**EECS 470 *Midterm Exam***

Fall 2009

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ unique name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sign the honor code:

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Scores:

|  |  |
| --- | --- |
| # | Points |
| 1 | **/20** |
| 2&3 | **/10** |
| 4 | **/10** |
| 5 | **/7** |
| 6 | **/10** |
| 7 | **/12** |
| 8 | **/12** |
| 9 | **/4** |
| 10 | **/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 **11** pages, including this one.
1. Multiple choice/fill-in-the-blank. Pick the best answer.
**[20 points, -2 per wrong/blank answer, min 0]**
	1. A given processor uses an average of X Watts and averages 100 MIPS on a given application that has a near 100% cache hit rate. If voltage scaling were used to increase

	the performance to 110 MIPS, we’d expect it to be using about \_\_\_\_\_\_\_\_\_\_ Watts. If we wanted to use voltage scaling to reduce the power to .8X Watts, we’d expect to get a performance of

	around \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ MIPS.
	2. The compiler often has difficulty moving a load above a store in program order because ***there might be a true dependence / there might be a name dependence / it could cause an exception that wouldn’t otherwise occur.***
	3. In a directory based system, a load request must go to the home node unless the cache has it in the ***S or E or I / S or E or M / E or M / M*** state.
	4. In IA-64, a *speculative* load can be used to hoist a load above a ***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***. If that load causes a fault, the fault is ***deferred / handled immediately/ ignored***.
	5. A processor using Tomasulo’s algorithm avoids all *but* ***RAW / WAW / WAR / name***  dependencies by renaming the architected registers.
	6. On a snoopy bus using the MESI protocol a cache line will end up in the “E” state if the processor performs a ***load / store***, misses in the cache, and sees ***HIT / HITM / either HIT or HITM / neither HIT nor HITM*** asserted when it places the memory transaction on the bus.
	7. In T2, a processor with 2 RSes, 16 RoB entries, and 8 architected registers would have a RAT
	whose size in bits is ***10 / 16 / 20 / 32 / 40 / 96 / 128*** ignoring state bits (like valid etc.). If that same design were using the algorithm we’ve called T3, the RAT would have ***10 / 16 / 20 / 32 / 40 / 96 / 128*** bits (again ignoring state bits).
	8. In the algorithm we are calling T3, the RAT is updated by a *committing* instruction if ***the instruction had a destination register / only if the current RAT value for the associated architected register points to the PRF entry of the committing instruction / only if the instruction is a mispredicted branch***.
	9. A return address stack helps to predict the ***direction / address / address and direction*** of a ***function call / function return / function call and function return***.
2. Consider the following access pattern: A, B, C, A. Assume that A, B, and C are memory addresses each of which are in a different block of memory. Further, assume the addresses A, B, and C are generated in a random way (each address having an equal probability of occurring) and that a "true" LRU replacement algorithm is used. What is the probability that the second instance of "A" will be a hit for the following caches? *Show your work.***[6 points]**
	1. The cache is a 2-way associative cache with 8 lines. **[3]**

* 1. The cache is a direct-mapped cache with 8 lines. **[3]**
1. *Spartan Computers* just released a high-end microprocessor with 96K 6-way set-associative L1 I and D caches. Assuming that the processor uses 8K pages, is it possible to do parallel address translation when accessing the cache (i.e., access TLB and cache at the same time)? Why or why not? **[4 points]**
2. Say you are designing an L1 cache and are considering three different designs:
	1. 1024-line, 32-byte block, direct-mapped cache
	2. 512-line, 64-byte block, two-way associative cache.
	3. 512-line, 64-byte block, fully associative cache, pseudo LRU.

In simulation you find that “a” will get a 90% hit rate, “b” will get a 93% hit rate and “c” will get a 96% hit rate.

Now answer the following questions. Your answer should be of the form “A”, “B”, “C”, any combination of the three, or “all the same”. ***Provide a brief explanation for each answer.* [10 points, 2 each]**

* + 1. Which of these caches would you expect to have the highest access time on a hit?

* + 1. Which will have the largest memory bandwidth requirements?

* + 1. Which requires the largest tag-store?

* + 1. Which requires the largest data-store?

* + 1. Which design would cause your L2 cache to get the worst hit rate (measured as hits per L2 request)?

1. Write a Verilog module that implements the 4 to 1 mux shown below. *Minor* syntax errors will be ignored. **[7 points]**

A

B

C

D

0

1

2

3

S[1:0]

OUT

1. Say we have the following code segment in pseudo-assembly:

If(R1==4) goto SKIP

R5=R5+R1

If(R1==5) goto SKIP

R5=R5+1

SKIP:

And say we’ve got the following instruction available to us: **CMOV(Rx,Ry,Rz)** Where the instruction sets Ry=Rz if and only if Rx!=0. **[10 points]**

* 1. Rewrite the above code to remove all branches. You may only use R6 and R7 as “scratch” registers: all other registers are holding live values. For comparisons you may use instructions such as R1=(R5==R6). You may not read or write memory. **[5]**

* 1. Under what circumstances would you expect removing the above branches and replacing them with CMOVs would improve performance? **[5]**
1. *Intel* has just released a new 4-core processor that uses a shared snoopy bus. Each core has a 2-way associative, 64KB cache with 32-byte cache lines and keeps data in the M, S or I states. There is a shared, on-chip, L2. On a given benchmark:
* One third of the processor’s memory transactions to the cache are stores (the rest being loads).
* 90% of loads and 90% of stores don’t generate a bus transaction.
* 25% of all evictions are of dirty data.
* Each core sends 85 million transactions on the bus per second. 10 million of those are BILs.

Answer the following questions. You are to assume that the cores aren’t bandwidth limited. **[12 points]**

1. How many transactions per second would you expect of each of the following types of transactions? Show your work **[6]**

|  |  |
| --- | --- |
| **BRILs** | Million/sec |
| **BRLs** | Million/sec |
| **BWLs** | Million/sec |

1. If we were to add an “E” state to the processor, the rate of what type (or types) of transactions would be impacted? How would they be impacted (go up or go down)? Explain your answer. **[6]**
2. 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 for the primary transaction only*. Finally, indicate the state of the processor 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** |
| 1 | 0x100 | Read |  |  |  |
| 1 | 0x130 | Read |  |  |  |
| 2 | 0x100 | Read |  |  |  |
| 1 | 0x100 | Write |  |  |  |
| 1 | 0x130 | Write |  |  |  |
| 1 | 0x200 | Read |  |  |  |
| 3 | 0x200 | Write |  |  |  |
| 2 | 0x210 | Read |  |  |  |
| 3 | 0x100 | Read |  |  |  |
| 1 | 0x210 | Write |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Proc 1** |  |  | **Proc 2** |  |  | **Proc 3** |
|  | **Address** | **State**  |  |  | **Address** | **State**  |  |  | **Address** | **State**  |
| **Set 0** |  |  |  | **Set 0** |  |  |  | **Set 0** |  |  |
| **Set 1** |  |  |  | **Set 1** |  |  |  | **Set 1** |  |  |

1. According to Robert Colwell…
	1. Intel made a mistake by doing what with the first working version of the IA-64 processor? [2]

* 1. Why was it a mistake? [2]
1. Consider the following tables that represent the state of a processor that implements what we have called Tomasulo’s second algorithm:

|  |  |  |
| --- | --- | --- |
| **RAT** |  | **ROB** |
| **Arch Reg. #** | **ROB#****(-- if in ARF)** |  | **Buffer****Number** | **PC** | **Done with EX?** | **Dest. Arch Reg #** | **Value**  |
| **0** | -- |  | 0 | 20 | N | 1 | -- |
| **1** | 2 |  | 1 | 24 | N | 4 | -- |
| **2** | 4 |  | 2 | 28 | Y | 1 | 6 |
| **3** | -- |  | 3 | 32 | Y | -- | -- |
| **4** | 1 |  | 4 | 36 | Y | 2 | 8 |
| **5** | -- |  | 5 |  |  |  |  |
|  |  |  | 6 |  |  |  |  |
|  |  |  | 7 |  |  |  |  |
|  |  |  | 8 |  |  |  |  |

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ARF** | **Reg#** | **0** | **1** | **2** | **3** | **4** | **5** |
| **Value** | 6 | 5 | 4 | 3 | 2 | 1 |

The instruction at PC 32 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 200, where the following code resides:

R3=R3+R1 // A

R1=R1+R4 // B

R5=R5+R4 // C

R1=R1\*R3 // D

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C and D have progressed as far along as possible. *Be sure to label the head and tail of the ROB.* Please place instruction A in slot 5 of the ROB. When other arbitrary decisions need to be made, you are to just make them. **[15]**

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

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

|  |  |  |
| --- | --- | --- |
| **RAT** |  | **ROB** |
| **Arch Reg. #** | **ROB#****(-- if in ARF)** |  | **Buffer****Number** | **PC** | **Done with EX?** | **Dest. Arch Reg #** | **Value**  |
| **0** | -- |  | 0 | 20 | N | 1 | -- |
| **1** | 2 |  | 1 | 24 | N | 4 | -- |
| **2** | 4 |  | 2 | 28 | Y | 1 | 6 |
| **3** | -- |  | 3 | 32 | Y | -- | -- |
| **4** | 1 |  | 4 | 36 | Y | 2 | 8 |
| **5** | -- |  | 5 |  |  |  |  |
|  |  |  | 6 |  |  |  |  |
|  |  |  | 7 |  |  |  |  |
|  |  |  | 8 |  |  |  |  |

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ARF** | **Reg#** | **0** | **1** | **2** | **3** | **4** | **5** |
| **Value** | 6 | 5 | 4 | 3 | 2 | 1 |

The instruction at PC 32 is a branch that has been predicted not-taken, but it is actually taken. The destination of the branch is PC 200, where the following code resides:

R3=R3+R1 // A

R1=R1+R4 // B

R5=R5+R4 // C

R1=R1\*R3 // D

Show the state of the above tables if instruction A has retired, inst B has not finished executing, while C and D have progressed as far along as possible. *Be sure to label the head and tail of the ROB.* Please place instruction A in slot 5 of the ROB. When other arbitrary decisions need to be made, you are to just make them. **[15]**