Recall: Making it real: X86 Memory model with segmentation (16/32-bit)

Recall: In Machine Structures (eg. 61C) …
- Caching is the key to memory system performance

Average Memory Access Time (AMAT)
= (Hit Rate x HitTime) + (Miss Rate x MissTime)
Where HitRate + MissRate = 1
HitRate = 90% => AMAT = (0.9 x 10) + (0.1 x 110) = 20ns
HitRate = 99% => AMAT = (0.99 x 10) + (0.01 x 110) = 11ns
MissTime includes HitTime\(_L1\) + MissPenalty\(_L1\) = HitTime\(_L1\) + AMAT\(_L2\)
Recall: A Summary on Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
  - “Cold” fact of life: not a whole lot you can do about it
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
- **Capacity:**
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- **Coherence** (Invalidation): other process (e.g., I/O) updates memory

Review: Direct Mapped Cache

- Direct Mapped $2^N$ byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = $2^M$)
- Example: 1 KB Direct Mapped Cache with 32 B Blocks
  - Index chooses potential block
  - Tag checked to verify block
  - Byte select chooses byte within block

Review: Fully Associative Cache

- **Fully Associative:** Every block can hold any line
  - Address does not include a cache index
  - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
  - We need N 27-bit comparators
  - Still have byte select to choose from within block

Review: Set Associative Cache

- **N-way set associative:** N entries per Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - Two tags in the set are compared to input in parallel
  - Data is selected based on the tag result

Review: Fully Associative Cache

- **Fully Associative:** Every block can hold any line
  - Address does not include a cache index
  - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
  - We need N 27-bit comparators
  - Still have byte select to choose from within block
Recall: Where is Block Placed in a Cache?

- Example: Block 12 placed in 8 block cache

<table>
<thead>
<tr>
<th>32-Block Address Space:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block no.</td>
</tr>
</tbody>
</table>

Direct mapped: block 12 can go only into block 4 (12 mod 8)

| Block no. | 0 1 2 3 4 5 6 7 |

Set associative: block 12 can go anywhere in set 0 (12 mod 4)

| Block no. | 0 1 2 3 |
| Set 0      |

| Block no. | 0 1 2 3 4 5 6 7 |
| Set 1      |

Fully associative: block 12 can go anywhere

| Block no. | 0 1 2 3 4 5 6 7 |

Miss rates for a workload:

<table>
<thead>
<tr>
<th>Size</th>
<th>2-way LRU Random</th>
<th>4-way LRU Random</th>
<th>8-way LRU Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2% 5.7%</td>
<td>4.7% 5.3%</td>
<td>4.4% 5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9% 2.0%</td>
<td>1.5% 1.7%</td>
<td>1.4% 1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15% 1.17%</td>
<td>1.13% 1.13%</td>
<td>1.12% 1.12%</td>
</tr>
</tbody>
</table>

Which block should be replaced on a miss?

- Easy for Direct Mapped: Only one possibility
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Caching Applied to Address Translation

- Question is one of page locality: does it exist?
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some...

- Question is one of page locality: does it exist?
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some...

- Can we have a TLB hierarchy?
  - Sure: multiple levels at different sizes/speeds

What happens on a write?

- Write through (WT): The information is written to both the block in the cache and to the block in the lower-level memory
- Write back (WB): The information is written only to the block in the cache
  - Modified cache block is written to main memory only when it is replaced
  - Question is block clean or dirty?
- Pros and Cons of each?
  - WT:
    » PRO: read misses cannot result in writes
    » CON: Processor held up on writes unless writes buffered
  - WB:
    » PRO: repeated writes not sent to DRAM processor not held up on writes
    » CON: More complex
      Read miss may require writeback of dirty data

Caching Applied to Address Translation

- Question is one of page locality: does it exist?
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some...

- Can we have a TLB hierarchy?
  - Sure: multiple levels at different sizes/speeds
What Actually Happens on a TLB Miss?

- **Hardware traversed page tables:**
  - On TLB miss, hardware in MMU looks at current page table to fill TLB (may walk multiple levels)
    - If PTE valid, hardware fills TLB and processor never knows
    - If PTE marked as invalid, causes Page Fault, after which kernel decides what to do afterwards

- **Software traversed Page tables (like MIPS):**
  - On TLB miss, processor receives TLB fault
  - Kernel traverses page table to find PTE
    - If PTE valid, fills TLB and returns from fault
    - If PTE marked as invalid, internally calls Page Fault handler

- Most chip sets provide hardware traversal
  - Modern operating systems tend to have more TLB faults since they use translation for many things
  - Examples:
    - shared segments
    - user-level portions of an operating system

### Administrative (1/2)

- **Happy π Day!!!**
  - 40 digits are sufficient to calculate circumference of visible universe to atomic dimensions:

  - Here are 40 decimal places:
    \[ 3.1415926535897932384626433832795028841971 \]

- Best formula for PI is from Ramanujan:
  \[
  \frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103+26390k)}{(k!)^439916k}
  \]

  - Google announced today (3/14/19) that Emma Haruka Iwao had just calculated pi to 31,415,926,535,897 digits (new record…)

### Administrative (2/2)

- Project 1 Peer evaluations are up!
  - It is very important that you fill these out!
  - It is as important as getting to know your TA.
  - The project grades are a zero-sum game; if you do not contribute to the project, your points might be distributed to those who do!

- Project 2 Release Today (3/14)

- Midterm 2: Thursday 4/4
  - Ok, this is a few weeks and after Spring Break
  - Will definitely include Scheduling material (lecture 10)
  - Up to and including some material from lecture 17
  - Will have a Midterm review in early part of that week…. Stay tuned

### Transparent Exceptions: TLB/Page fault

- How to transparently restart faulting instructions?
  - (Consider load or store that gets TLB or Page fault)
  - Could we just skip faulting instruction?
    - No: need to perform load or store after reconnecting physical page
  - Hardware must help out by saving:
    - Faulting instruction and partial state
      - Need to know which instruction caused fault
      - Is single PC sufficient to identify faulting position????
    - Processor State: sufficient to restart user thread
      - Save/restore registers, stack, etc
  - What if an instruction has side-effects?
Consider weird things that can happen

- What if an instruction has side effects?
  - Options:
    » Unwind side-effects (easy to restart)
    » Finish off side-effects (messy!)
  - Example 1: `mov (sp)+,10`
    » What if page fault occurs when write to stack pointer?
    » Did `sp` get incremented before or after the page fault?
  - Example 2: `strcpy (r1), (r2)`
    » Source and destination overlap: can’t unwind in principle!
    » IBM S/370 and VAX solution: execute twice – once read-only

- What about “RISC” processors?
  - For instance delayed branches?
    » Example: `bne somewhere
     ld r1,(sp)`
    » Restart after page fault: need two PCs, PC and nPC!
  - Delayed exceptions:
    » Example: `div r1, r2, r3
     ld r1, (sp)`
    » What if takes many cycles to discover divide by zero, but load has already caused page fault?

Precise Exceptions

- Precise ⇒ state of the machine is preserved as if program executed up to the offending instruction
  - All previous instructions completed
  - Offending instruction and all following instructions act as if they have not even started
  - Same system code will work on different implementations
  - Difficult in the presence of pipelining, out-of-order execution, ...
  - MIPS takes this position

- Imprecise ⇒ system software has to figure out what is where and put it all back together
- Performance goals often lead designers to forsake precise interrupts
  - system software developers, user, markets etc. usually wish they had not done this

- Modern techniques for out-of-order execution and branch prediction help implement precise interrupts

What happens on a Context Switch?

- Need to do something, since TLBs map virtual addresses to physical addresses
  - Address Space just changed, so TLB entries no longer valid!

- Options?
  - Invalidate TLB: simple but might be expensive
    » What if switching frequently between processes?
  - Include ProcessID in TLB
    » This is an architectural solution: needs hardware

- What if translation tables change?
  - For example, to move page from memory to disk or vice versa…
  - Must invalidate TLB entry!
    » Otherwise, might think that page is still in memory!
  - Called “TLB Consistency”

What TLB Organization Makes Sense?

- Needs to be really fast
  - Critical path of memory access
    » In simplest view: before the cache
    » Thus, this adds to access time (reducing cache speed)
  - Seems to argue for Direct Mapped or Low Associativity

- However, needs to have very few conflicts!
  - With TLB, the Miss Time extremely high!
  - This argues that cost of Conflict (Miss Time) is much higher than slightly increased cost of access (Hit Time)

- Thrashing: continuous conflicts between accesses
  - What if use low order bits of page as index into TLB?
    » First page of code, data, stack may map to same entry
  - Need 3-way associativity at least?
  - What if use high order bits as index?
    » TLB mostly unused for small programs
TLB organization: include protection

- How big does TLB actually have to be?
  - Usually small: 128–512 entries
  - Not very big, can support higher associativity
- TLB usually organized as fully-associative cache
  - Lookup is by Virtual Address
  - Returns Physical Address + other info
- What happens when fully-associative is too slow?
  - Put a small (4-16 entry) direct-mapped cache in front
  - Called a “TLB Slice”
- Example for MIPS R3000:

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFA00</td>
<td>0x0003</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>R/W</td>
<td>34</td>
</tr>
<tr>
<td>0x0040</td>
<td>0x0010</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>R</td>
<td>0</td>
</tr>
<tr>
<td>0x0041</td>
<td>0x0011</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>R</td>
<td>0</td>
</tr>
</tbody>
</table>

Example: R3000 pipeline includes TLB “stages”

MIPS R3000 Pipeline

<table>
<thead>
<tr>
<th>Inst Fetch</th>
<th>Dcd/ Reg</th>
<th>ALU / E.A</th>
<th>Memory</th>
<th>Write Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB</td>
<td>I-Cache</td>
<td>RF</td>
<td>Operation</td>
<td>E.A.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TLB
- 64 entry, on-chip, fully associative, software TLB fault handler

Virtual Address Space

<table>
<thead>
<tr>
<th>ASID</th>
<th>V. Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>20</td>
<td>12</td>
</tr>
</tbody>
</table>

0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among 64 user processes without TLB flush

Reducing translation time further

- As described, TLB lookup is in serial with cache lookup:

Overlapping TLB & Cache Access (1/2)

- Main idea:
  - Offset in virtual address exactly covers the “cache index” and “byte select”
  - Thus can select the cached byte(s) in parallel to perform address translation

```
virtual address
physical address
```

- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  - Works because offset available early
Overlapping TLB & Cache Access (1/2)

• Here is how this might work with a 4K cache:
  10 2
  4 bytes
  index

• What if cache size is increased to 8KB?
  – Overlap not complete
  – Need to do something else. See CS152/252
• Another option: Virtual Caches
  – Tags in cache are virtual addresses
  – Translation only happens on cache misses

Putting Everything Together: Address Translation

Putting Everything Together: TLB

Putting Everything Together: Cache
Next Up: What happens when …

Where are places that caching arises in OSes?

- Direct use of caching techniques
  - TLB (cache of PTEs)
  - Paged virtual memory (memory as cache for disk)
  - File systems (cache disk blocks in memory)
  - DNS (cache hostname => IP address translations)
  - Web proxies (cache recently accessed pages)

- Which pages to keep in memory?
  - All-important “Policy” aspect of virtual memory
  - Will spend a bit more time on this in a moment

Impact of caches on Operating Systems

- Indirect - dealing with cache effects
  - Maintaining the correctness of various caches
  - E.g., TLB consistency:
    - With PT across context switches?
    - Across updates to the PT?
- Process scheduling
  - Which and how many processes are active? Priorities?
  - Large memory footprints versus small ones?
  - Shared pages mapped into VAS of multiple processes?
- Impact of thread scheduling on cache performance
  - Rapid interleaving of threads (small quantum) may degrade cache performance
    - Increase average memory access time (AMAT) !!!
- Designing operating system data structures for cache performance

Working Set Model

- As a program executes it transitions through a sequence of “working sets” consisting of varying sized subsets of the address space
Cache Behavior under WS model

- Amortized by fraction of time the Working Set is active
- Transitions from one WS to the next
- Capacity, Conflict, Compulsory misses
- Applicable to memory caches and pages. Others?

Another model of Locality: Zipf

- Likelihood of accessing item of rank \( r \) is \( \alpha \frac{1}{r^\alpha} \)
- Although rare to access items below the top few, there are so many that it yields a “heavy tailed” distribution
- Substantial value from even a tiny cache
- Substantial misses from even a very large cache

Demand Paging

- Modern programs require a lot of physical memory
  - Memory per system growing faster than 25%-30%/year
- But they don’t use all their memory all of the time
  - 90-10 rule: programs spend 90% of their time in 10% of their code
  - Wasteful to require all of user’s code to be in memory
- Solution: use main memory as cache for disk

Disk is larger than physical memory

- In-use virtual memory can be bigger than physical memory
- Combined memory of running processes much larger than physical memory
  - More programs fit into memory, allowing more concurrency
- Principle: Transparent Level of Indirection (page table)
  - Supports flexible placement of physical data
  - Data could be on disk or somewhere across network
  - Variable location of data transparent to user program
  - Performance issue, not correctness issue
Demand Paging is Caching

- Since Demand Paging is Caching, must ask:
  - What is block size?
    » 1 page
  - What is organization of this cache (i.e. direct-mapped, set-associative, fully-associative)?
    » Fully associative: arbitrary virtual→physical mapping
  - How do we find a page in the cache when look for it?
    » First check TLB, then page-table traversal
  - What is page replacement policy? (i.e. LRU, Random...)
    » This requires more explanation... (kinda LRU)
  - What happens on a miss?
    » Go to lower level to fill miss (i.e. disk)
  - What happens on a write? (write-through, write back)
    » Definitely write-back. Need dirty bit!

Review: What is in a PTE?

- What is in a Page Table Entry (or PTE)?
  - Pointer to next-level page table or to actual page
  - Permission bits: valid, read-only, read-write, write-only
- Example: Intel x86 architecture PTE:
  - Address same format previous slide (10, 10, 12-bit offset)
  - Intermediate page tables called "Directories"

| Page Frame Number (Physical Page Number) | Free (OS) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| P: Present (same as “valid” bit in other architectures) | W: Writeable | U: User accessible | PWT: Page write transparent: external cache write-through | PCD: Page cache disabled (page cannot be cached) | A: Accessed: page has been accessed recently | D: Dirty (PTE only): page has been modified recently | L: L=1⇒4MB page (directory only). Bottom 22 bits of virtual address serve as offset |
| 31-12 | 11-9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Demand Paging Mechanisms

- PTE helps us implement demand paging
  - Valid ⇒ Page in memory, PTE points at physical page
  - Not Valid ⇒ Page not in memory; use info in PTE to find it on disk when necessary
- Suppose user references page with invalid PTE?
  - Memory Management Unit (MMU) traps to OS
    » Resulting trap is a "Page Fault"
  - What does OS do on a Page Fault?:
    » Choose an old page to replace
    » If old page modified ("D=1"), write contents back to disk
    » Change its PTE and any cached TLB to be invalid
    » Load new page into memory from disk
    » Update page table entry, invalidate TLB for new entry
    » Continue thread from original faulting location
- TLB for new page will be loaded when thread continued!
  - While pulling pages off disk for one process, OS runs another process from ready queue
    » Suspended process sits on wait queue

Summary: Steps in Handling a Page Fault
Summary (1/2)

• The Principle of Locality:
  – Program likely to access a relatively small portion of the address space at any instant of time.
    » Temporal Locality: Locality in Time
    » Spatial Locality: Locality in Space

• Three (+1) Major Categories of Cache Misses:
  – Compulsory Misses: sad facts of life. Example: cold start misses.
  – Conflict Misses: increase cache size and/or associativity
  – Capacity Misses: increase cache size
  – Coherence Misses: Caused by external processors or I/O devices

• Cache Organizations:
  – Direct Mapped: single block per set
  – Set associative: more than one block per set
  – Fully associative: all entries equivalent

Summary (2/2)

• “Translation Lookaside Buffer” (TLB)
  – Small number of PTEs and optional process IDs (< 512)
  – Fully Associative (Since conflict misses expensive)
  – On TLB miss, page table must be traversed and if located PTE is invalid, cause Page Fault
  – On change in page table, TLB entries must be invalidated
  – TLB is logically in front of cache (need to overlap with cache access)

• Precise Exception specifies a single instruction for which:
  – All previous instructions have completed (committed state)
  – No following instructions nor actual instruction have started

• Can manage caches in hardware or software or both
  – Goal is highest hit rate, even if it means more complex cache management