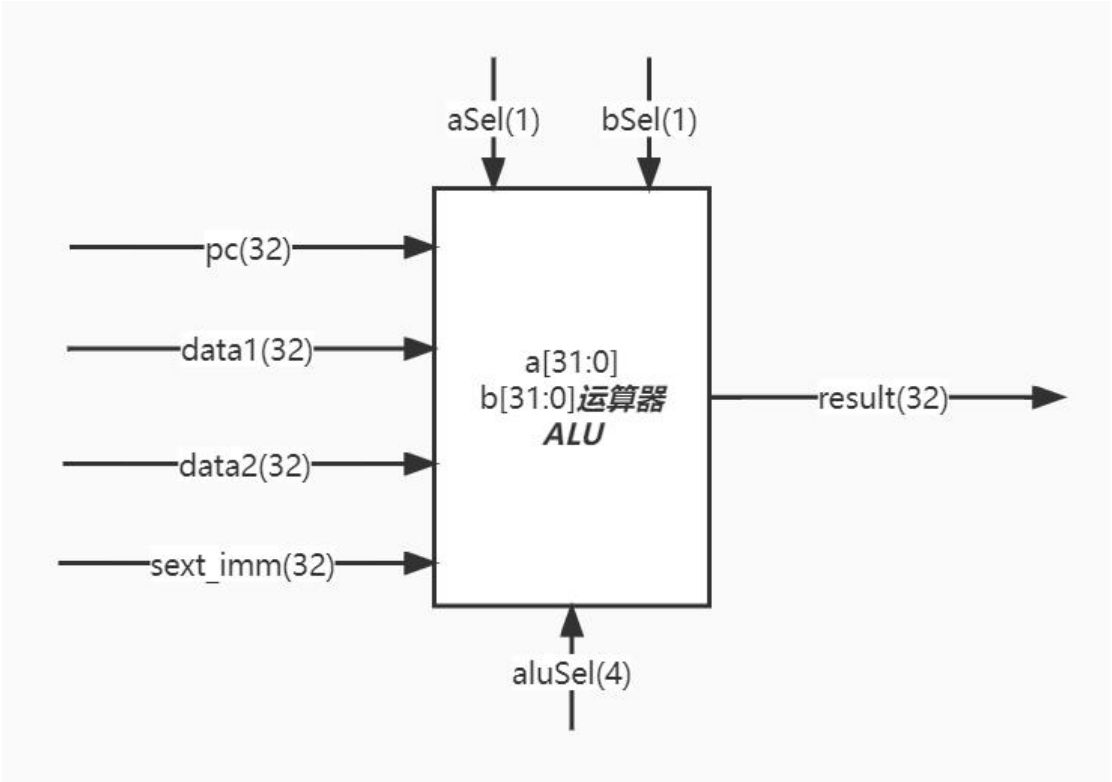
根据控制信号对 25 位的数 imm 进行相应立即数扩展为 32 位的 sext_imm_1。当 immSel_1 为`I_SEXT 时，进行 I 型立即数扩展；当 immSel_1 为`S_SEXT 时，进行 S 型立即数扩展；当 immSel_1 为`B_SEXT，进行 B 型立即数扩展；当 immSel_1 为`U_SEXT 时，进行 U 型立即数扩展；当 immSel_1 为`J_SEXT 时，进行 J 型立即数扩展；其余情况默认设置 sext_imm_1 为 0。（与单周期设计相同）


rst(1)
stop(1)
flush(1)
useful_1(1)
pc_1(32)
wreg_1(5)
sext_imm_1(32)
data1_1(32)
data2_1(32)
opcode_1(7)
regWEn_1(1)
brOp_1(3)
aSel_1(1)
bSel_1(1)
aluSel_1(4)
memRW_1(1)
wbSel_1(2)
ID/EX
流水级
useful_2(1)
pc_2(32)
wreg_2(5)
sext_imm_2(32)
data1_2(32)
data2_2(32)
opcode_2(7)
regWEn_2(1)
brOp_2(3)
aSel_2(1)
bSel_2(1)
aluSel_2(4)
memRW_2(1)
wbSel_2(2)

所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：
当复位信号有效，即 rst=1 时，或要清空流水级，即 flush（EX 阶段得到的 pcSel）=0 时，要将相应的寄存器清零，useful_2 设为'b0；pc_2 设为 32'b0； wreg_2 设为 5'b0；sext_imm_2 设为 32'b0；data1_2 设为 32'b0；data2_2 设为 32'b0；opcode_2 设为 7'b0；regWEn_2 设为 0；brOp_2 设为'b111；aSel_2 设为 1；bSel_2 设为 1；aluSel_2 设为`NOOP；memRW_2 设为 0；wbSel_2 设为`FROM_ALU。
当来自停顿判断逻辑的停顿信号有效，即 stop=1 时，useful_2 设为'b0；pc_2 设为 pc_1； wreg_2 设为 wreg_1；sext_imm_2 设为 sext_imm_1；data1_2 设为 data1_1；data2_2 设为 data2_1；opcode_2 设为 opcode_1；regWEn_2 设为 0；brOp_2 设为 brOp_1；aSel_2 设为 aSel_1；bSel_2 设为 bSel_1；aluSel_2 设为 aluSel_1；memRW_2 设为 0；wbSel_2 设为 wbSel_1。
其余情况默认将 useful_2 设为 useful_1；pc_2 设为 pc_1； wreg_2 设为 wreg_1；sext_imm_2 设为 sext_imm_1；data1_2 设为 data1_1；data2_2 设为 data2_1；opcode_2 设为 opcode_1；regWEn_2 设为 regWEn_1；brOp_2 设为 brOp_1；aSel_2 设为 aSel_1；bSel_2 设为 bSel_1；aluSel_2 设为 aluSel_1；memRW_2 设为 memRW_1；wbSel_2 设为 wbSel_1。

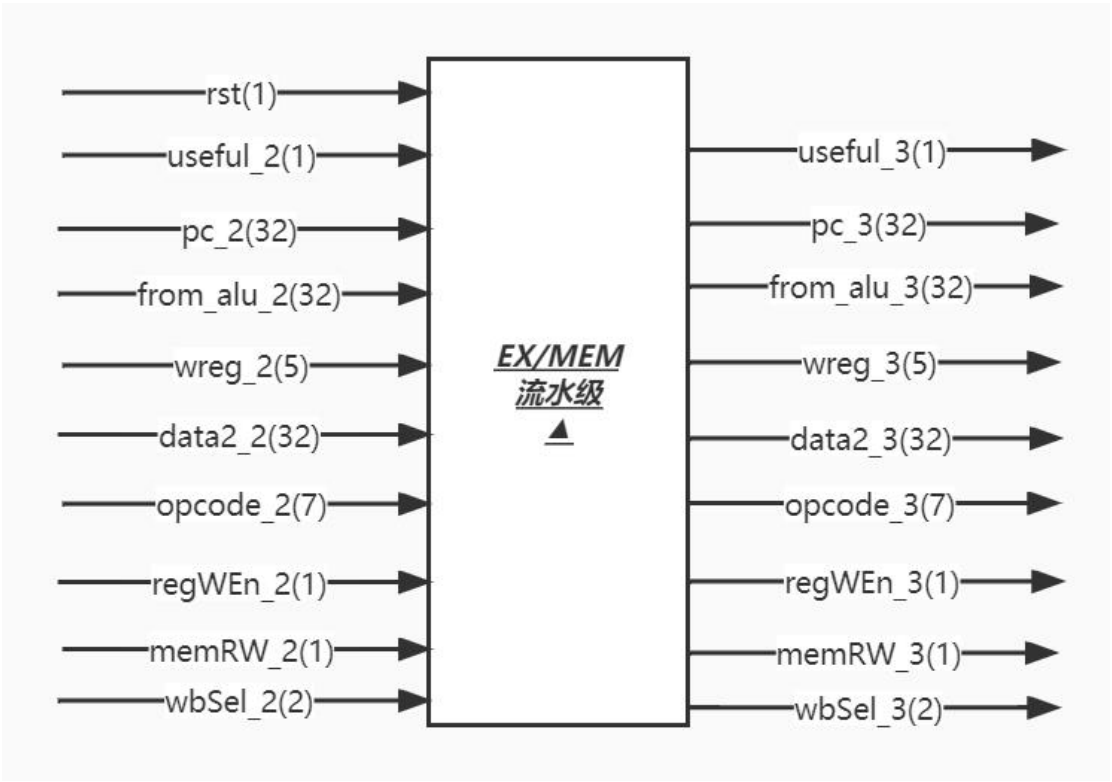


brOp_2 为`BEQ 时，data1_2==data2_2 则将 brAns 设为 0，否则为 1；为`BNE 时，与`BEQ 相反；为`BLT 时，data1_2<data2_2 则将 brAns 设为 0，否则为 1；为`BGE 时，与`BLT 相反。
同时根据 EX 阶段所执行的指令类型，即 opcode_2 判断 pcSel，当指令为 R 型指令、I 型指令、LW 指令、S 型指令、U 型指令时，指令不跳转，pcSel_2 设为 1；当指令为 B 型指令时，由比较结果判断指令跳转与否，pcSel_2 设为 brAns；当指令为 J 型指令或 jalr 指令，指令跳转，pcSel_2 设为 0；其余情况默认指令不跳转，pcSel_2 设为 1。

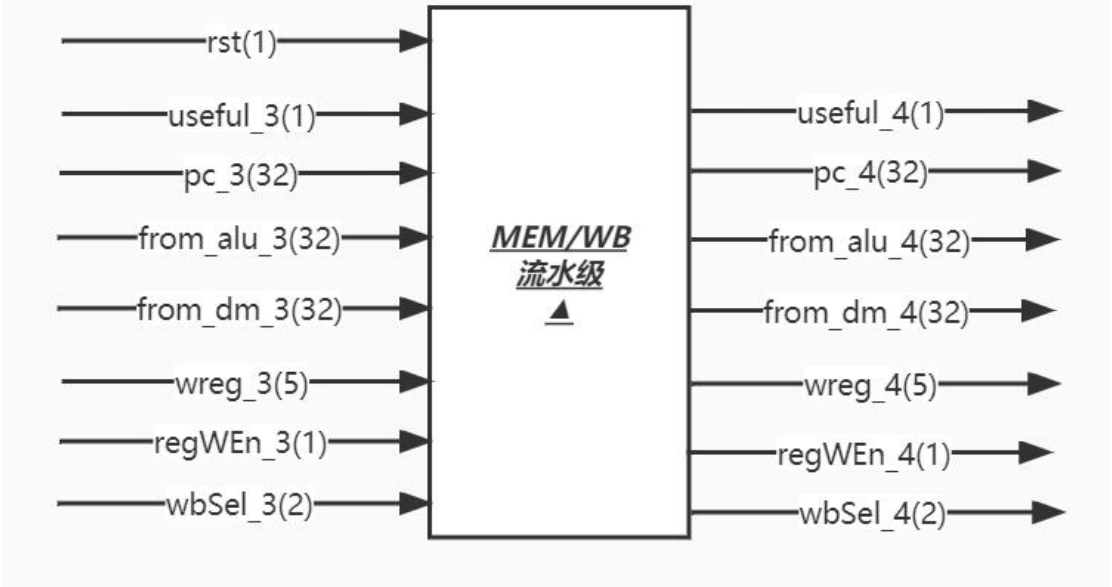


运算器采用组合逻辑。

aSel 为 0 时，ALU 第一个操作数 a 选择 pc，为 1 时选择 reg1 存储的数。bSel 为 0 时，ALU 第二个操作数 b 选择 reg2 存储的数，为 1 时选择扩展后的立即数。当 aluSel 为`ADD 时，ALU 对两操作数进行加法操作（result=a+b）；当为`SUB 时，做减法操作（result=a-b）；当为`AND 时，做与操作（result=a&b）；当为`OR 时，做或操作（result=a|b）；当为`XOR 时，做异或操作（result=a^b）；当为`SLL 时，做左移操作（result=a<<b[4:0]）；当为`SRL 时，做逻辑右移操作（result=a>>b[4:0]）；当为`SRA 时，做算数右移操作（result=($signed(a))>>>b[4:0]）；当为`LUI 时，直接将 b 赋值给 result（result=b）；其他情况默认将 result 直接赋值为 0。（与单周期设计相同）

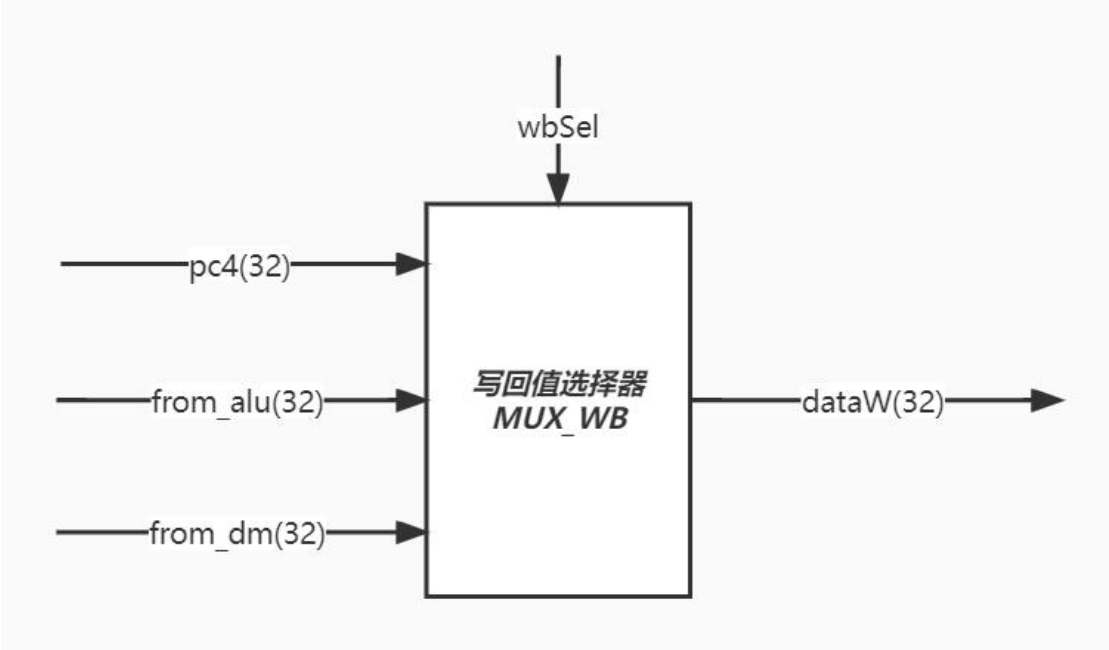


所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：
当复位信号有效，即 rst=1 时，useful_3 设为'b0；pc_3 设为 32'b0；from_alu_3 设为 32'b0；wreg_3 设为 5'b0；data2_3 设为 32'b0；opcode_3 设为 7'b0；regWEn_3 设为 0；memRW_3 设为 0；wbSel_3 设为`FROM_ALU。
其余情况默认将 useful_3 设为 useful_2；pc_3 设为 pc_2；from_alu_3 设为 from_alu_2；wreg_3 设为 wreg_2；data2_3 设为 data2_2；opcode_3 设为 opcode_2；regWEn_3 设为 regWEn_2；memRW_3 设为 memRW_2；wbSel_3 设为 wbSel_2。

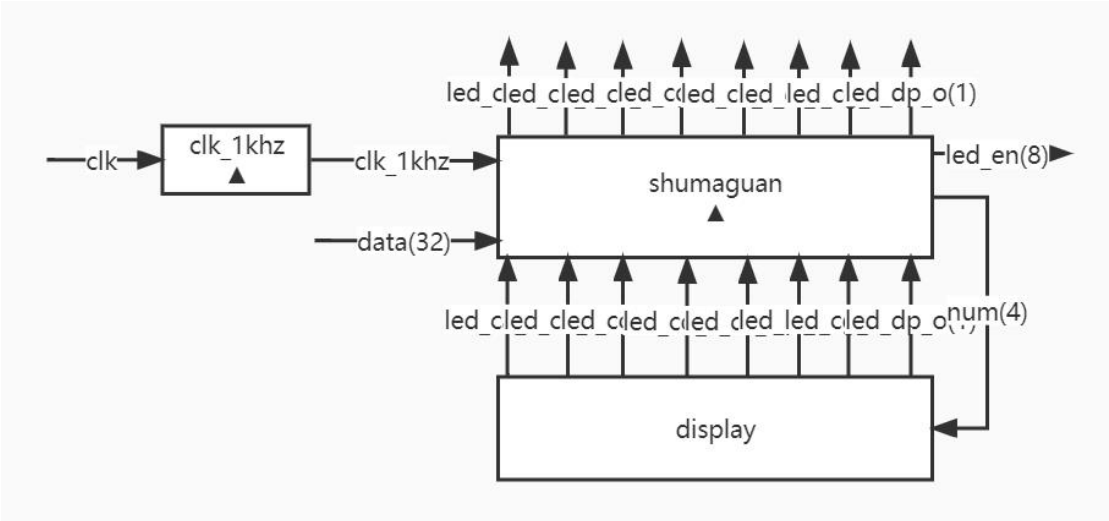


所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：
当复位信号有效，即 rst=1 时，useful_4 设为'b0；pc_4 设为 32'b0；from_alu_4

设为 32'b0;from_dm_4 设为 32'b0;wreg_4 设为 5'b0;regWEn_4 设为 0；wbSel_4 设为`FROM_ALU。
其余情况默认将 useful_4 设为 useful_3；pc_4 设为 pc_3；from_alu_4 设为 from_alu_3；from_dm_4 设为 from_dm_3；wreg_4 设为 wreg_3；regWEn_4 设为 regWEn_3；wbSel_4 设为 wbSel_3。



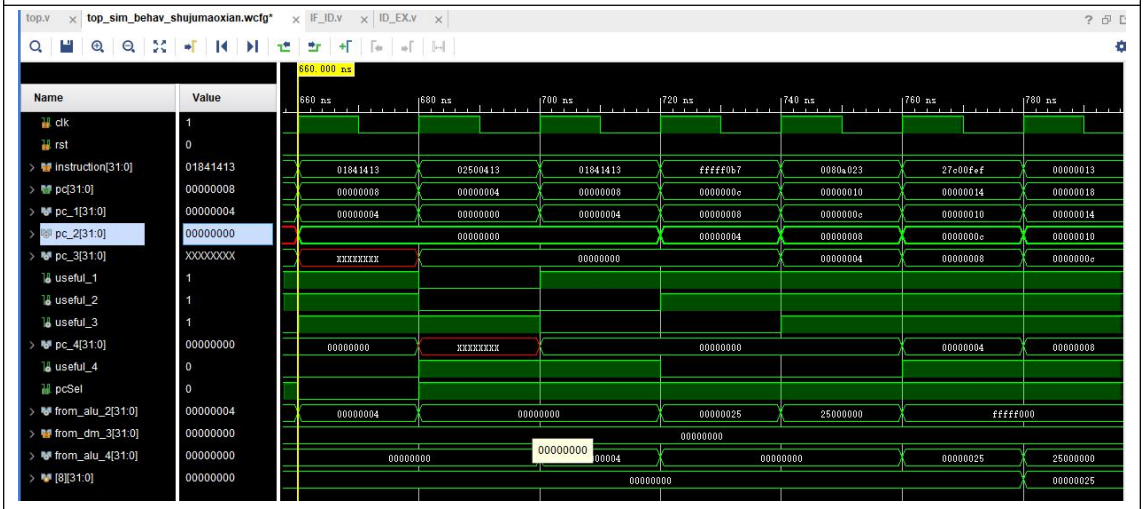
寄存器堆写回值 dataW 的赋值采用组合逻辑。当 wbSel=`PC4 时，选择 pc+4；当 wbSel=`FROM_ALU 时，选择 ALU 的计算结果；当 wbSel=`FROM_DM 时，选择从 DM 中取出的结果。（与单周期设计相同）



数码管显示逻辑。Top 模块获取数码管所需显示的数字 data 和进行时钟分频得到时钟信号 clk_1khz，传入 shumaguan 模块，由 shumaguan 模块计算出每个分频后时钟周期显示的数字 num 和数码管使能信号。（与单周期设计相同）

2.4 流水线 CPU 仿真及结果分析

要求：包含控制冒险和数据冒险三种情形的仿真截图，以及波形分析。若仅实现了理想流水，则此处贴上理想流水的仿真截图及详细的波形分析。

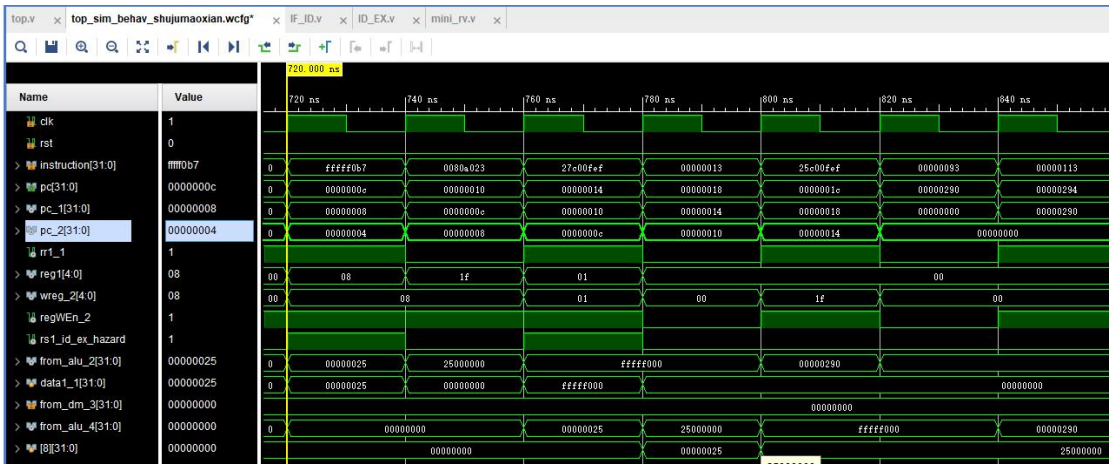


控制冒险：

0:　0040006f　　jal　x0,4 <reset_vector>

4:　02500413　　addi　x8,x0,37

我采用静态预测解决控制冒险，总是预测跳转。图中当第一条指令到达 EX 阶段(pc_2=0x0)时，得知下条指令该跳转(pcSel=0)，跳转目的地址为 from_alu_2=0x4，预测错误，一个时钟周期后，pc=0x4，清空 IF/ID 流水级(pc_1=0)、ID/EX 流水级(pc_2=0)，ID、EX 阶段的指令设为无效(useful_1=0、useful_2=0)。在经过四个时钟周期后，第二条指令执行完成，又一个时钟周期后将其计算结果 0x25 写入寄存器 reg8。

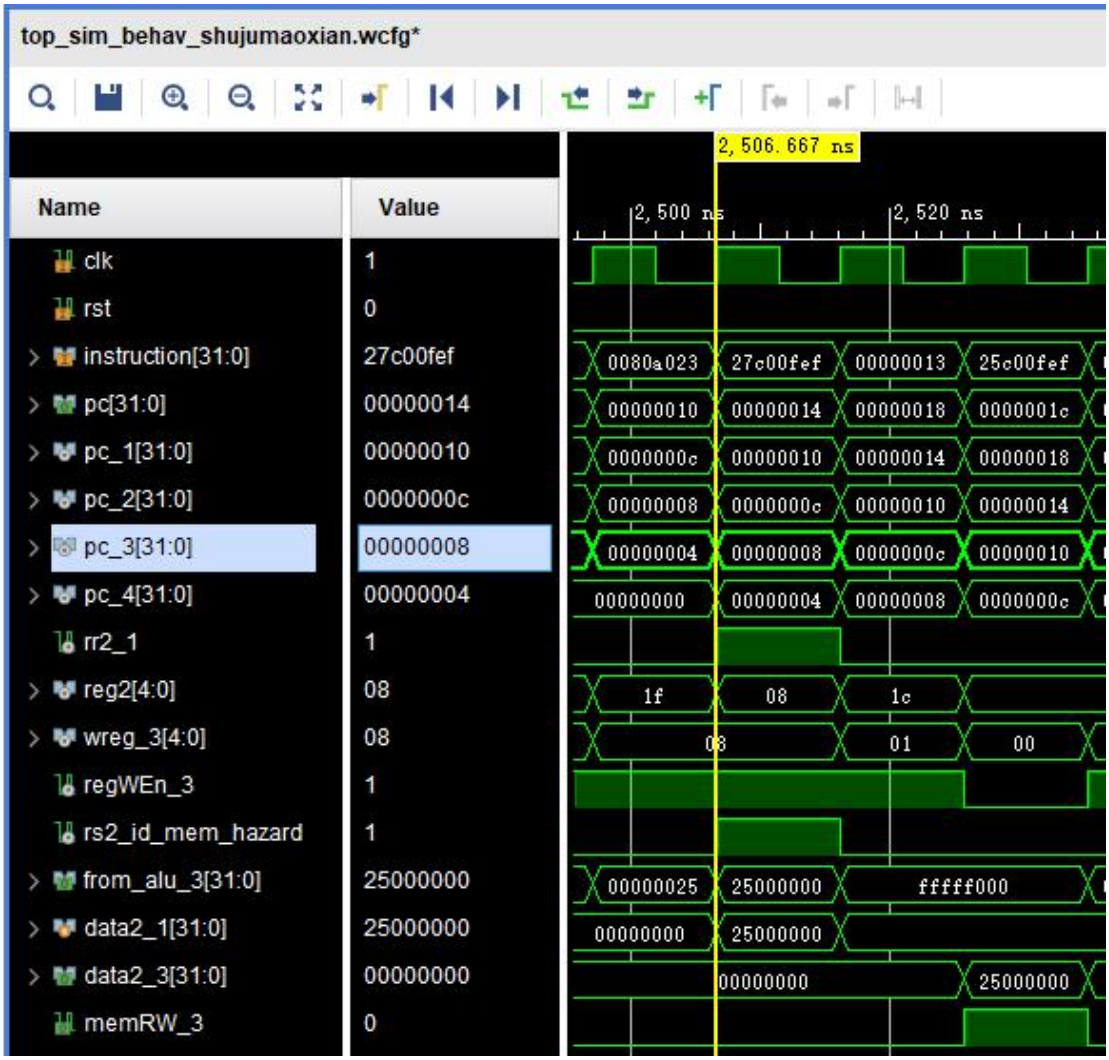


RAW 情形 A：

4:　02500413　　addi　x8,x0,37

8:　01841413　　slli　x8,x8,0x18

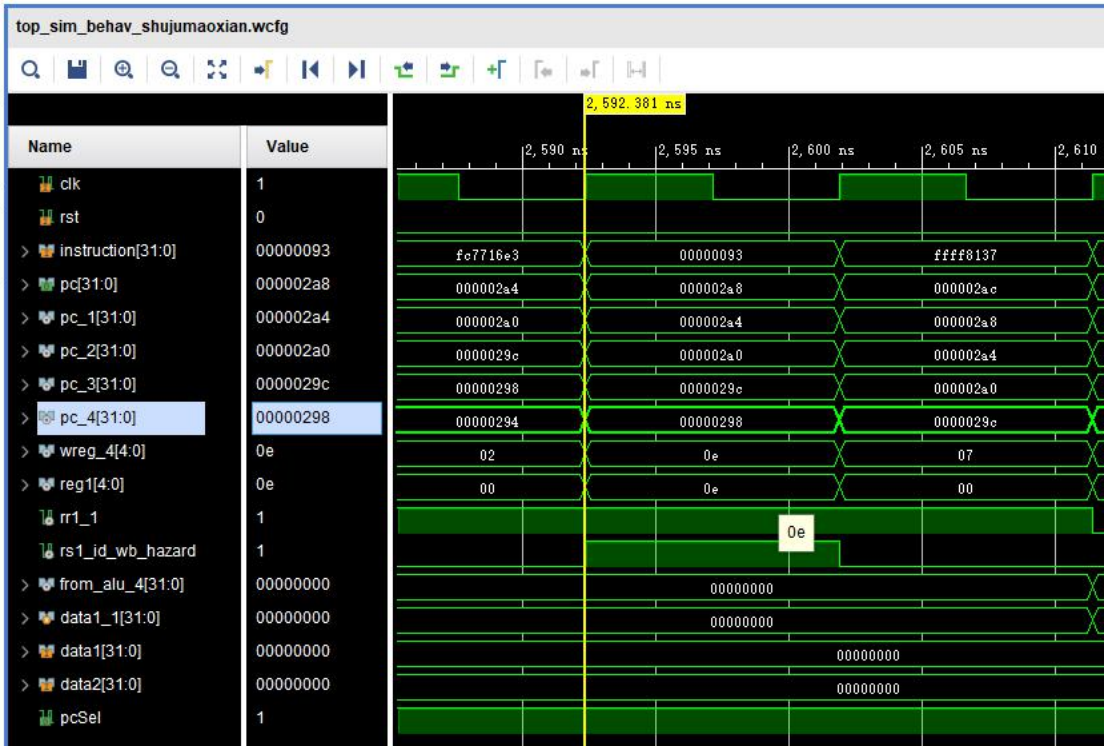
我采用前递解决该数据冒险。当第一条指令进入 EX 阶段、第二条指令在 ID 阶段时，检测到 RAW 情形 A(rs1_id_ex_hazard=1)，于是采用组合逻辑直接将 EX 阶段 ALU 计算结果 from_alu_2 前递给 data1_1，在下一个时钟周期直接作为 ALU 的第一个操作数，以保证第二条指令计算结果的正确。故又过三个时钟周期后将两条指令正确执行所得结果 0x25000000 写回寄存器 reg8。



8: 01841413 slli x8,x8,0x18
c: fffff0b7 lui x1,0xfffff
10: 0080a023 sw x8,0(x1) # fffff000 <_end+0xffffaf90>

RAW 情形 B：

我采用前递解决该数据冒险。当第一条指令进入 MEM 阶段、第三条指令在 ID 阶段时，检测到 RAW 情形 B(rs2_id_mem_hazard=1)，于是采用组合逻辑直接将 MEM 阶段 ALU 计算结果 from_alu_3 前递给 data2_1，在下一个时钟周期直接作为 ALU 的第二个操作数，以保证第三条指令计算结果的正确。故又过两个时钟周期后将要写入内存中的正确内容 data2_3=0x25000000 写入内存地址 from_alu_3=0xfffff000。



298: 00208733 add x14,x1,x2
29c: 00000393 addi x7,x0,0
2a0: 06600193 addi x3,x0,102
2a4: fc7716e3 bne x14,x7,270 <fail>

RAW 情形 C：

我采用前递解决该数据冒险。当第一条指令进入 WB 阶段、第四条指令在 ID 阶段时，检测到 RAW 情形 C(rs1_id_wb_hazard=1)，于是采用组合逻辑直接将 WB 阶段 ALU 计算结果 from_alu_4 前递给 data1_1，在下一个时钟周期直接作为 BranchComp 的第一个操作数，以保证第四条指令比较结果的正确。故一个时钟周期后，BranchComp 中两操作数 data1=data2，指令不跳转(pcSel=1)。

3 设计过程中遇到的问题及解决方法

要求:包括设计过程中遇到的有价值的错误,或测试过程中遇到的有价值的问题。所谓有价值，指的是解决该错误或问题后，能够学到新的知识和技巧，或加深对已有知识的理解和运用。

1. 在比较器 BranchComp 中进行有符号数的比较时，直接用“>”、“≤”判断时，默认进行的是无符号数的判断，改正后应该考虑分为两操作数同号和异号两种情况。
2. 对于时序逻辑电路，输出信号相比时钟有效边沿存在一定的输出延时，在单周期 CPU 设计时，若寄存器堆的读取采用组合逻辑，可能读取到 pc 还未来得及更新时的错误值，上板验证后发现结果也确实不正确，且存在电路不稳定的现象，改正后采取在时钟的上升沿更新 pc，在时钟的下降沿读取寄存器堆 RF。而在流水线 CPU 设计时，各个阶段总在执行不同的指令，为保证指令获得正确的执行结果，寄存器堆的读取操作应改成组合逻辑。
3. 运算器中对操作数 a 进行移位操作时，最多移动 31 位，故仅取操作数 b 的低 5 位作为移位位数 shamt。考虑到算数右移有符号数的情况，代码应写成“result=($signed(a))>>>b[4:0]”。
4. 对于获取外设显示数据，相当于 S 型指令，设计单周期 cpu 时，采用组合逻辑，而设计流水线 cpu 时，应采用时序逻辑。而读取拨码开关，相当于 LW 指令，单周期 cpu 和流水线 cpu 均采用组合逻辑。
5. 复位信号 rst 为特殊信号，在 if-else 等条件判断语句中应单独列为一种情形，与其它条件分开，比如我原先写的 if(rst | !flush)…应写成 if(rst)…else if(!flush)…
6. 针对数码管显示全 0 但仿真波形无误，我所遇到的原因和解决方法：单周期时时钟分频有误，应根据时钟 IP 核传入的 clk 时钟频率 25MHZ，经分频后数码管刷新频率 2ms 刷新一次来进行分频。流水线时原采用组合逻辑获取数码管要显示的数字，导致时许有误，应采用和 CPU 相同时钟的时序逻辑，在每个 clk_i 的上升沿写数码管和 led。
7. 在流水线 CPU 设计中，含有多个时序部件，尽量都采用同一个时钟沿。我原先流水级采用下降沿，跟新 pc、读寄存器堆、写内存等采用上升沿，仿真比较难看，但也算成功反应理想情况下结果正确，但板子可能就会反应不过来，这也可能是我数码管显示全零的原因之一。
8. 流水线 CPU 设计过程中，采用无脑停顿虽说比前递好想，但前者会降低 cpu 整体效率，而且实际上后者的代码并不比前者的复杂。

4 总结

要求：谈谈学完本课程后的个人收获以及对本课程的建议和意见。请在认真总结和思考后填写总结。

因为我有一位比我大一级的计科学长给我做了一个好榜样，他去年取得了这门课程的满分，同时我对这门课程兴趣浓厚，所以在考完试后我就开始认真着手于这门课程。

伊始到完成单周期计算器的下板，我花了近一周的时间，主要是刚开始我花了几天时间把指导书上提及的 MobaXterm、VS Code、虚拟机等下载到自己的电脑里，熟悉软件、使用远程平台等（我还是这类经验太欠缺，以前配置个环境、移动个软件目录都要花很长时间，捂脸.jpg）。单周期 cpu 主体的设计还是比较简单，但因为之前数逻知识的遗忘，下板的时候出现了很多问题，甚至因为降频失败导致数码管显示全零，这个问题还想了我很久。后面四五天的时间里设计流水线 CPU，我最大的困难和犯的错误出现在设计流水级等时序部件的时许上，还常常疑惑我这 trace 和仿真都没毛病哇，为啥下板就不行？现在我明白，不能总在理想情况下思考问题，也得为板子想想，捂脸.jpg。

我自觉对于《数字逻辑设计》和《计算机组成原理》的掌握并不全面深刻，还有很多遗忘，脑子也不灵活，所以算上我们班上课的半天时间，我几乎是把电脑搬到机房第一排放一整天。我见过的 T2210 上课的老师们都很好，不论是不是自己班里的学生，都很热心地帮我解决搓 cpu 过程中各式奇奇怪怪的问题，还老开玩笑说我又来白嫖老师 hh，谢谢老师和同学们~

流水线 180mhz（比我学长高 10mhz，嘿嘿）上板成功的时候，我在心里满满的充实感和成就感，通过这几天的学习，我也对《数字逻辑设计》和《计算机组成原理》的知识有了进一步的熟悉和掌握，收获满满。

我还有一点小小的建议是对指导书的修改，对于“单周期 CPU 时序”这一块，我采用“（3）将 RF 的读逻辑更改为组合逻辑”，结果失败。还有建议下板测试 trace 的.xci 文件和 IP 核的修改、更新、使用说明能再详细点，我之前在到 vscode 测试 trace 文件、跑仿真、下板 trace 测试、下板计算器反复折腾，脑子都混乱了，捂脸.jpg。