Associative Memory

Tanmay Bhowmik

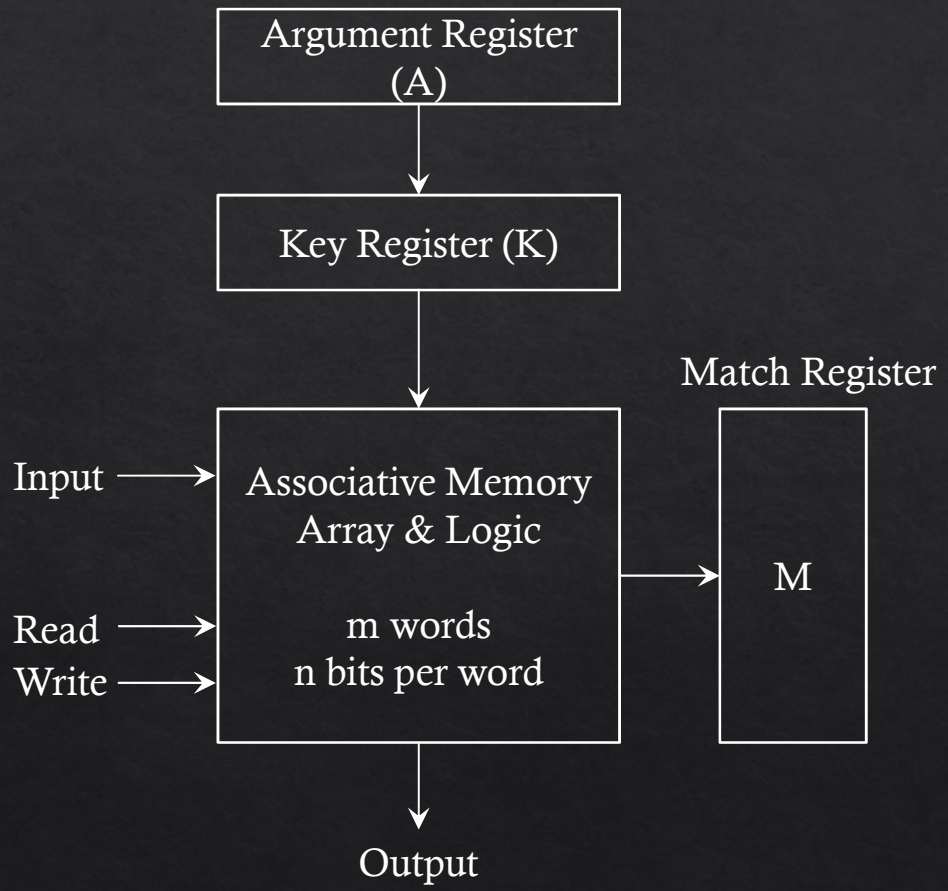
Introduction

- To search a particular data in memory, data is read from certain address and compared.
- If the match is not found content of the next address is accessed and compared.
- This goes on until required data is found. The number of access depend on the location of data and efficiency of searching algorithm.
- This searching time can be reduced if data is searched on the basis of content.

Introduction

- A memory unit accessed by content is called Associative Memory or Content Addressable Memory (CAM) or associative storage or associative array.
- This type of memory is accessed simultaneously and in parallel on the basis of data content.
- Memory is capable of finding empty unused location to store the word.

Associative Memory Organization



Block Diagram of Associative Memory Organization

- This block diagram consists of a memory array and logic for m words and n bits per word.
- The argument register A and key register K each have n bits, one for each bit of a word.
- The match register M has m bits, one for each memory word.
- Each word in memory is compared in parallel with the content of the argument register.
- The words that match the bits of the argument register set a corresponding bit in the match register M.
- After the matching process, those bit in the match register that have been set, indicate the corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register is set.

Associative Memory Organization

Associative Memory is organized in such a way:

- Argument register (A): It contains the word to be searched. It has n bits (one for each bit of the word).
- Key Register (K): This specifies which part of the argument word needs to be compared with words in memory. If all bits in register are 1, The entire word should be compared. Otherwise, only the bits in the argument register that have 1 in the corresponding position of the key register will be compared.
- Associative memory array: It contains the words which are to be compared with the argument word.
- Match Register (M): It has m bits, one bit corresponding to each word in the memory array. After the matching process, the bits corresponding to matching words in match register are set to 1.

Associative Memory Organization

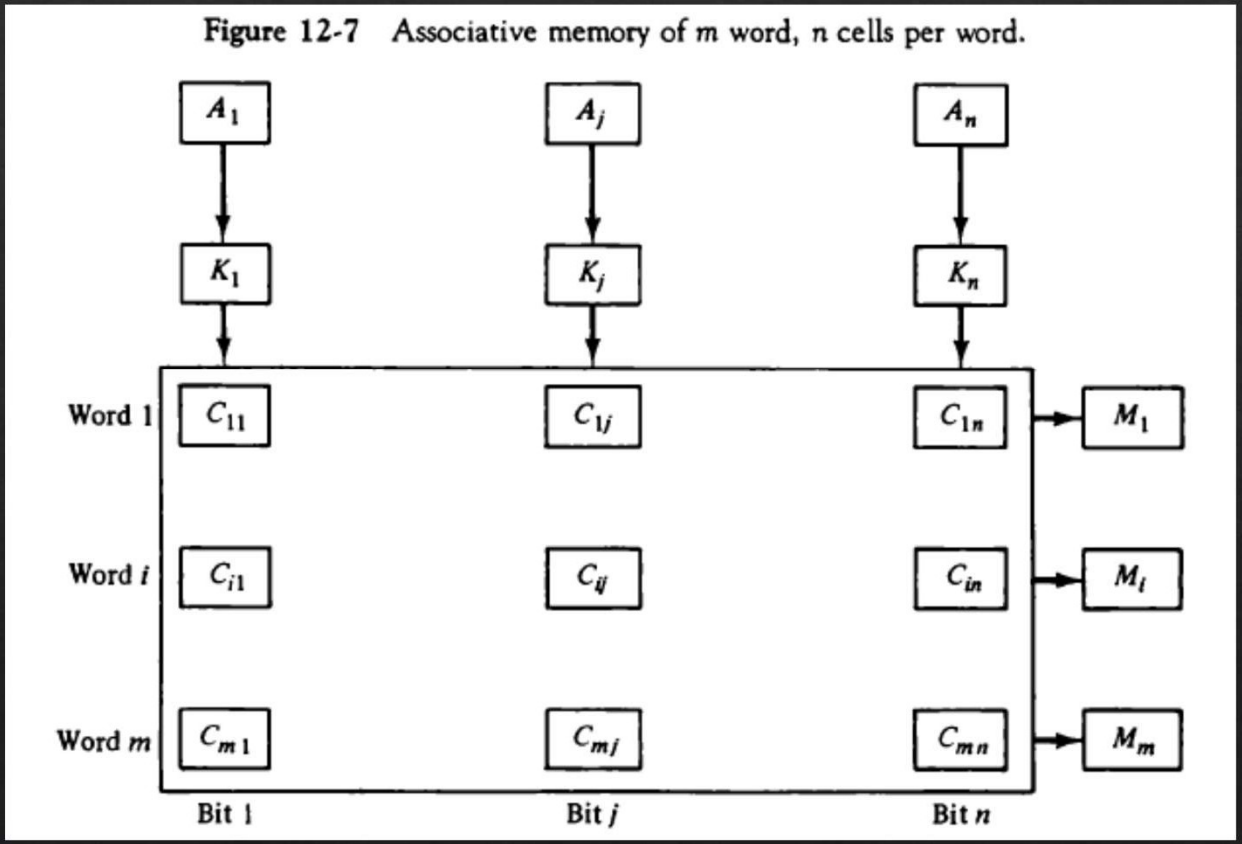
- Key register provide the mask for choosing the particular field in A register.
- The entire content of A register is compared if key register content all 1.
- Otherwise only bit that have 1 in key register are compared.
- If the compared data is matched corresponding bits in the match register are set.

Associative Memory Organization

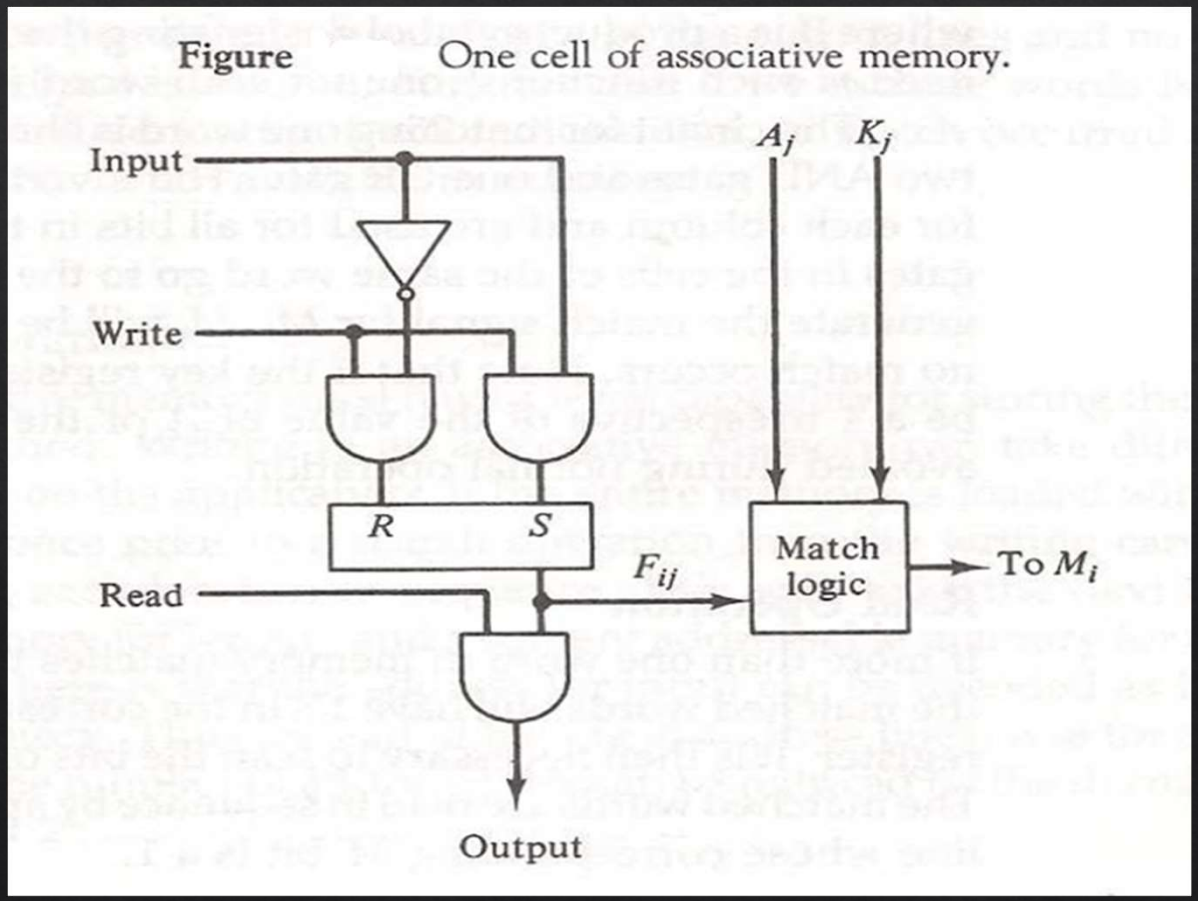
- Reading is accomplished by sequential access in memory for those words whose bit are set.

A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Associative Memory Organization



Associative Memory Organization



Associative Memory Organization

Match Logic

- Word ***i*** is equal to the argument in ***A*** if $A_j = F_{ij}$ for $j = 1, 2, \ldots, n$.
- Two bits are equal if they are both 1 or both 0.
- The equality of two bits can be expressed logically by the Boolean function

$$x_j = AjFij + A'_jF'_{ij}$$

where $x_j = 1$ if the pair of bits in position j are equal; otherwise, $x_j = 0$.

- For a word i to be equal to the argument in A we must have all x_j variables equal to 1.
- This is the condition for setting the corresponding match bit M_j to 1.
- The Boolean function for this condition is $M_i = x_1x_2x_3 \ldots x_n$

Associative Memory Organization

- Let us now include key register.
- If $K_j = 0$ then there is no need to compare A_j *and* F_{ij}.
- Only when $K_j = 1$, comparison is required.
- This requirement is achieved by OR-ing each term with K'_j, thus:

$$x_j + K_j' = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

- When $K_j = 1$, we have $K'_j = 0$ and $x_j + 0 = xj$.
- When $K_j = 0$, we have $K'_j = 1$ and $x_j + 1 = 1$.
- A term $(x_j + K'_j)$ will be in the 1 state if its pair of bits is not compared.
- So an output of 1 will have no effect.
- The comparison of bits has an effect only when $K_j = 1$.

Associative Memory Organization

- So the match logic for word i in an associative memory can now be expressed by the following Boolean function:

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

- Each term in this expression is equal to 1 if its corresponding $K_j = 0$.
- If $K_j = 1$ then each term is either 0 or 1 depending on the value of x_j.
- A match will occur and M_i will be equal to 1 if all terms are equal to 1.
- If we substitute the original definition of x_j, the above Boolean function can be expressed as

$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A_j' F_{ij}' + K_j')$$

Associative Memory Organization

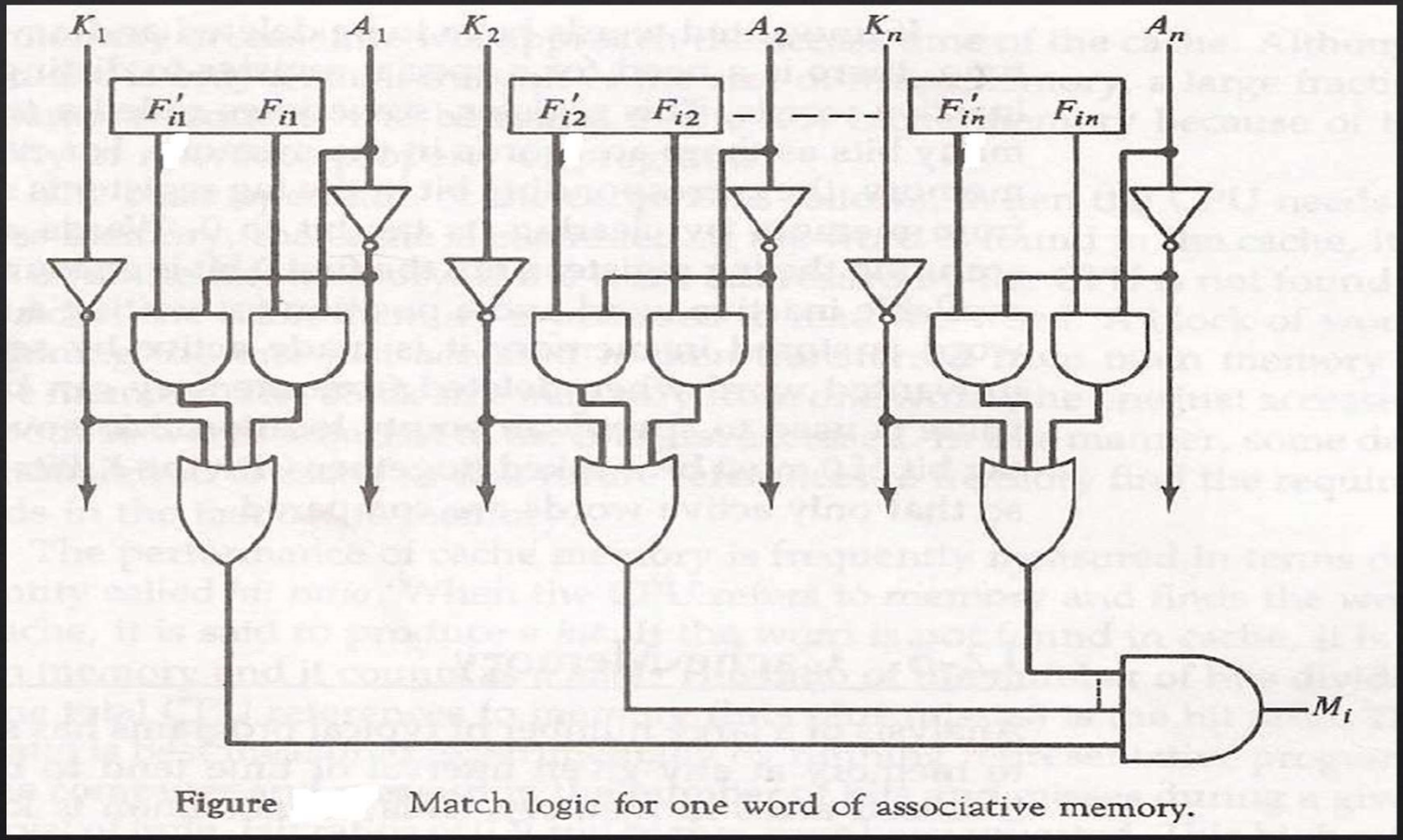


Figure Match logic for one word of associative memory.

Associative Memory Organization

- **Write operation**:
- If the entire memory is loaded with new information at once prior to search operation then writing can be done by addressing each location in sequence.
- Tag register contain as many bits as there are words in memory.
- It contain 1 for active word and 0 for inactive word.
- If the word is to be inserted, tag register is scanned until 0 is found and word is written at that position and bit is change to 1.

Associative Memory Organization

- **Read Operation:**
- When a word is to be read from an associative memory, the contents of the word, or a part of the word is specified.
- If more than one word match with the content, all the matched words will have 1 in the corresponding bit position in match register.
- Matched words are then read in sequence by applying a read signal to each word line.
- In most application, the associative memory stores a table with no two identical items under a given key.

Associative Memory Architecture

- It is a hardware search engines, a special **type of computer memory** used in certain very high searching applications.
- composed of conventional semiconductor memory (usually **SRAM**) with added comparison circuitry that enable a search operation to complete in a single clock cycle.
- **SRAM** is a type of semiconductor memory that uses bistable latching circuitry to store each bit.

Types of Associative memory

There are two types of Associative memory, which both are used in different conditions.

- **Auto-associative**

  Auto-associative memory takes back(retrieves) a previously stored pattern that most closely resembles the current pattern.

- **Hetero-associative**

  Hetero-associative memory, the retrieved pattern is in general, different from the input pattern not only in content but possibly also in type and format.

  Neural networks are used to implement these associative memory models called NAM (Neural associative memory).

Advantage of Associative memory

- This is suitable for parallel searches. It is also used where search time needs to be shorten.
- Associative memory is often used to speed up databases, in neural networks and in the page tables used by the virtual memory of modern computers.
- CAM-design challenge is to reduce power consumption associated with the large amount of parallel active circuitry, without sacrificing speed or memory density.

Disadvantage of Associative memory

- An associative memory is more expensive than a random access memory because each cell must have an extra storage capability as well as logic circuits for matching its content with an external argument.

- Usually associative memories are used in applications where the search time is very critical and must be very short.

LOCALITY

PRINCIPAL OF LOCALITY is the tendency to reference data items that are near other recently referenced data items, or that were recently referenced themselves.

TEMPORAL LOCALITY : memory location that is referenced once is likely to be **referenced multiple times** in near future.

SPATIAL LOCALITY : memory location that is referenced once, then the program is likely to be **referenced at a nearby memory location** in near future.

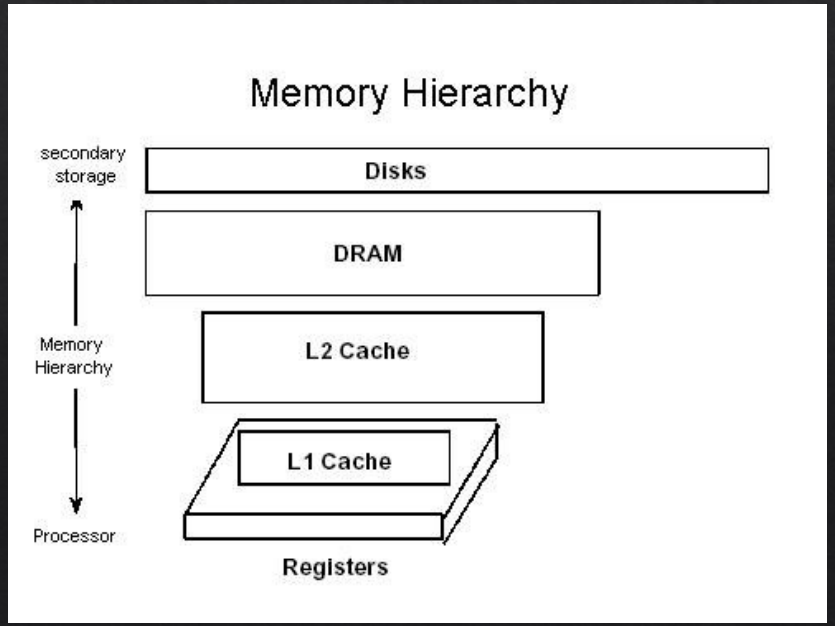
CACHE MEMORY

- Principle of locality helped to speed up main memory access by introducing small fast memories known as CACHE MEMORIES that hold blocks of the most recently referenced instructions and data items.

- Cache is a small fast storage device that holds the operands and instructions most likely to be used by the CPU.

Memory Hierarchy of early computers: 3 levels

- CPU registers
- DRAM Memory
- Disk storage



Due to increasing gap between CPU and main Memory, small SRAM memory called L1 cache inserted.

L1 caches can be accessed almost as fast as the registers, typically in 1 or 2 clock cycle.

Due to even more increasing gap between CPU and main memory, Additional cache: L2 cache inserted between L1 cache and main memory : accessed in fewer clock cycles.

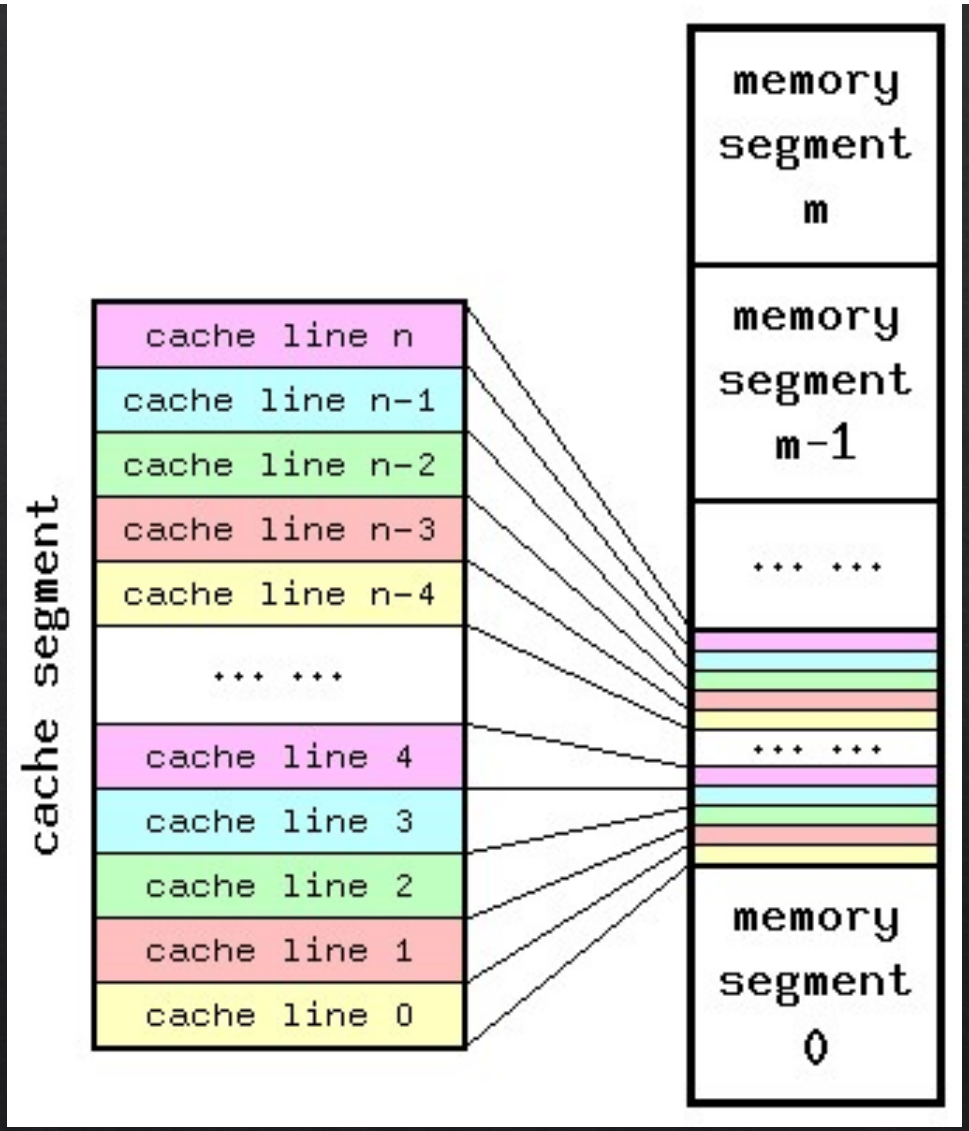
- L2 cache attached to the memory bus or to its own cache bus.
- Some high performance systems also include additional L3 cache which sits between L2 and main memory. It has different arrangement but principle same.
- The cache is placed both physically closer and logically closer to the CPU than the main memory.

CACHE LINES / BLOCKS

- Cache memory is subdivided into cache lines
- Cache Lines / Blocks: The smallest unit of memory that can be transferred between the main memory and the cache.

- Example:

  Memory segments and cache segments are exactly of the same size. Every memory segment contains equally sized N memory lines. Memory lines and cache lines are exactly of the same size. Therefore, to obtain an address of a memory line, it needs to determine the number of its memory segment first and the number of the memory line inside of that segment second, then to merge both numbers. Substitute the segment number with the tag and the line number with the index, and you should have realized the idea in general.


cache segment
cache line n
cache line n-1
cache line n-2
cache line n-3
cache line n-4
... ...
cache line 4
cache line 3
cache line 2
cache line 1
cache line 0
memory segment m
memory segment m-1
... ...
... ...
memory segment 0

Therefore, cache line's tag size depends on 3 factors:

- Size of cache memory;
- Associativity of cache memory;
- Cacheable range of operating memory.

$$S_{tag} = \log_2\left(\frac{S_{memory} \times A}{S_{cache}}\right)$$

Here,

S_{tag} — size of cache tag, in bits;
S_{memory} — cacheable range of operating memory, in bytes;
S_{cache} — size of cache memory, in bytes;
A — associativity of cache memory, in ways.

TAG / INDEX

- Every address field consists of two primary parts: a dynamic (tag) which contains the higher address bits, and a static (index) which contains the lower address bits.

- The first one may be modified during run-time while the second one is fixed.

VALID BIT / DIRTY BIT

- When a program is first loaded into main memory, the cache is cleared, and so while a program is executing, a valid bit is needed to indicate whether or not the slot holds a line that belongs to the program being executed.

- There is also a dirty bit that keeps track of whether or not a line has been modified while it is in the cache. A slot that is modified must be written back to the main memory before the slot is reused for another line.

CACHE HITS / MISSES

- Cache Hit: a request to read from memory, which can satisfy from the cache without using the main memory.

- Cache Miss: A request to read from memory, which cannot be satisfied from the cache, for which the main memory has to be consulted.

CACHE MEMORY : MAPPINGS

There are three commonly used methods to translate main memory addresses to cache memory addresses.

- Associative Mapped Cache
- Direct-Mapped Cache
- Set-Associative Mapped Cache

The choice of cache mapping scheme affects cost and performance, and there is no single best method that is appropriate for all situations

CACHE MEMORY : MAPPINGS

- In the discussion of these three mapping techniques we will use a specific example of a memory organization as shown in following figure.
- The main memory can store 32K words ($32K = 2^5 \times 2^{10} = 2^{15}$, so address bit = 15) of 12 bits each. The cache is capable of storing 512 of these words at any given time.
- For every word in cache there is a duplicate copy in main memory.
- The CPU first sends 15 bits address to the cache. If there is a hit, CPU accepts 12 bit data from the cache.
- If there is a miss, the CPU reads word from main memory and the word is then transferred to cache memory.

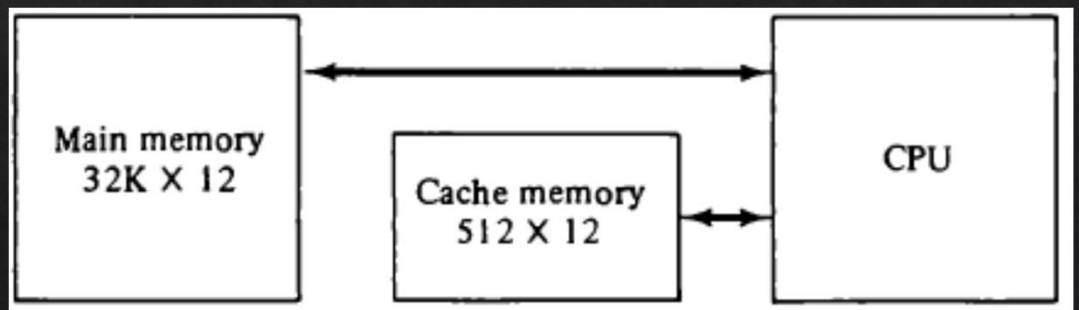
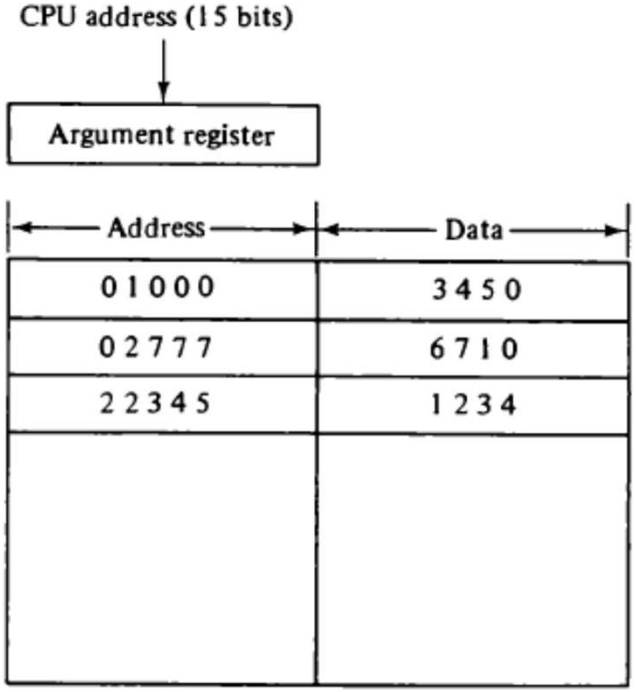


Figure 12-10 Example of cache memory.

Associative Mapping

- A block in the Main Memory can be mapped to any block in the Cache Memory available (not already occupied).
- Advantage: Flexibility. An Main Memory block can be mapped anywhere in Cache Memory.
- Disadvantage: Slow or expensive. A search through all the Cache Memory blocks is needed to check whether the address can be matched to any of the tags.

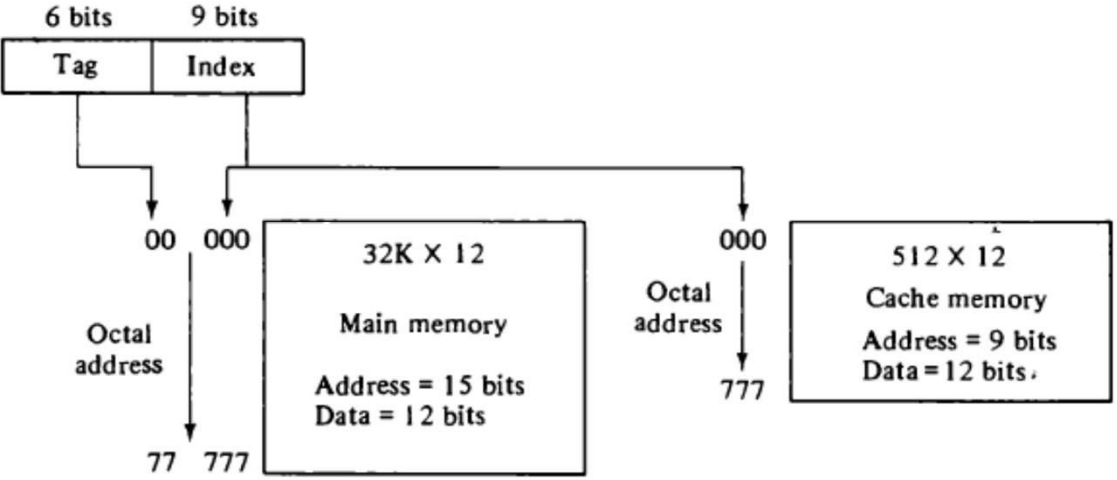
Figure 12-11 Associative mapping cache (all numbers in octal).



Direct Mapping

- To avoid the search through all CM blocks needed by associative mapping, this method only allows to map each block of main memory into only one possible cache line.
- In general there are 2^n words in main memory and 2^k words in cache memory.
- The n bit address is divided into two parts, k bits for index and n – k bits for the tag.

Figure 12-12 Addressing relationships between main and cache memories.



- Advantage: Direct mapping is faster than the associative mapping as it avoids searching through all the CM tags for a match.

- Disadvantage: But it lacks mapping flexibility. For example, if two MM blocks mapped to same CM block are needed repeatedly (e.g., in a loop), they will keep replacing each other, even though all other CM blocks may be available.

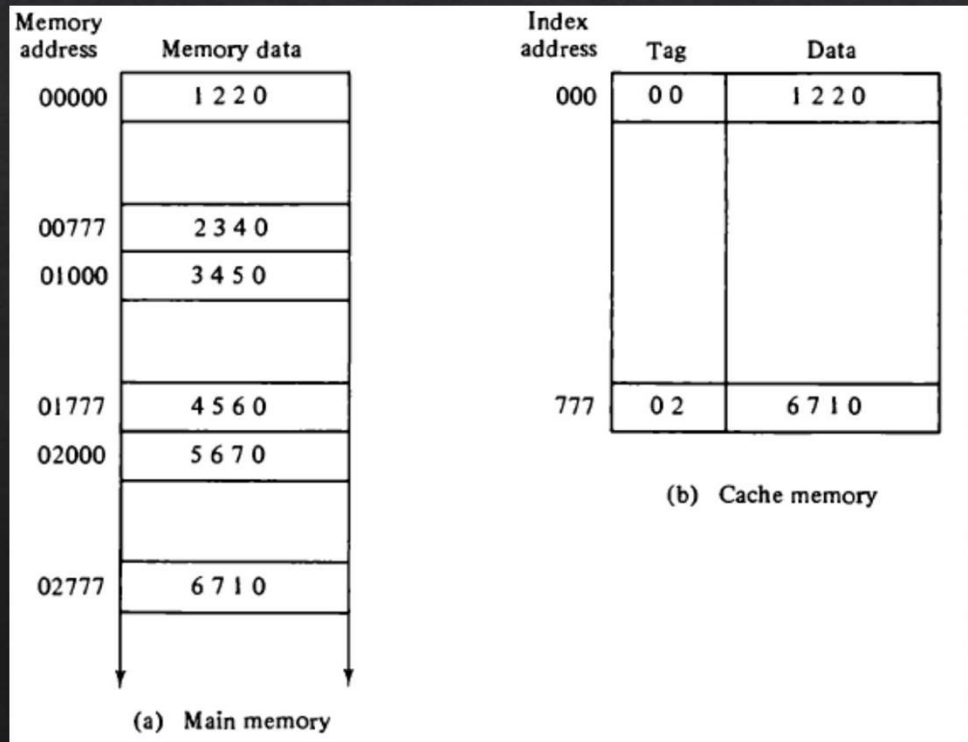


Figure 12-13 Direct mapping cache organization.

Set-Associative Mapping

- This is a trade-off between associative and direct mappings where each address is mapped to a certain set of cache locations.

- The cache is broken into sets where each set contains "N" cache lines, let's say 4. Then, each memory address is assigned a set, and can be cached in any one of those 4 locations within the set that it is assigned to. In other words, within each set the cache is associative, and thus the name.

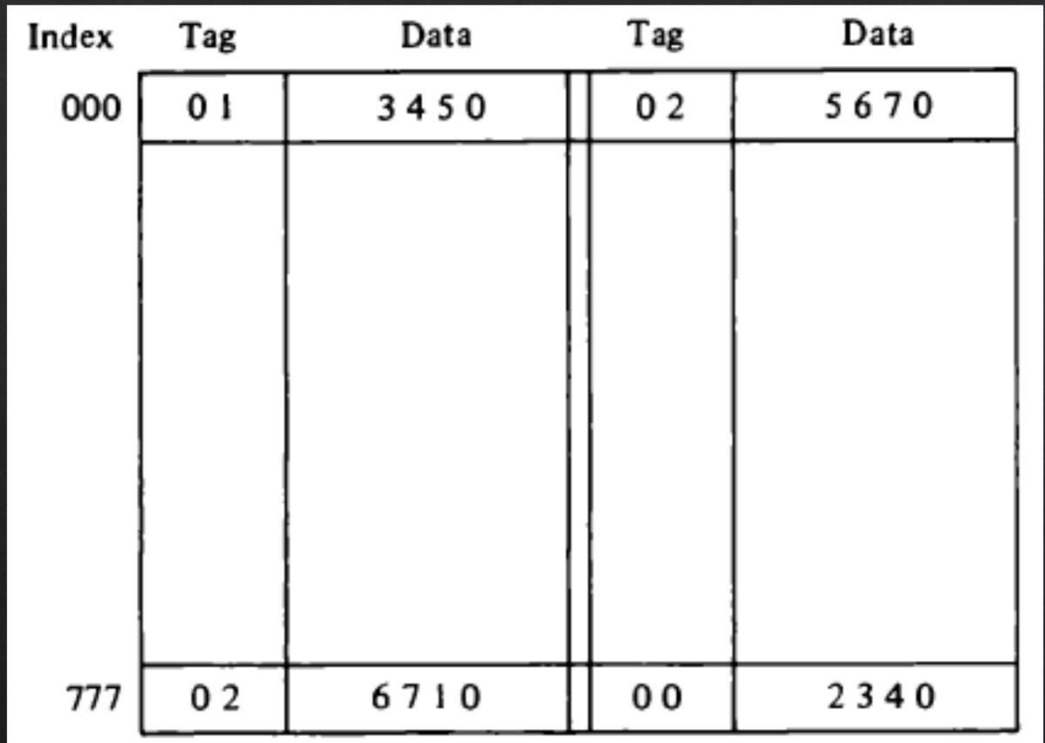


Figure 12-15 Two-way set-associative mapping cache.

DIFFERENCE BETWEEN LINES, SETS AND BLOCKS

- In direct-mapped caches, sets and lines are equivalent. However in associative mapping caches, sets and lines are very different things and the terms cannot be interchanged.

- BLOCK: fixed sized packet of information that moves back and forth between a cache and main memory.
- LINE: container in a cache that stores a block as well as other information such as the valid bit and tag bits.
- SET: collection of one or more lines. Sets in direct-mapped caches consist of a single line. Set in fully associative and set associative caches consists of multiple lines.

ASSOCIATIVITY

- Associativity : N-way set associative cache memory means that information stored at some address in operating memory could be placed (cached) in N locations (lines) of this cache memory.

- The basic principle of logical segmentation says that there is only one line within any particular segment to be capable of caching information located at some memory address.

REPLACEMENT ALGORITHM

- Optimal Replacement: replace the block which is no longer needed in the future. If all blocks currently in Cache Memory will be used again, replace the one which will not be used in the future for the longest time.

- Random selection: replace a randomly selected block among all blocks currently in Cache Memory.

- FIFO (first-in first-out): replace the block that has been in Cache Memory for the longest time.
- LRU (Least recently used): replace the block in Cache Memory that has not been used for the longest time.
- LFU (Least frequently used): replace the block in Cache Memory that has been used for the least number of times.

- The optimal replacement is the best but is not realistic because when a block will be needed in the future is usually not known ahead of time.
- The LRU is suboptimal based on the temporal locality of reference, i.e., memory items that are recently referenced are more likely to be referenced soon than those which have not been referenced for a longer time.
- FIFO is not necessarily consistent with LRU therefore is usually not as good.
- The random selection, surprisingly, is not necessarily bad.

HIT RATIO and EFFECTIVE ACCESS TIMES

- Hit Ratio : The fraction of all memory reads which are satisfied from the cache

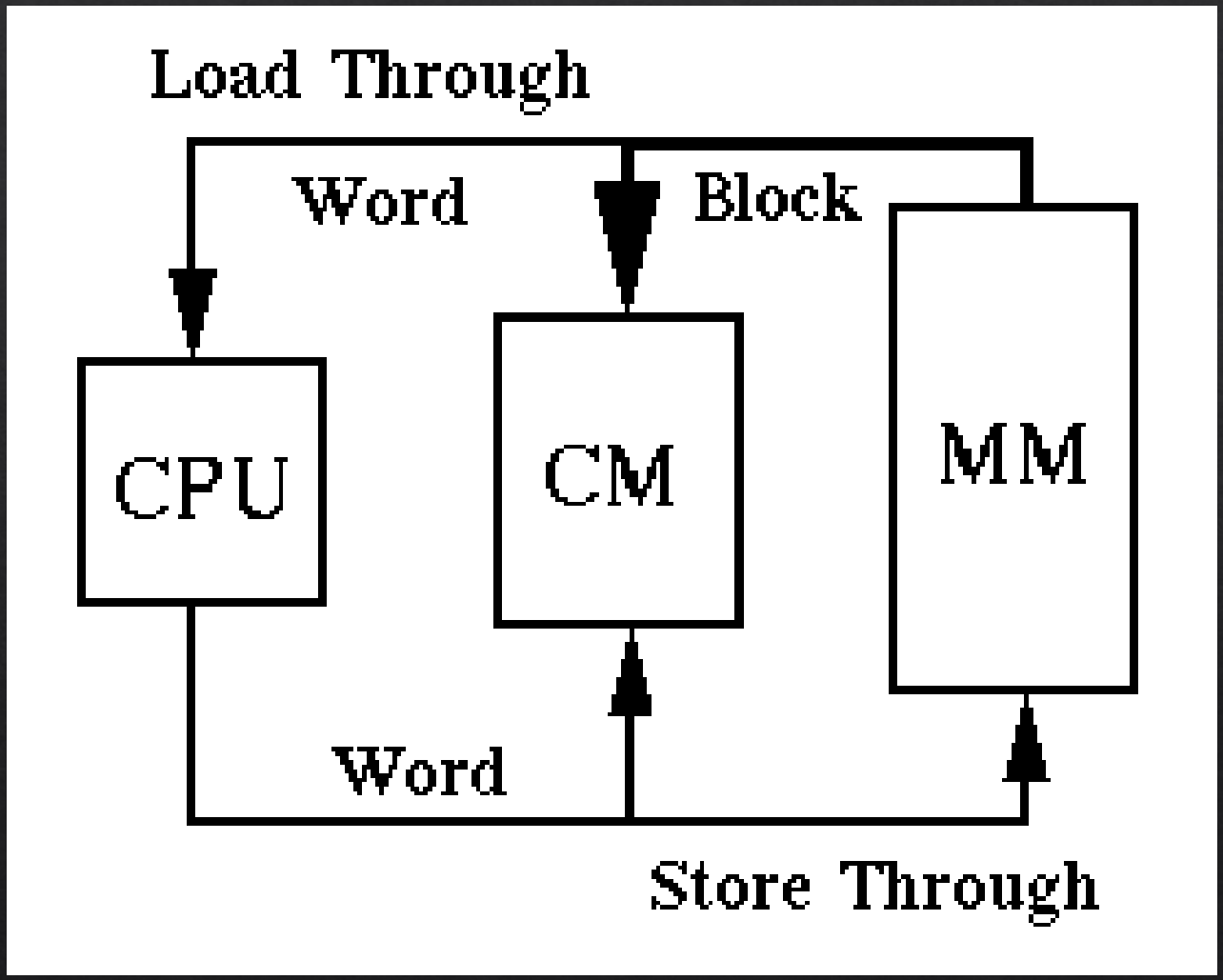
$$Hit\ ratio = \frac{No.\ times\ referenced\ words\ are\ in\ cache}{Total\ number\ of\ memory\ accesses}$$

$$Effective\ access\ time = \frac{(\#hits)\ (time\ per\ hit) + (\#\ misses)\ (time\ per\ miss)}{Total\ number\ of\ memory\ access}$$

LOAD-THROUGH STORE-THROUGH

- Load-Through : When the CPU needs to read a word from the memory, the block containing the word is brought from MM to CM, while at the same time the word is forwarded to the CPU.

- Store-Through : If store-through is used, a word to be stored from CPU to memory is written to both CM (if the word is in there) and MM. By doing so, a CM block to be replaced can be overwritten by an in-coming block without being saved to MM.


Load Through
Word
Block
CPU
CM
MM
Word
Store Through

WRITE METHODS

- Note: Words in a cache have been viewed simply as copies of words from main memory that are read from the cache to provide faster access. However this view point changes.

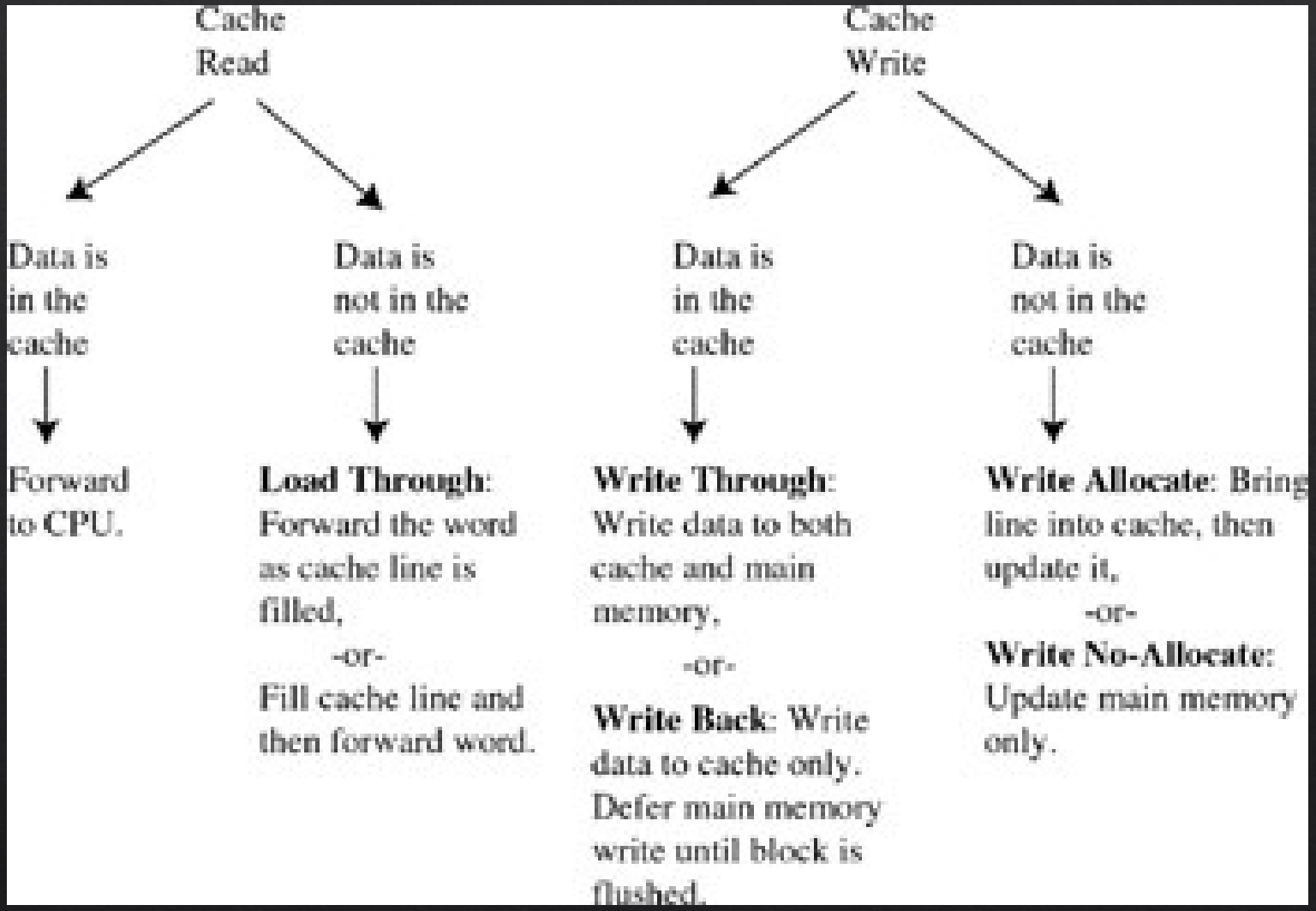
- There are 3 possible write actions:
  - Write the result into the main memory
  - Write the result into the cache
  - Write the result into both main memory and cache memory

- Write Through: A cache architecture in which data is written to main memory at the same time as it is cached.

- Write Back / Copy Back: CPU performs write only to the cache in case of a cache hit. If there is a cache miss, CPU performs a write to main memory.

When the cache is missed :

- Write Allocate: loads the memory block into cache and updates the cache block
- No-Write allocation: this bypasses the cache and writes the word directly into the memory.


Cache Read
Cache Write
Data is in the cache
Data is not in the cache
Data is in the cache
Data is not in the cache
Forward to CPU.
Load Through: Forward the word as cache line is filled,
-or-
Fill cache line and then forward word.
Write Through: Write data to both cache and main memory,
-or-
Write Back: Write data to cache only. Defer main memory write until block is flushed.
Write Allocate: Bring line into cache, then update it,
-or-
Write No-Allocate: Update main memory only.

CACHE CONFLICT

- A sequence of accesses to memory repeatedly overwriting the same cache entry.
- This can happen if two blocks of data, which are mapped to the same set of cache locations, are needed simultaneously.

- EXAMPLE: In the case of a direct mapped cache, if arrays A, B, and C map to the same range of cache locations, thrashing will occur when the following loop is executed:

```
for (i=1; i<n; i++)
    C[i] = A[i] + B[i];
```

Cache conflict can also occur between a program loop and the data it is accessing.

CACHE COHERENCY

- The synchronization of data in multiple caches such that reading a memory location via any cache will return the most recent data written to that location via any (other) cache.

- Some parallel processors do not cache accesses to shared memory to avoid the issue of cache coherency.

- If caches are used with shared memory then some system is required to detect when data in one processor's cache should be discarded or replaced because another processor has updated that memory location. Several such schemes have been devised.