

Caches continued and Virtual Memory

◆ Last Time

- Cache organization
- Dealing with Stores

◆ Today

- Quiz
- Cache entry replacement
- Virtual Memory

◆ Reminders/Announcements

- HW#5 due Tuesday, 3/14, beginning of class
- Final exam, Monday 3/20, 3-6pm, Peterson 110

Three types of cache misses

◆ Compulsory (or cold-start) misses

- first access to the data.

◆ Capacity misses

- we missed only because the cache isn't big enough.

◆ Conflict misses

- we missed because the data maps to the same line as other data that forced it out of the cache.

address string:
4 00000100
8 00001000
12 00001100
4 00000100
8 00001000
20 00010100
4 00000100
8 00001000
20 00010100
24 00011000
12 00001100
8 00001000
00000100

tag	data

DM cache

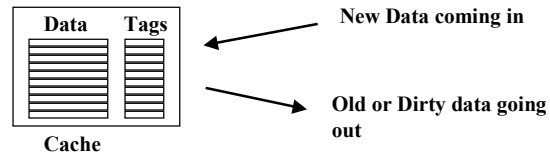
So, then, how do we decrease...

- ◆ Compulsory misses?
- ◆ Capacity misses?
- ◆ Conflict misses?

Replacement

- ◆ What happens to the data in the cache, when we bring data in?
- ◆ Direct Mapped caches
 - No choice which data to replace
 - Only one place to put the incoming data
 - Write the data back to the memory
- ◆ Associative Caches
 - Several possible places for the data
 - Choosing which to replace => Replacement policy
 - FIFO, Random, Least Recently Used: don't matter much for small degrees of associativity
- ◆ Writing the data back to the memory

Writing Cache Data Back



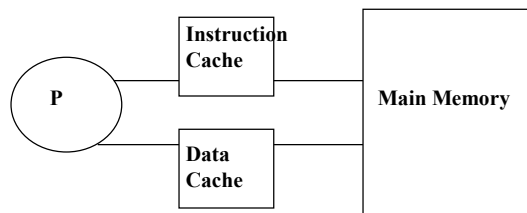
- ◆ Data in cache may have been modified (dirty)
- ◆ If so, must write it back to ensure updates are not lost
- ◆ Two possibilities:
 - Write data back always (safe)
 - Keep track of when data has been modified, write it back
 - Typical scheme but requires hardware support (detect writes)

Replacement algorithms

- ◆ LRU, FIFO, random, others?
- ◆ only need replacement for associative caches
- ◆ LRU
 - requires one bit for 2-way set-associative, 8 bits for 4-way, 24 bits for 8-way.
 - can be emulated with $\log n$ bits (NMRU)
 - can be emulated with *use* bits for highly associative caches (like page tables)
- ◆ FIFO
 - requires $\log n$ bits
- ◆ Random?

Instruction and Data Caches

- ◆ Caches necessary to support pipelined instruction execution at high speeds (600 - 1000 Mhz)
- ◆ MIPS instruction set has 1-2 memory references per instruction x Superscalar!
- ◆ => Need two caches, one for instructions and one for data to keep up



Caches in Current Processors

- ◆ Often DM at highest level (closest to CPU), associative further away
- ◆ split I and D close to the processor, unified further away (for throughput rather than miss rate).
- ◆ write-through and write-back both common, but never write-through all the way to memory.
- ◆ 32-byte cache lines very common

- ◆ Non-blocking
 - processor doesn't stall on a miss, but only on the use of a miss (if even then)
 - this means the cache must be able to handle multiple outstanding accesses.

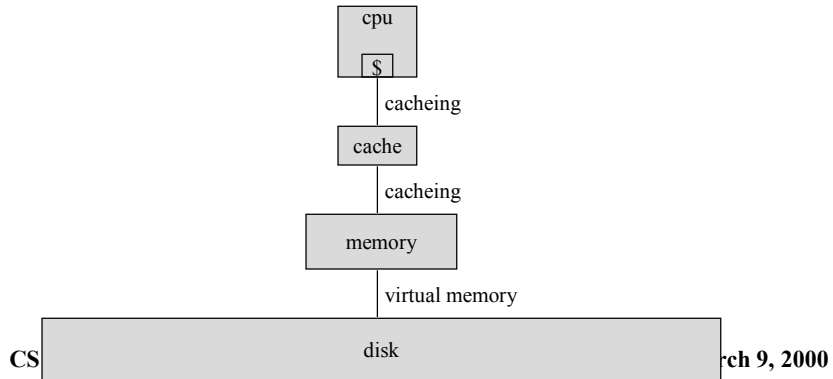
Multiple Caches and Consistency?

- ◆ **Instruction cache supports Instruction Fetch**
- ◆ **Data cache supports Load/Store operations**
- ◆ **Do they ever share data?**
 - Self-modifying code
 - Compiler produces code and links it dynamically
- ◆ **Hardware must support consistency.**
 - Flush Instruction Cache when necessary (delete contents)
- ◆ **Cache Consistency in Multiprocessors is an important topic.**

Virtual Memory

Virtual Memory

- ◆ It's just another level in the cache/memory hierarchy
- ◆ *Virtual memory* is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).



Virtual Memory

- ◆ is just cacheing, but uses different terminology

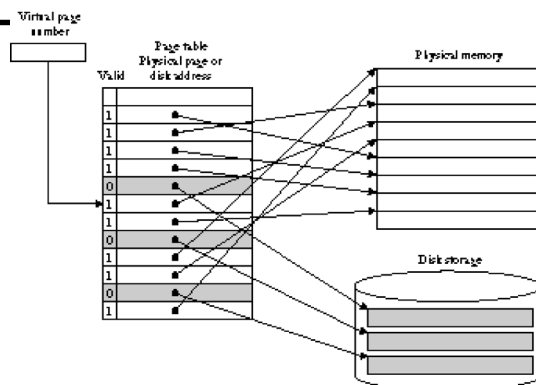
<u>cache</u>	<u>VM</u>
block	page
cache miss	page fault
address	virtual address
index	physical address (sort of)

Virtual Memory

- ◆ What happens if another program in the processor uses the same addresses that yours does?
- ◆ What happens if your program uses addresses that don't exist in the machine?
- ◆ What happens to “holes” in the address space your program uses?

- ◆ So, virtual memory provides
 - performance (through the caching effect)
 - protection
 - ease of programming/compilation
 - efficient use of memory

Virtual Memory

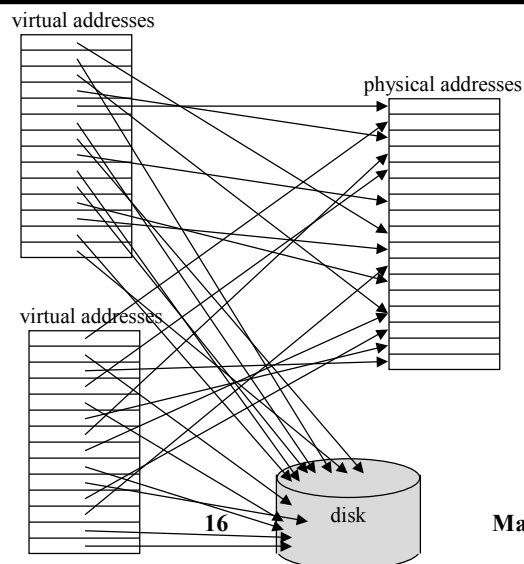


- ◆ is just a mapping function from virtual memory addresses to physical memory locations, which allows caching of virtual pages in physical memory.

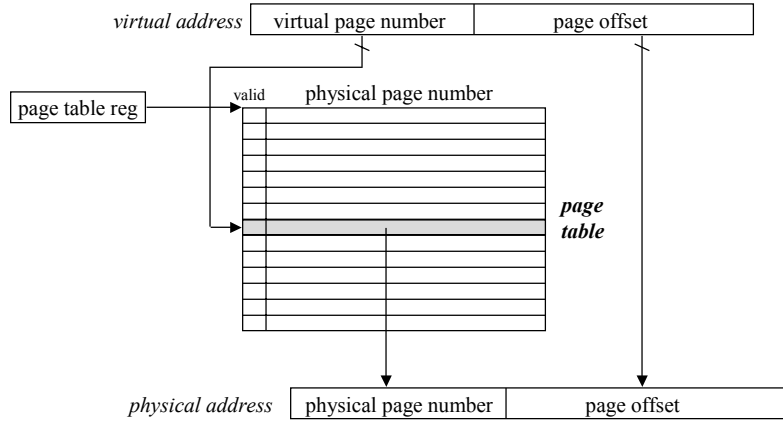
What makes VM different than memory caches

- ◆ *MUCH higher miss penalty (millions of cycles)!*
- ◆ **Therefore**
 - large pages [equivalent of cache line] (4 KB to MBs)
 - associative mapping of pages (typically fully associative)
 - software handling of misses (but not hits!!)
 - *write-through* not an option, only write-back
- ◆ **Generally a functionality enhancement (programs run), not a performance enhancement (NOT illusion of a large memory).**

Virtual Memory mapping



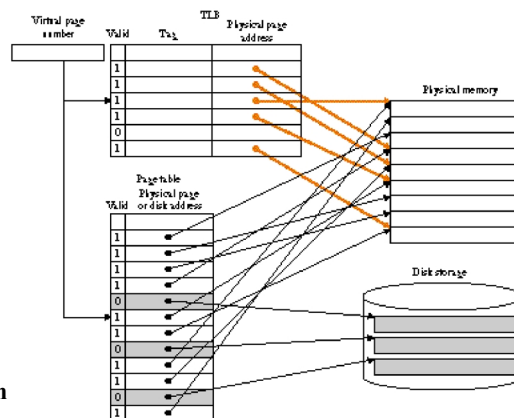
Address translation via Page Tables



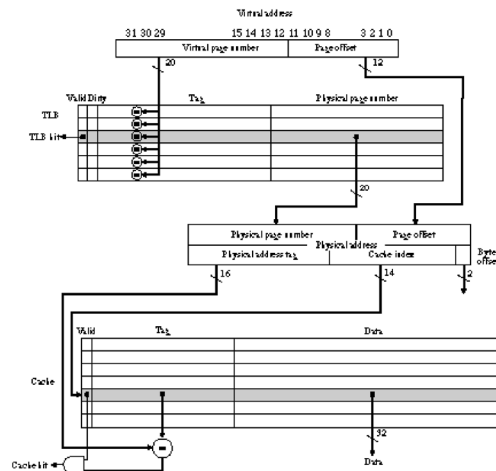
- ◆ all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)
- ◆ so why is this fully associative???

Making Address Translation Fast

- ◆ A cache for address translations: translation lookaside buffer



TLBs and caches



Virtual Memory Summary

- ◆ **Caches: Location, Organization (block size and associativity), Replacement**
- ◆ **Virtual memory provides**
 - protection, sharing, illusion of large main memory
- ◆ **Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.**
- ◆ **Three things can go wrong on a memory access: cache miss, TLB miss, page fault.**