EECS 252
Graduate Computer Architecture
Lecture 3

$\aleph_0$ (continued)
Review of Caches and Virtual Memory

January 27th, 2010

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Review: Control and Pipelining

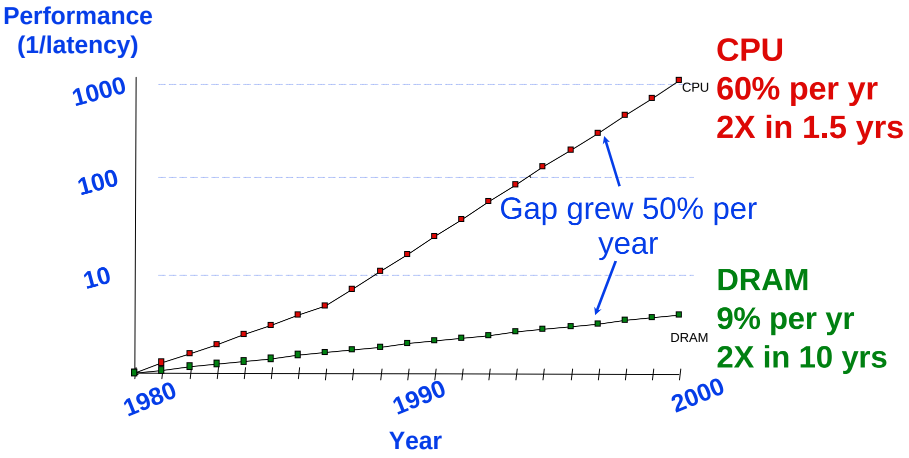
- Control VIA State Machines and Microprogramming
- Just overlap tasks; easy if tasks are independent
- Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data (RAW,WAR,WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction
- Exceptions, Interrupts add complexity

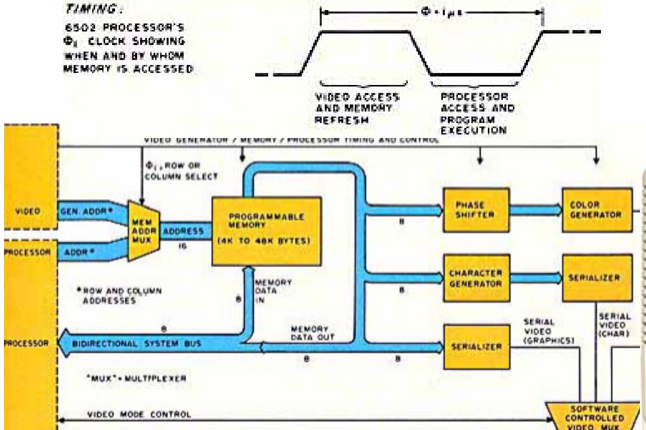
Memory Hierarchy Review

Since 1980, CPU has outpaced DRAM ...



- How do architects address this gap?
  - Put small, fast "cache" memories between CPU and DRAM.
  - Create a "memory hierarchy"

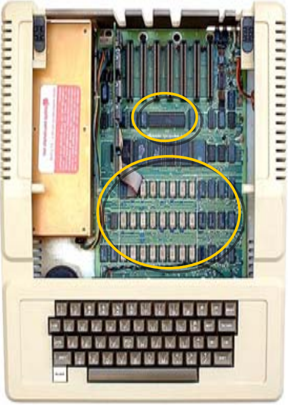
1977: DRAM faster than microprocessors



Apple][(1977)

CPU: 1000 ns

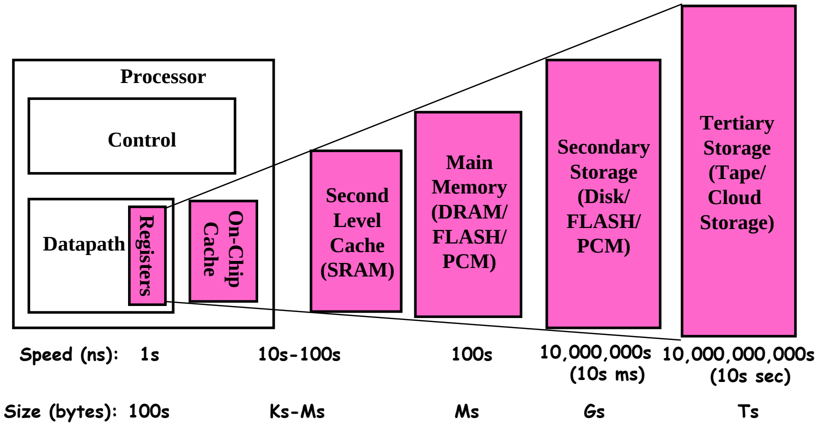
DRAM: 400 ns



RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

Memory Hierarchy

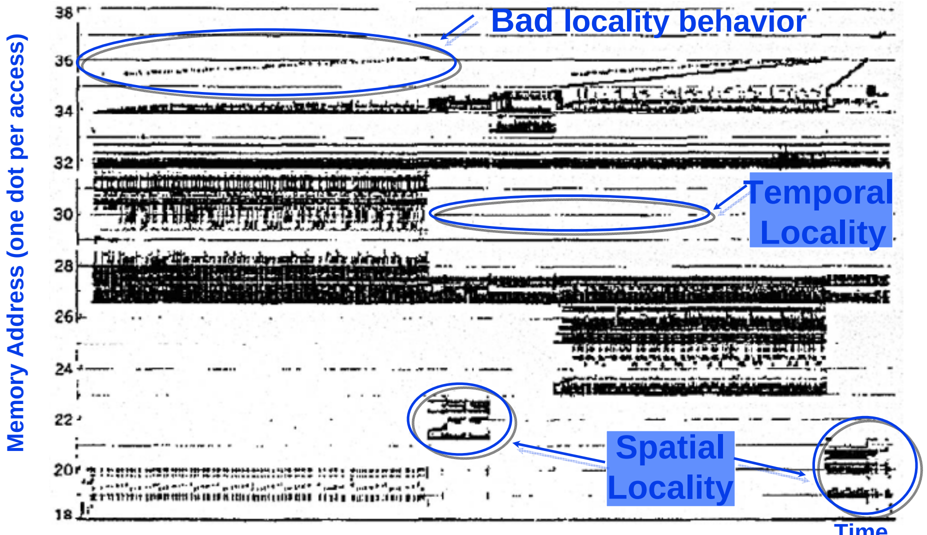
- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology


Processor
Control
Datapath
Registers
On-Chip Cache
Second Level Cache (SRAM)
Main Memory (DRAM/ FLASH/ PCM)
Secondary Storage (Disk/ FLASH/ PCM)
Tertiary Storage (Tape/ Cloud Storage)
Speed (ns): 1s
10s-100s
100s
10,000,000s (10s ms)
10,000,000,000s (10s sec)
Size (bytes): 100s
Ks-Ms
Ms
Gs
Ts

The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW relied on locality for speed

Programs with locality cache well ...



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Memory Hierarchy: Apple iMac G5

Managed by compiler

Managed by hardware

Managed by OS, hardware, application

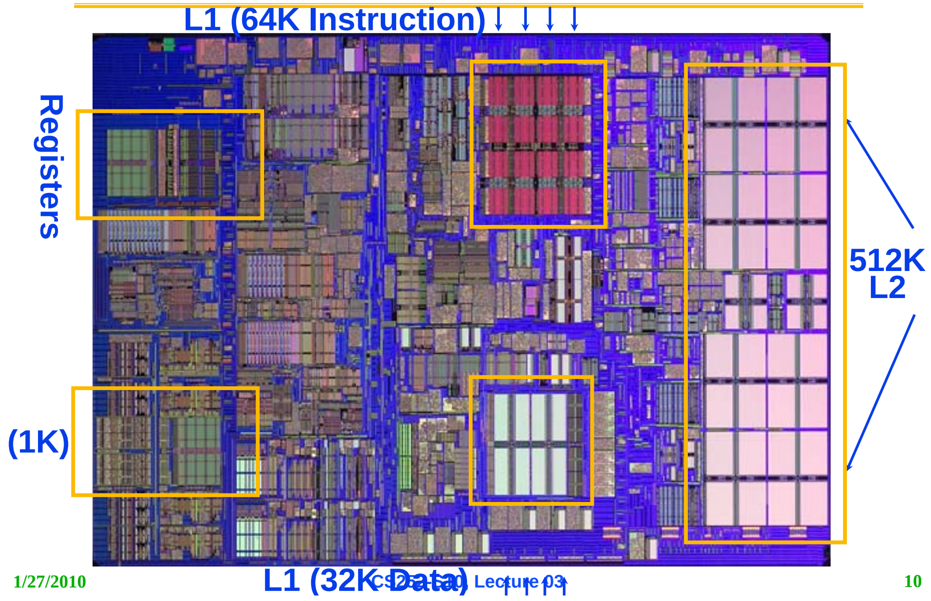
07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

iMac G5
1.6 GHz

Goal: Illusion of large, fast, cheap memory

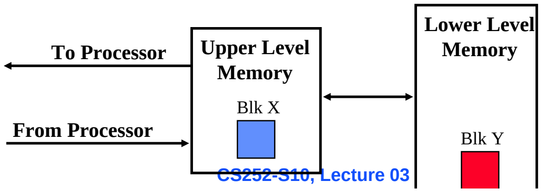
Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

iMac's PowerPC 970: All caches on-chip



Memory Hierarchy: Terminology

- **Hit: data appears in some block in the upper level (example: Block X)**
  - **Hit Rate: the fraction of memory access found in the upper level**
  - **Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss**
- **Miss: data needs to be retrieve from a block in the lower level (Block Y)**
  - **Miss Rate = 1 - (Hit Rate)**
  - **Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor**
- **Hit Time << Miss Penalty (500 instructions on 21264!)**

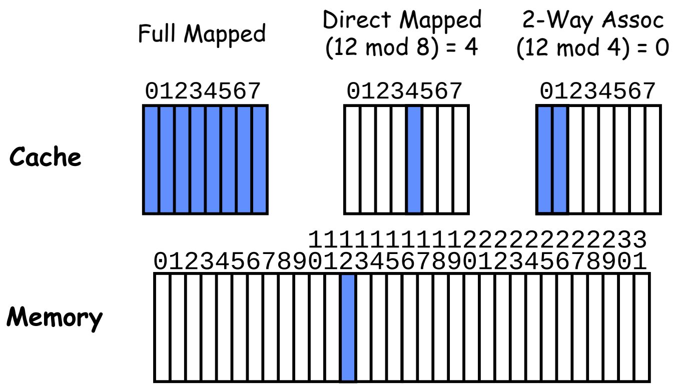


4 Questions for Memory Hierarchy

- **Q1: Where can a block be placed in the upper level?** ***(Block placement)***
- **Q2: How is a block found if it is in the upper level?** ***(Block identification)***
- **Q3: Which block should be replaced on a miss?** ***(Block replacement)***
- **Q4: What happens on a write?** ***(Write strategy)***

Q1: Where can a block be placed in the upper level?

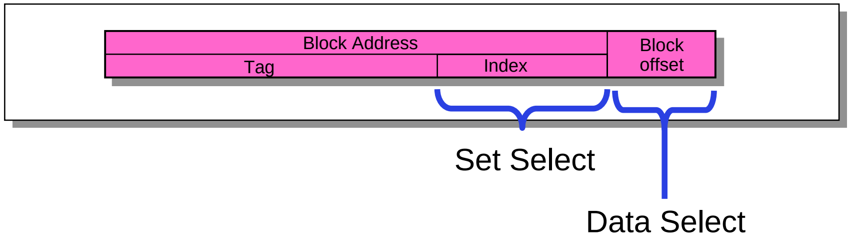
- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



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

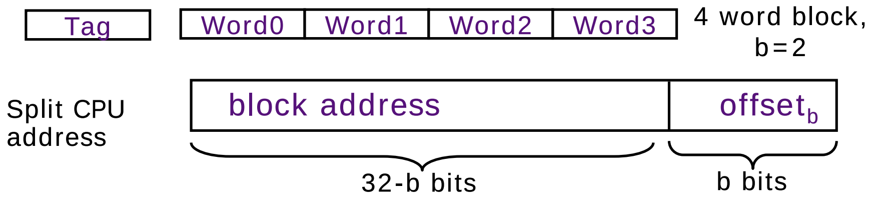
Q2: How is a block found if it is in the upper level?



- 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

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



2^b = block size *a.k.a* line size (in bytes)

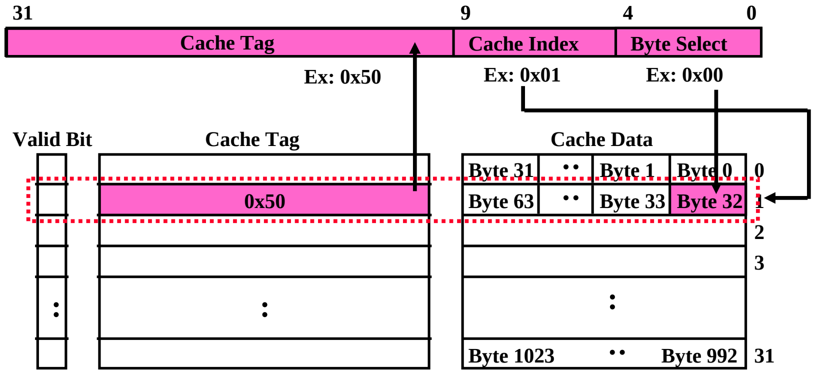
Larger block size has distinct hardware advantages
- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

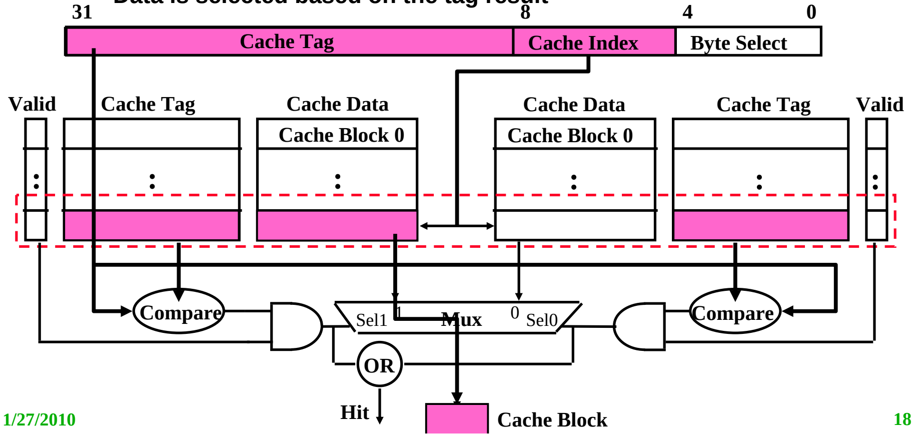
Review: Direct Mapped Cache

- 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



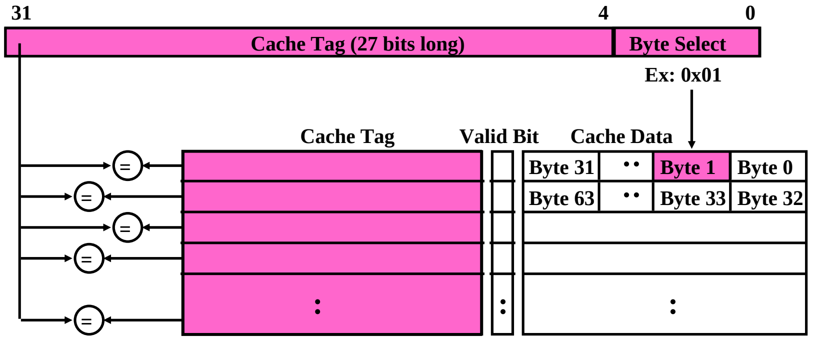
Review: Set Associative Cache

- 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



Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - LRU (Least Recently Used): Appealing, but hard to implement for high associativity
  - Random: Easy, but – how well does it work?

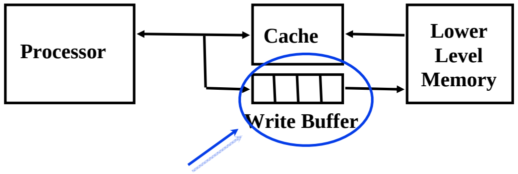
Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16K	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64K	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256K	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ? A. So CPU doesn't stall

Q. Why a buffer, why not just one register ? A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer? A. Yes! Drain buffer before next read, or check write buffers for match on reads

5 Basic Cache Optimizations

- Reducing Miss Rate

1. Larger Block size (compulsory misses)
2. Larger Cache size (capacity misses)
3. Higher Associativity (conflict misses)

- Reducing Miss Penalty

4. Multilevel Caches

- Reducing hit time

5. Giving Reads Priority over Writes
   - E.g., Read complete before earlier writes in write buffer

Administrivia

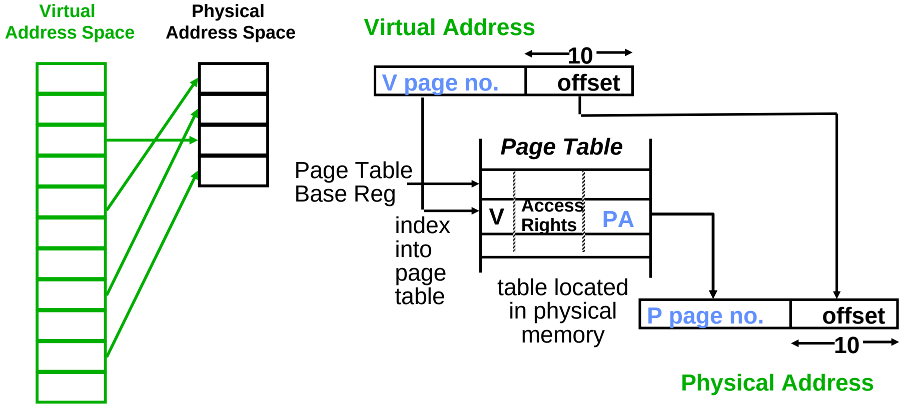
- Paper readings: important for your graduate career
- Remember: everything on web site:
  - HTTP://www.cs.berkeley.edu/~kubitron/cs252
- WebSite signup
  - Make sure to signup for the class if you haven't yet
- Don't forget the ISCA retrospective

RISC: The integrated systems view (Discussion of Papers)

- “The Case for the Reduced Instruction Set Computer”
  - Dave Patterson and David Ditzel
- “Comments on ‘The Case for the Reduced Instruction Set Computer’”
  - Doug Clark and William Strecker
- “"Retrospective on High-Level Computer Architecture"
  - David Ditzel and David Patterson

In-class discussion of these papers

What is virtual memory?



- Virtual memory => treat memory as a cache for the disk
- Terminology: blocks in this cache are called “Pages”
  - Typical size of a page: 1K — 8K
- Page table maps virtual page numbers to physical frames
  - “PTE” = Page Table Entry

What is in a Page Table Entry (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”

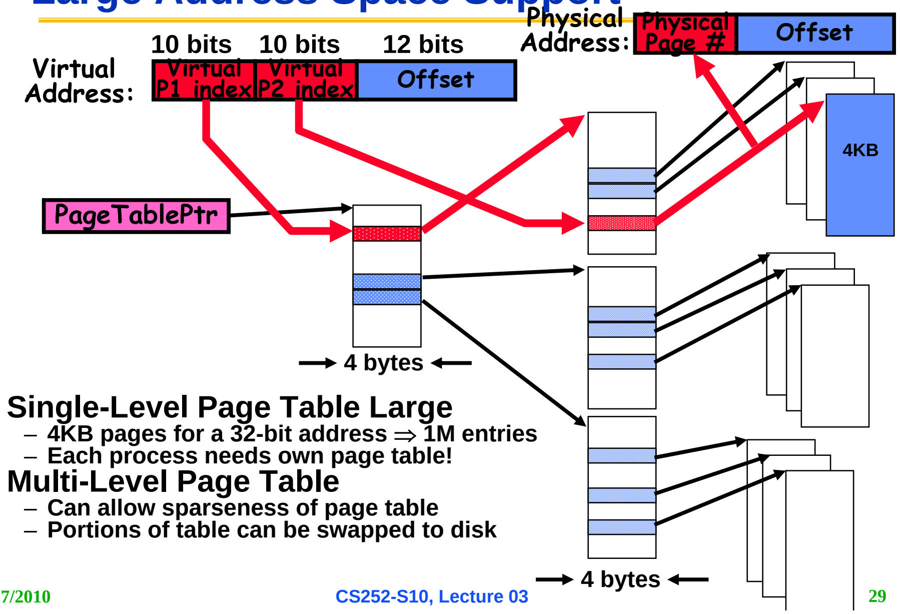
Page Frame Number (Physical Page Number)	Free (OS)	0	L	D	A	PCD	PWT	U	W	P
31-12	11-9	8	7	6	5	4	3	2	1	0

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

Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program (“Working Set”) must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - » (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- Sharing:
  - Can map same physical page to multiple users (“Shared memory”)

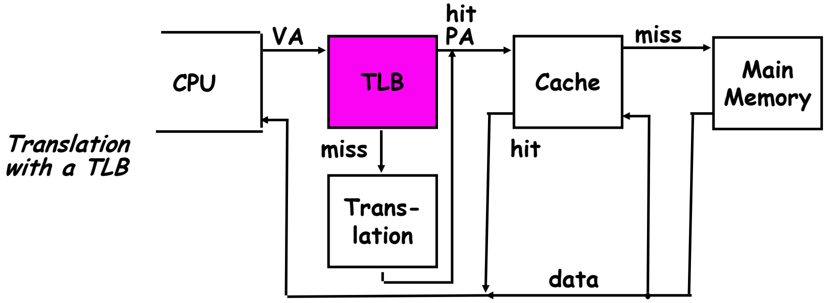
Large Address Space Support



- **Single-Level Page Table Large**
  - 4KB pages for a 32-bit address ⇒ 1M entries
  - Each process needs own page table!
- **Multi-Level Page Table**
  - Can allow sparseness of page table
  - Portions of table can be swapped to disk

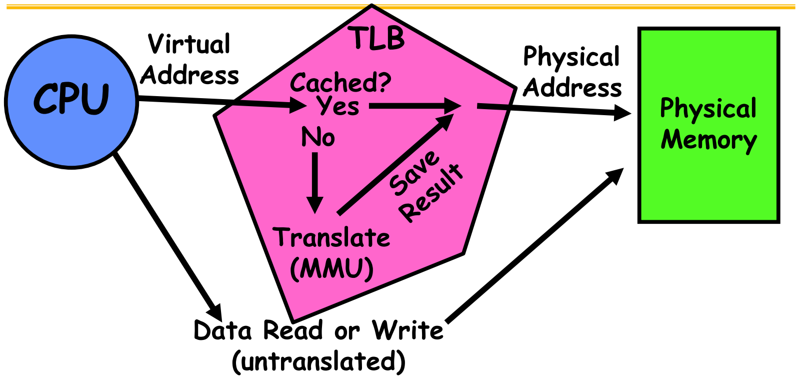
Translation Look-Aside Buffers

- **Translation Look-Aside Buffers (TLB)**
  - Cache on translations
  - Fully Associative, Set Associative, or Direct Mapped



- **TLBs are:**
  - Small – typically not more than 128 – 256 entries
  - Fully Associative

Caching Applied to Address Translation

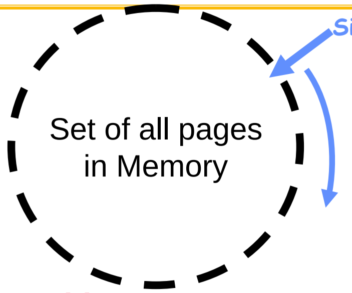


- **Question is one of page locality: does it exist?**
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some…
- **Can we have a TLB hierarchy?**
  - Sure: 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

Clock Algorithm: Not Recently Used



Single Clock Hand:
Advances only on page fault!
Check for pages not used recently
Mark pages as not used recently

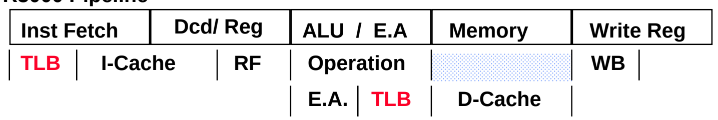
Page Table

dirty	used	
1	0	...
1	0	
0	1	
1	1	
0	0	

- **Clock Algorithm:**
  - Approximate LRU (approx to approx to MIN)
  - Replace an old page, not the oldest page
- **Details:**
  - Hardware "use" bit per physical page:
    - » Hardware sets use bit on each reference
    - » If use bit isn't set, means not referenced in a long time
  - On page fault:
    - » Advance clock hand (not real time)
    - » Check use bit: 1→used recently; clear and leave alone
      0→selected candidate for replacement

Example: R3000 pipeline

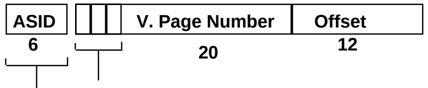
MIPS R3000 Pipeline



TLB

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

Virtual Address Space

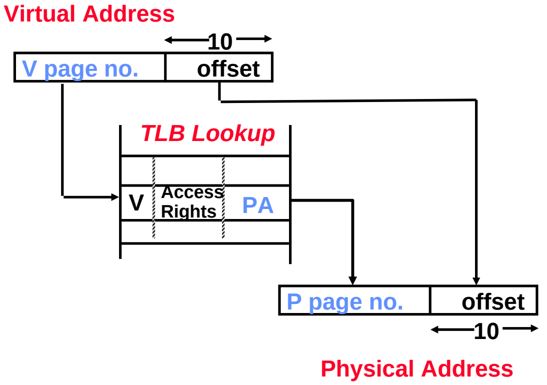


0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among
64 user processes without TLB flush

Reducing translation time further

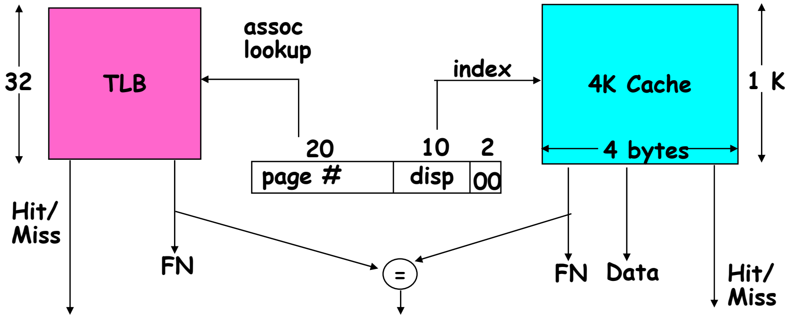
- As described, TLB lookup is in serial with cache lookup:



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

Overlapping TLB & Cache Access

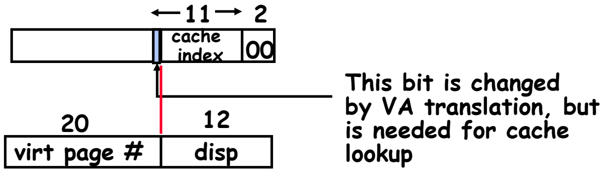
- Here is how this might work with a 4K cache:



- **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

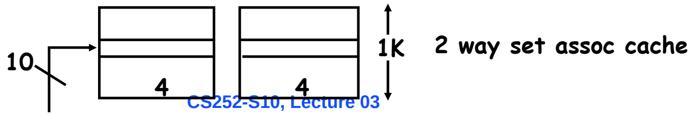
Problems With Overlapped TLB Access

- Overlapped access requires address bits used to index into cache *do not change* as result translation
  - This usually limits things to small caches, large page sizes, or high
  - n-way set associative caches if you want a large cache
- Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



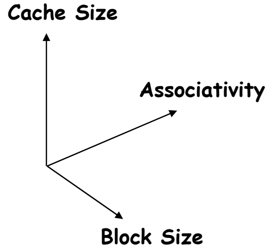
This bit is changed by VA translation, but is needed for cache lookup

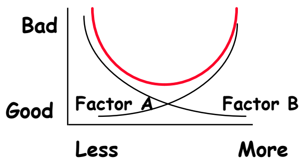
Solutions:
go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]



Summary #1/3: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins





Summary #2/3: Caches

- The Principle of Locality:
  - Program 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 Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Write Policy: Write Through vs. Write Back
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): affects Compilers, Data structures, and Algorithms

Summary #3/3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy benefits, but computers insecure
- Prepare for debate + quiz on Wednesday