Quiz 2 Review

Atalay Mert Ileri & Miguel Gomez

(Adapted from prior course offerings)

Quiz 2 logistics

- Time: 1pm EST on Wednesday, November 9
- Location: 32-141
- Handout posted on website/Piazza

Topics

- Advanced memory operations
- Multithreading
- On-chip Networks
  - Topology
  - Routing
  - Flow control
  - Router micro-architecture
- Cache coherence
  - Snooping-based vs. Directory-based
  - VI, MSI, MESI, MOSI, …
  - Transient states
  - Synchronization primitives
- Memory consistency model
  - Sequential consistency
  - Total Store Order (TSO)
  - Relaxed consistency

Advanced memory operations

- Write policy
  - Hits: write through vs. write back
  - Misses: write allocate vs. write no allocate

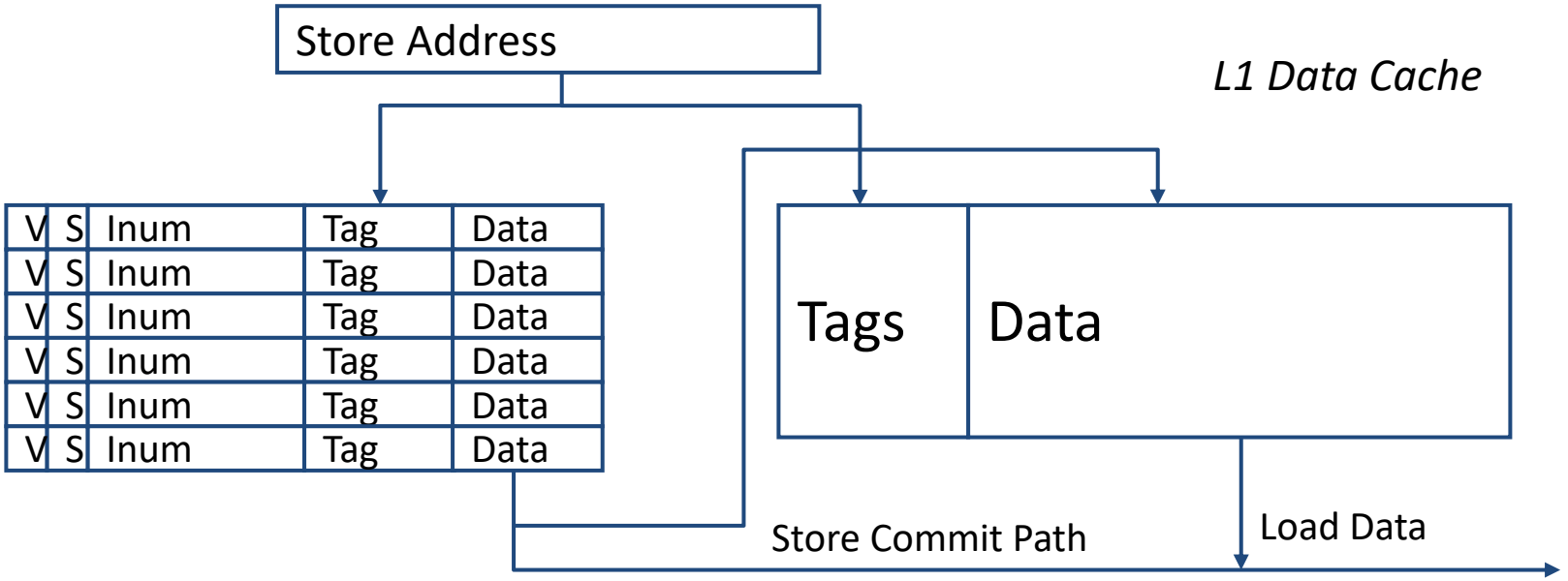
- Speculative loads/stores
  - Cause 1: control dependency: All instructions are speculative until commit
    - Just like other instructions
    - Solution: buffer the stores and commit them in order
  - Cause 2: (memory-location-based) data dependency
    - Simple solution: buffer stores; loads search addresses of all previous stores
    - Problem: addresses of previous stores may be unknown
    - Solution: speculate no data dependency
      - Use a data structure to keep track of this speculation: speculative load buffer

Store Buffer

» Enables data forwarding

» Handles OoO stores

» Handles speculative stores



» On store execute:

- mark valid and speculative; save tag, data and instruction number.

» On store commit:

- clear speculative bit and eventually move data to cache

» On store abort:

- clear valid bit

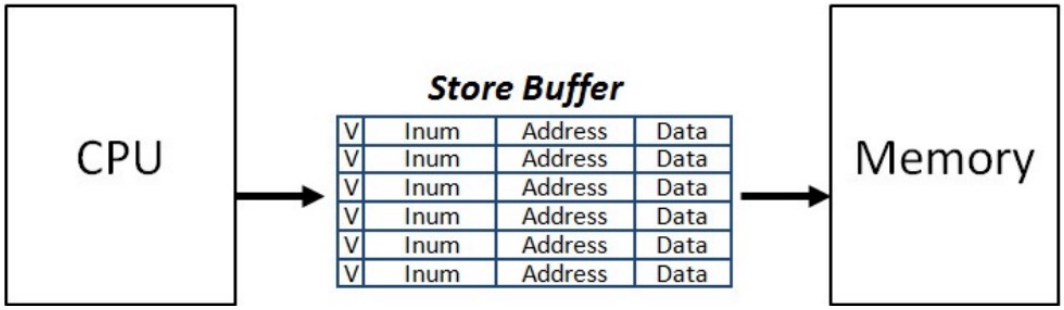
» One entry per store

» Written by stores

» Searched by loads

» Writes to data cache

Ben Bitdiddle designed an out-of-order vector machine with store buffers. This machine executes memory operations (both load and store) "in order", although other instructions can be executed out-of-order. (All the instructions are committed "in order".) There is a load/store issue queue to maintain the execution order of all the memory operations. Every load must check if the value it needs is in the store buffer, and to determine the proper value, every instruction is assigned a unique instruction number (Inum).

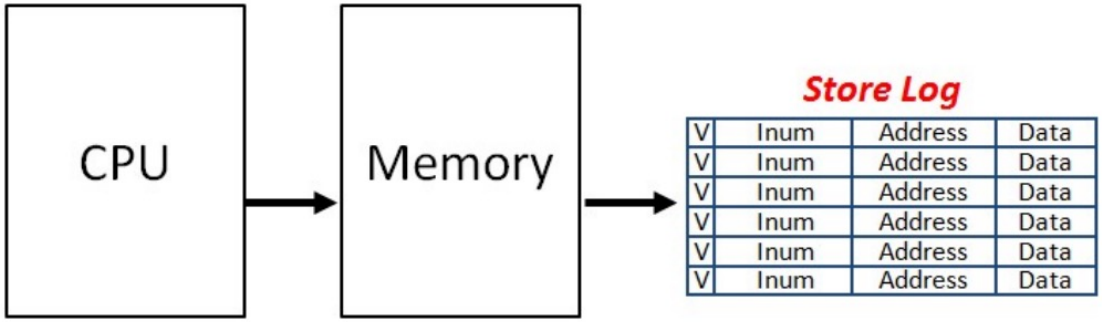


Problem M9.2.A

Suppose 10% of instructions are stores and the average lifetime of instructions in the store buffer is 100 cycles. Assuming the desired throughput of this machine to be 1 instruction per cycle, how many entries will the store buffer be holding at a given time on average?

Does this machine store into memory greedily or lazily?

Ben did not like the store buffers in the previous design, because the store buffers need to be looked up for every load instruction, and implementing a fast lookup to the buffers was too expensive. Thus, instead of using the store buffers, Ben decided to directly update the memory during execution (before the store instruction actually commits), and keep the old values in "**store logs**". **Each entry in the store logs consists of a valid bit, Inum, memory address and data value**. The data field holds the value that was in the memory before the store to the location writes to the memory.



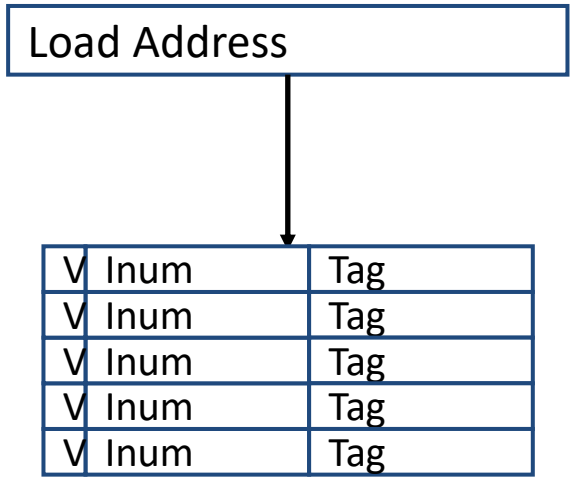
Is this a greedy update or a lazy update?

Assume an arithmetic exception occurred, and thus, the processor state and memory need to be recovered appropriately. Which machine design (store buffer machine or store log machine) has the higher recovery cost, and why?

Load Buffer

» On load execute:
  - mark entry valid, and instruction number and tag of data.
» On load commit:
  - clear valid bit
» On load abort:
  - clear valid bit

Speculative Load Buffer



» One entry per load
» Written by loads
» Searched by stores

» Enables aggressive load scheduling
» Detects ordering violations

Multithreading

- Fine-grain multithreading
- Coarse-grain multithreading
- Simultaneous multithreading
  - Scheduling policies
    - Round-robin: Equalize *throughput* between threads
    - ICOUNT: Equalize *instr. in flight* between threads

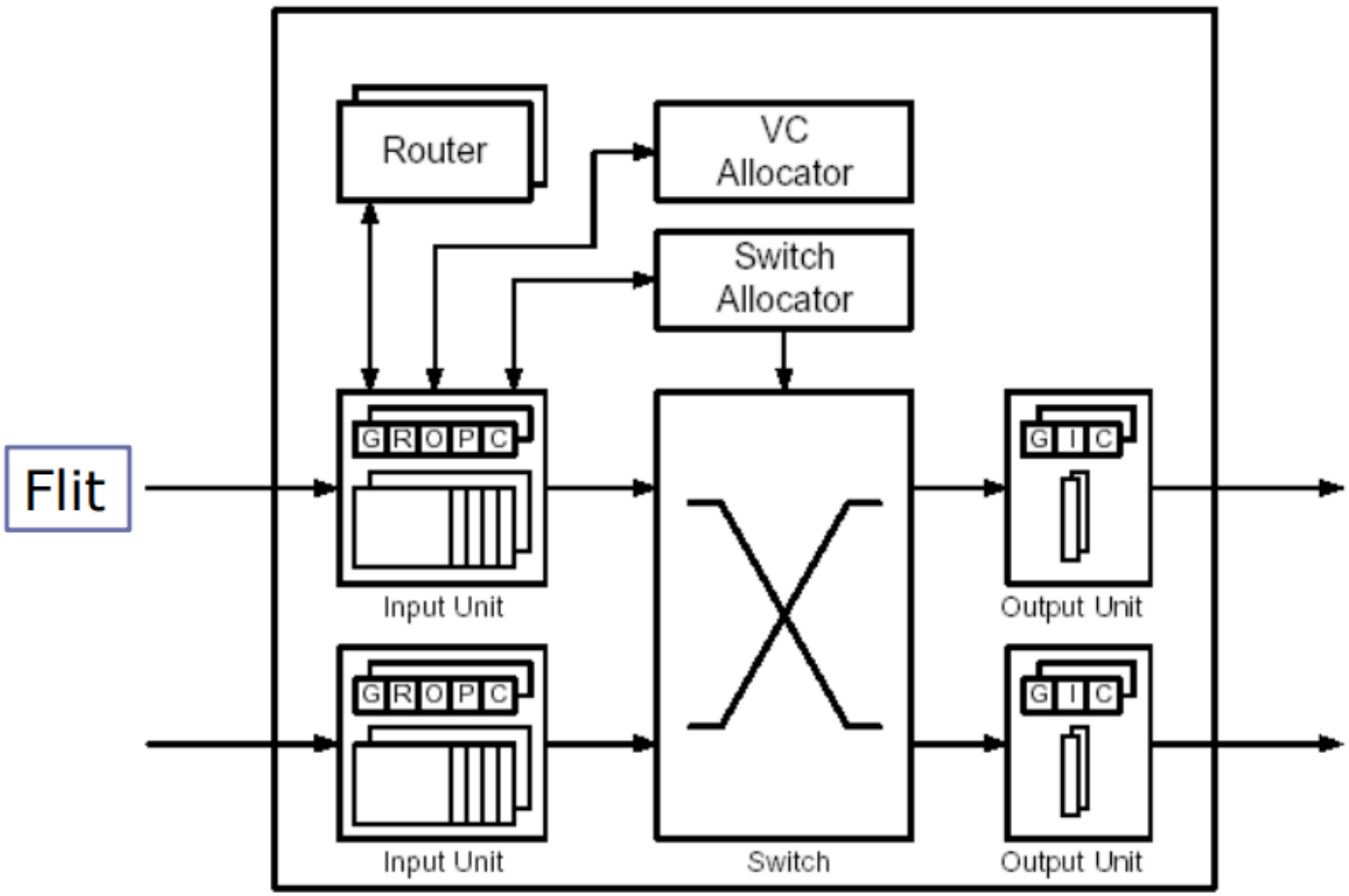
On-chip networks

- Allow sharing communication resource

- Topology
  - Metrics: routing distance, diameter, average distance, bisection bandwidth, …
- Routing
  - Properties: deterministic, adaptive, deadlock-free, …

On-chip networks

- Flow control
  - Bufferless
    - Circuit switching, dropping, misrouting, …
  - Buffered
    - Store-and-forward, virtual cut-through, wormhole, virtual channel
- Router architecture


Router
VC Allocator
Switch Allocator
Flit
G R O P C
Input Unit
G R O P C
Input Unit
Switch
G I C
Output Unit
G I C
Output Unit

Problem M12.6.A

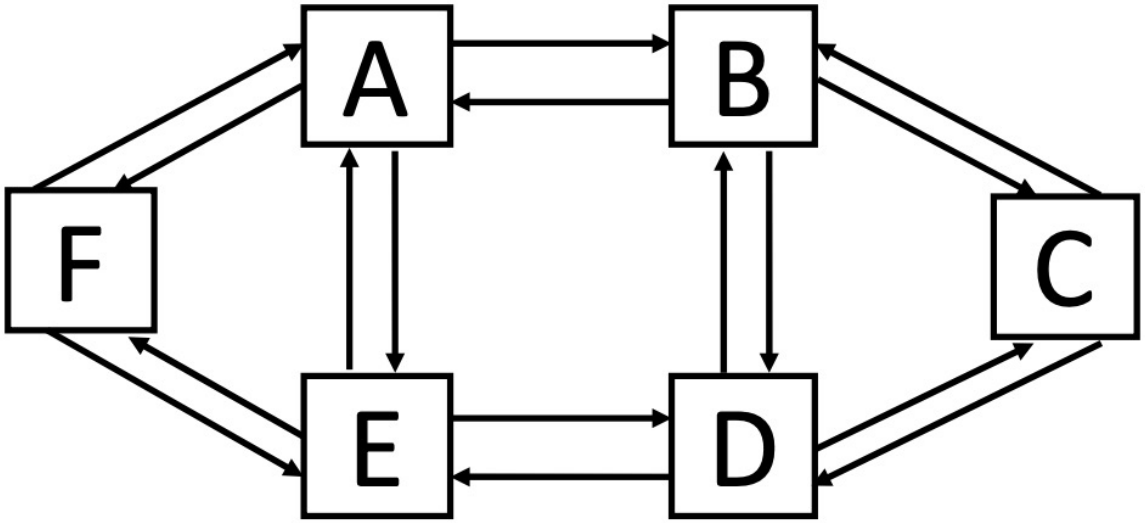
Determine whether the following routing algorithms are deadlock-free for a 2D-mesh. State your reasoning.

a) (3 points) Randomized dimension-order: All packets are routed minimally. Half of the packets are routed completely in the X dimension before the Y dimension, and the other packets are routed in the Y dimension before the X dimension.

b) (3 points) Less randomized dimension-order: All packets are routed minimally. Packets whose minimal direction is increasing in both X and Y always route **X before Y**. Packets whose minimal direction is decreasing in both X and Y always route **Y before X**. All other packets choose randomly between X before Y and vice-versa.

Problem M12.6.B

Consider the following topology:



(a) (2 points) What is the diameter of this topology?

(b) (2 points) What is the bisection bandwidth (in flits/cycle) of this topology?

Cache coherence

- Simplify building shared memory systems

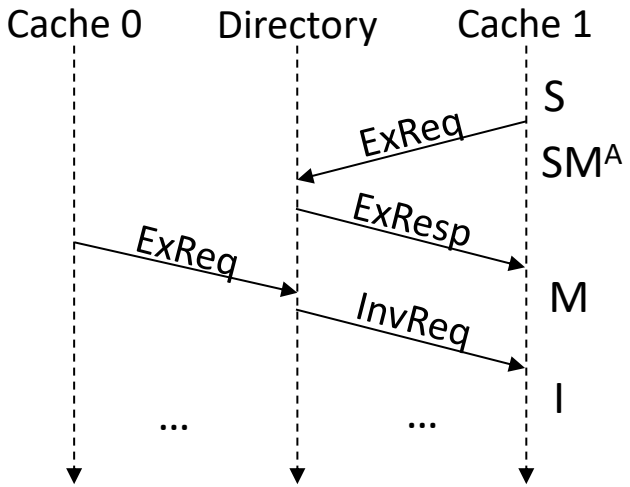
- Definition:
  - Write propagation — Liveness: do something good
    - Writes eventually become visible to all processors
  - Write serialization — Safety: don't do anything bad
    - Writes to the same location are serialized (all processors see them in the same order)

Cache coherence

- Transient states: required by lack of atomicity
  - Two types
    - Split states: to implement one transaction
      - E.g., S transitions to SM^A (instead of M), waiting for an ExResp ("A" denotes acknowledgement)
    - Race states: to handle overlaps of two transactions
      - Not all such overlaps require transient states
      - See the following examples

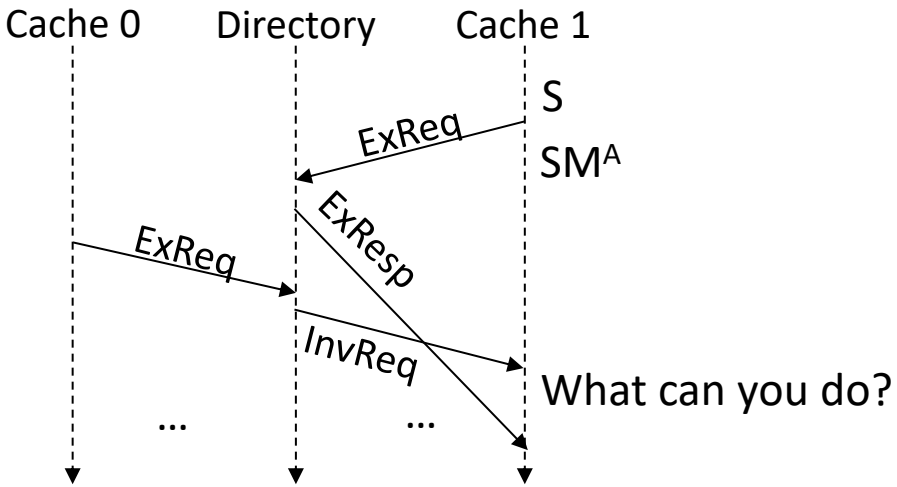
Cache coherence

- Split example
  - SM^A



Cache coherence

- Race example



If the arriving message is from a younger transaction:

- Either defers processing it
- Or handles it immediately and transitions to a race state (e.g., SM^AI)

Memory (consistency) model

- Concerns reads/writes to multiple memory locations
- Interacts with many parts and optimizations of the system
  - Probably more than what you would have imagined…
- Coherence is an useful (but not necessary) building block
  - Recall: Coherence guarantees writes are visible in some global order.

Sequential consistency

- Definition
  - "The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program"
  - Arbitrary order-preserving interleaving of memory references of sequential programs
- Implementation
  - In-order instruction execution + atomic loads and stores
- Advantage: easy to understand
- Disadvantage: limits performance
  - Uniprocessor optimizations often violate them!
    - E.g., committed store buffers, non-blocking caches, speculative execution, memory address speculation, …

Total Store Order (TSO)

- Allows loads to go ahead of stores waiting in the store buffer

- Implementation
  - Sequential consistency implementation + per-core FIFO store buffer with store-load bypassing

Relaxed memory consistency

- Allows more reordering
  - Store-load
  - Store-store
  - Load-load
  - Load-store

- Re-ordering can be disabled by fences/barriers

Tips on consistency problems

- Keep definitions in mind
- Think systematically
  - E.g., For questions asking all allowed execution results: search invariants to minimize brute-force search
  - E.g., For questions asking to add minimal barriers/fences: find the precise reordering that violates the target model

Wish you all the best!