COMPUTER ORGANIZATION (IS F242)

LECT 38_39: CACHE MEMORY

Cache Design

- Cache access
  - Look through, Look aside
- Block Placement
  - Direct, Fully Associative, Set-Associative
- Block Identification
  - TAG, INDEX, OFFSET
- Block Replacement
  - LRU, PLRU, LFU, OPT, FIFO, RANDOM
- Write Policies
  - Write Through, Write Back, Write Buffer
- Coherency
  - Snooping, MESI

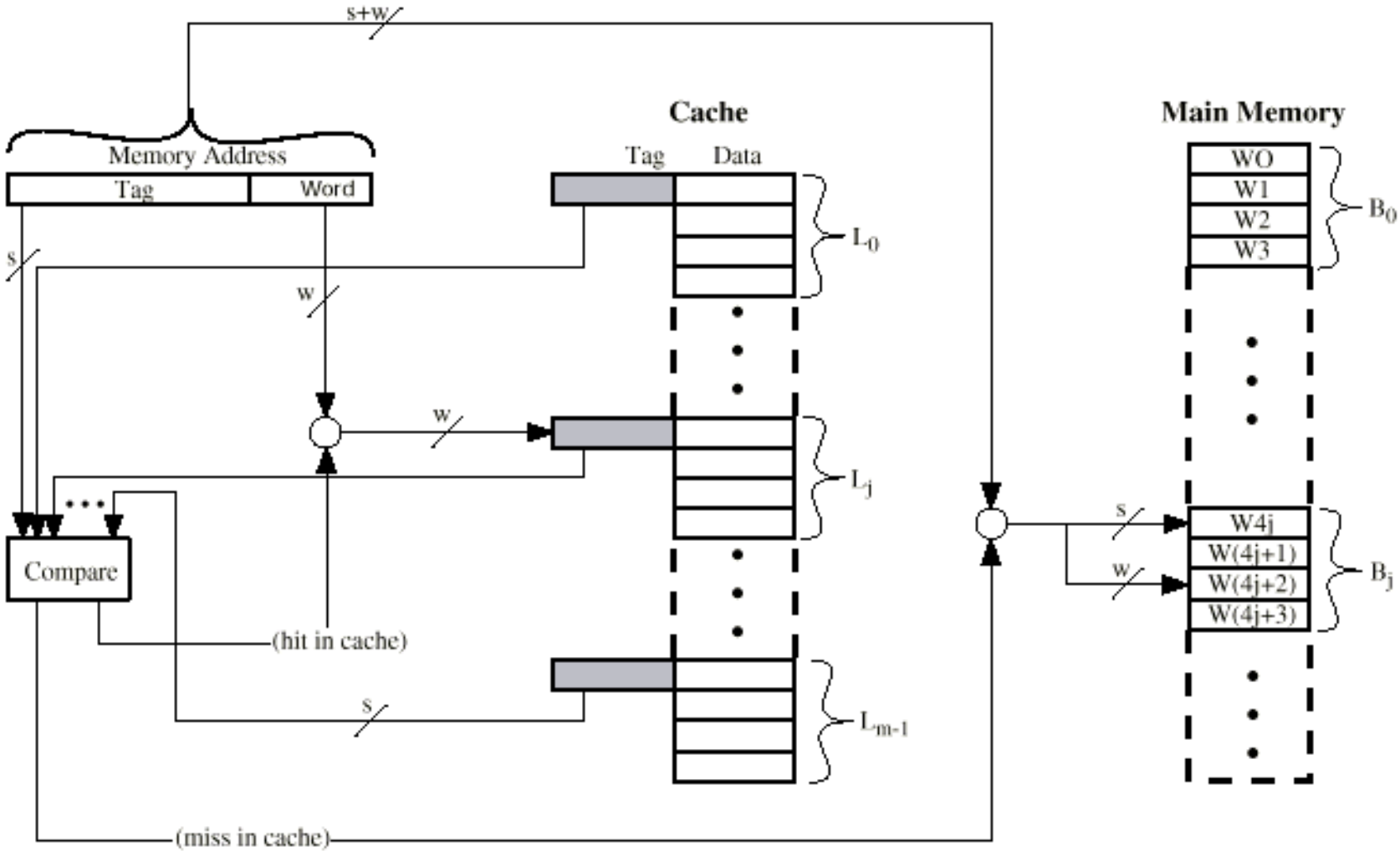
Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses (conflict misses) are very high

Associative Mapping

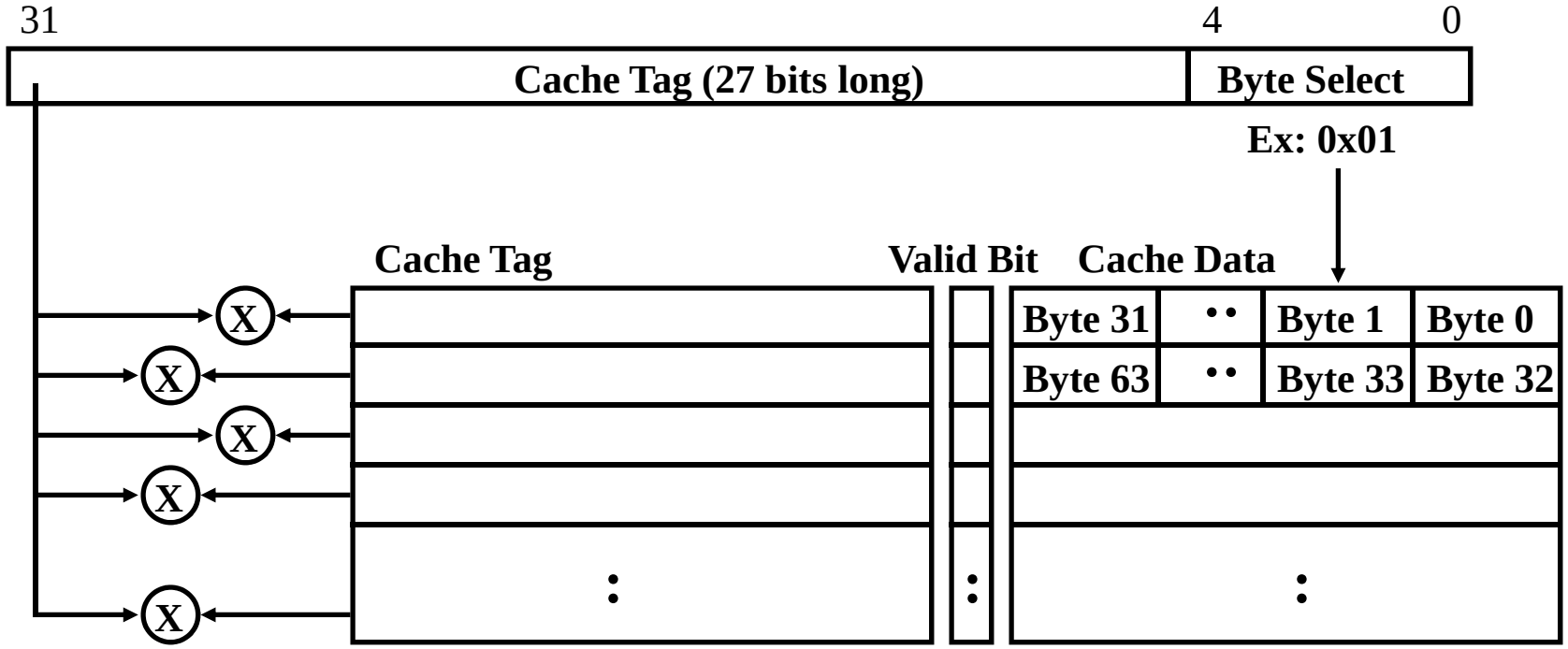
- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Fully Associative Cache Organization



Fully Associative Cache

- Fully Associative Cache (e.g., 32 B block)
  - compare tags in parallel



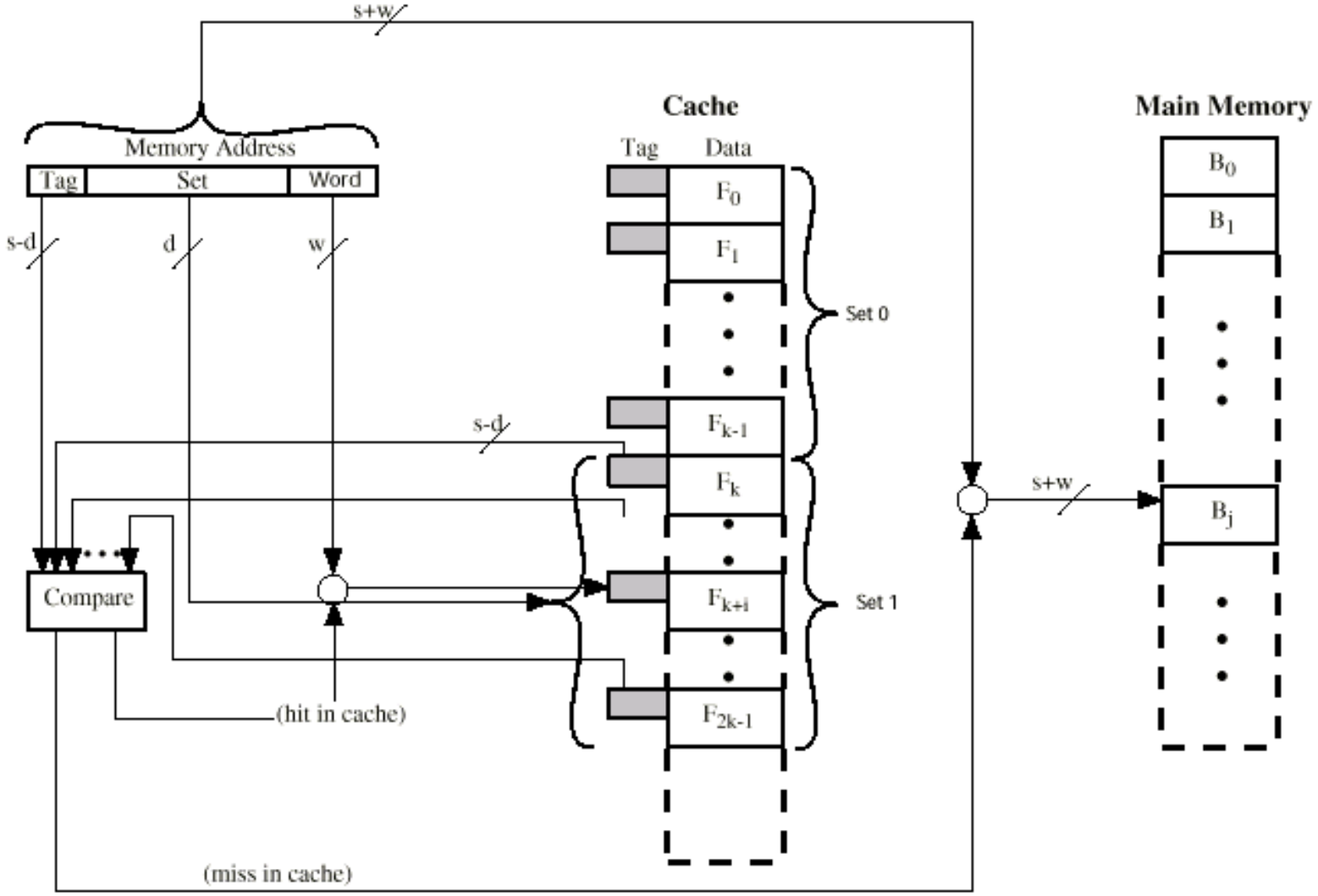
Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = Total Number of cache Blocks
- Size of tag = s bits

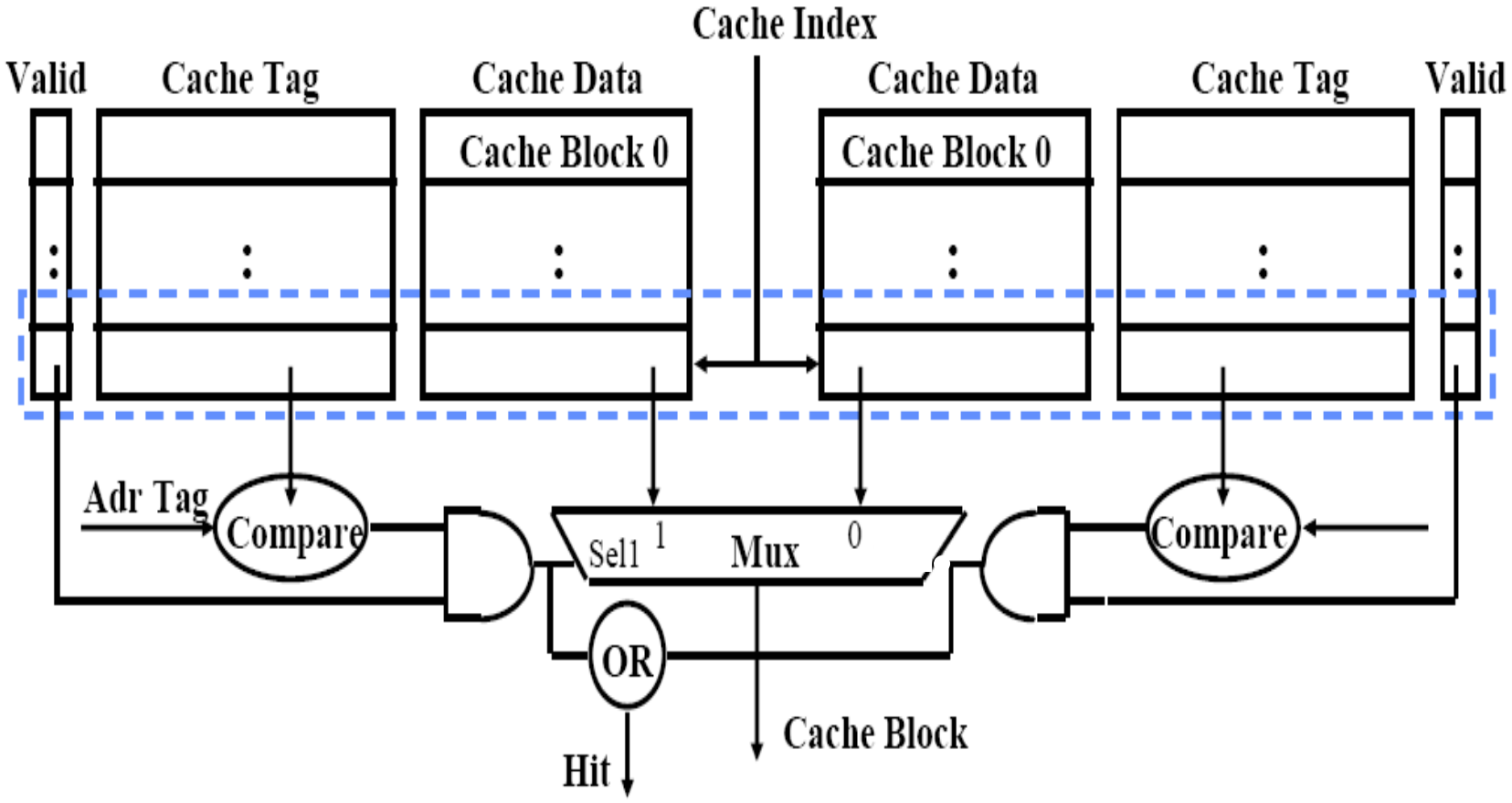
Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

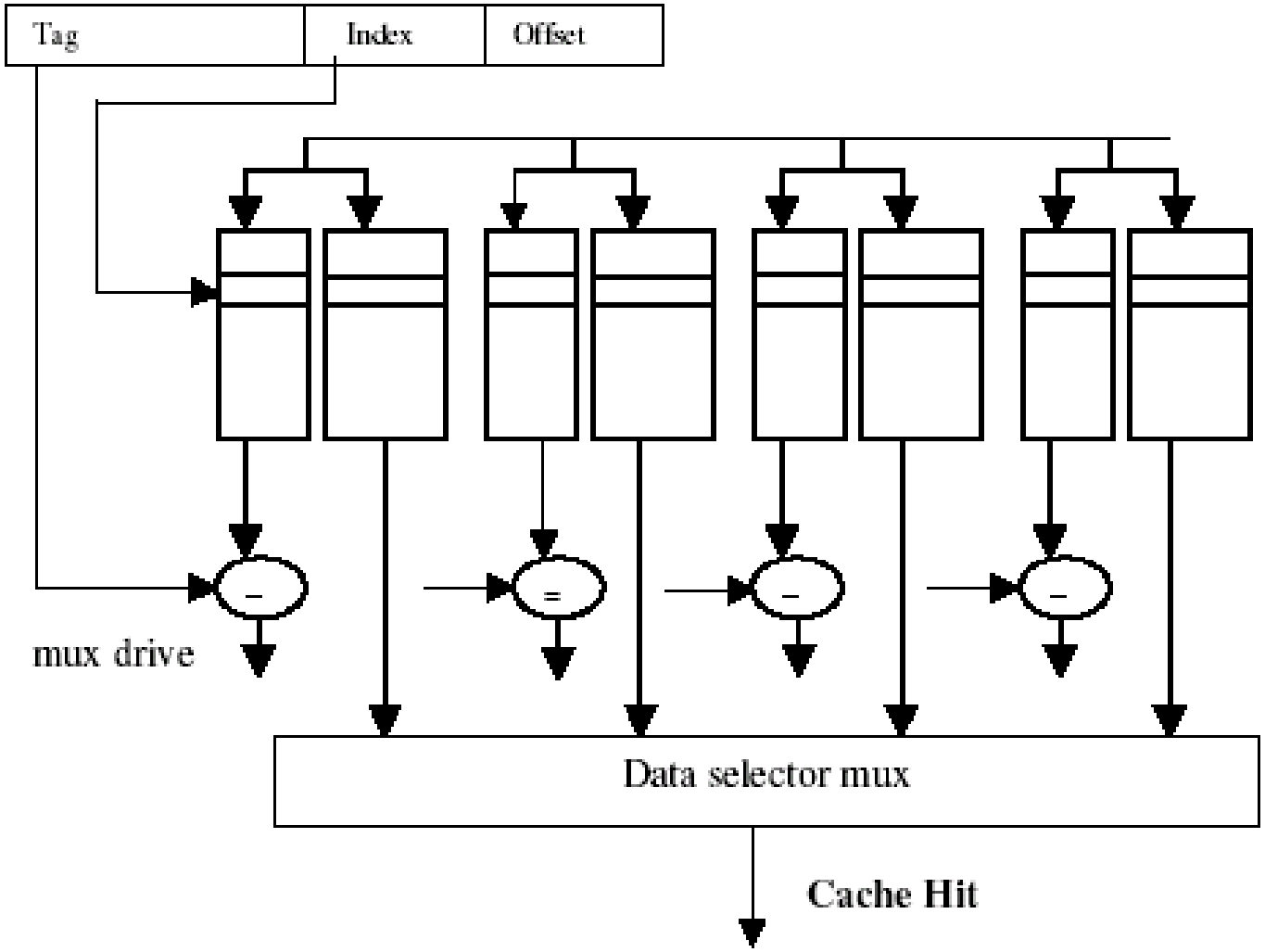
K-Way Set Associative Cache Organization



2 –way set associative cache



4 –way set associative cache



Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = 2^{s}
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2^{d}
- Size of tag = (s – d) bits

Disadvantage of Set Associative Cache

- N-way Set Associative Cache Vs. Direct Mapped Cache:
  - N comparators Vs 1
  - Extra mux delay for the data
  - Data comes after hit/miss
  - In a direct map cache, cache block is available before hit/miss
  - Number of misses
    - DM > SA > FA
  - Access latency
    - DM < SA < FA

Internal Organisation

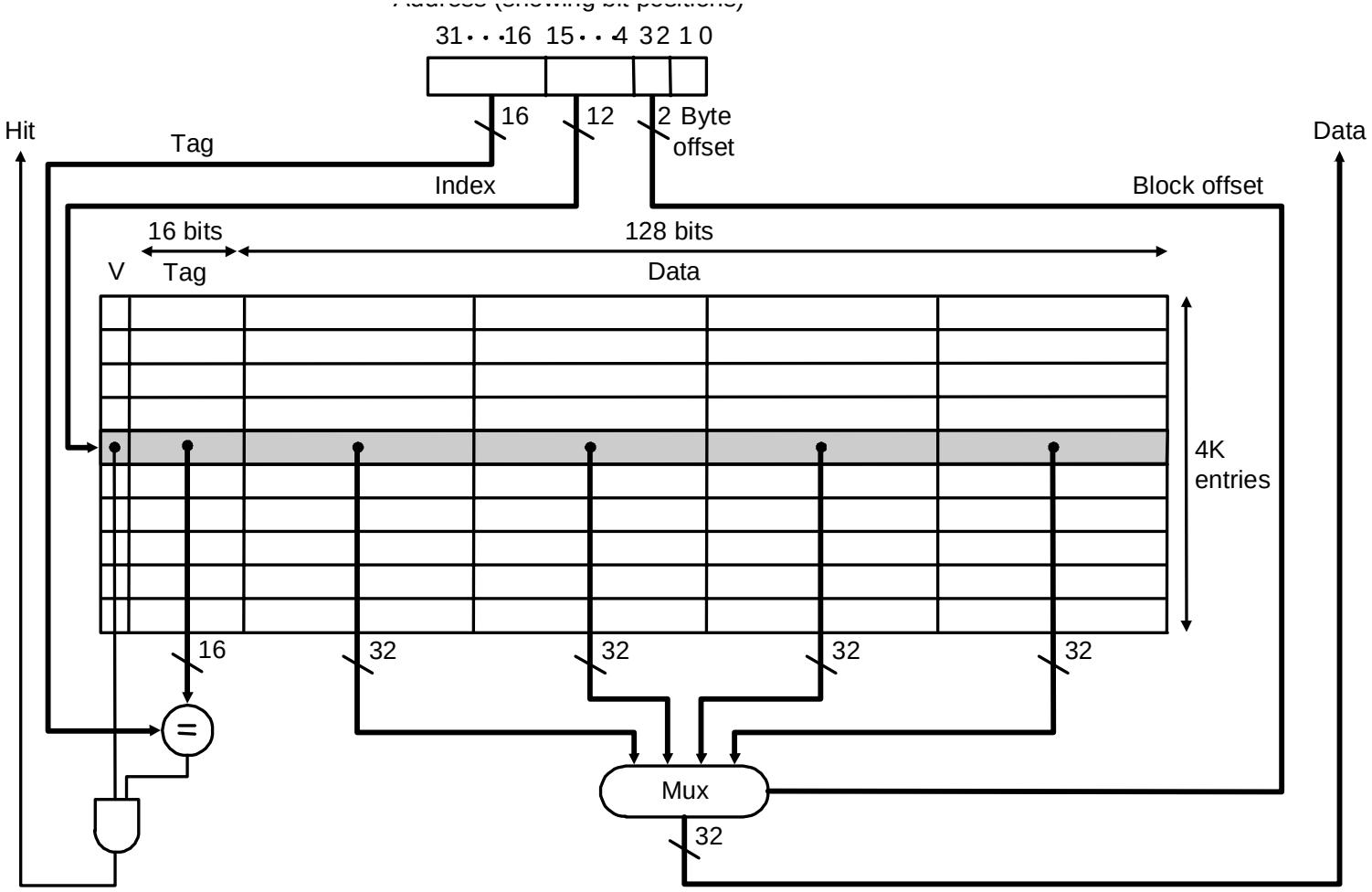
- Size
- Valid Bit?
- How many bits?
  - Direct
  - Fully Associative
  - Set Associative
- Comparison

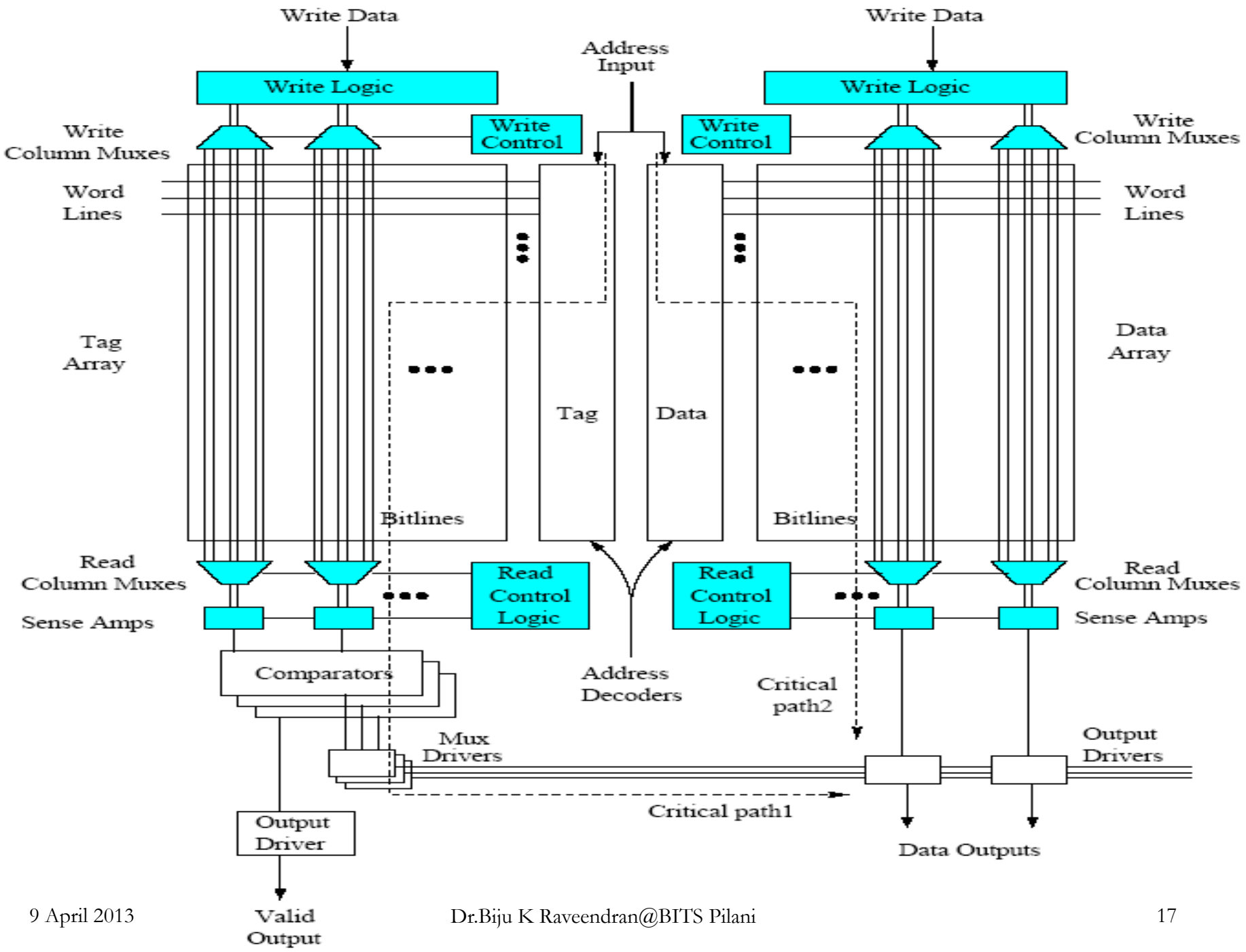
A byte addressable system with 34 bits physical address has 256KB Cache Memory with 32 Byte cache line size

- Find the total storage required if the cache has valid/invalid bit and dirty bit for the following configuration [assume the replacement scheme is not requiring additional storage]
  - Direct mapped
  - Fully associative
  - 4 – way Set Associative

Address (showing bit position)

1 word= 4 bytes




Write Data
Write Data
Address Input
Write Logic
Write Logic
Write Control
Write Control
Write Column Muxes
Write Column Muxes
Word Lines
Word Lines
Tag Array
Data Array
Tag
Data
Bitlines
Bitlines
Read Column Muxes
Read Column Muxes
Read Control Logic
Read Control Logic
Sense Amps
Sense Amps
Comparators
Address Decoders
Critical path2
Mux Drivers
Output Drivers
Critical path1
Output Driver
Data Outputs
Valid Output

Sources of Cache Misses

- Compulsory
- Capacity
- Conflict / Collision / Interference
- Coherence / Invalidation

Compulsory Misses

- The first access to a block is not in the cache, so the block must be brought into the cache.
- Also called cold start misses or first reference misses
- This cache miss will be there even if the cache is INFINITE in size.
- Occurs in all cache sizes and configurations
- Usually occurs when the program starts its execution
  - cache does not contain any of that program's data yet, so misses are bound to occur
  - can't be avoided easily, so won't focus on these in this course

Capacity Misses

- If the cache cannot contain all the blocks needed during execution of a program (its working set)
- Occurs due to blocks being discarded and later retrieved.
- The miss that occurs because the cache has a limited size
- Major source of misses in Fully Associative Cache
- Solutions
  - 1: Increase cache size
  - 2: Restructure program

Conflict Misses

- In SA or DM, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set.
- occurs because multiple distinct memory locations mapped to the same cache location
- Also called Collision misses or Interference misses
- Results in under utilizing cache memory
- A serious problem for DM caches as 2 memory locations mapped to the same cache location can be accessed frequently
  - Will not Occur in Fully Associative caches

Dealing with Conflict Misses

- Solution 1: Make the cache size bigger
  - fails at some point
- Solution 2: Multiple distinct blocks can fit in the same Cache Index
  - Increase the Associativity

Coherence Misses

- Other processor updates memory
- Also called as Invalidation misses
- We will discuss about this later

How to reduce Misses?

- Compulsory Misses?
  - Can not avoid
  - Can be reduced by increasing cache line size
- Capacity Misses?
  - Can be reduced by increasing the cache size
  - Can be reduced by Restructuring the program
- Conflict Misses?
  - Can be reduced by Increasing the associativity

Set Associative & Fully Associative Caches

- To Improve Associative cache performance
  - Design a better replacement scheme
    - Ideally, the replacement scheme should select the cache block (from the set) which we will not use in near future

Replacement Algorithms - Direct mapping

- No choice
- Each block only maps to one line
- Replace that line
- No replacement circuit needed

Replacement Algorithms - Associative & Set Associative caches

- Hardware implemented algorithm (speed)
- Categorized based on
  - Optimal
  - Random
  - FIFO
  - LFU, LFU-DA
  - LRU, Pseudo LRU

Replacement Algorithms

- Optimal - Not implementable
  - Ideal scenario for for comparison study
- Random - block replaced is a random pick
  - Very effective in cache as most recently used blocks only will be available in cache
  - Minimum hardware complexity – No storage required and access time is not affected
    - Hardware requirement is just a pseudo random number generator
    - Hardware complexity does not vary with cache size, associativity and line size
  - Does not guarantee a steady cache hit performance

Replacement Algorithms

- FIFO
  - For an N-way set associative cache
    - Implementation 1
      - Use N-bit register per cache line to store arrival time information
      - On cache miss – registers of all cache line in the set are compared to choose the victim cache line
    - Implementation 2
      - Maintain a FIFO queue
      - Register with ($\log_2$ N) bits per cache line
      - On cache miss – cache line corresponding to register value 00 will be the victim.
      - Decrement all other registers in the set by 1 and set the victim register with value N-1

Replacement Algorithms

- FIFO
  - Advantages
    - Predictable cache behavior with minimum hardware complexity
    - Replacement hardware is activated and modified only during cache miss
    - The cache access time is not affected by the replacement strategy (not in critical path)
  - Disadvantages
    - Cache hit performance is poor compared to LRU and LFU
    - Not suitable for high performance systems
    - Replacement circuit complexity increases with increase in associativity and number of cache lines

Replacement Algorithms

- Frequency
  - Least Frequently Used (LFU)
    - Requires a register per cache line to save number of references (frequency count)
    - If cache access is hit, then increase frequency count of the corresponding register by 1
    - If cache miss, find the victim cache line as the cache line corresponding to minimum frequency count in the set
    - Reset the register corresponding to victim cache line as 0
    - LFU can not differentiate between past references and recent references
  - Least Frequently Used – Dynamic Aging (LFU-DA)
    - When any frequency count register in the set reaches its maximum value, all the frequency count registers in that set is shifter one position right (divide by 2)

Replacement Algorithms

- LFU
  - Advantages
    - For small and medium caches LFU works better than FIFO and Random replacements
    - Suitable for high performance systems whose memory pattern follows frequency order
  - Disadvantages
    - Cache pollution is possible (Can overcome by LFU-DA)
    - The register should be updated in every cache access
    - Affects the critical path
    - The replacement circuit becomes more complicated when increasing the associativity and number of lines

Replacement Algorithms

- Least Recently Used
  - The most widely used replacement strategy
  - Replaces the Least Recently Used Cache line
  - Requires a register per cache line
  - If cache hit, Update all the registers in the set
  - If cache miss, the cache line corresponding to 0 as register value will be the victim line

Replacement Algorithms

- LRU
  - Advantages
    - Works better than almost all the other algorithms
    - Most suitable for high performance systems
    - Performs close to optimal when associativity is less
    - Gives very good results whose memory pattern follows temporal locality
  - Disadvantages
    - Requires additional hardware & complicated control circuitry
    - The registers should be updated in every cache access
    - Affects the critical path
    - The replacement circuit becomes more complicated when increasing the associativity and number of lines