FALL 2010
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination
Monday, September 20, 2010

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:

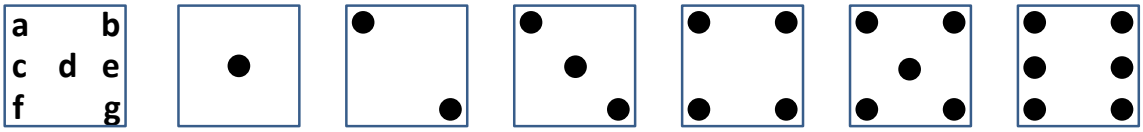
The Exam Committee tries to proofread the exam as carefully as possible. Nevertheless, the exam sometimes contains misprints and ambiguities. If you are convinced a problem has been stated incorrectly, mention this to the proctor. If necessary, the proctor can contact a representative of the area to 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. Logic design

An electronic game uses an array of seven LEDs to display the results of a roll of a dice. A decoder is to be designed to illuminate the proper dots for each of the legal six die values.

The roll of the dice is available as a 3-bit signal X2 X1 X0 for values 1 through 6. Assume 000 and 111 never occur.
Design a digital circuit that produces values of 1 (representing an illuminted dot) and 0 representing (an unilluminated dot) that will depict the a dice role based on the values. Your circuit must produce outputs a through g.

The assignment of the variables to location on the dice are shown below:



You can assume you have the basic logic gates (AND, OR, NOT, XOR, NAND etc). No more than 4-inputs to any such gate.

2. Instruction supply

An out-of-order processor is quite good at extracting and exploiting available parallelism in an instruction stream but to do so its instruction fetch mechanisms must fetch a good supply of instructions every cycle: if you want to sustain a processing rate of 4 instructions per cycle, the fetch mechanisms must fetch at least 4 instructions per cycle. There are two aspects of instruction fetching: i) knowing which instructions to fetch, and ii) fetching the instructions from where they are stored.

To simplify, the former requires a good branch predcitor and the latter requires adequate memory hierarchy structures. However, in most cases these two aspects are closely coupled and it is not easy or even possbible to separate all of the microarchitctural structures to independently deal with each aspect.

You are the chief architect of a family of microprocessors with a variety of different microarchitecures, ranging from a single-issue, in-order, pipelined microarchitecture to a 16-issue, out-of-order superscalar microarchitecture. They all execute the same RISC-like ISA, with fixed-length instructions.

Describe and explain your choice of the microarchitecture of the instruction supply mechanisms for the following three microarchitectures:

A) Simple: a single-issue, in-order, pipelined microarchitecture

B) Mid: a 4-issue, out-of-order, pipelined microarchitecture

C) Beast: a 16-issue, out-of-order, pipelined microarchitecture

3. Predication and Speculation

Overcoming the performance impediments of branch instructions, a.k.a the branch problem, is one of the first issues that a designer of a high-performance processor has to address. Over the years two main approaches to dealing with the branch problem have been proposed: predicated execution and speculative execution (using branch prediction). When these approaches were being considered for adoption, there was significant debate about the relative pros and cons of each approach.

A) Briefly describe predicated execution and discuss its pros and cons.

B) Briefly describe speculative branch execution and discuss its pros and cons.

C) Does it make sense to use speculative branch execution for an implementation of an architecture that uses an ISA that supports predicated execution? Argue why or why not (pick one and present your arguments).

4. From Big Parallel Machines to CMPs

Computer design has a long history of innovations first being made in high-end machines and eventually finding their way into mainstream processor microarchitectures. For example, pipelining was used in IBM mainframes in the early 1960s and eventually found its way into microprocessorsin the mid 1980s. Other examples include caches, branch predictors, reorder buffers, and the like.

Large-scale parallel processors have been around for a while. Examples in your reading list include: the Cray T3E, the SGI Origin and the Sun Wildfire. Through other reading, you may be familiar with others.

With CMPs becoming ubiquituous and with a continuing increase in the number of processing cores on a CMP, one might expect techniques developed for large-scale parallel processors to start being used in the CMP context.

A) Which synchronization features from large-scale machines might you expect to see being used in CMPs? Discuss.

B) Which communication features from large-scale machines might you expect to see being used in CMPs? Discuss.

5. Cache Coherence, Power and Programmability

Some architects have argued that as the number of cores per die continues to increase and power dissipation becomes increasingly critical, that future multicore processors will tend to eschew hardware cache coherence as being too power inefficient. Others have argued that the key problem is making future multicores more programmable and that eliminating cache coherence would be a giant step backwards.

A) Explain the major sources of power inefficiency that are introduced by a snooping cache coherent system.

B) Explain whether (and why) a directory cache coherent system tends to be more or less power efficient than a snooping system.

C) Discuss the programmability implications of eliminating hardware cache coherence.

6. The End of Moore's Law

Moore's Law shows signs of finally coming to an end. The amount of charge stored in a standard memory device is projected to become so small that they cannot reliably store a value for a reasonable amount of time. Supposedly identical logic devices will perform so differently that they may get different answers (or at least take very different amounts of time to get the same answer). For example, an AND gate may not behave like an AND gate all the time or the delay of two identically designed AND gates could be different. And the "utilization law" restricts the number of devices that may be simultaneously active and still meet a chip's power and thermal limits.

i) Discuss techniques to build reliable processors and caches from unreliable logic components.
ii) Discuss how increasing failure rates affect the utility of the techniques you presented in part (i), for building reliable systems from unreliable components.
iii) Discuss how the power limitations implied by the utilization wall affect the utility of the techniques you presented in part (i), for building reliable systems from unreliable components.