

# CSE 161 – Design and Architecture of Computer Systems

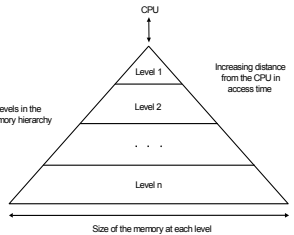
## Chapter Seven

## Memories: Review

- **SRAM:**
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)
- **DRAM:**
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)

## Exploiting Memory Hierarchy

- **Users want large and fast memories!**
  - SRAM access times are 2 - 25ns at \$100 to \$250 / MB
  - DRAM access times are 60-120ns at \$5 to \$10 / MB **1997!**
  - Disk access times are 10 to 20 million ns at \$.10 to \$.20 / MB
- **Speed and cost today?**
- **Try and give it to them anyway**
  - build a memory hierarchy



## Locality

- **A principle that makes having a memory hierarchy a good idea**
- **If an item is referenced,**
  - **temporal locality:** it will tend to be referenced again soon
  - **spatial locality:** nearby items will tend to be referenced soon.
- Why does code have locality?*
- **Our initial focus: two levels (upper, lower)**
  - **block:** minimum unit of data
  - **hit:** data requested is in the upper level
  - **miss:** data requested is not in the upper level

## Cache

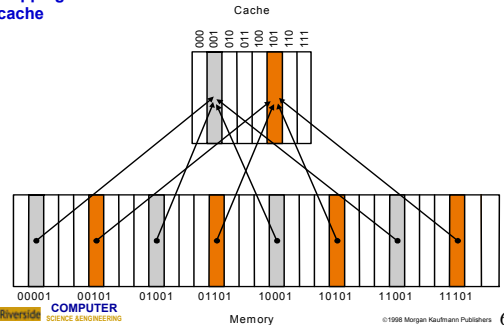
- **Two issues:**
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- **Our first example:**
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

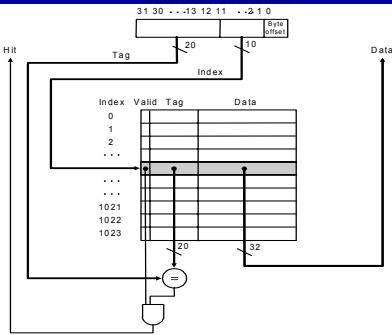
## Direct Mapped Cache

- **Mapping: address is modulo the number of blocks in the cache**



## Direct Mapped Cache

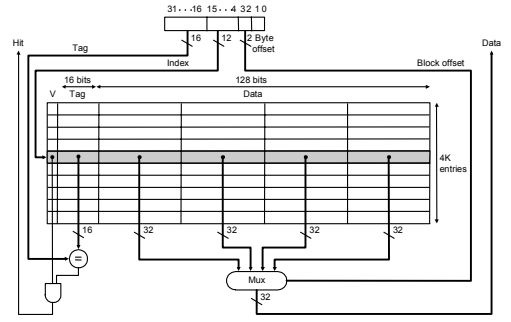
### For MIPS:



What kind of locality are we taking advantage of?

## Direct Mapped Cache

### Taking advantage of spatial locality:



## Hits vs. Misses

### Read access:

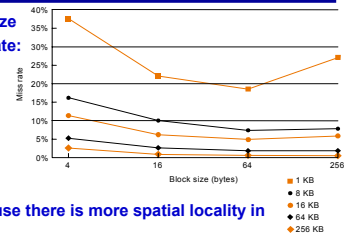
- Hits - this is what we want!
- Misses - stall the CPU, fetch block from memory, deliver to cache, restart

### Write access:

- Write Hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- Write misses:
  - read the entire block into the cache, then write the word
- Write allocate: should we fetch a block on a write miss?
  - When do we have to?

## Performance

### Increasing the block size tends to decrease miss rate:



### Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

## Performance

### Simplified model:

$$\text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}$$

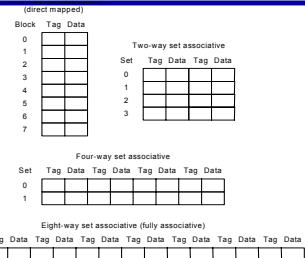
$$\text{stall cycles} = \# \text{ of mem. instructions} \times \text{miss ratio} \times \text{miss penalty}$$

### Two ways of improving performance:

- decreasing the miss ratio
- decreasing the miss penalty

What happens if we increase block size?

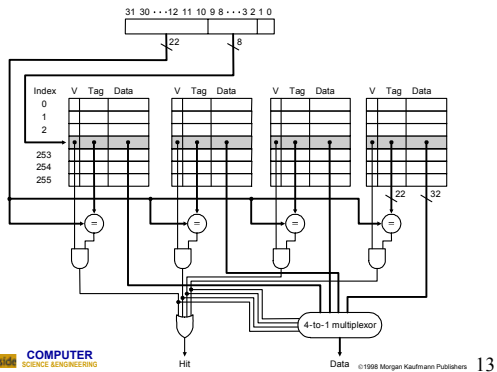
## Decreasing miss ratio with associativity



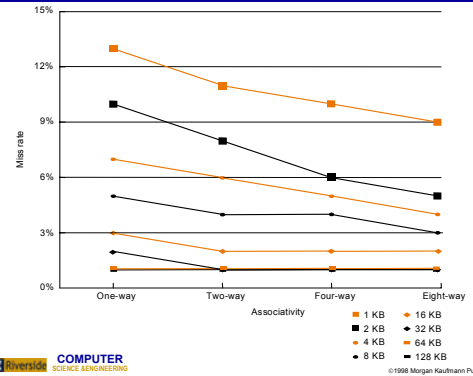
Compared to direct mapped, give a series of references that:

- results in a lower miss ratio using a 2-way set associative cache
- results in a higher miss ratio using a 2-way set associative cache assuming we use the "least recently used" replacement strategy

## An implementation



## Performance



## Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- **Example:**
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache