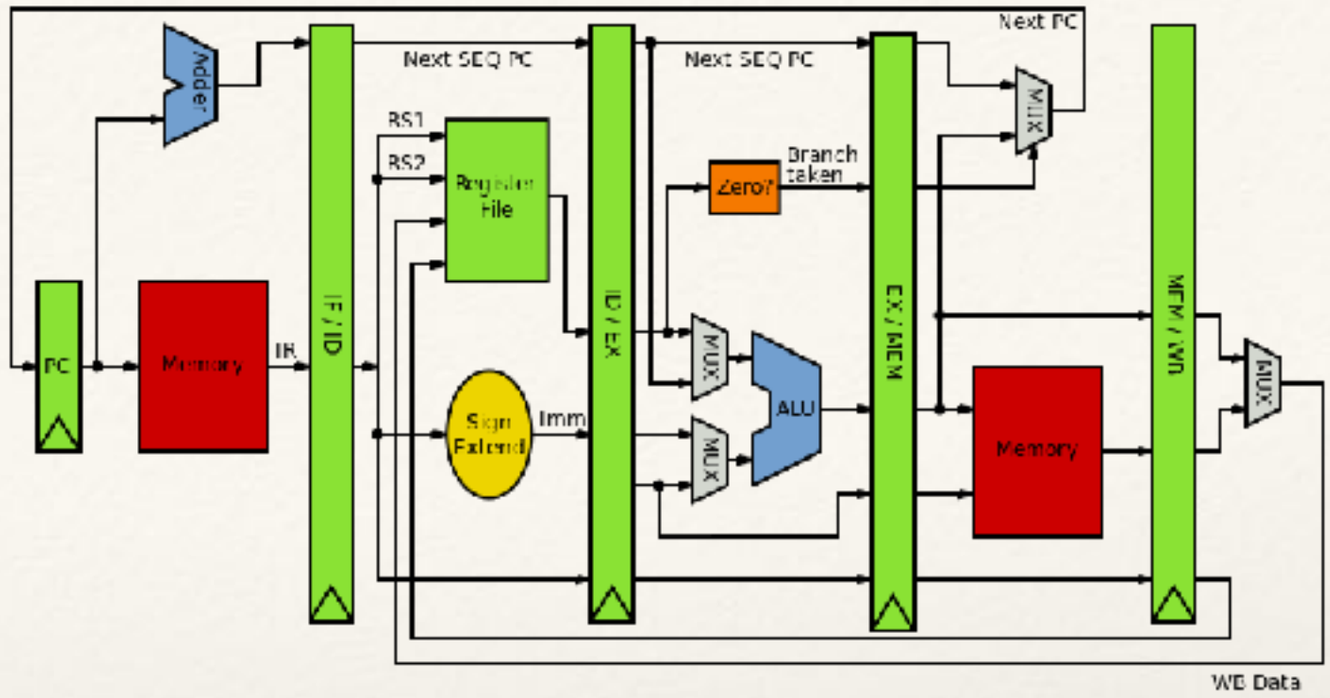


COMP2120 Computer Organization

Cache Memory

Chapter 4

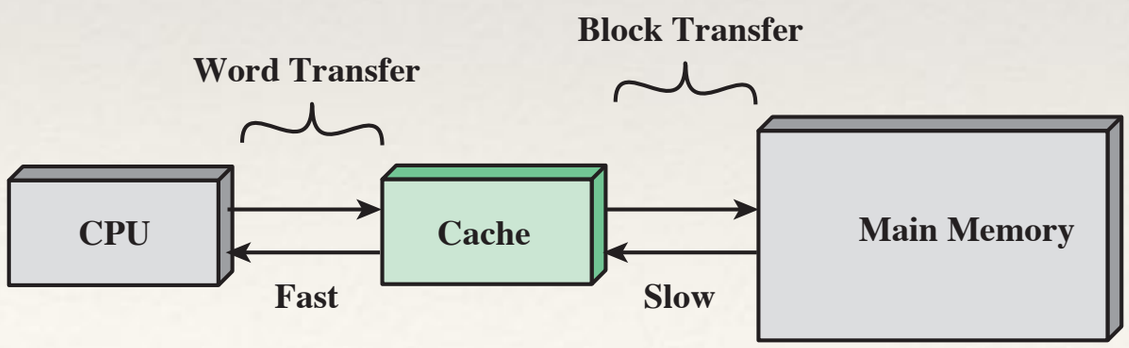
Dr. Ronald H.Y. Chung

THE UNIVERSITY OF HONG KONG

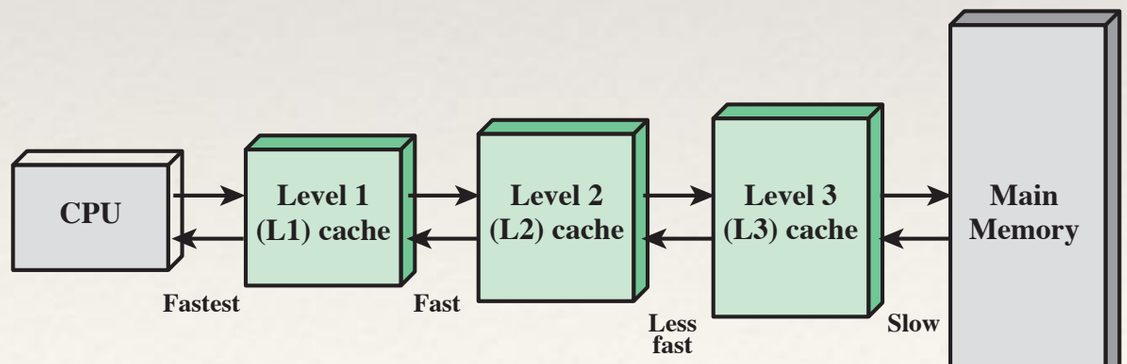
DEPARTMENT OF COMPUTER SCIENCE

Cache Memory

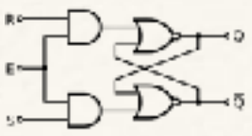
- Cache is introduced to bridge the speed gap between CPU and Primary Memory
  - Cycle time of CPU: ~1ns (1Ghz)
  - Cycle time of Primary Memory: ~50-60ns (memory access time)
- Static RAM is usually used because it is faster than Dynamic Ram
- Cache Memory is transparent to (hidden from) the program (i.e. the user)
- The cache contains a copy of portions of main memory.
  - When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor.
  - If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor.
- Because of the phenomenon of locality of reference, when a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block.



(a) Single cache

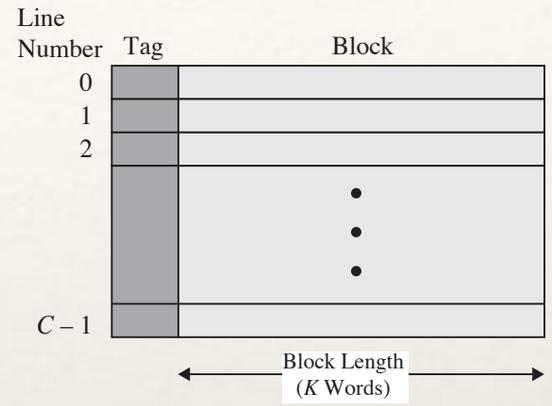


(b) Three-level cache organization

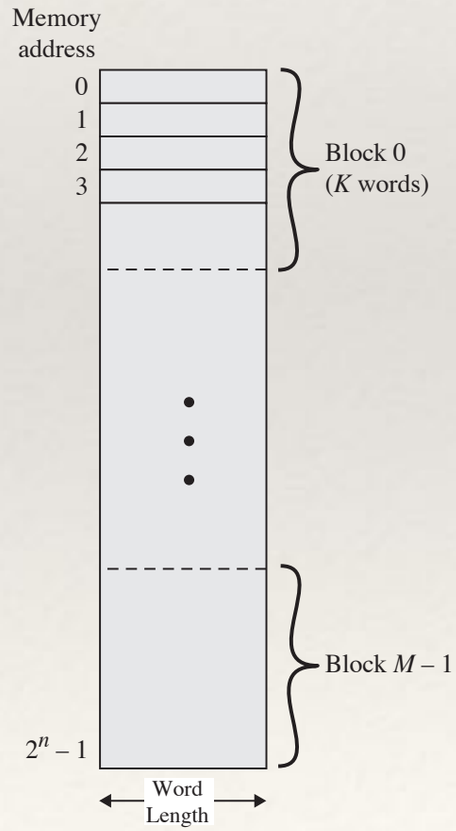


Cache/Main-Memory Structure

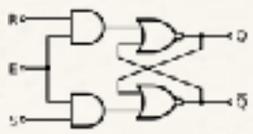
- Divide the memory cache into equal-sized blocks
  - A block in cache is also referred to as a ***cache line***
- When a memory block is referenced, the entire block is moved into the cache
- The next time that block is referenced, it will be retrieved from the cache, instead of from the main memory again
- The cache is addressed by the block number of the ***main memory***, not by the address of the cache
- When the data is not in the cache, the entire block will be brought into the cache
  - If the cache is full, then a block is selected to move out of the cache to provide space for the incoming block
- The principle of Multi-level cache is similar to that of single-level cache



(a) Cache

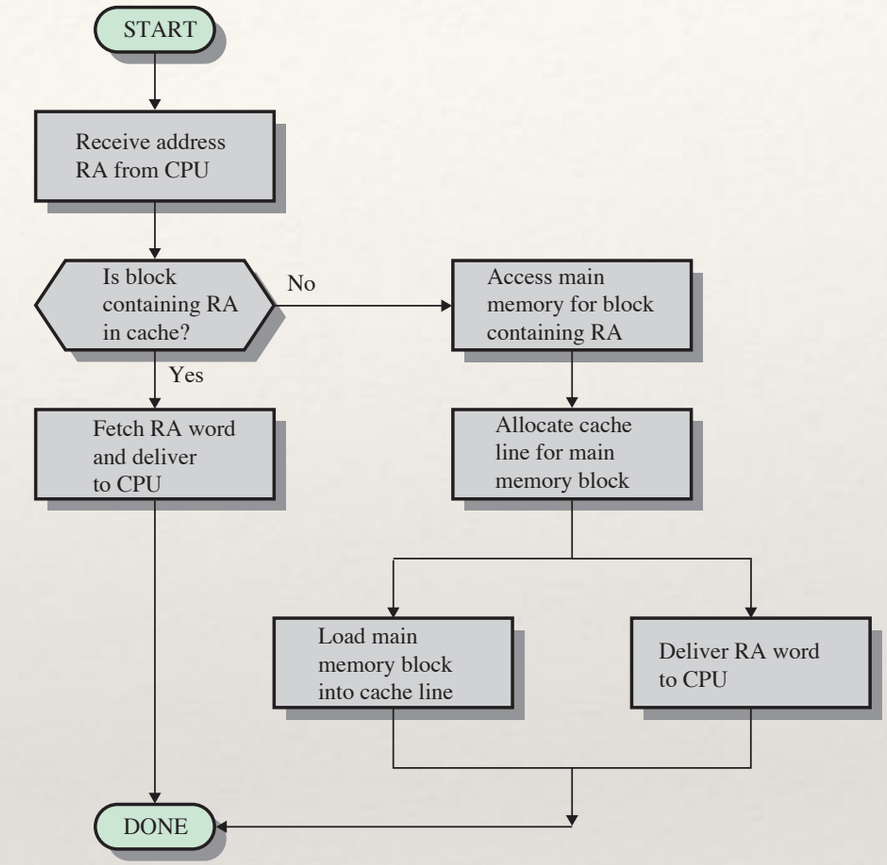


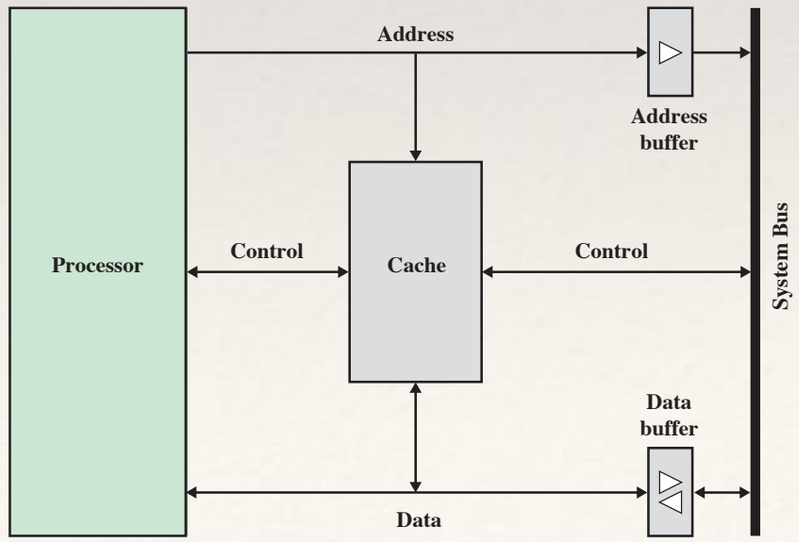
(b) Main memory

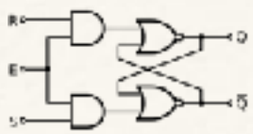


Cache Read Operations

- The processor generates the read address (RA) of a word to be read
  - If the word is contained in the cache, it is delivered to the processor
    - The data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic
  - Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor
    - The desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor
- The last two operations can occur in parallel

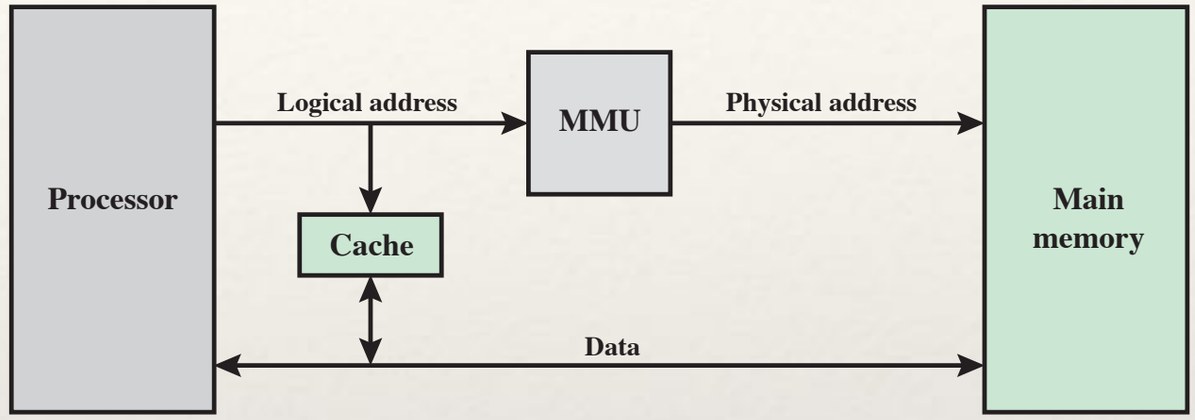




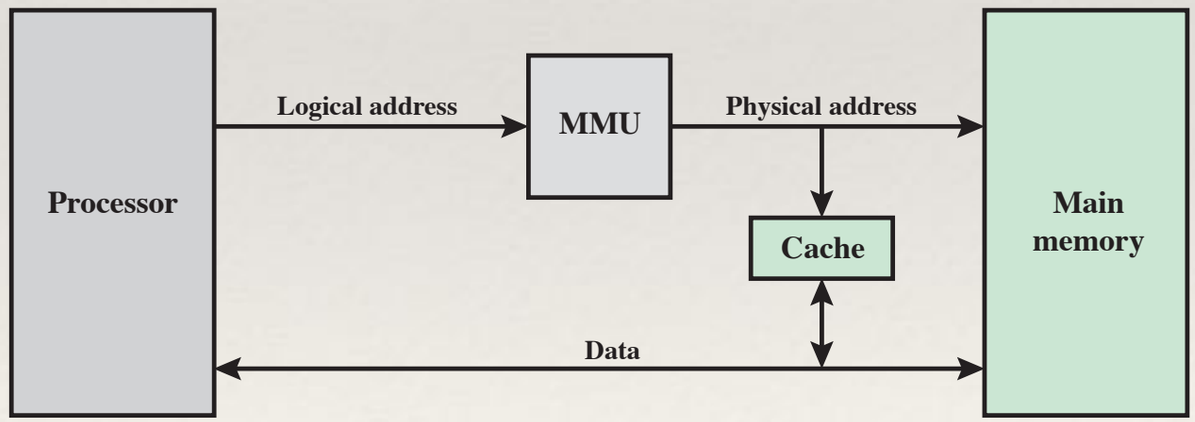


Logical Cache Address

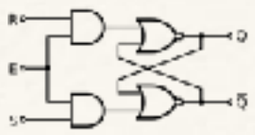
- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware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



(a) Logical Cache

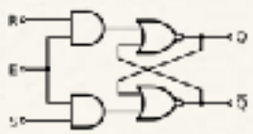


(b) Physical Cache



Cache Size

- Cache has to be small enough so that the overall average cost per bit is close to that of main memory alone
- Large enough so that the overall average access time is close to that of the cache alone
  - The larger the cache, the larger the number of gates involved in addressing the cache
  - Thus, large caches tend to be slightly slower than small ones
- Principle of diminishing return
  - The gain for more cache diminishes



Mapping Function

- 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

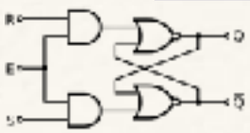
- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

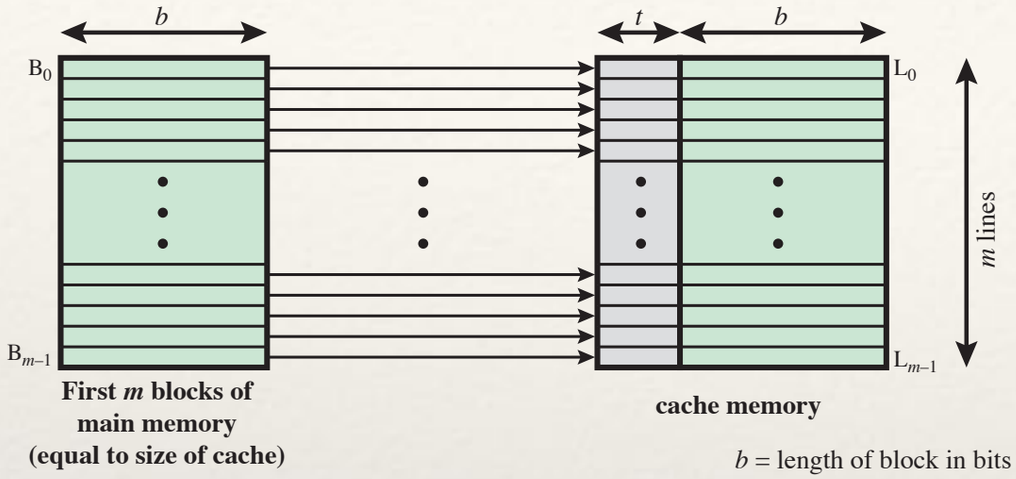
- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

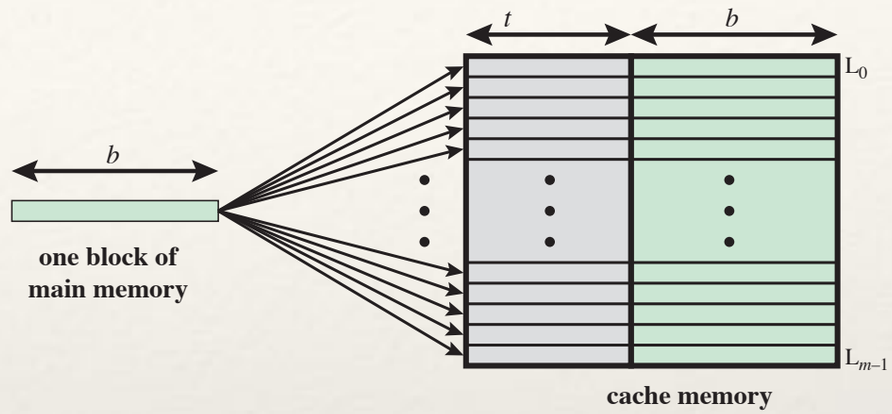
- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages



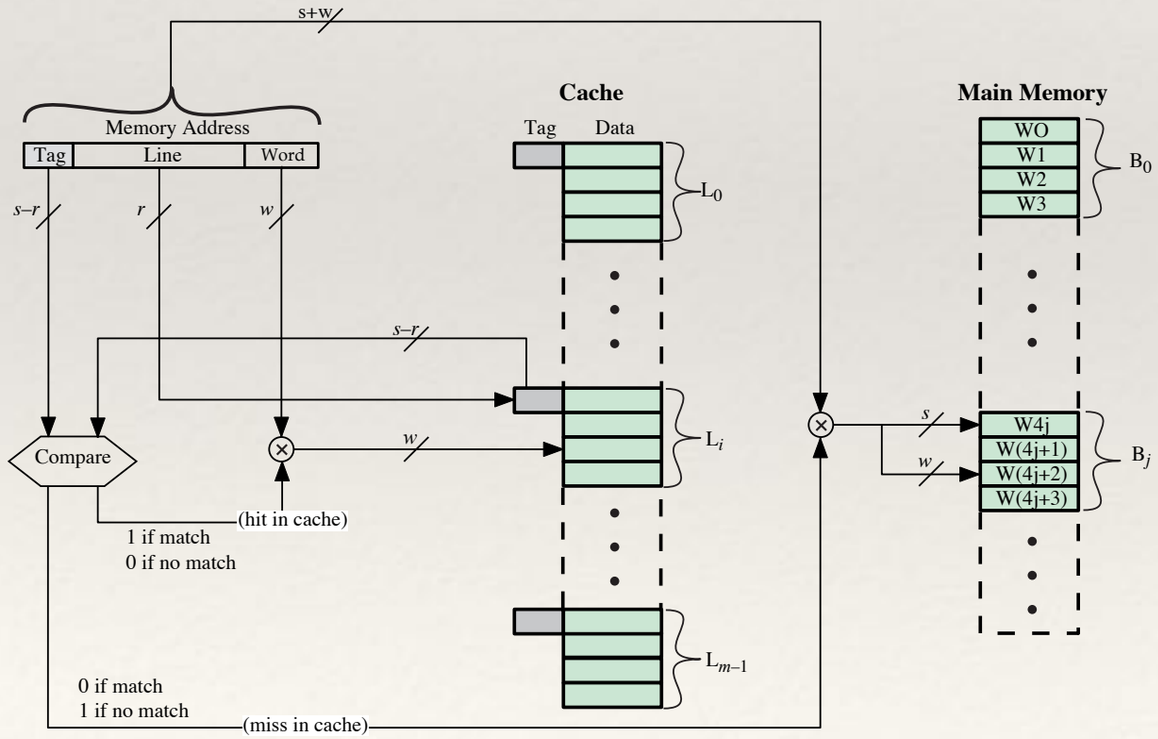
Direct vs Associate Mapping

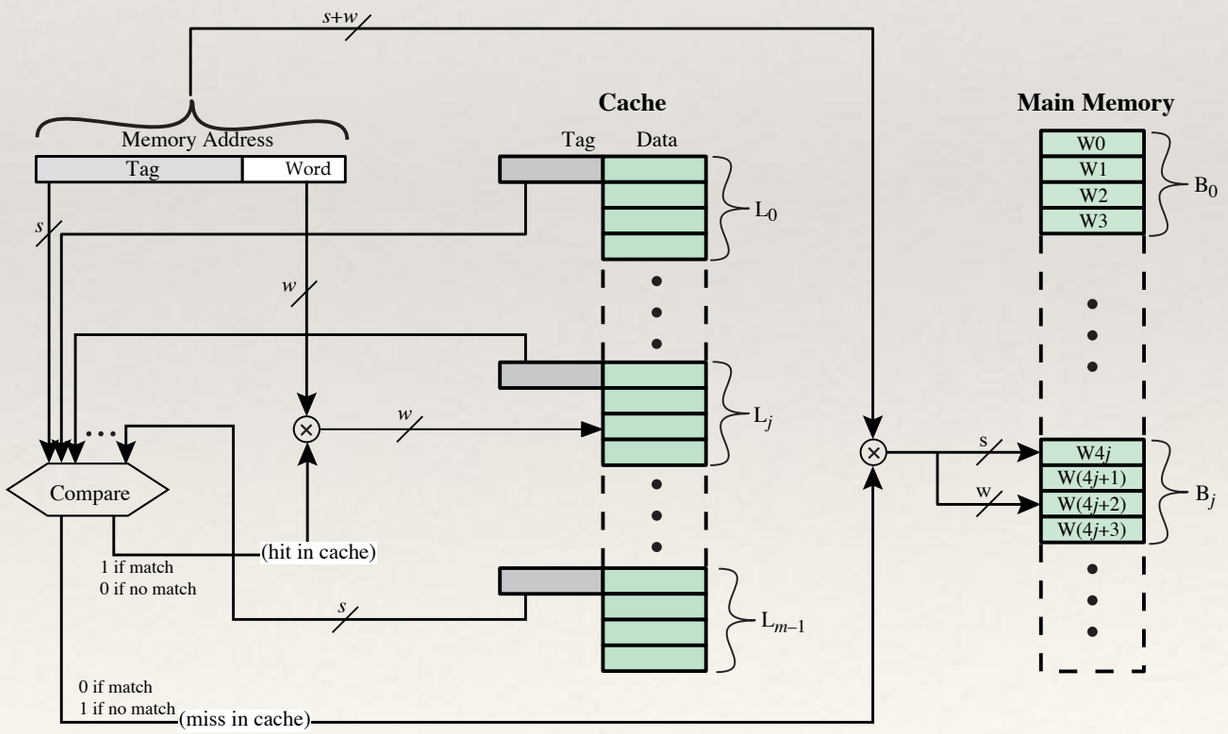


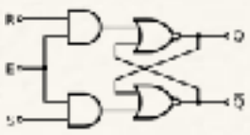
(a) Direct mapping



(b) Associative mapping

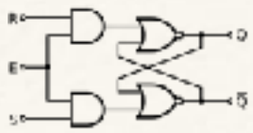






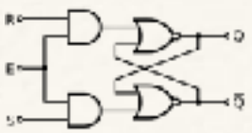
Direct vs Associate Mapping (Cont'd)

- Consider the following scenario
  - Number of blocks in main memory = 2^s
  - Block size = line size = 2^w words or bytes
  - Address length = $(s + w)$ bits
  - Number of addressable units = 2^{s+w} words or bytes
- Direct Mapping
  - Number of lines in cache = $m = 2^r$
  - Size of tag = $(s - r)$ bits
- Associate Mapping
  - Number of lines in cache = undetermined
  - Size of tag = s bits



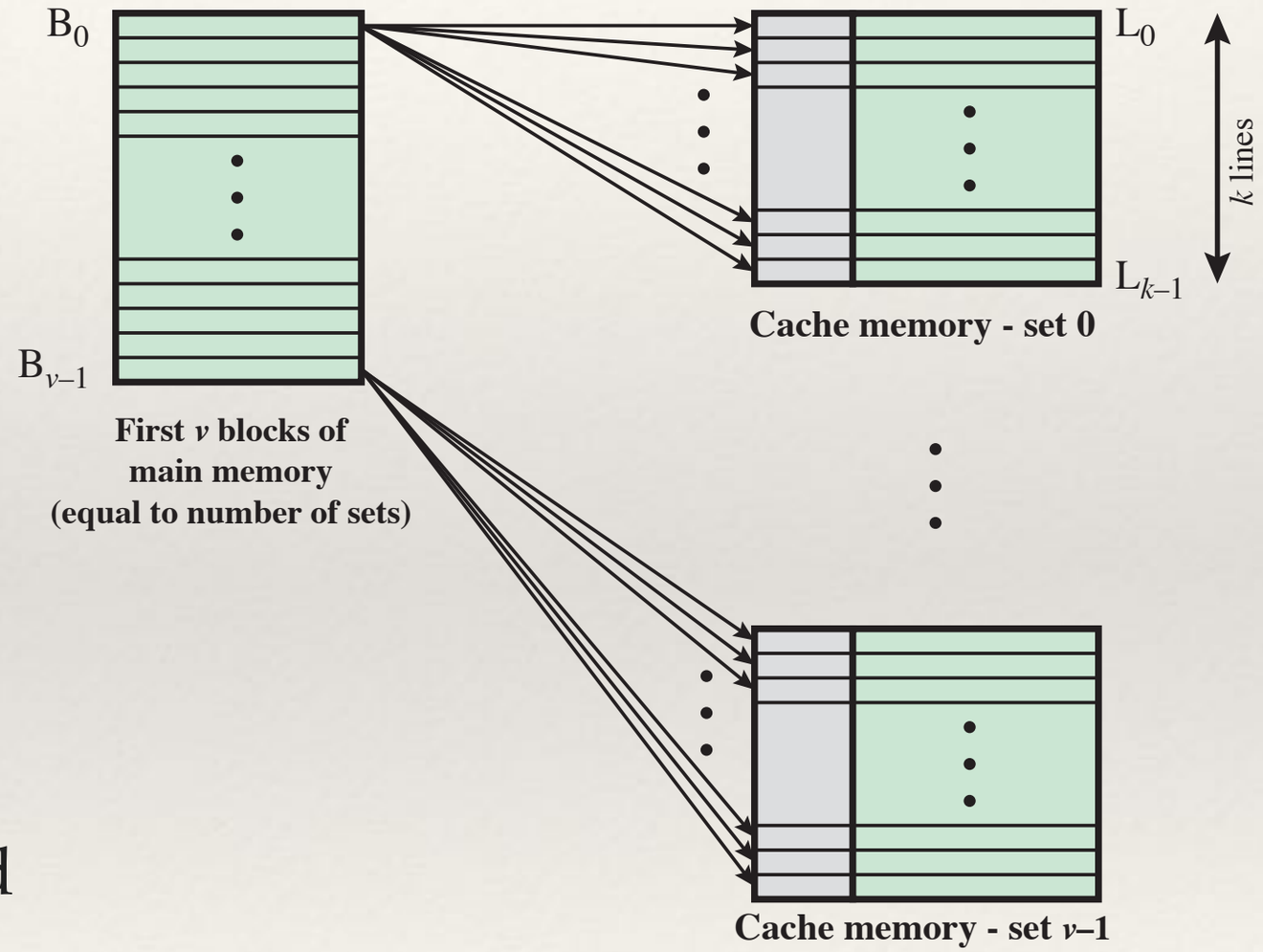
Set Associate Mapping

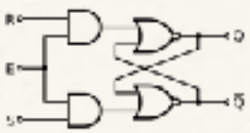
- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
  - Let number of sets = v
- Each set contains a number of lines
  - Let number of lines in each sets = k
  - Total number of lines in the cache is $m = v \times k$
- A given block (with block number j) maps to any line in a given cache set with cache set number i
  - where $i = j$ modulo v
- The cache under this setting is called k-way set-associative mapping
- e.g. 2 lines per set
  - 2 way associative mapping ($k = 2$)
  - A given block can be in one of 2 lines in only one set



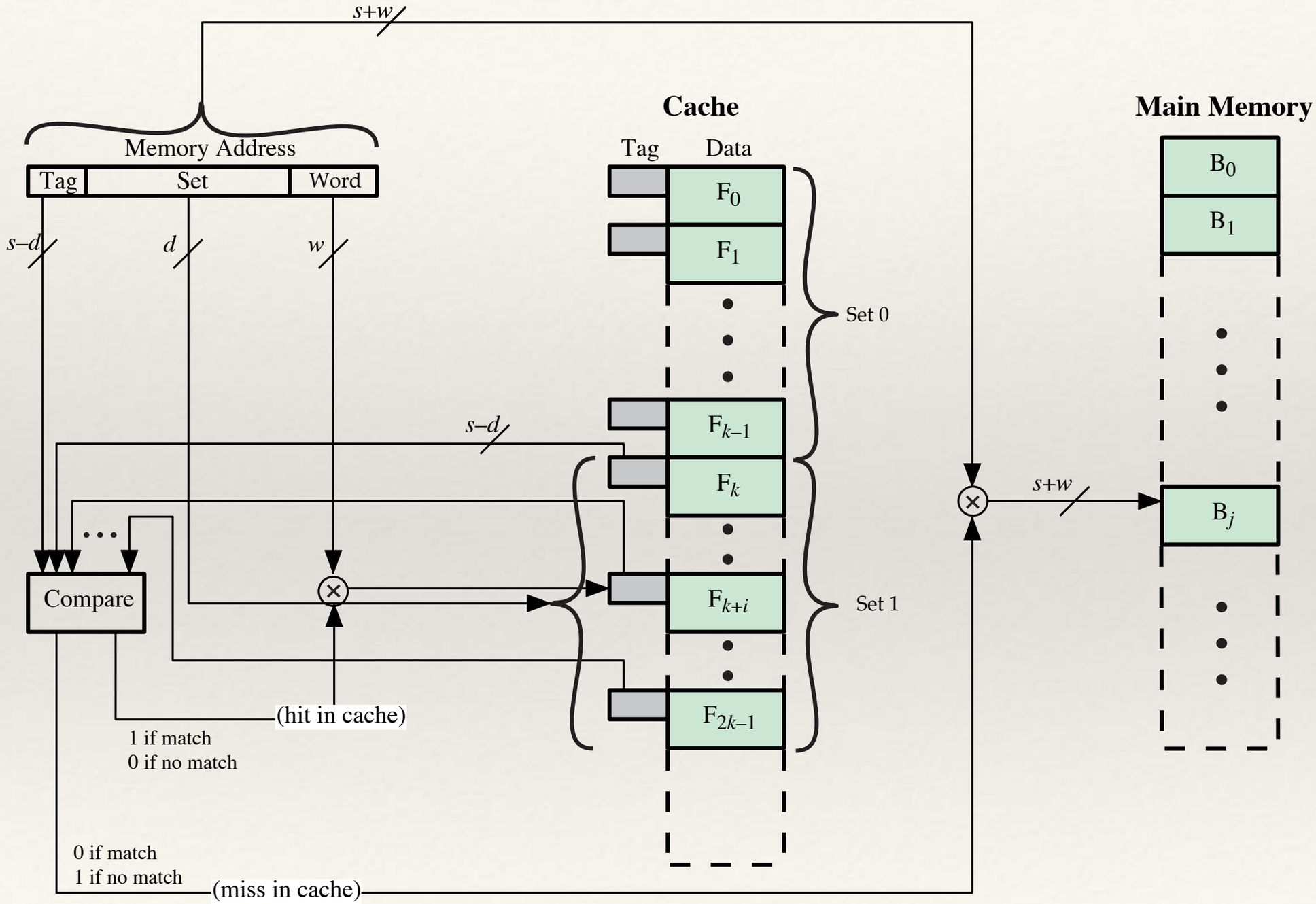
k-way Set-Associative

- 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 B_0 maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as v associative caches.



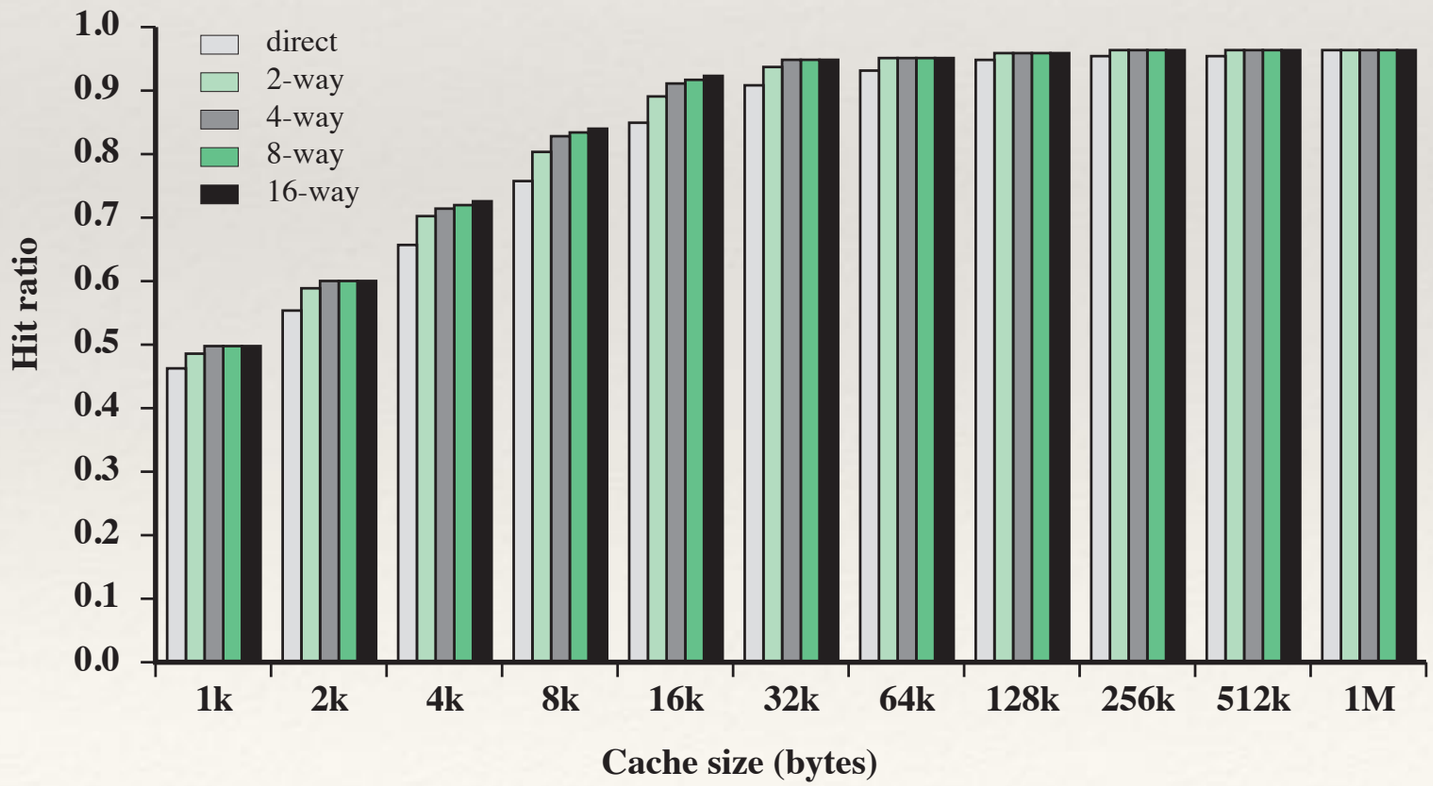


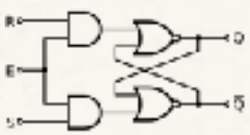
k-Way Set-Associative Cache Organization



Varying Associativity over Cache Size

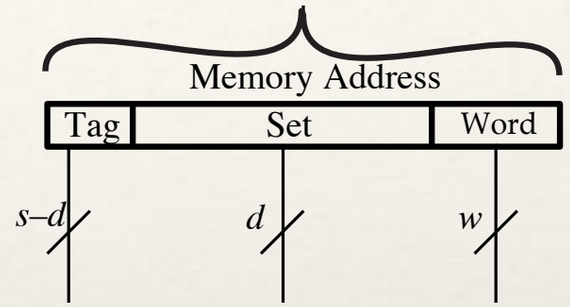
- The difference in performance between direct and two-way set associative is significant up to at least a cache size of 64 kB
- Note also that the difference between two-way and four-way at 4 kB is much less than the difference in going from for 4 kB to 8 kB in cache size.
- The complexity of the cache increases in proportion to the associativity, and in this case would not be justifiable against increasing cache size to 8 or even 16 Kbytes
- Beyond about 32 kB, increase in cache size brings no significant increase in performance
- The results illustrated in the figure below are based on simulating the execution of a GCC compiler. Different applications may yield different results.



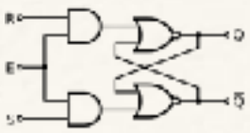


Set Associative Mapping Summary

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes

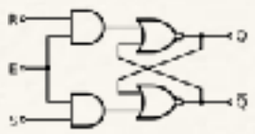


- Number of blocks in main memory = 2^{s}
- Number of lines in set = k
- Number of sets = $v = 2^{d}$
- Number of lines in cache = $m = k \times v = k \times 2^{d}$
- Size of cache = $k \times 2^{d+w}$ words or bytes
- Size of tag = $(s - d)$ bits



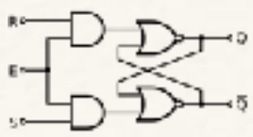
Replacement Algorithm

- 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



Common Replacement Algorithms

- 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

When a block that is resident in the cache is to be replaced there are two cases to consider:

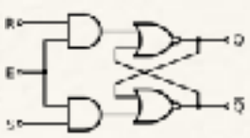
If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

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

There are two problems to contend with:

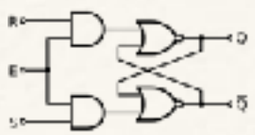
More than one device may have access to main memory

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
  - 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
- Write back
  - Minimizes memory writes
  - 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
  - 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

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

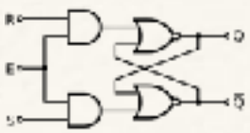
As the block size increases more useful data are brought into the cache

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

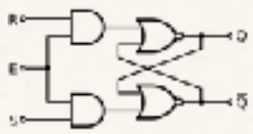
As the block size increases the hit ratio will at first increase because of the principle of locality

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

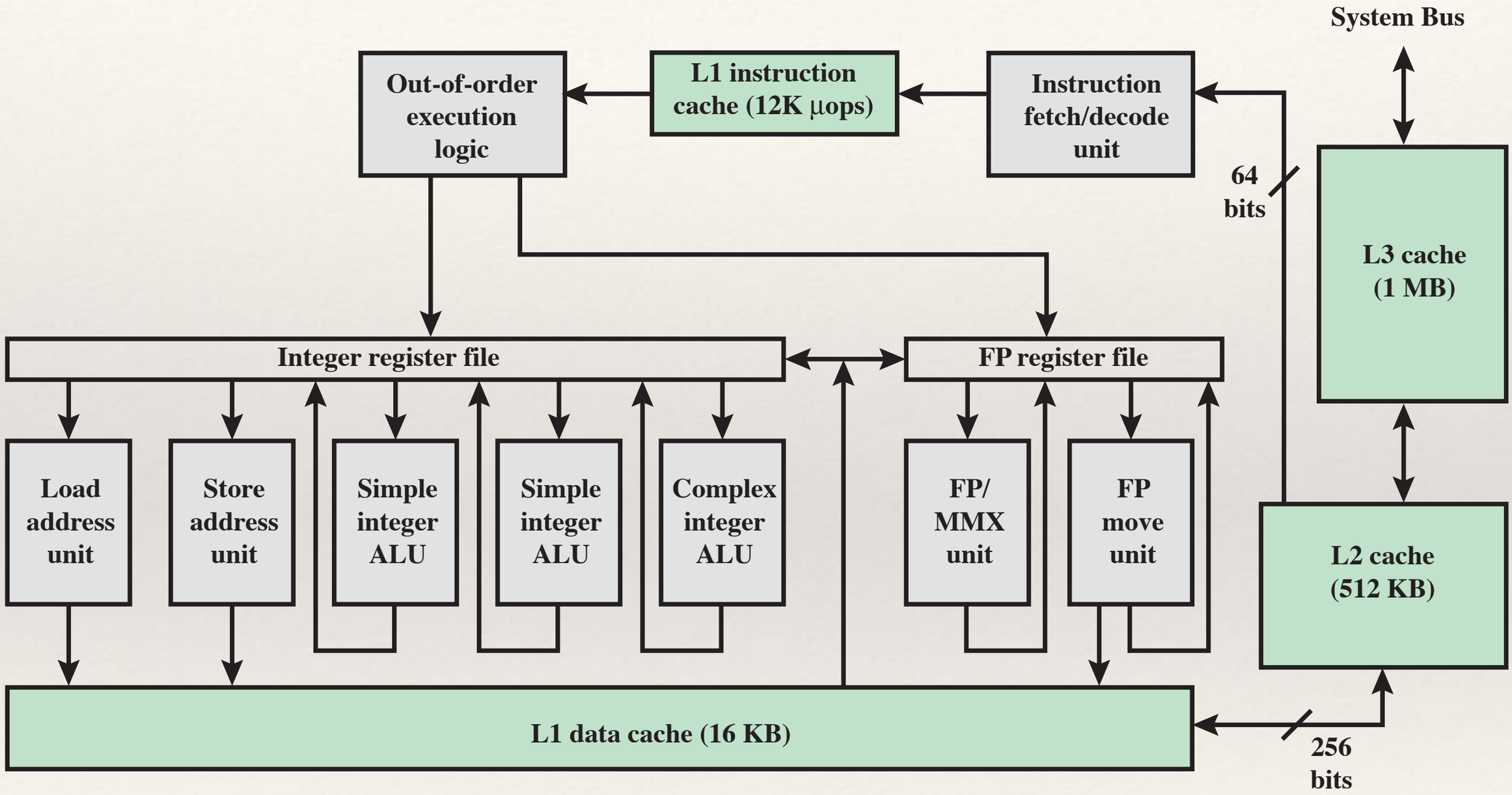


Unified Versus Split Caches

- 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
- 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



Pentium 4 Block Diagram



Chapter 4 - Summary

- Computer memory system overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache memory principles
- Pentium 4 cache organization
- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches