CS61C
Improving Cache Memory and Virtual Memory Introduction

Lecture 18

April 2, 1999

Dave Patterson
(http.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs61c/schedule.html

Review 1/1

- Tag, index, offset to find matching data, support larger blocks, reduce misses
- Where in cache? Direct Mapped Cache
  - Conflict Misses if memory addresses compete
- Fully Associative to let memory data be in any block: no Conflict Misses
- Set Associative: Compromise, simpler hardware than Fully Associative, fewer misses than Direct Mapped
- LRU: Use history to predict replacement

Outline

- Improving Miss Penalty
- Improving Writes
- Administrivia, More Survey results
- Virtual Memory
- Virtual Memory and Cache
- Translation Lookaside Buffer
- Conclusion

Improving Caches

- In general, minimize Average Access Time:

  = Hit Time x (1 - Miss Rate)

  \+ Miss Penalty x Miss Rate
- So far, have look at improving Miss Rate:
  - Larger block size
  - Larger Cache
  - Higher Associativity
- What about Miss Penalty?

Improving Miss Penalty

- When caches started becoming popular, Miss Penalty was about 10 processor clock cycles
- Today 500 MHz Processor (2 nanoseconds per clock cycle) and 200 ns to go to DRAM
  ⇒ 100 processor clock cycles!
- Solution: Place another cache between memory and the processor cache: <u>Second Level (L2) Cache</u>

Analyzing a multi-level cache hierarchy

- We consider the L2 hit and miss times to include the cost of not finding the data in the L1 cache
- Similarly, the L2 cache hit rate is only for accesses which actually make it to the L2 cache

So how do we calculate it out?

- Access time = L1 hit time * L1 hit rate + <u>L1 miss penalty</u> * L1 miss rate
- Access time = L1 hit time * L1 hit rate + (<u>L2 hit time * L2 hit rate + L2 miss penalty * L2 miss rate</u>) * L1 miss rate
- We simply calculate the L1 miss penalty as being the access time for the L2 cache

Do the numbers for L2 Cache

- Assumptions:
  - L1 hit time = 1 cycle, L1 hit rate = 90%
  - L2 hit time (also L1 miss penalty) = 4 cycles, L2 miss penalty= 100 cycles, L2 hit rate = 90%
- Access time = L1 hit time * L1 hit rate + (L2 hit time * L2 hit rate + L2 miss penalty * (1 - L2 hit rate)) * L1 miss rate

= 1*0.9 + (4*0.9 + 100*0.1) *(1-0.9)

= 0.9 + (13.6) * 0.1 = 2.26 clock cycles

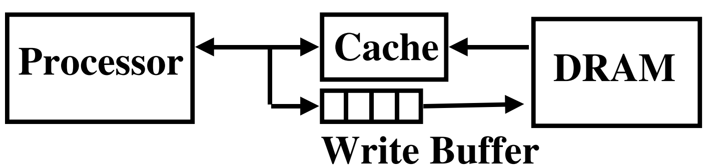
What would it be without the L2 cache?

- Assume that the L1 miss penalty would be 100 clock cycles
- 1 *0.9 + (100)* 0.1
- 10.9 clock cycles
- So gain a benefit from having the second, larger cache before main memory
- Today's L1 cache size 16 KB-64 KB, L2 cache may be 512 KB to 4096 KB

What about Writes to Memory?

- Simplest Policy: The information is written to both the block in the cache and to the block in the lower-level memory
- Problem: Writes operate at speed of lower level memory!

Improving Cache Performance: Write Buffer



- A <u>Write Buffer</u> is added between Cache and Memory
  - Processor: writes data into cache & write buffer
  - Controller: write buffer contents to memory
- Write buffer is just a First In First Out queue:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Improving Cache Performance: Write Back

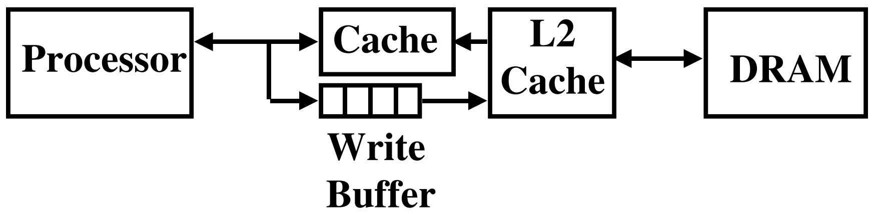
- Option 2: data is written <u>only</u> to cache block
- Modified cache block is written to main memory <u>only when it is replaced</u>
  - Block is unmodified (<u>clean</u>) or modified (<u>dirty</u>)
- This scheme called "<u>Write Back</u>"; original scheme called "<u>Write Through</u>"
- Advantage of Write Back
  - Repeated writes to same location stay in cache
- Disadvantage of Write Back
  - Any block replacement can ⇒ write to memory

What to do on a Write Miss?

- On a read miss, you must bring the block into the cache so that you can complete the read
- Option 1: Just like read; bring whole block into cache and then modify bytes needed; "Write Allocate"
  - Write indicates nearby access in future?
- Option 2: Only update lower level memory, nothing in cache; "No Write Allocate"
  - Perhaps just clearing memory, no reuse?
- Preference for Write Back vs. Write Thru?

Avoiding Write Buffer Saturation

- Store frequency > 1 / DRAM write cycle
  - Store buffer will overflow no matter how long you make queue
- Solution for write buffer saturation:
  - Use a write back 1st level cache, or
  - Install a 2nd level cache (write back)



Administrivia

- Project 5: Due 4/14: design and implement a cache (in software) and plug into instruction simulator
- 8th homework: Due 4/7 7PM
  - Exercises 7.7, 7.8, 7.20, 7.22, 7.32
- Readings 7.2, 7.3, 7.4

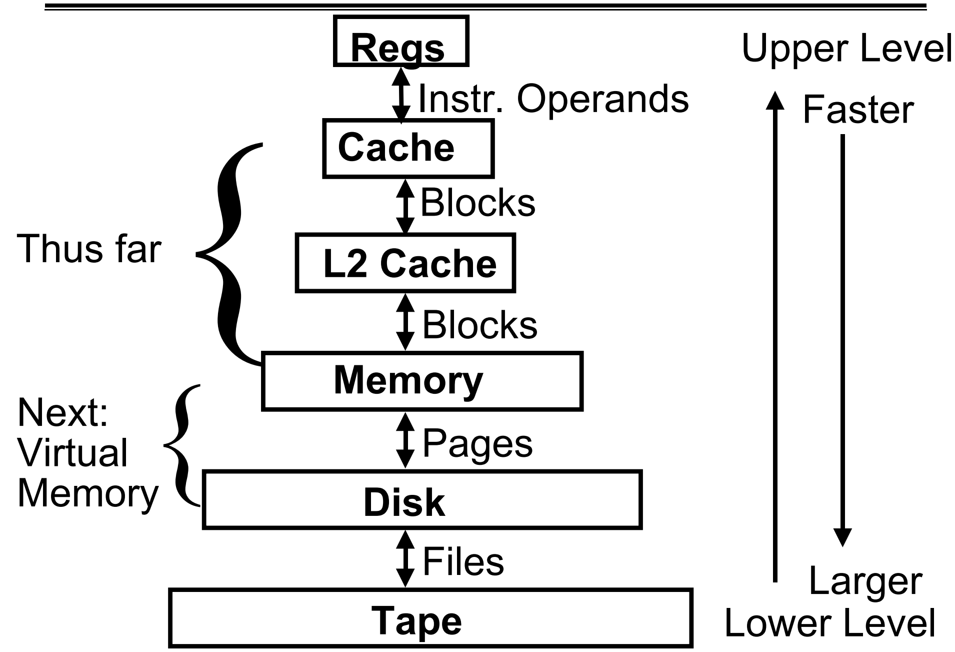
Suggestions

- Microphone volume?
  - Will try wired mike
- Separate Project, HW due dates?
  - Will look at separating project, HW if same week
- Why not more credit for projects?
  - Reduce reward for not doing own work
- Why not more tests? (Note:point totals = F98)
  - Testing vs. lecture time; time to make good test
- Why not more units?
  - Workload matches units by end of semester?

Questions

- **Look at X86 since its so prevalent?**
  - How many lectures dedicate to this? Can make much dent in 1-2 lectures?
- **Why recommended 7PM with 8AM deadline?**
  - Human nature? Goal + allowed slip
- **Why front load course?**
  - Many courses have more work in 2nd half; also, too many students trying 61C Spring '99
- **Why not finish in lab?**
  - What non trivial task can everyone finish in 50 minutes? Reduced homeworks, expanded lab

Another View of the Memory Hierarchy



Virtual Memory

- **If Principle of Locality allows caches to offer (usually) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?**
- **Called <u>Virtual Memory</u>**
  - Also allows OS to share memory, protect programs from each other
  - Today, more important for <u>protection</u> vs. just another level of memory hierarchy
  - Historically, it predates caches

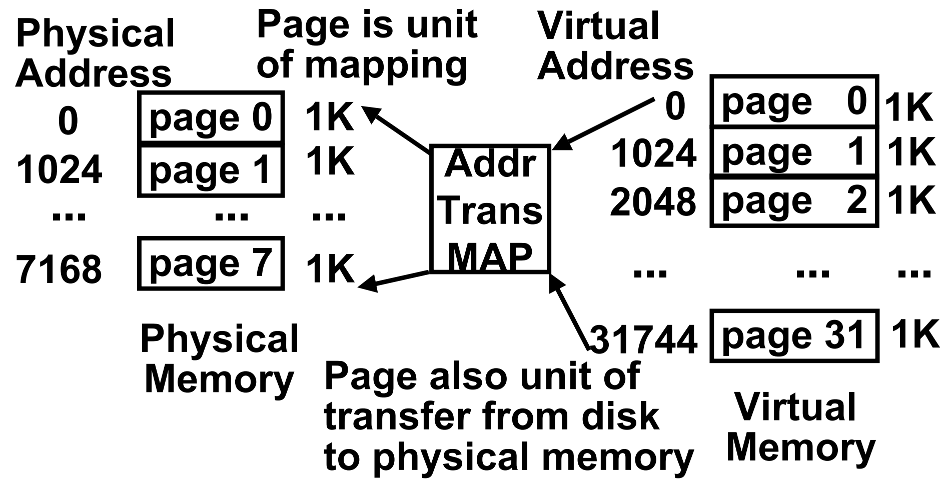
Comparing the 2 levels of hierarchy

Cache Version	Virtual Memory vers.
Block or Line	<u>Page</u>
Miss	<u>Page Fault</u>
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

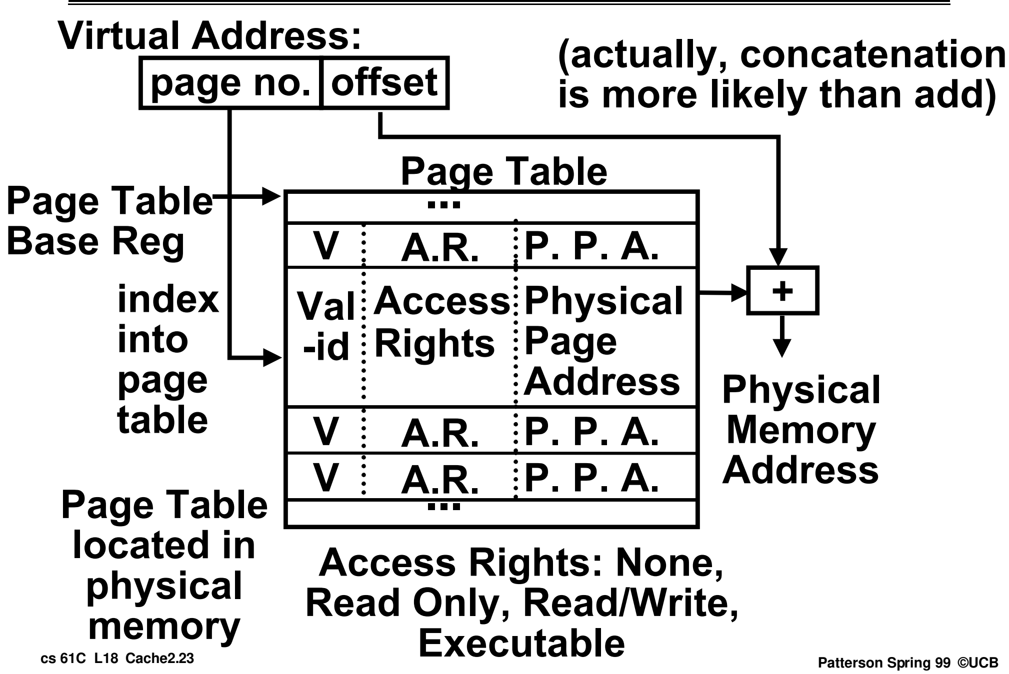
Protection leads to New Terms

- Each program is said to have "virtual address space" (e.g., 2^{32} Bytes)
- Each computer has "physical address space" (e.g., 128 MegaBytes DRAM); or real memory
- Address translation: map page from virtual address to physical address
  - Allows multiple programs to use (different pages of physical) memory at same time
  - Can only access what OS permits
  - Also allows some pages of virtual memory to be represented on disk, not in main memory (memory hierarchy)

Paging Organization (assume 1 KB pages)



Address Mapping: Page Table



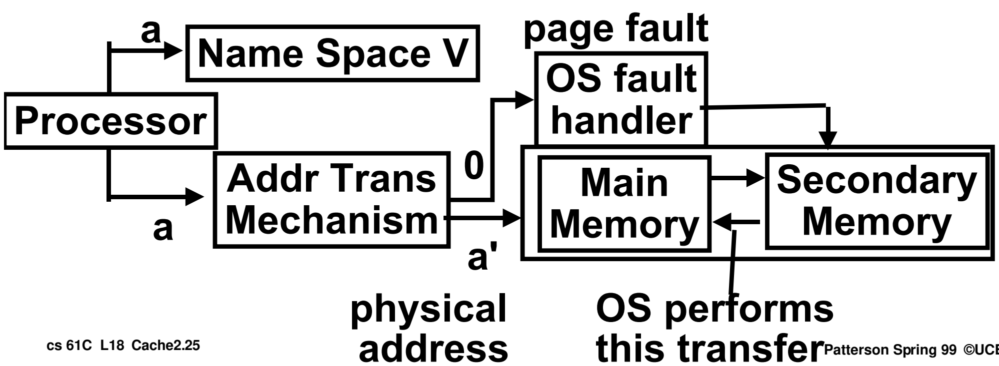
Page Table

- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the operating system has its own page table

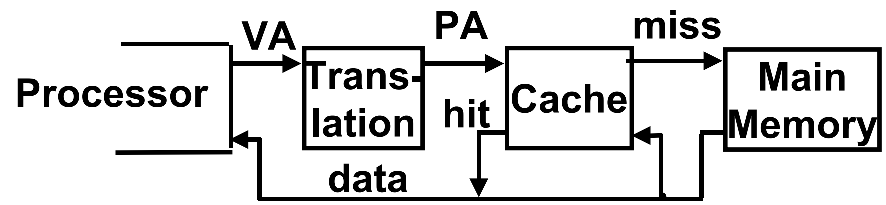
Address Map, Mathematically Speaking

V = {0, 1, . . . , n - 1}   virtual address space (n > m)
M = {0, 1, . . . , m - 1} physical address space
MAP: V --> M U {θ} address mapping function

MAP(a) = a' if data at virtual address <u>a</u> is present in physical address <u>a'</u> and <u>a'</u> in M
= θ if data at virtual address a is not present in M



Virtual Address and a Cache



- Extra memory access to translate VA to PA!
- This makes cache access very expensive, and you want to go as fast as possible

- Why access cache with physical addr. at all?
- Virtually Addressed caches have a problem!
  <u>*synonym / alias problem*</u>*:* 2 different virtual addresses map to same physical address
  $\Rightarrow$ 2 different cache entries holding data for the same physical address!

How Translate Fast?

- Observation: since locality in pages of data, must be locality in virtual addresses of those pages
- Why not use a cache of virtual to physical address translations to make translation fast? (small is fast)
- For historical reasons, cache is called a <u>Translation Lookaside Buffer</u>, or <u>TLB</u>

Typical TLB Format

Virtual Address	Physical Address	Dirty	Ref	Valid	Access Rights

- <u>Dirty</u>: since use write back, need to know whether or not to write page to disk when replaced
- <u>Ref</u>: Used to help calculate LRU on replacement

- TLB just a cache on the page table mappings

- TLB access time comparable to cache (much less than main memory access time)

TLB Miss (simplified format)

- If the address is not in the TLB, MIPS traps to the operating system
  - When in the operating system, we don't do translation
- The operating system knows which program caused the TLB fault, page fault, and knows what the virtual address desired was requested
  - So we look the data up in the page table

valid	virtual	physical
1	2	9

If the data is in memory

- We simply add the entry to the TLB, evicting an old entry from the TLB

valid	virtual	physical
1	7	32
1	2	9

What if the data is on disk?

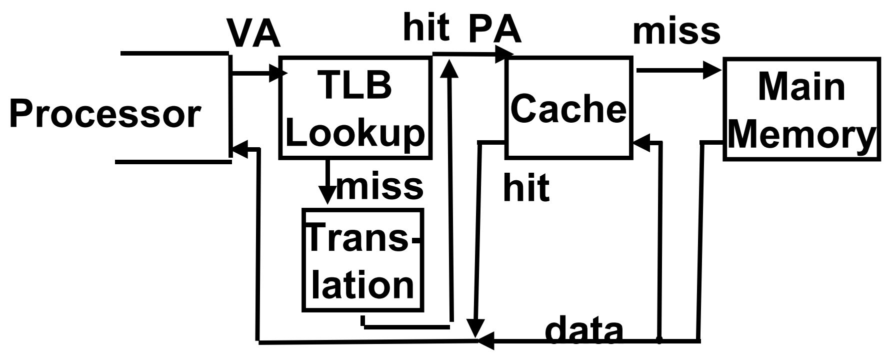
- We load the page off the disk into a free block of memory, using a DMA transfer
  - Meantime we switch to some other process waiting to be run
- When the DMA is complete, we get an interrupt and update the process's page table
  - So when we switch back to the task, the desired data will be in memory

What if we don't have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (other copy up-to-date), just overwrite that data
  - We chose the page to evict based on replacement policy
- And update that program's page table to reflect the fact that its memory moved somewhere else

Translation Look-Aside Buffers

- TLBs usually small, typically 128 - 256 entries
- Like any other cache, the TLB can be fully associative, set associative, or direct mapped



"And in Conclusion" 1/1

- Apply Principle of Locality Recursively
- Reduce Miss Penalty? add a (L2) cache
- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings vs. tag/data in cache
- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
- Next: WrapUp Memory Hierarchy