Final Exam

CSC 252

9 May 2009

Directions; PLEASE READ

This exam has 20 questions, some of which have subparts. Each question indicates its point value. The total is 88 points. Questions 18 and 20 are for extra credit only, and not included in the 88; they won't factor into your exam score, but may help to raise your end-of-semester letter grade.

This is a *closed-book* exam. You must put away all books and notes (except for a dictionary, if you want one). Please confine your answers to the space provided. For multiple choice questions, darken the circle next to the best answer. Be sure to read all candidate answers before choosing.

In the interest of fairness, the proctor will decline to answer questions during the exam. If you are unsure what a question is asking, make a reasonable assumption and state it as part of your answer.

Remaining exams will be collected promptly at 11:30.

Freebie

1. (3 points) Put your name on every page (so if I lose a staple I won't lose your answers).

Multiple choice (3 points each). Choose the most reasonable answer in each case.

2. The purpose of *bias* in IEEE floating-point numbers is to

- ⊗ **a.** ensure that nonnegative numbers are ordered the same as in integer arithmetic
- ◯ **b.** avoid the floating-point "gap" near zero
- ◯ **c.** distinguish normal and "not a number" (NaN) values
- ◯ **d.** increase numeric stability in the face of rounding errors

3. Which of the following can be represented precisely in single-precision IEEE floating point?

- ◯ **a.** 1.25×2^{-150}
- ◯ **b.** 6.3
- ⊗ **c.** 9.375
- ◯ **d.** all of the above

4. Many operations that can be performed in a single instruction on the x86 require more than one instruction on most RISC machines. Which of the following usually *does* have a single-instruction equivalent?

⊗ **a.** `ja foo`, where `foo` is a label in the current subroutine

◯ **b.** `incl foo`, where `foo` is a global variable

◯ **c.** `leal -12(%ebp,%esi,4)`

◯ **d.** `movl $0x1a2b3c4d, %eax`

5. Why didn't RISC machines appear before the early 1980s?

◯ **a.** Nobody thought of them until then.

⊗ **b.** Until then you couldn't fit enough transistors on a chip to build a pipelined machine.

◯ **c.** That's when IBM's patent on pipelining expired.

◯ **d.** That's when the gap between processor and memory speeds reached the point at which pipelining became profitable.

6. What is the cycle time of a 2.5GHz processor?

◯ **a.** 4 ns

◯ **b.** 2.5 ns

⊗ **c.** 0.4 ns

◯ **d.** 0.25 ns

7. Why are the keyboard and main memory usually connected to different buses?

◯ **a.** To allow the keyboard to communicate with the processor at the same time that the disk is communicating with memory.

⊗ **b.** To reduce the cost of the keyboard interface.

◯ **c.** To minimize the cost of handling keyboard interrupts.

◯ **d.** all of the above

8. A memory architect might increase the associativity of a cache in order to

◯ **a.** decrease hit time

◯ **b.** decrease miss penalty

◯ **c.** exploit spatial locality

⊗ **d.** reduce the likelihood of thrashing

Short answer (5 points each). Each answer in this section should be a *single sentence*.

9. Compared to processors of the 1970s, why is branch prediction so much more important today?

   Answer: Because modern processors are deeply pipelined, and we need to predict through branches in order to avoid bubbles.

10. Compared to processors of the 1970s, why are caches so much more important today?

    Answer: Because processor speed has improved much more than memory speed, so the distance to memory in cycles has grown dramatically.

11. What is the principal difference in Linux between a *process* (created with `fork`) and a *thread* (created with `pthread_create`)?

    Answer: Every process has its own address space. Threads of the same program share an address space.

12. Why can't we use a lock to protect a data structure shared between the main program and a signal handler?

    Answer: Because if the lock is held by the main program when the signal occurs, deadlock can result.

13. (4 points) Consider the improvements made over the last 20 years in

3	processor speed	1	memory (DRAM) speed
2	memory capacity	4	disk capacity

    Number these from smallest improvement (1) to largest improvement (4).

14. (4 points) Consider the following divisions of disk storage space:

3	cylinder	4	platter
1	sector	2	track

    Number these from smallest size (1) to largest size (4).

Essay/problem solving Points as marked.

15. (9 points) As we discussed in class, the time to run a program is the product of (a) the number of instructions executed, (b) the average number of cycles per instruction, and (c) the cycle time. For each of these factors, give two examples of techniques that could be used (by the programmer, compiler, or architect) to improve system performance. (Be specific. For example, don't say "reduce the cycle time"; tell me what an architect could do that would allow the cycle time to be reduced.)

Answer: There are *many* possible answers to this question. Here are just a few.

We can reduce the number of instructions executed by picking a better algorithm, applying compiler techniques that eliminate redundant computation, or designing an ISA that averages more work per instruction.

We can reduce the average number of cycles per instruction through deeper or wider pipelining, better branch prediction, register renaming, out-of-order execution, etc. In the compiler, we can choose instructions that execute more quickly (e.g., shift instead of multiply), schedule instructions to minimize pipeline stalls, or rename registers in software to eliminate W–W and R–W dependences.

We can reduce the cycle time if we use smaller transistors, higher voltage, or shallower circuits (fewer logic levels per cycle).

16. (9 points) Consider a 2 MB, 4-way associative, unified level-2 cache with a 32-byte block size and 32-bit physical addresses.

    (a) How many sets does the cache have?

    Answer: Each set contains four 32-byte blocks, for a total of $128 = 2^7$ bytes per set. Since the whole cache is $2\text{MB} = 2^{21}$ bytes, we have $2^{21-7} = 16\text{K}$ sets.

    (b) How many bits are required to hold the tag of a line?

    Answer: We need 5 bits ($\log_2 32$) to specify a byte index within a block and 14 to specify the set, leaving $32 - 5 - 14 = 13$ bits for the tag.

    (c) How many different memory blocks may map to the same set in the cache?

    Answer: $2^{13} = 8\text{K}$ blocks.

17. (9 points) Consider a machine with 64-bit addresses and an 8KB page size.

    (a) How many bits are required to represent a page number?

    Answer: 8KB is 2^{13}, so we need $64 - 13 = 51$ bits for a page number.

    (b) Suppose our TLB has 512 entries, and our machine has 4GB of RAM. What fraction of physical memory can be accessed without suffering a TLB miss?

    Answer: 512 TLB entries can cover 4MB of memory, which is 1/1024-th of the physical memory of the machine. (Obviously we need a lot of temporal locality to use the TLB well.)

    (c) If each page table entry requires 8 bytes, how much space will be required for an inverted (hash-table style) page table?

    Answer: The inverted page table has one entry for each physical frame. With 4GB of RAM, we have $2^{32-13} = 2^{19}$ entries. At 8 bytes each, that's $2^{22} = 4\text{MB}$ of space.

18. (Extra Credit)

    (a) (6 EC points) Recall that an inverted page table is shared by all processes running on the machine. Could the page table of part (c) in the previous question reasonably be implemented in SRAM instead of DRAM? Could it be put on chip? Explain.

Answer: 4MB is not an unreasonable amount of SRAM, in terms of dollar cost. It's comparable to the space devoted to on-chip cache, however, so finding that much extra space on-chip would probably be asking too much. Moreover we just happen to have 4GB of RAM. We have to design an inverted page table to accommodate the maximum amount of RAM someone might cram into the machine. Even if the architects limit that to, say, one TB, we're talking 1GB of inverted page table. That's still the same small fraction of main memory, but far too much to put on-chip—and the situation would get worse with larger physical memory limits. The next release of Mac OS X will support up to 64TB of RAM. That would imply a 64GB page table.

(b) (6 EC points) Older SPARC processors used a three-level tree-structured page table, with hardware TLB reload. More recent models use software TLB reload instead. Why do you suppose the designers might have made this change?

Answer: Recall the basic tradeoff between hardware and software TLB reload: hardware is a little faster, but ties the OS to a particular choice of page table structure. With the move to 64-bit addresses on the SPARC, even tree-structured tables can grow unacceptably large: we simply can't afford to devote several bytes to every page of virtual memory in every address space. Denser segment-by-segment structures are essential, but these are too complex to manage in hardware. We can choose to put some subset of the address space in a hardware table—this is what x86-64 does—and basically treat the hardware page table as a second-level TLB. The SPARC designers presumably did some performance studies and concluded that the benefit of that second-level structure (with its fairly low miss penalty) wasn't worth the hardware design complexity.

19. (9 points) Consider the following two-thread program fragment, executing on a two-processor machine connected by a bus with MESI (snooping) cache coherence.

Thread 1	Thread 2
`...`	`while (!f) /* spin */ ;`
`f = true;`	`...`

Suppose Thread 1 and Thread 2 are running on different processors; that `f` is initially false; that neither processor has `f` initially cached; and that Thread 2's first read of `f` occurs before Thread 1's write. For simplicity, assume that there is only one level of cache.

Trace the sequence of cache operations (loads, stores, hits, misses, bus messages, invalidations) that will occur in the process of executing the program fragment.

Answer: Thread 2's initial load of `f` causes a cold miss. It sends a message out on the bus and memory provides the data, which Thread 2's cache keeps in exclusive state. Subsequent loads of `f` hit in Thread 2's cache.

Thread 1's store to `f` causes another cold miss. It requests the data in exclusive mode and either memory or Thread 2's cache responds. Either way, Thread 2's copy is invalidated. Thread 2's next load of `f` suffers a coherence miss, because of the invalidation. It requests the data. Thread 1's cache responds, dropping its copy from

modified to shared, and Thread 2's cache stores it as shared as well. Because the value has changed, Thread 2's loop terminates.

20. (Extra Credit)

    (a) (4 EC points) In 2008, the stock market fell 4% in the 1st quarter, 10% in the 2nd quarter, 6% in the 3rd quarter, and 17% in the 4th quarter. Give an equation for the average quarterly loss. (You don't have to actually do the calculation—just show how.)

    Answer: We're looking for the rate that, if held steady over four quarters, would have led to the same overall decline. This calls for the geometric mean:

    $$\text{ave} = (0.04 \times 0.10 \times 0.06 \times 0.17)^{1/4}$$

    This turns out, by the way, to be almost exactly 8%. Note that the arithmetic mean would have yielded 9.25%.

    (b) (4 EC points) In a 4-leg relay race, runner A averages 9.7 m/s, runner B averages 9.4 m/s, runner C averages 9.2 m/s, and runner D averages 9.8 m/s. Give an equation for the average speed of the team as a whole. (Again, the formula alone will suffice.)

    Answer: We're looking for the speed that, if achieved by all four runners, would have led to the same finish time. This calls for the harmonic mean:

    $$\text{ave} = \left[\left(\tfrac{1}{9.7} + \tfrac{1}{9.4} + \tfrac{1}{9.2} + \tfrac{1}{9.8}\right)/4\right]^{-1}$$

    This turns out to be about 9.519 m/s. The arithmetic mean would have yielded 9.525 m/s—very close in this case, but only because the runners were fairly evenly matched.