CSCE 692 - HOMEWORK 5 (CHAPTER 5)

(100 Points)

Due: NLT 1100 Tuesday, 5 March 2019

Solutions

**Instructions:**

* Number each page (last name-pg)
* Show your work
* Clearly indicate your answer

**Hennessey and Patterson Problems (25 points each): 5.1, 5.2,**

**Additional problems 1, 2 (25 pts each)**

Notes:

5.1

* In each part of the problem, start back in the initial state of Figure 5.37
  + I.e., the memory accesses do not build upon one another.
* Modify (M) is the same as our Exclusive (E).
* The mapping from memory starts at block 0, 1, 2, 3, and just repeats (hex).
  + 00 → 0
  + 08 → 1
  + 10 → 2
  + 18 → 3
  + 20 → 0
  + 28 → 1
  + 30 → 2
  + 38 → 3

5.2

* Read the problem setup carefully, there are many assumptions
* In each part of the problem, start back in the initial state of Figure 5.37
  + I.e., the memory accesses from (b) do not build upon those from (a).

Additional Problems 1, 2

* DI indicating Directory Invalid is the same as saying that value in the directory is currently *uncached* and resides only in memory.
* Assume a Write-Back policy for L1 cache, and a Write-Through for L2 cache

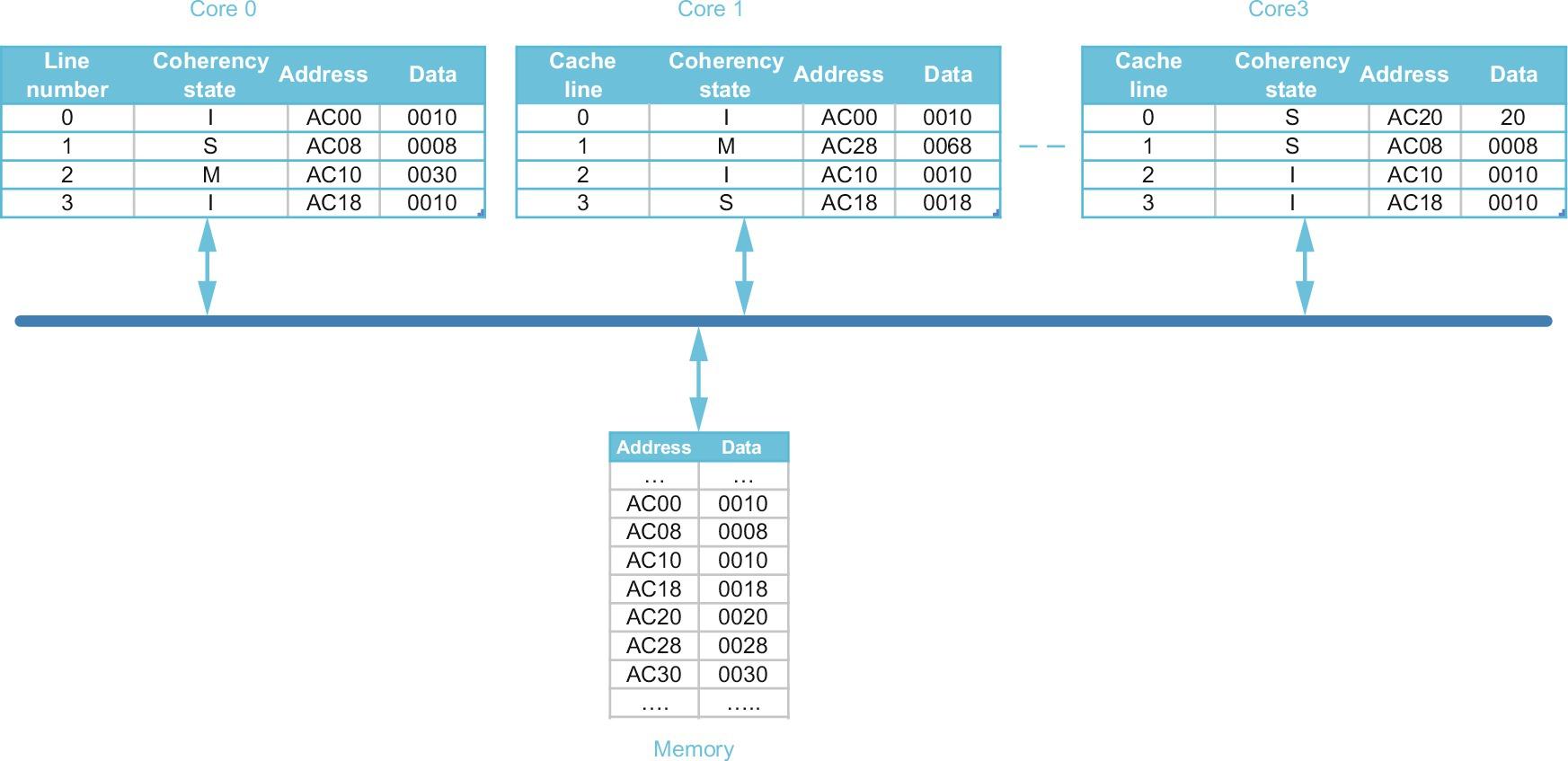


Figure 5.37 Multicore (point-to-point) multiprocessor

**5.1 – Snooping Coherence** Given initial state in Fig 5.37 (above), Show only the cache lines that experience some state change; for example: C0.0: (I, AC20, 0001) indicates that line 0 in core 0 is in an “invalid” coherence state (I), holds (invalid) data of 0001 from memory address AC20. Furthermore, represent any changes to the memory state as M.< address >: <value>. Assume the actions in all parts are applied to the initial cache and memory states; e.i., different parts (a) through (g) do not depend on one another. Show blocks that are changed after:

1. C0: R, AC20 // read miss

C0.0: (S, AC20, 0020), // read returns 0020

1. C0: W, AC20 ← 80 // Write miss

C3.0: (I, AC20, 0020), // block shared, goes to invalidate

C0.0: (M, AC20, 0080), // Writes 80 to address AC20

1. C3: W, AC20 ← 80 // Write hit

C3.0: (M, AC20, 0080) // Updates data, goes to modified state

1. C1: R, AC10 // Read miss

C0.2: (S, AC10, 0030) // Exclusive owner shares AC10

M.AC10(0030) // Data written back IAW slide 37

C1.2: (S, AC10, 0030) // Read returns 0030

1. C0: W, AC08 ← 48 // Write hit

C0.1: (M, AC08, 0048) // Updates local copy of data

C3.1: (I, AC08, 0008) // C3.1 holds invalid data of 0008 for AC08

1. C0: W, AC30 ← 78 // Write miss

M.AC10(0030) // Writes back AC10 on replacement

C0.2: (M, AC30, 0078) // Satisfied from memory

1. C3: W, AC30 ← 78 // Write miss

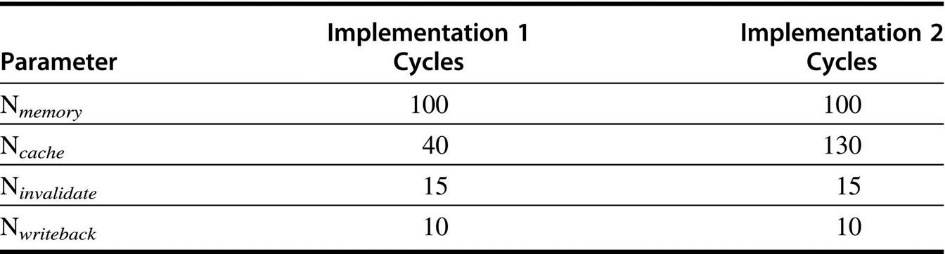
C3.2: (M, AC30, 0078), // Satisfied from memory 

Figure 5.38 Snooping coherence latencies

5.2 --- Snooping Caches

For the following sequences of operations, how many stall cycles are generated by each implementation?

**Assumptions:**

Ninvalidate indicates an “Invalid” on the bus, not just a block moving to invalid

a. C0: R, AC20 // Read miss, satisfied by memory

C0: R, AC28 // Read miss, satisfied by core1 cache

C0: R, AC30 // Read miss, satisfied by memory

[Nmem] + [Ncache+Nwback] + [Nmem+Nwback]

Implementation 1: 100 + (40 + 10) + (100 + 10) = 260 stall cycles

Implementation 2: 100 + (130 + 10) + (100 + 10) = 350 stall cycles

b. C0: R, AC00 // Read miss, satisfied by memory

C0: W, AC08 < -- 48 // Write hit, invalidate

C0: W, AC30 < -- 78 // Write miss, satisfied by memory, WB

Implementation 1: 100 + 15 + (100 + 10 ) = 225 stall cycles

Implementation 2: 100 + 15 + (100 + 10 ) = 225 stall cycles

c. C1: R, AC20 // Read miss, satisfied by memory

C1: R, AC28 // Read hit

C1: R, AC30 // Read miss, satisfied by memory

Implementation 1: 100 + 0 + 100 = 200 stall cycles

Implementation 2: 100 + 0 + 100 = 200 stall cycles

d. C1: R, AC00 // Read miss, satisfied by memory

C1: W, AC08 < -- 48 // Write miss, satisfied by memory

// Write back, no invalidate IAW slide 37

C1: W, AC30 < -- 78 // Write miss, satisfied by memory

Implementation 1: 100 + (100 + 10) + (100) = 310 stall cycles

Implementation 2: 100 + (100 + 10) + (100) = 310 stall cycles

Additional Problem 1 – Directory Caches

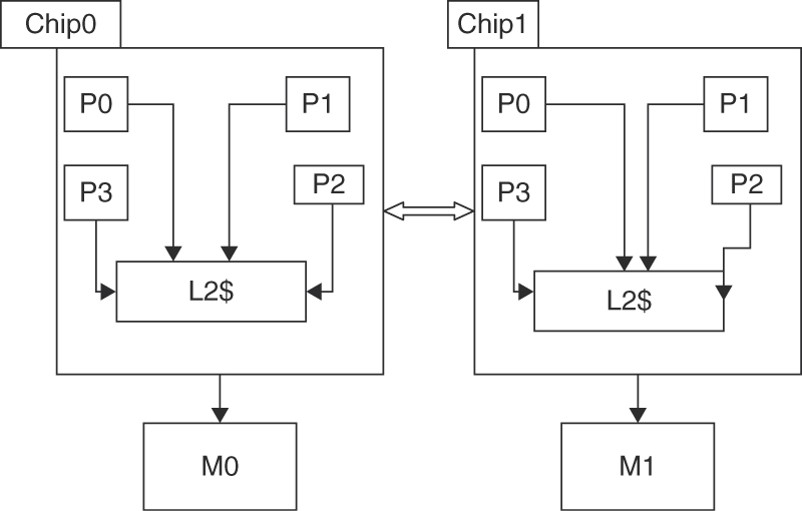


Figure HW5.1 Multichip, multicore multiprocessor with Distributed, Shared Memory

Consider the distributed shared-memory system illustrated in Figure HW5.1 consisting of two four-core chips. The processors in each chip share a Level 2 cache (L2$), and the two chips are connected via a point-to-point interconnect. The system memory is distributed across the two chips. Figure HW5.2 zooms in on part of this system.

Pi,j denotes Processor i in Chip j. Each processor has a single direct-mapped L1 cache that holds two blocks, each holding two words. Each chip has a single direct-mapped L2 cache that holds four blocks, each holding two words. To simplify the illustration, the cache address tags contain the full address and each word shows only two hex characters, with the least significant word on the right.

The L1 cache states are denoted M, S, and I for Modified, Shared, and Invalid. Both the L2 caches and memories have directories. The directory states are denoted DM, DS, and DI for Directory Modified, Directory Shared, and Directory Invalid. The simple directory protocol is described in Figures 5.20 and 5.21 (H&P Ed 6).

The L2 directory lists the local sharers/owners and additionally records if a line is shared externally in another chip; for example, P1,0;E denotes that a line is shared by local processor P1,0 and is externally shared in some other chip. The memory directory has a list of the chip sharers/owners of a line; for example C0,C1 denotes that a line is shared in Chips 0 and 1.

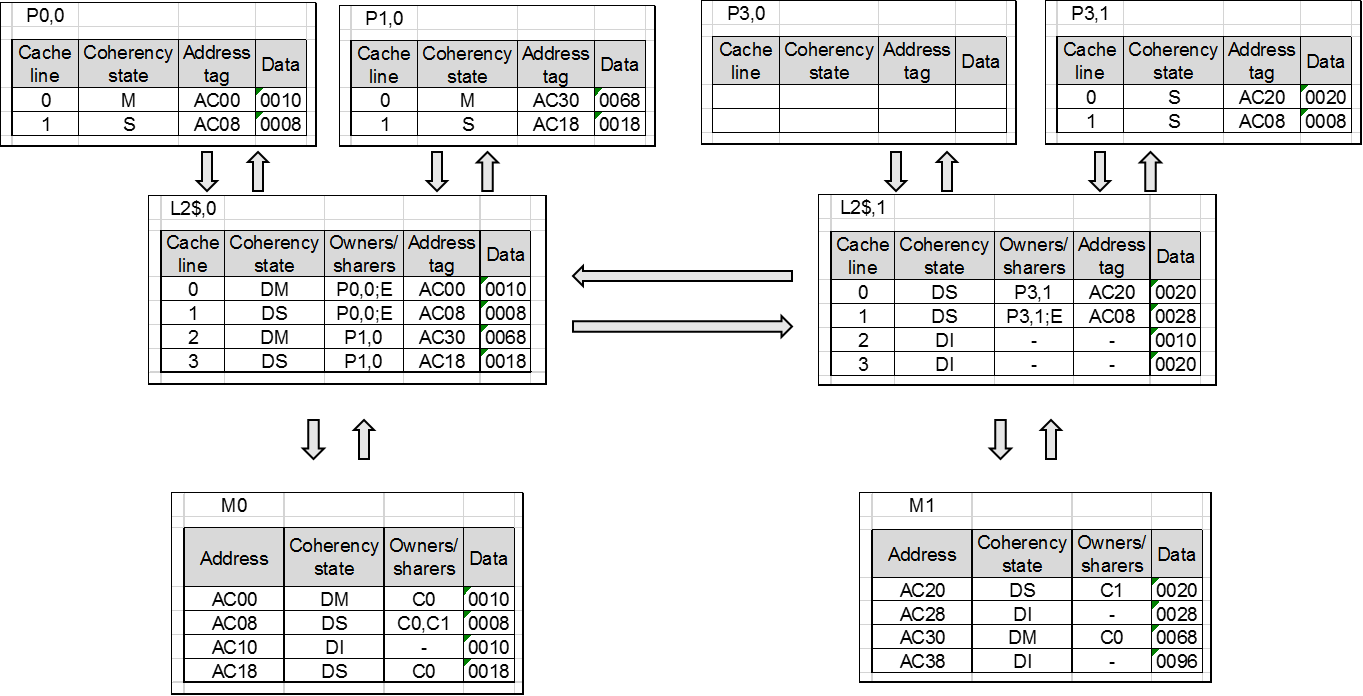


Figure HW5.2 Cache and memory states in the multichip, multicore multiprocessor

Additional Problem 1 – Directory Caches

For each part of this exercise, assume the initial cache and memory state in Figure HW5.2. Each part of this exercise specifies a sequence of one or more CPU operations of the form: P#: <op> <address> [ 🡨 <value> ] , where P# designates the processor (e.g., P0,0) <op> is the CPU operation (e.g., read or write). <address> denotes the memory address, and <value> indicates the new word to be assigned on a write operation. What is the final state (i.e., coherence stat, sharers/owners, tags, and data) of the caches and memory after the given sequence of CPU operations has completed? Also, what value is returned by each read operation?

a. P0,0: read AC00 // Hit in L1, P0,0 had as M so no change

// Read returns 0010

b. P0,0: read AC28 // miss in both L1 & L2 – evict AC08

P0,0.1: (S, AC28, 00 28) // shared, address AC28 with data 0028

L2,0.1: (DS, P0,0; AC28, 0028) // L2 cache now has AC28 with data 0028

// owned by P0,0

M0.AC08: (DS, C1, 00 08) // Now only C1 has a copy of AC08

L2-1.1: (DS, P3,1; AC08, 00 08) // L2-1.1 is no longer shared externally

M1.AC28: (DS, C0, 0028) // M1.AC28 is shared to C0 with data 0028

// Read returns 0028

c. P0,0: write AC28 <-- 78 // Miss in L1 & L2, evict AC08

P0,0.1: (M, AC28, 0078) // Modifies AC28 with value 0078

L2,0.1: (DM, P0,0; AC28, 0028) // Cache is in modified state, holds previous

value from bringing in AC28 from memory

M0.AC08: (DS, C1, 0008) // No longer shared by C0

L2-1.1: (DS, P3,1; AC08, 0008) // No longer externally shared

M1.AC28: (DM, C0, 0028) // Modified state, shared with C0

Additional Problem 1 (continued)

d. **P0,0: read AC20** // Miss in L1, hit in L2,1 – evict AC00

P0,0.0: (S, AC20, 0020) // Brings AC20 – 0020 from L2,1.0

L2,0.0: (DS, P0,0;E; AC20, 0020) // L2,0 evicts AC00

M0.AC00: (DI, - , 0010) // Value of AC00 written back

L2,1.0: (DS, P3,1;E; AC20, 0020) // L2,1.0 is now externally shared to chip 0.

M1.AC20: (DS; C0, C1; 0020) // AC20 shared to C0 and C1

e. **P0,0: read AC20** // Miss in L1, hit in L2,1 – evict AC00

P0,0.0: (S, AC20, 0020) // Brings AC20 – 0020 from L2,1.0

L2,0.0: (DS, P0,0;E; AC20, 0020) // L2,0 evicts AC00

M0.AC00: (DI, - , 0010) // Value of AC00 written back

L2,1.0: (DS, P3,1;E; AC20, 0020) // L2,1.0 is now externally shared to chip 0.

M1.AC20: (DS; C0, C1; 0020) // AC20 shared to C0 and C1

**P1,0: read AC20** // Hit in L2 – evict AC30 from L1, AC30 stays in L2 cache

P1,0.0: (S, AC20, 0020) // Replaces AC30 with AC20

L2,0.0: DS, {P0,0; P1,0; E}; AC20, 0020) // Shared by Proc. 1 & 2 on chip 0;

// and on chip 1

L2,0.2: (DI, - , AC30, 0068) // Directory invalid because there are no owners

M1.AC30 ( DI, - , 0068) // Memory reflects no ownership/sharing of AC30

f. **P0,0: read AC20** // Miss in L1, hit in L2,1 – evict AC00

P0,0.0: (S, AC20, 0020) // Brings AC20 – 0020 from L2,1.0

L2,0.0: (DS, {P0,0; E}; AC20, 0020) // L2,0 evicts AC00

M0.AC00: (DI, - , 0010) // Value of AC00 written back

L2,1.0: (DS, P3,1;E; AC20, 0020) // L2,1.0 is now externally shared to chip 0.

M1.AC20: (DS; C0, C1; 0020) // AC20 shared to C0 and C1

**P1-0: write AC20 <-- 80** // Miss in L1, hit in L2,0 – evict AC30

P1,0.0: (M, AC20, 0080) // Updated value written

P0,0.0: (I, AC20, 0020) // AC20 invalid in P0,0.0

L2,0,0: (DM, P1,0; AC20, 0020) // P1,0 is the exclusive owner

value not written back

L2,0,2: (DI, - , -, -) // AC30 written back into L2

M.AC30: (DI, - , 0068) // AC30 written back, no owners

L2,1,0: (DI, - , - , -) // AC20 modified, does not have

updated value

P3,1.0: (I, AC20, 0020) // AC20 now invalid

M.AC20: (DM, C0, 0020) // AC20 owned by P1,0

Additional Problem 1 (continued)

g. **P0,0: write AC20 <-- 80** // Miss in L1, hit in L2,1 – evict AC00

P0,0.0: (M, AC20, 0080) // Brings AC20 – 0020 from L2,1.0

L2,0.0: (DM, P0,0; AC20, 0020) // L2,0 evicts AC00, doesn’t have

updated value of AC20 yet

M0.AC00: (DI, - , 0010) // Value of AC00 written back, no owners

P3,1.0: (I, AC20, 0020) // Invalid because of write

L2,1.0: (DM, E; AC20, 0020) // L2,1.0 is now externally shared/owned externally.

// It is DM because of the write

M1.AC20: (DM; C0; 0020) // AC20 exclusively owned by C0

**P1,0: read AC20** // Read miss in L1 – evict AC30

P0,0.0: (S, AC20, 0080) // Replies to read miss with updated value

L2,0.0: (DS, {P0,0, P1,0}; AC20, 0080) // Stores updated value

P1,0.0: (S, AC20, 0080) // Receives updated reply

L2,0.2: (DI, - , -, -) // Directory invalid for AC30 because it’s not in L1

M.AC30: (DI, - , 0068) // Value of AC30 written back to memory

L2,1.0: (DS, {P3,1;E}, AC20, 0080) // P3,1 now shares AC20 value

h. **P0-0: write AC20 <-- 80** // Miss in L1, hit in L2,1 – evict AC00

P0,0.0: (M, AC20, 0080) // Brings AC20 – 0020 from L2,1.0

L2,0.0: (DM, P0,0; AC20, 0020) // L2,0 evicts AC00, doesn’t have

updated value yet

M0.AC00: (DI, - , 0010) // Value of AC00 written back

P3,1.0: (I, AC20, 0020) // Invalid because of write

L2,1.0: (DM, E; AC20, 0020) // L2,1.0 is now externally shared/owned externally.

// It is DM because of the write

M1.AC20: (DM; C0; 0020) // AC20 exclusively owned by C0

**P1-0: write AC20 <-- 90** // Miss in L1 – evict AC30

P0,0.0: (I, AC20, 0080) // Replies with current data, but invalid b/c of write

L2,0.0: (DM, P1,0; AC20, 0080) // Owned by P1,0; but gets value from P0,0

P1,0.0: (M, AC20, 0090) //P1,0.0 has new value

L2,0.2: (DI, - , -, -) // P1,0 no longer has AC30

M1.AC30: (DI, - , 0068) // Value of AC30 written back to memory

M1.AC20(DM, C0, 0080) // M1 has the previously written value for AC20

L2,1.0: (DM, E, AC20, 0080) // Has written back value from P0-0, but not P1-0.

Additional Problem 1 – Directory Caches

Consider the 8 processor system in Figure HW5.1, under the assumption that the caches which are not shown in Figure HW5.2 have invalid blocks.

Identify which nodes (chip/processor cache, chip/L2 cache, chip/memory) receive each request and invalidate message for the sequences below:

a. P0,0: write AC00 <-- 80 // Write hit

b. P0,0: write AC08 <-- 88 // Places invalidate on the bus

// Write Hit

L2,0.1 // Receives invalidate Makes P0,0 the exclusive owner

L2,1.1 // Receives invalidate

P3,1.1 // Receives invalidate

c. P0,0: write AC18 <-- 90 // Send write miss

L2,0 // Receives write miss from P0,0

// Sends a data value reply message to P0,0

// with current value of AC18

// Invalidates its own cache block 3

P0,0 // Receives data value reply from L2,0

P1,0.1 // Receives invalidate from L2,0

d. P1,0: write AC28 <-- 98 // Sends a write miss

L2,0 // Receives a write miss from P1,0

// Sends fetch/invalidate to M1 for data address AC28

M.AC28(DS, C0, 0028) // M.AC28 receives fetch/invalidate

// Sends data value reply to chip 0

// Invalidates local data for AC28

L2,0 // Receives data value reply from M.AC28

// Sends data value reply to P1,0

// Invalidates cache block 1 containing AC28

P1,0.1 // Receives data value reply from L2,0