Course Administration

- Homework 6 (optional) due on Friday April 21, 2017 (bonus points)

- Project is due on Friday April 21 by 11:59 PM; a softcopy of the project should be sent to TA in pdf format ONLY

- Final exam on Monday April 24 from 12:20 – 2:20 PM
Virtual Memory

- Use main memory as a “cache” for secondary memory
  - Allows efficient and safe sharing of memory among multiple programs
  - Provides the ability to easily run programs larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)

- What makes it work? – again the Principle of Locality
  - A program is likely to access a relatively small portion of its address space during any period of time

- Each program is compiled into its own address space – a “virtual” address space
  - During run-time each virtual address must be translated to a physical address (an address in main memory)

Two Programs Sharing Physical Memory

- A program’s address space is divided into pages (all one fixed size) or segments (variable sizes)
  - The starting location of each page (either in main memory or in secondary memory) is contained in the program’s page table

![Diagram of two programs sharing physical memory](image)
Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software

Virtual Address (VA)

<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical page number

<table>
<thead>
<tr>
<th>Physical page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Address (PA)

- So each memory request first requires an address translation from the virtual space to the physical space
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault

Address Translation Mechanisms

![Diagram showing address translation mechanisms]
Virtual Addressing with a Cache

- Thus it takes an *extra* memory access to translate a VA to a PA

- This makes memory (cache) accesses very expensive (if every access was really two accesses)

- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

Making Address Translation Fast

- Diagram showing the process of address translation with TLB and page table.
Translation Lookaside Buffers (TLBs)

- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped.

<table>
<thead>
<tr>
<th>V</th>
<th>Virtual Page #</th>
<th>Physical Page #</th>
<th>Dirty</th>
<th>Ref</th>
<th>Access</th>
</tr>
</thead>
</table>

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches).
  - TLBs are typically not more than 128 to 256 entries even on high end machines.

A TLB in the Memory Hierarchy

- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB.
    - Takes 10’s of cycles to find and load the translation info into the TLB.
  - If the page is not in main memory, then it’s a true page fault.
    - Takes 1,000,000’s of cycles to service a page fault.

- TLB misses are much more frequent than true page faults.
Class Problem 1

- Main Memory size is 256K words, Cache size is 16K words, block size is 32 bytes

- Show the mapping in terms of tag, index and block offset for
  - Direct-mapped
    - Tag = \( \log_2(256K \times 4) - (\text{Index} + \text{Block Offset}) = 20 - (11 + 5) = 4 \)
    - Index = Cache size - Block Offset = \( \log_2(16K \times 4) - \log_2(32) = 16 - 5 = 11 \)
    - Block Offset = \( \log_2(32) = 5 \)
  - 4-way set-associative
    - Tag = \( \log_2(256K \times 4) - (\text{Block Offset}) = 20 - 5 = 15 \)
    - Index = Cache size - (Block Offset + Associativity) = \( \log_2(16K \times 4) - (\log_2(32) - \log_2(4)) = 16 - (5 - 2) = 9 \)
    - Block Offset = \( \log_2(32) = 5 \)
  - Fully associative
    - Tag = \( \log_2(256K \times 4) - (\text{Block Offset}) = 20 - 5 = 15 \)
    - Block Offset = \( \log_2(32) = 5 \)
Class Problem 2

- Given the following: instruction breakdown, LW 25%, Add 20%, BEQ 20%, SW, 15%, Nand 20%; branches are taken 60% of the time, LW followed by immediate use occurs 20% of the time. The instruction cache hit rate is 90% and the data cache hit rate is 80%. Assume each cache miss requires 10 stall cycles.
  - Data hazards – detect and forward
  - Control hazards – predict not taken and squash (3 cycle penalty)
  - Calculate the CPI

Base CPI is 1, assume 100 instructions
Stalls due to LW-use: 25 LWs\* 0.2 = 5; 5 \* 1 cycle each = 5 cycles
Stalls due to mispred BEQs: 20 BEQs\* 0.6 = 12; 12 \* 3 cycles each = 36 cycles
Stalls due to Icache misses: 100 instrs\* 0.1 = 10; 10 \* 10 cycles each = 100 cycles
Stalls due to Dcache misses: 40 instrs\* 0.2 = 8; 8 \* 10 cycles each = 80 cycles
Stall CPI = (5 + 36 + 100 + 80)/100 = 2.21
CPI = baseCPI + stallCPI = 1.0 + 2.21 = 3.21