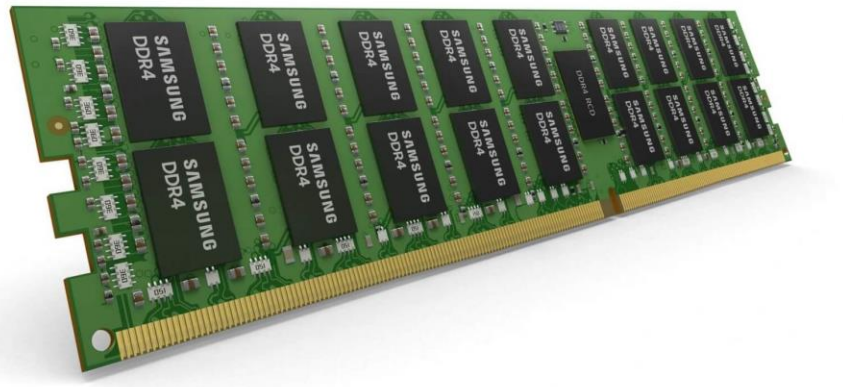


MEMORY

CHAPTER 5

CS2400 SPRING 2020

LOCALITY

- Locality of reference
  - ***Temporal locality***: The locality principle stating that if a data location is referenced then it will tend to be referenced again soon.
  - ***Spatial locality***: The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

WORKSHEET Q2

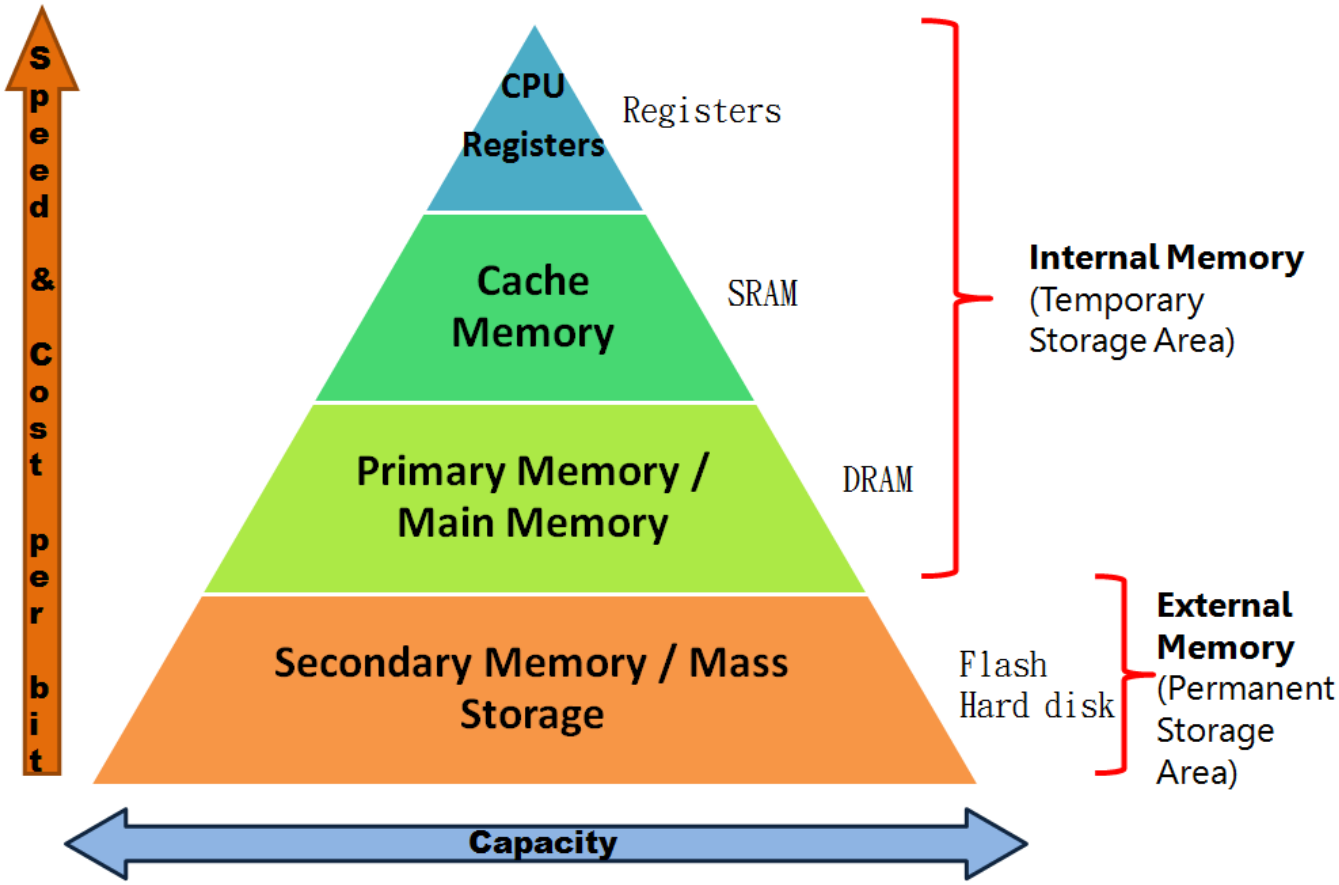
1. Given the following loop, the high likelihood of accessing multiple elements within array A is an example of _____ locality.

```
while (i < 10){
A[i] = A[i] + 2;
i = i + 1; }
```

2.Given the following loop, the high likelihood of accessing `i = i + 1` repeatedly is an example of _____ locality.

MEMORY HIERARCHY

- ***Memory hierarchy***: A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.



MEMORY

- ***Hit rate***: The fraction of memory accesses found in a level of the memory hierarchy.
- ***Miss rate***: The fraction of memory accesses not found in a level of the memory hierarchy.

- ***Hit time***: The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.
- ***Miss penalty***: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

MEMORY TECHNOLOGY

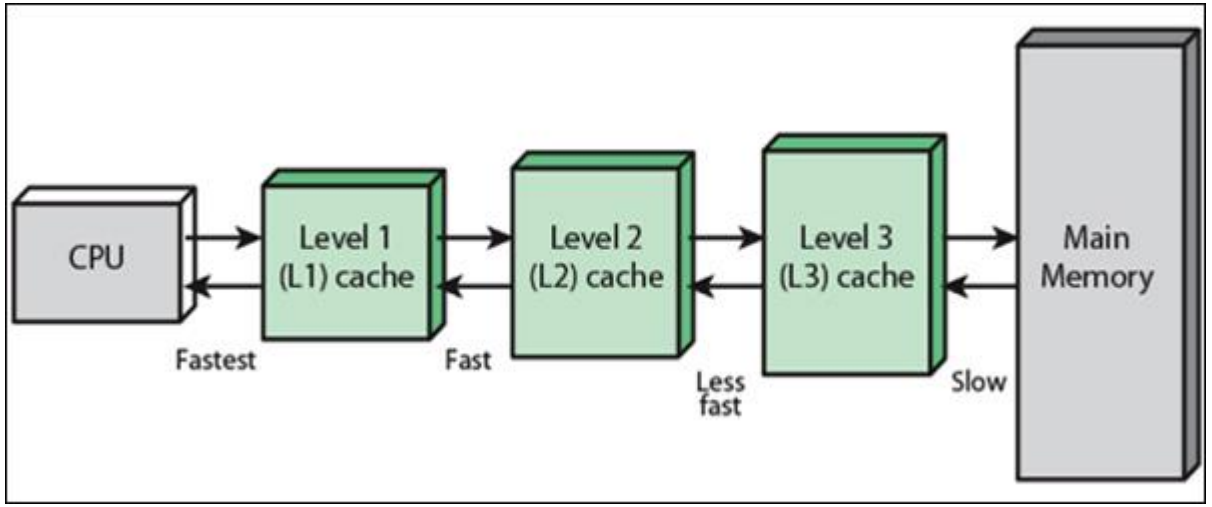
- Random Access memory
  - Data and machine code currently being used.

- Static RAM (SRAM)
  - Cache Memory
  - Fast
  - Six transistors to store data.

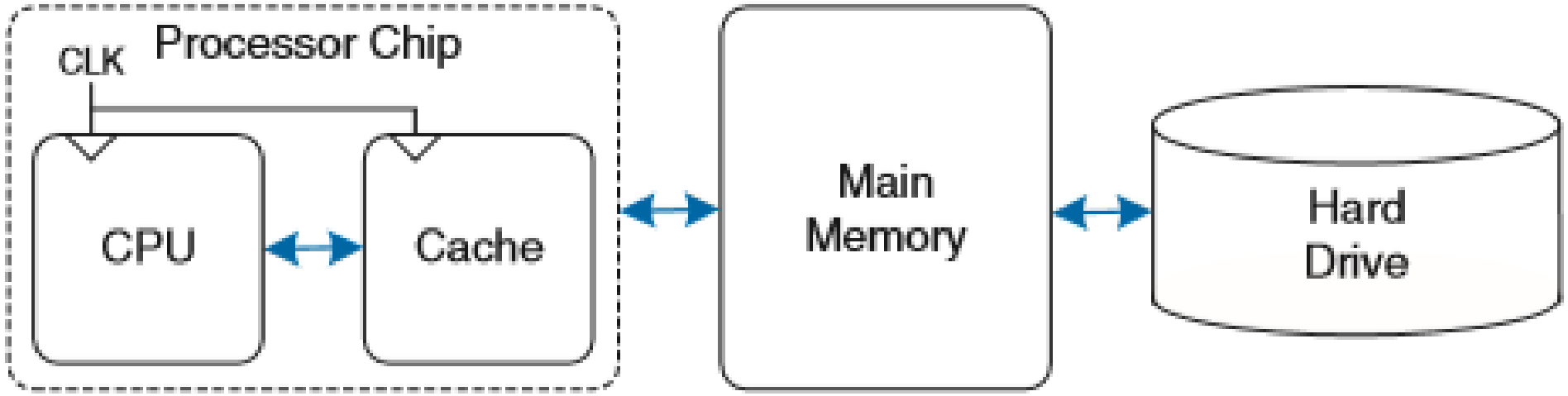
Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

- Dynamic RAM (DRAM)
  - Main Memory
  - Relatively slow.

- Cache Memory is the Memory which is very near to the central processing unit .
- Lesser access time than memory .
- Very expensive.
- Less Capacity.
- A computer can have different levels and sizes of cache depending on the CPU architecture. The most common levels of cache are L1 and L2 cache, where L1 is closest to the CPU and hence its access time is much faster compared to L2 cache



CACHE



CACHE

- ***Direct-mapped cache***: A cache structure in which each memory location is mapped to exactly one location in the cache.
- Cache index = Block address % Number of Blocks in Cache

- Example
- 8-block cache
- Cache index = Block address % 8
- Q1 in Worksheet ?
- 011 and 101

CACHE

- ***Index:*** *For 8 cache line =>2^3 ; last 3 bits represent the index.*

- ***Tag***: A field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word. All bits other than the index is tag.

- ***Valid bit***: A field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains valid data.

CACHE MAPPING

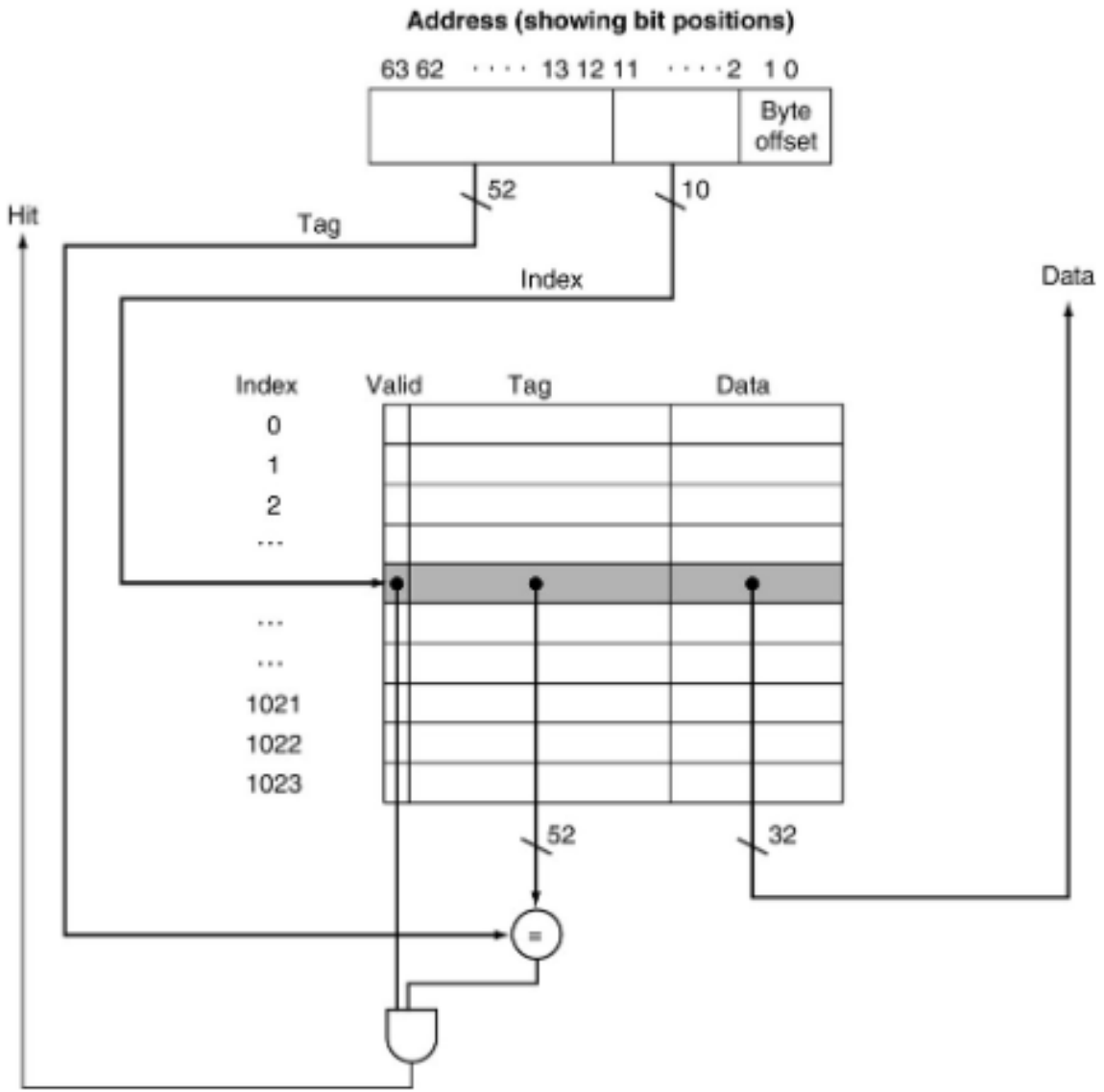
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

CACHE

- Example
- 10111
- 10000
- 11101
- 11101

- 10111(Hit)
- 10000(Miss)
- 11101(Miss)
- 11101(Hit)

Index	V	Tag	Data
000	Y	01_{two}	Memory (01000_{two})
001	Y	11_{two}	Memory (11001_{two})
010	N		
011	N		
100	Y	00_{two}	Memory (00100_{two})
101	N		
110	N		
111	Y	10_{two}	Memory(10111_{two})


Address (showing bit positions)
63 62 · · · · 13 12 11 · · · · 2 1 0
Byte offset
52
10
Tag
Index
Hit
Data
Index
Valid
Tag
Data
0
1
2
...
...
...
1021
1022
1023
52
32
=

Q2 IN WORKSHEET

Address	Hit or Miss
10110	Miss
11010	Miss
10110	Hit
11010	Hit
10000	Miss
00011	Miss
10000	Hit
10010	Miss
10000	Hit

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

HANDLING CACHE MISS AND WRITES

- ***Cache miss***: A request for data from the cache that cannot be filled because the data are not present in the cache.
  - Stall
- ***Write-through***: A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data are always consistent between the two.
- ***Write-back***: A scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.
- ***Split cache***: A scheme in which a level of the memory hierarchy is composed of two independent caches that operate in parallel with each other, with one handling instructions and one handling data.

MEASURING AND IMPROVING CACHE PERFORMANCE

- Improving Performance.
  - One focuses on reducing the miss rate by reducing the probability that two distinct memory blocks will contend for the same cache location.
  - Second technique reduces the miss penalty by adding an additional level to the hierarchy [*multilevel caching]* .

- CPU time can be divided into the clock cycles that the CPU spends executing the program and the clock cycles that the CPU spends waiting for the memory system.

MEASURING AND IMPROVING CACHE PERFORMANCE

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

- The memory-stall clock cycles come primarily from cache misses.
- Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from writes:

- Memory-stall clock cycles=(Read-stall cycles + Write-stall cycles)

READS / WRITES

- Reads:
- The read-stall cycles can be defined in terms of the number of read accesses per program, the miss penalty in clock cycles for a read, and the read miss rate:
- Read-stall cycles = Reads-Program × Read miss rate × Read miss penalty

- Writes
- Write-stall cycles = (Writes Program × Write miss rate × Write miss penalty) + Write buffer stalls

MEASURING AND IMPROVING CACHE PERFORMANCE

- Memory-stall clock cycles= $\frac{Memory\ Access}{Program}$ × Miss Rate × Miss penalty

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

- **Q3 in worksheet**
- The miss rate of an instruction cache is 2% and the miss rate of the data cache 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. Assume the frequency of all loads and stores is 36%.
- ? Instruction miss cycle , data miss cycle , CPI time (including stall)

SOLUTION

- The number of memory miss cycles for instructions in terms of the Instruction count (I) is
- Instruction miss cycles= $I \times 2\% \times 100 = 2.00 \times I$
- The frequency of all loads and stores is 36%, we can find the number of memory miss cycles for data references:
- Data miss cycles=$I \times 36\% \times 4\% \times 100 = 1.44 \times I$

- The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I.
- CPI including memory stalls is 2 + 3.44 = 5.44.
- For a perfect Cache (without stall) = 2

- Performance = $CPI_{stall} / CPI_{perfect} = 5.44 / 2 = 2.72$

AMAT

- *Average memory access time* (AMAT) as a way to examine alternative cache designs. Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses.

- AMAT = Time for a hit + Miss rate × Miss penalty

- Q4 in worksheet ?

- The average memory access time per instruction is

  AMAT= Time for a hit + Miss rate × Miss penalty

  =1+0.05×20

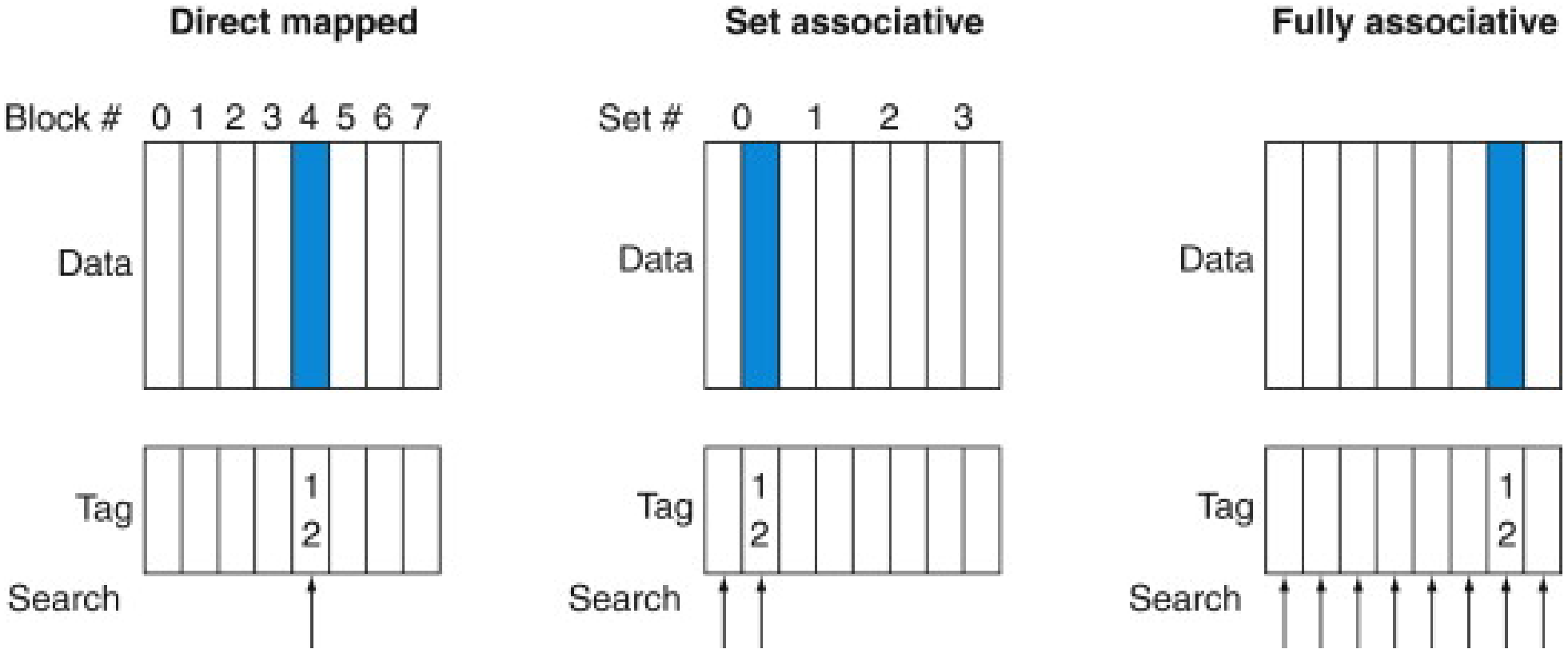
  =2 clock cycles or 2 ns

ASSOCIATIVE CACHES

- ***Fully associative cache***:
  - A cache structure in which a block can be placed in any location in the cache.

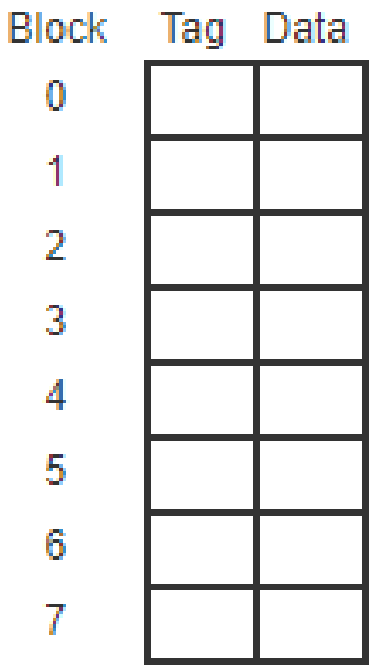
- ***Set-associative cache***:
  - A cache that has a fixed number of locations (at least two) where each block can be placed.

CACHE MAPPING



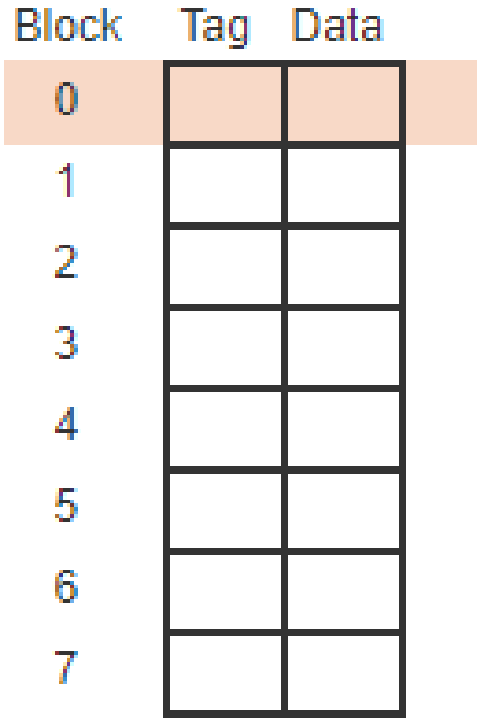
EXAMPLE

- An eight-block cache can be configured with different associativity types.

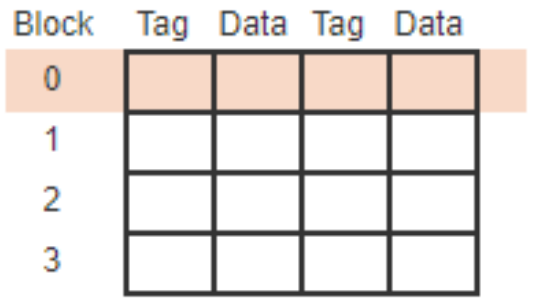


EXAMPLE

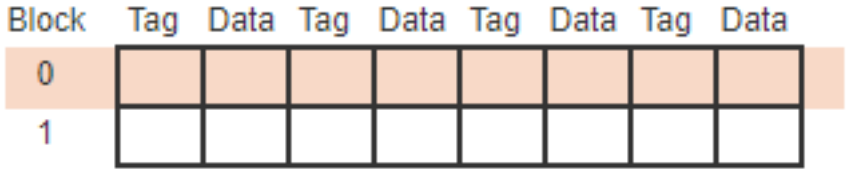
- Directed Mapped : Each cache entry holds one block and each set has one element.



- Increasing the associativity decreases the number of sets while increasing the number of elements per set. A two-way set-associative cache contains four sets; each entry set has two elements.



- A four-way set-associative cache contains two sets; each entry set has four elements.



Q 5

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

SOLUTION - WORKSHEET Q5

- For Directed Cache Map - 0, 8, 0, 6, and 8.

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			
		Block 0	Block 1	Block 2	Block 3

SOLUTION

- Directed Mapped Cache

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]	

SOLUTION

- Set Associativity -2 way - 0, 8, 0, 6, and 8.

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]		

Replacement Policy

Least recently used (***LRU***): A replacement scheme in which the block replaced is the one that has been unused for the longest time.

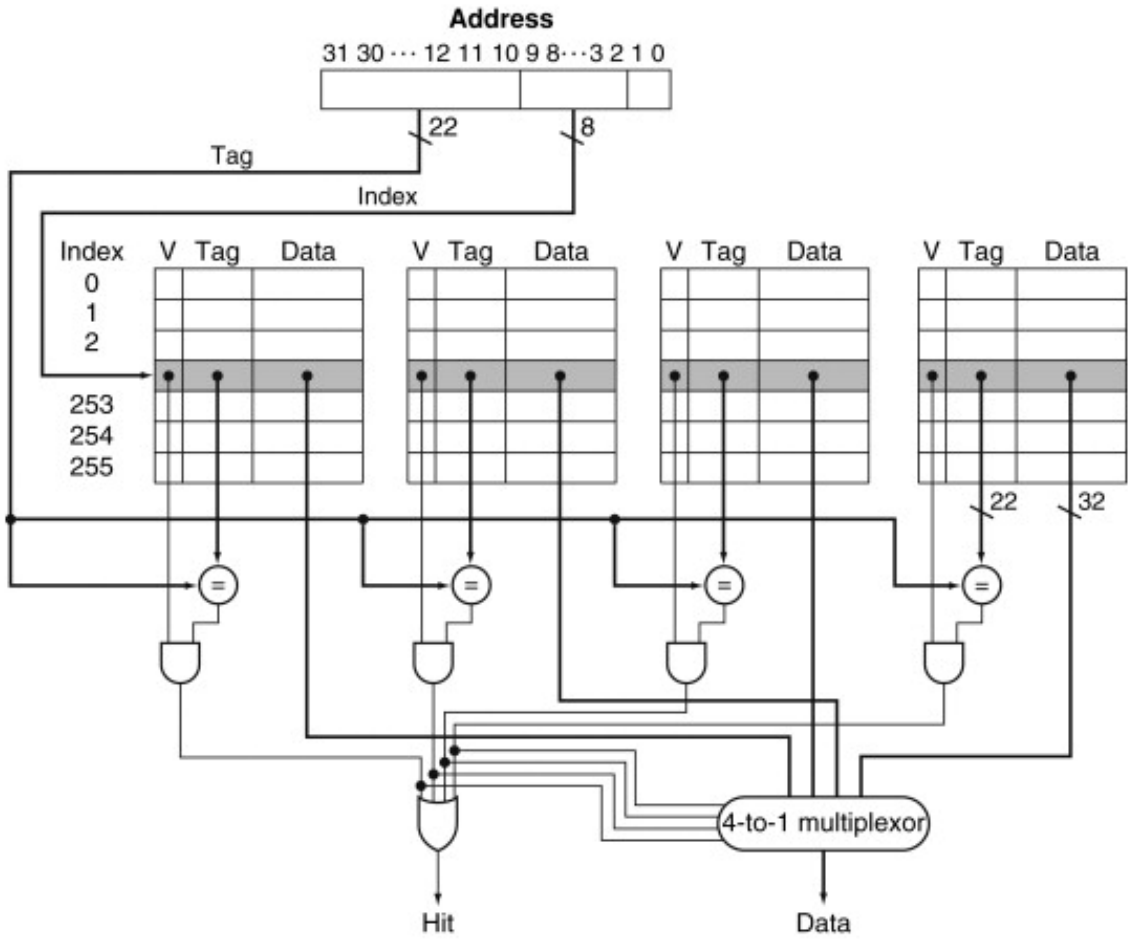
SOLUTION

- 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]	

CAM

- *A Content Addressable Memory (CAM) is a circuit that combines comparison and storage in a single device*



TAG, INDEX, OFFSET BIT , Q 1

- If we have a standard 32 bit address machine with 4GB RAM . A 4 way set associative CPU data Cache that uses 32 Byte blocks. How many bits will be used for Tag, Index and offset.
- Total size of cache is 16 KiB.

- 32 Byte Blocks –> 3 bits for offset
- 16 KiB = 16 X 1024 Bytes
- Number of Blocks = 16 X 1024 / 32 =512 Blocks
- 4 Way = 512 /4 = 128 Sets -> 7 Bits for index
- Rest is Tag -> 22 bits

Q 2

- Additional specs for the 16 Kbyte cache include: - Each block will hold 32 bytes of data .The cache would be 2-way set associative. Data is addressed to the word are 32 bits.

- How many blocks would be in this cache? **512**
- How many sets **256**
- What is Tag, Index and offset ? **21, 8, 3**

Q3

- Find the number of misses for each cache organization (Direct Map) given the following sequence of block addresses 8, 0, 5, 32, 0, 42, 9 and the address format is 8 bit ,if the cache has Capacity: 16B and Block size: 4B .

For 8 bit address

For 16 bit address

For 32 bit address

- Address 8 = (binary)00001000 [for 8 bit address format]

Q4 SOLUTION

- Memory addresses are 16 bits, The cache has 8 rows (i.e., 8 cache lines) , Each cache row (line) holds 16 bytes of data . **Tag = 10 Index=3 Offset=3**

Memory Address	Tag	Index	Offset Copied	Hit or Miss
0010 1101 1011 0011		110	0010110110110 [000,001,010,.....,111]	M
0000 0110 1111 1100		111	0010110110 111[000,001,.....,111]	M
0010 1101 1011 1000		111		M
1010 1010 1010 1011		101	1010101010 101 [000,.....,111]	M

Q4 MODIFIED QUESTION

- Memory addresses are 16 bits, The cache has 8 rows (i.e., 8 cache lines) , Each cache row (line) holds 16 bytes of data . **Tag = 10 Index=3 Offset=3**

Memory Address	Tag	Index	Offset Copied	Hit or Miss
0010 1101 1011 0011		110	0010110110110 [000,001,010,.....,111]	M
0000 0110 1111 1100		111	0010110110 111[000,001,.....,111]	M
0000 0110 1111 1000		111		H
1010 1010 1010 1011		101	1010101010 101 [000,.....,111]	M

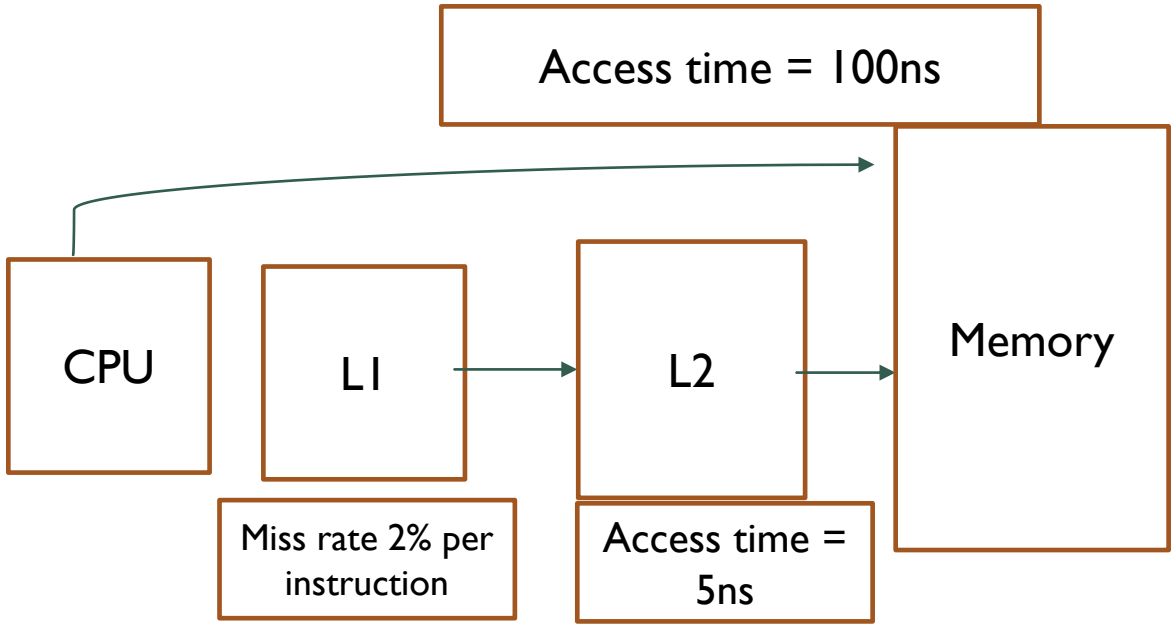
PERFORMANCE OF MULTILEVEL CACHE

- Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 4 GHz. Assume a main memory access time of 100 ns, including all the miss handling. Suppose the miss rate per instruction at the primary cache is 2%. How much faster will the processor be if we add a secondary cache that has a 5-ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%?
- Clock cycle per instruction (CPI) = 1
- Clock rate = 4 GHz

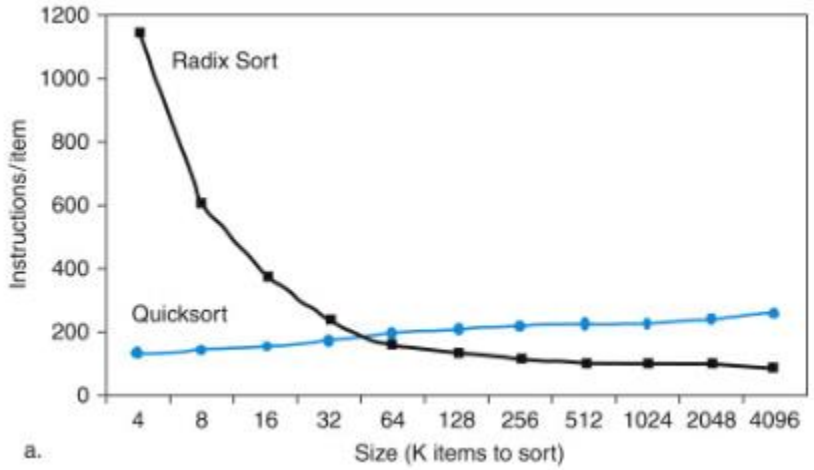
4 Ghz in 1 sec = pow(10,9) ns ; 1 period = pow(10,9) / 4GHz
1 period= pow(10,9) / 4 X pow(10,9) = 1/4 =0.25 ns

Miss penalty to Main memory = 100 / .25 =400 clock cycles
CPI = Base CPI + Memory Stall cycles / instruction
= 1 + 2% X 400 =1+8 =9
Miss penalty of L2 = 5 / .25 = 20 clock cycles
CPI with L2 = Base CPi+ Stalls in L1+Stalls in L2
= 1 + 2% X 20 + 0.5% X 400 = 1+ .4 + 2 = 3.4

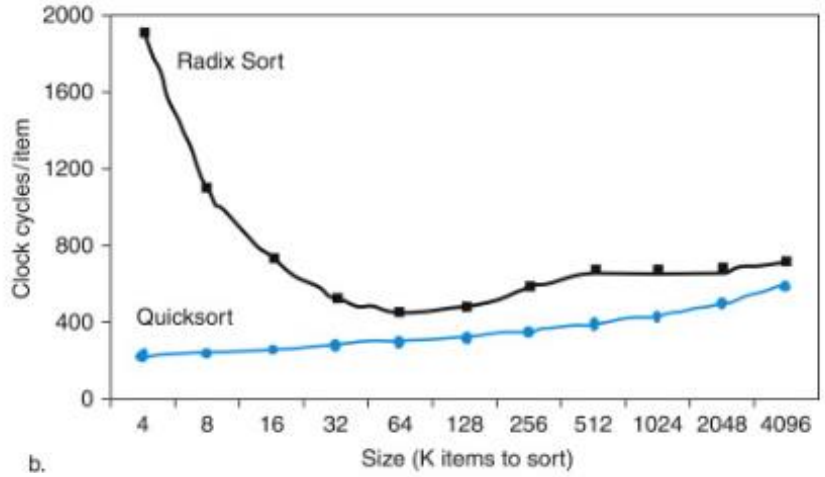
CPI =9 , with L2 CPI=3.4 , 9/3.4 times faster .



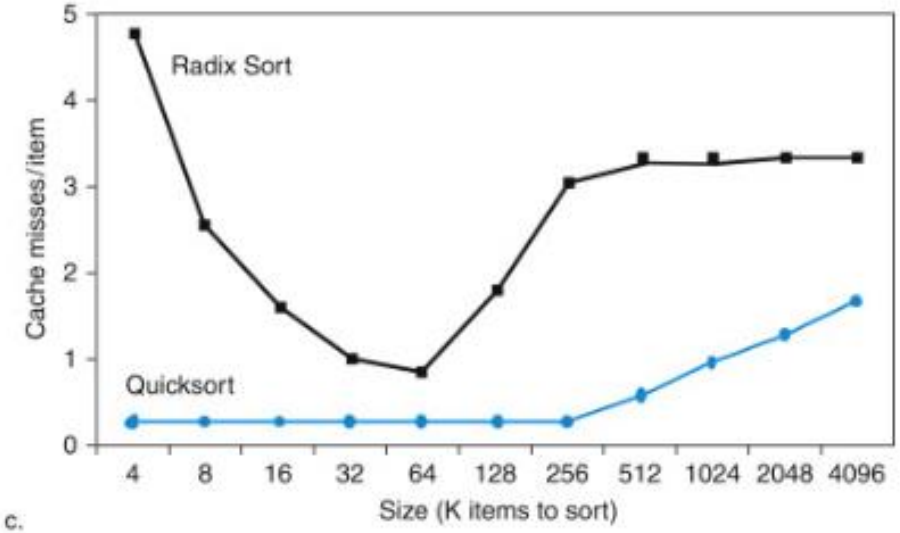
COMPARING QUICKSORT AND RADIX SORT



a.



b.



c.

ERROR CODES

- **Error detection codes** − are used to detect the error(s) present in the received data (bit stream). These codes contain some bit(s), which are included (appended) to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data (bit stream).
- **Example** − Parity code, Hamming code.

- **Error correction codes** − are used to correct the error(s) present in the received data (bit stream) so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. **Example** − Hamming code.

PARITY CODE

- Even Parity Code
  - *The value of even parity bit should be zero, if even number of ones present in the binary code.*
  - *Otherwise, it should be one.*

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

ODD PARITY

- Odd Parity Code
  - The value of odd parity bit should be zero, if odd number of ones present in the binary code.
  - Otherwise, it should be one.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

PARITY BIT

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

PARITY BIT

- Example: even parity
  - `1000000(1)`
  - `1111101(0)`
  - `1001001(1)`
- Example: odd parity
  - `1000000(0)`
  - `1111101(1)`
  - `1001001(0)`

PARITY BIT

- Assume we are using parity bit with 7-bit ASCII.
- The letter V in 7-bit ASCII is encoded as 0110101.
- How will the letter V be transmitted?

- For even parity ? **01101010**

- For odd parity ? **01101011**

- How will we do this using logic ?

FINDING EVEN PARITY

- XOR Truth Table

	0	0	1	1
^	0	1	0	1
	0	**1**	**1**	**0**

Even

- 1011 check number of 1s => 1 ^ 0 ^ 1 ^ 1 = 1
- 1100 1^1=0^0=0^0= 0
- 100100 1^0=1^0=1^1=0^0=0^0=0
- 111000 1

EXAMPLE –EVEN PARITY

- To transmit: **1001**
- Compute parity bit value:
  - 1+0+0+1 (mod 2) = **0** 0r 1^ 0 ^ 0^ 1 = **0**
- Adds parity bit and sends: **1001**0
- Receives: **10010**
- Computes parity: 1+0+0+1+0 (mod 2) = **0** 0r 1^ 0 ^ 0^ 1 = **0**
- Correct transmission after observing expected even result .

EXAMPLE –EVEN PARITY

- To transmit: **1001**
- Compute parity bit value:
  - 1+0+0+1 (mod 2) = **0** 0r 1^ 0 ^ 0^ 1 = **0**

- Adds parity bit and sends: **10010**
- **Transmission error**
- Receives: **11010**
- Computes parity: 1+1+0+1+0 (mod 2) = **1** 0r 1^ 1 ^ 0^ 1 = **1**
- Incorrect transmission after observing unexpected odd result.

PARITY BIT

- Limitations
  - Error in the parity bit ?
  - If even number of bits have errors ?

HAMMING CODE – ERROR CORRECTION

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		…
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

HAMMING (15,11)

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	1	1	0	1	0	1	1	0	0	0	1	1	0	0	1	
	P1	P2		P4				P8								
P1	?	X	0	X	0	X	1	X	0	X	1	X	0	X	1	1
P2		?	0	X	X	1	1	X	X	0	1	X	X	0	1	0
P4				?	0	1	1	X	X	X	X	1	0	0	1	0
P8								?	0	0	1	1	0	0	1	1

Parity Coverage

p1
p2 in 2, p4 in 4,p8 in 8
2+4+8 = 14

NEXT HW

- Write an ARM Assembly program to check a bit stream for even parity .