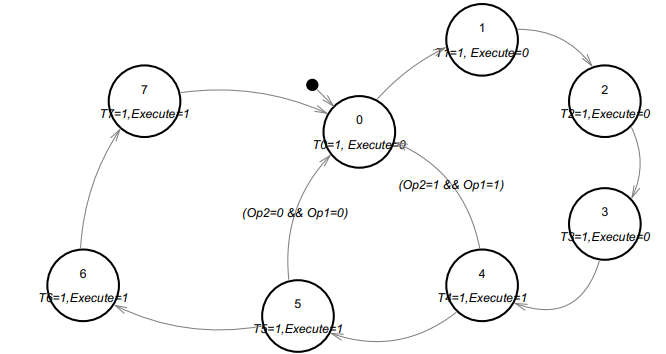
Tyler Kim

Digital Logic Design

***Instruction Sequencer***

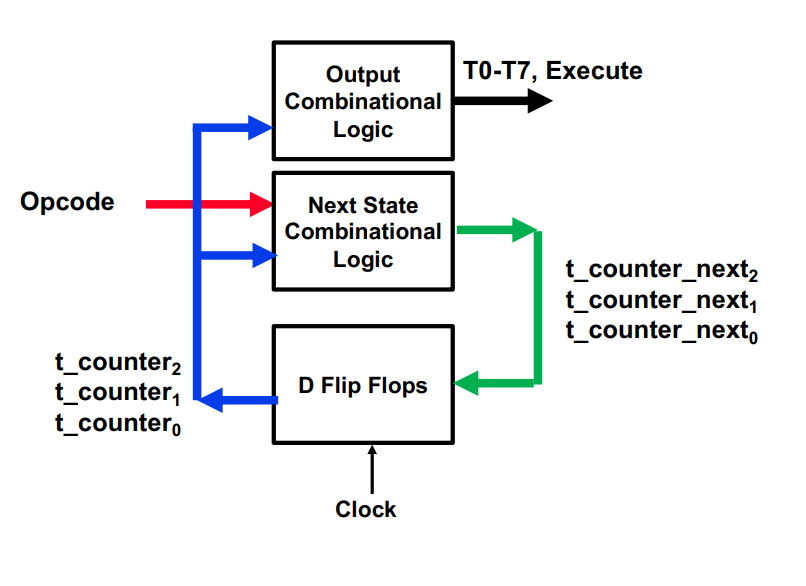
**Description**

The purpose of this learning activity is to create an instruction sequencer for the cpu. The instruction sequencer will take in three inputs: *opcode(2..0)*, *reset*, and *clock* and will have eight outputs: *t0*-*t7* and *execute*. This component can be represented as a finite state machine (FSM) with eight states ranging from state 0 to state 7. At each state, the component will return a particular output.

****

**Figure 1: Finite State Machine of Instruction Sequencer (from Document)**

Figure 1 represents the FSM for the instruction sequencer. In detail, the machine will start at state 0 and move to states 1 – 4 immediately. If *opcode(2)* is 1 and *opcode(1)* is 1, then the FSM will go back to state 0; if not, then it will continue and the same logic applies at state 5. The FSM can be represented in multiple tables.

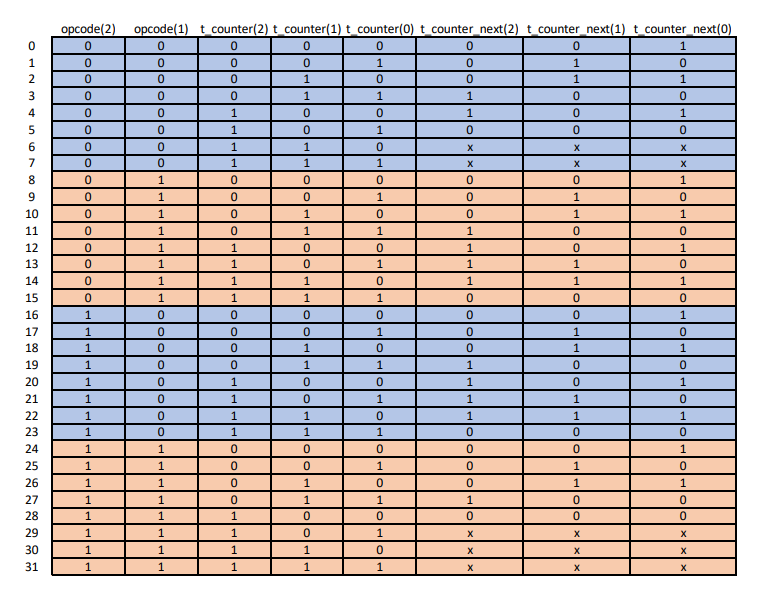
****

**Figure 2: Overview of the Instruction Sequencer (from Slides)**

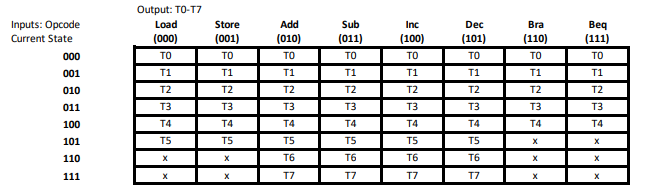
Figure 2 highlights a flow diagram of the instruction sequencer. In detail, the *opcode* will be fed into to calculate the current and next states of the FSM like a Mealy Machine. The instruction sequencer will store the current and next states and output a signal to *t0*-*t7* and *execute*.

**Design and Schematic**

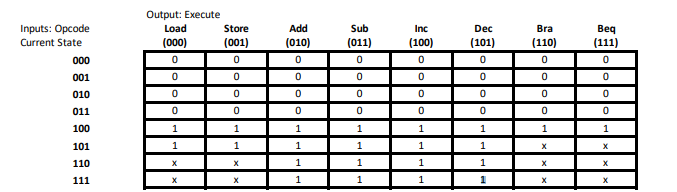
The design for the instruction sequencer requires multiple tables for the following: outputs *t0-t7*, next states, and output *execute*.

****

**Figure 3: Next State Table (from Slides)**

****

**Figure 4: T0-T7 Output Table (from Document)**

****

**Figure 5: Execute Output Table (from Document)**

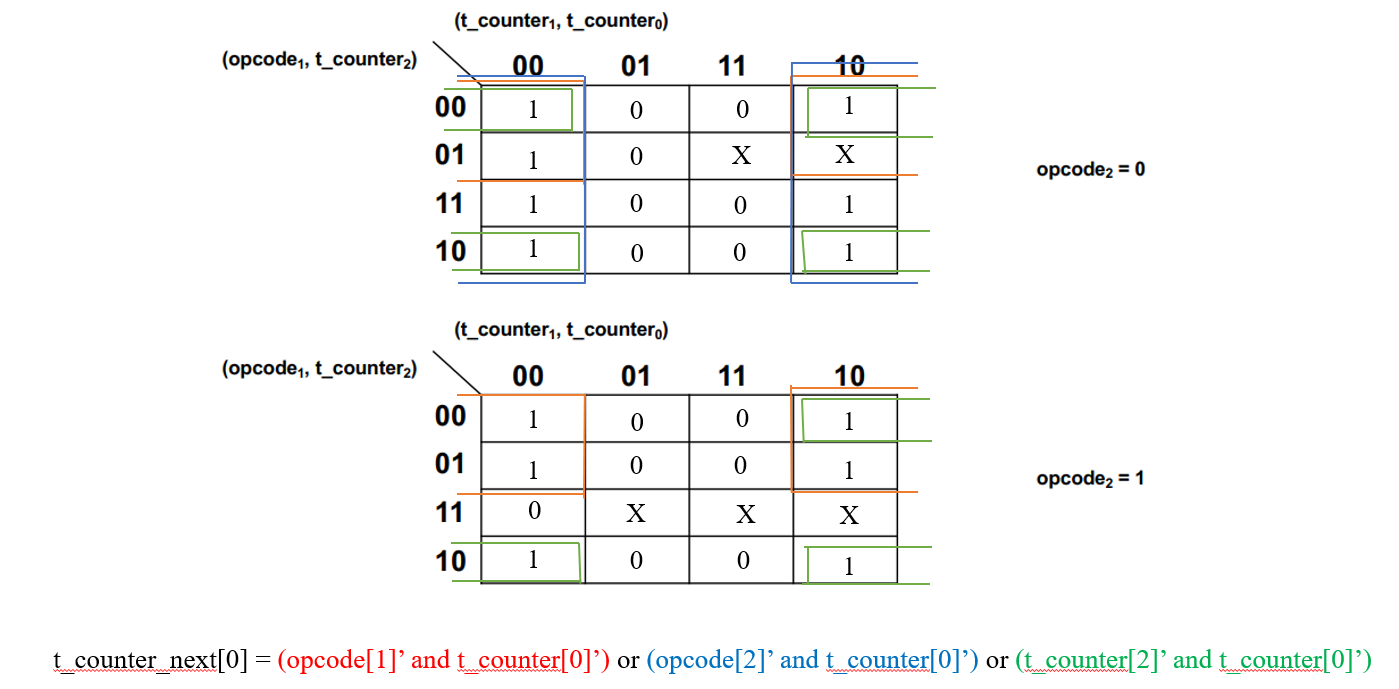
Developing the schematic required three steps: the current state logic, the next state logic, and the output logic.

Current State Logic

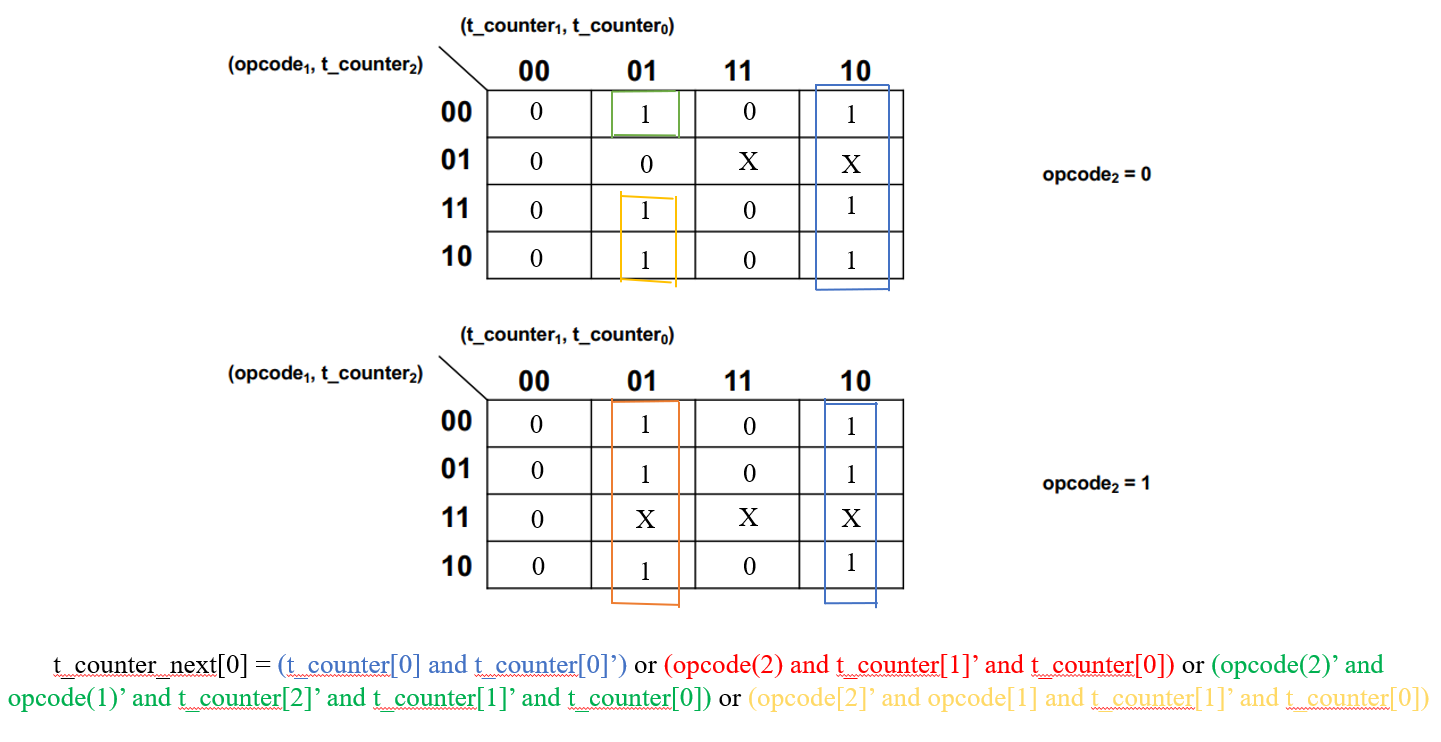
Designing the current state logic only required d-flipflops. Since the Mealy machine required 8 states, three state variables were used to represent all eight states. D-flipflops were used to store each state variable where the output *q* represents the current state and the input *d* stores the next state. Every time a state transition occurs, the flipflop will update the current state to the next state; thus, the cycle continues. In other words, current state logic only required storing the current state as *q* in the d-flipflops.

Next State Logic

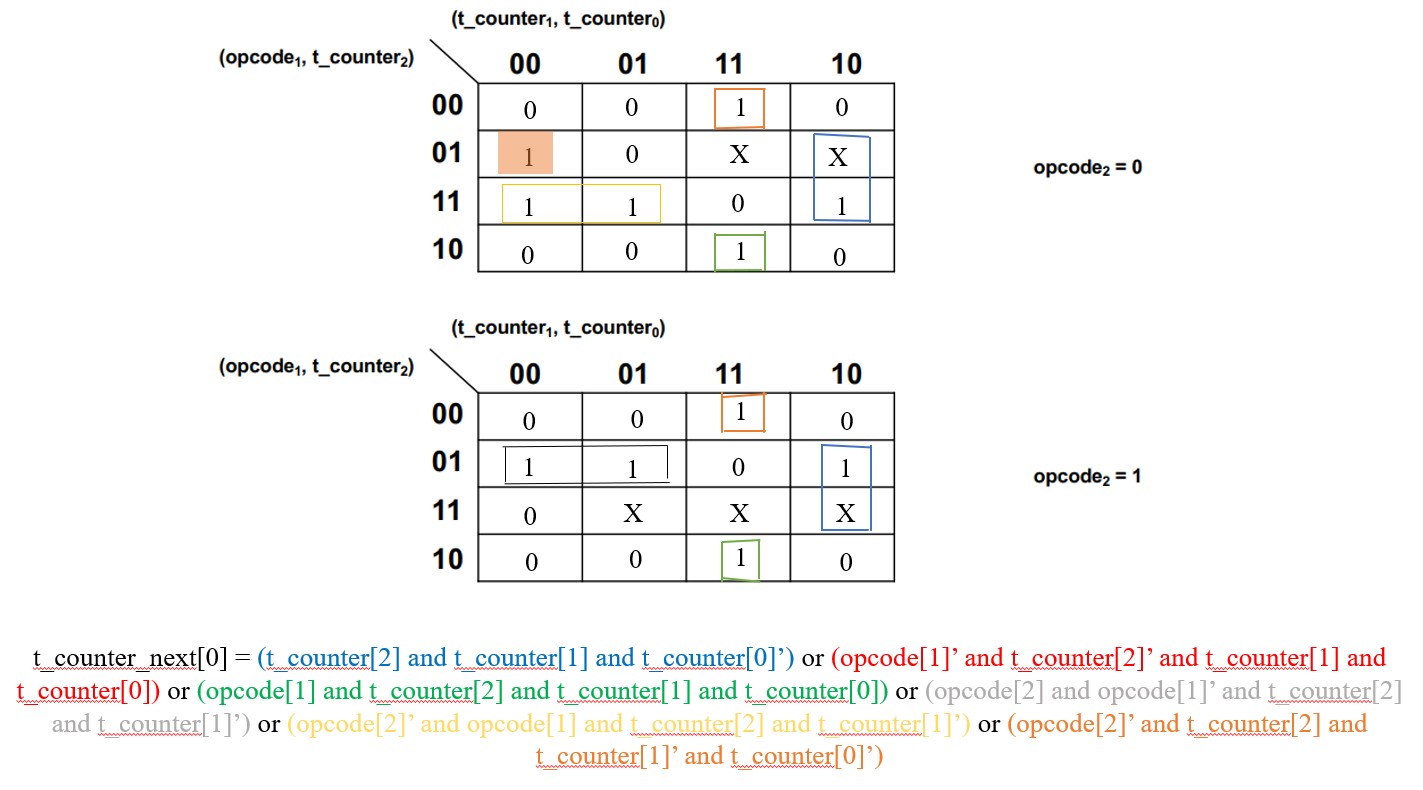
The next state logic required Karnaugh maps in order to develop the schematic for each of the three next state variables. Figure 3 represents the next state table and was thus used to develop the Karnaugh maps.

****

**Figure 6: Karnaugh Map for t\_counter\_next[0]**

****

**Figure 7: Karnaugh Map for t\_counter\_next[1]**

****

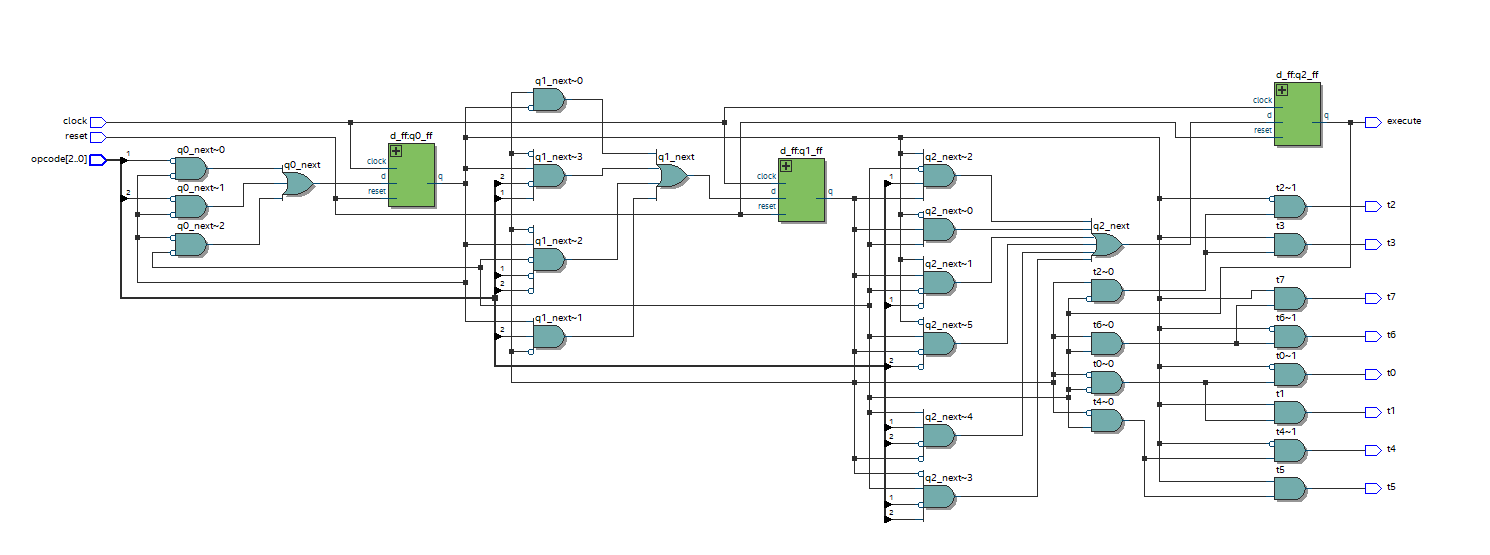
**Figure 8: Karnaugh Map for t\_counter\_next[2]**

Figures 6-8 show the k-maps for each of the next state variables using the table from Figure 3. The d-flipflops input *d* were assigned the values of their respective next state variables so that they can be loaded when transitioning to the next state. The next state logic itself was created with combinational circuits.

Output Logic

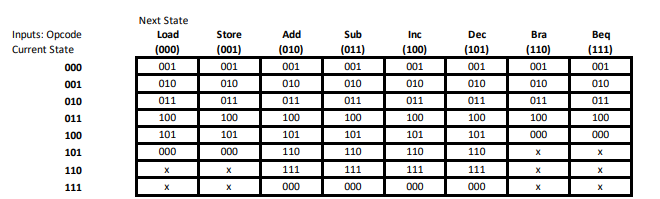
The final step to create the output logic was primarily combinational logic. The *t0*-*t7* output was created using Figure 4. A K-map was not needed for designing the output logic for *t0*-*t7* because the outputs were associated with the combination of the state variables. In other words, if the state variables were 010, then the FSM would output *t2* as 1. Simple AND and OR gates were used to create the outputs from *t0-t7*.

In the case for the *execute*, a K-map was not needed because *execute* yield 1 of the combination of the state variables yielded a decimal value greater than or equal to 4. In other words, *execute* was equal to the most significant bit *t\_counter[2]* or *q[2]* in the case of my particular implementation of the instruction sequencer.

****

**Figure 9: Schematic for Instruction Sequencer**

**Verification**

****

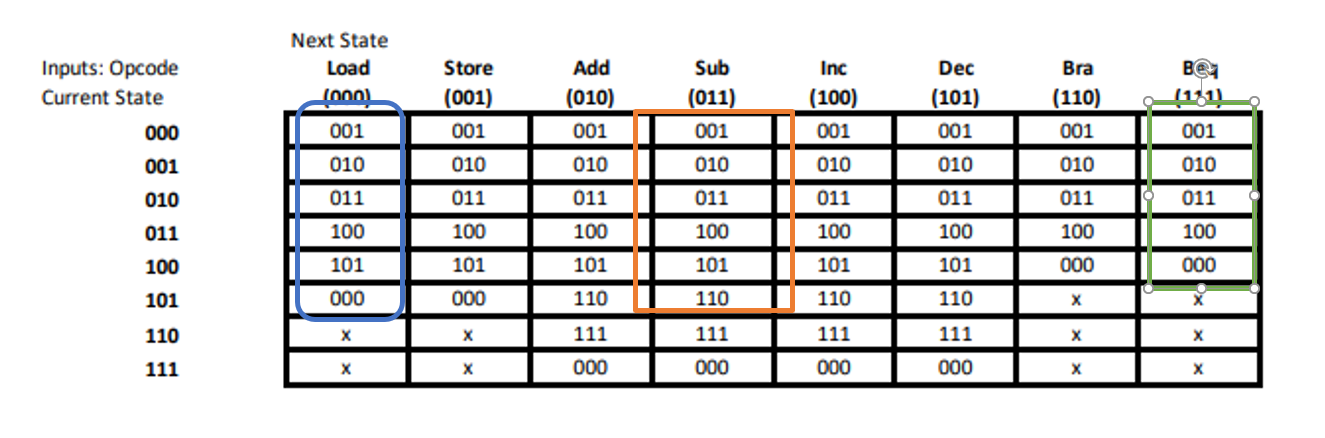
**Figure 10: Next State Transition Table (from Document)**

Figure 9 highlights the schematic for the instruction sequencer. Verification for the instruction sequencer was done in two ways: signal waves and assertions. The results of the signal waves were compared with Figure 10 and the assertion provide a quicker way to evaluate the correctness of the schematic.

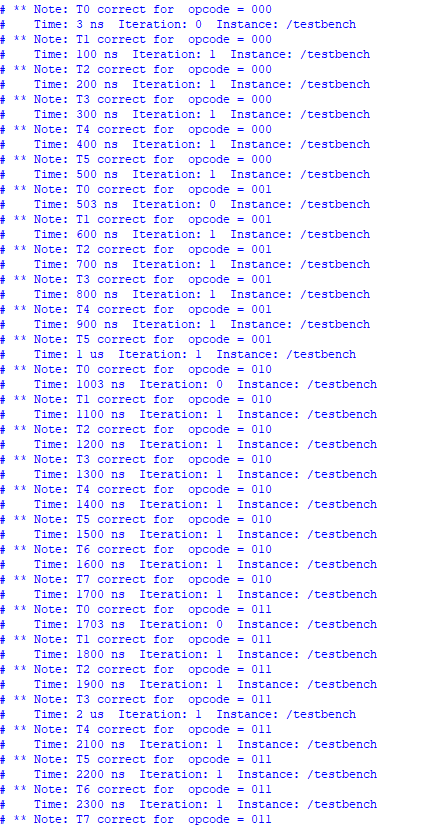
Timeline

Description automatically generated

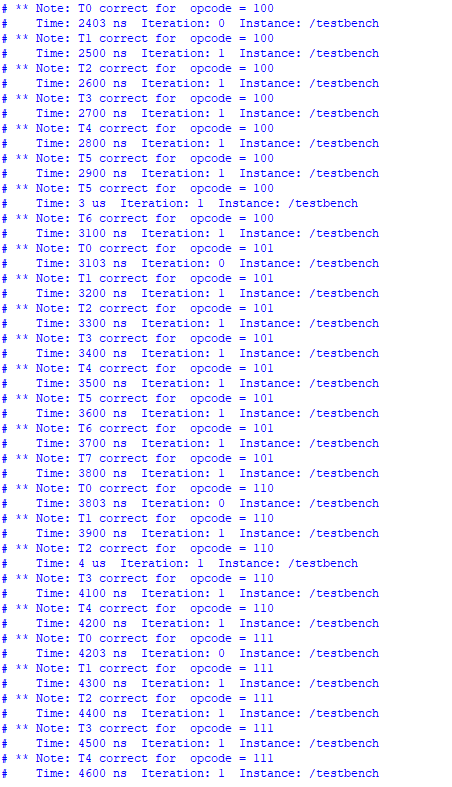
**Figure 10: Annotated Verification**



**Figure 11: Verification with Signal Waves and Comparison with Table**

****

**Figure 12: Verification for Opcodes 000 – 011**

****

**Figure 13: Verification for Opcodes 100 – 111**

As evident from the assertion statements and verification, the instruction sequencer successfully fulfilled the constraints and requirements of the document. Therefore, the instruction sequencer was successfully designed.