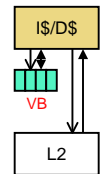


Two (of many possible) Optimizations

- **Victim buffer:** for conflict misses
- **Prefetching:** for capacity/compulsory misses

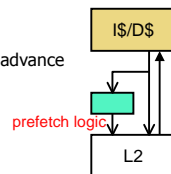
Victim Buffer

- Conflict misses: not enough associativity
 - High-associativity is expensive, but also rarely needed
 - 3 blocks mapping to same 2-way set and accessed (ABC)*
- **Victim buffer (VB):** small FA cache (e.g., 4 entries)
 - Sits on I\$/D\$ fill path
 - VB is small → very fast
 - Blocks kicked out of I\$/D\$ placed in VB
 - On miss, check VB: hit? Place block back in I\$/D\$
 - 8 extra ways, shared among all sets
 - + Only a few sets will need it at any given time
 - + Very effective in practice



Prefetching

- **Prefetching:** put blocks in cache proactively/speculatively
 - Key: anticipate upcoming miss addresses accurately
 - Can do in software or hardware
- Simple example: **next block prefetching**
 - Miss on address **X** → anticipate miss on **X+block-size**
 - Works for insns: sequential execution
 - Works for data: arrays
- **Timeliness:** initiate prefetches sufficiently in advance
- **Accuracy:** don't evict useful data

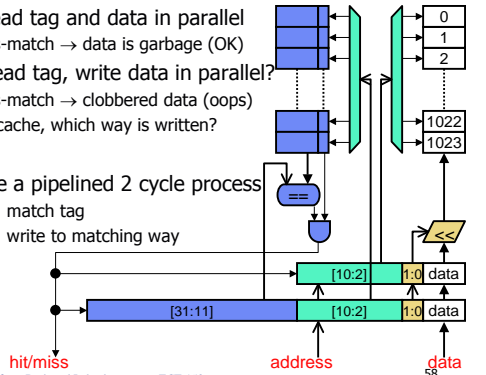


Write Issues

- So far we have looked at reading from cache (loads)
- What about writing into cache (stores)?
- Several new issues
 - Tag/data access
 - Write-through vs. write-back
 - Write-allocate vs. write-not-allocate

Tag/Data Access

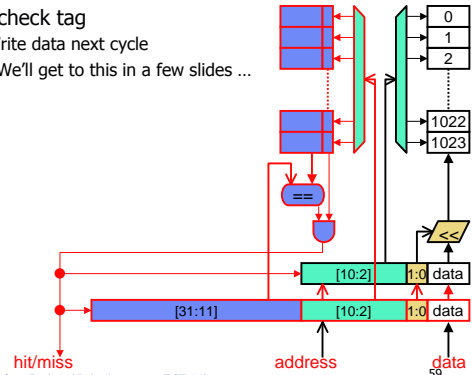
- Reads: read tag and data in parallel
 - Tag mis-match → data is garbage (OK)
- Writes: read tag, write data in parallel?
 - Tag mis-match → clobbered data (oops)
 - For SA cache, which way is written?
- Writes are a pipelined 2 cycle process
 - Cycle 1: match tag
 - Cycle 2: write to matching way



© 2008 Daniel J. Sorin from Roth and Lebeck ECE 152

Tag/Data Access

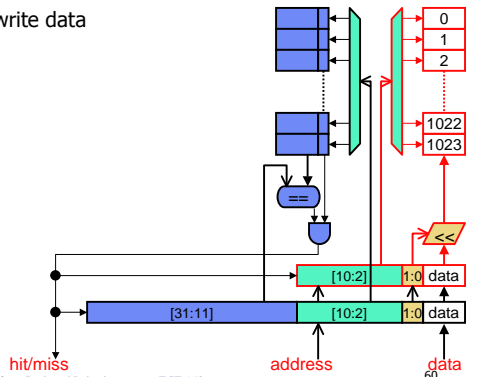
- Cycle 1: check tag
 - Hit? Write data next cycle
 - Miss? We'll get to this in a few slides ...



© 2008 Daniel J. Sorin from Roth and Lebeck ECE 152

Tag/Data Access

- Cycle 2: write data



© 2008 Daniel J. Sorin from Roth and Lebeck ECE 152

Write-Through vs. Write-Back

- When to propagate new value to (lower level) memory?
 - **Write-through:** immediately
 - + Conceptually simpler
 - + Uniform latency on misses
 - Requires additional bus bandwidth
 - **Write-back:** when block is replaced
 - + Minimal bus bandwidth
 - Only write back dirty blocks
 - Non-uniform miss latency
 - Clean miss: one transaction with lower level (fill)
 - Dirty miss: two transactions (writeback & fill)

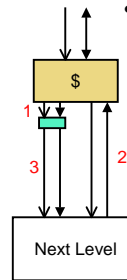
© 2008 Daniel J. Sorin from Roth and Lebeck ECE 152

61

Write-allocate vs. Write-non-allocate

- What to do on a write miss?
 - **Write-allocate**: read block from lower level, write value into it
 - + Decreases read misses
 - Requires additional bandwidth
 - Use with write-back
 - **Write-non-allocate**: just write to next level
 - Potentially more read misses
 - + Uses less bandwidth
 - Use with write-through

Write Buffer



- **Write buffer**: between cache and memory
 - Write-through cache? Helps with store misses
 - + Write to buffer to avoid waiting for memory
 - Store misses become store hits
 - Write-back cache? Helps with dirty misses
 - + Allows you to do read (important part) first
 1. Write dirty block to buffer
 2. Read new block from memory to cache
 3. Write buffer contents to memory

Typical Processor Cache Hierarchy

- First level caches: optimized for t_{hit} and parallel access
 - Insns and data in separate caches (**I\$, D\$**)
 - Capacity: 8–64KB, block size: 16–64B, associativity: 1–4
 - Other: write-through or write-back
 - t_{hit} : 1–4 cycles
- Second level cache (**L2**): optimized for $\%_{miss}$
 - Insns and data in one cache for better utilization
 - Capacity: 128KB–1MB, block size: 64–256B, associativity: 4–16
 - Other: write-back
 - t_{hit} : 10–20 cycles
- Third level caches (**L3**): also optimized for $\%_{miss}$
 - Capacity: 1–8MB
 - t_{hit} : 30 cycles

Performance Calculation Example

- Parameters
 - Reference stream: 20% stores, 80% loads
 - L1 D\$: $t_{hit} = 1ns$, $\%_{miss} = 5\%$, write-through + write-buffer
 - L2: $t_{hit} = 10ns$, $\%_{miss} = 20\%$, write-back, 50% dirty blocks
 - Main memory: $t_{hit} = 50ns$, $\%_{miss} = 0\%$
- What is $t_{avgL1D\$}$ without an L2?
 - Write-through+write-buffer means all stores effectively hit
 - $t_{missL1D\$} = t_{hitM}$
 - $t_{avgL1D\$} = t_{hitL1D\$} + \%_{loads} * \%_{missL1D\$} * t_{hitM} = 1ns + (0.8 * 0.05 * 50ns) = 3ns$
- What is $t_{avgD\$}$ with an L2?
 - $t_{missL1D\$} = t_{avgL2}$
 - Write-back (no buffer) means dirty misses cost double
 - $t_{avgL2} = t_{hitL2} + (1 + \%_{dirty}) * \%_{missL2} * t_{hitM} = 10ns + (1.5 * 0.2 * 50ns) = 25ns$
 - $t_{avgL1D\$} = t_{hitL1D\$} + \%_{loads} * \%_{missL1D\$} * t_{avgL2} = 1ns + (0.8 * 0.05 * 25ns) = 2ns$

Summary

- Average access time of a memory component
 - $t_{avg} = t_{hit} + \%_{miss} * t_{miss}$
 - Hard to get low t_{hit} and $\%_{miss}$ in one structure → hierarchy
- Memory hierarchy
 - Cache (SRAM) → memory (DRAM) → swap (Disk)
 - Smaller, faster, more expensive → bigger, slower, cheaper
- SRAM
 - Analog technology for implementing big storage arrays
 - Cross-coupled inverters + bitlines + wordlines
 - Delay $\sim \sqrt{\#bits} * \#ports$

Summary, cont'd

- Cache ABCs
 - Capacity, associativity, block size
 - 3C miss model: compulsory, capacity, conflict
- Some optimizations
 - Victim buffer for conflict misses
 - Prefetching for capacity, compulsory misses
- Write issues
 - Pipelined tag/data access
 - Write-back vs. write-through/write-allocate vs. write-no-allocate
 - Write buffer

Next Course Unit: Main Memory