1.

Replacement Algorithm talks about what should be done when a new block is brought into the Cache, once the cache has been filled. For the associative and set associative we need an algorithm to determine which block should be replaced with the block we fetched from the main memory. This algorithm is needed to be implemented on the hardware side to achieve high speed. A number of algorithms are available for this purpose. Some of the examples of replacement algorithms are LRU, MFU, LFU, FIFO etc..

We don't need a block replacement algorithm for direct mapping, this is because a main memory block can map only to a particular line of the cache. there is only one possible line for any particular block, and no choice is possible.

2.

Write-through:
In a write-through scheme, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.
Advantages:
- Read miss never results in writes to main memory
- Easy to implement
- Main memory always has current copy of data ensuring consistency

Disadvantages:
- Write is slower
- Every write needs a memory access
- Uses more memory bandwidth

Write-back:
With write back, updates are made only in the cache. When an update occurs, a **dirty bit**, or **use bit**, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.
Advantages:
- Writes occur at the speed of the cache memory
- Multiple writes within a block only require one write to main memory
- Uses less memory bandwidth

Disadvantages:
- Harder to implement
- Main memory is not always consistent with cache
- Reads that result in replacement may cause writes of dirty blocks to main memory

3.

1) Associate a 3-bit counter with each of the 8 blocks in a set.
2) Initially, arbitrarily set the 8 values to 0, 1, 2, ….. 7 respectively.
3) When a hit occurs, the counter of the block that is referenced is set to 0. The other counters in the set with values originally lower than the referenced counter are incremented by 1. The remaining counters are unchanged.
4) When a miss occurs, the block in the set whose counter value is 7 is replaced and its counter set to 0. All other counters in the set are incremented by 1.

For example if we have the block counter values as [1,3,4,5,7,0,2,6] and if a cache hit is occurred at block 2 of the 8 blocks, then we have the values as [2,0,4,5,7,1,3,6]. Next if there is a Cache miss the current values are changed and the new block replaces the block whose counter value is 7, the values become [3,1,5,6,0,2,4,7].

4.

It is not necessary that the cache hit ratio will increase when the line size increases. As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced in the near future. As the block size increases, more useful data are brought into the cache. The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced. There could be two effects possible because of this. Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched. As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future. It is complex to understand the relation between block size and cache hit ratio.

Possible approaches to cache coherency include the following:
1.Bus watching with write through: Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. This strategy depends on the use of a write-through policy by all cache controllers.
2.Hardware transparency: Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches. Thus, if one processor modifies a word in its cache, this update is written to main memory. In addition, any matching words in other caches are similarly updated.
3.Noncacheable memory: Only a portion of main memory is shared by more than one processor, and this is designated as noncacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache. The noncacheable memory can be identified using chip-select logic or high-address bits.

There are two advantages of unified cache over split cache.
1.For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. If an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions. If an execution pattern involves relatively more data fetches, the opposite will occur.
2. Only one cache needs to be designed and implemented.

5.

Spatial Locality:

Spatial Locality means that the instructions which are stored close to the recently executed instruction have a higher chance of execution. It refers to the tendency of the processor to access instructions sequentially. It also reflects the tendency of a program to access data locations sequentially, such as when processing a table of data.

Temporal Locality:

Temporal Locality means that the instruction which is executed recently have a higher chance of executing again. It refers to the tendency for a processor to access memory locations that have been used recently.

A. For the given code example of spatial locality will be reference of a[i] in the loop for(i=0;i< 40;i++) As we can assume that the locations a[0] and a[1] will be next to each other in the memory. As data locations are accessed in sequential order this is spatial locality.

B.

For the given code example of temporal locality is a[i] and j during the execution of the loop for(j=0;j<10;j++). Here both locations a[i] and j are accessed again and again. As memory locations are

6.

A.

8 GB = 2^{36} bits
Given C_m = $\$10^{-7}$ /bit
Cost of 8GB of main memory = $2^{36} * 10^{-7}$
= **$ 6871.94**

B.

8 GB = 2^{36} bits
Given C_c = $\$10^{-6}$ /bit
Cost of 8GB of main memory = $2^{36} * 10^{-6}$
= **$ 68719.476**

C.

The effective system access time is given by the formula
$T_s = H * Tc + (1 - H) * (Tc + Tm)$

Given T_s *is 15% greater than* Tc *, that is* $Ts = 1.15\ Tc$

$H = \frac{Tc + Tm - Ts}{Tm}$

$H = \frac{Tm - 0.15Tc}{Tm}$

We get ***H = 0.985***
Hence the Cache Hit ratio is H = 98.5%

7.

$T_s = H_c * T_c + (1 - H_c) * (T_{mavg})$
$T_{mavg} = H_m * T_{mc} + (1 - H_m) * (T_{dc})$

Given H_c = 0.94 , H_m = 0.5 , T_{mc} = 50 +15 = 65ns , T_{dc} = 10000000 + 70 +15 = 10000085 ns

T_{mavg} = 0.5 * 65 + 0.5 * 10000085 = 5000075 ns

T_s = 0.94* 15 + (0.06) * 5000075 = **300018.6 ns**

Hence the average time required to access a reference word in the system = 300018.6 ns