Notes and Updates

- Office hours for Dr. Simon: Thurs before final 9-12 and 1:30-2:30
- Today:
  - Virtual Memory
  - Questions about final
  - Your suggestions on making this class better
- Review session: Tuesday Center 109 6:30-7:50pm
- Surveys?
- Questions?

Assembly uses Virtual Addressing

- Each PC address and memory address used by a program is a virtual address
  - That way the compiler can use any range of addresses it wants
- At runtime, the OS assigns a program into a particular physical memory space
  - Which can provide protection, if you try to access out off your space...
- But now EVERY ACCESS to memory must be translated
The basic idea (page 7.19)

- Physical memory is broken into a number of pages
  - A range of P addresses that map to a similar range of V addresses
  - Reduces bookkeeping
- A page may be in PM or NOT (on disk)
- LIKE A ___________
Virtual Address Translation (pg 513)

- **Cache**
  
<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
<th>offset</th>
</tr>
</thead>
</table>

- **VA**
  
  | 32 bits |

- **PA**
  
  However many Bits you need to Access your memory size

Block/Line size vs page size

- Page size large enough to “cover” or “amortize” time to swap a page
  - 4-16KB typical
- Page fault (miss) EXPENSIVE - millions of cycles
  - Use software controlled fully associative
  - Spend some TIME in software alg to get the best possible replacement scheme
- Write back!
  - Pages on disk will be out of date
  - Ever unplugged your computer while it was running?
  - What happens on a “save” from word?
Hidden detail: address translation mechanism

• A Page Table: indexed by VP number

Hidden detail: The translation mechanism

• Since any virtual page # can map to any physical page # (ie fully assoc)
  - How do we get at it FAST?
  - We need to do this in significantly less than one cycle! (IF (PC) and MEM (Id/st) stages)
**TLB: Translation-Lookaside Buffer (pg522)**

- **AKA: Cache for the Page Table**

![Diagram of TLB structure](image)

**TLB generalities**

- **Size:** 16-512 entries
- **Block/Line Size:** 1-2 page table entries
  - $32\log_2(\text{page size})$ bits for tag +
    - 1 or 2 physical addresses +
      - $db+vb+rb \approx 100 \text{ bits}$
- **Hit time FAST:** .5-1 cycle
- **Miss penalty:** 10-100 cycles (eek)
- **Miss Rate:** 0.01%-1% (whew)
- **Associativity:** -- varies
Putting it all together
(the last thing you have to know)
(though page 528)

Pipelining in Today's Most Advanced Processors

- Not fundamentally different than the techniques we discussed
- Superpipelining
  - Deepens pipeline
- Superscalar execution: replicates pipeline
  - In order (mostly called "superscalar")
    - Replicates pipelines in parallel (ie, the laundromat option)
  - VLIW (very-long-instruction-word)
    - Replicates pipeline, but requires compiler to schedule precisely
    - Doesn't check for WAW, WAR or RAW stalls in hardware!
  - Out-of-order execution
    - Can have multiple pipelines, stalls can be avoided by allowing later instructions to "pass" earlier ones.
Processor under-utilization

- **Vertical Waste** - Introduced when the processor issues no instructions in a cycle.
- **Horizontal Waste** - Introduced when not all issue slots (superscalar pipeline) can be filled in a cycle.
- Modern superscalars introduce horizontal waste
  - Longer pipes (while giving faster frequencies) increase vertical waste.
- Multithreaded architectures attack vertical waste. (but only allow one thread to issue per cycle)

### Figure 1: Empty issue slots can be defined as either vertical waste or horizontal waste. Vertical waste is introduced when the processor issues no instructions in a cycle. Horizontal waste when not all issue slots can be filled in a cycle. Superscalar execution (as opposed to single-issue execution) both introduces horizontal waste and increases the amount of vertical waste.

---

Why do we stall?

- **Shorter Stalls:**
  - L1 cache miss
  - Branch
  - Mispredictions

- **Longer Stalls:**
  - L2 cache miss
  - TLB miss

---
Why do we stall?

- Longer Stalls:
  - L2 cache miss
  - TLB miss

- Shorter Stalls:
  - L1 cache miss
  - Branch xMispredictions

Figure 2: Sources of all unused issue cycles in an 8-issue superscalar processor. Purple bars represent the utilized issue slots; all others represent wasted issue slots.

Traditional multithreading and where do stall come from?

- Shorter Stalls:
  - L1 cache miss
  - Branch xMispredictions

- Longer Stalls:
  - L2 cache miss
  - TLB miss
**Simultaneous Multithreading**

- **Shorter Stalls:**
  - L1 cache miss
  - Branch mispredictions

- **Longer Stalls:**
  - L2 cache miss
  - TLB miss

---

**What is Simultaneous Multi-Threaded?**

- **Simultaneous Multi-Threaded (SMT)**
  - allows several independent threads to “start” an instruction in one cycle

- **Goal:**
  - increase processor utilization reducing empty slots
  - attacks both horizontal and vertical waste
    - Horizontal - allows more than one “process” per cycle
    - Vertical - if one process misses in cache, another can keep executing
Does it work?

![Graph showing instructions issued per cycle vs. number of threads.
(c) SM: Full Simultaneous Issue

Preparing for the Final

- A cheat sheet will be provided - like midterm
- Quizzes
- Homeworks (plus recommended problems, plus problems NEAR the homework problems)
- Questions suggested in class
- Midterm
- STUDY GROUPS
  - Explain things to each other
- SLEEP
  - At LEAST 6 hours before the exam
Questions about final?

How can I make this class better?
THANKS!