Computer Systems Architecture I

CSE 560M
Lecture 18
Guest Lecturer: Shakir James
Plan for Today

• **Announcements**
  – No class meeting on Monday, meet in project groups
  – Project demos < 2 weeks, Nov 23rd

• **Questions**

• **Review of project logistics**

• **Today’s discussion: Main and Virtual Memory**
Project Logistics

• Five major deadlines
  – **Project Demo** Nov 23
  – Report due Dec 7

<table>
<thead>
<tr>
<th>Task</th>
<th>Due</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Milestone 1</td>
<td>Oct 19</td>
<td>Processor Design</td>
</tr>
<tr>
<td>Project Milestone 2</td>
<td>Nov 2</td>
<td>Detailed Design &amp; VHDL</td>
</tr>
<tr>
<td>Design Presentations</td>
<td>Nov 4</td>
<td>Presentation of detailed design</td>
</tr>
<tr>
<td>Project Demo</td>
<td>Nov 23</td>
<td>Demo working VHDL model</td>
</tr>
<tr>
<td>Final Report</td>
<td>Dec 7</td>
<td>Final report due</td>
</tr>
</tbody>
</table>
**Cache Memory (DRAM)**

**Speed:**
- 0.25 ns
- 1 ns
- 100 ns
- 5 ms

* © 2003 Elsevier Science (USA). All rights reserved.
Main Memory

- Recall that on a cache miss, need to access memory. This process has three components
  - Send the address
  - Get the contents of the memory location
  - Transfer contents
- Parameters for dynamic random access memory (DRAM)
  - *Access time*: time between the read is requested and the desired word arrives
  - *Cycle time*: minimum time between requests to memory
    - cycle time > access time to allow address lines to stabilize and to write back data (i.e., reads are destructive)
    - In 2002, cycle times were 80 ns (compared to 250ns in 1980)
DRAM

• Bits stored in single transistors
  – Reads are destructive
  – Transistors “leak” (i.e. Dynamic) so must be refreshed

• Address lines split into row and column addresses. A read consists of
  – RAS – row access strobe
  – CAS – column access strobe

• Access time is typically RAS + CAS, but periodic refreshing can stall requests
DRAM Structure

- Address buffer
- Row decoder
- Column decoder
- Sense amps and I/O
  - Memory array (16,384 x 16,384)
  - Word line
  - Storage cell
- Bit line
- Data in (D)
- Data out (Q)
DRAM and SRAM

- Both are a memory organization using transistors
- S stands for static
  - Traditionally uses 6 Ts/bit (some use 4 or less)
  - No refresh
  - No need to write after read
- Main memory uses DRAM; on-chip caches use SRAM; on-board caches have been built with both
- DRAM semiconductor fabs use specific processes to create chips
  - Different layers, fewer layers
  - Different physical properties
Improving Main Memory Bandwidth

• Sending address
  – Can’t really be improved
  – Can use split-transaction bus to allow overlap

• Make memory wider
  – Send one address, return more than one word
  – Wider main memory increases memory increment size
  – Complicates error correcting codes when writing one word in a multi-word memory
Interleaving

• Memory is organized in banks
  – Bank \(i\) stores all words at address \(j \mod I\)
  – Requests to different banks can occur in consecutive clock cycles
    • All banks can read a word in parallel
  – Number of banks should match L2 block size
  – Bus does not need to be wider
  – Writes to individual banks for different addresses can proceed without waiting for the preceding write to finish

• Problem: number of banks limited by increasing chip capacity
  – With 1M*1 bit chips, it takes 64*8=512 chips to get 64MB (easy to put 16 banks of 32 chips)
  – With 64M*1 bit chips, it takes only 8 chips (only one bank)
Memory Width Illustration

(a) One-word-wide memory organization
(b) Wide memory organization
(c) Interleaved memory organization

CPU
Cache
Bus
Memory

Multiplexor
Cache
Bus
Memory

CPU
Cache
Bus

Memory bank 0
Memory bank 1
Memory bank 2
Memory bank 3

© 2003 Elsevier Science (USA). All rights reserved.
Page-mode, Synchronous and Double Data Rate DRAMs

- Fast page mode: Add a row buffer to DRAM
  - Keep most recently accessed row (i.e. page) open as a buffer
  - Hit: read out of buffer, skip RAS
  - Miss: cost of miss + RAS + CAS

- Synchronous: Add a clock
  - Traditional DRAMs were asynchronous; adding a clock eliminates sync overhead at each request
  - Also adds a “width” register to control number of words to be returned

- DDR:
  - Transfer data on the rising and falling edge of clock
  - Doubles data rate

- All three add a small amount of logic to exploit internal DRAM bandwidth (small cost, large improvement)
Virtual Memory

- Purpose: give program(mer) illusion of full-sized, unshared main memory
- Hardware assists for address translation and memory protection
  - Largely invisible to programmer
- Prior to virtual memory
  - Overlays
  - Relocation registers
  - Paging
  - Segmentation
- Paging systems predate caches
  - But same questions arise (mapping, replacement and write policies)
- An enormous difference: miss penalty
Paged Virtual Memory Illustrated
Two Extremes in the Memory Hierarchy

<table>
<thead>
<tr>
<th>Parameter</th>
<th>L1 Cache</th>
<th>Paging System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16-64 bytes</td>
<td>4K-8K bytes</td>
</tr>
<tr>
<td>Miss (fault) time</td>
<td>10-100 cycles</td>
<td>Millions</td>
</tr>
<tr>
<td></td>
<td>(20-1000 ns)</td>
<td>(3-20 ms)</td>
</tr>
<tr>
<td>Miss (fault) rate</td>
<td>1-10%</td>
<td>0.00001-0.001%</td>
</tr>
<tr>
<td>Memory size</td>
<td>4K-64K bytes</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Other Extreme Differences

• Mapping: Restricted (L1) vs. general (Paging)
  – Hardware assist for virtual address translation (TLB)

• Miss handler
  – Hardware only for caches
  – Software only for paging system (context switch)
  – Hardware/Software for TLB

• Replacement algorithm
  – Not important for caches
  – Critical for paging systems

• Write policy
  – Always write back for paging systems
Paging and Segmentation

• Paging: fixed block sizes
  – Easy to map, translate, replace, transfer to/from disk
  – Does not correspond to semantic objects, hence internal fragmentation

• Segmentation: variable block sizes
  – Requires two addresses per word: segment and offset
  – “Difficult” to translate (need to check segment length), place and replace (external fragmentation), transfer to/from disk
  – Allocation can be problematic (need contiguous memory)
  – Corresponds to semantic objects

• Segmentation with paging (segment holds $N$ pages)
  – Good for large objects
  – Helps allocation problem
Page Tables

- Page tables contain page table entries (PTEs):
  - Physical page number, as well as valid, protection, dirty, and use bits
    • virtual page number is often known implicitly based on the address, but can be stored explicitly as below

- Hardware register points to the page table of the running process

- Page table structures
  - Traditional structure needs one entry per virtual page
    • 32-bit addresses, 4KB pages gives $2^{20}=1$M entries
    • Assuming 4 bytes per entry, size is 4MB
  - Some systems have inverted page tables
    • Goal: reduce size of page table
    • One entry for each physical page
    • Hash the virtual page number to form table index
    • Size now due to physical page count, rather than virtual
      • 512MB main memory needs a 1MB table, assuming an additional 4 bytes per entry to store the virtual address
  - In all modern systems, page table entries are cached in a translation lookaside buffer (TLB)
TLBs

- Cache for page table entries (PTEs)
  - Usually fully associative

- TLB miss handled by hardware or by software
  - TLB miss 10-100 cycles, means you don’t need to context switch

- Addressed in parallel with access to cache

- TLB is smaller than page table, admits faster access
  - Can be on the processor’s critical path

- For a given TLB size (number of entries)
  - Larger page size implies larger effective capacity (ie, each TLB entry translates more addresses)
  - Modern O.S.s have a page size parameter
Virtually Indexed, Physically Tagged Caches

- Does the cache use virtual or physical addresses?

- Why not all physical?
  - Need to translate every time: serialize hierarchy

- Why not all virtual?
  - Protection (page level protection is checked at translation time?)
  - Context switches (each switch changes the meaning of virtual addresses)
    - Can add PID to cache address tag
  - Aliasing (OS and apps can use *synonyms* to refer to same locations)
  - I/O uses physical addresses

- Idea: use page offset, which is the same for virtual and physical addresses, to index a small cache
  - TLB entry (virtual page number) and Cache tag entry (portion of the page offset) are indexed in parallel
    - Tags are physical addresses
  - Limits the size of direct-mapped cache to a page!
Caches and Address Translation
I/O and Caches
(a first look at cache coherence)

• Alternatives
  – I/O data passes through the cache on its way from/to memory
    • No coherency problem but contention for cache access
  – I/O interacts directly with main memory
    • Coherency problem
### I/O Consistency Illustrated

<table>
<thead>
<tr>
<th>I/O</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cache</td>
<td>Cache</td>
<td>Cache</td>
</tr>
<tr>
<td>A</td>
<td>100</td>
<td>A</td>
<td>550</td>
</tr>
<tr>
<td>B</td>
<td>200</td>
<td>B</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>Memory</td>
<td>Memory</td>
</tr>
<tr>
<td>A</td>
<td>100</td>
<td>A</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>200</td>
<td>B</td>
<td>440</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
</tr>
<tr>
<td>(a) Cache and memory coherent: A = A and B = B</td>
<td>(b) Cache and memory incoherent: A ≠ A (A stale)</td>
<td>(c) Cache and memory incoherent: B ≠ B (B stale)</td>
<td></td>
</tr>
</tbody>
</table>
I/O Consistency - Software Approach

• On Output:
  – If write through cache, no problem
  – If write back cache, purge the cache via O.S. interaction

• On Input:
  – Both WT and WB:
    • use a non-cacheable buffer space
    • After input is completed, flush the cache of blocks mapping to these addresses and make the buffer cacheable.
I/O Consistency - Hardware Approach

• Subset of the shared-bus multiprocessor cache coherence protocol

• Cache controller snoops on the bus
  – On output, if there is a match in tags: if WT nothing to do, else if WB *send the cache entry instead*
  – On input, if there is a match in tags: store in both the cache and main memory (or invalidate the cache entry)
Assignment

• Project demos in < 2 weeks
• Readings
  – For Monday
    • No class meeting, meet in project groups