Reminders / Announcements

• Final Exam is on Saturday
  – Exam is comprehensive; expect it to be ~50% longer
  – Logistics much like the midterm:
    • Available 8am-8pm US/Pacific
    • Single-shot, forward progress only
    • 3 hour time limit (100% more time; should reduce time pressure)
• Friday lecture will be an exam review session
  – Come with questions! I stop when you do* [or at 2pm]
• One, final optional participation quiz from this week’s content
  – Will be posted after today’s lecture, deadline Tuesday, as always
  – Will count as a make-up for any missed participation quiz
• HW Solutions?
  – HW7 solutions will be posted right after the deadline
VIRTUAL MEMORY
Virtual Memory

- It’s just another level in the cache/memory hierarchy
- *Virtual memory* is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).
Virtual Memory

- is just cacheing, but uses different terminology (and different storage/lookup techniques)

<table>
<thead>
<tr>
<th>cache</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>page</td>
</tr>
<tr>
<td>cache miss</td>
<td>page fault</td>
</tr>
<tr>
<td>address</td>
<td>virtual address</td>
</tr>
<tr>
<td>index</td>
<td>physical address (sort of)</td>
</tr>
</tbody>
</table>
Virtual Memory

• What happens if another program in the processor uses the same addresses that yours does?
• What happens if your program uses addresses that don’t exist in the machine?
• What happens to “holes” in the address space your program uses?

• So, virtual memory provides
  – performance (through the cacheing effect)
  – protection
  – ease of programming/compilation
  – efficient use of memory
Virtual Memory...

...is just a mapping function from virtual memory addresses to physical memory locations, which allows caching of virtual pages in physical memory.
What makes VM different than memory caches

- **MUCH** higher miss penalty (millions of cycles)!
- Therefore
  - large pages [equivalent of cache line] (4 KB to MBs)
  - associative mapping of pages (typically fully associative)
  - software handling of misses (but not hits!!)
  - write-through not an option, only write-back
Virtual Memory mapping
Virtual Address Translation

- We do not need to translate/change all bits of the address.

- We’ll only change high order bits, and leave the low order bits alone – the number of low bits we do not change defines the “page size”.
  - Page size (virtual memory) is analogous to block size (caches) – it is the chunk of memory that gets moved as a unit on a miss.
Address translation via the page table

• All page mappings are in the page table, so hit/miss is determined solely by the valid bit

• So why is this fully associative???
  – Where are our “tag” “index” and “block offset”?

• Biggest problem – this is slow. Why?
Making Address Translation Fast

- A cache for address translations: **translation lookaside buffer (TLB)**

  TLB accessed first, page table only accessed on TLB miss.
TLBs and caches
Virtual Memory & Caches

• Cache lookup is now a **serial** process
  1. V->P translation through TLB
  2. Get index
  3. Read tag from cache
  4. Compare

• How can we make this faster?
  1.
  2.
Virtual Caches

• Which addresses are used to lookup data in cache/store in tag?
  – Virtual Addresses?
  – Physical Addresses?

• Pros/Cons?
  – Virtual
  – Physical
Fast Index Translation

• Can do
  1. V->P translation through TLB
  2. Get index

in parallel, if the v->p translation does not change the index.
TLBs and caches
Virtual Memory Key Points

- How does virtual memory provide:
  - protection?
  - sharing?
  - performance?
  - illusion of large main memory?

- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.

- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.