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*)
Cache, Virtual Memory so far

Break:
Meeting today (bring signup sheet?)
Next Assignment: Any trouble with cache simulator 8.7
Review of questions so far?
Discussion questions for class: 8.12, 8.22, 8.23, 8.24
Pizza & beer Wednesday 11:30-1:30 $5 (make reservation)

Today: Advanced caches concepts & discussion questions

Instruction Prefetch Buffer

Variable sized instructions, when instruction memory bandwdith higher than CPU can consume instructions

See 8.31 (FIFO)

Register Windows

See 8.32, 8.33

8.35 shows that can reduce stores a lot, loads some

Compiler differences can overcome these differences

Putting it all together on the VAX

8.48

Ask qeustions what happens after a branch

1: PC virtual address sent to TLB

2: process/system bit + 5 bit index access both TLB banks

3: valid bit + page address (- 5 bits) compared

4: hit returns physical page address via 2:1 MUX

5: 2 CPU registers get virtual page address (VIBA) & physical address (PIBA)

6: physical address goes to cache; same steps occur

7: prefetch unit loaded with 4 bytes, with valid bits set; PIBA then asks for next 4 bytes

8: assume this instruction writes memory

9: address to TLB set dirty bit (trap in microcode if not set)

10: as before, except update cache block

11: write through, so update cache and send to write buffer

2.1 CPI out of 10.6 (OK out of 10; not so great from base of 1)

Advanced Topics

Fig 8.36 User miss rate vs. user+OS miss rate (user < 50% of actual miss rate!)

Faster Writes:

VAX 8800: Look for slot in write pipeline

Check tag

Write data

Separate tag from data; Check new tag, write old data to address!

Subbblock placement: Figure 8.37: partially valid blocks (just write to word & set valid bit)

Work with write through or write back? No - just write through since only valid copy may be in the cache

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.
- *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: Executive 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)