CS 351 Exam 3 Mon. 5/5/2014

Name: ______________________________

Rules and Hints

- You may use one handwritten 8.5 × 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- Include step-by-step explanations and comments in your answers, and show as much of your work as possible, in order to maximize your partial credit.
- You may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.

Grade

	Your Score	**Max Score**
Problem 1: Cache mappings		30
Problem 2: Cache write policies and coherence		15
Problem 3: Virtual memory		20
Problem 4: I/O and RAID		25
Problem 5: Task- and data-parallelism		10
Total		**100**

Problem 1: Cache mappings (30 points)

For all of the questions below, assume that addresses in main memory are 12 bits.

Part A: Direct-mapped cache (5 points)

Consider a cache that

- Has 8 blocks (entries)
- Is direct-mapped
- Has 8 bytes per block
- Is write-back

Draw this cache, showing any necessary metadata. Assuming that the cache is initially empty, show the state of the cache after accessing address 5 (000...101).

Part B: Fully-associative cache (5 points)

Consider a cache that

- Has 8 blocks (entries)
- Is fully associative
- Has 2 bytes per block
- Is write-through

Draw this cache, showing any necessary metadata. Assuming that the cache is initially empty, show the state of the cache after accessing address 5 (000...101).

Part C: Set-associative cache (5 points)

Consider a cache that

- Has 8 blocks (entries)
- Is 2-way set-associative
- Has 2 bytes per block
- Is write-back

Draw this cache, showing any necessary metadata. Assuming that the cache is initially empty, show the state of the cache after accessing address 5 (000...101).

Part D: Conflict misses (10 points)

For *each* of the above caches, show a sequence of one or more memory references that will cause your data at address 5 to be evicted from the cache.

Part E: Sizing (5 points)

Pick your favorite cache from Parts (A) through (C). How many *bytes* is it (data and metadata)?

Problem 2: Cache write policies & coherence (15 points)

Assume that you have a dual-core processor with private L1 caches as specified in Problem 1A, and a shared L2 cache.

Part A: Coherence definition (5 points)

Assume that this processor does not have hardware-enforced cache coherence. Show a sequence of references (e.g. “Read X”, “Read Y”) for each processor that could potentially lead to a coherence problem, and explain.

Part B: Write policies and coherence (5 points)

Continue to assume that this processor does not have hardware-enforced cache coherence. Show a sequence of references for each processor, and explain an ordering of these references that would lead to a coherence problem *only* if the L1 caches are write-back, but not if they are write-through.

Part C: False sharing (5 points)

Now assume that this processor *does* have hardware-enforced cache coherence, using a write-invalidate protocol that is enforced for each cache block. Show a sequence of references for each processor that could potentially lead to a false sharing problem, and explain.

Problem 3: Virtual memory (20 points)

Assume a system with 32 GB of main memory, 64-bit virtual addresses, and 16 KB pages.

Part A: Page table sizing (10 points)

- How many virtual pages does each process have?
- How many physical pages does the entire system have?
- How many page table entries does each process have?
- If the TLB is invalidated on every context switch (so it doesn't need to store process IDs), and each TLB entry has 3 LRU bits and stores 3 protection bits, how many total bits are needed for each TLB entry?

Part B: Address translation (10 points)

Assume that virtual page 50 is currently mapped to physical page 92. Your process requests some data from this page. This specific byte is currently in the L2 cache but not the L1 cache.
Assume that your system has a TLB and has flat per-process page tables. Describe the steps in accessing this data, starting with the processor's request for the virtual address and ending with the data being returned to the processor.

Problem 4: I/O and RAID (25 points)

A certain hard disk has the following specifications:

- Average seek time: 10 ms
- Rotation speed: 10,000 rpm
- Data transfer rate: 200 MB/sec
- Capacity: 2 TB

Part A: Performance (4 points)

How long does this disk take to do a 2 MB transfer?

Part B: Energy (2 points)

The same disk has the following power requirements:

- Power during seek: 5 W
- Power while just spinning: 4.5 W
- Power during read/write: 5.5 W

How much *energy* will this disk use during the course of the 2 MB transfer?

Part C: RAID 1 (9 points)

You construct a RAID 1 array from 10 of these disks, which includes redundant disks.

- How much data (in TB) can you store on this array, not counting redundant data?
- Assuming that small accesses are evenly distributed across your disks, how many small reads can you do simultaneously across the array?
- Assuming that small accesses are evenly distributed across your disks, how many small writes can you do simultaneously across the array?
- Assume that large accesses from a single JBOD disk take time $L + T$ (latency plus transfer time). In terms of L and T, how long does a large read take on your RAID 1 array?
- How long does a large write take on your RAID 1 array?

Part D: RAID 5 (10 points)

You construct a RAID 5 array from 10 of these disks, which includes redundant disks.

- How much data (in TB) can you store on this array, not counting redundant data?
- Assuming that small accesses are evenly distributed across your disks, how many small reads can you do simultaneously across the array?
- Assuming that small accesses are evenly distributed across your disks, how many small writes can you do simultaneously across the array?
- Assume that large accesses from a single JBOD disk take time $L + T$ (latency plus transfer time). How long does a large read take on your RAID 5 array?
- How long does a large write take on your RAID 5 array?

Problem 5: Task- and data-parallelism (10 points)

Part A: GPUs (3 points)

Which of the following would be better suited to running on a GPU? Explain.

- Matrix multiplication
- Inserting into a doubly linked list

Part B: Metrics (3 points)

Draw a graph of threads vs. speedup. Show two lines:

- One line should show strong scalability from 1 to 256 threads.
- One line should show an *efficiency* of approximately 75% at 32 threads and 50% efficiency at 256 threads.

Part C: Task parallelism (4 points)

Consider the following code.

```
A = f1()       // Line 1
E = f2(A)      // Line 2
F = f3()       // Line 3
G = f4(E, F)   // Line 4
H = f5(A)      // Line 5
```

Draw the dependency graph for this code, with one node per line of code. Label each node with an amount of time that will lead to the best possible speedup when this graph is split across multiple threads.