```
/*
   Ben Davis
   2/15/24
   EE 371
   Lab 4, Task 2

   This module takes in a 8 bit input and a preloaded ram and
   searches to see if the ram holds the input, using binary
   search.  It has a clear, hold, and comp input. THey respectively
   clear the data, hold the search, and execute one cycle of the search.
   There are three outputs, L is the address of the input data in the
   RAM, finish tells the fsm that the process is finished, and
   found is true if the data is in the RAM.

*/

module data_bin (
               input logic clk, //clock
               input logic [7:0] A, //input data
               input logic clear, //clears data -- like reset
               input logic comp, //executes search
               input logic hold, //stalls search
               output logic [4:0] L, //data input address in RAM
               output logic finish, //search is done
               output logic found); //RAM contains input data

   //intermediate logic for the binary search
   logic [7:0] ram_out; //contents of the RAM
   int count; //counter fo the search process
   logic [4:0] addr; //address
   logic [4:0] addr_temp; //next address for the ram

   ram32x8 ram (.address(addr_temp), .clock(clk), .data(8'b0), .wren(0), .q(ram_out));

   always_ff @(posedge clk) begin
      /* if in the clear state, continuously reset the address
      to 16--the middle of the ram.*/
      if(clear) begin
         addr <= 5'b10000;
         addr_temp <= 5'b10000;
      end else if(hold) begin
      //in the stall state, assign address to next address
         addr_temp <= addr;
      end else if(comp) begin
      //executes the search
         if(A > ram_out) begin
         //if greater than, multiply address by 1.5
            addr <= addr + (addr >> 1);
         end else if(A < ram_out) begin
         //if less than, divide addr by two
            addr <= addr - (addr >> 1);
         end else begin
         //if its equal, hold the address
            addr <= addr;
         end
         //increment counter
         count <= count + 1;
      end
   end

   assign L = addr;
   //if the ram found the input
   assign found = (ram_out == A);
   //the ram has 32 elements. binary search should not
   //take more than five "recursions"
   assign finish = (count >=5);

endmodule
//testbench
`timescale 1ps/1ps
module data_bin_tb ();

   //recall variables
```

```
74      logic clk, clear, comp, hold;
75      logic [7:0] A;
76      logic [4:0] L;
77      logic finish, found;
78
79      data_bin dut (.clk, .A, .clear, .comp, .hold, .L, .finish, .found);
80
81      //clcok setup
82      parameter clk_pd = 100;
83      initial begin
84         clk <= 0;
85         forever #(clk_pd /2) clk <= ~clk;
86      end //of clock setup
87
88      //tests an instance where it takes in an input of "8," where it runs through
89      //five "recursions"
90      initial begin
91         A <= 8'b00001010; clear <= 1; comp <= 0; hold <= 0; @(posedge clk);
92         A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
93         A <= 8'b00001010; clear <= 0; comp <= 1; hold <= 0; @(posedge clk);
94         A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
95         A <= 8'b00001010; clear <= 0; comp <= 1; hold <= 0; @(posedge clk);
96         A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
97         A <= 8'b00001010; clear <= 0; comp <= 1; hold <= 0; @(posedge clk);
98         A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
99         A <= 8'b00001010; clear <= 0; comp <= 1; hold <= 0; @(posedge clk);
100        A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
101        A <= 8'b00001010; clear <= 0; comp <= 1; hold <= 0; @(posedge clk);
102        A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
103        A <= 8'b00001010; clear <= 0; comp <= 0; hold <= 1; @(posedge clk);
104
105        $stop;
106     end
107  endmodule //test end
```