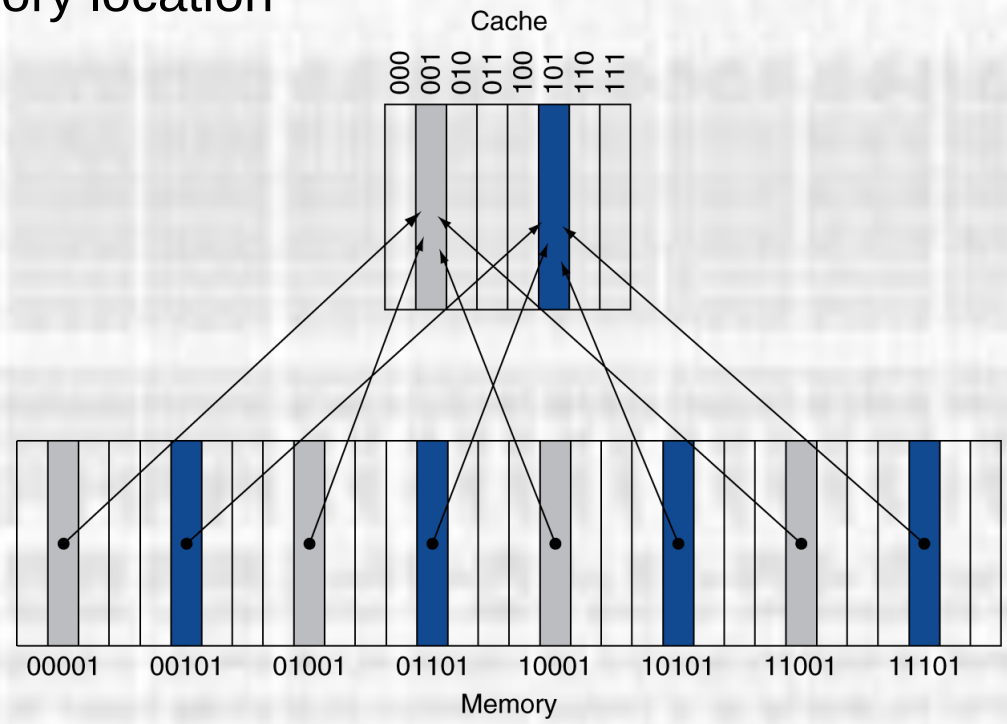
ELE 655
Microprocessor System Design

Class 4 – Cache Review

Cache Basics

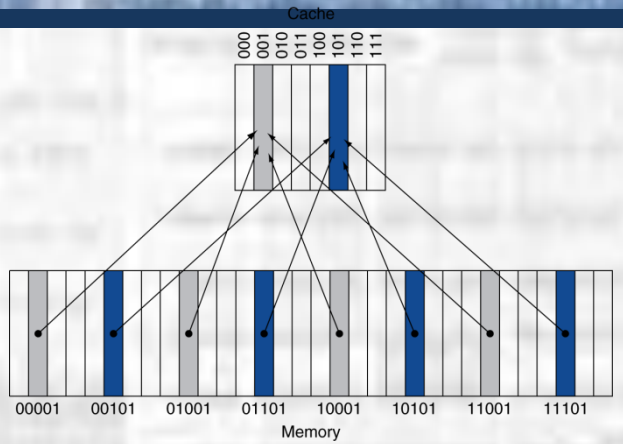
Cache Memory

- Direct Mapped Cache
  - Every higher level memory location is mapped to a single cache memory location



Cache Basics

Cache Memory



- Direct Mapped Cache
  - Cache size is built to be a power of 2
  - Cache block =
    (Block Address) mod (# of cache blocks)
    - Eg. Assume a 256 block cache
      Where does the memory block from address 0x2A3F map to?

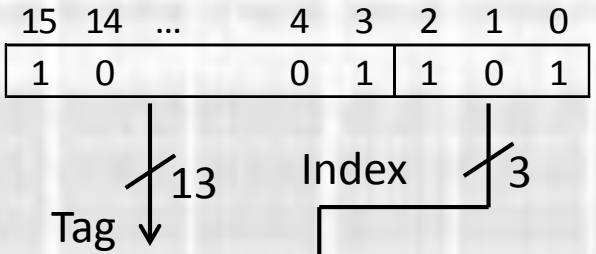
      0x2A3F mod 256_{10} = 0x3F = 63_{10}
  - As long as we follow this convention (cache size = 2^n)
    - **Cache block address = last n bits of the memory address***

*** for 1 byte block sizes**

Cache Basics

Cache Memory

- Direct Mapped Cache
  - 8 block cache, 1byte/block, 16 bit address space



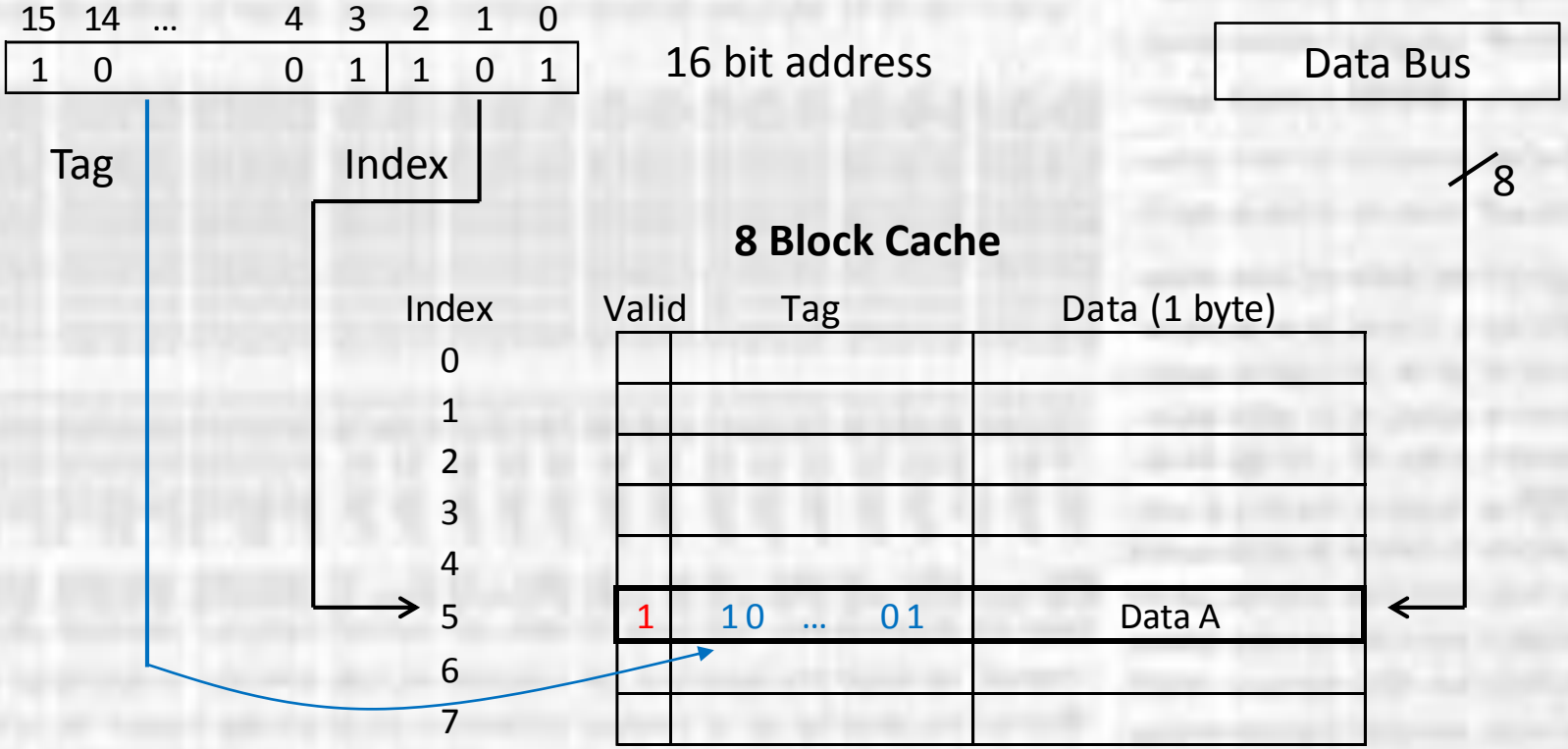
8 Block Cache

Index	Valid	Tag	Data (1 byte)
0			
1			
2			
3			
4			
5			
6			
7			

Cache Basics

Cache Memory

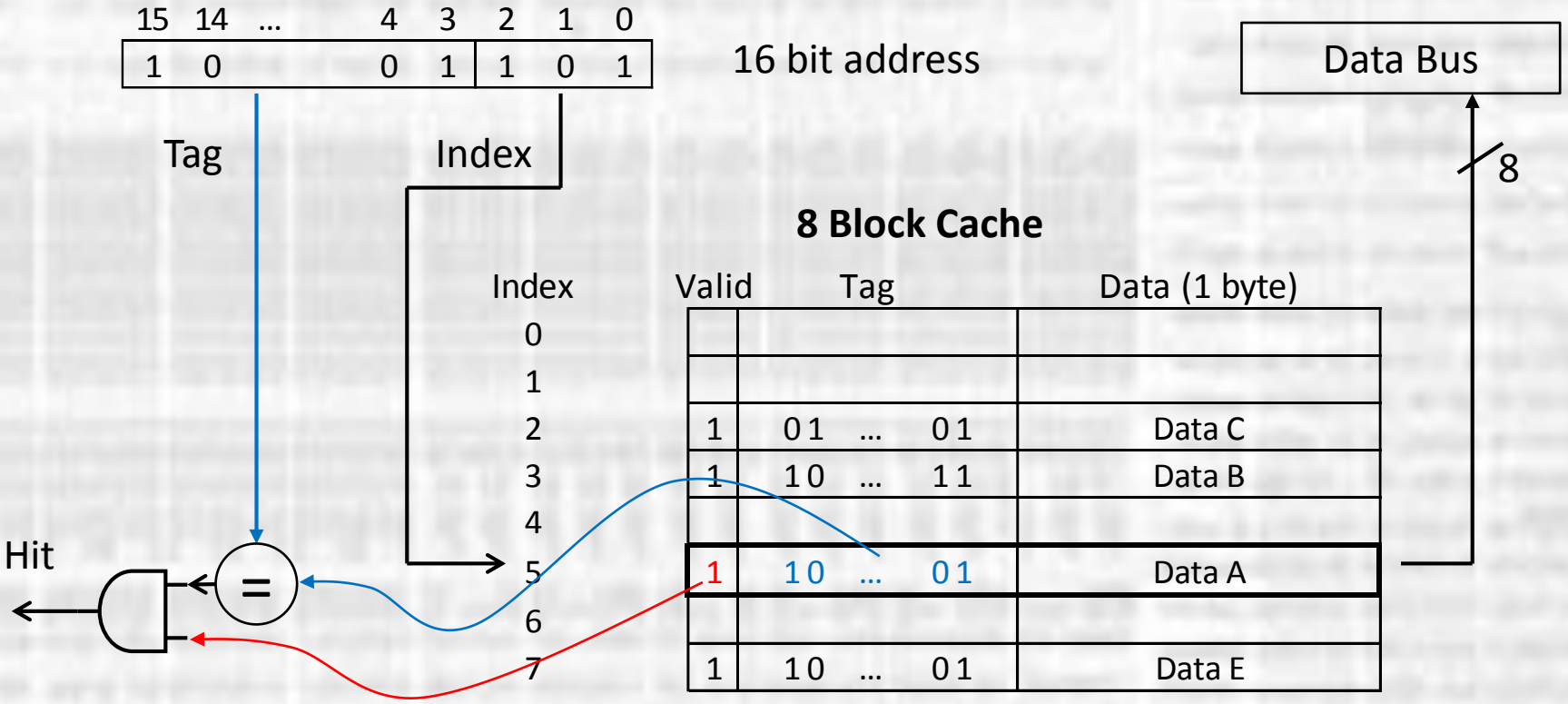
- Direct Mapped Cache
  - 8 block cache - Write



Cache Basics

Cache Memory

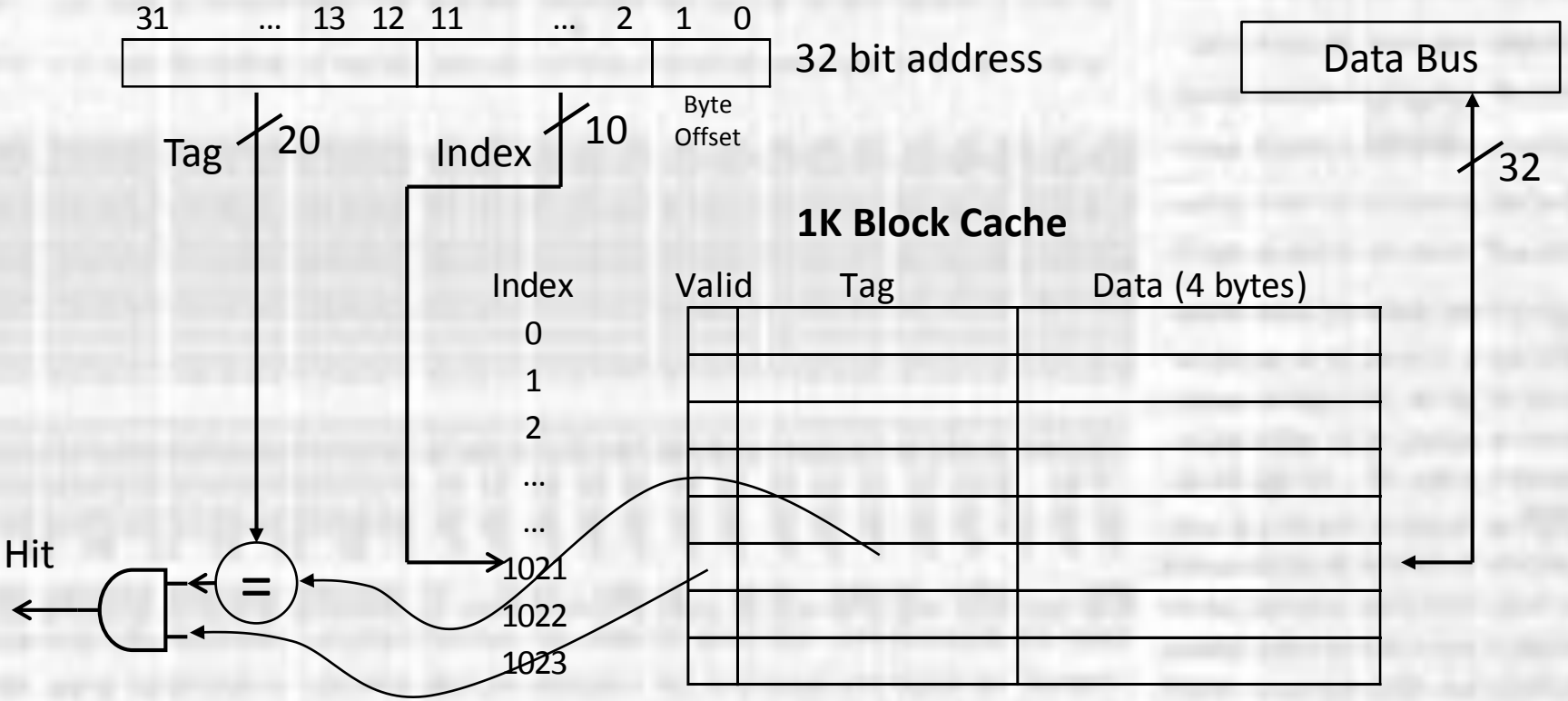
- Direct Mapped Cache
  - 8 block cache - Read



Cache Basics

Cache Memory

- Direct Mapped Cache
  - 1K block cache, 1 word block, 32 bit data word, 32 bit address space

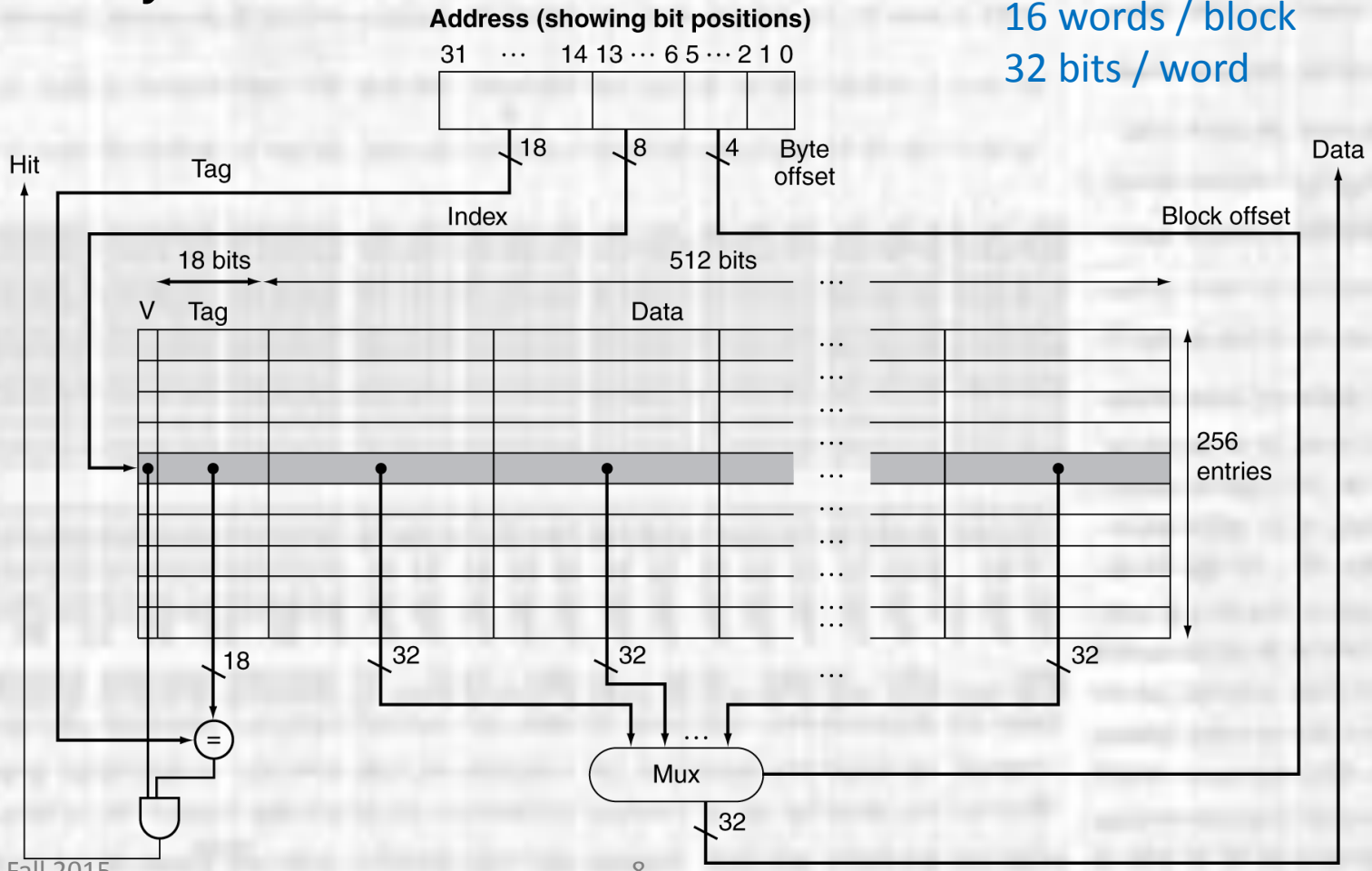


Cache Basics

Cache Example

- Intrinsity FastMATH Processor

256 Blocks
16 words / block
32 bits / word



Cache Performance

Measurement

- CPU performance
  - CPU Time
    - Clock Cycle Time x (CPU execution cycles + CPU stall cycles)
  - CPU Stall Cycles
    - Hazard stall cycles + Read stall cycles + Write stall cycles
    - let Hazard stall cycles go to zero with various techniques
    - CPU stall cycles = Memory stall cycles = Read stall cycles + Write stall cycles

Cache Performance

Measurement

- CPU performance
  - Read Stall Cycles
    - Stalls due to read misses
    - $$\text{Read stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

Cache Performance

Measurement

- CPU performance
  - Write Stall Cycles (write through)
    - Stalls due to write misses
      and
    - Write buffer stalls (buffer full)
    - $$\text{Write stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$
    - Design our system to make Write buffer stalls negligible
      - Fast L2 memory
      - Deep write buffer
    - $$\text{Write stall cycles} = \frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}$$

Cache Performance

Measurement

- CPU performance
  - Read and Write miss penalty is the same
    - In both cases the penalty is the time to read the value from memory
  - Define a Miss Rate which measures the miss rate for memory accesses – read or write
  - $$\text{Memory stall cycles} = \frac{\text{Memory Accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

    or
  - $$\text{Memory stall cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance

Measurement

- CPU performance - example

$CPI_{ideal} = 2$
2% instruction miss rate
4% data miss rate
100 cycle miss penalty
36% of instructions are Loads or Stores

Instruction Miss Cycles = Icount x 2%miss/inst x 100cycles/miss
= 2 x Icount

Data Miss Cycles = Icount x 36%LS/inst x 4%miss/LS x
100cycles/miss
= 1.44 x Icount

Cache Performance

Measurement

- CPU performance – example cont'd

Memory Stall Cycles = 2 Icount + 1.44 Icount = 3.44 Icount

This is almost 3.5 stalls per instruction !!!

$CPI = CPI_{ideal}$ + 3.44 clocks/inst = 5.44 clocks/inst

Only achieving 37% of the ideal performance

Cache Performance

Measurement

- CPU performance – example cont'd

If we improve the processor to a $CPI_{ideal} = 1$ (better pipeline)

$CPI = CPI_{ideal} + 3.44$ clocks/inst = 4.44 clocks/inst

This improves the performance – but not linearly

Only achieving 22.5% of the ideal performance

Cache Performance

Measurement

- CPU performance
  - We have assumed a 1 clock cycle Hit time – this may or may not be true
  - Use the Average Memory Access Time to measure performance
  - AMAT = Time for a hit + (Miss Rate x Miss penalty) seconds
    or
    AMAT = Clock cycle time x (Hit Cycles + Miss Rate x Miss Penalty)

Cache Performance

Measurement

- CPU performance - example

  1GHz clock
  1 cycle cache access time
  5% miss rate
  20 cycle miss penalty

AMAT = 1ns/clk x (1 clk/hit + 5% x 20clk/miss) = 2ns

Cache Performance

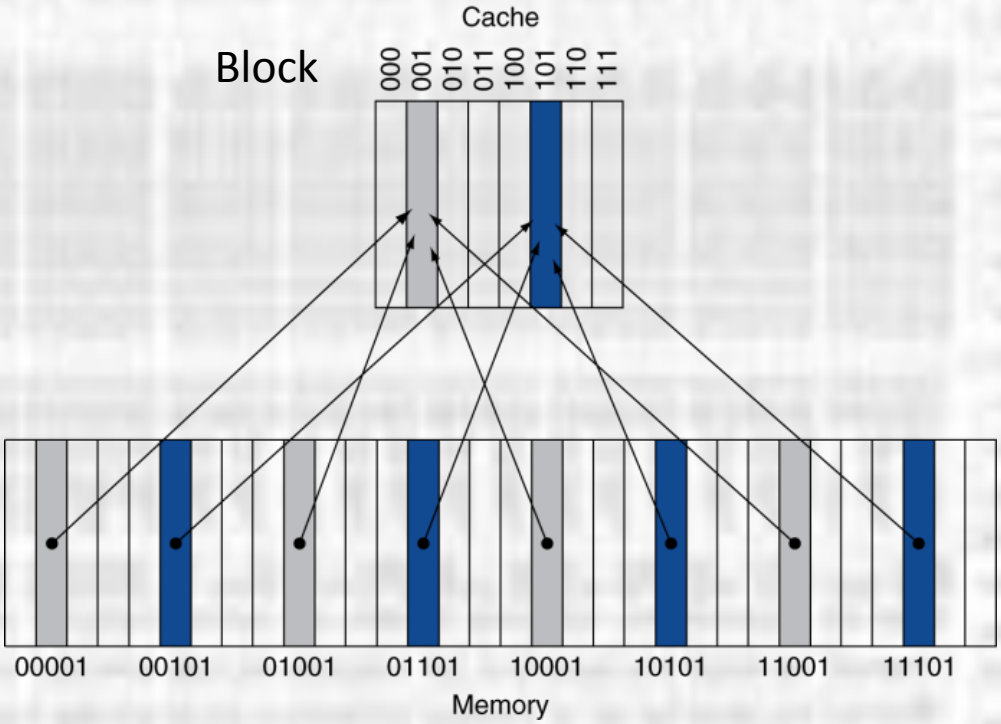
Measurement

- CPU performance
  - Memory performance is critical to overall performance
    - Impacts CPI
    - Impacts AMAT

Cache Performance

Set Associative Cache

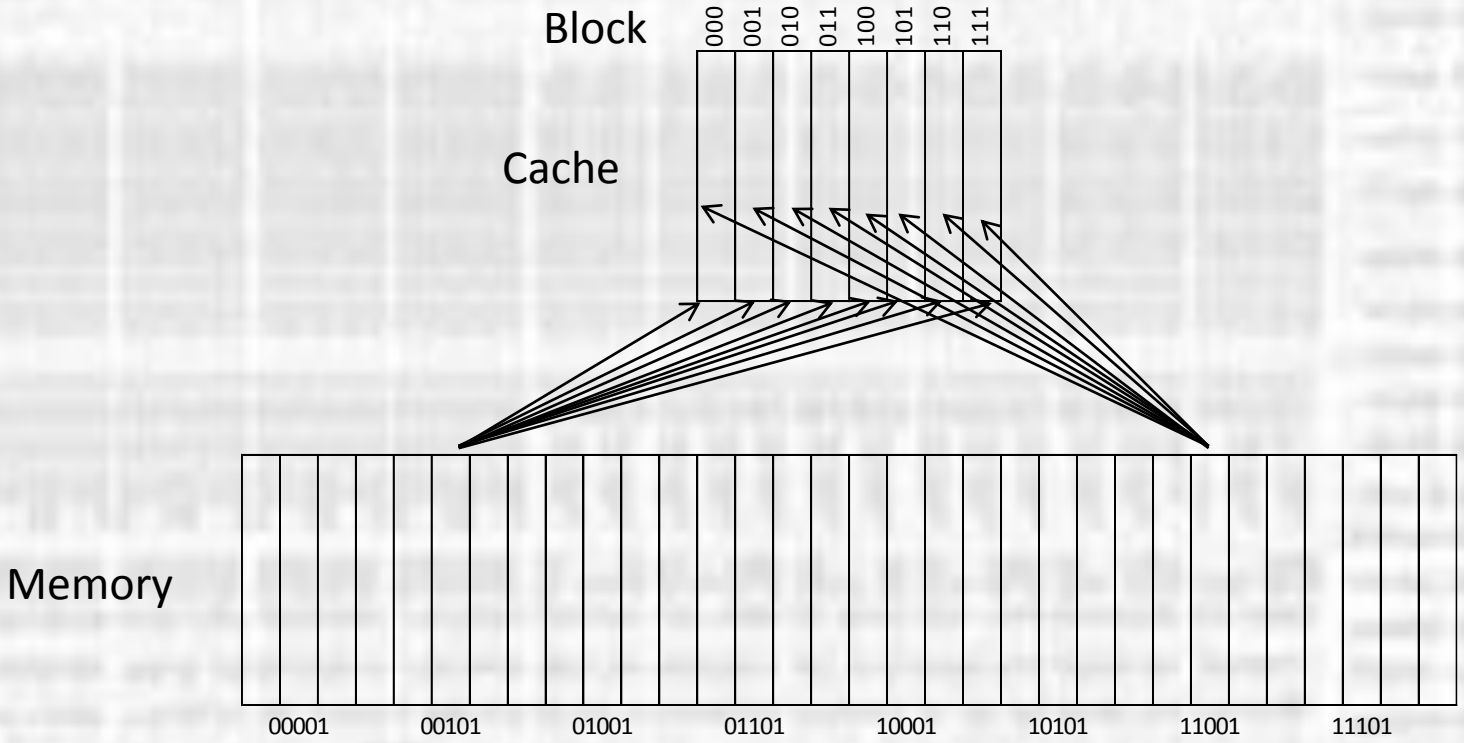
- Direct Mapped Cache
  - Maps each memory location into a single cache location



Cache Performance

Set Associative Cache

- Fully Associative Cache
  - Maps each memory location to any cache block



Cache Performance

Set Associative Cache

- Fully Associative Cache
  - Maps each memory location to any cache block
  - Reduces the number of mapping conflicts
  - Reduces the number of Misses

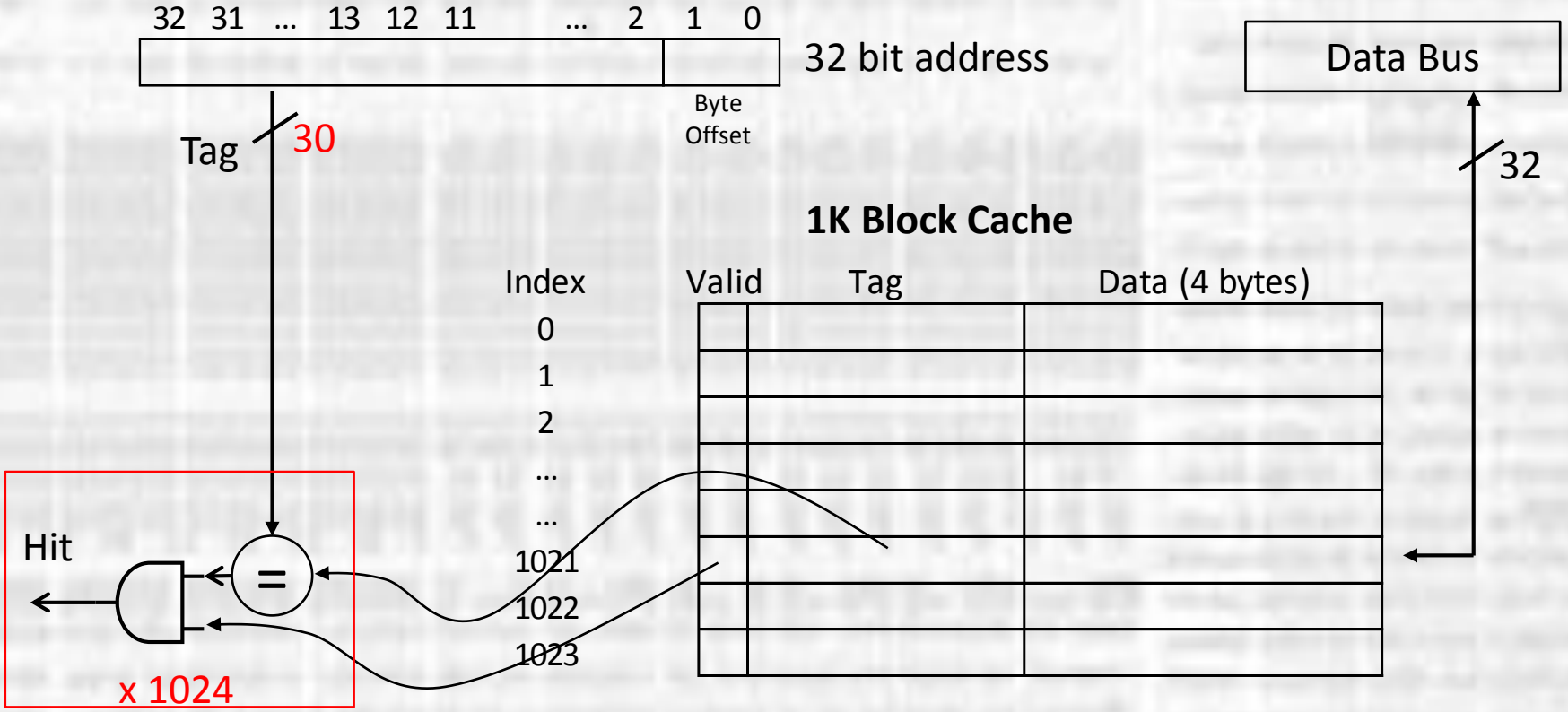
  but

  - Very inefficient
    - Increases total number of bits
    - Must search each tag field
      - Increases the amount of compare logic

Cache Performance

Set Associative Cache

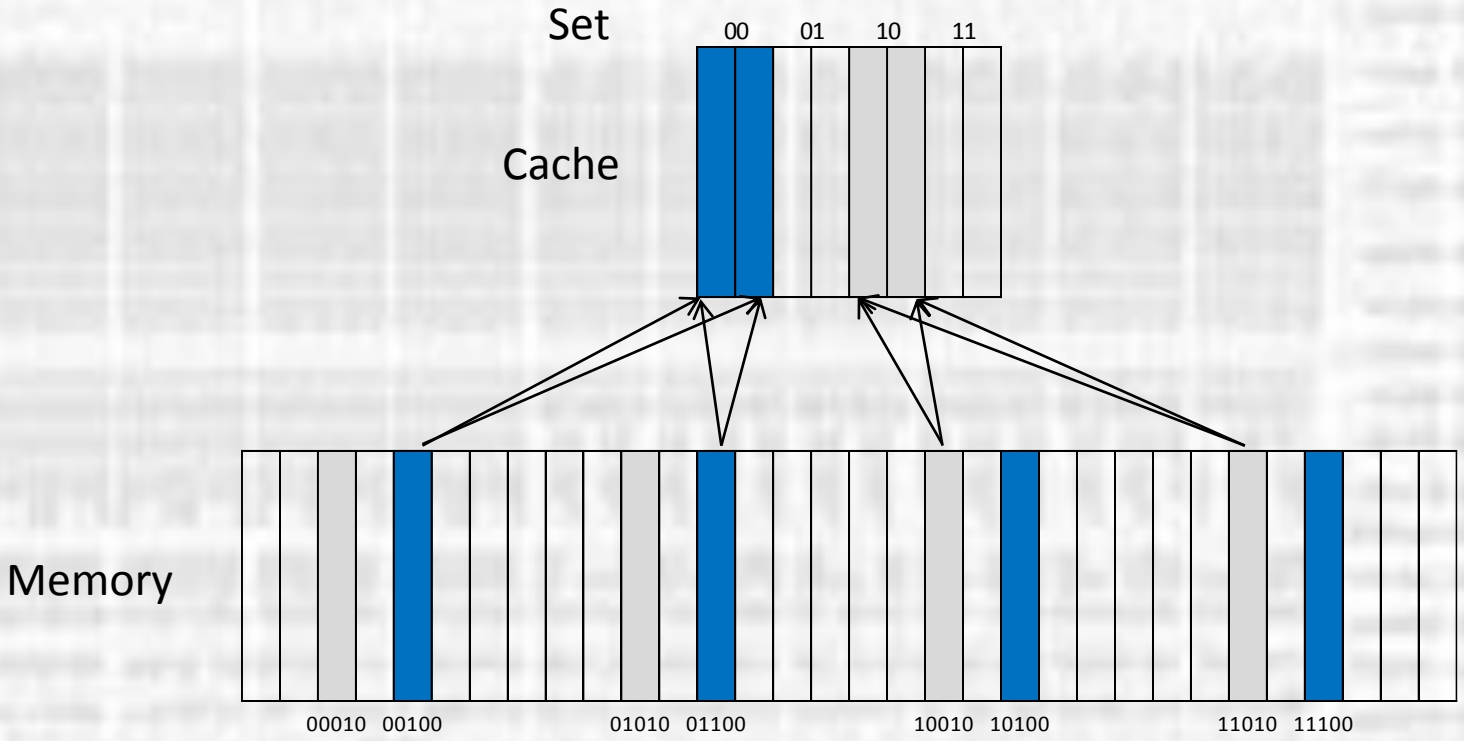
- Fully Associative Cache
  - 1K block cache, 32 bit word



Cache Performance

Set Associative Cache

- Set Associative Cache
  - Maps each memory location to a limited number of blocks



Cache Performance

Set Associative Cache

- Set Associative Cache
  - M block, N-way Set Associative Cache
    - N-way → each set consists of N blocks
    - M block → total number of blocks is M
  - 64 block, 2-way set associative cache
    - 32 sets of 2 blocks
    - Each memory location can be mapped to 2 blocks
    - There are 32 mapping groups

Cache Performance

Set Associative Cache

- Cache Comparison
  - 64 Block Cache
  - Direct Mapped
    - block location = (block number) modulo (# of blocks)
    - 1000 mod 64 = block 40
  - 2-way Set Associative
    - set location = (block number) modulo (# of sets)
    - 1000 mod 32 = set 8
  - Fully Associative
    - looks like a 64-way set associative cache → 1 set
    - 1000 mod 1 = set 0

Cache Performance

Set Associative Cache

- Cache Comparison
  - 8 Block Cache

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Cache Performance

Set Associative Cache

- Cache Comparison
  - 4 Block Cache – address sequence = 0,8,0,6,8
  - Direct Mapped

Block Address	Cache Block
0	0 mod 4 = 0
6	6 mod 4 = 2
8	8 mod 4 = 0

Address of memory block addressed	Hit or Miss	Contents of Cache after reference			
		0	1	2	3
0	miss	mem[0]			
8	miss	mem[8]			
0	miss	mem[0]			
6	miss	mem[0]		mem[6]	
8	miss	mem[8]		mem[6]	

5 accesses
5 misses

Cache Performance

Set Associative Cache

- Cache Comparison
  - 4 Block Cache – address sequence = 0,8,0,6,8
  - 2-way Set Associative

Block Address	Cache Block
0	0 mod 2 = 0
6	6 mod 2 = 0
8	8 mod 2 = 0

Address of memory block addressed	Hit or Miss	Contents of Cache after reference			
		Set 0		Set 1	
0	miss	mem[0]			
8	miss	mem[0]	mem[8]		
0	hit	mem[0]	mem[8]		
6	miss	mem[0]	mem[6]*		
8	miss	mem[8]*	mem[6]		

* least recently used block

5 accesses
4 misses

Cache Performance

Set Associative Cache

- Cache Comparison
  - 4 Block Cache – address sequence = 0,8,0,6,8
  - Fully Associative

Block Address	Cache Set
0	0 mod 1 = 0
6	6 mod 1 = 0
8	8 mod 1 = 0

Address of memory block addressed	Hit or Miss	Contents of Cache after reference			
		Set 0			
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]	

5 accesses
3 misses

Cache Performance

Set Associative Cache

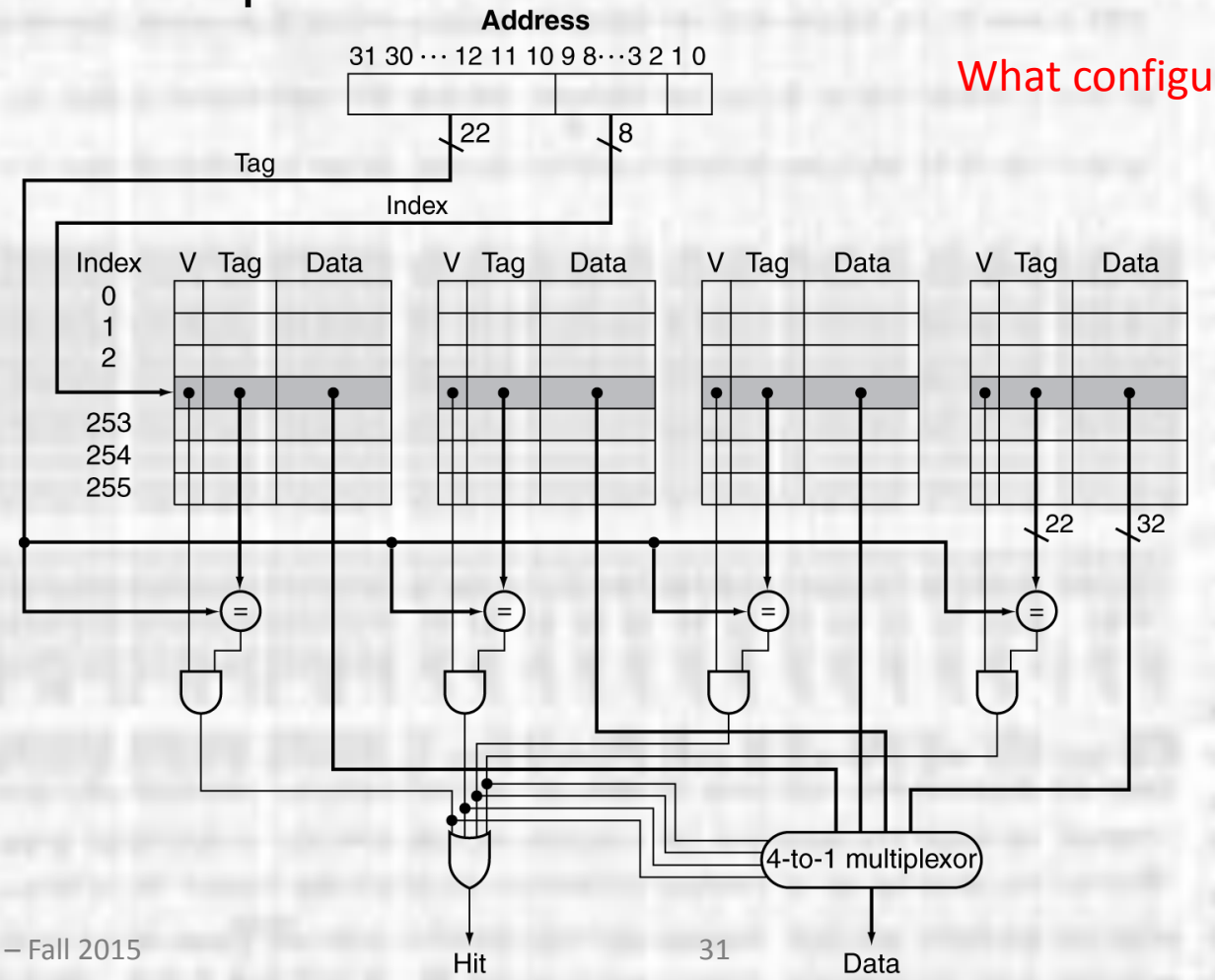
- Cache Comparison
  - As associativity increases:
    - Hit rate goes up
    - Complexity goes up
      - Cost
      - Usually leads to slow down
  - SPEC2000 benchmarks – 64KB Cache, 16 word block

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

Cache Performance

Set Associative Cache

- Cache Implementation

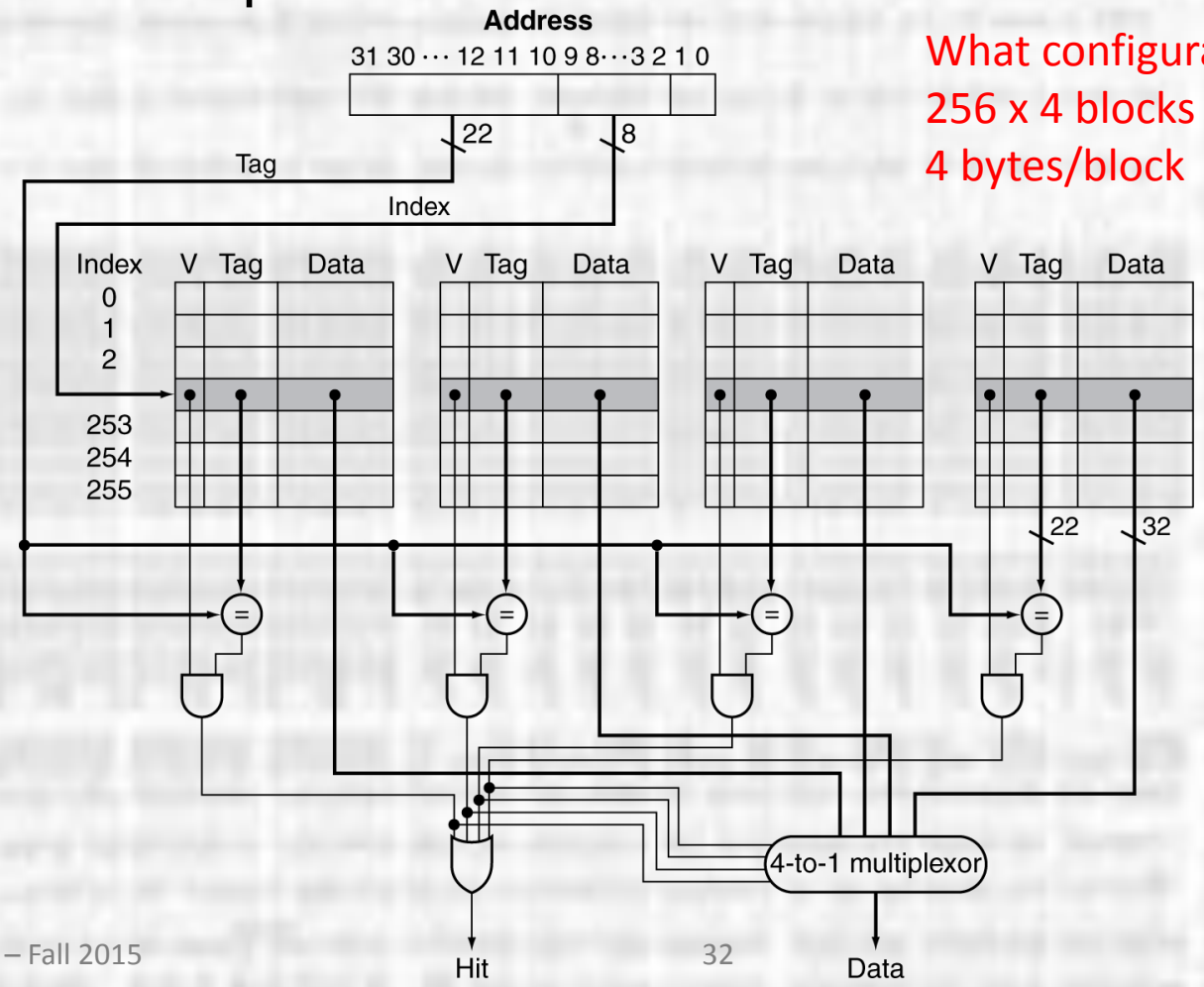


What configuration is this cache?

Cache Performance

Set Associative Cache

- Cache Implementation



What configuration is this cache?
256 x 4 blocks = 1K Block, 4 way
4 bytes/block → 4KByte, 4 way

Cache Performance

Set Associative Cache

- Replacement Policies
  - Set associativity introduces the need to choose which block to replace
    - Random
      - Implement pseudo-random block selection with-in a set
    - Least Recently Used (LRU)
      - Leverages temporal locality
    - First-in, first-out (FIFO)
      - Replace the oldest block
      - Simpler than LRU but frequently results in similar performance

Cache Performance

Set Associative Cache

- Replacement Policies
  - Data Cache Misses
    - 1000 instructions, SPEC2000, Alpha Architecture

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

src. Computer Architecture, Hennessy and Patterson, 5th ed.

Cache Performance

Set Associative Cache

- Performance Review

Data misses / 1000 instructions — Associativity

Size	Two-way LRU	Two-way Random	Two-way FIFO	Four-way LRU	Four-way Random	Four-way FIFO	Eight-way LRU	Eight-way Random	Eight-way FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

src. Computer Architecture, Hennessy and Patterson, 5th ed.

- Bigger cache → fewer misses
- LRU < FIFO < Random - but differences small
- Associativity reduces misses for smaller caches – but diminishing
- For large caches, associativity becomes less important

Cache Performance

Multi-level Caches

- Single level Cache Issues
  - Cache miss penalties are very high when a miss goes to main memory
    - Many stall cycles
  - Large caches are slower
    - Slowing down the processor

  → Multi-level Cache

Cache Performance

Multi-level Caches

- Multi-level Cache
  - 2 on chip Caches
    - Smaller – L1 cache
    - Larger – L2 cache
  - L1
    - Targeted at allowing the processor to run as fast as possible
    - Focus is on hits
      - Fewer ways
      - smaller blocks
  - L2
    - Targeted at reducing the number of main memory accesses
    - Focus is on misses
      - More ways
      - bigger blocks

Cache Performance

Multi-level Caches

- Multi-level Cache
  - Local Miss Rate
    - misses / access – for each cache level
    - Miss $rate_{L1}$, Miss $rate_{L2}$
  - Global Miss Rate
    - misses / processor accesses
    - Global miss $rate_{L1}$ = Local miss $rate_{L1}$
    - Global miss $rate_{L2}$ = Local miss $rate_{L1}$ x Local miss $rate_{L2}$

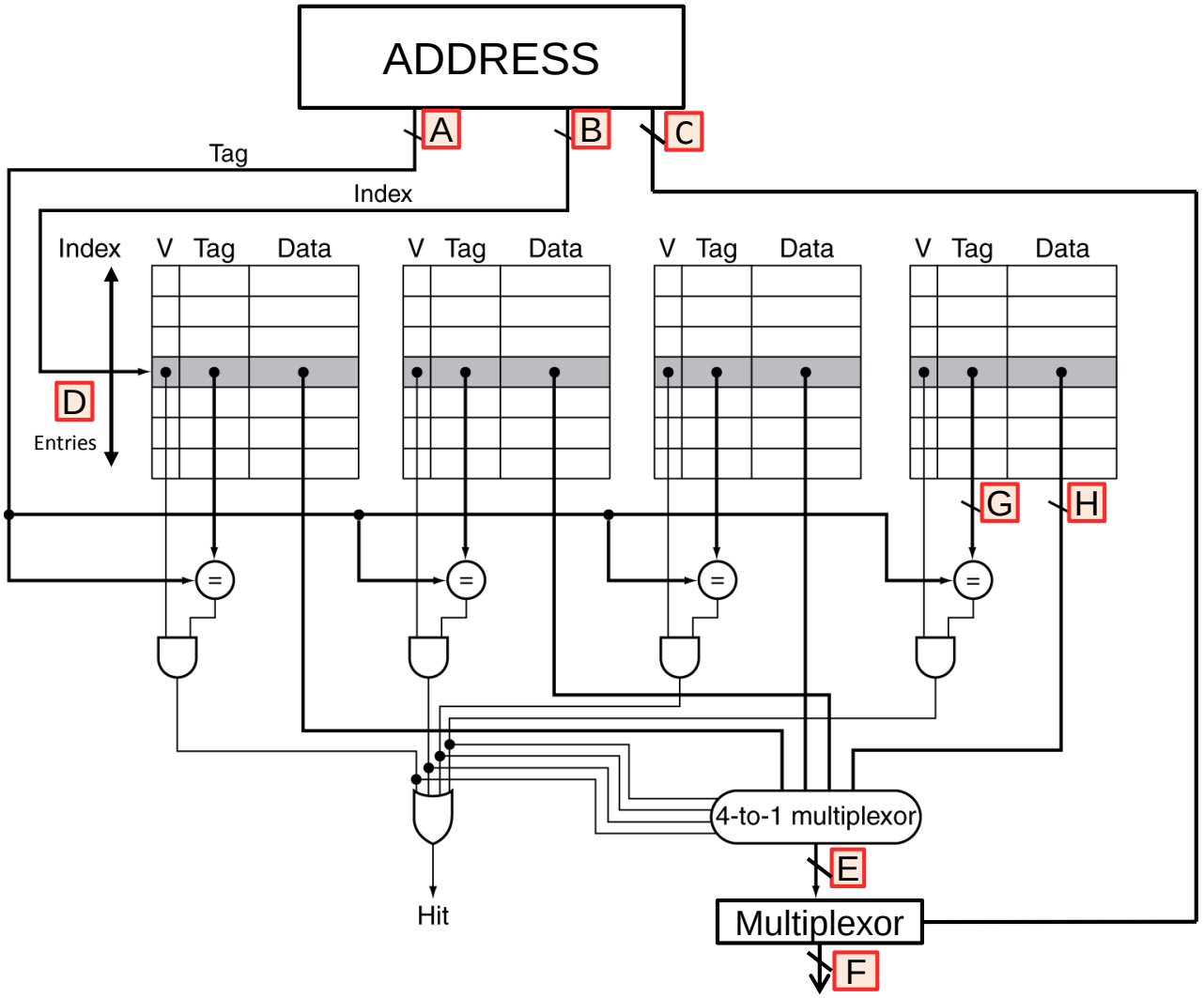
4) Given an 1K block direct mapped cache and a 20 bit address space, fill in the table below, given the following memory references (assume 32 bit words) - 15 pts

Memory Location (hex)	Data value (hex)
2B3B2	0x12212345
790F9	0x0110ABCD
93951	0xFFEE9876

use HEX for all values

Index	Valid	Tag	Data (4 bytes)
0			
...			
	1		
...			
	1		
...			
	1		
...			
0x3FF			

3) Given a 8kB 4-way set associative cache with 16 bit words, 16 word blocks, and 32 bit addressing – fill in the table referencing each letter. - 15 pts



A	
B	
C	
D	
E	
F	
G	
H	

6) Calculate the effective CPI given the following: - 15 pts

$CPI_{ideal} = 1$
1.5% instruction miss rate
75 cycle instruction miss penalty
3% data miss rate
90 cycle data miss penalty
15% of instructions are loads
15% of instructions are stores
15% of instructions are add immediate