1. Read request from L1 data.
   1. CheckforHit
      1. Hit
         1. Modified
            1. ReadL2
            2. UpdateLRU
            3. MESI - Stays M
            4. WriteL1
         2. Exclusive
            1. ReadL2
            2. UpdateLRU
            3. MESI - Stays E
            4. WriteL1
         3. Shared
            1. ReadL2
            2. UpdateLRU
            3. MESI – Stays S
            4. WriteL1
         4. Invalid (shouldn’t have a hit then an invalid but must have a fail-safe)
            1. readSharedBus: read

HIT/HITM

WriteL2

UpdateLRU

MESI – S

writeL1

MISS (HIT')

readSharedBus: RFO

writeL2

updateLRU

MESI – E

writeL1

* + 1. Miss
       1. readSharedBus: read
          1. HIT/HITM from other processor

writeL2

updateLRU

MESI – S

writeL1

* + - * 1. MISS from other processor

readSharedBus: RFO

writeL2

updateLRU

MESI - E

writeL1

1. Write request from L1 data
   1. CheckforHit
      1. Hit
         1. Modified
            1. writeL2
            2. updateLRU
            3. MESI – Stays M
         2. Exclusive
            1. sendInvalidate
            2. writeL2
            3. updateLRU
            4. MESI – M
         3. Shared
            1. sendInvalidate
            2. writeL2
            3. updateLRU
            4. MESI – M
         4. Invalid
            1. readSharedBus:read

HIT (other processor has it in E/S state)

writeL2

updateLRU

MESI - M

HITM (other processor has it in M state)

readSharedBus: RFO

writeL2

updateLRU

MESI - M

MISS (no other processor has it)

writeL2

updateLRU

MESI - M

* + 1. Miss
       1. readSharedBus:read
          1. HIT (other processor has it in E/S state)

writeL2

updateLRU

MESI - M

* + - * 1. HITM (other processor has it in M state)

readSharedBus: RFO

writeL2

updateLRU

MESI - M

* + - * 1. MISS (no other processor has it)

writeL2

updateLRU

MESI - M

1. Read request from L1 instruction
   1. CheckforHit
      1. Hit
         1. Modified (shouldn’t happen but must have something as fail safe)
            1. readL2
            2. updateLRU
            3. MESI – M
            4. writeL1
         2. Exclusive
            1. readL2
            2. updateLRU
            3. MESI - Stays E
            4. writeL1
         3. Shared
            1. readL2
            2. updateLRU
            3. MESI – Stays S
            4. writeL1
         4. Invalid
            1. readSharedBus: read

HIT

writeL2

updateLRU

MESI – S

writeL1

HITM

readSharedBus: RFO

writeL2

updateLRU

MESI – E

writeL1

MISS (HIT')

writeL2

updateLRU

MESI – E

writeL1

* + 1. Miss
       1. readSharedBus: read
          1. HIT (other processor has in E/S state)

writeL2

updateLRU

MESI – S

writeL1

* + - * 1. HITM (other processor has in M state)

writeL2

updateLRU

MESI – S

writeL1

* + - * 1. MISS (no other processor has it)

readSharedBus: RFO

writeL2

updateLRU

MESI – E

writeL1

1. Snooped invalidate command (Another processor is modifying the data)
   1. CheckforHit
      1. HIT/THIM (we have it)
         1. Modified
            1. PutSnoopResult
            2. writeSharedBus
            3. invalidateL2
         2. Exclusive
            1. invalidateL2
         3. Shared
            1. invalidateL2
         4. Invalid (should mean a miss)
            1. Do nothing
      2. MISS (we don’t have it)
         1. Do nothing
2. Snooped read request (another processor is trying to read)
   1. CheckforHit
      1. HIT/HITM (we have it)
         1. Modified
            1. PutSnoopResult
            2. WriteSharedBus
            3. updateLRU
            4. MESI – S
         2. Exclusive
            1. PutSnoopResult
            2. updateLRU
            3. MESI – S
         3. Shared
            1. PutSnoopResult
            2. updateLRU
            3. MESI – Stay S
         4. Invalid (should mean a miss)
            1. Do nothing
      2. MISS
         1. PutSnoopResult: MISS
            1. Do nothing
3. Snooped write request (another processor is trying to write)
   1. CheckforHit
      * 1. HIT/HITM (we have it)
           1. Modified

PutSnoopResult

WriteSharedBus

MESI: I

* + - * 1. Exclusive

MESI: I

* + - * 1. Shared

MESI: I

* + - * 1. Invalid

Do nothing

* + 1. MISS
       1. Do nothing

1. Snooped read with intent to modify(another processor has ownership and we are snooping)
   1. CheckforHit
      1. HIT/HITM
         1. Modified
            1. PutSnoopResult
            2. WriteSharedBus
            3. MESI – I
      2. Exclusive
         1. MESI – I
      3. Share
         1. MESI – I
      4. Invalid
         1. Do nothing
   2. MISS
      1. Do nothing
2. Clear and reset all
   1. Invalidate all memory locations. It is not necessary to clear the memory locations.
3. Print contents and state of each valid line
   1. Print contents and states of all(only) valid lines

Notes:

1. Read For Ownership (RFO) – Do a memory read but indicate to snooping processors on FSB our intent to write that memory location (slide 62)
2. Invalidate – “Invalidate your copy and I have the only valid one now.” Address only transaction (i.e.: Puts address on the FSB but doesn’t request data from memory)
3. IMPORTANT: Slide 68
   1. Constraints are met
   2. If miss in both L1 and L2, line is placed in both caches
      1. If a dirty line is evicted from L1 it will be held in L2 (IP).
      2. If a dirty line is evicted from L2 it must also be evicted from L1 (IP)
   3. Miss to L1 and hit to L2
      1. Due to a line being evicted from L1 where it is written back to L2 (IP)
   4. On a snoop hit to L2, L2 instructs L1 to invalidate the line in its cache
      1. Minimize invalidates by keeping single bit/line in L2 to indicate in L1
   5. Relaxed constraints
      1. L2 line size = C \* L1 line size (where C >= 1)
      2. Need additional inclusion bits if C > 1
      3. Associativity L2 / L2 line size >= Associativity L1 / L1 line size
4. IMPORTANT: Slide 65
   1. Has a MESI FSM that was used by PowerPC that incorporates the MESI protocol and the required functions for all cases into the one FSM. FSM and notes found below.

