

# Final Exam

CSC 252

12 May 2007

## Directions; PLEASE READ

This exam has 20 questions, many of which have subparts. Each question indicates its point value. The total is 85 points. Questions 17(b), 18(b) and 20 are for extra credit only; they are not part of the 85 points. They will not factor into your exam score, but may help to raise your letter grade for the semester.

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.

Please put your name on every page. That way if I lose a staple I won't lose your answers. (Please do this **NOW** so you don't forget.) Scratch paper is available if you need it, but the proctor will collect **only the exams**.

In the interest of fairness, the proctor has been instructed not 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.

You will have a maximum of 3 hours to complete the exam. The proctor will collect any remaining exams promptly at 11:30 am. Good luck!

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

1. Dynamic linking allows us to

- a. economize on disk space requirements
- b. economize on physical memory requirements
- c. adapt old programs to new OS releases automatically
- d. all of the above

2. Address translation hardware allows us to

- a. avoid the need for load-time relocation
- b. implement virtual memory
- c. implement copy-on-write
- d. all of the above

3. To the nearest power of ten, how many nanoseconds does it take to access main memory on a modern PC if we miss in all levels of the cache?

- a. 10
- b. 100
- c. 1,000
- d. 10,000

4. Consider the following two *unsigned* integer addition operations. Which will overflow on a 32-bit machine?

$$\begin{array}{r} (1) \text{ 0x6000 0000} \\ + \text{0xa000 0000} \\ \hline \end{array} \qquad \begin{array}{r} (2) \text{ 0x6000 0000} \\ + \text{0x7000 0000} \\ \hline \end{array}$$

- a. (1) only
- b. (2) only
- c. both (1) and (2)
- d. neither (1) nor (2)

5. Consider the same two addition operations as in the previous question, but now interpret the hexadecimal bit patterns as *two's complement signed integers*. Which will overflow on a 32-bit machine?

- a. (1) only
- b. (2) only
- c. both (1) and (2)
- d. neither (1) nor (2)

6. Consider a simple 5-stage pipelined processor like the PIPE implementation of the Y86. Suppose the pipeline stages require 250, 300, 350, 325, and 275ps (picoseconds) each to execute, and that it takes 50ps to propagate a value through a pipeline register. At what clock rate can we reasonably expect to run the machine?

- a. 2.5GHz
- b. 300MHz
- c. 667MHz
- d. none of the above

7. The bit pattern 0x4040 0000, interpreted as an IEEE single-precision floating point number, represents what decimal quantity?
- a.  $1 \times 10^{128}$
  - b. 3
  - c. 10
  - d. not-a-number
8. Which of the following is likely to provide the *highest bandwidth* communication between Rochester and Buffalo?
- a. a leased phone line
  - b. the Internet
  - c. satellite transmission
  - d. a truck full of magnetic tape
9. Which is likely to provide the *lowest latency* communication between Rochester and Buffalo?
- a. a leased phone line
  - b. the Internet
  - c. satellite transmission
  - d. a truck full of magnetic tape
10. Which of the following page table organizations is *not* a reasonable option for modern processors?
- a. characteristic array
  - b. search tree
  - c. hash table
  - d. TLB only—trap to software on a TLB miss
11. The harmonic mean ( $\text{ave} = n/(\sum_i(1/r_i))$ ) would be most appropriate for which of the following?
- a. the average dollar cost of three computers
  - b. their average power consumption in W
  - c. their average clock rate in MHz
  - d. their average up-time in days

**Short answer.** Point values as marked.

12. (6 points.) What is microprogramming? Why was its invention considered such a breakthrough?

**Answer:** Microprogramming is an implementation technique in which we build a simpler processor than the programmer expects, and let it *interpret* certain ISA instructions. Microprogramming made it feasible to provide interoperability across an entire family of processors, from slow-but-cheap to fast-and-pricey. Later, it made it possible to fit the first microprocessors on a single chip.

13. (6 points.) State two significant limitations of the `gprof` profiling tool.

**Answer:** Its data is only a statistical approximation. It doesn't work on parallel programs. It isn't accurate for short program runs. It requires special compilation arguments. It naively attributes a subroutine's run time to its callers in proportion to the number of calls. It lumps together the statistics of mutually recursive subroutines. It can't see inside system calls or library routines that were not specially compiled.

14. (8 points.) Compare hardware exceptions and software signals. In what ways are they alike? In what ways are they different?

**Answer:** There are many possible answers. Here are a few.

Alike: Both turn an asynchronous event into a spontaneous subroutine call. Both mask new events until done handling the current one. Neither can safely acquire a lock that might be held by the main program.

Different: Obviously, one happens in kernel space, the other in user space. Exception handlers have a well-defined maximum processing time to ensure that interrupts are never lost. Signal handlers can't make similar guarantees, so they have to be written to check for lost signals. Signals are *virtualized*—delivered to the “right” process; interrupts are delivered to the machine.

15. (One point each.) For each of the following, indicate whether it is a feature of the processor's instruction set architecture (A) or its implementation (I):

A The number of register names

I The number of physical registers

I The number of pipeline stages

A The maximum displacement of a PC-relative branch

I The number of cycles that must elapse after a floating-point add before its result is available for use

16. (6 points.) Consider an embedded processor with 16-bit addresses and a 4KB on-chip cache. Fill in the following table describing various ways in which the cache might be organized.

block size	ways	sets	# of address bits used to specify		
			tag	set index	block offset
4	1	<b>1K</b>	<b>4</b>	<b>10</b>	<b>2</b>
8	2	<b>256</b>	<b>5</b>	<b>8</b>	<b>3</b>
32	4	<b>32</b>	<b>4</b>	<b>5</b>	<b>5</b>

17. Consider the following function in C:

```
void add_to_A(int A[], int size, int *p) {
    int i;
    for (i = 0; i < size; i++) {
        A[i] += *p;
    }
}
```

- (a) (6 points.) What is the most likely cause of poor performance in this code? How might you rewrite it to make it better?

**Answer:** Parameter `p` will be loaded from memory in every iteration of the loop, because the compiler doesn't know whether `p` points to an element of `A`. The programmer probably expects that this will never happen, in which case the following would allow the compiler to cache `*p` in a register:

```
void add_to_A(int A[], int size, int *p) {
    int i;
    int t = *p;
    for (i = 0; i < size; i++) {
        A[i] += t;
    }
}
```

- (b) (Extra Credit; 5 points max.) If the compiler were particularly smart (smarter than most), how might it generate code that is both fast *and* safe for this function?

**Answer:** It could generate code that checks for aliasing *at run time*. Though it doesn't really re-write source code, the assembler it produces would be the equivalent of the following:

```
void add_to_A(int A[], int size, int *p) {
    int i;
    if (p <= &A[-1] || p >= &A[size]) {
        int t = *p;
        for (i = 0; i < size; i++) {
            A[i] += t;
        }
    }
}
```

```

        } else
            for (i = 0; i < size; i++) {
                A[i] += *p;
            }
        }
    }
}

```

18. (a) (5 points.) Briefly explain how the Pentium III processor described in the book and in lecture is able to perform address translation and L1 lookup concurrently.

**Answer:** The designers chose cache parameters such that the set index and block offset bits lie entirely within the page offset portion of the virtual address, which is always the same as in the physical address. The frame number is used for the tag, which is needed for comparison only after the appropriate set has been retrieved from the cache.

- (b) (Extra Credit; 5 points max.) Discuss the limitations of the mechanism described in your answer to part (a). How might you escape these limitations?

**Answer:** To keep the set index bits within the page offset, the Pentium III used relatively high associativity (4) and relatively small L1 size (16KB). Higher associativity probably isn't an option—too slow. To increase the cache size while still permitting concurrency, we could (1) increase the page size, which moves the low bit of the page/frame number left, or (2) use virtual addresses to index into the L1 cache. Solution (2) introduces the problem of *virtual address aliasing*: if a process has two virtual addresses that map to the same physical address (this is possible for several reasons), the kernel has to make sure that if any bits of page number that contribute to the cache index are different in the two addresses, then no more than one of the addresses is marked valid in the page table at any given time (this can lead to thrashing). Alternatively, we can give up on concurrency and do address translation before cache lookup.

19. Cache coherence.

- (a) (5 points.) What is the *cache coherence problem*?

**Answer:** If data can be cached in more than one location, we have to prevent processors from reading stale copies of anything that has been written elsewhere.

- (b) (5 points.) In class we saw a three-state (invalid, shared, modified) snooping cache coherence protocol. We also saw a four-state protocol, which adds an *exclusive* state. Exclusive is like modified, except that the line is clean. What benefit does this new state provide?

**Answer:** If a processor reads and then writes a non-shared variable, the exclusive state allows it to avoid going back out to the bus on the upgrade from exclusive to modified.

20. (Extra Credit; 5 points max.) Your solution to the final programming assignment probably worked correctly only for sequential (single-threaded) programs. How might you extend your allocator to accommodate multithreaded (e.g., `pthread`-based) programs?

**Answer:** You'll need to make sure that concurrent calls by different threads don't corrupt the heap. The simplest approach is simply to create a lock for the allocator, and begin and end every allocator routine with acquire and release operations. This gives correct code, but precludes concurrent operation. A better option is to use more fine-grain locks: e.g., one per free list. The best concurrent allocators maintain separate free lists for each thread, so allocations can proceed in parallel. Tricky parts include what to do when the local free list is empty (or very long); and what to do when a block is freed by a different thread from the one that allocated it.