```
`timescale 1ns / 1ps

module fpu_64bit (
    input  wire        clk,
    input  wire        rst,
    input  wire        start,
    input  wire [1:0]  op,         // 00: add, 01: sub, 10: mul, 11: div
    input  wire [63:0] a,
    input  wire [63:0] b,
    output wire [63:0] result,
    output reg         ready
);

    // Internal signals from unpack
    wire        sign_a, sign_b;
    wire [10:0] exp_a, exp_b;
    wire [52:0] mant_a, mant_b;
    wire        is_nan_a, is_nan_b, is_inf_a, is_inf_b, is_zero_a, is_zero_b;

    // Unpack operands
    fpu_unpack unpack_a (
        .in(a),
        .sign(sign_a),
        .exponent(exp_a),
        .mantissa(mant_a),
        .is_nan(is_nan_a),
        .is_inf(is_inf_a),
        .is_zero(is_zero_a)
    );

    fpu_unpack unpack_b (
        .in(b),
        .sign(sign_b),
        .exponent(exp_b),
        .mantissa(mant_b),
        .is_nan(is_nan_b),
        .is_inf(is_inf_b),
        .is_zero(is_zero_b)
    );

    // Determine global exceptions
    wire is_nan  = is_nan_a | is_nan_b;
    wire is_zero = is_zero_a & is_zero_b;
    wire is_inf  = (op != 2'b11) ? (is_inf_a | is_inf_b) : 1'b0;

    // Outputs from submodules
```

```
wire        addsub_sign, mul_sign, div_sign;
wire [10:0] addsub_exp,  mul_exp,  div_exp;
wire [52:0] addsub_mant, mul_mant, div_mant;
wire        addsub_ready, mul_ready, div_ready;

// ADD/SUB
fpu_addsub addsub (
    .clk(clk),
    .rst(rst),
    .op(op[0]), // 0: add, 1: sub
    .sign_a(sign_a),
    .exp_a(exp_a),
    .mant_a(mant_a),
    .sign_b(sign_b),
    .exp_b(exp_b),
    .mant_b(mant_b),
    .result_sign(addsub_sign),
    .result_exp(addsub_exp),
    .result_mant(addsub_mant),
    .ready(addsub_ready)
);

// MUL
fpu_mul mul (
    .clk(clk),
    .rst(rst),
    .sign_a(sign_a),
    .exp_a(exp_a),
    .mant_a(mant_a),
    .sign_b(sign_b),
    .exp_b(exp_b),
    .mant_b(mant_b),
    .result_sign(mul_sign),
    .result_exp(mul_exp),
    .result_mant(mul_mant),
    .ready(mul_ready)
);

// DIV
fpu_div div (
    .clk(clk),
    .rst(rst),
    .sign_a(sign_a),
    .exp_a(exp_a),
    .mant_a(mant_a),
    .sign_b(sign_b),
```

```
    .exp_b(exp_b),
    .mant_b(mant_b),
    .result_sign(div_sign),
    .result_exp(div_exp),
    .result_mant(div_mant),
    .ready(div_ready)
);

// Output selection
reg        final_sign;
reg [10:0] final_exp;
reg [52:0] final_mant;

always @(*) begin
    case (op)
        2'b00, 2'b01: begin
            final_sign = addsub_sign;
            final_exp  = addsub_exp;
            final_mant = addsub_mant;
            ready      = addsub_ready;
        end
        2'b10: begin
            final_sign = mul_sign;
            final_exp  = mul_exp;
            final_mant = mul_mant;
            ready      = mul_ready;
        end
        2'b11: begin
            final_sign = div_sign;
            final_exp  = div_exp;
            final_mant = div_mant;
            ready      = div_ready;
        end
        default: begin
            final_sign = 1'b0;
            final_exp  = 11'd0;
            final_mant = 53'd0;
            ready      = 1'b0;
        end
    endcase
end

// Final pack
fpu_pack pack (
    .sign(final_sign),
    .exponent(final_exp),
```

```
        .mantissa(final_mant),
        .is_zero(is_zero),
        .is_inf(is_inf),
        .is_nan(is_nan),
        .out(result)
    );

endmodule
```