Lecture 11 Memory Data Flow Techniques

CprE 581 Computer Systems Architecture

Readings

Readings

- Textbook pages 164-165
- Reference "Modern Processor Design" Sec. 5.3, (3)
- The Alpha 21264 Processor, IEEE Micro, No. 2, 1999

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. Read 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

Memory Data Flow

Unlike in register indices, memory addresses are not known until the end of execution

DIV.D F10, F12, F14
SW F10, 100(R8)
LW F16, 200(R10) ⇨ Dependent or not?
ADD F16, F16, F18

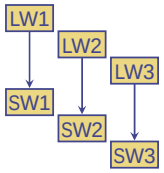
When should LW be executed?

Memory-level Parallelism

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

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

Note: F4 contains value 2



Significant performance gain if they can be executed in parallel

But what happens if there is a dependency?

Memory Dependences

Memory contents must be the same as in sequential execution

Hardware must handle data and name dependences correctly

Key differences with register dependences

1. Dependences are not known until memory addresses are calculated
2. Both data and name dependences are generally rare cases within a small instruction window

Memory Store: Correctness and Performance Considerations

Memory stores should proceed to memory in program order

- Only non-speculative store instructions may write to memory
- WAR and WAW dependences are not concerns

Memory Store: Correctness and Performance Considerations

Writes should be buffered to avoid memory bottleneck

- Cache memory has limited access ports
- Non-memory instructions won't wait for memory stores
- Memory dependences are ,in general, rare
- Memory loads are performance critical – they should have higher priority in memory access

Correctness and Performance Regarding Memory Loads

Memory loads are performance-critical

1. Reads may proceed to memory out-of-order
2. A read may bypass earlier writes if their addresses are different
3. A read may receive data forwarded from buffered writes if their address are identical

On the other hand, a speculative load with wrong data can be tolerated in Tomasulo scheduling.

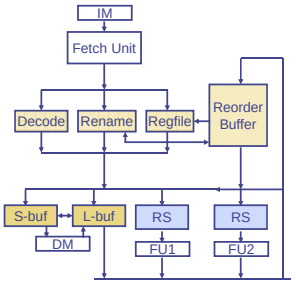
9

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

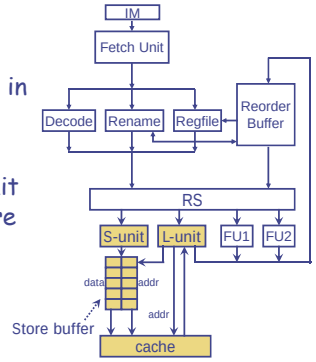


10

Load/store Unit with Centralized RS

Centralized RS includes part of load/store buffer in Tomasulo

Loads and stores wait in RS until they are ready

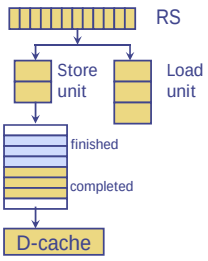


11

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.

This is a combination of *store buffer/queue* and *write buffer*



Source: Shen and Lipasti, page 197

12

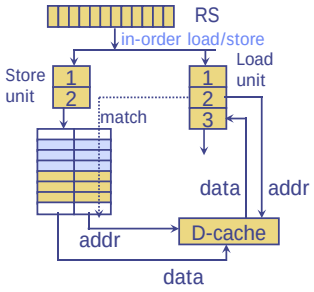
Load Bypassing and Memory Disambiguation

To exploit memory parallelism, loads may run ahead of earlier writes; but this may violate RAW dependences

Dynamic Memory Disambiguation (Memory Alias Analysis): Dynamic detection of memory dependences

- Compare load address with every older store address

Load Bypassing

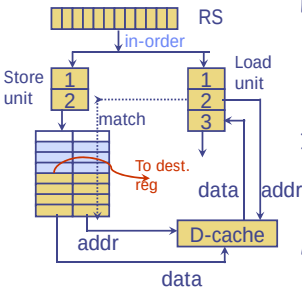


1. address calc.
2. address trans.
3. if no match, update dest reg

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

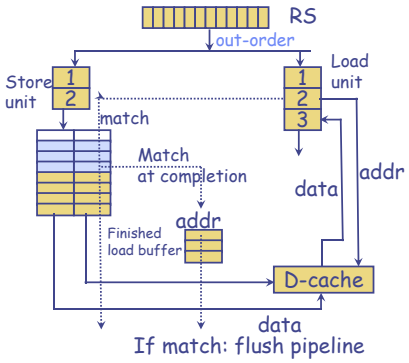
Now a load doesn't have to wait for stores to be committed and written to cache

However, loads and stores are serialized at inst scheduling.

```
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
```

Speculative Load Execution

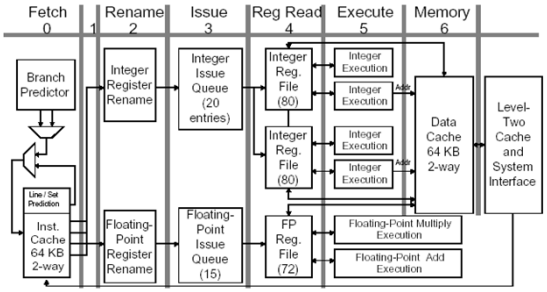


Forwarding does not always work if some addresses are unknown

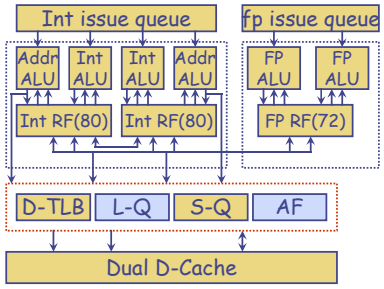
No match: predict a load has no RAW on older stores

Flush pipeline at commit if predicted wrong

Alpha 21264 Pipeline

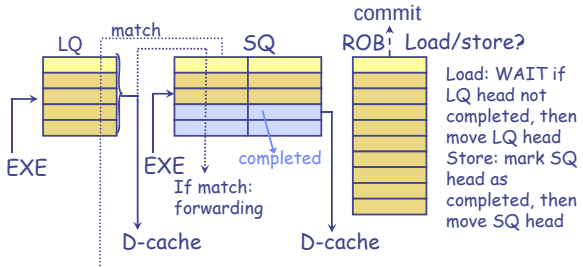


Alpha 21264 Load/Store Queues



32-entry load queue, 32-entry store queue

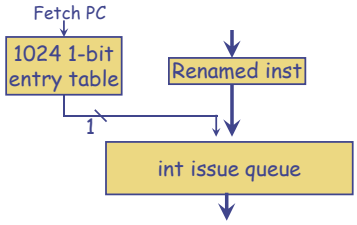
Load Bypassing, Forwarding, and RAW Detection



If match (and if the load was executed out-of-order): mark store-load trap to flush pipeline (at commit)

Speculative Memory Disambiguation

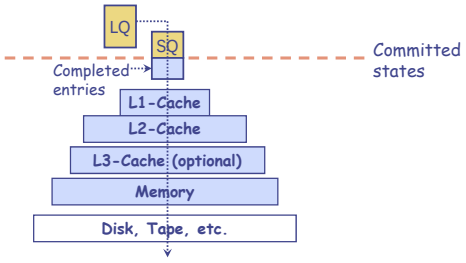
Load forwarding



- 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

21

Architectural Memory States



Memory request: search the hierarchy from top to bottom

22

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

23