CS / EE 320
Computer Organization and Assembly Language
Spring 2024
Lecture 24

QUIZ 5 Today

Shahid Masud

Topics: Direct Mapped Cache, Associative and Set Associative Caches, Cache Miss Processing, Example questions, Cache Performance

Topics

- Example of Direct Mapped Caches
- Associative Cache Design
- Set Associative Cache Design
- Examples of calculating different parameters in Cache Design
- Cache Replacement: LRU, LFU, FIFO, Random schemes
- Cache Writes – Write Back and Write Through schemes
- Some Examples of Cache Performance
- Introduction to Multi-level Caches (examples in next lecture)

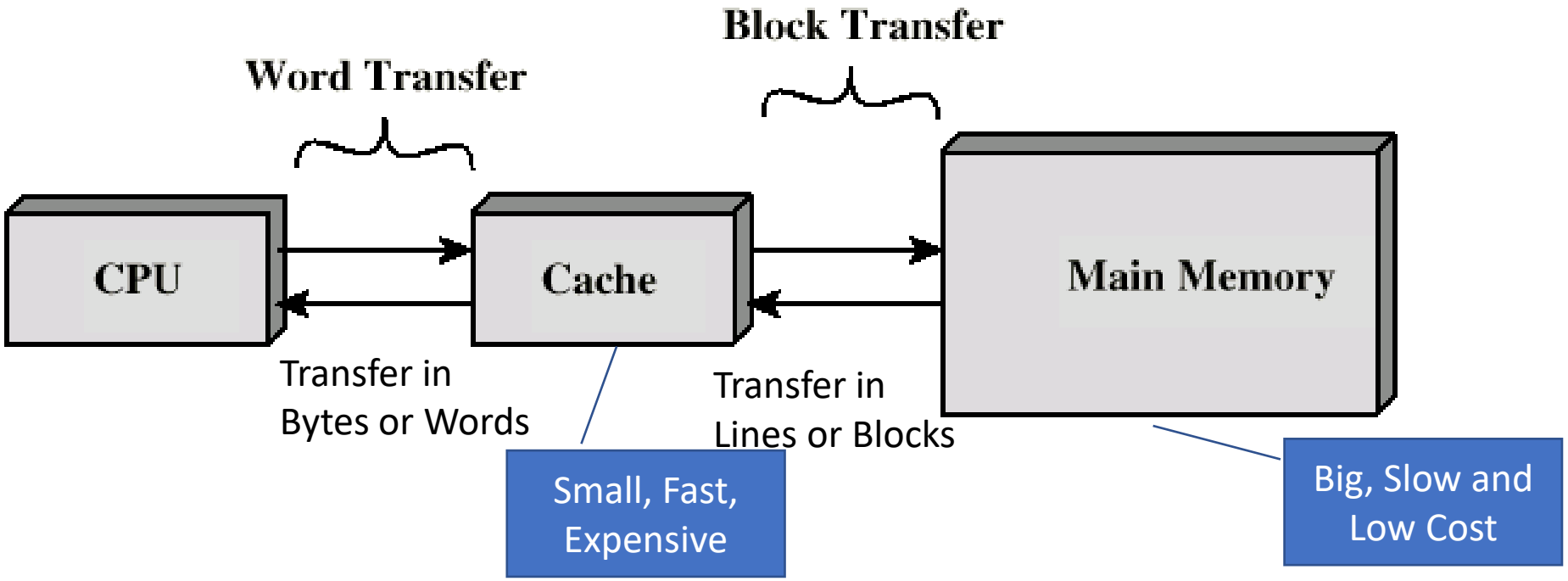
Principle of Locality

- Programs access a small proportion of their address space at any time
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

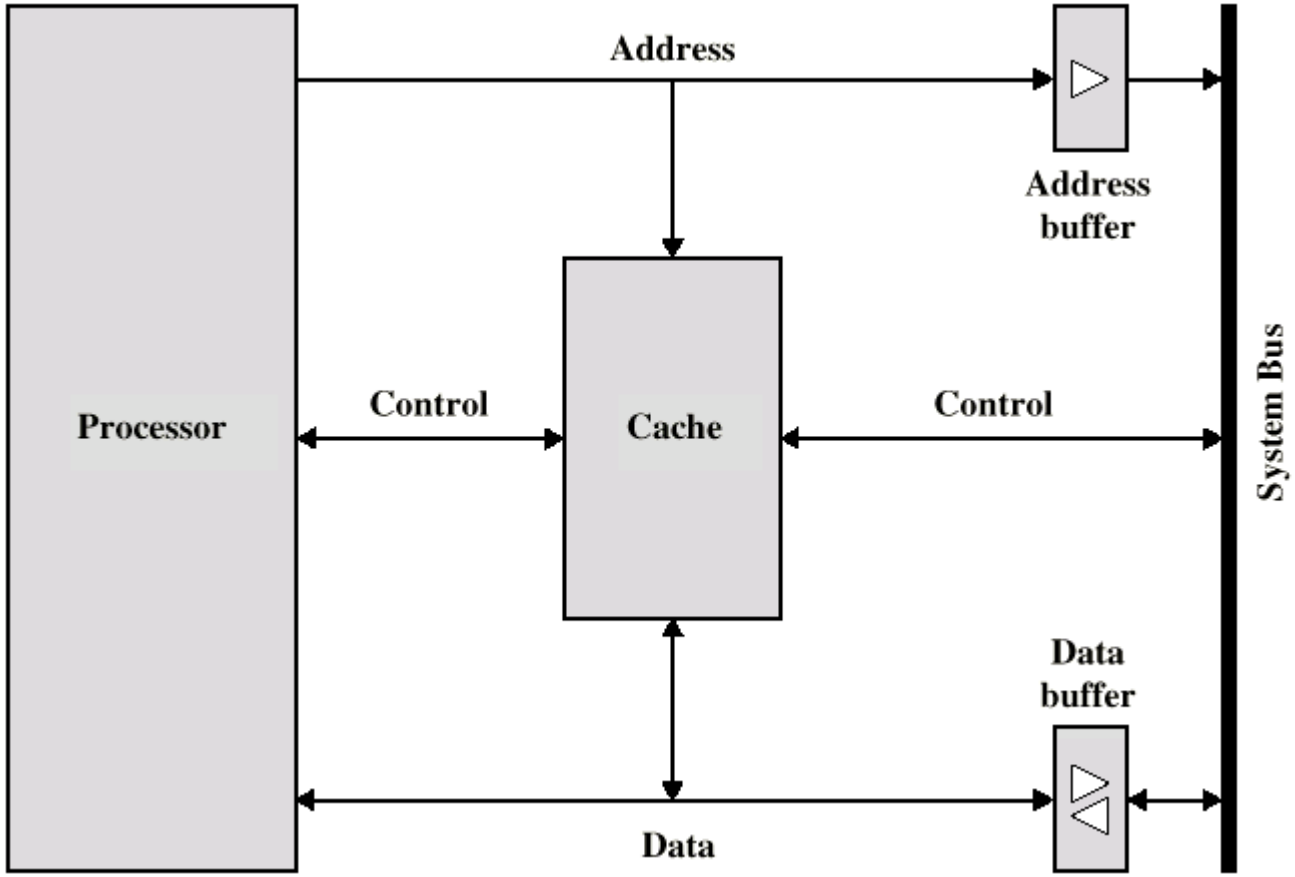
Cache Memory

Cache and Memory

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module



Typical Cache Connections



Cache operation - Overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

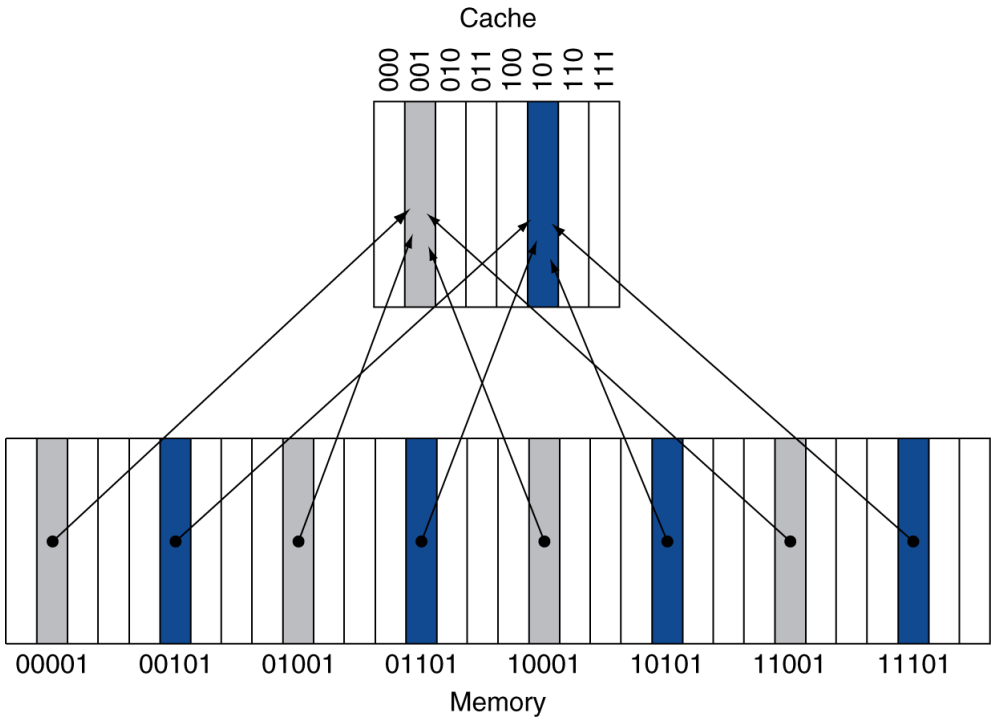
Cache Design Parameters – will discuss today

- Size and Capacity
  - How many lines?
  - How many Words per line?
- Mapping Function
  - Direct, Associative, Set Associative
- Replacement Algorithm
  - LRU, FIFO, etc.
- Write Policy
  - Write Through vs Write Back
- Number of Caches
  - Levels of Caches

Direct Mapped Cache

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (No. of Blocks in cache)



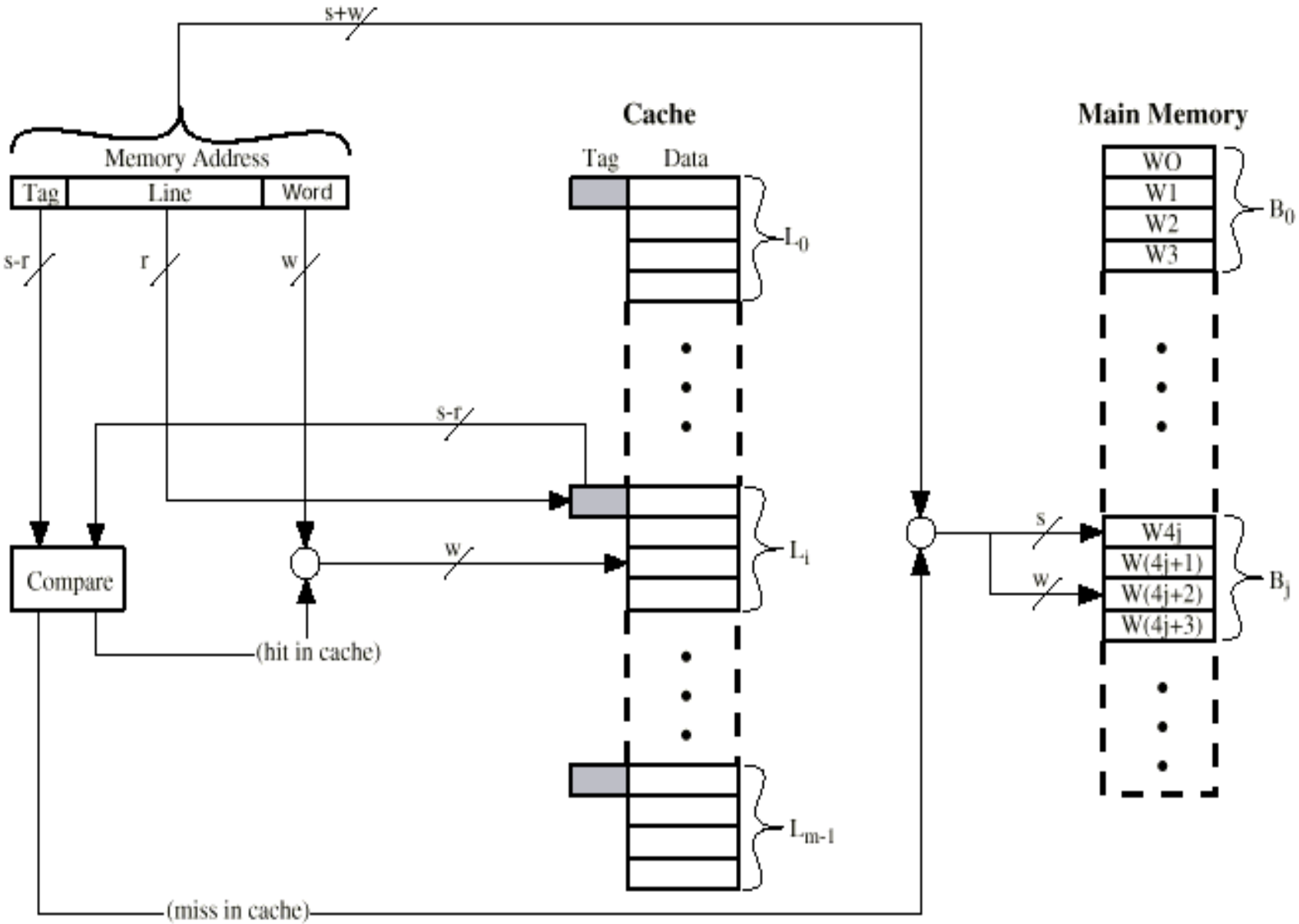
- No of Blocks is a power of 2
- Use low-order address bits

Direct Mapping Address Structure Example

Tag s-r	Line or Slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Direct Mapping Cache Organization – Read / Write



Mapping Function

- Cache of 64 Kilo Byte
- Cache block of 4 bytes
  - i.e. cache is 16k (2^{14}) lines of 4 bytes
- 16 Mega Bytes main memory
- 24 bit address
  - (2^{24}=16M)

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 are very high

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

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

V = Valid shows required data already present in cache or not

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Line Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

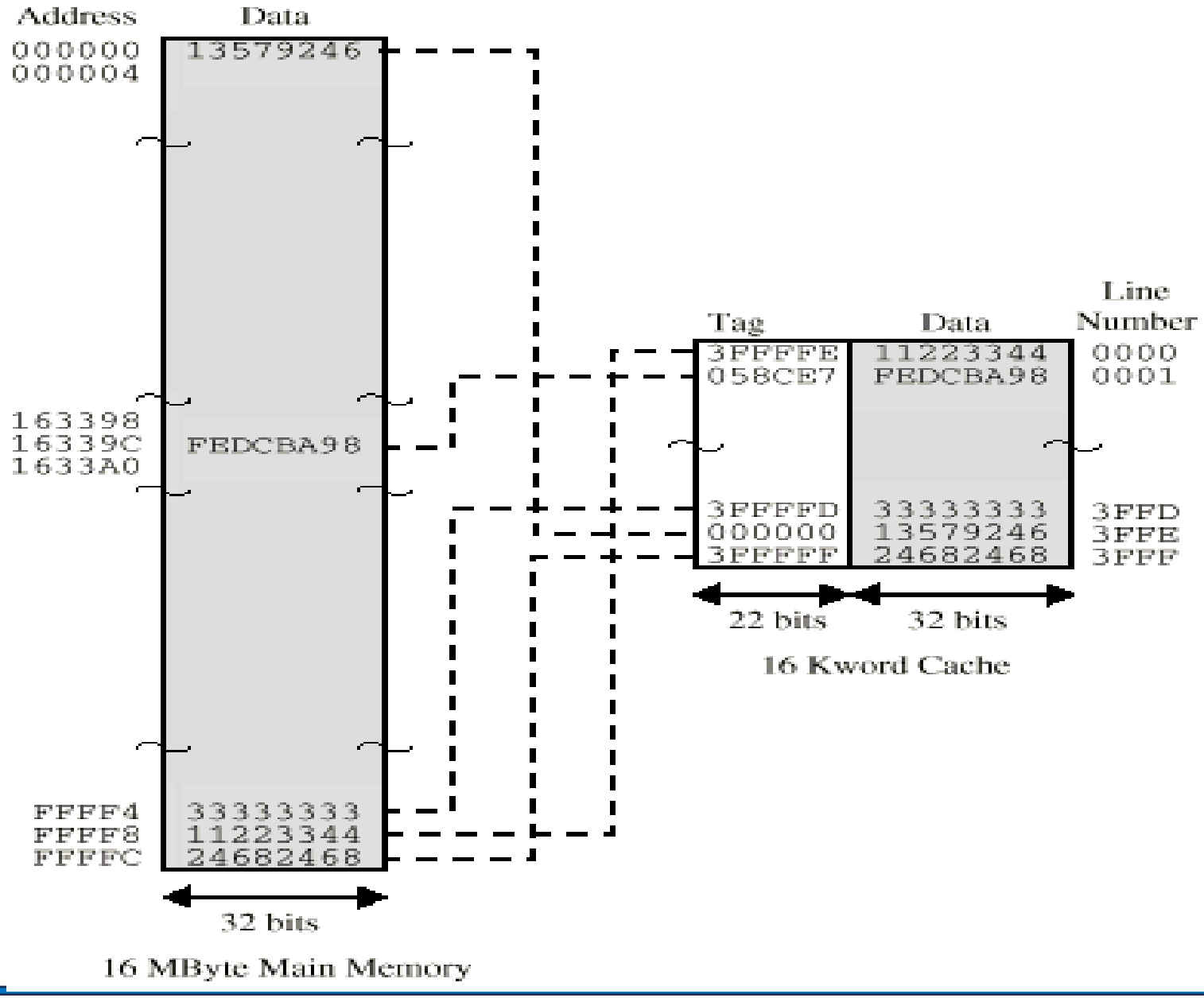
Line Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Associative Mapped Cache

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

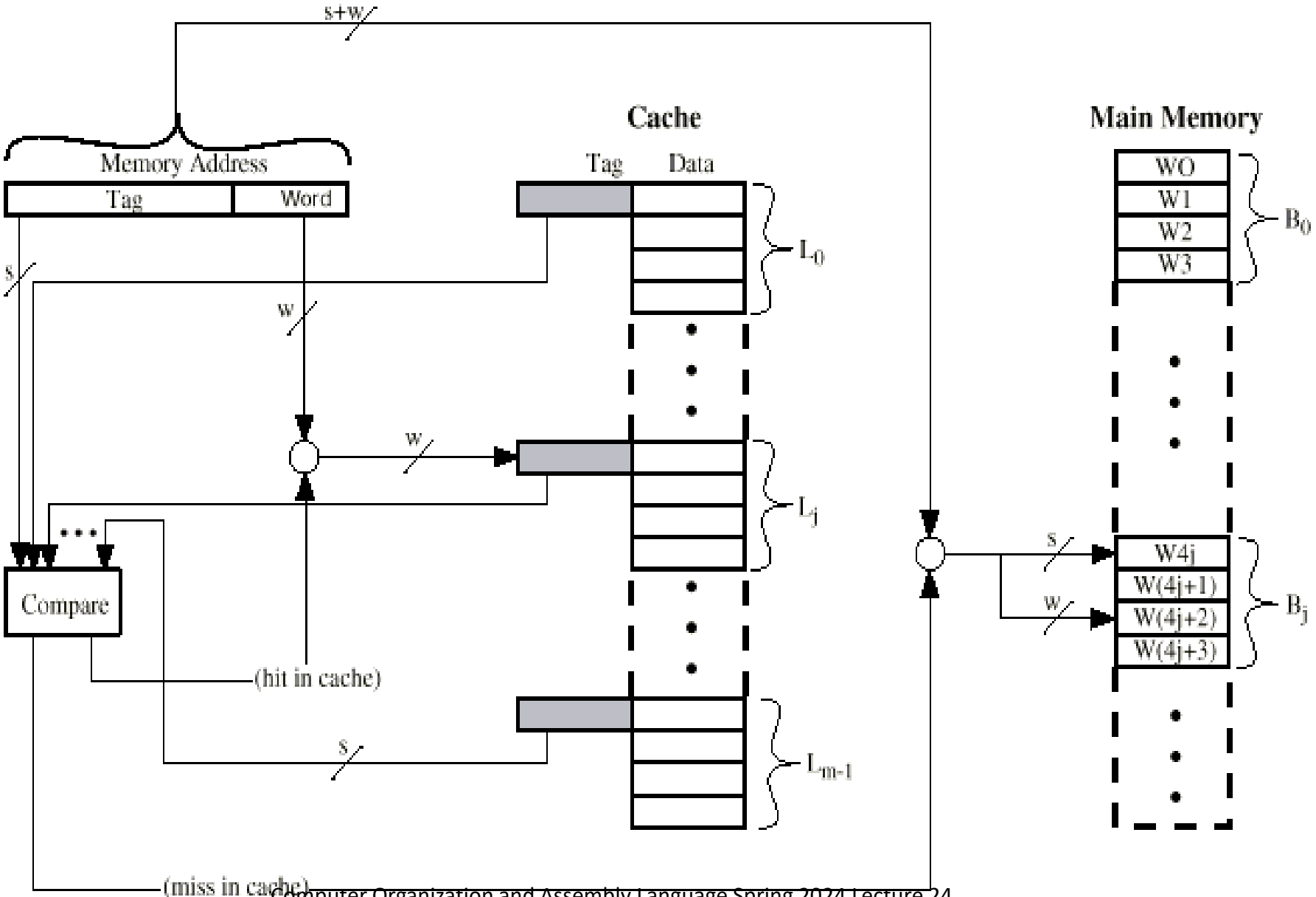
Associative Mapping Example



Associative Mapping Address Structure

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which byte (8bit) is required from 32 bit data block
- e.g. (based 16 number (4bit))
  - Address Tag Data Cache line
  - FFFC FFFC 24682468 3FFF

Fully Associative Cache Organization – Read / Write



How Much Associativity

- Increased associativity decreases miss rate
  - With diminishing returns
- Example: Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Mapped Cache

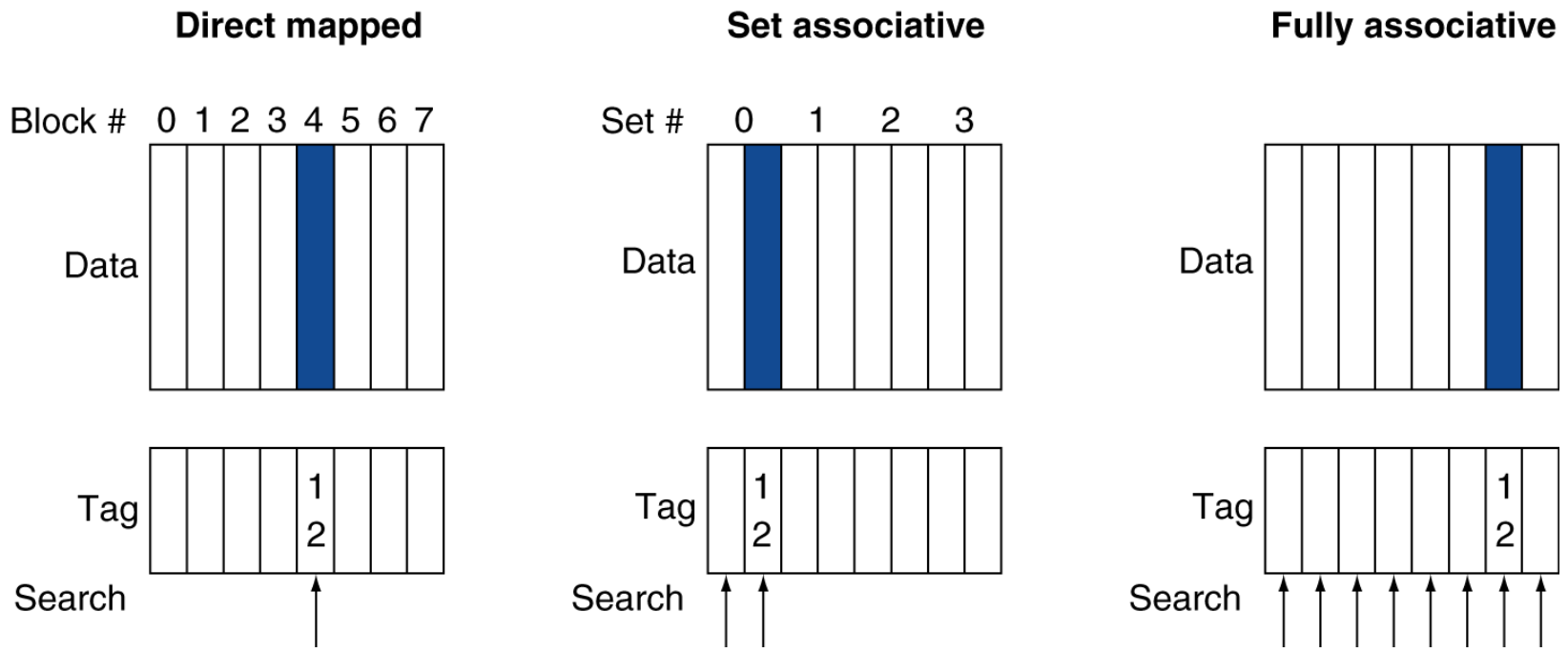
Set Associative Mapping

- 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

Associative Caches (and Set Associative)

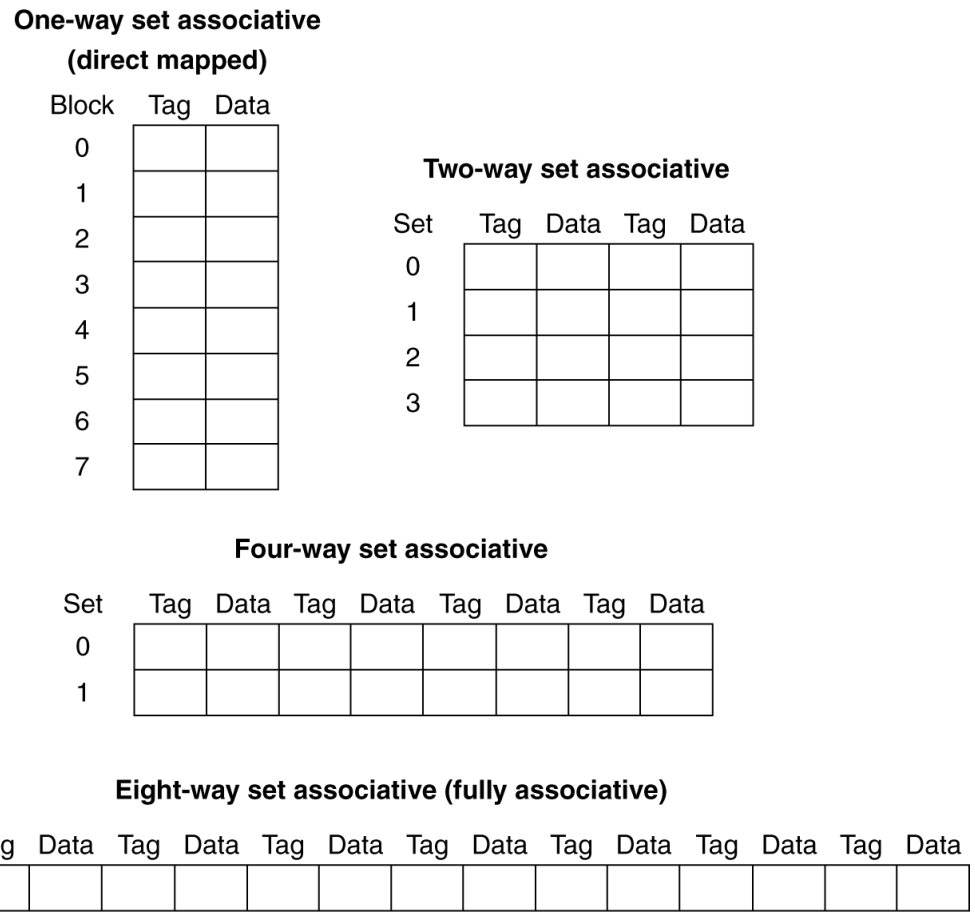
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries



Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

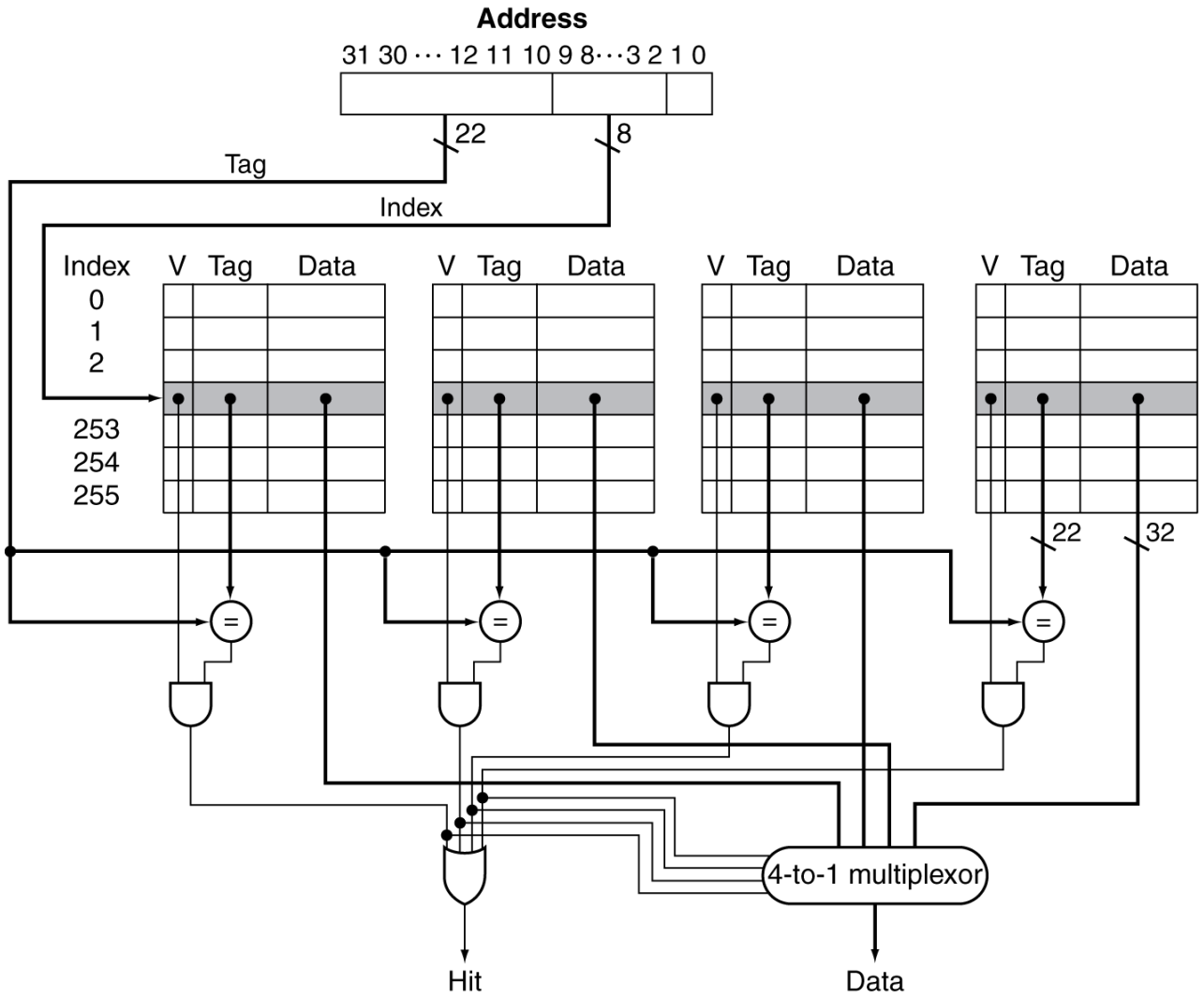
- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

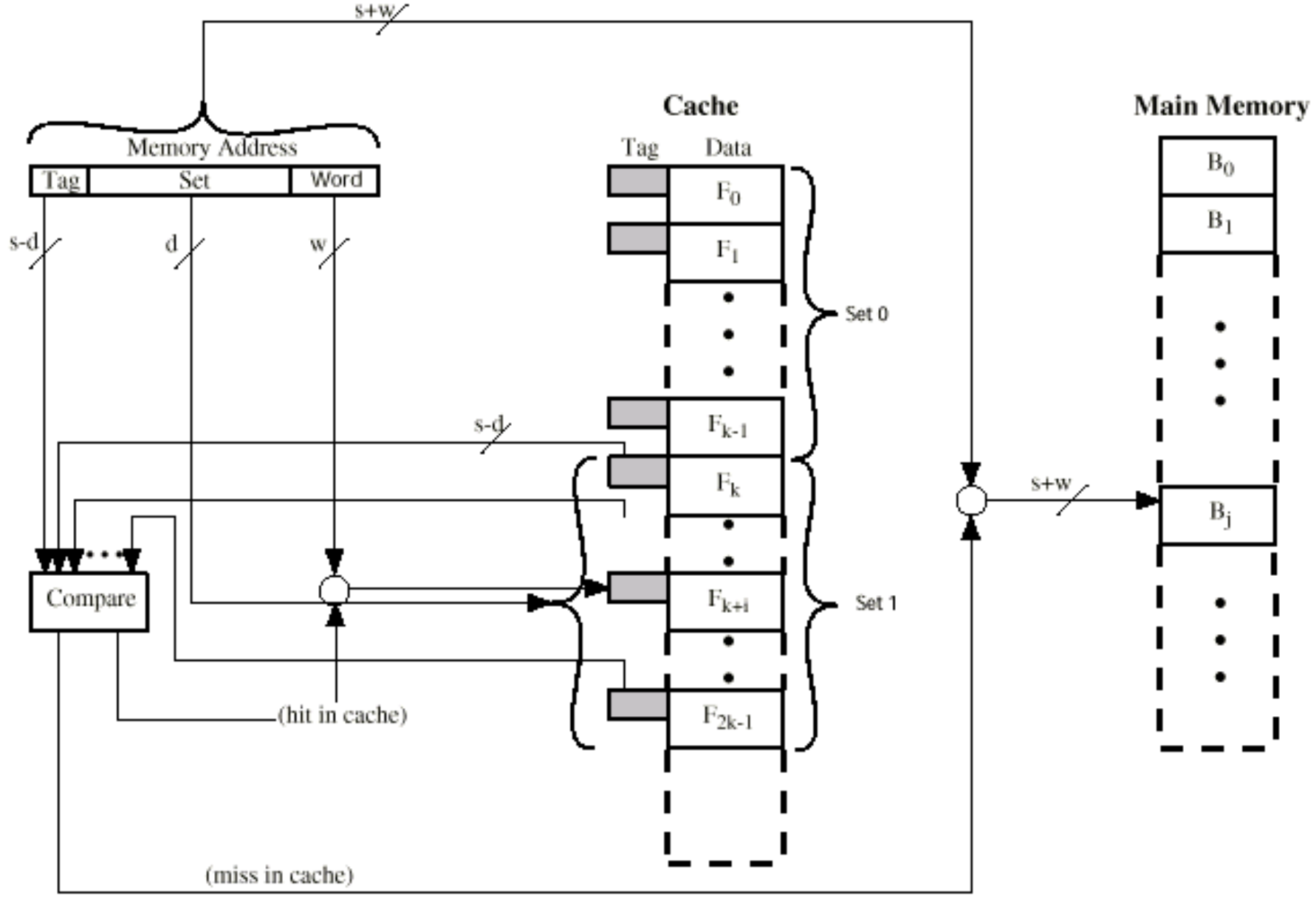
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

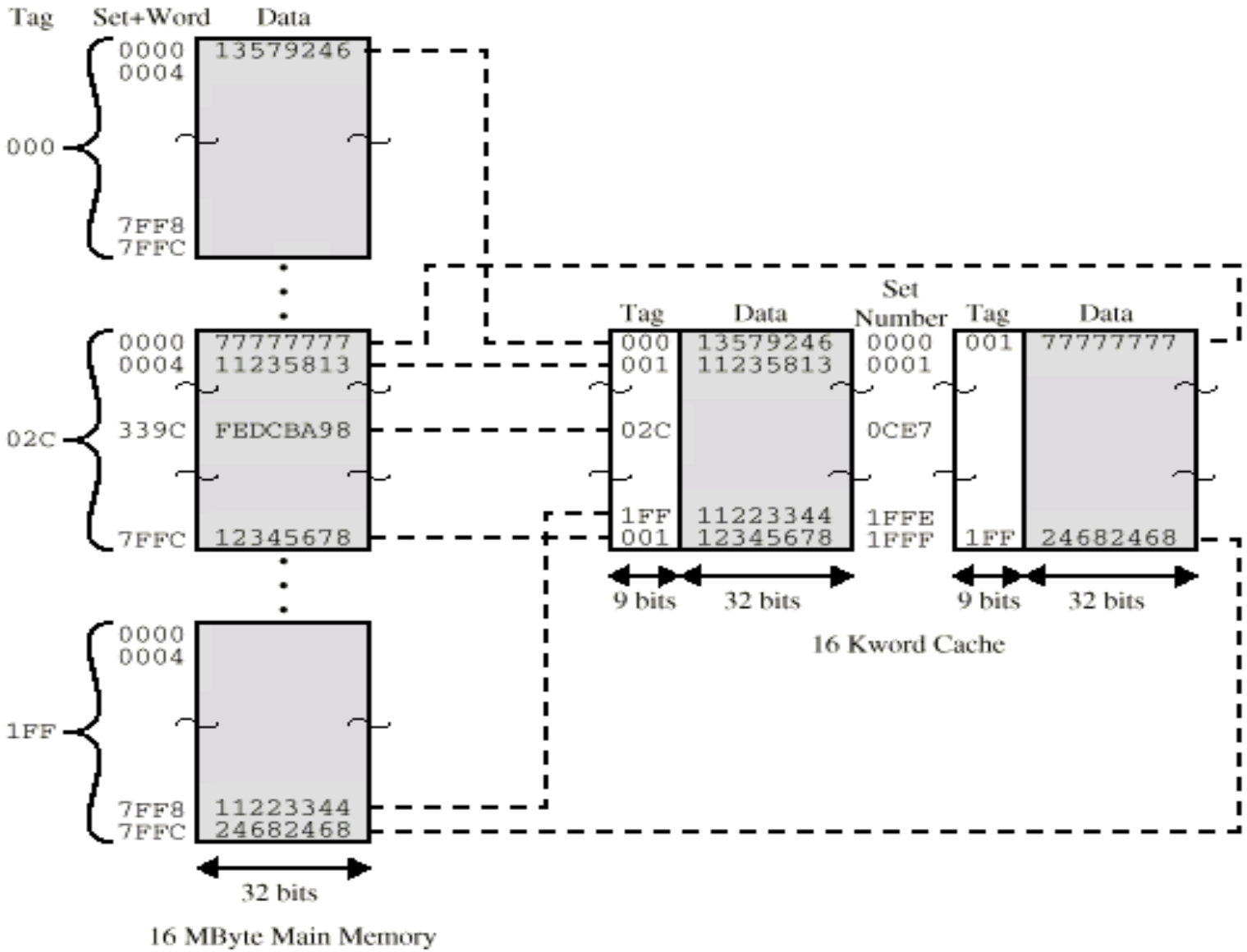
Set Associative Cache Organization



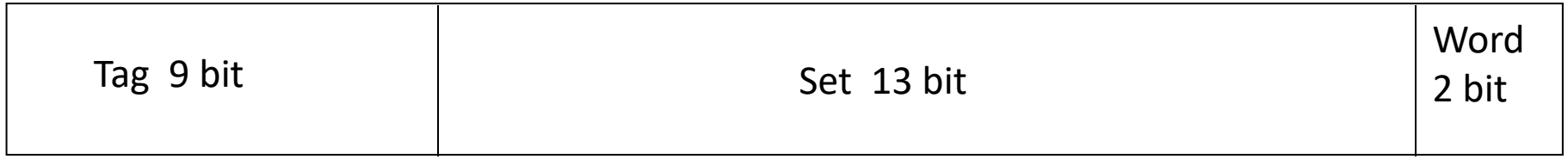
Set Associative Cache Organization



Two Way Set Associative Mapping Example



Set Associative Mapping Address Structure



- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g.
  - Address Tag Data Set number
  - 1FF 7FFC 1FF 12345678 1FFF
  - 001 7FFC001 11223344 1FFF

Dealing with Cache Miss and Cache Line Replacement

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Replacement Algorithms (1) Direct mapping

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

Replacement Algorithms (2)
Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
- e.g. in 2 way set associative
  - Which of the 2 block is lru?
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used
  - replace block which has had fewest hits
- Random

Write Policy

- Must not overwrite a cache block unless main memory is up to date
- Multiple CPUs may have individual caches
- I/O may address main memory directly

Write through

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Write Miss

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Block Placement

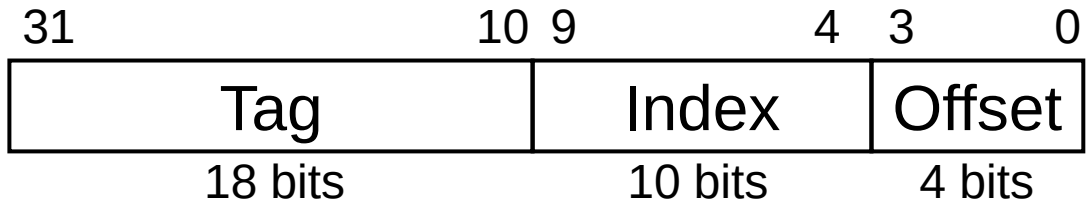
- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete



Multilevel On-Chip Caches

Characteristic	ARM Cortex-A53	Intel Core i7
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	Configurable 16 to 64 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	Two-way (I), four-way (D) set associative	Four-way (I), eight-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, variable allocation policies (default is Write-allocate)	Write-back, No-write-allocate
L1 hit time (load-use)	Two clock cycles	Four clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 2 MiB	256 KiB (0.25 MiB)
L2 cache associativity	16-way set associative	8-way set associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	12 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories
  - Caching gives this illusion
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors

Readings

- Chap 5 of P&H Textbook