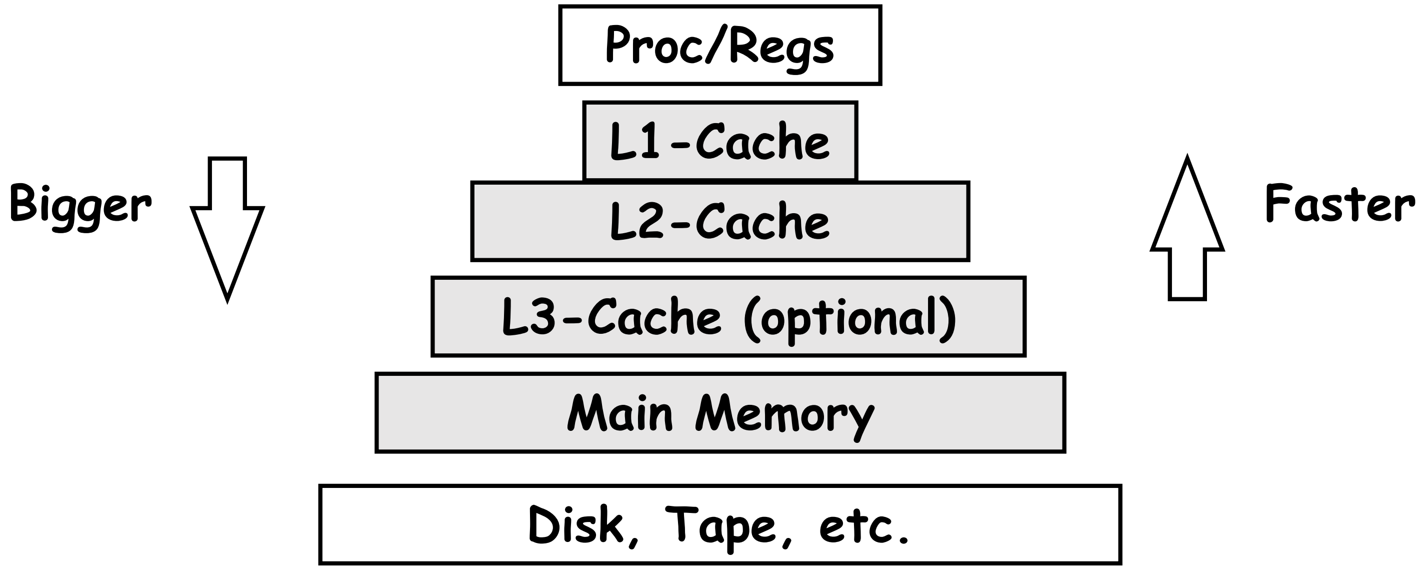
Memory Hierarchy
Design and Optimizations

Typical Memory Hierarchy



- Here we focus on L1/L2/L3 caches, main memory and virtual memory

Level of Memory

- Level-1 or Register: It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Example: Accumulator, Program counter, address register etc.
- Level-2 or Cache memory: It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- Level-3 or Main memory: In this memory, computer currently works. It is small in size. Once the power is off, data is no longer stays in this memory.
- Level-4 or Secondary Memory: It is an external memory which is not fast as main memory but data stays permanently in this memory.

What is the Role of a Cache?

- What is cache?
  - ➢A small, fast storage used to improve average access time to a slow memory.
- Why we need it?
  - ➢To improves memory system performance:
- Types of Cache:
  - ➢Primary Cache (Level-1): A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
  - ➢Secondary Cache (Level-2): Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2 cache). It is also located in the processor chip.

- Locality of reference (is also known as principle of locality) (Analysis of program)
- Since size of cache memory is less as compared to main memory. So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.
- Types of Locality of references

  - Temporal, - Spatial

  - Temporal locality tells us that we are likely to need this word again in the near future, so it is useful to place it in the cache where it can be accessed quickly. (Example: Loop)
  - Spatial locality, there is high probability that the other data in the block will be needed soon. (Example: accessing the data elements of an 1D-array)
- Cache block – *cache line (Locality of reference)*
  - *A set of contiguous address locations of some size*

Four Basic Questions

- Q1: Where can a block be placed in the cache? *(Block placement) (ADDRESS MAPPING)*
    - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the cache? *(Block identification)*
    - Tag/Block/word
- Q3: Which block should be replaced on a miss? *(Block replacement)*
    - Random, LRU, etc.
- Q4: What happens on a write? *(Write strategy)*
    - Write Back or Write Through (with Write Buffer)

Address Mapping/ Block Placement

- Converts physical address to cache address.
- 3 types of mapping.
    - Direct Mapping
    - Associative mapping
    - Set-associative mapping

Direct Mapping

- It is a simplest technique.
- It maps each block of main memory into only one possible cache line.
- The direct mapping selects (K mod N)th cache line for accommodating Kth main memory block. (where N is the no. of cache lines in cache memory).
- It divides the physical address into 3 fields.

TAG	CACHE BLOCK OFFSET	WORD OFFSET

- Word Offset= $log_2 B$
- Cache block offset = $log_2 N$
- Tag bits= $log_2 (\frac{M}{N})$
- (where M is the no. of blocks in Main memory, B is the block size).

Example-1:

- Main memory size =512 words, Cache memory size=64 words, 1 block size=16 words. Obtain TAG bits, cache block offset and word offset.
- Solution:
- No. of blocks in main memory =512/16=32 blocks
- No. of blocks/lines in cache memory=64/16= 4 blocks
- Word offset ?
- Cache block offset ?
- Tag bits?

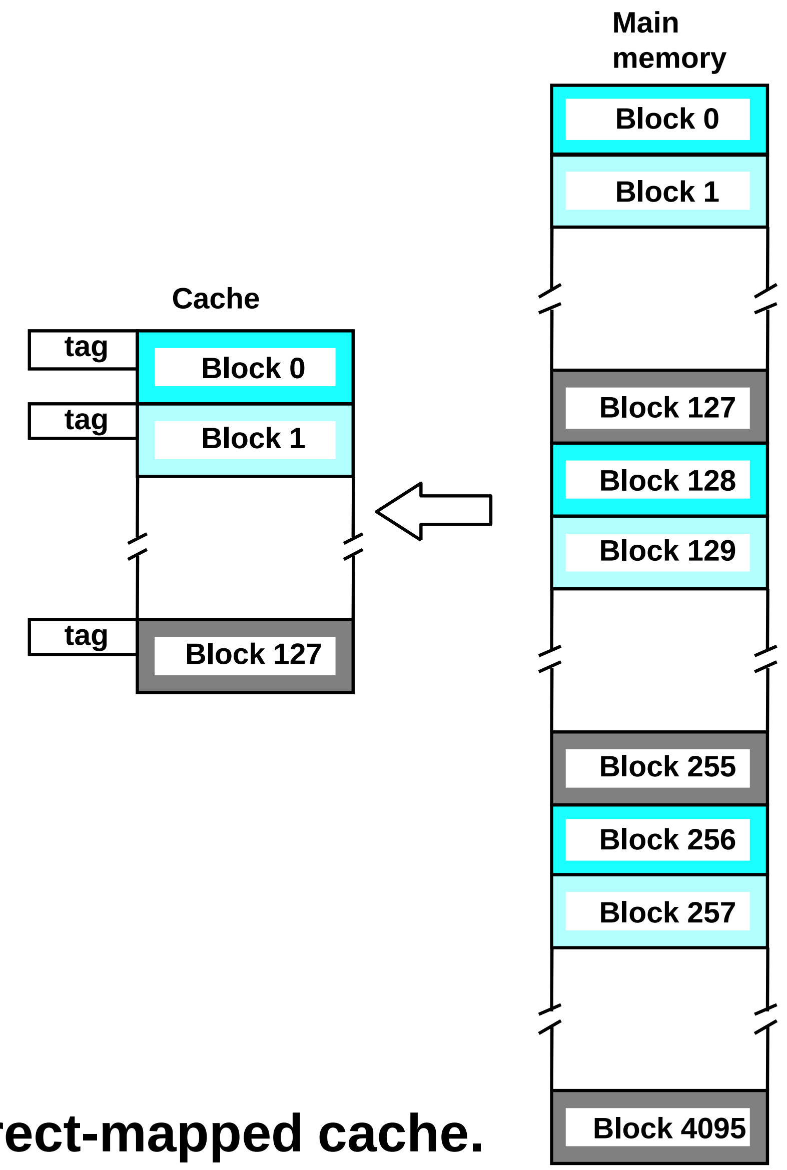
Tag (3 bits)	Cache Block Offset (2 bits)	Word Offset (4 Bits)

Example-2

4096 Main memory blocks

128 cache lines

Each block has 16 words

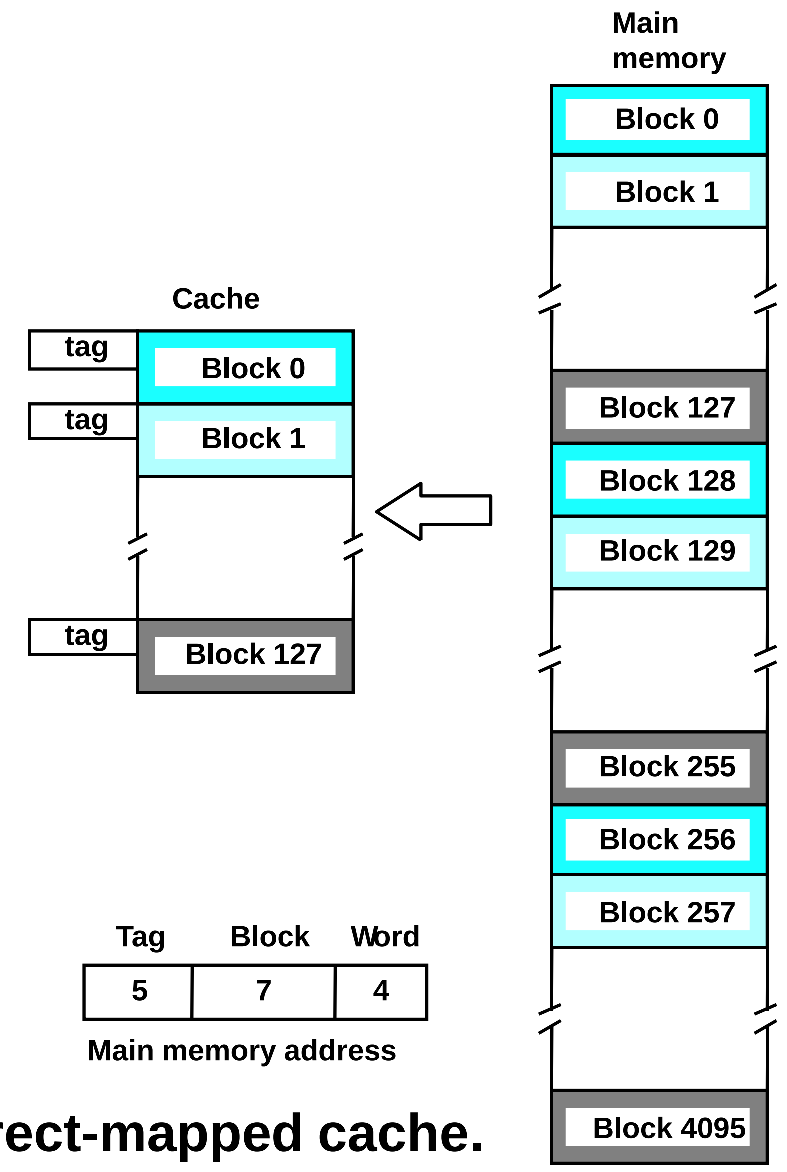


Direct-mapped cache.

4: one of 16 words. (each block has 16=2^4 words)

7: points to a particular block in the cache (128=2^7)

5: 5 tag bits are compared with the tag bits associated with its location in the cache. Identify which of the 32 blocks that are resident in the cache (4096/128).



Direct-mapped cache.

Limitation of Direct Mapping:

- Conflict penalties: It makes more blocks movements and hence the direct mapping is the slowest mapping technique.
- Example:
- M.M. Block: 0, 8, 0, 4, 8, 0, 8, 4, 12, 4, 12
  - Hit=0
- To overcome this problem, associative mapping is used.

Associative Mapping

- It avoids conflict penalty by relaxing the mapping rule.
- Any block of main memory can be placed anywhere in the cache.
- It is the fastest mapping but consumes more hardware.
- The associative mapping divides the physical address into 2 partitions.

Tag Bits	**Word Offset**

- Tag bits= $log_2 M$
- Word Offset= $log_2 B$
- M: no. of blocks in main memory, B: 1 Block size.

Example-1:

- Main memory size =512 words, Cache memory size=64 words, 1 block size=16 words. Obtain TAG bits and word offset.
- Solution:
- No. of blocks in main memory =512/16=32 blocks
- No. of blocks/lines in cache memory=64/16= 4 blocks
- Word offset ?
- Tag bits?

Tag (5 bits)	Word Offset (4 bits)

Example-2

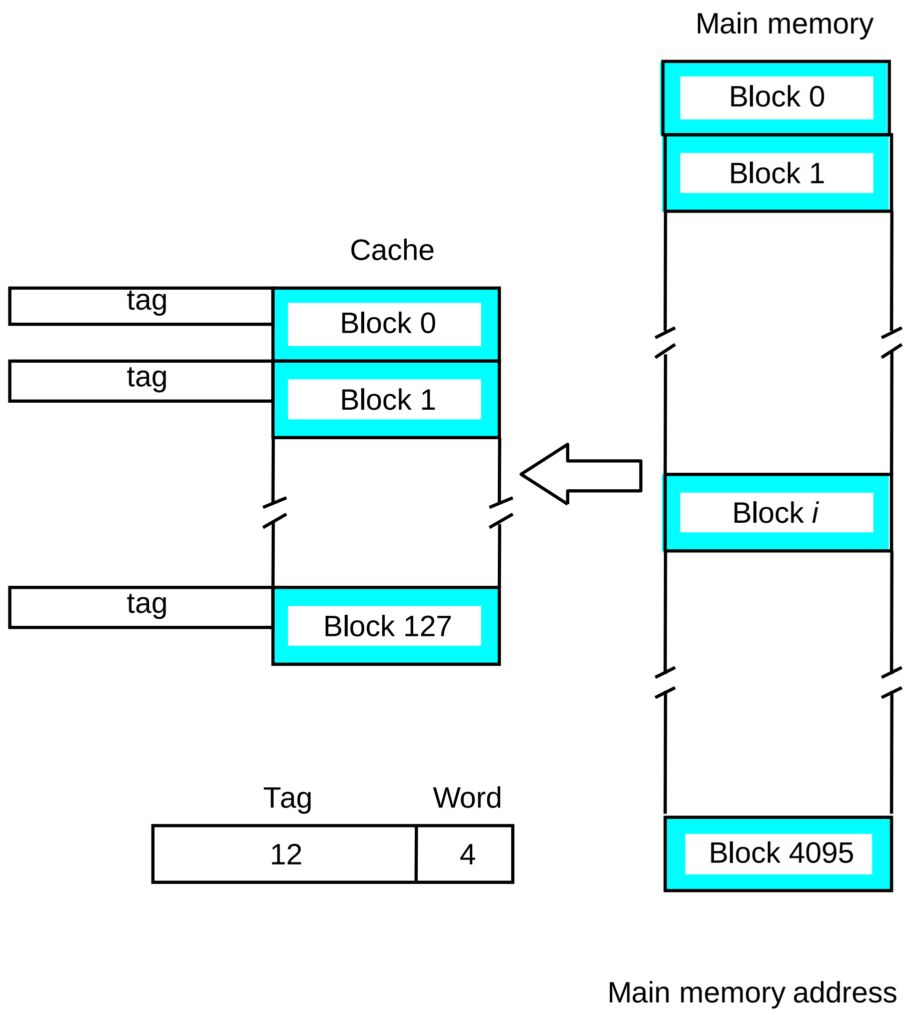
4096 Main memory blocks

128 cache lines

Each block has 16 words

4: one of 16 words. (each block has $16=2^4$ words)

12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache $4096=2^{12}$.



Associative-mapped cache.

Set Associative Mapping

- It divides the cache into logical sets.
- In *k*-way set association, each set contains k-cache blocks.
- No. of set (*S*) = *N*/*k*.

TAG	Set Offset	Word Offset

- Word Offset= $log_2 B$
- Set offset = $log_2 S$
- Tag bits= $log_2(\frac{M}{S})$

Example-1:

- Main memory size =512 words, Cache memory size=64 words, 1 block size=16 words. 2-way set-associative, Obtain TAG bits, set offset and word offset.
- Solution:
- No. of blocks in main memory =512/16=32 blocks
- No. of blocks/lines in cache memory=64/16= 4 blocks
- Word offset ?
- Set offset ?
- Tag bits?

Tag (4 bits)	**Set Offset (1 bits)**	**Word Offset (4 Bits)**

Example-2

4096 Main memory blocks

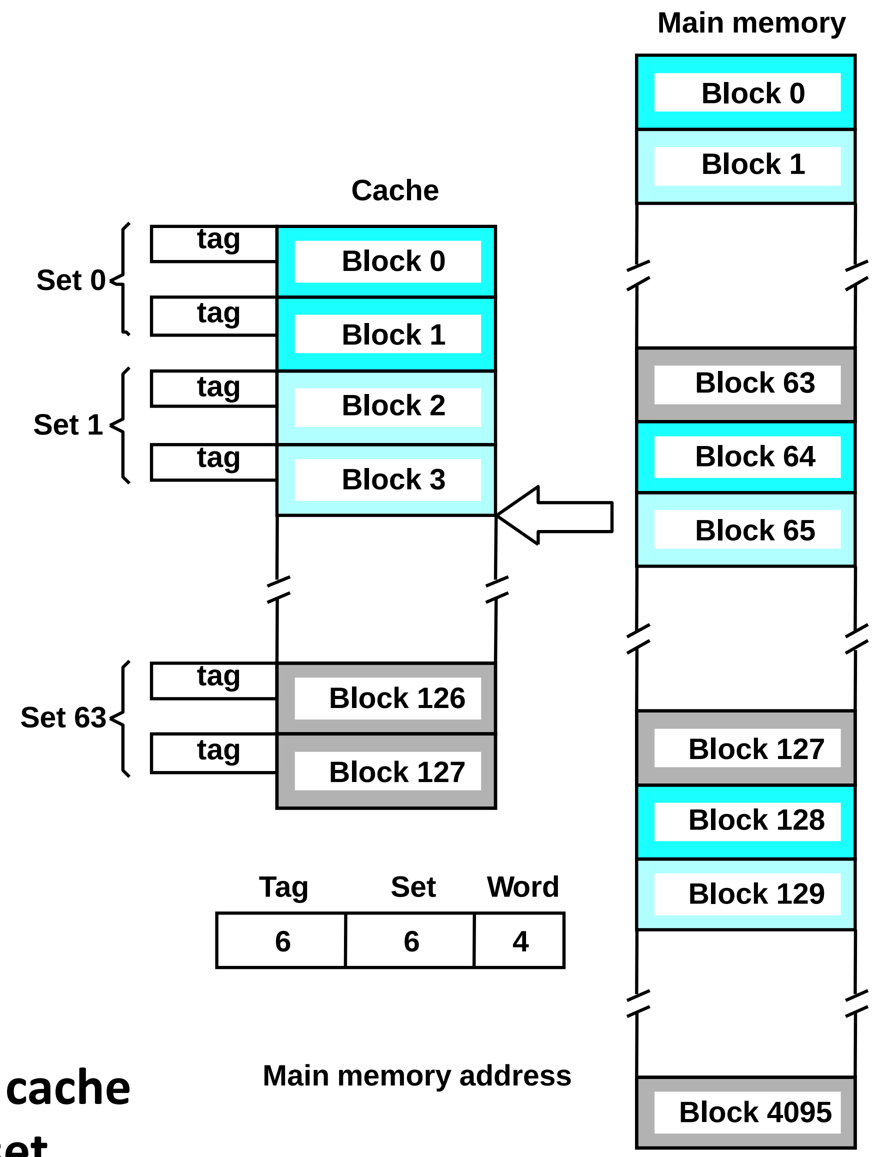
128 cache lines

Each block has 16 words

4: one of 16 words. (each block has $16=2^4$ words)

6: points to a particular set in the cache $(128/2=64=2^6)$

6: 6 tag bits is used to check if the desired block is present $(4096/64=2^6)$.



Set-associative-mapped cache with two blocks per set.

Address Mapping/Block Placement

- If a block has only one possible place in the cache: Direct Mapped
- If a block can be placed anywhere: Fully Associative
- If a block can be placed in a restricted subset of the possible places: Set Associative
    - If there are n blocks in each subset: n-way set associative
- Note that direct-mapped = 1-way set associative

Trade-offs

- n-way set associative becomes increasingly difficult (costly) to implement for large n
  - Most caches today are either 1-way (direct mapped), 2-way, or 4-way set associative
- The larger n the lower the likelihood of thrashing
  - e.g., two blocks competing for the same block frame and being accessed in sequence over and over

Question

Consider 2 GB main memory and 1MB cache memory. Block size is 256 word. Each word consumes 16 bits.

(i) How many bits are needs to physical address

(ii) How many blocks are presents in main memory and cache memory.

(iii) Identify no. of tag bits for the following

- (a) Direct mapping
- (b) Associative mapping
- (c) 4-way set associative mapping.

Solution

Memory is word addressable always. 1 word=16 bits=2 Byte

Main memory size= 2G Byte / 2 Byte=1G Words (30 bits)

Cache memory size= 1M Byte / 2 Byte = 512 K Words

1 block size=256 Words

No. of blocks in main memory= 1G/ 256= 4M Blocks

No. of blocks in cache memory= 512K/256=2K blocks

(a) Direct mapping: Tag=11

(b) Associative mapping: Tag=22

(c) Set-associative mapping: Tag=13

Four Basic Questions

- Q1: Where can a block be placed in the cache? *(Block placement) (ADDRESS MAPPING)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the cache? *(Block identification)*
  - Tag/Block/word
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU, etc.
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

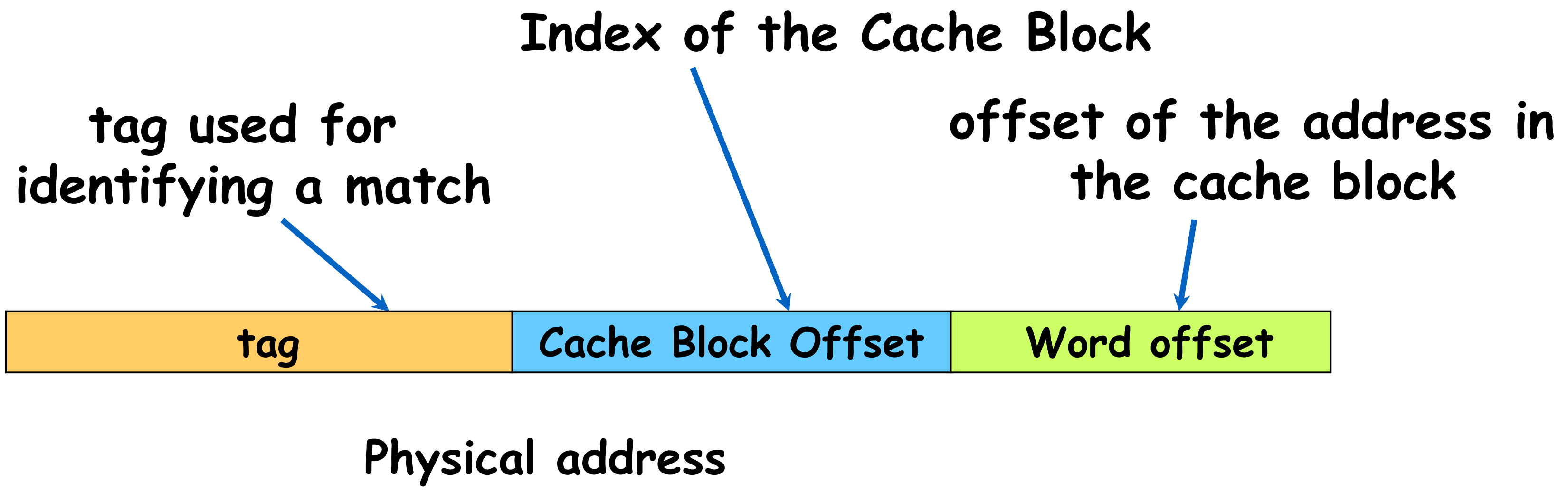
Block Identification

- Given an address, how do we find where it goes in the cache?
- Caches have an address tag on each block frame that gives the block address.
- Valid bit indicates that the cache block contains valid information.
- If the valid bit is not set, there can not be a match on this address.
- Physical address is divided into block address and block offset.

Block Address		Block Offset
Tag	Index	

Block Identification in Direct Mapping

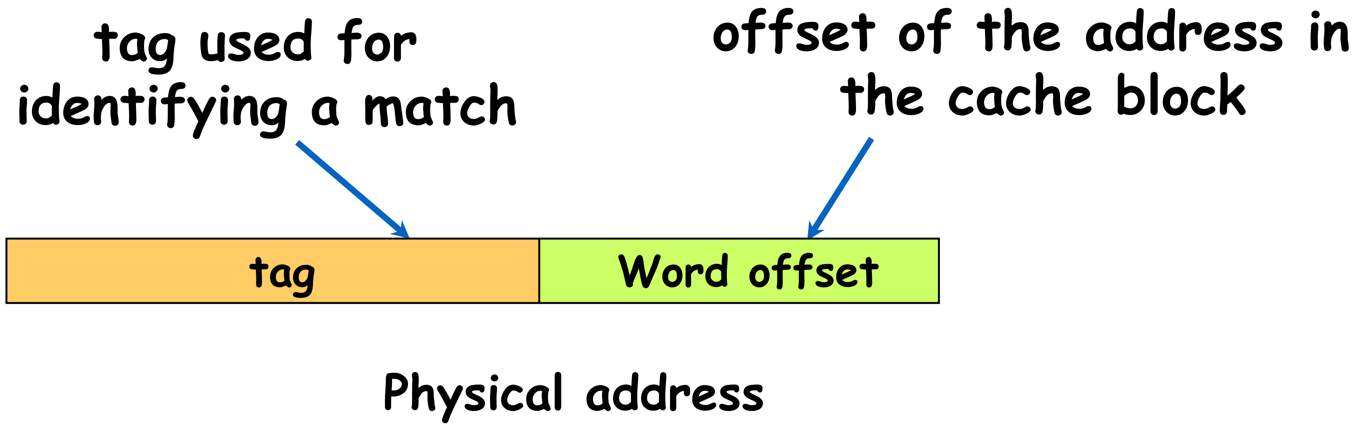
- This is done by first breaking down an address into three parts
- It requires one tag comparator.
- Inputs of tag comparator: Tag of Cache block, Tag of Physical address
- Output of tag comparator: Hit / Miss.



Index of the Cache Block
tag used for identifying a match
offset of the address in the cache block
tag
Cache Block Offset
Word offset
Physical address

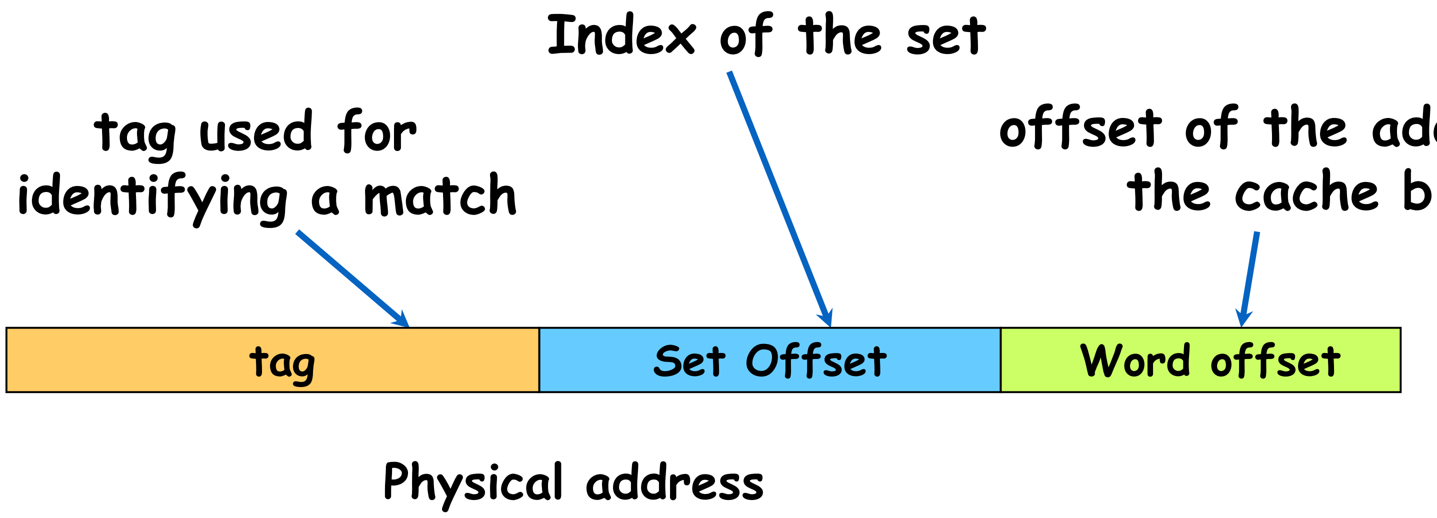
Block Identification in Associative Mapping

- This is done by first breaking down an address into two parts.
- It requires N tag comparator. (where N is the no. of cache blocks)


tag used for identifying a match
offset of the address in the cache block
tag
Word offset
Physical address

Block Identification in Set-Associative Mapping

- This is done by first breaking down an address into three parts
- It requires k tag comparator. (where k is the k-way set associative)


Index of the set
tag used for identifying a match
offset of the address in the cache block
tag
Set Offset
Word offset
Physical address

- Increasing in the associative
  - Increases the number of blocks in the set.
  - Decreases the size index field.
  - Increases the size tag field.
- Fully associative mapping has no index fields.

Four Basic Questions

- Q1: Where can a block be placed in the cache? *(Block placement) (ADDRESS MAPPING)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the cache? *(Block identification)*
  - Tag/Block/word
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - FIFO, LRU, Random, etc.
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

Block Replacement

- Which block should be replaced on a cache miss?
- When miss occurs, the cache controller must select a cache block to be replaced with the desired data.
- Three primary strategies employed for selecting which block to replace:

- FIFO: oldest block is replaced.

- LRU: the unused block for the longest time is replaced.

- Random: the random block is replaced.

Example-1

- Main Memory Block Reference: 1, 3, 0, 3, 5, 6, 3
- 4 blocks in cache memory
- Identify which blocks are to be replaced using FIFO rule?
- No of Hits?
- No. of Miss?
- Hit rate?
- Miss rate?

Types of Cache Miss

- Compulsory Miss
  - Initially cache is empty.
  - When processor starts its first request access for a word in the main memory, then a miss is occurred.
  - This type of miss occurred in all mapping technique.
- Conflict Miss
  - This type of miss occurred due to mapping rule in direct mapping and set associative mapping.
  - Even if cache memory is empty, still we cannot bring an address from main memory into cache memory.
- Capacity Miss
  - Due to smaller size of cache memory, all addresses of main memory can not be put into the cache memory.
  - This type of miss occurred in all mapping techniques.

Four Basic Questions

- Q1: Where can a block be placed in the cache? *(Block placement) (ADDRESS MAPPING)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the cache? *(Block identification)*
  - Tag/Block/word
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU, etc.
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

Cache Write Policies

- Write-through: Information is written to both the block in the cache and to the block in lower level memory.
- Write-back: Information is written only to the block in the cache. The modified block is written to main memory only when it is replaced.

Dirty bit in write back??

- To reduce the frequency of writing back blocks on replacements, a feature called dirty bit is commonly used.
- This status bit indicates whether the block is dirty (modified while in cache) or clean (not modified).
- If it is clean, the block is not written back on a miss, since both have identical information to the cache is found in lower levels.

Trade-offs

- Write back
  - Faster because writes occur at the speed of the cache, not the memory.
  - Faster because multiple writes to the same block is written back to memory only once, uses less memory bandwidth.
- Write through
  - Easier to implement

Write through with write buffer

- When the processor must wait for writes to complete during write through, the processor is said to be write stall.
- Write buffer is used to reduce the write stalls.
- It allows the process to continue as soon as the data are written to the buffer, thereby overlapping processor execution with memory updating.

Write Allocate, No-write Allocate

- On a read miss, a block has to be brought in from a lower level memory
- What happens on a write miss?
- Two options:
- Write allocate: A block is allocated in cache.
- No-write allocate: No block allocation, but just written to in main memory.

Write Allocate, No-write Allocate cont…

- In no-write allocate,
  - Only blocks that are read from can be in cache.
  - Write-only blocks are never in cache.
- But typically:
  - write-allocate used with write-back
  - no-write allocate used with write-through

Example-1: Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in square brackets):

Write Mem[100];

WriteMem[100];

ReadMem[200];

WriteMem[200];

WriteMem[100].

What are the number of hits and misses when using no-write allocate VS write allocate?

Answer

***For no-write allocate* the address 100 is not in the cache, and there is no allocation** on write, so the first two writes will result in misses.

Address 200 is also not in the cache, so the read is also a miss. The subsequent write to address 200 is a hit. The last write to 100 is still a miss.

- The result for no-write allocate is four misses and one hit.

For write allocate, the first accesses to 100 and 200 are misses, and the rest are hits since 100 and 200 are both found in the cache.

- Thus, the result for write allocate is two misses and three hits.

Example-2: Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in square brackets):

Write Mem[400];

WriteMem[500];

ReadMem[100];

WriteMem[500];

WriteMem[100].

What are the number of hits and misses when using no-write allocate VS write allocate?

Example-3: Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in square brackets):

Write Mem[100];

ReadMem[300];

WriteMem[200];

ReadMem[100];

ReadMem[400];

WriteMem[200];

WriteMem[300].

What are the number of hits and misses when using no-write allocate VS write allocate?