1. Suppose a CPU designer wanted to allow an instruction which would move the contents of a register to one memory location and increment a second memory location (e.g. `movinc R1, X, Y`). What would be the problems with the CPU supporting precise exceptions?

2. Why is a load-store queue used in dynamic scheduling? For example, are the instructions `lw $t2,24($sp)` and `sw $t3,-16($fp)` order dependent?

3. Why is branch prediction done? What is a (2,2) branch predictor? Give examples of a case where branch prediction is almost 100% effective and a case where branch prediction is no better than random guessing.

4. A branch target cache keeps track of previous branch instructions and both the prediction (yes/no) and the predicted target address. However, it has more uncertainty than doing branch prediction at the end of the DQ-cycle with, say, a (2,2) branch predictor; why, then, is it used?

5. What is register renaming? Where would it usually be used and what hazards (e.g. WAW, WAR, or RAW) are eliminated by its use?

6a. In Tomasulo’s design instructions are issued to Reservation Stations (RS) for (possibly) out-of-order execution. An instruction will begin execution if it is not waiting on an unresolved branch and if its source values and a corresponding functional unit are both available. How do L1-data cache misses affect its performance?

6b. Why is speculation and a Reorder Buffer (ROB) usually added to Tomasulo’s original design?

7. In a dynamically scheduled RISC processor with speculation, ROB, and multiple issue, the term *window* usually refers to the number of concurrent instructions which have been issued but have not yet committed. This number is usually in the range 32-128. What are some things which would keep a designer from using a much larger window, e.g. 2K?

8. The Intel Nehalem/i7 architecture is CISC by original design but its current internal implementation is very similar to dynamically scheduled RISC designs (with a scoreboard and ROB). Why add this complication to the design since it will certainly increase both the total number of transistors as well as the die size?

9. What is the argument for trying to discover and exploit parallelism at the compiler level rather than waiting until run time? What is “predicated code?”

10. What does the term “hoist a load instruction” refer to? Can this always be done? Why or why not?

11. In order to exploit multiprocessor/multi-core systems it is usually necessary to write *multi-threaded* programs. Discuss the strategy you would use to structure a multi-threaded program (using POSIX threads, mutexes, etc.) which would use 4 cpus/cores to multiply two matrices. Would you expect the speedup to be 4 times that of a single-threaded program? Why or why not?

12. In multiprocessor/multi-core designs which are tightly coupled (2-16 processors), it is necessary to implement some algorithm and mechanism for L2 cache coherency. How is this most commonly done?

13. In a multiprocessor/multi-core design using the MESI cache coherency protocol, where are the potential bottlenecks? Do these concern the programmer? Why or why not?
14. How do TLB Cache misses affect performance? Does a programmer have any control over TLB Cache misses?

15. In a typical page table entry there is a physical page number, a “valid (in-memory)” bit, a “modified” bit, a “lock-in-core” bit, an “uncacheable” bit, and a “copy-on-write” bit. What is each of these used for?

16. Assume that we have a dynamically scheduled RISC processor using Reservation Stations and a ReOrder Buffer, issuing instructions in-order, executing instructions out-of-order, and committing instructions in-order. Consider the following assembly language program fragment:

```
  lw $f2, 24($fp)
  lw $f0, 64($fp)
  mult $f6, $f0, $f2
  add $f4, $f2, $f12
  sub $f0, $f4, $f8
  add $f2, $f4, $f10
```

16a. Assume that the second instruction above is still executing due to a L1 and L2 cache misses on the address 64($fp). Which of the remaining instructions can finish execution (but not necessarily commit) and obtain their result?

16b. Assuming that the second instruction above is still executing due to L1 and L2 cache misses on the address 64($fp). Which of the remaining instructions can commit?