CIS501 – Computer Architecture	**Final Exam**
Prof. Martin	**Monday, Dec. 20, 2010**

This exam is an individual-work exam. Write your answers on these pages. Additional pages may be attached (with staple) if necessary. Please ensure that your answers are concise and legible. Read and follow the directions of each question carefully. Please attempt to answer all the questions (don't allow yourself to get stuck on a single question). You have 120 minutes to complete the exam (approximately one point per minute).

If you are taking this exam as a WPE-I:

Your WPE-I Number: _____

If you are *NOT* taking this exam as a WPE-I:

Name: _____

Problem	Page	Possible	Score
1	2	9	
2	3	8	
3	4	10	
4	5	9	
5	6	8	
6	7	10	
7	8	9	
8	9	9	
9	10	9	
10	11	8	
Total		89	

1. **[9 Points] True or False**. If a statement is "false," briefly explain how so by describing how the statement may most simply be made true. Unjustified (or poorly justified) "false" answers will be marked wrong. Simply stating the negation of the false statement is *not* sufficient justification. *Please be specific!*

   (a) *IQ clog* can be remedied by eating more fiber.

   (b) A fully-associative cache will never have conflict misses.

   (c) A store buffer (also called a write buffer) is primarily intended to hide store latency.

   (d) Translation lookaside buffers (TLBs) reduce the total size of the page table.

   (e) Each application process has its own page table, and that page table is maintained by the operating system.

   (f) A multiplexed single core, a multicore, and a multithreaded core all provide the same shared-memory programming model.

   (g) A memory consistency model specifies the "many readers" or "single writer" property for a single memory location.

   (h) Today's microprocessors use long vectors (64 32-bit elements or larger) to maximize the performance benefits of vectorization.

   (i) Power-related issues are relevant only when designing battery-powered devices.

2. **[8 Points] Pipelining and Clock Frequency**. Consider a workload with 40% branches and a 75% branch prediction accuracy (25% misprediction rate).

   (a) For a simple five-stage in-order scalar pipeline, what is the CPI of this workload assuming a three-cycle mis-prediction penalty and a single-cycle latency for all other instructions?

   (b) Consider a much deeper pipeline (but still scalar in-order) that has double the core clock frequency. Calculate under what condition does this deeper pipeline outperform the shallower pipeline?

   (c) Based on what you have learned about pipelines and calculated above, does this deeper pipeline seem like a reasonable design point? Why or why not?

   (d) If cache misses and memory latency were taken into account, would the above calculation and conclusion change? If so, how? If not, why?

3. **[10 Points] Branch Direction Prediction**

(a) What is a *gshare* predictor and why does it generally increase prediction accuracy?

(b) For fixed branch predictor size, sometimes a *bimodal* outperforms *gshare*. Give two distinct reasons this can occur.

1.

2.

(c) What technique is used to mitigate these tensions between bimodal and gshare?

(d) Consider a predictor with a single saturating counter which is either 1-bit or 2-bits. The 1-bit counter encodes either "Taken (T)" or "Not-taken (N)", and it is initialized to "Not-taken". The 2-bit counter encodes "Strongly Taken (T)", "Weakly Taken (t)", "Weakly Not-taken (n)" and "Strongly Not-taken (N)", and it is initialized to "Weakly Not-taken (n)".

i. Give a short sequence (4 or fewer) of branch directions (T or N) in which a 2-bit saturating counter is better than a 1-bit saturating counter. What is the prediction accuracy for *both* predictors?

ii. Give a short sequence (4 or fewer) of branch directions (T or N) in which a 1-bit saturating counter is better than a 2-bit saturating counter. What is the prediction accuracy for *both* predictors?

4. **[9 Points] Branch Target Prediction**

(a) What are the two purposes of a branch target buffer (BTB)?

1.

2.

(b) Why is a BTB generally tagged whereas the branch history table (BHT) used in the bimodal and gshare predictors is not?

(c) Even though the BTB is typically tagged, it often contains only a partial tag (just 8 or 16 bits) rather than a full tag (up to 64 bits on a 64-bit architecture). What is the advantage of using just a partial tag?

What is the disadvantage of using a partial tag?

Why is using the partial tag still a "correct" implementation?

(d) The BTB is used for even for unconditional branches, whereas a branch direction predictor is used for only conditional branches. Why?

(e) What is the purpose of a *return address stack*? Assuming a processor with a BTB, why might it also have a *return address stack* in addition to the BTB?

5. **[8 Points] Memory Hierarchy Performance**

Your job is to decided on the cache configuration of your companies next-generation chip. Two of your options are: (1) a direct-mapped cache with a **two-cycle hit latency** or (2) a set-associative cache with a **three-cycle hit latency**. In both cases, the miss latency is an *additional* 10 cycles.

(a) If the miss rate of the direct-mapped cache is 15%, what is its average latency of a memory operation?

(b) What miss rate must the set-associative cache obtain to have a lower average latency than the direct-mapped cache with a 15% miss rate?

(c) Alternatively, a victim cache could be used to reduce the miss latency of the cache to just 5 cycles when a miss hits in the victim cache (the victim cache and next-level of the hierarchy are accessed in parallel, so it is still just a 10 cycle miss penalty if it doesn't hit in the victim buffer). Assuming the direct-mapped cache miss rate is 15% and the victim cache has a local hit rate of 50% (that is, half of all misses find the block in the victim cache), what is the new average latency?

(d) Another alternative is to use use way prediction to obtain the miss rate of the set-associative cache but the hit latency of the direct-mapped cache. Assume that a way mis-prediction adds 5 cycles to the cache hit latency but doesn't impact the latency of misses.

If the set-associative cache and way-prediction cache both have a miss rate of 10%, how accurate must the way predictor be for the way prediction cache to out perform the original set-associative cache? That is, what percent of cache hits must be predict correctly?

6. **[10 Points] Register Renaming**

(a) What is the purpose of register renaming?

Consider the following code for a processor with four logical registers (r0-r3):

```
add r0, r1 -> r2
ld [r2] -> r1
add r0, r1 -> r1
ld [r0] -> r3
```

(b) Rename the above instructions assuming **eight** physical registers (p0-p7) (by replacing all the logical registers with the appropriate physical registers). Assume that r0 is initially mapped to p0, r1 to p1, r2 to p2, and r3 to p3. The remainder of the physical registers are on the free list in the order p4, p5, p6, p7.

(c) Rename the above instructions assuming a **six** physical registers (p0-p5). Assume the same initial mapping as above; the free list contains p4 and p5 (in that order).

(d) How does this fewer number of physical registers concretely impact the execution of this code?

7. **[9 Points] Store Sets**

(a) What problem is addressed by the Chrysos and Emer's *store sets* paper?

(b) In the paper, the predictor can be "wrong" in two different ways. What are those ways and what is the consequence of each?

1.

2.

(c) As defined in the paper, what is a load's *store set*?

(d) What key idea does the paper introduce to simplify the implementation of store sets? How does this simplify things?

(e) As described in the paper, the stores sets approach also constrains the execution order of stores. Why?

8. **[9 Points] Cache Coherence**

(a) What is the main advantage of the "MSI" protocol over the "VI" cache coherence protocol?

(b) What is the main advantage of the "MESI" protocol over the "MSI" cache coherence protocol?

(c) What is *false sharing*?

(d) How can software help prevent false sharing?

(e) What two issues with a *snooping* protocol does a *directory* cache coherence protocol fix? How does it fix each issue?

1.

2.

(f) When considering the future of multicore systems, what is the biggest technical challenge?

9. **[9 Points] Synchronization & Consistency**

(a) What is the primary disadvantage of coarse-grained locking?

(b) What are two disadvantages of employing fine-grained locking?
1.

2.

(c) What research proposal is aimed at mitigating the above issues with locking granularity?

(d) Assuming a system with a standard coherence protocol, describe the advantage of a "test-and-test-and-set" spin lock versus a simple "test-and-set" spin lock implementation?

(e) Give two specific implementations reasons that a hardware implementer might want to design a chip that is not *sequentially consistent* (that is, allows the unexpected behaviors we discussed).
1.

2.

(f) Even if the hardware does not reorder operations (that is, is sequential consistent), a programmer may still observe outcomes that are not sequentially consistent. Why?

10. **[8 Points] Dynamic Scheduling & Limits of ILP**. Consider the following C code:

```
double sum_square(double x[], int size) {
  double sum = 0.0;
  for(int i = 0; i < size; i++) {
    double value = x[i];
    double square = value * value;
    sum = sum + square;
  }
  return sum;
}
```

The loop body can be translated into six x86-like assembly instructions, one of which is a load:

```
.L2: movsd (%rdi,%rax) -> %xmm1    // load (-> value)
     mulsd %xmm1, %xmm1 -> %xmm1   // multiply (-> square)
     addsd %xmm1, %xmm0 -> %xmm0   // add (-> sum)
     addq  $8, %rax -> %rax        // add (-> i)
     cmpq  %rdx, %rax              // reads both registers, writes the flags
     jne   .L2                     // reads the flags
```

In this question, the loop executes an extremely large number of iterations (`size` is extremely large). Assume a fully idealized dynamically scheduled superscalar (perfect branch prediction, perfect infinitely wide fetch, rename, execute, commit and unbounded reorder buffer and instruction queues).

(a) If all instructions have a single-cycle latency, what is the maximum IPC? Why?

(b) If the `mulsd` and `addsd` have a four-cycle latency and all other instructions have a single-cycle latency, what is the maximum IPC? Why?

(c) If the `movsd` (the load) repeatedly misses in the cache, and thus has a 16-cycle latency and **all other instructions have a single-cycle latency**, what would the maximum IPC be? Why?

(d) For the 16-cycle load latency configuration above, **estimate** how large must the "instruction window" (say, size of the re-order buffer) be to achieve this IPC? Assume that instructions are magically fetched, decoded, and dispatched in a single cycle. Assume that instructions are cleared out of the reorder buffer (ROB) the same cycle the complete execution (assuming all older instructions have also completed, of course). **Explain your answer**. (Note: your reasoning is more important than the exact number of entries.)

— End of exam —

Scratch space