

15-213
"The course that gives CMU its Zip!"

Virtual Memory

March 18, 2004

Topics

- Motivations for VM
- Address translation
- Accelerating translation with TLBs

class18.ppt

Classic Motivations for Virtual Memory

Use Physical DRAM as a Cache for the Disk

- Address space of a process can exceed physical memory size
- Sum of address spaces of multiple processes can exceed physical memory

Simplify Memory Management

- Multiple processes resident in main memory.
Each process has its own address space
- Only "active" code and data is actually in memory
Allocate more memory to process as needed.

Provide Protection

- One process can't interfere with another.
Because they operate in different address spaces.
- User process cannot access privileged information
Different sections of address spaces have different permissions.

- 2 -

15-213, S04

Modern Motivations for VM

- **Memory sharing and control**
 - Copy on write: share physical memory among multiple processes until a process tries to write to it. At that point make a copy. For example, this eliminates the need for `vfork()`
 - Shared libraries
 - Protection (debugging) via Segment-Drivers (Solaris)
- **Sparse address space support (64bit systems)**
- **Memory as a fast communication device**
 - Part of memory is shared by multiple processes
- **Multiprocessing (beyond the scope of 15-213)**

- 3 -

15-213, S04

Why does VM Work?

It is not used!

- 4 -

15-213, S04

Motivation #1: DRAM a “Cache” for Disk

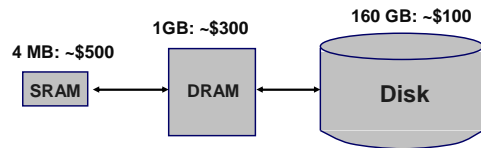
Full address space is quite large:

- 32-bit addresses: ~4,000,000,000 (4 billion) bytes
- 64-bit addresses: ~16,000,000,000,000,000 (16 quintillion) bytes

Disk storage is ~500X cheaper than DRAM storage

- 80 GB of DRAM: ~ \$25,000
- 80 GB of disk: ~ \$50

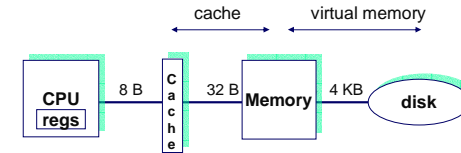
To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk



- 5 -

15-213, S04

Levels in Memory Hierarchy



	Register	Cache	Memory	Disk Memory
Size:	32 B	32 KB-4MB	1024 MB	100 GB
Latency:	< 1 ns	~2 ns	> 50 ns	>8 ms
\$/Mbyte:		\$125/MB	\$0.20/MB	\$0.001/MB
Line size:	8(16) B	32(64) B	4(64+) KB	

larger, slower, cheaper →

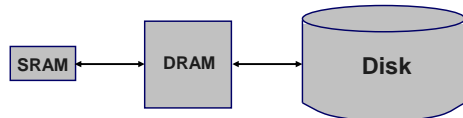
- 6 -

15-213, S04

DRAM vs. SRAM as a “Cache”

DRAM vs. disk is more extreme than SRAM vs. DRAM

- Access latencies:
 - DRAM ~10X slower than SRAM
 - Disk ~160,000X slower than DRAM !
- Importance of exploiting spatial locality:
 - First byte is ~160,000X slower than successive bytes on disk vs. ~4X improvement for page-mode vs. regular accesses to DRAM
- Bottom line:
 - Design decisions made for DRAM caches driven by enormous cost of misses



- 7 -

15-213, S04

Impact of Properties on Design

If DRAM was to be organized similar to an SRAM cache, how would we set the following design parameters?

- Line size?
 - Large, since disk better at transferring large blocks
- Associativity?
 - High, to minimize miss rate
- Write through or write back?
 - Write back, since can't afford to perform small writes to disk

What would the impact of these choices be on:

- miss rate
 - Extremely low. << 1%
- hit time
 - Must match cache/DRAM performance
- miss latency
 - Very high. ~10ms
- tag storage overhead
 - Low, relative to block size

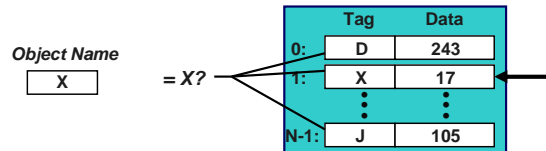
- 8 -

15-213, S04

Locating an Object in a "Cache"

SRAM Cache

- Tag stored with cache line
- Maps from cache block to memory blocks
 - From cached to uncached form
 - Save a few bits by only storing tag
- No tag for block not in cache
- Hardware retrieves information
 - can quickly match against multiple tags "Cache"



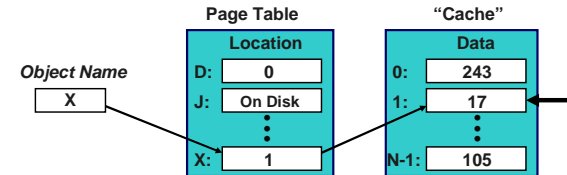
- 9 -

15-213, S04

Locating an Object in "Cache" (cont.)

DRAM Cache

- Each allocated page of virtual memory has entry in *page table*
- Mapping from virtual pages to physical pages
 - From uncached form to cached form
- Page table entry even if page not in memory
 - Specifies disk address
 - Only way to indicate where to find page
- OS retrieves information



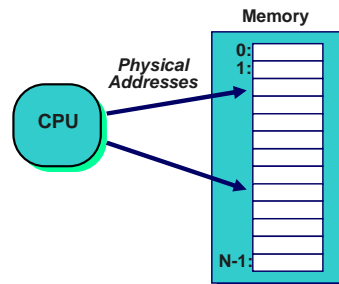
- 10 -

15-213, S04

A System with Physical Memory Only

Examples:

Most Cray machines, early PCs, nearly all embedded systems, etc.



- Addresses generated by the CPU correspond directly to bytes in physical memory

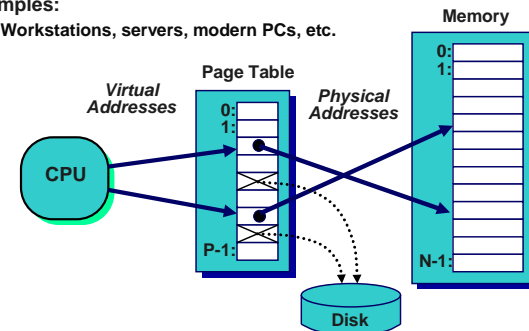
- 11 -

15-213, S04

A System with Virtual Memory

Examples:

Workstations, servers, modern PCs, etc.



- Address Translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)

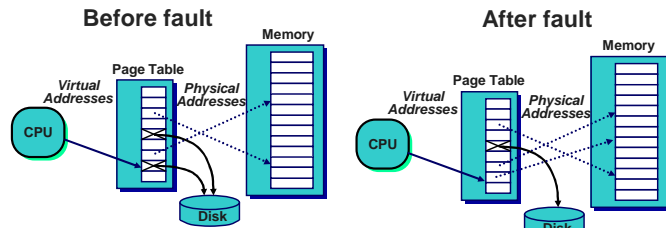
- 12 -

15-213, S04

Page Faults (like “Cache Misses”)

What if an object is on disk rather than in memory?

- Page table entry indicates virtual address not in memory
- OS exception handler invoked to move data from disk into memory
 - current process suspends, others can resume
 - OS has full control over placement, etc.



- 13 -

15-213, S04

Servicing a Page Fault

Processor Signals Controller

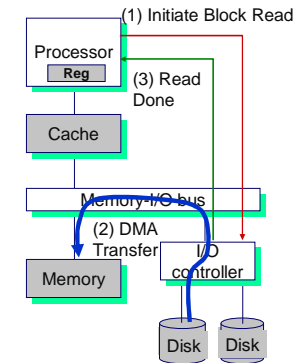
- Read block of length P starting at disk address X and store starting at memory address Y

Read Occurs

- Direct Memory Access (DMA)
- Under control of I/O controller

I/O Controller Signals Completion

- Interrupt processor
- OS resumes suspended process



- 14 -

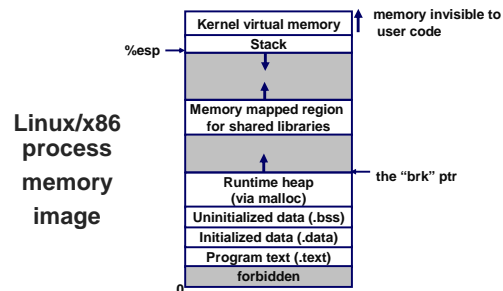
15-213, S04

Motivation #2: Memory Management

Multiple processes can reside in physical memory.

How do we resolve address conflicts?

- what if two processes access something at the same address?



- 15 -

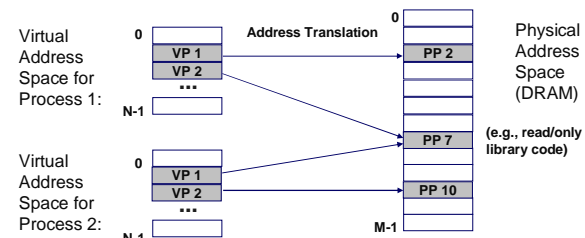
15-213, S04

Solution: Separate Virt. Addr. Spaces

- Virtual and physical address spaces divided into equal-sized blocks

Blocks are called “pages” (both virtual and physical)

- Each process has its own virtual address space
- Operating system controls how virtual pages as assigned to physical memory



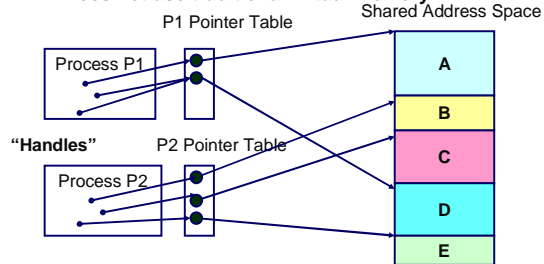
- 16 -

15-213, S04

Contrast: Macintosh Memory Model

MAC OS 1-9

- Does not use traditional virtual memory



All program objects accessed through “handles”

- Indirect reference through pointer table
- Objects stored in shared global address space

- 17 -

15-213, S04

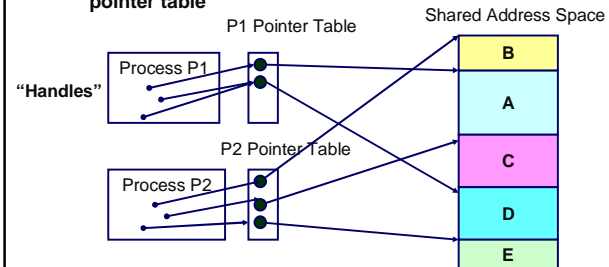
Macintosh Memory Management

Allocation / Deallocation

- Similar to free-list management of malloc/free

Compaction

- Can move any object and just update the (unique) pointer in pointer table



- 18 -

15-213, S04

Mac vs. VM-Based Memory Mgmt

Allocating, deallocating, and moving memory:

- can be accomplished by both techniques

Block sizes:

- Mac: variable-sized
 - may be very small or very large
- VM: fixed-size
 - size is equal to *one page* (4KB on x86 Linux systems)

Allocating contiguous chunks of memory:

- Mac: contiguous allocation is *required*
- VM: can map contiguous range of virtual addresses to disjoint ranges of physical addresses

Protection

- Mac: “wild write” by one process can corrupt another’s data

- 19 -

15-213, S04

MAC OS X

“Modern” Operating System

- Virtual memory with protection
- Preemptive multitasking*
 - Other versions of MAC OS require processes to voluntarily relinquish control

Based on MACH OS

- Developed at CMU in late 1980’s

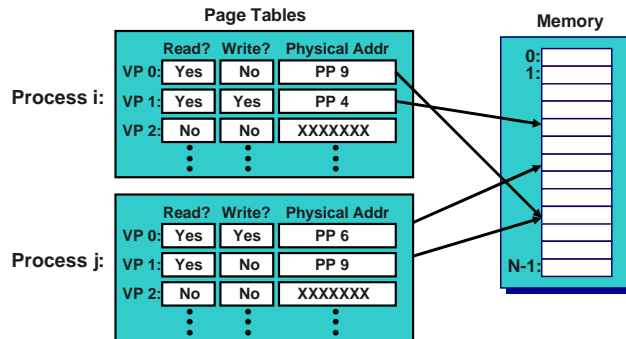
- 20 -

15-213, S04

Motivation #3: Protection

Page table entry contains access rights information

- hardware enforces this protection (trap into OS if violation occurs)



- 21 -

15-213, S04

VM Address Translation

Virtual Address Space

- $V = \{0, 1, \dots, N-1\}$

Physical Address Space

- $P = \{0, 1, \dots, M-1\}$
- $M < N$ (usually, but ≥ 4 Gbyte on an IA32 possible)

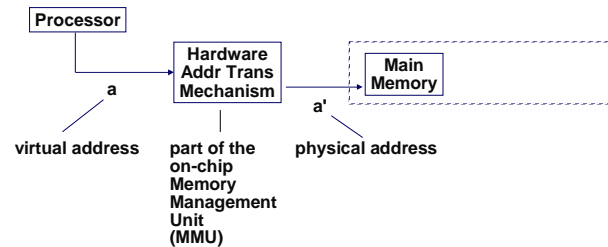
Address Translation

- MAP: $V \rightarrow P \cup \{\emptyset\}$
- For virtual address a :
 - MAP(a) = a' if data at virtual address a at physical address a' in P
 - MAP(a) = \emptyset if data at virtual address a not in physical memory
 - » Either invalid or stored on disk

- 22 -

15-213, S04

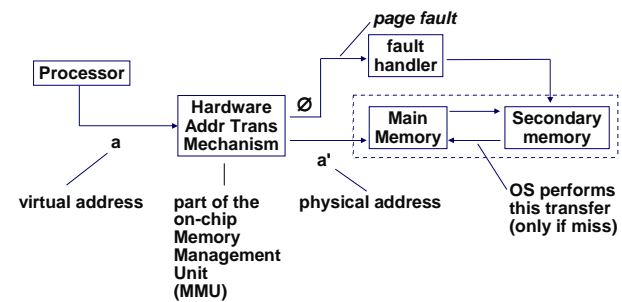
VM Address Translation: Hit



- 23 -

15-213, S04

VM Address Translation: Miss



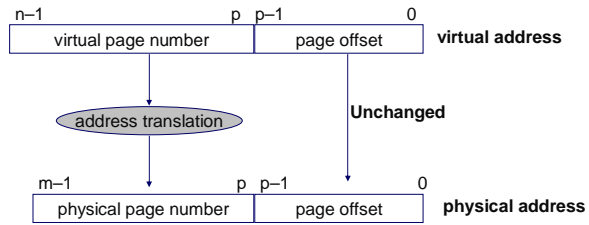
- 24 -

15-213, S04

VM Address Translation

Parameters

- $P = 2^p =$ page size (bytes).
- $N = 2^n =$ Virtual address limit
- $M = 2^m =$ Physical address limit

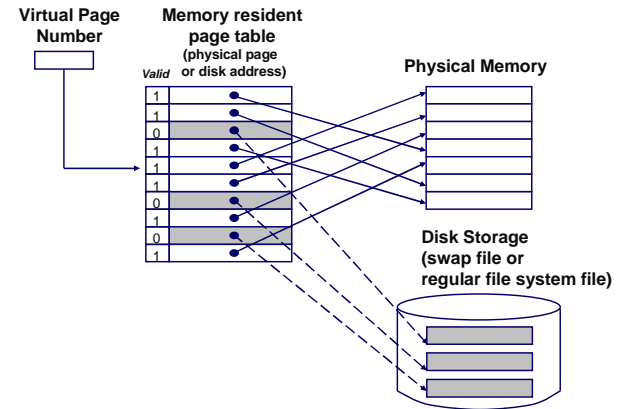


Page offset bits don't change as a result of translation

- 25 -

15-213, S04

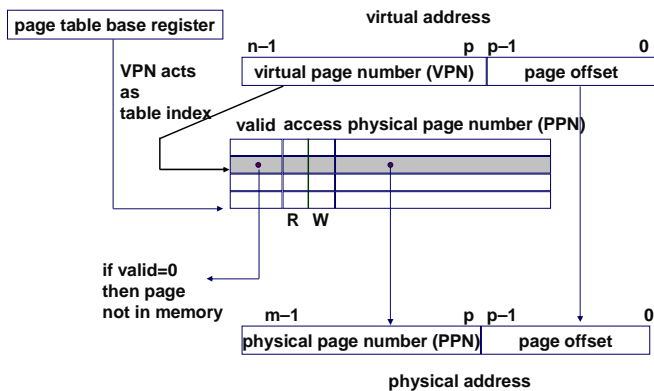
Page Tables



- 26 -

15-213, S04

Address Translation via Page Table



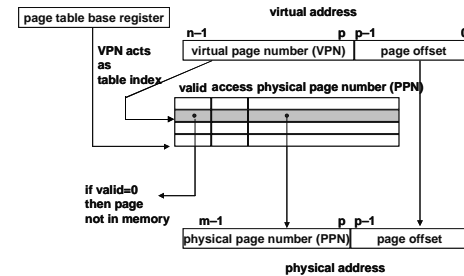
- 27 -

15-213, S04

Page Table Operation

Translation

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)



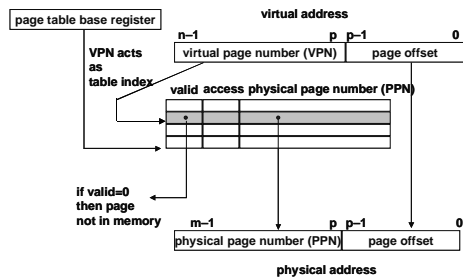
- 28 -

15-213, S04

Page Table Operation

Computing Physical Address

- Page Table Entry (PTE) provides information about page
 - if (valid bit = 1) then the page is in memory.
 - Use physical page number (PPN) to construct address
 - if (valid bit = 0) then the page is on disk
- Page fault



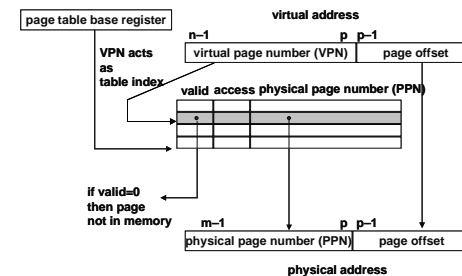
- 29 -

15-213, S'04

Page Table Operation

Checking Protection

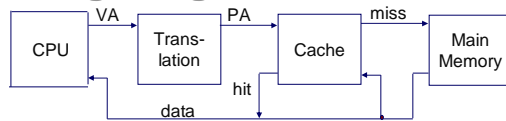
- Access rights field indicate allowable access
 - e.g., read-only, read-write, execute-only
 - typically support multiple protection modes (e.g., kernel vs. user)
- Protection violation fault if user doesn't have necessary permission



- 30 -

15-213, S'04

Integrating VM and Cache



Most Caches were "Physically Addressed"

- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn't need to be concerned with protection issues
 - Access rights checked as part of address translation

Perform Address Translation Before Cache Lookup

- But this could involve a memory access itself (of the PTE)
- Of course, page table entries can also become cached

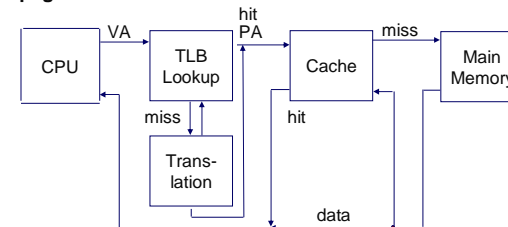
- 31 -

15-213, S'04

Speeding up Translation with a TLB

"Translation Lookaside Buffer" (TLB)

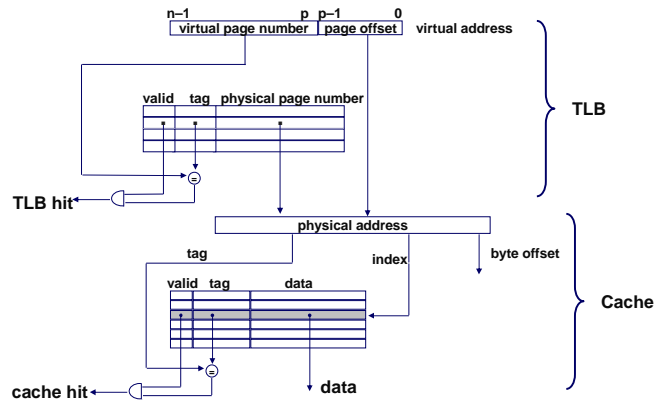
- Small hardware cache in MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages



- 32 -

15-213, S'04

Address Translation with a TLB



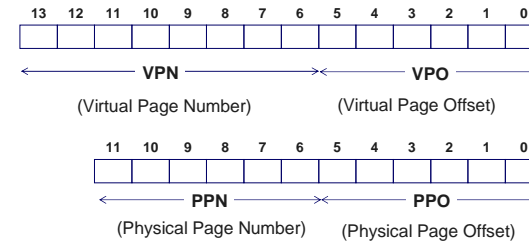
- 33 -

15-213, S'04

Simple Memory System Example

Addressing

- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes



- 34 -

15-213, S'04

Simple Memory System Page Table

- Only show first 16 entries (out of 256)

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

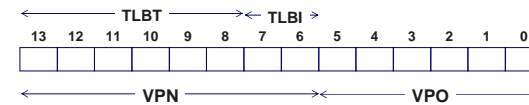
- 35 -

15-213, S'04

Simple Memory System TLB

TLB

- 16 entries
- 4-way associative



Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	-	0	09	0D	1	00	-	0	07	02	1
1	03	2D	1	02	-	0	04	-	0	0A	-	0
2	02	-	0	08	-	0	06	-	0	03	-	0
3	07	-	0	03	0D	1	0A	34	1	02	-	0

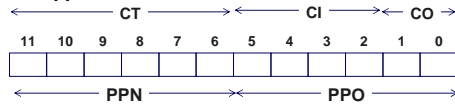
- 36 -

15-213, S'04

Simple Memory System Cache

Cache

- 16 lines
- 4-byte line size
- Direct mapped



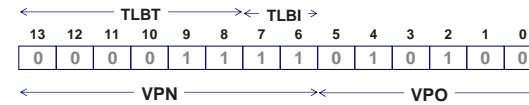
Idx	Tag	Valid	B0	B1	B2	B3	Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11	8	24	1	3A	00	51	89
1	15	0	-	-	-	-	9	2D	0	-	-	-	-
2	1B	1	00	02	04	08	A	2D	1	93	15	DA	3B
3	36	0	-	-	-	-	B	0B	0	-	-	-	-
4	32	1	43	6D	8F	09	C	12	0	-	-	-	
5	0D	1	36	72	F0	1D	D	16	1	04	96	34	15
6	31	0	-	-	-	-	E	13	1	83	77	1B	D3
7	16	1	11	C2	DF	03	F	14	0	-	-	-	-

- 37 -

15-213, S04

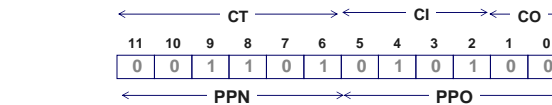
Address Translation Example #1

Virtual Address 0x03D4



VPN 0x0F TLBI 3 TLBT 0x03 TLB Hit? Y Page Fault? NO PPN: 0x0D

Physical Address



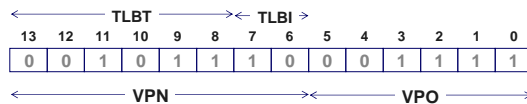
Offset 0 CI 0x5 CT 0x0D Hit? Y Byte: 0x36

- 38 -

15-213, S04

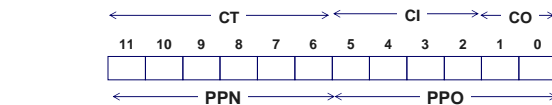
Address Translation Example #2

Virtual Address 0x0B8F



VPN 0x2E TLBI 2 TLBT 0x0B TLB Hit? NO Page Fault? YES PPN: TBD

Physical Address



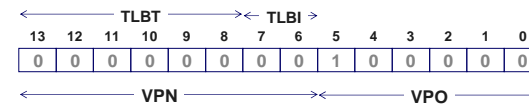
Offset CI CT Hit? Byte:

- 39 -

15-213, S04

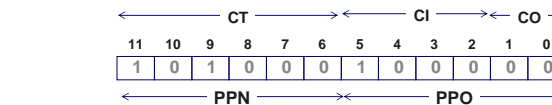
Address Translation Example #3

Virtual Address 0x0020



VPN 0x00 TLBI 0 TLBT 0x00 TLB Hit? NO Page Fault? NO PPN: 0x28

Physical Address



Offset 0 CI 0x8 CT 0x28 Hit? NO Byte: MEM

- 40 -

15-213, S04

Multi-Level Page Tables

Given:

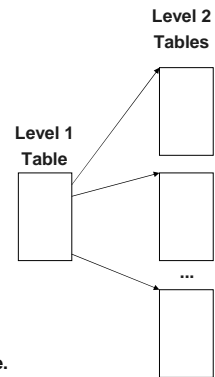
- 4KB (2^{12}) page size
- 32-bit address space
- 4-byte PTE

Problem:

- Would need a 4 MB page table!
 - $2^{20} * 4$ bytes

Common solution

- multi-level page tables
- e.g., 2-level table (P6)
 - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
 - Level 2 table: 1024 entries, each of which points to a page



- 41 -

15-213, S'04

Main Themes

Programmer's View

- Large "flat" address space
 - Can allocate large blocks of contiguous addresses
- Processor "owns" machine
 - Has private address space
 - Unaffected by behavior of other processes

System View

- User virtual address space created by mapping to set of pages
 - Need not be contiguous
 - Allocated dynamically
 - Enforce protection during address translation
- OS manages many processes simultaneously
 - Continually switching among processes
 - Especially when one must wait for resource
 - » E.g., disk I/O to handle page fault

- 42 -

15-213, S'04