FALL 2002
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination
Monday, September 23th 2002
3:00 – 7:00 PM
Room 382 Mechanical Engineering

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 Comparator Design

You are to design a comparator to compare 2 n-bit, unsigned integers, $A = A_{n-1}...A_1A_0$ and $B = B_{n-1}...B_1B_0$. The comparator has 2 outputs, X and Y, where X = 1 iff A = B and Y = 1 if A > B.

You are to design the comparator using basic gates (i.e., AND, OR, XOR, and NOT) with a maximum fan-in of 5 inputs. You are to use good hierarchical design, first designing single-bit comparators and carry lookahead blocks. Then you will combine these blocks into a multi-bit comparator.

(a) Design a single-bit comparator. This comparator has 2 single-bit inputs (A_i and B_i) and a 2-bit output (X_i and Y_i). $X_i = 1$ if $A_i = B_i$ and $Y_i = 1$ if $A_i > B_i$. What is the delay through this single-bit comparator (assume basic gates have unit delay)?

(b) Design a 4-wide carry lookahead block. This unit has 4 sets of inputs, where each input is a 2-bit tuple $<g_i, p_i>$, and a 2-bit output <G, P>. Thus the 4 sets of inputs are $<g_0, p_0>$, $<g_1, p_1>$, $<g_2, p_2>$ and $<g_3, p_3>$, and the output is <G, P>. This unit is very similar to the lookahead unit used in a carry lookahead adder, the only difference being that it does not have outputs for carry signals.

(c) Design a 4-bit lookahead comparator using the above blocks. Which signals correspond to the output of the 4-bit comparison? Calculate the delay through this comparator.

(d) Repeat part (c) but for a 16-bit comparator.

2. Non-blocking Caches

(a) What is a non-blocking cache and how does it differ from a blocking cache?

(b) Why do many modern processors use non-blocking caches in preference to blocking caches? Why might some processors prefer to use blocking caches?

(c) You and your friend Jim Bob are assigned the task of evaluating a non-blocking cache design for a future processor. Your friend argues that the best approach is to use traditional metrics used to evaluate caches (e.g., miss ratios and average access times).

(i) What arguments might he be using to support his decision?

(ii) Do you agree? Why or why not?

3. Branch prediction

It is well known that 60% or more of all branches are repeatedly decided the same way, and therefore can be reliably predicted. A number of techniques exists for identifying those branches that are likely to be unpredictable.

(a) Suggest a way to exploit the distinction between predictable and unpredictable branches in a “conventional” superscalar processor (e.g., the MIPS R10000).

(b) Some microarchitectures, e.g., Multiscalar and trace caches, propose to increase instruction-level parallelism by grouping instructions into thread segments that contain multiple branch instructions. In such systems, should unpredictable branches appear in the middle of a thread segment or between thread segments? Explain.

4. Cache Coherence

In most shared-memory multiprocessors, cache misses can only be serviced by a designated *home* memory module or by the cache currently designated as the *owner* of the requested block. However, in some cases a shared copy of the block may be much closer (and/or in faster memory) than either the home or the owner. Since latency is critical, it would obviously be desirable to retrieve the block from whatever source can provide the block most quickly. Few coherence protocols do this, though for a large network of processors with increasing distance between them, the benefits clearly increase.

(a) Give at least two reasons why it may be difficult to extend a coherence protocol to provide the block from the optimal source.

(b) For a bus-based snooping protocol, explain how such a protocol extension might be supported.

(c) For a CC-NUMA protocol, can you suggest a protocol extension that would accomplish this aim? (Hint: suppose that the nodes are clustered.)

5. Prefetching

As instruction issue speed has diverged from memory latency, prefetching operands has become more and more critical to achieving high performance. Recently, numerous researchers have investigated the notion that the best way to fetch data in advance is to have a speculative processor "executing" the program quickly—but possibly incorrectly—racing ahead of the verification processor, which actually executes the entire program correctly.

(a) Compare this approach against more traditional methods for prefetching operands.

(b) Some critics of this approach worry that the speculative processor will never get far ahead of the verification processor, limiting the effectiveness of the prefetching. What methods can be used to help the speculative processor race ahead of the verification processor?

6. Interconnection trends in shared-memory multiprocessors

During the last decade, high-end shared-memory multiprocessors have moved from using relatively simple busses to more complex switched interconnects.

(a) What are the fundamental trade-offs between busses and switched interconnects? What impact does this choice have on the logical and physical design of a shared-memory multiprocessor system?

(b) As multiprocessors move on chip (i.e., CMPs), what kinds of on-chip interconnection networks do you expect to see (i.e., buses or switched interconnects)? What are the factors that favor one approach over another? What trends will drive this design trade-off?