电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC4019: Real-Time Computer Systems and Architecture

Lecture 9

Memory Technologies (Part-2)

Elements of Cache Design (1 of 3)

- **Cache Addresses**
  - Logical
  - Physical
- **Cache Size**
- **Mapping Function**
  - Direct
  - Associative
  - Set Associative

Elements of Cache Design (2 of 3)

- **Replacement Algorithm**
  - Least recently used (LRU)
  - First in first out (FIFO)
  - Least frequently used (LFU)
  - Random

Elements of Cache Design (3 of 3)

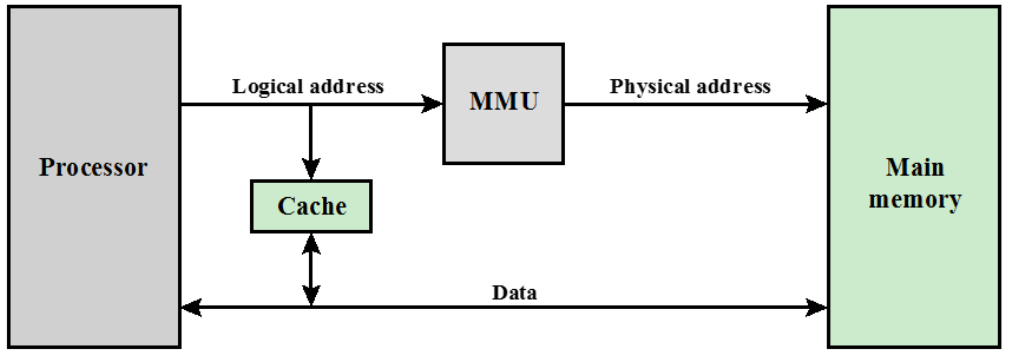
- **Write Policy**
  - Write through
  - Write back
- **Line Size**
- **Number of caches**
  - Single or two level
  - Unified or split

Cache Addresses

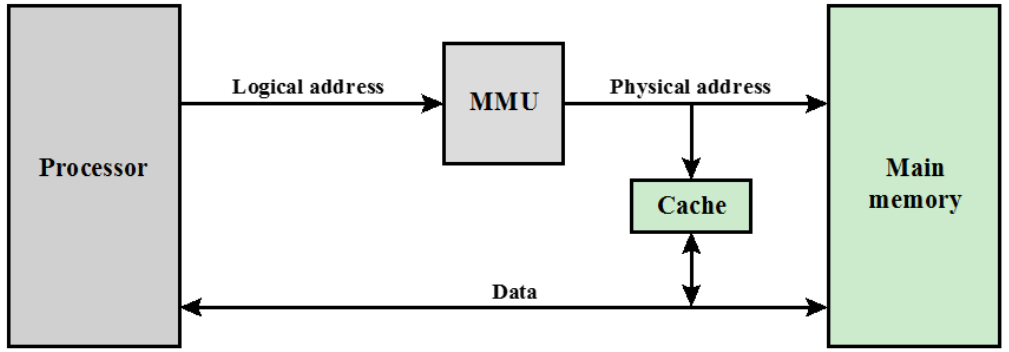
- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory

Logical and Physical Caches

- A **logical cache,** also known as a **virtual cache,** stores data using **virtual addresses**
- The processor accesses the cache directly, without going through the MMU
- A physical cache stores data using main memory **physical addresses**
- One obvious advantage of the logical cache is that cache access speed is faster than for a physical cache, because the cache can respond before the MMU performs an address translation



(a) Logical Cache



(b) Physical Cache

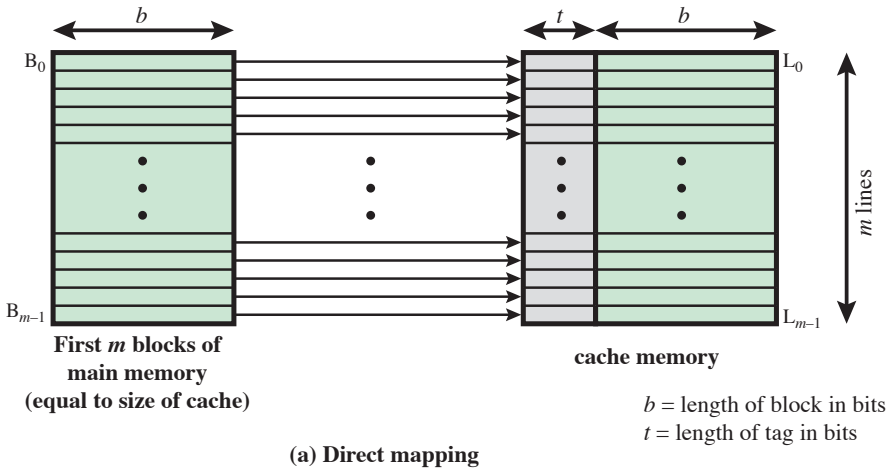
Mapping Function (1 of 2)

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines. Three techniques can be used:
- Direct
  - The simplest technique
  - Maps each block of main memory into only one possible cache line

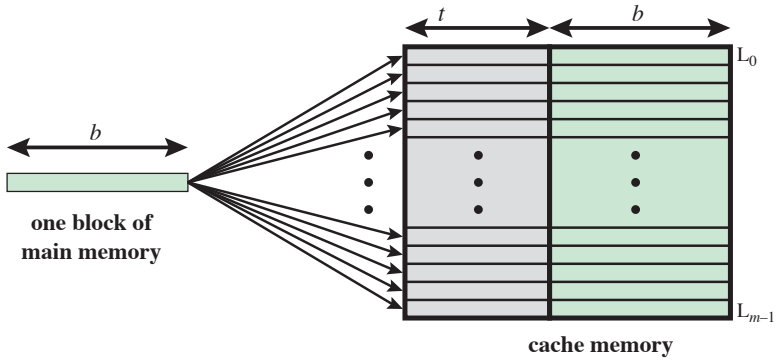
Mapping Function (2 of 2)

- **Associative**
  - Permits each main memory block to be loaded into any line of the cache
  - The cache control logic interprets a memory address simply as a Tag and a Word field
  - To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match
- **Set Associative**
  - A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

Mapping From Main Memory to Cache: Direct and Associative

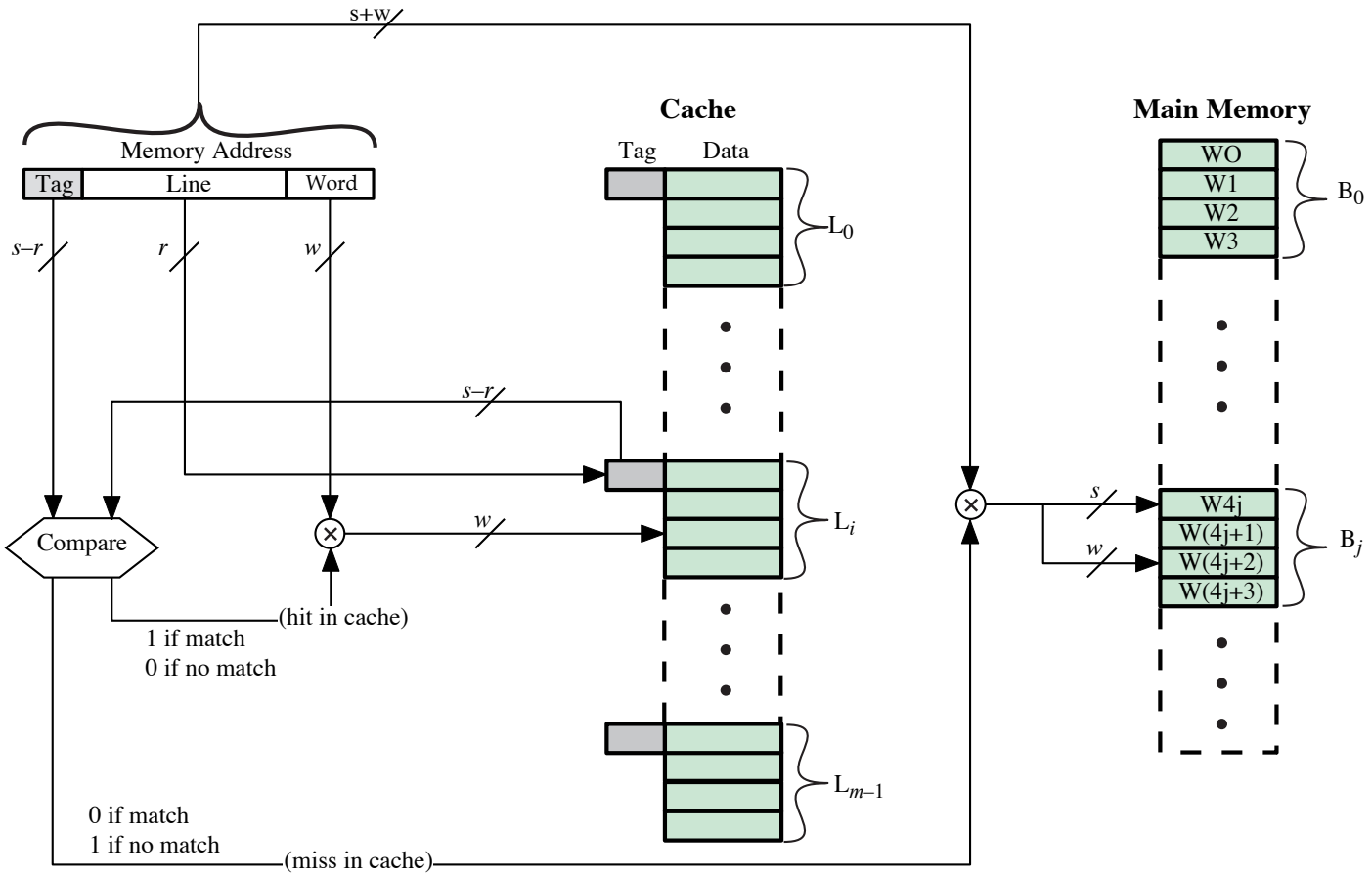


(a) Direct mapping

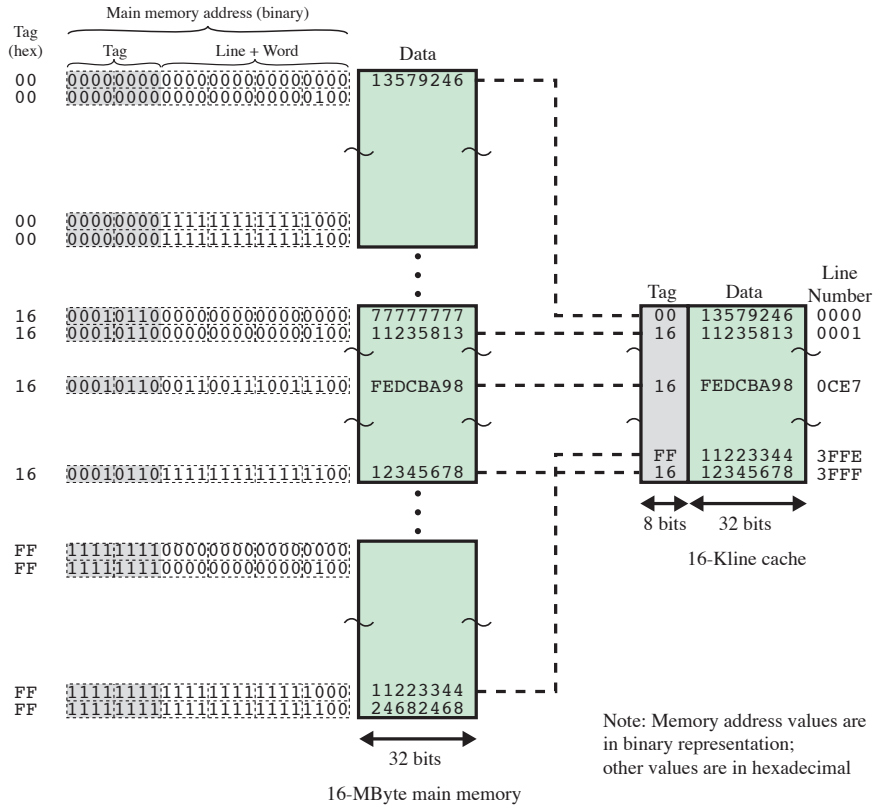


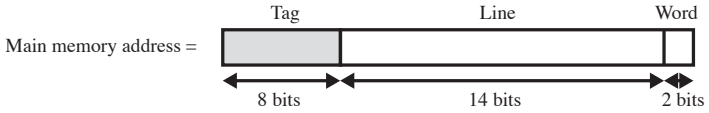
(b) Associative mapping

Direct-Mapping Cache Organization



Direct Mapping Example





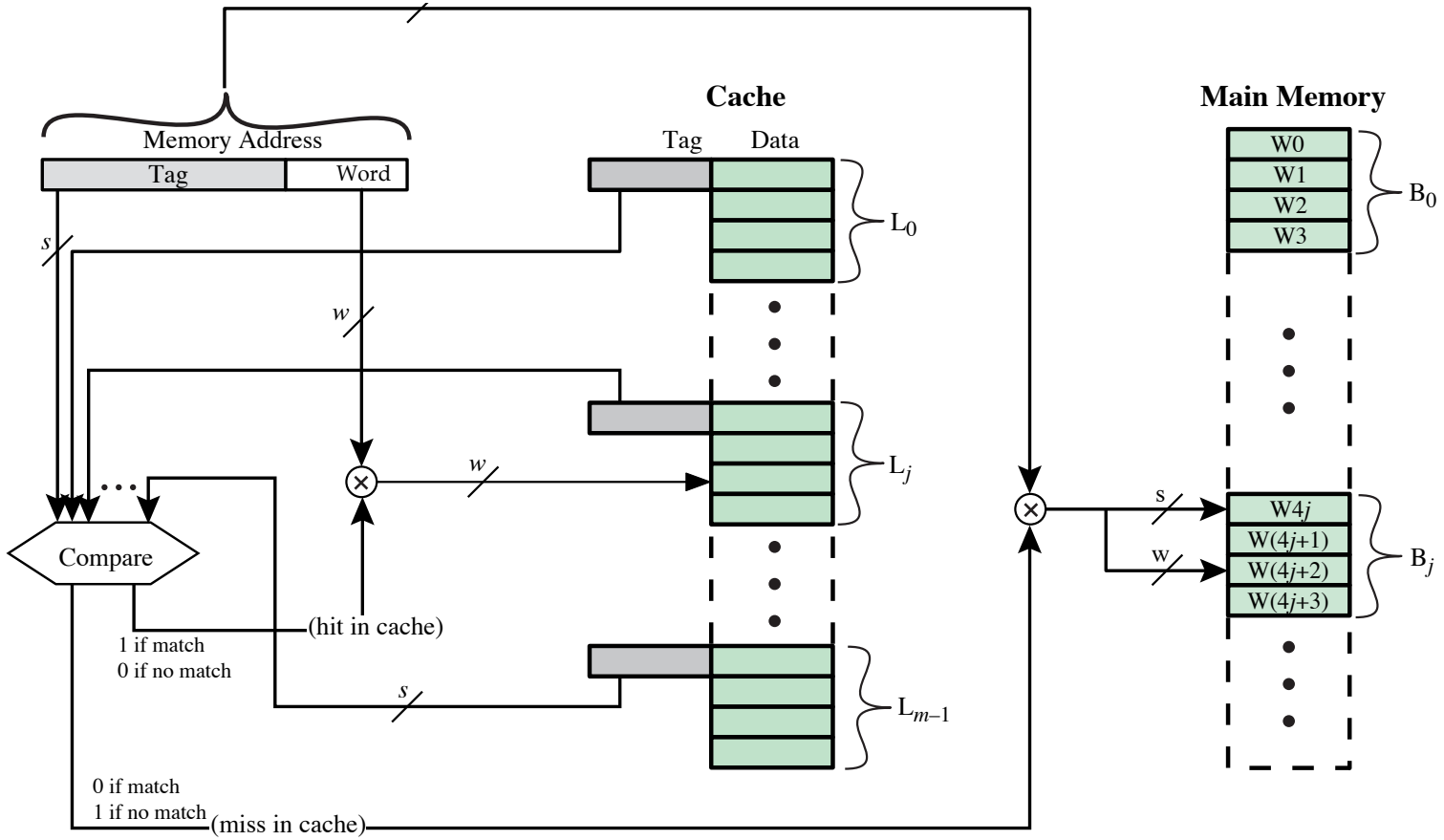
Direct Mapping Summary

- 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 = $2^{s+w} / 2^{w} = 2^{s}$
- Number of lines in cache = $m = 2^{r}$
- Size of tag = (s − r) bits

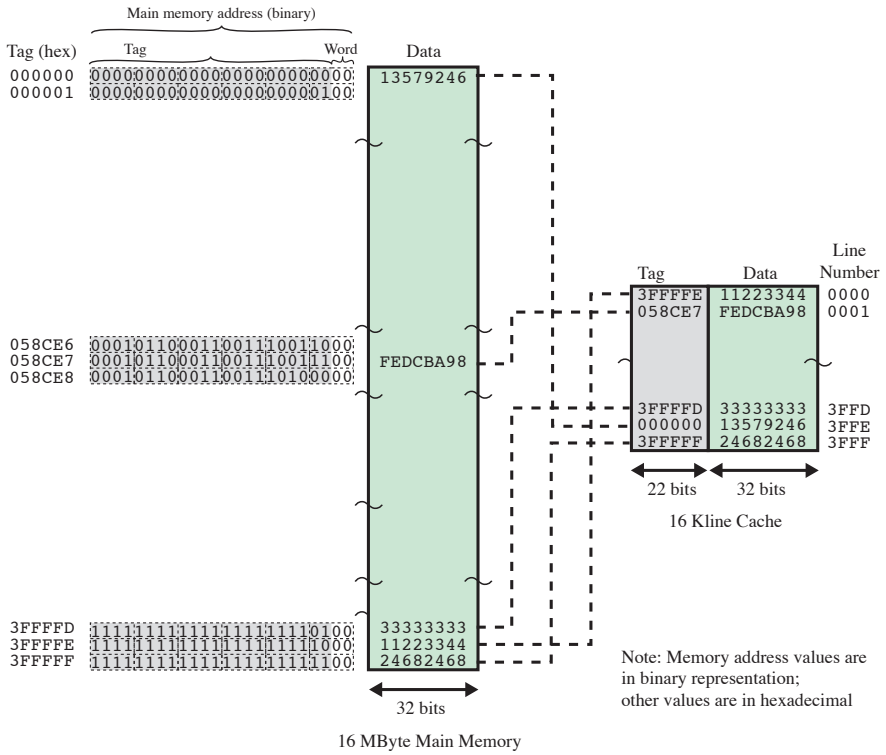
Victim Cache

- Originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time
  - Fully associative cache
  - Typical size is 4 to 16 cache lines
  - Residing between direct mapped L1 cache and the next level of memory

Fully Associated Cache Organization



Associative Mapping Example



Associative Mapping Summary

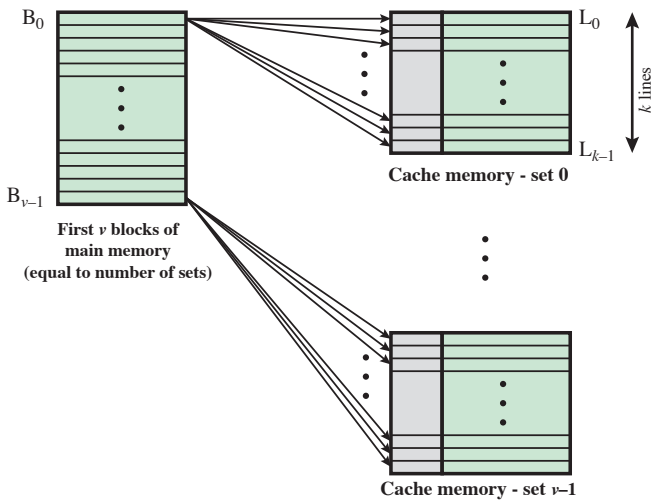
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w} / 2^{w} = 2^{s}$
- Number of lines in cache = Undetermined
- Size of tag = s bits

Set Associative Mapping

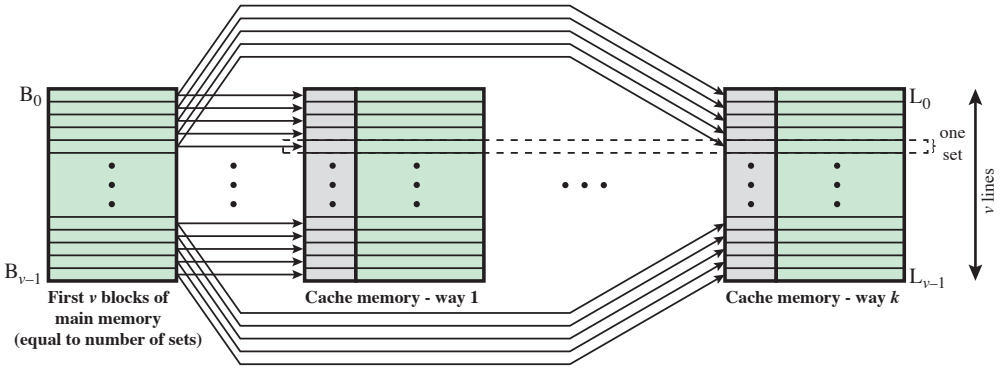
- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Mapping From Main Memory to Cache: k-way Set Associative

- For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block B_0 maps into set 0, and so on
- Thus, the set-associative cache can be physically implemented as n associative caches
- It is also possible to implement the set-associative cache as *k direct mapping caches*
- *Each direct-mapped cache is* referred to as a *way, consisting of v lines*

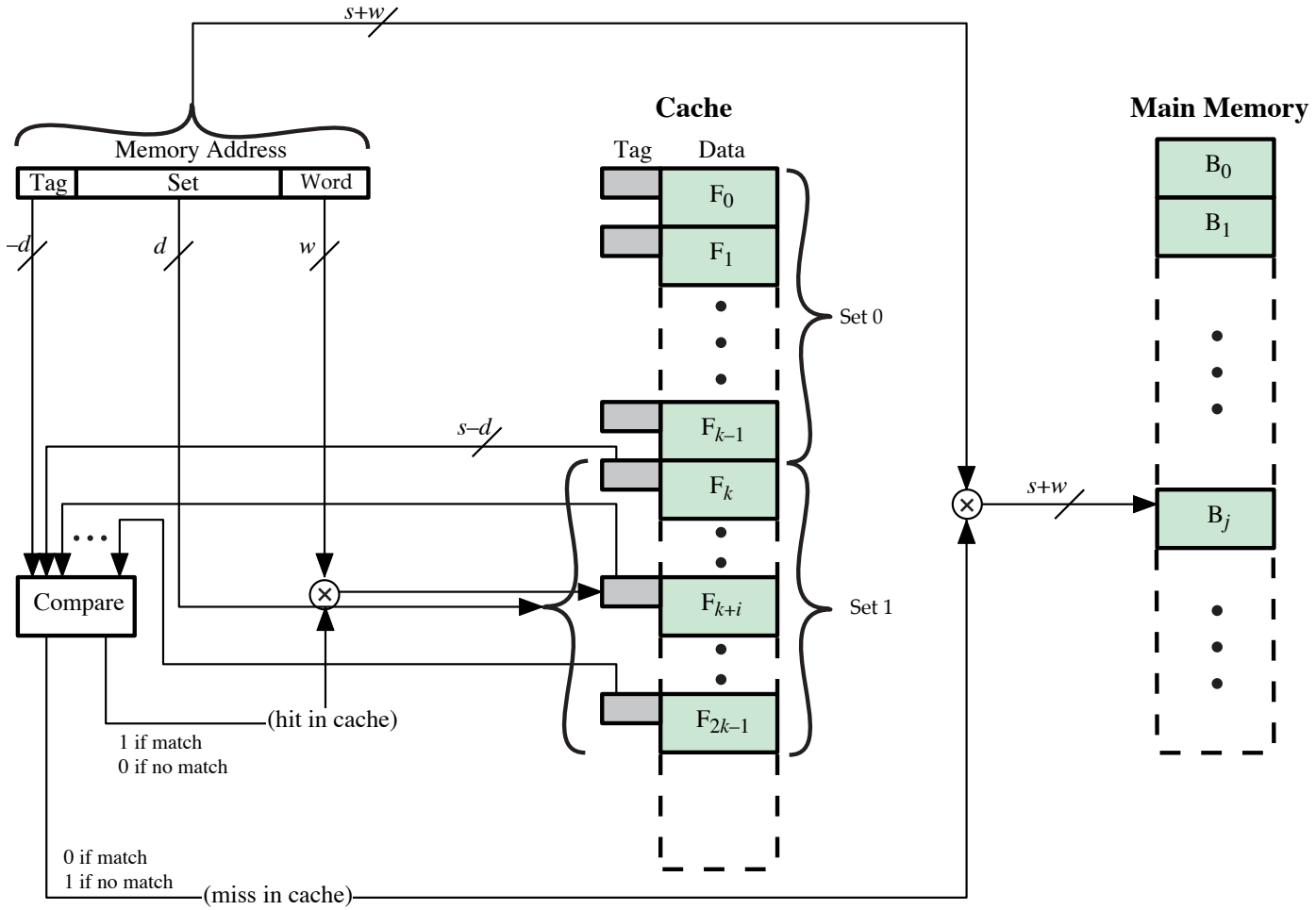


(a) *v* associative-mapped caches



(b) *k* direct-mapped caches

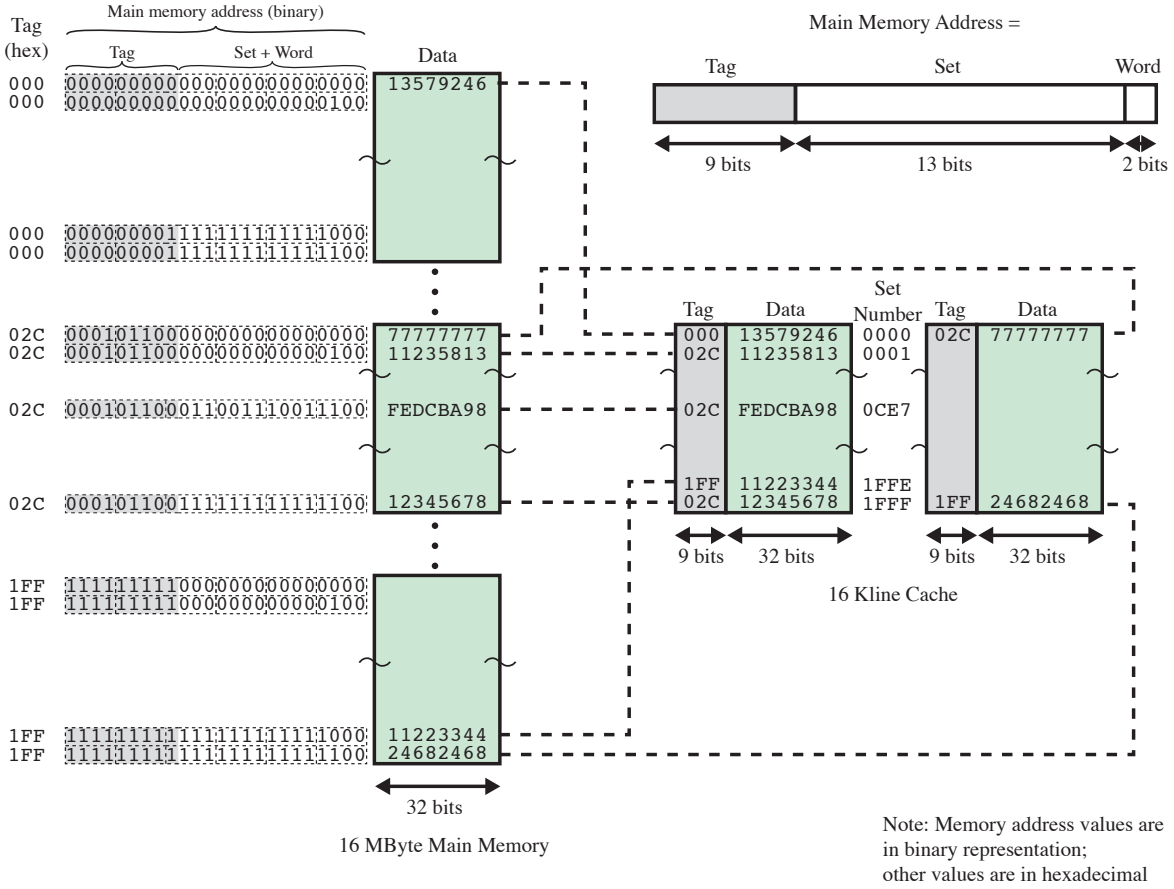
Fk-Way Set Associative Cache Organization



Set Associative Mapping Summary

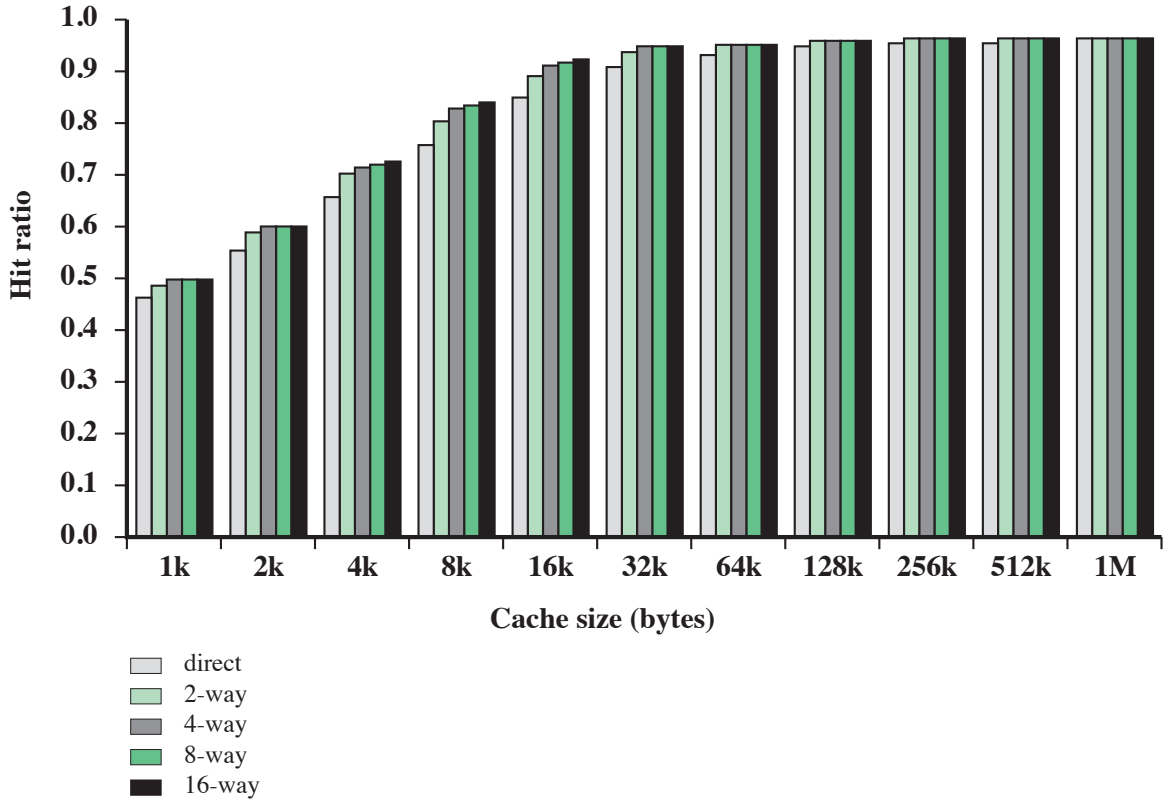
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w} / 2^{w} = 2^{s}$
- Number of lines in set = k
- Number of sets = $v = 2^{d}$
- Number of lines in cache = $m = kv = k * 2^{d}$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

Two-Way Set Associative Mapping Example



Varying Associativity over 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



Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

The Most Common Replacement Algorithms Are:

- **Least recently used (LRU)**
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- **First-in-first-out (FIFO)**
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- **Least frequently used (LFU)**
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line

Write Policy (1 of 2)

- 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

Write Policy (2 of 2)

- There are two problems to contend with:
  - More than one device may have access to main memory
  - A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Write Through and Write Back

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - 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

Line Size (1 of 2)

- 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 the hit ratio will at first increase because of the principle of locality
- As the block size increases more useful data are brought into the cache
- The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

Line Size (2 of 2)

- Two specific effects come into play:
  - Larger blocks reduce the number of blocks that fit into a cache
  - As a block becomes larger each additional word is farther from the requested word

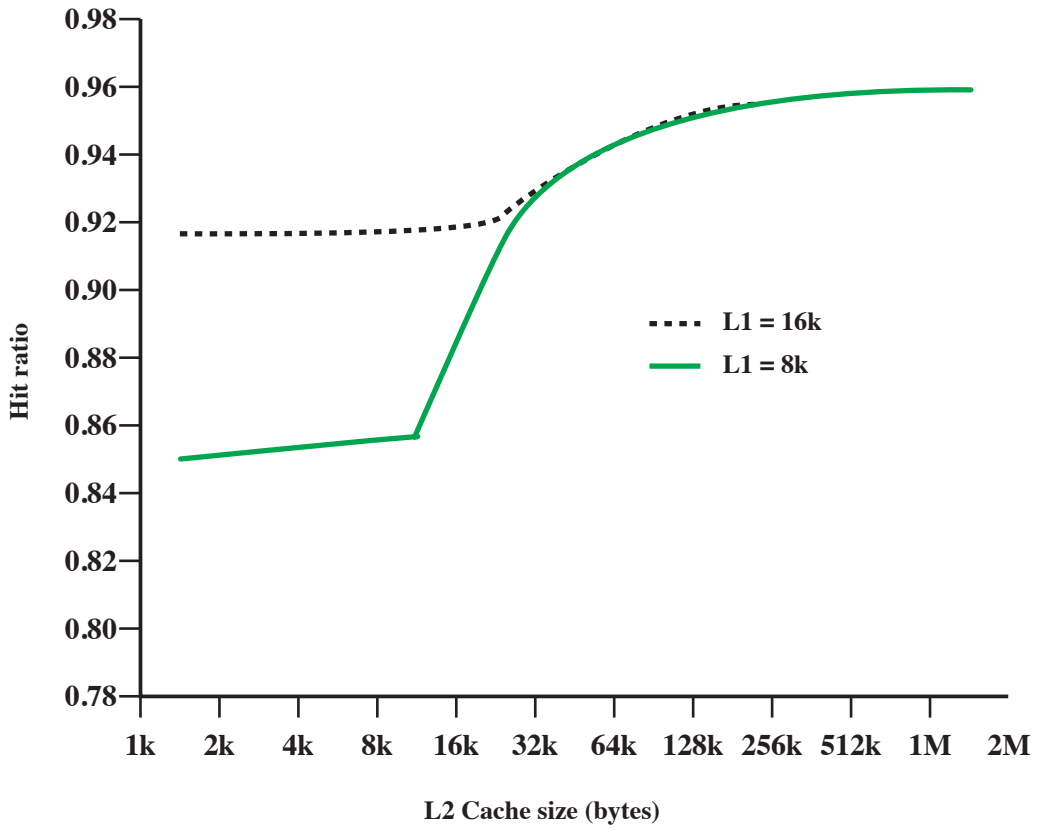
Multilevel Caches (1 of 2)

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers

Multilevel Caches (2 of 2)

- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1



Unified Versus Split Caches (1 of 2)

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented

Unified Versus Split Caches (2 of 2)

- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

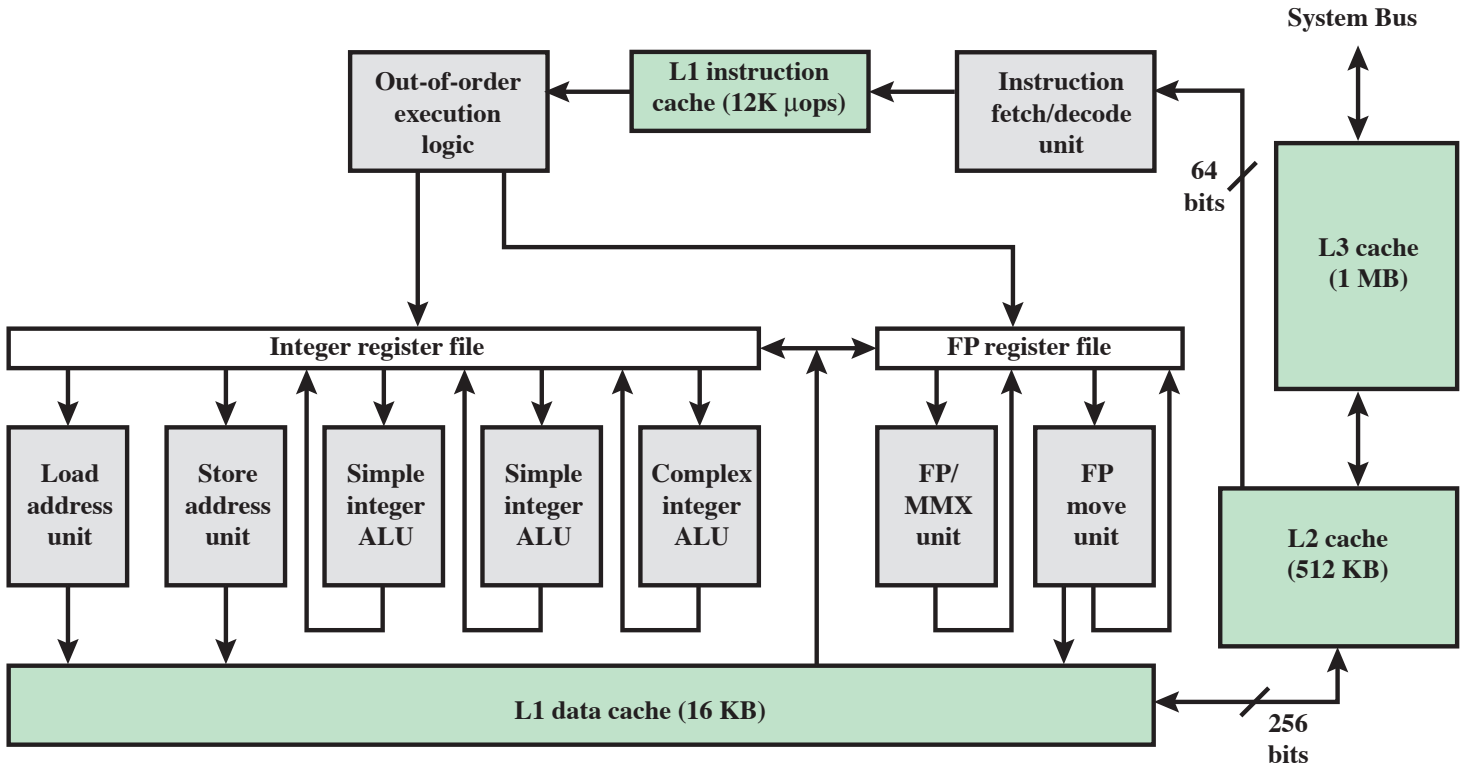
Intel Cache Evolution (1 of 2)

Problem	**Solution**	**Processor on which Feature First Appears**
External memory slower than the system bus.	Add external cache using faster memory technology	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on- chip, operating at the speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches	Pentium

Intel Cache Evolution (2 of 2)

Problem	**Solution**	**Processor on which Feature First Appears**
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium 2
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache	Pentium 3
	Move L3 cache on-chip.	Pentium 4

Pentium 4 Block Diagram



Pentium 4 Cache Operating Modes

Control Bits		**Operating Modes**		
CD	NW	Cache Fills	Write Throughs	Invalidates
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination