Large and Fast: Exploiting Memory Hierarchy

Chapter Five

Of

Book of David A. Patterson

Processor Performance

- ✔ CPU time = (CPU execution clock cycles + Memory stall clock cycles) × clock cycle time
- ✔ Memory stall clock cycles = read stall cycles + write stall cycles
- ✔ Read stall clock cycles = reads / program × read miss rate × read miss penalty
- ✔ Write stall clock cycles = (writes / program × write miss rate × write miss penalty) + write buffer stalls

Processor Performance

- ✔ Memory stall clock cycles = memory accesses / program × miss rate × miss penalty
- ✔ Memory stall clock cycles = instructions / program × miss / instruction × miss penalty

Processor's Performance- 1

- Instruction cache miss rate = 2%
- Data cache miss rate = 4%
- CPI = 2 without memory stalls
- Miss penalty = 100 cycles
- Frequency of load and store = 36%

1. Instruction miss cycles = I × 2% × 100 = 2.00 × I
2. Data miss cycles = I × 36% × 4% × 100 = 1.44 × I
3. Total = 2.00 × I + 1.44 × I = 3.44I
4. CPI with memory stalls = 2+ 3.44 = 5.44

102

Processor's Performance- 1

5. CPU time with stalls / CPU time with perfect cache = $(I \times CPI_{stall} \times \text{Clock cycle}) / (I \times CPI_{perfect} \times \text{Clock cycle})$ $= CPI_{stall} / CPI_{perfect} = 5.44/2 = 2.72$

6. CPU time with stalls = 2.72 CPU time with perfect cache

CPI ? 2

Processor's Performance - 2

Performance with lower CPI:

1. CPI is reduced to 1 from 2.
2. CPI with memory stalls = 1+3.44 = 4.44
3. CPU time with stalls / CPU time with perfect cache = CPI_{stall} / $CPI_{perfect}$ = 4.44 / 1 =4.44 faster
4. CPU time with stalls = 4.44 CPU time with perfect cache

2

Processor's Performance with Increased Clock Rate

Performance with Increased Clock Rate

1. Increase the clock rate by 2.
2. Miss penalty = 200 clock cycles.
3. Total miss cycles per instructions = (2% × 200) + 36% × (4% × 200) =6.88
4. CPI = 2+6.88 = 8.88
5. Execution time with slow clock / Execution time with fast clock = (IC × CPI$_{\text{slow clock}}$ × clock cycle) / (IC × CPI$_{\text{fast clock}}$ × clock cycle/2) = 5.44 / 8.88 *.5 = 1.23

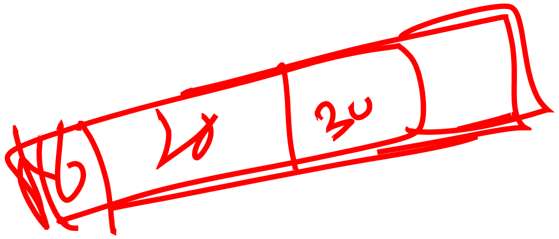
Reducing Cache Misses

- ✔ Two ways:
  1. Fully associative cache
  2. Set associative cache

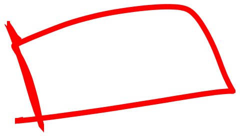
Fully Associative Cache

- ✔ A memory block can be placed in any location in the cache.
- ✔ To find a given block all the entries in the cache must be searched.
- ✔ A comparator is associated with each cache entry increasing the hardware cost.

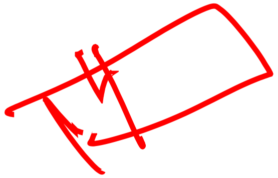
16



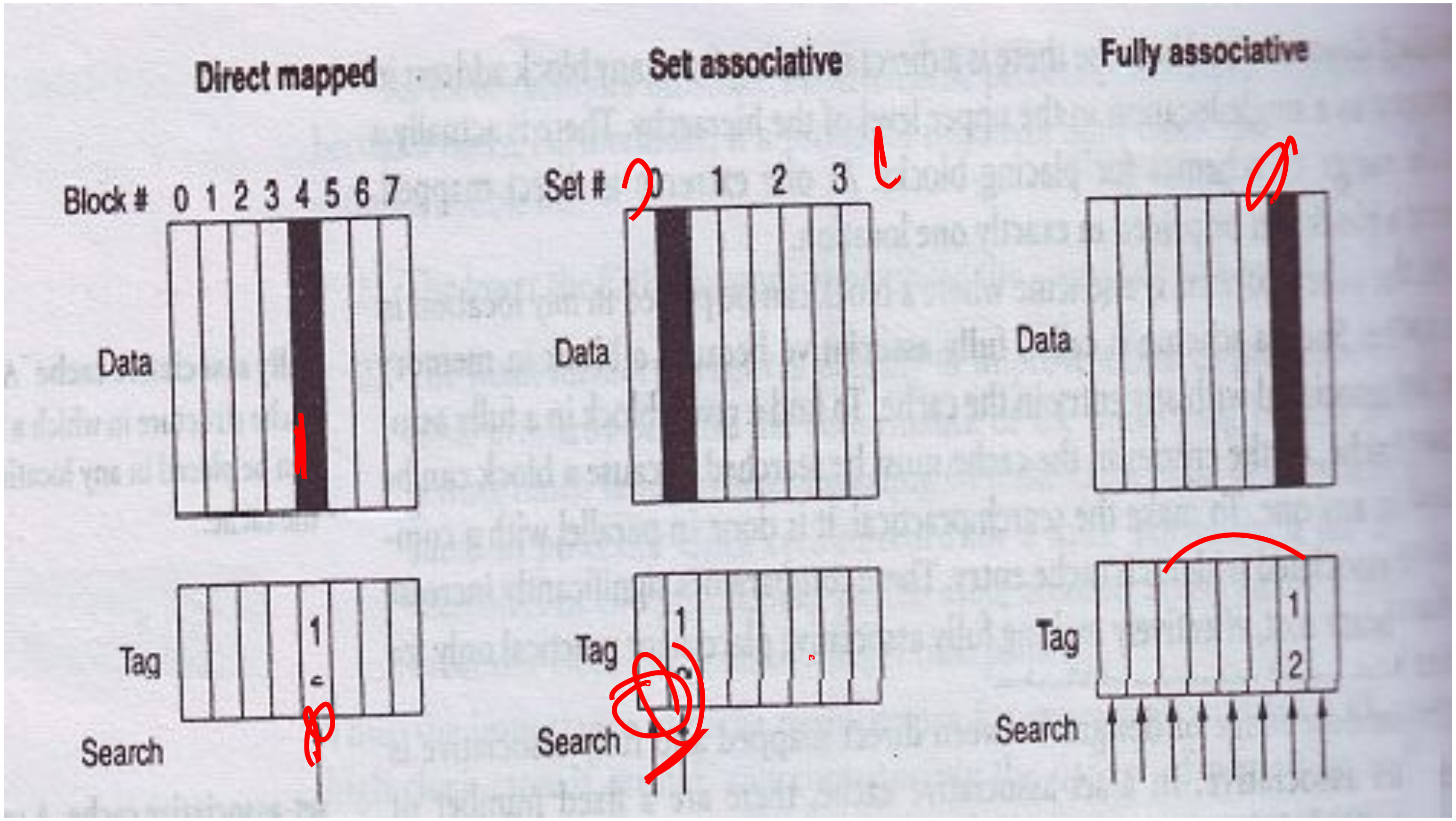
Set Associative Cache



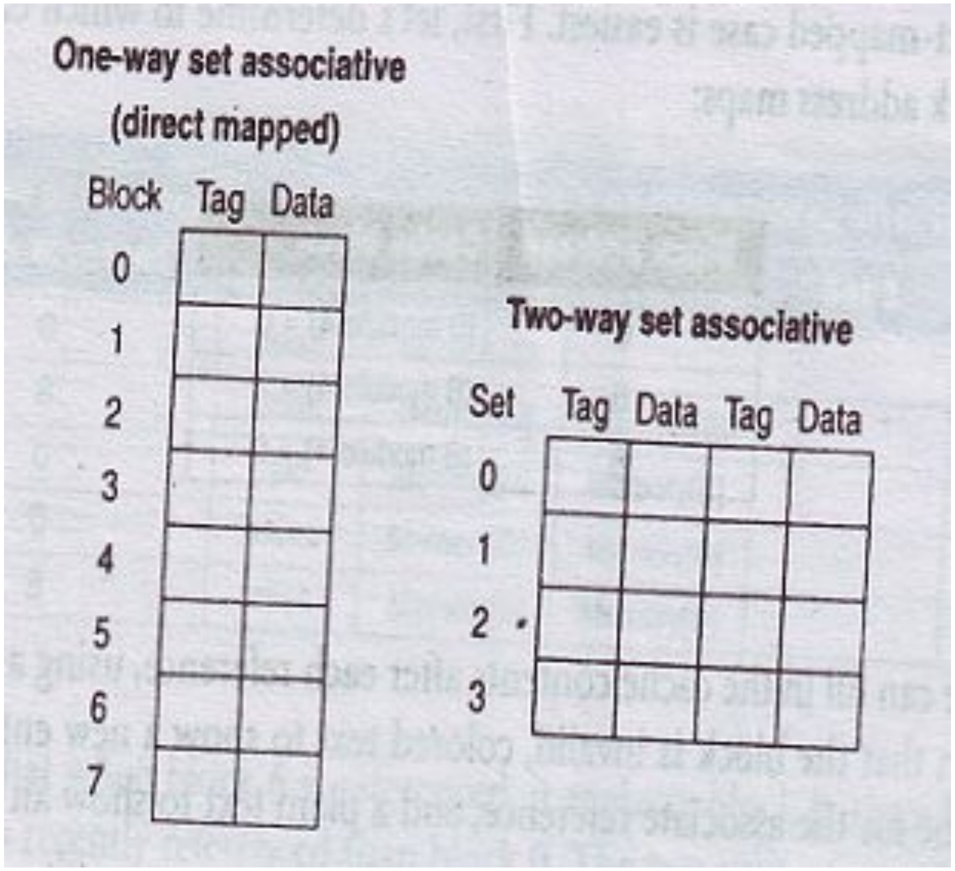
- ✔ There are a fixed number of locations where each block can be placed.
- ✔ A n-way set associative cache consists of a number of sets, each of which consists of n blocks.
- ✔ Each block in the memory maps to a unique set in the cache given by the index field and a block can be placed in any elements of the set.
- ✔ In combines both direct mapping and fully associative placement.
- ✔ Set number = (Block number) modulo (Number of sets in the cache).

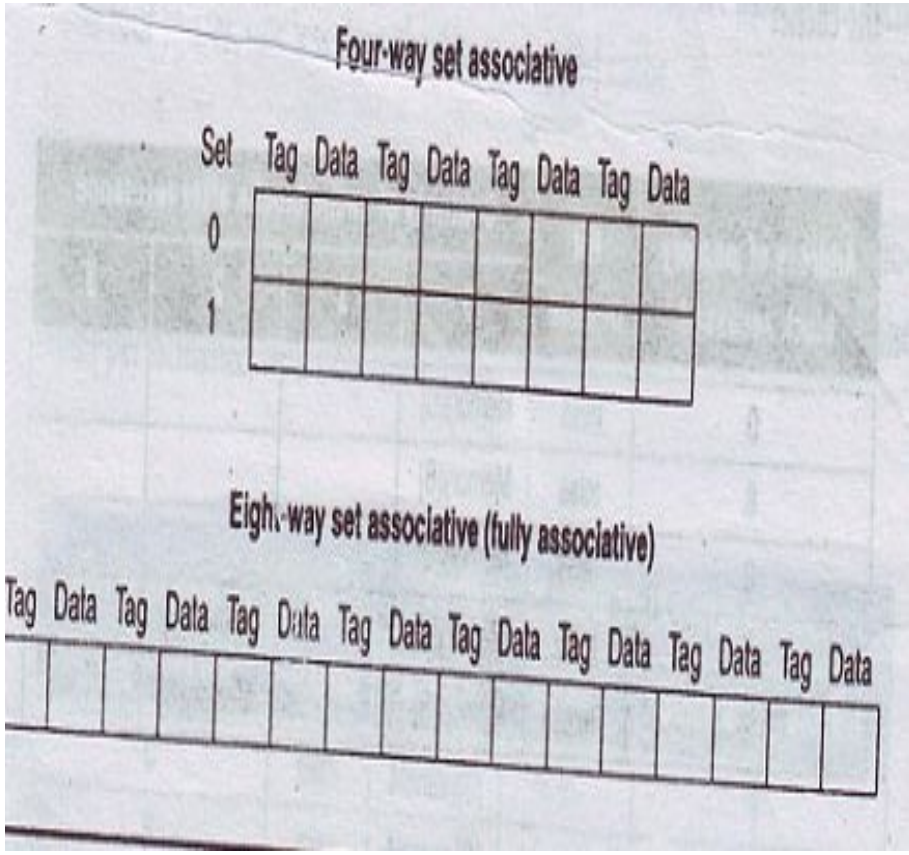


Memory Block Mapping in the Cache



Cache Configuration

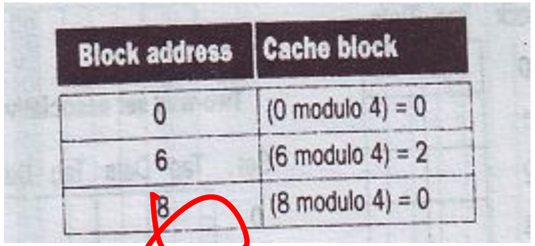




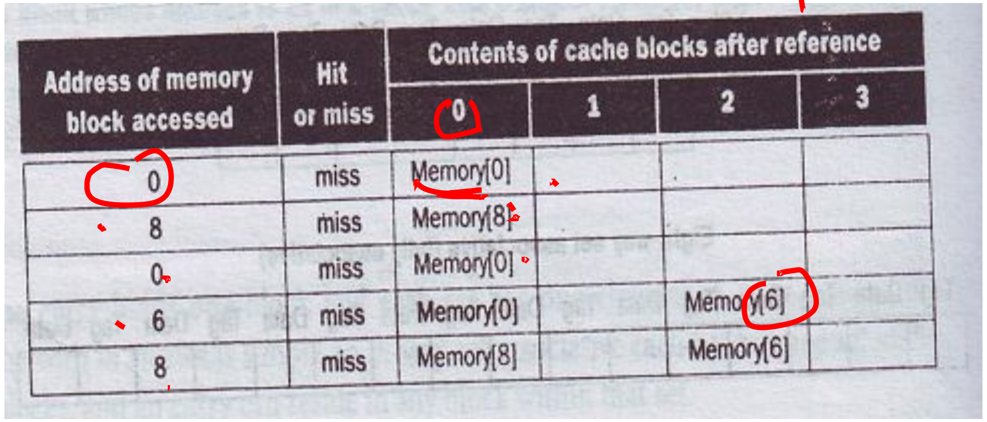
Memory Accesses in Cache

- ✔ Cache size = four one-word blocks.
- ✔ Block addresses = 0, 8, 0, 6, 8

<u>Direct Mapped Cache:</u>



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

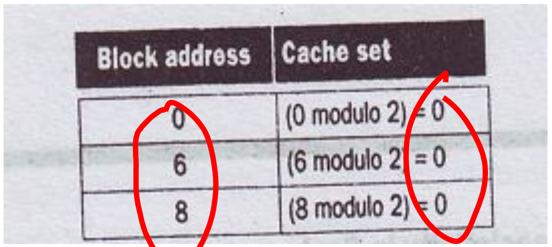


Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

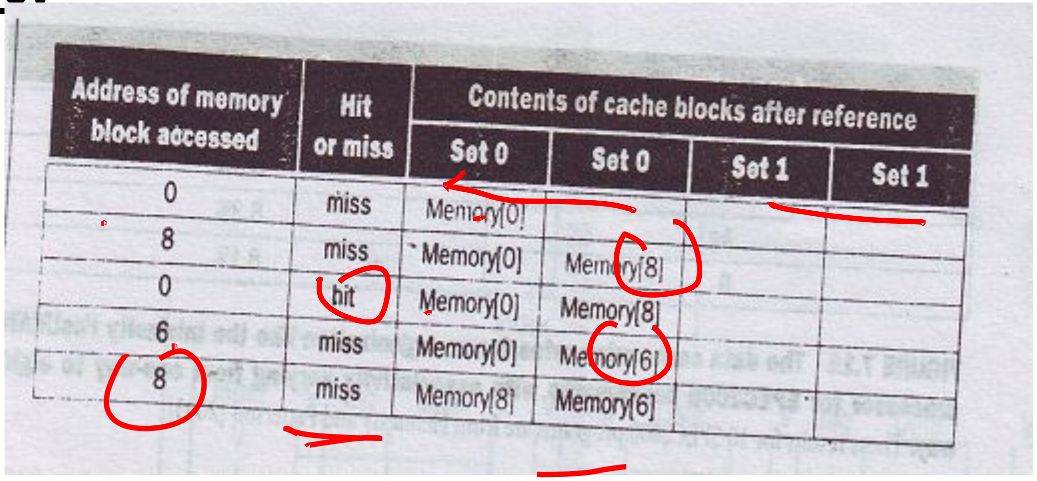
Five misses for five accesses.

Memory Accesses in Cache

<u>Two Way Set Associative:</u>



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

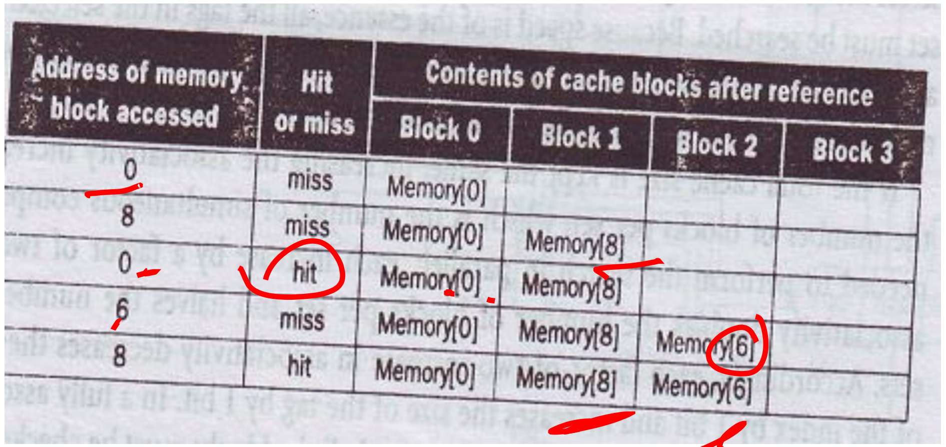


Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Four misses on five accesses

Memory Accesses in Cache

Fully Associative Cache:

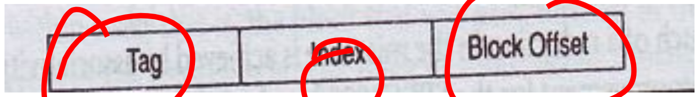


Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Three misses on five memory accesses

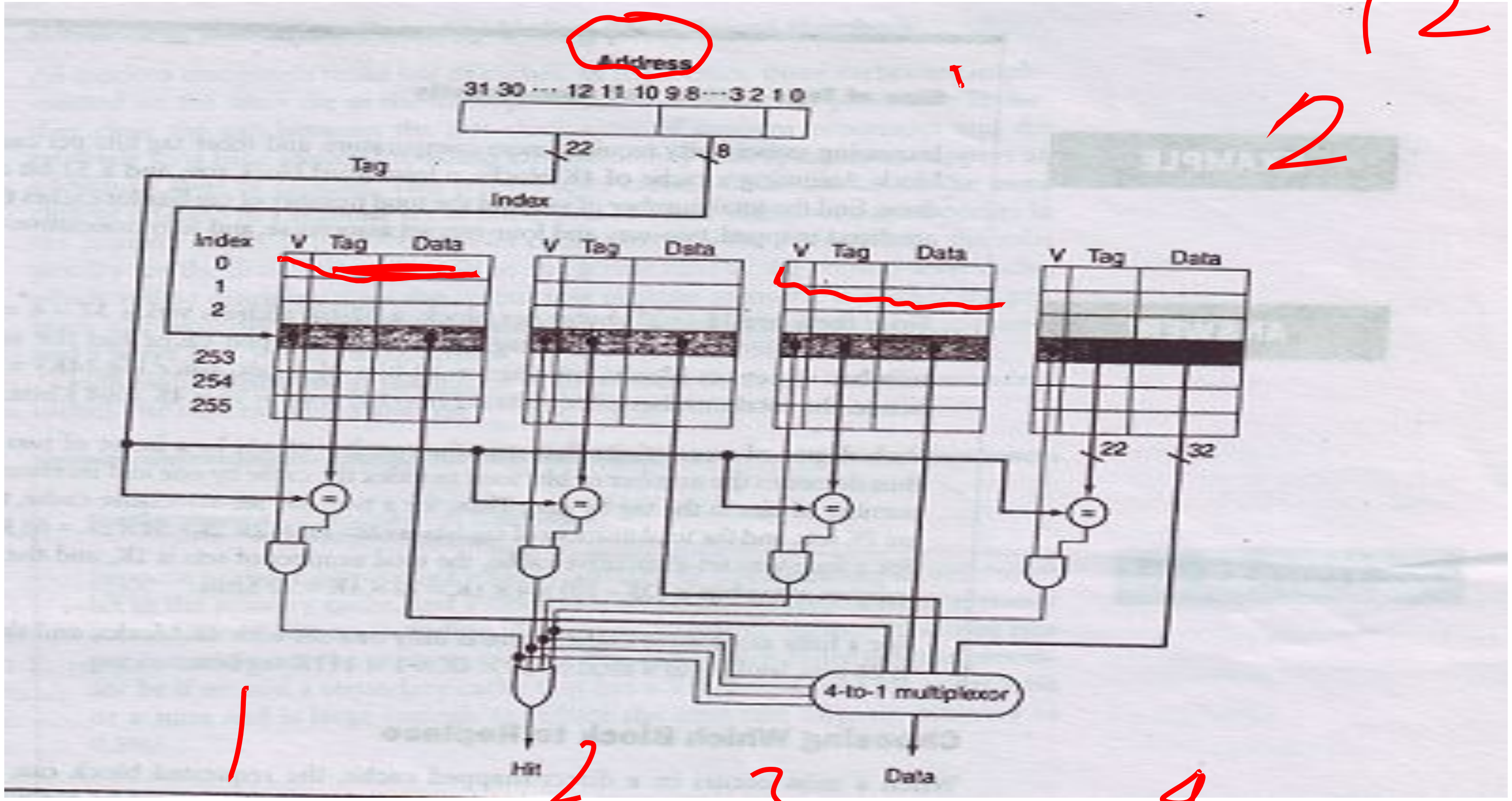
Locating Block in the Cache

- ✔ Memory address is decomposed as-



- ✔ Increasing associativity increases the number of blocks per set.
- ✔ Increase by a factor of 2 in associativity doubles the number of blocks in a set and halves the number of sets.
- ✔ Factor of 2 increase in associativity decrease the size of the index by 1 and increases the size of tag by 1 bit.
- ✔ There is no index for fully associative memory.

Implementation of Four-way Set Associative Cache



Size of Tag

- ✔ Cache size = 4K blocks
- ✔ Block size = 4 words
- ✔ Address = 32 bits

Direct mapped cache:

- ✔ Number of sets = 4K
- ✔ Index = 12 bits
- ✔ Number of tags=32-12-4= 16 bits.

$2^2 \times 2^{10} = 2^{12}$

block

Size of Tag

Two way Set Associative:

- ✔ Number of sets = 4K/2 = 2K
- ✔ Index = 11 bits.
- ✔ Tag = 32 - 11 -4 =17 bits.

Four way Set Associative:

- ✔ Number of sets = 1K
- ✔ Index = 10 bits
- ✔ Tag = 32 - 10 -4 =18

Size of Tag

Fully Associative:

- ✔ Number of sets = 1
- ✔ Index = 0
- ✔ Tag = 32 – 0 -4 =28