Review
- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each process vs. tag/data in cache
  - TLB is cache of Virtual-Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

Translation Look-Aside Buffers (TLBs)
- TLBs usually small, typically 128 - 256 entries
- Like any other cache, the TLB can be direct mapped, set associative, or fully associative

On TLB miss, get page table entry from main memory

Address Mapping: Page Table

Address Translation

Typical TLB Format

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
  - Dirty: since use write back, need to know whether or not to write page to disk when replaced
  - Ref: Used to help calculate LRU on replacement
  - Cleared by OS periodically, then checked to see if page was referenced
What if not in TLB?

• Option 1: Hardware checks page table and loads new Page Table Entry into TLB
• Option 2: Hardware traps to OS, up to OS to decide what to do
  - MIPS follows Option 2: Hardware knows nothing about page table

What if the data is on disk?

• We load the page off the disk into a free block of memory, using a DMA transfer (Direct Memory Access – special hardware support to avoid processor)
  - Meantime we switch to some other process waiting to be run
• When the DMA is complete, we get an interrupt and update the process’s page table
  - So when we switch back to the task, the desired data will be in memory

What if we don’t have enough memory?

• We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
• And update that program’s page table to reflect the fact that its memory moved somewhere else
• If continuously swap between disk and memory, called Thrashing

Question (1/3)

• 40-bit virtual address, 16 KB page

<table>
<thead>
<tr>
<th>Virtual Page Number (? bits)</th>
<th>Page Offset (? bits)</th>
</tr>
</thead>
</table>

• 36-bit physical address

<table>
<thead>
<tr>
<th>Physical Page Number (? bits)</th>
<th>Page Offset (? bits)</th>
</tr>
</thead>
</table>

• Number of bits in Virtual Page Number/ Page offset, Physical Page Number/Page offset?

1: 22/18 (VPN/PO), 22/14 (PPN/PO)
2: 24/16, 20/16
3: 26/14, 22/14
4: 26/14, 26/10
5: 28/12, 24/12

(1/3) Answer

• 40- bit virtual address, 16 KB (2^{14} B)

<table>
<thead>
<tr>
<th>Virtual Page Number (26 bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
</table>

• 36- bit virtual address, 16 KB (2^{14} B)

<table>
<thead>
<tr>
<th>Physical Page Number (22 bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
</table>

• Number of bits in Virtual Page Number/ Page offset, Physical Page Number/Page offset?

1: 22/18 (VPN/PO), 22/14 (PPN/PO)
2: 24/16, 20/16
3: 26/14, 22/14
4: 26/14, 26/10
5: 28/12, 24/12

Question (2/3): 40b VA, 36b PA

• 2-way set-associative TLB, 512 entries, 40b VA:

<table>
<thead>
<tr>
<th>TLB Tag (? bits)</th>
<th>TLB Index (? bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
</table>

• TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Virtual Page Number, Physical Page Number

<table>
<thead>
<tr>
<th>V D Access (2 bits)</th>
<th>TLB Tag (? bits)</th>
<th>Physical Page No. (? bits)</th>
</tr>
</thead>
</table>

• Number of bits in TLB Tag / Index / Entry?

1: 12 / 14 / 38 (TLB Tag / Index / Entry)
2: 14 / 12 / 40
3: 18 / 8 / 44
4: 18 / 8 / 58
### 2/3 Answer

- **2-way set-associative data cache, 256 (2^8) “sets”, 2 TLB entries per set => 8 bit index**

<table>
<thead>
<tr>
<th>TLB Tag (18 bits)</th>
<th>TLB Index (8 bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Page Number (26 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TLB Entry:** Valid bit, Dirty bit, Access Control (2 bits), Virtual Page Number, Physical Page Number

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Access (2 bits)</th>
<th>TLB Tag (18 bits)</th>
<th>Physical Page No. (22 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>12 / 14 / 38 (TLB Tag / Index / Entry)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>14 / 12 / 40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>18 / 8 / 44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>18 / 8 / 58</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Question (3/3)

- **2-way set-associative, 64KB data cache, 64B block**

<table>
<thead>
<tr>
<th>Cache Tag (7 bits)</th>
<th>Cache Index (9 bits)</th>
<th>Block Offset (7 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Address (36 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Data Cache Entry:** Valid bit, Dirty bit, Cache tag + ? bits of Data

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Cache Tag (7 bits)</th>
<th>Cache Data (7 bits)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Number of bits in Data cache Tag / Index / Offset / Entry?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)</td>
</tr>
<tr>
<td>2: 20 / 10 / 6 / 86</td>
</tr>
<tr>
<td>3: 20 / 10 / 6 / 534</td>
</tr>
<tr>
<td>4: 21 / 9 / 6 / 87</td>
</tr>
<tr>
<td>5: 21 / 9 / 6 / 535</td>
</tr>
</tbody>
</table>

### 3/3 Answer

- **2-way set-associative data cache, 64K/1K (2^10) “sets”, 2 entries per set => 9 bit index**

<table>
<thead>
<tr>
<th>Cache Tag (21 bits)</th>
<th>Cache Index (9 bits)</th>
<th>Block Offset (6 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Address (36 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Data Cache Entry:** Valid bit, Dirty bit, Cache tag + 64 Bytes of Data

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Cache Tag (21 bits)</th>
<th>Cache Data (64 Bytes = 512 bits)</th>
</tr>
</thead>
</table>

| 1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry) |
| 2: 20 / 10 / 6 / 86 |
| 3: 20 / 10 / 6 / 534 |
| 4: 21 / 9 / 6 / 87 |
| 5: 21 / 9 / 6 / 535 |

### 4 Qs for any Memory Hierarchy

- **Q1:** Where can a block be placed?
  - One place (direct mapped)
  - A few places (set-associative)
  - Any place (fully associative)

- **Q2:** How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)

- **Q3:** Which block is replaced on a miss?
  - Least recently used (LRU)
  - Random

- **Q4:** How are writes handled?
  - Write through (Level never inconsistent w/ lower)
  - Write back (Could be “dirty”, must have dirty bit)

### Q1: Where block placed in upper level?

- **Block #12 placed in 8 block cache:**
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping = Block # Mod # of Sets

- **Block 12:**
  - Fully associative block 12 can go anywhere
  - Direct mapped: block 12 can go only into block 4 (12 mod 4)
  - 2-way set associative: block 12 can go anywhere in set 0 (12 mod 4)

### Q2: How is a block found in upper level?

- **Block Address**
  - Tag
  - Index
  - Block Offset

- **Set Select**
  - Data Select

- **Direct indexing (using index and block offset), tag compares, or combination**

- **Increasing associativity shrinks index, expands tag**
Q3: Which block replaced on a miss?
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Miss Rates
Associativity: 2-way  4-way  8-way

<table>
<thead>
<tr>
<th>Size</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.4%</td>
<td>5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>1.4%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

Q4: What to do on a write hit?
- Write-through
  - update the word in cache block and corresponding word in memory
- Write-back
  - update word in cache block
  - allow memory word to be "stale"
  - add ‘dirty’ bit to each line indicating that memory be updated when block is replaced
  - OS flushes cache before I/O !!!

- Performance trade-offs?
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes

Three Advantages of Virtual Memory
1) Translation:
- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program (“Working Set”) must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

2) Protection:
- Different processes protected from each other
- Different pages can be given special behavior
  - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more “viruses” under Microsoft Windows
- Special Mode in processor (“Kernel mode”) allows processor to change page table/TLB

3) Sharing:
- Can map same physical page to multiple users (“Shared memory”)

Why Translation Lookaside Buffer (TLB)?
- Paging is most popular implementation of virtual memory (vs. base/bounds)
- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection / indirection
- Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast

And in Conclusion...
- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
- Virtual Memory allows protected sharing of memory between processes with less swapping to disk