Problem 1: State and define the hazards presented by instruction level parallelism. For each one, indicate how it can be resolved.

1. Data Hazards

- *RAW (Data dependence) cannot use a value before it is computed. Resolve by forwarding or stalling*
- *WAW (Output dependence) cannot write a value if a logically preceding instruction might overwrite it Resolve by pipeline design (in-order op-fetch + in-order WB), stalling on potential write to pending register, or renaming*
- *WAR (anti-dependence) cannot write a value before logically preceding instruction reading the previous value have done so. Resolve by pipeline design (in-order issue with in-order operand fetch), stalling or renaming.*

2. Structural Hazards

Attempt to use the same hardware resource for two different purposes at once. Resolve by adding hardware resources (as design time) or stalling

3. Control Hazards

Cannot determine the control flow until the condition of the branch is resolved. Resolve by stalling. Mitigated by predicting and discarding miss-predicts.

Problem 2. Your current version of ZippyCAD runs through a benchmark design in 43 minutes on your ZIPS10 computer. ZIPS has a new model that they are offering to sell to you. ZIPS30 is a scalar machine like ZIPS10, but 3 times faster. Or you can get the ZIPS1010 vector upgrade that performs vectorized code at 10 times the performance of ZIPS10. You know that ZippyCAD spends a lot of time in its numerical library, so you are intrigued. *How much of ZippyCAD would need to vectorize for the ZIPS1010 to beat the ZIPS30?*

$$SUvector = \frac{Tscalar}{Tvector} = \frac{1}{(1-f)+f/x} = \frac{1}{1-f+f/10} \geq 3$$

$$f \geq 20/27 \approx 74\%$$

Problem 3: Diagram a correlating branch predictor that uses 3 bits of global information, 5 bits of local information, 3-bit saturating counters. Include the dimensions of all the machine data structures. Diagram the state machine. Give brief pseudo code for how this operates.

Index a 256 x 3-bit RAM using the low 5 bits of the PC concatenated with 3 bits from the branch history shift register. The shift register has as input the branch direction. The book draws this as a two dimensional table. Same thing.

On decode, combine the 8 bits as above, index the prediction table, read the value and predict based on the most significant bit.

Retain prediction table index into the execute stage. At that point, update the table entry according to the saturating counter state-transition-diagram. Write the result to the table. Shift the taken/not-taken bit into the shift history.

There are two reasonable state machine. Both have state encodings going from 000 at the bottom to 111 at the top. The one on the right provides some hysteresis. You don't get stuck toggling between 011 and 100. Even a branch that alternates between taken and not-taken will get predicted right about half the time.