

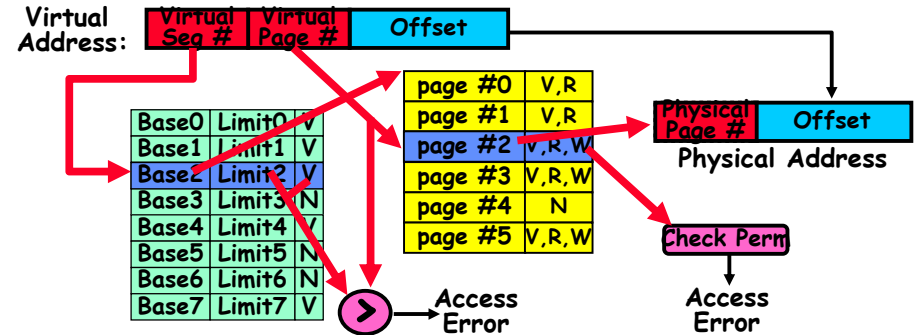
CS162 Operating Systems and Systems Programming Lecture 13

Address Translation (cont') Caches and TLBs

October 15, 2007
Prof. John Kubiatowicz
<http://inst.eecs.berkeley.edu/~cs162>

Review: Multi-level Translation

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



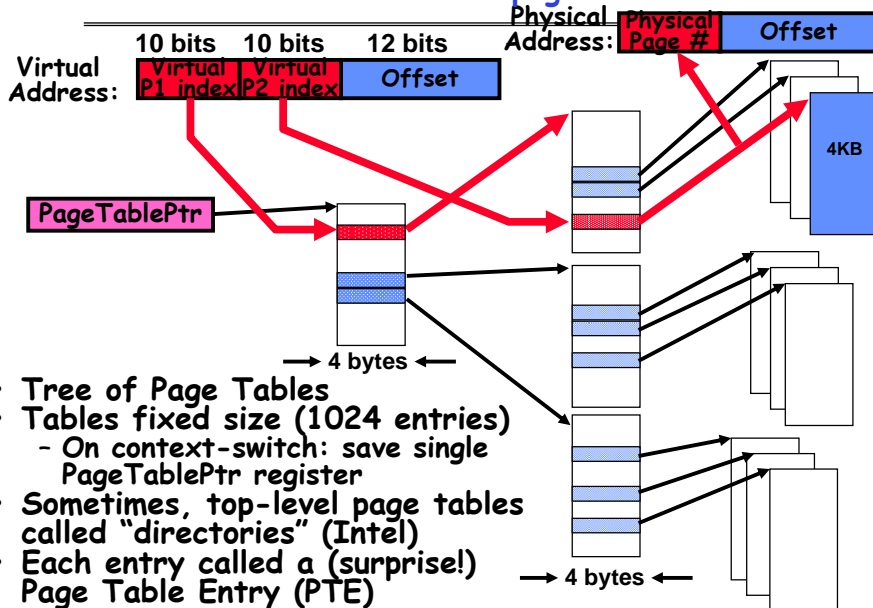
- What must be saved/restored on context switch?
 - Contents of top-level segment registers (for this example)
 - Pointer to top-level table (page table)

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.2

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)

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.3

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

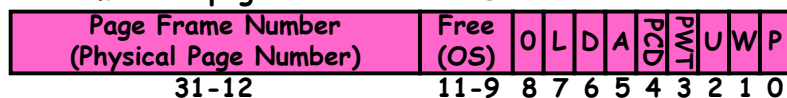
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.4

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:



- 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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.5

Examples of how to use a PTE

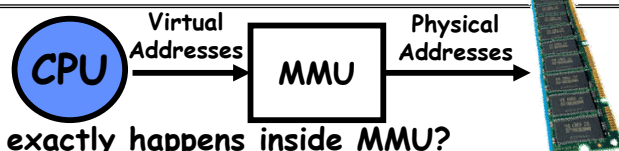
- How do we use the PTE?
 - Invalid PTE can imply different things:
 - » Region of address space is actually invalid or
 - » Page/directory is just somewhere else than memory
 - Validity checked first
 - » OS can use other (say) 31 bits for location info
- Usage Example: Demand Paging
 - Keep only active 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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.6

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
 - » 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!**

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.7

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 (Privilege Level) from 0 - 3
 - » PLO has full access, PL3 has least
 - Privilege Level set in code segment descriptor (CS)
 - Mirrored "IOPL" bits in condition register gives permission to programs to use the I/O instructions
 - Typical OS kernels on Intel processors only use PLO ("user") and PL3 ("kernel")

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.8

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 effect 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
 - What else needs to be protected?
- **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



10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.9

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)

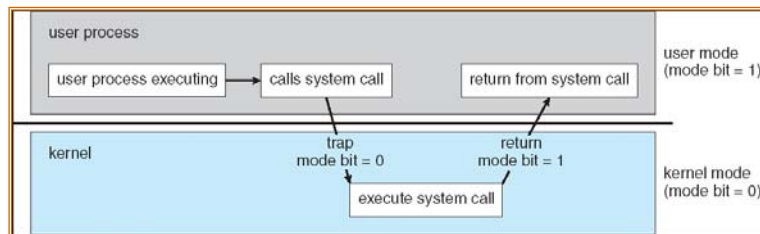
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.10

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.11

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!

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.12

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.13

Additions to MIPS ISA to support Exceptions?

- Exception state is kept in "Coprocessor 0"
 - Use mfc0 read contents of these registers:
 - » **BadVAddr (register 8)**: contains memory address at which memory reference error occurred
 - » **Status (register 12)**: interrupt mask and enable bits
 - » **Cause (register 13)**: the cause of the exception
 - » **EPC (register 14)**: address of the affected instruction
- 15 8 5 4 3 2 1 0
- | | | | | | | | |
|--------|------|-----|------|-----|---|---|---|
| Status | Mask | k | e | k | e | k | e |
| | | old | prev | cur | | | |
- Status Register fields:
 - **Mask**: Interrupt enable
 - » 1 bit for each of 5 hardware and 3 software interrupts
 - **k** = kernel/user: 0⇒kernel mode
 - **e** = interrupt enable: 0⇒interrupts disabled
 - **Exception⇒6 LSB shifted left 2 bits, setting 2 LSB to 0**:
 - » run in kernel mode with interrupts disabled

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.14

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)**

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.15

Administrivia

- Exam was too long
 - Sorry about that...
 - If it is any consolation, everyone in same boat
- Still grading exam
 - Will announce results as soon as possible
 - Also will get solutions up very soon!
- Project 2 is started!
 - Design document due date is Wednesday (10/17) at 11:59pm
 - Always keep up with the project schedule by looking on the "Lectures" page
- Make sure to come to sections!
 - There will be a lot of information about the projects that I cannot cover in class
 - Also supplemental information and detail that we don't have time for in class

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.16

Review: Monitor Summary

Basic structure of monitor-based program:

```
lock
while (need to wait) {
    condvar.wait();
}
unlock
```

} Check and/or update
state variables
Wait if necessary

do something so no need to wait

lock

```
condvar.signal();
```

} Check and/or update
state variables

unlock

Scheduling:

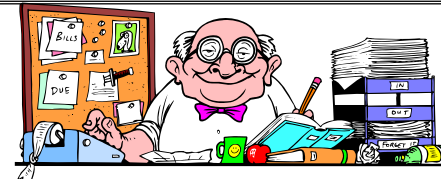
- Mesa: signaler puts waiter on ready queue, keeps CPU and lock (and keeps running)
- Hoare: signaler gives CPU and lock to waiting thread, signaler sleeps. Waiter returns CPU and lock to signaler when waiter releases lock or waits

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.17

Caching Concept



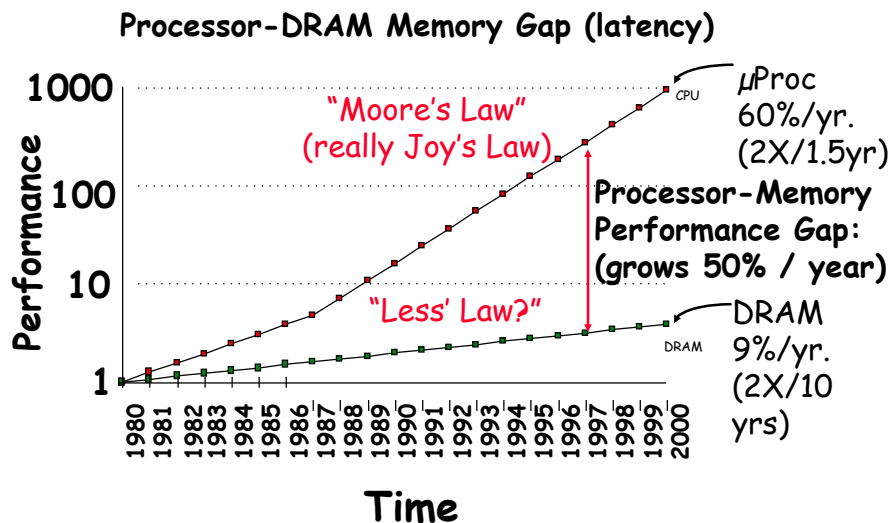
- **Cache:** a repository for copies that can be accessed more quickly than the original
 - Make frequent case fast and infrequent case less dominant
- Caching underlies many of the techniques that are used today to make computers fast
 - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc...
- Only good if:
 - Frequent case frequent enough and
 - Infrequent case not too expensive
- Important measure: Average Access time = $(\text{Hit Rate} \times \text{Hit Time}) + (\text{Miss Rate} \times \text{Miss Time})$

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.18

Why Bother with Caching?

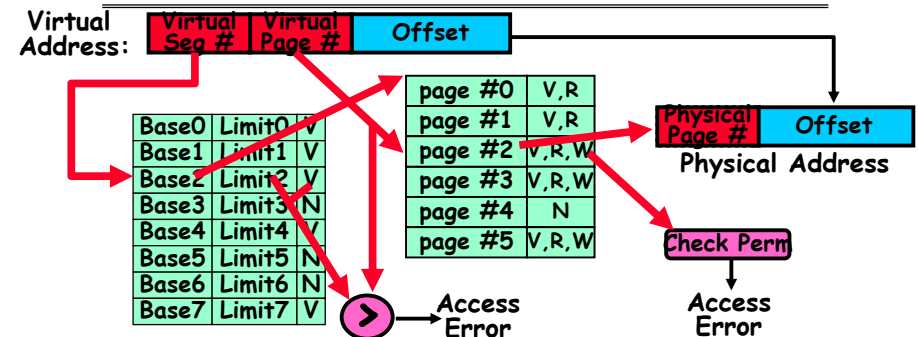


10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.19

Another Major Reason to Deal with Caching



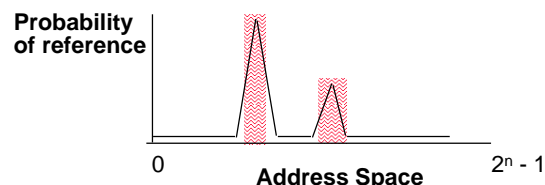
- Cannot afford to translate on every access
 - At least three DRAM accesses per actual DRAM access
 - Or: perhaps I/O if page table partially on disk!
- Even worse: What if we are using caching to make memory access faster than DRAM access???
- Solution? Cache translations!
 - Translation Cache: TLB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

10/15/07

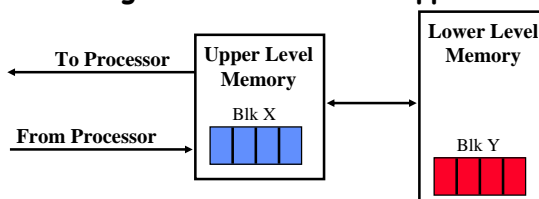
Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.20

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



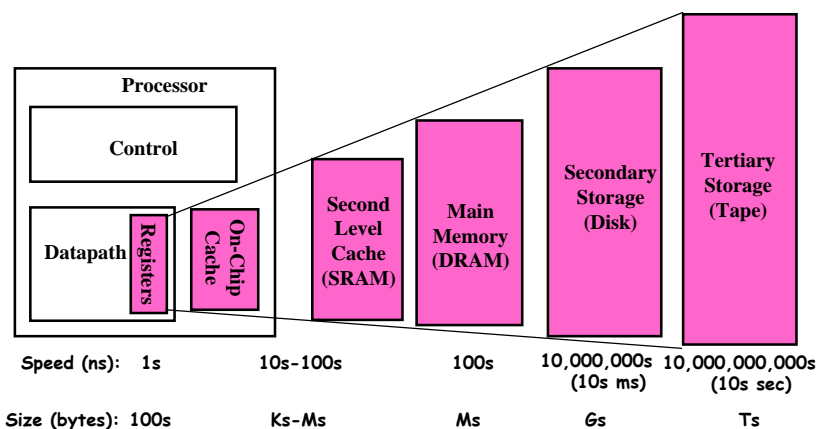
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.21

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



10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.22

A Summary on Sources of Cache Misses

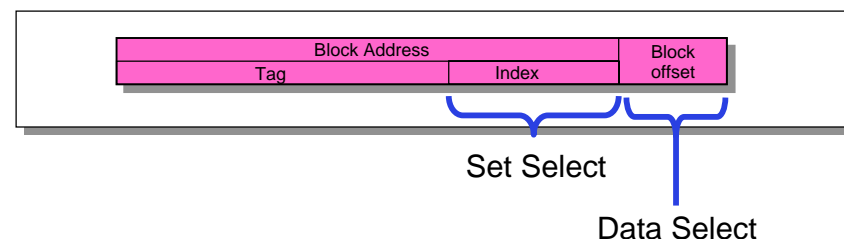
- **Compulsory** (cold start or process migration, first reference): first access to a block
 - "Cold" fact of life: not a whole lot you can do about it
 - Note: If you are going to run "billions" of instructions, Compulsory Misses are insignificant
- **Capacity:**
 - Cache cannot contain all blocks accessed 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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.23

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

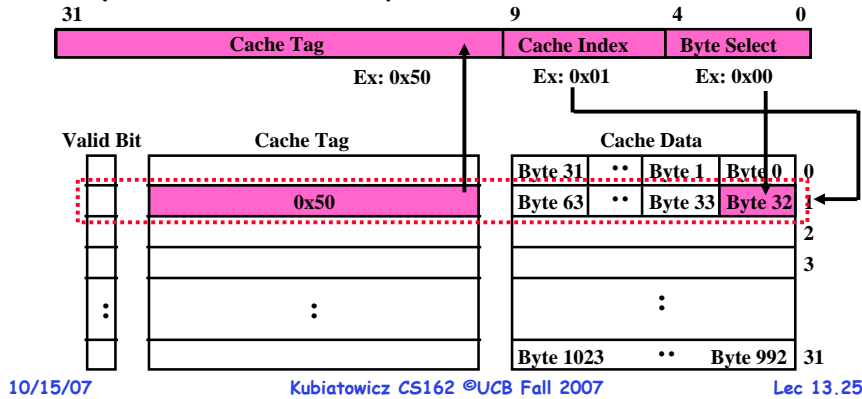
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.24

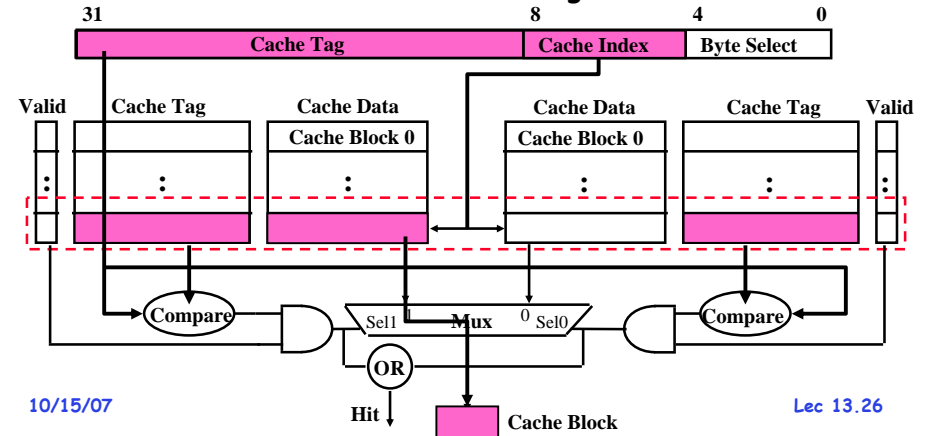
Review: Direct Mapped Cache

- **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



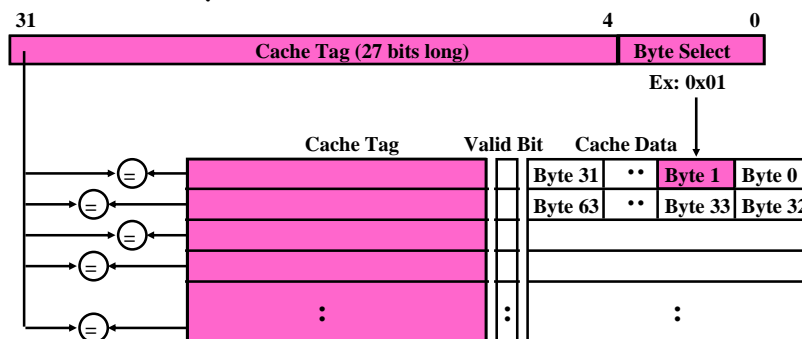
Review: Set Associative Cache

- **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

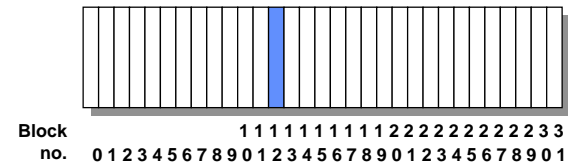
- **Fully Associative:** Every block can hold any line
 - Address does not include a cache index
 - Compare Cache Tags of all Cache Entries in Parallel
- **Example: Block Size=32B blocks**
 - We need N 27-bit comparators
 - Still have byte select to choose from within block



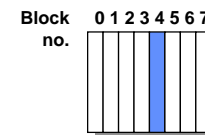
Where does a Block Get Placed in a Cache?

- **Example: Block 12 placed in 8 block cache**

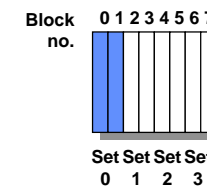
32-Block Address Space:



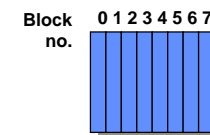
Direct mapped:
block 12 can go only into block 4 (12 mod 8)



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



Fully associative:
block 12 can go anywhere



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)

Size	2-way		4-way		8-way	
	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.29

Review: What happens on a write?

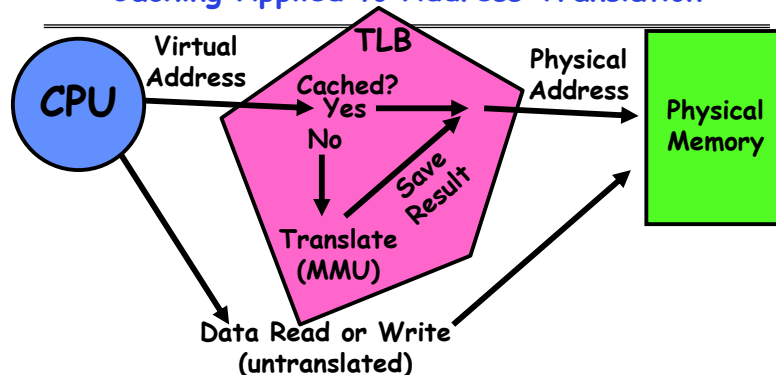
- **Write through:** The information is written to both the block in the cache and to the block in the lower-level memory
- **Write back:** The information is written only to the block in the cache.
 - Modified cache block is written to main memory only when it is replaced
 - Question is block clean 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 processor not held up on writes
 - » CON: More complex
Read miss may require writeback of dirty data

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.30

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.31

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.32

What happens on a Context Switch?

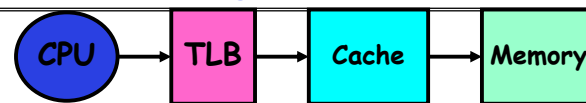
- Need to do something, since TLBs map virtual addresses to physical addresses
 - Address Space just changed, so TLB entries no longer valid!
- Options?
 - Invalidate TLB: simple but might be expensive
 - » What if switching frequently between processes?
 - Include ProcessID in TLB
 - » This is an architectural 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!

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.33

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.34

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:

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.35

Example: R3000 pipeline includes TLB "stages"

MIPS R3000 Pipeline

Inst Fetch	Dcd/ Reg	ALU / E.A	Memory	Write Reg
TLB	I-Cache	RF	Operation	WB
		E.A.	TLB	D-Cache

TLB

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

Virtual Address Space

ASID	V. Page Number	Offset
6	20	12

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

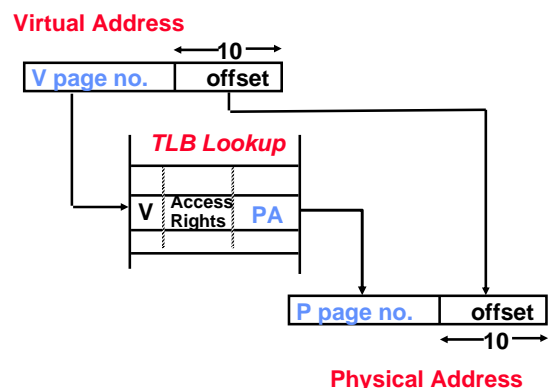
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.36

Reducing translation time further

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



- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
 - Works because offset available early

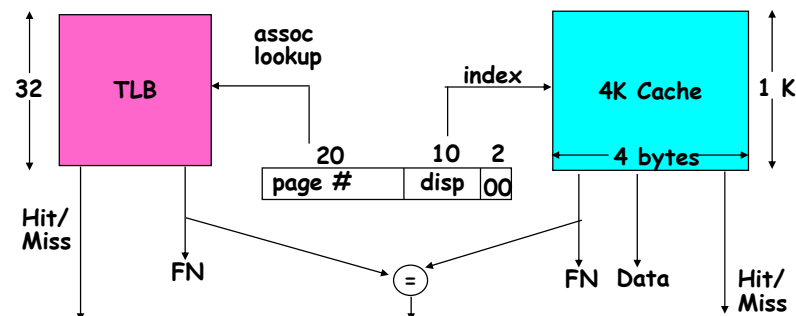
10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.37

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.38

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

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.39

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 a "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
- TLB is logically in front of cache
 - Thus, needs to be overlapped with cache access to be really fast

10/15/07

Kubiatowicz CS162 ©UCB Fall 2007

Lec 13.40