Memory Ordering

Out of order processing can lead to better performance, as long as the results of the program are correct. Which means ensuring the correct handling of dependencies.

- -Control dependencies are eliminated using Branch Prediction
- -False dependencies are eliminated using Register Renaming
- -Register dependencies are eliminated using Tomasulo-like scheduling

This lesson discusses a method for handling memory dependencies, the Load/Store Queue.

When Does Memory Write Happen

Writes to memory occur at Commit. If a memory write were to occur before this, it has the potential to be recalled - from an exception or misprediction.

The loads for memory should get the data as quickly as possible.

Load Store Queue Part 1

The Load-Store Queue has the following fields:

- -A bit for indicating a load or store
- -The address of the load or store
- -The data value to be stored in memory
- -A bit indicating completion

Loads and stores are placed in the queue in program order.

Store-to-Load Forwarding - When a load command is put into the queue, the address of the load is compared to all previous stores in the queue. If there is a match, this value in the queue is used, thus eliminating the need to go to memory.

When a store does not have its address when a load is checking the queue, the following can be done:

1. Wait for the store to get the address, which means perform the stores and loads in order. When the store gets the address, compare it to the address needed by the load and proceed from there. This method is slow and leads to unnecessary waiting.

2. Wait for all previous stores to get their addresses. If any of the stores have the same address as the load, then we know we have to wait for that store to complete. If there is no match, then go to memory to get the data.

3. The most aggressive method is to just go to memory to get the data. If there is not a previous store to that address, this is good. But if there is a previous store to the address, then we have just fetched the old data. A recover will have to be performed.

To recover the correct data must be obtained - the store instruction will now have the correct data. This data must be propagated to all loads that are accessing this memory address.
Modern processors use this method because it results in the best performance.

OOO Load/Store Execution
OOO Load/Store execution occurs when loads go to memory as soon as the address is available.
Sometimes this results in stale data being used and then it needs to be recovered.

In-Order Load/Store Execution
To solve the problems of OOO load/store execution, do the loads and stores in-order. A load cannot execute until all previous stores have their addresses. Then the load will check the stores, if there is a match in addresses, the load will wait for the completion of the store. If there is no match the load will go to memory.
All other instructions can complete out-of-order.

This method is not a high performance solution.

Store to Load Forwarding
Two questions that need to be answered:

1. A load instruction asks: Which earlier store has the required value?
2. A store instruction asks: Which later load(s) will need this value?

These answers can be found in the load/store queue (LSQ).
The values must be written to memory or data cache when the store is committed - not when it executes.

LSQ, ROB, RS
When issuing a Load/Store instruction, there must be a:
ROB entry available
LSQ entry available

When issuing a non-Load/Store instruction there must be a:
ROB entry available
RS available

When executing a Load/Store Instruction
-Compute the address
-Produce the value
-Write the results for Loads, which will broadcast it.

Commit the Load/Store

-Free the ROB entry
-Free the LSQ entry
-For the store - send the value to memory

When there is a Store then a Load to the same address - the data value is supplied by the LSQ.