### CSCE 692 – Chapter 5 Homework Problems

2nd Lt David Crow – 28 February 2019

(100 Points)

Due: NLT 1100 Tuesday, 5 March 2019

**Instructions:**

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

**Book Problems (25 points each): 5.1, 5.2**

**Additional Problems (25 points each): 1, 2**

**Notes:**

Book Problem 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

Book Problem 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

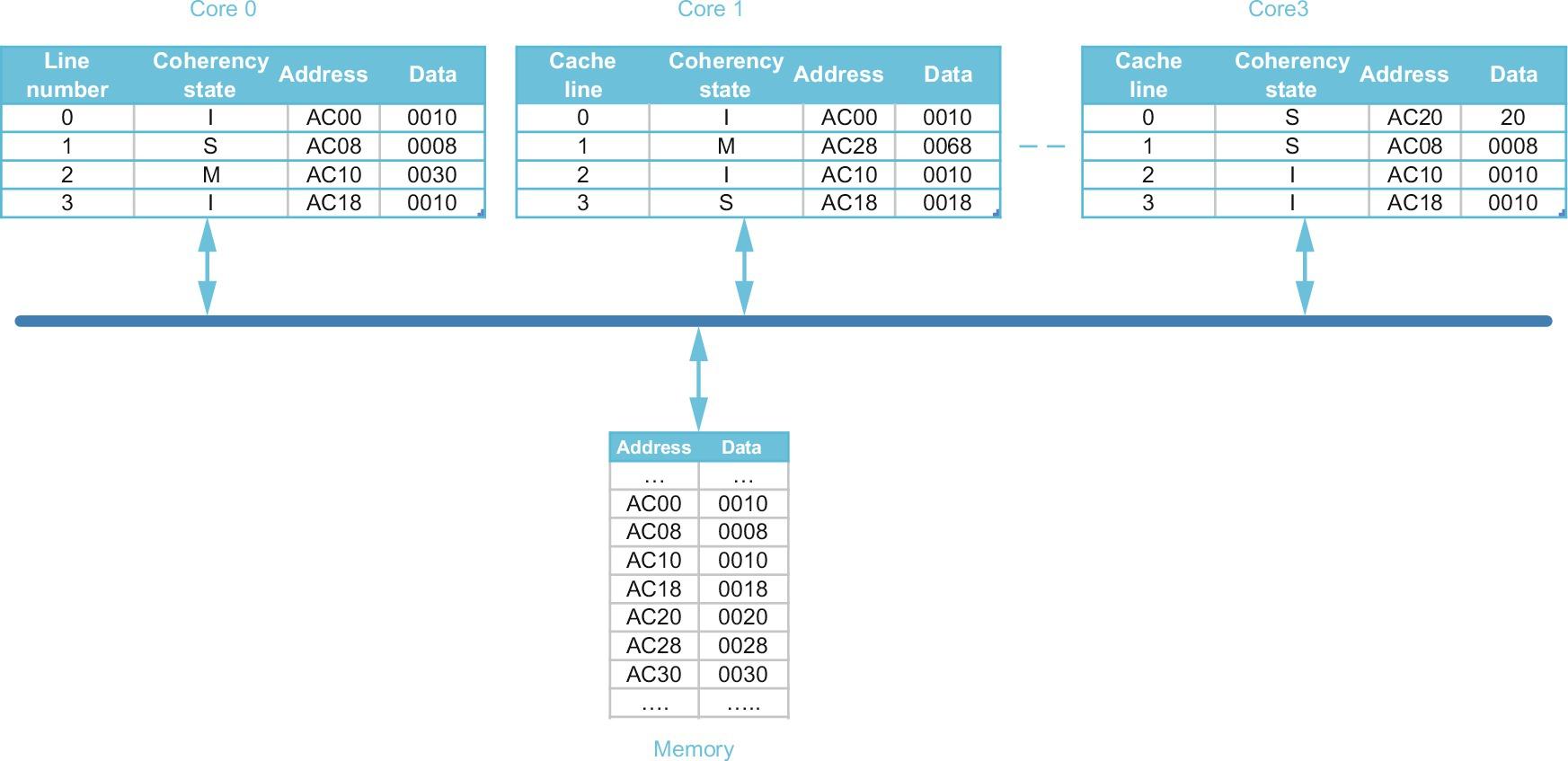


Figure 5.37 Multicore (point-to-point) multiprocessor

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) and 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; i.e., 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

C0.0: (M, AC20, 0080) // fetch and update AC20 in C0

C3.0: (I, AC20, 0020) // invalidate C3’s AC20

1. C3: W, AC20 ← 80 // write hit

C3.0: (M, AC20, 0080) // update C3’s AC20

1. C1: R, AC10 // read miss

M.AC10.0030 // write C0’s AC10 to memory

C0.2: (S, AC10, 0030) // share C0’s AC10

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

1. C0: W, AC08 ← 48 // write hit

C0.1: (M, AC08, 0048) // update C0’s AC08

C3.1: (I, AC08, 0008) // invalidate C3’s AC08

1. C0: W, AC30 ← 78 // write miss

C0.2: (M, AC30, 0078) // fetch and update AC30 in C0

M.AC10.0030 // writeback C0’s AC10

1. C3: W, AC30 ← 78 // write miss

C3.2: (M, AC30, 0078) // fetch and update AC30 in C3

5.2 – Snooping Caches

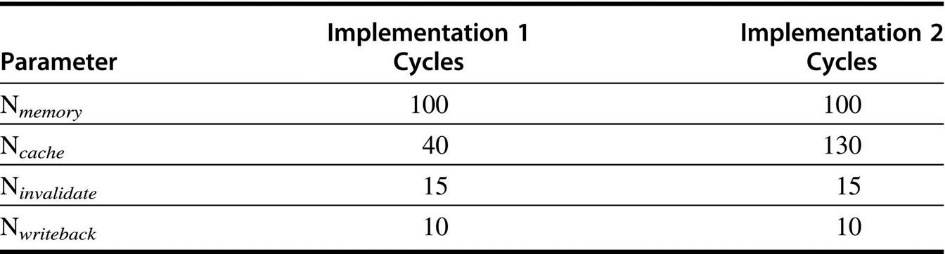


Figure 5.38: Snooping coherence latencies

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

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

C0: R, AC28 // read miss satisfied by C1; writeback

C0: R, AC30 // read miss satisfied by memory; writeback

Implementation 1: stall cycles

Implementation 2: 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; writeback

Implementation 1: stall cycles

Implementation 2: 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: stall cycles

Implementation 2: stall cycles

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

C1: W, AC08 ← 48 // write miss satisfied by memory; writeback; invalidate

C1: W, AC30 ← 78 // write miss satisfied by memory

Implementation 1: stall cycles

Implementation 2: stall cycles

Additional Problem 1 – Directory Caches

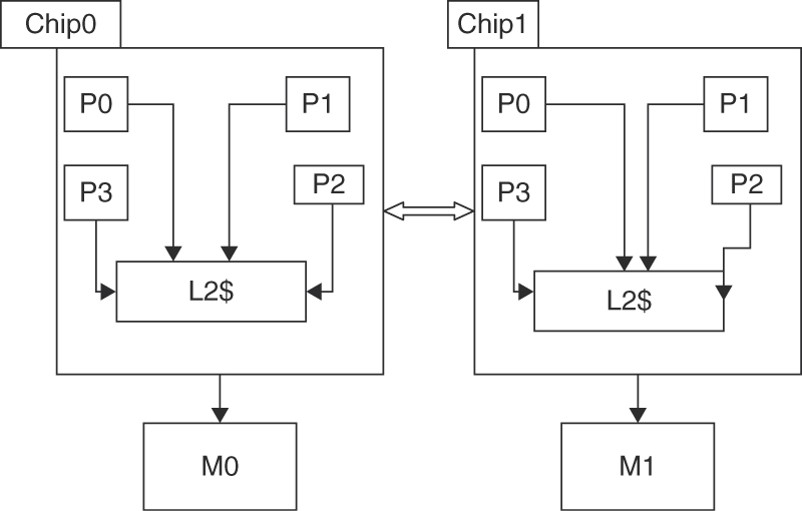
­

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.

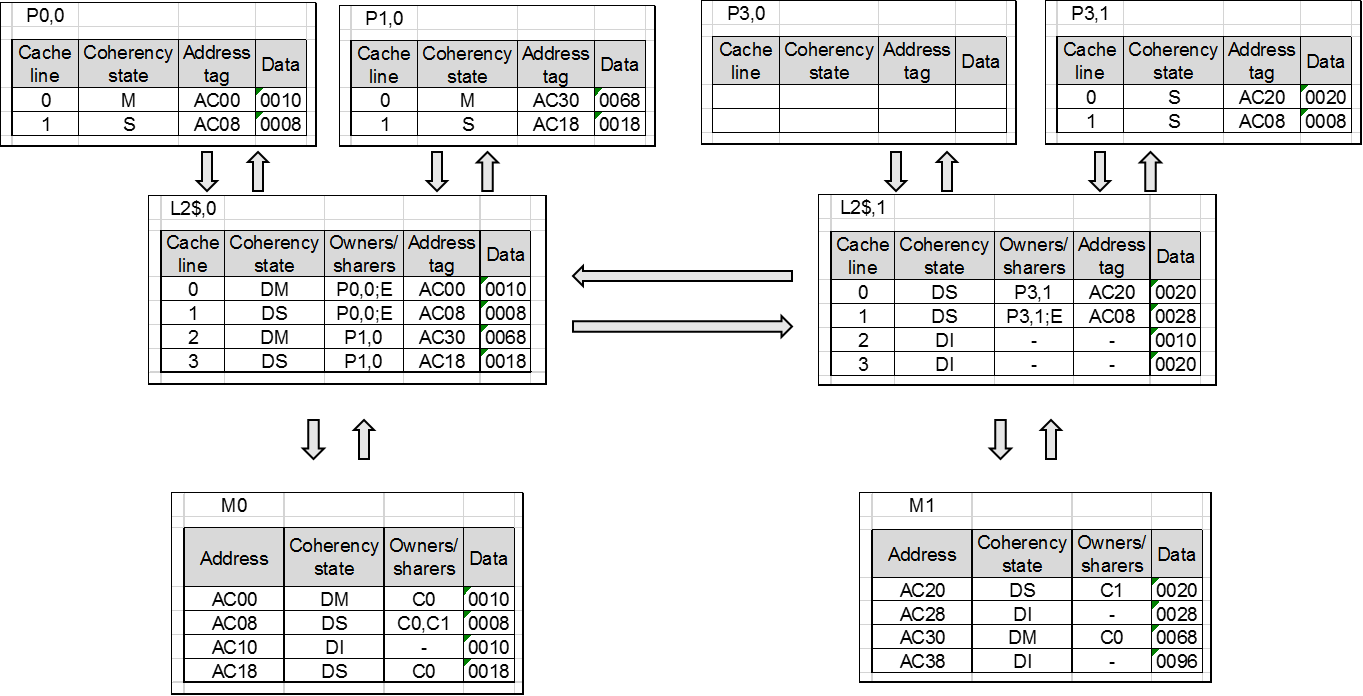


Figure HW5.2: Cache and memory states in the multichip, multicore multiprocessor

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?

1. P0,0: read AC00
2. P0,0: read AC28 // read returns 0028

P0,0.1: (S, AC28, 0028)

L2,0.1: (DS, P0,0, AC28, 0028)

M0.AC08: (DS, C1, 0008)

L2,1.1: (DS, P3,1, AC08, 0008)

M1.AC28: (DS: C0, 0028)

1. P0,0: write AC28 ← 78

P0,0.1: (M, AC28, 0078)

L2,0.1: (DM, P0,0, AC28, 0028)

M0.AC08: (DS, C1, 0008)

L2,1.1: (DS, P3,1, AC08, 0008)

M1.AC28: (DM, C0, 0028)

1. P0,0: read AC20 // read returns 0020

P0,0.0: (S, AC20, 0020)

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

M0.AC00: (DI, -, 0010)

L2,1.0(DS, P3,1/E, AC20, 0020)

M1.AC20: (DS, C0/C1, 0020)

1. P0,0: read AC20 // read returns 0020

P1,0: read AC20 // read returns 0020

P0,0.0: (S, AC20, 0020)

P1,0.0: (S, AC20, 0020)

L2,0.0: (DS, P0,0/P1,0/E, AC20, 0020)

L2,0.2: (DI, -, -, -)

M0.AC00: (DS, -, 0010)

L2,1.0: (DS, P3,1/E, AC20, 0020)

M1.AC20: (DS, C0/C1, 0020)

M1.AC30: (DS, -, 0034)

1. P0,0: read AC20 // read returns 0020

P1,0: write AC20 ← 80

P0,0.0: (I, AC20, 0020)

P1,0.0: (M, AC20, 0080)

L2,0.0: (DM, P1,0, AC20, 0020)

L2,0.2: (DI, -, -, -)

M0.AC00: (DI, -, 0010)

P3,1.0: (I, AC20, 0020)

L2,1.0: (DI, -, -, -)

M1.AC20: (DM, C0, 0020)

M1.AC30: (DI, -, 0034)

1. P0,0: write AC20 ← 80

P1,0: read AC20 // read returns 0080

P0,0.0: (S, AC20, 0080)

P1,0.0: (S, AC20, 0080)

L2,0.0: (DS, P0,0/P0,1, AC20, 0080)

L2,0.2: (DI, -, -, -)

M0.AC00: (DI, -, 0010)

P3,1.0: (I, AC20, 0020)

L2,1.0: (DI, -, -, -)

M1.AC20: (DS, C0, 0080)

M1.AC30: (DI, -, 0034)

1. P0,0: write AC20 ← 80

P1,0: write AC20 ← 90

P0,0.0: (I, AC20, 0080)

P1,0.0: (M, AC20, 0090)

L2,0.0: (DM, P1,0, AC20, 0080)

L2,0.2: (DI, -, -, -)

M0.AC00: (DI, -, 0010)

P3,1.0: (I, AC20, 0020)

L2,1.0: (DI, -, -, -)

M1.AC20: (DM, C0, 0020)

M1.AC30: (DI, -, 0034)

Additional Problem 2 – 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:

1. P0,0: write AC00 ← 80
2. P0,0: write AC08 ← 88

L2,0 receives Invalidate // sends and receives write hit to itself

L2,1 receives Invalidate

P3,1 receives Invalidate

1. P0,0: write AC18 ← 90

L2,0 receives Write Miss

M0 receives Fetch

L2,0 receives Data Value Reply

L2,0 receives Invalidate // for the AC08 in P0,0.1

P1,0 receives Invalidate

1. P1,0: write AC28 ← 98

L2,0 receives Write Miss

L2,1 receives Fetch

M1 receives Fetch

L2,1 receives Data Write Back

L2,0 receives Data Write Back

P1,0 receives Data Value Reply