Lecture 11: Memory Hierarchy
-- Cache Organization and Performance

CSCE 513 Computer Architecture

Department of Computer Science and Engineering
Yonghong Yan
yanyh@cse.sc.edu
https://passlab.github.io/CSCE513

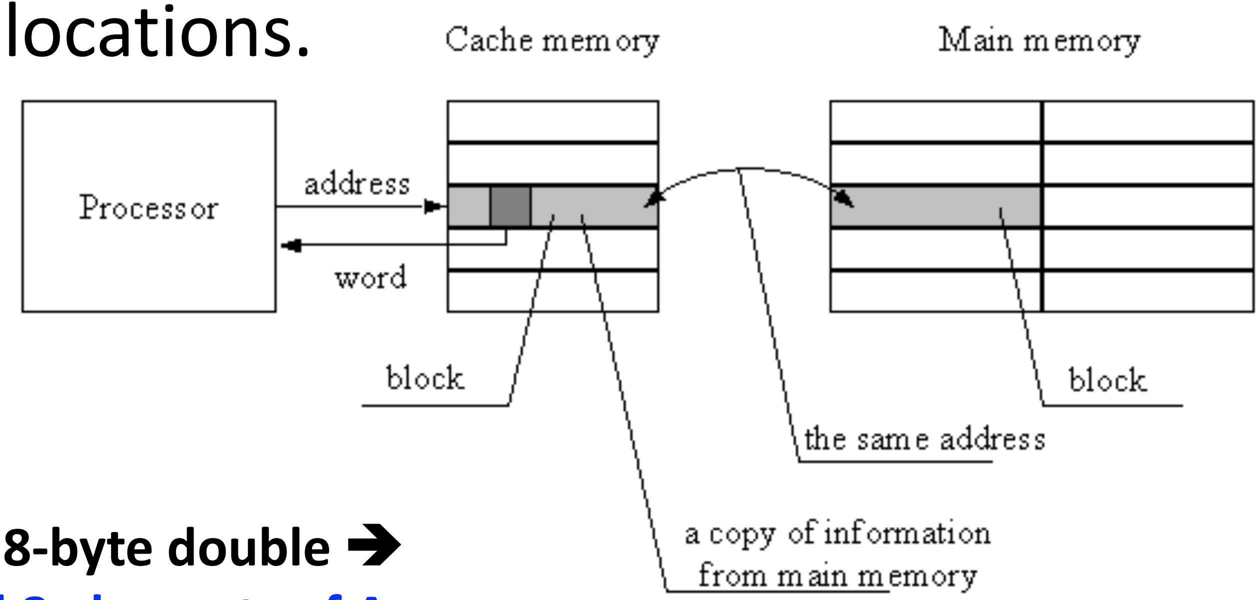
Topics for Memory Hierarchy

- Memory Technology and Principal of Locality
  - CAQA: 2.1, 2.2, B.1
  - COD: 5.1, 5.2
- ☛ Cache Organization and Performance
  - CAQA: B.1, B.2
  - COD: 5.2, 5.3
- Cache Optimization
  - 6 Basic Cache Optimization Techniques
    - CAQA: B.3
  - 10 Advanced Optimization Techniques
    - CAQA: 2.3
- Virtual Memory and Virtual Machine
  - CAQA: B.4, 2.4; COD: 5.6, 5.7
  - Skip for this course

Caching Exploits Both Types of Locality by Preloading and Keeping Data in Faster Memory

- Exploit temporal locality by remembering the contents of recently accessed locations.
- Exploit spatial locality by fetching blocks of data around recently accessed locations.

```
double A[N];
sum = 0;
for(i=0; i<N; i++)
  sum += A[i];
return sum;
```

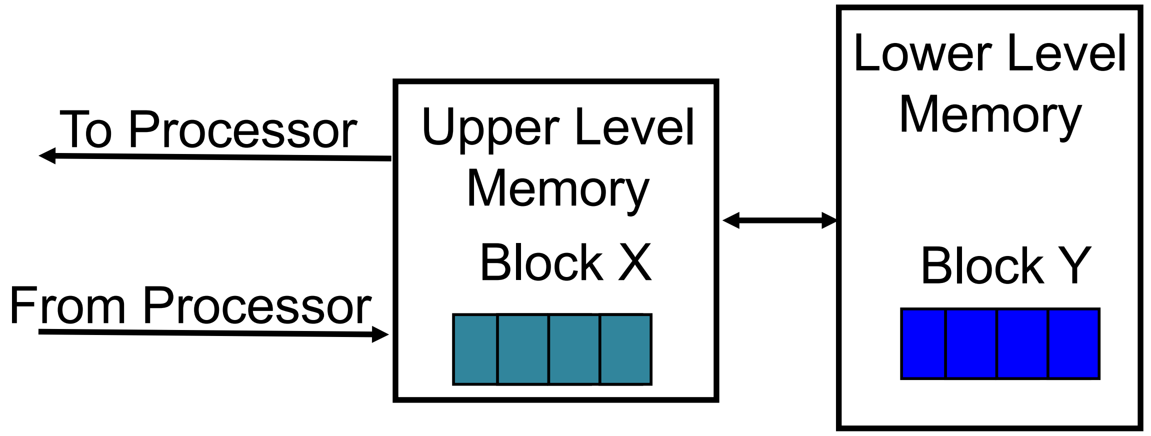


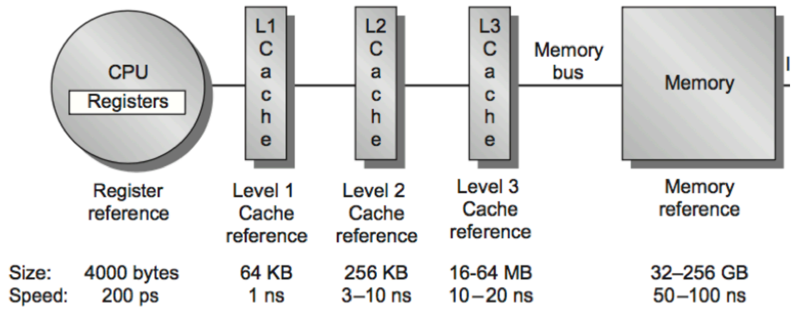
64-byte block size, A[i] is an 8-byte double ➔
A block (cache line) can hold 8 elements of A.

Referencing to A[0] will cause the memory system to bring A[0:7] to the cache ➔
Future reference to A[1:7] are all hits in cache ➔ faster access than reading from memory

Terminology for Memory Hierarchy Performance

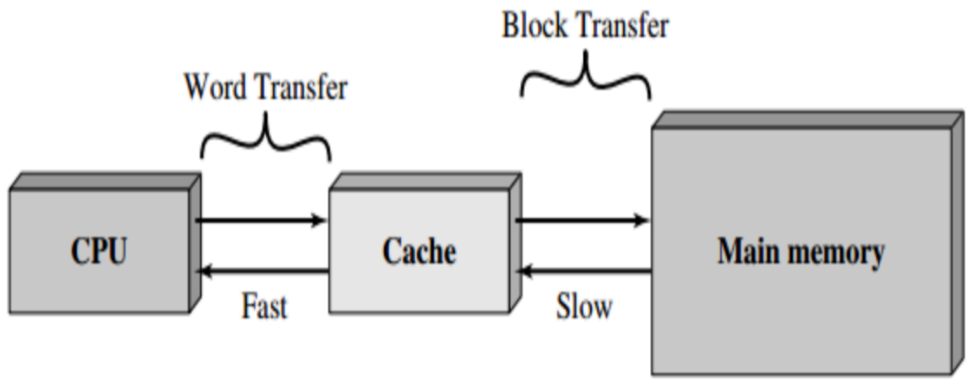
- **Hit:** Data appears in some block in upper level (example: Block X)
  - **Hit Rate:** the fraction of memory access found in the upper level.
  - **Hit Time:** Time to access the upper level which consists of
    - **Upper-level access time + Time to determine hit/miss.**
- **Miss**: data needs to be retrieve from a block in lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate), i.e. # misses / # memory access
  - **Miss Penalty:** Time to replace a block in the upper level
    - **Time to deliver the block the processor.**
- **Hit Time << Miss Penalty (500 instructions on 21264!)**





4 Questions for Cache Organization

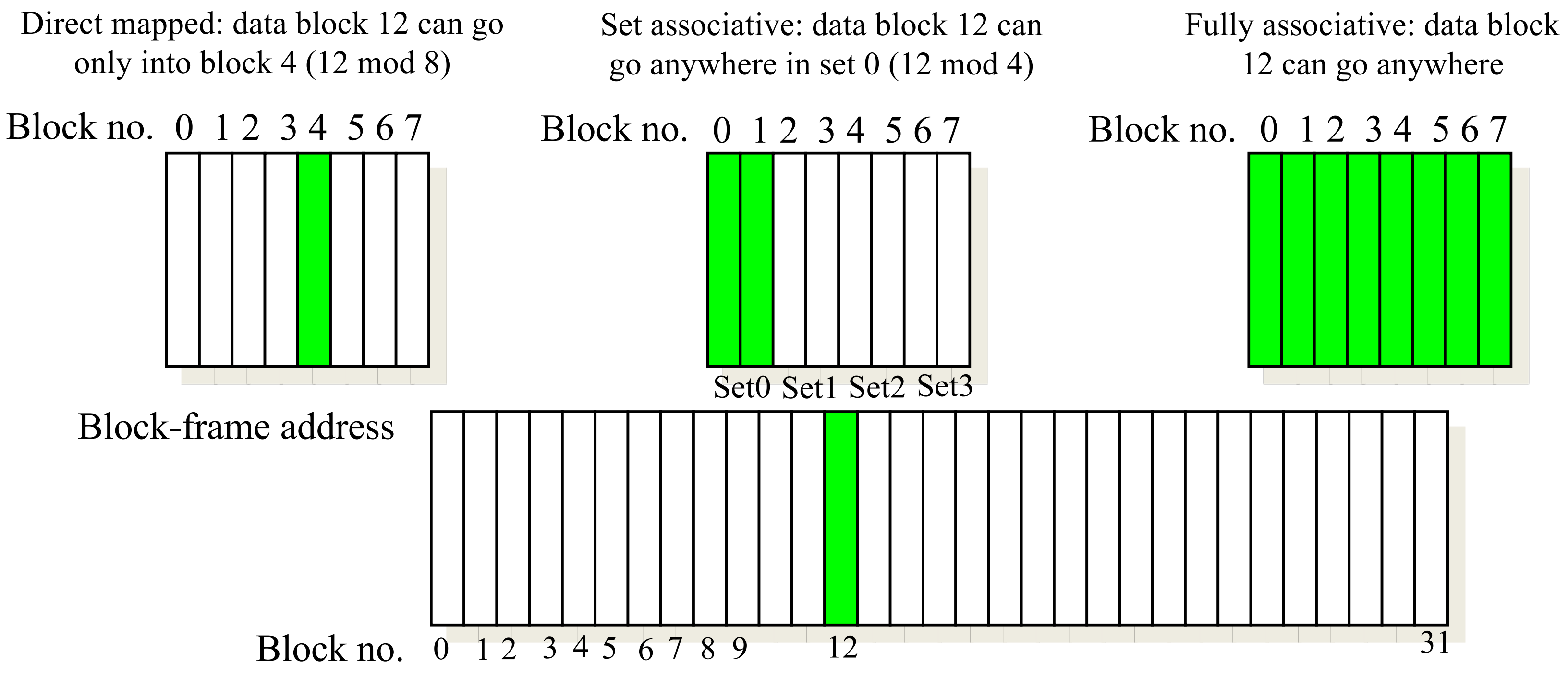
- Q1: Where can a block be placed in the upper level?
  - Block placement
- Q2: How is a block found if it is in the upper level?
  - Block identification
- Q3: Which block should be replaced on a miss?
  - Block replacement
- Q4: What happens on a write?
  - Write strategy



(a) Single cache

Q1: Where Can a Block be Placed in The Upper Level?

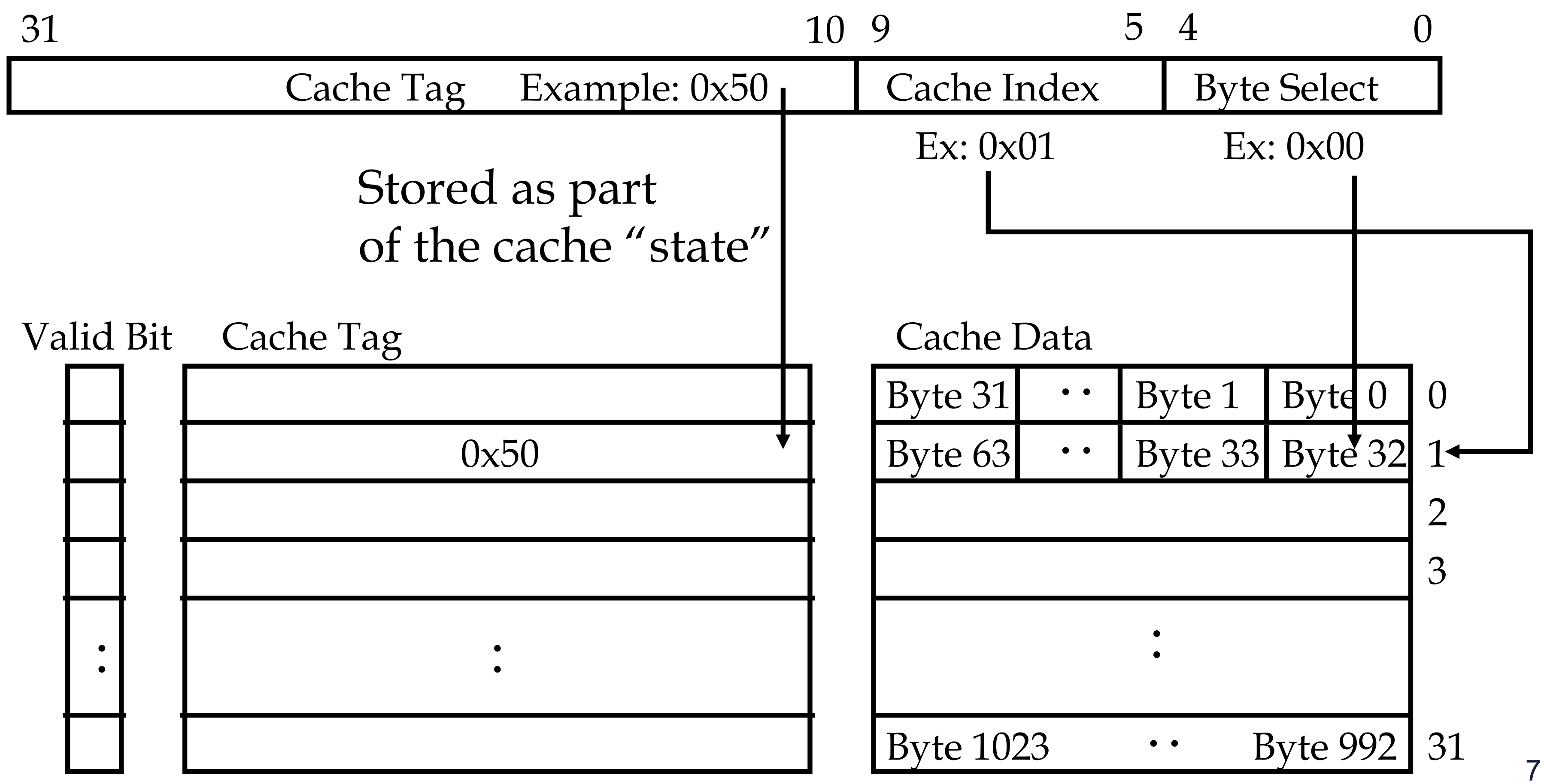
- Block Placement
  - Direct Mapped, Fully Associative, Set Associative
    - Direct mapped: (Block number) mod (Number of blocks in cache)
    - Set associative: (Block number) mod (Number of sets in cache)
      - # of set ≤ # of blocks
      - *n*-way: *n* blocks in a set
      - 1-way = direct mapped
    - Fully associative: # of set = 1



Direct mapped: data block 12 can go only into block 4 (12 mod 8)
Set associative: data block 12 can go anywhere in set 0 (12 mod 4)
Fully associative: data block 12 can go anywhere
Block no. 0 1 2 3 4 5 6 7
Block no. 0 1 2 3 4 5 6 7
Block no. 0 1 2 3 4 5 6 7
Set0 Set1 Set2 Set3
Block-frame address
Block no. 0 1 2 3 4 5 6 7 8 9 12 31

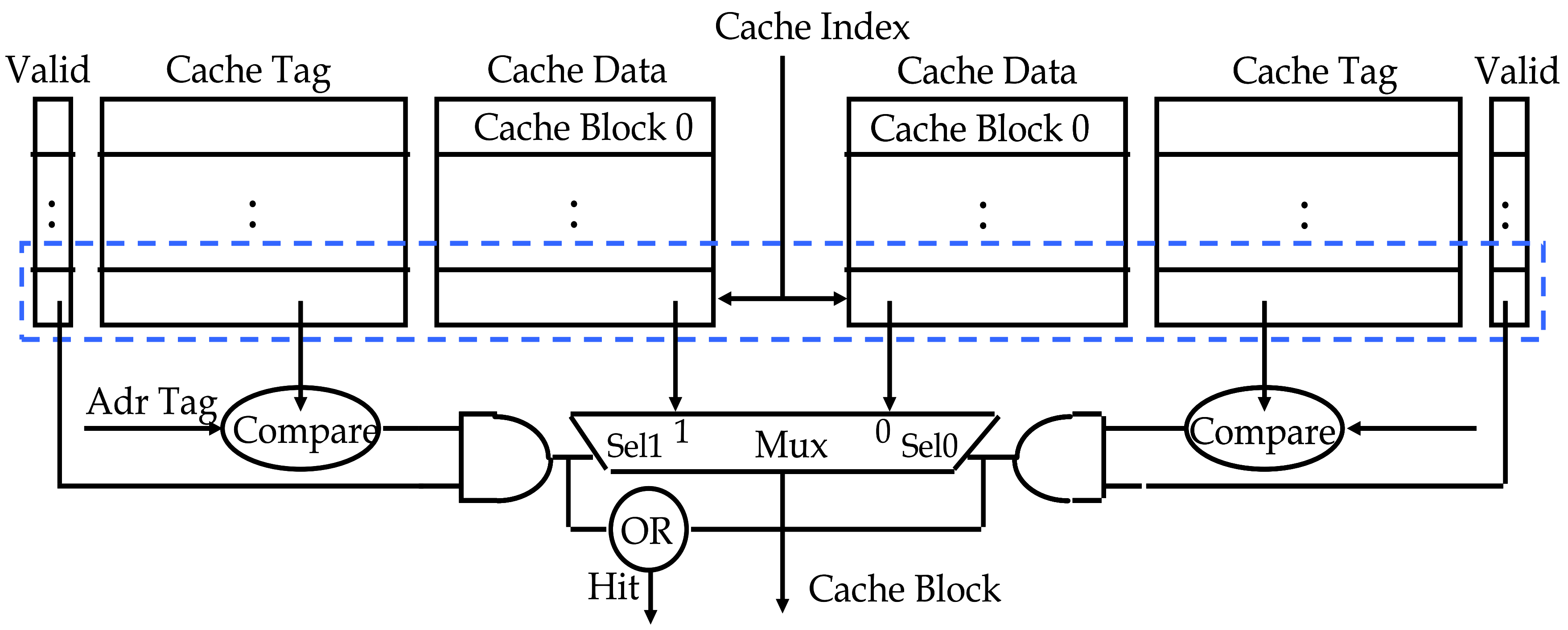
1 KB Direct Mapped Cache, 32B blocks

- For a 2^N byte cache
  - The uppermost (32 - *N*) bits are always the **Cache Tag**
  - The lowest *M* bits are the **Byte Select** (Block Size = 2^M)



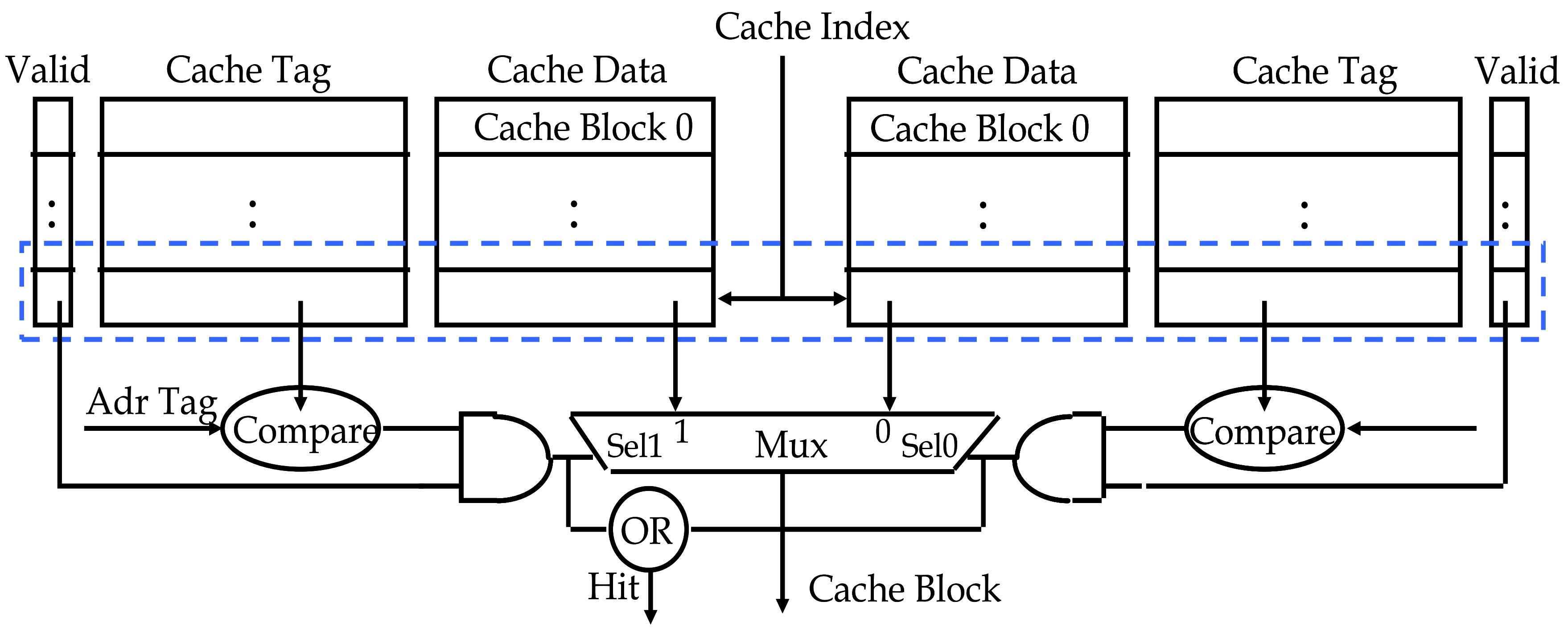
Set Associative Cache

- *N*-way set associative: *N* entries for each **Cache Index**
  - *N* direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - **Cache Index** selects a "set" from the cache;
  - The two tags in the set are compared to the input in parallel;
  - Data is selected based on the tag result.



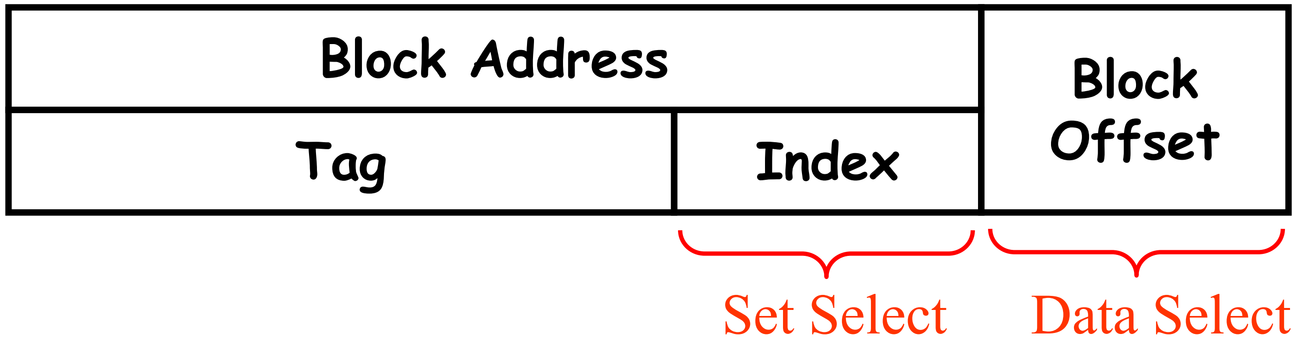
Disadvantage of Set Associative Cache

- *N*-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



Q2: Block Identification

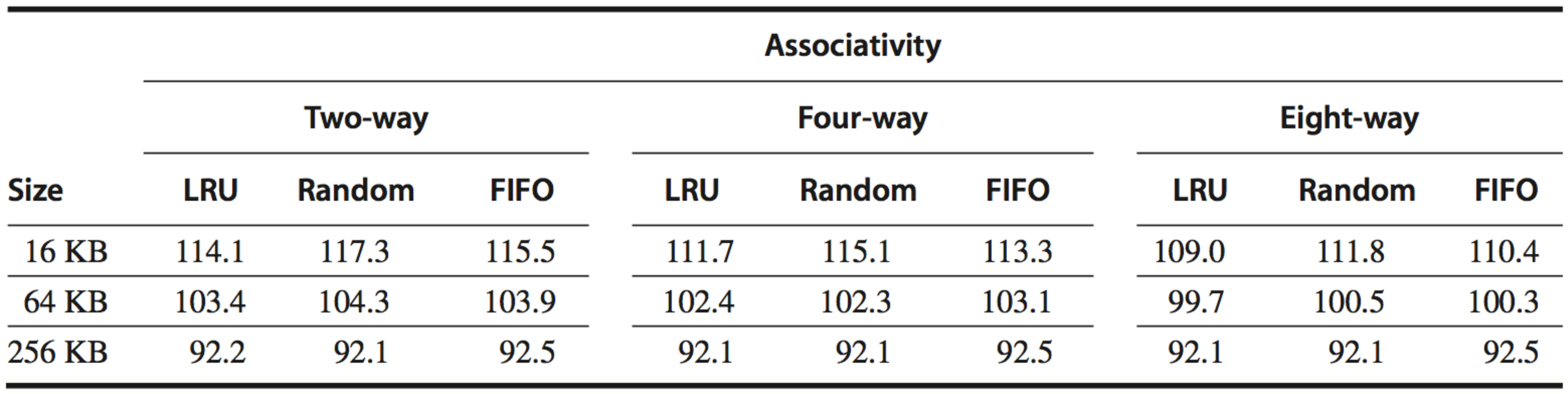
- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag



$$\text{Cache size} = \text{Associativity} \times 2^{\text{index\_size}} \times 2^{\text{offest\_size}}$$

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative
  - Random
  - LRU (Least Recently Used)
  - First in, first out (FIFO)



	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

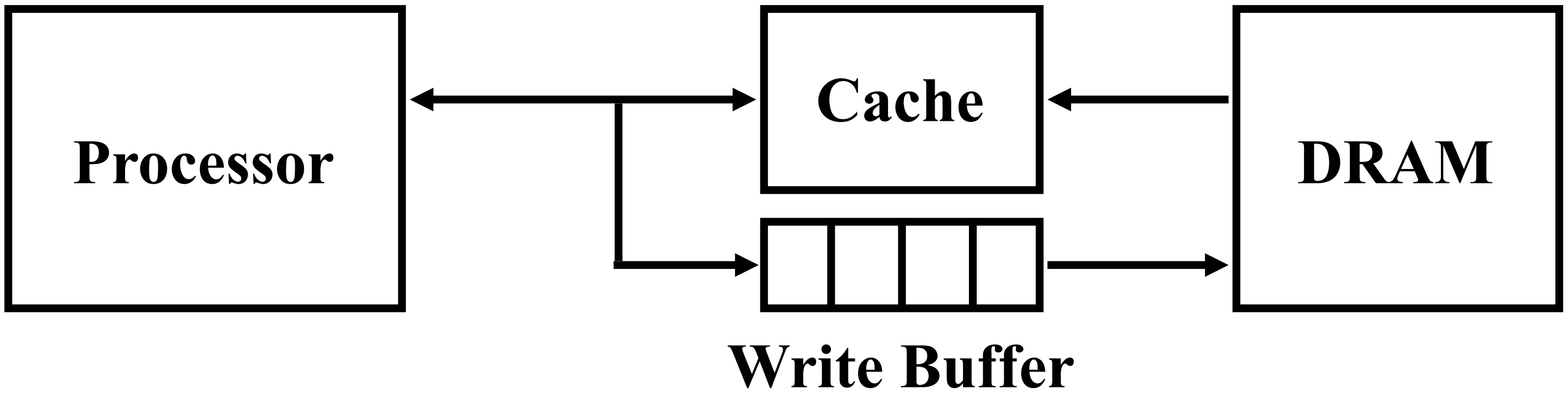
Figure B.4 Data cache misses per 1000 instructions comparing least recently used, random, and first in, first out replacement for several sizes and associativities. There is little difference between LRU and random for the largest size cache, with LRU outperforming the others for smaller caches. FIFO generally outperforms random in the smaller cache sizes. These data were collected for a block size of 64 bytes for the Alpha architecture using 10 SPEC2000 benchmarks. Five are from SPECint2000 (gap, gcc, gzip, mcf, and perl) and five are from SPECfp2000 (applu, art, equake, lucas, and swim). We will use this computer and these benchmarks in most figures in this appendix.

Q4: What Happens on a Write?

	Write-Through	**Write-Back**
Policy	Data written to cache block, also written to lower-level memory	1. Write data only to the cache 2. Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



- Q. Why a write buffer ?
  - A. So CPU doesn't stall
- Q. Why a buffer, why not just one register ?
  - A. Bursts of writes are common.
- Q. Are Read After Write (RAW) hazards an issue for write buffer?
  - A. Yes! Drain buffer before next read, or send read 1st after check write buffers.

Write-Miss Policy

- Two options on a write miss
  - Write allocate – the block is allocated on a write miss, followed by the write hit actions.
    - Write misses act like read misses.
  - No-write allocate – write misses do not affect the cache. The block is modified only in the lower-level memory.
    - Block stay out of the cache in no-write allocate until the program tries to read the blocks, but with write allocate even blocks that are only written will still be in the cache.

Write-Miss Policy Example

- Example: Assume a fully associative write-back cache with many cache entries that starts empty. Below is sequence of five memory operations (The address is in square brackets):

  Write Mem[100];
  Write Mem[100];
  Read Mem[200];
  Write Mem[200];
  Write Mem[100].

  What are the number of hits and misses (inclusive reads and writes) when using no-write allocate versus write allocate?

- *Answer*

No-write Allocate:		*Write allocate*:	
Write Mem[100];	1 write miss	Write Mem[100];	1 write miss
Write Mem[100];	1 write miss	Write Mem[100];	1 write hit
Read Mem[200];	1 read miss	Read Mem[200];	1 read miss
Write Mem[200];	1 write hit	Write Mem[200];	1 write hit
Write Mem[100].	1 write miss	Write Mem[100];	1 write hit
4 misses; 1 hit		2 misses; 3 hits	

What Happens on a Cache Miss in Pipeline?

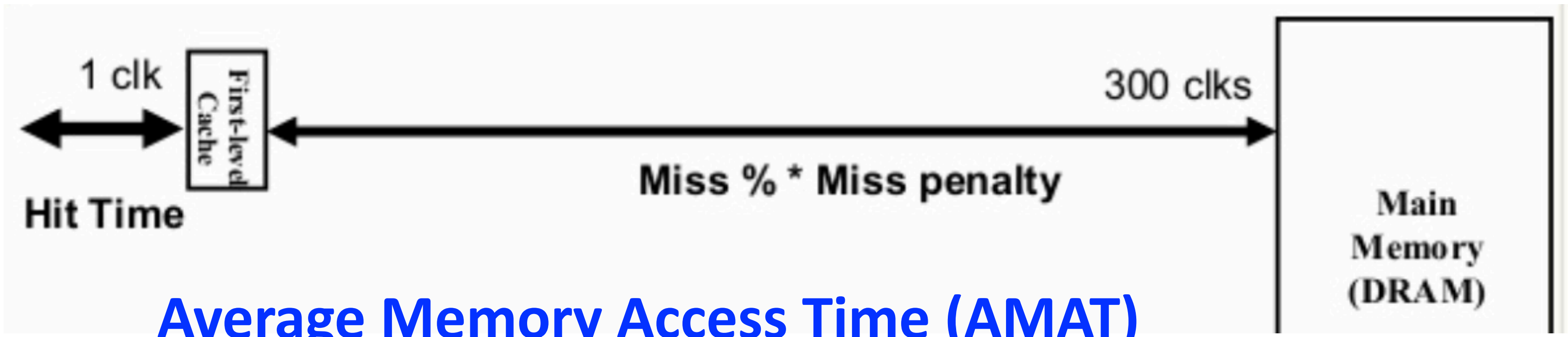
- For in-order pipeline, 2 options:
  - Freeze pipeline in Mem stage (popular early on: Sparc, R4000)

```
IF ID  EX  Mem stall stall stall … stall Mem   Wr
    IF  ID  EX     stall stall stall … stall stall     Ex   Wr
```

  - Release load from pipeline
    - MSHR = "Miss Status/Handler Registers" (Kroft)
      Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
      - Per cache-line: keep info about memory address.
      - For each word: register (if any) that is waiting for result.
      - Used to "merge" multiple requests to one memory line
    - New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
    - Attempt to use register before result returns causes instruction to block in decode stage.

Cache Performance

Average Memory Access Time (AMAT)



Average Memory Access Time (AMAT)
= Hit Time + Miss Rate * Miss Penalt
= T_{hit}(L1) + Miss% (L1) * T(Mem)

- **Miss penalty**: Time to fetch a block from lower memory level
  - Access time: function of latency
  - Transfer time: function of bandwith b/w levels
    - Transfer one “cache block/line” at a time
- Example:
  - Cache Hit = 1 cycle
  - Miss rate = 10%, Miss penalty = 300 cycles
  - **AMAT = 1 + 0.1 * 300 = 31 cycles**

Miss Rate: Miss per Memory Reference

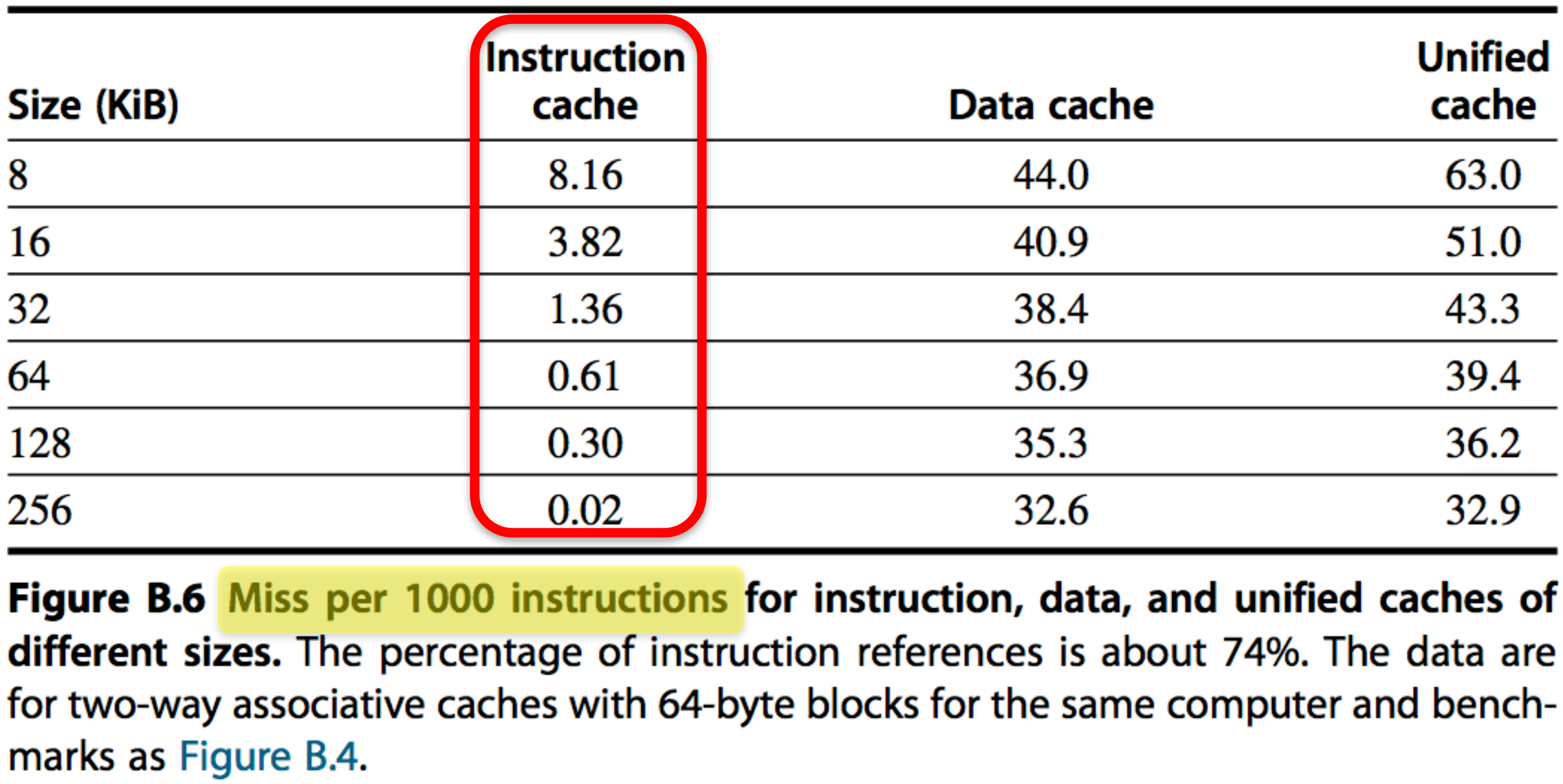
Miss Rate = # Misses / # Memory reference

- Memory access include both instruction access and data access
  - Each instruction needs to be read from memory: one I-mem access
  - LW/SW are memory access instructions: one D-mem access (in addition to one 1-mem access)
  - **Count # memory accesses: Each iteration, 14 instructions in total, 9 ld/lw/sw ➔ 14+9 or 9*2+5 = 23 memory accesses**
    - **9 are data-mem accesses**

```
int sum(int N, int a, int *X) {
    int i;
    int result = 0;
    for (i = 0; i < N; ++i)
        result += X[i];
    return result;
}
```

```
.L3:
        lw      a5,-20(s0)    /* a5 = i */
        sll     a5,a5,2       /* a5 = i<<2, which is i=i*4
        ld      a4,-48(s0)    /* a4 = X */
        add     a5,a4,a5      /* the &X[i] */
        lw      a5,0(a5)      /* the X[i] */
        lw      a4,-24(s0)    /* load result */
        addw    a5,a4,a5      /* result += X[i] */
        sw      a5,-24(s0)    /* store to result */
        lw      a5,-20(s0)    /* i */
        addw    a5,a5,1       /* i++ */
        sw      a5,-20(s0)    /* store i */
.L2:
        lw      a4,-20(s0)    /* i */
        lw      a5,-36(s0)    /* N */
        blt     a4,a5,.L3     /* if (i < N) goto .L3 */
```

Miss Per Instruction (Textbook B-15 and B-16)



Size (KiB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Figure B.6 Miss per 1000 instructions for instruction, data, and unified caches of different sizes. The percentage of instruction references is about 74%. The data are for two-way associative caches with 64-byte blocks for the same computer and benchmarks as Figure B.4.

Misses for instruction cache are very low, why?

Because instruction access has good locality!

Miss Rate ←→ Miss Per Instruction (Textbook B-15 and B-16)

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \boxed{\frac{\text{Memory accesses}}{\text{Instruction}}}$$

$$\Downarrow$$

$$\text{Miss rate} = \frac{\dfrac{\text{Misses}}{1000\ \text{Instructions}} / 1000}{\dfrac{\text{Memory accesses}}{\text{Instruction}}}$$

- **Memory Access Per Instruction:**
  - Each instruction needs to be read from memory: one I-mem access
  - LW/SW instructions: one D-mem access (in addition to one 1-mem access)
- E.g. 36% are load/store instruction
  - **Memory Access Per Instruction = 1 + 0.36 = 1.36**

Miss Rate ←→ Miss Per Instruction (Textbook B-15 and B-16)

Size (KiB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61		
128	0.30		
256	0.02		

Figure B.6 Miss per 1000 instru different sizes. The percentage o for two-way associative caches wit marks as Figure B.4.

NOTE: Very low instruction miss rate

$$\text{Miss rate} = \frac{\dfrac{\text{Misses}}{1000\ \text{Instructions}}/1000}{\dfrac{\text{Memory accesses}}{\text{Instruction}}}$$

Because every instruction access has exactly one memory access to fetch the instruction, the instruction miss rate is

$$\text{Miss rate}_{16\,\text{KB instruction}} = \frac{3.82/1000}{1.00} = 0.004$$

Because 36% of the instructions are data transfers, the data miss rate is

$$\text{Miss rate}_{16\,\text{KB data}} = \frac{40.9/1000}{0.36} = 0.114$$

The unified miss rate needs to account for instruction and data accesses:

$$\text{Miss rate}_{32\,\text{KB unified}} = \frac{43.3/1000}{1.00 + 0.36} = 0.0318$$

Miss Rate for Programmers

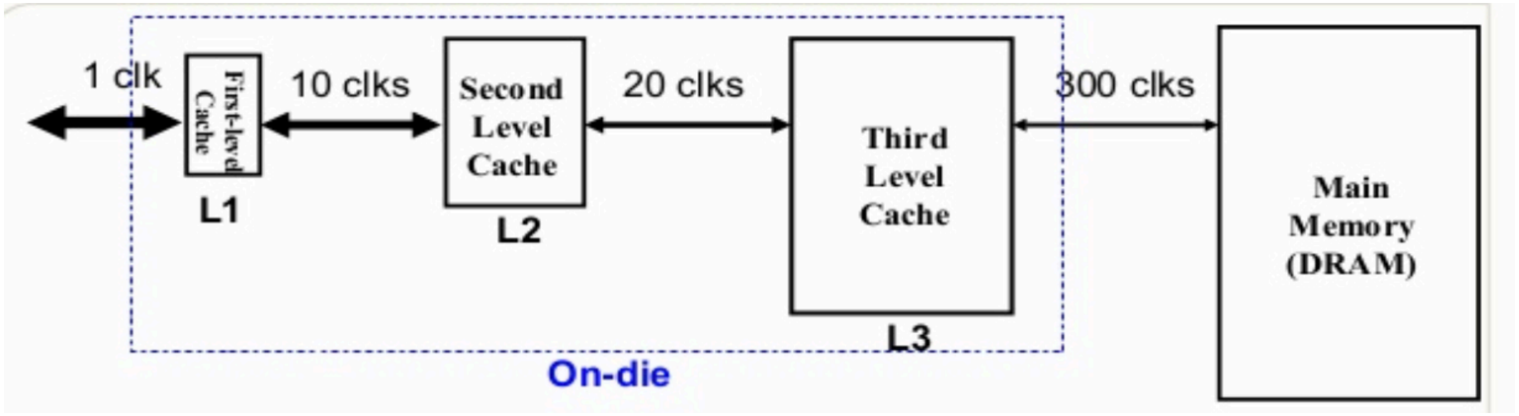
- Because instruction miss rate is very low → Only count data-memory access

  Counting data-mem access only needs to look at source code! and count array reference!

```
int sum(int N, int a, int *X) {
    int i;
    int result = 0;
    for (i = 0; i < N; ++i)
        result += X[i];
    return result;
}
```

  – **1 data mem access**
    - **Since scalar variables (i, result) are all in registers**
    - **Only X[i] is LW (right value)**

AMAT for Multi-Level Cache



- Average Memory Access Time (AMAT)

  = Hit Time + Miss Rate * Miss Penalty

Miss Penalty(L3) = T(Mem)

Miss Penalty(L2) = AMAT(L3) = T_{hit}(L3) + Miss%(L3) * Miss Penalty(L3)

Miss Penalty(L1) = AMAT(L2) = T_{hit}(L2) + Miss%(L2) * Miss Penalty(L2)

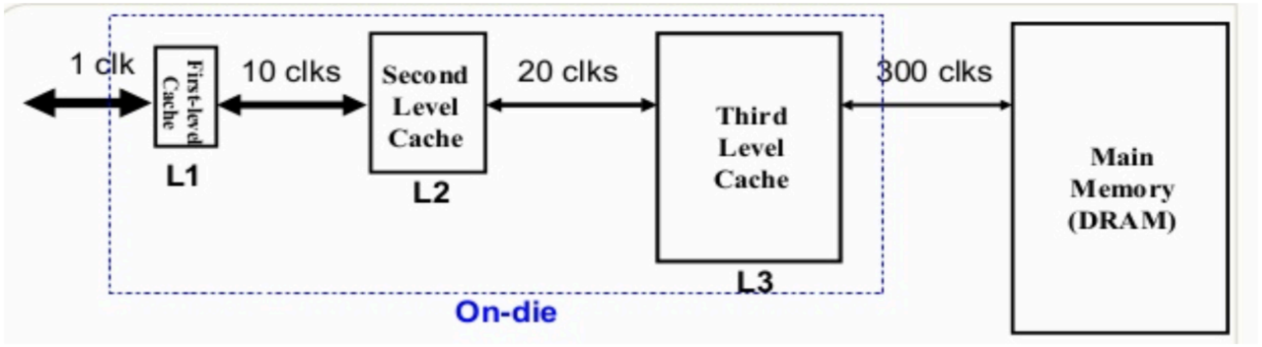
AMAT = T_{hit}(L1) + Miss%(L1) * Miss Penalty(L1)

= T_{hit}(L1) + Miss%(L1) * {T_{hit}(L2) + Miss%(L2) * Miss Penalty(L2)}

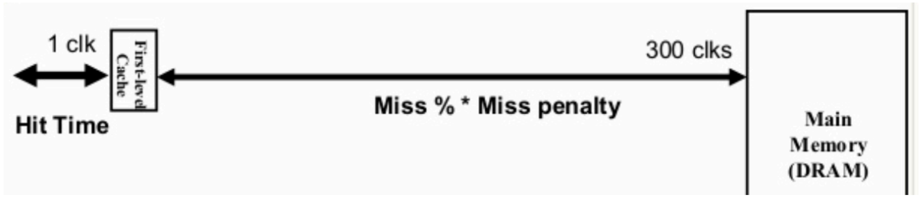
= T_{hit}(L1) + Miss%(L1) * {T_{hit}(L2) + Miss%(L2) * [T_{hit}(L3) + Miss%(L3) * *T(Mem)*] }

AMAT Example for Multi-Level Cache

AMAT = T_{hit}(L1) + Miss%(L1) * {T_{hit}(L2) + Miss%(L2) * [T_{hit}(L3) + Miss%(L3) * *T(Mem)*] }



- Miss rate L1=10%, T_{hit}(L1) = 1 cycle
- Miss rate L2 = 5%, T_{hit}(L2) = 10 cycles
- Miss rate L3 = 1%, T_{hit}(L3) = 20 cycles
- T(Mem) = 300 cycles



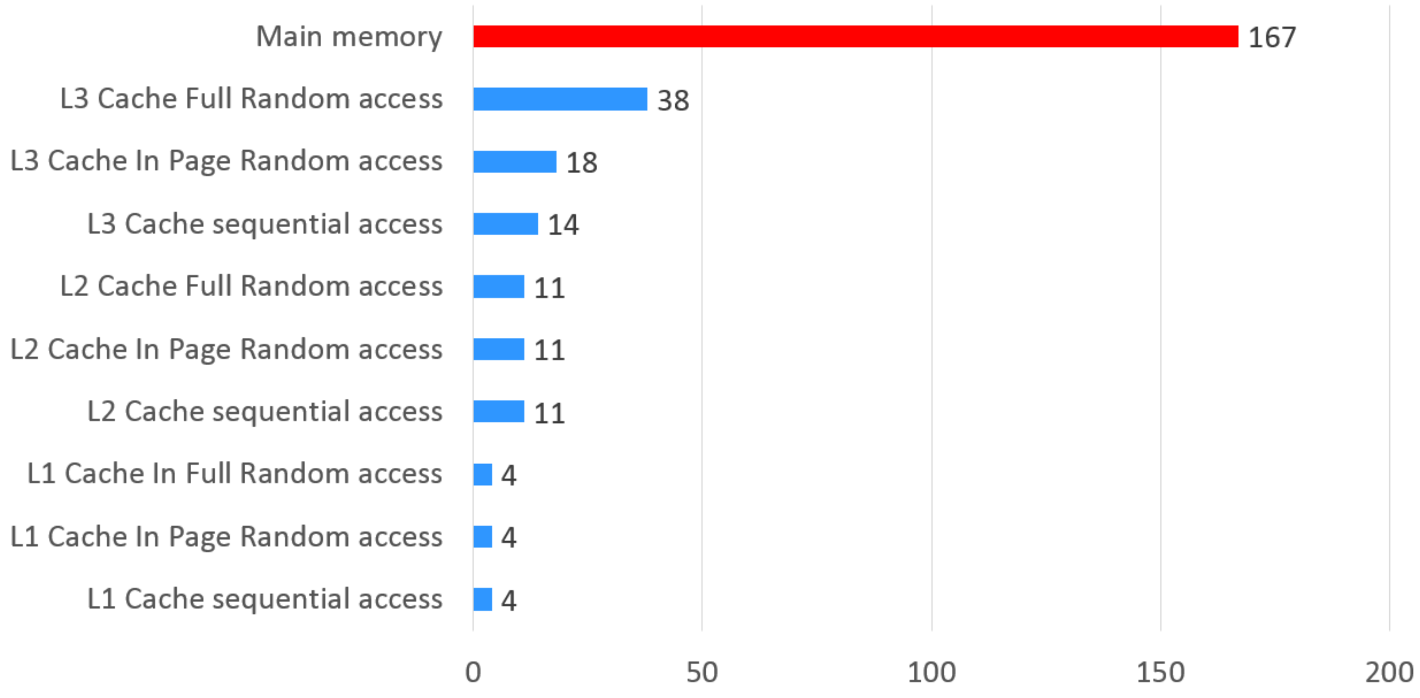
- **AMAT =**
  - **2.115 compared to 31 with single-level L1 cache**

CPU Performance Revisit

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware

$$CPU\ \text{Time} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

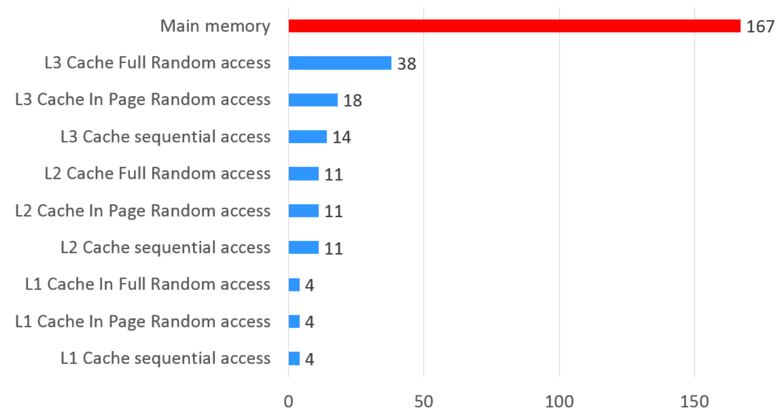
CPU Cache Access Latencies in Clock Cycles



CPU Performance with Memory Factor

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - **Memory stall cycles**
    - **Mainly from cache misses**

CPU Cache Access Latencies in Clock Cycles



- **Memory Stall Cycles:** the number of cycles during which the processor is stalled waiting for a memory access.

$$\boxed{\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}}$$

Memory Stall Cycles per Instruction

$$\mathit{CPU}\ \text{Time} = \frac{\text{Instructions}}{\text{Program}} * \boxed{\frac{\text{Cycles}}{\text{Instruction}}} * \frac{\text{Time}}{\text{Cycle}}$$

$$\text{CPU execution time} = (\text{CPU clock cycles} + \boxed{\text{Memory stall cycles}}) \times \text{Clock cycle time}$$

$$\Downarrow$$

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \boxed{\frac{\text{Memory stall clock cycles}}{\text{Instruction}}}\right) \times \text{Clock cycle time}$$

Memory Stall Cycles

- **Memory Stall Cycles:** the number of cycles during which the processor is stalled waiting for a memory access.
- Depends on both the number of misses and the cost per miss, i.e. the miss penalty:

$$\boxed{\begin{aligned}\text{Memory stall cycles} &= \text{Number of misses} \times \text{Miss penalty} \\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instrution}} \times \text{Miss Penalty} \\ &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instrution}} \times \text{Miss rate} \times \text{Miss Penalty}\end{aligned}}$$

† The advantage of the last form is the component can be easily measured.

Cache Performance

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$

- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Example (Page B-18 of CAQA Book) - 1/2

Let's use an in-order execution computer for the first example. Assume that the cache miss penalty is 200 clock cycles, and all instructions usually take 1.0 clock cycles (ignoring memory stalls). Assume that the average miss rate is 2%, there is an average of 1.5 memory references per instruction, and the average number of cache misses per 1000 instructions is 30. What is the impact on performance when behavior of the cache is included? Calculate the impact using both misses per instruction and miss rate.

Answer

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}}\right) \times \text{Clock cycle time}$$

The performance, including cache misses, is

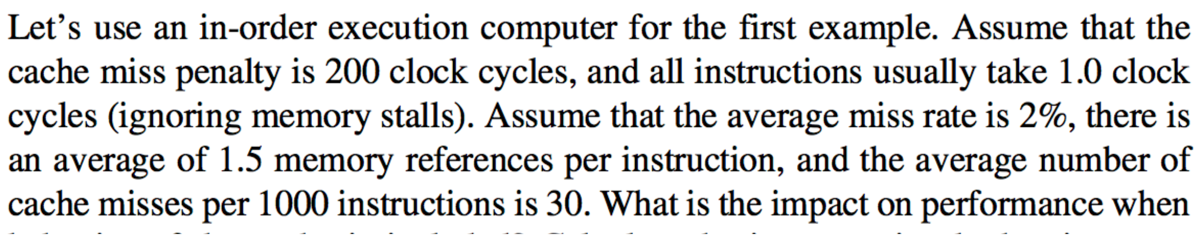
$$\begin{aligned}\text{CPU time}_{\text{with cache}} &= \text{IC} \times [1.0 + (30/1000 \times 200)] \times \text{Clock cycle time}\\ &= \text{IC} \times 7.00 \times \text{Clock cycle time}\end{aligned}$$

Now calculating performance using miss rate:

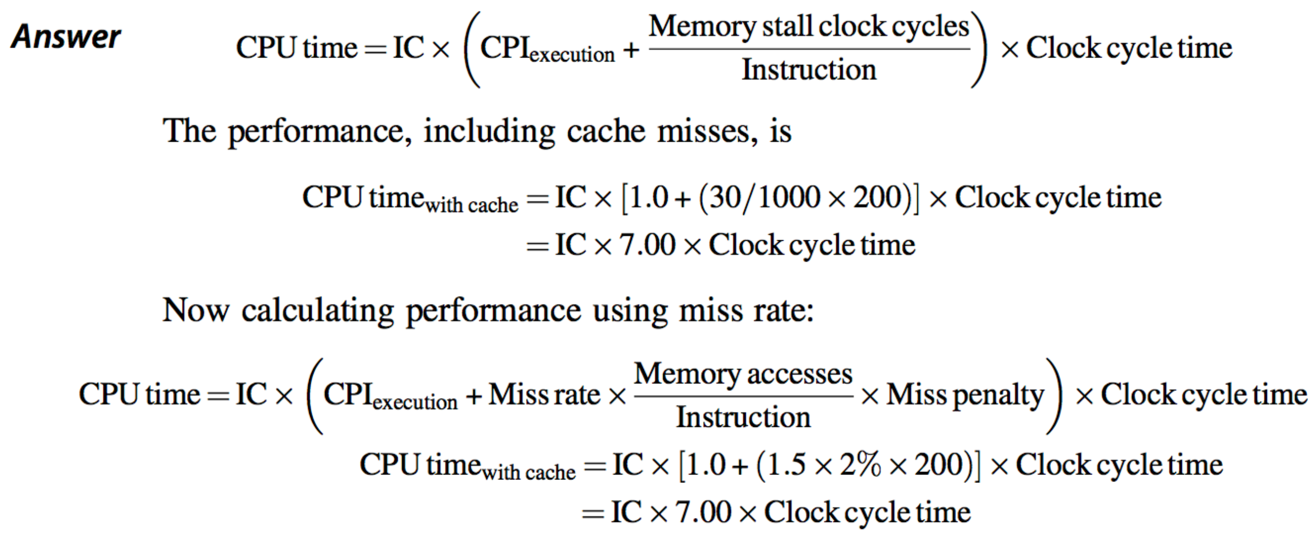
$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\begin{aligned}\text{CPU time}_{\text{with cache}} &= \text{IC} \times [1.0 + (1.5 \times 2\% \times 200)] \times \text{Clock cycle time}\\ &= \text{IC} \times 7.00 \times \text{Clock cycle time}\end{aligned}$$

Example (Page B-18 of CAQA Book)- 2/2



Let's use an in-order execution computer for the first example. Assume that the cache miss penalty is 200 clock cycles, and all instructions usually take 1.0 clock cycles (ignoring memory stalls). Assume that the average miss rate is 2%, there is an average of 1.5 memory references per instruction, and the average number of cache misses per 1000 instructions is 30. What is the impact on performance when



Answer

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}}\right) \times \text{Clock cycle time}$$

The performance, including cache misses, is

$$\begin{aligned}\text{CPU time}_{\text{with cache}} &= \text{IC} \times [1.0 + (30/1000 \times 200)] \times \text{Clock cycle time}\\ &= \text{IC} \times 7.00 \times \text{Clock cycle time}\end{aligned}$$

Now calculating performance using miss rate:

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\begin{aligned}\text{CPU time}_{\text{with cache}} &= \text{IC} \times [1.0 + (1.5 \times 2\% \times 200)] \times \text{Clock cycle time}\\ &= \text{IC} \times 7.00 \times \text{Clock cycle time}\end{aligned}$$

- **CPI for perfect cache (miss rate 0): 1.00**
- **CPU for 2% miss rate cache: 7**
- **CPI for No cache: = 1.0+200*1.5 = 301**
  - **a factor of more than 40 times longer than a system with a cache!**

Three Important Equations for Cache Performance

Average Memory Access Time (AMAT)

= Hit Time + Miss Rate * Miss Penalty

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

$$\begin{aligned}\text{Memory stall cycles} &= \text{Number of misses} \times \text{Miss penalty}\\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instrution}} \times \text{Miss Penalty}\\ &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instrution}} \times \text{Miss rate} \times \text{Miss Penalty}\end{aligned}$$

Summary of Performance Equations

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}}\right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{Total miss latency} - \text{Overlapped miss latency})$$

$$\text{Average memory access time} = \text{Hit time}_{\text{L1}} + \text{Miss rate}_{\text{L1}} \times (\text{Hit time}_{\text{L2}} + \text{Miss rate}_{\text{L2}} \times \text{Miss penalty}_{\text{L2}})$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}_{\text{L1}}}{\text{Instruction}} \times \text{Hit time}_{\text{L2}} + \frac{\text{Misses}_{\text{L2}}}{\text{Instruction}} \times \text{Miss penalty}_{\text{L2}}$$

Figure B.7 Summary of performance equations in this appendix. The first equation calculates the cache index size, and the rest help evaluate performance. The final two equations deal with multilevel caches, which are explained early in the next section. They are included here to help make the figure a useful reference.

Another Example

- Assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?
- Answer:

1. Compute the performance for the computer that always hits:

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$
$$= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle} = \text{IC} \times 1.0 \times \text{Clock cycle}$$

2. For the computer with the real cache, we compute memory stall cycles:

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$
$$= \text{IC} \times \boxed{(1 + 0.5)} \times 0.02 \times 25 = \text{IC} \times 0.75$$

1 instruction memory access + 0.5 data memory access

3. Compute the total performance

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$
$$= 1.75 \times \text{IC} \times \text{Clock cycle}$$

4. Compute the performance ratio which is the inverse of the execution times

$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times \text{IC} \times \text{Clock cycle}}{1.0 \times \text{IC} \times \text{Clock cycle}} = 1.75$$

Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- **Can’t neglect cache behavior when evaluating system performance**

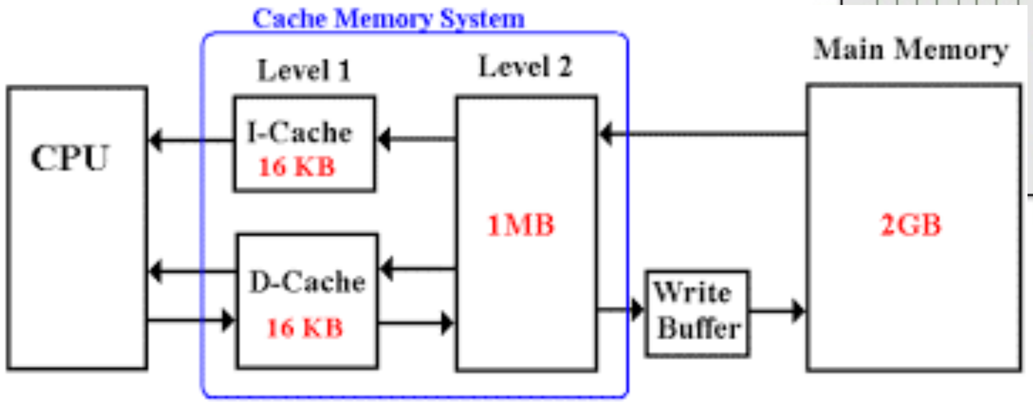
Midterm

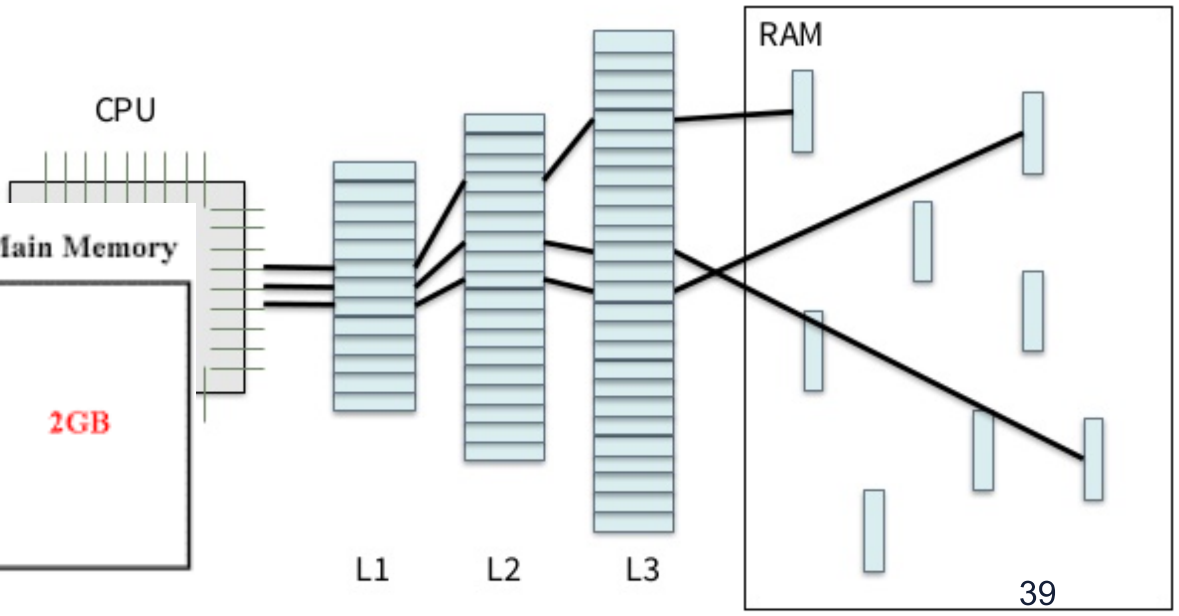
- Oct 15 Monday 8:05 - 9:20
- Close book and close note
- 1-page (one side of the paper) A4/Letter-size hand-written note
- Some sample questions:
  – https://passlab.github.io/CSCE513/Midterm_Fall2016_formatted_WithSolutions.pdf

Additional

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache





Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = $9/3.4 = 2.6$

Memory Hierarchy Performance

- Two indirect performance measures have waylaid many a computer designer.
  - *Instruction count* is independent of the hardware;
  - *Miss rate* is independent of the hardware.
- A better measure of memory hierarchy performance is the *Average Memory Access Time* (AMAT) per instructions

$$\text{AMAT} = \text{Hit time} + \text{Mis srate} \times \text{Miss penalty}$$

  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Example (B-16): Separate vs Unified Cache

- Which has the lower miss rate: a 16 KB instruction cache with a 16KB data or a 32 KB unified cache? Use the miss rates in Figure B.6 to help calculate the correct answer, assuming 36% of the instructions are data transfer instructions. Assume a hit take 1 clock cycle and the miss penalty is 100 clock cycles. A load or store hit takes 1 extra clock cycle on a unified cache if there is only one cache port to satisfy two simultaneous requests. Using the pipelining terminology of Chapter 2, the unified cache leads to a structure hazard. What is the average memory access time in each case? Assume write-through caches with a write buffer and ignore stalls due to the write buffer.
- Answer:

  First let's convert misses per 1000 instructions into miss rates. Solving the general formula from above, the miss rate is

$$\text{Miss rate} = \frac{\dfrac{\text{Misses}}{\text{1000 Instructions}}/1000}{\dfrac{\text{Memory accesses}}{\text{Instruction}}}$$

  Since every instruction access has exactly one memory access to fetch the instruction, the instruction miss rate is

$$\text{Miss rate}_{\text{16 KB instruction}} = \frac{3.82/1000}{1.00} = 0.004$$

Example (B-16)

Since 36% of the instructions are data transfers, the data miss rate is

$$\text{Miss rate}_{16\,\text{KB data}} = \frac{40.9/1000}{0.36} = 0.114$$

The unified miss rate needs to account for instruction and data access:

$$\text{Miss rate}_{32\,\text{KB unified}} = \frac{43.3/1000}{1.00+0.36} = 0.0318$$

As stated above, about 74% of the memory accesses are instruction references. Thus, the overall miss rate for the split caches is

$$(74\% \times 0.004) + (26\% \times 0.114) = 0.0326$$

Thus, a 32 KB unified cache has a slightly lower effective miss rate than two 16 KB caches.

The average memory access time formula can be divided into instruction and data accesses:

$$\text{Average memory access time} = \%\,\text{instructions} \times (\text{Hit time} + \text{Instruction miss rate} \times \text{Miss penalty})$$
$$\%\,\text{data} \times (\text{Hit time} + \text{Data miss rate} \times \text{Miss penalty})$$

Therefore, the time for each organization is

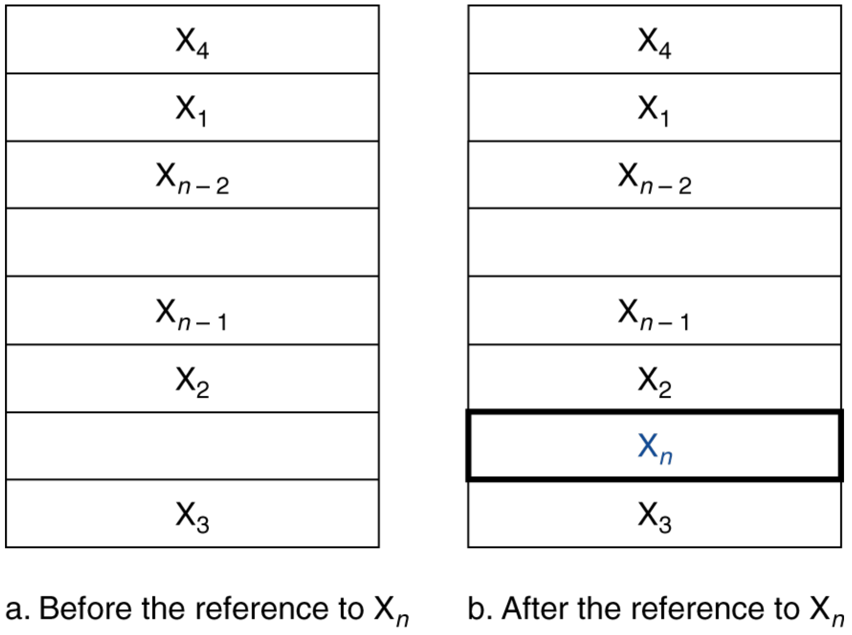
$$\text{AMAT}_{\text{split}} = 74\% \times (1 + 0.004 \times 200) + 26\% \times (1 + 0.114 \times 200) = 7.52$$

$$\text{AMAT}_{\text{unified}} = 74\% \times (1 + 0.0.0318 \times 200) + 26\% \times (1 + +1 + 0.0318 \times 200) = 7.62$$

Introduction to Cache Organization

Cache Memory

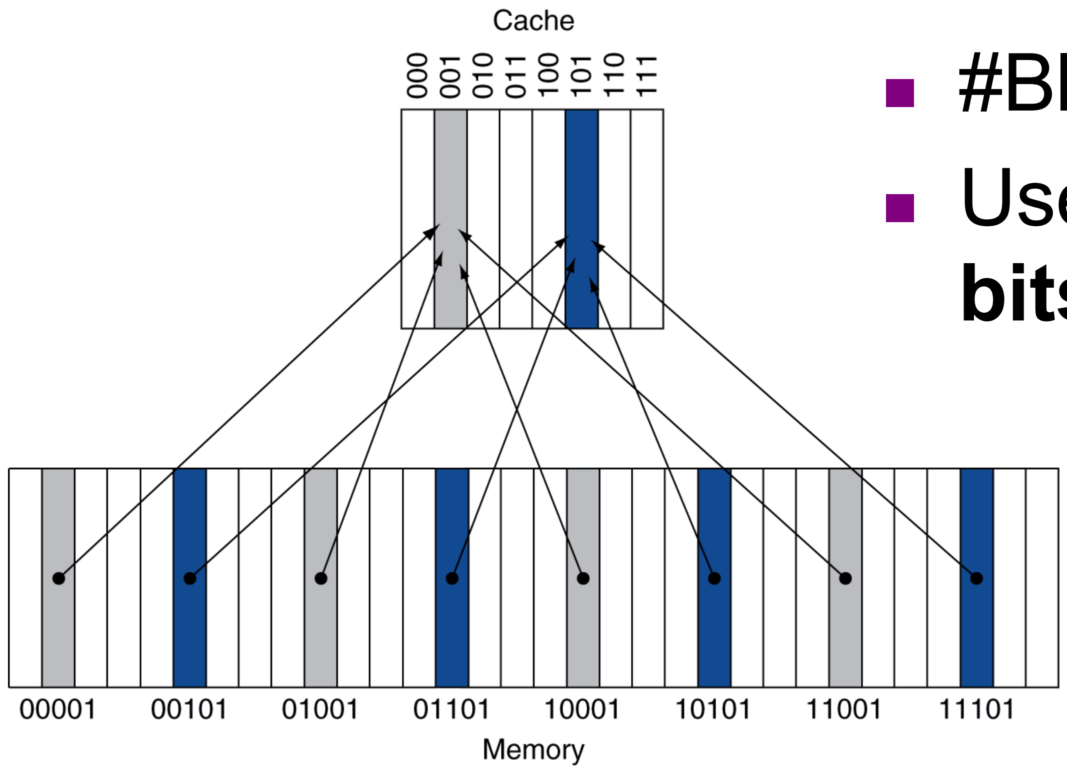
- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use **low-order address bits as index to the entry**

5-bit address space for total 32 bytes. This is simplified and a cache line normally contains more bytes, e.g. 64 or 128 bytes.

Tags and Valid Bits

- 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: the high-order bits**
- 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		

Cache Example

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

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

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		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	**10 110**	Hit	110
26	**11 010**	Hit	010

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		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	**10 000**	Miss	000
3	**00 011**	Miss	011
16	**10 000**	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

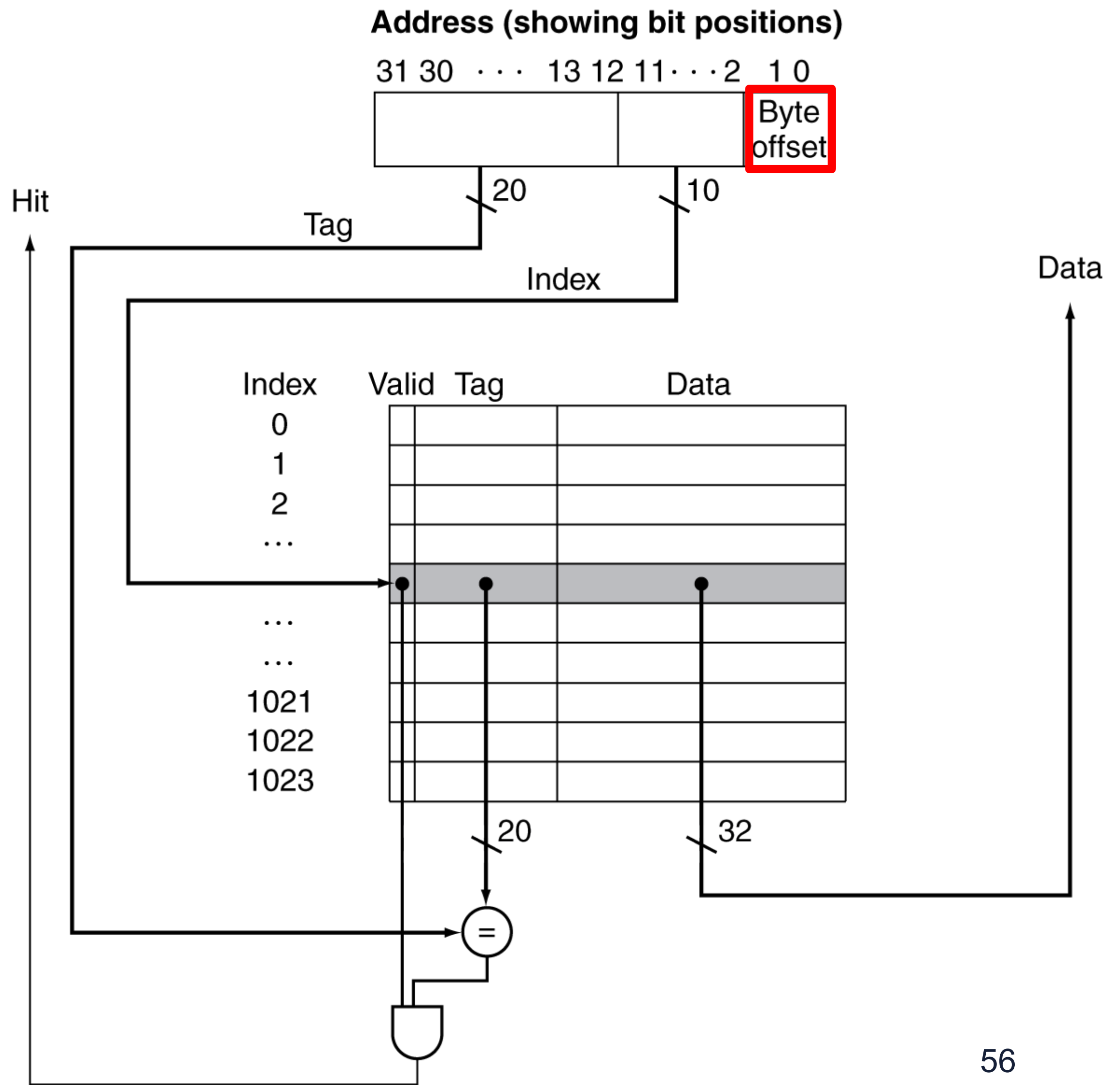
Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	**10 010**	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision

- 4-byte data per cache entry: block or line size
  - Cache line
- **Why do we keep multiple bytes in one cache line?**
  - **Spatial locality**

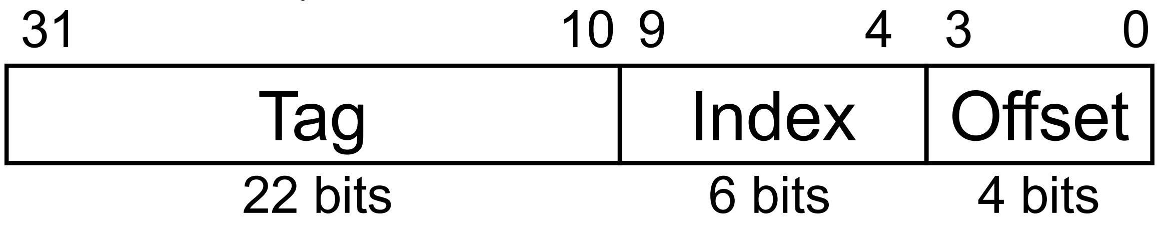


Example: Larger Block Size

- **64 blocks, 16 bytes/block**
  - To what block number does address 1200 (decimal) map?: 11

1200 = 0x4B0

$\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 01}\underbrace{\mathbf{00\ 1011}}\ \underbrace{\mathbf{0000}}$



- Map all addresses between 1200 - 1215

Block Size Considerations

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

Cache Misses

- On cache **hit**, CPU proceeds normally
  - IF or MEM stage to access instruction or data memory
  - 1 cycle

100: LW X1 100(X2)

- On cache **miss**: → x10 or x100 cycles
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy

- **Instruction cache miss**
  - **Restart instruction fetch**
- **Data cache miss**
  - **Complete data access**

Write-Through

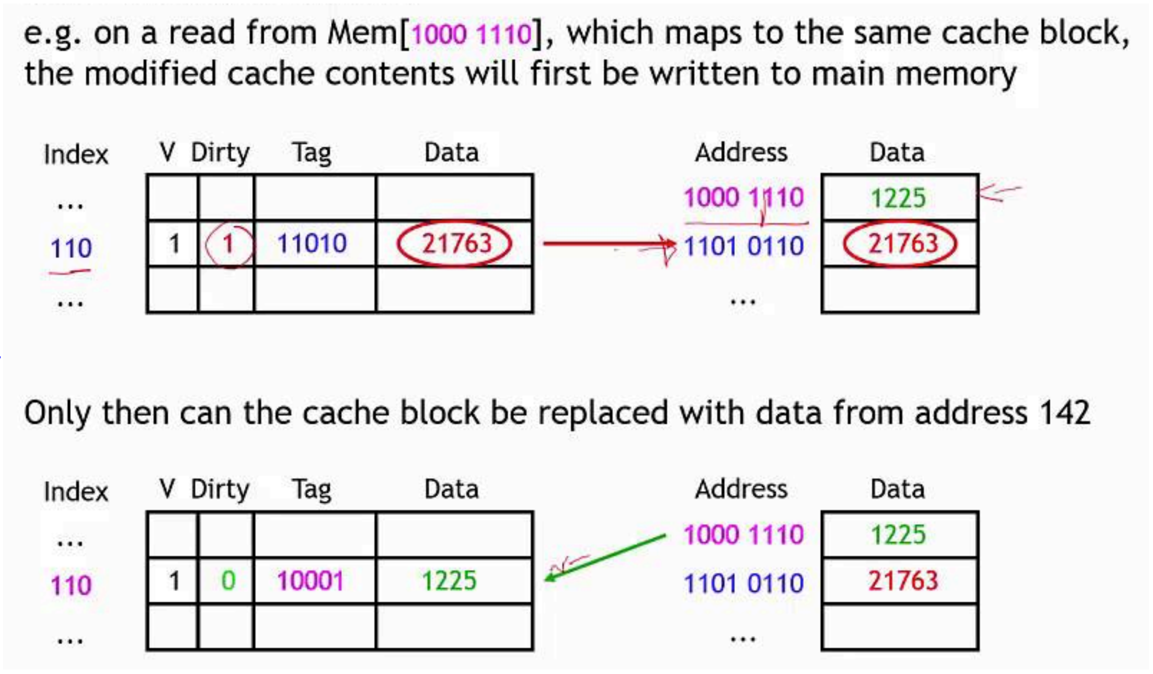
- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent

200: SW X1 100(X2)

- **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 and write multiple bytes a time

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



e.g. on a read from Mem[1000 1110], which maps to the same cache block, the modified cache contents will first be written to main memory

Index	V	Dirty	Tag	Data
...				
110	1	1	11010	21763
...				

Address	Data
1000 1110	1225
1101 0110	21763
...	

Only then can the cache block be replaced with data from address 142

Index	V	Dirty	Tag	Data
...				
110	1	0	10001	1225
...				

Address	Data
1000 1110	1225
1101 0110	21763
...	

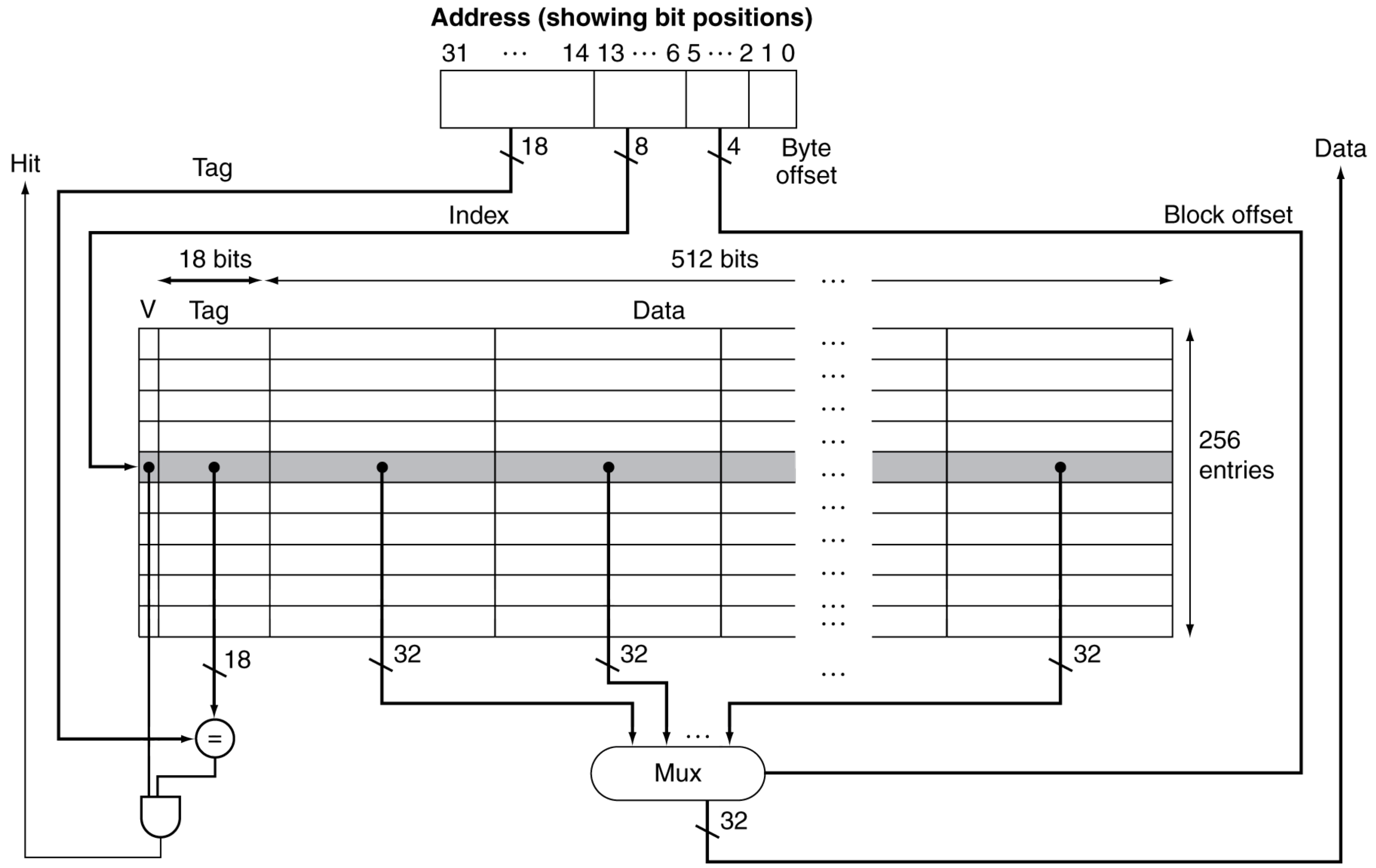
Write Allocation

- 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

Example: Intrinsity FastMATH

- 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%

Example: Intrinsity FastMATH



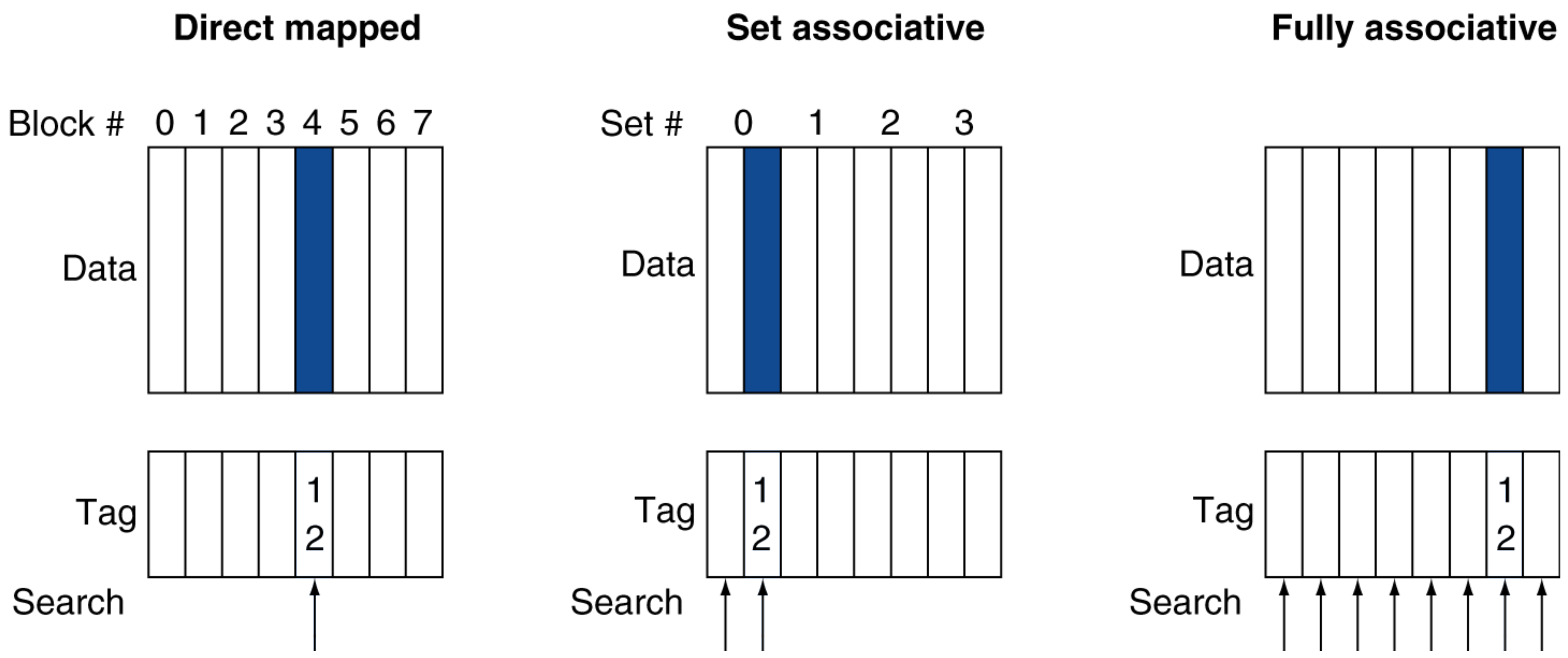
Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Associative Caches

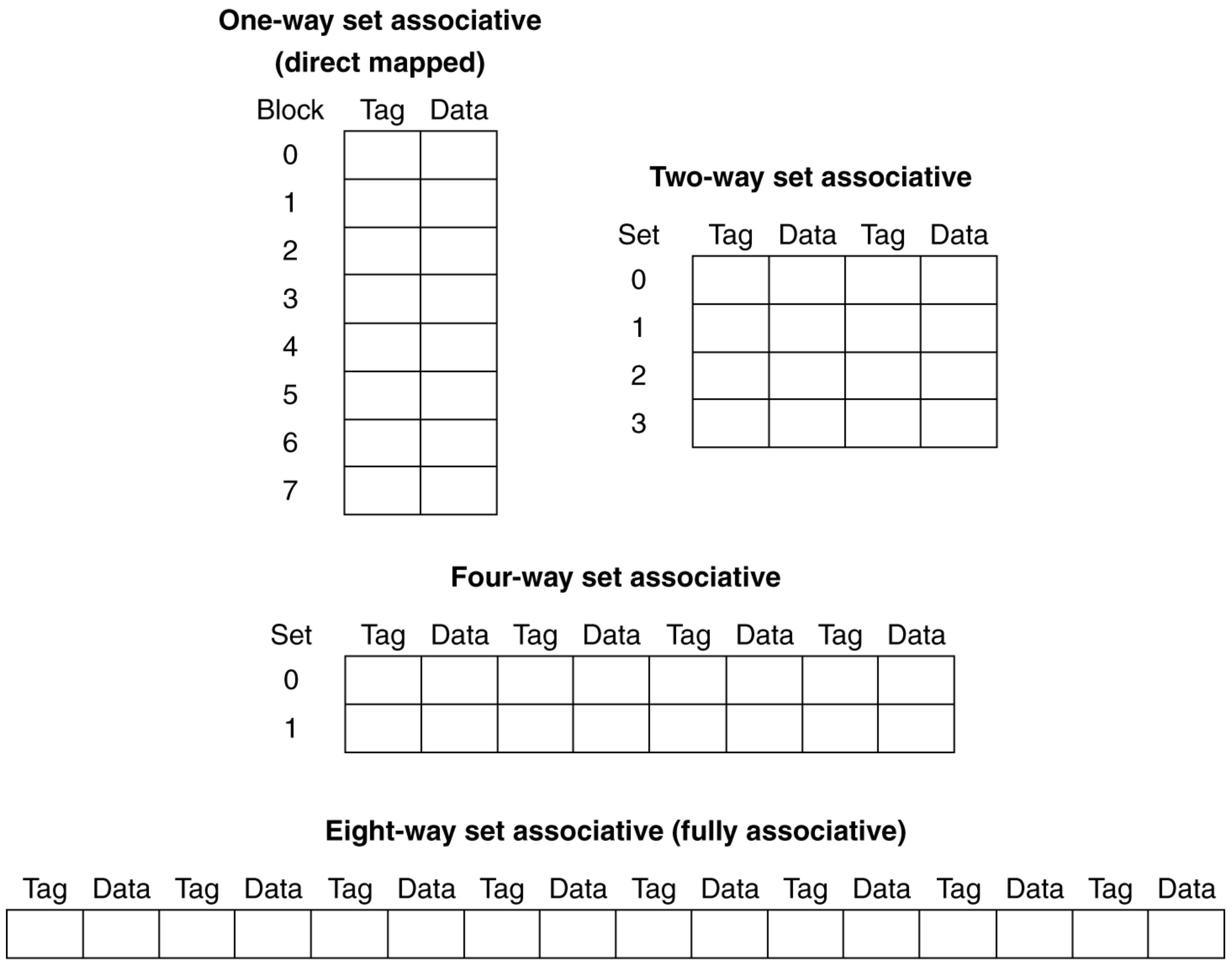
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  -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


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

Associativity Example

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

Access sequence

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]		

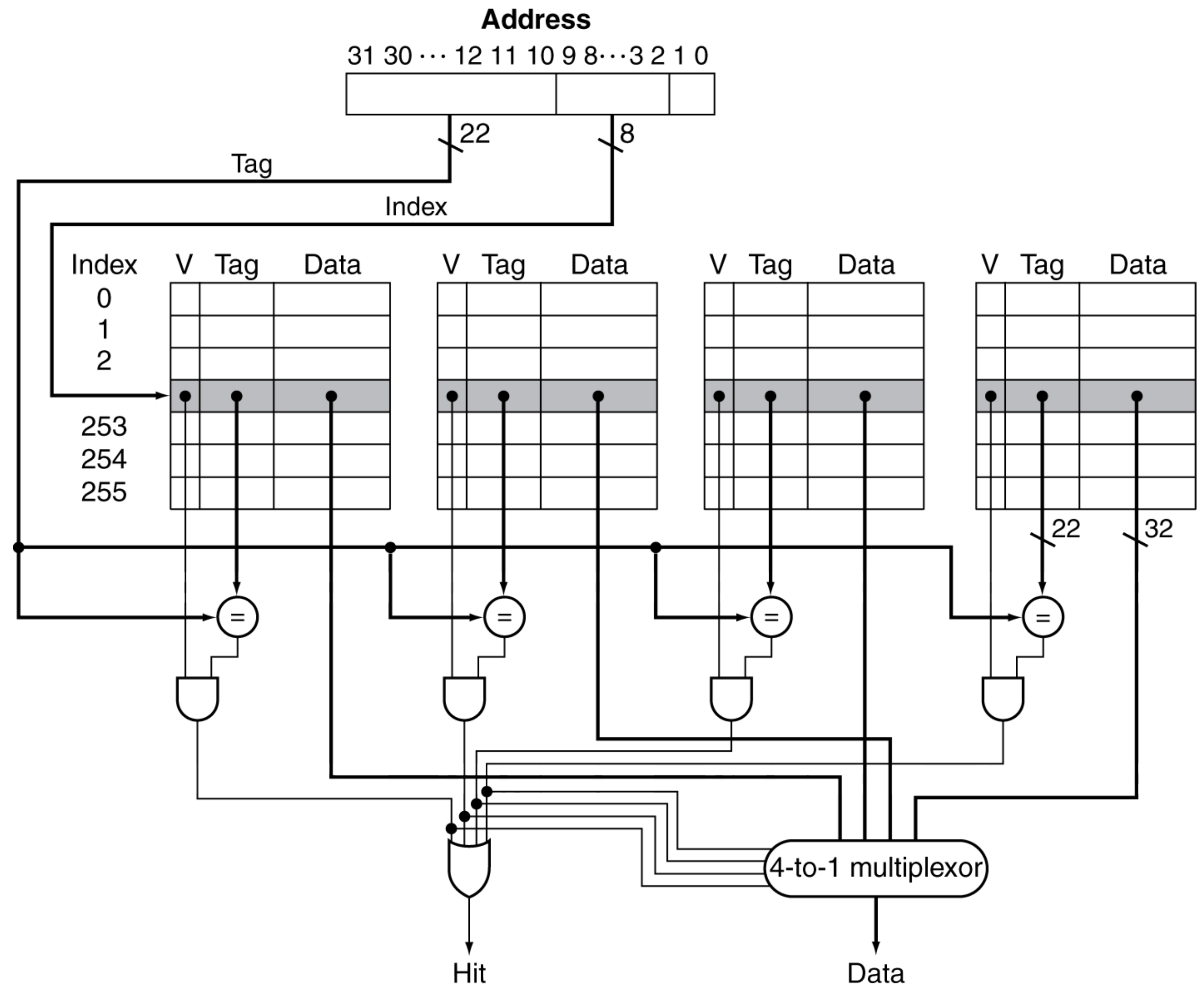
- Fully associative

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]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- 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 Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity