# CS 341l Fall 2008 Test #3

Name: _____Key_____

CS 341l, test #3. Friday, 31 October 2008. 100 points total, number of points each question is worth is indicated in parentheses. Answer all questions. Be as concise as possible while still answering the question. **You have 50 minutes exactly, I'll collect the tests at 1:50pm.** Closed book, closed notes, closed calculator, closed everything. **Check right now to make sure you have all pages of the test.**

#1. (15 points, 3 points each) Circle T for true or F for false.

*The miss rate is equal to 1 minus the miss penalty.

F

* An inverted page table's size is directly proportional to the size of the virtually addressable memory space, hence this technique is not applicable to 64-bit machines.

F

*A victim cache primarily helps with compulsory misses, not capacity or conflict.

F

*The Pentium 4 replaces the D-cache with a trace cache, but still has an I-cache, so is still considered to have a split L1 cache.

F

*When writing C code, it is helpful to write an arithmetic overflow exception handler so that arithmetic overflows will not crash your program.

F

#2. (20 points, 4 points each) For each of the following caches, what will be the number of sets, the number of lines per set, and the number of bits in the tag? Assume a 32-bit, physically addressed, byte-addressed, word-aligned cache (like a typical MIPS processor would have). Hint to help you with the arithmetic: a KB is $2^{10}$, a MB is $2^{20}$, so, *e.g.*, 8MB is $2^{23}$. You can divide powers of two by subtracting their exponents, so, for example, $2^{23}/2^{9}=2^{14}$. It's okay to write powers of two as your answer.

*8MB cache, 512-byte cache lines, 4-way set associative.

# of sets: ___*2^12*____ # cache lines per set: ____*4*____ #bits in the tag: ___*11*___

*512KB cache, 256-byte cache lines, direct-mapped.

# of sets: __*2^11*____ # cache lines per set: __*1*____ #bits in the tag: _*13*___

*512KB cache, 512-byte cache lines, fully associative.

# of sets: _*1*____ # cache lines per set: __*2^10*_____ #bits in the tag: __*23*___

*1MB cache, 128-byte cache lines, 2-way set associative.

# of sets: ___*2^12*__ # cache lines per set: __*2*_____ #bits in the tag: __*13*____

*16KB cache, 64-byte cache lines, 4-way set associative.

# of sets: _*2^6*____ # cache lines per set: ____*4*____ #bits in the tag: __*20*____

#3 (20 points, 4 points each). For each of the following paging schemes in "a + b + c ..." notation, where each subsequent number is the number of bits used to index increasing levels of page tables and the last number is the number of bits used to index within a page, write the size of each page, the number of entries in each second-level page table, the number of paging levels, and the size of the virtually addressable address space. No calculator necessary, write a power of 2 if you must. The first (the Pentium) is done for you. If an entry does not apply, write "N/A".

| | Page size | # Entries in second level page table | # Levels | Virtually addressable address space |
|---|---|---|---|---|
| 10+10+12 | 4 KB | 1024 | 2 | 2^32 = 4GB |
| 10+22 | *4 MB* | *N/A* | *1* | *4 GB* |
| 9+9+10+13 | *8 KB* | *512* | *3* | *2 TB* |
| 9+9+14 | *16 KB* | *512* | *2* | *4 GB* |
| 12+12+12+14 | *16 KB* | *4096* | *3* | *1 PB* |
| 10+10+10+24 | *16 MB* | *1024* | *4* | *16 PB* |

#4. (40 points, 5 points each) Multiple choice. Circe all answers that apply. Your answer could be all of them, none of them, or any subset. These will each be graded like five 1-point true-false questions.

*Which of the following measurement scenarios are possible:

***a. A superscalar processor has IPC > 1.0.***

b. A scalar pipelined processor with forwarding support has IPC > 1.0.

c. A scalar pipelined processor without forwarding support has IPC > 1.0.

***d. A superscalar processor has IPC < 1.0.***

e. A scalar pipeline has CPI < 1.0.

*Which of these is likely to happen if I increase associativity in a cache, but hold all else (total size, cache line size, *etc.*) the same:

a. The number of capacity misses will go down.

***b. The number of conflict misses will go down.***

c. The number of compulsory misses will go down

***d. The hit time will go up.***

e. The miss penalty for misses in that cache will go down.

*Which of these might happen if I increase the cache line size (or block size) in a cache, but hold all else (total size, associativity, *etc.*) the same:

***a. The miss rate for that cache might go up.***

***b. The miss rate for that cache might go down.***

***c. The number of conflict misses in that cache might increase.***

***d. I might be able to take advantage of more spatial locality.***

e. I might reduce the miss penalty for misses to that cache.

*Which of these is likely to happen if I increase the total size of a cache, but hold all else (associativity, cache line size, *etc.*) the same:

***a. The number of capacity misses will go down.***

***b. The number of conflict misses will go down.***

c. The number of compulsory misses will go down

***d. The hit time will go up.***

e. The miss penalty for misses in that cache will go down.

*Virtual memory can provide:

***a. Memory protection for multiple processes.***

b. A reduction in latency for accesses to DRAM.

***c. The illusion of a larger amount of available physical memory.***

d. A buffer for cache writes between the L2 cache and DRAM.

***e. For each process to assume its own flat view of memory.***

*Which of these structures are indexed with virtual addresses on a typical CPU with two-level paging, two levels of physically-addressed cache, a TLB, and a branch target buffer, assuming that virtual memory protections are enabled.

***a. A first-level page table***

***b. A second-level page table***

c. The L2 cache

***d. The TLB***

***e. The branch target buffer***

*Which of these schemes for handling writes in different levels of the memory hierarchy (assume a two-level cache) would be feasible for some reasonable application:

***a. Write-through from the L1 cache to the L2 cache.***

***b. Write-buffer from the L2 cache to the DRAM.***

***c. Write back from the L1 cache to the L2 cache.***

***d. Write-back from the L2 cache to the DRAM.***

e. Write through from the DRAM to the hard drive.

*Which of the following sequences of events in different levels of the memory hierarchy (assume a two-level cache) would be possible on a typical CPU for a single memory lookup, assuming that unnecessary lookups are not done by that CPU:

a. Hit in the TLB, hit in the page table, hit in the cache.

b. Hit in the TLB, hit in the page table, miss in the cache.

c. Miss in the TLB, miss in the page table, hit in the cache.

***d. Miss in the TLB, hit in the page table, miss in the cache.***

***e. Miss in the TLB, hit in the page table, hit in the cache.***

#5. (4 points), multiple choice, **circle only the one correct answer**. Suppose I have a 64-bit machine with a three-level, 12+12+12+16 virtual memory scheme. Which of the following is the correct calculation for how many bytes of memory a process that fills up its entire virtual memory space will consume just for its page tables? (Arithmetic is in C notation, so * is multiply).

***a. $2^{12}*8 + 2^{12}*2^{12}*8 + 2^{12}*2^{12}*2^{12}*8$ = over 500GB!***

b. $2^{12}*8 + 2^{12}*8 + 2^{12}*8 + 2^{16}*8$ = 608 KB

c. $2^{12}*64 + 2^{12}*64 + 2^{12}*64 + 2^{16}*64$ = almost 5MB

d. $2^{12+12+12+16}$= 4 petabytes

#6. (1 point) In *The Matrix Reloaded* (yes, I know it was a bad movie, but this part was good), Trinity uses a real-life network scanning tool to access a power plant's computer system. What is that tool called? *nmap*