FALL 2000
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination
Monday, September 18, 2000
3:00 – 7:00 PM
Room 1213 Engineering Hall

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. Decimal Carry Lookahead Addition

Binary Coded Decimal (BCD) represents a decimal digit with four bits using binary bit patterns 0000 (0) through 1001 (9). Assume that each digit of a decimal adder has inputs a<3:0>, b<3:0>, and carryin and outputs sum<3:0>, carry-generate g, and carry-propagate p. Also assume that all decimal numbers are non-negative.

(a) Specify g and p unambiguously (e.g., with C code or logic equations).

(b) Consider a block of three decimal digits. Let output G be asserted if the block generates a carry. Let output P be asserted if the block propagates a carry. Specify G and P.

(c) Specify how to create a decimal adder of nine digits using three of the blocks specified for part (b).

2. Branches

Branches are one of the biggest, if not the biggest, performance impediments in modern processors.

(a) How do branch instructions degrade the performance of a processor?

(b) Describe two *architectural* solutions to overcome the performance impediments of branch instructions. Discuss their pros and cons.

(c) Describe two *microarchitectural* solutions to overcome the performance impediments of branch instructions. Discuss their pros and cons.

3. Prefetching versus Non-Blocking Caches.

Some researchers have argued that non-blocking caches (a.k.a., lockup-free caches) are necessary to obtain high memory system bandwidth. Others have argued that hardware and/or software-directed prefetching support is necessary to achieve this goal.

(a) Explain how non-blocking caches can improve memory system bandwidth.

(b) Explain one software-directed prefetching mechanism and how it can improve memory system bandwidth.

(c) What demands do non-blocking caches and software-directed prefetching place on the processor, cache structure, system bus, and main memory to fully exploit their capabilities? Do these different schemes work together synergistically or do they conflict. Explain.

4. I/O Interface Implementation

Most modern computer systems have a system interconnect (e.g., the system bus) that connects the processor and caches to main memory. In addition, these systems have one or more I/O interconnects (e.g., an I/O bus), such as PCI, to connect to I/O devices. The operating system accesses these I/O devices using ordinary loads and stores.

(a) Describe one typical way that the I/O interconnect is physically connected to the rest of the computer system. Discuss issues that affect performance and correctness of operation, such as flow control and deadlock avoidance.

(b) Describe the way in which the hardware and software work to allow ordinary loads and stores to access I/O devices. Be sure to discuss addressing issues.

5. Synchronization

Locks are a common synchronization abstraction that provide mutual exclusion in shared-memory parallel programs. Locks can be implemented in a variety of different ways, including: (1) atomic memory primitives (e.g., test-and-set and fetch&add) (2) non-atomic memory primitives (e.g., load-linked/store-conditional), and (3) explicit hardware lock/unlock primitives (e.g., Cray Y-MP lock registers, Dash's lock/unlock operations on directory entries, and QOSB/QOLB)

(a) Discuss the tradeoffs of these different approaches. What limitations do they have? How complex are they to implement in hardware and in software? What is their impact on performance in both the contended and uncontended cases?

(b) If you were designing a processor specifically for a shared-memory parallel server, which mechanism(s) would you chose and why?

6. Computing with Molecules

As a possible successor to CMOS, chemists are proposing a technology to do computing with molecules. It appears, for example, that a molecular technology can implement the logical OR of two inputs. More work is needed to develop molecular technology to the point where is can support a *logic family* suitable for implementing programmable computers.

(a) *At All:* What primitives and properties does a molecular logic family need to make is possible to use it for implementing a programmable computer?

(b) *Better:* What additional properties might a molecular logic family possess to make it more likely that it could be used to implement computing products superior to those implemented with conventional technology?