CS150 Cache Checkpoint

Dan Yeager, Daiwei Li, James Parker
Current Architecture: Block Rams

- Essentially single write port, dual read ports
- Pros:
  - Simple
  - Fast (1 cycle)
  - Allows us to load programs into IMEM using SW
- Cons:
  - Max 5MB block ram space
Expanding Memory

![Diagram of memory expansion with CPU, Cache, Main Memory (DDR2), and Hard Disk, illustrating the speed and capacity trade-offs.]

<table>
<thead>
<tr>
<th>Technology</th>
<th>Cost/GB</th>
<th>Access Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>~ $10,000</td>
<td>~ 1 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>~ $100</td>
<td>~ 100 ns</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>~ $1</td>
<td>~ 10,000,000 ns</td>
</tr>
</tbody>
</table>

Images from DDCA
**DDR2 - Basic Idea**

- **Same interface:**
  - Dual read read ports for I & D
  - Writes now more complicated (to be discussed)
Concurrency

- Write 0xBAADDF00D to ICache 0x00000000
- Write 0xDEADBEEF to DCache 0x00000000

Caches not concurrent!
Concurrent Solution

1. Writing a new program:
   ○ Write to I & D simultaneously
     ■ Hardware enforced
   ○ Concurrency enforced

2. D-cache loads and stores:
   ○ Don't overwrite the program space
     ■ SW enforced
   ○ OK if not concurrent
- Write to I & D Cache simultaneously
- Run bios out of block ram
  - Like I/D mem from last checkpoint
## Memory Map

### Example Address:
yyyy_xxx_xxx_xxx_xxx_xxx_xxx_xxx_xxx_xxx

### Top Nibble

<table>
<thead>
<tr>
<th>Device</th>
<th>R/W</th>
<th>Top Nibble</th>
<th>Address Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>D$</td>
<td>R/W</td>
<td>xxx1</td>
<td>Mem</td>
</tr>
<tr>
<td>I$</td>
<td>R</td>
<td>xxx1</td>
<td>PC</td>
</tr>
<tr>
<td>I$</td>
<td>W</td>
<td>xx1x</td>
<td>Mem</td>
</tr>
<tr>
<td>BIOS</td>
<td>R</td>
<td>x1xx</td>
<td>PC/Mem</td>
</tr>
<tr>
<td>I/O</td>
<td>R/W</td>
<td>1xxx</td>
<td>Mem</td>
</tr>
</tbody>
</table>

### Diagram

- **D Cache**
- **I Cache**
- **BIOS (RAM)**
- **UART**
Memory Map Examples

Based on the map, what do these ops do?

- PC=0x4000010, MemAddr=0x30000000, SW
- PC=0x1000000, MemAddr=0x80000000, LW
- PC=0x1000000, MemAddr=0x30000000, SW

<table>
<thead>
<tr>
<th>Device</th>
<th>R/W</th>
<th>Top Nibble</th>
<th>Address Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>D$</td>
<td>R/W</td>
<td>xxx1</td>
<td>Mem</td>
</tr>
<tr>
<td>I$</td>
<td>R</td>
<td>xxx1</td>
<td>PC</td>
</tr>
<tr>
<td>I$</td>
<td>W</td>
<td>xx1x</td>
<td>Mem</td>
</tr>
<tr>
<td>BIOS</td>
<td>R</td>
<td>x1xx</td>
<td>PC/Mem</td>
</tr>
<tr>
<td>I/O</td>
<td>R/W</td>
<td>1xxx</td>
<td>Mem</td>
</tr>
</tbody>
</table>
Stalls

Remember:

Speed, cost, size tradeoff

CPU needs to wait (stall) for cache miss
Stall Implementation

- Implement & verify separate from your cache
  - Modularity is crucial for testability
- Test by stalling each instruction
  - @(posedge clk) stall <= ~stall;
- Ensure that no state elements are written while stall is asserted
Final Notes

- Start early
- Please read spec carefully
- Draw timing diagrams
- Instrument your code
- Survey of CP3 common mistakes (soon)