CS/ECE 752: Advanced Computer Architecture I

Prof. David A. Wood

Final Exam
May 12, 2003
Approximate Weight: 20%

CLOSED BOOK
TWO SHEETS OF NOTES

NAME: _____

DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO!

Read over the entire exam before beginning. Verify that your exam includes all 11 pages. Budget your time according to the weight of the questions, and your ability to answer them. Limit your answers to the space provided, if possible. If not, write on the BACK OF THE SAME SHEET. Use the back of the sheet for scratch work. WRITE YOUR NAME ON EACH SHEET.

Problem	Possible Points	Points
Problem 1	10	
Problem 2	25	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	20	
Problem 7	15	
Total	**100**	

NAME: _____

Problem 1: (10 points)

Circle either the word True or the word False for the questions below.

True or False: Compiler optimizations such as loop unrolling and if-conversion only improve program performance on statically scheduled processors, not dynamically scheduled processors.

True or False: Non-blocking caches improve performance by only caching requested words rather than multiword blocks.

True or False: Mark Hill's original 3C's classification broke cache misses down into three categories: Capacity, Conflict, and Coherence.

True or False: Increasing a cache's associativity always decreases cache misses.

True or False: Increasing a cache's associativity always increases cache access time.

True or False: Seznec's skewed-associative caches try to reduce conflict misses by hashing the address bits in the cache tag.

True or False: Stream buffers are one way to implement non-binding prefetching.

True or False: Virtual address aliases are never a problem for a virtually-tagged cache if the virtual memory page size times the associativity are less than the cache size.

True or False: TLB Reach can be increased both by increasing the page size and by increasing the number of TLB entries.

True or False: Implementing TLB reload (i.e., TLB miss handling) in hardware always results in lower TLB miss penalties than implementing it in software.

Problem 2: (25 points)

Consider a memory hierarchy with the following components and properties:

- An L1 data cache with 64 KByte capacity, 64 byte blocks, direct-mapped placement, a write-through policy, and physical tags. An L1 hit takes 2 cycles.
- A four-entry victim cache (i.e., victim buffer) between the L1 and L2 with fully-associative placement, LRU replacement, and a "swap-on-hit" policy. A reference that misses in the L1 cache but hits in the victim cache takes a total of 4 cycles.
- An L2 unified cache with 2 Mbyte capacity, 256 byte (address) blocks, 64 byte subblocks, 2-way set-associative, LRU replacement policy, a writeback policy, and physical tags. A reference that misses in both the L1 cache and victim cache but hits in the L2 cache takes a total of 15 cycles.
- A main memory system with 4 Gbyte capacity. A reference that misses in all caches and is satisfied by the main memory takes a total of 250 cycles.
- Physical addresses are 32 bits and the smallest addressable unit is one byte.
- Ignore the instruction cache and TLB.

Part A: (15 points)

How many bits are required to implement this memory system? Be sure to include the state needed to implement the various policies. Complete the table below. Show your work in the space below the table, using additional pages if necessary.

Table 1:

Cache	Bits per Entry/Block	Entries/Blocks per set	Bits per set	Total bits in cache
L1 Cache				
Victim Cache				
L2 Cache				

Part B: (10 points)

NAME: _____

Consider only the L1 data cache and victim cache and assume that all entries in both caches are initially invalid. For the memory reference stream below, determine which references hit or miss at each level (accessed from top to bottom). Assume that the victim cache is only accessed on L1 misses and all accesses that miss in the victim cache will hit in the L2 cache. Assume byte addressing and all accesses are 32-bit loads.

.

Table 2:

Memory Reference	L1/L2 Indexes	Hits in?
0x00000000		L2
0x00000004		
0x00010000		
0x00000014		
0xFFF01010		
0xEEE02000		
0xEEE02040		
0xFFF11000		
0xEEE12000		
0xEEE02040		
0x00010010		
0xEEE02004		
0xFFF01010		
0x00000000		
0x00000030		
0xFFF02010		
0xFFF03060		
0xFFF02020		

Complete the tables below. In the table at left, indicate in which cache each reference hits. In the rightmost table, indicate the number of hits, misses, and cycles for each cache.

Table 3:

Cache	Number of Hits	Number of Misses	Hit Latency	Total Hit Cycles
L1 Cache			2	
Victim Cache			4	
L2 Cache			15	
All				

Hint: Calculate the L1 and L2 index bits for each access in the middle column. Then track of which entries are added to or removed from victim cache.

Problem 3: (10 points)

The Itanium-2 implements an Advanced Load Address Table (ALAT). What is it, problem does it solve, and how is it used? Illustrate with block diagrams and code sequences as appropriate.

NAME: _____

Problem 4: (10 points)

Part A: (5 points)

The Cray-1 implemented an optimization called *chaining*. What is chaining and how does it help performance? In what ways is chaining similar and/or different from the corresponding scalar pipeline optimization?

Part B: (5 points)

VMIPS provides support for if-conversion. Write the VMIPS code for the following loop using if-conversion. The VMIPS instruction set is summarized on the next page.

```
      for I = 1, 64
            if (A[I] < 0) then
                  A[I] = - A[I]
```

NAME: _____

Instruction	Operands	Function
ADDV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2, then put each result in V1.
SUBV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i × R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i × R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1, ..., 63 × R1 into V1.
S--V.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1	VLR,R1	Move contents of R1 to the vector-length register.
MFC1	R1,VLR	Move the contents of the vector-length register to R1.
MVTM	VM,F0	Move contents of F0 to the vector-mask register.
MVFM	F0,VM	Move contents of vector-mask register to F0.

Figure G.3 The VMIPS vector instructions. Only the double-precision FP operations are shown. In addition to the vector registers, there are two special registers, VLR (discussed in Section G.3) and VM (discussed in Section G.4). These special registers are assumed to live in the MIPS coprocessor 1 space along with the FPU registers. The operations with stride are explained in Section G.3, and the use of the index creation and indexed load-store operations are explained in Section G.4.

Problem 5: (10 points)

Part A: (5 points)

Cache access latency is critical to memory system performance. Several recent microprocessors implement *way prediction* to reduce access latency. What is meant by way prediction? How does it help? Sketch one possible implementation of way prediction.

Part B: (5 points)

Several recent microprocessors implement *sum-address caches* as another approach to reduce effective cache access latency. Explain the key insight behind sum-address caches. Sketch a simple sum-address cache and explain why its access time will be smaller than a classical cache organization.

Problem 6: (20 points)

Part A: (5 points)

Rotenberg, et al. proposed something called a *trace cache*. What is a trace cache? What problems does it attempt to solve?

Part B: (5 points)

What is trace scheduling? Give a brief example. What problems does it attempt to solve?

Part C: (5 points)

The Intel Pentium IV implements a trace cache. How does this differ from the original trace cache proposal? What impact do these differences have on performance?

Part D: (5 points)

Transmeta's Crusoe processor combines aspects of trace caches and trace scheduling. Explain what this processor does. How is it similar to and/or different from the Pentium IV approach.

Problem 7: (15 points)

Consider a multiprocessor system that uses broadcast snooping cache coherence. Four processors, P1, P2, P3, and P4 perform the following sequence of loads and stores to/from lines A and B. Assume A and B do not conflict in the data caches. Assume the protocol described in the book, which uses the three states: Invalid (I), Shared (S), and Exclusive (E). The table below shows the state of the memory system as time flows down. The cache blocks are represented with the following notation: *address:(state, data)*. For example, *A:(I,0)* means that cache block A is in state I with data value 0. A data value of x means the value is unknown or undefined. Complete the table below, updating the cache and memory states in response to the sequence of loads and stores. Indicate actions taken by the cache and memory controllers: hits, requests to get a block shared or exclusive, and responses to requests. You may use arrows (as shown) to indicate that the state has not changed in that cycle.

P1	P2	P3	P4	MEM
A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:(I,x) B:(I,x)	A:0 B:0
load A miss, get A shared A:(S,0) B:(I,x)				respond with A A:0 B:0
	load B miss, get B shared A:(I,x) B:(S,0)			respond with B A:0 B:0
	load A			
	store B = 1			
		load A		
load B				
			store A = 3	
load A				
			load B	