Computer Organization
計算機組織

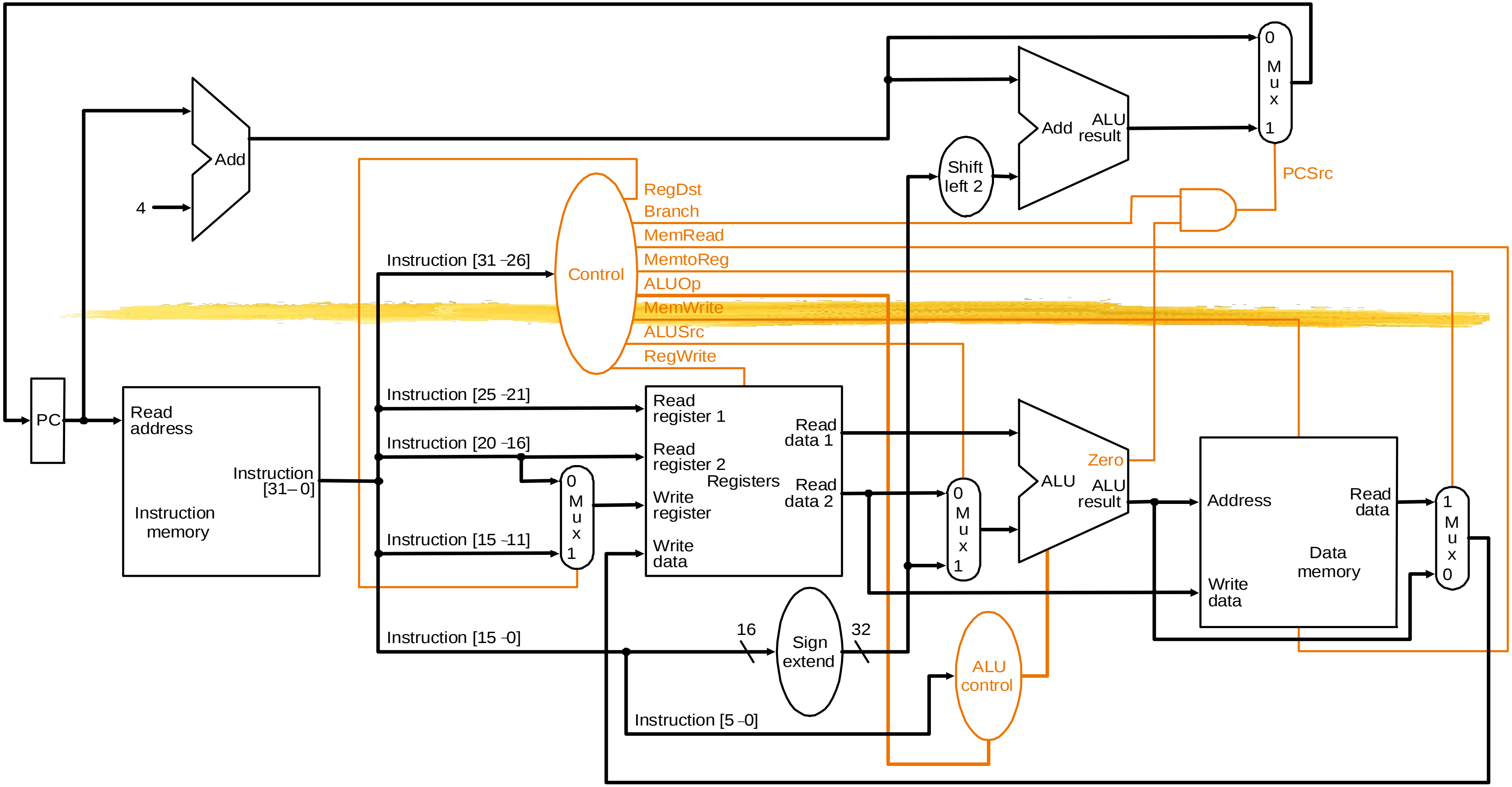
Cache

國立成功大學資訊工程學系
105 年度第二學期

What to Learn

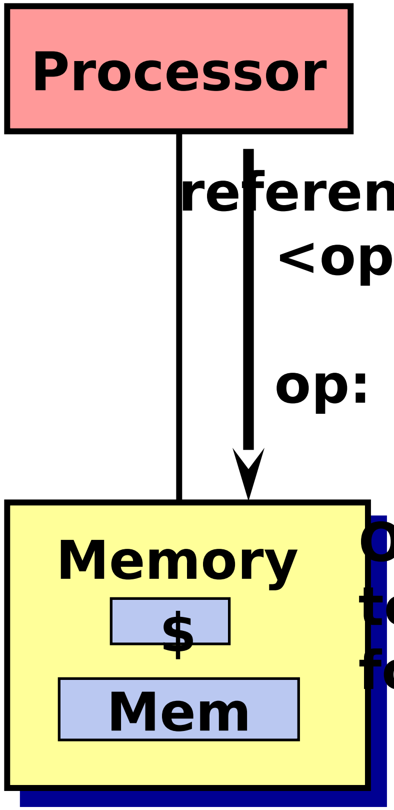
- Memory hierarchy
- Caching
- Cache organization
  - Direct mapped
  - Fully associative
  - Set associative

Single-Cycle MIPS



Memory References Workload

Workload or Benchmark programs



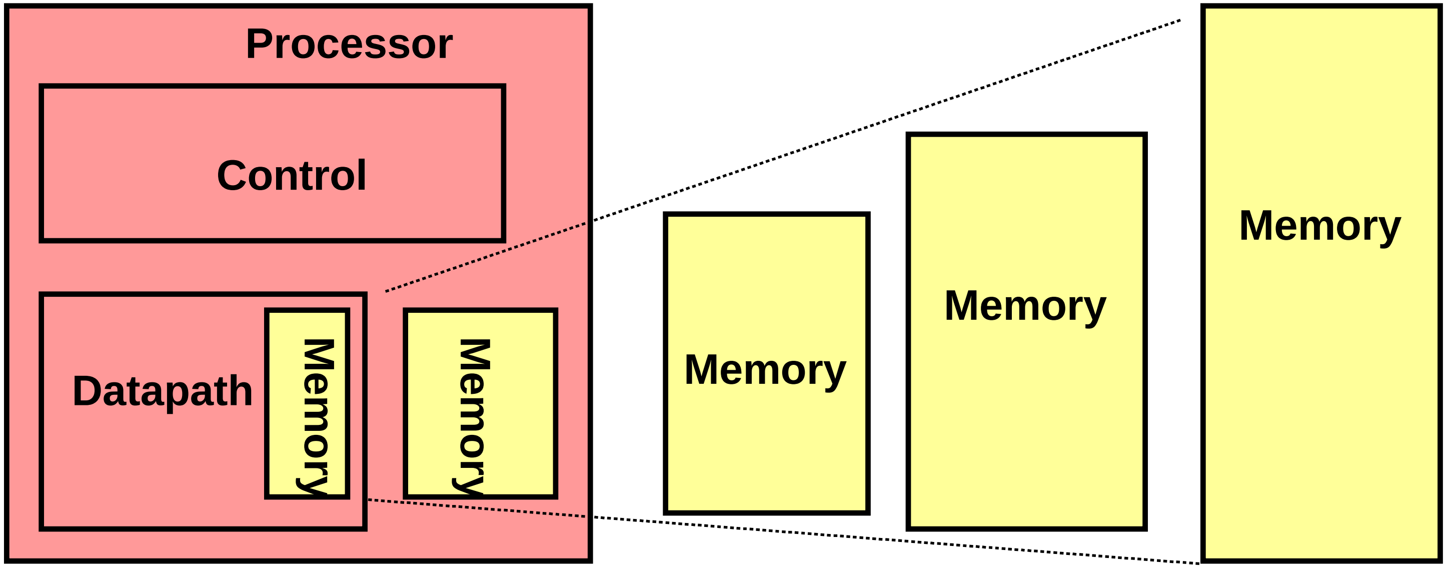
reference stream

<op,addr>, <op,addr>,<op,addr>,<op,addr>, . . .

op: instruction fetch, read (`lw`), and write (`sw`)

Optimize the memory system organization to minimize the average memory access time for typical workloads

Memory Hierarchy: Concept

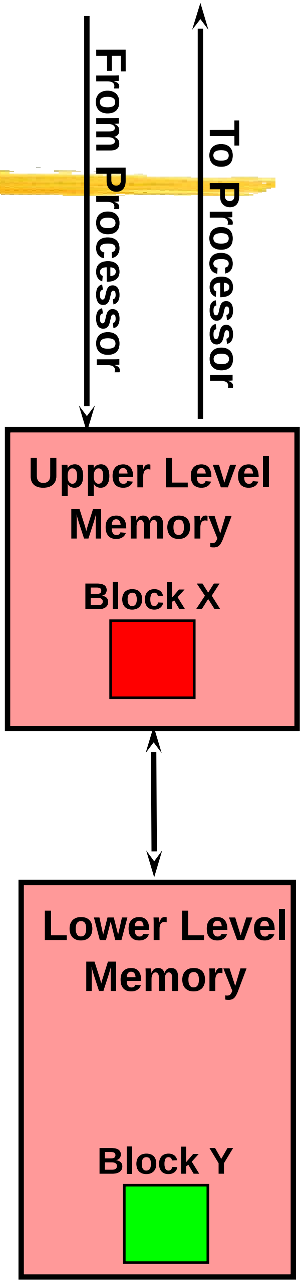


Speed: Fastest
Size: Smallest
Cost: Highest

Slowest
Biggest
Lowest

Memory Hierarchy: Terminology

- Hit: data appears in upper level (Block X)
  - Hit rate (ratio): fraction of memory access found in the upper level
  - Hit time: time to access the upper level
    - Time to determine hit/miss + RAM access time
- Miss: data needs to be retrieved from a block in the lower level (Block Y)
  - Miss rate = 1 - (Hit rate)
  - Miss penalty: time to place (replace) a block (with a block) in the upper level + time to deliver the block to the processor
- Hit Time << Miss Penalty
- For simplicity:
  - Upper level memory: cache in processor
  - Lower level memory: main memory



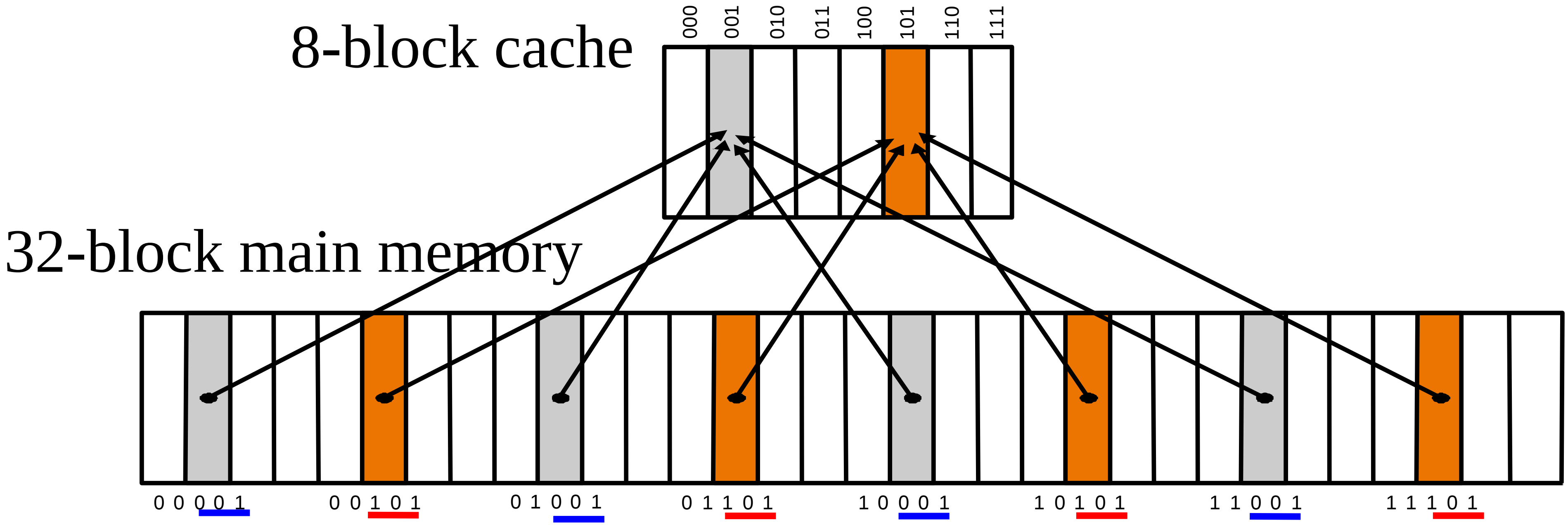
Why Hierarchy Works?

- Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time
  - 90/10 rule (power law): 10% of code executes 90% of time

- Two types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
    - E.g., some counter variable in a loop
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon
    - E.g., nearby array data A[i] and A[i+1]

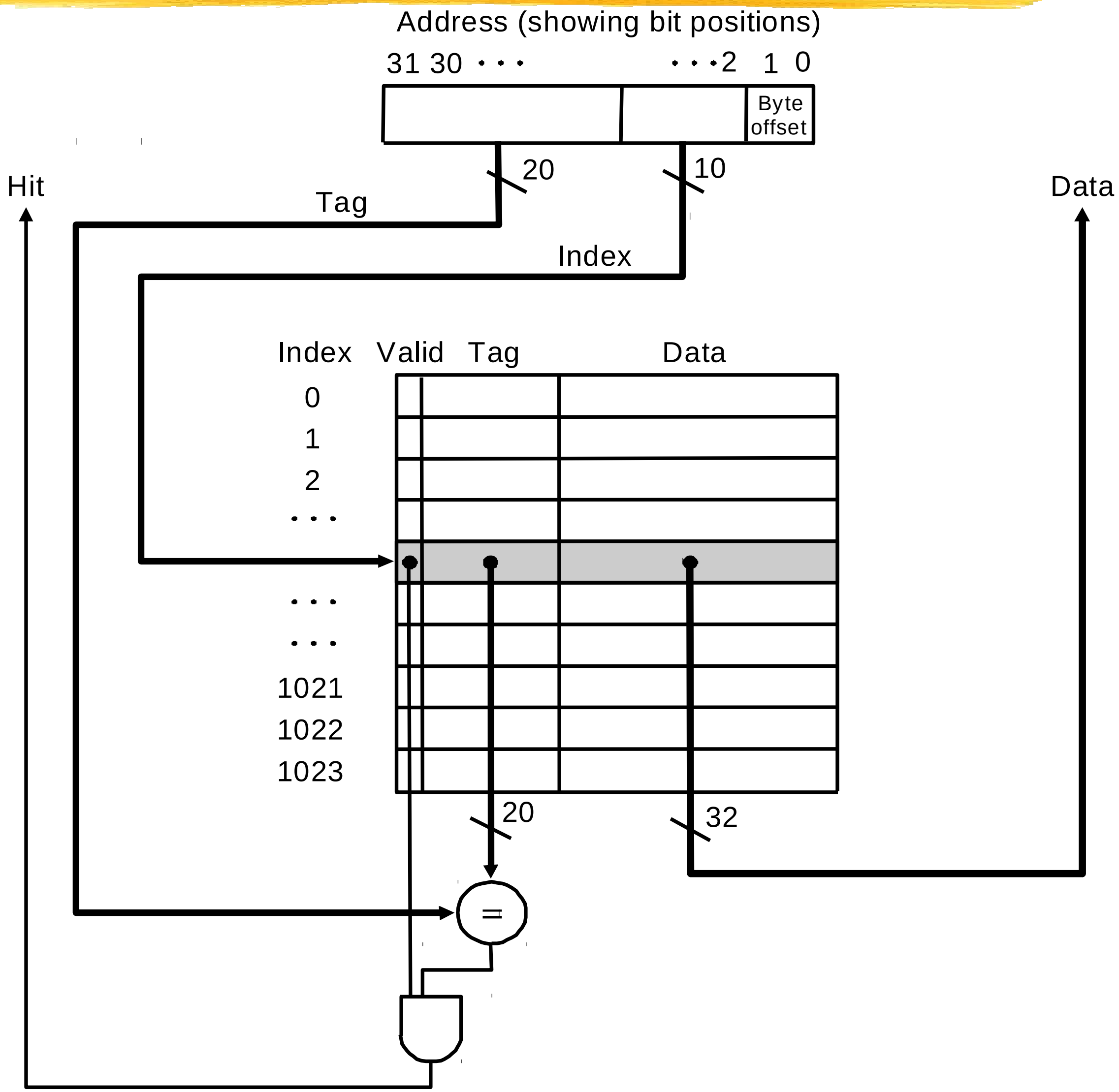
Basics of Cache

- **Direct-mapped cache (e.g., MIPS R2000)**
  - **Address mapping: modulo number of blocks**



Direct-Mapped Cache

- **Tag and valid bit**
- **Example**
  - **1K words, 1-word block:**
  - **Cache index: lower 10 bits**
  - **Cache tag: upper 20 bits**
  - **Valid bit (when start up, valid is 0)**



Example

(0) Initial state:

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

(1) Address referred 10110xx (*miss*):

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem(10110xx)**
111	N		

(2) Address referred 11010xx (*miss*):

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem(11010xx)**
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem(10110xx)**
111	N		

(3) Address referred 10110xx (*hit*):

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem(11010xx)**
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem(10110xx)**
111	N		

Hit

(4) Address referred 10010xx (*miss*):

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**10**	**Mem(10010xx)**
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem(10110xx)**

Miss and then replaced

Example Problem (1/2)

- How many total bits are required for a direct-mapped cache with 128 KB of data and 1-word block size, assuming a 32-bit address?

  - Cache data = 128 KB = 2^{17} bytes = 2^{15} words = 2^{15} blocks

  - Cache entry size = block data bits + tag bits + valid bit = 32 + (32 – 15 – 2) + 1 = 48 bits

  - Therefore, cache size = $2^{15} \times 48$ bits = $2^{15} \times (1.5 \times 32)$ bits = 1.5×2^{20} bits = 1.5 Mbits

Example Problem (2/2)

- Consider a direct-mapped cache with 64 blocks and a block size of 16 bytes. What block number does byte address 1200 map to?

  - As block size = 16 bytes:

byte address 1200 ⇒ block address $\lfloor 1200/16 \rfloor = 75$

  - As cache size = 64 blocks:

block address 75 ⇒ cache block (75 *mod* 64) = 11

More on Hits

- Read hits: this is what we want!

- Write hits: keep cache/memory consistent?
  - Write-through: write to cache and memory at same time => but memory is very slow!
  - Write-back: write to cache only (write to memory when that block is being replaced)
    - Need a dirty bit for each block

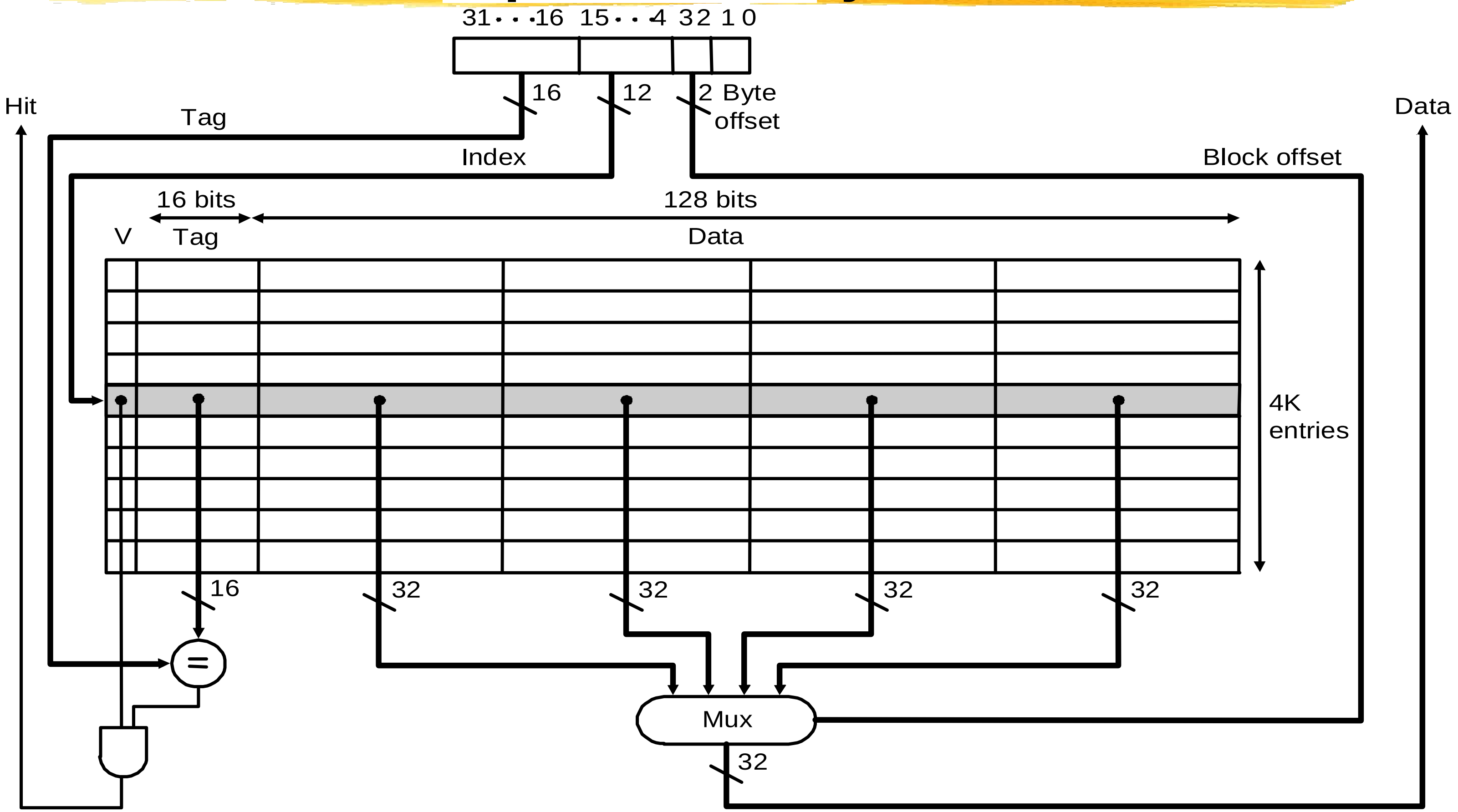
More on Misses

- Read misses
  - Stall CPU, freeze register contents, fetch block from memory, deliver to cache, restart
  - "Block replacement" may occur
    - Placement: ?
    - Replacement: ?

- Write misses
  - Write-allocated: read block into cache, write the word
    - Block replacement may occur
  - Write-non-allocate: write directly into memory

How to Exploit Spatial Locality?

- **Increase block size for spatial locality**

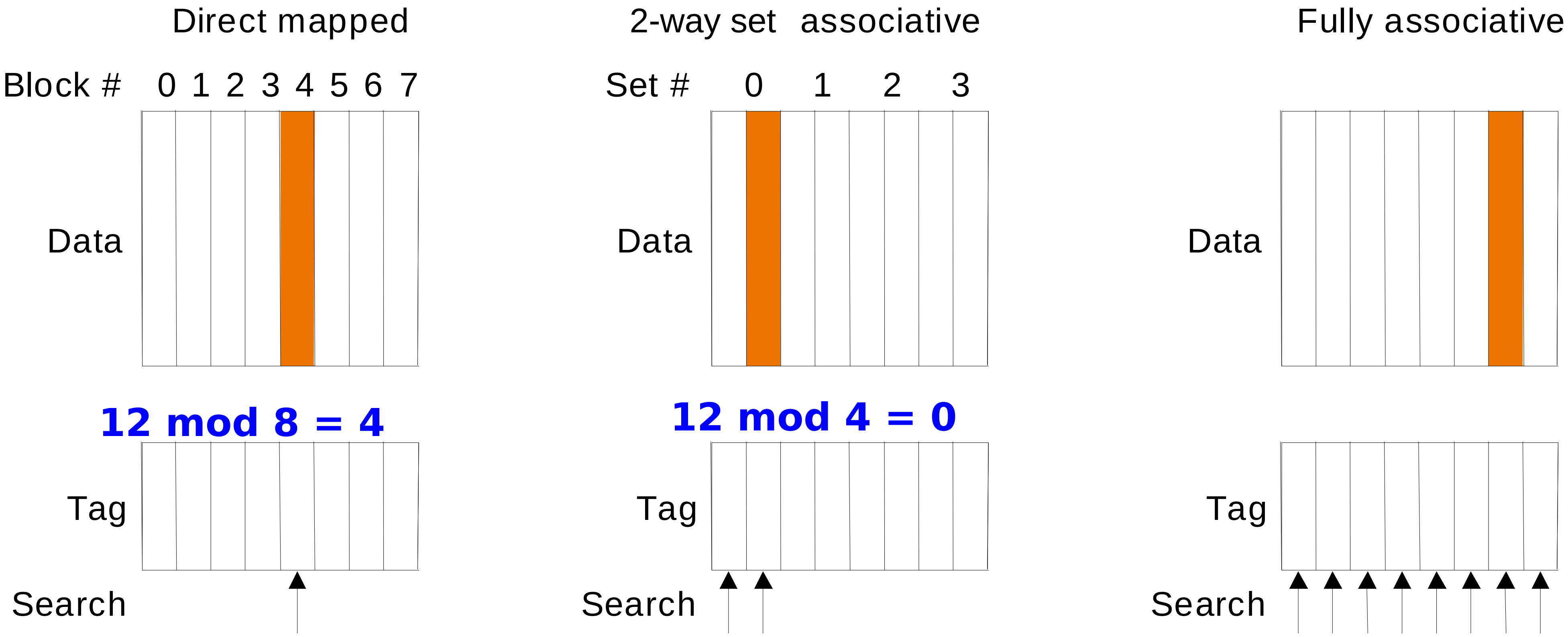


Cache with 4K 4-word blocks: *byte offset* (least 2 significant bits) is ignored, next 2 bits are *block offset*, and the next 12 bits are used to index into cache

Possible Cache Architectures

- **Direct mapped (have discussed)**
- **Fully associative: each memory block can locate anywhere in cache**
  - **all cache entries are searched (in parallel) to locate block**
- **Set associative: each memory block can place in a unique set of cache locations (any location in the set), and if the set is of size *n* it is *n*-way set-associative**
  - **cache set address = memory block address *mod* number of sets in cache**
  - **all cache entries in the corresponding set are searched (in parallel) to locate block**
  - **E.g., in a 2-way 1K-word set associate cache, there are 512 sets, each having 2 words**

Illustration



Location of a memory block with address 12 in a cache with 8 blocks with different degrees of associativity

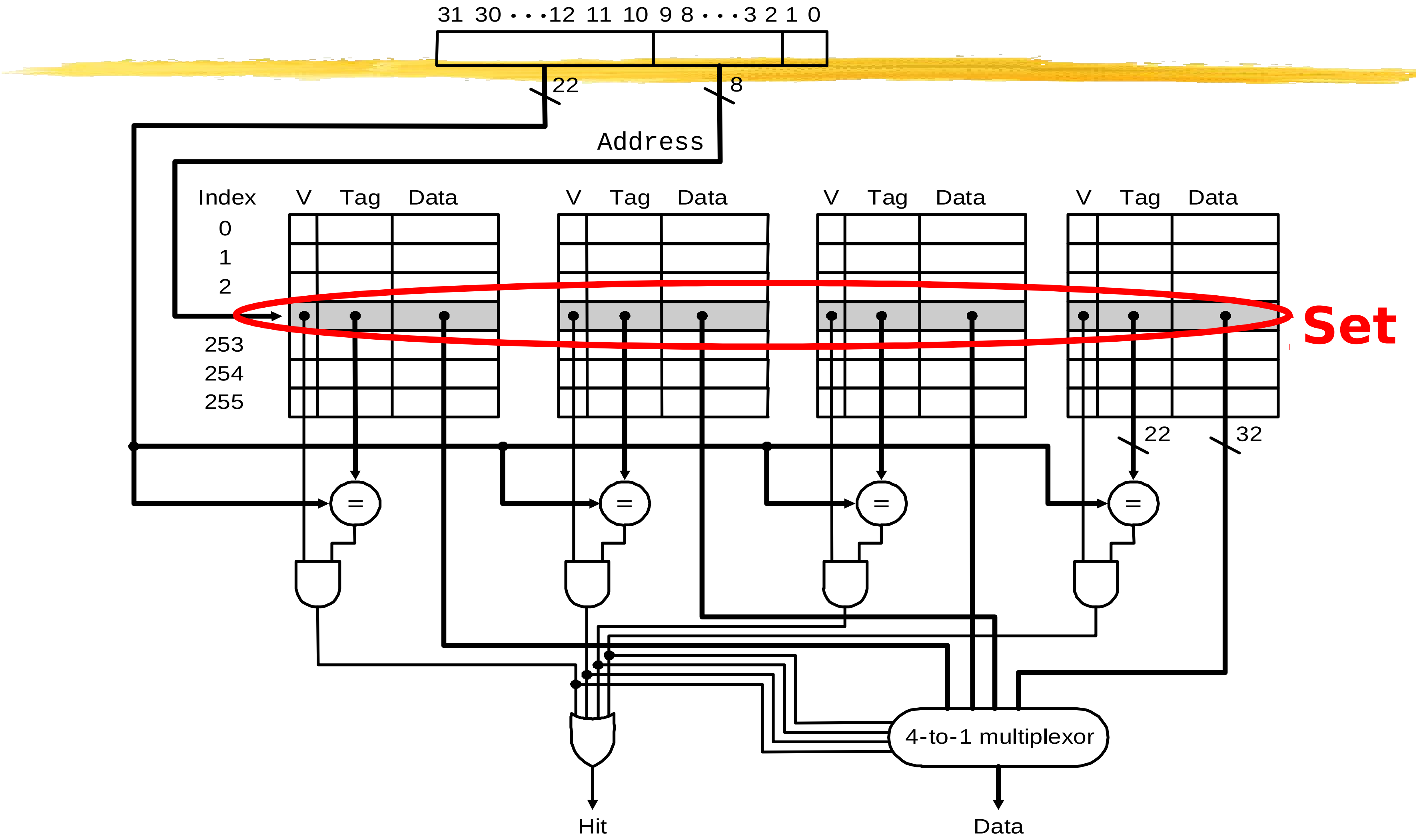
More on Associativity

Assume *N* cache blocks

- *K*-way set associativity = *K* directed mapped
  - No. of sets = *N* / *K*

- Fully associativity = *N* directed mapped
  - No. of sets = *N* / *N* = 1

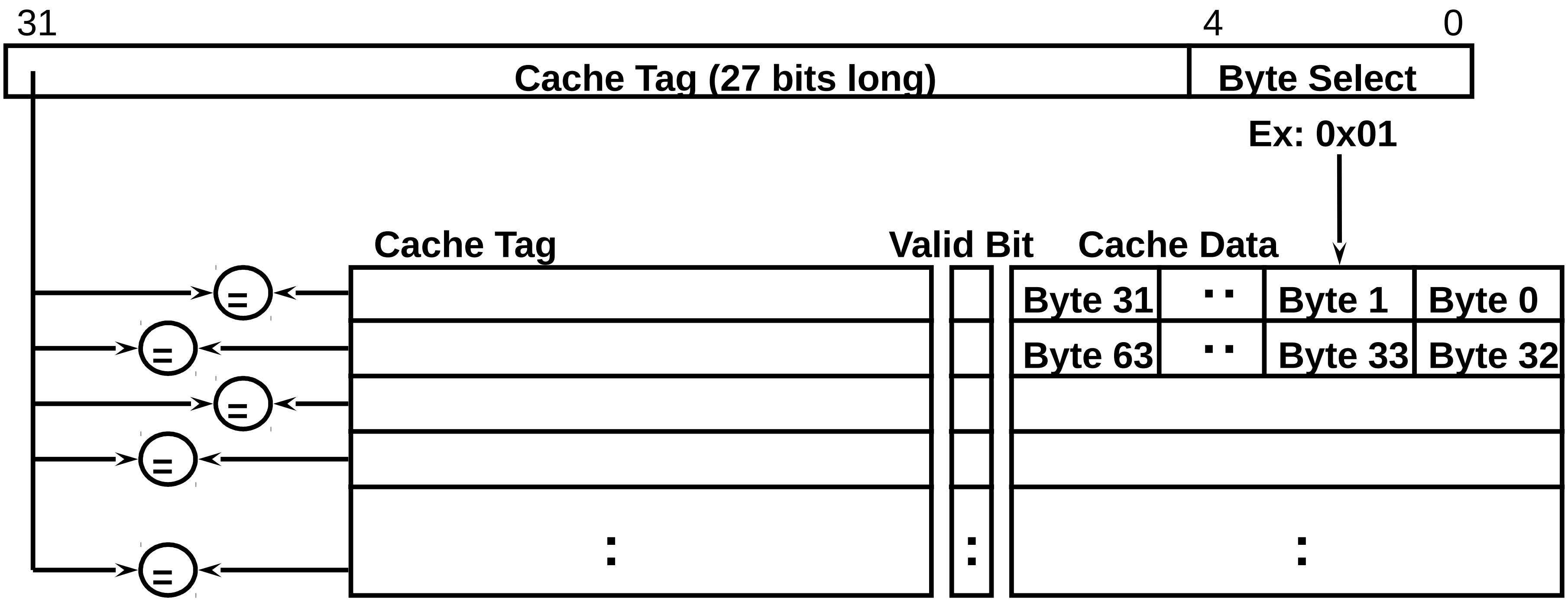
Implementing Set-Associative Cache



4-way set-associative cache with 4 comparators and one 4-to-1 multiplexor: size of cache is 1K blocks = 256 sets * 4-block set size

Implementing Fully Associate Cache

- Compare cache tags of all cache entries in parallel
- Ex.: Block Size = 8 words, N 27-bit comparators



Why Set Associate Cache?

- Directed mapped may introduce too many conflict misses
  - Conflict miss: >=1 memory blocks contend a cache line
- Fully associative is too expensive
  - Minimize the number of conflict misses
- Set associative intends to come up with a balance

Example Problem

- Find the number of misses for a cache with four 1-word blocks given the following sequence of word addresses of memory block accesses: 0, 8, 0, 6 and 8, for each of the following cache configurations
  1. direct mapped
  2. 2-way set associative (use LRU replacement policy)
  3. fully associative

- LRU replacement
  - If replacement, replace with a least recently used block
  - In a 2-way set associative cache LRU replacement can be implemented with one bit at each set whose value indicates the mostly recently referenced block

Direct Mapped

Block address translation in direct-mapped cache

Block address	Cache block
0	0 (= 0 *mod* 4)
6	2 (= 6 *mod* 4)
8	0 (= 8 *mod* 4)

Cache contents after each reference – red indicates new entry added

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

5 misses

Two-way Set Associate

Block address translation in a two-way set-associative cache

Block address	Cache set
0	0 (= 0 *mod* 2)
6	0 (= 6 *mod* 2)
8	0 (= 8 *mod* 2)

Cache contents after each reference – red indicates new entry added

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

4 misses

Fully Associate

Cache contents after each reference – red indicates new entry added

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	**Block 1**	**Block 2**	**Block 3**
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

3 misses