CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Final Exam
May 10, 2006
12:25-2:25pm, 1240 Computer Sciences
Approximate Weight: 25%

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 9 pages. It is a long exam, so use your time carefully. 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	10	
Problem 3	15	
Problem 4	20	
Problem 5	20	
Problem 6	20	
Total	**95**	

NAME: _____

Problem 1: (10 points)

Part A: (5 points) What is the difference between a *write-through* and a *writeback* cache?

In a write-through cache, stores always update memory. On cache hits, stores update the cache as well. On cache misses, stores *may* or may not allocate the block in the cache (if so, they update it).

In a write-back cache, stores always update cache and never update memory. Memory is updated only when the block is replaced from the cache. On a store miss, the block is normally allocated in the cache and then updated.

Part B: (5 points) What makes correcting disk errors with RAID-5 different from correcting memory errors with Hamming codes?

The only way to tell that a memory fault has occured is to use an error detection or correction code. This is because memory faults simply cause a bit to change from a 0 to a 1, or vice versa.

Disk errors may be detected because the device itself fails to respond or the device indicates that an error has occured (e.g., because the per-sector error detecting codes indicate that the data has been corrupted). Because of this self-identifying property, disk errors are more precisely called *erasures*.

Because an erasure indicates which bit has failed, a simple parity code is sufficient to recover the missing bit.

NAME: _____

Problem 2: (10 points)

Ideally, the 5-stage pipeline discussed in class will complete one instruction every cycle. Stall Cycles Per Instruction (SCPI) is a metric that measures the average number of stalls (i.e., pipeline bubbles) that get introduced per instruction. Thus the processor's CPI = 1 + SCPI. SCPI can be expressed as the sum of the SCPI's of different (independent) factors. For example, CPI = 1 + $SCPI_{data}$ + $SCPI_{branch}$ + $SCPI_{I\text{-}cache}$ +$SCPI_{D\text{-}cache}$ breaks the CPI down into stalls due to data dependence stalls, branch stalls, instruction cache stalls, and data cache stalls.

Part A: (5 points) What is the $SCPI_{D\text{-}cache}$ of a machine with a data cache miss penalty of 81 cycles and a load/store miss rate of 2%, when 28% of instructions are loads and stores?

.

28 ld-st/instr * .02 misses/ld-st * 81 cycles/miss = .45 cycles/instr

Part B: (5 points) Data forwarding (aka bypassing) can eliminate most, but not all, of the stalls due to true data dependences. For the MIPS 5-stage pipeline, give an example code sequence for a dependence that cannot be eliminated using forwarding and must generate a stall. Explain why the hazard cannot be eliminated in this pipeline organization.

load-use delays cannot be eliminated in this pipeline.

lw $1, 0($2) -- Value available at end of M

add $3, $2, $1 -- Value needed by beginning of X

	1	2	3	4	5	6	7
lw	F	D	X	M	W		
add		F	stall	D	X	M	W

NAME: _____

Problem 3: (15 points)

Part A: (3 points) Show the Booth recoding for the 8-bit multiplier -47_{ten}.

0 -1 1 -1 0 0 1 -1

Part B: (3 points) What is the 2-bit modified Booth recoding of the multiplier in Part A?

-1 1 0 1

Part C: (4 points) A 16-bit multiplier has been recoded using the 2-bit modified Booth recoding algorithm. The recoded multiplier is:

```
0 -2 0 2 -1 2 -1 1
```

What is the original 16-bit multiplier?

1110 0001 1101 1101

Part D: (5 points) What is a Wallace Tree? Why is it used?

A Wallace tree is a collection of carry-save-adders followed by a full-adder. Wallace trees are used in multipliers to sum the partial products. Each carry-save adder sums three bits of the same weight and produces one bit of the same weight and one bit with twice the weight. Because carry-save adders reduce the number of partial products from three to two, the height of a Wallace tree is O(log3/2 N). Note that the full adder is needed to reduce the final two partial products to a non-redundant form.

NAME: _____

Problem 4: (20 points)

Consider a memory system with the following parameters. The virtual address has 64 bits. The physical address has 41 bits. A unified (i.e., instructions and data) cache is writeback and has 32 kilobyte capacity with 64-byte blocks. The translation lookaside buffer (TLB) has 64 entries, is direct-mapped, and is accessed *before* the cache. Pages are 8 kilobytes. All addresses are byte addresses.

Part A: (3 points) Show the breakup of a virtual address into virtual page number and byte offset within a page. Indicate the number of bits in each field.

Virtual Page Number<63:13> . Page Offset <12:0>

Part B: (3 points) Show the breakup of a physical address into a page frame number and a byte offset with the page frame. Indicate the number of bits in each field.

Page Frame Number<40:13> . PageOffset<12:0>

Part C: (3 points) How many bits of storage does it take to implement this TLB? Assume the minimum number of bits possible to achieve a correct implementation of address translation.

64 entries * (VPN-index + valid + PFN) = 64 * ((51-6) + 1 + 28) = 464 * 74 = 736

Part D: (2 points) What other bits of state may a TLB maintain? What are these used for?

Reference = Used by OS to approximate LRU or other page replacement policy

Dirty = Used by OS to reduce writeback traffic to disk

Protection = Use by OS to limit access to pages

Part E: (3 points) Suppose the cache is two-way set associative with least-recently-used (LRU) replacement. Show the break up of the physical address into tag, index, and byte within block used to access the cache. Indicate the number of bits in each field.

Tag<40:14> . Index<13:6> . Offset<5:0>

25 8 6

Part F: (3 points) How many sets are there in the cache?

256 sets

Part G: (3 points) How many total *bits* does it take to implement the cache (i.e., tags, state, data, LRU, etc.).

tags & state = 512 blocks * (25 tag bits + valid + dirty) = 512 * 27

lru = 256 * 1

data = 32KB * 8 = 262144

total = 276,224

NAME: _____

Problem 5: (20 points)

Part A: (2 points) What is the Hamming distance between 01101101_{two} and 01010110_{two}?

5

Part B: (3 points) What is the minimum Hamming distance needed between any pair of valid code words to detect a single bit error?

2

Part C: (3 points) What is the minimum Hamming distance needed between any pair of valid code words to correct a single bit error?

3

Part D: (3 points) What is the minimum Hamming distance needed between any pair of valid code words to correct a single bit error AND detect a double bit error?

4

Part E: (3 points) The parity check matrix for a SECDED code is given below. In this matrix C_i's denote check bits and b_i's denote information bits. The codewords are stored in memory in the same bit order as shown in the parity check matrix.

	C_1	C_2	b_1	C_3	b_2	b_3	b_4	C_4
H =	1	0	1	0	1	0	1	0
	0	1	1	0	0	1	1	0
	0	0	0	1	1	1	1	0
	1	1	1	1	1	1	1	1

Consider the data word $b_1b_2b_3b_4 = 0011$. Assuming *odd parity*, what is the codeword that is stored in memory for the data word given the above parity check matrix?

0101 0111

NAME: _____

Part F: (3 points) Suppose that the word read from the memory is 0101 0101, calculate the syndrome using the parity check matrix above.

Syndrome = c_3 ^ c_3', c_2 ^ c_2', c_1 ^ c_1'

Syndrome = 1 ^ 0, 1 ^ 0. 0 ^ 1 = 111, bit 7 is in error

Part G: (3 points) What is the procedure for detecting a double error?

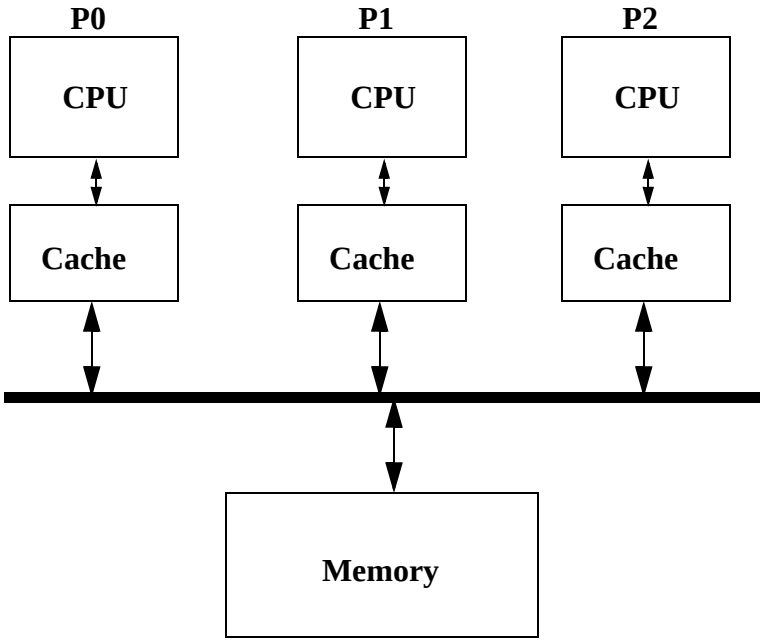
If the syndrome is non-zero AND the global parity of the word read from memory is odd (i.e., indicates an even number of errors) then we assume that there were two errors.

Part H: (3 points, extra credit) Was there a double bit error in Part F? If not, what was the value originally written into memory? Like most real systems, ignore the possibility of more than two errors.

No, there was only a single bit error. The word read from memory has an even number of ones.

The correct value is: 0101 0111, which means the original data value was 0011.

Problem 6: (20 points)



Symmetric Multiprocessors (SMPs) are the most common computer systems that use multiple processors. Most SMPs use a shared bus to connect the processors to a (logically) shared memory, as illustrated above. Each processor has one or more *writeback* caches to reduce both memory latency and contention for the shared bus. A *write-invalidate* cache coherence protocol is used to ensure that caches maintain a coherent view of the memory state.

Assume a three-state write-invalidate protocol, as discussed in class and in the textbook. The three states are:

```
Modified (M), also known as Read/Write (Dirty)
Shared (S), also known as Read Only (Clean)
Invalid (I)
```

Consider an SMP with the initial memory state as shown on the next page. For simplicity, the table shows only the memory block at address 100 and the block only contains one word. Consider a sequence of processor operations (time flows down the page). For each processor operation, write a one or two sentence description of the actions taken by the coherence protocol (I have completed the first one for you, as an example). Fill in the tables with the state of the caches and memory (for block 100) after each operation has completed.

Recall that $0 in MIPS always contains the value zero.

Initial state:

P0		P1		P2		Memory
State	**Data**	**State**	**Data**	**State**	**Data**	**Data**
I	**0**	**I**	**0**	**I**	**0**	**7**

Operation: Processor P0 executes lw $1, 100($0)

Action: P0 misses in the cache and reads the block from memory

Memory state:

P0		P1		P2		Memory
State	**Data**	**State**	**Data**	**State**	**Data**	**Data**
S	**7**	**I**	**0**	**I**	**0**	**7**

Operation: Processor P1 executes addi $1, $0, 13 and sw $1, 100($0)

Action: P1 misses, reads the block from memory and invalidates P0

Memory state:

P0		P1		P2		Memory
State	**Data**	**State**	**Data**	**State**	**Data**	**Data**
I	**7**	**M**	**13**	**I**	**0**	**7**

Operation: Processor P2 executes lw $1, 100($0)

Action: P2 misses, reads the block, P1 intervenes and supplies the block, updating memory

Memory state:

P0		P1		P2		Memory
State	**Data**	**State**	**Data**	**State**	**Data**	**Data**
I	**7**	**S**	**13**	**S**	**13**	**13**

Operation: Processor P1 executes addi $1, $0, 19 and sw $1, 100($0)

Action: P1 hits, but does not have permission to write. P1 invalidates P2's copy.

Memory state:

P0		P1		P2		Memory
State	**Data**	**State**	**Data**	**State**	**Data**	**Data**
I	**7**	**M**	**19**	**I**	**13**	**13**