Recall: Caching Concept

- **Cache**: a repository for copies that can be accessed more quickly than the original
  - Make frequent case fast and infrequent case less dominant
- Caching underlies many techniques used today to make computers fast
  - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc…

**Tribute**

- Sir Maurice Wilkes, inventor of EDSAC, Microprogramming, and the Translation Look-aside Buffer

**Big Really Basic Concept #1**

- “Large” memories are “slow”, “Fast” memories are “small”
- How do we create the illusion of a “large fast” memory? – on average.
Recall: In Machine Structures (eg. 61C) …

- Caching is the key to memory system performance

Average Access time = (Hit Rate \times \text{HitTime}) + (Miss Rate \times \text{MissTime})

\[ \text{HitRate} + \text{MissRate} = 1 \]

\[ \begin{align*}
\text{HitRate} = 90\% &\Rightarrow \text{Avg. Access Time} = (0.9 \times 10) + (0.1 \times 100) = 19\text{ns} \\
\text{HitRate} = 99\% &\Rightarrow \text{Avg. Access Time} = (0.99 \times 10) + (0.01 \times 100) = 10.9\text{ns}
\end{align*} \]

Big Really Really Basic Concept #2

- How many distinct values can be expressed in K bits?
- How many addresses can be expressed in K bits?
- How large a “namespace” can be addressed by N bits?

Big Basic Concept #3

- A program operates in a Virtual Address Space distinct from the Physical Address Space of the hardware in the machine.

Recall: How is the Translation Accomplished?

- What, exactly happens inside MMU?
  - Hardware Tree Traversal
    - For each virtual address traverses the page table in hardware
    - Generates a “Page Fault” if it encounters invalid PTE
      - Fault handler will decide what to do
      - More on this next lecture
    - Pros: Relatively fast (but still many memory accesses!)
    - Cons: Inflexible, Complex hardware
  - Another possibility: Software
    - Each traversal done in software
    - Pros: Very flexible
    - Cons: Every translation must invoke Fault!
- In fact, need way to cache translations for either case!
How to map VA => PA?

- What should be the unit of mapping? How?
- How many?

<table>
<thead>
<tr>
<th>Virtual addr 0</th>
<th>Physical addr 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual addr 1</td>
<td>Physical addr 1</td>
</tr>
<tr>
<td>Virtual addr m-1</td>
<td>Physical addr m-1</td>
</tr>
</tbody>
</table>

Virtual address 64 bits  ➔  map  ➔  Physical address 38 bits

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...

What Actually Happens on a TLB Miss? (1/2)

- 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

What Actually Happens on a TLB Miss? (2/2)

- 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
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!

Really really basic question #4

- How many virtual-to-physical translations occur per instruction?
- How much time do we have to do them?
- How many memory references occur per instruction?
- What is the difference between an interrupt and a fault (or trap) ?
  » Both care called Exceptions.
- Which kind is a Page Fault?
- A TLB miss?

Recall: User→Kernel (System Call)

- Can’t let inmate (user) get out of padded cell on own
  » Would defeat purpose of protection!
  » So, how does the user program get back into kernel?

- System call: Voluntary procedure call into kernel
  » Hardware for controlled User→Kernel transition
  » Can any kernel routine be called?
    » No! Only specific ones.
  » System call ID encoded into system call instruction
    » Index forces well-defined interface with kernel

But on a page-fault there is no "system call" instruction!
Transparent Exceptions: TLB/Page fault (2/2)

- 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)`
    » Precise exception state consists of two PCs: PC and nPC (next PC)
  - 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 \( \Rightarrow \) 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 \( \Rightarrow \) system software has to figure out what is where and put it all back together
- Performance goals often lead to forsaking 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

Recall: TLB Organization

- 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
Basic Concept #5

- If we use $K$ bits out of an $N$ bit address, how many of the $2^N$ “things” in the address space can we refer to?
- Which ones?

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

- Overlapping TLB & Cache Access

- Here is how this might work with a 4K cache:

  - 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
Where are all 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 (1/2)

- Indirect - dealing with cache effects (e.g., sync state across levels)
  - 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 caches on Operating Systems (2/2)

- 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 $\alpha 1/r^a$
- 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
Illusion of Infinite Memory (1/2)

- 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

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!

Recall: What is in a Page Table Entry

- 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”

<table>
<thead>
<tr>
<th>Page Frame Number (Physical Page Number)</th>
<th>Free (OS)</th>
<th>0</th>
<th>L</th>
<th>D</th>
<th>A</th>
<th>R</th>
<th>C</th>
<th>P</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-12</td>
<td>11-9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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
**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 continues!
    - While pulling pages off disk for one process, OS runs another process from ready queue
      - Suspended process sits on wait queue

**Summary**

- A cache of translations called a “Translation Lookaside Buffer” (TLB)
  - Relatively 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