```
module proc(DIN, Resetn, Clock, Run, DOUT, ADDR, W, Done);
    input [15:0] DIN;
    input Resetn, Clock, Run;
    output wire [15:0] DOUT;
    output wire [15:0] ADDR;
    output wire W;
    output Done;

    parameter T0 = 3'b000, T1 = 3'b001, T2 = 3'b010, T3 = 3'b011, T4 = 3'b100, T5 = 3'b101;
    reg [15:0] BusWires;
    reg [0:7] Rin, Rout;
    reg [15:0] Sum;
    reg IRin, ADDRin, Done, DINout, DOUTin, Ain, Gin, Gout, AddSub;
    reg [2:0] Tstep_Q, Tstep_D;
    reg z,c; // Zero and carry flags
    reg set_z,set_c;
    wire [2:0] I;
    wire [0:7] Xreg, Yreg;
    wire [15:0] R0, R1, R2, R3, R4, R5, R6, R7 /* pc */, A;
    wire [15:0] G;
    wire [1:9] IR;
    wire [1:10] Sel; // bus selector
    reg pc_inc, W_D;

    assign I = IR[1:3];
    dec3to8 decX (IR[4:6], 1'b1, Xreg);
    dec3to8 decY (IR[7:9], 1'b1, Yreg);

    // Control FSM state table
    always @(Tstep_Q, Run, Done)
    begin
        case (Tstep_Q)
            T0: // instruction fetch
                if (~Run) Tstep_D = T0;
                else Tstep_D = T1;
            T1: // wait cycle for synchronous memory
                Tstep_D = T2;
            T2: // this time step stores the instruction word in IR
                Tstep_D = T3;
            T3: // some instructions end after this time step
                if (Done) Tstep_D = T0;
                else Tstep_D = T4;
            T4: // always go to T5 after this
                Tstep_D = T5;
            T5: // instructions end after this time step
                Tstep_D = T0;
        endcase
    end

    /* Instruction Table
    *   000: mv     Rx,Ry       : Rx <- Ry
    *   001: mvi        Rx,D        : Rx <- D
    *   010: add        Rx,Ry       : Rx <- Rx + Ry
    *   011: sub        Rx,Ry       : Rx <- Rx - Ry
    *   100: ld     Rx,[Ry] : Rx <- [Ry]
    *   101: st     Rx,[Ry] : [Ry] <- Rx
    *   110: mvnz   Rx,Ry       : Rx <- Ry, if G != 0
    *   111: mvnc   Rx,Ry       : Rx <- Ry, if carry-out != 1
    *   OPCODE format: III XXX YYY , where
    *   III = instruction, XXX = Rx, and YYY = Ry. For mvi,
    *   a second word of data is read in the following clock cycle
    */
    // R7 is the program counter
    parameter
        mv = 3'b000, mvi = 3'b001, add = 3'b010, sub = 3'b011, ld = 3'b100, st = 3'b101, mvnz = 3'b110, mvnc = 3'b111;
    // Control FSM outputs
    always @(*)
```

```
68      begin
69          Done = 1'b0; Ain = 1'b0; Gin = 1'b0; Gout = 1'b0; AddSub = 1'b0;
70          IRin = 1'b0; DINout = 1'b0; DOUTin = 1'b0; ADDRin = 1'b0; W_D = 1'b0;
71          Rin = 8'b0; Rout = 8'b0; pc_inc = 1'b0; set_z = 1'b0; set_c = 1'b0;
72          case (Tstep_Q)
73              T0: // fetch the instruction
74                  begin
75                      Rout = 8'b00000001; // R7 is program counter (pc)
76                      ADDRin = 1'b1;
77                      pc_inc = Run; // to increment pc
78                  end
79              T1: // wait cycle for synchronous memory
80                  // in case the instruction turns out to be mvi, read memory
81                  begin
82                      Rout = 8'b00000001; // R7 is program counter (pc)
83                      ADDRin = 1'b1;
84                  end
85              T2: // store instruction on DIN in IR
86                  begin
87                      IRin = 1'b1;
88                  end
89              T3:   //define signals in T3
90                  case (I)
91                      mv: // mv Rx,Ry
92                      begin
93                          Rout = Yreg;
94                          Rin = Xreg;
95                          set_z = 1'b1;
96                          Done = 1'b1;
97                      end
98                      mvi: // mvi Rx,#D
99                      begin
100                         // data is available now on DIN
101                         DINout = 1'b1;
102                         Rin = Xreg;
103                         pc_inc = 1'b1;
104                         Done = 1'b1;
105                     end
106                     add, sub: //add, sub
107                     begin
108                         Rout = Xreg;
109                         Ain = 1'b1;
110                     end
111                     ld: // ld Rx,[Ry]
112                     begin
113                         Rout = Yreg;
114                         ADDRin = 1'b1;
115                     end
116                     st: // st Rx,[Ry]
117                     begin
118                         Rout = Yreg;
119                         ADDRin = 1'b1;
120                     end
121                     mvnz: // mvnz rX,rY
122                     begin
123                         if(!z)
124                         begin
125                             Rout = Yreg;
126                             Rin = Xreg;
127                             Done = 1'b1;
128                         end
129                     end
130                     mvnc: // mvnc rX,rY
131                     begin
132                         if(!c)
133                         begin
134                             Rout = Yreg;
135                             Rin = Xreg;
136                             Done = 1'b1;
```

```
                        end
                    end
                    default: ;
                endcase
            T4:   //define signals T4
                case (I)
                    add: // add
                    begin
                        Rout = Yreg;
                        Gin = 1'b1;
                        set_c = 1'b1;
                    end
                    sub: // sub
                    begin
                        Rout = Yreg;
                        AddSub = 1'b1;
                        Gin = 1'b1;
                        set_c = 1'b1;
                    end
                    ld: // ld Rx,[Ry]
                        ; // wait cycle for synchronous memory
                    st: // st Rx,[Ry]
                    begin
                        Rout = Xreg;
                        DOUTin = 1'b1;
                        W_D = 1'b1;
                    end
                    default: ;
                endcase
            T5:   //define T5
                case (I)
                    add, sub: //add, sub
                    begin
                        Gout = 1'b1;
                        Rin = Xreg;
                        set_z = 1'b1;
                        Done = 1'b1;
                    end
                    ld: // ld Rx,[Ry]
                    begin
                        DINout = 1'b1;
                        Rin = Xreg;
                        Done = 1'b1;
                    end
                    st: // st Rx,[Ry]
                    begin
                        Done = 1'b1;
                    end
                    default: ;
                endcase
            default: ;
        endcase
    end

    // Control FSM flip-flops
    always @(posedge Clock)
        if (!Resetn)
            Tstep_Q <= T0;
        else
            Tstep_Q <= Tstep_D;

    // Registers for z and c flags
    always @(posedge Clock)
    begin
        // z flag
        if(!Resetn) // Active low reset
            z <= 1'b1;
        else if(set_z) // The bus wires will either be G or rY
            z <= (BusWires==0);
```

```
206
207         // c flag
208         // If a carryout is generated then the sum is less than
209         // either value due to the concatenated carry out digit
210         // A borrow is required when A-B goes negative
211         if(!Resetn) // Active low reset
212             c <= 1'b1;
213         else if(set_c & !AddSub) // Addition
214             c <= (A > A+BusWires);
215         else if(set_c & AddSub) // Subtraction
216             c <= (A < BusWires);
217     end
218
219     regn reg_0 (BusWires, Rin[0], Clock, R0);
220     regn reg_1 (BusWires, Rin[1], Clock, R1);
221     regn reg_2 (BusWires, Rin[2], Clock, R2);
222     regn reg_3 (BusWires, Rin[3], Clock, R3);
223     regn reg_4 (BusWires, Rin[4], Clock, R4);
224     regn reg_5 (BusWires, Rin[5], Clock, R5);
225     regn reg_6 (BusWires, Rin[6], Clock, R6);
226
227     // R7 is program counter
228     // module pc_count(R, Resetn, Clock, E, L, Q);
229     pc_count pc (BusWires, Resetn, Clock, pc_inc, Rin[7], R7);
230
231     regn reg_A (BusWires, Ain, Clock, A);
232     regn reg_DOUT (BusWires, DOUTin, Clock, DOUT);
233     regn reg_ADDR (BusWires, ADDRin, Clock, ADDR);
234     regn #(.n(9)) reg_IR (DIN[8:0], IRin, Clock, IR);
235
236     flipflop reg_W (W_D, Resetn, Clock, W);
237
238     //  alu
239     always @(AddSub or A or BusWires)
240         begin
241         if (!AddSub)
242             Sum = A + BusWires;
243        else
244             Sum = A - BusWires;
245         end
246
247     regn #(.n(16)) reg_G (Sum, Gin, Clock, G);
248
249     // define the internal processor bus
250     assign Sel = {Rout, Gout, DINout};
251
252     always @(*)
253     begin
254         if (Sel == 10'b1000000000)
255             BusWires = R0;
256     else if (Sel == 10'b0100000000)
257             BusWires = R1;
258         else if (Sel == 10'b0010000000)
259             BusWires = R2;
260         else if (Sel == 10'b0001000000)
261             BusWires = R3;
262         else if (Sel == 10'b0000100000)
263             BusWires = R4;
264         else if (Sel == 10'b0000010000)
265             BusWires = R5;
266         else if (Sel == 10'b0000001000)
267             BusWires = R6;
268         else if (Sel == 10'b0000000100)
269             BusWires = R7;
270         else if (Sel == 10'b0000000010)
271             BusWires = G;
272     else BusWires = DIN;
273     end
274 endmodule
```

```
275
276
277  module pc_count(R, Resetn, Clock, E, L, Q);
278      input [15:0] R;
279      input Resetn, Clock, E, L;
280      output [15:0] Q;
281      reg [15:0] Q;
282
283      always @(posedge Clock)
284          if (!Resetn)
285              Q <= 9'b0;
286          else if (L)
287              Q <= R;
288          else if (E)
289              Q <= Q + 1'b1;
290  endmodule
291
292  module dec3to8(W, En, Y);
293      input [2:0] W;
294      input En;
295      output [0:7] Y;
296      reg [0:7] Y;
297
298      always @(W or En)
299      begin
300          if (En == 1)
301              case (W)
302                  3'b000: Y = 8'b10000000;
303             3'b001: Y = 8'b01000000;
304                  3'b010: Y = 8'b00100000;
305                  3'b011: Y = 8'b00010000;
306                  3'b100: Y = 8'b00001000;
307                  3'b101: Y = 8'b00000100;
308                  3'b110: Y = 8'b00000010;
309                  3'b111: Y = 8'b00000001;
310              endcase
311          else
312              Y = 8'b00000000;
313      end
314  endmodule
315
316  module regn(R, Rin, Clock, Q);
317      parameter n = 16;
318      input [n-1:0] R;
319      input Rin, Clock;
320      output [n-1:0] Q;
321      reg [n-1:0] Q;
322
323      always @(posedge Clock)
324          if (Rin)
325              Q <= R;
326  endmodule
327
```