Memory Data Flow Techniques

Load/store buffer design, memory-level parallelism, consistency model, memory disambiguation
Load/Store Execution Steps

Load: LW R2, 0(R1)

1. Generate virtual address; may wait on base register
2. Translate virtual address into physical address
3. Write data cache

Store: SW R2, 0(R1)

1. Generate virtual address; may wait on base register and data register
2. Translate virtual address into physical address
3. Write data cache

Unlike in register accesses, memory addresses are not known prior to execution
Load/store Buffer in Tomasulo

Support memory-level parallelism

Loads wait in load buffer until their address is ready; memory reads are then processed

Stores wait in store buffer until their address and data are ready; memory writes wait further until stores are committed
Load/store Unit with Centralized RS

Centralized RS includes part of load/store buffer in Tomasulo

Loads and stores wait in RS until there are ready
Memory-level Parallelism

for (i=0; i<100; i++)

Loop:
    L.S F2, 0(R1)
    MULT F2, F2, F4
    SW F2, 0(R1)
    ADD R1, R1, 4
    BNE R1, R3, Loop

F4 store 2.0

Significant improvement from sequential reads/writes
Memory Consistency

Memory contents must be the same as by sequential execution
Must respect RAW, WRW, and WAR dependences

Practical implementations:
1. Reads may proceed out-of-order
2. Writes proceed to memory in program order
3. Reads may bypass earlier writes only if their addresses are different
Store Stages in Dynamic Execution

1. Wait in RS until base address and store data are available (ready)
2. Move to store unit for address calculation and address translation
3. Move to store buffer (finished)
4. Wait for ROB commit (completed/committed)
5. Write to data cache

Stores always retire in for WAW and WRA Dep.

Source: Shen and Lipasti, page 197
Load Bypassing and Memory Disambiguation

To exploit memory parallelism, loads have to bypass writes; but this may violate RAW dependences.

Dynamic Memory Disambiguation:
Dynamic detection of memory dependences

- Compare load address with every older store addresses
Load Bypassing Implementation

1. address calc.
2. address trans.
3. if no match, update dest reg

Associative search for matching

Assume in-order execution of load/stores
Load Forwarding

Load Forwarding: if a load address matches an older write address, can forward data

If a match is found, forward the related data to dest register (in ROB)
Multiple matches may exist; last one wins
In-order Issue Limitation

for (i=0; i<100; i++)
  A[i] = A[i]/2;

Loop:
  L.S F2, 0(R1)
  DIV F2, F2, F4
  SW F2, 0(R1)
  ADD R1, R1, 4
  BNE R1, R3, Loop

Any store in RS station may blocks all following loads

❄ When is F2 of SW available?

❄ When is the next L.S ready?

Assume reasonable FU latency and pipeline length
Speculative Load Execution

Forwarding does not always work if some addresses are unknown

No match: predict a load has no RAW on older stores

Flush pipeline at commit if predicted wrong
Alpha 21264 Pipeline
Alpha 21264 Load/Store Queues

32-entry load queue, 32-entry store queue
Load Bypassing, Forwarding, and RAW Detection

- **LQ** (Load Queue)
  - If match: forwarding
  - 
- **D-cache**
  - If match: mark store-load trap to flush pipeline (at commit)
  - 
- **SQ** (Store Queue)
  - completed
  - 
- **ROB** (Ready Queue)
  - Load/store?
    - Load: WAIT if LQ head not completed, then move LQ head
    - Store: mark SQ head as completed, then move SQ head
  - 

**Commit**

Speculative Memory Disambiguation

Load forwarding

- When a load is trapped at commit, set stWait bit in the table, indexed by the load's PC
- When the load is fetched, get its stWait from the table
- The load waits in issue queue until old stores are issued
- stWait table is cleared periodically
Architectural Memory States

Memory request: search the hierarchy from top to bottom
Summary of Superscalar Execution

Instruction flow techniques
    Branch prediction, branch target prediction, and instruction prefetch

Register data flow techniques
    Register renaming, instruction scheduling, in-order commit, mis-prediction recovery

Memory data flow techniques
    Load/store units, memory consistency

Source: Shen & Lipasti