THE UNIVERSITY OF AUCKLAND

FIRST SEMESTER, 2010
Campus: City

COMPUTER SCIENCE
Advanced Computer Architecture
(Time Allowed: FIFTY minutes)

FAMILY NAME: Model Answers
PERSONAL NAMES:
STUDENT IDENTIFICATION NUMBER:
LOGIN NAME:
SIGNATURE:

For this test you are allowed to use printed material: books, printouts, notes, etc. You may not use any electronic devices.

This test has 6 questions, all of equal value. ***Answer any four***. If you answer more than four questions, the first four not crossed out will be graded.

Some questions are harder than others—they will be graded more generously.

Be concise. Long answers are not generally better answers.

QUESTION	MARKS	SCORE
	10	
	10	
	10	
	10	
TOTAL	40	

Question 1

In a recent New Zealand Herald feature about Bloom Energy, a company that builds fuel cells, the CEO K.R. Sridar was asked the question, "How can you ever be more efficient than a big power plant?" to which he answered, "The question to ask is, in a traditional power plant, is there a Moore's Law kind of learning that can happen? The answer is no. There are 100 years of history associated with that [technology]. Whereas we have shown in the past five years that every year we are able to improve upon the physics and chemistry to get more value out of the same material that we put in."

Is this a reference to the same Moore's Law discussed in CS703? Explain.

Question 2

Explain the difference between dynamic scheduling and static scheduling. Is dynamic scheduling always better? What are the arguments for static scheduling? What are the arguments for dynamic scheduling.

Question 3

What is the purpose of a reorder buffer (ROB)?

Question 4

A high-performance processor may prefetch operands from memory before previous instructions have completed. Consider the following code:

```
REP  mv    R1, R2        # Copy R2 to R1
     swap  R1, Lock      # Atomically swap location Lock with R1
     bne   R1, R2, REP   # Branch to REP if R1 != R2
     lw    R3, TailPtr   # Copy contents of memory location TailPTr into R3
     lw    R4, (R3)      # Copy contents of memory location with address in R3 into R4
     add   R3, R3, 1     # Increment R3
     sw    R3, (R4)      # Write contents of R3 into memory location with address in R4
```

What can happen if and only if the memory model allows load and store instructions to complete out of order?

Question 5

What problem does Test&Test&Set solve that Test&Set does not solve?

Question 6

Explain the difference between an atomic transaction as described by Herlihy and the classical critical section.

Continued...

Question #1

The speaker Sridar is clearly referring to the phenomenon of "the popular Moore's Law, namely, exponential increases in capability and decreases in cost over a relatively extended period of time. The notion of "Moore's Law" has become ever more widely recognized as we have seen increasing examples of technology growth at exponential rates. There is even an analogy with "the big power plant," namely, that mainframe "big-iron" computers experienced steady but limited improvements over decades, and was made obsolete by fast-moving technology tracking Moore's curve.

Note in both cases, the "law" is largely self-fulfilling, with businesses adjusting their investment to respond to, and anticipate, projections of change in the marketplace.

Question # 2

Straight from H&P section 2.4 (assigned reading):

"A simple statically scheduled pipeline fetches an instruction and issues it, unless there was a data dependence between an instruction already in the pipeline and the fetched instruction that cannot be hidden with bypassing or forwarding. (Forwarding logic reduces the effective pipeline latency so that the certain dependences do not result in hazards.) If there is a data dependence that cannot be hidden, then the hazard detection hardware stalls the pipeline starting with the instruction that uses the result. No new instructions are fetched or issued until the dependence is cleared.

"In [sec 2.4] we explore dynamic scheduling, in which the hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behaviour. Dynamic scheduling offers several advantages: It enables handling some cases when dependences are unknown at compile time (for example, because they may involve a memory reference), and it simplifies the compiler. Perhaps most importantly, it allows the processor to tolerate unpredictable delays such as cache misses, by executing other code while waiting for the miss to revolve. Almost as importantly, dynamic scheduling allows code that was compiled with one pipeline in mind to run efficiently on a different pipeline."

Static scheduling is much simpler to implement and a good compiler can achieve equal performance if instruction times are all predictable.

Question # 3

To assure the clean handling of exceptions and interrupts, instructions must commit in-order even if they are allowed to execute out of order. When instructions finish their execution, they are "retired" with each result being place in a buffer--the reorder buffer--and their results are available for future instructions, though not yet committed (i.e., made visible to the system). When all previous instructions have committed, the instruction may be committed, assigning the value to a register or memory location, thus making the operation visible.

The ROB further serves as an extension to the Tomasulo algorithm, allowing instructions to be executed speculatively.

Question # 4

This question had a mistake in the code, making it unclear exactly what was expected. Any answer was accepted if it identified a different result if two of the load or store instructions were executed out-of-order.

Question #5

Test&Set continuously creates serious memory contention when more than one processor is spin-waiting to acquire a lock held by another node. Test&Test&Set creates serious memory contention only after a lock is modified (a release or an acquire, whether successful or not). Because T&T&S does not attempt to write while spin-waiting, the contention ceases as soon as all processors can observe that the lock is held.

Continued...

Question # 6

The critical section is guaranteed exclusive access to a set of resources while holding a lock, obeying a protocol to provide mutual exclusion with regard to a set of variables guarded by the lock. The thread must acquire the lock using a mechanism that guarantees it will be the only holder of the lock, and release it when finished. The critical section exhibits the properties of atomicity, isolation, and consistency by mutual agreement defined by the protocol.

The atomic section is a language construct that informs the system that the block of code is to have the appearance of instantaneous execution, i.e., atomicity, isolation, and consistency, but the support for this atomic action is provided by the system and does not depend on the programmer to obey a protocol.

Continued...