RISC-Z2 处理器基础任务实验报告（任务 4-5）

任务 4：实现执行操作及算术逻辑指令

一. 设计思路

1.目标：实现 ADD、SUB、OR、SLL 指令的执行逻辑，通过 ALU 完成算术和逻辑运算，并将结果写回寄存器堆。

2.关键实现：

（1）ALU 模块：

输入为操作数 aluA、aluB 和功能码 alufun，输出运算结果 valE。支持加法（alufun=0）、减法（alufun=1）、逻辑或（alufun=2）、逻辑左移（alufun=3）。

（2）操作数来源

ADD/SUB/OR：aluA 为 rA 寄存器值，aluB 为 rB 寄存器值。

SLL：aluA 为 rA 寄存器值，aluB 为立即数 valC 零扩展至 32 位。

（3）写回逻辑

ALU 结果 valE 通过 dstE 写入目标寄存器 rd。

3.指令编码对应

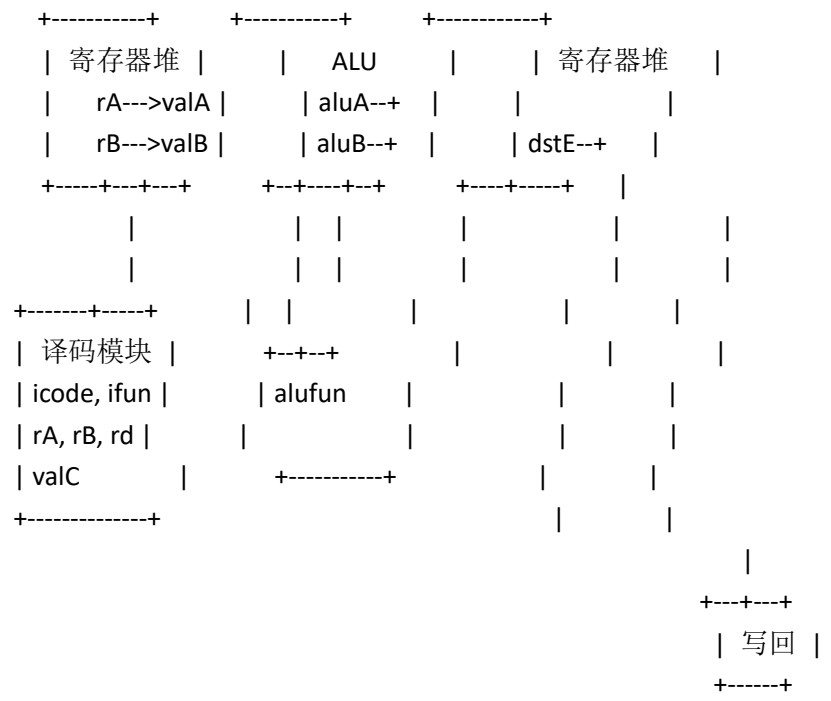
（1）ADD（icode=2, ifun=0）→ alufun=0

（2）SUB（icode=2, ifun=1）→ alufun=1

（3）OR（icode=3, ifun=0）→ alufun=2

（4）SLL（icode=3, ifun=1）→ alufun=3，aluB=valC[14:0]

二. 数据通路与控制通路框图



控制信号：alufun 由 icode 和 ifun 译码生成，dstE 为指令中的 rd 字段。

三. 验证材料

1. Testbench 代码说明

功能：写入 8 条指令，验证算术逻辑指令执行结果。

关键指令（学号 0xCAF6）

```
// 阶段 1：写入指令（working=0, wEn=1）
addr=0; wDat=32'b0001_0000_000_000_000_1100101011110110; //
IRMOV r0, 0xCAF6
addr=1; wDat=32'b0001_0000_000_000_001_1100101011110111; //
```

IRMOV r1, 0xCAF7

addr=2; wDat=32'b0001_0000_000_000_010_1100101011111000; //

IRMOV r2, 0xCAF8

addr=3; wDat=32'b0001_0000_000_000_011_1100101011111001; //

IRMOV r3, 0xCAF9

addr=4; wDat=32'b0010_0000_001_010_100_000000000000000; //

ADD r4, r1, r2

addr=5; wDat=32'b0010_0001_000_011_101_000000000000000; //

SUB r5, r0, r3

addr=6; wDat=32'b0011_0000_001_000_110_000000000000000; //

OR r6, r1, r0

addr=7; wDat=32'b0011_0001_010_000_111_000000000000011; //

SLL r7, r2, 3

验证点：

（1）寄存器 r0-r3 应分别加载 0xCAF6、0xCAF7、0xCAF8、0xCAF9。

（2）r4 = r1 + r2 = 0xCAF7 + 0xCAF8 = 0x195EF。

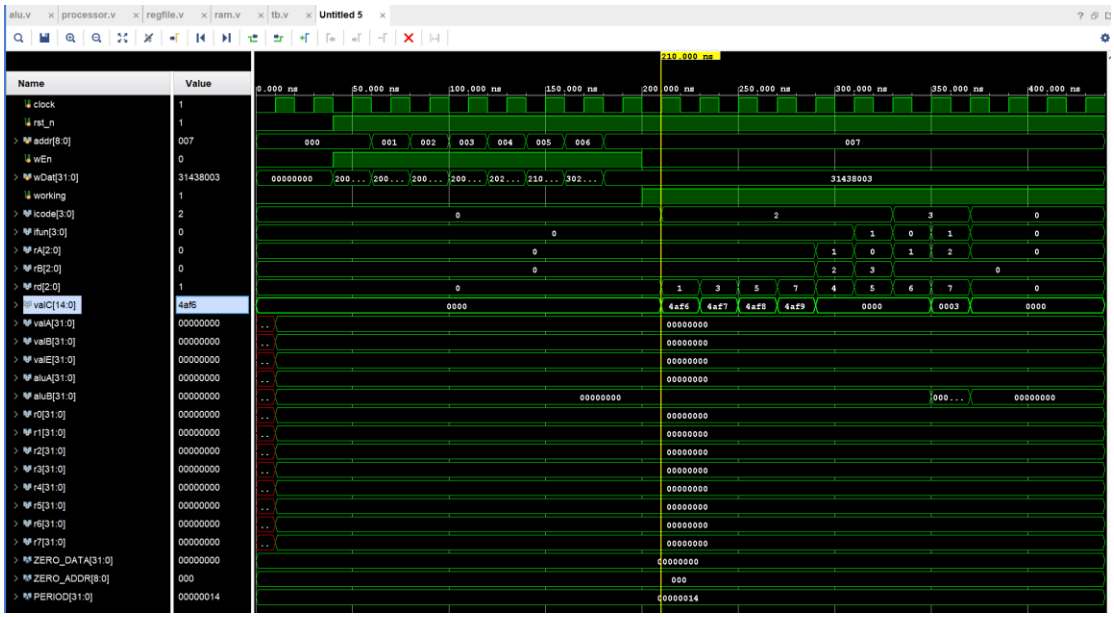
（3）r5 = r0 - r3 = 0xCAF6 - 0xCAF9 = 0xFFFFFFF3（补码表示 -3）。

（4）r6 = r1 | r0 = 0xCAF7。

（5）r7 = r2 << 3 = 0xCAF8 << 3 = 0x65FC0。

2.仿真结果截图与分析

（1）仿真结果



（2）信号说明：

r0-r7：寄存器值变化，验证写入结果。

valE：ALU 输出，观察 ADD 指令执行时 valE=0x195EF。

（3）结果分析

IRMOV 指令正确加载立即数至寄存器。

算术逻辑指令（ADD/SUB/OR/SLL）的运算结果与预期一致，验证 ALU 功能正确。

任务 5：四级流水线处理器的实现

一. 设计思路

1.流水线划分

取指（IF）：从存储器读指令，更新 PC（PC+1）。

译码（ID）：解析指令字段，读取寄存器 valA、valB。

执行（EX）：ALU 运算，生成 valE。

写回（WB）：将结果写入寄存器堆。

2.数据冒险处理

（1）RAW 冒险：当前指令的源寄存器为前一条指令的目标寄存器时，通过 ** 前递（Forwarding）** 机制，将 EX 阶段的 valE 直接传递至 ID 阶段的输入，避免流水线停顿。

（2）冒险检测：在 ID 阶段检查 rA/rB 是否为 EX 阶段的 dstE，若是则使用前递值。

3.流水线寄存器

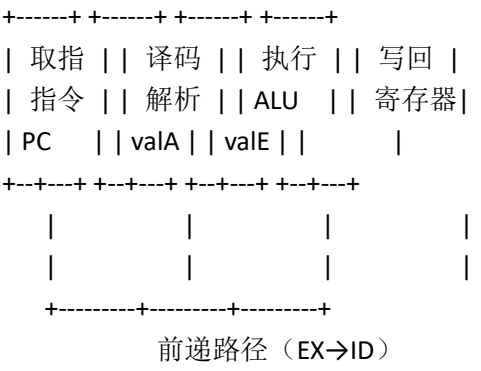
IF/ID：存储取指结果（指令、PC）。

ID/EX：存储译码结果（icode, ifun, valA, valB, valC, rd）。

EX/WB：存储执行结果（valE, rd）。

二. 数据通路与控制通路框图

IF 阶段　ID 阶段　EX 阶段　WB 阶段



前递逻辑：在 ID 阶段，若 rA == EX/WB.rd，则 valA 取 EX/WB.valE，而非寄存器堆值。

三. 验证材料

1.Testbench 代码说明

功能：写入 8 条指令，验证流水线执行和数据冒险处理。

关键指令（学号 0xCAF6）：

// 指令 4: ADD %r3, %r1, %r2 （r3 为目标寄存器，后续指令 5 使用 r3）

addr=4; wDat=32'b0010_0000_001_010_011_000000000000000;

// 指令 5: SUB %r6, %r0, %r3 （存在 RAW 冒险：r3 是指令 4 的结果）

addr=5; wDat=32'b0010_0001_000_011_110_000000000000000;

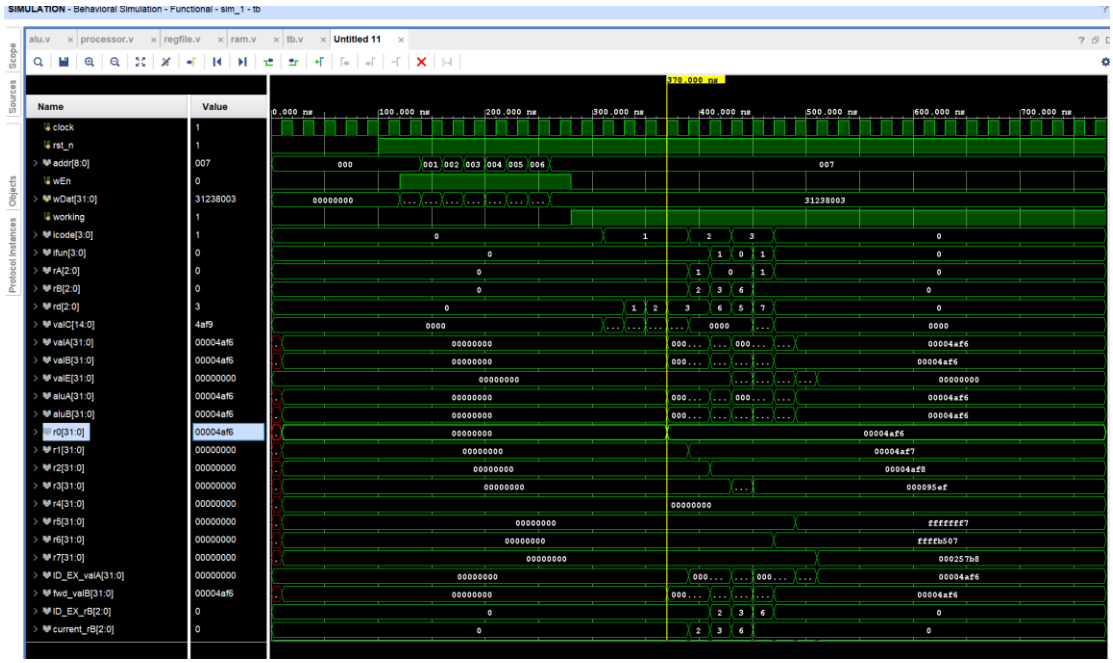
验证点

（1）流水线各阶段信号是否按周期推进。

（2）指令 5 是否通过前递机制获取指令 4 的 valE，避免停顿。

2.仿真结果截图与分析

（1）仿真结果截图



信号说明：

PC：依次递增，显示流水线取指顺序。

EX/WB.valE：指令 4 的 ALU 结果 0x195EF 在前递路径中传递至指令 5 的 ID 阶段。

结果分析：

指令按流水线阶段并行执行，无停顿，验证流水线结构正确。

指令 5 成功通过前递获取指令 4 的结果，RAW 冒险被消除，寄存器 r6 值为 0xCAF6 - 0x195EF = 0xFF7A0D07。

3.RAW 冒险识别

存在冒险的指令对：

指令 4（ADD %r3, %r1, %r2）与指令 5（SUB %r6, %r0, %r3）：

指令 5 的源寄存器 r3 是指令 4 的目标寄存器，通过前递解决冒险。

指令 5（SUB %r6, %r0, %r3）与指令 6（OR %r5, %r0, %r6）：

指令 6 的源寄存器 r6 是指令 5 的目标寄存器，需前递 valE。

四. 结果总结

1.任务 4：成功实现 ALU 和算术逻辑指令，寄存器结果与预期一致，验证了运算逻辑的正确性。

2.任务 5：四级流水线正确执行指令，通过前递机制解决数据冒险，流水线效率提升，无停顿现象。