SPRING 1997
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH. D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination

Monday, February 3, 1997
3:00 – 7:00 PM
2365 CSS

GENERAL INSTRUCTIONS:

1. Answer each question in a separate book.
2. Indicate on the cover of *each* book the area of the exam, your code number, and the question answered in that book. On *one* of your books list the numbers of *all* the questions answered. *Do not write your name on any answer book.*
3. Return all answer books in the folder provided. Additional answer books are available if needed.

SPECIFIC INSTRUCTIONS:

Answer *all* of the following *six* questions. The questions are quite specific. If, however, some confusion should arise, be sure to state all your assumptions explicitly.

POLICY ON MISPRINTS AND AMBIGUITIES:

The Exam Committee tries to proofread the exam as carefully as possible. Nevertheless, the exam sometimes contains misprints and ambiguities. If you are convinced a problem has been stated incorrectly, mention this to the proctor. If necessary, the proctor can contact a representative of the area to resolve problems during the *first hour* of the exam. In any case, you should indicate your interpretation of the problem in your written answer. Your interpretation should be such that the problem is nontrivial.

1. Fast Multiplication

Consider building a combinational multiplier that accepts two unsigned 8-bit integers **A** (a8,a7,..,a0) and **B** (b8,b7,..,b0) and produces an unsigned 16-bit product **C** (c16,c15,..,c0).

(a) Define the interface to an n-bit *carry propagate* adder. Use your carry propagate adder and simple gates to build an implementation of the above multiplier.

(b) Define the interface to an n-bit *carry save* adder. (Recall that a carry save adder takes three numbers and produces two outputs, one formed with the sums and the other with the carry-outs.) Use your carry save adder, your full adder, and simple gates to build an implementation of the above multiplier.

(c) Explain which implementation is faster using approximate critical path estimates.

2. Precise exceptions and out-of-order execution

Executing instructions out-of-order potentially increases performance by using function units even though an earlier instruction (in sequential order) is blocked on a data or resource dependence. Unfortunately, out-or-order execution can make recovering from exceptional conditions, e.g., arithmetic overflow, difficult.

(a) What is the definition of precise interrupts? What is the problem that they solve?

(b) How does a reorder buffer help implement precise interrupts?

(c) What is a future file and why might it be useful?

(d) How does the use of register renaming, ala the IBM RS/6000, affect the options for implementing precise interrupts?

3. Multiprocessor communication

Shared memory and message passing are two fundamental programming abstractions for parallel programs. Similarly, shared memory and message passing are two common sets of mechanisms for communicating between processors in a multiprocessor.

(a) Explain how the message-passing abstraction can be implemented on top of the shared-memory communication mechanisms.

(b) Explain how the shared-memory abstraction can be implemented on top of the message-passing communication mechanisms.

(c) Does it ever make sense to do either of b) or c)? Why or why not? Discuss the issues.

4. TLB Design

A translation lookaside buffer (TLB) caches recently-used virtual to physical translations. A "conventional" TLB from recent processors might have be fully associative and have translations for 64 4K-byte pages.

(a) What trends are making such a conventional TLB inadequate?

(b) Discuss the pros and cons of improving on a conventional TLB by (i) making it larger or (ii) building a two-level TLB (like an L1 and L2 cache).

(c) Discuss pros and cons of the additional alternatives of (i) making the page size larger or (ii) supporting multiple page sizes.

(d) In what ways does the advent of wide out-of-order speculative processors affect TLB design?

5. Memory Consistency Models

Memory consistency models are used to define the interaction of loads and stores in a shared-memory parallel computer.

(a) What is release consistency? Give a short code sequence that, under release consistency, could result in an execution that could not occur under sequential consistency. Illustrate one such execution, showing how it violates sequential consistency.

(b) Discuss how and when a system's memory consistency model affects allowable compiler optimizations.

(c) In what ways does the advent of wide out-of-order speculative processors affect the trade-offs between stronger consistency models (e.g., sequential consistency) and weaker consistency models (e.g., release consistency)?

6. Memory Gap

Historically there has been a "memory gap" resulting from the lack of memory devices significantly faster than disks but significantly cheaper than DRAMs. Computer system designs have accommodated this gap. For example, when an access is to any part of the memory hierarchy above the gap (i.e., with access times less than the gap) most processors do not switch tasks while waiting for the data. An access to any part of the memory below the gap generally results in a task switch for the processor.

(a) How might the availability of a new memory device in the 10 microsecond range affect when and how task switches occur?

(b) How would the volatility (i.e., whether or not the device is volatile) affect the options for using this device?

(c) How might computer architecture have developed differently if there had been a smooth curve of memory products available along the latency/price curve?