**FALL 2005**
**COMPUTER SCIENCES DEPARTMENT**
**UNIVERSITY OF WISCONSIN—MADISON**
**PH.D. QUALIFYING EXAMINATION**

Computer Architecture
Qualifying Examination
Monday, September 19, 2005

## 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 non-trivial.

## 1. Fast Adder Implementation

A *two-way carry-select adder* computes the most-significant “half” of a sum assuming both possible carry-in values and then selects the actual result using the actual carry-out from the least-significant “half”.

Implementations may use gates, flip-flops, etc. Combinational logic may be done with equations. You will be graded by clarity of design first and speed second.

(a) *Without* using a carry-select adder, implement a fast *four-bit* two’s complement adder that can also be used as a building block for part (b). Show how many gate delays are in the critical path (assume AND, OR, and NOT gates are 1 gate delay, XOR / XNOR gates are two).

(b) Implement an *8-bit two-way carry-select adder* using gates and your solution to part (a). Show how many gate delays are in the critical path.

(c) Consider implementing a larger *3-way carry-select* structure. Is it always fastest for “way” to have the same number of bits? If not, how should the bits be distributed across “ways”? Justify your answer.

## 2. Value Prediction

Processor micro-architectures have long predicted the value of pending branch outcomes. Over the last decade, architects have begun considering techniques that also predict full 32- or 64-bit values. Assume a load-store architecture when answering this question.

(a) In what ways might values be predictable (e.g., what patterns)?

(b) What hardware mechanisms might an architect use to predict values?

(c) While an architect could seek to predict all inputs to all types of instructions, might it be more effective to concentrate on selective value prediction? If so, what would you select for prediction and why? If not, why not?

(d) When, if at all, do you expect future chips to employ value prediction (beyond branch outcomes)? Justify your answer.

## 3. Error detection and correction in registers and caches

In the 1990’s, microprocessors went from having no memory error detection or correction, to using parity to detect single-bit memory errors, to using SECDED ECC codes to detect double and correct single bit memory errors. Similarly, over the past decade, microprocessors have gone through the same progression for L2 caches. Some systems are even using ECC codes for L1 caches and registers.

(a) Discuss the tradeoffs with using no coding, parity codes only, and SECDED ECC codes at different levels of a *uniprocessor* cache hierarchy.

(b) How does the choice of code and / or code word size interact with other cache hierarchy management policies, e.g., writethrough vs. writeback and inclusion vs. exclusion.

(c) Discuss whether or not the recent shift towards chip multiprocessors using multi-threaded cores and shared caches affects these tradeoffs and interactions.

## 4. Systems on a chip

Chip multiprocessors (CMPs) are a recent industry trend towards putting multiple processor cores on a single die. An othogonal trend—called System On a Chip (SOC)—is to integrate system functions, such as memory controllers, I/O bus interfaces, and network interfaces. While SOCs have long been used in the embedded market, recent proposals argue for greater integration for high-performance computers.

(a) Discuss the pros and cons of integrating the memory controller on the same chip as the processor. Do these tradeoffs change as industry moves towards CMPs?

(b) Discuss the pros and cons of integrating the I/O controller onto the processor chip.

## 5. Trace Cache for Instructions

(a) What are the motivations and advantages of building a trace cache for instructions? Please consider both architectural and micro-architectural issues.

(b) What are the disadvantages of building a trace cache for instructions?

(c) Some recent processors have implemented trace caches (e.g., Pentium IV) and some have used conventional instruction caches. When, if at all, do you expect future chips to employ trace caches? Justify your answer.

## 6. Speculative Multithreading

The increasing density of semiconductor technology and demand for power efficiency has led most microprocessor vendors to embrace chip multiprocessors (CMPs), where a single chip implements multiple, potentially multithreaded, processor cores. While such designs promise great throughput for multithreaded server workloads, they may provide no better, or even substantially worse, single-threaded performance.

Speculative multithreading is one approach to harnessing the power of these multiple cores to accelerate the performance of a single-threaded application. For example, Multiscalar processors partition the application into *tasks* (e.g., loop iterations) and run subsequent tasks as speculative threads on separate cores. By overlapping execution of both independent and dependent tasks, Multiscalar exploits coarser-grain parallelism than conventional ILP processors.

A central challenge for speculative multithreaded processors is to preserve sequential execution semantics, especially in the presence of dependences between the speculative threads. All speculative multithreaded processors must resolve cross-thread memory dependences, but Multiscalar is one of the few speculative multithreaded processors that handles cross-thread dependences through registers.

(a) Discuss the fundamental similarities between detecting and resolving cross-thread register and memory dependences.

(b) Discuss the fundamental differences between detecting and resolving cross-thread register and memory dependences.

(c) Discuss the implementation and performance tradeoffs between supporting both register and memory dependences versus only memory dependences.