FALL 2007
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination
Monday, September 17, 2007

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. Designing FIFOs

You must design a FIFO that can hold 64-bit data.

a) Implement a 4-entry FIFO that can hold 64-bit data. The FIFO should implement the functionality of a conventional first-in-first-out data structure. You may assume only edge-triggered D-flip-flops (D, Q, and clock inputs only) and basic logic gates (AND, OR, and NOT), but you may build more complex logic using good hierarchical design. The FIFO accepts new input each cycle that data_in_valid is asserted, unless it is full (indicated by fifo_full). Data that is "inserted" into a full FIFO is ignored. The data_out signals always drive the data at the head of the FIFO (the oldest data). The head of the FIFO is popped when pop_data is asserted. An empty FIFO drives zeros on data_out and asserts fifo_empty. Popping an empty FIFO has no affect. Asserting reset makes the FIFO empty. All outputs should change only in response to the clock edge.

```
Inputs: data_in[63:0], data_in_valid, pop_fifo, clk, reset
Outputs: data_out[63:0], fifo_empty, fifo_full
```

b) Now consider implementing a large FIFO that can hold 128 or 1024 entries each being 64 bits. Discuss scalability tradeoffs of your design in section (a) and whether you will design a different structure if you had access to more design elements like register files, SRAMs, and CAMs.

2. Memory disambiguation

The von Neumann execution model dictates sequential semantics: namely that all instructions must appear to execute one at a time and in program order. Control dependences (e.g., branches) are the first challenge. To achieve instruction level parallelism, most high-performance processors predict branches and speculatively execute based on that prediction. But memory dependences (e.g., loads and stores to the same address) also present challenges to out of order instruction execution. All speculative processors must respect memory dependences to ensure sequential semantics. More aggressive processors use additional prediction and speculation mechanisms to expose more parallelism.

(a) Give a pseudo-assembly-code example that illustrates a case where memory dependences might limit ILP (in the absence of additional prediction and speculation).

(b) Discuss hardware mechanisms that suffice to detect and enforce memory dependence order in out-of-order processors

(c) Discuss how prediction and speculation of memory dependences can increase instruction level parallelism.

3. Future commercial processors

For the past 30 years, computers targeting scientific computation have focused on architectures that exploit data parallelism (e.g., vectors and massively parallel machines) and provide fast floating point computation. Conversely, computers targeting commercial workloads have focused on integer data, caching, random memory accesses, and I/O.

Some pundits predict that the rise in multimedia data types and the increased reliance on complex search will lead future commercial computers to look more like classical scientific computers.

(a) Argue why future commercial processors will look increasingly like scientific computers, leading to unified general purpose machines for both.

(b) Argue why commercial and scientific and scientific machines will actually diverge, leading to increasingly specialized machines for each target domain.

4. Virtual Memory & Address Translation

Most computer architectures today support virtual memory with demand paging of, at least, fixed-sized base pages. Most computer implementations support virtual memory address translations with translation lookaside buffers (TLB) logically before level one (L1) caches. Assume that future code and data will actively use much more virtual memory than today's.

(a) What problems might the future use of more virtual memory cause if these workloads were run on today's computers?

(b) Compare and contrast options for dealing with larger memory use, assuming that the changes must be transparent to user- and system-mode software.

(c) Compare and contrast options for dealing with larger memory use, assuming that the changes must be transparent to user-mode software only.

5. Message-Passing vs. Shared-Memory Programming

Programmers of concurrent applications can use the message-passing and shared-memory programming models. In the past, message passing has been more commonly used on clusters and shared-memory more common on symmetric multiprocessors (SMPs).

(a) Why do programmers choose to author programs in message-passing rather than shared-memory?

(b) Why do programmers choose to author programs in shared-memory rather than message-passing?

(c) How might tradeoffs change as programers target 32-thread chip multiprocessors (CMPs)? What about CMPs supporting 1,000 threads?

6. Architecture Trends and Component Design

As the complexity of computers and computer architecture evolve, the concentration of design effort and innovation tends to move to components that are higher up in the component hierarchy. For example, several decades ago a considerable amount of research and design effort was devoted to components that carried out arithmetic, e.g., ALUs. Even the design of integer ALUs was a significant research topic. With increasing resources, the design of such elemental resources becomes routine, and the effort shifts to the higher level design which consists of several such elemental components, in particular in designing and managing the interaction and use such elemental components.

For the past 3 decades, the microarchitectural components of a processing core, and the processing core itself, have been the subject of extensive research and design efforts. Today we have multi-core processors with a small number of processing cores. Before too long we can expect to see chips with hundreds of processing cores, i.e., more than the number of integer ALUs that are present in a chip today. Some have even gone so far as to say that a processing core is "the transistor of the future" implying that the microarchitecture of a processing core will be inconsequential.

(a) Argue why the architecture and design of a processing core will continue to be important in the next two decades of semiconductor technology.

(b) Argue why the architecture and design of a processing core will become less relevant, or even inconsequential, in the next two decades of semiconductor technology.