Review: Single-Level Translation

- Advantages
  - Low translation overhead
  - Simplicity
- Disadvantages
  - Large page tables
    - E.g., 32b address space, 4KB pages → up to $2^{10} = 1$ million entries for each process
  - Expensive to share memory
    - E.g., 4KB pages, want to share 100MB → need to update 25,000 entries in page table

Review: Multi-level Translation

- What about a tree of tables?
  - Lowest level page table—memory still allocated with bitmap
  - Higher levels often segmented
- Could have any number of levels. Example (top segment):

Review: Two-level page table

- Tree of Page Tables
- Tables fixed size (1024 entries)
  - On context-switch: save single PageTablePtr register
  - Sometimes, top-level page tables called “directories” (Intel)
  - Each entry called a (surprise!) Page Table Entry (PTE)
Goals for Today

• Finish discussion of both Address Translation and Protection
• Caching and TLBs

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne

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”
  - Present (same as “valid” bit in other architectures)
  - Writeable
  - User accessible
  - Page write transparent: external cache write-through
  - Page cache disabled (page cannot be cached)
  - Accessed: page has been accessed recently
  - Dirty (PTE only): page has been modified recently
  - 4MB page (directory only).

Examples of how to use a PTE

• How do we use the PTE?
  - Invalid PTE can imply different things:
    » Region of address space is actually invalid or
    » Page/directory is just somewhere else than memory
  - Validity checked first
• Usage Example: Demand Paging
  - Keep only active pages in memory
  - Place others on disk and mark their PTEs invalid
• Usage Example: Copy on Write
  - UNIX fork gives copy of parent address space to child
  - Address spaces disconnected after child created
  - How to do this cheaply?
    » Make copy of parent’s page tables (point at same memory)
    » Mark entries in both sets of page tables as read-only
  - Page fault on write creates two copies
• Usage Example: Zero Fill On Demand
  - New data pages must carry no information (say be zeroed)
  - Mark PTEs as invalid; page fault on use gets zeroed page
  - Often, OS creates zeroed pages in background

How is the translation accomplished?

• What, exactly happens inside MMU?
  - One possibility: Hardware Tree Traversal
    » For each virtual address, takes page table base pointer
    » And traverses the page table in hardware
  - Generates a “Page Fault” if it encounters invalid PTE
    » More on this next lecture
    » 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!
**Dual-Mode Operation**

- Can Application modify its own translation tables?
  - If it could, could get access to all of physical memory
  - Has to be restricted somehow
- To Assist with Protection, **Hardware** provides at least two modes (Dual-Mode Operation):
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode (Normal program mode)
  - Mode set with bits in special control register only accessible in kernel-mode
- Intel processor actually has four "rings" of protection:
  - PL (Privileged Level) from 0 – 3
    - PL0 has full access, PL3 has least
  - Privilege Level set in code segment descriptor (CS)
  - Typical OS kernels on Intel processors only use PLO ("user") and PL3 ("kernel")

**For Protection, Lock User-Programs in Asylum**

- Idea: Lock user programs in padded cell with no exit or sharp objects
  - Cannot change mode to kernel mode
  - User cannot modify page table mapping
  - Limited access to memory: cannot adversely affect other processes
    - Side-effect: Limited access to memory-mapped I/O operations (I/O that occurs by reading/writing memory locations)
  - Limited access to interrupt controller
- A couple of issues
  - How to share CPU between kernel and user programs?
    - Kinda like both the inmates and the warden in asylum are the same person. How do you manage this?
  - How do programs interact?
  - How does one switch between kernel and user modes?
    - OS → user (kernel → user mode): getting into cell
    - User → OS (user → kernel mode): getting out of cell

**How to get from Kernel→User**

- What does the kernel do to create a new user process?
  - Allocate and initialize address-space control block
  - Read program off disk and store in memory
  - Allocate and initialize translation table
    - Point at code in memory so program can execute
    - Possibly point at statically initialized data
  - Run Program:
    - Set machine registers
    - Set hardware pointer to translation table
    - Set processor status word for user mode
    - Jump to start of program
- How does kernel switch between processes?
  - Same saving/restoring of registers as before
  - Save/restore PSL (hardware pointer to translation table)

**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
System Call Continued

• What are some system calls?
  - I/O: open, close, read, write, lseek
  - Files: delete, mkdir, rmdir, truncate, chown, chgrp, ...
  - Process: fork, exit, wait (like join)
  - Network: socket create, set options

• Are system calls constant across operating systems?
  - Not entirely, but there are lots of commonalities
  - Also some standardization attempts (POSIX)

• What happens at beginning of system call?
  » On entry to kernel, sets system to kernel mode
  » Handler address fetched from table/Handler started

• System Call argument passing:
  - In registers (not very much can be passed)
  - Write into user memory, kernel copies into kernel mem
    » User addresses must be translated
    » Kernel has different view of memory than user
  - Every Argument must be explicitly checked!

Closing thought: Protection without Hardware

• Does protection require hardware support for translation and dual-mode behavior?
  - No: Normally use hardware, but anything you can do in hardware can also do in software (possibly expensive)

• Protection via Strong Typing
  - Restrict programming language so that you can’t express program that would trash another program
  - Loader needs to make sure that program produced by valid compiler or all bets are off
  - Example languages: LISP, Ada, Modula-3 and Java

• Protection via software fault isolation:
  - Language independent approach: have compiler generate object code that provably can’t step out of bounds
    » Compiler puts in checks for every “dangerous” operation (loads, stores, etc). Again, need special loader.
    » Alternative, compiler generates “proof” that code cannot do certain things (Proof Carrying Code)
  - Or: use virtual machine to guarantee safe behavior (loads and stores recompiled on fly to check bounds)

User→Kernel (Exceptions: Traps and Interrupts)

• A system call instruction causes a synchronous exception (or “trap”)
  - In fact, often called a software “trap” instruction

• Other sources of Synchronous Exceptions:
  - Divide by zero, Illegal instruction, Bus error (bad address, e.g. unaligned access)
  - Segmentation Fault (address out of range)
  - Page Fault (for illusion of infinite-sized memory)

• Interrupts are Asynchronous Exceptions
  - Examples: timer, disk ready, network, etc...
  - Interrupts can be disabled, traps cannot!

• On system call, exception, or interrupt:
  - Hardware enters kernel mode with interrupts disabled
  - Saves PC, then jumps to appropriate handler in kernel
  - For some processors (x86), processor also saves registers, changes stack, etc.

  Actual handler typically saves registers, other CPU state, and switches to kernel stack

Administrivia

• Midterm in 1 week:
  - Monday, 3/9, 3:30-6:30pm, (277 Cory Hall - this room!)
  - Should be 2 hour exam with extra time
  - Closed book, one page of hand-written notes (both sides)

• No class on day of Midterm
  - Extra Office Hours: Next tuesday 1:00-3:00

• Midterm Topics
  - Topics: Everything up to Thursday 3/4
  - History, Concurrency, Multithreading, Synchronization, Protection/Address Spaces, TLBs

• Make sure to fill out Group Evaluations!

• Project 2
  - Initial Design Document due Thursday 3/4
  - Look at the lecture schedule to keep up with due dates!
### 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 of the techniques that are used today to make computers fast
  - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc...
- Only good if:
  - Frequent case frequent enough and
  - Infrequent case not too expensive
- Important measure: Average Access time = (Hit Rate x Hit Time) + (Miss Rate x Miss Time)

### Why Bother with Caching?

- Processor-DRAM Memory Gap (latency)
  - **"Moore’s Law"** (really Joy’s Law)
  - **"Less’ Law?"**
  - Processor-Memory Performance Gap: (grows 50% / year)
  - Processor-DRAM Memory Gap (latency)

### Another Major Reason to Deal with Caching

- Cannot afford to translate on every access
  - At least three DRAM accesses per actual DRAM access
  - Or: perhaps I/O if page table partially on disk!
- Even worse: What if we are using caching to make memory access faster than DRAM access?
- **Solution? Cache translations!**
  - Translation Cache: TLB ("Translation Lookaside Buffer")

### Why Does Caching Help? Locality!

- **Temporal Locality** (Locality in Time):
  - Keep recently accessed data items closer to processor
- **Spatial Locality** (Locality in Space):
  - Move contiguous blocks to the upper levels
Memory Hierarchy of a Modern Computer System

- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology

<table>
<thead>
<tr>
<th>Processor</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datapath</td>
<td>On-Chip Cache</td>
</tr>
<tr>
<td></td>
<td>Registers</td>
</tr>
<tr>
<td></td>
<td>Main Memory (DRAM)</td>
</tr>
<tr>
<td></td>
<td>Secondary Storage (Disk)</td>
</tr>
<tr>
<td></td>
<td>Tertiary Storage (Tape)</td>
</tr>
</tbody>
</table>

- Speed (ns): 1s, 10s-100s, 100s, 10,000,000s (10s ms), 10,000,000,000s (10s sec)
- Size (bytes): 100s, Ms, Gs, Ts

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

How is a Block found in a Cache?

- Index Used to Lookup Candidates in Cache
  - Index identifies the set
- Tag used to identify actual copy
  - If no candidates match, then declare cache miss
- Block is minimum quantum of caching
  - Data select field used to select data within block
  - Many caching applications don’t have data select field

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

<table>
<thead>
<tr>
<th>Cache Tag</th>
<th>Cache Index</th>
<th>Byte Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x50</td>
<td>0x001</td>
<td>0x01</td>
</tr>
<tr>
<td>Valid Bit</td>
<td>Cache Tag</td>
<td>Cache Data</td>
</tr>
<tr>
<td>1</td>
<td>0x50</td>
<td>Byte 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Byte 31</td>
</tr>
</tbody>
</table>

3/2/10 CS162 ©UCB Spring 2010 Lec 13.21
Review: Set Associative Cache

- N-way set associative: N entries per Cache Index
  - N direct mapped caches operate 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

Where does a Block Get Placed in a Cache?

- Example: Block 12 placed in 8 block cache

Review: 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)
Review: What happens on a write?

- **Write through**: The information is written to both the block in the cache and to the block in the lower-level memory.
- **Write back**: 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
    - **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

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!
### 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 Caching (TLB)

**PTE:** Page Table Entries
- Includes physical page number
- Control info (valid bit, writeable, dirty, user, etc)

**A cache of translations called the “Translation Lookaside Buffer” (TLB):**
- Relatively small number of entries (< 512)
- Fully Associative (Since conflict misses expensive)
- TLB entries contain PTE and optional process ID

**On TLB miss, page table must be traversed**
- If located PTE is invalid, cause Page Fault

**On context switch/change in page table**
- TLB entries must be invalidated somehow