Computer Organization & Design

Verilog - Appendix B Sec.4 (CDROM)

Verilog and VHDL

VHDL is patterned after ADA (a computer language pushed by the Dept. of Defense a decade ago, but is which now extinct).

VHDL, taught in ECE2031, is covered in the book *Rapid Prototyping of Digital Systems* by Prof. James Hamblen - http://www.ece.gatech.edu/~hamblen/book/bookse.htm

Verilog is described in Patterson and Hennessy, Section 5.8 (CDROM), the source of the following example - a MIPS multicycle CPU design.

Verilog to design and simulate multicycle CPU

```
module CPU (clock);
parameter LW = 6'b100011, SW = 6'b101011, BEQ=6'b000100, J=6'd2;
input clock; //the clock is an external input
// The architecturally visible registers and scratch registers for implementation
reg [31:0] PC, Regs[0:31], Memory [0:1023], IR, ALUOut, MDR, A, B;
reg [2:0] state; // processor state
wire [5:0] opcode; //use to get opcode easily
wire [31:0] SignExtend,PCOffset; //used to get sign extended offset field
assign opcode = IR[31:26]; //opcode is upper 6 bits
assign SignExtend = {{16{IR[15]}},IR[15:0]}; //sign extension of lower 16-bits of instruction
assign PCOffset = SignExtend << 2; //PC offset is shifted
// set the PC to 0 and start the control in state 0
initial begin PC = 0; state = 1; end

//The state machine--triggered on a rising clock
always @(posedge clock) begin
    Regs[0] = 0; //make R0 0 //short-cut way to make sure R0 is always 0
   case (state) //action depends on the state
      1: begin // first step: fetch the instruction, increment PC, go to next state
         IR <= Memory[PC>>2];
         PC <= PC + 4;
         state=2; //next state
      end
```

Verilog description - 2

```
  2: begin // second step: Instruction decode, register fetch, also compute branch address
      A <= Regs[IR[25:21]];
      B <= Regs[IR[20:16]];
      state= 3;
      ALUOut <= PC + PCOffset; // compute PC-relative branch target
    end


3: begin // third step: Load/store execution, ALU execution, Branch completion
       state =4; // default next state
       if ((opcode==LW) |(opcode==SW)) ALUOut <= A + SignExtend; //compute effective address
       else if (opcode==6'b0) case (IR[5:0]) //case for the various R-type instructions
            32: ALUOut= A + B; //add operation
            default: ALUOut= A; //other R-type operations: subtract, SLT, etc.
         endcase

         else if (opcode == BEQ) begin
                        if (A==B) PC <= ALUOut; // branch taken--update PC
                         state = 1;
          end
          else if (opocde=J) begin
                  PC = {PC[31:28], IR[25:0],2'b00}; // the jump target PC
                  state = 1;
          end    //Jumps
                  else ; // other opcodes or exception for undefined instruction would go here
  end
```


Verilog description - 3

```
          4: begin
              if (opcode==6'b0) begin //ALU Operation
                    Regs[IR[15:11]] <= ALUOut; // write the result
                    state =1;
              end //R-type finishes
                 else if (opcode == LW) begin // load instruction
                     MDR <= Memory[ALUOut>>2]; // read the memory
                     state =5; // next state
                  end
                        else if (opcode == LW) begin
                            Memory[ALUOut>>2] <= B; // write the memory
                            state = 1; // return to state 1
                        end //store finishes
                             else ; // other instructions go here
                 end


        5: begin // LW is the only instruction still in execution
                Regs[IR[20:16]] = MDR; // write the MDR to the register
                state = 1;
             end //complete a LW instruction
           endcase
       end
       endmodule
```