1. Explain the function of the following components in a PVM (in a single line): emulation engine, code cache manager, profile database, OS call emulator.

2. Explain intrinsic and extrinsic compatibility (with examples).

3. Explain the need to define a compatibility framework. What are the main components of the compatibility framework we employed during our discussions?

4. Give and explain code transformation examples where *trap*, *register*, and *memory* compatibility may not be maintained after the transformation.

5. *Name* the two methods of memory state mapping in a PVM. Explain their advantages and disadvantages. Which method is applicable in all PVM situations?

6. What aspects of the *memory architecture* does a PVM have to emulate?

7. Why does the PVM runtime memory need protection from the guest process? Describe one approach of providing this protection.

8. What is self-referencing code? What is self-modifying code? Why do these constructs present problems to binary translation in a PVM?

9. Explain the need for *staged emulation*.

10. Define: precise exceptions. Give an example each of ABI-visible and ABI-invisible exception.

11. Define *traps* and exceptions. Does their nature cause them to be implemented differently in a PVM? If so, how?

12. Is providing perfect OS call emulation always feasible for all guest and target combinations? Explain.

13. What is a *code cache*? How is a code cache different from ordinary caches?

14. Describe the algorithm and explain the advantages and drawbacks of the following code-cache replacement algorithms: (a) LRU, (b) Cache flush, (c) FIFO, (d) Coarse-grained FIFO.