UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Intro to Operating Systems

A. Arpaci-Dusseau
Spring 2000

Translation Lookaside Buffers (TLBs)

Questions answered in these notes

- How is a TLB used to speed up address translations?
- How is a TLB organized? What should be done on a context-switch?
- What do TLBs look like in real systems?
- What is the purpose of the copy-on-write optimization?
- For architects: How does the TLB interact with cache lookups?

Translation Look-Aside Buffer (TLB)

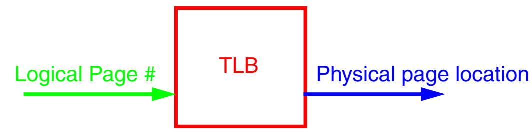
Locality: Process references few unique pages in time interval

TLB (cache) stores a few translation table entries

- Typical sizes: 64 to 2K entries
- Index by logical segment+page --> Physical base address of page

On each memory reference: Check TLB for page

- If present, fetch physical base address and append page offset
- Else, go through segment and page tables to get base address
    - Update TLB for next access
    - (TLB discards one of old entry)



Organize TLB to Find Entry Quickly?

Direct mapped

- Restrict location for each logical page to one TLB entry
    - Calculate TLB entry with bits of logical page number
    - Check remainder of logical page number for match
- Pros: Easy to implement, fast lookups
- Cons: Conflicts when two logical pages map to same TLB entry

Set associative

- Restrict location for each logical page to a **set** of TLB entries
    - Set size is typically 2, 4, or 8 (*n*-way associative)
- Pros: Easy and fast, must have (set size + 1) conflicts
- Cons: TLB replacement scheme

Fully associative

- Check all TLB entries in parallel for logical page number
- Pros and Cons: No conflicts but expensive hardware

Mapping Logical Addresses to TLB Entries

Some hash functions are better than others

- Better to use low-order page number bits or high-order bits?

Which TLB entry to replace?

- Direct mapped: Only one choice
- Associativity:
    - Random
    - Least Recently Used (LRU)

Other TLB options

- Separate TLBs for instructions and data
- Seperate TLB for user and OS

TLBs have been extremely successful in practice

- 98% hit ratio is typical for 128 entries

TLB on Context Switch

TLB usually hidden from operating system

- Possible exception: context switches
- Logical addresses across processes do not have same physical address

Two alternatives

- Flush TLB at each context switch
  - Automatically by hardware (Pentium II)
  - Extra instruction
- Store a Process ID (PID) in each TLB entry

Sharing

- What if two TLB entries have same physical address?
- What if two TLB entries have same logical address?

Example #1

MIPS R2000/R3000

CPU used in DecStations and SGIs workstations

Addresses are 32 bits: 12 bit page offset (i.e 4K pages)

TLB has 64 entries, fully associative

TLB entry format: (64 bits)

20	6	6	20	1	1	1	1	8
VPN	AID	0	PFN	N	D	V	G	0

VPN — Virtual page number
AID — Address space ID for entry
PFN — Physical address of the page
D — Dirty bit, page has been modified
N — Don't cache memory addresses
V — Entry is valid
G — Global, valid for any AID

TLB Example #2

64-bit SPARC v9 Architecture

- 44-bit virtual address space, 41-bit physical address space
- 64-entry Instruction TLB (iTLB) and 64-entry Data TLB (dTLB)
- Multiple page sizes (8KB, 64KB, 512KB, 4MB)
- Two segments (stack vs. data and text)

TLB entry format (128 bits)

1	2	13	6	42
G		AID		VA_tag <63:22>

1	2	1	1	9	9	28	6	1	1	1	1	1	1	1
V	Size		IE	Soft	Diag	PA<40:13>	Soft	L	CP	CV	E	P	W	G

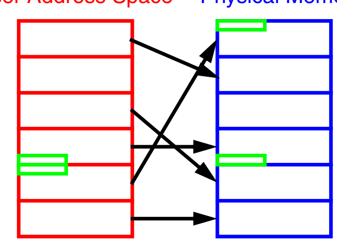
IE - Invert Endianness
L - Lock
CP, CV - Cacheable
E - Side-effect associated with memory (do not prefetch)
P - Priveleged
W - Writable

OS Interactions with User Processes

How does OS get information from user memory?

- User passes the OS logical addresses
  - OS must check logical address is valid for calling process
- Contiguous addresses in logical space may not be contiguous physically
  - Split I/O operations into multiple blocks

User Address Space Physical Memory



Optimization: Copy on Write

- Unix process forks child process
  - OS copies data and text segment of parent to child
  - Each has own address space, though same contents initially
  - Problem: Large overhead if large address space

- Solution: Copy on write
  - Set up page tables to initially point to same physical memory
  - Set up permissions in page table entry as **private**
  - First write to page:

    Copy page in physical memory

    Change page table entry (pointer and permissions)

- Lazy creation of new pages in memory

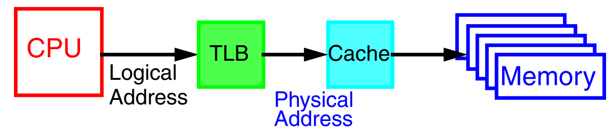
OS Interactions with Hardware Cache

Cache: Component of memory hierarchy

- Small, fast, expensive memory
- Principle of locality: Data and instructions not accessed randomly
- Hold memory anticipate will reference soon in future

Is cache indexed with logical or physical addresses?

Physical Address



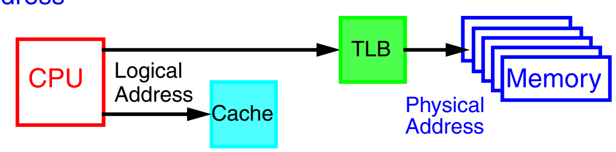
- Problem: TLB lookup and cache access are on critical path

  Want to complete in single clock cycle

  Implicitation: May slowdown clock speed

Indexing Cache with Logical Addresses

Logical Address



Advantage: TLB lookup is not on critical path

Disadvantages

- Synonyms or aliases

  Multiple logical addresses share same physical address

  Inconsistent copies in cache

- Must distinguish between logical addresses across address spaces

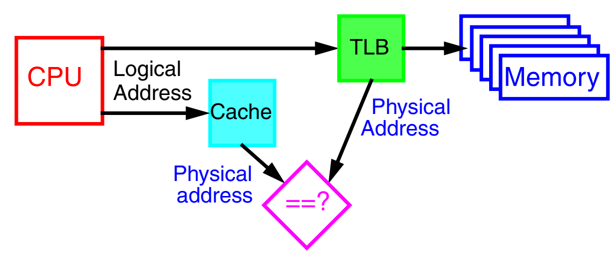
  Flush cache on context-switch

  Associate PID with each cache entry

TLB and Cache Solution

Cache is logically indexed, physically tagged

- Use part of logical address to find possible cache entries
- Compare physical address tag stored with entry to physical address



Index cache with what part of logical address?

- Bits not affected by TLB translation