2023

Graduation Project Documentation

RVF FLOATING POINT EXTENSION

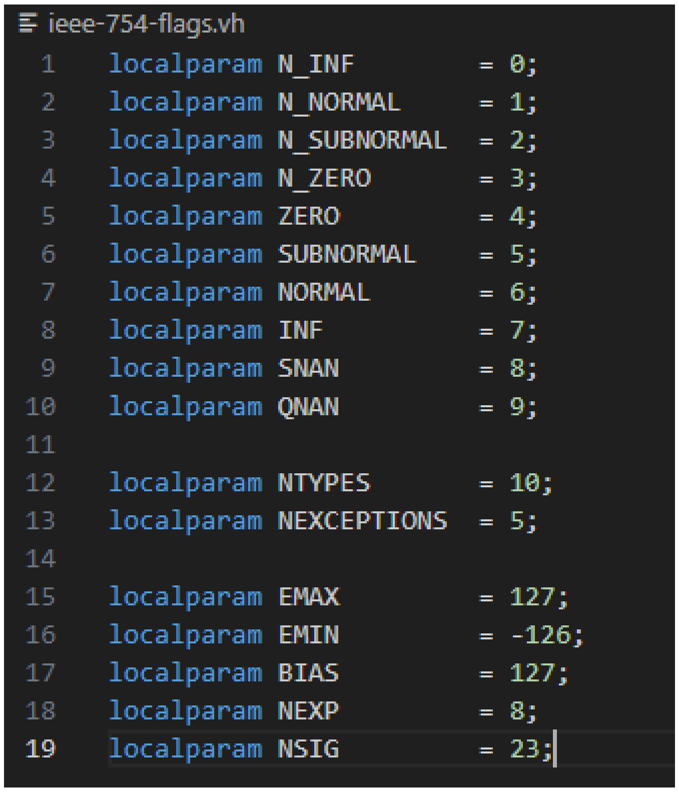
ABDEL RAHMAN ATEF YOUSSEF

9/12/2023

Contents

Codes:

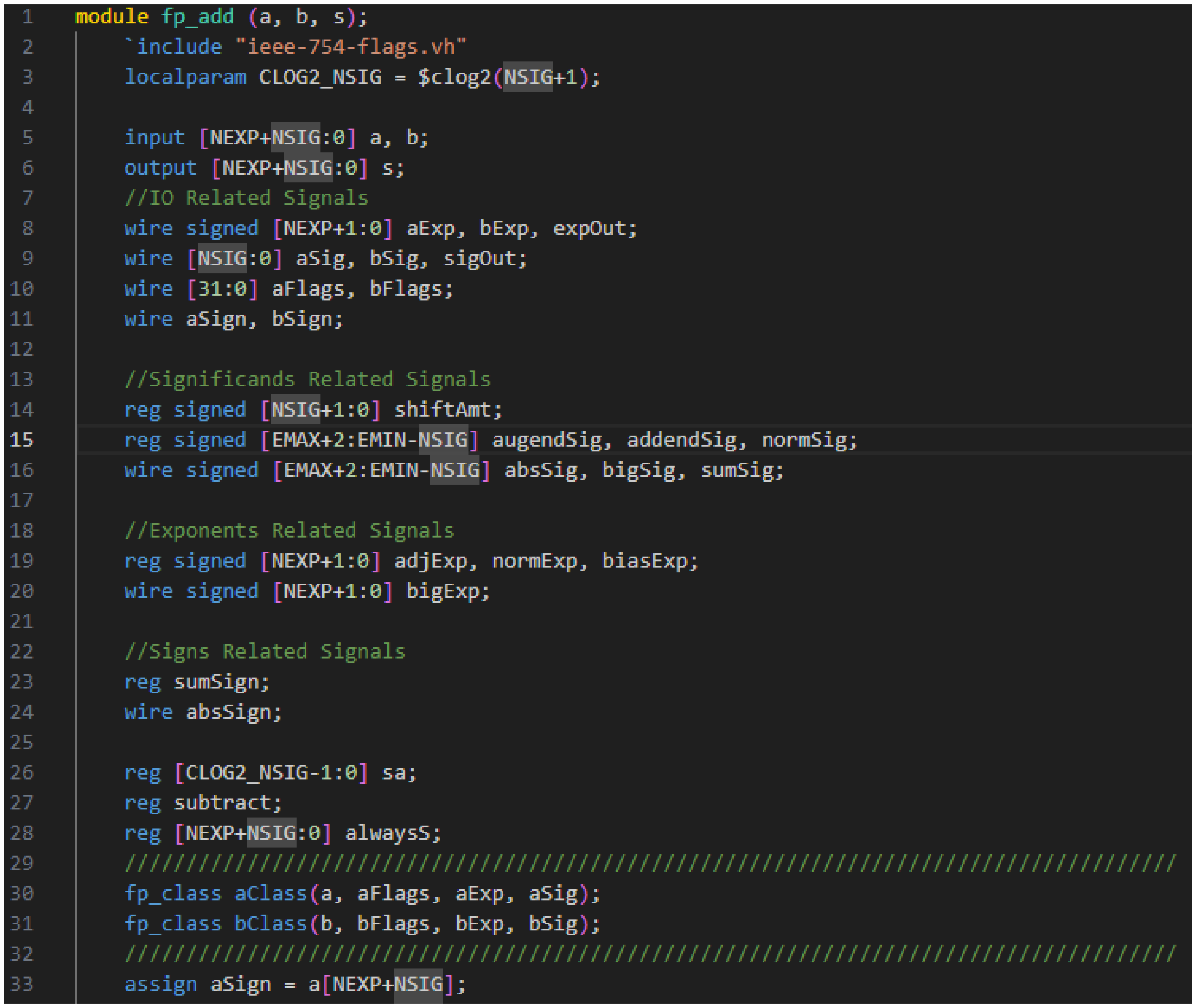
1)ieee-754-flags:



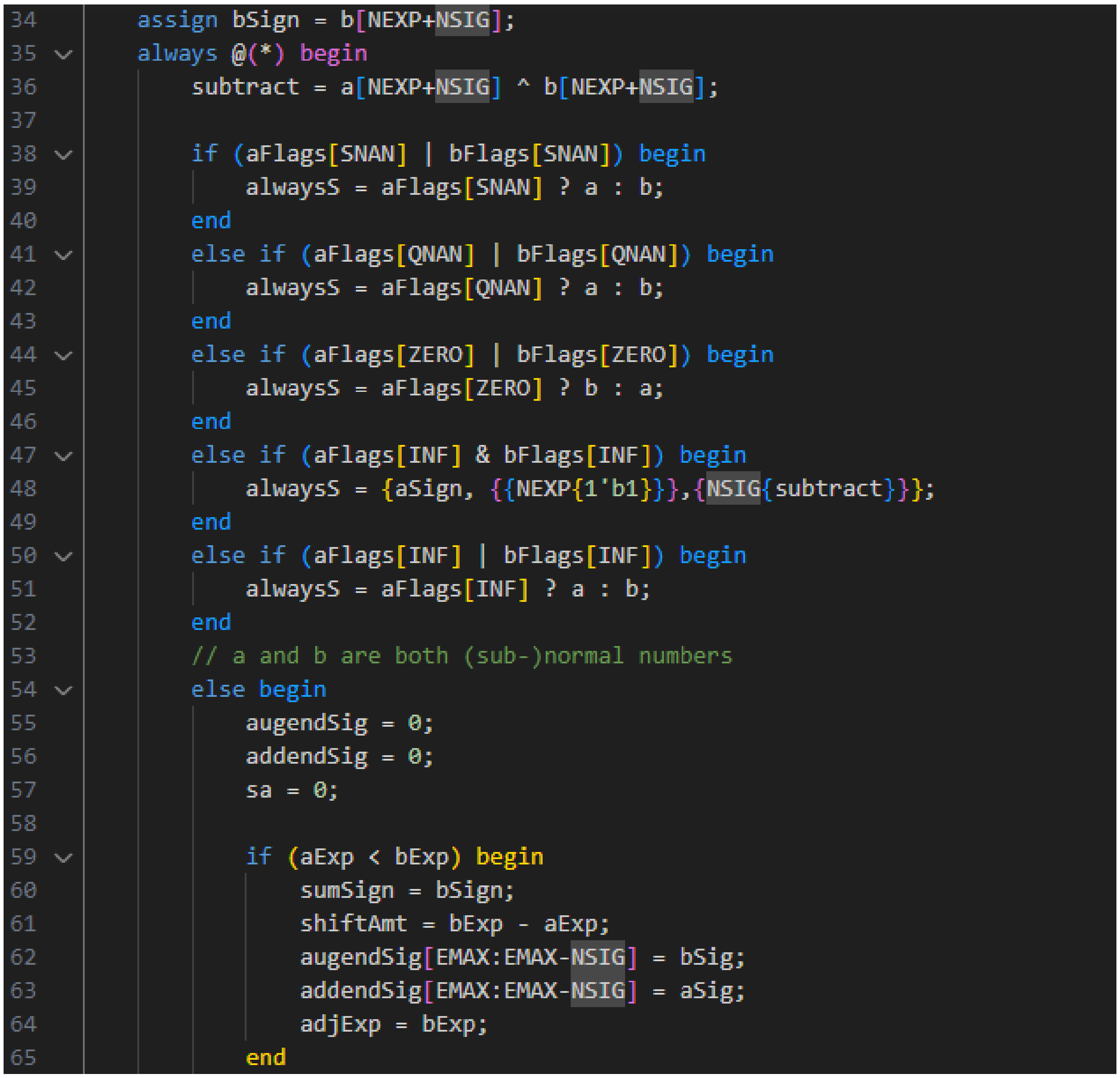
ieee-754-flags.vh

```
localparam N_INF        = 0;
localparam N_NORMAL     = 1;
localparam N_SUBNORMAL  = 2;
localparam N_ZERO       = 3;
localparam ZERO         = 4;
localparam SUBNORMAL    = 5;
localparam NORMAL       = 6;
localparam INF          = 7;
localparam SNAN         = 8;
localparam QNAN         = 9;

localparam NTYPES       = 10;
localparam NEXCEPTIONS  = 5;

localparam EMAX         = 127;
localparam EMIN         = -126;
localparam BIAS         = 127;
localparam NEXP         = 8;
localparam NSIG         = 23;
```

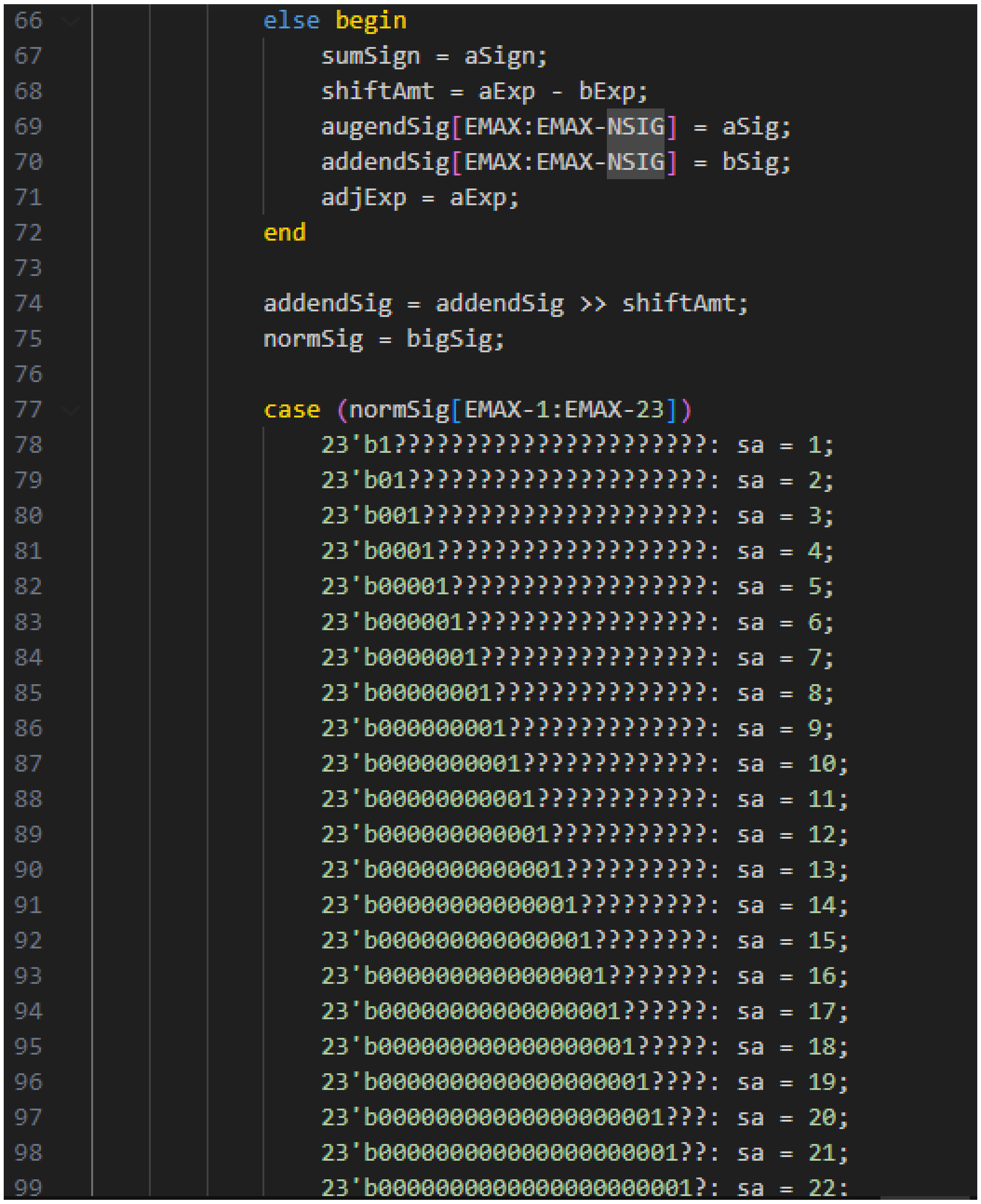
2) fp_add:



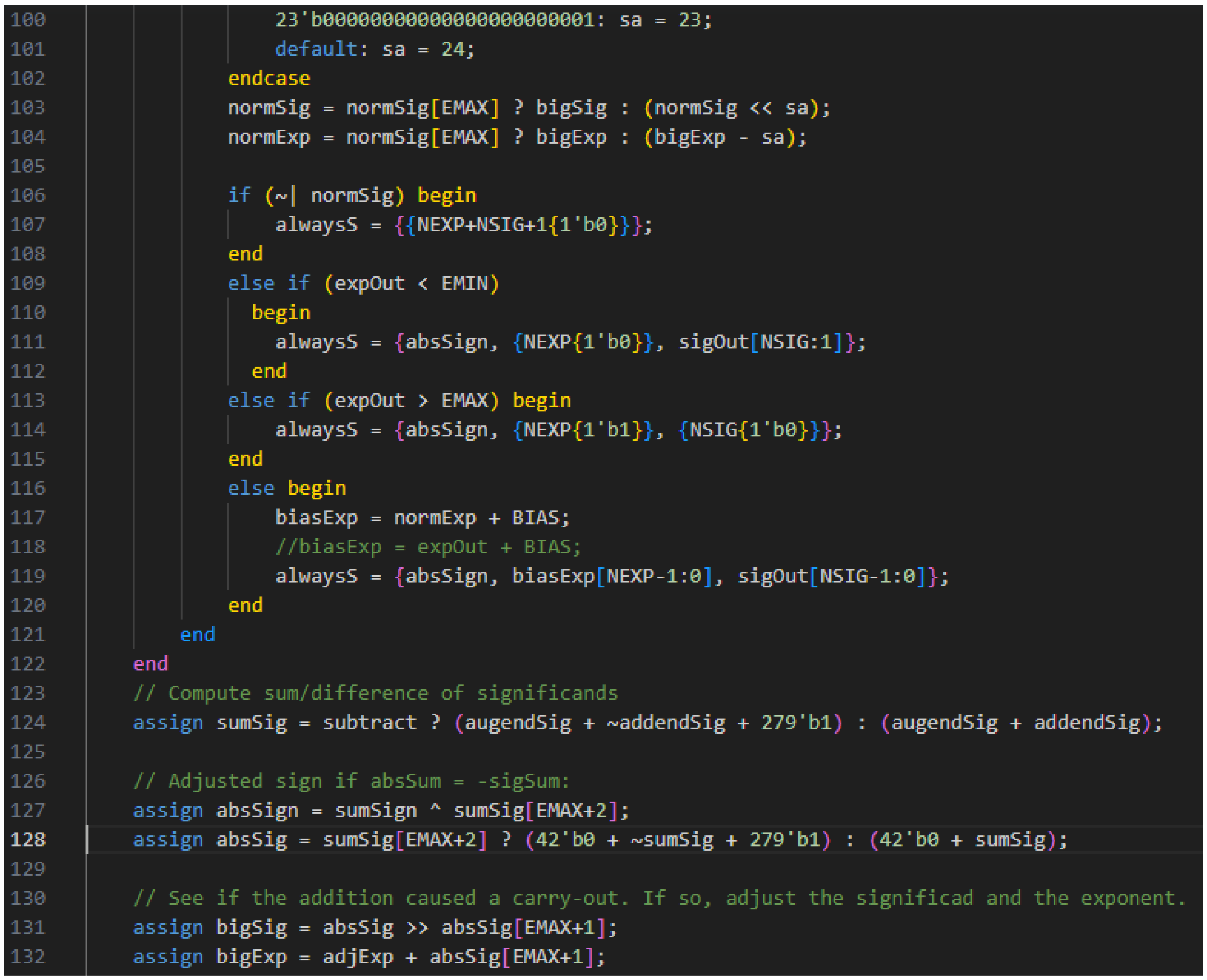
```verilog
module fp_add (a, b, s);
    `include "ieee-754-flags.vh"
    localparam CLOG2_NSIG = $clog2(NSIG+1);

    input [NEXP+NSIG:0] a, b;
    output [NEXP+NSIG:0] s;
    //IO Related Signals
    wire signed [NEXP+1:0] aExp, bExp, expOut;
    wire [NSIG:0] aSig, bSig, sigOut;
    wire [31:0] aFlags, bFlags;
    wire aSign, bSign;

    //Significands Related Signals
    reg signed [NSIG+1:0] shiftAmt;
    reg signed [EMAX+2:EMIN-NSIG] augendSig, addendSig, normSig;
    wire signed [EMAX+2:EMIN-NSIG] absSig, bigSig, sumSig;

    //Exponents Related Signals
    reg signed [NEXP+1:0] adjExp, normExp, biasExp;
    wire signed [NEXP+1:0] bigExp;

    //Signs Related Signals
    reg sumSign;
    wire absSign;

    reg [CLOG2_NSIG-1:0] sa;
    reg subtract;
    reg [NEXP+NSIG:0] alwaysS;
    ////////////////////////////////////////////////////////////////////////////////////////
    fp_class aClass(a, aFlags, aExp, aSig);
    fp_class bClass(b, bFlags, bExp, bSig);
    ////////////////////////////////////////////////////////////////////////////////////////
    assign aSign = a[NEXP+NSIG];
```



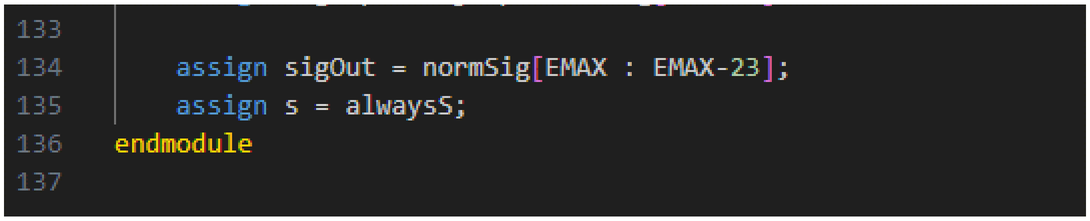
```
    assign bSign = b[NEXP+NSIG];
    always @(*) begin
        subtract = a[NEXP+NSIG] ^ b[NEXP+NSIG];

        if (aFlags[SNAN] | bFlags[SNAN]) begin
            alwaysS = aFlags[SNAN] ? a : b;
        end
        else if (aFlags[QNAN] | bFlags[QNAN]) begin
            alwaysS = aFlags[QNAN] ? a : b;
        end
        else if (aFlags[ZERO] | bFlags[ZERO]) begin
            alwaysS = aFlags[ZERO] ? b : a;
        end
        else if (aFlags[INF] & bFlags[INF]) begin
            alwaysS = {aSign, {{NEXP{1'b1}}},{NSIG{subtract}}};
        end
        else if (aFlags[INF] | bFlags[INF]) begin
            alwaysS = aFlags[INF] ? a : b;
        end
        // a and b are both (sub-)normal numbers
        else begin
            augendSig = 0;
            addendSig = 0;
            sa = 0;

            if (aExp < bExp) begin
                sumSign = bSign;
                shiftAmt = bExp - aExp;
                augendSig[EMAX:EMAX-NSIG] = bSig;
                addendSig[EMAX:EMAX-NSIG] = aSig;
                adjExp = bExp;
            end
```



```
            else begin
                sumSign = aSign;
                shiftAmt = aExp - bExp;
                augendSig[EMAX:EMAX-NSIG] = aSig;
                addendSig[EMAX:EMAX-NSIG] = bSig;
                adjExp = aExp;
            end

            addendSig = addendSig >> shiftAmt;
            normSig = bigSig;

            case (normSig[EMAX-1:EMAX-23])
                23'b1??????????????????????: sa = 1;
                23'b01?????????????????????: sa = 2;
                23'b001????????????????????: sa = 3;
                23'b0001???????????????????: sa = 4;
                23'b00001??????????????????: sa = 5;
                23'b000001?????????????????: sa = 6;
                23'b0000001????????????????: sa = 7;
                23'b00000001???????????????: sa = 8;
                23'b000000001??????????????: sa = 9;
                23'b0000000001?????????????: sa = 10;
                23'b00000000001????????????: sa = 11;
                23'b000000000001???????????: sa = 12;
                23'b0000000000001??????????: sa = 13;
                23'b00000000000001?????????: sa = 14;
                23'b000000000000001????????: sa = 15;
                23'b0000000000000001???????: sa = 16;
                23'b00000000000000001??????: sa = 17;
                23'b000000000000000001?????: sa = 18;
                23'b0000000000000000001????: sa = 19;
                23'b00000000000000000001???: sa = 20;
                23'b000000000000000000001??: sa = 21;
                23'b0000000000000000000001?: sa = 22;
```

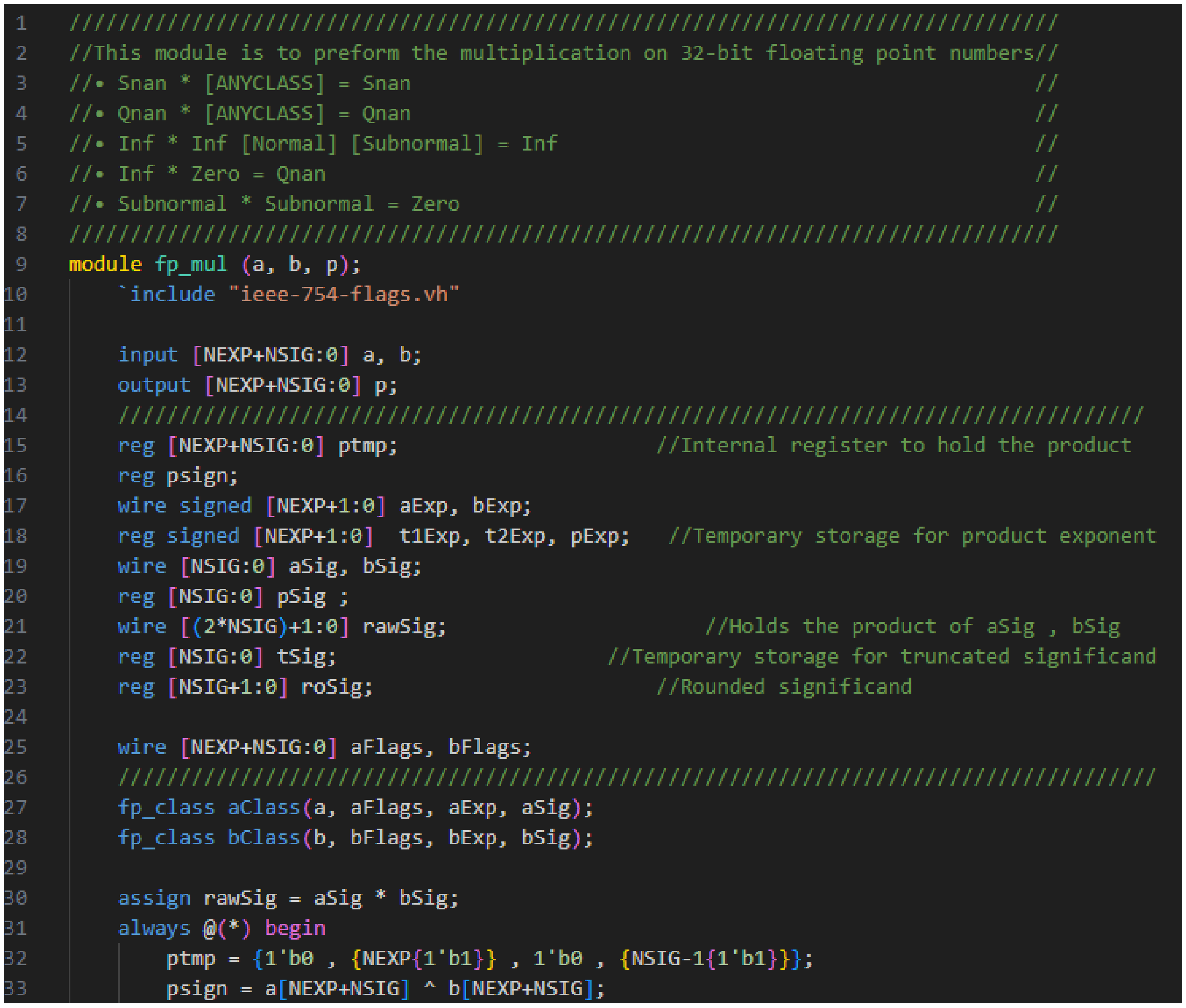


```verilog
                23'b00000000000000000000001: sa = 23;
                default: sa = 24;
            endcase
            normSig = normSig[EMAX] ? bigSig : (normSig << sa);
            normExp = normSig[EMAX] ? bigExp : (bigExp - sa);

            if (~| normSig) begin
                alwaysS = {{NEXP+NSIG+1{1'b0}}};
            end
            else if (expOut < EMIN)
              begin
                alwaysS = {absSign, {NEXP{1'b0}}, sigOut[NSIG:1]};
              end
            else if (expOut > EMAX) begin
                alwaysS = {absSign, {NEXP{1'b1}}, {NSIG{1'b0}}};
            end
            else begin
                biasExp = normExp + BIAS;
                //biasExp = expOut + BIAS;
                alwaysS = {absSign, biasExp[NEXP-1:0], sigOut[NSIG-1:0]};
            end
        end
    end
    // Compute sum/difference of significands
    assign sumSig = subtract ? (augendSig + ~addendSig + 279'b1) : (augendSig + addendSig);

    // Adjusted sign if absSum = -sigSum:
    assign absSign = sumSign ^ sumSig[EMAX+2];
    assign absSig = sumSig[EMAX+2] ? (42'b0 + ~sumSig + 279'b1) : (42'b0 + sumSig);

    // See if the addition caused a carry-out. If so, adjust the significad and the exponent.
    assign bigSig = absSig >> absSig[EMAX+1];
    assign bigExp = adjExp + absSig[EMAX+1];
```

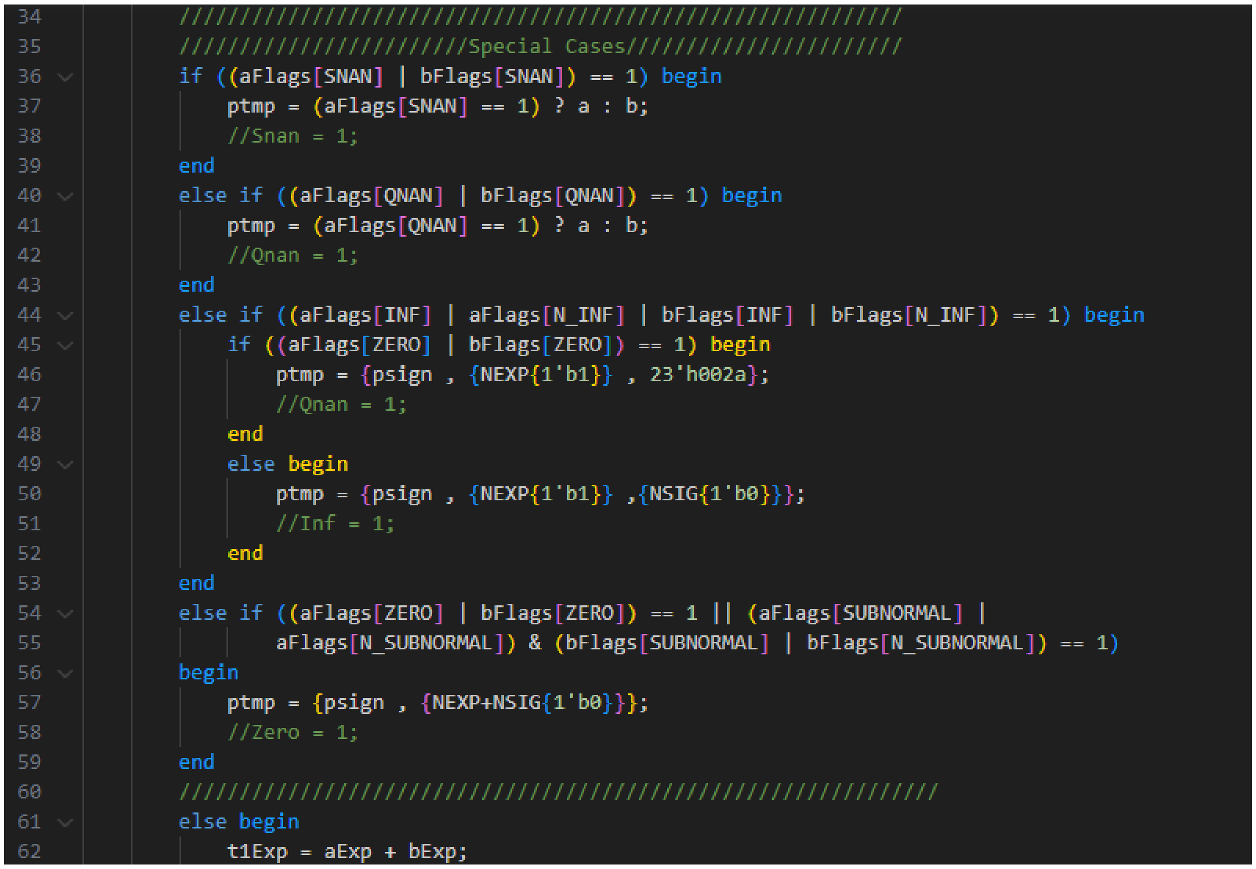


```verilog

    assign sigOut = normSig[EMAX : EMAX-23];
    assign s = alwaysS;
endmodule

```

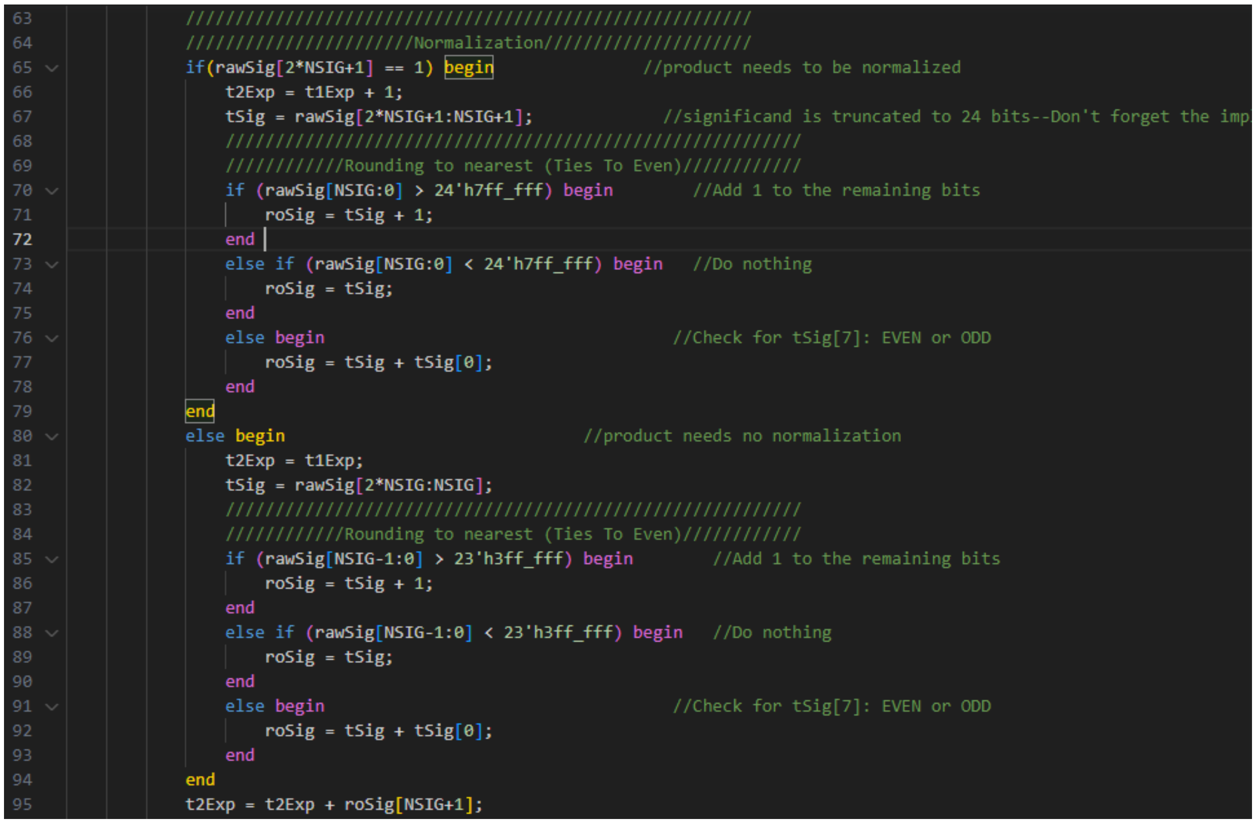
3)fp_mul:



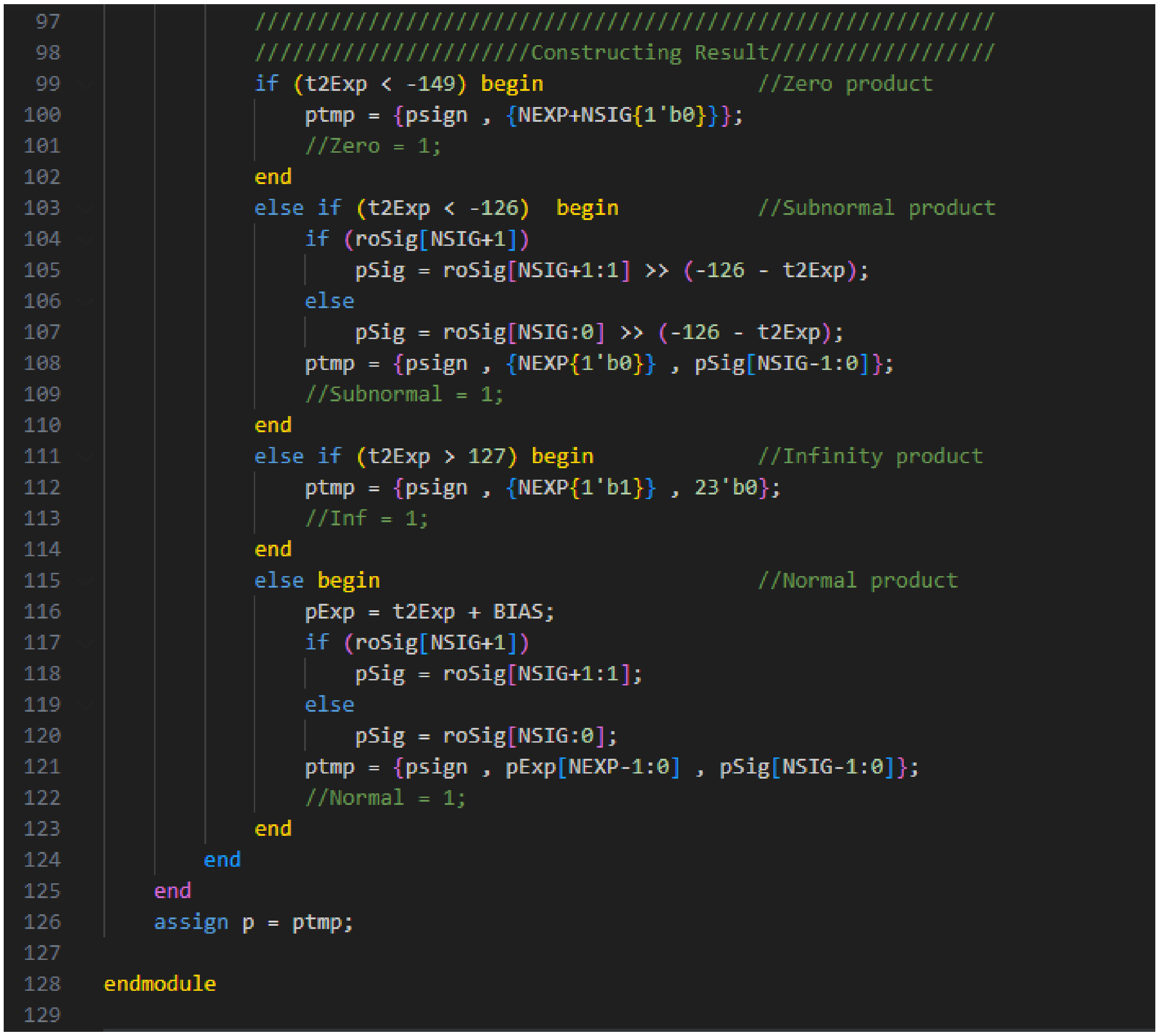
```
/////////////////////////////////////////////////////////////////////////////////
//This module is to preform the multiplication on 32-bit floating point numbers//
//• Snan * [ANYCLASS] = Snan                                                   //
//• Qnan * [ANYCLASS] = Qnan                                                   //
//• Inf * Inf [Normal] [Subnormal] = Inf                                       //
//• Inf * Zero = Qnan                                                          //
//• Subnormal * Subnormal = Zero                                               //
/////////////////////////////////////////////////////////////////////////////////
module fp_mul (a, b, p);
    `include "ieee-754-flags.vh"

    input [NEXP+NSIG:0] a, b;
    output [NEXP+NSIG:0] p;
    ////////////////////////////////////////////////////////////////////////////////////
    reg [NEXP+NSIG:0] ptmp;                     //Internal register to hold the product
    reg psign;
    wire signed [NEXP+1:0] aExp, bExp;
    reg signed [NEXP+1:0]  t1Exp, t2Exp, pExp;   //Temporary storage for product exponent
    wire [NSIG:0] aSig, bSig;
    reg [NSIG:0] pSig ;
    wire [(2*NSIG)+1:0] rawSig;                     //Holds the product of aSig , bSig
    reg [NSIG:0] tSig;                      //Temporary storage for truncated significand
    reg [NSIG+1:0] roSig;                       //Rounded significand

    wire [NEXP+NSIG:0] aFlags, bFlags;
    /////////////////////////////////////////////////////////////////////////////////////
    fp_class aClass(a, aFlags, aExp, aSig);
    fp_class bClass(b, bFlags, bExp, bSig);

    assign rawSig = aSig * bSig;
    always @(*) begin
        ptmp = {1'b0 , {NEXP{1'b1}} , 1'b0 , {NSIG-1{1'b1}}};
        psign = a[NEXP+NSIG] ^ b[NEXP+NSIG];
```



```verilog
////////////////////////////////////////////////////////////
///////////////////////Special Cases//////////////////////
if ((aFlags[SNAN] | bFlags[SNAN]) == 1) begin
    ptmp = (aFlags[SNAN] == 1) ? a : b;
    //Snan = 1;
end
else if ((aFlags[QNAN] | bFlags[QNAN]) == 1) begin
    ptmp = (aFlags[QNAN] == 1) ? a : b;
    //Qnan = 1;
end
else if ((aFlags[INF] | aFlags[N_INF] | bFlags[INF] | bFlags[N_INF]) == 1) begin
    if ((aFlags[ZERO] | bFlags[ZERO]) == 1) begin
        ptmp = {psign , {NEXP{1'b1}} , 23'h002a};
        //Qnan = 1;
    end
    else begin
        ptmp = {psign , {NEXP{1'b1}} ,{NSIG{1'b0}}};
        //Inf = 1;
    end
end
else if ((aFlags[ZERO] | bFlags[ZERO]) == 1 || (aFlags[SUBNORMAL] |
        aFlags[N_SUBNORMAL]) & (bFlags[SUBNORMAL] | bFlags[N_SUBNORMAL]) == 1)
begin
    ptmp = {psign , {NEXP+NSIG{1'b0}}};
    //Zero = 1;
end
///////////////////////////////////////////////////////////////
else begin
    t1Exp = aExp + bExp;
```

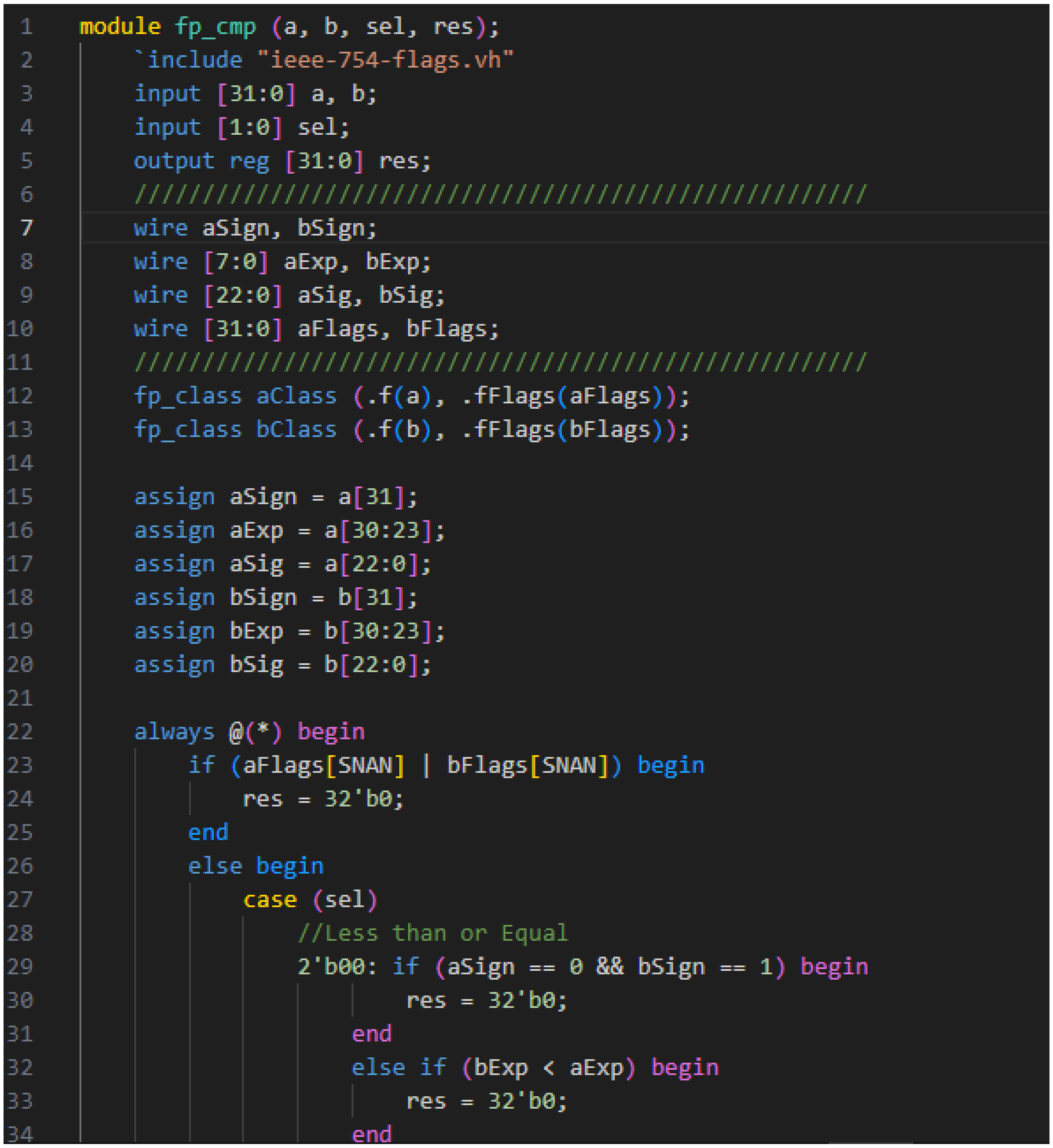


```verilog
////////////////////////////////////////////////////////////
//////////////////////Normalization///////////////////
if(rawSig[2*NSIG+1] == 1) begin             //product needs to be normalized
    t2Exp = t1Exp + 1;
    tSig = rawSig[2*NSIG+1:NSIG+1];             //significand is truncated to 24 bits--Don't forget the imp
    ////////////////////////////////////////////////////////////
    ///////////Rounding to nearest (Ties To Even)///////////
    if (rawSig[NSIG:0] > 24'h7ff_fff) begin        //Add 1 to the remaining bits
        roSig = tSig + 1;
    end
    else if (rawSig[NSIG:0] < 24'h7ff_fff) begin   //Do nothing
        roSig = tSig;
    end
    else begin                                  //Check for tSig[7]: EVEN or ODD
        roSig = tSig + tSig[0];
    end
end
else begin                                  //product needs no normalization
    t2Exp = t1Exp;
    tSig = rawSig[2*NSIG:NSIG];
    ////////////////////////////////////////////////////////////
    ///////////Rounding to nearest (Ties To Even)///////////
    if (rawSig[NSIG-1:0] > 23'h3ff_fff) begin        //Add 1 to the remaining bits
        roSig = tSig + 1;
    end
    else if (rawSig[NSIG-1:0] < 23'h3ff_fff) begin   //Do nothing
        roSig = tSig;
    end
    else begin                                  //Check for tSig[7]: EVEN or ODD
        roSig = tSig + tSig[0];
    end
end
t2Exp = t2Exp + roSig[NSIG+1];
```

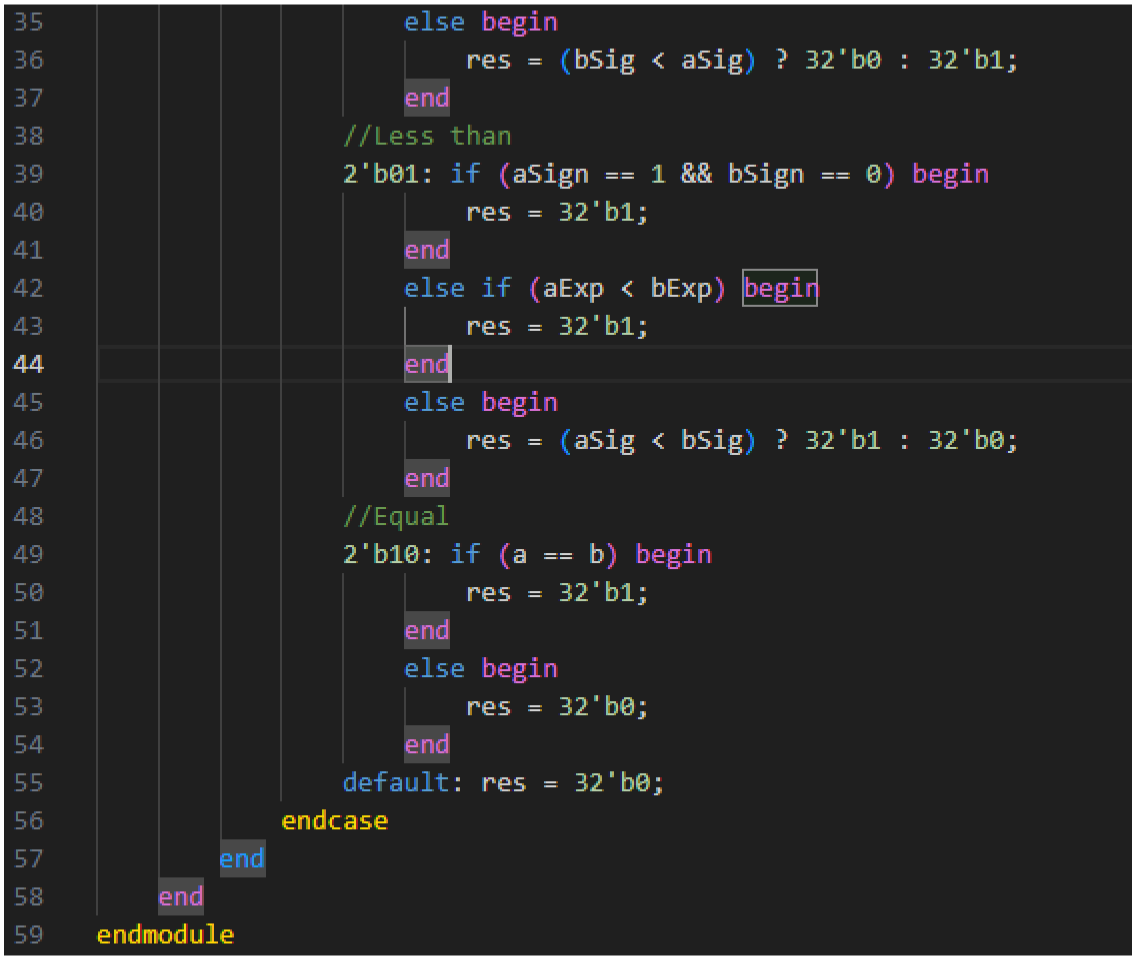


```
            ////////////////////////////////////////////////////////////
            //////////////////////Constructing Result/////////////////
            if (t2Exp < -149) begin                  //Zero product
                ptmp = {psign , {NEXP+NSIG{1'b0}}};
                //Zero = 1;
            end
            else if (t2Exp < -126)  begin            //Subnormal product
                if (roSig[NSIG+1])
                    pSig = roSig[NSIG+1:1] >> (-126 - t2Exp);
                else
                    pSig = roSig[NSIG:0] >> (-126 - t2Exp);
                ptmp = {psign , {NEXP{1'b0}} , pSig[NSIG-1:0]};
                //Subnormal = 1;
            end
            else if (t2Exp > 127) begin              //Infinity product
                ptmp = {psign , {NEXP{1'b1}} , 23'b0};
                //Inf = 1;
            end
            else begin                               //Normal product
                pExp = t2Exp + BIAS;
                if (roSig[NSIG+1])
                    pSig = roSig[NSIG+1:1];
                else
                    pSig = roSig[NSIG:0];
                ptmp = {psign , pExp[NEXP-1:0] , pSig[NSIG-1:0]};
                //Normal = 1;
            end
        end
    end
    assign p = ptmp;

endmodule
```

4)fp_cmp:

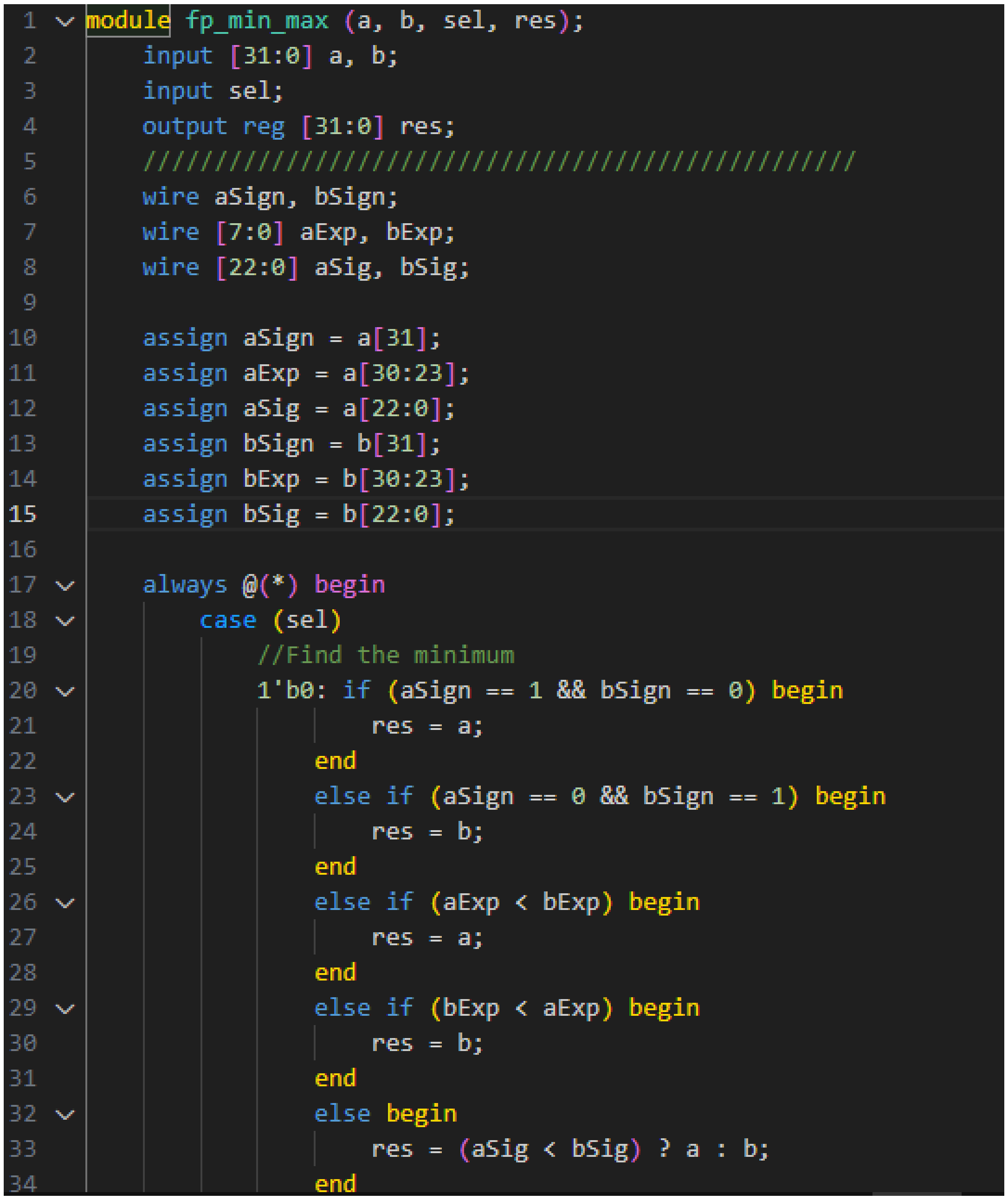


```
module fp_cmp (a, b, sel, res);
    `include "ieee-754-flags.vh"
    input [31:0] a, b;
    input [1:0] sel;
    output reg [31:0] res;
    ///////////////////////////////////////////////////////
    wire aSign, bSign;
    wire [7:0] aExp, bExp;
    wire [22:0] aSig, bSig;
    wire [31:0] aFlags, bFlags;
    ///////////////////////////////////////////////////////
    fp_class aClass (.f(a), .fFlags(aFlags));
    fp_class bClass (.f(b), .fFlags(bFlags));

    assign aSign = a[31];
    assign aExp = a[30:23];
    assign aSig = a[22:0];
    assign bSign = b[31];
    assign bExp = b[30:23];
    assign bSig = b[22:0];

    always @(*) begin
        if (aFlags[SNAN] | bFlags[SNAN]) begin
            res = 32'b0;
        end
        else begin
            case (sel)
                //Less than or Equal
                2'b00: if (aSign == 0 && bSign == 1) begin
                        res = 32'b0;
                    end
                    else if (bExp < aExp) begin
                        res = 32'b0;
                    end
```

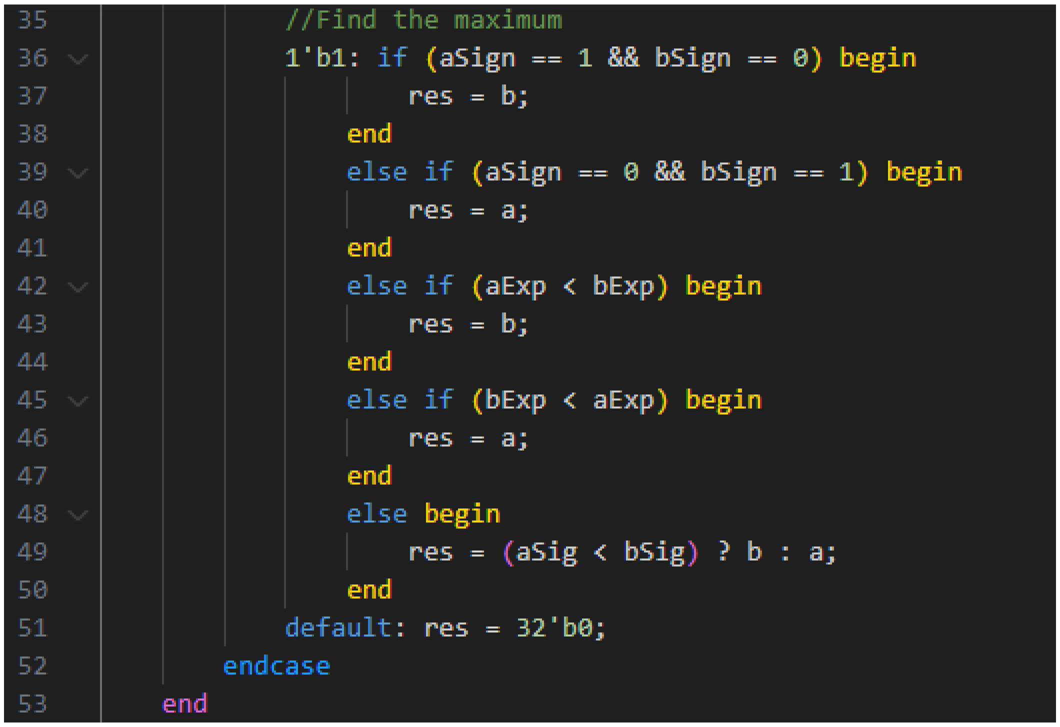


```
                    else begin
                        res = (bSig < aSig) ? 32'b0 : 32'b1;
                    end
                //Less than
                2'b01: if (aSign == 1 && bSign == 0) begin
                        res = 32'b1;
                    end
                    else if (aExp < bExp) begin
                        res = 32'b1;
                    end
                    else begin
                        res = (aSig < bSig) ? 32'b1 : 32'b0;
                    end
                //Equal
                2'b10: if (a == b) begin
                        res = 32'b1;
                    end
                    else begin
                        res = 32'b0;
                    end
                default: res = 32'b0;
            endcase
        end
    end
endmodule
```

5)fp_min_max:

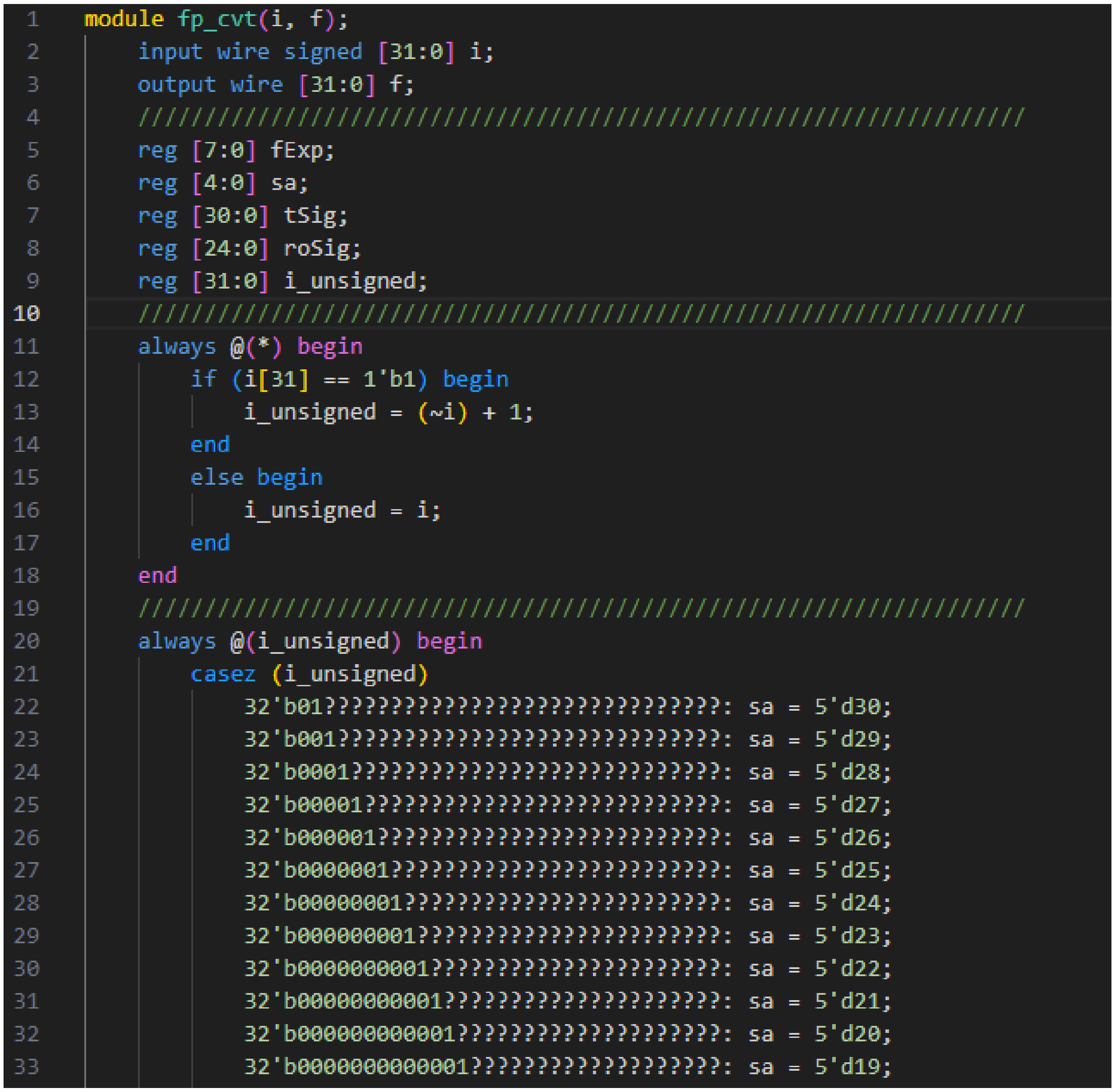


```
module fp_min_max (a, b, sel, res);
    input [31:0] a, b;
    input sel;
    output reg [31:0] res;
    ///////////////////////////////////////////////
    wire aSign, bSign;
    wire [7:0] aExp, bExp;
    wire [22:0] aSig, bSig;

    assign aSign = a[31];
    assign aExp = a[30:23];
    assign aSig = a[22:0];
    assign bSign = b[31];
    assign bExp = b[30:23];
    assign bSig = b[22:0];

    always @(*) begin
        case (sel)
            //Find the minimum
            1'b0: if (aSign == 1 && bSign == 0) begin
                    res = a;
                end
                else if (aSign == 0 && bSign == 1) begin
                    res = b;
                end
                else if (aExp < bExp) begin
                    res = a;
                end
                else if (bExp < aExp) begin
                    res = b;
                end
                else begin
                    res = (aSig < bSig) ? a : b;
                end
```

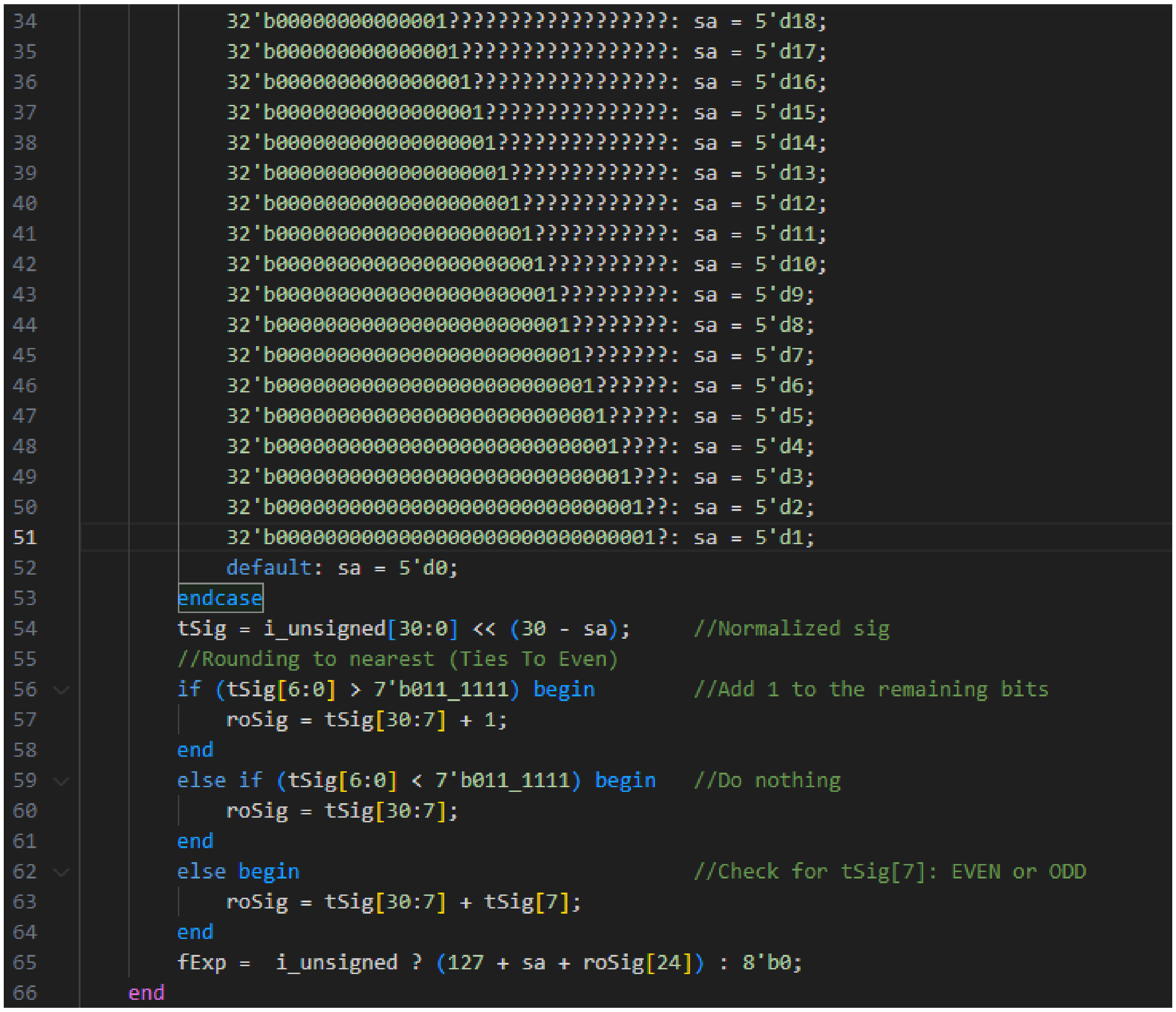


```
            //Find the maximum
            1'b1: if (aSign == 1 && bSign == 0) begin
                    res = b;
                end
                else if (aSign == 0 && bSign == 1) begin
                    res = a;
                end
                else if (aExp < bExp) begin
                    res = b;
                end
                else if (bExp < aExp) begin
                    res = a;
                end
                else begin
                    res = (aSig < bSig) ? b : a;
                end
            default: res = 32'b0;
        endcase
    end
```

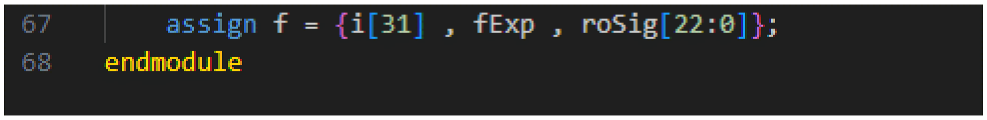
6)fp_cvt:



```
module fp_cvt(i, f);
    input wire signed [31:0] i;
    output wire [31:0] f;
    ////////////////////////////////////////////////////////////////////
    reg [7:0] fExp;
    reg [4:0] sa;
    reg [30:0] tSig;
    reg [24:0] roSig;
    reg [31:0] i_unsigned;
    ////////////////////////////////////////////////////////////////////
    always @(*) begin
        if (i[31] == 1'b1) begin
            i_unsigned = (~i) + 1;
        end
        else begin
            i_unsigned = i;
        end
    end
    ////////////////////////////////////////////////////////////////////
    always @(i_unsigned) begin
        casez (i_unsigned)
            32'b01??????????????????????????????: sa = 5'd30;
            32'b001?????????????????????????????: sa = 5'd29;
            32'b0001????????????????????????????: sa = 5'd28;
            32'b00001???????????????????????????: sa = 5'd27;
            32'b000001??????????????????????????: sa = 5'd26;
            32'b0000001?????????????????????????: sa = 5'd25;
            32'b00000001????????????????????????: sa = 5'd24;
            32'b000000001???????????????????????: sa = 5'd23;
            32'b0000000001??????????????????????: sa = 5'd22;
            32'b00000000001?????????????????????: sa = 5'd21;
            32'b000000000001????????????????????: sa = 5'd20;
            32'b0000000000001???????????????????: sa = 5'd19;
```

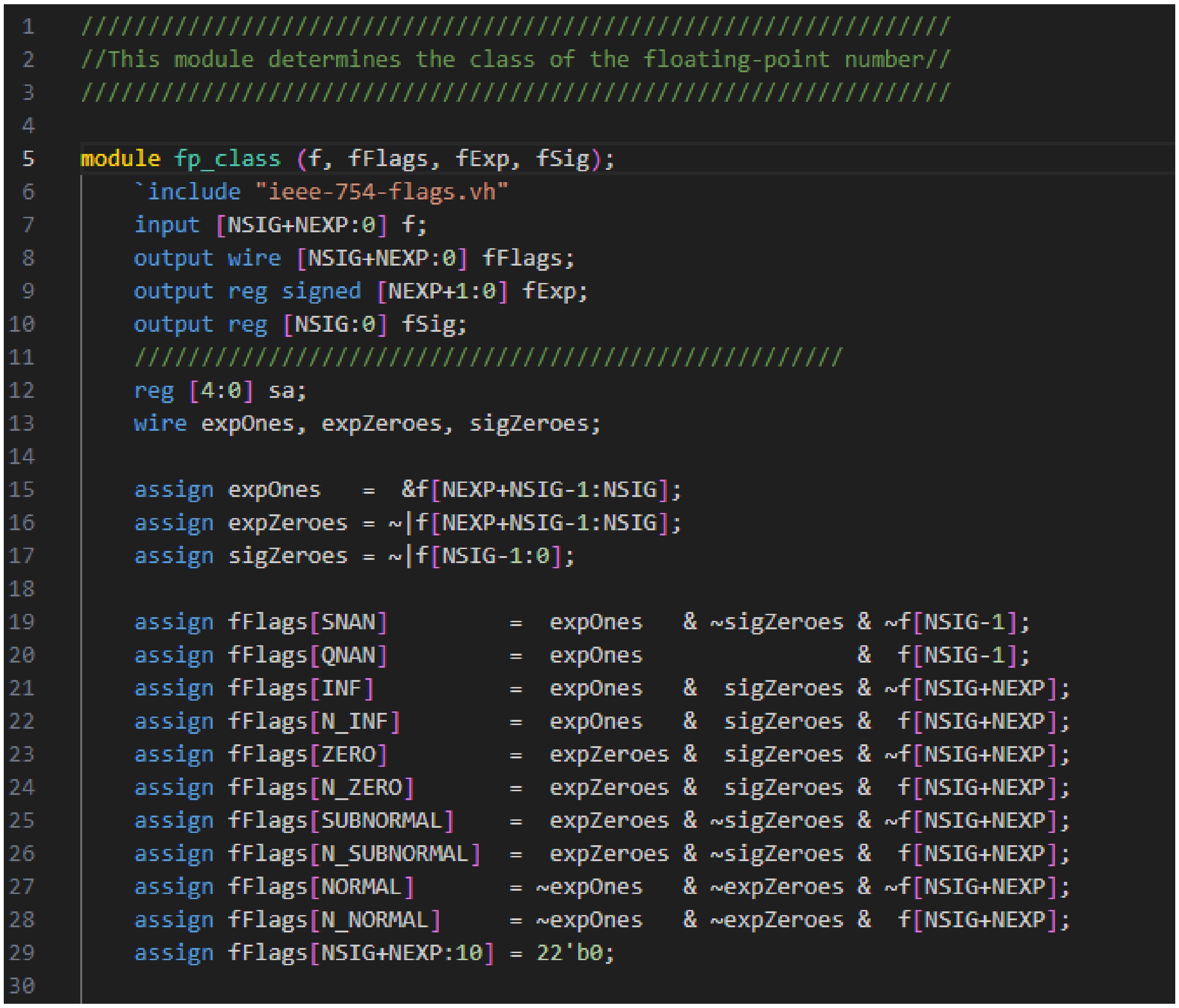


```verilog
            32'b00000000000001??????????????????: sa = 5'd18;
            32'b000000000000001?????????????????: sa = 5'd17;
            32'b0000000000000001????????????????: sa = 5'd16;
            32'b00000000000000001???????????????: sa = 5'd15;
            32'b000000000000000001??????????????: sa = 5'd14;
            32'b0000000000000000001?????????????: sa = 5'd13;
            32'b00000000000000000001????????????: sa = 5'd12;
            32'b000000000000000000001???????????: sa = 5'd11;
            32'b0000000000000000000001??????????: sa = 5'd10;
            32'b00000000000000000000001?????????: sa = 5'd9;
            32'b000000000000000000000001????????: sa = 5'd8;
            32'b0000000000000000000000001???????: sa = 5'd7;
            32'b00000000000000000000000001??????: sa = 5'd6;
            32'b000000000000000000000000001?????: sa = 5'd5;
            32'b0000000000000000000000000001????: sa = 5'd4;
            32'b00000000000000000000000000001???: sa = 5'd3;
            32'b000000000000000000000000000001??: sa = 5'd2;
            32'b0000000000000000000000000000001?: sa = 5'd1;
            default: sa = 5'd0;
        endcase
        tSig = i_unsigned[30:0] << (30 - sa);     //Normalized sig
        //Rounding to nearest (Ties To Even)
        if (tSig[6:0] > 7'b011_1111) begin        //Add 1 to the remaining bits
            roSig = tSig[30:7] + 1;
        end
        else if (tSig[6:0] < 7'b011_1111) begin   //Do nothing
            roSig = tSig[30:7];
        end
        else begin                                //Check for tSig[7]: EVEN or ODD
            roSig = tSig[30:7] + tSig[7];
        end
        fExp =  i_unsigned ? (127 + sa + roSig[24]) : 8'b0;
    end
```

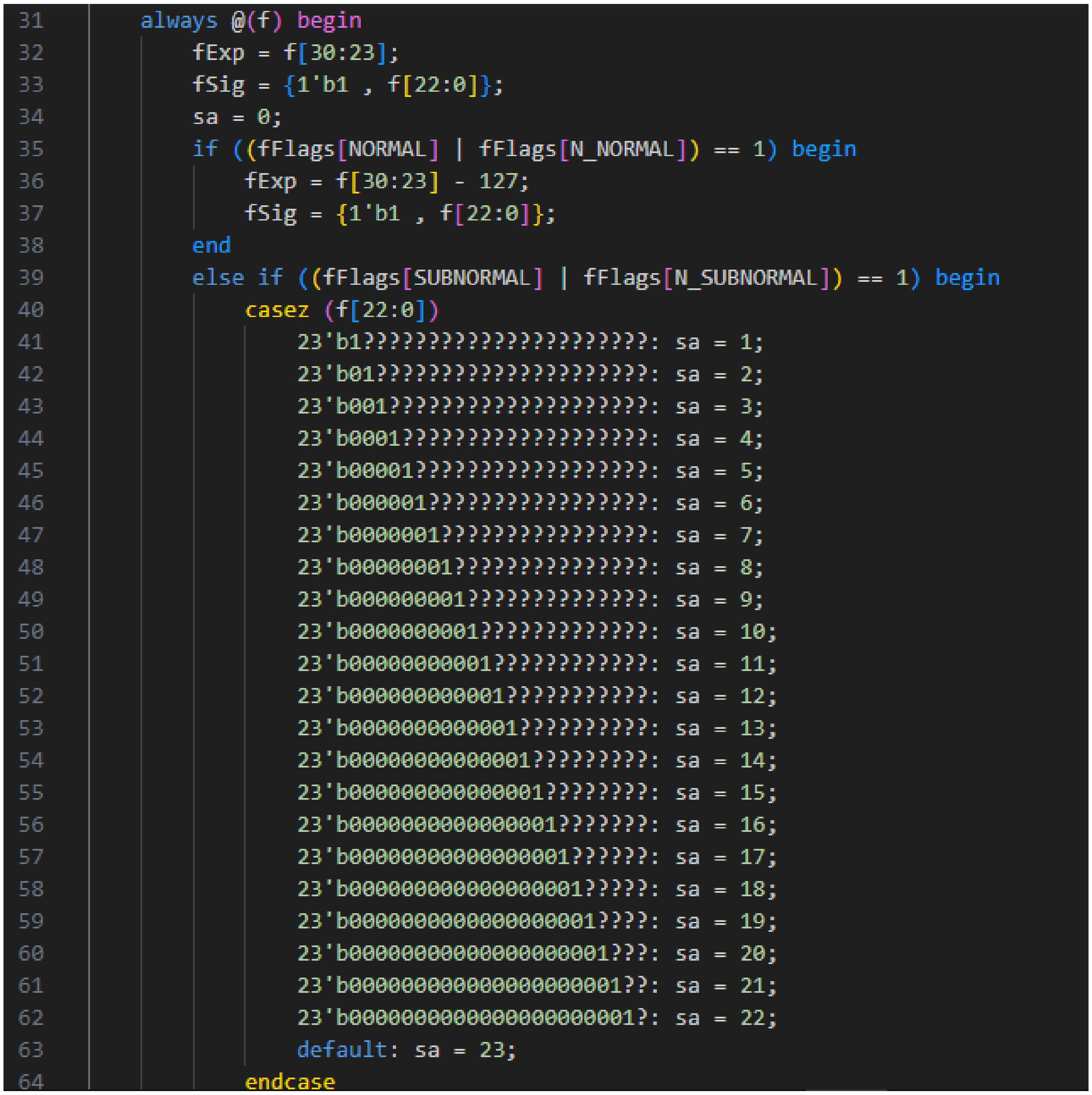


```verilog
    assign f = {i[31] , fExp , roSig[22:0]};
endmodule
```

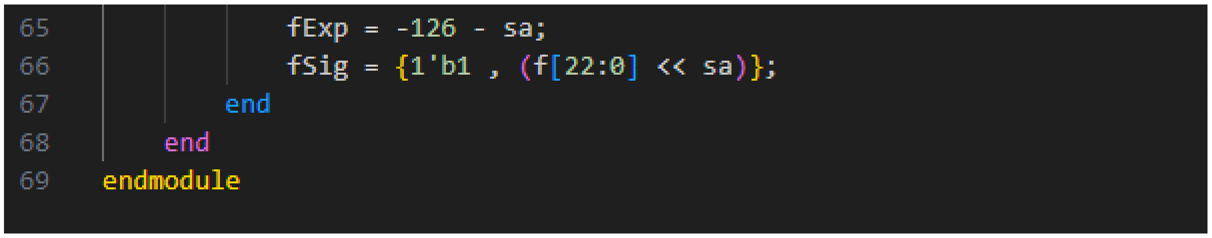
7)fp_class:



```verilog
/////////////////////////////////////////////////////////////////
//This module determines the class of the floating-point number//
/////////////////////////////////////////////////////////////////

module fp_class (f, fFlags, fExp, fSig);
    `include "ieee-754-flags.vh"
    input [NSIG+NEXP:0] f;
    output wire [NSIG+NEXP:0] fFlags;
    output reg signed [NEXP+1:0] fExp;
    output reg [NSIG:0] fSig;
    /////////////////////////////////////////////////////
    reg [4:0] sa;
    wire expOnes, expZeroes, sigZeroes;

    assign expOnes   =  &f[NEXP+NSIG-1:NSIG];
    assign expZeroes = ~|f[NEXP+NSIG-1:NSIG];
    assign sigZeroes = ~|f[NSIG-1:0];

    assign fFlags[SNAN]         =  expOnes   & ~sigZeroes & ~f[NSIG-1];
    assign fFlags[QNAN]         =  expOnes                &  f[NSIG-1];
    assign fFlags[INF]          =  expOnes   &  sigZeroes & ~f[NSIG+NEXP];
    assign fFlags[N_INF]        =  expOnes   &  sigZeroes &  f[NSIG+NEXP];
    assign fFlags[ZERO]         =  expZeroes &  sigZeroes & ~f[NSIG+NEXP];
    assign fFlags[N_ZERO]       =  expZeroes &  sigZeroes &  f[NSIG+NEXP];
    assign fFlags[SUBNORMAL]    =  expZeroes & ~sigZeroes & ~f[NSIG+NEXP];
    assign fFlags[N_SUBNORMAL]  =  expZeroes & ~sigZeroes &  f[NSIG+NEXP];
    assign fFlags[NORMAL]       = ~expOnes   & ~expZeroes & ~f[NSIG+NEXP];
    assign fFlags[N_NORMAL]     = ~expOnes   & ~expZeroes &  f[NSIG+NEXP];
    assign fFlags[NSIG+NEXP:10] = 22'b0;

```

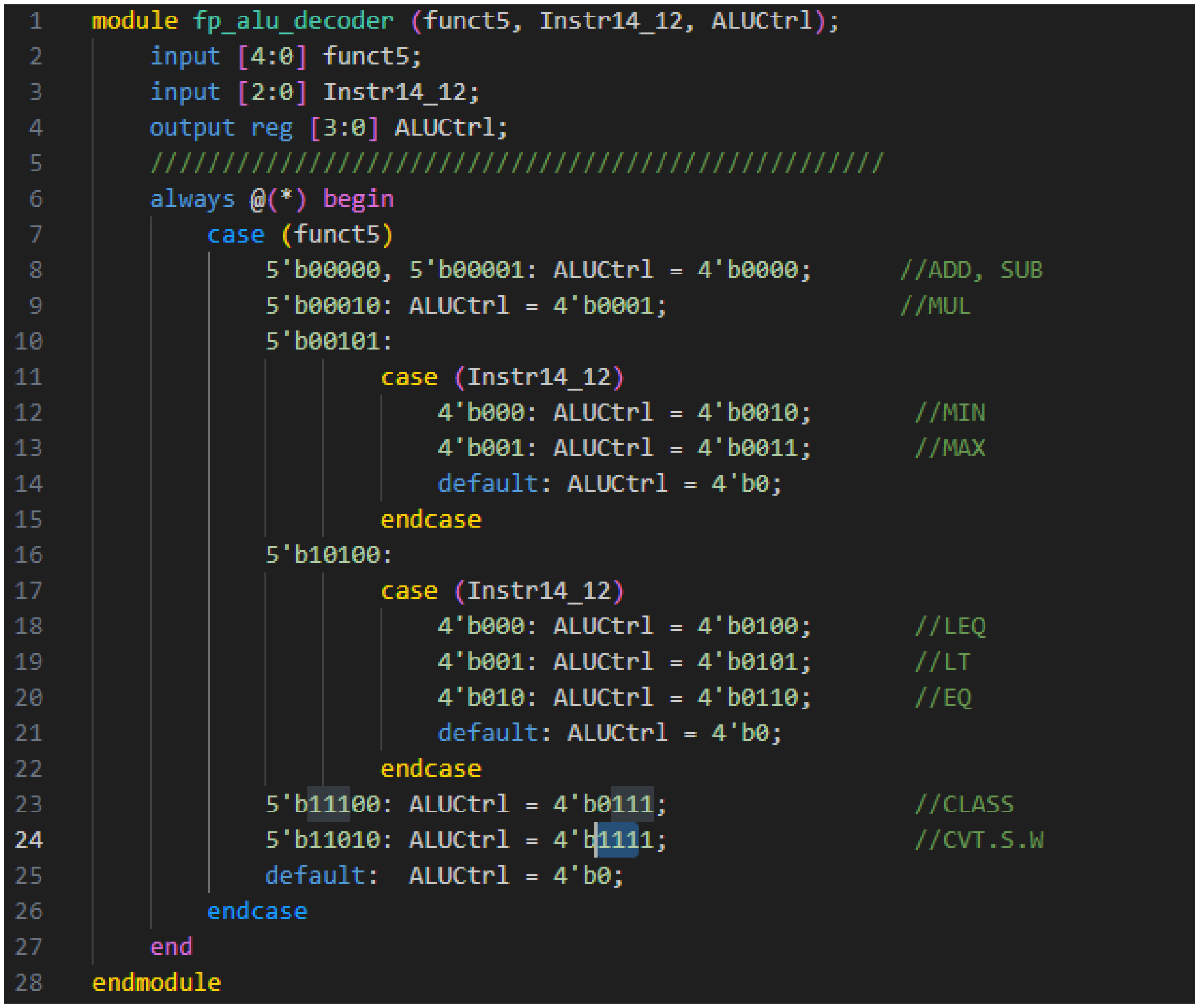


```verilog
    always @(f) begin
        fExp = f[30:23];
        fSig = {1'b1 , f[22:0]};
        sa = 0;
        if ((fFlags[NORMAL] | fFlags[N_NORMAL]) == 1) begin
            fExp = f[30:23] - 127;
            fSig = {1'b1 , f[22:0]};
        end
        else if ((fFlags[SUBNORMAL] | fFlags[N_SUBNORMAL]) == 1) begin
            casez (f[22:0])
                23'b1??????????????????????: sa = 1;
                23'b01?????????????????????: sa = 2;
                23'b001????????????????????: sa = 3;
                23'b0001???????????????????: sa = 4;
                23'b00001??????????????????: sa = 5;
                23'b000001?????????????????: sa = 6;
                23'b0000001????????????????: sa = 7;
                23'b00000001???????????????: sa = 8;
                23'b000000001??????????????: sa = 9;
                23'b0000000001?????????????: sa = 10;
                23'b00000000001????????????: sa = 11;
                23'b000000000001???????????: sa = 12;
                23'b0000000000001??????????: sa = 13;
                23'b00000000000001?????????: sa = 14;
                23'b000000000000001????????: sa = 15;
                23'b0000000000000001???????: sa = 16;
                23'b00000000000000001??????: sa = 17;
                23'b000000000000000001?????: sa = 18;
                23'b0000000000000000001????: sa = 19;
                23'b00000000000000000001???: sa = 20;
                23'b000000000000000000001??: sa = 21;
                23'b0000000000000000000001?: sa = 22;
                default: sa = 23;
            endcase
```



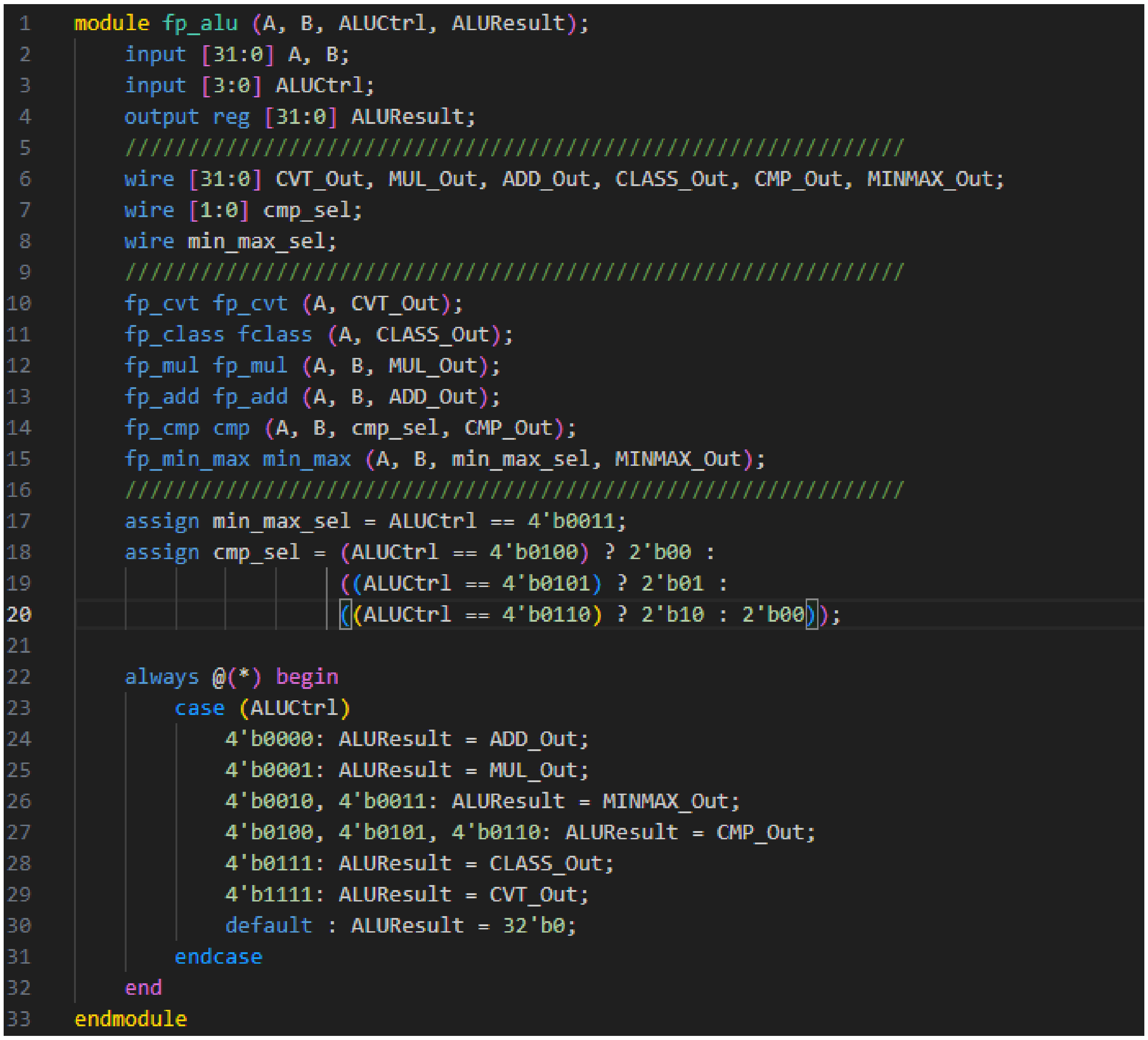
```verilog
            fExp = -126 - sa;
            fSig = {1'b1 , (f[22:0] << sa)};
        end
    end
endmodule
```

8)fp_alu_decoder:



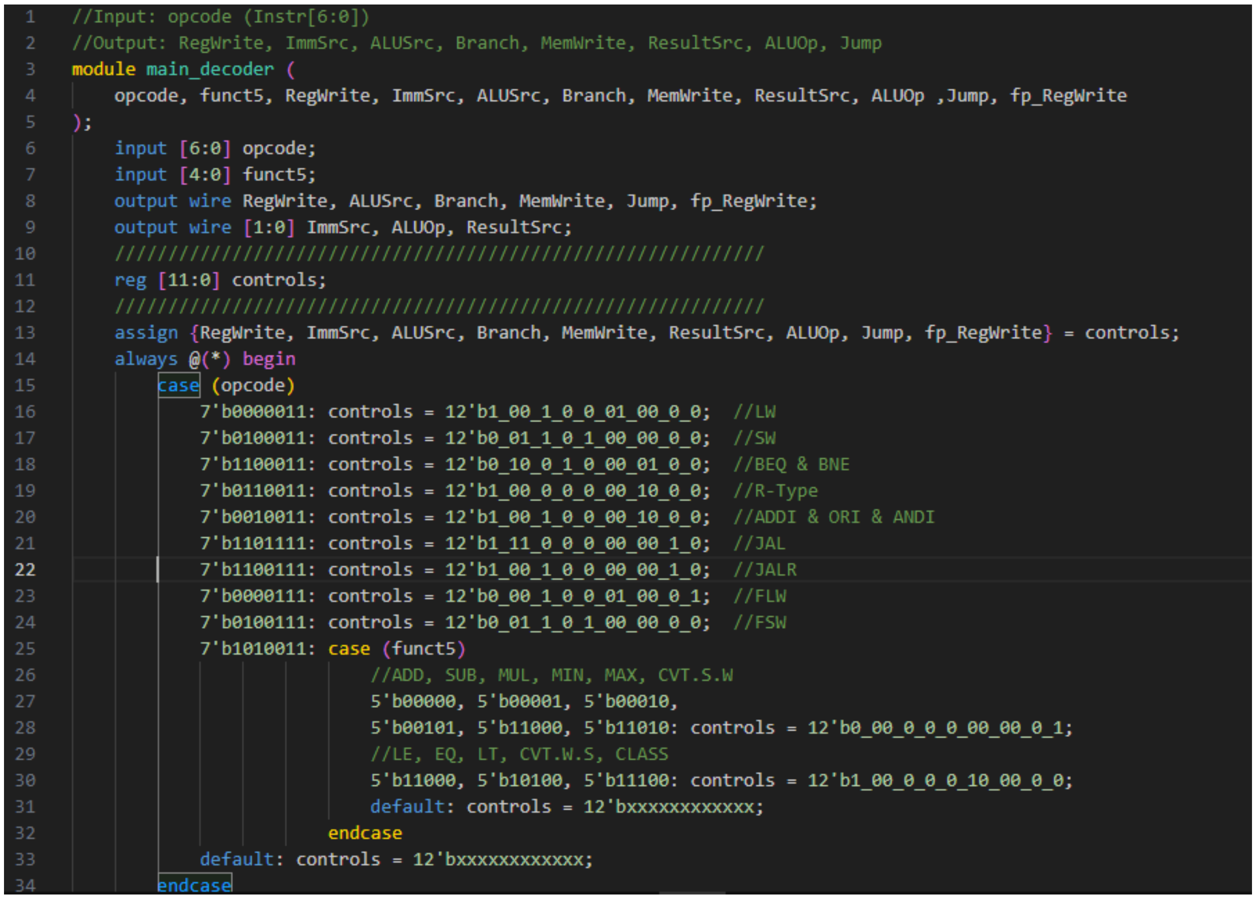
```
module fp_alu_decoder (funct5, Instr14_12, ALUCtrl);
    input [4:0] funct5;
    input [2:0] Instr14_12;
    output reg [3:0] ALUCtrl;
    ///////////////////////////////////////////////////
    always @(*) begin
        case (funct5)
            5'b00000, 5'b00001: ALUCtrl = 4'b0000;      //ADD, SUB
            5'b00010: ALUCtrl = 4'b0001;                //MUL
            5'b00101:
                    case (Instr14_12)
                        4'b000: ALUCtrl = 4'b0010;       //MIN
                        4'b001: ALUCtrl = 4'b0011;       //MAX
                        default: ALUCtrl = 4'b0;
                    endcase
            5'b10100:
                    case (Instr14_12)
                        4'b000: ALUCtrl = 4'b0100;       //LEQ
                        4'b001: ALUCtrl = 4'b0101;       //LT
                        4'b010: ALUCtrl = 4'b0110;       //EQ
                        default: ALUCtrl = 4'b0;
                    endcase
            5'b11100: ALUCtrl = 4'b0111;                 //CLASS
            5'b11010: ALUCtrl = 4'b1111;                 //CVT.S.W
            default:  ALUCtrl = 4'b0;
        endcase
    end
endmodule
```

9)fp_alu:

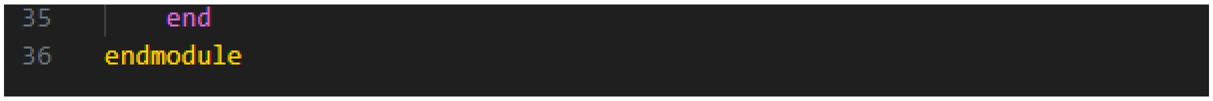


```verilog
module fp_alu (A, B, ALUCtrl, ALUResult);
    input [31:0] A, B;
    input [3:0] ALUCtrl;
    output reg [31:0] ALUResult;
    //////////////////////////////////////////////////////////////
    wire [31:0] CVT_Out, MUL_Out, ADD_Out, CLASS_Out, CMP_Out, MINMAX_Out;
    wire [1:0] cmp_sel;
    wire min_max_sel;
    //////////////////////////////////////////////////////////////
    fp_cvt fp_cvt (A, CVT_Out);
    fp_class fclass (A, CLASS_Out);
    fp_mul fp_mul (A, B, MUL_Out);
    fp_add fp_add (A, B, ADD_Out);
    fp_cmp cmp (A, B, cmp_sel, CMP_Out);
    fp_min_max min_max (A, B, min_max_sel, MINMAX_Out);
    //////////////////////////////////////////////////////////////
    assign min_max_sel = ALUCtrl == 4'b0011;
    assign cmp_sel = (ALUCtrl == 4'b0100) ? 2'b00 :
                     ((ALUCtrl == 4'b0101) ? 2'b01 :
                     ((ALUCtrl == 4'b0110) ? 2'b10 : 2'b00));

    always @(*) begin
        case (ALUCtrl)
            4'b0000: ALUResult = ADD_Out;
            4'b0001: ALUResult = MUL_Out;
            4'b0010, 4'b0011: ALUResult = MINMAX_Out;
            4'b0100, 4'b0101, 4'b0110: ALUResult = CMP_Out;
            4'b0111: ALUResult = CLASS_Out;
            4'b1111: ALUResult = CVT_Out;
            default : ALUResult = 32'b0;
        endcase
    end
endmodule
```

10)main_decoder:

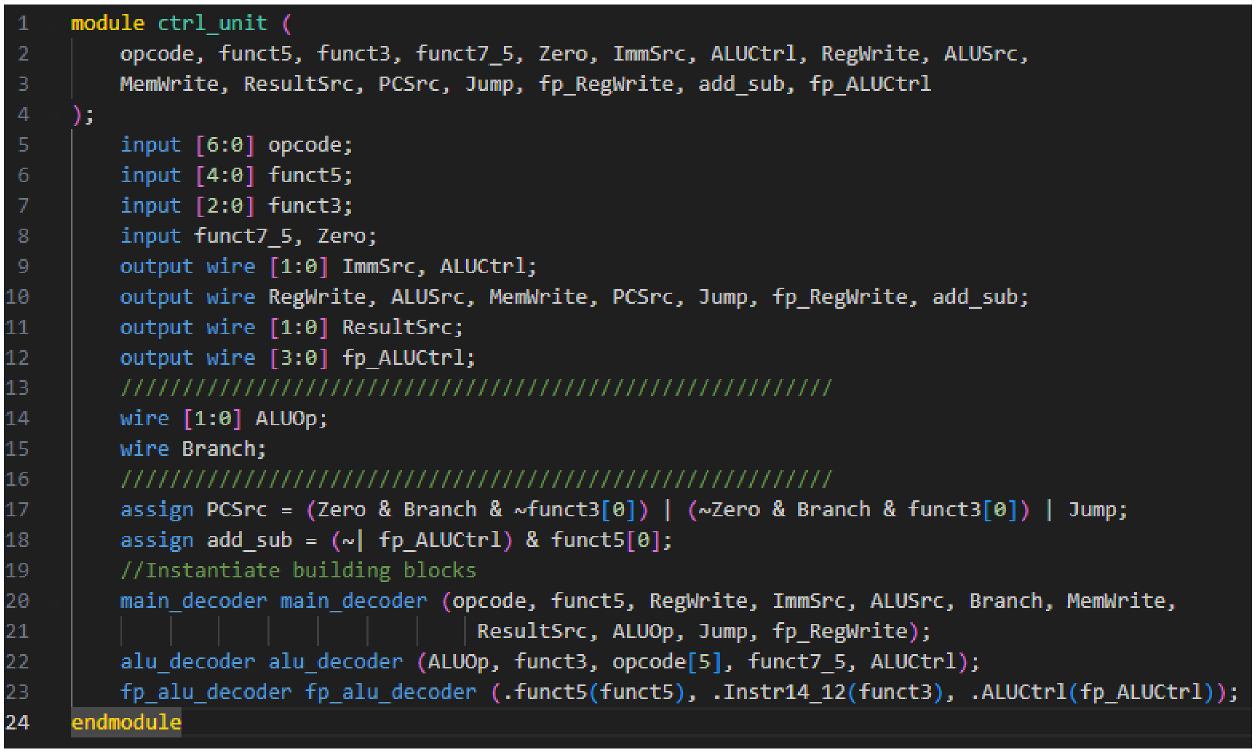


```verilog
//Input: opcode (Instr[6:0])
//Output: RegWrite, ImmSrc, ALUSrc, Branch, MemWrite, ResultSrc, ALUOp, Jump
module main_decoder (
    opcode, funct5, RegWrite, ImmSrc, ALUSrc, Branch, MemWrite, ResultSrc, ALUOp ,Jump, fp_RegWrite
);
    input [6:0] opcode;
    input [4:0] funct5;
    output wire RegWrite, ALUSrc, Branch, MemWrite, Jump, fp_RegWrite;
    output wire [1:0] ImmSrc, ALUOp, ResultSrc;
    /////////////////////////////////////////////////////////////
    reg [11:0] controls;
    /////////////////////////////////////////////////////////////
    assign {RegWrite, ImmSrc, ALUSrc, Branch, MemWrite, ResultSrc, ALUOp, Jump, fp_RegWrite} = controls;
    always @(*) begin
        case (opcode)
            7'b0000011: controls = 12'b1_00_1_0_0_01_00_0_0;  //LW
            7'b0100011: controls = 12'b0_01_1_0_1_00_00_0_0;  //SW
            7'b1100011: controls = 12'b0_10_0_1_0_00_01_0_0;  //BEQ & BNE
            7'b0110011: controls = 12'b1_00_0_0_0_00_10_0_0;  //R-Type
            7'b0010011: controls = 12'b1_00_1_0_0_00_10_0_0;  //ADDI & ORI & ANDI
            7'b1101111: controls = 12'b1_11_0_0_0_00_00_1_0;  //JAL
            7'b1100111: controls = 12'b1_00_1_0_0_00_00_1_0;  //JALR
            7'b0000111: controls = 12'b0_00_1_0_0_01_00_0_1;  //FLW
            7'b0100111: controls = 12'b0_01_1_0_1_00_00_0_0;  //FSW
            7'b1010011: case (funct5)
                            //ADD, SUB, MUL, MIN, MAX, CVT.S.W
                            5'b00000, 5'b00001, 5'b00010,
                            5'b00101, 5'b11000, 5'b11010: controls = 12'b0_00_0_0_0_00_00_0_1;
                            //LE, EQ, LT, CVT.W.S, CLASS
                            5'b11000, 5'b10100, 5'b11100: controls = 12'b1_00_0_0_0_10_00_0_0;
                            default: controls = 12'bxxxxxxxxxxxx;
                        endcase
            default: controls = 12'bxxxxxxxxxxxx;
        endcase
```



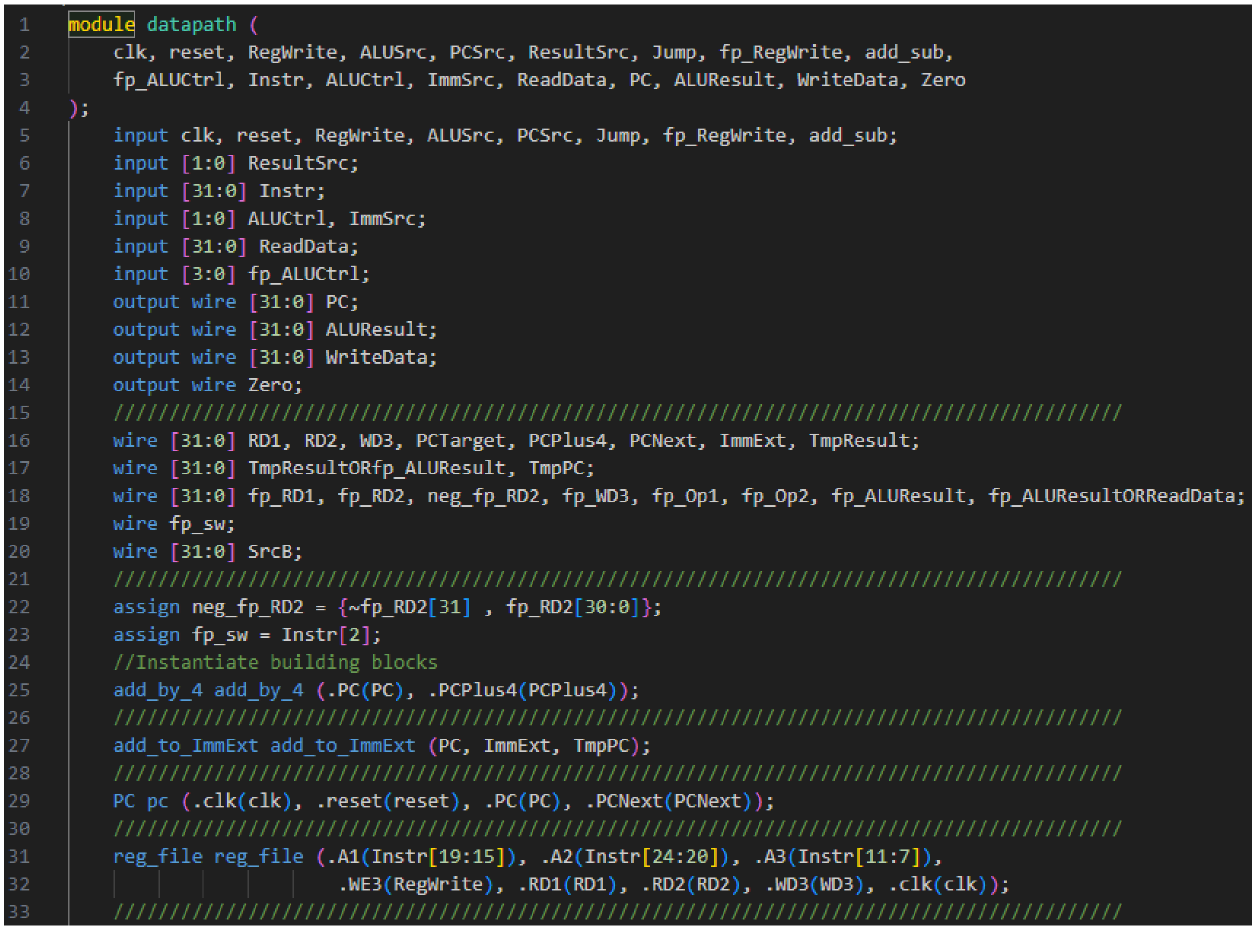
```verilog
    end
endmodule
```

11)ctrl_unit:

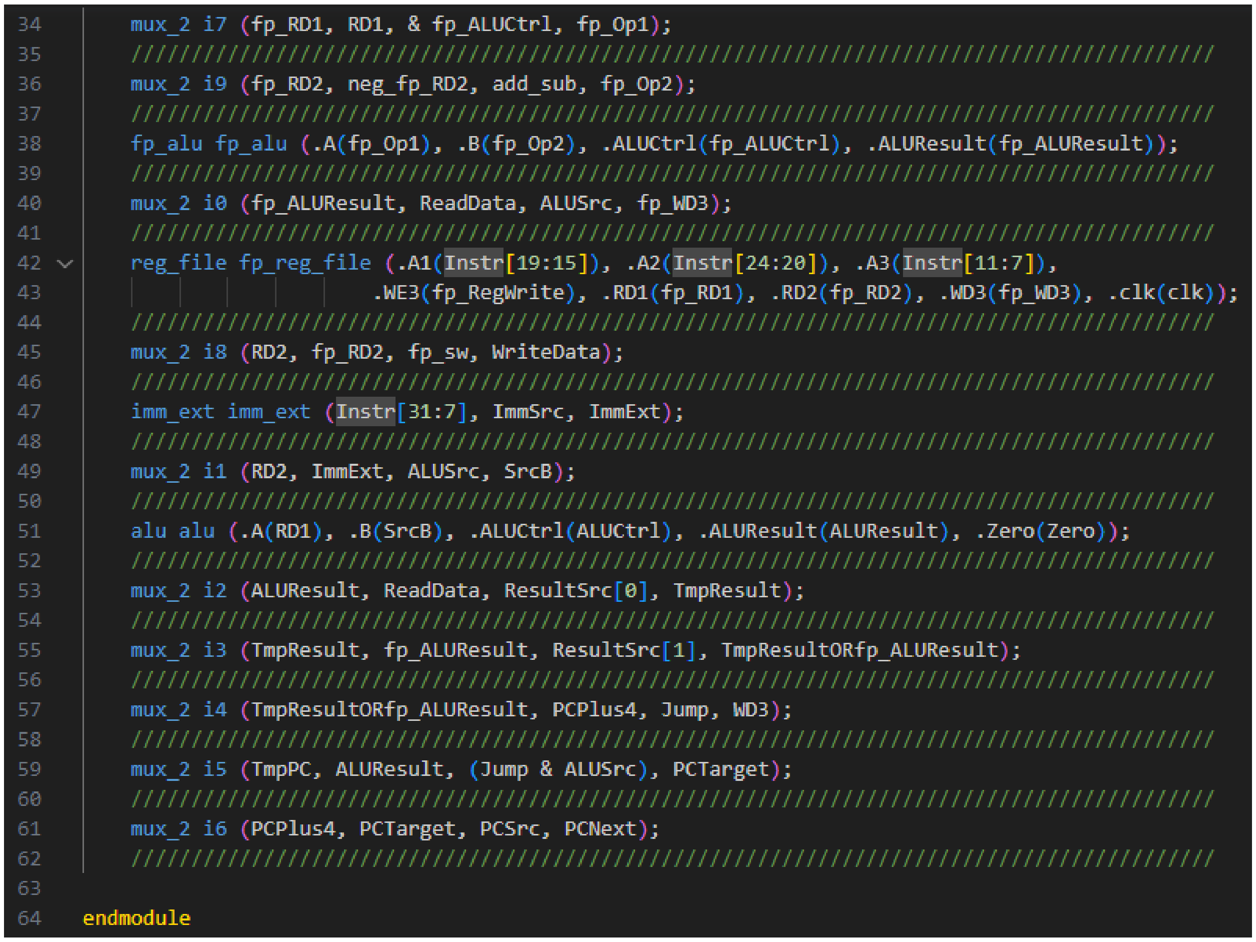


```verilog
module ctrl_unit (
    opcode, funct5, funct3, funct7_5, Zero, ImmSrc, ALUCtrl, RegWrite, ALUSrc,
    MemWrite, ResultSrc, PCSrc, Jump, fp_RegWrite, add_sub, fp_ALUCtrl
);
    input [6:0] opcode;
    input [4:0] funct5;
    input [2:0] funct3;
    input funct7_5, Zero;
    output wire [1:0] ImmSrc, ALUCtrl;
    output wire RegWrite, ALUSrc, MemWrite, PCSrc, Jump, fp_RegWrite, add_sub;
    output wire [1:0] ResultSrc;
    output wire [3:0] fp_ALUCtrl;
    ////////////////////////////////////////////////////////////
    wire [1:0] ALUOp;
    wire Branch;
    ////////////////////////////////////////////////////////////
    assign PCSrc = (Zero & Branch & ~funct3[0]) | (~Zero & Branch & funct3[0]) | Jump;
    assign add_sub = (~| fp_ALUCtrl) & funct5[0];
    //Instantiate building blocks
    main_decoder main_decoder (opcode, funct5, RegWrite, ImmSrc, ALUSrc, Branch, MemWrite,
                              ResultSrc, ALUOp, Jump, fp_RegWrite);
    alu_decoder alu_decoder (ALUOp, funct3, opcode[5], funct7_5, ALUCtrl);
    fp_alu_decoder fp_alu_decoder (.funct5(funct5), .Instr14_12(funct3), .ALUCtrl(fp_ALUCtrl));
endmodule
```

12)datapath:

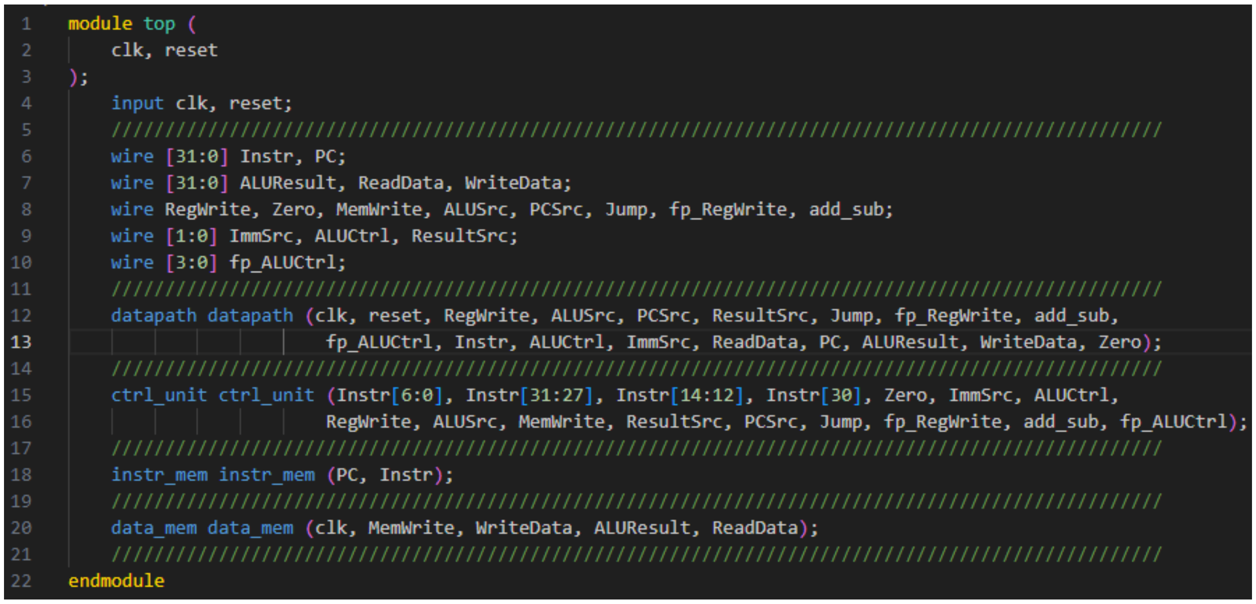


```
module datapath (
    clk, reset, RegWrite, ALUSrc, PCSrc, ResultSrc, Jump, fp_RegWrite, add_sub,
    fp_ALUCtrl, Instr, ALUCtrl, ImmSrc, ReadData, PC, ALUResult, WriteData, Zero
);
    input clk, reset, RegWrite, ALUSrc, PCSrc, Jump, fp_RegWrite, add_sub;
    input [1:0] ResultSrc;
    input [31:0] Instr;
    input [1:0] ALUCtrl, ImmSrc;
    input [31:0] ReadData;
    input [3:0] fp_ALUCtrl;
    output wire [31:0] PC;
    output wire [31:0] ALUResult;
    output wire [31:0] WriteData;
    output wire Zero;
    ////////////////////////////////////////////////////////////////////////////////////////////
    wire [31:0] RD1, RD2, WD3, PCTarget, PCPlus4, PCNext, ImmExt, TmpResult;
    wire [31:0] TmpResultORfp_ALUResult, TmpPC;
    wire [31:0] fp_RD1, fp_RD2, neg_fp_RD2, fp_WD3, fp_Op1, fp_Op2, fp_ALUResult, fp_ALUResultORReadData;
    wire fp_sw;
    wire [31:0] SrcB;
    ////////////////////////////////////////////////////////////////////////////////////////////
    assign neg_fp_RD2 = {~fp_RD2[31] , fp_RD2[30:0]};
    assign fp_sw = Instr[2];
    //Instantiate building blocks
    add_by_4 add_by_4 (.PC(PC), .PCPlus4(PCPlus4));
    ////////////////////////////////////////////////////////////////////////////////////////////
    add_to_ImmExt add_to_ImmExt (PC, ImmExt, TmpPC);
    ////////////////////////////////////////////////////////////////////////////////////////////
    PC pc (.clk(clk), .reset(reset), .PC(PC), .PCNext(PCNext));
    ////////////////////////////////////////////////////////////////////////////////////////////
    reg_file reg_file (.A1(Instr[19:15]), .A2(Instr[24:20]), .A3(Instr[11:7]),
                        .WE3(RegWrite), .RD1(RD1), .RD2(RD2), .WD3(WD3), .clk(clk));
    ////////////////////////////////////////////////////////////////////////////////////////////
```



```
    mux_2 i7 (fp_RD1, RD1, & fp_ALUCtrl, fp_Op1);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i9 (fp_RD2, neg_fp_RD2, add_sub, fp_Op2);
    ////////////////////////////////////////////////////////////////////////////////////////////
    fp_alu fp_alu (.A(fp_Op1), .B(fp_Op2), .ALUCtrl(fp_ALUCtrl), .ALUResult(fp_ALUResult));
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i0 (fp_ALUResult, ReadData, ALUSrc, fp_WD3);
    ////////////////////////////////////////////////////////////////////////////////////////////
    reg_file fp_reg_file (.A1(Instr[19:15]), .A2(Instr[24:20]), .A3(Instr[11:7]),
                        .WE3(fp_RegWrite), .RD1(fp_RD1), .RD2(fp_RD2), .WD3(fp_WD3), .clk(clk));
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i8 (RD2, fp_RD2, fp_sw, WriteData);
    ////////////////////////////////////////////////////////////////////////////////////////////
    imm_ext imm_ext (Instr[31:7], ImmSrc, ImmExt);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i1 (RD2, ImmExt, ALUSrc, SrcB);
    ////////////////////////////////////////////////////////////////////////////////////////////
    alu alu (.A(RD1), .B(SrcB), .ALUCtrl(ALUCtrl), .ALUResult(ALUResult), .Zero(Zero));
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i2 (ALUResult, ReadData, ResultSrc[0], TmpResult);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i3 (TmpResult, fp_ALUResult, ResultSrc[1], TmpResultORfp_ALUResult);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i4 (TmpResultORfp_ALUResult, PCPlus4, Jump, WD3);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i5 (TmpPC, ALUResult, (Jump & ALUSrc), PCTarget);
    ////////////////////////////////////////////////////////////////////////////////////////////
    mux_2 i6 (PCPlus4, PCTarget, PCSrc, PCNext);
    ////////////////////////////////////////////////////////////////////////////////////////////

endmodule
```

13)top:

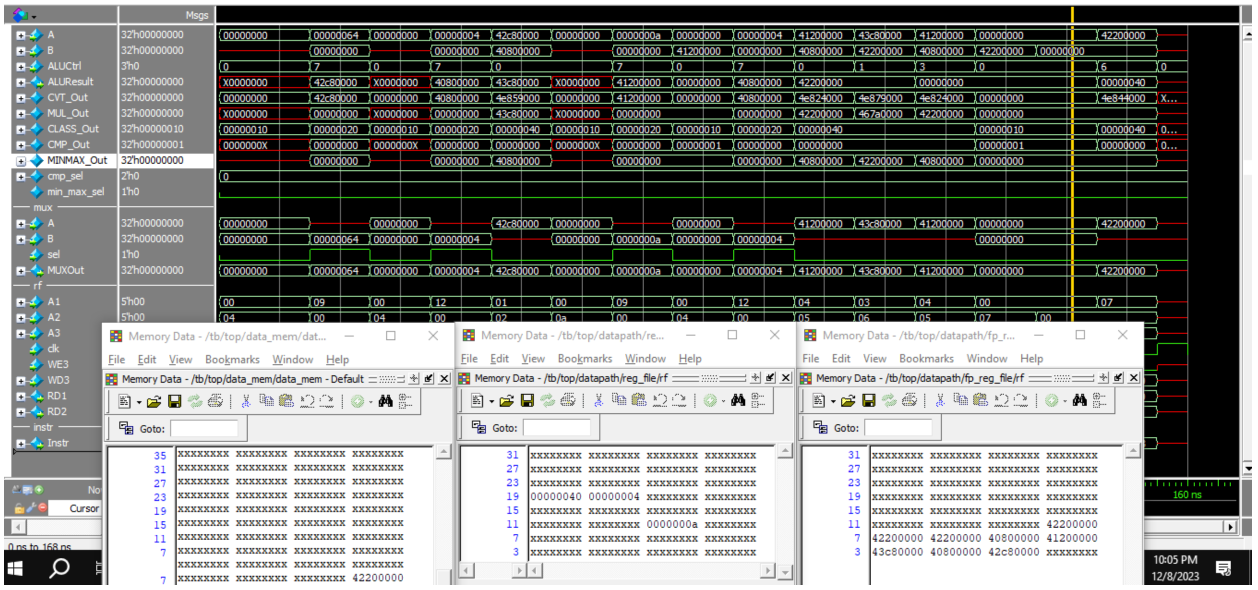


```
module top (
    clk, reset
);
    input clk, reset;
    ////////////////////////////////////////////////////////////////////////////////////////////
    wire [31:0] Instr, PC;
    wire [31:0] ALUResult, ReadData, WriteData;
    wire RegWrite, Zero, MemWrite, ALUSrc, PCSrc, Jump, fp_RegWrite, add_sub;
    wire [1:0] ImmSrc, ALUCtrl, ResultSrc;
    wire [3:0] fp_ALUCtrl;
    ////////////////////////////////////////////////////////////////////////////////////////////
    datapath datapath (clk, reset, RegWrite, ALUSrc, PCSrc, ResultSrc, Jump, fp_RegWrite, add_sub,
                        fp_ALUCtrl, Instr, ALUCtrl, ImmSrc, ReadData, PC, ALUResult, WriteData, Zero);
    ////////////////////////////////////////////////////////////////////////////////////////////
    ctrl_unit ctrl_unit (Instr[6:0], Instr[31:27], Instr[14:12], Instr[30], Zero, ImmSrc, ALUCtrl,
                        RegWrite, ALUSrc, MemWrite, ResultSrc, PCSrc, Jump, fp_RegWrite, add_sub, fp_ALUCtrl);
    ////////////////////////////////////////////////////////////////////////////////////////////
    instr_mem instr_mem (PC, Instr);
    ////////////////////////////////////////////////////////////////////////////////////////////
    data_mem data_mem (clk, MemWrite, WriteData, ALUResult, ReadData);
    ////////////////////////////////////////////////////////////////////////////////////////////
endmodule
```

Simulation results:

The Program:

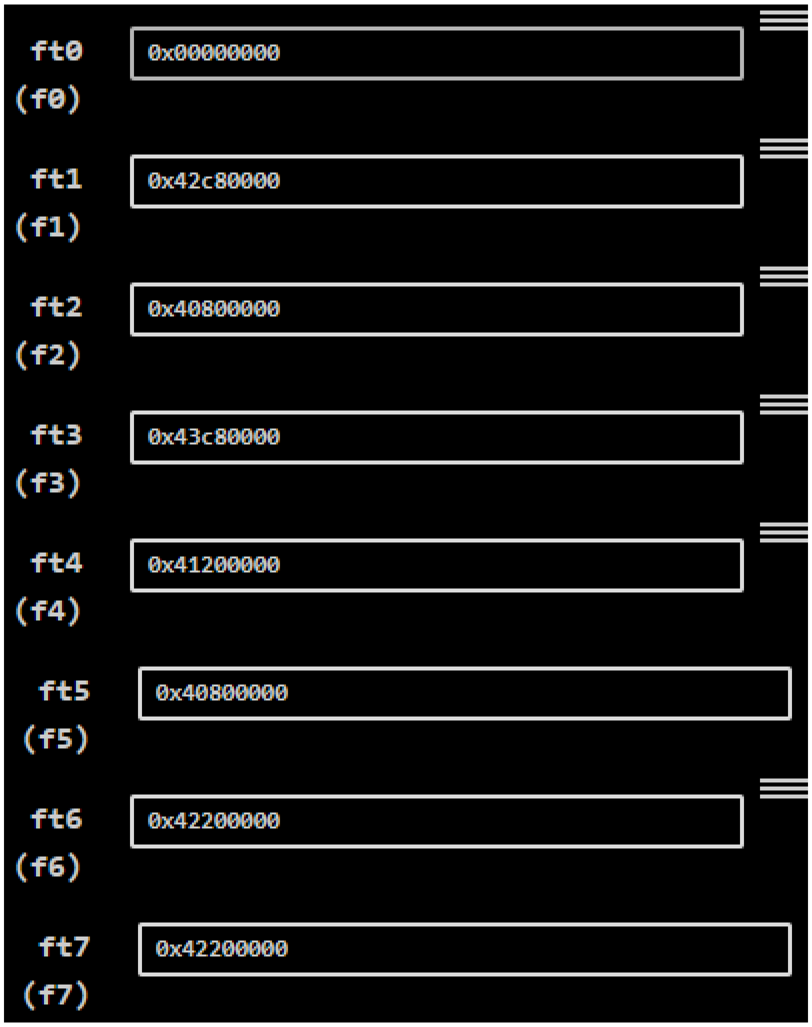
```
addi s1, x0, 100
fcvt.s.w f1, s1
addi s2, x0, 4
fcvt.s.w f2, s2
fmul.s f3,f1,f2      #f3 takes the value of 400
addi s1, x0, 10
fcvt.s.w f4,s1
addi s2, x0, 4
fcvt.s.w f5,s2
fmul.s f6,f4,f5      #f6 takes the value of 40
fmin.s f7,f3,f6      #f7 takes the value of f6 -> 40
fle.s s3,f4,f5       #s3 takes the value of 0 since f4 > f5
fsw   f7,84(x0)       #mem[0] = 40
flw   f8,84(x0)       #f8 takes the value of mem[0] -> 40
fclass.s s3,f7       #s3 takes the value of 0x40 since the value stored in f7 is
+ve normal
```



```
Memory Data - /tb/top/data_mem/data_mem - Default
35 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
31 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
27 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
23 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
19 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
15 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
 7 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
 7 xxxxxxxx xxxxxxxx xxxxxxxx 42200000
```

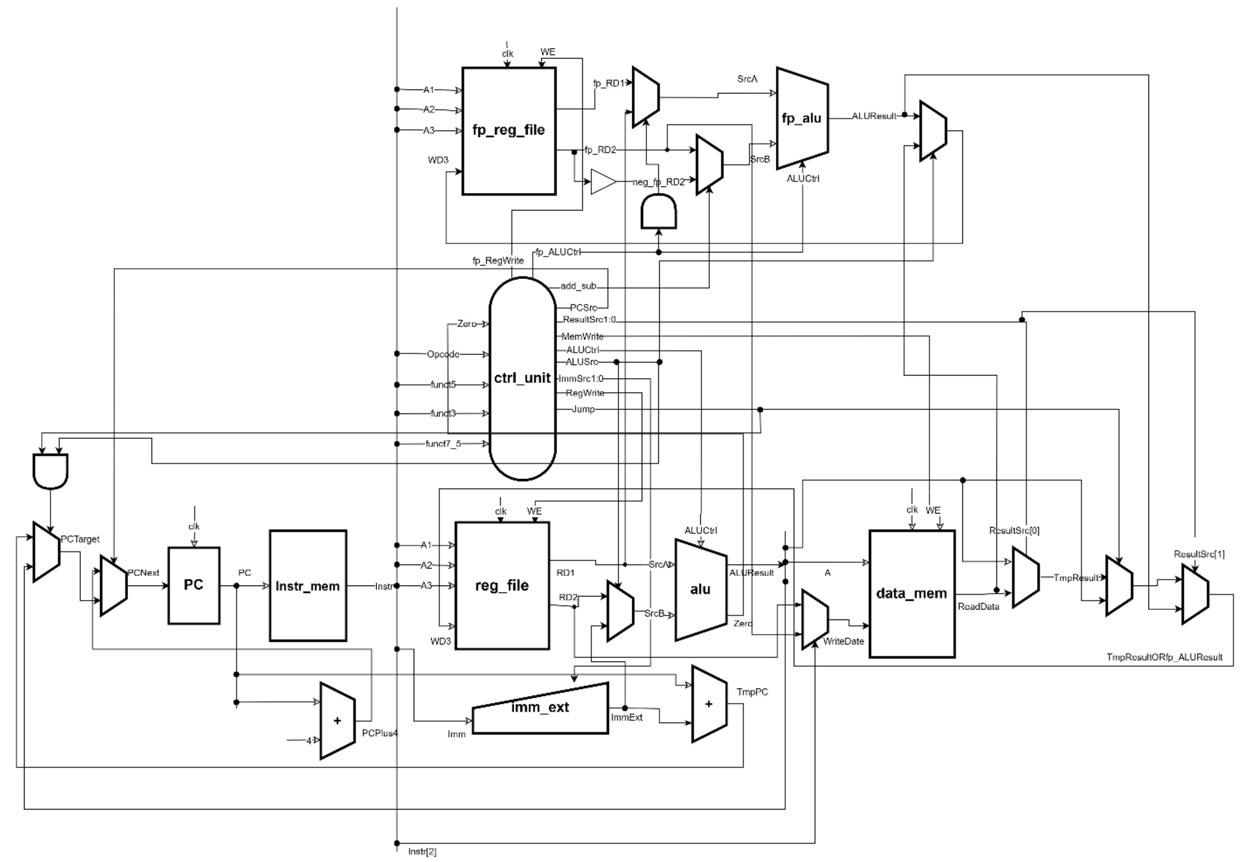
```
Memory Data - /tb/top/datapath/reg_file/rf
31 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
27 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
23 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
19 00000040 00000004 xxxxxxxx xxxxxxxx
15 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11 xxxxxxxx xxxxxxxx 0000000a xxxxxxxx
 7 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
 3 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

```
Memory Data - /tb/top/datapath/fp_reg_file/rf
31 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
27 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
23 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
19 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
15 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11 xxxxxxxx xxxxxxxx xxxxxxxx 42200000
 7 42200000 42200000 40800000 41200000
 3 43c80000 40800000 42c80000 xxxxxxxx
```



```
ft0   0x00000000
(f0)
ft1   0x42c80000
(f1)
ft2   0x40800000
(f2)
ft3   0x43c80000
(f3)
ft4   0x41200000
(f4)
ft5   0x40800000
(f5)
ft6   0x42200000
(f6)
ft7   0x42200000
(f7)
```

Architecture:



clk
WE
A1
A2
A3
fp_reg_file
WD3
fp_RD1
fp_RD2
neg_fp_RD2
SrcA
fp_alu
ALUResult
SrcB
ALUCtrl
fp_RegWrite
fp_ALUCtrl
add_sub
PCSrc
ResultSrc1:0
MemWrite
ALUCtrl
ALUSrc
ImmSrc1:0
RegWrite
Jump
Zero
Opcode
funct5
funct3
funct7_5
ctrl_unit
PCTarget
PCNext
clk
PC
PC
Instr_mem
Instr
clk
WE
A1
A2
A3
reg_file
WD3
RD1
RD2
SrcA
SrcB
ALUCtrl
alu
ALUResult
Zero
A
clk
WE
data_mem
ReadData
WriteDate
ResultSrc[0]
TmpResult
ResultSrc[1]
TmpResultORfp_ALUResult
imm_ext
Imm
ImmExt
+
TmpPC
+
4
PCPlus4
Instr[2]