```systemverilog
/* arm is the spotlight of the show and contains the bulk of the datapath and control logic. This module is split into two parts, the datapath and control.
*/

// clk - system clock
// rst - system reset
// Instr - incoming 32 bit instruction from imem, contains opcode, condition, addresses and or immediates
// ReadData - data read out of the dmem
// WriteData - data to be written to the dmem
// MemWrite - write enable to allowed WriteData to overwrite an existing dmem word
// PC - the current program count value, goes to imem to fetch instruciton
// ALUResult - result of the ALU operation, sent as address to the dmem

module arm (
    input  logic        clk, rst,
    input  logic [31:0] Instr,
    input  logic [31:0] ReadData,
    output logic [31:0] WriteData,
    output logic [31:0] PC, ALUResult,
    output logic        MemWrite
);

    // datapath buses and signals
    logic [31:0] PCPrime, PCPlus4, PCPlus8;  // pc signals
    logic [ 3:0] RA1, RA2;                   // regfile input addresses
    logic [31:0] RD1, RD2;                   // raw regfile outputs
    logic [ 3:0] ALUFlags;                   // alu combinational flag outputs
    logic [31:0] ExtImm, SrcA, SrcB;         // immediate and alu inputs
    logic [31:0] Result;                     // computed or fetched value to be written into regfile or pc

    // control signals
    logic PCSrc, MemtoReg, ALUSrc, RegWrite, FlagWrite;
    logic [1:0] RegSrc, ImmSrc, ALUControl;
    logic [3:0] FlagsReg, cond;


    /* The datapath consists of a PC as well as a series of muxes to make decisions about which data words to pass forward and operate on. It is
    ** noticeably missing the register file and alu, which you will fill in using the modules made in lab 1. To correctly match up signals to the
    ** ports of the register file and alu take some time to study and understand the logic and flow of the datapath.
    */
    //-------------------------------------------------------------------------------
    //                                      DATAPATH
    //-------------------------------------------------------------------------------


    assign PCPrime = PCSrc ? Result : PCPlus4;  // mux, use either default or newly computed value
    assign PCPlus4 = PC + 'd4;                  // default value to access next instruction
    assign PCPlus8 = PCPlus4 + 'd4;             // value read when reading from reg[15]

    // update the PC, at rst initialize to 0
    always_ff @(posedge clk) begin
        if (rst) PC <= '0;
        else     PC <= PCPrime;
    end

    // determine the register addresses based on control signals
    // RegSrc[0] is set if doing a branch instruction
    // RefSrc[1] is set when doing memory instructions
    assign RA1 = RegSrc[0] ? 4'd15        : Instr[19:16];
    assign RA2 = RegSrc[1] ? Instr[15:12] : Instr[ 3: 0];

    // Instantiates a register file to hold values.
    reg_file u_reg_file (
        .clk       (clk),
        .wr_en     (RegWrite),
        .write_data(Result),
        .write_addr(Instr[15:12]),
        .read_addr1(RA1),
        .read_addr2(RA2),
        .read_data1(RD1),
        .read_data2(RD2)
    );

    // two muxes, put together into an always_comb for clarity
    // determines which set of instruction bits are used for the immediate
    always_comb begin
        if      (ImmSrc == 'b00) ExtImm = {{24{Instr[7]}},Instr[7:0]};           // 8 bit immediate - reg operations
        else if (ImmSrc == 'b01) ExtImm = {20'b0, Instr[11:0]};                   // 12 bit immediate - mem operations
        else                     ExtImm = {{6{Instr[23]}}, Instr[23:0], 2'b00}; // 24 bit immediate - branch operation
    end

    // WriteData and SrcA are direct outputs of the register file, wheras SrcB is chosen between reg file output and the immediate
    assign WriteData = (RA2 == 'd15) ? PCPlus8 : RD2;           // substitute the 15th regfile register for PC
    assign SrcA      = (RA1 == 'd15) ? PCPlus8 : RD1;           // substitute the 15th regfile register for PC
    assign SrcB      = ALUSrc        ? ExtImm  : WriteData;     // determine alu operand to be either from reg file or from immediate


    // Instantiates an alu module to do arithmetic operations.
    alu u_alu (
        .a          (SrcA),
        .b          (SrcB),
        .ALUControl (ALUControl),
        .Result     (ALUResult),
        .ALUFlags   (ALUFlags)
    );

    // determine the result to run back to PC or the register file based on whether we used a memory instruction
    assign Result = MemtoReg ? ReadData : ALUResult;    // determine whether final writeback result is from dmemory or alu

    always_ff @(posedge clk) begin
         if (FlagWrite) FlagsReg = ALUFlags;
    end

    /* The control conists of a large decoder, which evaluates the top bits of the instruction and produces the control bits
    ** which become the select bits and write enables of the system. The write enables (RegWrite, MemWrite and PCSrc) are
    ** especially important because they are representative of your processors current state.
    */
    //-------------------------------------------------------------------------------
    //                                      CONTROL
    //-------------------------------------------------------------------------------
    assign cond = Instr[31:28];

    always_comb begin
        casez (Instr[27:20])

            // ADD (Imm or Reg)
            8'b00?_0100_0 : begin   // note that we use wildcard "?" in bit 25. That bit decides whether we use immediate or reg, but regardless we add
                PCSrc    = 0;
                MemtoReg = 0;
                MemWrite = 0;
                ALUSrc   = Instr[25]; // may use immediate
                RegWrite = 1;
                RegSrc   = 'b00;
                ImmSrc   = 'b00;
                ALUControl = 'b00;
                FlagWrite = 0;
            end

            // SUB (Imm or Reg)
            8'b00?_0010_0 : begin   // note that we use wildcard "?" in bit 25. That bit decides whether we use immediate or reg, but regardless we sub
                PCSrc    = 0;
                MemtoReg = 0;
                MemWrite = 0;
                ALUSrc   = Instr[25]; // may use immediate
                RegWrite = 1;
                RegSrc   = 'b00;
                ImmSrc   = 'b00;
                ALUControl = 'b01;
                FlagWrite = 0;
            end

            // AND
            8'b000_0000_0 : begin
                PCSrc    = 0;
                MemtoReg = 0;
                MemWrite = 0;
                ALUSrc   = 0;
                RegWrite = 1;
                RegSrc   = 'b00;
                ImmSrc   = 'b00;    // doesn't matter
                ALUControl = 'b10;
               FlagWrite = 0;
            end

            // ORR
            8'b000_1100_0 : begin
                PCSrc    = 0;
                MemtoReg = 0;
                MemWrite = 0;
                ALUSrc   = 0;
                RegWrite = 1;
                RegSrc   = 'b00;
                ImmSrc   = 'b00;    // doesn't matter
                ALUControl = 'b11;
                FlagWrite = 0;
            end

            // LDR
            8'b010_1100_1 : begin
                PCSrc    = 0;
                MemtoReg = 1;
                MemWrite = 0;
                ALUSrc   = 1;
                RegWrite = 1;
                RegSrc   = 'b10;    // msb doesn't matter
                ImmSrc   = 'b01;
                ALUControl = 'b00;  // do an add
                FlagWrite = 0;
            end

            // STR
            8'b010_1100_0 : begin
                PCSrc    = 0;
                MemtoReg = 0; // doesn't matter
                MemWrite = 1;
                ALUSrc   = 1;
                RegWrite = 0;
                RegSrc   = 'b10;    // msb doesn't matter
                ImmSrc   = 'b01;
                ALUControl = 'b00;  // do an add
                FlagWrite = 0;
            end

            // B
            8'b1010_???? : begin
               if((cond == 1110) ||
                  (cond == 0000 && FlagsReg[2]) ||
                  (cond == 0001 && !FlagsReg[2]) ||
                  (cond == 1010 && !FlagsReg[3]) ||
                  (cond == 1100 && !FlagsReg[3] && !FlagsReg[2]) ||
                  (cond == 1101 && (FlagsReg[3] || FlagsReg[2])) ||
                  (cond == 1011 && FlagsReg[3])
                  ) begin
                   PCSrc    = 1;
                   MemtoReg = 0;
                   MemWrite = 0;
                   ALUSrc   = 1;
                   RegWrite = 0;
```

```
                     RegSrc    = 'b01;
                     ImmSrc    = 'b10;
                     ALUControl  = 'b00;  // do an add
                     FlagWrite = 0;
                  end else begin
                     PCSrc     = 0;
                     MemtoReg = 0;
                     MemWrite = 0;
                     ALUSrc    = 0;
                     RegWrite = 0;
                     RegSrc    = 'b00;
                     ImmSrc    = 'b00;    // doesn't matter
                     ALUControl  = 'b00;
                     FlagWrite = 0;
                  end
//                PCSrc    = 1;
//                   MemtoReg = 0;
//                   MemWrite = 0;
//                   ALUSrc   = 1;
//                   RegWrite = 0;
//                   RegSrc   = 'b01;
//                   ImmSrc   = 'b10;
//                   ALUControl = 'b00;
//                   FlagWrite = 0;
               end

               // CMP
               8'b00?00101 : begin
                    PCSrc     = 0;
                    MemtoReg = 0;
                    MemWrite = 0;
                    ALUSrc    = Instr[25];
                    RegWrite = 1;
                    RegSrc    = 'b00;
                    ImmSrc    = 'b00;
                    ALUControl  = 'b01;  // subtract
                    FlagWrite = 1;
               end

            default: begin
                    PCSrc     = 0;
                    MemtoReg = 0; // doesn't matter
                    MemWrite = 0;
                    ALUSrc    = 0;
                    RegWrite = 0;
                    RegSrc    = 'b00;
                    ImmSrc    = 'b00;
                    ALUControl  = 'b00;  // do an add
                    FlagWrite = 0;
            end
           endcase
       end


endmodule
```