```verilog
`timescale 1ns / 1ps

`define PC 0
`define IR 1
`define Y 2
`define MAR 3
`define MDR 4
`define INPORT 5
`define OUTPORT 6
`define Z   7 //Z used for Enable
`define ZHI 8 //ZHI / ZLO used for outputs
`define ZLO 9
`define HI 10
`define LO 11
`define READ 12
`define C   13

`define ADD 0
`define SUB 1
`define NEG 2
`define MUL 3
`define DIV 4
`define AND 5
`define OR  6
`define ROR 7
`define ROL 8
`define SLL 9
`define SRA 10
`define SRL 11
`define NOT 12
`define INC 13

module Control (

   // External Inputs
   input reset, stop, clk,

   // Datapath Inputs
   input CON,
   input [31:27] IRop,

   // Execution Control Signals
   output reg clr, CONin, RAM_wr,

   // General Purpose Register Control
   output reg Gra, Grb, Grc, Rin, Rout, BAout,

   // Datapath Register Control
   output reg [15:0] DPin, DPout,

   // ALU Control
   output [15:0] ALUopp,

   // Status indicator
   output run
);




   //---------------- INTERNAL REGISTERS ----------------//

   reg [4:0] ps, ns; // State Registers
   reg [3:0] op; // ALU operation state.


   //--------------------------------//

   //---------------- PARAMETER DEFINITIONS ----------------//

   // IR op code labels
   parameter LD = 5'b00000, LDI = 5'b00001, ST = 5'b00010,
             ADD = 5'b00011, SUB = 5'b00100, AND = 5'b00101, OR = 5'b00110, ROR = 5'b00111,
ROL = 5'b01000, SHR = 5'b01001, SHRA = 5'b01010, SHL = 5'b01011,
             ADDI = 5'b01100, ANDI = 5'b01101, ORI = 5'b01110, DIV = 5'b01111, MUL =
5'b10000, NEG = 5'b10001, NOT = 5'b10010,
             BR = 5'b10011, JAL = 5'b10100, JR = 5'b10101, IN = 5'b10110, OUT = 5'b10111, MFLO = 5'b11000, MFHI = 5'b11001,
             NOP = 5'b11010, HALT = 5'b11011;

   // State variable labels
   parameter Treset = 5'b11111, Thalt = 5'b11110, T0 = 5'b00000, T1 = 5'b00001, T2 = 5'b00010, T3 = 5'b00011, // States common to all instructions
             T4ALU = 5'b00100, T4dm = 5'b00101, T4ALUimm = 5'b00110, T4nn = 5'b00111, T4br = 5'b01000, T4ldst = 5'b01001, T4jal = 5'b01010, // T4 states
             T5ALU = 5'b01011, T5br = 5'b01100, T5ldi = 5'b01101, T5ld = 5'b01110, T5st = 5'b01111, // T5 states
             T6dm = 5'b10000, T6br = 5'b10001, T6ld = 5'b10010, T6st = 5'b10011, // T6 states
             T7ld = 5'b10100, T7st = 5'b10101; // T7 states


   //-------------------------------//


   //---------------- NEXT-STATE LOGIC AND CONTROL SIGNAL ASSERTION ----------------//


   always @(*) begin
      // Default all control signals to zero.
      zeroes ();

      // Assert next state behaviour and control signals according to present state.
      case (ps)
         // INSTRUCTION FETCH STAGE
         T0:
         begin
            ns = T1;
            DPout[`PC] = 1'b1;
            DPin[`MAR] = 1'b1;
            DPin[`Z] = 1'b1;
         end

         T1:
         begin
            ns = T2;
            DPout[`ZLO] = 1'b1;
            DPin[`PC] = 1'b1;
            DPin[`MDR] = 1'b1;
            DPin[`READ] = 1'b1;
         end

         T2:
         begin
            ns = T3;
            DPout[`MDR] = 1'b1;
            DPin[`IR] = 1'b1;
         end

         // DECODE STAGE
         T3: case(IRop[31:27])
                  MUL, DIV:
                  begin
                     ns = T4dm;
                     Rout = 1'b1;
                     Gra = 1'b1;
                     DPin[`Y] = 1'b1;
                  end

                  ADD, SUB, AND, OR, ROR, ROL, SHR, SHRA, SHL:
                  begin
                     ns = T4ALU;
                     Rout = 1'b1;
                     Grb = 1'b1;
                     DPin[`Y] = 1'b1;
                  end

                  ADDI, ANDI, ORI:
                  begin
                     ns = T4ALUimm;
                     Rout = 1'b1;
                     Grb = 1'b1;
                     DPin[`Y] = 1'b1;
```

```
                end

                NEG, NOT:
                begin
                   ns = T4nn;
                   Rout = 1'b1;
                   Grb = 1'b1;
                   DPin[`Y] = 1'b1;
                end

                BR:
                begin
                   ns = T4br;
                   Rout = 1'b1;
                   Gra = 1'b1;
                   CONin = 1'b1;
                end

                LD, LDI, ST:
                begin
                   ns = T4ldst;
                   BAout = 1'b1;
                   Grb = 1'b1;
                   DPin[`Y] = 1'b1;
                end

                JAL:
                begin
                   ns = T4jal;
                   Rin = 1'b1;
                   Grb = 1'b1;
                   DPout[`PC] = 1'b1;
                end

                JR:
                begin
                   ns = T0;
                   Rout = 1'b1;
                   Gra = 1'b1;
                   DPin[`PC] = 1'b1;
                end

                HALT:
                   ns = Thalt;

                IN:
                begin
                   ns = T0;
                   Rin = 1'b1;
                   Gra = 1'b1;
                   DPout[`INPORT] = 1'b1;
                end

                OUT:
                begin
                   ns = T0;
                   Rout = 1'b1;
                   Gra = 1'b1;
                   DPin[`OUTPORT] = 1'b1;
                end

                MFLO:
                begin
                   ns = T0;
                   Rin = 1'b1;
                   Gra = 1'b1;
                   DPout[`LO] = 1'b1;
                end

                MFHI:
                begin
                   ns = T0;
                   Rin = 1'b1;
                   Gra = 1'b1;
                   DPout[`HI] = 1'b1;
                end
```

```


                    default: ns = T0;
                endcase

            // T4-T7 ACCORDING TO DECODE STAGE.
            // T4 Steps.
            T4dm:
            begin
                ns = T5ALU;
                Rout = 1'b1;
                Grb = 1'b1;
                DPin[`Z] = 1'b1;
            end

            T4ALU:
            begin
                ns = T5ALU;
                Rout = 1'b1;
                Grc = 1'b1;
                DPin[`Z] = 1'b1;
            end

            T4ALUimm:
            begin
                ns = T5ALU;
                DPout[`C] = 1'b1;
                DPin[`Z] = 1'b1;
            end

            T4nn:
            begin
                ns = T5ALU;
                Rout = 1'b1;
                Grb = 1'b1;
                DPin[`Z] = 1'b1;
            end

            T4br: if (CON) begin
                    ns = T5br;
                    DPout[`PC] = 1'b1;
                    DPin[`Y] = 1'b1;
                end
                else ns = T0;

            T4ldst:
            begin
                DPout[`C] = 1'b1;
                DPin[`Z] = 1'b1;

                if (IRop[31:27] == ST) ns = T5st;
                else if (IRop[31:27] == LD) ns = T5ld;
                else ns = T5ldi;
            end

            T4jal:
            begin
                ns = T0;
                Rout = 1'b1;
                Gra = 1'b1;
                DPin[`PC] = 1'b1;
            end

            // T5 Steps.
            T5ALU:
            begin
                DPout[`ZLO] = 1'b1;
                if (IRop[31:27] == MUL || IRop[31:27] == DIV) begin
                    ns = T6dm;
                    DPin[`LO] = 1'b1;
                end
                else begin
                    ns = T0;
                    Rin = 1'b1;
                    Gra = 1'b1;
                end
            end

            T5br:
            begin
               ns = T6br;
               DPout[`C] = 1'b1;
               DPin[`Z] = 1'b1;
            end

            T5st:
            begin
               ns = T6st;
               DPout[`ZLO] = 1'b1;
               DPin[`MAR] = 1'b1;
            end

            T5ld:
            begin
               ns = T6ld;
               DPout[`ZLO] = 1'b1;
               DPin[`MAR] = 1'b1;
            end

            T5ldi:
            begin
               ns = T0;
               DPout[`ZLO] = 1'b1;
               Gra = 1'b1;
               Rin = 1'b1;
            end

            T6st:
            begin
               ns = T7st;
               DPin[`MDR] = 1'b1;
               Gra = 1'b1;
               Rout = 1'b1;
            end

            T6ld:
            begin
               ns = T7ld;
               DPin[`MDR] = 1'b1;
               DPin[`READ] = 1'b1;
            end

            T6br:
            begin
               ns = T0;
               DPout[`ZLO] = 1'b1;
               DPin[`PC] = 1'b1;
            end

            T6dm:
            begin
               ns = T0;
               DPout[`ZHI] = 1'b1;
               DPin[`HI] = 1'b1;
            end

            T7ld:
            begin
               ns = T0;
               DPout[`MDR] = 1'b1;
               Gra = 1'b1;
               Rin = 1'b1;
            end

            T7st:
            begin
               ns = T0;
               RAM_wr = 1'b1;
            end

            Treset:
            begin
```

```
                clr = 1'b1;
                ns = T0;
             end
             Thalt: ns = Thalt;

             default: ns = Thalt;
          endcase
       end

       //--------------------------------//


       //---------------- STATE TRANSITIONS ----------------//

       always @(posedge clk) begin
          if (reset) begin
             ps <= Treset;
             op <= 4'b0;
          end
          else if (stop)
             ps <= Thalt;
          else
             ps <= ns;
             if (ps == T3)  // Assign op only when exiting T3
                op <= op_to_alu(IRop[31:27]);
       end

       //--------------------------------//


       //---------------- MODULE FUNCTIONS ----------------//

          function [3:0] op_to_alu (input [4:0] op_code);
          begin
             case (op_code)
                ADD, ADDI, LD, ST, LDI, BR:   op_to_alu = `ADD;
                SUB:         op_to_alu = `SUB;
                AND, ANDI:   op_to_alu = `AND;
                OR , ORI :   op_to_alu = `OR ;
                ROR:         op_to_alu = `ROR;
                ROL:         op_to_alu = `ROL;
                SHR:         op_to_alu = `SRL;
                SHRA:        op_to_alu = `SRA;
                SHL:         op_to_alu = `SLL;
                DIV:         op_to_alu = `DIV;
                MUL:         op_to_alu = `MUL;
                NEG:         op_to_alu = `NEG;
                NOT:         op_to_alu = `NOT;
                default:     op_to_alu = 4'b0;
             endcase
          end
       endfunction

       assign ALUopp = (ps == T0) ? 1'b1 << `INC : 1'b1 << op;
       assign run = ~(ps == Thalt);

       task zeroes();
          begin
             clr = 0; CONin = 0;
             Gra = 0; Grb = 0; Grc = 0; Rin = 0; Rout = 0; BAout = 0; RAM_wr = 0;
             DPin = 16'b0; DPout = 16'b0;
          end
       endtask


    endmodule




    `timescale 1ns/1ps

    module Control_tb;

        // Inputs to DUT
        reg          reset, stop, clk, CON;
        reg  [31:27] IR;  // Only bits 31:27 are used in decoding

        // Outputs from DUT
        wire        clr, CONin, RAM_wr;
        wire        Gra, Grb, Grc, Rin, Rout, BAout;
        wire [15:0] DPin, DPout, ALUopp;
        wire        run;

        // Instantiate the Device Under Test (DUT)
        Control uut (
            .reset(reset),
            .stop(stop),
            .clk(clk),
            .CON(CON),
            .IR(IR),
            .clr(clr),
            .CONin(CONin),
            .RAM_wr(RAM_wr),
            .Gra(Gra),
            .Grb(Grb),
            .Grc(Grc),
            .Rin(Rin),
            .Rout(Rout),
            .BAout(BAout),
            .DPin(DPin),
            .DPout(DPout),
            .ALUopp(ALUopp),
            .run(run)
        );

        // Clock generation: 10 ns period
        initial begin
            clk = 0;
            forever #5 clk = ~clk;
        end

        // Task to run simulation for a given number of clock cycles
        task run_cycles(input integer n);
            integer i;
            begin
                for(i = 0; i < n; i = i + 1)
                    @(posedge clk);
            end
        endtask

        // Task to apply an instruction and wait for the expected number of cycles.
        // We use the expected cycle count to wait until the FSM should have returned to T0.
        // Optionally, the branch instruction takes a CON input.
        task apply_instruction(input [4:0] instr, input integer expected_cycles, input con_val);
            begin
                $display("\n[%0t] Applying instruction: %b, CON = %b" , $time, instr, con_val);
                IR = instr;
                CON = con_val;

                run_cycles(expected_cycles);

                // Check if the FSM has returned to T0 (the instruction fetch state)
                // (Note: In this design T0 is used as the starting/fetch state.)
                if(uut.ps !== 5'b00000)
                    $display("WARNING: FSM did not return to T0 after %0d cycles. Current state
    = %b", expected_cycles, uut.ps);
                else
                    $display("PASSED: FSM returned to T0 as expected after %0d cycles" ,
    expected_cycles);
            end
        endtask

        // Main stimulus sequence
        initial begin
            $display("Starting Control Module Testbench" );
            // Initialize signals
            reset = 1; stop = 0; CON = 0; IR = 5'b0;
            run_cycles(2);  // Hold reset for a couple of cycles
            reset = 0;
            run_cycles(2);  // Allow FSM to complete reset and move into T0

            //----------------------------------------------------------------------
            // Test 1: ALU Operation (e.g. ADD)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4ALU -> T5ALU -> T0 (~6 cycles)
            // Instruction code for ADD is 5'b00011.
            apply_instruction(5'b00011, 6, 0);

            //----------------------------------------------------------------------
            // Test 2: ALU Immediate Operation (e.g. ADDI)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4ALUimm -> T5ALU -> T0 (~6 cycles)
            // Instruction code for ADDI is 5'b01100.
            apply_instruction(5'b01100, 6, 0);

            //----------------------------------------------------------------------
            // Test 3: Negative Operation (NEG)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4nn -> T5ALU -> T0 (~6 cycles)
            // Instruction code for NEG is 5'b10001.
            apply_instruction(5'b10001, 6, 0);

            //----------------------------------------------------------------------
            // Test 4: Multiply (MUL) Operation
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4dm -> T5ALU -> T6dm -> T0 (~7 cycles)
            // Instruction code for MUL is 5'b10000.
            apply_instruction(5'b10000, 7, 0);

            //----------------------------------------------------------------------
            // Test 5: Division (DIV) Operation
            // FSM path: Similar to MUL: T0 -> T1 -> T2 -> T3 -> T4dm -> T5ALU -> T6dm -> T0
    (~7 cycles)
            // Instruction code for DIV is 5'b01111.
            apply_instruction(5'b01111, 7, 0);

            //----------------------------------------------------------------------
            // Test 6: Branch (BR) with false condition
            // FSM path: When CON is false, T4br goes directly to T0:
            // T0 -> T1 -> T2 -> T3 -> T4br -> T0 (~5 cycles)
            // Instruction code for BR is 5'b10011.
            apply_instruction(5'b10011, 5, 0);

            //----------------------------------------------------------------------
            // Test 7: Branch (BR) with true condition
            // FSM path: When CON is true, branch takes the full path:
            // T0 -> T1 -> T2 -> T3 -> T4br -> T5br -> T6br -> T0 (~7 cycles)
            apply_instruction(5'b10011, 7, 1);

            //----------------------------------------------------------------------
            // Test 8: Memory Load (LD)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4ldst -> T5ld -> T6ld -> T7ld -> T0 (~8 cycles)
            // Instruction code for LD is 5'b00000.
            apply_instruction(5'b00000, 8, 0);

            //----------------------------------------------------------------------
            // Test 9: Memory Store (ST)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4ldst -> T5st -> T6st -> T7st -> T0 (~8 cycles)
            // Instruction code for ST is 5'b00010.
            apply_instruction(5'b00010, 8, 0);

            //----------------------------------------------------------------------
            // Test 10: Jump and Link (JAL)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T4jal -> T0 (~5cycles)
            // Instruction code for JAL is 5'b10100.
            apply_instruction(5'b10100, 5, 0);

            //----------------------------------------------------------------------
            // Test 11: Jump Register (JR)
            // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~4 cycles) because the control goes
    directly to T0 in JR.
            // Instruction code for JR is 5'b10101.
            apply_instruction(5'b10101, 4, 0);

            //----------------------------------------------------------------------
            // Test 12: IN Instruction
            // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
            // Instruction code for IN is 5'b10110.
            apply_instruction(5'b10110, 4, 0);

            //----------------------------------------------------------------------
            // Test 13: OUT Instruction
            // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
            // Instruction code for OUT is 5'b10111.
            apply_instruction(5'b10111, 4, 0);

            //------------------------------------------------------------------------
            // Test 14: MFLO Instruction
            // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
            // Instruction code for MFLO is 5'b11000.
            apply_instruction(5'b11000, 4, 0);

            //------------------------------------------------------------------------
            // Test 15: MFHI Instruction
            // FSM path: T0 -> T1 -> T2 -> T3 -> T0 (~5 cycles)
            // Instruction code for MFHI is 5'b11001.
            apply_instruction(5'b11001, 4, 0);

            //------------------------------------------------------------------------
            // Test 16: HALT Instruction
            // FSM path: T0 -> T1 -> T2 -> T3 -> Thalt and then remain in Thalt (~5 cycles)
            // Instruction code for HALT is 5'b11011.
            apply_instruction(5'b11011, 4, 0);

            // End simulation after a short delay
            #20;
            $display("Simulation complete.");
            $stop;
        end

        // Optional: Monitor important signals and the FSM state
        // (Accessing the internal state variable (ps) hierarchically from the DUT)
        initial begin
            $monitor("Time=%0t | FSM State=%b | IR=%b | clr=%b | CONin=%b | RAM_wr=%b | Gra=%b
    | Grb=%b | Grc=%b | Rin=%b | Rout=%b | BAout=%b" ,
                     $time, uut.ps, IR, clr, CONin, RAM_wr, Gra, Grb, Grc, Rin, Rout, BAout);
        end

    endmodule
```