1a. Some scientific machines have a very large level-3 (L3) cache, so that if the L1, and L2 caches are too small to be effective, some additional speedup can be obtained. This L3 cache is slower than the other caches, though. Suppose that transfers to the L3 cache take 25 nSec., transfers to main DRAM memory take 100 nSec, and the buses are 32-bits wide (4-bytes) with each word able to be used. What transfer rates can we expect with 0% L3 cache hits, with 50% L3 cache hits, with 100% L3 cache hits (give your answer in Mbytes/sec)?

1b. Rather than deal with the complexity of caches, why don’t hardware designers forget about slow dynamic RAM (50-100 nSec access time) and fabricate main memory entirely out of fast (5–6 nSec. access time) static RAM?

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

3. 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?

4. Complicated instruction set (CISC) computers usually offer the programmer the opportunity to do arithmetic directly on memory locations. For example, the VAX allows $ADD X, Y, Z$ where $X, Y, \text{ and } Z$ 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?

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

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

7. MIPs Assembly (along with most other assembly languages) supports pseudo-instructions such as move, blt, and la. What is the purpose of this feature and how does the $at$ register fit into this?

8. In a function call, room is left in on the stack for the first four parameters but they are actually placed in the $a0, a1, a2, a3$ registers. Why?

9. In the MIPs processor line, what is the $gp$ register used for?

10. In Unix/Linux systems a trap such as a CPU exception (e.g. divide by zero, invalid memory access, etc.) is converted to a signal which is then sent to the process. Why is it done this way?

11. The canonical model of a process requires a minimum of three data segments: code, data, and stack. What is the function of each of these segments with regard to execution of the process? How does MIPs keep track of these segments? What permissions (read, write, execute) are usually put on each of the segments? What happens if the process, after invocation, creates additional threads, or allocates additional memory with new or malloc?

12. Using pointers to reference data (e.g. in the case of linked lists or trees) provides the programmer with a great deal of flexibility. Are there any disadvantages to using
pointers (as, for example, against using arrays) in the case of time-critical code?

13. It is often necessary for the CPU (especially when executing operating system code) to be able to exchange the contents of a register variable and a memory variable as one atomic operation. Will the following code do this? If not, how would you fix it?

   ```
   lw $t1, 0($s1)
   sw $t0, 0($s1)
   move $t0, $t1
   ```

14. Describe the process of creating an executable (i.e. compiling, assembling, linking) from a C-program source file. What is the difference between a statically-linked executable and a dynamically-linked executable? What are the advantages and disadvantages of dynamic linking?