float_add.v

```
module float_add (
  input [7:0] aIn,
  input [7:0] bIn,
  output reg [7:0] result
);

wire [7:0] big_aOut, big_bOut;

big_number_first step1 (
  .aIn(aIn),
  .bIn(bIn),
  .aOut(big_aOut),
  .bOut(big_bOut)
);

wire [2:0] aExp;
wire [4:0] aMan;
wire [2:0] bExp;
wire [4:0] bMan;

assign aExp = big_aOut[7:5];
assign aMan = big_aOut[4:0];
assign bExp = big_bOut[7:5];
assign bMan = big_bOut[4:0];

wire [2:0] distance1;
assign distance1 = aExp - bExp;

wire [2:0] bExpShift;
wire [4:0] bManShift;

assign bExpShift = bExp +
distance1;

shifter step2 (
  .in(bMan),
  .distance(distance1),
  .direction(1'b1),
  .out(bManShift)
);

wire [4:0] adderSum;
wire adderCout;

adder step3 (
  .A(aMan),
  .B(bManShift),
  .S(adderSum),
  .Cout(adderCout)
);

wire [2:0] distance2;
assign distance2 = {2'b00,
adderCout};

wire [2:0] resultExp;
wire [4:0] resultMan;

assign resultExp = bExpShift +
adderCout;

shifter step4 (
  .in(adderSum),
  .distance(distance2),
  .direction(1'b1),
  .out(resultMan)
);

reg [4:0] resultManShift;
reg [2:0] distance3;
wire [4:0] resultManShift2;

reg [2:0] numToShift1;

wire [2:0] resultExpShift;
assign resultExpShift =
resultExp - distance3;

shifter step5(
  .in(resultManShift),
  .distance(distance3),
  .direction(1'b0),
  .out(resultManShift2)
);
```

float_add.v (continued)

```
always @* begin
  case(adderCout)
    1'b0: resultManShift =
resultMan;
    1'b1: resultManShift =
{1'b1, resultMan[3:0]};
  endcase

  if(bExpShift == 3'b111 &&
adderCout == 1'b1)
    // Saturation case
    result = 8'b11111111;
  else if(resultManShift ==
5'b00000)
    result = 8'b00000000;
  else
    // Handling the carry bit
from the adder
    begin
      if(resultExp != 3'b000 &&
resultManShift[4] == 1'b0)
        begin
          // Normalizing -
determining how many bits to
shift mantissa
          casex(resultManShift)
            5'b00001:
numToShift1 = 3'b100;
            5'b0001x:
numToShift1 = 3'b011;
            5'b001xx:
numToShift1 = 3'b010;
            5'b01xxx:
numToShift1 = 3'b001;
            default: numToShift1
= 3'b000;
          endcase
        end
      else
        numToShift1 = 3'b000;

      // Normalizing -
determining how many bits to
shift mantissa
      if(resultExp <
numToShift1)
        distance3 = resultExp;
      else if (resultExp >=
numToShift1)
        distance3 = numToShift1;
      else
        distance3 = 3'b000;

      result = {resultExpShift,
resultManShift2};
    end
end

endmodule
```

big_number_first.v

```
module big_number_first (
  input [7:0] aIn,
  input [7:0] bIn,
  output reg [7:0] aOut,
  output reg [7:0] bOut
);

wire [2:0] aExp;
wire [4:0] aMan;
wire [2:0] bExp;
wire [4:0] bMan;

assign aExp = aIn[7:5];
assign aMan = aIn[4:0];
assign bExp = bIn[7:5];
assign bMan = bIn[4:0];

always @* begin
  if((aExp < bExp) || (aExp ==
bExp && aMan < bMan))
    begin
      aOut = {bExp, bMan};
      bOut = {aExp, aMan};
    end
  else
    begin
      aOut = {aExp, aMan};
      bOut = {bExp, bMan};
    end
end

endmodule
```

shifter.v

```
module shifter (
  input [4:0] in,
  input [2:0] distance,
  input direction,
  output reg [4:0] out
);

always @* begin
  case(direction)
    1'b0: out = in << distance;
    1'b1: out = in >> distance;
  endcase
end

endmodule
```

adder.v

```
module adder(
  input [4:0] A,
  input [4:0] B,
  output [4:0] S,
  output Cout
);

wire Cin = 1'b0;
wire Cout0;
wire Cout1;
wire Cout2;
wire Cout3;

fullAdder a0 (
  .A(A[0]),
  .B(B[0]),
  .Cin(Cin),
  .Cout(Cout0),
  .S(S[0])
);

fullAdder a1 (
  .A(A[1]),
  .B(B[1]),
  .Cin(Cout0),
  .Cout(Cout1),
  .S(S[1])
);

fullAdder a2 (
  .A(A[2]),
  .B(B[2]),
  .Cin(Cout1),
  .Cout(Cout2),
  .S(S[2])
);

fullAdder a3 (
  .A(A[3]),
  .B(B[3]),
  .Cin(Cout2),
  .Cout(Cout3),
  .S(S[3])
);

fullAdder a4 (
  .A(A[4]),
  .B(B[4]),
  .Cin(Cout3),
  .Cout(Cout),
  .S(S[4])
);

endmodule
```

---

fullAdder.v

```
module fullAdder (
  input A,
  input B,
  input Cin,
  output S,
  output Cout
);

wire AxB;
assign AxB = A ^ B;

wire AxBaCin;
assign AxBaCin = (A ^ B) & Cin;

wire AaB;
assign AaB = A & B;

assign Cout = AaB | AxBaCin;
assign S = AxB ^ Cin;

endmodule
```

float_add_tb.v

```
module float_add_tb ();

reg [7:0] sim_aIn;
reg [7:0] sim_bIn;
wire [7:0] sim_result;

float_add test(
  .aIn(sim_aIn),
  .bIn(sim_bIn),
  .result(sim_result)
);

initial begin
  sim_aIn = 8'b00000000;
  sim_bIn = 8'b00000000;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00000000);

  sim_aIn = 8'b00001000;
  sim_bIn = 8'b00000011;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00001011);

  sim_aIn = 8'b00110001;
  sim_bIn = 8'b00001100;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00110111);

  sim_aIn = 8'b10010010;
  sim_bIn = 8'b01011111;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b10011001);

  sim_aIn = 8'b00011110;
  sim_bIn = 8'b00011000;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00111011);

  sim_aIn = 8'b11111110;
  sim_bIn = 8'b11111000;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b11111111);

  sim_aIn = 8'b11111111;
  sim_bIn = 8'b00111111;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b11111111);

  sim_aIn = 8'b11111110;
  sim_bIn = 8'b00111111;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b11111110);

  sim_aIn = 8'b11100000;
  sim_bIn = 8'b11100000;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00000000);

  sim_aIn = 8'b00100000;
  sim_bIn = 8'b00000100;
  #5
  $display("Output is %b, we
expected %b", sim_result,
8'b00000100);

  $stop;
end
endmodule
```

big_number_first_tb.v

```
module big_number_first_tb ();

reg [7:0] sim_aIn, sim_bIn;
wire [7:0] sim_aOut, sim_bOut;

big_number_first test (
  .aIn(sim_aIn),
  .bIn(sim_bIn),
  .aOut(sim_aOut),
  .bOut(sim_bOut)
);

initial begin
  sim_aIn = 8'b00001000;
  sim_bIn = 8'b00000011;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_aIn,
sim_bIn);

  sim_aIn = 8'b00000011;
  sim_bIn = 8'b00001000;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_bIn,
sim_aIn);

  sim_aIn = 8'b00110001;
  sim_bIn = 8'b00001100;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_aIn,
sim_bIn);

  sim_aIn = 8'b00001100;
  sim_bIn = 8'b00110001;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_bIn,
sim_aIn);

  sim_aIn = 8'b11111110;
  sim_bIn = 8'b11111111;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_bIn,
sim_aIn);

  sim_aIn = 8'b11111111;
  sim_bIn = 8'b11111110;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_aIn,
sim_bIn);

  sim_aIn = 8'b11011111;
  sim_bIn = 8'b11111111;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_bIn,
sim_aIn);

  sim_aIn = 8'b11111111;
  sim_bIn = 8'b11011111;
  #5
  $display("Output is %b and %b,
we expected %b and %b",
sim_aOut, sim_bOut, sim_aIn,
sim_bIn);

  $stop;
end

endmodule
```

shifter_tb.v

```
module shifter_tb ();

reg [4:0] sim_in;
reg [2:0] sim_distance;
reg sim_direction;
wire [4:0] sim_out;

shifter test (
  .in(sim_in),
  .distance(sim_distance),
  .direction(sim_direction),
  .out(sim_out)
);

initial begin
  sim_in = 5'b11001;
  sim_distance = 3'b000;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b11001);

  sim_in = 5'b11001;
  sim_distance = 3'b001;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b10010);

  sim_in = 5'b11001;
  sim_distance = 3'b010;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00100);

  sim_in = 5'b11001;
  sim_distance = 3'b011;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b01000);

  sim_in = 5'b11001;
  sim_distance = 3'b100;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b10000);

  sim_in = 5'b11001;
  sim_distance = 3'b101;
  sim_direction = 1'b0;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00000);

  sim_in = 5'b11001;
  sim_distance = 3'b000;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b11001);

  sim_in = 5'b11001;
  sim_distance = 3'b001;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b01100);

  sim_in = 5'b11001;
  sim_distance = 3'b010;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00110);
```

shifter_tb.v (continued)

```
  sim_in = 5'b11001;
  sim_distance = 3'b011;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00011);

  sim_in = 5'b11001;
  sim_distance = 3'b100;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00001);

  sim_in = 5'b11001;
  sim_distance = 3'b101;
  sim_direction = 1'b1;
  #5
  $display("Output is %b, we
expected %b", sim_out,
5'b00000);

  $stop;
end

endmodule
```

---

adder_tb.v

```
module adder_tb ();

reg [4:0] sim_A;
reg [4:0] sim_B;

wire [4:0] sim_S;
wire sim_Cout;

adder test (
  .A(sim_A),
  .B(sim_B),
  .Cout(sim_Cout),
  .S(sim_S)
);

initial begin
  sim_A = 5'b0;
  sim_B = 5'b0;

  #5;

  $display("Sum is %b, we
expected %b", sim_S, 5'b0);
  $display("Cout is %b, we
expected %b", sim_Cout, 1'b0);

  sim_A = 5'b00001;
  sim_B = 5'b00010;

  #5;

  $display("Sum is %b, we
expected %b", sim_S, 5'b00011);
  $display("Cout is %b, we
expected %b", sim_Cout, 1'b0);

  sim_A = 5'b00111;
  sim_B = 5'b00111;

  #5;

  $display("Sum is %b, we
expected %b", sim_S, 5'b01110);
  $display("Cout is %b, we
expected %b", sim_Cout, 1'b0);

  sim_A = 5'b10000;
  sim_B = 5'b11000;

  #5;

  $display("Sum is %b, we
expected %b", sim_S, 5'b01000);
  $display("Cout is %b, we
expected %b", sim_Cout, 1'b1);

  $stop;
end

endmodule
```