```
module Datapath (ALUOp, RegDst, MemtoReg, MemRead, MemWrite, IorD, RegWrite, IRWrite,
PCWrite,   PCWriteCond, ALUSrcA, ALUSrcB, PCSource, opcode, clock); // the control inputs + clock
input [1:0] ALUOp, ALUSrcB, PCSource; // 2-bit control signals
input RegDst, MemtoReg, MemRead, MemWrite, IorD, RegWrite, IRWrite, PCWrite, PCWriteCond,
ALUSrcA,    clock; // 1-bit control signals
output [5:0] opcode ;// opcode is needed as an output by control
reg [31:0] PC, Memory [0:1023], MDR,IR, ALUOut; // CPU state + some temporaries
wire [31:0] A,B,SignExtendOffset, PCOffset, ALUResultOut, PCValue, JumpAddr, Writedata, ALUAin,
    ALUBin,MemOut; / these are signals derived from registers
wire [3:0] ALUCtl; //. the ALU control lines
wire Zero; the Zero out signal from the ALU
wire[4:0] Writereg;// the signal used to communicate the destination register
 initial PC = 0; //start the PC at 0

//Combinational signals used in the datapath

// Read using word address with either ALUOut or PC as the address source
assign MemOut = MemRead ? Memory[(IorD ? ALUOut : PC)>>2]:0;
assign opcode = IR[31:26];// opcode shortcut

// Get the write register address from one of two fields depending on RegDst
assign Writereg = RegDst ? IR[15:11]: IR[20:16];

// Get the write register data either from the ALUOut or from the MDR
assign Writedata = MemtoReg ? MDR : ALUOut;

// Sign-extend the lower half of the IR from load/store/branch offsets
assign SignExtendOffset = {{16{IR[15]}},IR[15:0]}; //sign-extend lower 16 bits;

// The branch offset is also shifted to make it a word offset
assign PCOffset = SignExtendOffset << 2;

// The A input to the ALU is either the rs register or the PC
assign ALUAin = ALUSrcA ? A : PC; //ALU input is PC or A

// Compose the Jump address
assign JumpAddr = {PC[31:28], IR[25:0],2'b00}; //The jump address
```

FIGURE e4.14.6 A Verilog version of the multicycle MIPS datapath that is appropriate for synthesis. This datapath relies on several units from Appendix B. Initial statements do not synthesize, and a version used for synthesis would have to incorporate a reset signal that had this effect. Also note that resetting R0 to 0 on every clock is not the best way to ensure that R0 stays 0; instead, modifying the register file module to produce 0 whenever R0 is read and to ignore writes to R0 would be a more efficient solution.