FALL 1996
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH. D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination

Monday, September 16, 1996
3:00 – 7:00 PM
115 Psychology

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 nontrivial.

1. Branch Prediction

In static branch prediction, the compiler uses heuristics to guess whether or not a branch is likely to be taken.

(a) Give an example of one such compiler heuristic.

(b) What are the possible ways that the compiler can convey its predictions to the hardware?

In dynamic branch prediction, the hardware attempts to use the history of past branches to predict whether branches will be taken on future executions.

(c) Give an example of one such dynamic branch prediction technique.

(d) What are the pros and cons of dynamic branch prediction schemes versus static branch prediction?

2. Memory System Design

Hit ratio is a classic measure of cache memory performance, and this simple measure has been used to evaluate a wide variety of design parameters of a cache. Hit ratio, however, does not accurately predict performance in all cases. A more robust measure of performance is *effective access time*, which is the mean time from when the address is available until the data is available.

(a) Give two examples of how a change that results in improvement in the hit ratio might actually degrade performance.

(b) Explain why effective access time is a better measure of program performance than hit ratio.

(c) Even effective access time is not a reliable measure of program performance for modern, high-performance processors. Give an example of a feature of a recent architecture that reduces the reliability of effective access time as a performance measure.

3. RAIDs

Consider the first block of data on five disks in a RAID Level 5. Rather than the usual 4KB block, our example will use a 4-bit block. Each `ai` and `bi` stands for a `0` or a `1`.

Disk	Data in 1st block
info1	1 0 1 0
info2	a3 a2 a1 a0
info3	1 1 0 0
info4	b3 b2 b1 b0
check	1 1 1 0

(a) Assume that a program wants to write the block ‘‘`1 0 0 1`’’ onto disk info3. What must it do to complete the write? What are the new values in the check disk? State any additional assumptions you need to make.

(b) A RAID Level 5 does error correction with parity, while a memory system can only detect errors with parity. What assumptions does a RAID use to make correction possible? Are these assumptions reasonable?

(c) Assume that each disk in the above RAID has a mean time to failure of $MTTF_{disk}$ and a mean time to repair of $MTTR_{disk}$. Let the RAID’s mean time to failure be $MTTF_{raid}$. Suppose a new hot insertion technique cuts the mean time to repair of a disk by a factor of 3. Give the intuition for how this improvement will affect the RAID’s mean time to failure. Write an expression of the RAID’s new mean time to failure. State any additional assumptions you need to make.

4. Vectors vs. Superscalar

Traditional vector supercomputers like the Cray-1 are designed specifically to provide high performance on regular computations (e.g., dense linear algebra calculations). In the mid-1980's, IBM's John Cocke argued that superscalar RISC machines could be a more effective architecture for scientific computing because these new machines would perform just as well for regular computations and significantly better for irregular problems.

(a) Make the argument that superscalar RISCs can execute scientific codes nearly as well or better than vector machines.

A common counter-argument is that while superscalars may have the same or better peak instruction execution rates as traditional vector machines, they do not have sufficient memory bandwidth to sustain the computations. Nor are superscalar's cache memories appropriate for large scientific codes.

(b) Explain why traditional vector supercomputers generally provide vector registers rather than caches. What are the pros and cons of the two approaches?

(c) Superscalar advocates have argued that compilers can manage cache behavior, so that they perform much like a vector register. Discuss how a compiler might improve cache performance. Are there any hardware extensions that might allow the compiler to do a better job, or to do the same job more easily?

5. Implementing Registers

Microprocessors that exploit instruction-level parallelism must deal with read-after-write (RAW), write-after-read (WAR), and write-after-write (WAW) hazards in both integer and floating-point registers. Some microprocessors mitigate the effect of hazards by *not* requiring an instruction to get a register value from a specific storage location. Instead, multiple uses of the same architectural register can use different implementation locations. Discuss two implementations of this approach from real machines. Illustrate how they can mitigate the negative effects of hazards.

6. Shared-Memory Multiprocessing

Shared-memory multiprocessor implementations can be classified several ways. One taxonomy divides them as follows:

UMA:	no cache coherence and uniform memory access (that is, accesses to any memory from any processor incur the same average latency),
NUMA:	no cache coherence and non-uniform memory access,
CC-UMA:	cache coherence and uniform memory access, and
CC-NUMA:	cache coherence and non-uniform memory access.

Discuss how programmers (or compiler-writers) might change programs to make them perform better for these four different implementation classes. Concentrate on ordinary data references (not synchronization accesses). When appropriate, give examples of real machines that fall into these categories.