Today’s topics:

- Cache access details
- Examples
Locality

• Why do caches work?
  ▪ Temporal locality: if you used some data recently, you will likely use it again
  ▪ Spatial locality: if you used some data recently, you will likely access its neighbors

• No hierarchy: average access time for data = 300 cycles

• 32KB 1-cycle L1 cache that has a hit rate of 95%:
  average access time = 0.95 x 1 + 0.05 x (301) = 16 cycles
Accessing the Cache

Direct-mapped cache: each address maps to a unique location in cache

8-byte words

Sets

Offset

101000

Byte address

Data array

8 words: 3 index bits
The Tag Array

Direct-mapped cache: each address maps to a unique address

8-byte words

Tag array

Data array

Byte address

Tag

Compare

101000
Example Access Pattern

Direct-mapped cache: each address maps to a unique address

Byte address

Tag array

Tag

Compare

101000

Data array

8-byte words

Assume that addresses are 8 bits long
How many of the following address requests are hits/misses?
4, 7, 10, 13, 16, 68, 73, 78, 83, 88, 4, 7, 10…

Tag array

Data array
Increasing Line Size

- A large cache line size → smaller tag array, fewer misses because of spatial locality.
- 32-byte cache line size or block size.

Diagram:
- Byte address
- Tag
- Offset
- 10100000
- Tag array
- Data array

- Tag array with a byte address 10100000
- Data array with offset

- 32-byte cache line size or block size
Associativity

Byte address

10100000

Tag

Tag array

Set associativity $\rightarrow$ fewer conflicts; wasted power because multiple data and tags are read

Compare

Way-1

Way-2

Data array
Associativity

How many offset/index/tag bits if the cache has 64 sets, each set has 64 bytes, 4 ways
Example 1

• 32 KB 4-way set-associative data cache array with 32 byte line sizes

• How many sets?

• How many index bits, offset bits, tag bits?

• How large is the tag array?
Example 1

• 32 KB 4-way set-associative data cache array with 32 byte line sizes

    cache size = #sets x #ways x block size

• How many sets?  256

• How many index bits, offset bits, tag bits?
    8       5       19

• How large is the tag array?
    tag array size = #sets x #ways x tag size
    = 19 Kb = 2.375 KB
Example 2

• A pipeline has CPI 1 if all loads/stores are L1 cache hits
  40% of all instructions are loads/stores
  85% of all loads/stores hit in 1-cycle L1
  50% of all (10-cycle) L2 accesses are misses
Memory access takes 100 cycles
What is the CPI?
Example 2

- A pipeline has CPI 1 if all loads/stores are L1 cache hits
- 40% of all instructions are loads/stores
- 85% of all loads/stores hit in 1-cycle L1
- 50% of all (10-cycle) L2 accesses are misses

Memory access takes 100 cycles

What is the CPI?

Start with 1000 instructions
1000 cycles (includes all 400 L1 accesses)
+ 400 (ld/st) x 15% x 10 cycles (the L2 accesses)
+ 400 x 15% x 50% x 100 cycles (the mem accesses)
= 4,600 cycles

CPI = 4.6
Assume that addresses are 8 bits long. How many of the following address requests are hits/misses?

4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4
Example 3

Assume that addresses are 8 bits long.
How many of the following address requests are hits/misses?
4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4
M H M H M M M H M M H M M M M

Way-1

Way-2

Data array

8-byte blocks
Cache Misses

• On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)

• On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  ➢ no choice for a direct-mapped cache
  ➢ randomly pick one of the ways to replace
  ➢ replace the way that was least-recently used (LRU)
  ➢ FIFO replacement (round-robin)
Writes

• When you write into a block, do you also update the copy in L2?
  - write-through: every write to L1 $\rightarrow$ write to L2
  - write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2

• Writeback coalesces multiple writes to an L1 block into one L2 write

• Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data
Types of Cache Misses

- Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

- Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache