ELE 455/555
Computer System Engineering

Section 3 – Memory

Class 2 – Cache Operation

Cache Basics

Memory Hierarchy

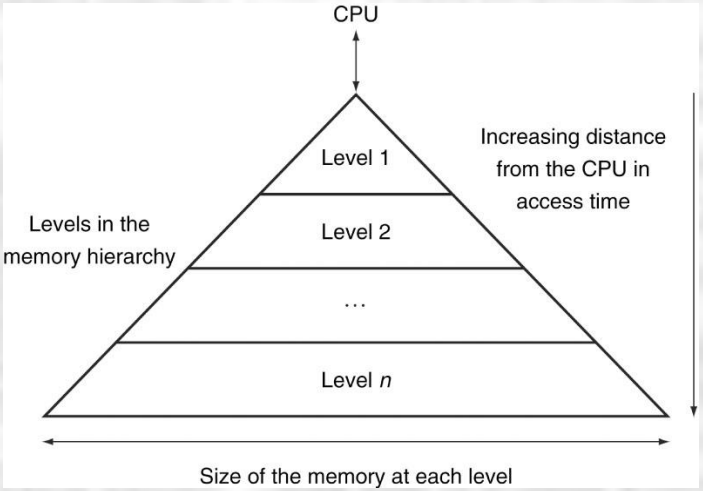
- Memory Hierarchy Considerations
  - Typical System

    Registers

    Cache (SRAM)

    Main Memory (DRAM)

    Storage (HDD or Flash)



  - Advanced systems may have 2,3,4 levels of cache
    - Each is progressively slower and larger
    - Size is targeted at holding entire applications

Cache Basics

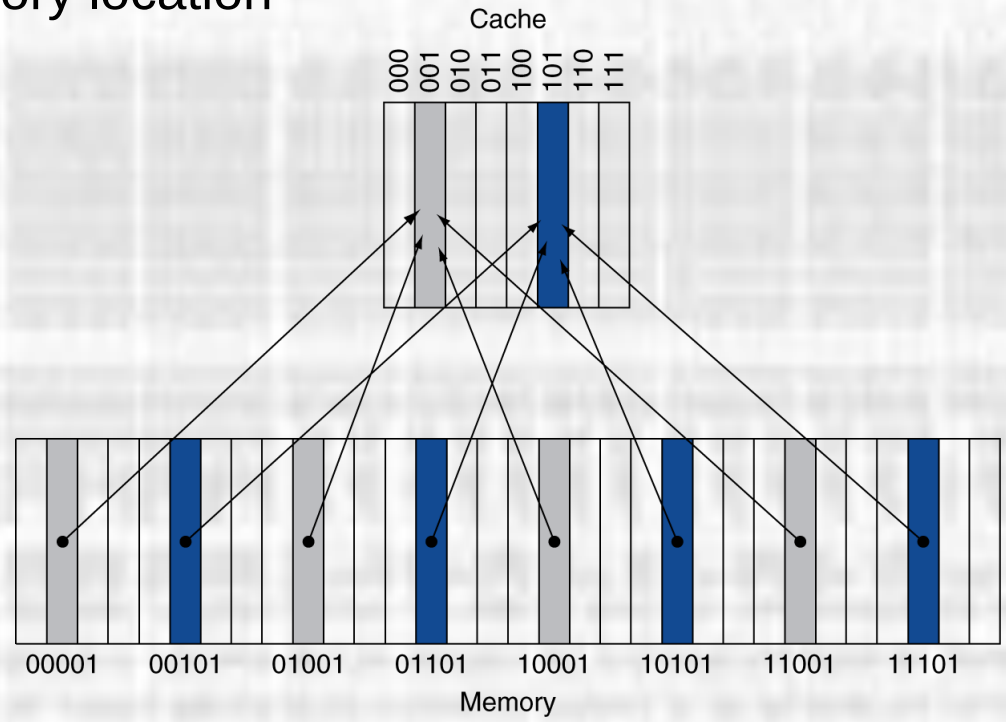
Cache Memory

- Cache Overview
  - Closest memory to the CPU
  - SRAM
    - Fast
    - Not too large (Kbytes)
  - Must MAP a larger address space into a small memory
    - Direct Mapped
    - Set Associative

Cache Basics

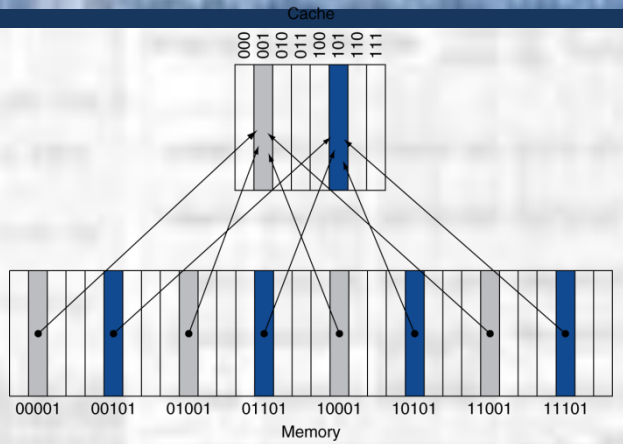
Cache Memory

- 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?

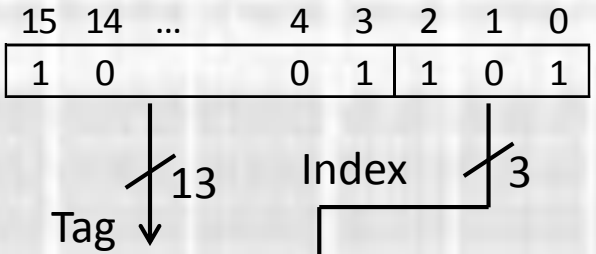
      0x2A3F mod 256_{10} = 0x3F = 63_{10}
  - As long as we follow this convention (cache size = 2^n)
    - **Cache block address = last n bits of the memory address***

*** for 1 byte block sizes**

Cache Basics

Cache Memory

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



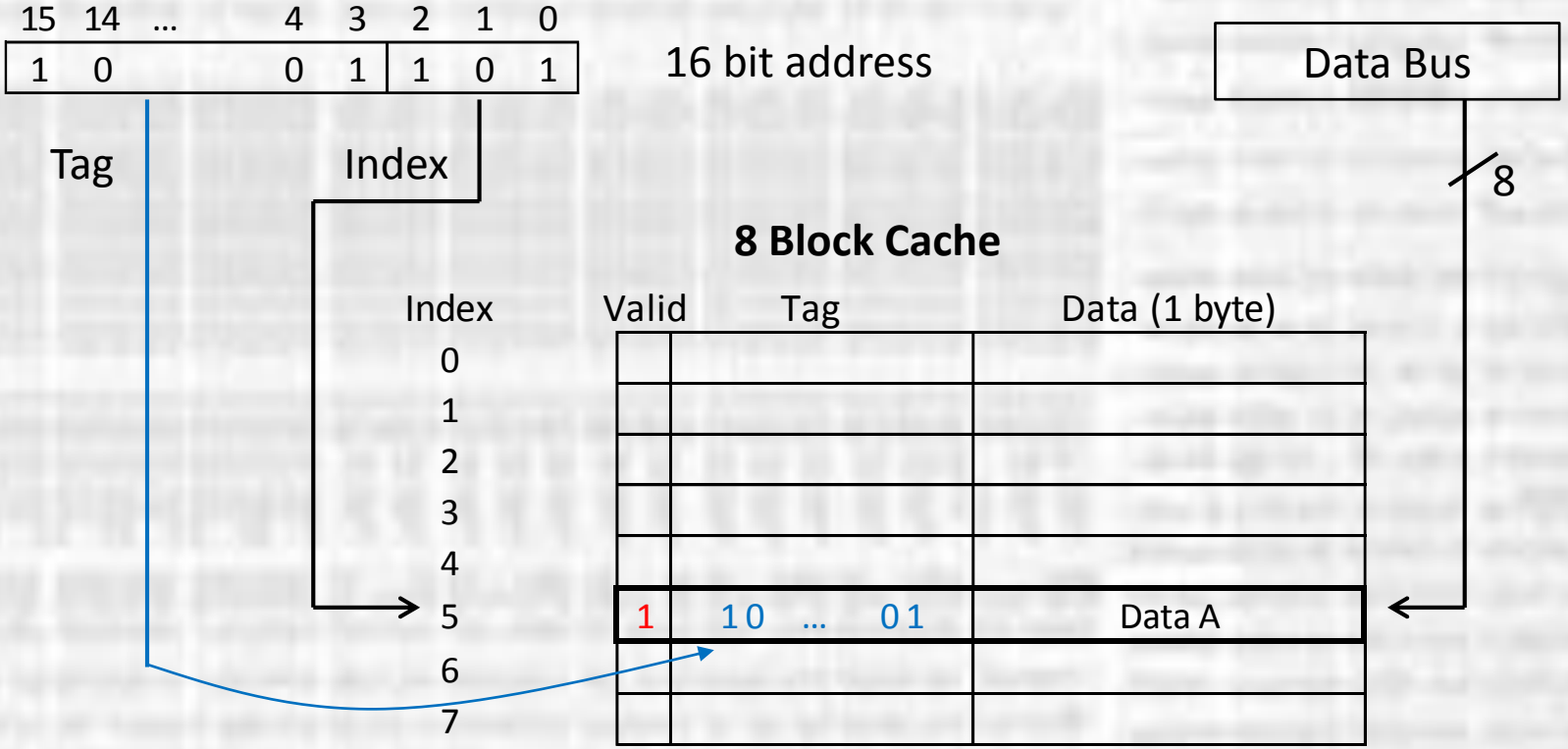
8 Block Cache

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

Cache Basics

Cache Memory

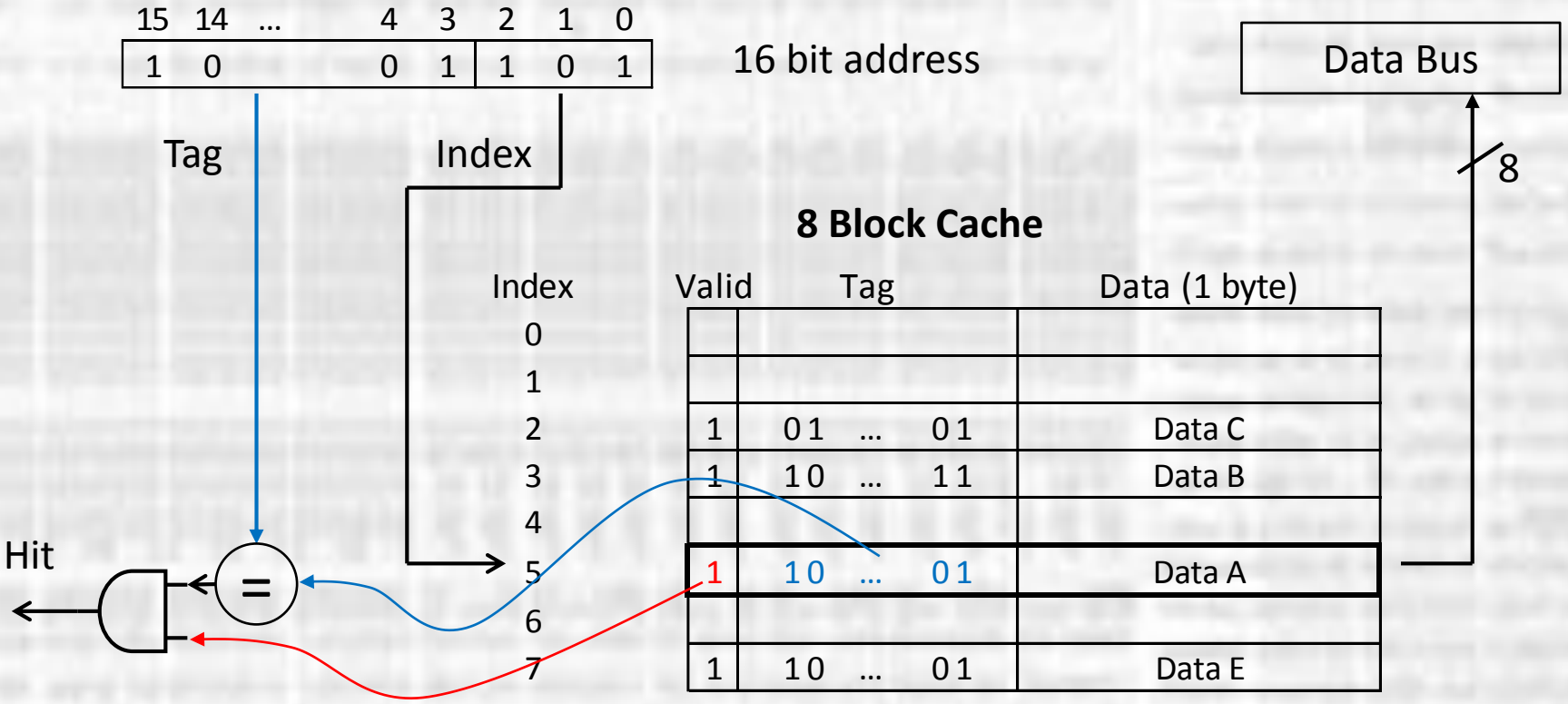
- Direct Mapped Cache
  - 8 block cache - Write



Cache Basics

Cache Memory

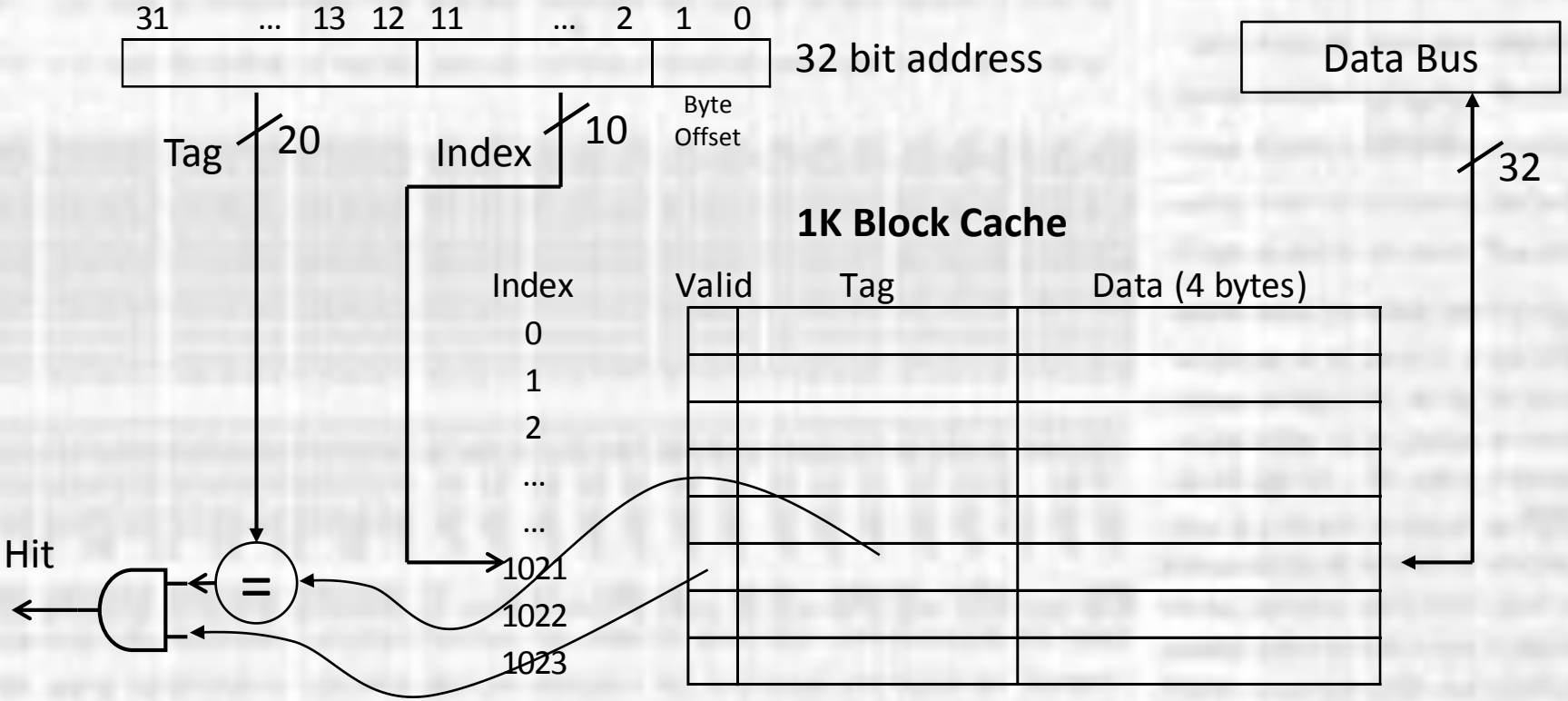
- Direct Mapped Cache
  - 8 block cache - Read



Cache Basics

Cache Memory

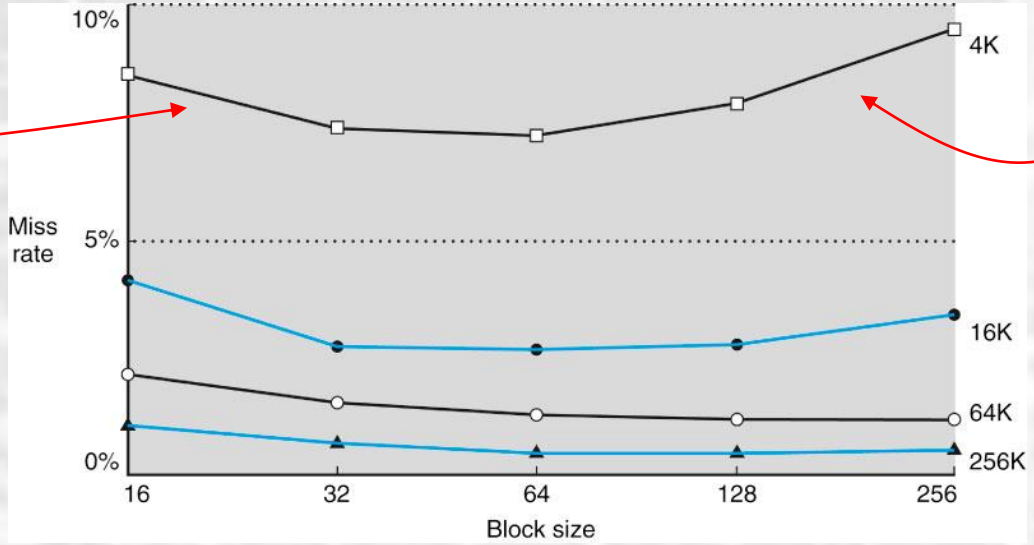
- Direct Mapped Cache
  - 1K block cache, 1 word block, 32 bit data word, 32 bit address space



Cache Basics

Cache Memory

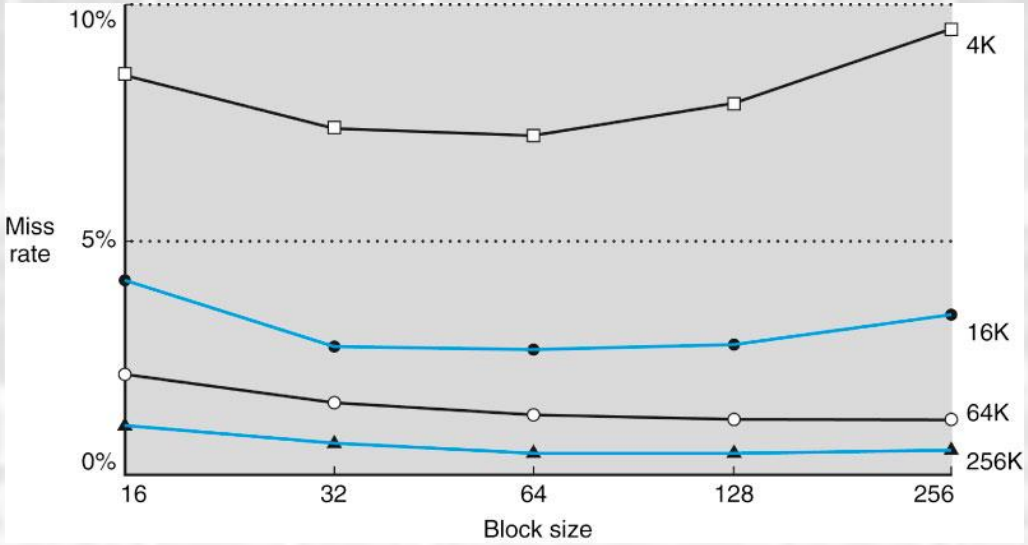
- Direct Mapped Cache
  - Large block sizes → reduced miss rate
    - Due to spatial locality
  - Assuming a fixed cache size
    - Larger blocks → fewer blocks → higher miss rate



Cache Basics

Cache Memory

- Direct Mapped Cache
  - Larger Cache → reduced miss rate
    → slower access



Cache Basics

Cache Read

- Cache Read Miss - Program Memory
  - On a miss we do not have the requested program memory value available (current instruction)
  - In the mean time the PC has incremented (+4 for MIPS)
  - We must stall the processor while we wait for the instruction

Cache Basics

Cache Read

- Cache Read Miss - Program Memory
  - Actually have 2 control circuits (controllers)
    - Processor controller
    - Memory controller
      - Separate due to timing and latencies associated with the memory
  - Processor control will stall the processor
    - Wait for a signal to restart
  - Memory controller
    - Sends the original program memory address to memory with a read request (current PC - 4)
    - When available: write data, tag, and valid bit in cache
    - Signal the processor to restart at the fetch stage

Cache Basics

Cache Read

- Cache Read Miss – Data Memory
  - On a miss we do not have the requested data memory value available (cannot complete the instruction - Load)
  - We must stall the processor while we wait for the data

Cache Basics

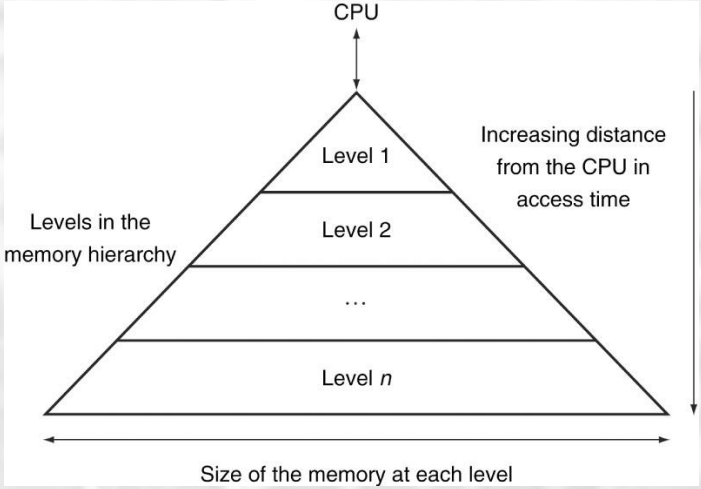
Cache Read

- Cache Read Miss - Data Memory
  - Actually have 2 control circuits (controllers)
    - Processor controller
    - Memory controller
      - Separate due to timing and latencies associated with the memory
  - Processor Control will stall the processor
    - Wait for a signal to restart
  - Memory controller
    - Sends the original data memory address to memory with a read request
    - When available: write data, tag, and valid bit in cache
    - Signal the Processor to restart with the memory read

Cache Basics

Cache Write

- Memory Consistency
  - Our memory hierarchy needs to remain consistent
    - All levels must contain the same value for a given memory location
    - If not – which is right?
    - Not a problem for reads
    - Can be a problem for writes



Cache Basics

Cache Write

- Write-through
  - Simple approach to ensure memory consistency
  - Every write to the cache → write to main memory
  - Write Miss
    - The desired memory value is not in the cache
    - Read the desired memory value from main memory
    - Write it into the cache
    - Modify it (since this was started with a write instruction to begin with)
    - Write a copy back to main memory

Cache Basics

Cache Write

- Write-through
  - Simple approach – but very inefficient
  - Every write to the cache → write to main memory
  - Main memory writes are very slow (why we have a hierarchy)
  - Example
    - Main memory clock cycles/write = 100
    - 1% of instructions are stores
    - No-cache CPI = 1

    1% of instructions will take 100 clock cycles

    New CPI = 1 + 1 = 2 clocks/instruction

    All that work to reduce the CPI has been foiled!

Cache Basics

Cache Write

- Write-through
  - Partial fix
  - Create a write buffer
    - Holds the write value while the processor goes on to the next instruction
    - Works as long as the rate of writes from the processor does not exceed the buffers capability to store data
  - Example – 1 word write buffer
    - Main memory clock cycles/write = 100
    - 1% of instructions are stores
    - No-cache CPI = 1

    1% of instructions will take 100 clock cycles – but will be stored in the write buffer – allowing the CPU to continue

    CPI = 1

Cache Basics

Cache Write

- Write-through
  - Partial fix – complication
  - Even though the overall % of writes is small enough to allow the write buffer to work – If the writes are “bursty” – the buffer will be overwhelmed and the processor will stall
  - Leads to deep write buffers

Cache Basics

Cache Write

- Write-through
  - Alternative approaches within the write-through process
  - Allocate a specific block for write misses in the cache
    - On miss – read the value from main memory and place it in a special block of the cache
    - Called - write allocate
    - Make changes
    - Write the value back into memory
    - Assumes you likely will not reuse the value soon
  - Don’t load the block into cache at all
    - Write the value directly to memory
    - Also assumes you will not reuse the block soon
    - Typical for zeroing out a page of memory
    - Called - no write allocate

Cache Basics

Cache Write

- Write-Back
  - Alternative to write-through
  - Only write back to main memory when the cache block is being replaced
    - And only when it is “dirty”, i.e. been changed
  - Provides a similar performance advantage as the cache read process
    - 10% of instructions are writes but only 10% are cache misses, leading to a write-back rate of 1%

Cache Basics

Cache Write

- Write-back vs. Write-through
  - Write-through
    - Can write to the cache and determine if there is a miss at the same time
      - If hit – write is OK
      - If miss – no harm since the value over-written has already been stored in memory
        - Process moves forward as usual – but only replacing the parts of the block that were not just overwritten
      - All writes can occur in 1 clock cycle
  - Write-back
    - Must write the block back to memory on a miss (and dirty)
      - 2 clock cycles: one to determine hit or miss, one to initiate write back on misses
      - Or use a write buffer to pipeline the process → 1 clock cycle
      - Or use a store buffer to hold the stored value while the write-back occurs then updates the cache on the next available cache write cycle

Cache Basics

Cache Example

- Intrinsity FastMATH Processor
  - Embedded MIPS processor
    - 12-stage pipeline
    - Instruction and data access on each cycle
  - Split cache: separate I-cache and D-cache
    - Each 16KB: 256 blocks × 16 words/block
    - D-cache: write-through or write-back
  - SPEC2000 miss rates
    - I-cache: 0.4%
    - D-cache: 11.4%
    - Weighted average: 3.2%

Cache Basics

Cache Example

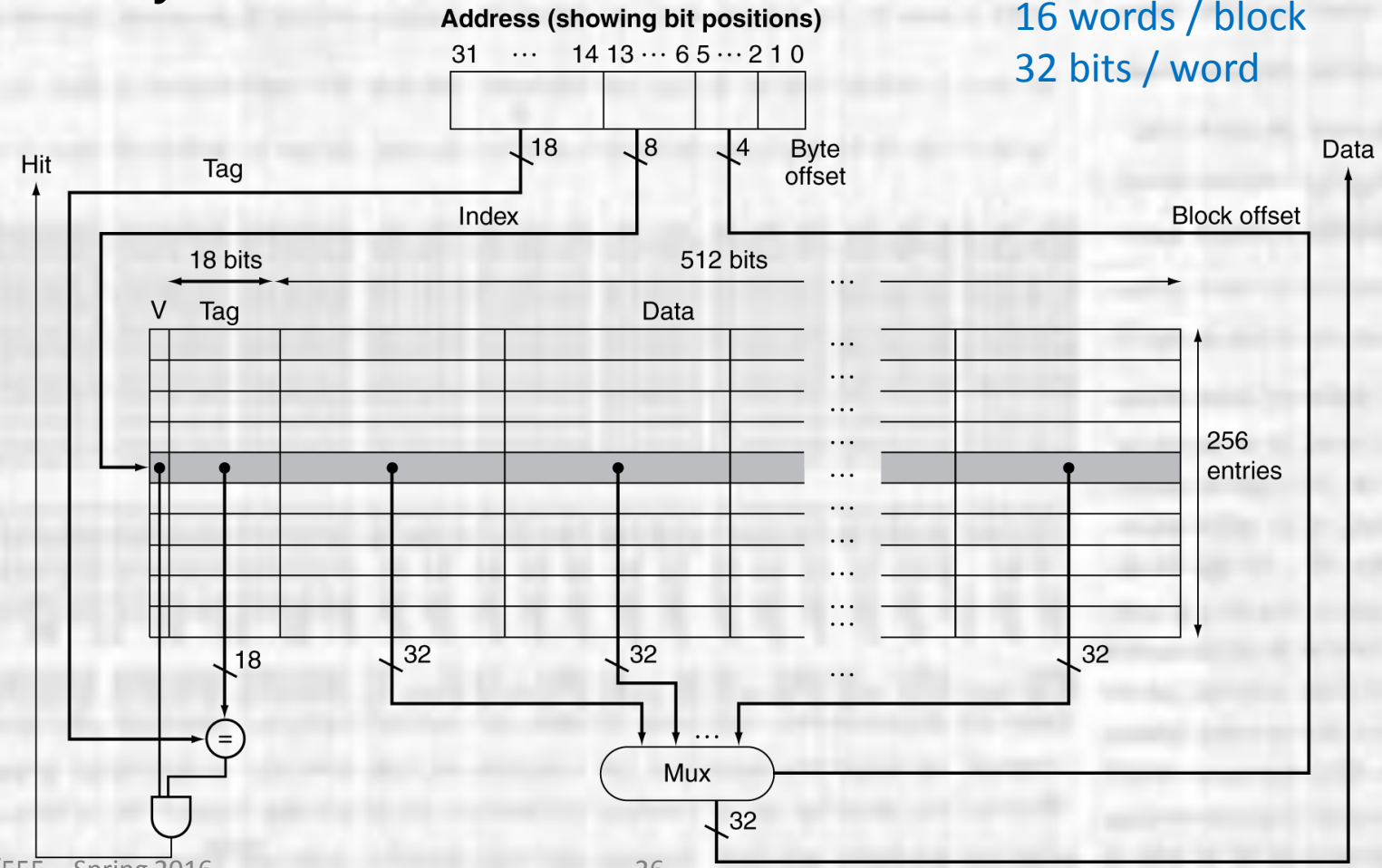
- Intrinsity FastMATH Processor
  - Read
    - Send requested address to cache
      - Instruction or data
    - Hit
      - Data available
      - Must select from the 16 possible words in the block
      - Block offset field is used – addr[5:2]
    - Miss
      - Send address to main memory
      - When available – update the cache
      - Continue normal operation – read value from cache
  - Write
    - Has both write-through and write back options
    - OS determines which to use
      - based on instruction usage by each application

Cache Basics

Cache Example

- Intrinsity FastMATH Processor

256 Blocks
16 words / block
32 bits / word



Cache Basics

Cache Configuration

- Split vs. Single Cache
  - Single cache to support I and D
    - Larger (same as 2 together) → better hit rate
      - Allows more flexibility for how much is data and how much is instruction
        - consider a small program operating on a lot of data vs. a big program using almost no data

Cache Size	Split Cache Miss Rate	Combined Cache Miss Rate
32KB	3.24%	3.18%

  - Split I and D cache
    - Allows for concurrent I and D access – 2x bandwidth
    - Far outweighs the flexibility advantage of a combined cache