CSCI 210: Computer Architecture Lecture 33: Caches II

Stephen Checkoway Slides from Cynthia Taylor

CS History: Jerry Lawson



Museum of Play / Estate of Jerry Lawson

- Born in Brooklyn in 1940
- "Father of the video game cartridge"
- Worked at Fairchild Semiconductor
- Developed the swappable video game cartridge
- Member of the Homebrew Computer Club
 - Interviewed Steve Wozniak for a position at Fairchild and didn't hire him

CACHE REPLACEMENT POLICIES

Cache policy for loads

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy and replace an existing one
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

Cache replacement policy

• On a hit, return the requested data

• On a miss, load block from lower level in the memory hierarchy and write in cache; return the requested data

 Policy: Where in cache should the block be written? (With direct-mapped caches, there's only one possible location: block_address % number_of_blocks_in_cache)

Cache policy for stores

- Policy choice for a hit: Where do we write the data?
 - Write-back: Write to cache only
 - Write-through: Write to cache and also to the next lower level of the memory hierarchy
- Policy choice for a miss
 - Write-allocate: Bring the block into cache and then do the write-hit policy
 - Write-around: Write only to memory

Store-hit policy: write-through

- Update cache block AND memory
- Makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

Store-hit policy: write-back

- Only update the block in cache
 - Keep track of whether each block is "dirty" (i.e., it has a different value than in memory)
- When a dirty block is replaced ("evicted")
 - Write it back to memory
 - Can use a write buffer
- Faster than write-through, but more complex

V	D	Тад	Data			
1	0	000042	FE	FF	3C	
0						
1	1	001234	65	82	5C	
0						
0						
1	0	000F3C	00	00	00	•••
0						
0						

W a	hat value(s) will we address 0xFFFF1234 and the cache block until afte	eventually write ? Assume a wr for 0xFFFF1234 er the three write	te to memory at ite back cache, its not evicted tes
A.	4	\$t3 holds Averent 234	
В.	5	\$t1 holds 4 \$t2 holds 5	
C.	6	\$t4 holds 6	
D.	We will write 4, then overwrite it with 5, then overwrite that with 6	sw \$t1, 0(\$t3) sw \$t2, 0(\$t3) sw \$t4, 0(\$t3)	Write-Back Policy: Only update the block in cache
E.	None of the above		When a dirty block is evicted write it back to memory

Store-miss policy: write-around

• Only write the data to memory

 Good for initialization where lots of memory is written at once but won't be read again soon

Store-miss policy: write-allocate

• Read a block from memory into the cache (just like a load miss)

• Perform the write according to the store-hit policy (i.e., write in cache or write in both cache and memory)

 Good for when data is likely to be read shortly after being written (temporal locality)

Common policy choices

- Write-back + write-allocate
 - Dirty blocks are written to memory only when replaced
 - Stores bring block into cache
 - Subsequent loads/stores will cause cache hits (unless the block is evicted)
- Write-through + write-around
 - Writes always go to memory
 - Cache is mostly for loads

ASSOCIATIVE CACHES

Direct-mapped Cache

- Each block goes into **1** spot
- Only search one entry on lookup
- Associativity = 1

• What if we allow blocks to go into more than one spot?



Direct mapped

Fully-associative Cache

- Allow a given block to go in any cache entry
- Requires all entries to be searched at once
- Comparator per entry (expensive)

Fully associative





n-way Set-associative Cache

- Each set contains *n* entries
- Block number determines which set
 - (Block address) modulo (#Sets in cache)
- Search all entries in a given set at once
- n comparators (less expensive)





Spectrum of associativity for 8-entry cache

One-way set associative

(direct mapped)





Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data														

Memory addresses, block addresses, offsets

- Block size of 32 bytes (not bits!)
- 16-block, 2-way set associative cache
- Each address
 - A (32 5)-bit block address (in purple and blue)
 - A 5-bit offset into the block (in green)
- Block address can be divided into
 - A (32 3 5)-bit **tag** (purple)
 - A 3-bit cache index (blue)

V	Тад	Data	V	Тад	Data
0			0		
0			0		
0			1	3F2084	•••
0			0		
0			0		
1	15C9AC		1	28477D	•••
0			0		
0			0		

Set Associative Cache Organization



Given a 256-entry, 8-way set associative cache with a block size of 64 bytes, how many bits are in the tag, index, and offset?

	Tag bits	Index bits	Offset bits
А	32 - 5 - 6 = 21	5	6
В	32 – 3 – 5 = 24	3	5
С	32 - 8 - 6 = 18	8	6
D	32 - 6 - 5 = 21	6	5
Е	32 - 6 - 3 = 23	6	3

Given a 256-entry, fully associative cache with a block size of 64 bytes, how many bits are in the tag, index, and offset?

	Tag bits	Index bits	Offset bits
А	32 - 5 - 6 = 21	1	6
В	32 - 3 - 5 = 24	3	5
С	32 - 8 - 6 = 18	8	6
D	32 - 6 - 5 = 21	6	5
E	32 - 0 - 6 = 26	0	6

Replacement Policy

- Direct mapped: no choice
- Set associative
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set
 - Goal: Choose an entry we will not use in the future

Replacement Policy

- Least-recently used (LRU)
 - Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
 - Gives approximately the same performance as LRU for high associativity

Associativity Example

- Compare 4-block caches
 - Direct mapped, 2-way set associative, fully associative
 - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block	Cache	Hit/miss	Cache content after access					
address	index		0	1	2	3		
0	0							
8	0							
0	0							
6	2							
8	0							

Associativity Example: 0, 8, 0, 6, 8

• 2-way set associative

Block	Cache	Hit/miss	Cache content after access			
address	index		Set 0		Set 1	
0	0					
8	0					
0	0					
6	0					
8	0					

Fully associative

Block address	Hit/miss	Cache content after access				
0						
8						
0						
6						
8						

Reading

- Next lecture: More Caches!
 - Section 6.4