

# Final Exam

CSC 252

4 May 2005

## Directions; PLEASE READ

This exam has 8 questions, all of which have subparts. Each question indicates its point value. The total is 100 points. **Questions 7(d) and 8 are for extra credit only**; they are not part of the 100 points.

**Please show your work here on the exam, in the space given.** Do not write on the backs or in the margins. Put your name on every page. If your answer won't fit in the given space then you're trying to write too much (or *way* too big). For multiple choice questions, darken the circle next to the best answer. Be sure to read all candidate answers before choosing. This is a *closed-book* exam. You must put away all books and notes. Scratch paper is available if you need it, but the TA will collect **only the exams**.

I have tried to make the questions as clear and self-explanatory as possible. If you do not understand what a question is asking, make some reasonable assumption and *write that assumption down* next to your answer. The TA has been instructed not to try to answer questions during the exam.

You will have a maximum of 3 hours to work, though it shouldn't take that long. Good luck!

### 1. Cache design.

- (a) (4 points.) Explain the principal rationale for having separate L1 instruction and data caches. Describe at least one disadvantage.

**Answer:** *Separate I and D caches can easily be accessed simultaneously by different stages of the pipeline. A single cache that supports two simultaneous accesses is substantially more complex, and potentially slower. A split I&D design also provides higher effective associativity. On the down side it introduces a coherence problem: if a program writes instructions and then tries to execute them we have to do something special to avoid fetching stale data from the L1 cache. Separate caches may also be less space efficient than a unified design if the I/D ratio of a program's working set differs from that of the hardware.*

- (b) (4 points.) Why do modern workstation-class processors have both L1 and L2 caches on-chip, as opposed to a single large L1?

**Answer:** *A cache can't be simultaneously large and fast. The L1 cache is typically made as large as possible while still supporting a single-cycle access time. Remaining chip real estate is devoted to the L2.*

(c) (4 points.) Why aren't most caches fully associative?

**Answer:** *Fully associative caches can be fast only if they are tiny. Some TLBs are fully associative. Data and instruction caches never are.*

(d) (4 points.) Explain the distinction between cold-start, capacity, conflict, and coherence misses.

**Answer:** *A cold-start miss is obligatory: it occurs the first time a datum is accessed. A capacity miss occurs when a datum that was accessed in the past is no longer available because there wasn't enough room to hold everything used in the interim. A conflict miss occurs when a datum that was accessed in the past is no longer available because there wasn't enough associativity to hold everything that mapped to the same set in the interim. A coherence miss occurs when a datum that was accessed in the past is no longer available because it was written by another processor in the interim.*

## 2. Cache parameters.

Consider a machine with the following memory system characteristics:

- direct-mapped 64KB L1 instruction cache
- 2-way associative 128KB L1 data cache
- 4-way associative 2MB unified L2 cache
- 32 byte line size in every cache

(a) (3 points.) How many sets are there in each cache?

**Answer:**  $2^{11} = 2048$  each in L1D and L1I.  $2^{14} = 16384$  in L2.

(b) (3 points.) For each cache, indicate the number of bits in a 32-bit address that will be devoted to tag, index, and offset.

**Answer:** *L1 (I & D): 16-bit tag, 11-bit index, 5-bit offset. L2: 13-bit tag, 14-bit index, 5-bit offset.*

(c) (3 points.) Again assuming a 32-bit address, how many distinct blocks in memory would map to the same line in the L1 instruction cache?

**Answer:**  $2^{16} = 64K$ .

(d) (4 points.) If a virtual memory page is 8KB, will we need virtual indexing in order to do L1 lookup in parallel with address translation? Explain.

**Answer:** *Yes: the 13 bits of page offset are less than the 16 bits of block offset and index; we need 3 bits of page number to complete the value used to index the L1 cache (either I or D).*

## 3. Memory management.

(a) (4 points.) Most processors associate a *use bit* and a *dirty bit* with each entry in the TLB or page table. A page's use bit is set every time the page is accessed. A page's dirty bit is set every time the page is written. Explain why these bits are useful to the operating system.

**Answer:** *The dirty bit tells the operating system whether a page needs to be written back to disk if it is evicted from memory. If the operating system turns the use bit off and then checks it a little while later, it can tell whether the page has been used recently. Pages that haven't been used recently are good candidates for eviction.*

- (b) (4 points.) Summarize the argument for dynamically linked (shared) libraries (as opposed to statically linked).

**Answer:** *Dynamic linking saves space on disk and in main memory. It also allows libraries to be upgraded without having to recompile all the existing program binaries.*

- (c) (4 points.) On a TLB miss, some machines (e.g., the Pentium, the Sparc, and the PowerPC) do page table look-up and TLB reload in hardware. Other machines (e.g., the Alpha and the MIPS) generate an exception, and let the operating system do the look-up and reload. Summarize the comparative advantages of these two strategies.

**Answer:** *Hardware TLB load is a little bit faster. Software TLB load makes the hardware simpler, and allows the OS designer to choose the layout of the page table.*

- (d) (4 points.) Suppose `p` refers to an `n`-byte string and I wish it were `m` bytes instead (where `m > n`). Other than brevity, what is the advantage of saying `p = (char *) realloc(p, m)` instead of

```
{ char *q = (char *) malloc(m);
  if (q) {
    memcpy(q, p, n);
    free(p);
  }
  p = q;
}
```

**Answer:** *If there is sufficient free space following `*p` in the heap, `realloc` simply enlarges the string, avoiding the need to copy.*

#### 4. Architectural constants.

- (a) (3 points.) Order the following Greek prefixes from smallest to largest (they're currently in alphabetical order): giga, kilo, mega, micro, milli, nano, pico, tera.

**Answer:** *pico, nano, micro, milli, kilo, mega, giga, tera.*

- (b) (3 points.) Order the following from slowest to fastest rate of improvement over time: CPU speed, hard disk capacity, memory (DRAM) capacity, memory latency.

**Answer:** *Memory latency, CPU speed, memory (DRAM) capacity, hard disk capacity.*

Multiple choice (2 points each). Choose the most reasonable answer in each case, for a modern workstation-class (deskside) machine.

(c) A load that misses in (all levels of) the cache and is serviced by main memory takes about how long to complete?

- a. one nanosecond
- b. one microsecond
- c. one millisecond
- d. one picosecond

(d) A page fault that forces a read from disk takes about how long to complete?

- a. 10 milliseconds
- b. 100 milliseconds
- c. 100 microseconds
- d. 10 microseconds

(e) An off-chip cache might reasonably be how big?

- a. 4 KB
- b. 256 KB
- c. 4 MB
- d. 256 MB

(f) Such a cache is most likely to be

- a. 4-way associative
- b. 16-way associative
- c. fully associative
- d. direct-mapped

(g) The processor chip is likely to consume approximately

- a. 1 Watt
- b. 10 Watts
- c. 100 Watts
- d. 1000 Watts

## 5. Programming in C.

(a) (4 points.) Explain the difference in C between `int *a()` and `int (*a)()`.

**Answer:** `int *a()` is a function returning a pointer to an integer; `int (*a)()` is a pointer to a function returning an integer.

For each of the C programs on this page and the next, identify a significant problem with either correctness or performance (4 points each). Note: you do not necessarily have to write a new version of the code; just explain what's wrong.

(b) 

```
int my_var;
...
scanf("%d", my_var);
```

**Answer:** This uses the value of `my_var` where it ought to use its address. The second argument to `scanf` should be `&my_var`.

(c) 

```
int average(double scores[], int len, double *ave) {
    int i;
    *ave = 0.0;
    for (i = 0; i < len; i++) {
        if (scores[i] < 0.0 || scores[i] > 100.0)
            return -1;    /* failure */
        *ave += scores[i];
    }
    *ave /= len;
    return 0;    /* success */
}
```

**Answer:** Because `ave` is accessed through a pointer, and there's no way to be sure that `*scores` and `*ave` don't overlap, the compiler will repeatedly access `*ave` in memory. Faster code (and probably more correct, in the rare case where `*scores` and `*ave` do overlap) will result from

```
int average(double scores[], int len, double *ave) {
    int i;
    double temp = 0.0;
    for (i = 0; i < len; i++) {
        if (scores[i] < 0.0 || scores[i] > 100.0)
            return -1;    /* failure */
        temp += scores[i];
    }
    *ave = temp/len;
    return 0;    /* success */
}
```

```
(d) int A[100];
    int i, j;
    ... /* initialize i and j to point into A, with i < j */
    int *p = &A[i];
    int *q = &A[j];
    int *copy = (int *) malloc(q-p);
    if (copy) {
        int *t;
        for (t = copy; p < q; ) *t++ = *p++;
    }
}
```

**Answer:** This doesn't allocate enough space for copy. The argument to malloc should be (q-p) \* sizeof(int).

```
(e) struct int_node {
    int val;
    struct int_node *next;
}
...
void delete_list(struct int_node *n) {
    while (n) {
        free(n);
        n = n->next;
    }
}
```

**Answer:** This uses a dangling reference. The body of the while loop needs to be something like

```
    struct int_node *t = n->next;
    free(n);
    n = t;
```

## 6. Averages.

Recall the following definitions:

$$\begin{aligned} \text{arithmetic\_mean}(v_1, v_2, \dots, v_n) &= \frac{1}{n} \left( \sum_{i=1}^n v_i \right) \\ \text{harmonic\_mean}(v_1, v_2, \dots, v_n) &= n \left( \sum_{i=1}^n \frac{1}{v_i} \right)^{-1} \\ \text{geometric\_mean}(v_1, v_2, \dots, v_n) &= \left( \prod_{i=1}^n v_i \right)^{1/n} \end{aligned}$$

Which mean should you use for each of the following computations (3 points each)?

- (a) Program  $k$  executes  $v_k$  instructions per cycle. What is the average number of instructions per cycle across all programs?

- a. arithmetic mean
- b. harmonic mean
- c. geometric mean
- d. none of these

(b) Program  $k$  executes  $v_k$  instructions. How many instructions does the average program execute?

- a. arithmetic mean
- b. harmonic mean
- c. geometric mean
- d. none of these

(c) In the process of optimizing a program I make a series of  $n$  improvements. Improvement  $k$  reduces total run time by a factor  $v_k$ . What is the average reduction in run time across all improvements?

- a. arithmetic mean
- b. harmonic mean
- c. geometric mean
- d. none of these

## 7. Miscellaneous.

(a) (4 points.) Let  $A = 00101010$  and  $B = 11111101$  be 8-bit two's complement integers. Compute  $A + B$  and  $A \times B$ , and express the results as 8-bit two's complement integers. (Feel free to convert to decimal, do the math, and convert back again if you like.)

**Answer:**  $A = 42_{10}$ ;  $B = -3_{10}$ .  $A + B = 39_{10} = 00100111_2$ ;  $A \times B = -126_{10} = 10000010_2$ .

(b) (3 points.) What's the principal difference between a multithreaded processor like the "hyperthreaded" Intel Pentium 4 and a chip-level multiprocessor like the IBM Power 4?

**Answer:** *The two threads on the Pentium share a single set of functional units. The two processors of the Power 4 have separate sets.*

(c) (3 points.) What's the principal difference in Unix between a *thread* (created with `pthread_create`) and a *process* (created with `fork`)?

**Answer:** *Threads share an address space; processes don't.*

(d) (**Extra Credit; 3 points max.**) Why do most machines have separate user-mode and kernel-mode stack pointers, but only a single copy of the other registers? (Hint: think about exceptions/interrupts.)

**Answer:** When an exception happens, the processor needs to push state onto the stack, in hardware. The user stack pointer may not point to a place that's convenient for the kernel. In fact there's no guarantee that the location it points to will even be valid. If we use a separate kernel pointer, however, any other registers that the exception handler needs can be saved onto the stack before use.

8. **Cache-friendly code (extra credit).**

Consider the following code in C:

```
double A[N][N];
...
for (j = 1; j < N; j++) {
    for (i = 0; i < N-1; i++) {
        A[i][j] -= A[i+1][j-1];
    }
}
```

- (a) **(Extra Credit; 3 points max.)** How well does this code use the cache? If  $N$  is large, approximately how many cache misses can we expect?

**Answer:** This code walks the matrix in column-major order, which is the opposite of allocation order in C, so the code makes very poor use of the cache. If  $N$  is large we can expect  $O(N^2)$  cache misses.

- (b) **(Extra Credit; 4 points max.)** In lecture we saw a program (to sum all elements of a matrix) in which cache locality could be improved by interchanging the order of the loops. Unfortunately, that change (making the outer loop iterate over  $i$  and the inner loop over  $j$ ) would not be valid in the example above. Explain why not.

**Answer:** Iteration  $(j, i)$  uses a value produced in iteration  $(j - 1, i + 1)$ . If we interchange the order of the loops we'll end up using this value before it is produced. The code won't be correct.

- (c) **(Extra Credit; 5 points max.)** Is there some other way to transform the code so that it will be both fast *and* correct?

**Answer:** The code will work correctly if we reverse the direction of the inner ( $i$ ) loop and then interchange the loops:

```
for (i = N-2; i >= 0; i--) {
    for (j = 1; j < N; j++) {
        A[i][j] -= A[i+1][j-1];
    }
}
```

Now the value on the right-hand side of the assignment is still computed before it is used.