Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Pop Quiz

- Reduce ideal CPI from 2->1
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- CPI from previous slide: 5.44
  - We spent 3.44/5.44 (63%) on memory stalls
- How much on CPU w/ improved CPI?
- A) 59% B) 63%
- C) 77% D) 82%

Clock Rate and Cache Performance

- If we double the clock rate of the processor, we don't change:
  - Cache miss rate
  - Miss penalty (memory is not likely to change)
- The cache will not improve, so the speedup is not close to double!

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

Reducing Miss Rate

- Obviously a larger cache will reduce the miss rate!
- We can also reduce miss rate by reducing the *competition* for cache slots.
  - allow a block to be placed in one of many possible cache slots.

Destroy Direct Mapped Cache

- Assume that every 64th memory element maps to the same cache slot

```
for (i=0;i<10000;i++) {
  a[i] = a[i] + a[i+64] + a[i+128];
  a[i+64] = a[i+64] + a[i+128];
}
```

- a[i], a[i+64], and a[i+128] use the same cache slot!

Associative Caches

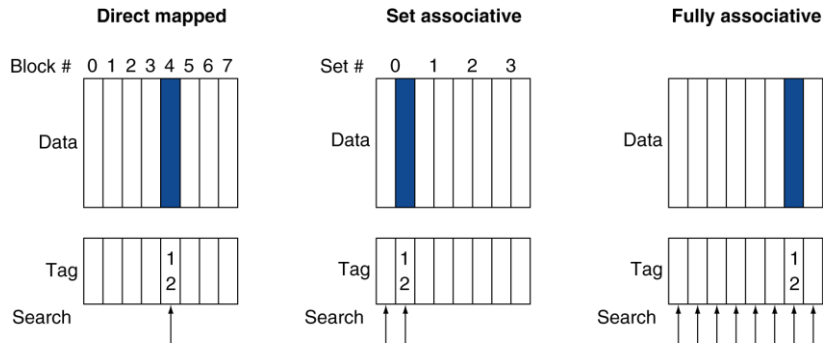
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Pop Quiz

- Caches with larger associativity miss less frequently than an identical cache with a smaller associativity.

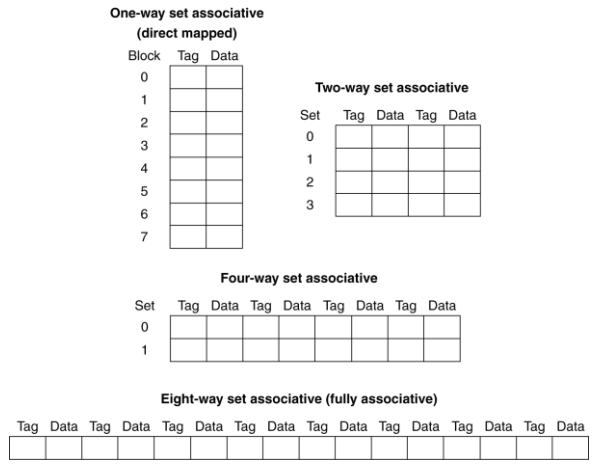
- A: True
- B: False

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries



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 address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	