```java
/*
 * CSC252 - Sim 3
 * File: Sim3_ALUElement.java
 * Author: Rithvik
 * Purpose: 1-bit ALU element with combinational logic operations.
 */

public class Sim3_ALUElement {
    // ALU operation bits (3 bits selecting the operation)
    public RussWire[] aluOp;
    // Indicates whether b input should be inverted (for subtraction)
    public RussWire bInvert;
    // Data inputs
    public RussWire a, b;
    // Carry input for addition
    public RussWire carryIn;

    // Output of this ALU element
    public RussWire result;
    // Result from addition stage (sum)
    public RussWire addResult;
    // Carry out from addition stage
    public RussWire carryOut;

    // Less input (for set-less-than)
    public RussWire less;

    // Internal wires for adjusted inputs and intermediate sums/carries
    private RussWire bAdjusted;
    private RussWire sum;
    private RussWire carry;

    /** Constructor initializes wires */
    public Sim3_ALUElement() {
        aluOp = new RussWire[3];
        for (int i = 0; i < 3; i++)
            aluOp[i] = new RussWire();

        bInvert = new RussWire();
```

```
        a = new RussWire();
        b = new RussWire();
        carryIn = new RussWire();

        result = new RussWire();
        addResult = new RussWire();
        carryOut = new RussWire();

        less = new RussWire();

        bAdjusted = new RussWire();
        sum = new RussWire();
        carry = new RussWire();
    }

    /**
     * First stage: Compute adjusted b input considering bInvert,
     * then calculate sum and carry for addition.
     */
    public void execute_pass1() {
        boolean bin = bInvert.get();
        // Invert b if bInvert=1 (for subtraction)
        bAdjusted.set(b.get() ^ bin);

        boolean aVal = a.get();
        boolean bVal = bAdjusted.get();
        boolean cin = carryIn.get();

        // Full adder sum and carry calculations
        boolean s = (aVal ^ bVal) ^ cin;
        boolean c = (aVal && bVal) || ((aVal ^ bVal) && cin);

        sum.set(s);
        carry.set(c);

        addResult.set(s);
        carryOut.set(c);
    }

    /**
```

```
     * Second stage: Determine final ALU output by selecting
     * between AND, OR, ADD, LESS, XOR based on aluOp signals.
     */
    public void execute_pass2() {
        boolean op0 = aluOp[0].get();
        boolean op1 = aluOp[1].get();
        boolean op2 = aluOp[2].get();

        boolean andRes = a.get() && bAdjusted.get();
        boolean orRes = a.get() || bAdjusted.get();
        boolean xorRes = a.get() ^ bAdjusted.get();

        boolean lessVal = less.get();

        // Decode aluOp bits into operations
        boolean isAnd = (!op2 && !op1 && !op0);
        boolean isOr = (!op2 && !op1 && op0);
        boolean isAdd = (!op2 && op1 && !op0);
        boolean isLess = (!op2 && op1 && op0);
        boolean isXor = (op2 && !op1 && !op0);

        // Select result based on operation
        boolean res = (isAnd && andRes) ||
                      (isOr && orRes) ||
                      (isAdd && addResult.get()) ||
                      (isLess && lessVal) ||
                      (isXor && xorRes);

        result.set(res);
    }
}
```