EECS 470 *Final Exam*

Fall 2013

Name: _____ unique name: _____

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

_____

Scores:

Page#	Points
2	**/21**
3	**/8**
4	**/12**
5	**/10**
6	**/12**
7	**/10**
8	**/12**
9	**/15**
Total	***/100***

NOTES:

- Open book and Open notes
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam.
- There are **10** pages, including this one.
- The last page is a duplicate—please be sure cross out the one you don't want graded.

1) Multiple choice/fill-in-the-blank. Pick the best answer.
[17 points, -2 per wrong/blank answer, min 0]

a) Adding ***architected / physical*** registers can provide the complier more flexibility in avoiding ***true / name / RAW / structural*** dependencies.

b) One thing that does **not** limit the hoisting of loads by the compiler is ***the number of architected registers / the number of physical registers / stores before the load / branches before the load***.

c) Loop unrolling will generally reduce the I-cache ***size / associativity / hit rate / bandwidth*** but usually increases the amount of ***ILP /instruction-stream locality / branches / loads*** in the program.

d) A ***gshare / bimodal / two-level / tournament*** predictor actually consists of two (or more) predictors, where one is usually global and the other local. The predictor that chooses which predictor to use only is updated when the two underling predictors ***predict taken / disagree / predict not taken / alias with each other***.

e) A ***no-write-allocate / write-through / write-back /write-allocate / skew*** data cache doesn't need dirty bits.

f) Victim caches are most effective when added to ***direct-mapped / skew / write-back / write-through*** caches.

g) Given a 32-KB two-way associative cache with 32-byte lines, the index into the cache will be _____ bits. If the same cache were direct-mapped, you'd need _____ index bits.

h) Robert Colwell was the chief architect for the ***Power PC / Itanium / P6 / Intel's 386***.

2) When using the MESI coherence protocol and bus protocol described in class, which of the following are true? Assume the cache is write-back and write-allocate. *LIST all correct answers*.
[**4 points, -2 per wrong/missing answer]**

a) If a given cache line is held in the "E" state and that cache's processor wishes to do a **write**, the transaction need not be sent on the bus.
b) If a given cache line is held in the "S" state and that cache's processor wishes to do a **read**, the transaction need not be sent on the bus.
c) If a given cache line is held in the "S" state and that cache's processor wishes to do a **write**, a BRIL (read and invalidate) transaction is placed on the bus
d) The only state which indicates that the data is "dirty" is the "M" state.
e) When a cache line held in the "E" state is evicted, a BWL transaction is placed on the bus.

_____ ← Place your answer here

3) Answer the following about the out-of-order algorithm we have called “P6”. **[8 points]**

a) When does an instruction write a value to the RoB? **[2]**

b) What structures could an instruction get its value from? **[3]**

c) When a mispredicted branch hits the head of the RoB what happens to the following structures? Your answer should be a few words or a sentence for each. **[3]**

- RSes
- RoB
- ARF

4) Short answer **[12 points]**

a) Branch alignment is a compiler optimization that converts strongly biased taken branches to strongly biased not-taken branches, by rearranging instruction layout in memory. How can this optimization be used to improve BTB performance? (Keep your answer to a few sentences.) **[4]**

b) When we speculatively wakeup a load and it turns out that there was an L1 miss and we've misspeculated, we need to replay all instructions in the load's "shadow". What does it mean for an instruction to be in the load's shadow? **[4]**

c) Provide the shortest possible reference stream where a direct-mapped cache will get a hit, while a 2-way associative cache will get a miss. Assume both caches are 1KB in size and have 16-byte lines. **[4]**

5) Page coloring (sometimes called “cache coloring”) can be described as requiring the low-order bits of the physical page number and the virtual page number to be the same. So for example, we might require that the two least significant bits of the virtual and physical page number not change (so virtual page 0x00 could map to physical page 0x04 but not physical page 0x03 as the two least significant digits are different). In this case it would be said there are four page “colors”, (00, 01, 10, and 11). **[10 points]**

   For the rest of this problem, assume we are studying a computer with the following characteristics:

   - 4KB pages
   - 16KB physically addressed instruction cache that is direct-mapped
   - A fully associative 32-entry TLB

   a) What effect would you expect page coloring to have on your page fault rate? Explain. **[4]**

   b) Say you that the average hit-rate of the SPEC benchmark suite improved from 95% with no page coloring to 95.4% with two page colors, and that with four or more page colors your hit rate is around 95.6%. What would explain these results? **[4]**

   c) If the cache were instead virtually addressed and physically tagged, how would page coloring help? **[4]**

6) Consider processor which, when running a given application performs 1 billion loads and 500 million stores per second. **[12 points]**

Assume the following is true:

- The processor's multi-level cache system gets a 95% hit rate on both loads and stores.
- Cache lines are 32-bytes in size
- There is no prefetching and the instruction cache never misses.
- 30% of all lines evicted from the last level of the cache are dirty.
- The cache is write-back and write-allocate.
- There are no coherence misses.
- All loads and stores are to 4-byte values.

a) What is the read bandwidth (bytes/second) on the bus? Show your work. **[4]**

b) What is the write bandwidth (bytes/second) on the bus? Show your work. **[4]**

c) What would the write bandwidth (bytes/second) be if the cache were write-through and no-write-allocate? Assume the bus can support 4-byte writes and that the hit rates drop to 90% on writes and 93% on reads. **[4]**

7) A given code segment has one load, one store, and 200 ALU operations. When the segment is run on a baseline superscalar out-of-order processor the load hits in the L1 cache and it takes 210 cycles to execute. Adding a "good" load-store queue to the processor (loads can pass stores, stores can forward to loads) causes the code segment to run in 120 cycles. If the load and store are to different addresses, what could account for this rather large speedup? Sketch what the code segment must look like. **[5 points]**

8) Consider the pipeline you were to implement for your third programming assignment, but assume that the structural hazard has been removed. A given program consists of 25% loads, 5% stores, 10% branches and 60% ALU operations. If 40% of the branches are not-taken and 40% of all instructions are dependent on the instruction in front of them, what is the expected CPI of the processor on this program? Show you work. **[5 points]**

9) Consider a case of having 3 processors using a snoopy MESI protocol where the memory can snarf data. All three have a 2 line direct-mapped cache with each line consisting of 16 bytes. The caches begin with all lines marked as invalid. Fill in the following table indicating

- If the processor gets a hit or a miss in its cache
- If a HIT or HITM (or neither) occurs on the bus during snoop.
- What bus transaction(s) (if any) the processor performs (BRL, BWL, BRIL, BIL)

In the event more than one bus transaction occurs due to a given memory read/write indicate the Hit/Miss and HIT/HITM <u>for the primary transaction only</u>. For the misses, indicate what type of "4C's" miss it is. Finally, indicate the state of the caches after all of these memory operations have completed. The operations occur in the order shown. **[12 points, -1 per wrong or blank, minimum of zero]**

Processor	**Address**	**Read/ Write**	**Bus transaction(s)**	**Hit/Miss**	**HIT/ HITM**	**4C's miss type.**
1	0x100	Write				
1	0x200	Read				
2	0x200	Read				
1	0x200	Write				
1	0x110	Write				
1	0x110	Read				
3	0x110	Write				
2	0x100	Read				
3	0x100	Read				
1	0x100	Write				

	<u>Proc 1</u>	
	Address	**State**
Set 0		
Set 1		

	<u>Proc 2</u>	
	Address	**State**
Set 0		
Set 1		

	<u>Proc 3</u>	
	Address	**State**
Set 0		
Set 1		

10) Consider the following tables that represent the state of a processor that implements what we have called the P6 algorithm. **[15 points]**

RAT	
Arch Reg. #	**ROB# (-- if in ARF)**
0	0
1	--
2	--
3	--
4	--
5	--

ROB					
	Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**
← Head	0	40	N	0	--
← Tail	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				

RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Add	Y	5	Y	2	0
1						
2						
3						
4						

ARF	**Reg#**	**0**	**1**	**2**	**3**	**4**	**5**
	Value	1	5	4	3	2	1

Place the following instructions into the processor:

```
R3=R3+R1      // A
R1=R1*R4      // B
R5=R1+R4      // C
R1=R3*R3      // D
R2=R5+R4      // E
```

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C, D, and E have progressed as far along as possible. *Be sure to update the head and tail of the ROB*. When arbitrary decisions need to be made, you are to just make them. Assume instructions are 4 bytes each. ***Clearly cross out any data that is no longer valid.***

(A second copy is available on the following page, ***please cross out the one you don't want graded!***)

Consider the following tables that represent the state of a processor that implements what we have called the P6 algorithm. **(This is a copy of the previous page, cross out the one you don't want graded!)**

RAT	
Arch Reg. #	**ROB# (-- if in ARF)**
0	0
1	--
2	--
3	--
4	--
5	--

ROB					
	Buffer Number	**PC**	**Done with EX?**	**Dest. Arch Reg #**	**Value**
← Head	0	40	N	0	--
← Tail	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				

RS						
RS#	**Op type**	**Op1 ready?**	**Op1 RoB/value**	**Op2 ready?**	**Op2 RoB/value**	**Dest ROB**
0	Add	Y	5	Y	2	0
1						
2						
3						
4						

ARF							
	Reg#	**0**	**1**	**2**	**3**	**4**	**5**
	Value	1	5	4	3	2	1

Place the following instructions into the processor:

```
R3=R3+R1      // A
R1=R1*R4      // B
R5=R1+R4      // C
R1=R3*R3      // D
R2=R5+R4      // E
```

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C, D, and E have progressed as far along as possible. *Be sure to update the head and tail of the ROB.* When arbitrary decisions need to be made, you are to just make them. Assume instructions are 4 bytes each. ***Clearly cross out any data that is no longer valid.***

(This is a copy of the previous page. ***Please cross out the one you don't want graded!***)