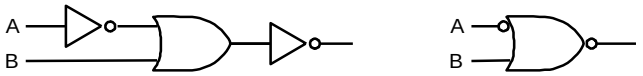
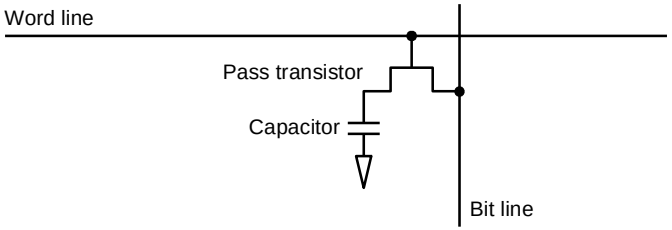
Chapter Five
Exploiting Memory Hierarchy

5.1 Introduction

- **SRAM:**
  - **value is stored on a pair of inverting gates**
  - **very fast but takes up more space than DRAM (4 to 6 transistors)**

- **DRAM:**
  - **value is stored as a charge on capacitor (must be refreshed)**
  - **very small but slower than SRAM (factor of 5 to 10)**





- **There are three primary technologies used in building memory hierarchies:**
  1. **DRAM (main memory)**
  2. **SRAM (caches)**
  3. **Magnetic Disk**

Memory technology	Typical access time	$ per GB in 2004
SRAM	0.5–5 ns	$4000–$10,000
DRAM	50–70 ns	$100–$200
Magnetic disk	5,000,000–20,000,000 ns	$0.50–$2

Locality

- **A principle that makes having a memory hierarchy a good idea**

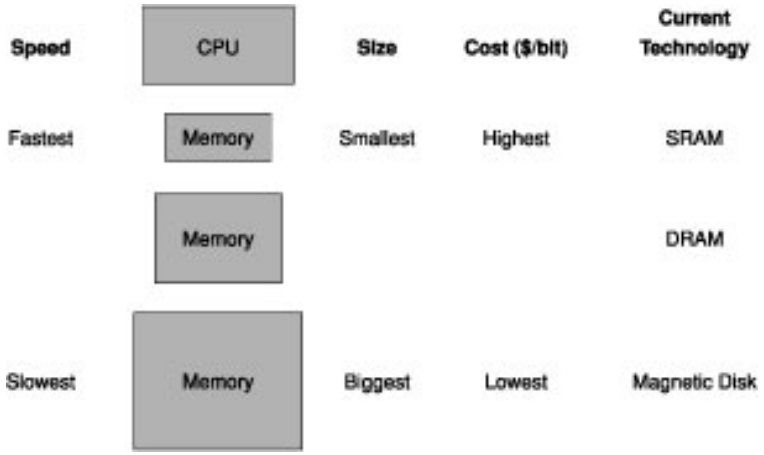
- **If an item is referenced,**

  temporal locality: it will tend to be referenced again soon
  spatial locality: nearby items will tend to be referenced soon.

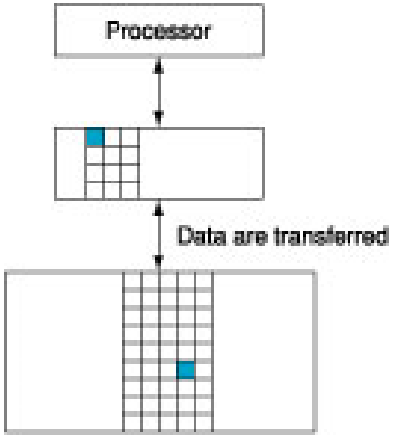
Why does code have locality?

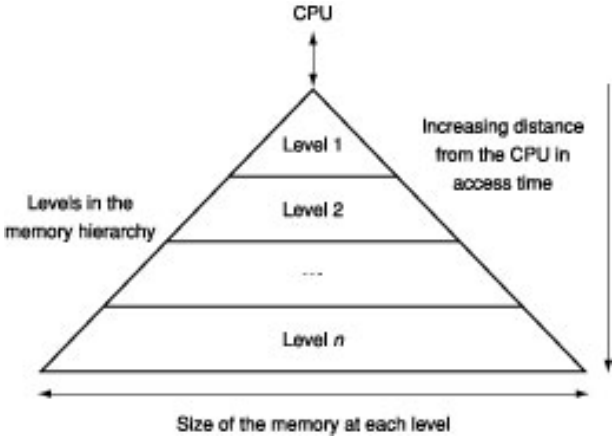
- **Our initial focus: two levels (upper, lower)**
  - **block: minimum unit of data**
  - **hit: data requested is in the upper level**
  - **miss: data requested is not in the upper level**

- **Users want large and fast memories!**
- **Build memory as a hierarchy of levels (fastest is close to the processor.**



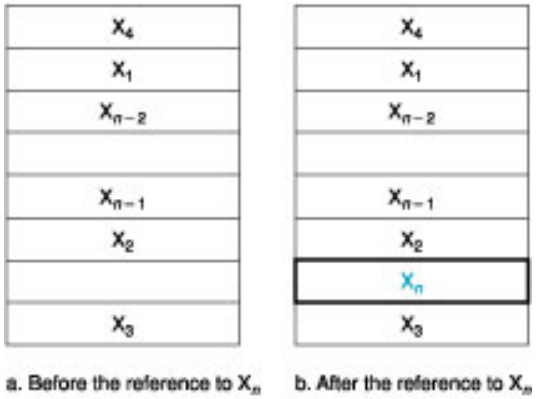
- Hit rate
- Miss rate
- Hit time
- Miss penalty





5.2 The Basic of Caches

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"



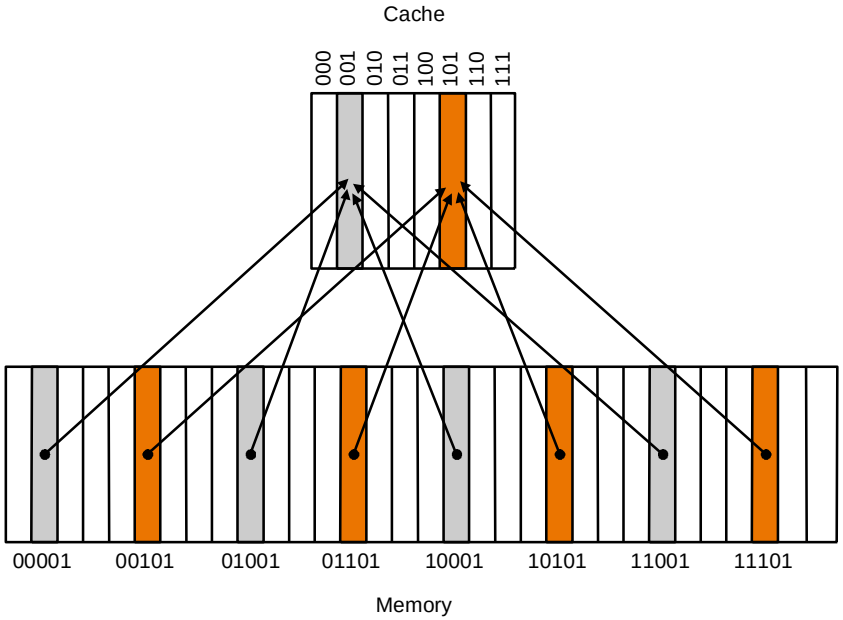
a. Before the reference to X_n b. After the reference to X_n

For each item of data at the lower level,
there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

Direct Mapped Cache

- **Mapping: address is modulo the number of blocks in the cache**



Accessing a Cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Example: An eight-word direct-mapped ca

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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

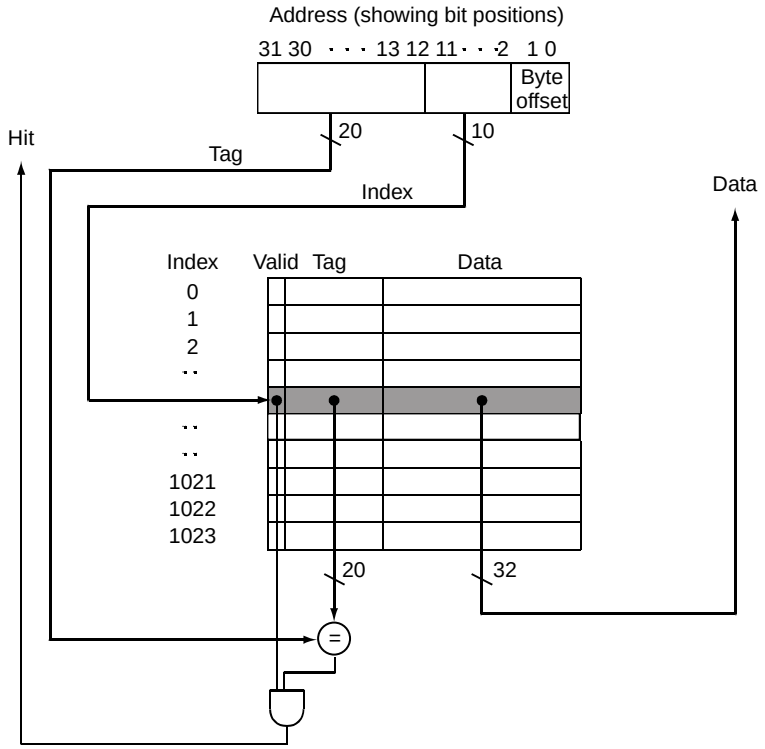
e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

Direct Mapped Cache

- **For MIPS:**



What kind of locality are we taking advantage of?

Bits in a Cache

- 2^n blocks 2^m words/block $\Rightarrow$ 32-(n+m+2)

$\Rightarrow$ total number of bits in a cache = $2^n \times$(block size + tag size + valid size)

Example: Bits in a Cache

How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?

16KB = 4K words = 2^{12} words

Block size = 4 words(2^2), 2^{10} blocks

Each block has 4×32=128 bits of data plus a tag(32-10-2-2 bits)

Thus:

the total cache size = $2^{10} \times$(128+(32-10-2-2)+1) = $2^{10} \times 147$ = 147 Kbits

Multiword Cache Block

Example: Mapping an Address to a Multiword Cache Block

Consider a cache with 64 blocks and a block size of 16 bytes. What block number does byte address 1200 map to?

(Block address) modulo (Number of cache blocks)

where the address of the block is:

$$\frac{Byte\ address}{Bytes\ per\ block}$$

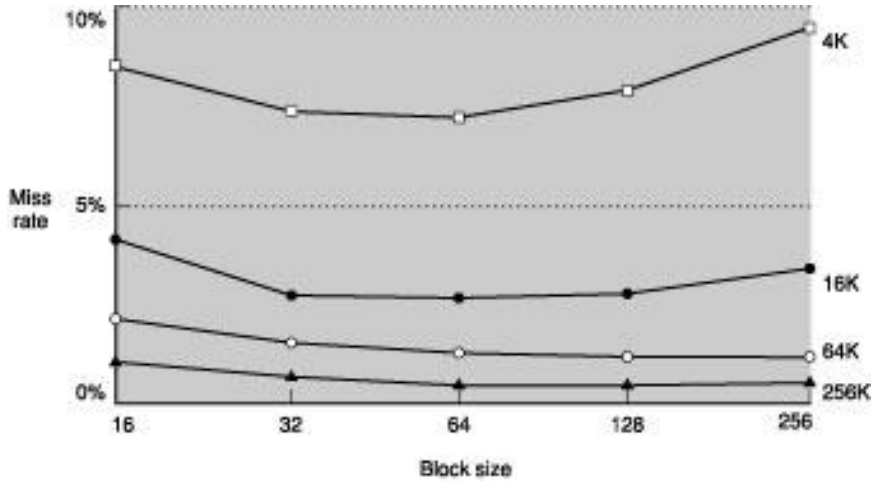
Notice that this block address is the block containing all address between:

$$\left\lfloor \frac{Byte\ address}{Bytes\ per\ block} \right\rfloor \times Byte\ per\ block \quad \&\& \quad \left\lfloor \frac{Byte\ address}{Bytes\ per\ block} \right\rfloor \times Byte\ per\ block + (Byte\ per\ block - 1)$$

Thus, with 16 bytes per block $\Rightarrow$ $\left\lfloor \frac{1200}{16} \right\rfloor = 75$

which map to cache number (75 modulo 64) = 11

Block Size && Miss Rate



Handling Cache Misses

- **Cache misses**
- **Step to be taken on an instruction cache miss:**
  1. **Send the PC (current PC-4) to the memory**
  2. **Instruct main memory to perform a read and wait**
  3. **Write the cache entry, update the data portion, tag field, and valid bit.**
  4. **Restart the instruction execution.**

Handling Write

- **write-through**
- **write buffer**
- **write-back**

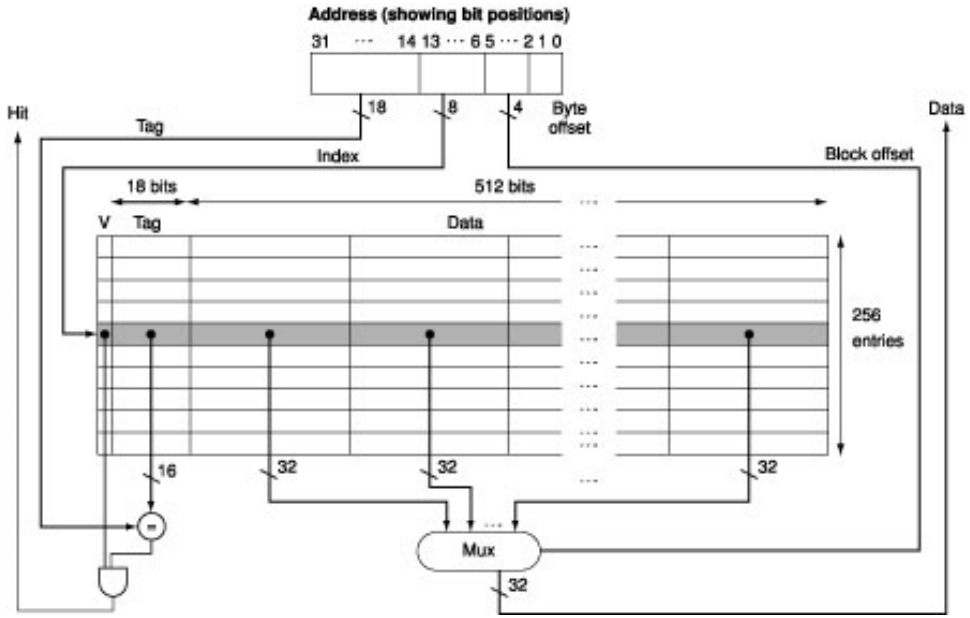
An Example Cache: The Intrinsity FastMATH processor

Cache size = 4k word

2^8=256 blocks

2^4=16 word blocks

Tag=32-8-4-2



- **The steps for a read request:**
  1. Send the address to the appropriate cache. The address come either from the PC (for an instruction) or from the ALU (for data).
  2. If the cache signal hit, the requested word is available.
  3. If the cache signals miss, send the address to the main memory….

Hits vs. Misses

- **The miss rates (instruction + data) for the FastMATH:**

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

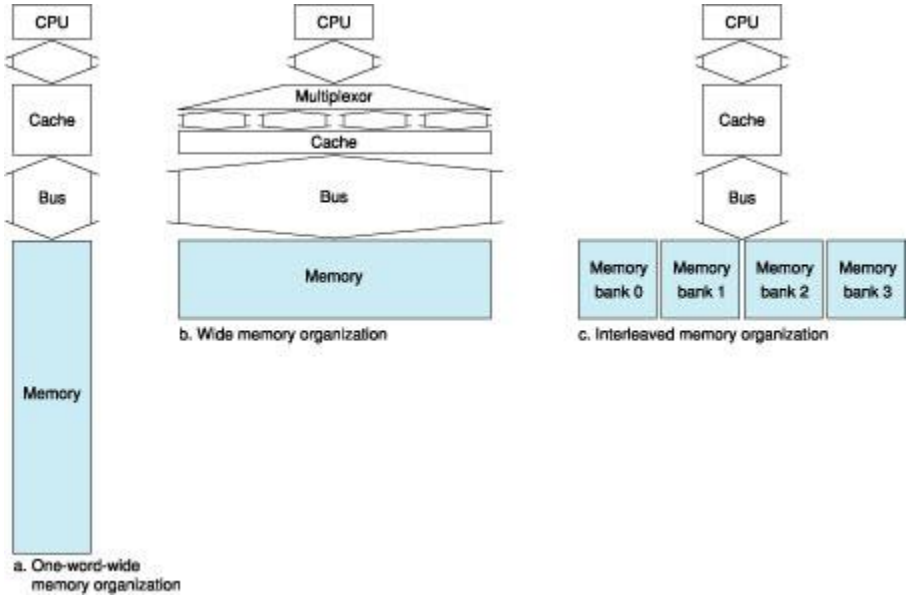
- **Read hits**
  - **this is what we want!**

- **Read misses**
  - **stall the CPU, fetch block from memory, deliver to cache, restart**

- **Write hits:**
  - **can replace data in cache and memory (write-through)**
  - **write the data only into the cache (write-back the cache later)**

- **Write misses:**
  - **read the entire block into the cache, then write the word**

Designing the Memory System to Support Caches



- **Assume:**
  1. **1 memory bus clock cycle to send the address**
  2. **15 memory bus clock cycles for each DRAM access initiated**
  3. **1 memory bus clock to send a word of data**
- **If we have a cache block of 4 words and a one-word-wide bank of DRAMs,**
  the miss penalty = 1 + 4×15 + 4×1 = 65
  Thus, bytes transferred per clock for a single miss = $\frac{4 \times 4}{65} = 0.25$

- **Widening the memory and the bus:**
  - **memory width of 2 words:**

    the miss penalty = 1 + 2×15 + 2×1 = 33

    bytes transferred = 16/33 = 0.48

  - **memory width of 4 words:**

    the miss penalty = 1 + 15 + 1 = 17

    bytes transferred = 16/17 = 0.94
- **Interleaved memory organization:**

  the miss penalty = 1 + 1×15 + 4×1 = 20

  bytes transferred = 16/20 = 0.80

5.3 Measuring and Improving Cache Performance

- **CPU time = (CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time**

- **Memory-stall clock cycles = Read-stall cycles + Write-stall cycles**

  Where,

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

  Write-stall cycles =($\frac{\text{Writes}}{\text{Program}}$ × Write miss rate × Write miss penalty) + Write buffer stalls

- **In most write-through cache, the read and write miss penalties are the same. If we assume that the write buffer stalls are negligible, we can combine the read and writes by using a single miss rate and the miss penalty:**

  Memory-stall clock cycles = $\frac{\text{Memory access}}{\text{Program}}$ × Miss rate × Miss penalty

  We can also factor this as:

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

Calculating Cache Performance

Example:

Assume an instruction cache miss rate for a program is 2% and a data cache miss rate is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed.

Instruction miss cycles = I×2%×100=2.00×I

The frequency of load and store in SPECint2000 is 36%

Data miss cycles = I × 36% ×4% × 100 = 1.44×I

The total number of memory stall cycles = 2×I + 1.44×I = 3.44×I

Accordingly, CPI = 2 + 3.44 = 5.44

Since there is no change in Ic or clock rate, the ratio of the CPU execution time is:

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I_c \times CPI_{stall} \times Clock\ cycle}{I_c \times CPI_{perfect} \times Clock\ cycle}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

The performance with the perfect cache is better by $\frac{5.44}{2} = 2.72$

Continue

- **Suppose we speed up the computer in the previous example by reducing its CPI from 2 to 1 without changing the clock rate,**

  CPI = 1 + 3.44 = 4.44

  The system with the perfect cache would be:

$$\frac{4.44}{1} = 4.44 \text{ time faster}$$

  The amount of execution time spent on memory stalls would have risen from

$$\frac{3.44}{5.44} = 63\%$$

  to

$$\frac{3.44}{4.44} = 77\%$$

Cache Performance with Increased Clock Rate

Example:

Suppose we increase the performance of the computer in the previous example by double its clock rate. Since the main memory speed is unlikely to change, assume that the absolute time to handle a cache miss does not change. How much faster will the computer be with the faster clock, assuming the same miss rate as the previous example?

The new miss penalty will be twice as many clock cycles, or 200 cycles. Hence:

Total miss cycles per instruction = (2%×200) + 36%×(4%×200) = 6.88

CPI = 2 + 6.88 = 8.88 compared to 5.44 for the slower computer.

the relative performance:

$$\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{\text{execution time with slow clcok}}{\text{execution time with fast clcok}}$$

$$= \frac{\text{Ic} \times \text{CPI}_{\text{slow clock}} \times \text{Clock cycle}}{\text{Ic} \times \text{CPI}_{\text{fast clock}} \times \frac{\text{Clock cycle}}{2}}$$

$$= \frac{5.44}{8.88 \times \frac{1}{2}} = 1.23$$

Thus, the computer with the faster clock is about 1.2 time faster rather than 2 time faster.

Reducing Cache Misses by More Flexible Placement of Blocks

Placement Schemes:

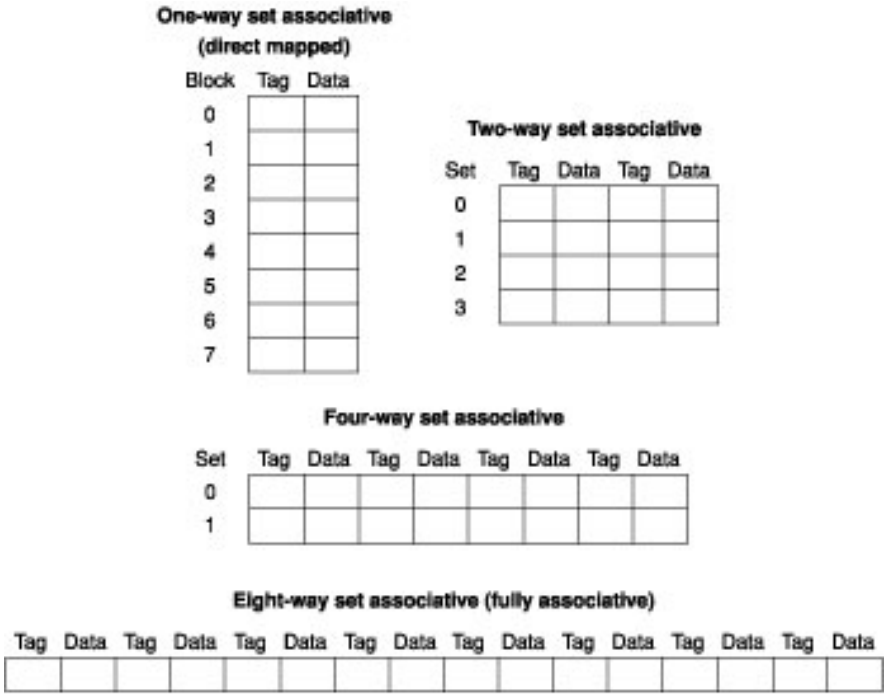
1. **Direct Mapped:** a block can go in exactly one place
2. **Set Associative:** a fixed number of locations where each block can be placed
3. **Fully Associative:** a block can be placed in any location

Direct Mapped: ⇒ (Block number) modulo (Number of cache blocks)

Set Associative: ⇒ (Block number) modulo (Number of sets in the cache)

Continue

- **The possible associatively structures for an eight-block cache.**



Misses and Associativity in Caches

Example:

Three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set associative, and the third is direct mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, 8.

Direct mapped:

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Example (Continue)

Set associative:

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

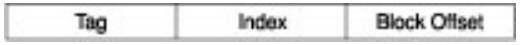
Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Fully associative:

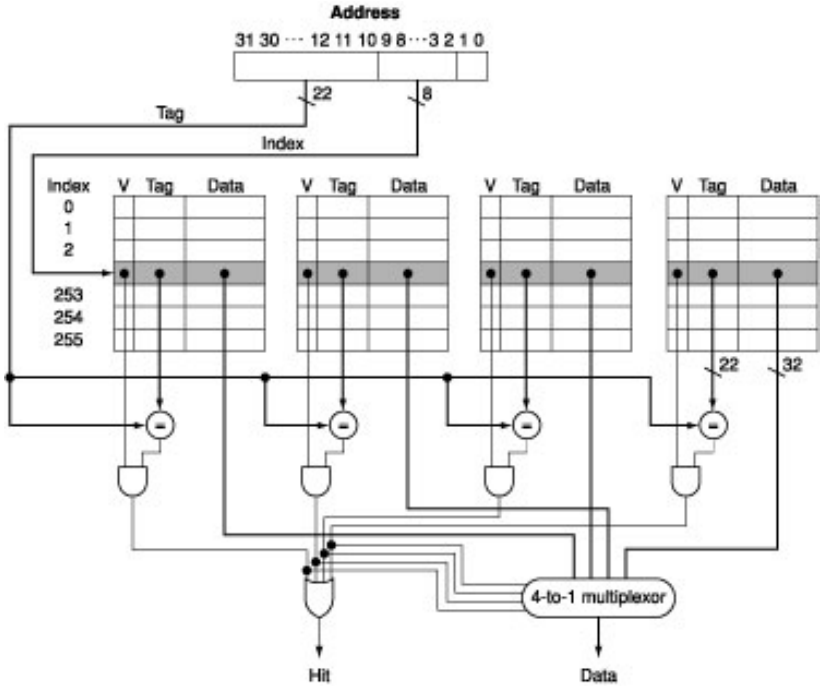
Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Locating a Block in the Cache

- All tags in the selected set are searched in parallel
- Address decomposition:



  - Tag
  - Index
  - Block Offset



Choosing Which Block to Replace

- **Least recently Used (LRU)**

Reducing the Miss Penalty Using Multilevel Caches

- **Primary Cache**
- **Secondary Cache**

Example: