```verilog
module proc(DIN, Resetn, Clock, Run, Done);
    input [15:0] DIN;
    input Resetn, Clock, Run;
    output Done;

    reg [15:0] BusWires;
    reg [3:0] Sel; // BusWires selector
    reg [0:7] Rin;
    reg [15:0] Sum;
    reg IRin, Done, Ain, Gin, AddSub;
    reg [2:0] Tstep_Q, Tstep_D;
    wire [2:0] III, rX, rY; // instruction opcode and register operands
    wire [0:7] Xreg;
    wire [15:0] R0, R1, R2, R3, R4, R5, R6, R7, A;
    wire [15:0] G;
    wire [15:0] IR;
    wire IMM;

    assign III = IR[15:13];
    assign IMM = IR[12];
    assign rX = IR[11:9];
    assign rY = IR[2:0];
    dec3to8 decX (rX, Xreg);

    parameter T0 = 3'b000, T1 = 3'b001, T2 = 3'b010, T3 = 3'b011;

    // Control FSM state table
    always @(Tstep_Q, Run, Done)
        case (Tstep_Q)
            T0: // data is loaded into IR in this time step
                if (~Run) Tstep_D = T0;
                else Tstep_D = T1;
            T1: // some instructions end after this time step
                if (Done) Tstep_D = T0;
                else Tstep_D = T2;
            T2: // always go to T3 after this
                Tstep_D = T3;
            T3: // instructions end after this time step
                Tstep_D = T0;
            default: Tstep_D = 3'bxxx;
       endcase

    /* OPCODE format: III M XXX DDDDDDDDD, where
    *     III = instruction, M = Immediate, XXX = rX.
    *     If M = 0, DDDDDDDDD = 000000YYY = rY
    *     If M = 1, DDDDDDDDD = #D is the immediate operand
    *
    *  III M  Instruction   Description
    *  --- -  -----------   -----------
    *  000 0: mv   rX,rY    rX <- rY
    *  000 1: mv   rX,#D    rX <- D (0 extended)
    *  001 1: mvt  rX,#D    rX <- D << 8
    *  010 0: add  rX,rY    rX <- rX + rY
    *  010 1: add  rX,#D    rX <- rX + D
    *  011 0: sub  rX,rY    rX <- rX - rY
    *  011 1: sub  rX,#D    rX <- rX - D */
    parameter mv = 3'b000, mvt = 3'b001, add = 3'b010, sub = 3'b011;
    // selectors for the BusWires multiplexer
    parameter Sel_R0 = 4'b0000, Sel_R1 = 4'b0001, Sel_R2 = 4'b0010, Sel_R3 = 4'b0011,
        Sel_R4 = 4'b0100, Sel_R5 = 4'b0101, Sel_R6 = 4'b0110, Sel_R7 = 4'b0111, Sel_G = 4'b1000,
            Sel_D /* immediate data */  = 4'b1001, Sel_D8 /* immediate data
        << 8 */ = 4'b1010;
        // Control FSM outputs
        always @(*) begin
            // default values for control signals
            Done = 1'b0; Ain = 1'b0; Gin = 1'b0; AddSub = 1'b0; IRin = 1'b0;
         Rin = 8'b0;
            Sel = 4'bxxxx;
            case (Tstep_Q)
                T0: // store instruction on DIN in IR
                    IRin = 1'b1;
                T1: // define signals in T1
                    case (III)
                        mv: begin
                            if (IMM == 1'b0) Sel = rY;
                            else Sel = Sel_D;
                            Rin = Xreg;
                            Done = 1;
                        end
                        mvt: begin
                            Sel = Sel_D8;
                            Rin = Xreg;
                            Done = 1;
                        end
                        add, sub: begin
                            Sel = rX;
                            Ain = 1'b1;
                        end
                        default: ;
                    endcase
                T2: // define signals T2
                    case (III)
                        add: begin
                            if (IMM == 1'b0) Sel = rY;
                            else Sel = Sel_D;
                            AddSub = 1'b0;
                            Gin = 1'b1;
                        end
                        sub: begin
                            if (IMM == 1'b0) Sel = rY;
                            else Sel = Sel_D;
                            AddSub = 1'b1;
                            Gin = 1'b1;
                        end
                        default: ;
                    endcase
                T3: // define T3
                    case (III)
                        add, sub: begin
                            Sel = Sel_G;
                            Rin = Xreg;
                            Done = 1;
                        end
                        default: ;
                    endcase
                default: ;
            endcase
        end

        // Control FSM flip-flops
        always @(posedge Clock)
            if (!Resetn)
```

```
                Tstep_Q <= T0;
            else
                Tstep_Q <= Tstep_D;

        regn reg_0 (BusWires, Rin[0], Clock, R0);
        regn reg_1 (BusWires, Rin[1], Clock, R1);
        regn reg_2 (BusWires, Rin[2], Clock, R2);
        regn reg_3 (BusWires, Rin[3], Clock, R3);
        regn reg_4 (BusWires, Rin[4], Clock, R4);
        regn reg_5 (BusWires, Rin[5], Clock, R5);
        regn reg_6 (BusWires, Rin[6], Clock, R6);
        regn reg_7 (BusWires, Rin[7], Clock, R7);

        regn reg_A (BusWires, Ain, Clock, A);
        regn reg_IR (DIN, IRin, Clock, IR);

        // alu
        always @(*)
            if (!AddSub)
                Sum = A + BusWires;
            else
                Sum = A - BusWires;
        regn reg_G (Sum, Gin, Clock, G);

        // define the internal processor bus
        always @(*)
            case (Sel)
                Sel_R0: BusWires = R0;
                Sel_R1: BusWires = R1;
                Sel_R2: BusWires = R2;
                Sel_R3: BusWires = R3;
                Sel_R4: BusWires = R4;
                Sel_R5: BusWires = R5;
                Sel_R6: BusWires = R6;
                Sel_R7: BusWires = R7;
                Sel_G: BusWires = G;
                Sel_D: BusWires = {7'b0000000, IR[8:0]};
                Sel_D8: BusWires = {IR[7:0], 8'b00000000};
                default: BusWires = 16'bxxxxxxxxxxxxxxxx;
            endcase
    endmodule

    module dec3to8(W, Y);
        input [2:0] W;
        output [0:7] Y;
        reg [0:7] Y;

        always @(*)
            case (W)
                3'b000: Y = 8'b10000000;
                3'b001: Y = 8'b01000000;
                3'b010: Y = 8'b00100000;
                3'b011: Y = 8'b00010000;
                3'b100: Y = 8'b00001000;
                3'b101: Y = 8'b00000100;
                3'b110: Y = 8'b00000010;
                3'b111: Y = 8'b00000001;
            endcase
    endmodule

    module regn(R, Rin, Clock, Q);
        parameter n = 16;
        input [n-1:0] R;
        input Rin, Clock;
        output [n-1:0] Q;
        reg [n-1:0] Q;

        always @(posedge Clock)
            if (Rin)
                Q <= R;
    endmodule

```