Last (family) name: _____

First (given) name: _____

Student I.D. #: _____

Department of Electrical and Computer Engineering
University of Wisconsin - Madison

ECE/CS 752 Advanced Computer Architecture I

Midterm Exam 2

Distributed Friday, May 5, 2008 / Due by 5pmon Monday, May 12, 2008
Please place completed exam in Prof. Lipasti's mailbox on the first floor of EH4613.

Instructions:

1. This exam is open books, open notes, and open all handouts (including previous homeworks and exams, project descriptions, textbook, and readings). However, **it must be your own work--do not discuss any aspects of the exam problems with other students until after** all exams have been turned in
2. **You must show your complete work**. Points will be awarded only based on what appears in your answers.
3. **If you prefer**, you can type your answers using a word processor, rather than hand-writing them on this exam.
4. Failure to follow instructions may result in forfeiture of your exam and will be handled according to UWS 14 Academic misconduct procedures.

Problem	*Type*	*Points*	*Score*
1	Discussion Questions	20	
2	Power Modeling	10	
3	Modern Cache Coherence Protocols	15	
4	Virtually-indexed Coherent Caches	15	
5	Out-of-order Scheduling Replay	20	
Total		80	

1. Discussion Questions [20 pts]

a. [3 pts] Fine-grained multithreading, coarse-grained multithreading, and simultaneous multithreading are all used to interleave the execution of multiple threads on a single processor. Explain when and how instructions from multiple threads are introduced into and execute within the processor under each scheme.

Not provided (in the book).

b. [3 pts] Modern processors separate wakeup and select and data forwarding (or data capture) into separate pipeline stages to improve cycle time. In order to achieve high performance, this requires speculative scheduling. Explain why, and identify two challenging requirements that the recovery mechanism for a speculative scheduler must satisfy.

Cycle time reasons. Predict all loads hit the cache (fixed latency). Must recover on cache misses. Recovery mechanism should be faster than wakeup propagation, otherwise it will never catchup. Also it must reach the transitive closure of dependent ops.

c. [2 pts] In a multiscalar processor, what major functions did the sequencer perform?

Not provided (in the paper).

d. [2 pts] Describe the purpose and operation of the Multiscalar ARB.

Not provided (in the paper).

e. [3 pts] Compare and contrast the key differences between FPM, EDO, SDRAM, and RDRAM main memory technologies.

Not provided (in the paper)

f. [4 pts] Sequential consistency requires all memory references to appear to other processors in the system as if they were executed in original program order. Describe the mechanism used by modern out-of-order processors to enable out-of-order execution of load instructions while still maintaining the appearance of in-order execution. Also explain why this mechanism is sufficient for guaranteeing sequential consistency.

Loads are issued out of order but tracked in the load queue. Remote writes (bus upgrades or bus writes) show up as invalidates; these are checked against the load queue. Any conflict indicates that load was prematurely issued. A local replay of the load will force a miss and reorder that load with respect to the remote processor. This is sufficient since all stores are performed at commit, in order, so the fact that remote store has reached commit indicates it should be ordered first. All references that do not conflict need not be ordered, since the fact that they are not ordered is not "visible" to the programmer or user of the system.

g. [3 pts] Explain how redundant execution can be used to improve reliability. Distinguish both temporal and spatial redundancy and identify the sets of faults that each can be applied to.

Execution instructions twice and comparing results can detect errors. Executing three times allows one to vote and identify the error.
Temporal redundancy uses the same hardware twice. This will catch transient (soft) faults.
Spatial redundancy uses dedicated hardware for redundancy. This will catch transient as well as permanent faults.

2. Power Modeling [10 pts]

Assume a processor with the following energy consumption characteristics for the event types listed.

Event	Energy per instruction (nJ)
Fetch, Decode, Dispatch	**2.0**
Issue	**1.3**
Reg file read (per instruction)	**1.1**
Int/branch execute	**0.4**
Load execute	**0.9**
Store execute	**0.6**
Reg file write	**0.65**
Commit	**0.3**

Assume an instruction mix of 25% loads, 15% stores, 20% branches and jumps, and 40% integer ALU instructions. Assume that all loads and integer ALU ops write a register, and that this is a physical-register file machine where registers are written speculatively right after an instruction executes. Finally, assume the processor runs at 2GHz.

a. [4 pts] Assume that on average the processor commits 1.4 IPC, executes 1.8 IPC, issues 2.0 IPC, and fetches/decodes/dispatches 3.0 IPC. What is the power consumption (in watts) of this processor running this workload?

1.4 x 0.3 + 1.8 x [1.1 + 0.4x0.6 + 0.9x0.25 + 0.6x0.15 + 0.65x(0.25+0.4)] + 2.0x1.3 + 3.0x2.0 = 12.74nJ/cyc x 2bcyc/sec = 25.48W

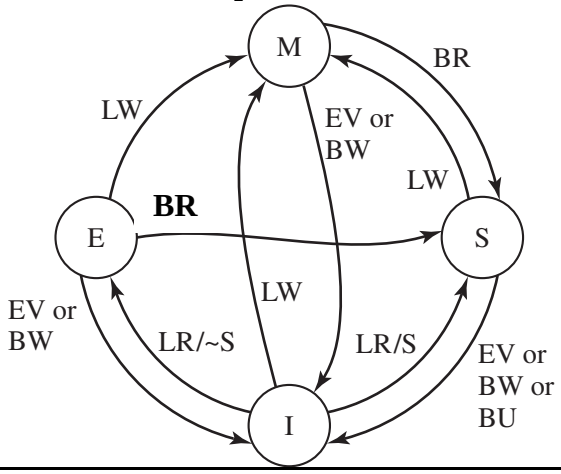
b. [4 pts] You decide to implement a branch confidence estimator that slows down fetch whenever you have predicted a branch with low confidence. Assume that the power consumption of the confidence estimator is negligible. This approach reduces the commit rate to 1.35 IPC, execution rate to 1.7 IPC, issue rate to 1.8 IPC, and fetch/decode/dispatch rate to 2.75 IPC. Now what is the power consumption?

1.35 x 0.3 + 1.7 x [1.1 + 0.4x0.6 + 0.9x0.25 + 0.6x0.15 + 0.65x(0.25+0.4)] + 1.8x1.3 + 2.75x2.0 = 11.75nJ/cyc x 2bcyc/sec = 23.5W

c. [2 pts] Given your answer for (b), does your confidence scheme pass or fail the 3:1 rule of thumb discussed in lecture? Why or why not?

Power savings = 7.7%, perf reduction is 3.57%, so it fails the 3:1 test.

3. Modern Cache Coherence Protocols [15 pts]



Current State s	Event and Local Coherence Controller Responses and Actions (s' refers to next state)					
	Local Read (LR)	**Local Write (LW)**	**Local Eviction (EV)**	**Bus Read (BR)**	**Bus Write (BW)**	**Bus Upgrade (BU)**
Invalid (I)	Issue bus read if no sharers then s' = E else s' = S	Issue bus write s' = M	s' = I	Do nothing	Do nothing	Do nothing
Shared (S)	Do nothing	Issue bus upgrade s' = M	s' = I	Respond shared	s' = I	s' = I
Exclusive (E)	Do nothing	s' = M	s' = I	Respond shared s' = S	s' = I	Error
Modified (M)	Do nothing	Do nothing	Write data back; s' = I	Respond dirty; Write data back; s' = S	Respond dirty; Write data back; s' = I	Error

Many modern systems use a MOESI cache coherence protocol, where the semantics of the additional O state are that the line is shared-dirty: i.e., multiple copies may exist, but the other copies are in S state, and the cache that has the line in O state is responsible for writing the line back if it is evicted.

a. [2 pts] Explain what benefit accrues from the addition of O state to the MESI protocol.
Can delay writing back the dirty line until it is evicted from the cache. This can also streamline directly providing the dirty line to the requesting processor, since there is no (slow) writeback involved.

[11 pts] **Redraw** the state diagram and table above (from the textbook) to accommodate the O state.

(add new O state which is entered from M on a BR, and which transitions to I on LE,BW).

Current State s	**Event and Local Coherence Controller Responses and Actions (s' refers to next state)**					
	Local Read (LR)	**Local Write (LW)**	**Local Eviction (EV)**	**Bus Read (BR)**	**Bus Write (BW)**	**Bus Upgrade (BU)**
Invalid (I)	Issue bus read if !shared & !dirty then s' = E else s' = S	Issue bus write s' = M	s' = I	Do nothing	Do nothing	Do nothing
Shared (S)	Do nothing	Issue bus upgrade s' = M	s' = I	Respond shared	s' = I	s' = I
Exclusive (E)	Do nothing	s' = M	s' = I	Respond shared s' = S	s' = I	Error
Owned (O)	Do nothing	Issue bus upgrade s' = M	Write data back; s' = I	Respond dirty; Supply data	Respond dirty; Supply data s'= I	s'= I
Modified (M)	Do nothing	Do nothing	Write data back; s' = I	Respond dirty; Supply data s'= O	Respond dirty; Supply data s' = I	Error

The protocol can also be implemented so that the requestor goes to O, and the supplier downgrades from M to S. This increases the likelihood that the O block will not get evicted, since it will be in the MRU position at the receiver.

b. [2 pts] The base protocol has two snoop responses (shared and dirty). Does the addition of the 'O' state require any new responses? If so, what are they?

It can be implemented without any extra responses.

4. Virtually-indexed Coherent Caches [15 pts]

Assume a processor similar to the Hewlett-Packard PA-8500, with only a single level of data cache. Assume the cache is virtually-indexed but physically tagged, is 4-way associative with 128B lines, and is 512 KB in size. This processor supports multiprocessor coherence, but bus commands specify physical addresses rather than virtual addresses. In order to snoop coherence messages from the bus, a reverse-address translation table (RAT) is used to store physical-to-virtual address mappings stored in the cache.

a. [2 pts] Assuming a fully-associative RAT and 4KB pages, how many entries must it contain so that it can map the entire data cache? Would you consider this a reasonable and buildable structure? Why or why not?

   There are 4K blocks in the L2 (512KB/128B = 4K); each could have a unique physical page number in the tag. Hence, a total of 4K entries are needed in the RAT to guarantee that all cache blocks can be mapped. Of course, a 4K-entry fully-associative RAT is quite expensive and possibly too slow since it needs to be accessed for each snoop.

b. [3 pts] Given the assumptions above, describe a reasonable set-associative organization for the RAT that is still able to map the entire data cache. Would you consider this a reasonable and buildable structure? Why or why not?

   There are 12 untranslated bits in each address, and only 7 of these are used for block offset. Hence, 5 bits are left to use as part of the index into the L2. Hence, the RAT could also use these 5 bits to index into the RAT, giving 2^5 = 32 sets. Hence, you would have 4k/32 = 128-way set associative RAT, which is much more feasible.

c. [2 pts] Identify another use for the RAT in this design that is not related to multiprocessor coherence. Describe why it is necessary and describe when and how the RAT is used for this purpose.

   A virtually-indexed cache cannot allow aliased blocks to reside at different locations in the cache. Whenever a block is inserted in the cache, the RAT must be consulted to make sure there is no pre-existing aliased block; if there is one, it must be evicted.

d. [4 pts] Explain the implications of a RAT that is not able to map all possible entries in the data cache. Describe the sequence of events that must occur whenever a RAT entry is evicted due to replacement.

This limits the freedom of the cache, since the RAT must maintain inclusion over the cache. In this case, if one RAT entry is evicted to make room for a new block being inserted into the cache, all cache entries covered by the old RAT entry must also be evicted from the cache. This can be quite expensive, since a single 4K-page RAT entry can map up to 4K/128 = 32 blocks, which must be individually checked.

e. [4 pts] The PA-8500 was derived from the PA-8000, which had no on-chip caches (at the time, HP's proprietary process technology did not effectively support large amounts of on-chip SRAM; the PA-8500 was fabbed by Intel, so it did not have this limitation). Architecturally, the PA-8000 was similar in that it had a single level of cache, implemented using off-chip SRAM. Why do you think HP designers continued the unconventional single-level approach, even when they had access to on-chip SRAM for the PA-8500? Do you think this was the right decision? Justify your answer.

Adding 2nd level is a fairly significant ripup of the existing design, particularly in a cache-coherent multiprocessor (protocols for multilevel caches can get quite involved). Given that HP was moving away from PA-RISC and wanted to minimize its investment and risk, this was probably the right business decision, though arguably not the right technical decision.

5. Out-of-order Scheduling Replay [20 pts]

Given the following code, assume that array A begins at address 0x0, and array B is placed consecutively after A in memory, and that each "double" array entry consumes 8 bytes (64 bits). The data cache is initially empty.

```
double A[1024], B[1024];
for(int i=0; i<1000; i++) {
    A[i] = 35.0 * B[i];
}
```

The corresponding machine/assembly code for the loop body looks like this:

```
loop:  addi   r1,r1,1       // increment i
       cmp    r1,#1000      // compare to 1000
       bge    done          // exit loop if done
       load   f1,0(r2)      // r2 points to B[i]
       fmul   f2,f1,f4      // f4 preloaded with 35.0
       store  f2,0(r4)      // r4 points to A[i]
       addi   r2,r2,8
       addi   r4,r4,8
       jump   loop
done:  …
```

You are to analyze the behavior of a 2-wide speculative scheduler in terms of the number of unnecessarily executed instructions under varying scheduling recovery models, as described in the Kim reading ("Understanding Scheduling Replay Schemes").

Assume the following:

- The same 4-stage pipeline from schedule to execute shown in Fig. 5 of [Kim]. Cache misses are detected at the end of this pipeline, so there are up to 4x2=8 instructions in flight before the scheduler finds out that a cache miss occurred.
- Load instructions have 1-cycle latency (that is, they execute in the 4th stage from issue)
- Integer, branch, and store-address have a 1-cycle latency
- The floating-point multiply (fmul) instructions have 3-cycle latency that extends beyond the 1 EX cycle shown in the paper
- All execution units are fully pipelined
- There are two of each type of functional unit, so any instruction pair can issue in each cycle
- Stores are split into two micro-ops: a store-address micro-op, which issues speculatively as soon as its address-source register is ready, and the store-data microp, which never issues speculatively, but executes at commit
- Loads issue speculatively, even if prior store addresses have not yet issued
- The select logic will always issue the oldest of the ready instructions
- Perfect branch prediction and an infinitely large issue queue and reorder buffer, so there are always more instructions to issue (from future iterations of the loop)
- 128-byte cache lines, leading to a 6.25% steady-state miss rate for loads from B[] (1 load in 16 will miss).

a. [5 pts] For steady-state execution, determine the fraction of instructions that have to be squashed and replayed under non-selective replay (Sec. 3.3 in the paper).

jmp	addi
addi	addi
fmul	sta
bge	load

These ops are in flight when the load misses. All eight are squashed and replay. In steady state, this is 8/9 instructions per loop, with one miss per 16 loops. This works out to 8/9 x 1/16 = 1/18, or 5.5%

b. [5 pts] For steady-state execution, determine the fraction of instructions that have to be squashed and replayed under position-based selective replay (Sec. 3.4.3)

Only the fmul must be squashed, so 1/9 per loop, with one miss per 16 loops, which works out to 1/9 x 1/16 = 1/144 = 0.69%

c. [5 pts] For steady-state execution, determine the fraction of instructions that have to be squashed and replayed under ID-based selective replay (Sec. 3.4.1).

Only the fmul must be squashed, so 1/9 per loop, with one miss per 16 loops, which works out to 1/9 x 1/16 = 1/144 = 0.69%

d. [3 pts] ID-based selective replay is difficult to scale to large instruction windows, since each in-flight load needs its own "name". Describe how the token-based technique in Sec. 4 addresses this problem.

Only loads that are likely to miss get assigned a token. Since most misses are caused by a small fraction of loads (in typical programs), we need fewer tokens in flight.

e. [2 pts] Do you believe the token-based technique (Sec. 4) will be effective for the loop studied in this problem? Why or why not?

No, since all misses are caused by the same static load, there will be no reduction in the number of tokens needed.