Lecture: Cache Hierarchies

- Topics: cache access basics/examples
Out-of-Order Loads/Stores

What if the issue queue also had load/store instructions?
Can we continue executing instructions out-of-order?
Memory Dependence Checking

- The issue queue checks for *register dependences* and executes instructions as soon as registers are ready.

- Loads/stores access memory as well – must check for RAW, WAW, and WAR hazards for memory as well.

- Hence, first check for register dependences to compute effective addresses; then check for memory dependences.
### Memory Dependence Checking

- Load and store addresses are maintained in program order in the Load/Store Queue (LSQ).
- Loads can issue if they are guaranteed to not have true dependences with earlier stores.
- Stores can issue only if we are ready to modify memory (can not recover if an earlier instr raises an exception) – happens at commit.

<table>
<thead>
<tr>
<th>Ld</th>
<th>0x abcdef</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ld</td>
<td></td>
</tr>
<tr>
<td>St</td>
<td></td>
</tr>
<tr>
<td>Ld</td>
<td>0x abcdef</td>
</tr>
<tr>
<td>Ld</td>
<td></td>
</tr>
<tr>
<td>St</td>
<td>0x abcd00</td>
</tr>
<tr>
<td>Ld</td>
<td>0x abc000</td>
</tr>
<tr>
<td>Ld</td>
<td>0x abcd00</td>
</tr>
</tbody>
</table>
The Alpha 21264 Out-of-Order Implementation

Branch prediction and instr fetch

R1 ← R1+R2  
R2 ← R1+R3  
BEQZ R2  
R3 ← R1+R2  
R1 ← R3+R2  
LD R4 ← 8[R3]  
ST R4 → 8[R1]

Instr Fetch Queue

Decode & Rename

Instr 1  
Instr 2  
Instr 3  
Instr 4  
Instr 5  
Instr 6  
Instr 7

Reorder Buffer (ROB)

Instr 1  
Instr 2  
Instr 3  
Instr 4  
Instr 5  
Instr 6  
Instr 7

Committed Reg Map

R1 → P1  
R2 → P2

P33 ← P1+P2  
P34 ← P33+P3  
BEQZ P34  
P35 ← P33+P34  
P36 ← P35+P34  
P37 ← 8[P35]  
P37 → 8[P36]

Issue Queue (IQ)

P37 ← [P35 + 8]  
P37 → [P36 + 8]

Register File P1-P64

ALU  
ALU  
ALU

Results written to regfile and tags broadcast to IQ

ALU

D-Cache
Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1</td>
<td>[R2]</td>
<td>3</td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3</td>
<td>[R4]</td>
<td>6</td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5</td>
<td>[R6]</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>LD</td>
<td>R7</td>
<td>[R8]</td>
<td>2</td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9</td>
<td>[R10]</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>R11</td>
<td>[R12]</td>
<td>1</td>
<td>abba</td>
</tr>
</tbody>
</table>
Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD R1</td>
<td>[R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD R3</td>
<td>[R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST R5</td>
<td>[R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD R7</td>
<td>[R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST R9</td>
<td>[R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>LD R11</td>
<td>[R12]</td>
<td>1</td>
<td>abba</td>
<td>2</td>
</tr>
</tbody>
</table>
Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1</td>
<td>3</td>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>R3</td>
<td>6</td>
<td>adde</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>R5</td>
<td>5</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7</td>
<td>2</td>
<td>abce</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>R9</td>
<td>1</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11</td>
<td>2</td>
<td>abba</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>5</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>1</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>2</td>
<td></td>
<td>abba</td>
</tr>
</tbody>
</table>
Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. **Assume memory dependence prediction.**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td></td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td></td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td></td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>1</td>
<td></td>
<td>abba</td>
</tr>
</tbody>
</table>
Problem 4

Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. **Assume memory dependence prediction.**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ⇐ [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ⇐ [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ⇐ [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ⇐ [R12]</td>
<td>1</td>
<td>abba</td>
<td>2</td>
</tr>
</tbody>
</table>
The Cache Hierarchy

Core → L1 → L2 → L3 → Off-chip memory
Problem 1

• Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

  With L3: 1000 + 10x20 + 30x10 + 300x5 = 3000
  Without L3: 1000 + 10x20 + 10x300 = 4200
Accessing the Cache

Direct-mapped cache: each address maps to a unique address

Data array

Sets

8-byte words

Offset

Byte address

101000

8 words: 3 index bits
The Tag Array

Direct-mapped cache: each address maps to a unique address.

8-byte words

Data array

Tag array

Compare

Tag

Byte address

101000
Increasing Line Size

- Byte address
- Tag
- Offset

10100000

Tag array

Data array

A large cache line size $\rightarrow$ smaller tag array, fewer misses because of spatial locality

32-byte cache line size or block size
Associativity

Byte address

Tag

10100000

Set associativity $\rightarrow$ fewer conflicts; wasted power because multiple data and tags are read

Tag array

Compare

Data array

Way-1

Way-2
Problem 2

- Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 2

• Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH MM H MM HM HMM M M M M
Problem 3

• Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 3

• Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH M MH MM HM HMM M H M
Problem 4

• 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address

• How many sets?

• How many index bits, offset bits, tag bits?

• How large is the tag array?

Equations:
Data array size (cache size) = \#sets x \#ways x blocksize
Tag array size = \#sets x \#ways x tagsize
Index bits = \log_2 (\#sets)
Offset bits = \log_2 (blocksize)
Tag bits + index bits + offset bits = address width
Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address

- How many sets? 64

- How many index bits (6), offset bits (6), tag bits (28)?

- How large is the tag array (28 Kb)?
Problem 5

- 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address

- How many sets? How many ways?

- How many index bits, offset bits, tag bits?

- How large is the tag array?
Problem 5

- 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address

- How many sets (1) ? How many ways (128) ?

- How many index bits (0), offset bits (6), tag bits (34) ?

- How large is the tag array (544 bytes) ?