Part 3

Cach memory

• Cache memory is a fast small memory where the active portion of the program and data are placed in, so the average memory access time is reduced.

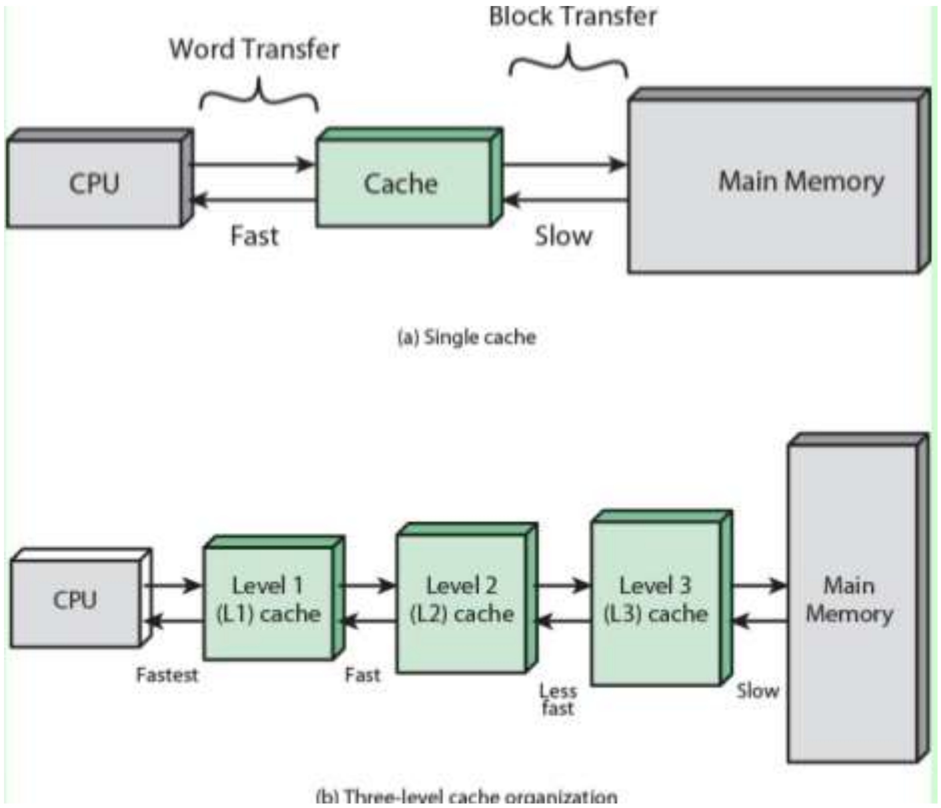
• Thus reducing the total execution time of the program.

•Cache memory is in between CPU and main memory.

• CPU access data from cache.

• L-1 cache fabricated on CPU chip(on chip).

•L-2 cache btween main memory and CPU(off chip).



- When CPU needs to access a memory, cache is examined
- If memory location found then OK! Read (Hit)
- If not, then memory location is searched in main memory and block that contained the required location is transffered to the cahe and read by CPU (Miss).

Principle of Locality:

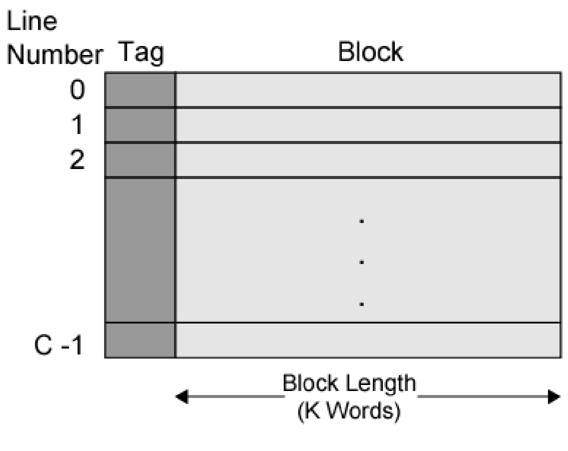
Programs tends to reuse data and instructions near those they heve used recently. There are two types of locality:

- Temporal locality: Recently referenced items are likely to be referenced in the near future.
- Spatial locality: a neighbor of a recently referenced memory location is likely to be refrenced.

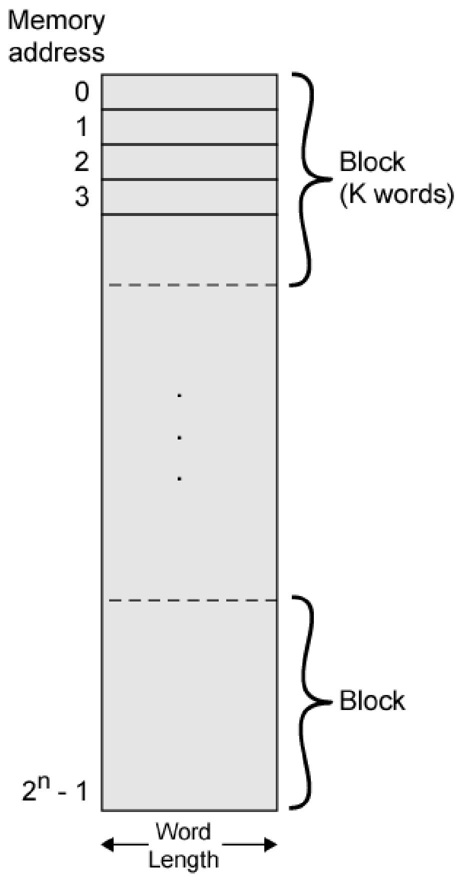
Cache performance:

- The performance of cache memory is measured by Hit Ratio.
- Hit Ratio=(total hit)/(total hit +total miss).
- Hit Ratio of 0.9 and higher have been reported.

Cache/Main Memory Structure



(a) Cache



(b) Main memory

Basic elements of cache design:

- Size
- Mapping function
- Replacement algorithm
- Write policy
- Block size
- Number of caches

Cache-Mapping Function :

• The transformation of data from main memory to cache memory is referred to as memory **mapping** process.
• This is one of the functions performed by the memory management unit (**MMU**).
• Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
• There are three different types of mapping functions in common use and are **direct**, **associative** and **set associative**

The three techniques are discussed below:

1- Direct Mapping :This is the simplest among the three techniques. Its simplicity stems from the fact that it places an incoming main memory block into a specific fixed cache block location. The placement is done based on a fixed relation between the incoming block number, i, the cache block number, j, and the number of cache blocks, N: $j = i \bmod N$

The main **advantage** of the direct-mapping technique is its simplicity in determining where to place an incoming main memory block in the cache.

Its main **disadvantage** is the inefficient use of the cache. This is because a number of main memory blocks may compete for a given cache block even if there exist other empty cache blocks. This disadvantage should lead to achieving a low cache hit ratio.

According to the direct-mapping technique the MMU interprets the address issued by the processor by dividing the address into three fields:

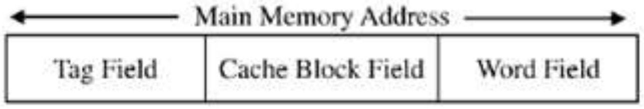
1. ***Word*** field = $\log_2 B$, where B is the size of the block in words.
2. ***Block*** field = $\log_2 N$, where N is the size of the cache in blocks.
3. ***Tag*** field = $\log_2 (M/N)$, where M is the size of the main memory in blocks.
4. The number of **bits in the main memory address** = $\log_2 (B \times M)$

Example 1: Consider, for example, the case of a main memory consisting of **4K** blocks, a cache memory consisting of **128** blocks, and a block size of **16** words.

Example 2 Compute the above four parameters for Example 1.

Word field $= \log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits
Block field $= \log_2 N = \log_2 128 = \log_2 2^7 = 7$ bits
Tag field $= \log_2(M/N) = \log_2(2^2 \times 2^{10}/2^7) = 5$ bits

The number of bits in the main memory address $= \log_2(B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ bits.



2-Fully Associative Mapping: According to this technique, an incoming main memory block can be placed in any available cache block. Therefore, the address issued by the processor need only have two fields. These are the ***Tag*** and ***Word*** fields. The first uniquely identifies the block while residing in the cache. The second field identifies the element within the block that is requested by the processor.

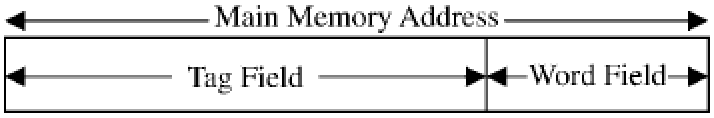


Figure 6.7 Associative-mapped address fields

1. **Word** field = $\log_2$ B, where B is the size of the block in words
2. **Tag** field = $\log_2$ M, where M is the size of the main memory in blocks
3. The **number of bits in the main memory address** = $\log_2$ (B x M)

the tags are stored in an **associative memory** (content addressable). This allows the entire contents of the tag memory to be searched in parallel (associatively), hence the name, associative mapping.

The main **advantage** of the associative-mapping technique is the efficient use of the cache. This stems from the fact that there exists no restriction on where to place incoming main memory blocks. Any unoccupied cache block can potentially be used to receive those incoming main memory blocks.

The main **disadvantage** of the technique, is the hardware overhead required to perform the associative search conducted in order to find a match between the tag field and the tag memory as discussed above.

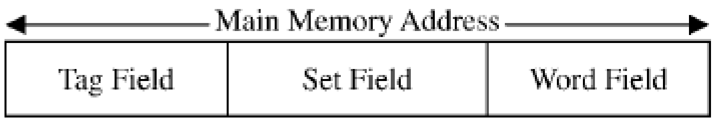
Example 3: Compute the above three parameters for a memory system having the following specification: size of the main memory is **4K** blocks, size of the cache is **128** blocks, and the block size is **16** words. Assume that the system uses associative mapping:

Word field = $\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits

Tag field = $\log_2 M = \log_2 2^7 \times 2^{10} = 12$ bits

The number of bits in the main memory address = $\log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ bits.

3- Set-Associative Mapping: In the set-associative mapping technique, the cache is divided into a number of sets. Each set consists of a number of blocks. A given main memory block maps to a specific cache set based on the equation $s = i \bmod S$, where S is the number of sets in the cache, i is the main memory block number, and s is the specific cache set to which block i maps.
However, an incoming block maps to any block in the assigned cache set. Therefore, the address issued by the processor is divided into three distinct fields. These are the ***Tag*, *Set*,** and ***Word*** fields.



Set-associative-mapped address fields

The length, in bits, of each of the fields of is given by:

1. **Word** field = $\log_2 B$, where B is the size of the block in words

2. **Set** field = $\log_2 S$, where S is the number of sets in the cache

3. **Tag** field = $\log_2 (M/S)$, where M is the size of the main memory in blocks.

S = N/Bs, where N is the number of cache blocks and Bs is the number of blocks per set

4. The **number of bits in the main memory address** = $\log_2 (B \times M)$

Example 4: Compute the above three parameters (Word, Set, and Tag) for a memory system having the following specification: size of the main memory is **4K** blocks, size of the cache is **128** blocks, and the block size is **16** words.
Assume that the system uses set-associative mapping with **four** blocks per set.
S = 128/4 = 32 sets:
1. Word field = $\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits

2. Set field = $\log_2 32 = 5$ bits

3. Tag field = $\log_2 (4 \times 2^{10}/32) = 7$ bits
The number of bits in the main memory address = $\log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ bits.

<u>Replacement Techniques</u>

When all lines are occupied, bringing in a new block requires that an existing line be overwritten.

For Direct mapping :

- No choice possible with direct mapping
- Each block only maps to one line
- Replace that line

For Associative and set- associative:
A number of replacement techniques can be used. These include a:

1-Random selection: Let us assume that when a computer system is <u>powered up</u>:
• A random number generator starts generating numbers between 0 and (N- 1).
• <u>A cache block for replacement</u> is done based <u>on the output of the random number generator</u> at the time of replacement.
• This technique is simple and does not require much additional overhead.

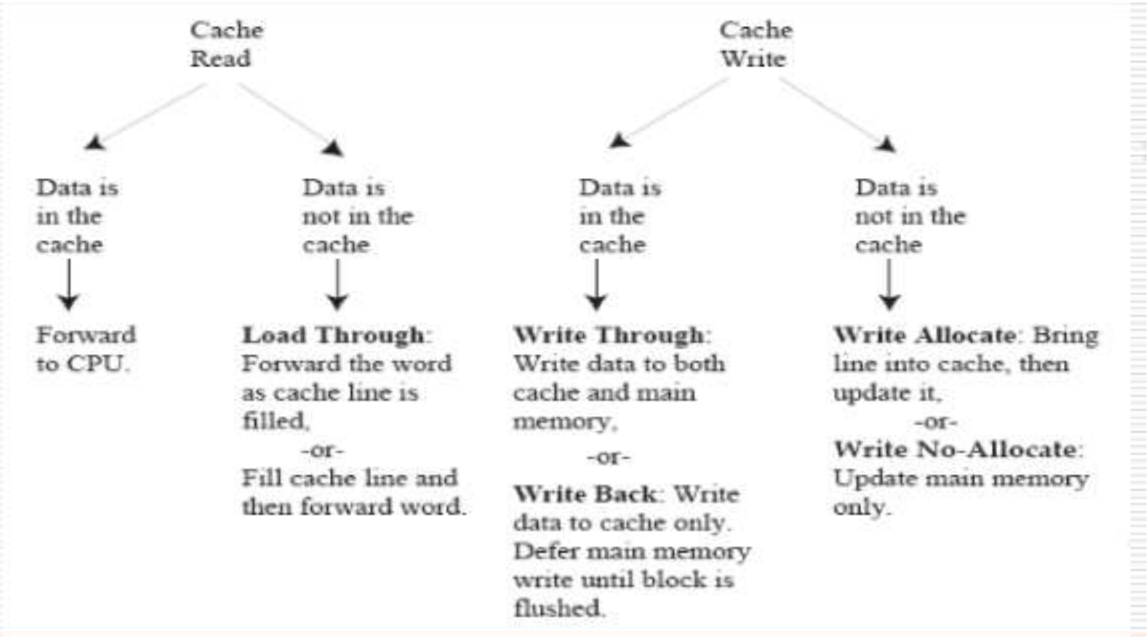
2-FIFO:
• Takes the time spent by a block in the cache as a measure for replacement.
• The block that has been in the cache <u>the longest</u> is selected for replacement <u>regardless of the recent pattern of access to the block.</u>
• This technique requires keeping track of the lifetime of a cache block.
• Therefore, it is not as simple as the random selection technique.

3- Least Recently used (LRU):
• Replace that block in the set which has been in cache longest with no reference to it
• Implementation: having a <u>USE</u> bit for each line . When a block is read into cache, use the line whose USE bit is set to 0, then set its USE bit to one and
•The other line's USE bit to 0.
Probably the most effective method

4- Least-frequently-used (LFU):
• Replace that block in the set which has experienced the fewest references or hits
• Implementation: associate a counter with each slot and increment when used

Cache write policies

- If cpu has to write a word in memory location


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