Lecture 10: Memory Dependence Detection and Speculation

Memory correctness, dynamic memory disambiguation, speculative disambiguation, Alpha 21264 Example

Register and Memory Dependences

Store: SW Rt, A(Rs)

1. Calculate effective memory address ⇒ dependent on Rs
2. Write to D-Cache ⇒ dependent on Rt, and cannot be speculative

Compare "ADD Rd, Rs, Rt"
What is the difference?

LW Rt, A(Rs)

1. Calculate effective memory address ⇒ dependent on Rs
2. Read D-Cache ⇒ could be memory-dependent on pending writes!

When is the memory dependence known?

Memory Correctness and Performance

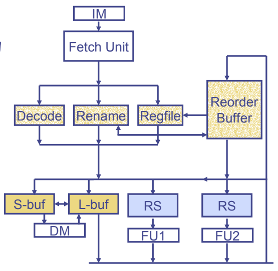
Correctness conditions:

- Only committed store instructions can write to memory
- Any load instruction receives its memory operand from its parent (a store instruction)
- At the end of execution, any memory word receives the value of the last write

Performance: Exploit memory level parallelism

Load/store Buffer in Tomasulo

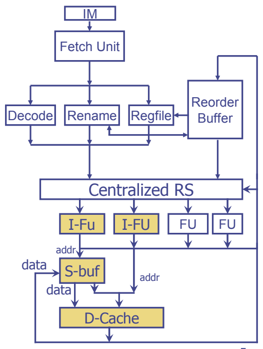
- Original Tomasulo: Load/store address are pre-calculated before scheduling
- Loads are not dependent on other instructions
- Stores are dependent on instructions producing the store data
- Provide dynamic memory disambiguation: check the memory dependence between stores and loads



Dynamic Scheduling with Integer Instructions

Centralized design example:

- Centralized reservation stations usually include the load buffer
- Integer units are shared by load/store and ALU instructions
- What is the challenge in detecting memory dependence?



Load/Store with Dynamic Execution

- Only committed store instructions can write to memory

⇒ Use store buffer as a temporary place for write instruction output

- Any memory word receives the value of the last write

⇒ Store instructions write to memory in program order

- Any memory word receives the value of the last write
- Memory level parallelism be exploited

⇒ Non-speculative solution: load bypassing and load forwarding

⇒ Speculative solution: speculative load execution

Store Buffer Design Example

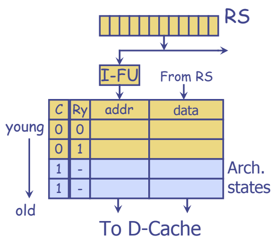
Store instruction:

- Wait in RS until the base address and data are ready
- Calculate address, move to store buffer
- Move data directly to store buffer
- Wait for commit

If no exception/mis-predict

5. Wait for memory port
6. Write to D-cache

Otherwise flushed before writing D-cache



Memory Dependence

Any load instruction receives the memory operand from its parent (a store instruction)

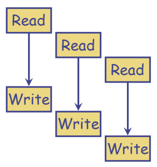
- If any previous store has not written the D-cache, what to do?
- If any previous store has not finished, what to do?

Simple Design: Delay all following loads; but how about performance?

Memory-level Parallelism

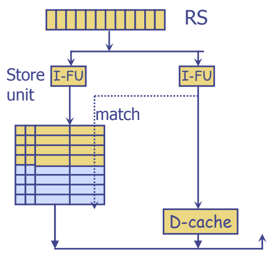
```
for (i=0;i<100;i++)
  A[i] = A[i]*2;

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

F4 store 2.0



Significant improvement from sequential reads/writes

Load Bypassing and Load Forwarding



Non-speculative solution

- Dynamic Disambiguation: Match the load address with all store addresses
- Load bypassing: start cache read if no match is found
- Load forwarding: using store buffer value if a match is found
- In-order execution limitation: must wait until all previous store have finished

In-order Execution Limitation

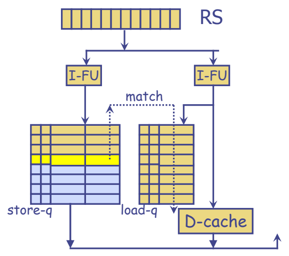
```
Example 1:
for (i=0;i<100;i++)
  A[i] = A[i]/2;
Loop:L.S F2, 0(R1)
     DIV F2, F2, F4
     SW F2, 0(R1)
     ADD R1, R1, 4
     BNE R1, R3,Loop
Example 2:
a->b->c = 100;
d = x;
```

Example 1: When is the SW result available, and when can the next load start?

Possible solution: start store address calculation early ⇒ more complex design

Example2: When is the address "a->b->c" available?

Speculative Load Execution

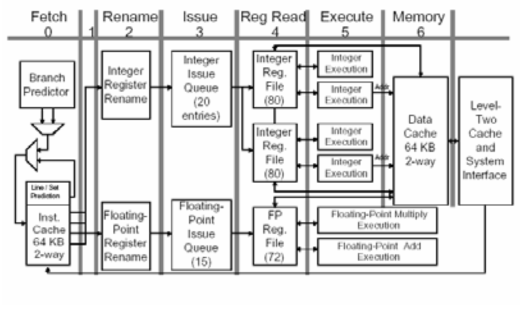


If no dependence predicted

- Send loads out even if dependence is unknown
- Do address matching at store commits
  1. Match found: memory dependence violation, flush pipeline;
  2. Otherwise: continue

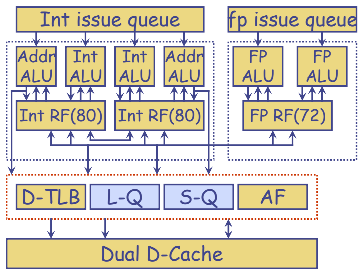
Note: may still need load forwarding (not shown)

Alpha 21264 Pipeline



13

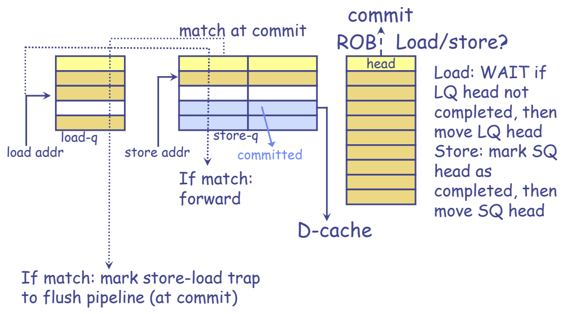
Alpha 21264 Load/Store Queues



32-entry load queue, 32-entry store queue

14

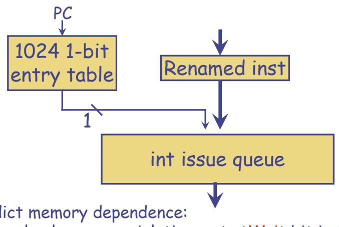
Load Bypassing, Forwarding, and RAW Detection



Load: WAIT if LQ head not completed, then move LQ head
Store: mark SQ head as completed, then move SQ head

15

Speculative Memory Disambiguation

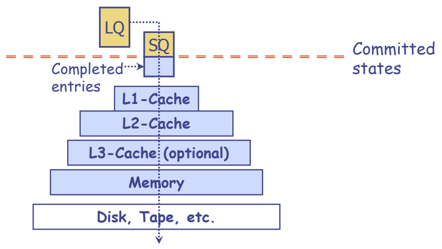


To help predict memory dependence:

- Whenever a load causes a violation, set stWait bit in the table
- When the load is fetched, get its stWait from the table, send to issue queue with the load instruction
- A load waits there if its swWait is set and any previous store exists
- The tale is cleared periodically

16

Architectural Memory States



Memory request: search the hierarchy from top to bottom

17

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 reference book

18