SPRING 1993
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Depth Examination

Monday, September 20, 1993
3:00 – 7:00 PM
2365 Computer Sciences

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 tires 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. Implementing condition codes

Consider an ALU and condition code logic for a hypothetical 16-bit computer implementation.

> **ALU.**
> Inputs: `A(15:0)`, `B(15:0)`, `ADD`, and `2SC`, where bit 15 is most significant;
> Outputs: `C(16:0)`, a 17-bit result;
> Function: Produces `C(16:0)` using addition (`ADD`==1) or subtraction (`ADD`==0) of two's complement (`2SC`==1) or unsigned numbers (`2SC`==0).
>
> **CC-logic.**
> Inputs: *to be determined by you*;
> Outputs: `ZERO`, `NEGATIVE`, and `OVERFLOW`;
> Function: `ZERO` is 1 if the result is 0, `NEGATIVE` is 1 if the result is less than zero; `OVERFLOW` is 1 if truncating the result back to 16 bits produces a number that is not the correct sum or difference.

Your task is to design the **CC-logic**.

(1) What are the minimum number of inputs that can you use? Do not use any logic outside of the ALU and CC-logic blocks.

(2) Unambiguously specify the logic functions for `ZERO`, `NEGATIVE`, and `OVERFLOW` using equations or gates so that an implementor need not know how two's complement and unsigned numbers are represented.

2. Parallel Computing

Consider the execution of the following Fortran code fragment on the ILLIAC-IV (with 64 PEs).

```
      DO 10 I = 2, 64
10    A(I) = A(I-1) + B(I)
```

(1) Explain how the above code fragment would be executed (in parallel, of course). List the number of steps, and the actions of the PEs at each step. You need only give the pseudo-code, illustrating the main actions at each step; ILLIAC-specific details are not important.

(2) How many words pass through the network during the algorithm? State any assumptions that you make.

3. Precise Exceptions

(1) Define the conditions necessary for an exception to be considered precise.

(2) Explain which types of exceptions are straightforward to make precise and which types can be more difficult. Which implementation techniques make maintaining precise exceptions more difficult? Explain why.

(3) What are the general philosophical approaches to making exceptions appear precise?

(4) Smith and Pleskun describe several techniques for implementing precise interrupts in pipelined processors. How does a history buffer differ from a future file? Which of the general approaches does each employ?

4. Memory Order

Consider a multiprocessor organized in the style of the IBM RP3: a "dance hall" architecture with no data caches. Memory accesses are achieved through a network. Both the network and the memory system exhibit long delay relative to a processor's instruction execution time. For purposes of this problem, assume that instructions are never modified.

The network delivers memory requests in order between a given processor and a given memory module. To assure Sequential Consistency but minimize delay, it is planned to use a write buffer. After a store instruction is issued, the address and data are *always* stored in the write buffer (assume there is always sufficient space), and eventually forwarded through the network to memory. Later an acknowledge signal is received from the memory system, indicating that the write request and data have been received by the memory system.

The write buffer logic controls when both loads and stores are *injected*— *i.e.*, a request is permitted to enter the network — and must delay injections as necessary to guarantee sequential consistency. The following questions pertain to this write buffer control logic.

(1) When a store instruction has been injected, but not yet acknowledged, can a succeeding store instruction to the *same address* be injected without delay?

(2) When a store instruction has been injected, but not yet acknowledged, can a succeeding store instruction to a *different address* be injected without delay?

(3) When a store instruction has been injected, but not yet acknowledged, can a *load* instruction to the *same address* be satisfied from the data in the write buffer?

(4) When a store instruction has been injected, but not yet acknowledged, can a *load* instruction to a *different address* be issued and injected without delay?

5. Processor/Memory Tradeoffs

You are the chief architect for a company whose main business is the manufacture of DRAMs. Your marketing department is looking for ways to capture new markets by incorporating additional logic onto a DRAM chip, giving it increased capabilities.

One market they are considering is text retrieval, where the application is to search unordered text for a particular string, possibly including complex wild-card expressions. For example, text retrieval technology is frequently used by Wall Street firms to search incoming news reports for mention of company names, etc. Currently, most text retrieval systems are software systems that are very slow, requiring expensive hardware systems to perform the search in real-time. Your marketing department has decided that this market is sufficiently large to justify a new product development, and has asked you to come up with a design.

The marketing department has proposed a range of design options, each of which is a different way to implement the desired functionality. All three options are single, standalone chip designs that require little or no glue logic.

Option 1 is to add a "full RISC processor" to the DRAM chip.

Option 2 is to add a "very small processor" to each bit of storage in the DRAM array(s). These processors would, by necessity, be SIMD processors controlled by some on-chip control unit.

Option 3 is to add a "small processor" for each column, or group of columns, in the storage array(s). These processors could be somewhat bigger than in option 2, but must presumably still be SIMD processors.

(1) Evaluate each of these options, taking into account cost, cost/performance, peak and sustained performance, ease of programmability, and any other factors that you think are relevant.

(2) Which of these options do you think has the most promise? Why?

(3) Would you recommend that your company build one of these chips? Why or why not?

6. Interconnection Networks

You are responsible for designing the network for a large-scale parallel machine, with 1024 processing elements. It has been decided that the processors of the machine will be multi-threaded, and will use multiple contexts to tolerate latencies. You have narrowed down the choice to: (i) a 1024×1024 Omega network or (ii) a 32×32 mesh network.

(1) What are the issues that influence your choice of the network, and why?

(2) Use these issues to argue in favor of using the mesh network.

(3) Use these issues to argue in favor of using the Omega network.

(4) Which network would you recommend?