1. Is making faster and more powerful IC’s simply a matter of reducing the feature size (currently 22–45 nanometers) and using more transistors? What role does capacitance \( C \) play with regard to propagation delay and power consumption?

2. A program spends 70% of its time in one, slow loop. A programmer has the option of speeding up the non-loop code by a factor of 100 or by speeding up the loop code by a factor of 5. Which is better and why?

3. Complicated instruction set (CISC) computers usually offer the programmer the opportunity to do arithmetic directly on memory locations. For example, the VAX allows \( \text{ADD } X, Y, Z \) where \( X, Y, \) and \( Z \) are memory locations. How would this computation be done by a load-store (or register-register) architecture? What level of parallelism is possible for each of these architectures?

4. Which of the following addressing modes do most RISC processors support, Register, Immediate, Displacement, Indexed, Memory Indirect, and why?

5. What are the advantages of having (almost all) instructions fit in one (aligned) machine word? What are the disadvantages?

6. MIPS Assembly (along with most other assembly languages) supports pseudo-instructions such as \( \text{move}, \text{blt}, \) and \( \text{la} \). What is the purpose of this feature and how does the \( \text{at} \) register fit into this?

7. In a function call, room is left on the stack for the first four parameters but they are actually placed in the \( \text{a0, a1, a2, a3} \) registers. Why?

8. In the MIPS processor line, what is the \( \text{gp} \) register used for?

9. Early MIPS processors (with no floating point operations) used a 5-stage integer pipeline (\( \text{IF, ID, EX, MEM, WB} \)). What specifically is done in each stage, considering only I-format and J-format instructions?

10. What is a pipeline stall, what causes it, and how does it result in loss of performance?

11. Why is the L1-Cache split into two separate sections, one for instructions and one for data? What does the term “program footprint” refer to?

12. What is a structural hazard and what can be done to eliminate such a hazard?

13. Data hazards can be classified as RAW, WAW, or WAR. In the original MIPS 5-stage integer pipeline (with no floating point operations) only one of these is a problem. Which one is it and how is it usually handled?

14. What are the main reasons why the MIPS designers chose to implement a branch delay slot (so that the instruction right after the branch is always executed)?

15. Consider the code below. Assume branches are done in the ID stage of the pipeline with a branch delay slot. You are allowed to assume forwarding. Can you reorder the code to eliminate stalls without using a \( \text{nop} \) in the branch delay slot? Draw the timing diagram for execution.

\[
\begin{align*}
\text{addi } &\text{t0, t0, 1} \\
\text{lw } &\text{t1, 28(t2)} \\
\text{beq } &\text{t0, t1, target} \\
\text{add } &\text{t3, t1, t2}
\end{align*}
\]

16. When floating point support is added to the MIPS 5-stage integer pipeline,
the DQ stage is expanded into a 3-stage decode queue (DQ) and the EX stage is replaced with multiple functional units (executing in parallel – see floating.pt.txt). We could call this the MIPs statically scheduled floating point design. Why are these changes made? What limitations does this design (still) have?

17. In order to overcome the limitations of static scheduling we introduce a “scoreboard” with reservation stations for instructions which are in the process of being executed. This is essentially Tomasulo’s design. Which problems does this design solve? Which problems remain?