**EECS 470 *Final Exam***

Winter 2011

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 | **/9** |
| 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. The compiler often has difficulty moving a load above a branch 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.***
   2. In the MESI bus protocol the ***M to I / E to I / S to M*** and the ***M to S / E to M / S to E*** transitions can be silent (that is with no information related to the transition appearing on the bus).
   3. A return address stack is used to predict the ***direction / address / address and direction*** of a ***function call / function return / function call and function return***.
   4. Given a 32-bit address space, a 16 KB cache with 32-byte lines that is four-way set-associative  
        
      will have \_\_\_7\_\_\_\_ index bits and the total size of the tag store (not including things like valid   
        
      bits, MESI bits, etc.) will be \_\_**29 \* 20 bits/(8 bits/byte) = 1280**\_\_ ***bytes***.
   5. The IA-64 has support for software pipelining. One feature it **doesn’t** have is:

* ***a special purpose register for the loop counter***
* ***a mechanism for removing the extra code associated with prologues and epilogues***
* ***dynamic register renaming to move high-latency instructions***.  
  1. A standard 2000mAh double-A battery has about ***0.3Wh / 3.0 Wh / 30 Wh / 0.3kWh / 3kWh*** of energy.
  2. In class Dr. Austin discussed how failure rates come in three stages: infant mortality, breakdown period and grace period. Together those three terms are used to model the fact that:
     + ***devices tend to fail either early or late in their lifetime***
     + ***devices tend to fail only after a long period of time***
     + ***soft and hard error rates tend to diverge over time***.
  3. Loop unrolling allows the compiler to ***reorder true dependencies / reduce the number of instructions in the executable / have a bigger window to reorder instructions statically***.   
       
     However it also ***makes things considerably harder for a dynamic scheduler / increases the number of instruction in the executable / shrinks the window for reordering instructions statically.***
  4. When using dynamic voltage scaling to reduce the power draw by 50% will result in the performance being reduced to about ***95 / 85 / 80 / 65 / 50 / 35*** % of what it was.
  5. In the algorithm we are calling T3, if the RS has 8 entries, the ROB has 32 entries and the ARF has 16 entries, then you’d not expect to have more than ***8/ 16 / 32 / 48 / 56 / 64*** entries in the PRF.

1. Early branch resolution **[7 points]**
   1. If we are implementing early branch resolution using Branch Register Alias Tables (BRATs), at what point will we need to allocate a BRAT? At what point can we free it (assuming there are no mispredictions in front of it)? **[3]  
        
      Allocate BRAT when we send instructions to the RS and ROB.   
      Free BRAT once branch has finished execution (or been cancelled by another mispredicted branch…).**
   2. In order to recover the PRF’s free list, we typically keep a free list associated with each BRAT. If we wish to be able to simply copy that free list over to the main free list on a misprediction we need to do a number of things. *Circle* ***all*** *the things that need to happen*. You may assume this processor is not superscalar. **[4, no partial credit]**
      * Copy the main free list to the BRAT free list ***after*** updating the free list to reflect any destination register(s) the branch might have.
      * Copy the main free list to the BRAT free list ***before*** updating the free list to reflect any destination register(s) the branch might have.
      * Update the BRAT free list as new instructions ***issue*** to the RS and RoB.
      * Update the BRAT free list as instructions ***commit.***
2. Consider a purely in-order processor with a cache connected to a main memory with a bus (which is 32 bits wide). A read access by the processor that hits in the cache takes 1 cycle. On a miss, the entire block must be fetched from main memory over the memory bus. A bus transaction consists of one address cycle to send an address (32 bits) to the memory, four cycles of idle-time for main memory access, and one cycle to transfer each word (32 bits) in the block to the cache. (Assume that the processor continues execution only after the last word of the block has arrived.) The following table gives the average cache miss rates of a 1Mbyte direct-mapped cache for various block sizes.[[1]](#footnote-1) Show your work. **[9 points]**

|  |  |
| --- | --- |
| **Block size (B) in words** | **Miss ratio (m) %** |
| 1 | 4.5 |
| 4 | 2.4 |
| 8 | 1.6 |
| 16 | 1.0 |

* 1. What block size yields the best average memory access time? **[5]**  
       
     tmem = thit + pmiss\*tmiss, tmiss = 1 + 4 + B

B(1) = 1 + 0.045\*(1 + 4 + 1) = 1.27 cycles

B(4) = 1 + 0.024\*(1 + 4 + 4) = 1.216 cycles

B(8) = 1 + 0.016\*(1 + 4 + 8) = 1.208 cycles (best average memory access time)

B(16) = 1 + 0.010\*(1 + 4 + 16) = 1.21 cycles

* 1. Assuming a 32-bit address space, what will be the size of the tag store (in bits) for the cache with a block size of 8? For a block size of 32? **[4]**  
       
     B(8) = 220/25 = 215 cache blocks 12 bits per block 393,216 bits

B(32)= 98,304

1. Short-answer cache questions **[9 points]**
   1. 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 if the cache is a 2-way associative cache and has 32 lines?**[4]**

For A to miss a second time, B, and C must both map to the A’s set.

phit = 1 – pmiss **=** 1 – (1/16 \* 1/16) = 0.99609375

* 1. What is the main ***advantage*** of a virtually addressed cache over a physically addressed cache? **[2]**The cache can be immediately accessed with a virtual address, thus the latency of address translation is not exposed on a cache access.
  2. What is the primary ***disadvantage*** of a virtually-indexed, physically tagged cache? Briefly explain the root cause is of that problem. **[3]**

Shared virtually pages that map to the same physical page MUST have the same virtual cache index bits, otherwise, memory sharing will be broken.

1. Write a Verilog module that implements a 4 to 2 encoder. *Minor* syntax errors will be ignored.   
   **[7 points]**
2. For multiprocessor systems we often find that the higher-level caches of a processor are required to be *inclusive* of the lower-level ones. Answer the following questions about cache inclusion.   
   **[13 points]**
   1. Why is cache inclusion helpful to multi-processor systems? **[3]**It reduces the bandwidth requirements on the L1 cache for multiprocessor snooping, since the inclusion property will ensure that anything seen in the L2 will be a superset of what is contained in the L1 cache.
   2. Cache inclusion can be tricky. Consider a cache structure where the L1 cache is 128KB in size and two-way set-associative while the L2 cache is 1MB in size and four-way set-associative. Both have 32-byte cache lines, use true-LRU replacement, and place the data in their cache on a miss (so an L2 miss results in the data being placed in both the L1 and L2). Find the shortest sequence of load addresses (with addresses in 8-digit hex) that would cause the L1 cache to have a piece of data but the L2 cache to not have that data. Your answer must start with the address 0x00000000. **[5]**
   3. Given the above, *briefly* describe a reasonable algorithm for maintaining cache inclusion. What is the negative impact of your algorithm? **[5]**

On L1 miss, bring data into L1 and L2 caches.

On L1 replacement, fetch line into L2 cache (if not present).

On L2 replacement, evict replaced address from L1 cache.

1. Consider a 4-core processor that uses a shared snoopy bus. Each core has a private 2-way associative 64-KB cache with 32-byte cache lines and keeps the data in one of the MESI states. You have found that a given program generates 1 billion loads and 250 million stores per second on one core when the other 3 cores are idle. You see 150 million BRLs, 50 million BWLs, 25 million BRILs, and no BILs[[2]](#footnote-2) (each per second) from this single core when the others are idle. **[14 points]**
   1. What is the hit rate of loads on this core? Of stores? **[4]**  
        
      Load hit rate=\_\_\_1 - 150m/1000m = 85%\_\_ Store hit rate=\_1 – 25m/250m = 90%\_\_
   2. What percent of misses in the cache result in the eviction of dirty data? **[2]**pevict = 50m/(150m + 25m) = 28.6%
   3. All four cores are then turned on (running the same program). All 4 cores end up generating the same number of loads and stores per second, but each core now generates 200 million BRLs, 40 million BWLs, 50 million BRILs and 10 million BILs (each per second). Of those transactions 10% of the BRLs and BRILs result in a HITM (that is the data is dirty in another cache and is supplied by that cache).   
      1. What might explain the drop in the number of BWLs? **[2]**  
           
         Dirty lines are being pulled to other processors’ caches before being displaced to memory during eviction.
      2. What percentage of the load misses are likely due to coherency misses? **[3]**  
           
         (200m – 150m)/200m = 25%
      3. What percentage of the store misses are likely due to coherency misses? **[3]**

(50m – 25m)/50m = 50%

1. You are talking to some of your relatives about computers over the summer. They are quite computer and math literate but don’t really know anything about computer architecture. When they ask you why modern computers have more than one core (and that number is rising), you tell them that it has to do with it being too hard to power a single core. That of course confuses them and they ask “Why can we power a lot of cores but not one core?” Explain the issue to them. Feel free to use math but be sure to explain the ideas behind the math.   
     
   This essay will be graded on the basis of A) hitting the relevant topics correctly and B) clearly and ***concisely*** explaining your answer. Grammar and spelling will only be considered if they make your explanation unclear (i.e. we can’t be sure of what you are trying to say). Your answer must be no more than 200 words. Please try to write neatly. **[10 points]**
2. 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 |  |  |  |  |
| **1** | -- |  | 1 |  |  |  |  |
| **2** | -- |  | 2 |  |  |  |  |
| **3** | -- |  | 3 |  |  |  |  |
| **4** | -- |  | 4 |  |  |  |  |
| **5** | -- |  | 5 |  |  |  |  |
|  |  |  | 6 |  |  |  |  |
|  |  |  | 7 |  |  |  |  |
|  |  |  | 8 |  |  |  |  |

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

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

Place the following instructions into the processor:

R3=R3+R1 // A

R1=R1\*R4 // B

R5=R3+R4 // C

R2=R1\*R3 // D

R2=R2+R1 // E

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 0 of the ROB and B in slot 1. When arbitrary decisions need to be made, you are to just make them. ***Clearly cross out any data that is no longer valid.* [15 points]**

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

1. (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 |  |  |  |  |
| **1** | -- |  | 1 |  |  |  |  |
| **2** | -- |  | 2 |  |  |  |  |
| **3** | -- |  | 3 |  |  |  |  |
| **4** | -- |  | 4 |  |  |  |  |
| **5** | -- |  | 5 |  |  |  |  |
|  |  |  | 6 |  |  |  |  |
|  |  |  | 7 |  |  |  |  |
|  |  |  | 8 |  |  |  |  |

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

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

Place the following instructions into the processor:

R3=R3+R1 // A

R1=R1\*R4 // B

R5=R3+R4 // C

R2=R1\*R3 // D

R2=R2+R1 // E

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 0 of the ROB and B in slot 1. When arbitrary decisions need to be made, you are to just make them. ***Clearly cross out any data that is no longer valid.* [15 points]**

1. Taken in part from a U. of Wisconsin PhD qualifier. [↑](#footnote-ref-1)
2. BRL=Bus Read Line BRIL=Bus Read and Invalidate Line, BWL=Bus Write Line, BIL=Bus Invalidate Line [↑](#footnote-ref-2)