Memory Hierarchy Organization

Arquitecturas Paralelas de Computadoras

Profr. Carlos Ernesto Carrillo Arellano[1]

[1]Universidad Autónoma Metropolitana - Unidad Azcapotzalco
Departamento de Ingeniería Electrónica
Correo electrónico: *ceca@xanum.uam.mx*
Personal web page: *http://ecarrillo.ddns.net*

Mayo, 2019

Contenido

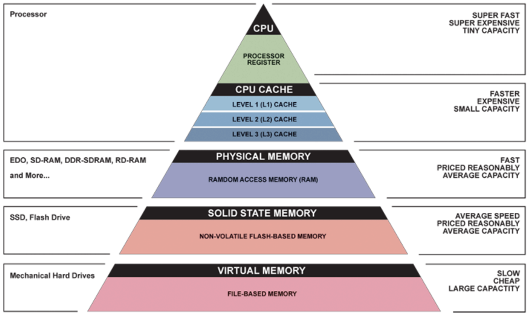
Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Contenido

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Introduction

1. The memory hierarchy consists of levels of temporary storage managed by hardware, referred to as **caches**.
2. To get the **most performance from a parallel program**, it is necessary to tune it to conform to the machine architecture on which it runs.
3. One of the **biggest obstacle** in parallel program performance tuning is probably the complex memory hierarchy present in current computer systems.
4. **Access to data**: upper level cache, access time of only a few processor clock cycles, lower level cache, access time of tens of clock cycles, local main memory, access time of hundreds of clock cycles, remote memory, access time of up to thousands of clock cycles.

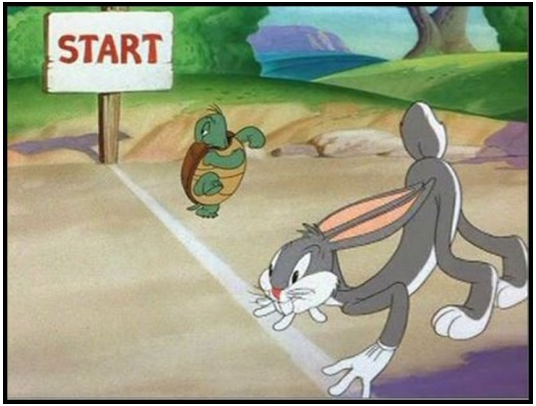


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Motivation

1. A good understanding of how memory hierarchy is organized is crucial for (1) knowing how to **tune the performance of parallel programs**
2. The increase in **CPU speed** has been much faster than the **decrease in the access latency** of the main memory.
3. CPU speed as measured in its clock frequency grew at the rate of **55 %** annually, while the memory speed grew at the rate of only **7 %** annually
4. **In the past**: a load instruction could get the needed datum from main memory in one CPU clock cycle
5. **Today**: It requires hundreds of processor clock cycles to get a datum from the main memory.
6. **Memory wall** problem



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

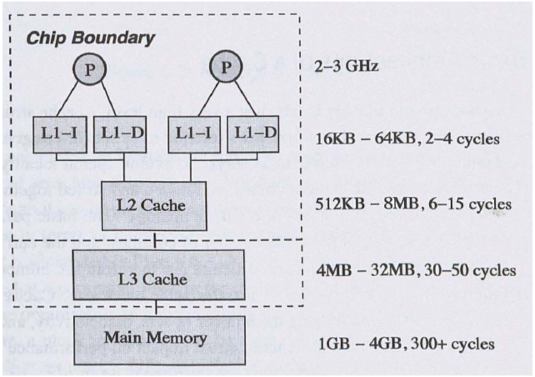
Motivation

1. Dependences between a load instruction (**producer**) and instructions that use the loaded value (**consumers**) dictate that the consumer instructions must wait until the load obtains its datum before they can execute
2. With the latency of loading datum from the main memory in the order of **hundreds of cycles**, the **CPU will stall for much** of that time because it runs out of instructions that are not dependent on the load.
3. It is critical to performance that **most data accesses are supplied to the CPU with low latencies**
4. **Caches** provide such support.

Metropolitana
Casa abierta al tiempo Azcapotzalco

Cache

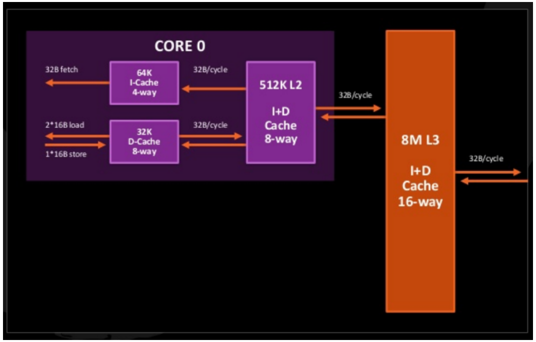
1. A **cache** is a relatively small and temporary storage for keeping data that is likely needed by the requestor.
2. Current memory hierarchy example
3. Private Levell (LI) data cache and a Level 1 instruction cache for each processor
4. Single Level 2 (L2) cache that holds both instructions and data.

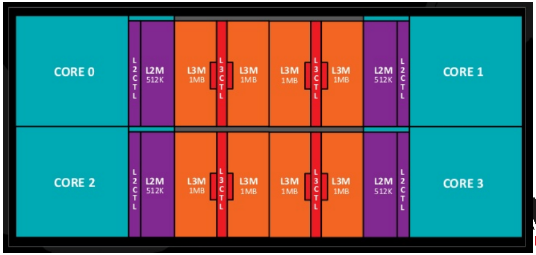


Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Example: AMD Ryzen^{TM} Threadripper^{TM}

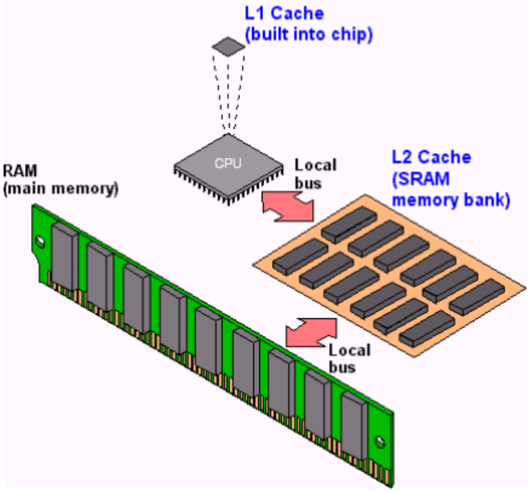
1. 32 Cores
2. Cache L1 total 3MB (64KB I and 32KB D per core)
3. Caché L2 total 16MB (512KB per core)
4. Caché L3 total 64MB (8MB Shared among 4 cores)





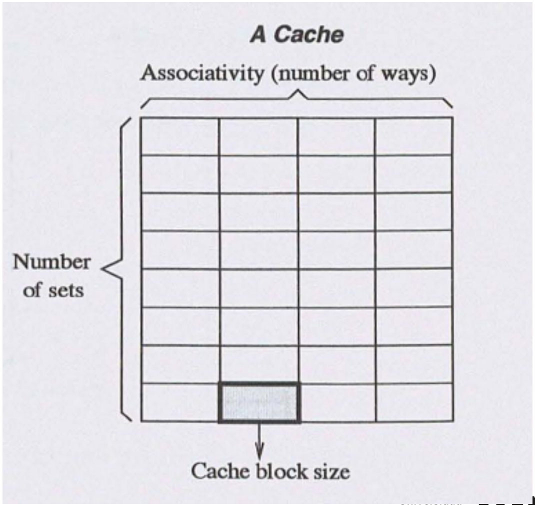
Cache Memory

1. Due to **temporal locality behavior** naturally found in programs, **caches are an effective structure to keep most useful data closer to the processor**.
2. By simply keeping more **recently used data in the cache** while discarding **less recently used data, caches achieve that goal**.
3. If the processor requests data that is available in the cache, it is returned quickly. This is called a **cache hit**
4. Otherwise, the processor retrieves the data from main memory (DRAM). This is called a **cache miss**.

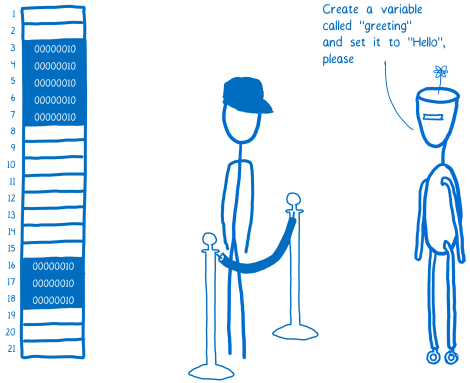


Basic Architectures of a Cache

1. At a very high level, a cache structure can be thought of as a table
2. With multiple rows referred to as **sets** or **congruence classes**, and multiple columns referred to as **ways**.
3. To exploit spatial locality and reduce management overheads, multiple bytes are **fetched and stored together as a cache block**.
4. The basic parameters of a cache structure are the **number of sets**, **associativity** (or the number of ways), and the **cache block size**.
5. **Cache size** can be computed simply by multiplying the number of sets, associativity, and the block size.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco



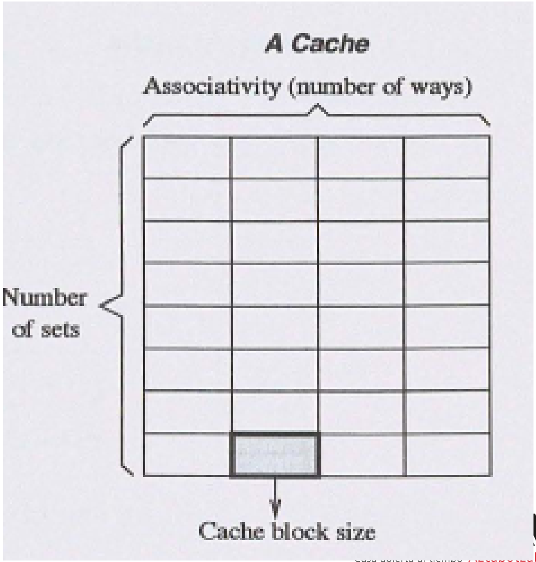
Management of cache

1. To manage a cache, several aspects must be considered.
2. The first is **placement policy** that decides where a memory block can be placed in the cache.
3. The second is **replacement policy** that decides which block needs to be evicted to make room for a new block.
4. The third is **write policy** that decides that when parts of a cache block is written, when the new value should be propagated to the lower level memory hierarchy component.

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Placement policy

1. The placement policy decides **where a block in memory can be placed in the cache**.
2. Must be **locatable** in future accesses to the block.
3. To simplify the process of searching a block, the placement of a block is **restricted to a single set**, although it can be placed in any way in that set.
4. To locate a block in the cache, a cache employs **three steps**:
5. **Determine the set** to which the block can map.
6. Access the set to **determine if any of the blocks in the set matches** the one that is being requested.
7. If the block is found, **find the byte or word that is requested**, and return it to the processor.



Cache memory

1. An ideal cache would anticipate all of the data needed by the processor and fetch it from main memory ahead of time so that the cache has a zero miss rate.
2. The computer system designer must choose what subset of the main memory is kept in the cache.
3. When the processor attempts to access data, it first checks the cache for the data If the cache hits, the data is available immediately.
4. If the cache misses, the processor fetches the data from main memory and places it in the cache for future use.

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

$$\textit{Miss Rate} = \frac{\textit{Number of misses}}{\textit{Number of total memory accesses}} = 1 - \textit{Hit Rate}$$

$$\textit{Hit Rate} = \frac{\textit{Number of hits}}{\textit{Number of total memory accesses}} = 1 - \textit{Miss Rate}$$

Miss and Hit Rate

1. Quantitative ways to measure the performance of memory systems to evaluate the cost-benefit trade-offs of various alternatives.
2. Miss Rate
3. Hit rate

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Example

1. Suppose a program has 2000 data access instructions (loads or stores), and 1250 of these requested data values are found in the cache. The other 750 data values are supplied to the processor by main memory or disk memory. What are the miss and hit rates for the cache?

Average memory access time (AMAT)

1. Average time a processor must wait for memory per load or store instruction.
2. The processor first looks for the data in the cache. If the cache misses, the processor then looks in main memory.
3. If the main memory misses, the processor accesses virtual memory on the hard disk.

$$AMAT = t_{\text{cache}} + MR_{\text{cache}}(t_{MM} + MR_{MM}t_{VM})$$

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

AMAT Example

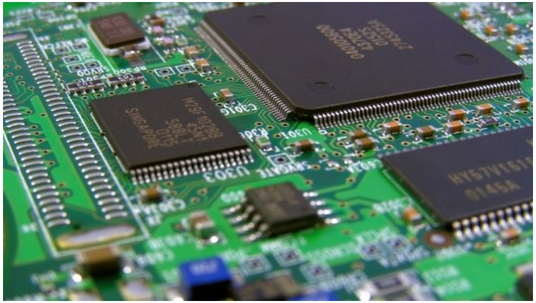
1. Suppose a computer system has a memory organization with only two levels of hierarchy, a cache and main memory. What is the average memory access time given the access times and miss rates?
2. What cache miss rate is needed to reduce the average memory access time to 1.5?

Memory Level	Access Time (Cycles)	Miss Rate
Cache	1	10%
Main Memory	100	0%

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Temporal and Spatial locality

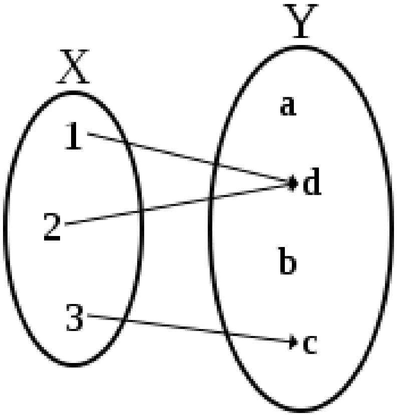
1. An ideal cache would **anticipate** all of the data needed by the processor and fetch it from main memory so that the cache has a zero miss rate.
2. It is impossible to predict the future with perfect accuracy, the cache must guess what data will be needed based on the pattern of memory accesses.
3. **Temporal locality** means that the processor is likely to access a piece of data again soon if it has accessed that data recently.
4. **Spatial locality** means that, when the processor accesses a piece of data, it is also likely to access data in nearby memory locations.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Mapping

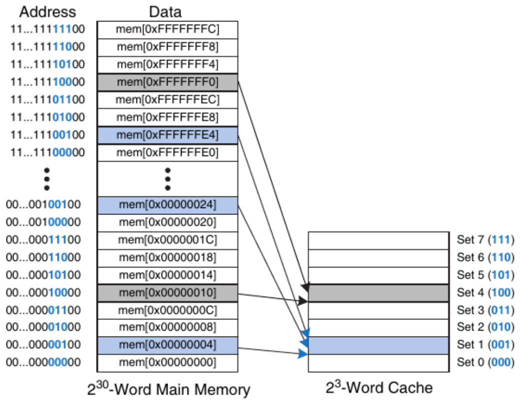
1. The relationship between the address of data in main memory and the location of that data in the cache is called the mapping.
2. Each memory address maps to exactly one set in the cache.
3. Some of the address bits are used to determine which cache set contains the data.
4. If the set contains more than one block, the data may be kept in any of the blocks in the set.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Direct Mapped Cache

1. Caches are categorized based on the **number of blocks** in a set.
2. In a **direct mapped cache**, each set contains exactly one block
3. Thus, a particular main memory address maps to a unique block in the cache.
4. 32-bit addresses and 32-bit words. The memory is byte-addressable, and each word is four bytes, so the memory consists of 2^{30} words aligned on word boundaries.



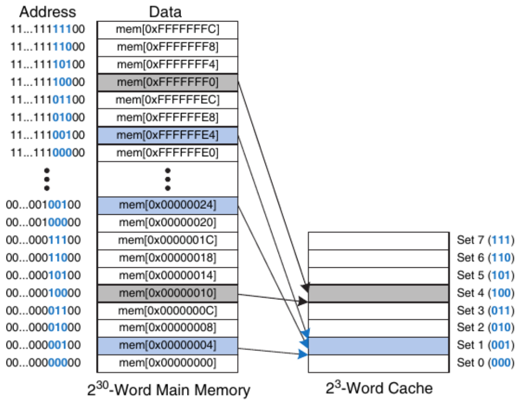
Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Direct Mapped Cache

1. A direct mapped cache has one block in each set, so it is organized into **S = B sets**.
2. An address in block 0 of main memory maps to set 0 of the cache. An address in block 1 of main memory maps to set 1 of the cache, and so forth
3. Block B of main memory maps to block 0 of the cache.
4. Each main memory address maps to exactly one set in the cache.

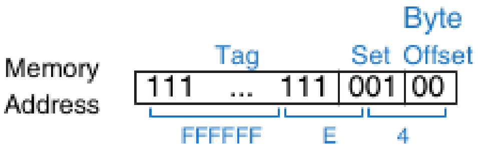
Example

To what cache set does the word at address 0x00000014 map?



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Direct Mapped Cache



Example

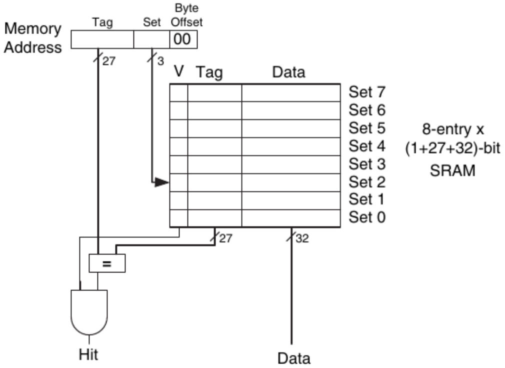
Find the number of set and tag bits for a direct mapped cache with 1024 sets and address size is 32 bits.

1. Because many addresses map to a single set, the cache must also keep track of the address of the data actually contained in each set.
2. Tag + Set + Byte Offset
3. **Tag** indicates which of the many possible addresses is held in the set.
4. Byte offset they indicate the byte within the word.
5. Set indicates which set holds the data.

Universidad Autónoma Metropolitana Casa abierta al tiempo Azcapotzalco

Direct Mapped Cache

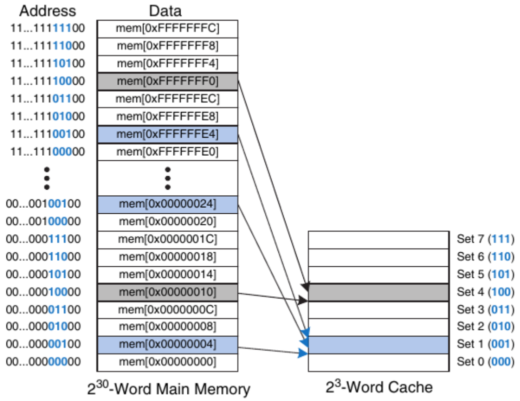
1. Each entry, or set, contains one line consisting of 32 bits of data, 27 bits of tag, and 1 valid bit.
2. The cache is accessed using the 32-bit address.
3. Byte offset bits, are ignored for word accesses.
4. Set bits, specify the entry or set in the cache.
5. If the tag matches the most significant 27 bits of the address and the valid bit is 1, the **cache hits and the data is returned to the processor**.
6. Two recently accessed addresses map to the same cache block, a conflict occurs, and the most recently accessed address evicts the previous one from the block.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Example: What is the miss rate?

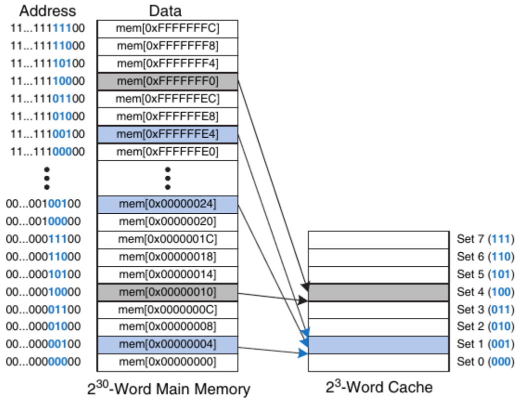
```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Example: What is the miss rate?

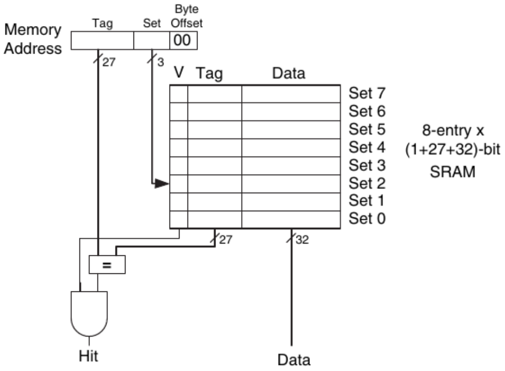
```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Example: What is the miss rate?

```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

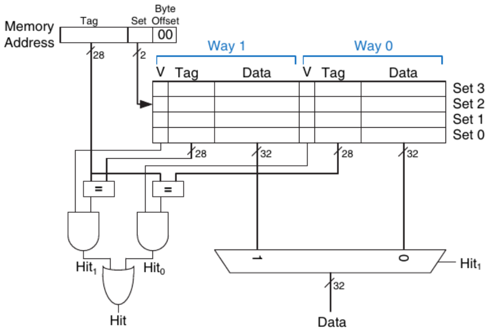
Example: What is the miss rate?

```
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Multi-way Set Associative Cache

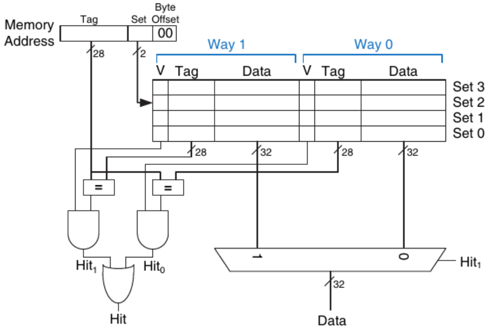
1. An N-way set associative cache reduces conflicts by providing N blocks in each set where data mapping to that set might be found.
2. Each memory address still maps to a specific set, but it can map to any one of the N blocks in the set. N is also called the degree of associativity of the cache.
3. N is also called the degree of associativity of the cache.
4. The cache reads blocks from both ways in the selected set and checks the tags and valid bits for a hit. If a hit occurs in one of the ways, a multiplexer selects data from that way.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Multi-way Set Associative Cache

1. Set associative caches generally have lower miss rates than directm mapped caches of the same capacity, because they have fewer conflicts.
2. Set associative caches are usually slower and somewhat more expensive to build because of the output multiplexer and additional comparators.



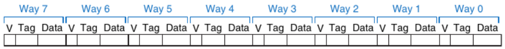
Example: What is the miss rate?

```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0x24($0)
       addi $t0, $t0, -1
       j    loop
done:
```

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Fully Associative Cache

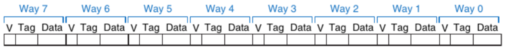
1. A fully associative cache contains a single set with B ways, where B is thenumber of blocks.
2. A memory address can map to a block in any of these ways.
3. Upon a data request, eight tag comparisons must be made, because the data could be in any block.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Fully Associative Cache

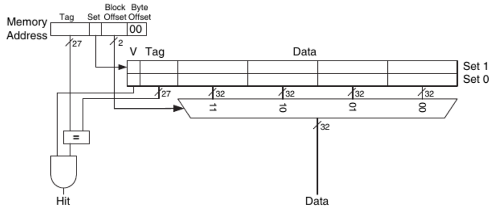
1. Fully associative caches tend to have the fewest conflict misses for a given cache capacity, but they require more hardware for additional tag comparisons
2. A memory address can map to a block in any of these ways.
3. Upon a data request, eight tag comparisons must be made, because the data could be in any block.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Block Size

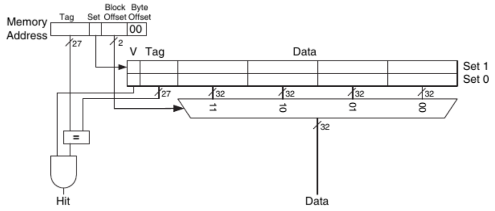
1. To exploit spatial locality,a cache uses larger blocks to hold several consecutive words.
2. The advantage of a block size greater than one is that when a miss occurs and the word is fetched into the cache, the adjacent words in the block are also fetched.
3. Subsequent accesses are more likely to hit because of spatial locality
4. A large block size means that a fixed-size cache will have fewer blocks.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Block Size

1. It takes more time to fetch the missing cache block after a miss, because more than one data word is fetched from main memory.
2. The time required to load the missing block into the cache is called the **miss penalty**.
3. If the adjacent words in the block are not accessed later, the effort of fetching them is wasted.
4. Most real programs benefit from larger block sizes.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Replacement Policy

1. In a **direct mapped cache**, each address maps to a unique block and set. If a set is full when new data must be loaded, the block in that set is **replaced with the new data.**
2. In **set associative and fully associative caches**, the cache must choose which block to **evict** when a cache set is full.
3. The principle of temporal locality suggests that the **best choice is to evict the least recently used block (LRU)**, because it is least likely to be used again soon.
4. The LRU replace exploits temporal locality in code because recently accessed data tend to be accessed again in the near future.

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Least Recently Used

1. LRU can be implemented **cheaply** for lowly-associative caches, but is **expensive** to implement for highly-associative caches,
2. Designs for highly-associative caches use **approximations to LRU** instead of true LRU
3. A possible approximation to LRU is by keeping track of only a few most recently used blocks, and on replacement, a block from among blocks that are not the few most recently used ones is selected, either randomly or based on a certain algorithm

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Least Recently Used

1. In a two-way set associative cache, a use bit, **U**, indicates which way within a set was least recently used.
2. Each time one of the ways is used, **U is adjusted to indicate the other way**.
3. For set associative caches with more than two ways, tracking the least recently used way becomes complicated.
4. To simplify the problem, the ways are often divided into two groups and U indicates which group of ways was least recently used. Upon replacement, the new block replaces a random block within the least recently used group. Such a policy is called **pseudo-LRU** and is good enough in practice.

```
lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

	Way 1			Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	0	00...010	mem[0x00...24]	1	00...000	mem[0x00...04]	Set 1 (01)
0	0			0			Set 0 (00)

(a)

	Way 1			Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	1	00...010	mem[0x00...24]	1	00...101	mem[0x00...54]	Set 1 (01)
0	0			0			Set 0 (00)

(b)

Write Policy

1. A cache is a temporary storage.
2. The main storage for programs is the main memory.
3. Memory stores, or writes, follow a similar procedure as loads.
4. Upon a memory store, the processor checks the cache.
5. If the cache misses, the cache block is fetched from main memory into the cache, and then the appropriate word in the cache block is **written**.
6. If the cache hits, the word is simply **written** to the cache block.

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Important Question

Question

So intuitively, an important question in cache design is that when a processor changes the value of bytes at a cache block, when should that change be propa- g.a ted to the lower level memory hierarchy?

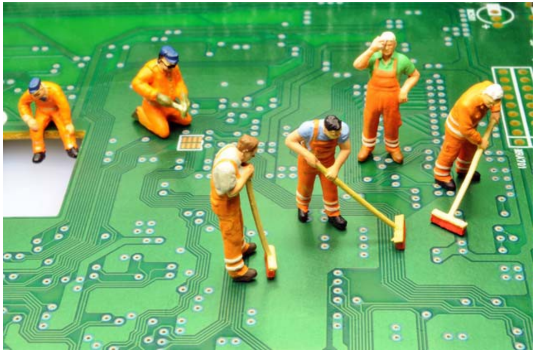
Write-through or write-back

1. There are two choices for implementation: **write-through** and **write back** policies.
2. In a **write-through** cache, the data written to a cache block is **simultaneously** written to main memory.
3. In a **write-back** cache, a dirty bit (D) is associated with each cache block.
4. When a new cache block is installed in the cache, its dirty bit is cleared.
5. D is 1 when the cache block has been modified and 0 otherwise.
6. Dirty cache blocks are written back to main memory only when they are evicted from the cache.

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Write-through or write-back

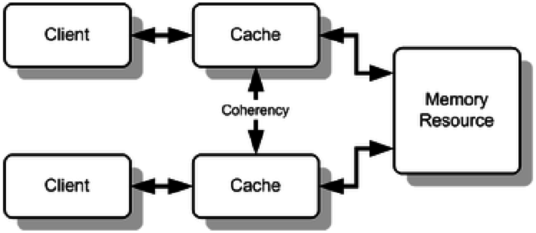
1. The write back policy tends to **conserve bandwidth** usage between the cache and its lower level memory hierarchy
2. There are often multiple writes to a cache block during the time the cache block resides in the cache.
3. In the write-through policy, each write propagates its value down so it **consumes bandwidth** on each write.



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Memory hierarchy coherence

1. Memory coherence is a fundamental property of any memory hierarchy and it becomes critical in multiprocessors.
2. In a single core system, instructions may be executed out-of-order and speculatively
3. At the end, **the result of any execution must be the same as if instrcutios were executed one at a time in process order**
4. A *load* instruction always return the value of the pervious *store* to the same address
5. To maintain coherence in the hierarchy, **updates are propagated to the lower level** in the memory hierarchy



Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco