

Processor Architecture

Caches

Last modified 4/20/20

Cache Basics

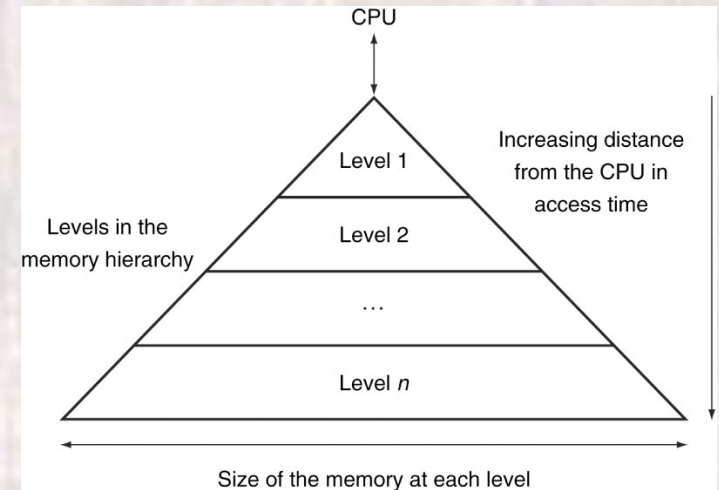
- Memory Hierarchy Considerations
 - Typical System

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

- Memory Hierarchy Considerations

- Typical System - 2GHz

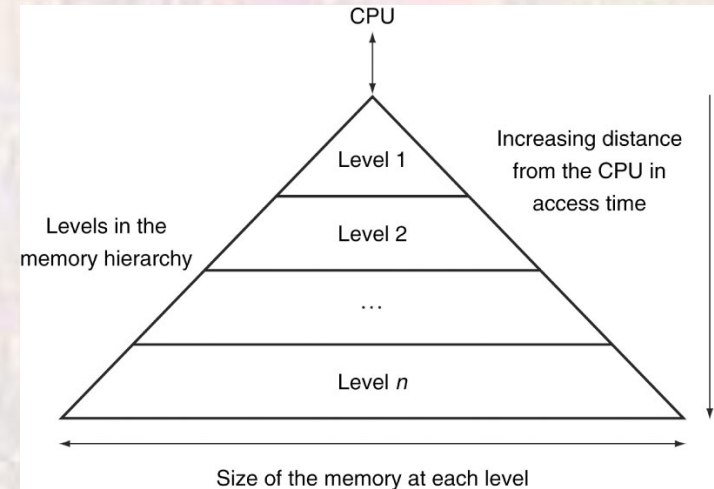
Registers - 1 Clk access

Cache (SRAM) – 1 – 10 Clk access

Main Memory (DRAM) – 50 Clk access

Storage (HDD or Flash) – 10,000 Clk access

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

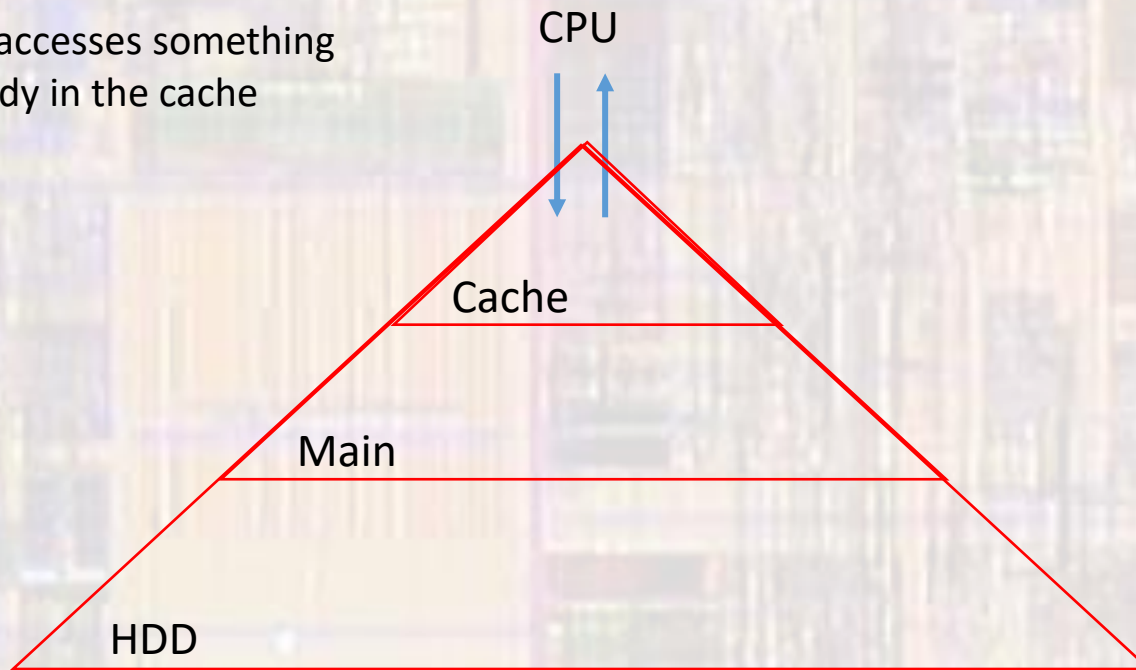


Cache Basics

- Memory Hierarchy Considerations

CPU accesses something
already in the cache

1 clk

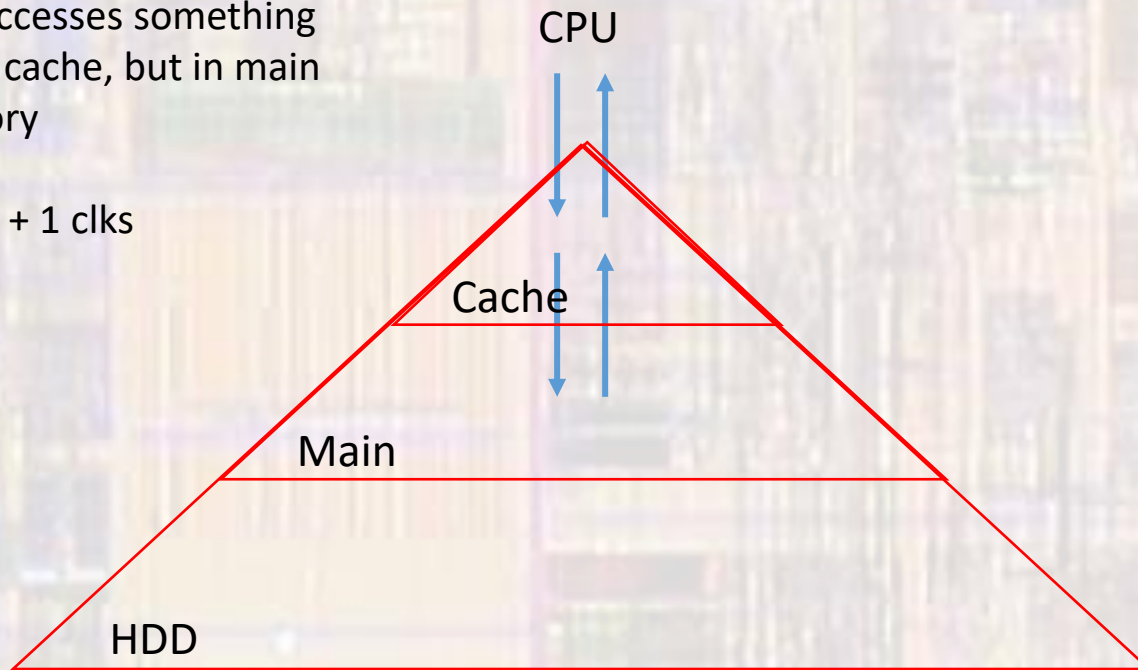


Cache Basics

- Memory Hierarchy Considerations

CPU accesses something
not in cache, but in main
memory

1 + 50 + 1 clks

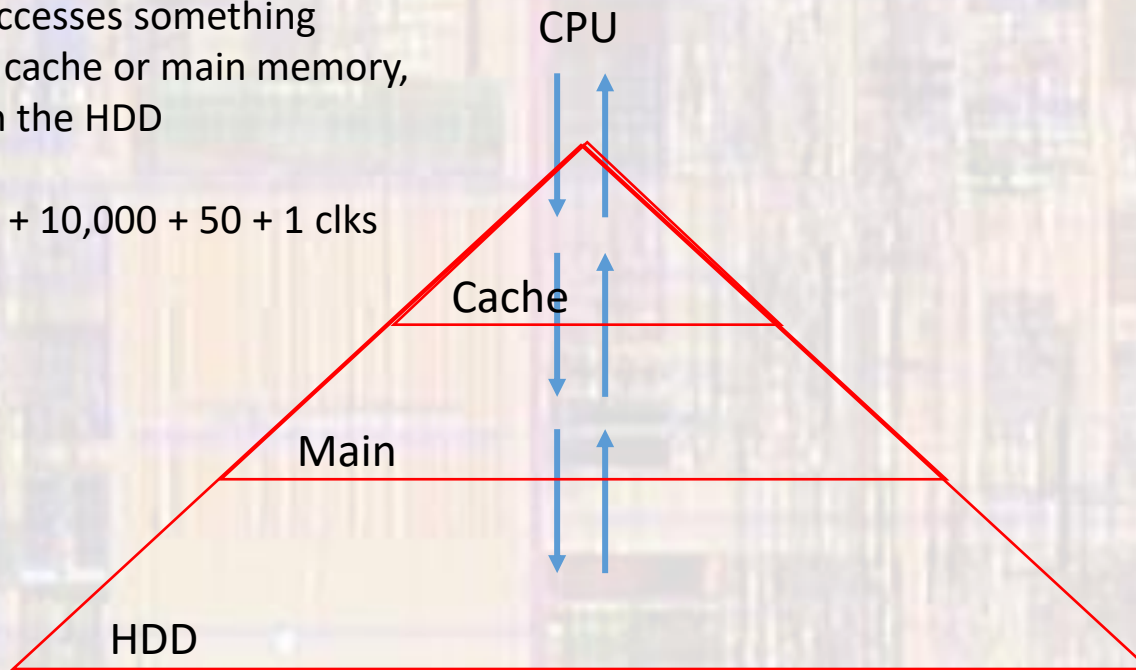


Cache Basics

- Memory Hierarchy Considerations

CPU accesses something
not in cache or main memory,
but on the HDD

$1 + 50 + 10,000 + 50 + 1$ clks



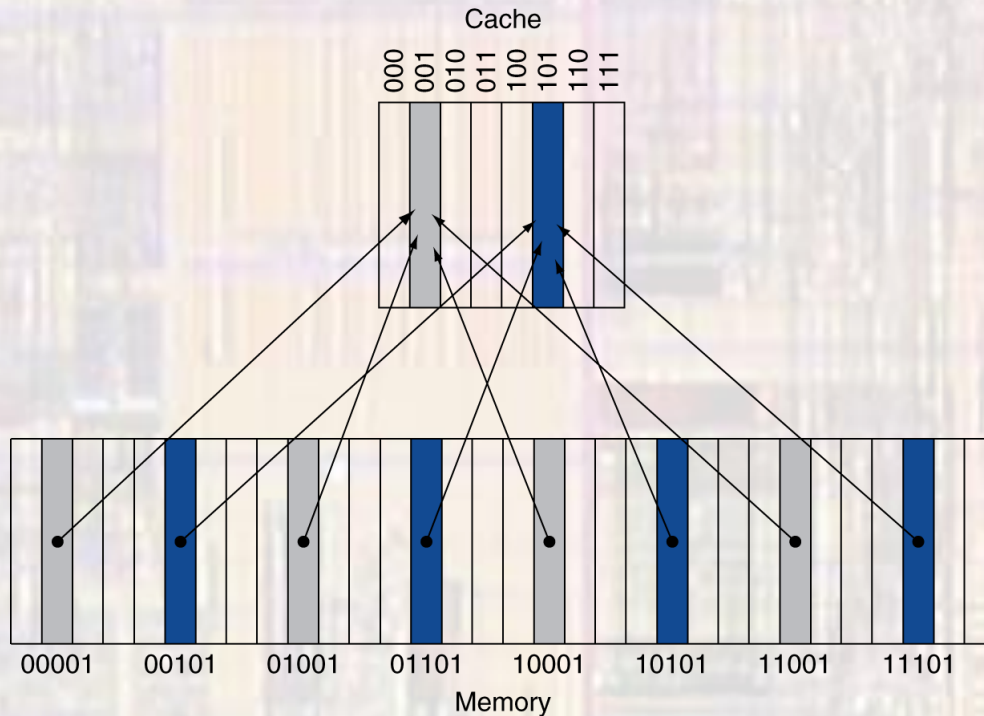
Cache Basics

- 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

- Direct Mapped Cache

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



Cache Basics

- Direct Mapped Cache

- Cache size is built to be a power of 2

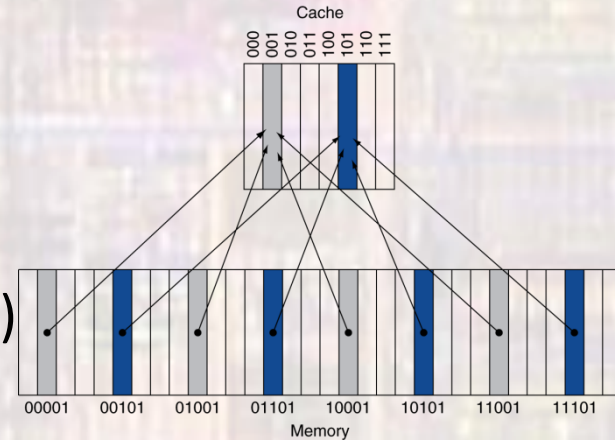
- Cache block =
(Block Address) mod (# of cache blocks)

- Eg. Assume a 256 block cache
Where does the memory block from address 0x2A3F map to?

$$0x2A3F \bmod 256_{10} = 0x3F = 63_{10}$$

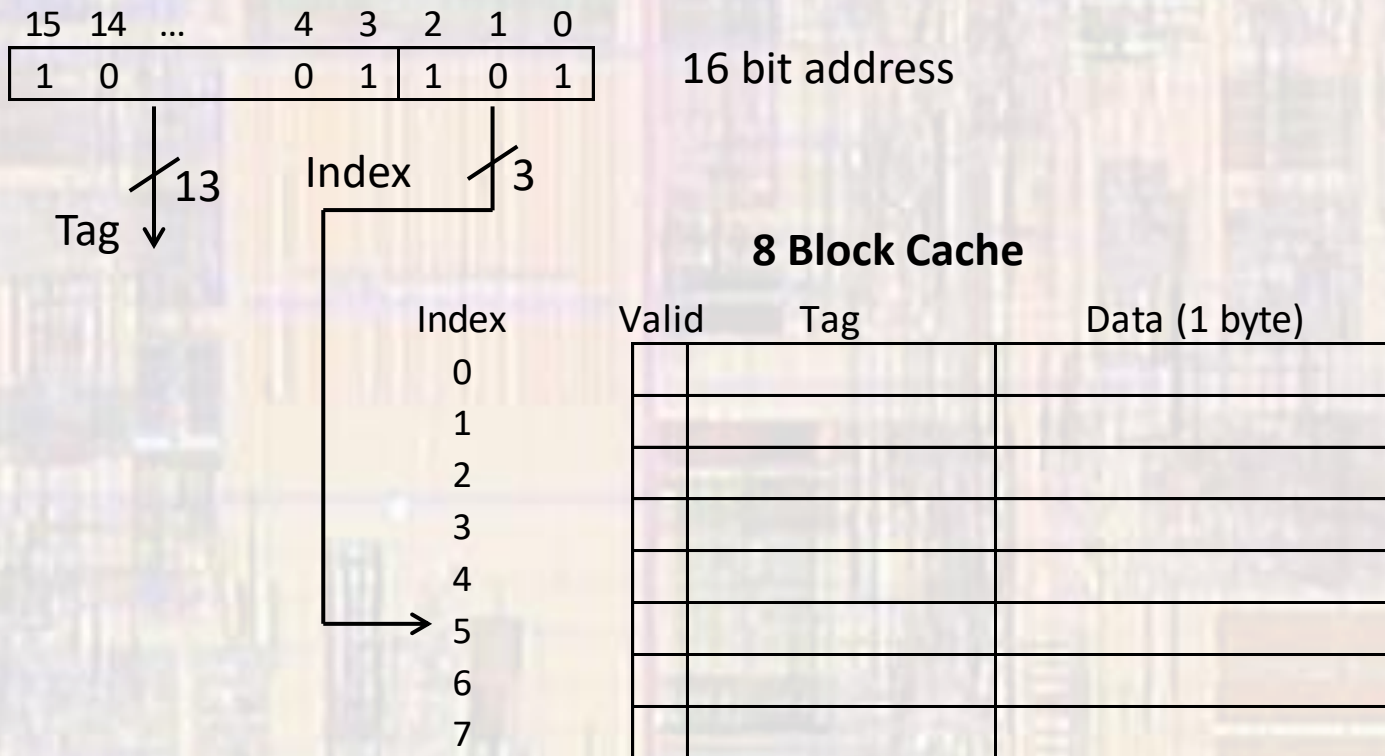
- As long as we follow this convention (cache size = 2^n)
 - **Cache block address = last n bits of the memory address***

* for 1 byte block sizes



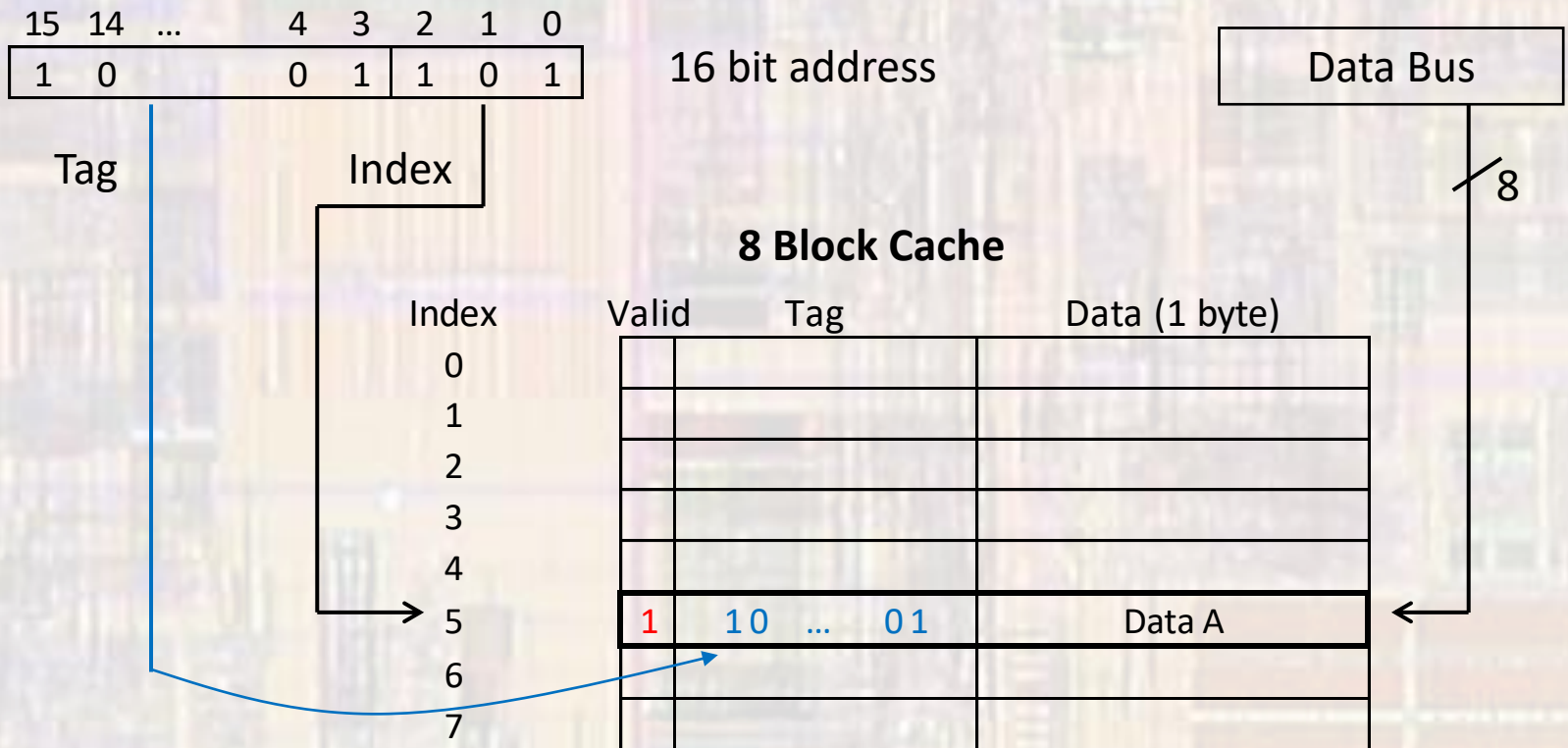
Cache Basics

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



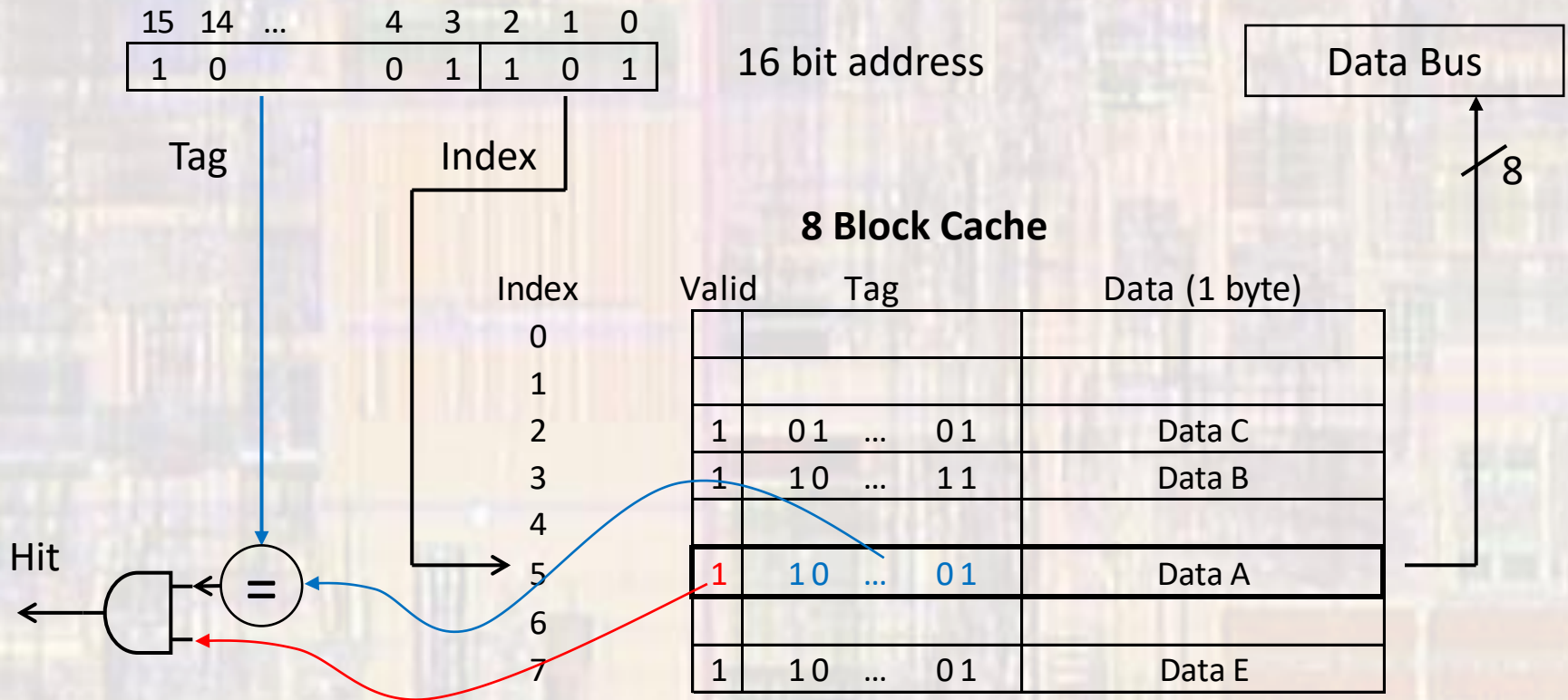
Cache Basics

- Direct Mapped Cache
 - 8 block cache - Write



Cache Basics

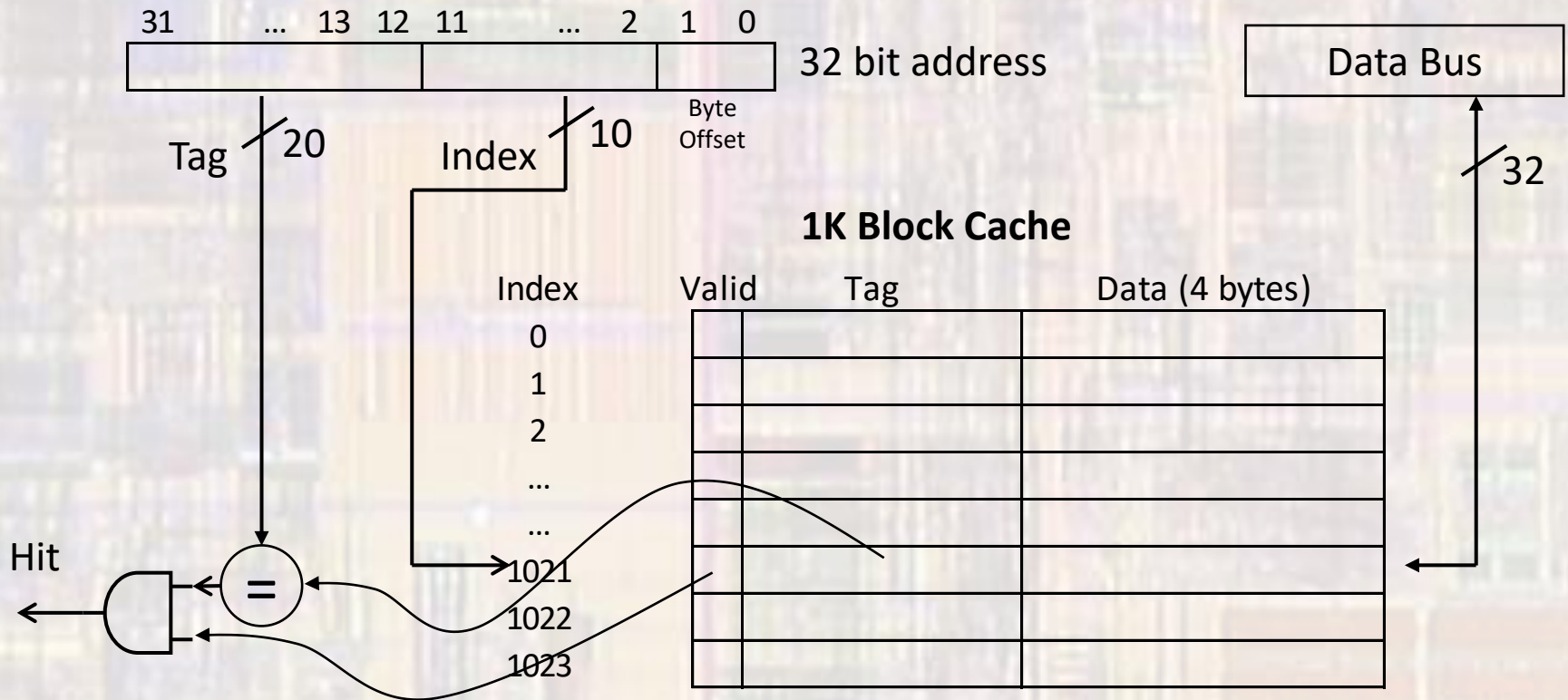
- Direct Mapped Cache
 - 8 block cache - Read



Cache Basics

- Direct Mapped Cache

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



Cache Basics

- 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 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

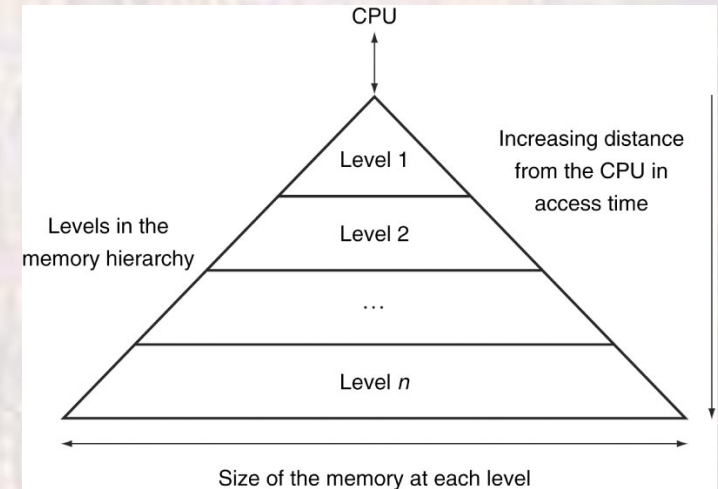
- Cache Read Miss – Data Memory
 - On a miss we do not have the requested data memory value available (cannot complete the instruction - Load)
 - We must stall the processor while we wait for the data

Cache Basics

- Cache Read 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

- 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

- 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

- 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

- 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

- 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

- 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

Cache Performance

- 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

- CPU performance

- Read Stall Cycles

- Stalls due to read misses

- Read stall cycles = $\frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$

Cache Performance

- CPU performance

- Write Stall Cycles (write through)

- Stalls due to write misses
and

- Write buffer stalls (buffer full)

- 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

- Write stall cycles = $\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}$

Cache Performance

- 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
 - Memory stall cycles = $\frac{\text{Memory Accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$

or

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

Cache Performance

- CPU performance - example

$$\text{CPI}_{\text{ideal}} = 2$$

2% instruction miss rate

4% data miss rate

100 cycle miss penalty

36% of instructions are Loads or Stores

$$\begin{aligned}\text{Instruction Miss Cycles} &= \text{Icount} \times 2\% \text{miss/inst} \times 100 \text{cycles/miss} \\ &= 2 \times \text{Icount}\end{aligned}$$

$$\begin{aligned}\text{Data Miss Cycles} &= \text{Icount} \times 36\% \text{LS/inst} \times 4\% \text{miss/LS} \times \\ &\quad 100 \text{cycles/miss} \\ &= 1.44 \times \text{Icount}\end{aligned}$$

Cache Performance

- CPU performance – example cont'd

$$\text{Memory Stall Cycles} = 2 \text{ Icount} + 1.44 \text{ Icount} = 3.44 \text{ Icount}$$

This is almost 3.5 stalls per instruction !!!

$$\text{CPI} = \text{CPI}_{\text{ideal}} + 3.44 \text{ clocks/inst} = 5.44 \text{ clocks/inst}$$

Only achieving 37% of the ideal performance

Cache Performance

- CPU performance – example cont'd

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

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

This improves the performance – but not linearly

Only achieving 22.5% of the ideal performance

Cache Performance

- 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
 - $AMAT = \text{Time for a hit} + (\text{Miss Rate} \times \text{Miss penalty})$ seconds
or
 $AMAT = \text{Clock cycle time} \times (\text{Hit Cycles} + \text{Miss Rate} \times \text{Miss Penalty})$

Cache Performance

- CPU performance - example

1GHz clock

1 cycle cache access time

5% miss rate

20 cycle miss penalty

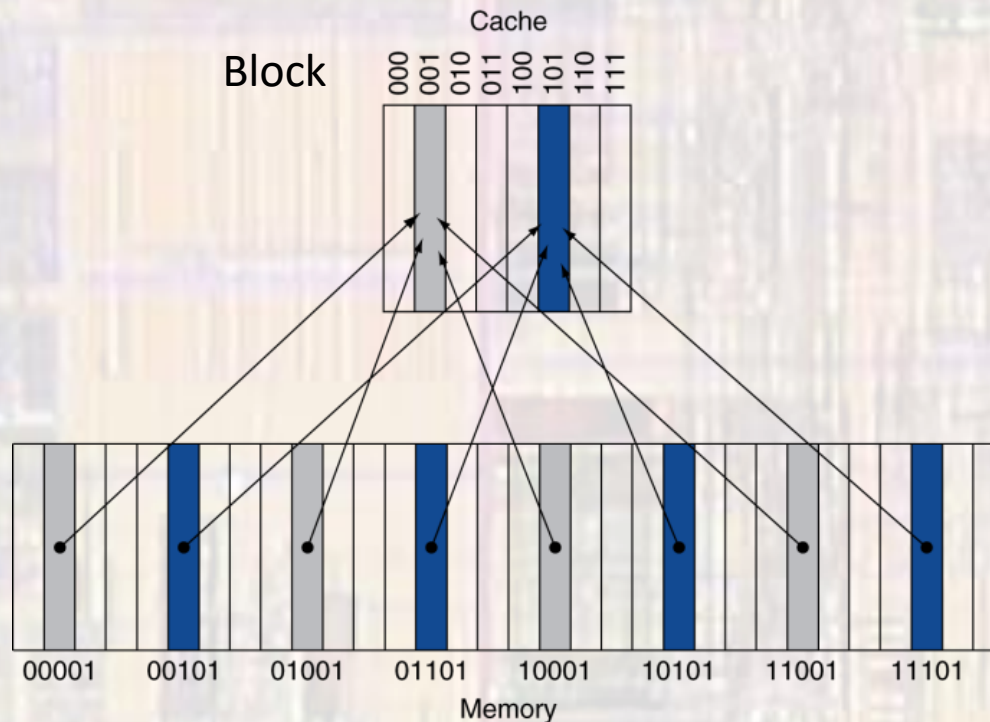
$$\text{AMAT} = 1\text{ns}/\text{clk} \times (1 \text{ clk}/\text{hit} + 5\% \times 20\text{clk}/\text{miss}) = 2\text{ns}$$

Cache Performance

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

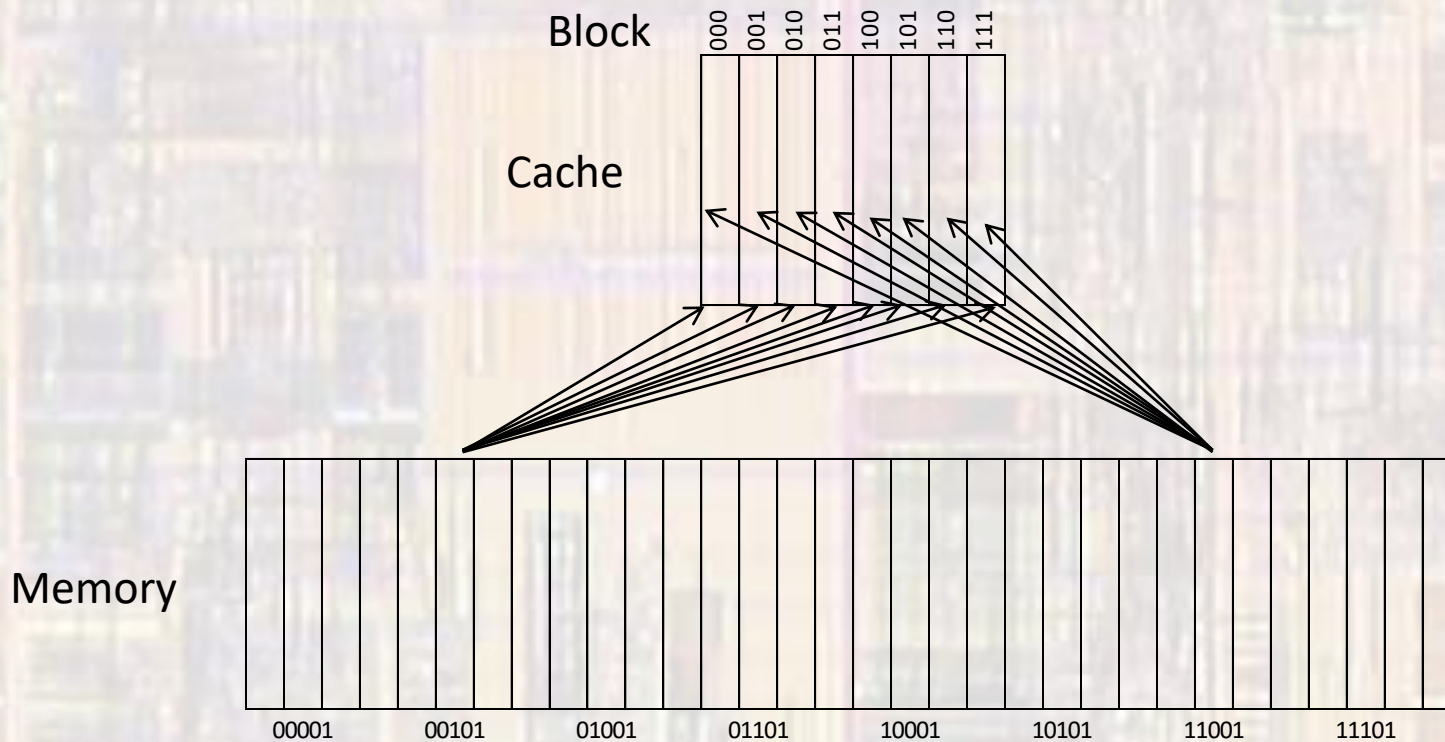
Cache Performance

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



Cache Performance

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



Cache Performance

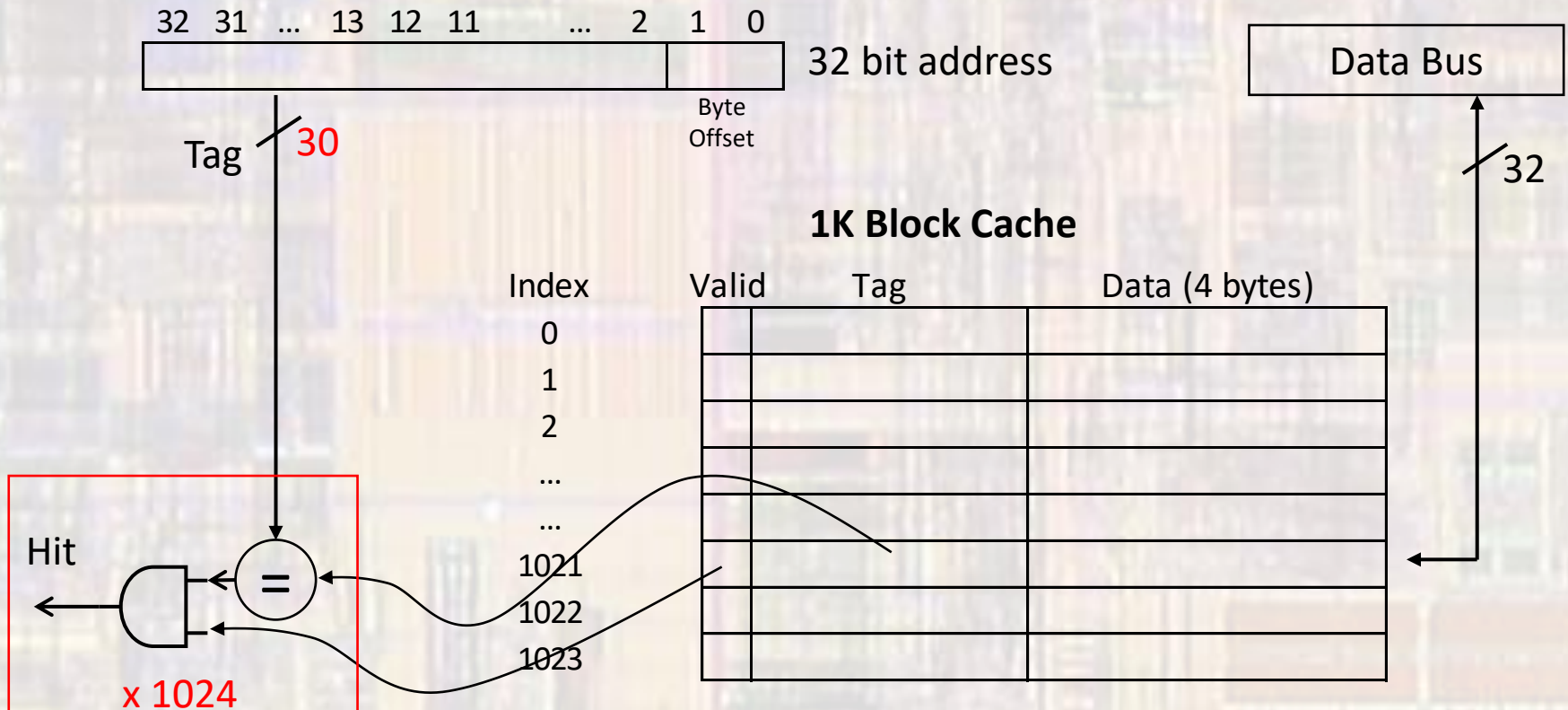
- Fully Associative Cache
 - Maps each memory location to any cache block
 - Reduces the number of mapping conflicts
 - Reduces the number of Misses

but

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

Cache Performance

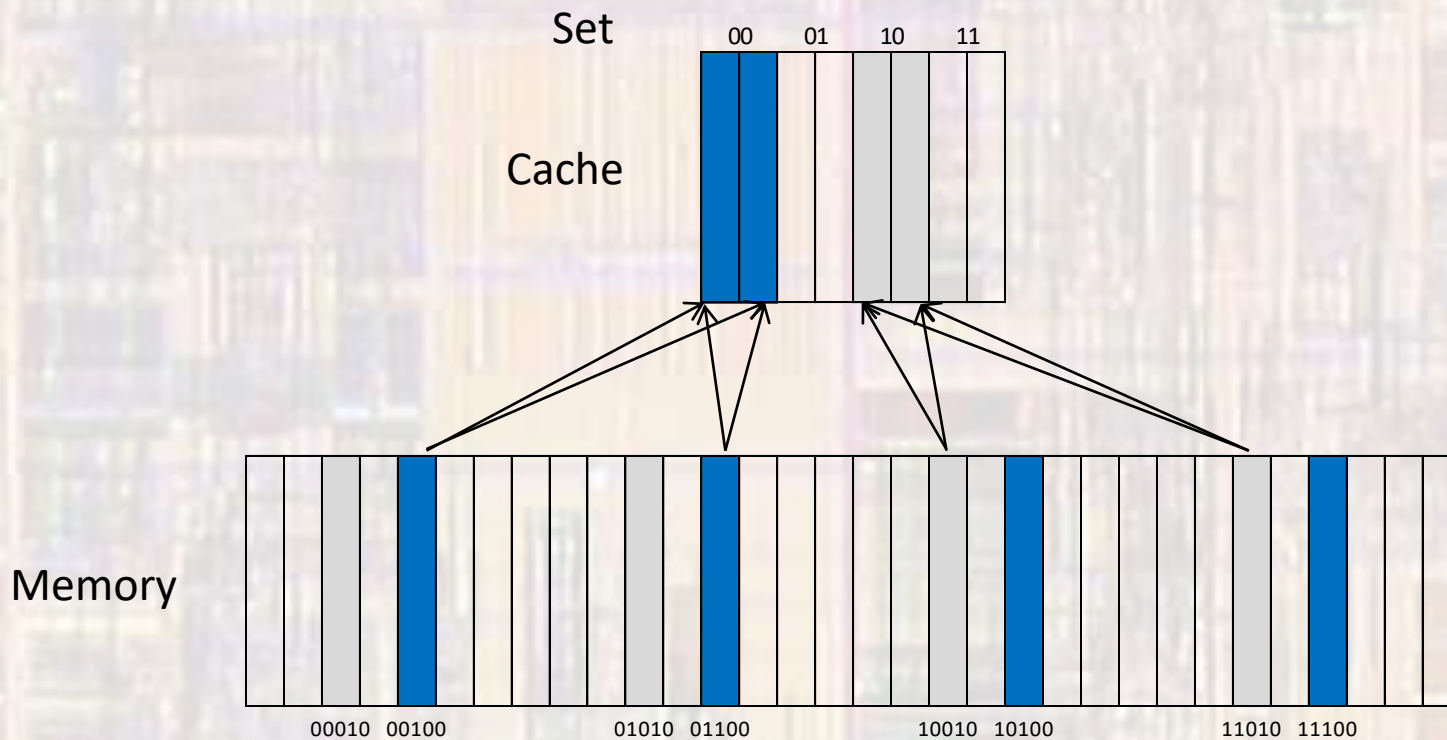
- Fully Associative Cache
 - 1K block cache, 32 bit word



Cache Performance

- Set Associative Cache

- Maps each memory location to a limited number of blocks



Cache Performance

- Set Associative Cache
 - M block, N-way Set Associative Cache
 - N-way \rightarrow each set consists of N blocks
 - M block \rightarrow 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

- Cache Comparison
 - 64 Block Cache
 - Direct Mapped
 - block location = (block number) modulo (# of blocks)
 - $1000 \bmod 64 = \text{block } 40$
 - 2-way Set Associative
 - set location = (block number) modulo (# of sets)
 - $1000 \bmod 32 = \text{set } 8$
 - Fully Associative
 - looks like a 64-way set associative cache \rightarrow 1 set
 - $1000 \bmod 1 = \text{set } 0$

Cache Performance

- 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

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

Block Address	Cache Block
0	$0 \bmod 4 = 0$
6	$6 \bmod 4 = 2$
8	$8 \bmod 4 = 0$

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

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

Block Address	Cache Block
0	$0 \bmod 2 = 0$
6	$6 \bmod 2 = 0$
8	$8 \bmod 2 = 0$

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

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

Block Address	Cache Set
0	$0 \bmod 1 = 0$
6	$6 \bmod 1 = 0$
8	$8 \bmod 1 = 0$

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

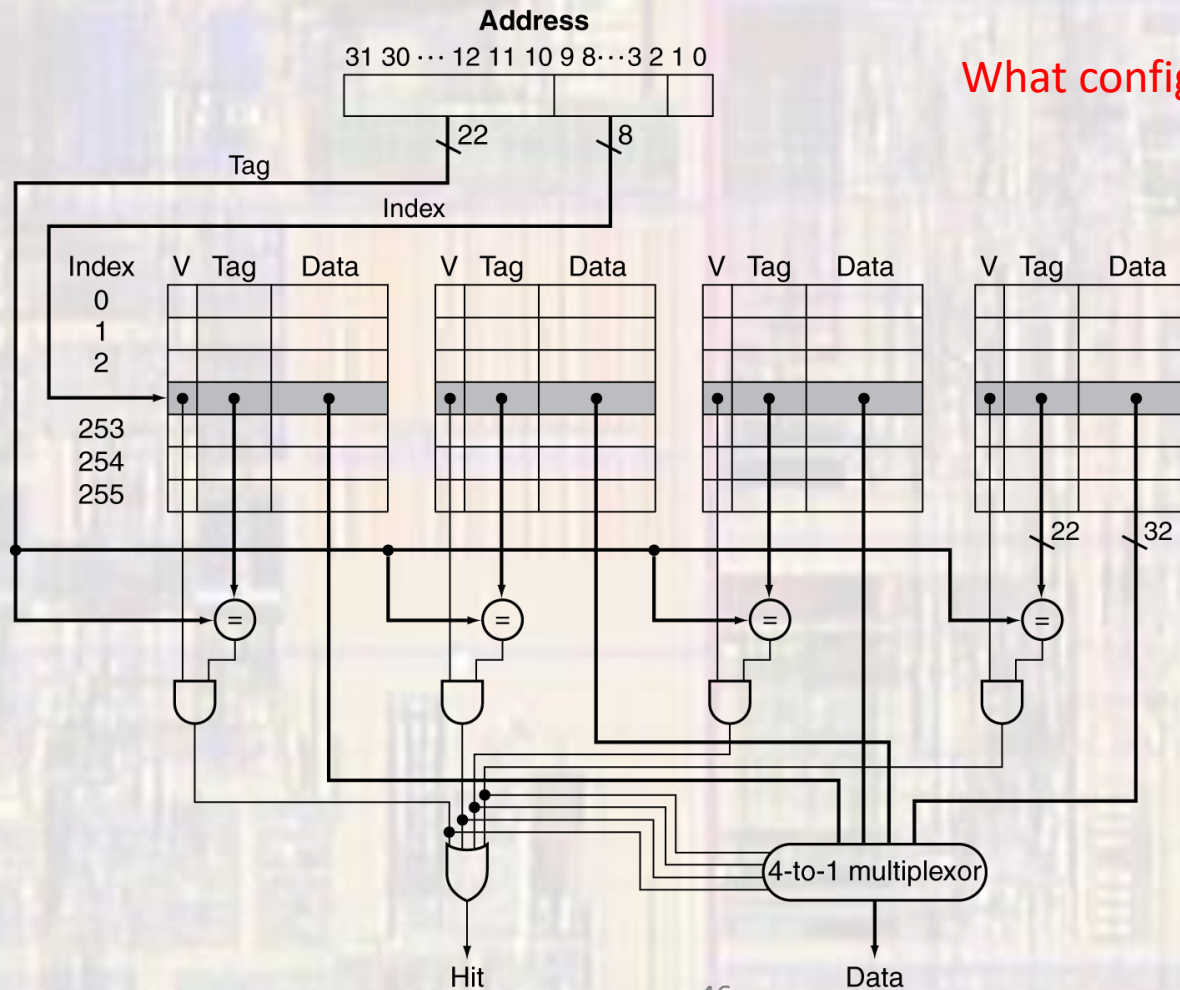
- Cache Comparison
 - As associativity increases:
 - Hit rate goes up
 - Complexity goes up
 - Cost
 - Usually leads to slow down

- 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

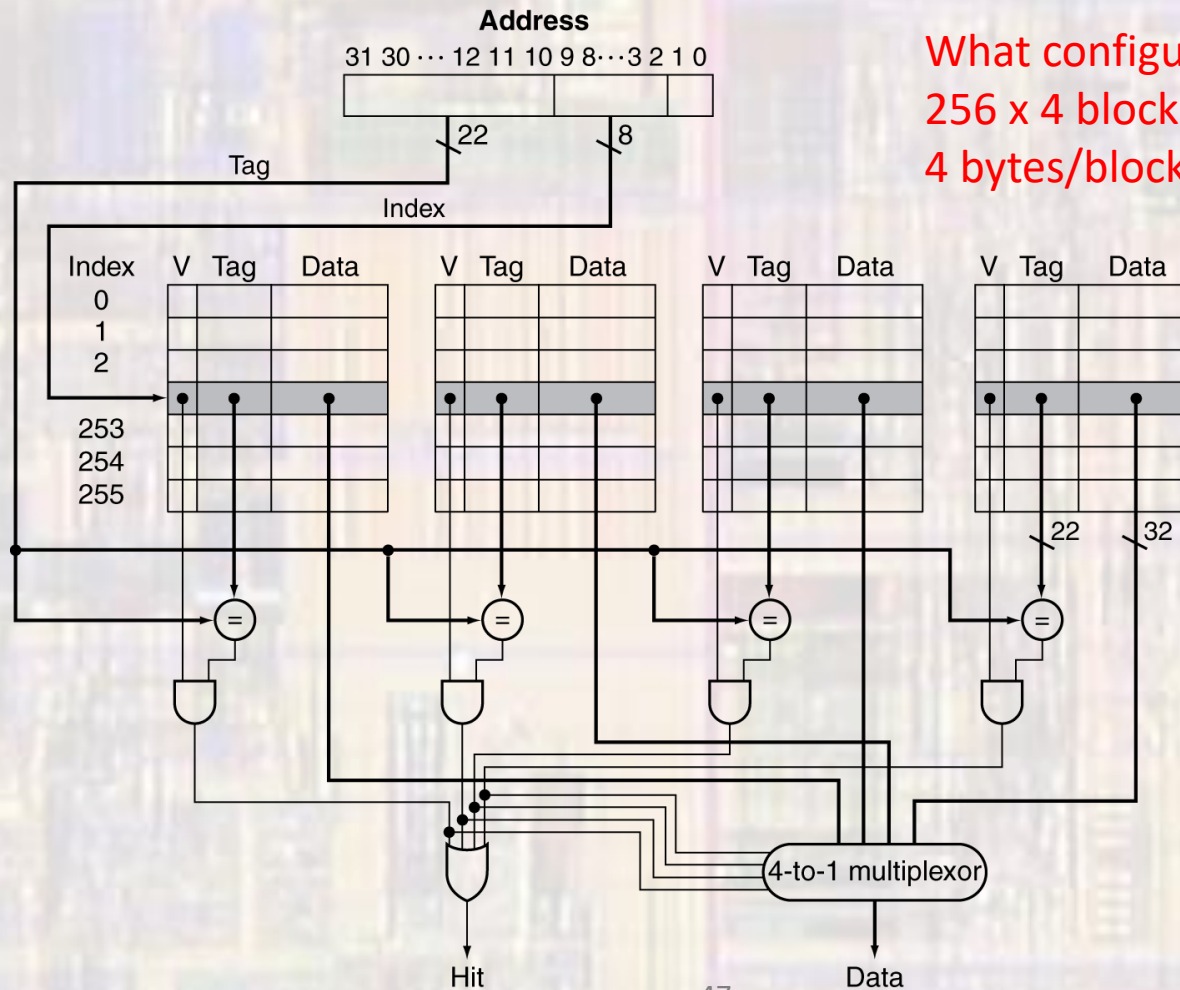
- Cache Implementation



What configuration is this cache?

Cache Performance

- Cache Implementation



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

Cache Performance

- 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

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

Size	Associativity								
	Two-way			Four-way			Eight-way		
	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

- Performance Review

Size	Data misses / 1000 instructions								
	Two-way			Four-way			Eight-way		
	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.

- **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

- 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 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 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}