中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Computer Architecture

计 算 机 体 系 结 构

第12讲：Memory（2）

张献伟

xianweiz.github.io

DCS3013, 11/14/2022

Quiz Questions

For remote attendees, plz email to zhangxw79@mail.sysu.edu.cn (ddl: 14:40).

- Q1: explain SIMT.

  Single inst multi threads. GPU is a typical SIMT arch (e.g., warp).

- Q2: list 3 differences between CPU and GPU.

  O3 vs IO, few cores vs massive, latency vs. throughput, …

- Q3: for GPU shared and local mem, which is faster? Why?

  Shared. SMEM is part of L1, inside SM; LMEM is global memory.

- Q4: for *kernel_A*, calculate thread occupancy on the utilized SM of A100 GPU.

  2 SMs will be used. 512 / 2048 = 25%.

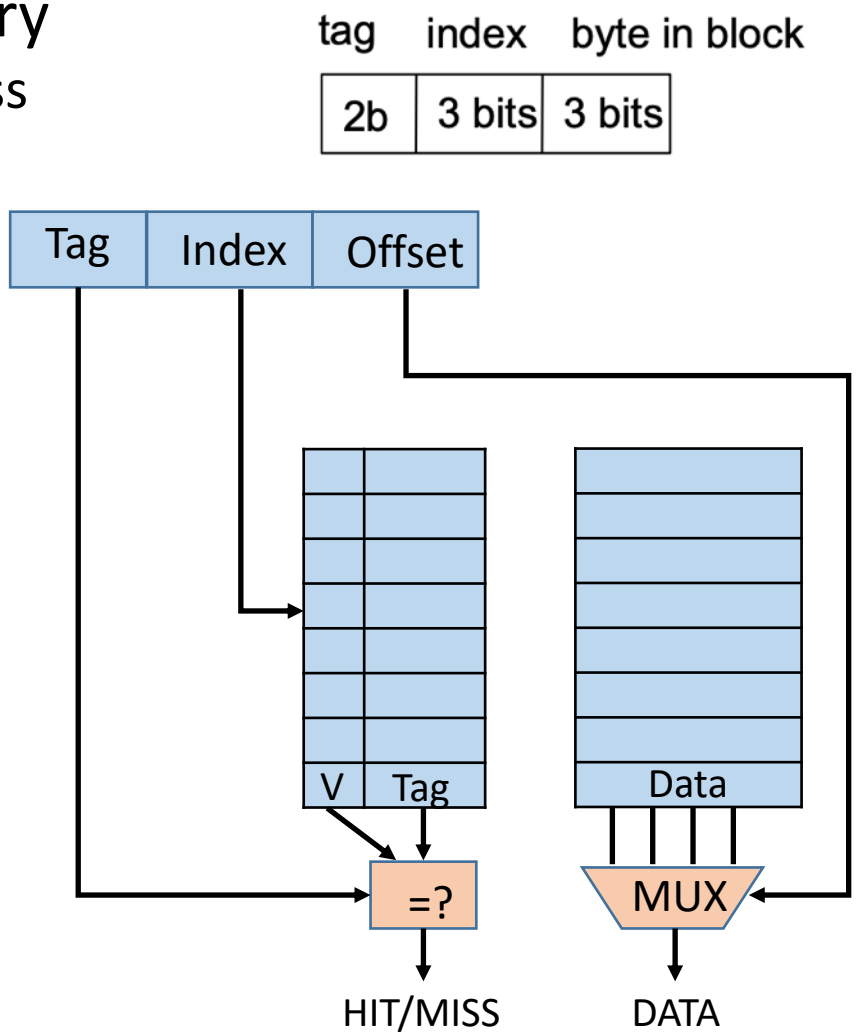
```
kernel_A<<<2, 512, 0>>>()
kernel_B<<< 8, 512, 0, 2>>>()
kernel_C<<< 8, 512, 0, 3>>>()
```

- Q5: can *kernel_A*/*_B*/*_C* fully overlapped? Why?

  No. *kernel_A* uses default stream 0, synchronous with all streams.

Cache Basics

- Assume byte-addressable memory
  - Capacity: 256 bytes → 8-bit address
  - Block: 8 bytes → 3-bit offset
  - #blocks: 32 (256/8)
- Assume cache
  - Capacity: 64 bytes → 3-bit index
    - Holding 8 blocks (64/8)
- What is a tag store?
  - Tag
  - Metadata
    - Valid bit
    - Replacement policy bits
    - Dirty bit
    - ECC

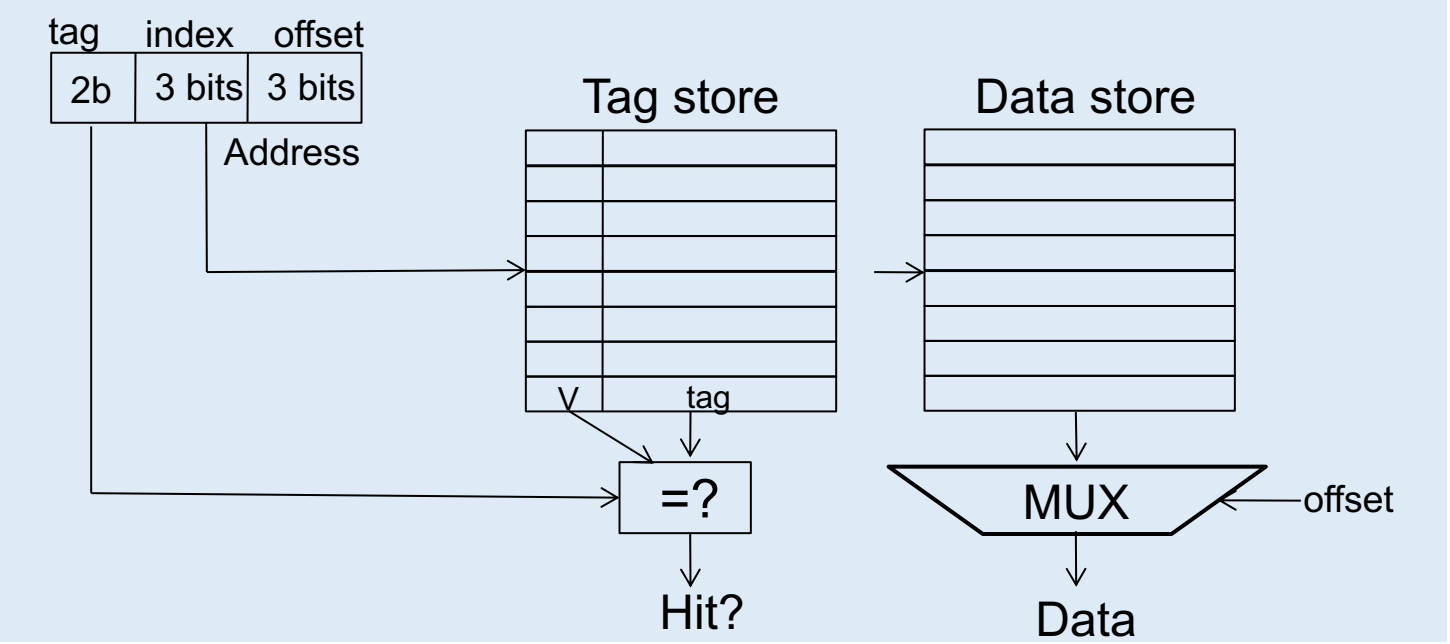


MUX: multiplexer (数据选择器)

Direct Mapped[直接映射]

- For each item (block) of data in memory, there is **exactly one** location in the cache where it might be
- Two blocks in memory that map to the same index in the cache cannot be present in the cache at the same time
  - Addresses A/B have the same index bits but different tag bits
  - A, B, A, B, A, B, A, B, … → all misses

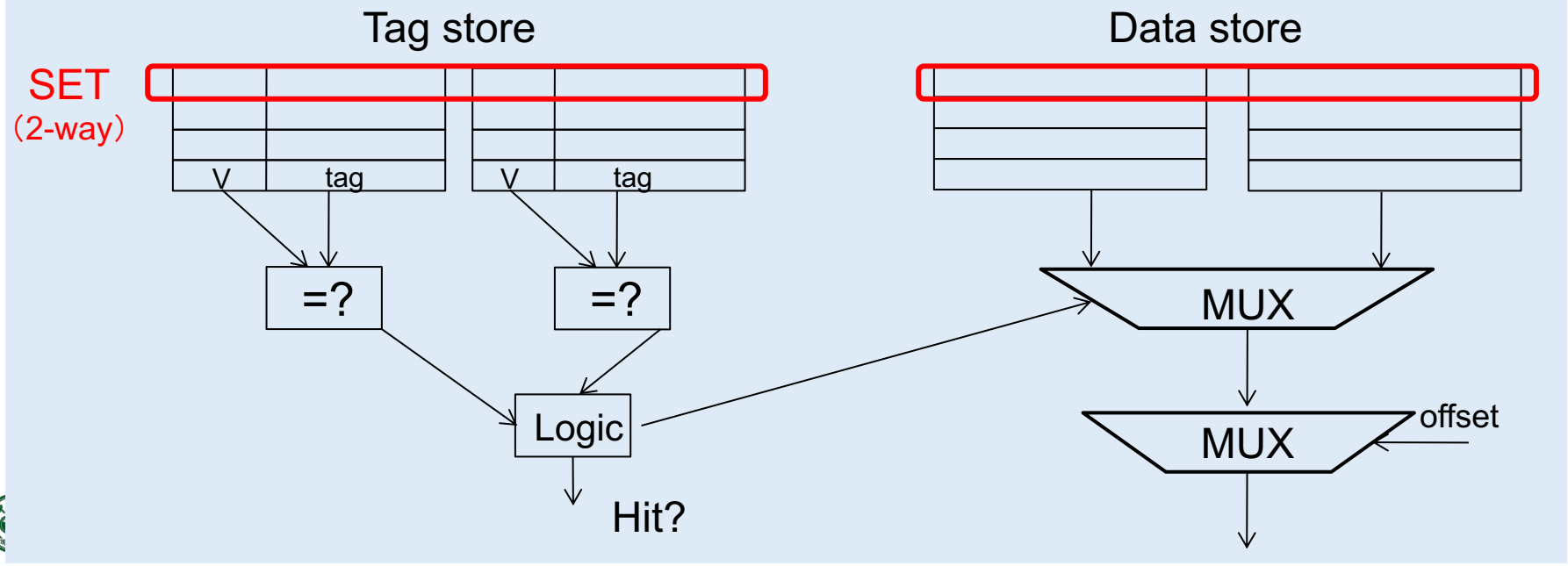
tag	index	offset
2b	**3b**	3b



Set-Associative[组相连]

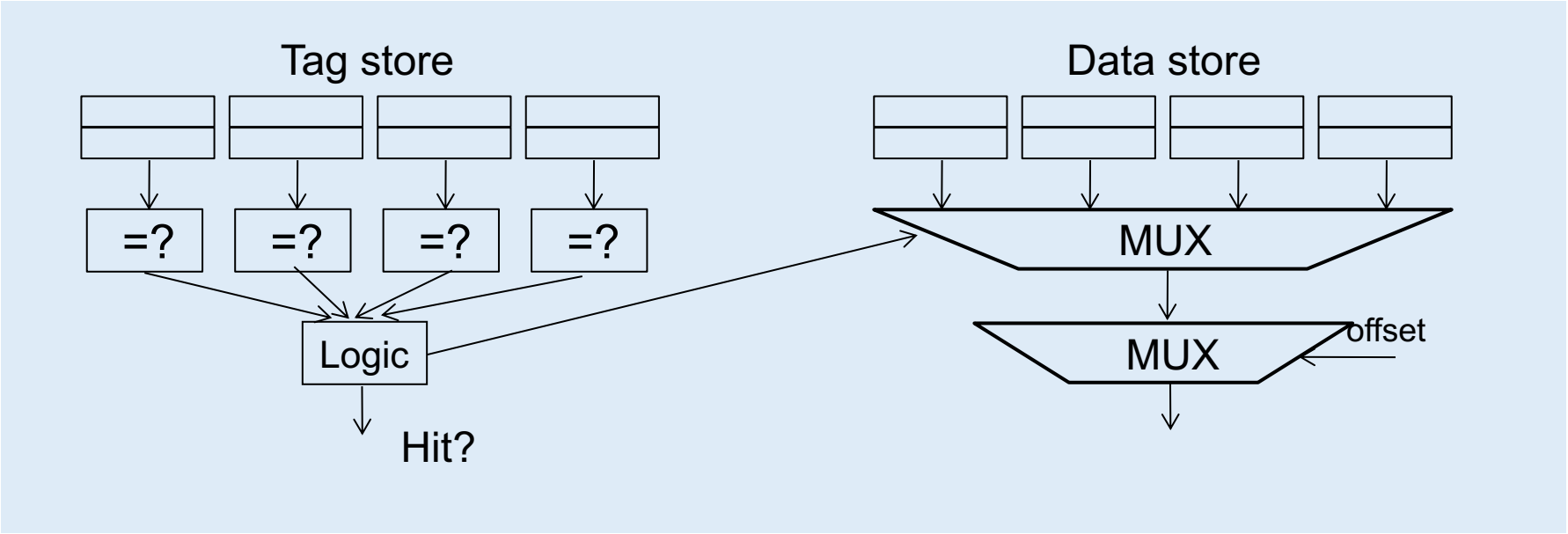
- For the direct mapped
  - Addresses *0* and *8* always conflict in direct mapped cache
  - Instead of having one column of *8* blocks, have *2* columns of *4*
- Key idea: associative memory within the set
  - + Accommodates conflicts better (fewer conflict misses)
  - -- More complex, slower access, larger tag store

tag	index	offset
3b	2b	3b



Higher Associativity[高相连度]

- 2-way → 4-way
  - + Likelihood of conflict misses even lower
  - -- More tag comparators and wider data mux
  - -- larger tags

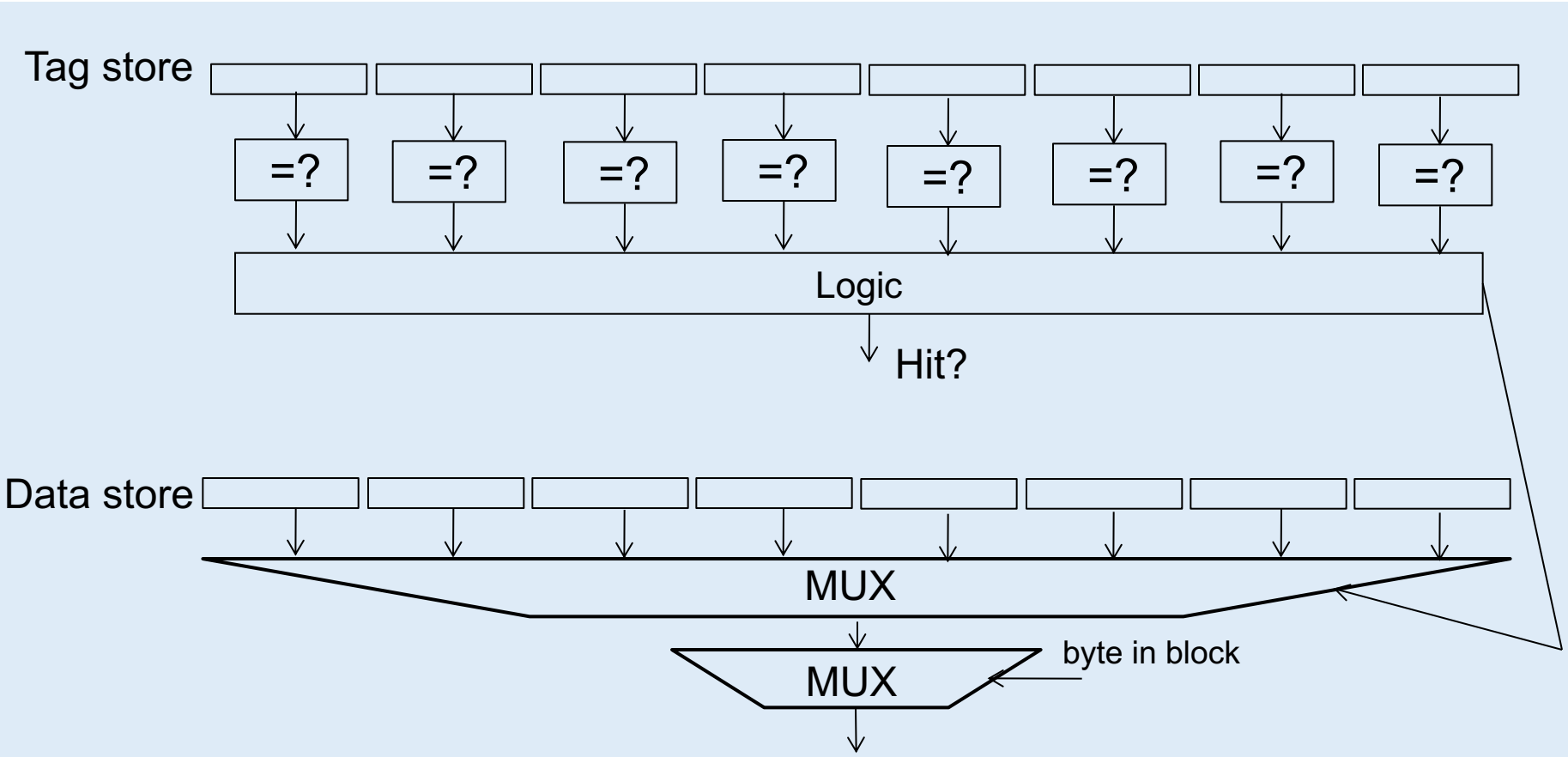


tag	index	offset
4b	1b	3b

Fully-Associative[全相联]

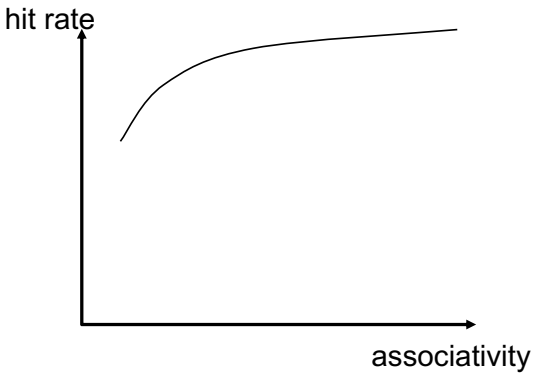
- A block can map *anywhere* in the cache
  - Most efficient use of space
  - Least efficient to check

tag	index	offset
5b	0b	3b



Issues of Set-Associative[一些问题]

- Degree of associativity[相连度]: how many blocks can map to the same index (or set)?
- Higher associativity
  - ++ Higher hit rate
  - -- Slower cache access time (hit latency and data access latency)
  - -- More expensive hardware (more comparators)
- Diminishing returns from higher associativity



- Block replacement[块替换]
  - Not an issue for Direct-Mapped
  - Set Associative or Fully Associative
    - Random, LRU (Least Recently Used), FIFO

Handling Writes[写]

- When do we write the modified data in a cache to next level?
  - Write back: when the block is evicted
  - Write through: at the time the write happens
- Write-back[写回]
  - + Can consolidate multiple writes to the same block before eviction
    - Potentially saves bandwidth between cache levels + saves energy
  - -- Need a bit in the tag store indicating the block is "dirty/modified"
- Write-through[写通]
  - + Simpler
  - + All levels are up to date
    - Consistency: simpler cache coherence because no need to check lower-level caches
  - -- More bandwidth intensive; no coalescing of writes

Handling Writes (cont.)

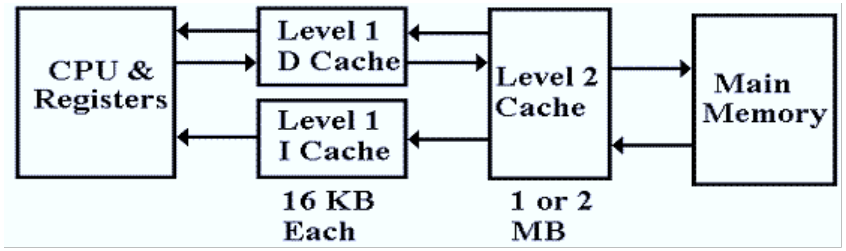
- Do we allocate a cache block on a write miss?
  - Allocate on write miss: Yes
  - No-allocate on write miss: No
- Allocate on write miss[写分配]
  - + Can consolidate writes instead of writing each of them individually to next level
  - + Simpler because write misses can be treated the same way as read misses
  - -- Requires (?) transfer of the whole cache block
- No-allocate[写不分配]
  - + Conserves cache space if locality of writes is low (potentially better cache hit rate)

Handling Writes (cont.)

- Write-through with write-allocate
  - On hits, writes to both cache and memory
  - On misses, updates the block in memory and brings it to cache
- Write-through with write-no-allocate
  - Ditto
  - On misses, updates the block in memory but not brings it to cache
- Write-back with write-allocate
  - On hits, writes to cache, sets “dirty” bit for the block (memory is not updated)
  - On misses, updates the block in memory and brings the block to cache
- Write-back with write-no-allocate
  - Ditto
  - On misses, updates the block in memory but not brings the block to cache

Instruction vs. Data Caches

- Separate or Unified?



- Unified[一体]
  - + Dynamic sharing of cache space: no overprovisioning that might happen with static partitioning (i.e., split I and D caches)
  - -- Instructions and data can thrash each other (i.e., no guaranteed space for either)
  - -- I and D are accessed in different places in the pipeline. Where do we place the unified cache for fast access?
- First level caches are almost always split
  - Mainly for the last reason above
- Second and higher levels are almost always unified

Evaluation Metrics[评价指标]

- Cache hit ratio
  - (# hits) / (# hits + # misses) = (# hits) / (# accesses)
- Average memory access time (AMAT)
  - (hit-ratio * hit-latency) + (miss-ratio * miss-latency)
- Cache hit rate: # of misses per kilo instructions (MPKI)

Example: Assume that
Processor speed = 1 GHz (1 n.sec. clock cycle)
Cache access time = 1 clock cycle
Miss penalty = 100 n.sec (100 clock cycles)
I-cache miss ratio = 1%, and D-cache miss ratio = 3%
74% of memory references are for instructions and 26% for data

Effective cache miss ratio = 0.01 * 0.74 + 0.03 * 0.26 = 0.0152
Av. (effective) memory access time = 1 + 0.0152 * 100 = 2.52 cycles = 2.52 n.sec