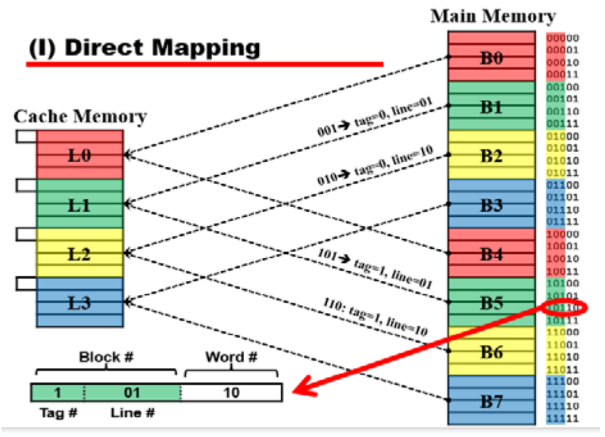
Cache Memory:

$T_{av} = T_c + (1 - H)\, T_m$

1. Mapping:

(I) Direct Mapping



- Each MM block maps to only one cache line.
- Block #**b** maps to line #(**b modulo m**).
- If a block is in cache, it must be in one specific place.

Address Format:

- Memory address is split (based on block size) into:
  1. Least significant **w** bits ➔ identify **word** in block.
     - ➢ w = $\log_2$ (# of words in block).
  2. Most significant **s** bits ➔ identify **block** in MM.
     - ➢ s = $\log_2$ (# of blocks in MM).
     - ➢ The **s** bits are split (based on cache size) into:
       1. Least significant **r** bits ➔ identify cache **line**.
          - ➢ r = $\log_2$ (# of lines in cache).
       2. Most significant **s – r** bits ➔ **tag** (identify group).

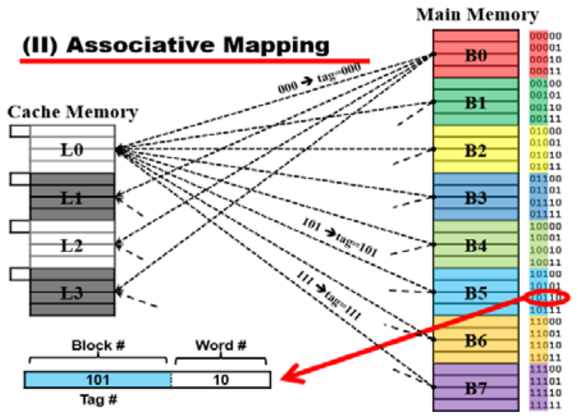
Address Format Summary:

- Address length = (s + w) bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^w words.
  - ➢ w = $\log_2$ (# of words in block).
- Number of blocks in MM = M = $2^{s+w}/2^w = 2^s$.
  - ➢ s = $\log_2$ (# of words in MM / # of words in block)
- Number of lines in cache = m = 2^r.
  - ➢ r = $\log_2$ (# of words in cache / # of words in line).
- Size of tag = (s – r) bits.
  - ➢ (s-r) = $\log_2$ (# of words in MM / # of words in cache)

Block # (s bits)		Word # (w bits)
Tag # (s-r bits)	Line # (r bits)	

(II) Fully-Associative Mapping:

- A main memory block can load into any line of cache.
- Memory address is interpreted as tag and word.
- Tag uniquely identifies block of memory.
- Every line's tag is examined for a match.
- Cache searching gets expensive.



Address Format:

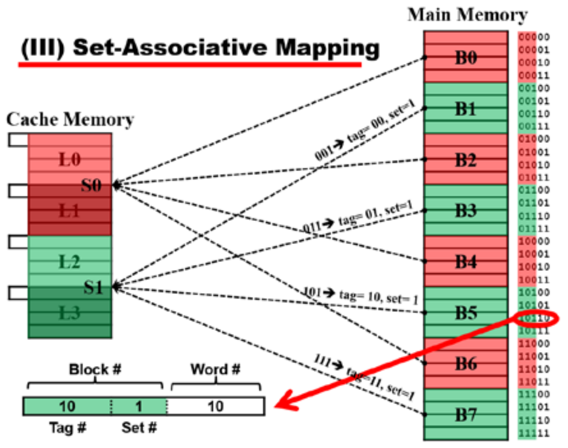
- Memory address is split (based on block size) into:
  1. Least significant **w** bits ➔ identify **word** in block.
     - ➢ w = $\log_2$ (# of words in block).
  2. Most significant **s** bits ➔ identify **block** in MM, and used as a **tag**.
     - ➢ s = $\log_2$ (# of blocks in MM).
- In the "tiny example": w=2, s=3.

Address Format Summary:

- Address length = (s + w) bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^w words.
  - ➢ w = $\log_2$ (# of words in block).
- Number of blocks in MM = M = $2^{s+w}/2^w = 2^s$.
  - ➢ s = $\log_2$ (# of words in MM / # of words in block)
- Number of lines in cache = m ➔ unknown!
- Size of tag = s.

Tag # (s bits)	Word # (w bits)

(III) Set-Associative Mapping:



- Cache is divided into a number of **sets** (v) of equal size.
- Each set contains a number of lines (k).
    - ➢ **k-way** set-associative mapping!
- A block b could map to **any line** in a set i if and only if i = b modulo v.

Address Format:

- Memory address is split (based on block size) into:
    1. Least significant **w** bits ➔ identify **word** in block.
        - ➢ w = $\log_2$ (# of words in block).
    2. Most significant **s** bits ➔ identify **block** in MM.
        - ➢ s = $\log_2$ (# of blocks in MM).
        - ➢ The **s** bits are split (based on cache size) into:
            1. Least significant **d** bits ➔ identify cache **set**.
                - ➢ d= $\log_2$ (# of sets in cache).
            2. Most significant **s – d** bits ➔ **tag** (identify group).
- In the "tiny example": w=2, s=3, d=1, s-d=2.

Address Format Summary:

- Address length = (s + w) bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^w words.
    - ➢ w = $\log_2$ (# of words in block).
- Number of blocks in MM = M = $2^{s+w}/2^w = 2^s$.
    - ➢ s = $\log_2$ (# of words in MM / # of words in block)
- Number of lines in set = k ➔ **k-way set-associative**
- Number of sets = v = 2^d.
- Number of lines in cache = m = k * v = k * 2^d.
    - ➢ d = $\log_2$ (# of words in cache / # of words in set).
- Size of tag = (s – d) bits.
    - ➢ (s-d) = $\log_2$ (# of blocks in MM / # of sets in cache).

Block # (**s** bits)		Word # (**w** bits)
Tag # (**s**-**d** bits)	Set # (**d** bits)	

2. Replacement Algorithms:

Direct mapping:

- No choice.
- Each block only maps to one line.
- Replace that line.

Associative and Set Associative:

- **Hardware implemented algorithms!**

- **Most common ones:**

    1. **Least Recently Used (LRU)**
        - Replace the block that has not been recently accessed.

    2. **First In First Out (FIFO)**
        - Replace the oldest block.

    3. **Least Frequently Used**
        - Replace the block which has had the fewest hits.

    4. **Random**
        - Replace any block!

3. Read/Write Policies:

Read Policy (Hit & Miss):

- **Reads dominate processor cache accesses.**
  - — All instructions need to be fetched ➔ reads.
  - — Some instructions may read one or more operands.
  - — Most instructions do not write to memory!

- **What if CPU wants to read a word from MM?**
  - — **Read-hit policy:** if read hits in cache (i.e., container block is in cache), just read word from cache!
  - — **Read-miss policy:** if read misses in cache (i.e., container block is in not cache),
    1. Read-through: read word from MM, and bring container block to cache.
    2. No-read-through: bring container block to cache, and then read word from cache.

Write Policy (Hit):

- What if CPU wants to write a word to memory?
  - **Write-hit policy:** if write hits (i.e., container block is in cache)
    1. **Write-through:** write word to both cache and MM
       + **Pros**: easier to implement, MM is always consistent with cache, read miss never results in writes to MM.
       + **Cons**: Every write needs a MM access, slower writes, more MM bandwidth consumption.

    2. **Write-back:** write word to cache only, do not write word to MM until cached version of container block is replaced.
       + In addition to tag bits, each cache line stores a "dirty bit".
       + **Pros**: multiple writes within block require only one write to MM, faster writes, less MM bandwidth consumption.
       + **Cons**: harder to implement, MM is not always consistent with cache, reads may cause writes of dirty blocks to MM.

Write Policy (Miss):

  - **Write-miss policy:** if write misses (i.e., container block is in not cache)
    1. **Write-allocate:** bring container block to cache, and then act according to write-hit policy.

    2. **No-write-allocate:** write word to container block in MM, and do not bring block to cache.

- **Note**: either write-miss policy could be used with write-through or write-back, however:
  - Write-back caches generally use write-allocate
    - Hoping that subsequent writes to block get captured by cache
  - Write-through caches often use no-write-allocate
    - since subsequent writes block will still have to go to MM