HW4 Answer

5.1

Cx.y is cache line y in core x.

a. C0: R AC20 → C0.0: (S, AC20, 0020), returns 0020

b. C0: W AC20 ← 80 → C0.0: (M, AC20, 0080)
C3.0: (I, AC20, 0020)

c. C3: W AC20 ← 80 → C3.0: (M, AC20, 0080)

d. C1: R AC10 → C1.2: (S, AC10, 0010) returns 0010

e. C0: W AC08 ← 48 → C0.1 (M, AC08, 0048)
C3.1: (I, AC08, 0008)

f. C0: W AC30 ← 78 → C0.2: (M, AC30, 0078)
M: AC10 ← 0030 (write-back to memory)

g. C3: W AC30 ← 78 → C3.2 :(M, AC30, 0078)

5.2

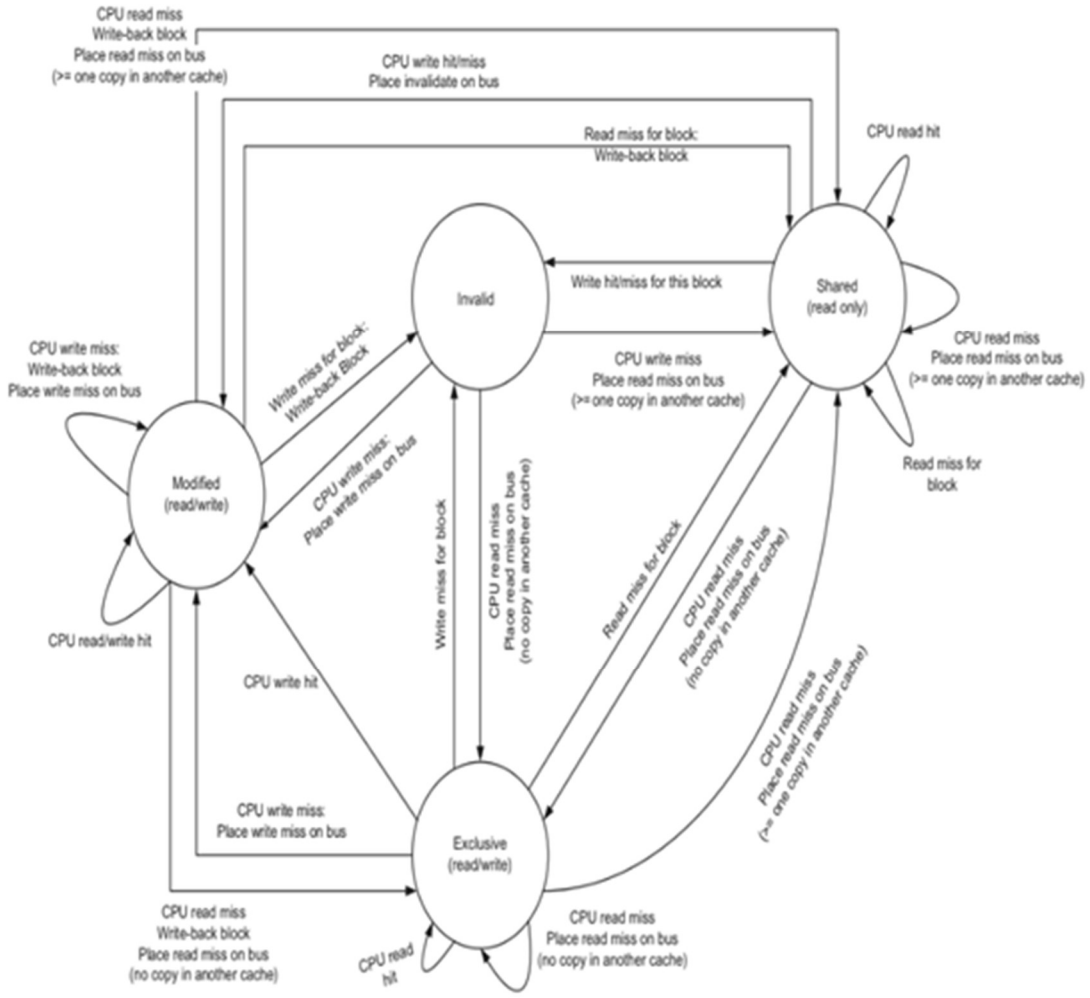
a. C0: R AC20 Read miss, satisfied by memory
C0: R AC28 Read miss, satisfied by C1's cache
C0: R AC20 Read miss, satisfied by memory, write-back 110
Implementation 1: 100 + 40+ 10 + 100 + 10 = 260 stall cycles
Implementation 2: 100 + 130 + 10 + 100 + 10 = 350 stall cycles

b. C0: R AC00 Read miss, satisfied by memory
C0: W AC08 ← 48 Write hit, sends invalidate
C0: W AC20 ← 78 Write miss, satisfied by memory, write back 110
Implementation 1: 100 + 15 + 10 + 100 = 225 stall cycles
Implementation 2: 100 + 15 + 10 + 100 = 225 stall cycles

c. C1: R AC20 Read miss, satisfied by memory
C1: R AC28 Read hit
C1: R AC20 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 AC28
C1: W AC20 ← 78 Write miss, satisfied by memory
Implementation 1: 100 + 100 + 10 + 100 = 310 stall cycles
Implementation 2: 100 + 100 + 10 + 100 = 310 stall cycles

5.3



5.4 a. C0: R AC00, Read miss, satisfied in memory, no sharers MSI: S, MESI: E
C0: W AC00 ← 40 MSI: send invalidate, MESI: silent transition from E to M
MSI: 100 + 15 = 115 stall cycles
MESI: 100 + 0 = 100 stall cycles

b. C0: R AC20, Read miss, satisfied in memory, sharers both to S
C0: W AC20 ← 60 both send invalidates
Both: 100 + 15 = 115 stall cycles

c. C0: R AC00, Read miss, satisfied in memory, no sharers MSI: S, MESI: E
C0: R AC20, Read miss, memory, silently replace 120 from S or E
Both: 100 + 100 = 200 stall cycles, silent replacement from E

d. C0: R AC00, Read miss, satisfied in memory, no sharers MSI: S, MESI: E
C1: W AC00 ← 60, Write miss, satisfied in memory regardless of protocol
Both: 100 + 100 = 200 stall cycles, don't supply data in E state (some protocols do)

e. C0: R AC00, Read miss, satisfied in memory, no sharers MSI: S, MESI: E
C0: W AC00 ← 60, MSI: send invalidate, MESI: silent transition from E to M
C1: W AC00 ← 40, Write miss, C0's cache, write-back data to memory
MSI: 100 + 15 + 40 + 10 = 165 stall cycles
MESI: 100 + 0 + 40 + 10 = 150 stall cycles

5.16

P1:	P2:
A = 1;	B = 1;
A = 2;	While (A <> 1);
While (B == 0);	B = 2;

Without an optimizing compiler the threads, SC will allow different orderings. Depending on the relative speeds of P1 and P2, "While (A <> 1);" may be legitimately executed

a. Zero times:
   B = 1; → A = 1;→ While (A <> 1);→ B = 2; → A = 2; While (B == 0);
   B will be set to 2

b. Infinite number of times:
   B = 1; → A = 1;→ A = 2; →While (A <> 1); ……
   B will be set to 1

c. A few times (A is initially 0)
   B = 1; → While (A <> 1);→A = 1;→ B = 2;→ A = 2; ….
   B will be set to 2

An optimizing compiler might decide that the assignment "A = 1;" is extraneous (because A is not read between the two assignments writing to it) and remove it. In that case, "while A .." will loop forever.

a.

The general form for Amdahl's Law is

$$\text{Speedup} = \frac{Execution\ time_{old}}{Execution\ time_{new}} = T/t$$

To compute the formula for speedup we need to derive the new execution time. The exercise states that for the portion of the original execution time that can use i processors is given by

$F(i, p)$. The time for running the application on p processors is given by summing the times required for each portion of the execution time that can be sped up using i processors, where i is between one and p. This yields

$$t = T * \sum_{i=1}^{p} \frac{f(i,p)}{i}$$

The new speedup formula is then $1/\sum_{i=1}^{p} \frac{f(i,p)}{i}$.

b.

New run time for 8 processors = T(0.2/1 + 0.2/2 + 0.1/4 + 0.05/6 + 0.45/8) = 0.39 *T (speedup = 2.57)

c.

New run time for 32 processors = T(0.2/1 + 0.2/2 + 0.1/4 + 0.05/6 + 0.15/8 + 0.2/16 + 0.1/32) = 0.369 *T (speedup = 2.72)

(c) New runtime for infinite processors = T(0.2/1 + 0.2/2+ 0.1/4 + 0.05/6 + 0.15/8 + 0.2/16 + 0.1/128) = 0.365 *T (speedup = 2.74)

5.22

i. 64 processors arranged as a ring→largest number of communication hops = 32
   100+10x32=420 ns
ii. 64 processors arranged as a 8x8 grid→largest number of communication hops = 14
   100+10x14=240 ns
iii. 64 processors arranged as a hypercube→largest number of communication hops = 6 (log64)
   100+10x6=160 ns

b.

i. Worst case CPI = 0.75+0.2/100x(420)x**2.0 = 2.43**
ii. Worst case CPI = 0.75+0.2/100x(240)x**2.0 = 1.71**
iii. Worst case CPI = 0.75+0.2/100x(160)x**2.0 = 1.39**

5.23

To keep the figures from becoming cluttered, the coherence protocol is split into two parts. Figure S.3 presents the CPU portion of the coherence protocol, and Figure S.4 presents the bus portion of the protocol.

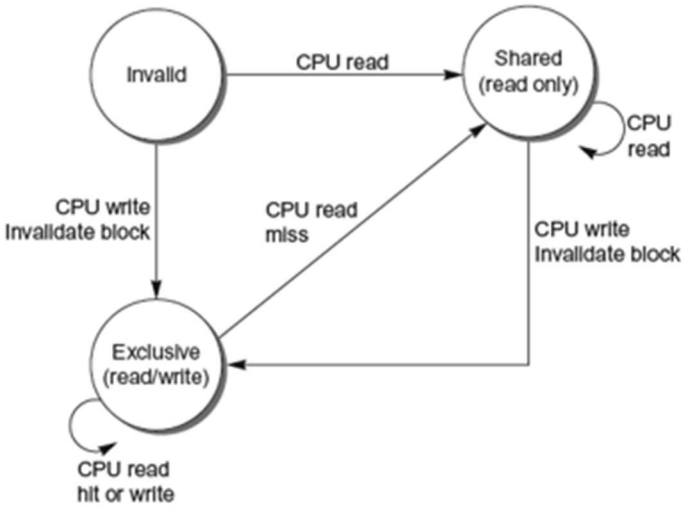


Figure S.3 CPU portion of the simple cache coherency protocol for write-through caches.

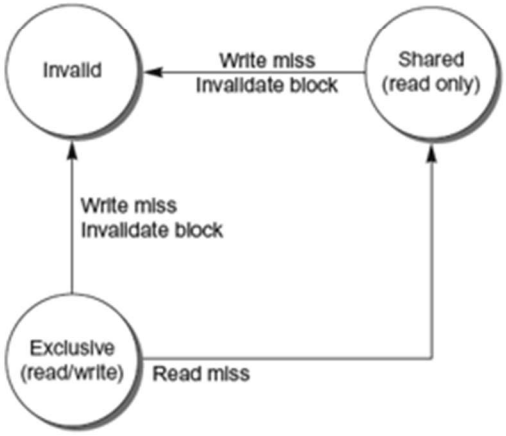


Figure S.4 Bus portion of the simple cache coherency protocol for write-through caches.

The major change introduced in moving from a write-back to write-through cache is the elimination of the need to access dirty blocks in another processor's caches. As a result, in the write-through protocol it is no longer necessary to provide the hardware to force write back on read accesses or to abort pending memory accesses. As memory is updated during any write on a write-through cache, a processor that generates a read miss will always retrieve the correct information from memory. It is not possible for valid cache blocks to be incoherent with respect to main memory in a system with write-through caches.