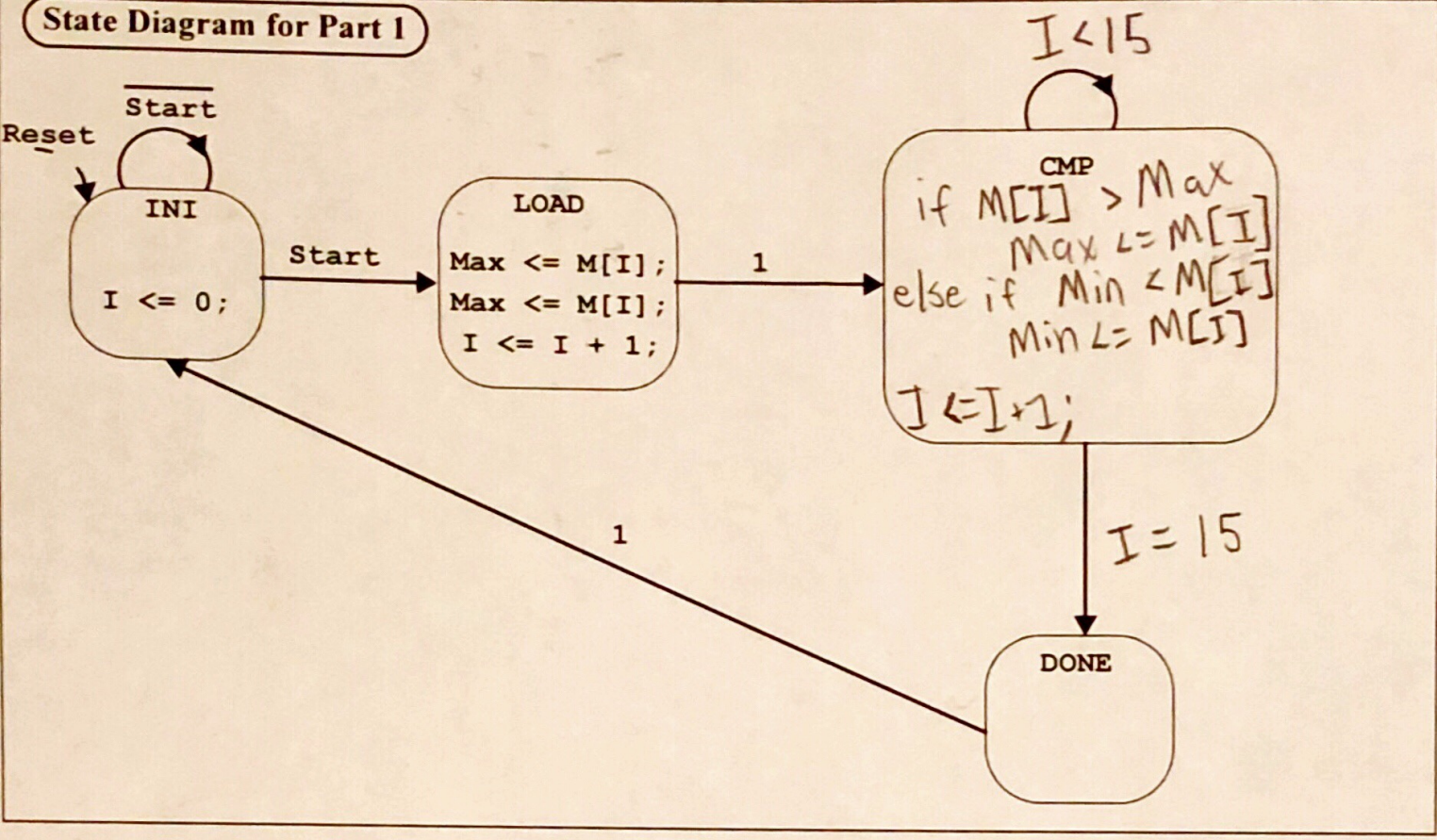


Part of the incomplete Verilog code given to you for part 1(**min_max_finder_part1 .v**):

```
module min_max_finder_part1
(Max, Min, Start, Clk, Reset,
Qi, Ql, Qc, Qd);

input Start, Clk, Reset;
output [7:0] Max, Min;
output Qi, Ql, Qc, Qd;

reg [7:0] M [0:15];
//reg [7:0] X;
reg [3:0] state;
reg [7:0] Max;
reg [7:0] Min;
reg [3:0] I;

localparam
INI  =  4'b0001,
LOAD =  4'b0010,
COMP =  4'b0100,
DONE =  4'b1000;

assign {Qd, Qc, Ql, Qi} = state;
```

To facilitate use of symbolic names for states and also user defined state encoding

```
always @(posedge Clk, posedge Reset)

  begin : CU_n_DU
    if (Reset)
       begin
          state <= INI;
          I <= 4'bXXXX;
          Max <= 8'bXXXXXXXX;
          Min <= 8'bXXXXXXXX;
       end
    else
       begin
          case (state)
            INI :
              begin
                 // state transitions
                 if (Start)
                   state <= LOAD;
                 // RTL operations
                I <= 0;
              end
            LOAD :    // to be completed
              begin
```

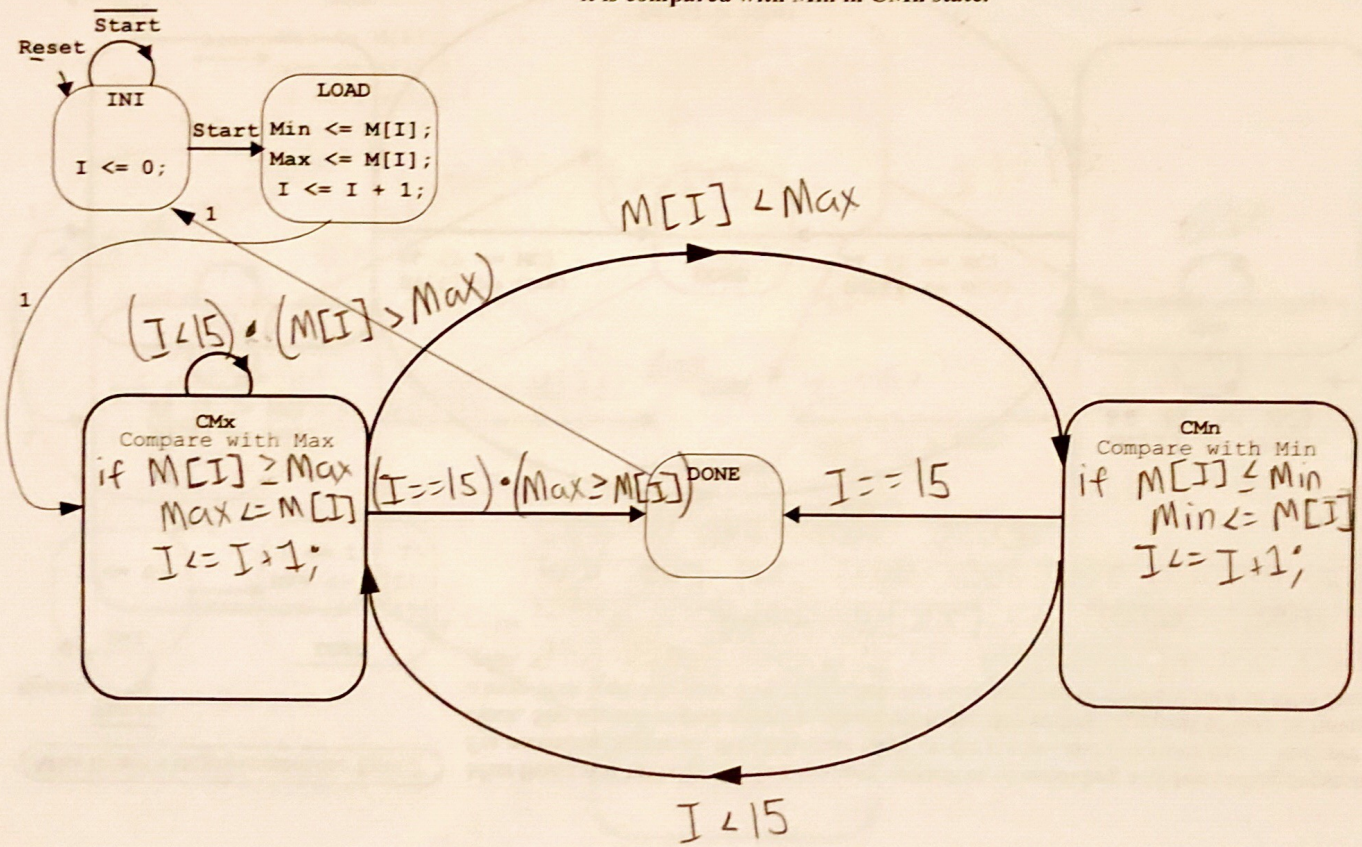
To avoid unnecessary recirculating muxes controlled by Reset

"case" statement to describe both CU and DPU

Part 2 (one comparator)

State Diagram for Part 2 for 1 comparator

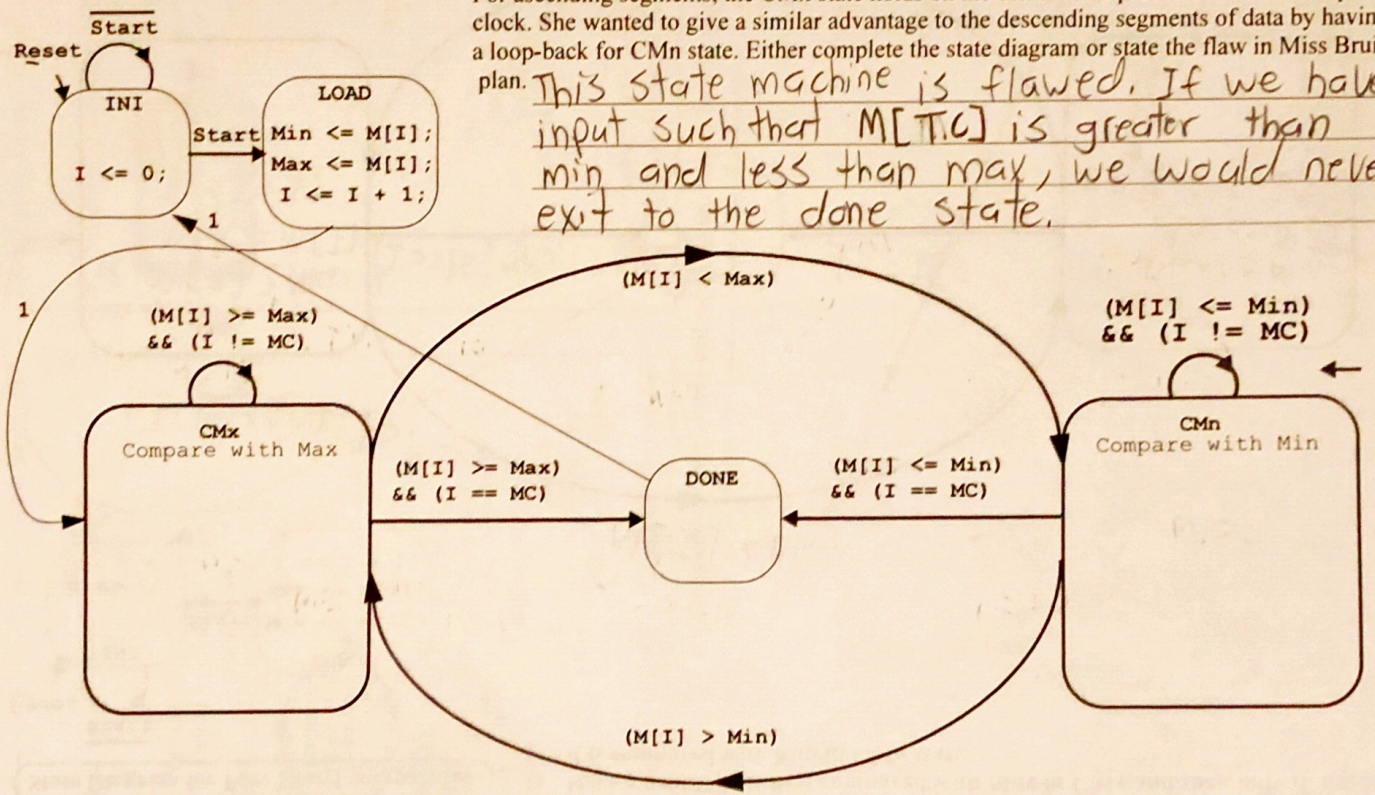
Here a new M[I] is first compared with Max in CMx and then only if, needed, it is compared with Min in CMn state.



100, 1, 2, 50

Miss Bruin's improvement for Part 2

Miss Bruin was told that the data has long sequences of ascending and descending segments. For ascending segments, the CMx state holds on the control and processes data 1 item per clock. She wanted to give a similar advantage to the descending segments of data by having a loop-back for CMn state. Either complete the state diagram or state the flaw in Miss Bruin's plan. This state machine is flawed. If we have input such that M[I] is greater than min and less than max, we would never exit to the done state.



MC = Max_Count or Terminal_Count

"I" would need to only increment in CMn if we had already checked it against max, and vice versa, otherwise we won't know when a specific M[I] has been checked against both Min & Max, so we would need to add a flag

State Diagram for Part 3 Method 1

State Diagram for Part 3 Method 3 (similar to Method 1)

Earlier CMnF is now merged with CMn state. When control is received from CMx to CMn, the Flag is received in SET condition for the CMn state to recognize that the current M[I] is already compared with the Max and hence the "I" can be incremented unconditionally at the end of the current clock irrespective of whether M[I] is less than (or equal) to Min.
Similarly the earlier CMxF is now merged with CMx.

State Diagram for Part 3 Method 4 (similar to Method 2)

Questions:

Answers to the these end-of-lab questions (and any waveform with your analysis/justification if asked in future labs) is an **individual effort** (not a team task). For this lab, you need to submit hard copies of (a) the 6 completed state diagrams (completed in hand) and (b) answers to the following questions. No waveforms are needed.

1. Is your Part 1 control unit a Moore machine or a Mealy machine? How about Part 2? How can you tell whether it is a Moore or a Mealy machine? Which of the two options (Moore or Mealy) results in saving clock cycles? Is saving clock cycles always better, or can it possibly result in widening the clock (lowering the frequency)? Discuss briefly using a simple example.

   Part 1 is mealy, we use state + inputs to control what happens. Part 2 is also mealy as it too, uses inputs, and not just state, to determine what to do. Mealy results in saving clock cycles but normally results in a longer clock, so it may not actually be faster. It is possible to merge multiple operations into one clock by widening it, but this may not be better. 5ops x 5ns clock < 2 ops x 15ns clock

2. You know that it may not be a good idea to pack too many operations into one state as the total time it takes to complete all the operations may be fairly long resulting in a slow clock. Here, in this design, we assumed that the memory is a register array and hence is very fast for accessing data. Hence, in a clock, we not only access the memory **M** with the index **I**, but also compare the **M[I]** with **Max** and/or **Min** in the same clock. On the other hand, if we are given a *slow* memory needing a significant access time comparable to the comparison time, will it be advantageous in part 1 to split the memory access and comparing with the **Max** and **Min** into two separate states so that the clock rate is higher and overall speed is better? Discuss.

   If the memory access was super slow we would be better off breaking the one state into two. In this way we avoid doubling the length of our clock cycle by roughly 2x and instead double the number of states. While this will make us take more clocks it will make the clocks shorter and since the clock must be long enough for the longest possible delay it is advantageous to shorten the clock and use more clocks

3. All of you have designed the control unit for the data unit for part 2, utilizing a SINGLE comparator. Our strategy was to compare **M(I)** with current **Max** first and then, if necessary, with current **Min** also. We noted that the total number of clock cycles taken by our control unit is data-dependent. For example, if the data in the memory is already sorted in ascending order, it would take the least number of clock cycles. If the data is in the descending order it would take maximum number of clock cycles. As we do not have any prior knowledge of data distribution we can not optimize the control unit design.

   Now for the sake of our exercise, assume that (you know that) the data has many large chunks of numbers arranged in ascending order and similarly it has also many large chunks of numbers arranged in descending order. Then, to optimize your design, it is desirable that once you go into one of the two states, **CMx** (compare with **Max**) or **CMn** (compare with **Min**), you stay there as long as possible. Unlike in your original design where you go back to **CMx** from **CMn** (as long as it is not max-count), here you perhaps would like to stay in **CMn** unless the data item being compared warrants comparison with **Max** (the current maximum).

   Your friend, Miss Bruin, thought that she could easily do this problem and arrived at the state diagram given in previous pages. See the page #4 with her state diagram. Do you agree with her design? Is there any flaw in her thinking? Do you increment the counter **I** *conditionally* or *unconditionally* in state **CMx**? And similarly in State **CMn**? If there is no flaw in her design, complete the state diagram. If you find a flaw in her (we mean in her design), state what is the flaw and how to correct it. This exercise leads to Part 3 (methods 1, 2, 3, and 4). Answer this question on the state diagram sheet itself.