PART 3

Code:

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

    reg [15:0] BusWires;
    reg [3:0] Sel; // BusWires selector
    reg [0:7] Rin;
    reg [15:0] Sum;
    reg IRin, ADDRin, Done, DOUTin, Ain, Gin, AddSub, ALUand;
    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, PC, A;
    wire [15:0] G;
    wire [15:0] IR;
    reg pc_inc, W_D;
    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, T4 = 3'b100, T5 = 3'b101;

    // Control FSM state table
    always @(Tstep_Q, Run, Done)
        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;
            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
   *  100 0: ld   rX,[rY]  rX <- [rY]
   *  101 0: st   rX,[rY]  [rY] <- rX
   *  110 0: and  rX,rY    rX <- rX & rY
   *  110 1: and  rX,#D    rX <- rX & D */
   parameter mv = 3'b000, mvt = 3'b001, add = 3'b010, sub = 3'b011, ld = 3'b100, st = 3'b101,
             and_ = 3'b110;
   // 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_PC = 4'b0111, Sel_G =
4'b1000,
      Sel_D /* immediate data */ = 4'b1001, Sel_D8 /* immediate data << 8 */ = 4'b1010,
      Sel_DIN /* data-in from memory */ = 4'b1011;
   // 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; Sel = 4'bxxxx;
      DOUTin = 1'b0; ADDRin = 1'b0; W_D = 1'b0; Rin = 8'b0; pc_inc = 1'b0; ALUand = 1'b0;
      case (Tstep_Q)
         T0: begin // fetch the instruction
            Sel = Sel_PC; // put pc onto the internal bus
            ADDRin = 1'b1;
            pc_inc = Run; // to increment pc
         end
         T1: // wait cycle for synchronous memory
            ;
         T2: // store instruction on DIN in IR
            IRin = 1'b1;
         T3: // define signals in T1
            case (III)
               mv: begin
                  if (!IMM) Sel = rY;   // mv rX, rY
                  else Sel = Sel_D;     // mv rX, #D
                  Rin = Xreg;
                  Done = 1'b1;
               end
               mvt: begin
                  // ... your code goes here
                                                        Sel = Sel_D8;
```

```
                                                    Rin = Xreg;
                                                    Done = 1'b1;

        end
        add, sub, and_: begin
            // ... your code goes here

                                                    Sel = rX;
                                                    Ain = 1'b1;


        end
        ld, st: begin
            // ... your code goes here

                                                    Sel = rY;
                                                    ADDRin = 1'b1;

        end
        default: ;
    endcase
T4: // define signals T2
    case (III)
        add: begin
            // ... your code goes here

                                                    if (IMM == 1'b1)
                                                        begin

                                                            Sel = Sel_D;
                                                        end
                                                    else if (IMM == 0)
                                                        begin
                                                            Sel = rY;
                                                        end

                                                    AddSub = 1'b0;
                                                    Gin = 1'b1;
        end
        sub: begin
            // ... your code goes here
                                                    if (IMM == 1'b1)
                                                        begin
                                                            Sel = Sel_D;
                                                        end
                                                    else if (IMM == 0)
                                                        begin
                                                            Sel = rY;
                                                        end

                                                    AddSub = 1'b1;
                                                    Gin = 1'b1;
```

```
        end
        and_: begin
            // ... your code goes here

                                                    if (IMM == 1'b1)
                                                        begin
                                                            Sel = Sel_D;
                                                        end
                                                    else if (IMM == 0)
                                                        begin
                                                            Sel = rY;
                                                        end

                                                    ALUand = 1'b1;
                                                    Gin = 1'b1;

        end
        ld: // wait cycle for synchronous memory
            ;
        st: begin
            // ... your code goes here
                                                    Sel = rX;

                                                    W_D = 1'b1;
                                                    DOUTin = 1'b1;

        end
        default: ;
    endcase
T5: // define T3
    case (III)
        add, sub, and_: begin
            // ... your code goes here

                                                    Sel = Sel_G;
                                                    Rin = Xreg;
                                                    Done = 1'b1;

        end
        ld: begin
            // ... your code goes here
                                                    Sel = Sel_DIN;
                                                    Rin = Xreg;
                                                    Done = 1'b1;

        end

        st: // wait cycle for synhronous memory
            // ... your code goes here

                                                    Done = 1'b1;
```

```
                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);

// R7 is program counter
// module pc_count(R, Resetn, Clock, E, L, Q);
pc_count pc (BusWires, Resetn, Clock, pc_inc, Rin[7], PC);

regn reg_A (BusWires, Ain, Clock, A);
regn reg_DOUT (BusWires, DOUTin, Clock, DOUT);
regn reg_ADDR (BusWires, ADDRin, Clock, ADDR);
regn reg_IR (DIN, IRin, Clock, IR);

flipflop reg_W (W_D, Resetn, Clock, W);

// alu
always @(*)
    if (!ALUand)
        if (!AddSub)
            Sum = A + BusWires;
        else
            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_PC: BusWires = PC;
            Sel_G: BusWires = G;
            Sel_D: BusWires = {7'b0000000, IR[8:0]};
            Sel_D8: BusWires = {IR[7:0], 8'b00000000};
            default: BusWires = DIN;
        endcase
endmodule

module pc_count(R, Resetn, Clock, E, L, Q);
    input [15:0] R;
    input Resetn, Clock, E, L;
    output [15:0] Q;
    reg [15:0] Q;

    always @(posedge Clock)
        if (!Resetn)
            Q <= 9'b0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1'b1;
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
```

PART 4

Code:

```
// Data written to registers R0 to R5 are sent to the H digits
module seg7 (Data, Addr, Sel, Resetn, Clock, H5, H4, H3, H2, H1, H0);
    input [6:0] Data;
    input [2:0] Addr;
    input Sel, Resetn, Clock;
    output [6:0] H5, H4, H3, H2, H1, H0;

    wire [6:0] nData;
    assign nData = ~Data;

    regne reg_R0 (nData, Clock, Resetn, Sel & (Addr == 3'b000), H0);
            regne reg_R1 (nData, Clock, Resetn, Sel & (Addr == 3'b001), H1);
            regne reg_R2 (nData, Clock, Resetn, Sel & (Addr == 3'b010), H2);
            regne reg_R3 (nData, Clock, Resetn, Sel & (Addr == 3'b011), H3);
            regne reg_R4 (nData, Clock, Resetn, Sel & (Addr == 3'b100), H4);
            regne reg_R5 (nData, Clock, Resetn, Sel & (Addr == 3'b101), H5);
            // ... add code here
endmodule

module regne (R, Clock, Resetn, E, Q);
    parameter n = 7;
    input [n-1:0] R;
    input Clock, Resetn, E;
    output [n-1:0] Q;
    reg [n-1:0] Q;

    always @(posedge Clock)
        if (Resetn == 0)
            Q <= {n{1'b1}};  // turn OFF all segments on reset
        else if (E)
            Q <= R;
endmodule
```

PART 5

Code:

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

    reg [15:0] BusWires;
    reg [3:0] Sel; // BusWires selector
```

```
reg [0:7] Rin;
reg [16:0] Sum;
reg IRin, ADDRin, Done, DOUTin, Ain, Gin, AddSub, ALUand;
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, PC, A, CondIn;
wire [15:0] G;
wire [15:0] IR;
reg pc_inc, W_D;
wire IMM, c;
        reg z;

        assign c = Sum[16];

        always @(*)
        if(G == 0)
                z = 1'b1;
        else
                z = 1'b0;

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, T4 = 3'b100, T5 = 3'b101;

// Control FSM state table
always @(Tstep_Q, Run, Done)
   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;
      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
   *  100 0: ld   rX,[rY]  rX <- [rY]
   *  101 0: st   rX,[rY]  [rY] <- rX
   *  110 0: and  rX,rY    rX <- rX & rY
   *  110 1: and  rX,#D    rX <- rX & D */
   parameter mv = 3'b000, mvt = 3'b001, add = 3'b010, sub = 3'b011, ld = 3'b100, st = 3'b101,
             and_ = 3'b110, b = 3'b111, eq = 3'b001, ne = 3'b010, cc = 3'b011, cs = 3'b100;
   // 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_PC = 4'b0111, Sel_G =
4'b1000,
      Sel_D /* immediate data */ = 4'b1001, Sel_D8 /* immediate data << 8 */ = 4'b1010,
      Sel_DIN /* data-in from memory */ = 4'b1011, Sel_COND = 4'b1100;
   // 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; Sel = 4'bxxxx;
      DOUTin = 1'b0; ADDRin = 1'b0; W_D = 1'b0; Rin = 8'b0; pc_inc = 1'b0; ALUand = 1'b0;
      case (Tstep_Q)
         T0: begin // fetch the instruction
            Sel = Sel_PC; // put pc onto the internal bus
            ADDRin = 1'b1;
            pc_inc = Run; // to increment pc
         end
         T1: // wait cycle for synchronous memory
            ;
         T2: // store instruction on DIN in IR
            IRin = 1'b1;
         T3: // define signals in T1
            case (III)
               mv: begin
                  if (!IMM) Sel = rY;   // mv rX, rY
                  else Sel = Sel_D;     // mv rX, #D
                  Rin = Xreg;
                  Done = 1'b1;
               end
               mvt: begin
                  // ... your code goes here
                                                        Sel = Sel_D8;
                                                        Rin = Xreg;
                                                        Done = 1'b1;

               end
```

```
add, sub, and_: begin
   // ... your code goes here

                                        Sel = rX;
                                        Ain = 1'b1;


end
ld, st: begin
   // ... your code goes here

                                        Sel = rY;
                                        ADDRin = 1'b1;

end
                                b: begin
                                        case(rX)
                                        mv: begin

                                        Sel = Sel_D;
                                        Rin[7] = 1'b1;
                                        Done = 1'b1;
                                        end

                                        eq: begin

                                        if(z == 1'b1)
                                        begin
                                                Sel = Sel_D;
                                                Rin[7] = 1'b1;
                                                Done = 1'b1;
                                        end

                                        end
                                        ne: begin

                                        if(z == 1'b0)
                                        begin
                                                Sel = Sel_D;
                                                Rin[7] = 1'b1;
                                                Done = 1'b1;
                                        end

                                        end
                                        cc: begin

                                        if(c == 1'b0)
                                        begin
                                                Sel = Sel_D;
                                                Rin[7] = 1'b1;
                                                Done = 1'b1;
                                        end
```

```
                                        end
                                        cs: begin

                                        if(c == 1'b1)
                                        begin
                                                Sel = Sel_D;
                                                Rin[7] = 1'b1;
                                                Done = 1'b1;
                                        end

                                        end

                                endcase
                        end

        default: ;
    endcase
T4: // define signals T2
    case (III)
        add: begin
            // ... your code goes here

                                        if (IMM == 1'b1)
                                                begin

                                                        Sel = Sel_D;
                                                end
                                        else if (IMM == 0)
                                                begin
                                                        Sel = rY;
                                                end

                                        AddSub = 1'b0;
                                        Gin = 1'b1;

        end
        sub: begin
            // ... your code goes here
                                        if (IMM == 1'b1)
                                                begin
                                                        Sel = Sel_D;
                                                end
                                        else if (IMM == 0)
                                                begin
                                                        Sel = rY;
                                                end

                                        AddSub = 1'b1;
                                        Gin = 1'b1;

        end
        and_: begin
            // ... your code goes here
```

```
                                        if (IMM == 1'b1)
                                            begin
                                                Sel = Sel_D;
                                            end
                                        else if (IMM == 0)
                                            begin
                                                Sel = rY;
                                            end

                                        ALUand = 1'b1;
                                        Gin = 1'b1;

        end
        ld: // wait cycle for synchronous memory
            ;
        st: begin
            // ... your code goes here
                                        Sel = rX;

                                        W_D = 1'b1;
                                        DOUTin = 1'b1;

        end
        default: ;
    endcase
T5: // define T3
    case (III)
        add, sub, and_: begin
            // ... your code goes here

                                        Sel = Sel_G;
                                        Rin = Xreg;
                                        Done = 1'b1;

        end
        ld: begin
            // ... your code goes here
                                        Sel = Sel_DIN;
                                        Rin = Xreg;
                                        Done = 1'b1;

        end

        st: // wait cycle for synhronous memory
            // ... your code goes here

                                        Done = 1'b1;


        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);

// R7 is program counter
// module pc_count(R, Resetn, Clock, E, L, Q);
pc_count pc (BusWires, Resetn, Clock, pc_inc, Rin[7], PC);

regn reg_A (BusWires, Ain, Clock, A);
regn reg_DOUT (BusWires, DOUTin, Clock, DOUT);
regn reg_ADDR (BusWires, ADDRin, Clock, ADDR);
regn reg_IR (DIN, IRin, Clock, IR);

flipflop reg_W (W_D, Resetn, Clock, W);

// alu
always @(*)
    if (!ALUand)
        if (!AddSub)
          Sum = A + BusWires;
        else
          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_PC: BusWires = PC;
```

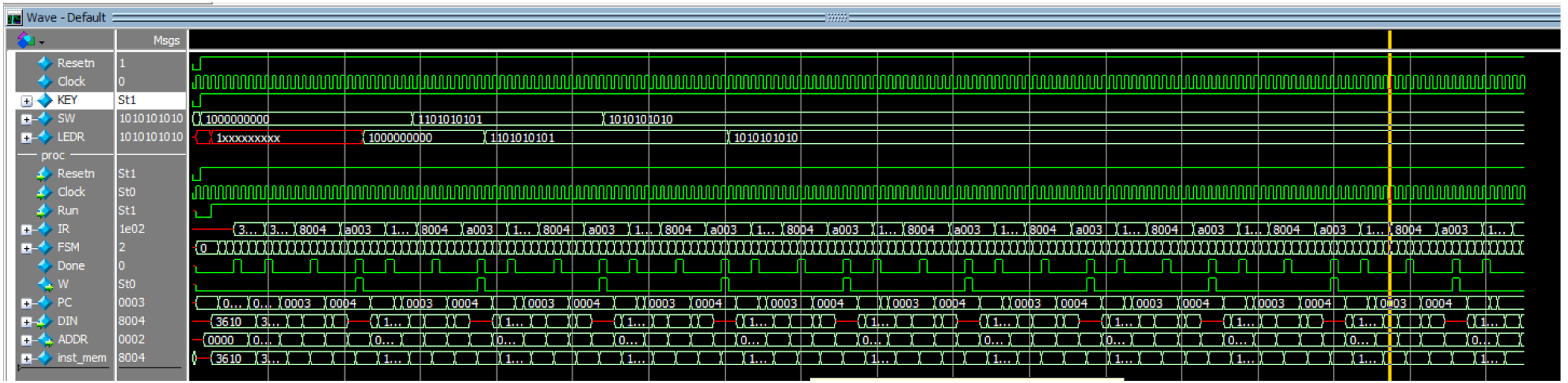
```
            Sel_G: BusWires = G;
            Sel_D: BusWires = {7'b0000000, IR[8:0]};
            Sel_D8: BusWires = {IR[7:0], 8'b00000000};
                                    Sel_COND: BusWires = CondIn;
            default: BusWires = DIN;
        endcase
endmodule

module pc_count(R, Resetn, Clock, E, L, Q);
    input [15:0] R;
    input Resetn, Clock, E, L;
    output [15:0] Q;
    reg [15:0] Q;

    always @(posedge Clock)
        if (!Resetn)
            Q <= 9'b0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1'b1;
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
```

PART 6

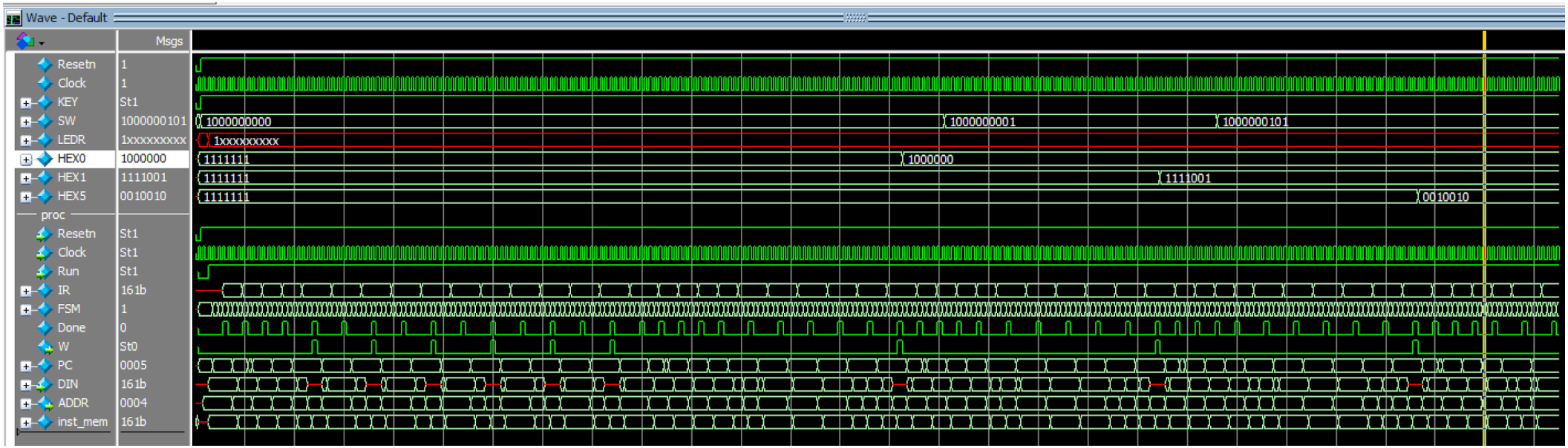
Code:

```
.define LED_ADDRESS 0x1000
.define SW_ADDRESS 0x3000

mv r1, #0
mvt r3, #LED_ADDRESS
mvt r4, #SW_ADDRESS

InLoop: st r1, [r3]
        ld r0, [r4]
        add r0, #1

        add r1, #1

IDKLoop:    mv r2, #0xff

DelayLoop:  sub r2, #1
            bne #DelayLoop
            sub r0, #1
            bne #IDKLoop

            b #InLoop
```

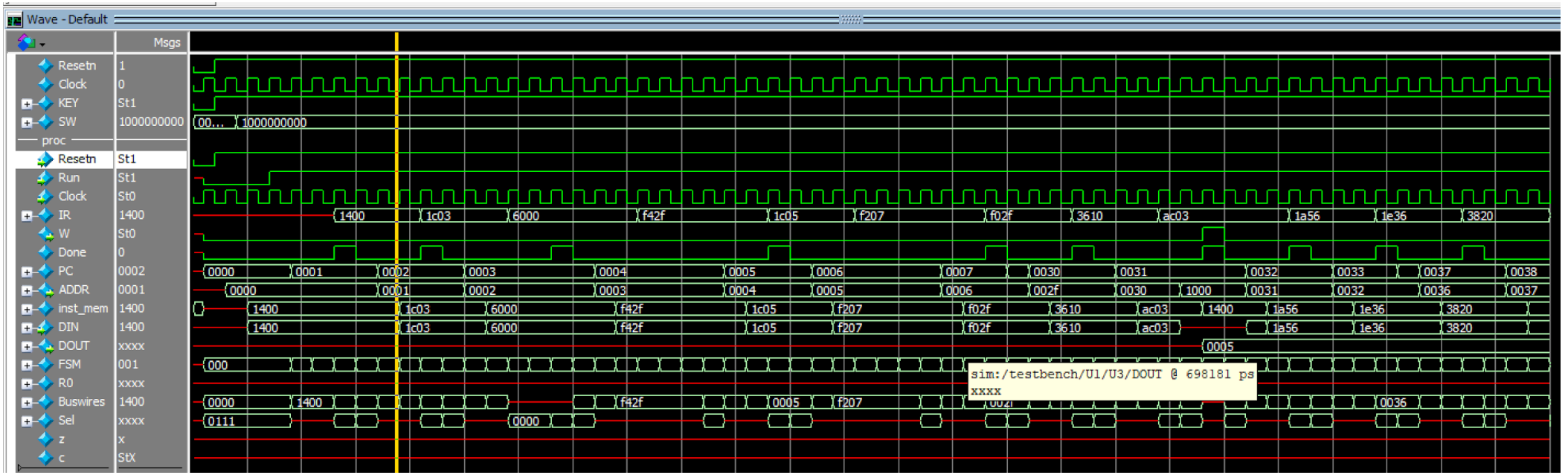
Part 3


Wave - Default
Msgs
Resetn
Clock
KEY
SW
LEDR
proc
Resetn
Clock
Run
IR
FSM
Done
W
PC
DIN
ADDR
inst_mem

Part 4


Wave - Default
Msgs
Resetn
Clock
KEY
SW
LEDR
HEX0
HEX1
HEX5
proc
Resetn
Clock
Run
IR
FSM
Done
W
PC
DIN
ADDR
inst_mem
1
1
St1
1000000101
1xxxxxxxxx
1000000
1111001
0010010
St1
St1
St1
16'b
1
0
St0
0005
16'b
0004
16'b
1000000000
1000000001
1000000101
1xxxxxxxxx
1111111
1000000
1111111
1111001
1111111
0010010

Part 5


Wave - Default
Msgs
Resetn
Clock
KEY
SW
proc
Resetn
Run
Clock
IR
W
Done
PC
ADDR
inst_mem
DIN
DOUT
FSM
R0
Buswires
Sel
z
c
sim:/testbench/U1/U3/DOUT @ 698181 ps
xxxx