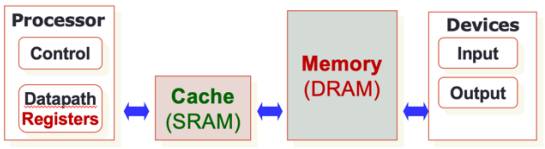
Cache and Memory

Cache



make SLOW main memory appear faster

Cache: a small but fast SRAM near CPU

Hardware managed: Transparent to programmer

Basic Idea: Keep the frequently and recently used data in **smaller but faster** memory

Refer to bigger and slower memory: Only when you cannot find data/instruction in the faster memory

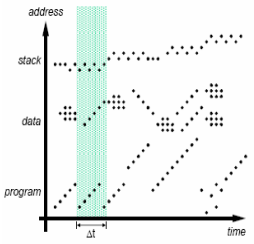
Principle of Locality: Program accesses only a small portion of the memory address space within a small time interval

Temporal locality: If an item is referenced, **it** will tend to be referenced again soon

Spatial locality: If an item is referenced, **nearby items** will tend to be referenced soon

Working Set: Set of locations accessed during Δt.

- Different phases of execution may use different working sets.
- Our aim is to **capture the working set** and keep it in the memory **closest** to CPU



Terminology

Hit: Data is in cache

Hit rate: Fraction of memory accesses that hit

Hit time: Time to access cache

Miss: Data is not in cache

Miss rate = 1-Hit rate

Miss penalty: Time to replace cache block + hit time

Hit time < Miss penalty

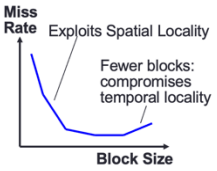
Average Access Time

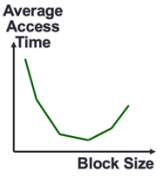
Hit rate x Hit Time + (1-Hit rate) x Miss penalty

Cache Block/Line: Unit of transfer between memory and cache

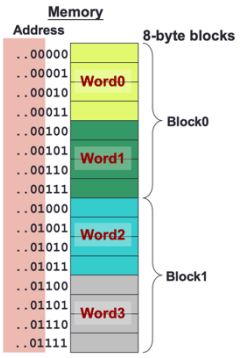
Trade-off (Larger block size)

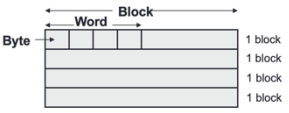
- Takes advantage of spatial locality
- Larger miss penalty: Takes longer time to fill up the block
- If block size is too big relative to cache size→Too few cache blocks→miss rate will go up





Direct Mapped Cache





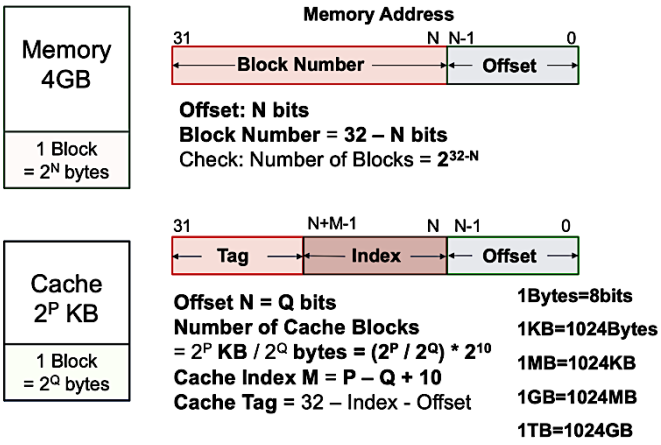
Observations:

1. 2^N-byte blocks are aligned at 2^N-byte boundaries
2. The addresses of words within a 2^N-byte block have identical (32-N) most significant bits (MSB).
3. Bits [31:N] ➔ the **block number**
4. Bits [N-1:0] ➔ the **byte offset** within a block

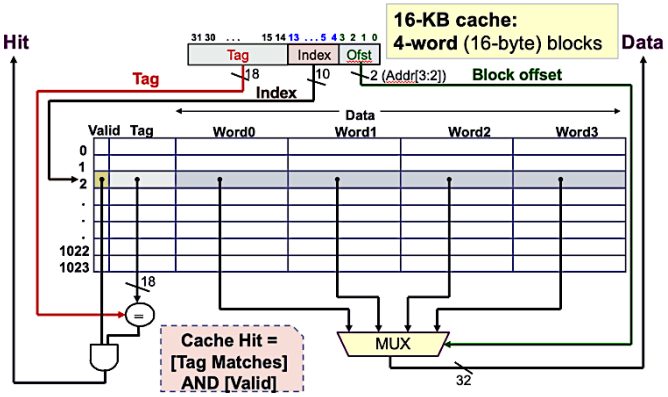
Mapping Function

Cache Index = (Block Number) mod (# of Cache Blocks)

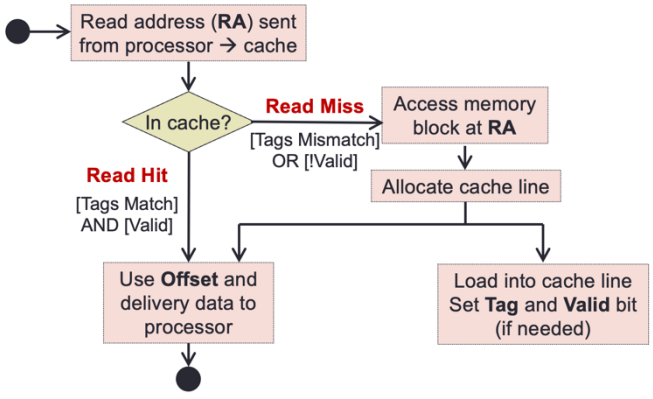
Tag=Block number / Number of Cache Blocks



Cache Circuitry



Read Data Example



Set Associative (SA) Cache

N-way Set Associative Cache⇒N cache blocks in a set

- Cache consists of a number of sets. Each set contains N cache blocks
- Each memory block maps to a **unique** cache set
- Within the set, a memory block can be placed in any of the N cache blocks in the set

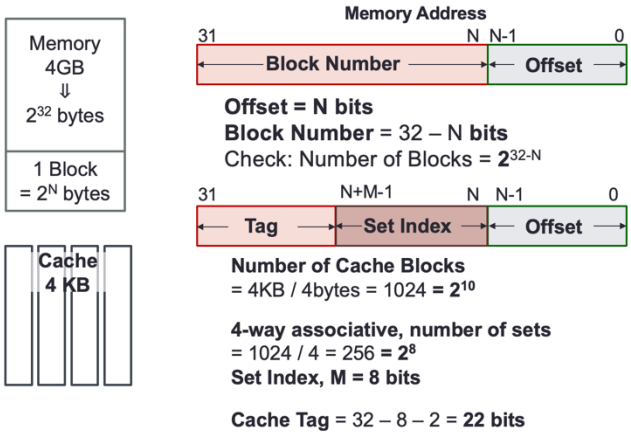
Map Function:

Cache Set Index = (Block Number) mod (# of Cache Sets)

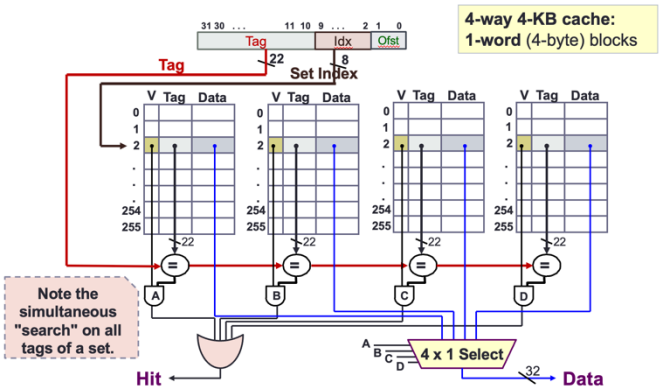
Cache Block size = 2^N bytes
Number of cache sets = 2^M
Offset = **N bits**
Set Index = **M bits**
Tag = **32 – (N + M) bits**

Observation: It is essentially unchanged from the direct-mapping formula

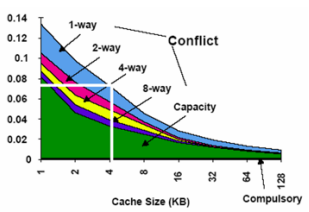
Example



Circuitry



Rule of Thumb: A **direct-mapped** cache of size N has about the **same miss rate** as a **2-way set associative cache** of size **N/2**



Fully Associative Cache: A memory block can be placed in any location in the cache.

- Memory block placement is no longer restricted by cache index or cache set index
- Can be placed in any location
- Need to search all cache blocks for memory access
- **No conflict miss (since data can go anywhere)**

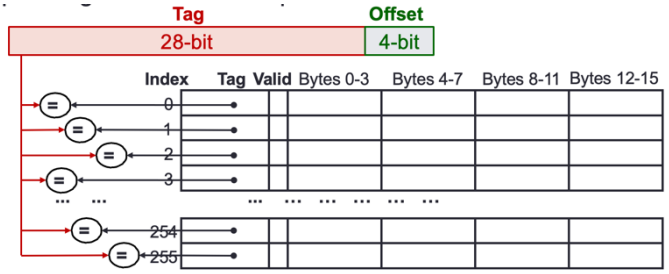
Performance:

1. Cold/compulsory miss remains the **same** irrespective of cache size/associativity.
2. For the same cache size, conflict miss goes down with increasing associativity.
4. For the same cache size, capacity miss remains the **same** irrespective of associativity.
5. Capacity miss **decreases** with increasing cache size.

Total Miss = Cold miss + Conflict miss + Capacity miss

Capacity miss (FA) = Total miss (FA) – Cold miss (FA), when Conflict Miss→0

Circuitry: 4KB cache size and 16-Byte block size. Compare tags and valid bit in parallel



Block Replacement Policy for SA/FA cache

- **Least Recently Used (LRU)**: replace the block that has not been accessed for the longest time [**disadvantages:** hard to keep track if there are many choices]
- FIFO (first in first out)
- RR (random replacement)
- LFU (least frequently used)

Cache Misses

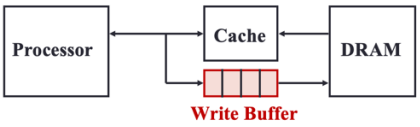
- **Compulsory misses:** On the first access to a block; the block must be brought into the cache. Also called **cold start misses** or **first reference misses**
- **Conflict misses:** Occur in the case of direct mapped cache or set associative cache, when several blocks are mapped to the same block/set. Also called **collision misses** or **interference misses**
- **Capacity misses:** Occur when blocks are discarded from cache as cache cannot contain all blocks needed (Occurs in Fully Associative Cache)

Write Policy

Problem: Cache and main memory are inconsistent. Modified data only in cache, not in memory!

Solution 1: Write-through cache

- Write data both to cache and to main memory
- Put a write buffer between cache and main memory



Solution 2: Write-back cache

- Only write to cache. Write to main memory only when cache block is replaced (evicted)
- Add an additional bit (**Dirty bit**) to each cache block
- Write operation will change dirty bit to **1**, Only cache block is updated, **no** write to memory
- When a cache block is replaced: Only write back to memory if dirty bit is 1

Handling Cache Misses

Read Miss: Data loaded into cache and then load from there to register

Write Miss:

(1) Write allocate

- Load the **complete block** into cache
- Change only the required word in cache
- Write to main memory depends on write policy

(2) Write around

- Do not load the block to cache
- Write **directly** to main memory only

Cache Framework

Block Placement: Where can a block be placed in cache?

Direct Mapped:	N-way Set-Associative:	Fully Associative:
• Only one block defined by index	• Any one of the **N** blocks within the set defined by index	• Any cache block

Block Identification: How is a block found if it is in the cache?

Direct Mapped:	N-way Set Associative:	Fully Associative:
• Tag match with only one block	• Tag match for all the blocks within the set	• Tag match for all the blocks within the cache

Block Replacement: Which block should be replaced on a cache miss?

Direct Mapped:	N-way Set-Associative:	Fully Associative:
• No Choice	• Based on replacement policy	• Based on replacement policy

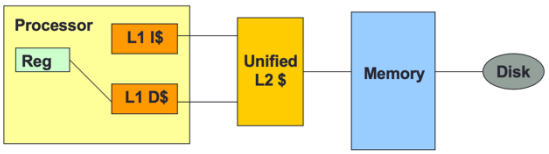
Write Strategy: What happens on a write?

Write Policy: Write-through vs write-back

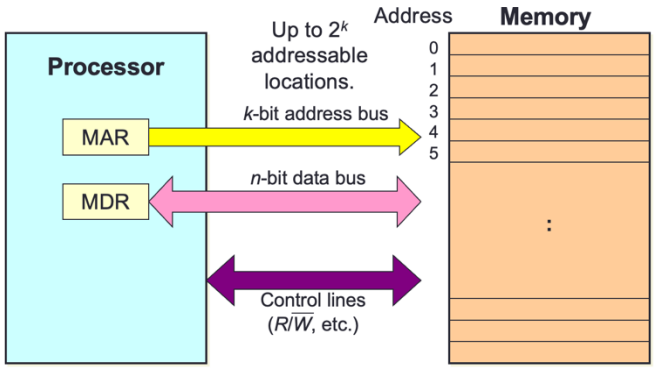
Write Miss Policy: Write allocate vs write no allocate

Exploration: Multilevel Cache

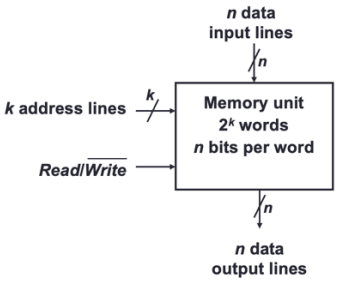
- Options:
  - Separate data and instruction caches, or a unified cache
- Sample sizes:
  - **L1**: 32KB, 32-byte block, 4-way set associative
  - **L2**: 256KB, 128-byte block, 8-way associative
  - **L3**: 4MB, 256-byte block, Direct mapped



Memory



Memory Unit



- Stores **binary information** in groups of bits called *words*.
- The data consists of *n* lines (for *n*-bit words).
  - Data input lines provide the information to be stored (*written*) into the memory
  - Data output lines carry the information out (*read*) from the memory.
- The address consists of *k* lines which specify which word (among the 2^k words available) to be selected for reading or writing.
- The control lines *Read* and *Write* (usually combined into a single control line *Read/Write*) specifies the direction of transfer of the data.

Write operation: Transfers the address of the desired word to the address lines.

- Transfers the data bits (the word) to be stored in memory to the data input lines.
- Activates the ***Write* control line** (set *Read/Write* to 0).

Read operation: Transfers the address of the desired word to the address lines.

- Activates the ***Read* control line** (set *Read/Write* to 1).

Memory Enable	*Read/Write*	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

SRAM and DRAM

	SRAM	DRAM
Transistor(s) per memory cell	6	1
access latency	Fast (0.5-5ns)	Slow (50-70ns)
Memory Cell	use flip-flops as memory cells.	use capacitor charges to represent data. have to be constantly refreshed
Speed:	Hard disk < DRAM < SRAM < Register	
Size:	Hard disk > DRAM > SRAM > Register	

A single memory cell of the static RAM