Lecture 24, Lecture 25 and Lecture 26
Cache Memory

CSC208A-Computer Organisation and Architecture

B. Tech. 2016

Course Leader:

Chaitra S.

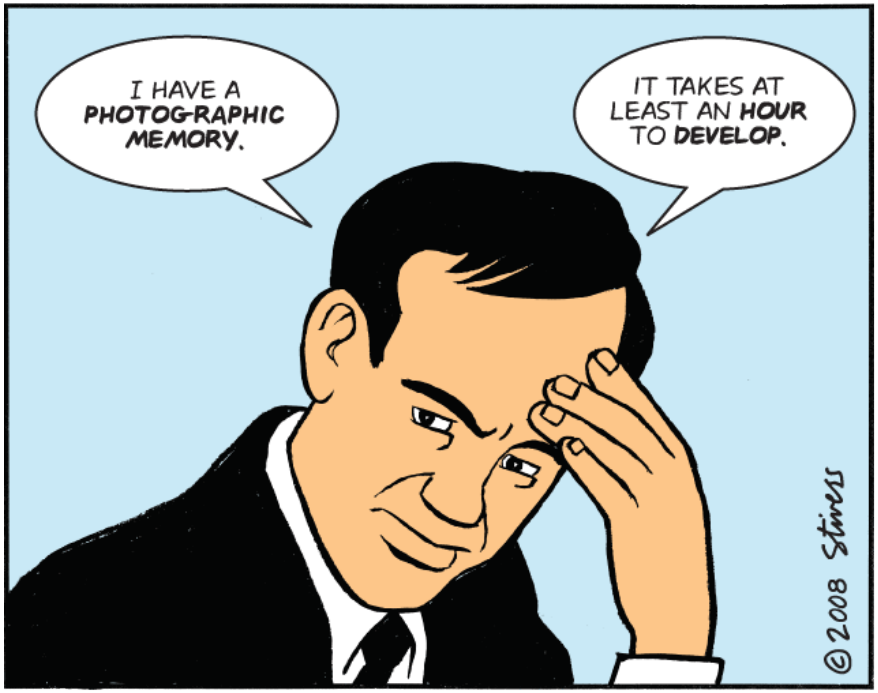
Naveeta

Objectives

- At the end of these lectures, students will be able to
  - Identify the need for the cache memory system
  - Distinguish between the cache mapping techniques
  - Describe basic functional concepts of caching such as hits, misses, write through and write back
  - Analyze the effect that cache has on the performance of the system

Contents

- Design and Overview of Operation
- Cache Mapping
  - Direct Mapped
  - Fully Associative
  - Set Associative
- Cache hits and misses
- Write Through and Write Back
- Cache Performance



Cache Memory

- Analysis of large number of programs has shown that a number of instructions are executed repeatedly. This may be in the form of a simple loops, nested loops, or a few procedures that repeatedly call each other. It is observed that many instructions in each of a few localized areas of the program are repeatedly executed, while the remainder of the program is accessed relatively less. This phenomenon is referred to as **locality of reference.**

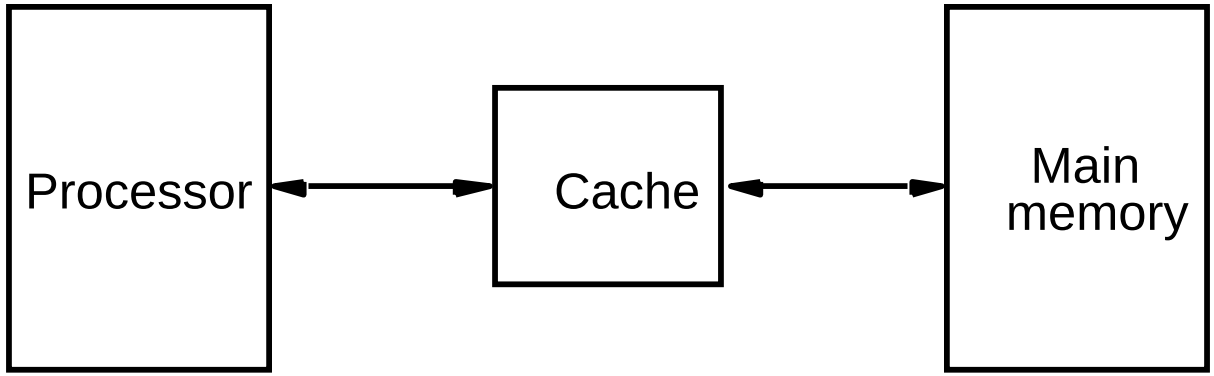


Figure : Cache memory between Processor and the main memory

Cont..

- Now, if it can be arranged to have the active segments of a program in a fast memory, then the total execution time can be significantly reduced.
- CPU is a faster device and memory is a relatively slower device. Memory access is the **main bottleneck** for the **performance efficiency**. If a faster memory device can be inserted between main memory and CPU, the efficiency can be increased. The faster memory that is inserted between CPU and MM is termed as **Cache memory**. To make this arrangement effective, the cache must be considerably faster than the MM, and typically it is 5 to 10 time faster than the main memory.
- This approach is more economical than the use of fast memory device to implement the entire MM. This is also a feasible due to the locality of reference that is present in most of the program, which reduces the frequent data transfer between MM and cache memory. The inclusion of cache memory between CPU and MM is shown in Figure.

Cache Memory

- Processor is much faster than the main memory.
- As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
  - Major obstacle towards achieving good performance.
- Speed of the main memory cannot be increased beyond a certain point.
- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- Cache memory is based on the property of computer programs known as “locality of reference”.

Locality of Reference

- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.

- These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.

- This is called “locality of reference”.

Locality of Reference

- Temporal locality of reference:
  - Recently executed instruction is likely to be executed again very soon.

- Spatial locality of reference:
  - Instructions with addresses close to a recently instruction are likely to be executed soon.

Cache Memories

- Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.
- Subsequent references to the data in this block of words are found in the cache.
- At any given time, only some blocks in the main memory are held in the cache.
  - Which blocks in the main memory are in the cache is determined by a “mapping function”.
- When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced.
  - This is determined by a “replacement algorithm”.

Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Cache Hit

- Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.
- If the data is in the cache it is called a Read or Write hit.
- Read hit:
  - The data is obtained from the cache.

Cache Hit

- Write hit:
  - Cache has a replica of the contents of the main memory.
  - Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol.
  - Update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol.

Cache Miss

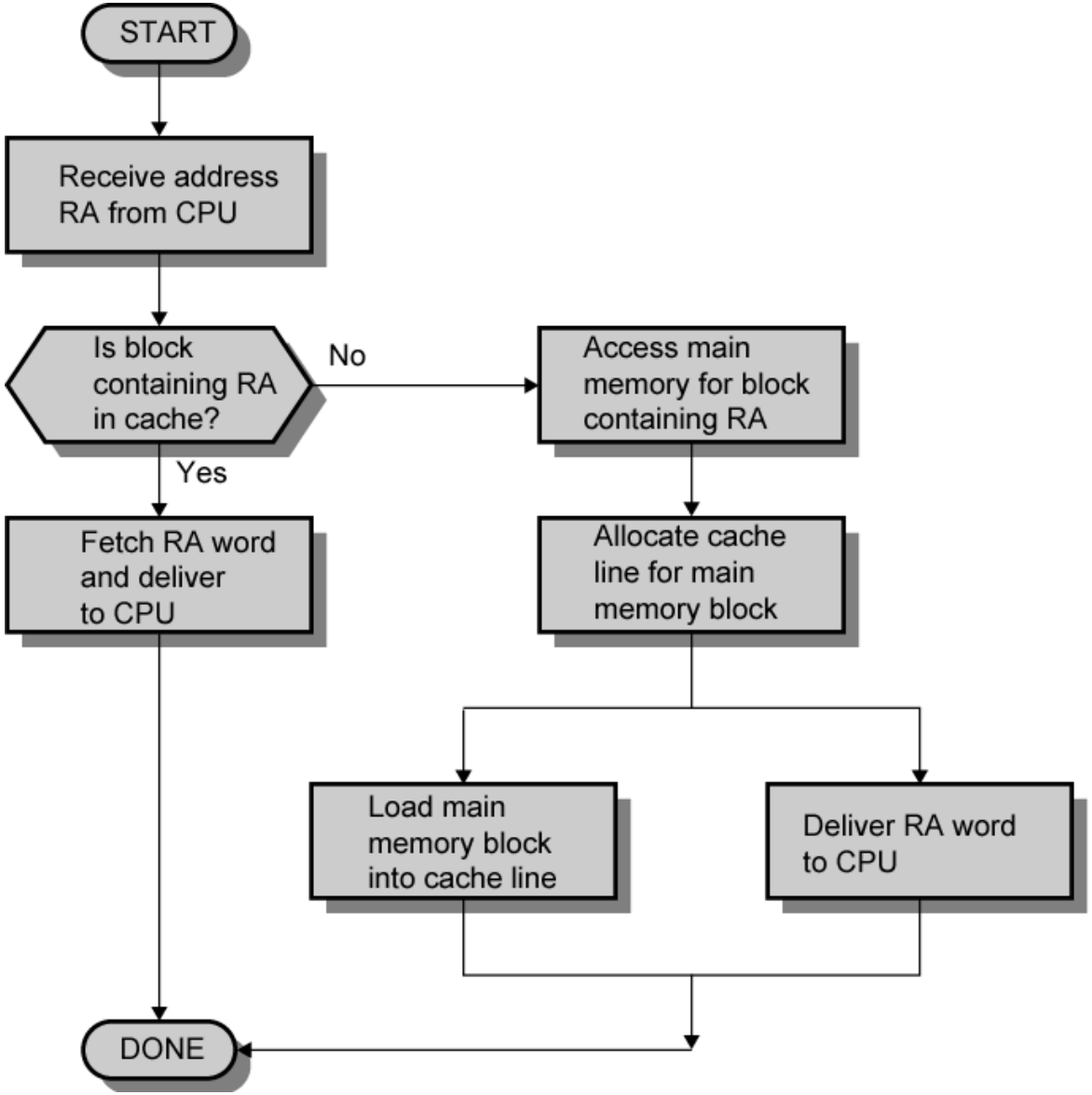
- If the data is not present in the cache, then a Read miss or Write miss occurs.

- Read miss:
  - Block of words containing this requested word is transferred from the memory.
  - After the block is transferred, the desired word is forwarded to the processor.
  - The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.

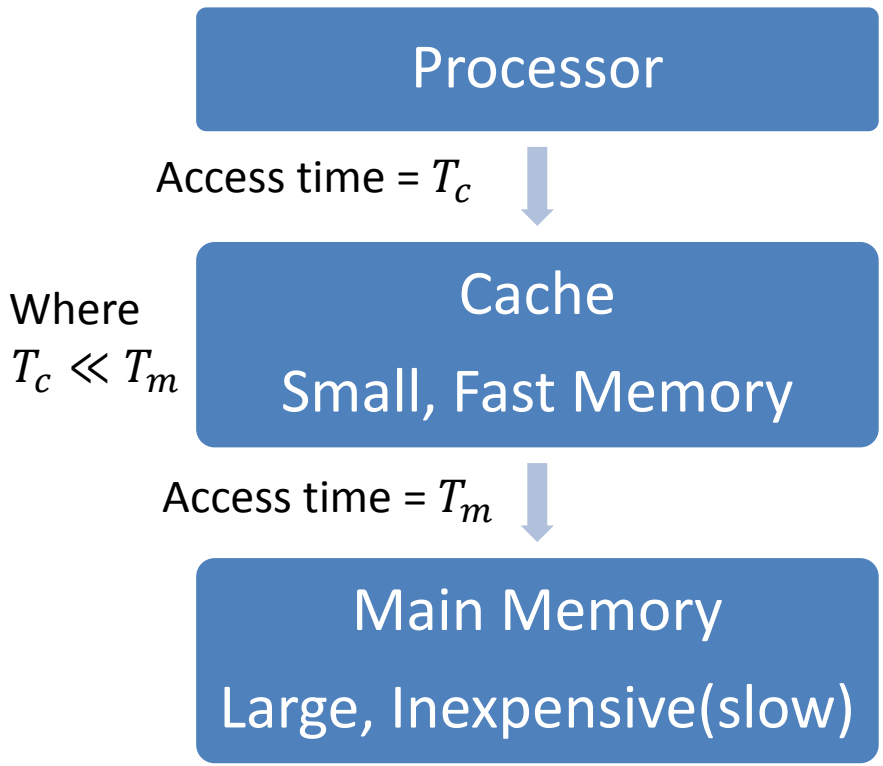
Cache Miss

- Write-miss:
  - Write-through protocol is used, then the contents of the main memory are updated directly.
  - If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.

Cache Read Operation - Flowchart



Cache Performance

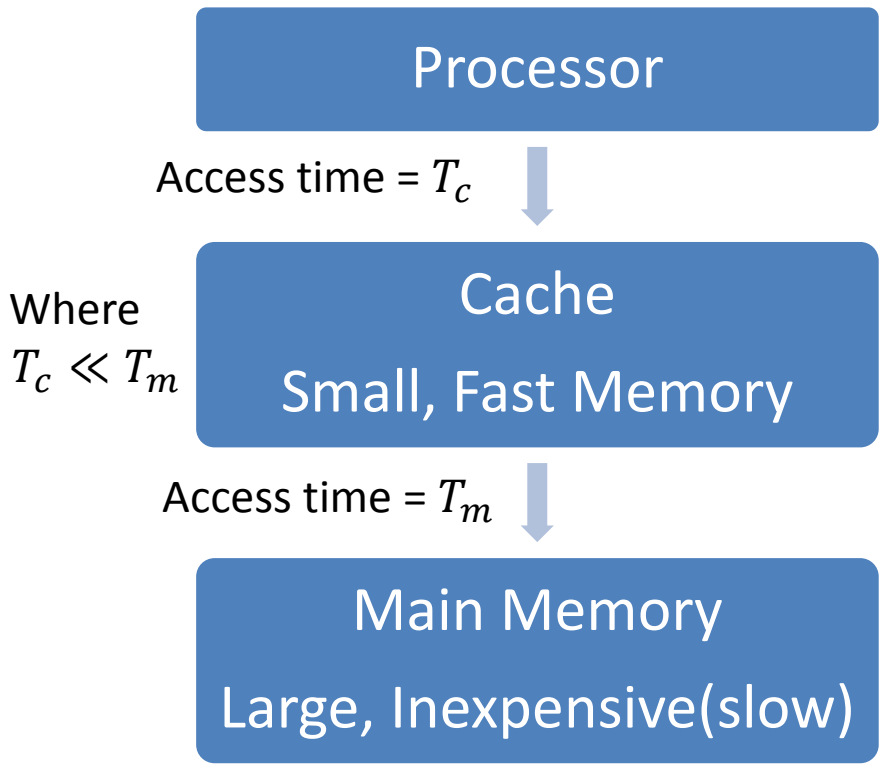


- **Average access time**

  $= T_c \times h + (T_m + T_c) \times (1 - h)$

  $= T_c + T_m \times (1 - h)$
- Where
- T_c = Cache access time(small)
- T_m = Memory access time(large)
- h = Hit rate($0 \leq h \leq 1$)
- Hit rate is also known as hit ratio.
- Miss rate = $1 - Hit\ rate$

Cont..

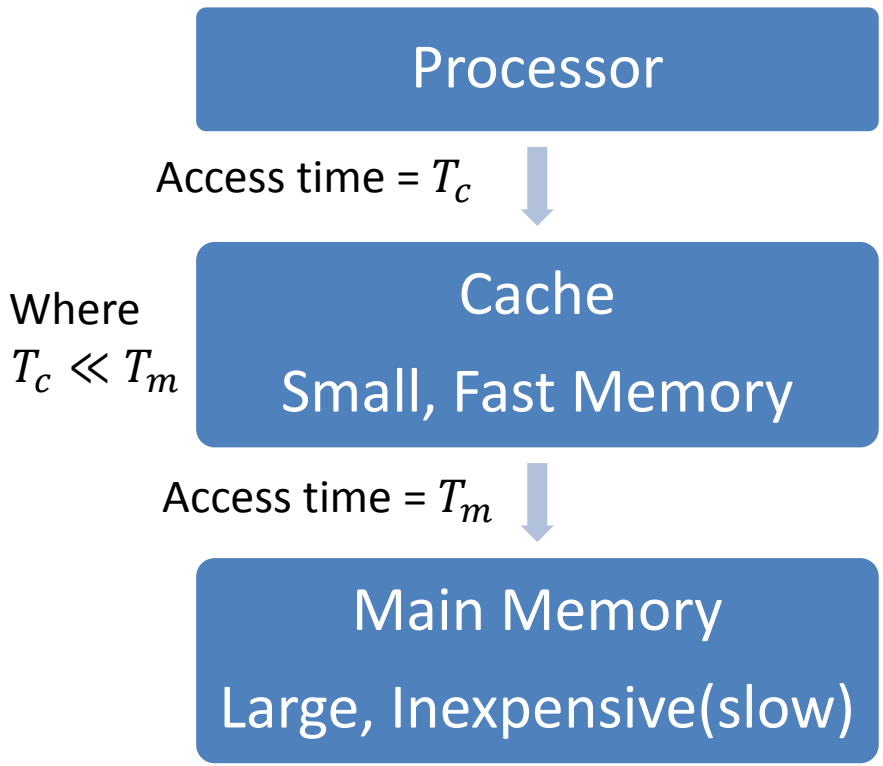


- **Average access time**

  $= T_c \times h + (T_m + T_c) \times (1 - h)$

  Where
- T_c = Cache access time
- T_m = Memory access time
- h = Hit rate
- **Example:** ***T_c = 1ns, T_m = 30ns, hit rate = 0.9***
- Total No of access = 50
- Access Tome(**without cache**) =
- Access Tome(**with cache**) =

Cont..



- **Average access time**

  $= T_c \times h + (T_m + T_c) \times (1 - h)$

  Where
- T_c = Cache access time
- T_m = Memory access time
- h = Hit rate
- **Example:** ***T_c = 1ns, T_m = 30ns, hit rate = 0.9***
- Total No of access = **50**
- Access Time(**without cache**)

  = 50×30 = **1500 ns**
- Access Time(**with cache**)

  = [50×0.9×1+50×0.1×(30+1)] ns

  = **200 ns**

Cache Coherence Problem

- A bit called as “valid bit” is provided for each block.
- If the block contains valid data, then the bit is set to 1, else it is 0.
- Valid bits are set to 0, when the power is just turned on.
- When a block is loaded into the cache for the first time, the valid bit is set to 1.
- Data transfers between main memory and disk occur directly bypassing the cache.

Cache Coherence Problem

- When the data on a disk changes, the main memory block is also updated.
- However, if the data is also resident in the cache, then the valid bit is set to 0.
- What happens if the data in the disk and main memory changes and the write-back protocol is being used?
- In this case, the data in the cache may also have changed and is indicated by the dirty bit.
- The copies of the data in the cache, and the main memory are different. This is called the cache coherence problem.
- One option is to force a write-back before the main memory is updated from the disk.

Cache Mapping

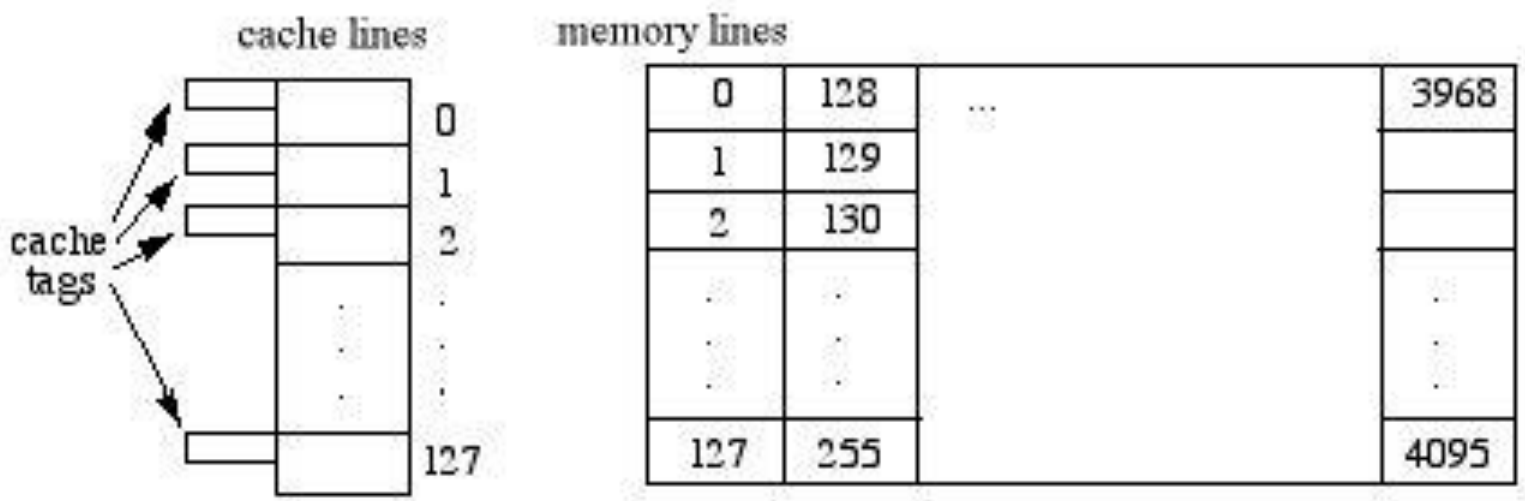
- Mapping functions determine how memory blocks are placed in the cache
- A simple processor example:
  - Cache consisting of $128(2^7)$ blocks of $16(2^4)$ words each.
  - Total size of cache is $128(2^7) \times 16(2^4) = 2048(\ 2^{11}\ ) = 2 \times 1024 = 2K$ words.
  - Main memory is addressable by a 16-bit address.
  - Main memory has $2^{16} = 64 \times 1024\ = 64K$ words.
  - Main memory has 4Kblocks of 16 words each.

Cache Mapping

- Direct mapping
- Associative mapping
- Set-associative mapping

Cont..

- Example, suppose the cache has 2^7 = 128 lines, each with 2^4 = 16 words. Suppose the memory has a 16-bit address, so that 2^{16} = 64K words are in the memory's address space.



Direct Mapping

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- The mapping is expressed as follows: $i = j \; modulo \; m$
  - Where i = cache line number, j = main memory block number, m = number of lines in cache
- MM address can be viewed as consisting of 3 fields
  - Least Significant w bits identify unique word within a block of MM
  - Most Significant s bits specify one of 2^s memory block
  - The MSBs are split into a cache line field r and a tag of s-r (most significant). It identifies one of the m = 2^r lines of the cache.

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w}$ = 2s
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits

Direct Mapping Address Structure

Tag s-r	Line or Slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (2^2 = 4 byte block)
- 22 bit block identifier (for MM)
  - 14 bit slot or line for cache
  - 8 bit tag (= 22-14)
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Cont..

	Tag s-r	Line or Slot r	Word w
Bits:	5	7	4

- Suppose the cache has 2^7 = 128 lines, each with 2^4 = 16 words. And the memory has a 16-bit address, so that 2^{16} = 64K words are in the memory's address space.
- Here, the "Word" field selects one from among the 16 addressable words in a line. The "Line" field defines the cache line where this memory line should reside. The "Tag" field of the address is then compared with that cache line's 5-bit tag to determine whether there is a hit or a miss.
- If there's a miss, we need to swap out the memory line that occupies that position in the cache and replace it with the desired memory line.

Cont..

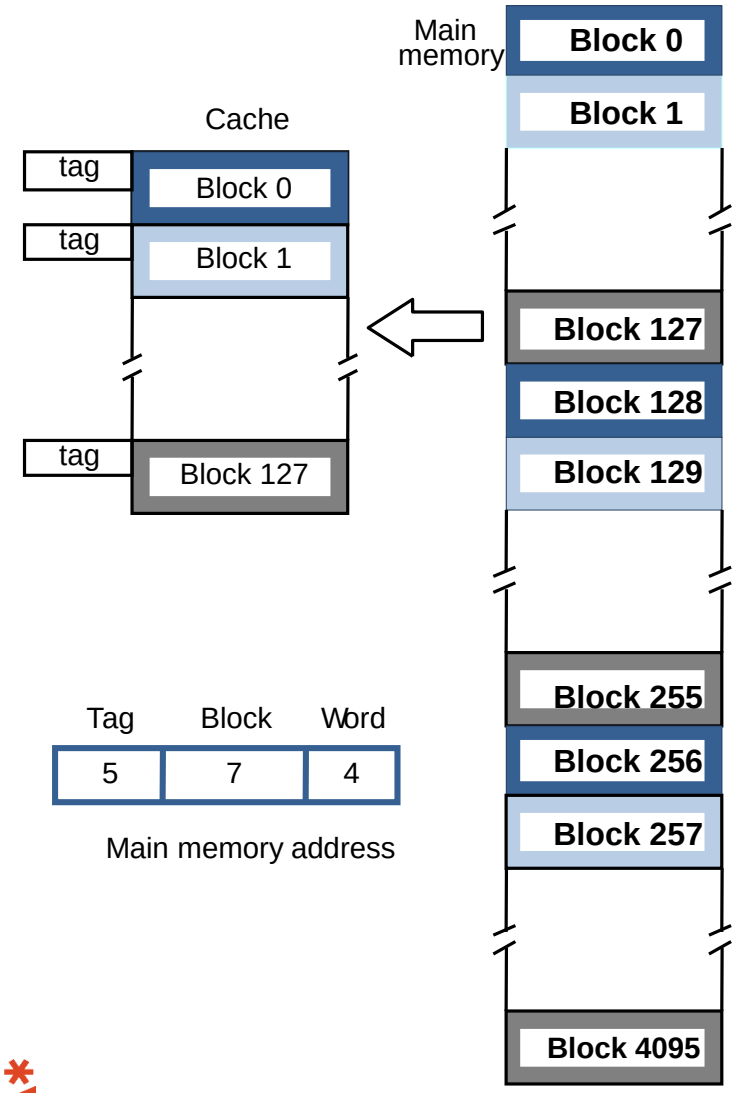
- Example, Suppose we want to read or write a word at the address 357A, whose 16 bits (in binary) are 0011(3) 0101(5) 0111(7) 1010(A).
- This translates to Tag = 6(00110, 5 bits), line = 87(1010111, 7 bits), and Word = 10 (1010, 4 bits)(all in decimal). If line 87 in the cache has the same tag (6), then memory address 357A is in the cache. Otherwise, a miss has occurred and the contents of cache line 87 must be replaced by the memory line 001101010111 = 855 before the read or write is executed.
- Direct mapping is the most efficient cache mapping scheme, but it is also the least effective in its utilization of the cache - that is, it may leave some cache lines unused.

Direct Mapping Cache Line Table

- The effect of this mapping is that blocks of main memory are assigned to lines of the cache as follows:

Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2s-m
1	1,m+1, 2m+1…2s-m+1
m-1	m-1, 2m-1,3m-1…2s-1

Direct Mapping



Tag	Block	Word
5	7	4

Main memory address

- Block j of the main memory maps to j modulo 128 of the cache.
  - 0 maps to 0, 129 maps to 1.

Memory Blocks 0,128,256, ...maps to cache block 0. Memory Blocks 1, 129, 257, ...maps to cache block 1. And so on.

- More than one memory block is mapped onto the same position in the cache.
  - May lead to contention for cache blocks even if the cache is not full.
- Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.
- Memory address is divided into three fields
  - Low order 4 bits determine one of the 16 words in a block.
  - When a new block is brought into the cache, the the next 7 bits determine which cache block this new block is placed in.
  - High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits. = *No of Blocks of memory / No of Blocks of cache* = 4096/128 = 32 = 2^5
- Simple to implement but not very flexible.

Direct Mapping Cache Organization

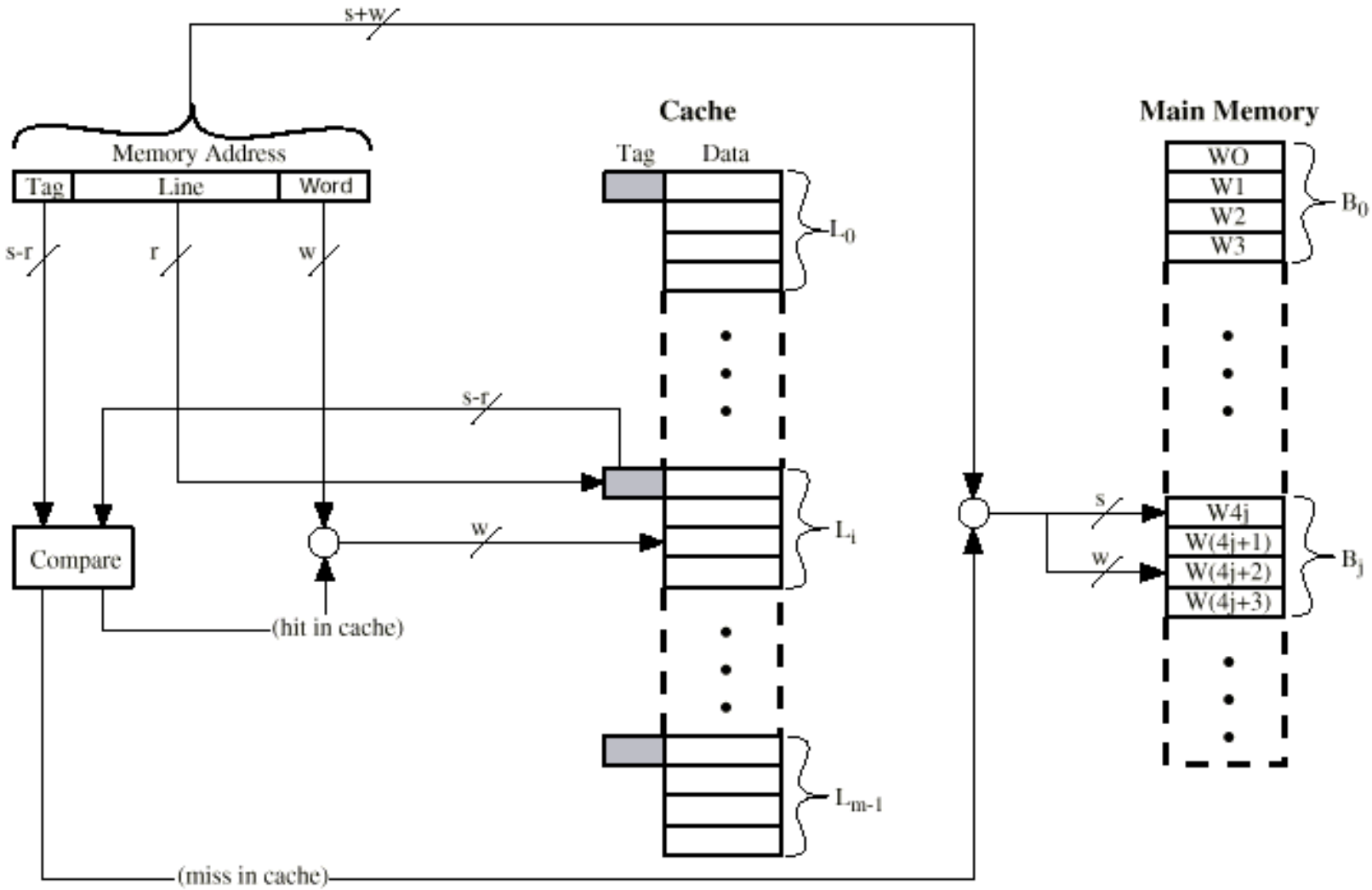
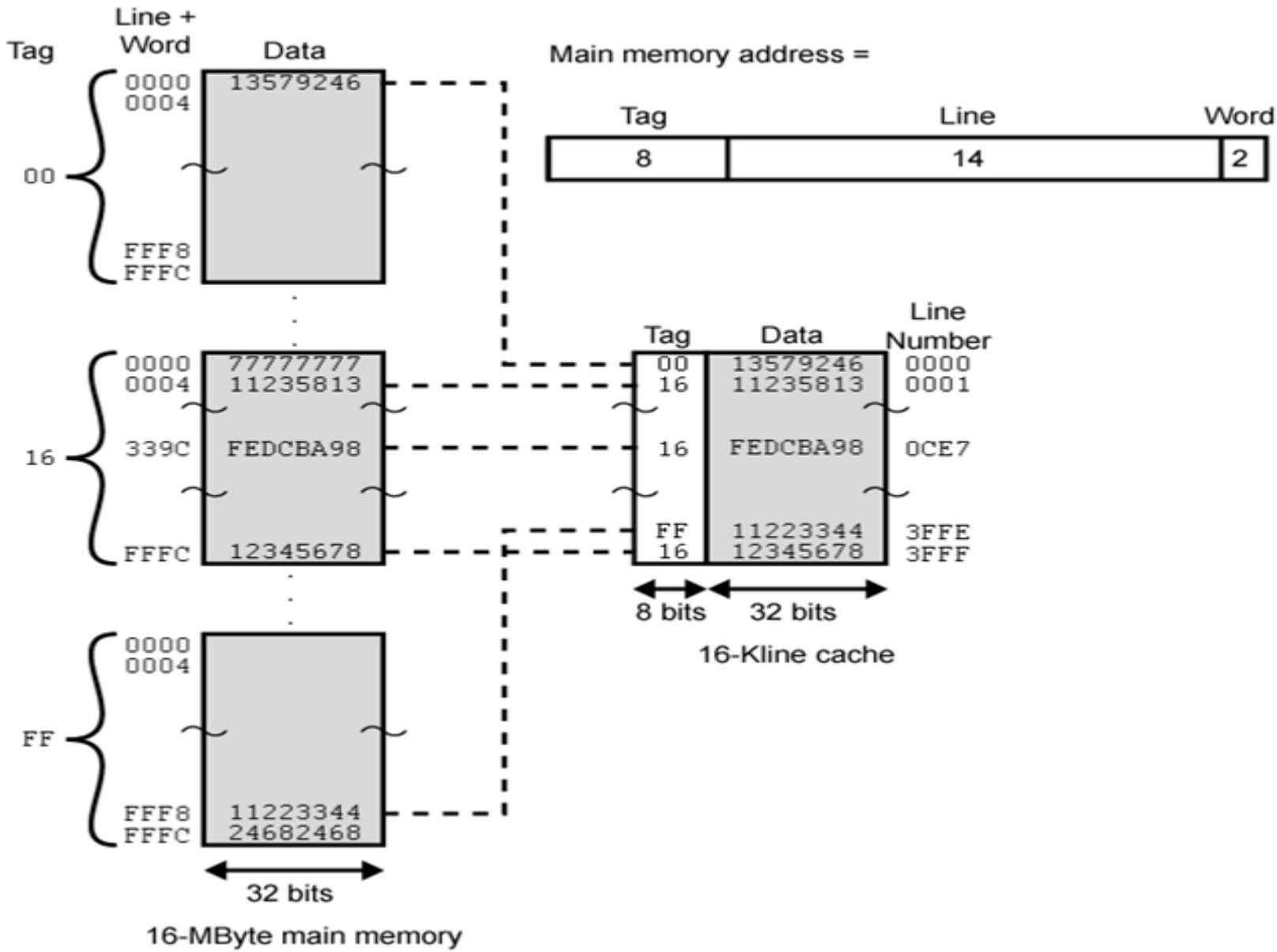


Figure illustrates the general mechanism of Direct Mapping.

Direct Mapping Example



Cont..

- Note: no two blocks that map into the same line number have the same tag number. Thus, blocks with starting addresses 000000, 010000, FF0000 have tag numbers 00, 01, FF, respectively.
- The cache system is presented with a 24-bit address. The 14-bit line number is used as an index into the cache to access a particular line. If the 8-bit tag number matches the tag number currently stored in that line, then the 2-bit word number is used to select one of the 4 bytes in that line.
- Otherwise, the 22-bit tag-plus-line field is used to fetch a block from MM. The actual address that is used for the fetch is the 22-bit tag-plus-line concatenated with two 0 bits, so that 4 bytes are fetched starting on a block boundary.

Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed Cache location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high
  - Thus, if a program have to refer words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as **thrashing**).

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

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w}$ = 2^{s}
- Number of lines in cache = undetermined
- Size of tag = s bits

Cont..

- This mapping scheme attempts to improve cache utilization, but at the expense of speed.
- Here, the cache line tags are 12 bits, rather than 5, and any memory line can be stored in any cache line. The memory address looks like this:

	Tag s	Word w
Bits:	12	4

- Suppose the memory has a 16-bit address, so that 2^{16} = 64K words are in the memory's address space, each with 2^4 = 16 words.

Cont..

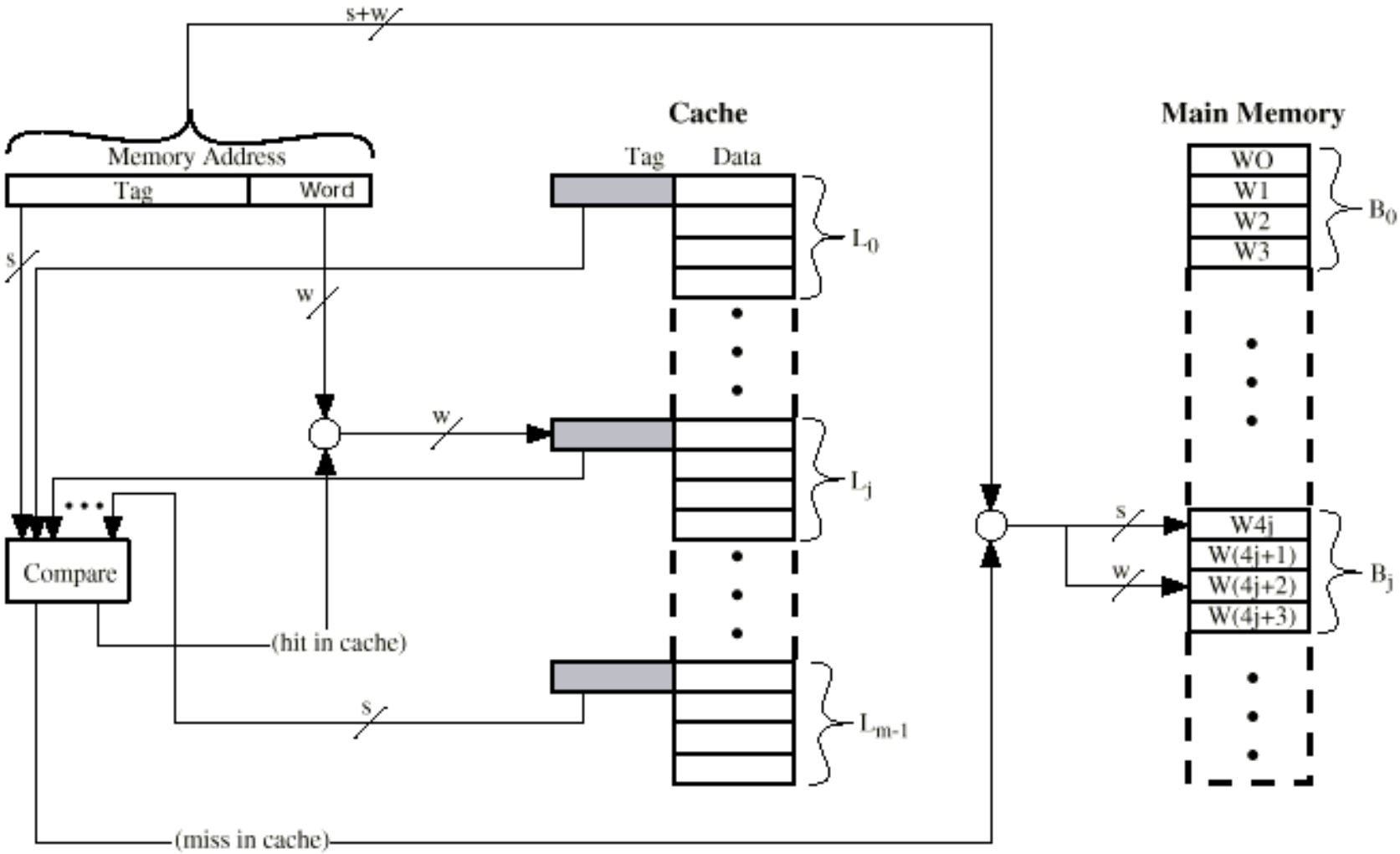
	Tag s	Word w
Bits:	12	4

- Here, the "Tag" field identifies one of the 2^{12} = 4096 memory lines;
- All the cache tags are searched to find out whether or not the Tag field matches one of the cache tags.
- If so, we have a hit, and if not there's a miss and we need to replace one of the cache lines by this line before reading or writing into the cache.
- The "Word" field again selects one from among 16 addressable words (bytes) within the line.

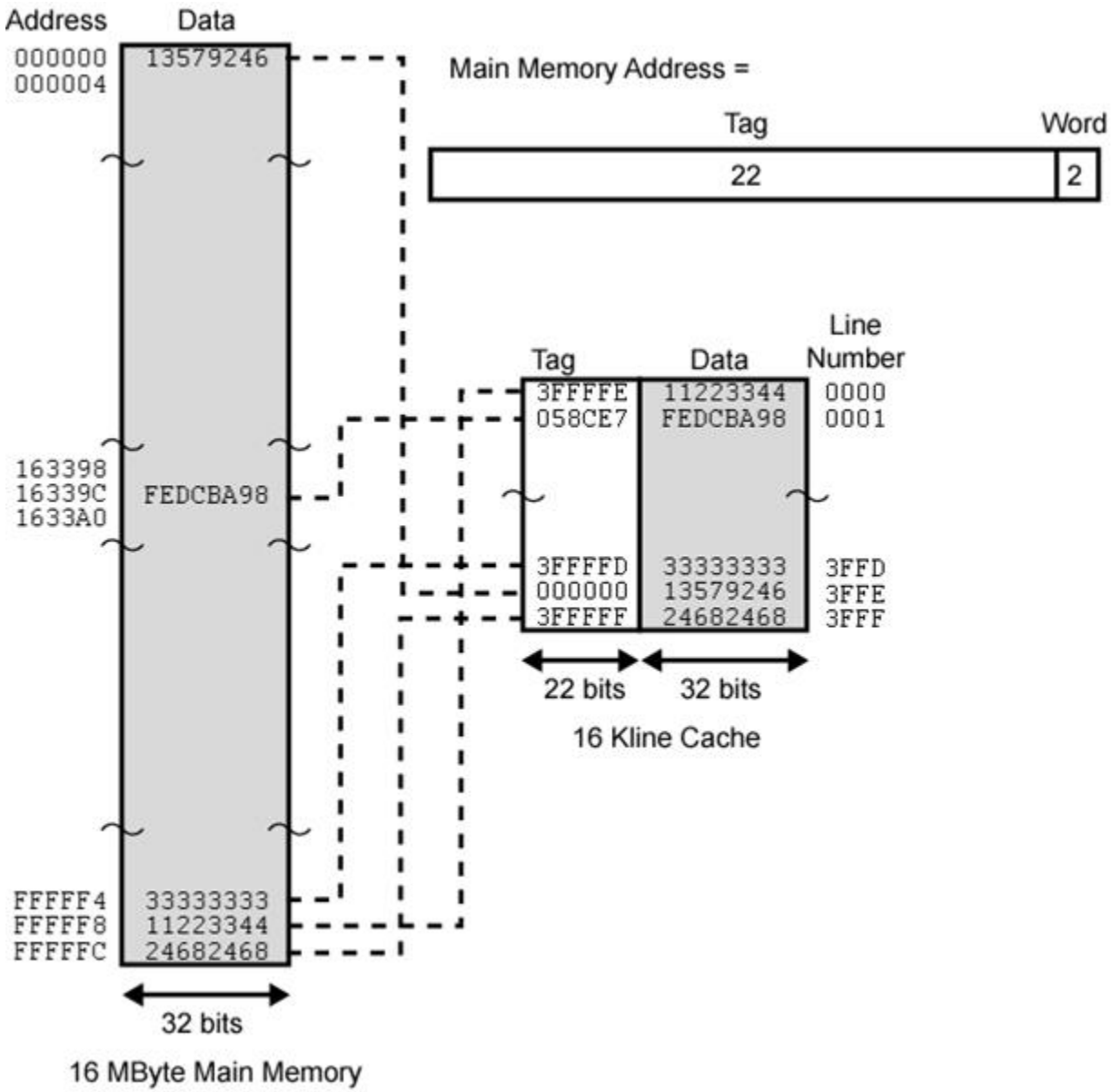
Cont..

- Example: Suppose we want to read or write a word at the address 357A, whose 16 bits (in binary) are 0011(3) 0101(5) 0111(7) 1010(A).
- Under associative mapping, this translates to Tag = 855(001101010111, 12 bits) and Word = 10 (1010, 4 bits)(in decimal).
- So we search all of the 128 cache tags to see if any one of them will match with 855.
- If not, there's a miss and we need to replace one of the cache lines with line 855 from memory before completing the read or write. The search of all 128 tags in the cache is time-consuming. However, the cache is fully utilized since none of its lines will be unused prior to a miss (recall that direct mapping may detect a miss even though the cache is not completely full of active lines).

Fully Associative Cache Organization



Associative Mapping Example



Associative Mapping Address Structure

Tag	Word
22 bit	2 bit

- MM address consists of a 22-bit tag and a 2-bit word. The 22-bit tag must be stored with the 32-bit block of data for each line in the cache. Note that it is the leftmost (most significant) 22 bits of the address that form the tag. Thus, the 24-bit hexadecimal address 16339C has the 22-bit tag 058CE7. This is easily seen in binary notation:

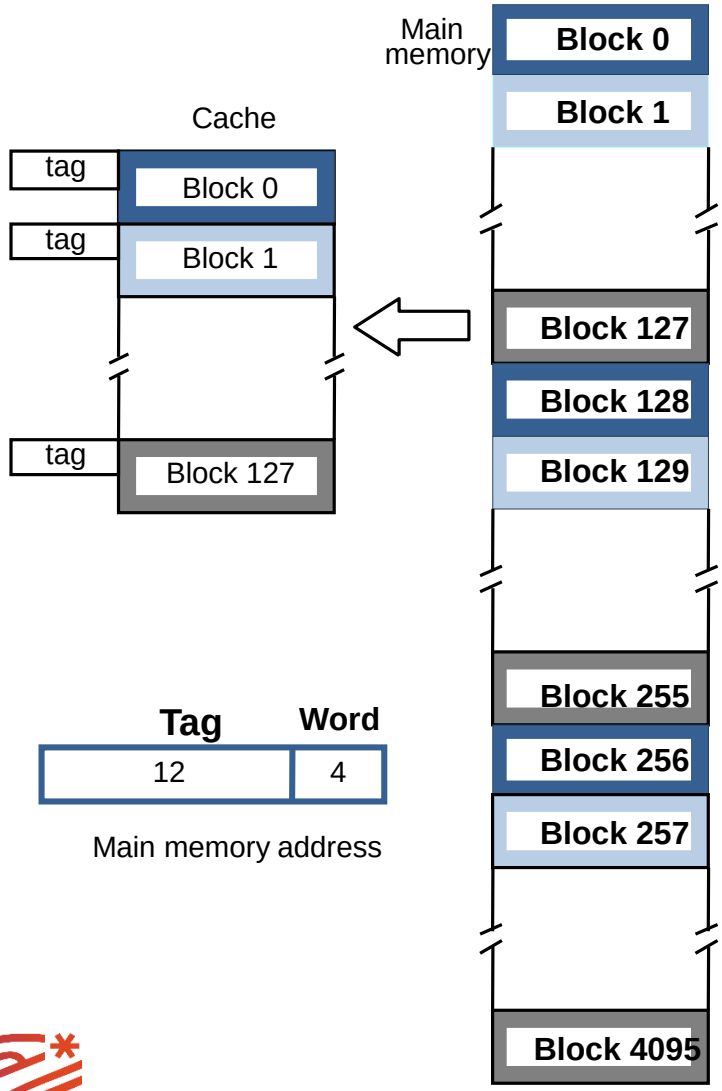
• memory address	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
• tag (leftmost 22 bits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)

Associative Mapping Address Structure

Tag	Word
22 bit	2 bit

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block
- e.g.
  - Address Tag Data Cache line
  - FFFFFC 3FFFFF 24682468 3FFF
  - 1111(F), 1111(F), 1111(F), 1111(F), 1100(C) [Address(24 bit)]
  - 11(3)1111(F) 1111(F) 1111(F) 1111(F) 11 11(F) [Tag(22bit)]

Associative Mapping



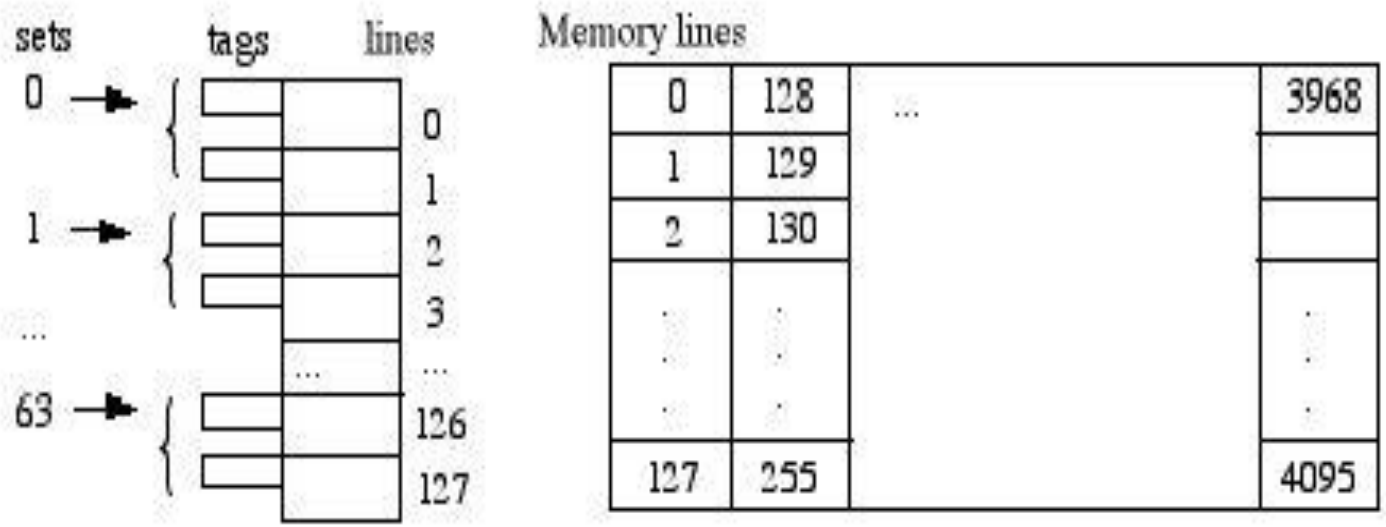
- Main memory block can be placed into any cache position.
- Memory address is divided into two fields:
  - Low order 4 bits identify the word within a block.
  - High order 12 bits or tag bits identify a memory block when it is resident in the cache.
- Flexible, and uses cache space efficiently.
- Replacement algorithms can be used to replace an existing block in the cache when the cache is full.
- Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Cont..

- This scheme is a compromise between the direct and associative schemes.
- Here, the cache is divided into sets of tags, and the set number is directly mapped from the memory address (e.g., memory line j is mapped to cache set j mod 64), as suggested by the diagram below:



Cont..

- Here the cache consists of a number of sets, each of which consists of a number of lines. The relationships are
  - $m = v \times k$
  - $i = j \; modulo \; v$

where

  - i = *cache set number*
  - j = *main memory block number*
  - m = *number of lines in the cache*
  - v = *number of sets*
  - k = *number of lines in each set*

- This is referred to as k-way set-associative mapping. With set-associative mapping, block B_j can be mapped into any of the lines of set j.

Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory =$2^{s+w}/2^{w}$ = 2^{s}
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2^{d}
- Size of tag = (s – d) bits

Cont..

- The memory address is now partitioned to like this:

	Tag s-d	Set d	Word w
Bits:	6	6	4

- Suppose the memory each with 2^4 = 16 words has a 16-bit address, so that 2^{16} = 64K words are in the memory's address space.
- Here, the "Tag" field identifies one of the 2^6 = 64 different memory lines in each of the 2^6 = 64 different "Set" values. Since each cache set has room for only two lines at a time, the search for a match is limited to those two lines (rather than the entire cache). Number of lines in cache $=2 \times 2^6 = 2 \times 64 = 128$
- If there's a match, we have a hit and the read or write can proceed immediately. Otherwise, there's a miss and we need to replace one of the two cache lines by this line before reading or writing into the cache. (The "Word" field again select one from among 16 addressable words inside the line.)
- In set-associative mapping, when the number of lines per set is n, the mapping is called n-way set associative. The above example is 2-way set associative.

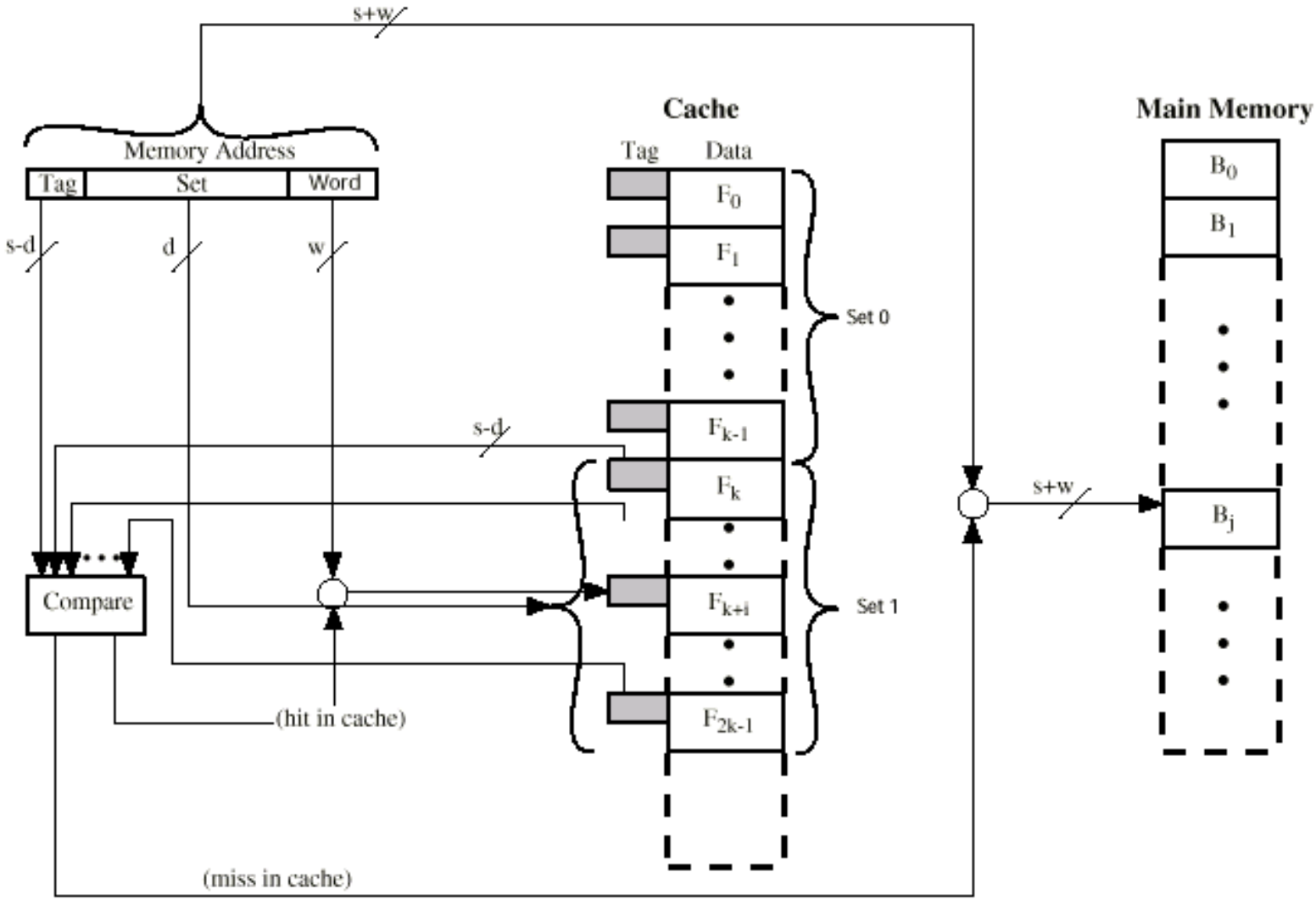
Cont..

- Example, Suppose we want to read or write a word at the address 357A, whose 16 bits (in binary) are 0011(3) 0101(5) 0111(7) 1010(A).
- Under set-associative mapping, this translates to Tag = 13(001101, 6 bits), Set = 23(010111, 6 bits), and Word = 10 (1010, 4 bits)(all in decimal).
- So we search only the two tags in cache set 23 to see if either one matches tag 13. If so, we have a hit. Otherwise, one of these two must be replaced by the memory line being addressed (line 855) before the read or write can be executed.

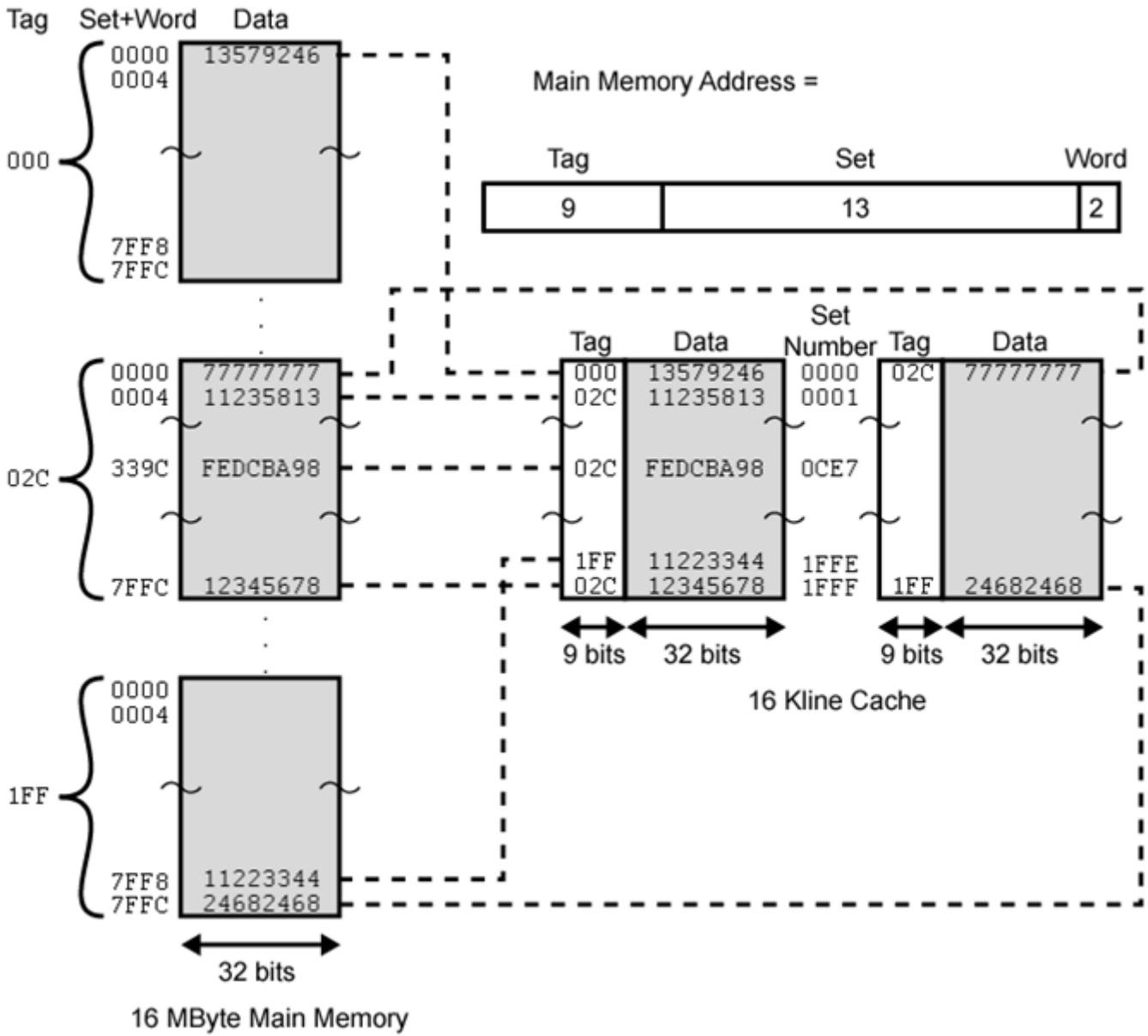
Set Associative Mapping Example

- 13 bit set number identifies a unique set of two lines within the cache.
- Block number in main memory is modulo 2^{13}
- This determines the mapping of blocks into lines. 000000, 00A000, 00B000, 00C000 … map to same cache set 0.
- Any of those blocks can be loaded into either of the two lines in the set.
- Note: no two blocks that map into the same cache set have the same tag number.

Two Way Set Associative Cache Organization



Cont..

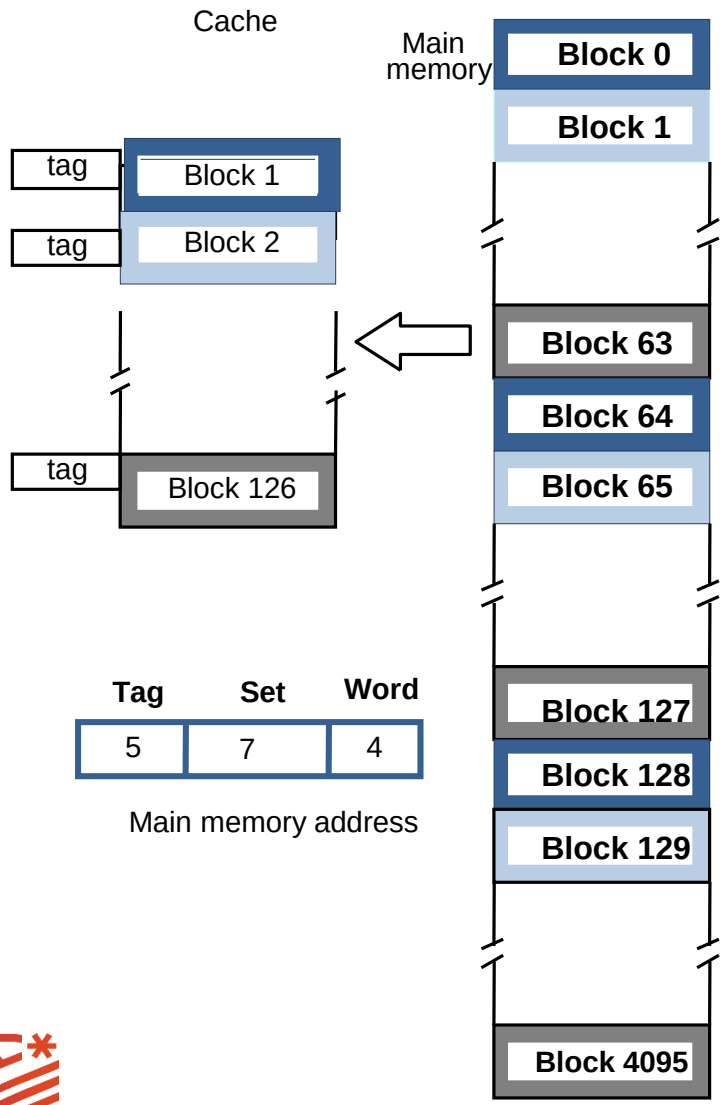


Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g
  - Address Tag Data Set number
  - 1FF 7FFC 1FF 12345678 1FFF
  - 001 7FFC 001 11223344 1FFF

Set Associative Mapping



- Blocks of cache are grouped into sets.
- Mapping function allows a block of the main memory to reside in any block of a specific set.
- Divide the cache into 64 sets, with two blocks per set.
- Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.
- Memory address is divided into three fields:
  - 6 bit field determines the set number.
  - High order 6 bit fields are compared to the tag fields of the two blocks in a set.
- Set-associative mapping combination of direct and associative mapping.
- Number of blocks per set is a design parameter.
  - One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping)
  - Other extreme is to have one block per set, is the same as direct mapping.

Example 1

- A computer has a 256 KByte, 4-way set associative, write back data cache with block size of 32 Bytes. The processor sends 32 bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. The number of bits in the tag field of an address is:

- Sol:-
- Number of blocks = Cache-Size/Block-Size = 256 KB / 32 Bytes = 2^{13}
- Number of Sets = 2^{13} / 4 (given, 4-way set associative)= 2^{11}
- Tag + Set offset + word = 32(given)
  Tag + 11 + 5 = 32
  Tag = 16

Example 2

- An 8KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following.
  - 1 Valid bit
  - 1 Modified bit
  - As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache?
- Sol:- Cache size = 8 KB = 8 × 1024 bytes = 2^{13}
- Block size = 32 bytes= = 2^5 , w=5 • MM address = 32 bit =s
- Number of cache lines = Cache size / Block size = $2^{13}/ 2^5 = 2^8$ r=8,s-r=19

	Tag s-r	Line or Slot r	Word w
Bits:	19	8	5

- total bits required to store meta-data of 1 line = 1 + 1 + 19 = 21 bits
- total memory required = 21 × 2^8 = 21 × 256 = 5376 bits

Summary

- Cache may be located on CPU chip or module
- Write-through means writing to both cache and main memory when a miss occurs (to avoid inconsistent or untrue memories)
- Direct Mapped: Each block has only one place that it can appear in the cache.
- Fully associative: Each block can be placed anywhere in the cache.
- Set associative: Each block can be placed in a restricted set of places in the cache.
- Write through: The information is written to both the block in the cache and to the block in the lower-level memory
- Write back: The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced