Processor pipeline (20 points)
(a) (7 points) If a processor encounters frequent stalls due to write-after-write (WAW) hazards, it may be a sign of a poorly-designed pipeline. Diagnose the possible problem(s) that may be causing such stalls and suggest solution(s).

(b) (13 points) Consider a system with a write-through L1 cache and a dual-banked L2 cache. The access latency (not pipelined) of each L2 bank is 10 cycles. Some workloads (say, Class A workloads) have an even distribution of accesses across the two L2 banks. Other workloads (say class B) workloads achieve a 70%/30% split across the two L2 banks. What is the maximum IPC that the processor can sustain if 10% of instructions are stores for each workload class?

Memory Hierarchy and Virtual Memory (20 points)
(a) (7 points) In a system with 4KB virtual memory pages, what should the associativity of a 16KB L1 cache be if we want to use virtual indexing with physical tagging while also avoiding any restrictions on virtual-to-physical page mappings.

(b) (6 points) Briefly explain why (1) L1 caches and TLBs are typically split designs (with separate I- and D- caches/TLBs) whereas (2) lower level caches and TLBs are unified. Explain both trends.

(c) (7 points) Explain why victim caches have diminishing returns with increasing associativity of the main cache.

Multicore (30 points)
(a) (5 points) Explain why the following code could potentially (but not necessarily) have poor performance on a dual core, bus-based shared-memory multiprocessor with coherent caches (based on the MESI coherence protocol). Assume that the processor wordsize is 4 bytes and that the cacheblock size is 64 bytes.

```c
struct foo {
    int a, b;
};
struct foo q_array[1024];
```

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>for(int i=0; i&lt;1024; i++) {</td>
<td>for(int j=0; j&lt;1024; j++) {</td>
</tr>
<tr>
<td>q_array[i].a =</td>
<td>q_array[j].b =</td>
</tr>
<tr>
<td>sqrt(q_array[i].a);</td>
<td>sqrt(q_array[j].b);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>
(b) (5 points) Clearly state the conditions under which there are no performance problems with the code.

(c) (10 points) Briefly describe one change you could make to the code (NOT the data layout) to guarantee the elimination of the potential performance problem. Your modified code must maintain load balance on the two cores.

(d) (5 points) Explain one change you would make to the data-layout (NOT the code) to guarantee the elimination of the potential performance problem. Your modified code must maintain load balance on the two cores.

Fundamentals: (30 points)
A newly developed RAM technology called PrimeRAM is proposed as an alternative to DRAM. PrimeRAM’s access latency is not uniform; it varies by address. The delay of accessing an address in PrimeRAM depends on the delay since the previous access to that address. In general, it is faster to access a PrimeRAM address if it has been recently accessed. Specifically, if a memory block (of arbitrary size) was accessed at time T1, and then accessed again at time T2, the access latency for the second access is equal to CEILING((T2-T1)/100) cycles. In addition, the access latency never exceeds Tmax cycles. For example, access 1 occurred at T1=100 and incurs Tmax cycles of access latency; access 2 occurs at T2= 200 and incurs a 1 cycle latency.

Q1. (15 points) Given PrimeRAM’s behavior, briefly explain how you would redesign current memory hierarchies to exploit PrimeRAM’s characteristics. (Hint: Think about the characteristics offered by traditional memory hierarchies.)

Q2. If PrimeRAM’s access-latency behavior is limited to individual word-sized memory locations (instead of arbitrary block sizes), answer the following questions about the resulting implications.

   Q2a. (10 points) Does the PrimeRAM based memory system from Q1 achieve comparable performance as the traditional SRAM-based cache hierarchy? Briefly explain why or why not?

   Q2b. (5 points) Based on your answer above, is it realistic to expect that the PrimeRAM-based system will benefit applications broadly. If not, how would your design change without adding buffering or caching?