Review: (Write on the board before class)
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*)

Break:
Enjoyed meeting today
Regular lecture on April 2; Cray tape on April 6 (faculty retreat)
Reminder on exercise evaluation

Today: Advanced caches concepts & Pitfalls and fallacies

Faster Reads: Don't wait:

Early restart - As soon as requested word arrives

Out-of-order fetch - fetching missing word first and then continue

Lock-up free cache - continue in presence of miss

Problem with all ideas is spatial locality mkes it likely to request other parts of block soon (perhaps works better with data)

Faster hits: virtual caches

if cache is smaller than a page, then just use physical portion of address (set associativity multiplies cache size: IBM 3033 is 16-way)

must flush on conxtext switch unless use Process IDs

aliases: 2 virtual addresses for same physical address

Reducing Miss Penaly wtih 2-level caches:
L1 and L2 refer to a first-level and a second-level cache,

Average memory-access time = Hit $time_{L1}$ + Miss $rate_{L1}$ * (Hit $time_{L2}$ + Miss $rate_{L2}$ * Miss $penalty_{L2}$)

- *Local miss rate*—The number of misses in the cache divided by the total number of memory accesses to this cache; this is miss $rate_{L2}$ above.

n *Global miss rate*—The number of misses in the cache divided by the total number of memory accesses generated by the CPU; using the terms above, this is miss $rate_{L1} * miss\ rate_{L2}$.

Fig 8.39: linear and log plots of miss rates
Local 72% to 15%
Global 3% to 1%
Single cache miss rate (same size as global) 8% to 1%

Fig 8.40: Execution time vs speed and size of L2 cache

I/O and caches

Fig 8.43 explains coherency

I/O to cache OK, but knocks out valid data

I/O to memory: write through solves output
input to buffer: OS flushes buffer from cache or buffer in noncacheable page

Multiprocessor cohernecy

Directory: info on block is one place (loically a single directory)
Snooping: a copy of info every place there is a data

Snooping popular because single bus systems share the wires to each processor

Cost: duplicate the tags so that don't slow down all caches on any miss

Lots of options on cohernency. Especially what happens on a write
Write invalidate: writing processor invalidates all others
Write broadcast: writing processor updates all others

Figure 8.44 shows a protocol: 2 FSM to make it easier to follow
upper is normal, from local processor
lower is transitions that occur due to signals on bus (other processors)

Show the overall picture 8.52:

Fallacies and Pitfalls:

1) Pitfall: Too small an address space.

5 after DEC and CMU designed PDP-11, a fatal flaw. A IBM 360, 6 years **before** the PDP-11 is still thriving, with minor modifications, 25 years later. Bell and Strecker [1976] put it like this:

There is only one mistake that can be made in computer design that is difficult to recover from—not having enough address bits for memory addressing and memory management. The PDP-11 followed the unbroken tradition of nearly every known computer. [p. 2]

The reason the address size is so hard to change is that it determines the minimum width of anything that can contain an address: PC, register, memory word, and effective-address arithmetic. If there is no plan to expand the address from the start, then the chances of successfully changing address size are so slim that it normally means the end of that computer family.

2) Fallacy: Given the hardware resources, the computer designer who selects a set-associative cache over a direct-mapped cache of the same size will get a faster computer.

lower costs, faster hit times, and therefore smaller average access times for large, direct-mapped caches.

10% for off-the-shelf
3% for custom

3) *Pitfall: Address space can be extended by adding segments on top of a flat address space.*

During the 1970s, many programs grew to the point they couldn't address all of the code and data with just a 16-bit address. Machines were then revised to offer 32-bit addresses, either through a flat 32-bit address space or by adding 16 bits of segment to the existing 16-bit address. From the point of view of marketing, adding segments solves the addressing problem. Unfortunately, there is trouble any time a programming language wants an address that is larger than one segment, such as indices for large arrays, unrestricted pointers, or reference parameters. Moreover, adding segments can turn every address into two words—one for the segment number and one for the segment offset—causing problems in the use of addresses in registers. In the 1990s, 32-bit addresses will be exhausted

Discussion Questions:

8.22 [Discussion] <8.6> Some people have argued that with increasing capacity of memory storage per chip, virtual memory is an idea whose time has passed, and they expect to see it dropped from future computers. Find reasons for and against this argument.

8.24 [Discussion] <8.4> Recent research has tried to use compilers to improve cache performance (see McFarling [1989] and Samples and Hilfinger [1988]):

a. Which of the 3C's are compilers trying to improve and which are they not? Why?

b. Which mapping is best for compiler improvement? Why?

8.23 [Discussion] <8.6> So far, few computer systems take advantage of the extra security available with gates and rings found in a machine like the Intel 80286. Construct some scenario whereby the computer industry would switch over to this model of protection.

Conclusion: locatity: register windowsn (nesting depth), instruction buffer (spatial), TLB, cache, cache 2, virtual memory, file cache, disk cache,...

Smaller HW is faster and locality dominate memory system design; expect will do so for many years;

Next the Cinderella of Computer Architecture, the neglected stepchild, who triumphs in the end: I/O

Network speeds

1975 3Mb; 1982 10Mb; 1991? 100Mb: about 25% per year

Close to processor speedup, butr this is a step function at best adequate for 1991; rest of decade at 50% year? Processors at 40 to 400 times faster by end of decade

	Register windows	*Instruction-prefetch buffer*	*TLB*	*First-level cache*	*Second-level cache*	*Virtual memory*
Block size	64 bytes	1 byte	4 – 8 (1 PTE)	4 – 128 bytes	32–256 bytes	512 – 8192 bytes
Hit time	1 clock	1 clock	1 clock	1 – 4 clocks	4 – 10 clocks	1 – 10 clocks
Miss penalty	32 – 64 clocks	2 – 6 clocks	10 – 30 clocks	8 – 32 clocks	30 – 80 clocks	100,000 – 600,000 clocks
Miss rate (local)	1% – 3%	10% – 25%	0.1% – 2%	1% – 20%	15% – 30%	0.00001% – 0.001%
Size	512 bytes	6 – 12 bytes	32 – 8192 (8 – 1024 PTEs)	1 KB – 256 KB	256 KB – 4 MB	4 MB – 2048 MB
Backing store	First-level cache	First-level cache	First-level cache	Second-level cache	Static-column DRAM	Disks
Q1: block placement	Circular buffer	N.A. (Queue)	Set asso-ciative	Direct mapped	Set asso-ciative	Fully associative
Q2: block identification	2 registers: high and low	Valid bits + 1 register	Tag/ block	Tag/ block	Tag/ block	Table
Q3: block replacement	First in– first out	N.A. (Queue)	Random	N.A. (Direct mapped)	Random	LRU
Q4: write strategy	Write back	Flush on write to IB (if possible)	Flush on write to page table	Write through or write back	Write through or write back	Write back