```
// Signal for a taken branch: instruction is BEQ and registers are equal
assign takebranch = (IFIDIR[31:26]==BEQ) && (Regs[IFIDIR[25:21]]== Regs[IFIDIR[20:16]]);
   reg [5:0] i; //used to initialize registers
   initial begin
      PC = 0;
     IFIDIR = no-op; IDEXIR = no-op; EXMEMIR = no-op; MEMWBIR = no-op; // put no-ops in pipeline registers
      for (i = 0;i<=31;i = i+1) Regs[i] = i; //initialize registers--just so they aren't don't cares
   end

   always @ (posedge clock) begin
   if (~stall) begin // the first three pipeline stages stall if there is a load hazard
      if (~takebranch) begin     // first instruction in the pipeline is being fetched normally
          IFIDIR <= IMemory[PC>>2];
          PC <= PC + 4;

     end else begin // a taken branch is in ID; instruction in IF is wrong; insert a no-op and reset the PC
        IFDIR <= no-op;
        PC <= PC + 4 + ({{16{IFIDIR[15]}}, IFIDIR[15:0]}<<2);
        end

      // second instruction is in register fetch
       IDEXA <= Regs[IFIDIR[25:21]]; IDEXB <= Regs[IFIDIR[20:16]]; // get two registers

      // third instruction is doing address calculation or ALU operation
        IDEXIR <= IFIDIR;  //pass along IR
if ((IDEXop==LW) |(IDEXop==SW))  // address calculation & copy B
         EXMEMALUOut <= IDEXA +{{16{IDEXIR[15]}}, IDEXIR[15:0]};
       else if (IDEXop==ALUop) case (IDEXIR[5:0]) //case for the various R-type instructions
           32: EXMEMALUOut <= Ain + Bin;  //add operation
           default: ; //other R-type operations: subtract, SLT, etc.
          endcase
       EXMEMIR <= IDEXIR; EXMEMB <= IDEXB; //pass along the IR & B register
     end
   else EXMEMIR <= no-op; /Freeze first three stages of pipeline; inject a nop into the EX output

      //Mem stage of pipeline
       if (EXMEMop==ALUop) MEMWBValue <= EXMEMALUOut; //pass along ALU result
          else if (EXMEMop == LW) MEMWBValue <= DMemory[EXMEMALUOut>>2];
            else if (EXMEMop == SW) DMemory[EXMEMALUOut>>2] <=EXMEMB; //store

      // the WB stage
MEMWBIR <= EXMEMIR; //pass along IR
      if ((MEMWBop==ALUop) & (MEMWBrd != 0)) Regs[MEMWBrd] <= MEMWBValue; // ALU operation

      else if ((EXMEMop == LW)& (MEMWBIR[20:16] != 0)) Regs[MEMWBIR[20:16]] <= MEMWBValue;

   end
endmodule
```

FIGURE e4.14.4 A behavioral definition of the five-stage MIPS pipeline with stalls for loads when the destination is an ALU instruction or effective address calculation. (*Continued*)