CIS501– Computer Architecture	**Final Exam**
Prof. Martin	**Tuesday, Dec. 22, 2009**

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	7	
2	3	8	
3	4	11	
4	5	9	
5	6	9	
6	7	10	
7	8	10	
8	10	15	
9	11	9	
10	12	8	
11	13	14	
Total		110	

1. **[7 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) "The Alpha 21264 architecture is very dynamic."

   (b) Moore's law predicts an exponential improvement in transistor switching speeds over time.

   (c) The cost to manufacture a chip is proportional to the area of the chip.

   (d) A compiler optimization can increase performance yet hurt CPI.

   (e) Clustering (such as that used by the Alpha 21264) is one approach for tackling the n^2 problem of dependence cross checking logic.

   (f) Assuming similar pipeline depths, high branch prediction accuracy is generally more important in a superscalar (multiple-issue) processor than in a scalar (single-issue) processor.

   (g) An easy way to exploit data parallelism in your programs is to call library code that has been optimized to use vector instructions.

2. **[8 Points] Multiplying Performance**.

(a) Consider a computation that consists of calculating the sum of thousands of integers. Using a simple single-cycle datapath as a starting point, what are four techniques or approaches we discussed that *each* can increase performance by a factor of four or more (that is, altogether these four techniques could result in a 256x speedup!)

(Single word answers are sufficient.)

1.

2.

3.

4.

3. **[11 Points] Performance & ISAs (Part 2)**.

An alternative way of accelerating the computation from the previous question is introducing a new three-input `ADD3` instruction to your favorite ISA. This new instruction operates on normal 64-bit registers only and performs the computation $A = B + C + D$.

(a) What impact–*if any*–would adding the `ADD3` instruction have on the following aspects of a pipelined datapath:

Instruction fetch:

Stall logic:

Bypassing:

Register file:

Execution units:

Data cache:

(b) What impact–*if any*–would adding this `ADD3` instruction have on the following aspects of a dynamically-scheduled pipeline?

Register renaming logic:

Instruction wakeup logic:

Instruction commit logic:

(c) An alternative way of implementing `ADD3` would be by using micro-ops. What would be the key advantage and disadvantage of implementing `ADD3` in this way?

4. **[9 Points] Virtual Memory**.

Consider a system with a **42-bit virtual address** space and **4KB pages**. Consider a single-level, two-level, and three-level page table organization (with the indexing bits evenly divided among the levels). Each entry in the page table nodes is **4 bytes**. For a program that uses the entire first **8GB (**2^{33} **bytes)** of the virtual memory space, how many bytes are allocated for each level of the page table under each organization?

(a) How many bits in the page offset:

(b) How many bits in the virtual page number:

(c) Single-level page table:

i. Bytes for first (only) level:

(d) Two-level page table:

i. Bytes for first (root) level:

ii. Bytes for second level:

(e) Three-level page table:

i. Bytes for first (root) level:

ii. Bytes for second level:

iii. Bytes for third level:

(f) In addition to a page table per process (which maps virtual to physical pages), the operating system also keeps a single "inverse" page table (which maps physical to virtual pages). Why?

5. **[9 Points] Caching**.

Consider the following progression of on-chip caches for three generations of chips (loosely modeled after the Alpha 21064, 21164, and 21264).

(a) Assume the first generation chip had a direct-mapped 8KB single-cycle first-level data cache. Assume t_{hit} for this cache is one cycle, t_{miss} is 100 cycles (to accesses off-chip memory), and the miss rate is 20%. What is the average memory access latency?

(b) The second generation chip used the same direct-mapped 8KB single-cycle first-level data cache, but added a 96KB second-level cache on the chip. Assume the second-level cache has a 10-cycle hit latency. If an access misses in the second-level cache, it takes an additional 100 cycles to fetch the block from the off-chip memory. The second-level cache has a global miss rate of 4% of memory operations (which corresponds to a local miss rate of 20%). What is the average memory access latency?

(c) The third generation chip replaced the two-levels of on-chip caches with a single-level of cache: a set-associative 64KB cache. Assume this cache has a 5% miss rate and the same 100 cycle miss latency as above. Under what conditions does this new cache configuration have an average memory latency lower than the second generation configuration?

(d) Notably, the second generation chip was an in-order pipeline, whereas the third generation was an out-of-order pipeline design. How might this difference have influenced the design of the memory hierarchies described above?

6. **[10 Points] Pipelining**.

The standard pipeline discussed in class has five stages: fetch (F), decode (D), execute (X), memory (M), and writeback (W). The pipeline is fully bypassed and branches resolve at the end of the X stage. Consider the following pipelines, formed either by spreading the logic of a single pipeline stage across two stages (six-stage pipelines) or by combining pipeline stages (four-stage pipelines).

(a) What impact would splitting `F` into two stages (`F1` and `F2`) have on CPI?

(b) What impact would splitting `D` into two stages (`D1` and `D2`) have on CPI?

(c) What three distinct impacts would splitting `X` into two stages (`X1` and `X2`) have on CPI?

1.

2.

3.

(d) What impact would splitting `M` into two stages (`M1` and `M2`) have on CPI?

(e) How would combining the `D` and `X` stages impact CPI?

(f) How would combining the `X` and `M` stages impact CPI?

(g) How could combining the `M` and `W` stages positively impact the clock frequency of the pipeline?

(h) How might combining the `F` and `D` stages impact the implementation of branch prediction?

7. **[10 Points] Conflicts and Tagged Predictors**.

You're a microprocessor designer, and your trace-based simulations of an important workload indicate that branch instructions at the following two 32-bit addresses are executed frequently:

- Address `A`: `1000 1101 1100 0011 0110 1011 0100 1001`
- Address `B`: `1000 1101 0110 1001 1110 1011 0100 1001`

(a) The above two addresses cause frequent and repeated conflicts in a small direct-mapped instruction cache. Instead of implementing a set-associative cache, you instead decide to increase the size of the direct-mapped cache. How large must the direct-mapped cache be before these two addresses no longer conflict with each other?

(b) The branch direction predictors that we discussed are all implicitly direct-mapped because they don't have tags. Your simulations of a simple "bimodal" predictor of two-bit saturating counters indicate that the branches at addresses A and B also map to the same entry in the branch predictor (and have conflicting biases). How many entries must the predictor contain to prevent these two branches from interfering (conflicting) with each other? How many total bytes?

[Continued on next page]

(c) You have a brilliant idea: "Why not create a set-associative branch direction predictor?". Your simulations indicate that a predictor with just 2k entries (512 bytes) would be sufficient if it wasn't for these two trouble branches. Consider a two-way set-associative predictor that uses a straightforward tagging strategy (one tag for each two-bit counter, instructions have 32-bit addresses). How large (in KBs) would a two-way set-associative 2k-entry tagged predictor be?

(d) You have *another* brilliant idea: "Because this is just a predictor, it can be wrong, so it doesn't actually need the full tags, just enough of a tag to avoid this particular conflict". With this new insight, (1) how large should the tags be and (2) what is the total size in KBs of this predictor?

(e) How might a predictor capture both the conflict-mitigating benefits of a tagged set-associative predictor and most of the area efficiency of a tag-less predictor?

8. **[15 Points] Dynamic Scheduling**.

Consider the following seven-stage dynamically scheduled pipeline. It uses aggressive load scheduling, and it is similar to both the Alpha 21264 and the pipelined used in the lecture notes, but for simplicity the various execute, memory, and writeback stages have been coalesced into a single "X" stage. The stages are:

F Fetch
R Rename
D Dispatch
I Issue
RR Register Read
X Execute, Memory, and Writeback
C Commit

(a) In the table below, fill in each empty box with the shorthand name of the stage (that is, `F`, `R`, `D`, `I`, `RR`, `X`, or `C`) in which the operation (column) is performed on the structure (rows):

Structure	Entry Allocated	Entry Read	Entries Searched	Entry Written	Entries Updated	Entry Deallocated
Register File			—		—	
Instruction Queue		—				
Store Queue					—	
Load Queue		—			—	
Data Cache	—		—		—	—

(b) *Store queue writes* are triggered by (circle one): loads stores both

(c) *Store queue searches* are triggered by (circle one): loads stores both

(d) *Load queue writes* are triggered by (circle one): loads stores both

(e) *Load queue searches* are triggered by (circle one): loads stores both

(f) When using *conservative memory scheduling*, is the store queue necessary? Why or why not?

(g) When using *conservative memory scheduling*, is the load queue necessary? Why or why not?

(h) How does the Alpha 21264 (which uses aggressive memory scheduling) avoid repeated memory ordering violations?

9. **[9 Points] Thread-Level Parallelism and Multicore**.

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

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

2.

(c) What is the primary advantage of the "MSI" protocol over the simpler "VI" cache coherence protocol?

(d) What is the primary advantage of the "MESI" protocol over the simpler "MSI" cache coherence protocol?

(e) Describe *false sharing*. How can software help avoid false sharing?

(f) What is a *memory fence* (also known as *memory barrier*)? Where are they typically used?

10. **[8 Points] Limits of Performance**.

(a) Give two key reasons not to build a processor with a **1000-stage pipeline**:
Implementation reason:

More fundamental reason:

(b) Give two key reasons not to build **1000-issue superscalar** processor:
Implementation reason:

More fundamental reason:

(c) Give two key reasons not to build a processor with a **1000-instruction scheduling window**:
Implementation reason:

More fundamental reason:

(d) Give two key reasons not to build a **1000-core multicore**:
Implementation reason:

More fundamental reason:

11. **[14 Points] General Themes & Concepts**.

(a) We have seen the general idea of "learning from the past to predict future behavior" employed in several different contexts. Give three distinct examples of prediction and the problem it is used to attack in each case:

1.

2.

3.

(b) Another general technique we have seen is "adding a level of indirection". Give two distinct examples of indirection and the problem it is used to attack in each case:

1.

2.

(c) Caching is another general technique. *Besides processor caches* for data and instructions, we discussed several other applications of the concept of caching or locality (spatial or temporal). Give two examples.

1.

2.

— End of exam —

For reference:

Hex	Binary	Decimal
0x0	0000	0
0x1	0001	1
0x2	0010	2
0x3	0011	3
0x4	0100	4
0x5	0101	5
0x6	0110	6
0x7	0111	7
0x8	1000	8
0x9	1001	9
0xA	1010	10
0xB	1011	11
0xC	1100	12
0xD	1101	13
0xE	1110	14
0xF	1111	15

Power	Bytes	Kilobytes	Megabytes	Gigabytes
2^{1}	2			
2^{2}	4			
2^{3}	8			
2^{4}	16			
2^{5}	32			
2^{6}	64			
2^{7}	128			
2^{8}	256	0.25 KB		
2^{9}	512	0.5 KB		
2^{10}	1024	1 KB		
2^{11}	2048	2 KB		
2^{12}	4096	4 KB		
2^{13}	8192	8 KB		
2^{14}	16,384	16 KB		
2^{15}	32,768	32 KB		
2^{16}	65,536	64 KB		
2^{17}		128 KB		
2^{18}		256 KB	0.25 MB	
2^{19}		512 KB	0.5 MB	
2^{20}		1024 KB	1 MB	
2^{21}		2048 KB	2 MB	
2^{22}		4096 KB	4 MB	
2^{23}		8192 KB	8 MB	
2^{24}		16,384 KB	16 MB	
2^{25}		32,768 KB	32 MB	
2^{26}		65,536 KB	64 MB	
2^{27}			128 MB	
2^{28}			256 MB	0.25 GB
2^{29}			512 MB	0.5 GB
2^{30}			1024 MB	1 GB
2^{31}			2048 MB	2 GB
2^{32}			4096 MB	4 GB
2^{33}			8192 MB	8 GB
2^{34}			16,384 MB	16 GB
2^{35}			32,768 MB	32 GB
2^{36}			65,536 MB	64 GB