Computer Architecture

Lecture-07

Md. Biplob Hosen

Lecturer, IIT-JU.

Email: biplob.hosen@juniv.edu

Reference

- “Computer Organization and Architecture” by William Stallings; 8th Edition (Chapter-04).
  - Any later edition is fine.

Content

- Elements of Cache Design
- Cache Organization

Elements of Cache Design - Details from Book

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	Write once
Mapping Function	**Line Size**
Direct	**Number of caches**
Associative	Single or two level
Set Associative	Unified or split
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Mapping Function

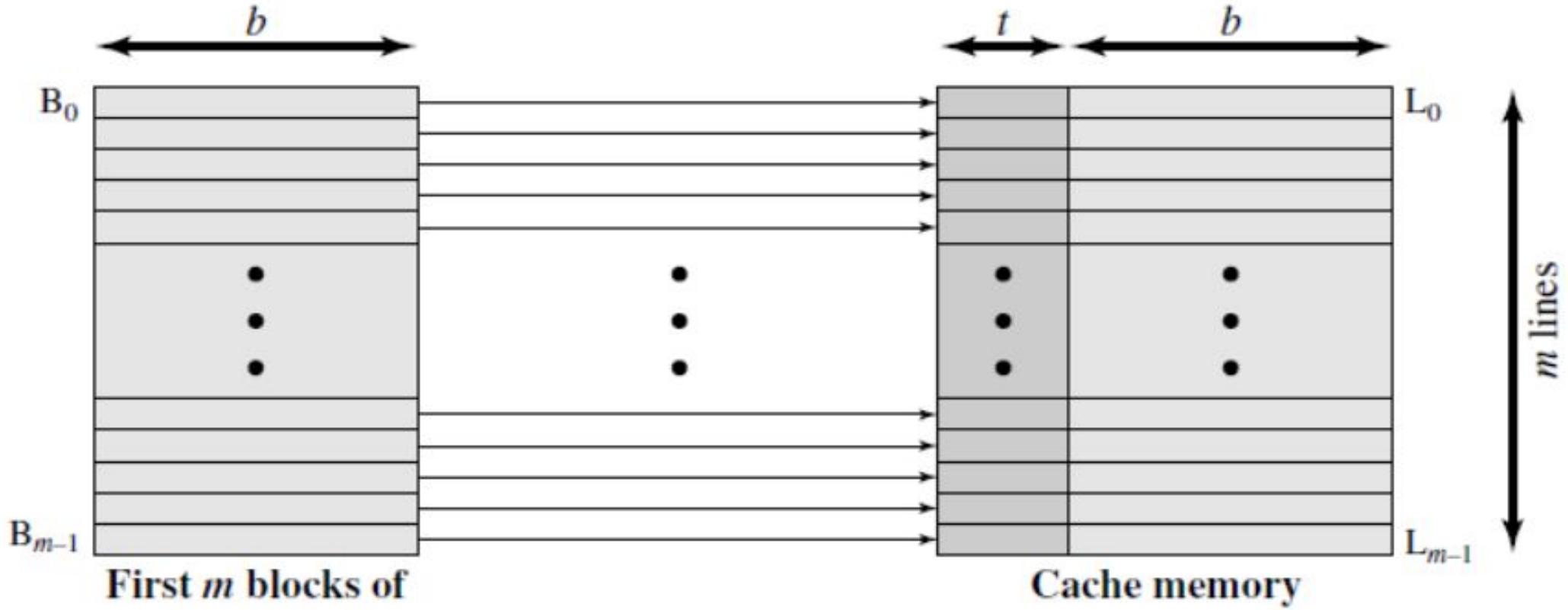
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- Further, a means is needed for determining which main memory block currently occupies a cache line.
- The choice of the mapping function dictates how the cache is organized.
- Three techniques can be used:
  - Direct
  - Associative
  - Set associative.
- We examine each of these in turn.
- In each case, we look at the general structure and then a specific example.

Direct Mapping

- The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line.
- The mapping is expressed as: i = j modulo m

  where, i cache line number, j main memory block number, m number of lines in the cache.
- Figure shows the mapping for the first blocks of main memory.
- Each block of main memory maps into one unique line of the cache.
- The next blocks of main memory map into the cache in the same fashion; that is, block Bm of main memory maps into line L0 of cache, block Bm+1 maps into line L1, and so on.

Continue…

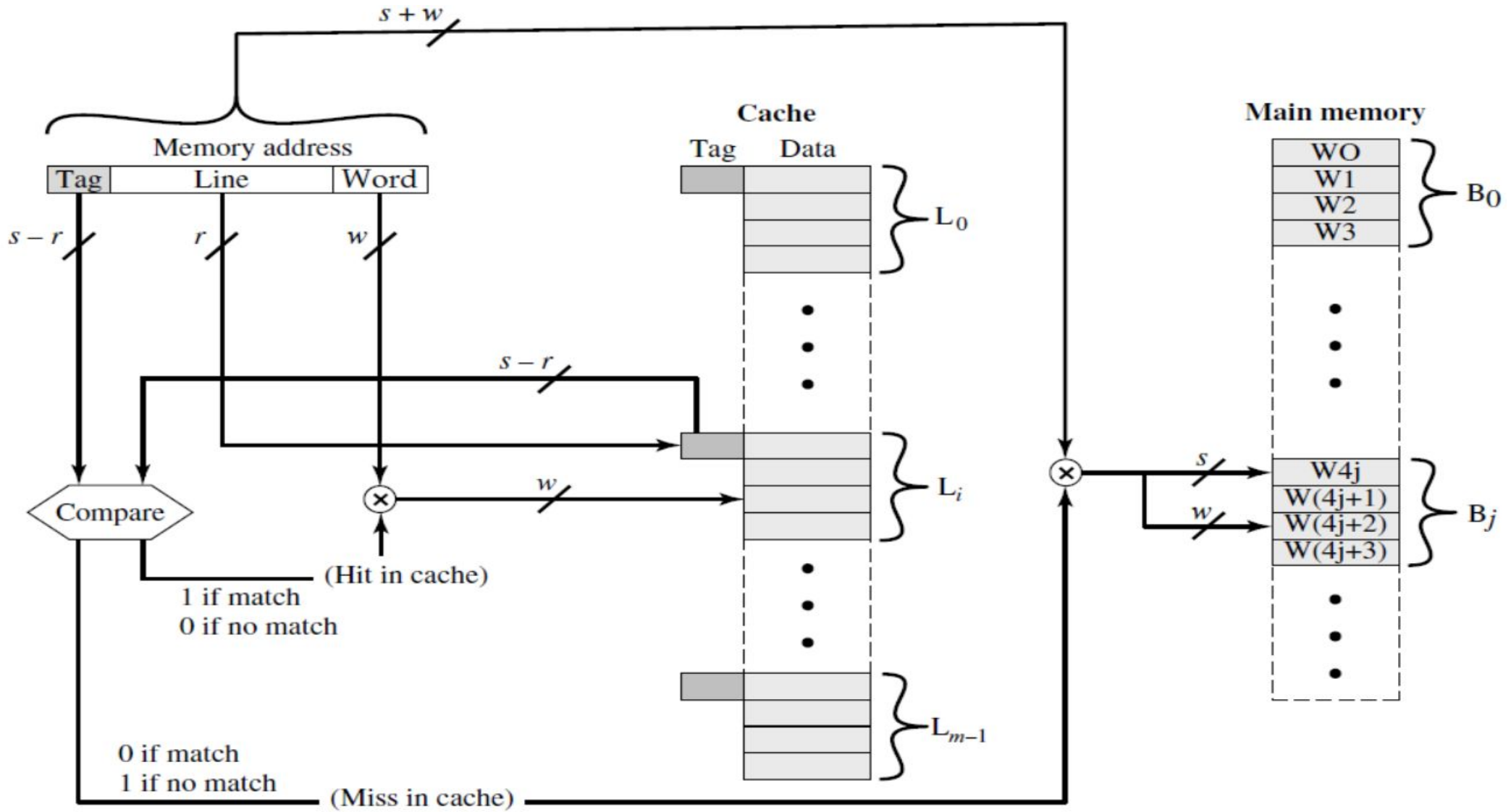


(a) Direct mapping

Continue…

- The mapping function is easily implemented using the main memory address.
- Figure illustrates the general mechanism.
- For purposes of cache access, each main memory address can be viewed as consisting of three fields.
- The least significant w bits identify a unique word or byte within a block of main memory; in most contemporary machines, the address is at the byte level.
- The remaining s bits specify one of the 2s blocks of main memory.
- The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits.
- This latter field identifies one of the m 2r lines of the cache.
- To summarize,
  - 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 $= \frac{2^{s+w}}{2^{w}} = 2^{s}$
  - Number of lines in cache $= m = 2^{r}$
  - Size of cache $= 2^{r+w}$ words or bytes
  - Size of tag $= (s - r)$ bits

Continue…


s + w
Memory address
Tag
Line
Word
s − r
r
w
Cache
Tag
Data
L0
Li
Lm−1
s − r
w
Compare
(Hit in cache)
1 if match
0 if no match
0 if match
1 if no match
(Miss in cache)
Main memory
WO
W1
W2
W3
B0
W4j
W(4j+1)
W(4j+2)
W(4j+3)
Bj
s
w

Continue…

- 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 assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

- Thus, the use of a portion of the address as a line number provides a unique mapping of each block of main memory into the cache.
- When a block is actually read into its assigned line, it is necessary to tag the data to distinguish it from other blocks that can fit into that line.
- The most significant s-r bits serve this purpose.

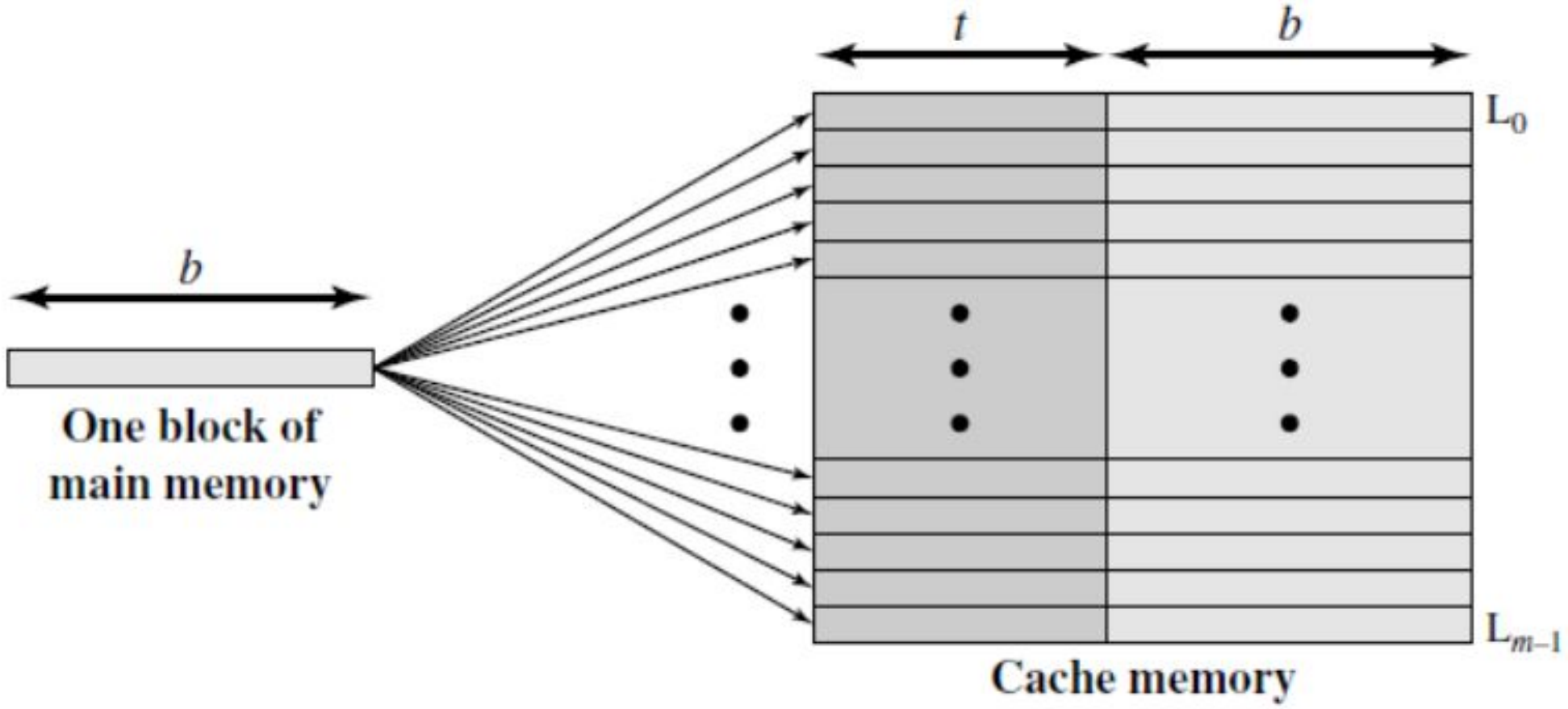
Continue…

- The direct mapping technique is simple and inexpensive to implement.
- Its main disadvantage is that there is a fixed cache location for any given block.
- Thus, if a program happens to reference 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).*
- One approach to lower the miss penalty is to remember what was discarded in case it is needed again.
- Since the discarded data has already been fetched, it can be used again at a small cost.
- Such recycling is possible using a victim cache.
- Victim cache was originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time.
- Victim cache is a fully associative cache, whose size is typically 4 to 16 cache lines, residing between a direct mapped L1 cache and the next level of memory.

Associative Mapping

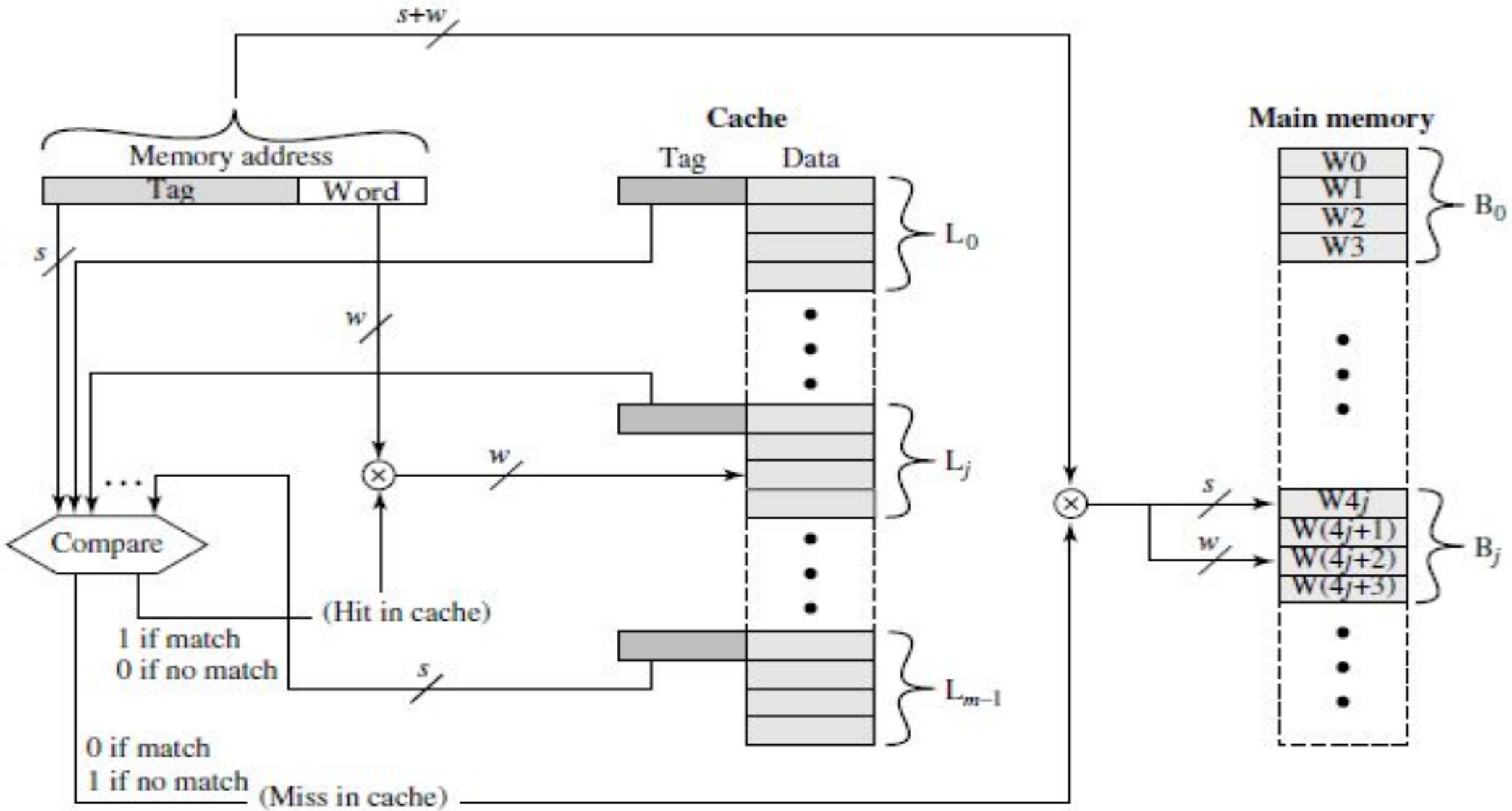
- Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache.
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field. The Tag field uniquely identifies a block of main memory.
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match.
- Note that no field in the address corresponds to the line number, so that the number of lines in the cache is not determined by the address format.
- To summarize,
  - 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 $= \frac{2^{s+w}}{2^{w}} = 2^{s}$
  - Number of lines in cache $=$ undetermined
  - Size of tag $= s$ bits

Associative Mapping



(b) Associative mapping

Fully Associative Mapping



Set-Associative Mapping

- Set-associative mapping is a compromise that exhibits
- the strengths of both the direct and associative approaches while reducing their disadvantages.
- In this case, the cache consists of a number sets, each of which consists of a number of lines.
- The relationships are:

$$m = v \times k$$
$$i = j \text{ modulo } v$$

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

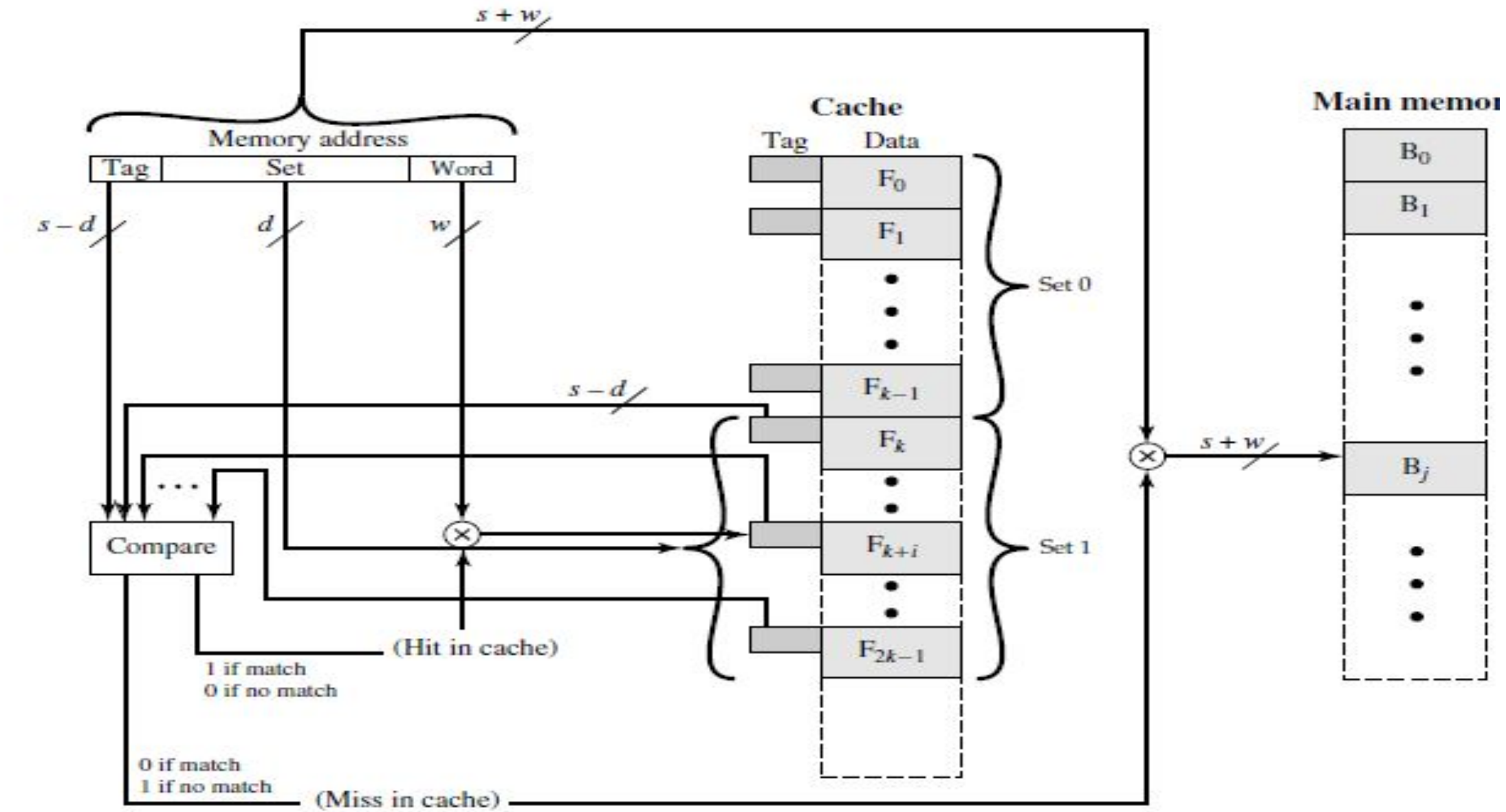
Continue…

- This is referred to as k-way set-associative mapping. With set-associative mapping, block Bj can be mapped into any of the lines of set j.
- Figure illustrates this mapping for the first blocks of main memory.
- As with associative mapping, each word maps into multiple cache lines. For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block B0 maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as associative caches.
- It is also possible to implement the set-associative cache as k direct mapping caches, as shown in Figure.
- Each direct-mapped cache is referred to as a *way, consisting of lines.*
- *The first lines of main memory* are direct mapped into the lines of each way; the next group of lines of main memory are similarly mapped, and so on.
- The direct-mapped implementation is typically used for small degrees of associativity (small values of k) while the associative-mapped implementation is typically used for higher degrees of associativity.

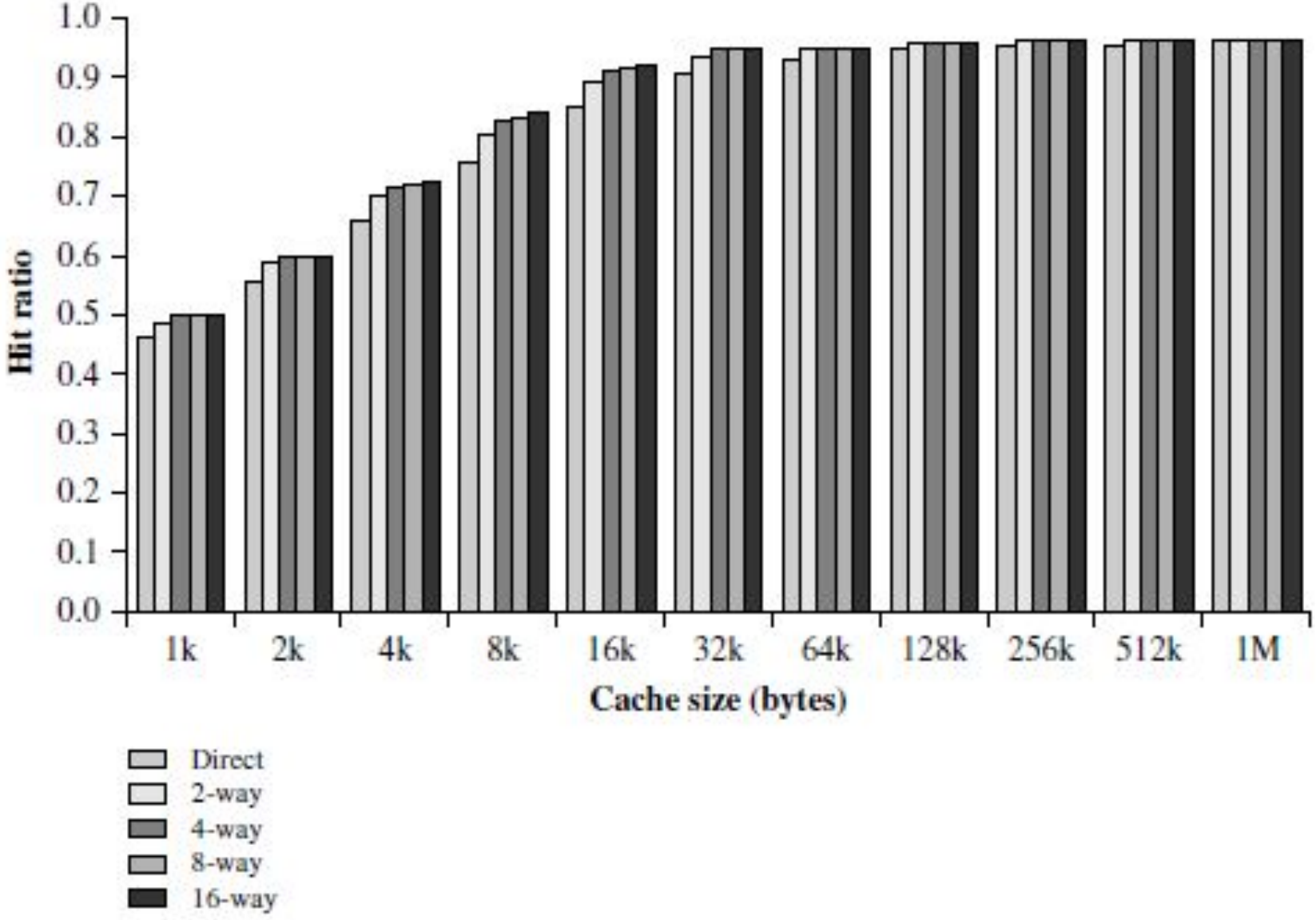
Continue…

- For set-associative mapping, the cache control logic interprets a memory address as three fields: Tag, Set, and Word.
- The d set bits specify one of 2d sets.
- The s bits of the Tag and Set fields specify one of the 2s blocks of main memory.
- Figure illustrates the cache control logic.
- With fully associative mapping, the tag in a memory address is quite large and must be compared to the tag of every line in the cache.
- With k-way set-associative mapping, the tag in a memory address is much smaller and is only compared to the k tags within a single set.
- To summarize,
  - 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 $= \frac{2^{s+w}}{2^{w}} = 2^{s}$
  - Number of lines in set $= k$
  - Number of sets $= v = 2^{d}$
  - Number of lines in cache $= m = kv = k \times 2^{d}$
  - Size of cache $= k \times 2^{d+w}$ words or bytes
  - Size of tag $= (s - d)$ bits

Set Associative Mapping



Varying Associativity over Cache Size



Continue…

- It significantly improves the hit ratio over direct mapping. Four-way set associative (n = m/4, k = 4) makes a modest additional improvement for a relatively small additional cost.
- Further increases in the number of lines per set have little effect.
- Figure shows the results of one simulation study of set-associative cache performance as a function of cache size.
- The difference in performance between direct and two-way set associative is significant up to at least a cache size of 64 kB.
- Note also that the difference between two-way and four-way at 4 kB is much less than the difference in going from for 4 kB to 8 kB in cache size.
- The complexity of the cache increases in proportion to the associativity, and in this case would not be justifiable against increasing cache size to 8 or even 16 Kbytes.
- A final point to note is that beyond about 32 kB, increase in cache size brings no significant increase in performance.
- The results of Figure are based on simulating the execution of a GCC compiler.
- Different applications may yield different results.
- For example, [CANT01] reports on the results for cache performance using many of the CPU2000 SPEC benchmarks.
- The results of [CANT01] in comparing hit ratio to cache size follow the same pattern as Figure 4.16, but the specific values are somewhat different.

Write Policy

- When a block that is resident in the cache is to be replaced, there are two cases to consider.
- If the old block in the cache has not been altered, then it may be overwritten with a new block without first writing out the old block.
- If at least one write operation has been performed on a word in that line of the cache, then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block.
- A variety of write policies, with performance and economic trade-offs, is possible.
- There are two problems to contend with.
- First, more than one device may have access to main memory. For example, an I/O module may be able to read-write directly to memory. If a word has been altered only in the cache, then the corresponding memory word is invalid. Further, if the I/O device has altered main memory, then the cache word is invalid.
- A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache. Then, if a word is altered in one cache, it could conceivably invalidate a word in other caches.

Continue…

- The simplest technique is called **write through.** Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.
- Any other processor–cache module can monitor traffic to main memory to maintain consistency within its own cache.
- The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck.
- An alternative technique, known as **write back,** minimizes memory writes.
- With write back, updates are made only in the cache.
- When an update occurs, a dirty bit, or use bit, associated with the line is set.
- Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set. The problem with write back is that portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache.
- This makes for complex circuitry and a potential bottleneck.

Continue…

- In a bus organization in which more than one device (typically a processor) has a cache and main memory is shared, a new problem is introduced.
- If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word).
- Even if a write-through policy is used, the other caches may contain invalid data.
- A system that prevents this problem is said to maintain cache coherency.
- Possible approaches to cache coherency include the following:
- **Bus watching with write through:** Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. This strategy depends on the use of a write-through policy by all cache controllers.
- **Hardware transparency:** Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches. Thus, if one processor modifies a word in its cache, this update is written to main memory. In addition, any matching words in other caches are similarly updated.
- **Non-cacheable memory:** Only a portion of main memory is shared by more than one processor, and this is designated as non-cacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache. The non-cacheable memory can be identified using chip-select logic or high-address bits.

Line Size

- Another design element is the line size.
- When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words are retrieved.
- As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced in the near future.
- As the block size increases, more useful data are brought into the cache.
- The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced.
- Two specific effects come into play:
- Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
- As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future.
- The relationship between block size and hit ratio is complex, depending on the locality characteristics of a particular program, and no definitive optimum value has been found.
- A size of from 8 to 64 bytes seems reasonably close to optimum.

Number of Caches

- When caches were originally introduced, the typical system had a single cache.
- More recently, the use of multiple caches has become the norm. Two aspects of this design issue concern:
  - The number of levels of caches
  - The use of unified vs. split caches
- Level of Cache:
  - Single Level
  - Multi-Level:
    - On-Chip Cache (L1): on-processor.
    - Off-Chip Cache (L2): External

Continue…

Unified vs. Split Cache:

- When the on-chip cache first made an appearance, many of the designs consisted of a single cache used to store references to both data and instructions.
- More recently, it has become common to split the cache into two: one dedicated to instructions and one dedicated to data.
- These two caches both exist at the same level, typically as two L1 caches.
- When the processor attempts to fetch an instruction from main memory, it first consults the instruction L1 cache, and when the processor attempts to fetch data from main memory, it first consults the data L1 cache.

There are two potential advantages of a unified cache:

- For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. That is, if an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches, the opposite will occur.
- Only one cache needs to be designed and implemented.

Continue…

- Despite these advantages, the trend is toward split caches, particularly for superscalar machines such as the Pentium and PowerPC, which emphasize parallel instruction execution and the pre-fetching of predicted future instructions.
- The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit.
- This is important in any design that relies on the pipelining of instructions.
- Typically, the processor will fetch instructions ahead of time and fill a buffer, or pipeline, with instructions to be executed.
- Suppose now that we have a unified instruction/data cache.
- When the execution unit performs a memory access to load and store data, the request is submitted to the unified cache.
- If, at the same time, the instruction pre-fetcher issues a read request to the cache for an instruction, that request will be temporarily blocked so that the cache can service the execution unit first, enabling it to complete the currently executing instruction.
- This cache contention can degrade performance by interfering with efficient use of the instruction pipeline.
- The split cache structure overcomes this difficulty.

Thank You