This exam has nine (9) problems. You should submit your answers to six (6) of these nine problems. You should not submit answers for the remaining three problems. Since all problems are equally weighted (carry the same number of points), carefully choose which six problems you will answer.

Write your answers/solutions clearly and legibly. Illegible answers will be treated as wrong answers. You can attach additional sheets of paper if you wish, in which case you should clearly mark on top of each page which problem the sheet corresponds to, and you should not use the same sheet for two problems (it is fine to use multiple sheets per problem). Although you can use additional paper, try to keep your answers and solutions short and to the point.

Good luck!
Problem 1: Multiprocessor Consistency

Different consistency models provide different tradeoffs between performance and ease of programming for multi-processor machines. Consider this tradeoff between sequential consistency (SC) and total store ordering (TSO) consistency.

a) Which of the two (SC or TSO) provides better performance and why? Provide an example execution on a modern out-of-order superscalar processor that shows this performance difference. You should show the actual program order, the actual execution order, and when the consistency model causes delays.

b) Which of the two (SC or TSO) is easier to program and why? Provide example code that behaves as expected in one of these consistency models but not in the other, and explain its “unexpected” behavior.

c) Explain what kind of processor is needed (in terms of pipelining, out of order execution, caching, branch prediction) to guarantee that SC and TSO always behave in the same way? Hint: consider what kinds of ordering can be different in SC and TSO and which processor features can result in such ordering.
Problem 2: Approaches to Shared-Memory Multiprocessing

Describe the difference between distributed shared memory (DSM), symmetric multiprocessing (SMP), and simultaneous multi-threading (SMT, a.k.a Hyperthreading). In particular:

a) What are the differences in terms of hardware cost, design complexity, scalability to many parallel threads, and ease of achieving good performance.

b) A multi-programming workload is one in which a parallel machine is used to run independent programs. Multi-programming workloads raise the issue of fairness, because one thread can get good performance but cause the other to have poor performance. Describe an example of how a thread can be “unfair” if it runs in an SMT machine, but fair if it runs in a DSM or SMP machine. Also describe an example of a thread that is “unfair” in SMT and SMP, but not in DSM machines.
Problem 3: Interactions between Hardware Structures

Modern processors use multiple cache-like structures, such as instruction caches, data caches, translation lookaside buffers (TLBs), and branch prediction tables. These structures normally keep different information and, as a result, data updates in one structure do not need to be propagated into the other. However, some of these structures can interact in this way. For example, with self-modifying code writes in the data cache can affect the instructions in the instruction cache.

a) Describe all interactions of the four structures mentioned above (i-cache, d-cache, TLB, branch predictor) can interact and what kind of program or system behavior causes this interaction.

b) For each type of interaction from part (a), describe a high-performance solution (cost is a secondary concern here) that allows correct program and system operation.

c) For each type of interaction from part (a), describe a low-cost solution (performance is a secondary concern here) that allows correct program and system operation.
Problem 4: Dynamic Scheduling

One formulation of the classic Tomasulo-style dynamic scheduling algorithm assumes wakeup, arbitration, execution, and result and tag broadcast all occur in the same cycle. However, modern processors decouple the scheduling procedure (wakeup+select+tag broadcast) from the execution procedure (operand read+execute+result bypass).

Consider a baseline processor with a single cycle schedule-to-execute (S2E) delay (if the scheduler determines that an instruction is ready on cycle N then the instruction will start execution on cycle N+1). What is the impact (pros and cons) on the design of the microarchitecture if the S2E delay increases to K cycles? In particular, fully address:

a) Why and how does performance change in terms of both clock frequency and IPC (and you must discuss the impact of variable latency instructions such as loads).

b) Changes in hardware structures required to implement multi-cycle S2E.

c) Power, both dynamic and static. Specifically relate the components of the power equations to the microarchitectural changes discussed in parts (a) and (b). Discuss how these changes in power in turn affect the temperature of the chip.

d) Some processors read instruction operands from the physical register file prior to inserting these instructions into the scheduler/reservation stations (e.g., Pentium-Pro, Pentium-M), while other processors read operands from the physical register file after scheduling as the instructions issue from the reservation stations (e.g., Pentium-4, 21264). Discuss why or why not a longer S2E delay makes sense for each of these pipeline organizations. Justify your answers in terms of performance, power and complexity.
Problem 5: Branch Prediction

Computer architects have been researching better ways to perform branch prediction for decades now. Many of the more recent proposals involve multiple tables of counters or weights, complex hash functions, and additional logic (e.g., trees of adders) to compute the final prediction. As a result, these branch predictors are quite accurate, but require a lot of area, power and latency.

a) Discuss the impact on the front-end pipeline design of a branch predictor that requires multiple cycles to provide a prediction. In particular, discuss the hardware changes and the dependence of the performance impact on the pipeline depth, the issue width and maximum instruction window size.

b) Under what conditions does it make sense to implement a more accurate predictor even if it increases the branch predictor power consumption? Consider the impact on performance, power, the underlying microarchitecture, interaction with voltage/frequency scaling, and thermal hotspots (in particular the throttling of the processor when the chip gets too hot).
Problem 6: Memory management

Efficient management of memory is crucial to performance be it a uniprocessor or a multiprocessor. Historically, this is one place where the symbiosis between hardware and system software can be seen pretty clearly. For example, in most parallel machines while the hardware may be responsible for cache coherence, the system software is usually responsible for TLB consistency.

As multicore architectures become more common place it is natural to revisit this boundary between hardware and software once again. This question pertains to efficient memory management in the context of multicore architectures. With respect to efficient memory management,

a) What problems remain the same compared to uniprocessors?

b) What problems are new compared to uniprocessors?

c) What problems remain the same compared to SMPs?

d) What problems are new compared to SMPs?

e) Give your ideas on how you will design an efficient memory manager for a parallel operating system running on an SMP built out of multicore parts. Be sure to address the partitioning of responsibilities between hardware and software with qualitative (and if possible) quantitative reasons.
Problem 7: Core OS Synchronization

There have been several proposals for higher-level interprocess/interthread synchronization dating back to 70's (such as monitors, conditional critical regions, and Roy Campbell's path expressions).

a) First, give the pros and cons of adopting such proposals in system software from an implementation point of view (discuss both architectural and OS related issues).

b) Second, discuss the relative merits of these constructs from the point of view of an application programmer.

c) Third, enumerate the mechanisms that are typically found in modern operating systems such as Sun Solaris and Microsoft XP for thread synchronization. In particular discuss why such operating systems chose to have or not to have such high-level synchronization constructs. Once again your discussion should address architectural and OS related issues.
Problem 8: Prefetching

Prefetching is an effective technique to hide memory latency. Distinguish between hardware and software prefetching illustrating pros and cons of both with examples. It is proposed to design a joint hardware/software mechanism to increase the effectiveness of prefetching and to reduce bad side effects such as cache pollution. Which are the key factors you will consider and provide a design that works as a software solution with hardware support. Show details of which information you will keep in hardware and how will you use it to make prefetching decisions. You may assume you have mechanisms already in place to detect critical load/stores.
Question 9: Hardware/Software Interaction

Pointer-intensive codes are heavily memory bound. Pointer-chasing in data structures such as linked lists, trees, queues, etc. leads to bursts of memory accesses that also have dependencies (can not be re-ordered). Show a hardware solution that tackles this problem by keeping some information - ie which may reduce pointer chasing or may have allow some re-ordering through speculation. To make the technique more effective one could also add some compiler generated information (such as points-to sets found through static analysis). Show the benefits of such an approach and how will you provide effective mechanisms to use such information?