1.6 Microprogrammed Design [4 points]

In lecture, we discussed a design principle for microprogrammed processors. We said that it is a good design principle to generate the control signals for cycle $N+1$ in cycle N.

Why is this a good design principle? Be concise in your answer.

Answer: Likely keeps the critical path short (it follows the critical path design principle).
Explanation: By generating the control signals in advance, we can make the critical path of the circuit likely shorter. Shorter critical path can increase the frequency of the processor.

1.7 Processor Performance [10 points]

Assume that we test the performance of two processors, A and B, on a benchmark program. We find the following about each:

- Processor A has a CPI of 2 and executes 4 Billion Instructions per Second.
- Processor B has a CPI of 1 and executes 8 Billion Instructions per Second.

Which processor has higher performance on this program? Circle one.
Recall that CPI stands for Cycles Per Instruction.

A. Processor A
B. Processor B
C. They have equal performance
[✓ D. Not enough information to tell]

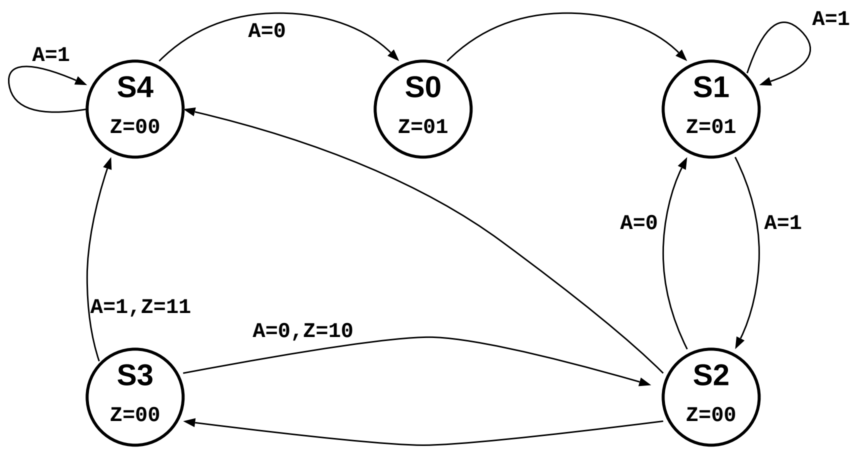
Explain concisely your answer in the box provided below. Show your work.

Answer: Neither of these metrics nor their combination provide execution time.
Explanation: Although information about the CPI and the instructions/second is provided, it is not enough to reason about the processors' performance. The processors may support different Instruction Set Architectures, in which case the benchmark program will be compiled into a different assembly code. The fact that one of the processors execute more instructions per second does not necessarily mean that the processor makes more progress on the benchmark.

2 Finite State Machines

This question has three parts.

(a) [20 points] An engineer has designed a deterministic finite state machine with a one-bit input (A) and a two-bit output (Z). He started the design by drawing the following state transition diagram:



Although the exact functionality of the FSM is not known to you, there are **at least three mistakes** in this diagram. Please list **all** the mistakes.

There are four problems with this diagram

(a) Most states have a Moore labelling (output state in the bubble), one has a Mealy type labelling (output given with input transitions) (5 points)

(b) There are two different transitions both with $A = 1$ from state $S1$. What will happen with $A = 0$ is missing (5 points)

(c) There are two different transitions from state $S2$, without labeling which input triggers them (5 points)

(d) There is no reset state (5 points)

(b) [25 points] After learning from his mistakes, your colleague has proceeded to write the following Verilog code for a much better (and **different**) FSM. The code has been verified for syntax errors and found to be OK.

```
module fsm (input CLK, RST, A, output [1:0] Z);

  reg [2:0] nextState, presentState;

  parameter start   = 3'b000;
  parameter flash1 = 3'b010;
  parameter flash2 = 3'b011;
  parameter prepare   = 3'b100;
  parameter recovery = 3'b110;
  parameter error = 3'b111;

  always @ (posedge CLK, posedge RST)
     if (RST)  presentState <= start;
     else      presentState <= nextState;

  assign Z = (presentState == recovery) ? 2'b11 :
             (presentState == error)    ? 2'b11 :
             (presentState == flash1)   ? 2'b01 :
             (presentState == flash2)   ? 2'b10 : 2'b00;

  always @ (presentState, A)
    case (presentState)
      start    : nextState <= prepare;
      prepare  : if (A) nextState <= flash1;
      flash1   : if (A) nextState <= flash2;
                 else   nextState <= recovery;
      flash2   : if (A) nextState <= flash1;
                 else   nextState <= recovery;
      recovery : if (A) nextState <= prepare;
                 else   nextState <= error;
      error    : if (~A) nextState <=start;
      default  : nextState <= presentState;
    endcase

endmodule
```