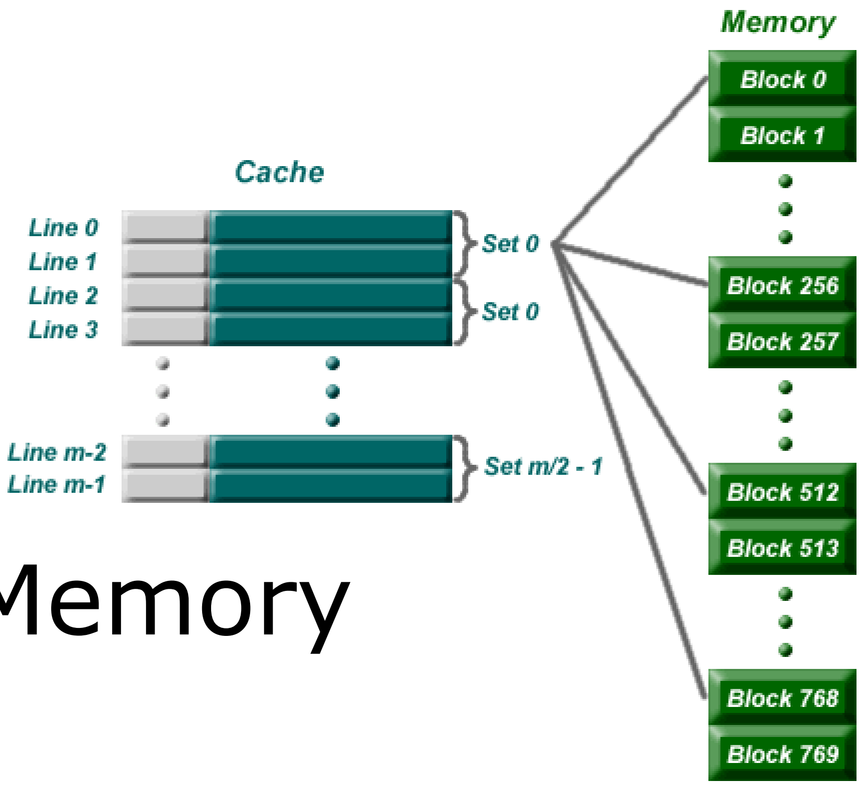


11. Cache Memory

Chapter 4: section 4.3

Mapping Techniques

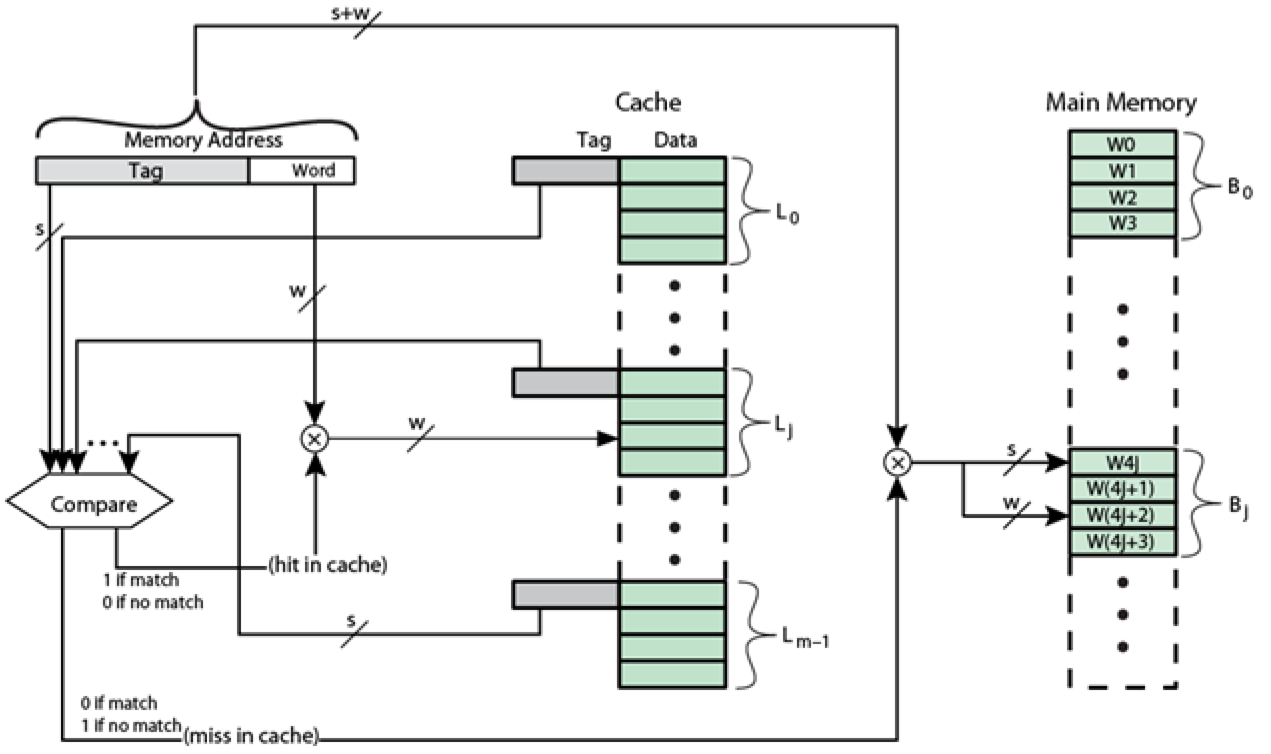
- Cache is much smaller than memory
- We need a technique to map memory blocks onto cache lines
- Three techniques:
  - Direct Mapping
  - Associative Mapping
  - Set-Associative Mapping

ASSOCIATIVE MAPPING

Associative Mapping

- With associative mapping, any block of memory can be loaded into any line of the cache.
- A memory address is simply a tag and a word (note: there is no field for line #).
- To determine if a memory block is in the cache, each of the tags are simultaneously checked for a match.

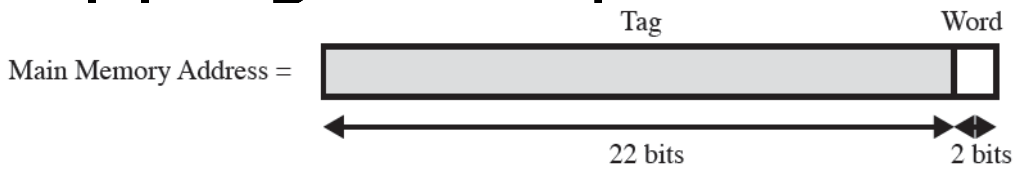
Associative Mapping

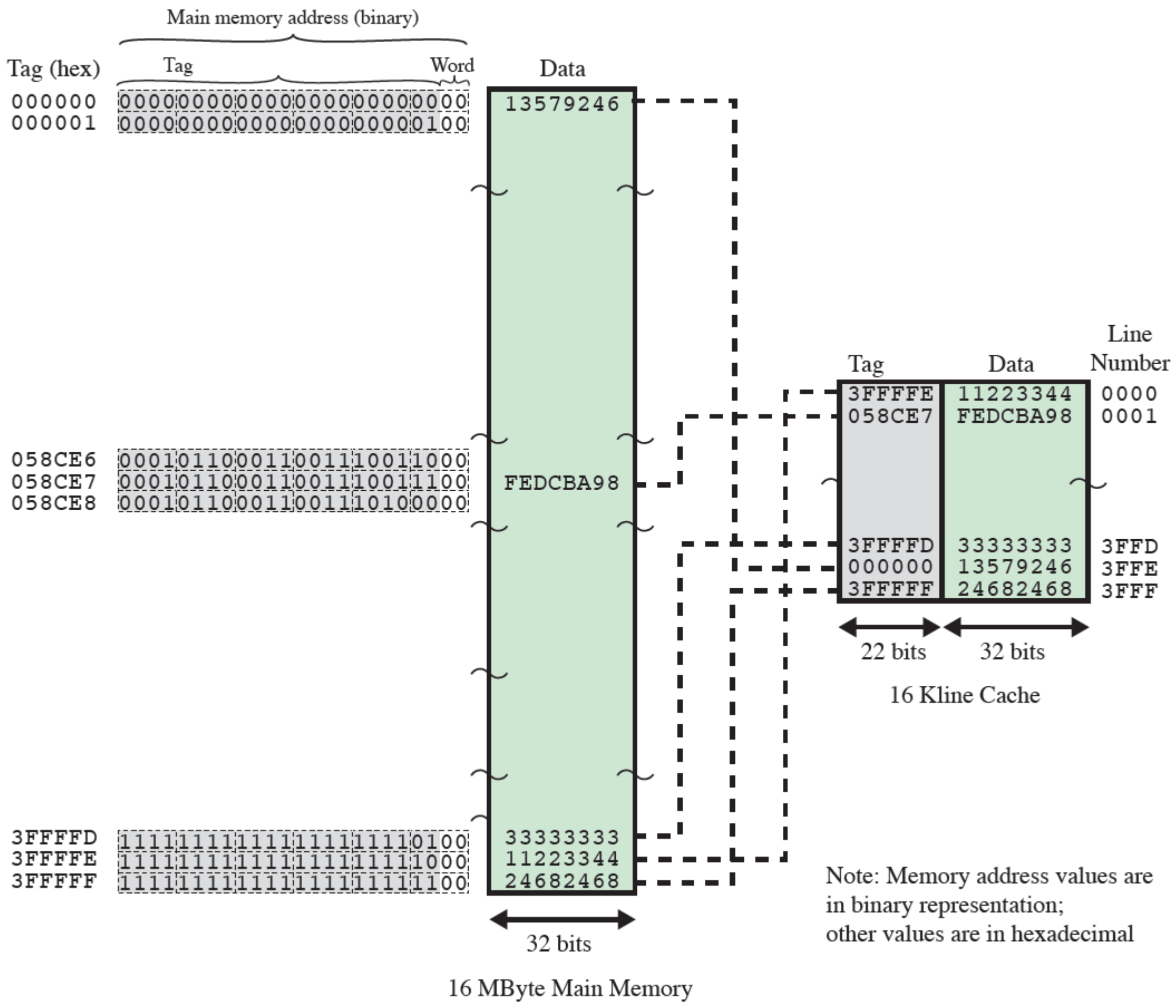


Associative Mapping

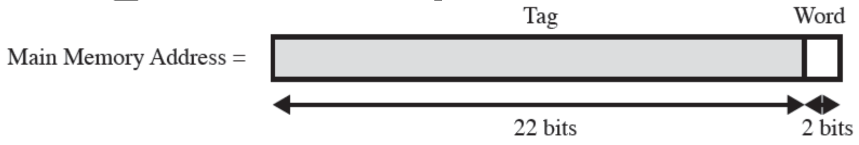
- Address Length is (s + w) bits
- Number of addressable units is 2^{s+w} bytes
- Block size = line size = 2^{w} bytes
- Number of blocks in main memory is $\frac{2^{s+w}}{2^{w}} = 2^{s}$
- Number of cache lines is undetermined
- Tag size is (s) bits

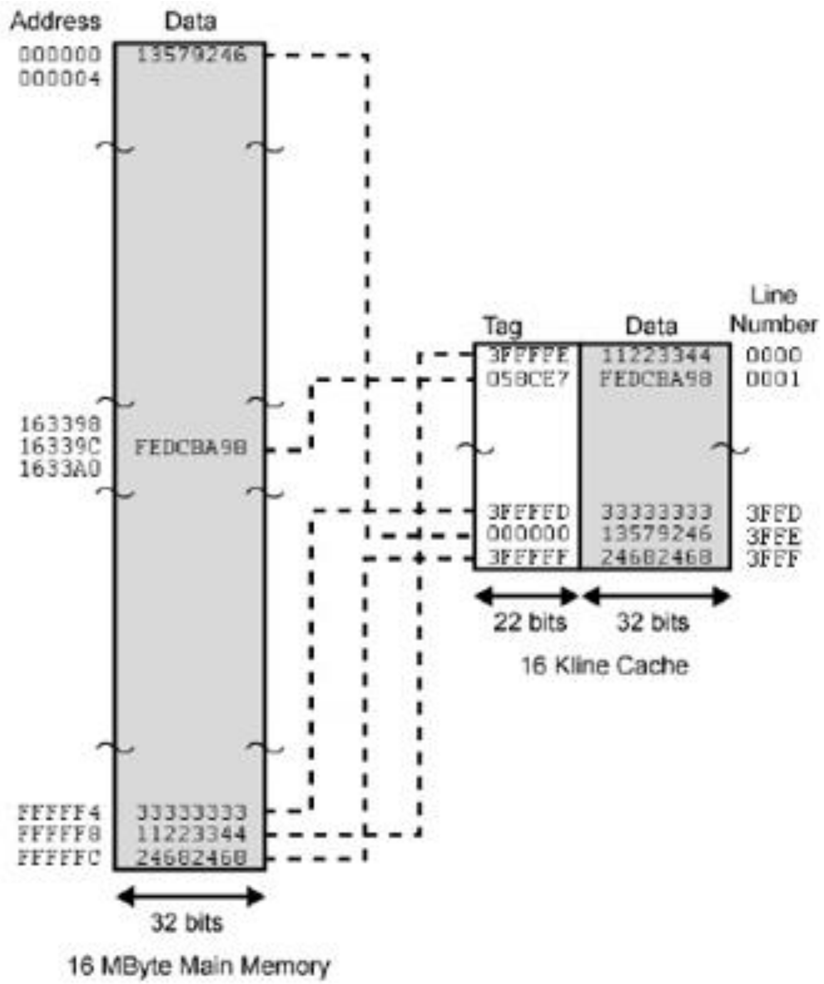
Associative Mapping Example




Main memory address (binary)
Tag (hex)
Tag
Word
Data
000000 0000 0000 0000 0000 0000 0000 13579246
000001 0000 0000 0000 0000 0000 0100
058CE6 0001 0110 0011 0011 1001 1000
058CE7 0001 0110 0011 0011 1001 1100 FEDCBA98
058CE8 0001 0110 0011 0011 1010 0000
3FFFFD 1111 1111 1111 1111 1111 0100 33333333
3FFFFE 1111 1111 1111 1111 1111 1000 11223344
3FFFFF 1111 1111 1111 1111 1111 1100 24682468
32 bits
16 MByte Main Memory
Tag
Data
Line Number
3FFFFE 11223344 0000
058CE7 FEDCBA98 0001
3FFFFD 33333333 3FFD
000000 13579246 3FFE
3FFFFF 24682468 3FFF
22 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation; other values are in hexadecimal

Associative Mapping Example





Associative Mapping

- Advantage of associative mapping:
  - There is flexibility when mapping a block to any line of the cache
- Disadvantages of associative mapping:
  - A replacement algorithm must be used to determine which line of cache to swap out
  - More space is needed for the tag field
  - The most important disadvantage is the complex circuitry needed to examine all of the tags in parallel in the cache

SET-ASSOCIATIVE MAPPING

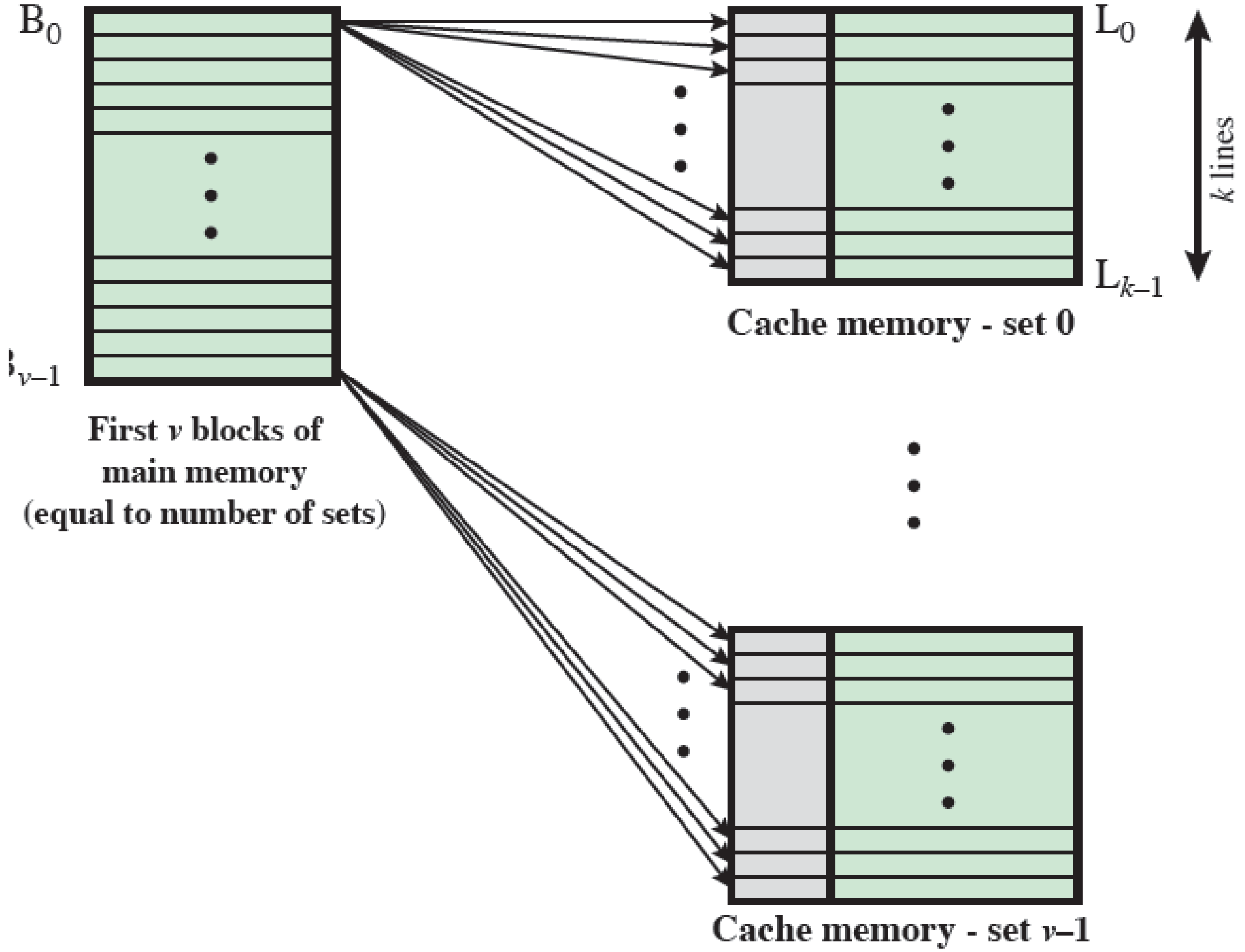
Set Associative Mapping

- Utilizes the strengths of direct and associative mapping while trying to reduce their disadvantages

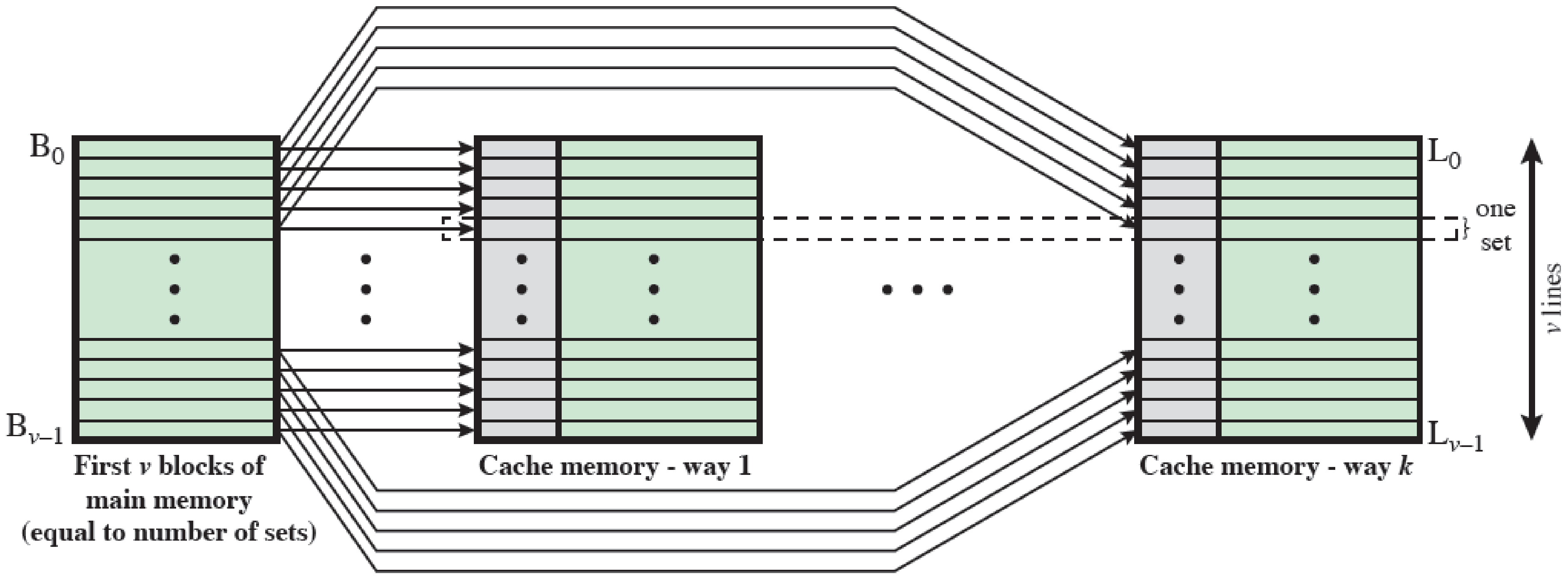
Set-Associative Mapping

- The cache is divided into v sets of k lines per set
  - $m = v \times k$
  - $i = j \ modulo \ v$
- Where:
  - i = cache set number
  - j = main memory block number
  - m = number of lines in cache
  - v = number of sets
  - k = number of lines in each set

Physical Implementation: v associative caches

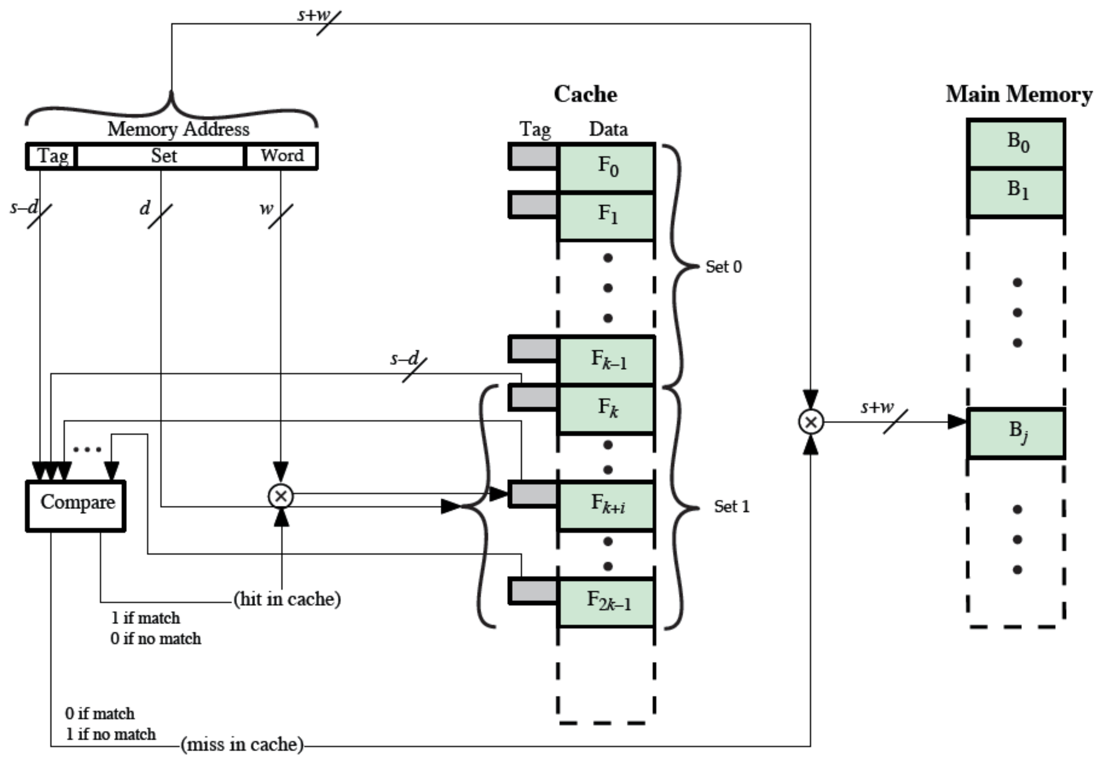


Physical Implementation: k direct mapping caches

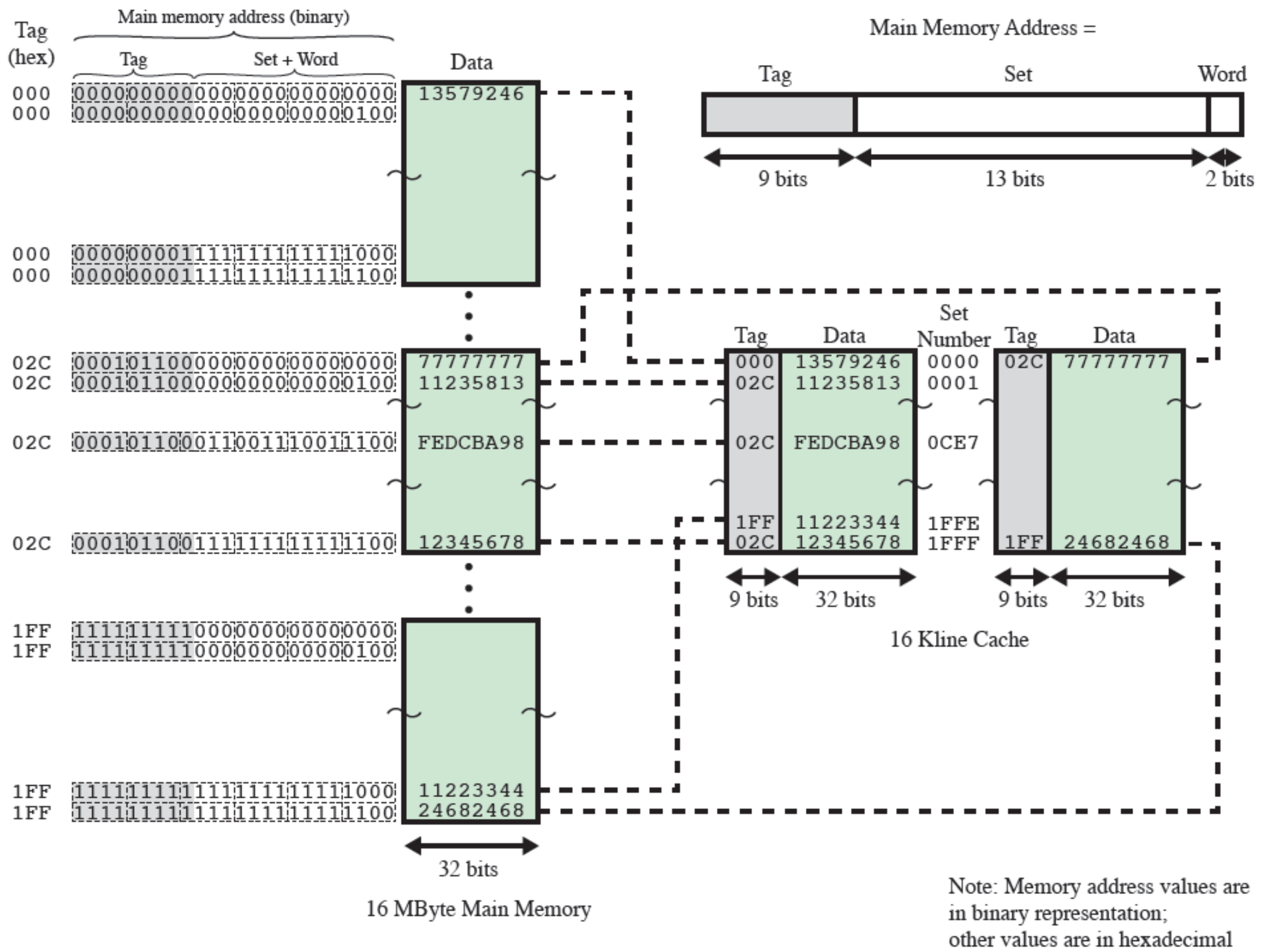


k-way set associative

- What are the values of each of the following:
- Block size
- # blocks in main MM
- # lines in set
- # of sets
- # lines in cache
- size of the cache
- tag size

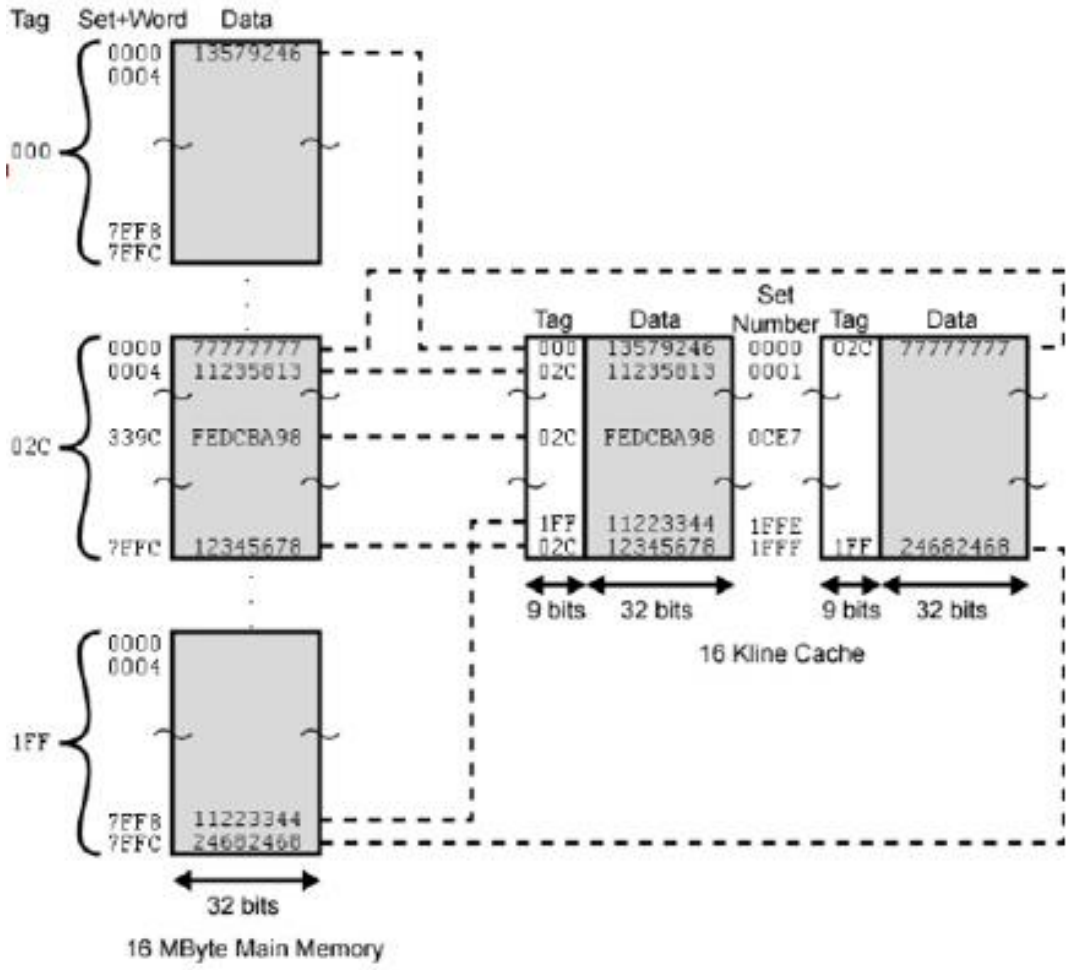


Two-way set-associative Example



Main memory address (binary)
Tag (hex)
Tag
Set + Word
Data
000 0000 0000 0000 0000 0000 0000 13579246
000 0000 0000 0000 0000 0000 0100
000 0000 0001 1111 1111 1111 1000
000 0000 0001 1111 1111 1111 1100
02C 0001 0110 0000 0000 0000 0000 77777777
02C 0001 0110 0000 0000 0000 0100 11235813
02C 0001 0110 0011 0011 1001 1100 FEDCBA98
02C 0001 0110 0111 1111 1111 1100 12345678
1FF 1111 1111 1000 0000 0000 0000
1FF 1111 1111 1000 0000 0000 0100
1FF 1111 1111 1111 1111 1111 1000 11223344
1FF 1111 1111 1111 1111 1111 1100 24682468
32 bits
16 MByte Main Memory
Main Memory Address =
Tag
Set
Word
9 bits
13 bits
2 bits
Tag
Data
Set Number
Tag
Data
000 13579246 0000 02C 77777777
02C 11235813 0001
02C FEDCBA98 0CE7
1FF 11223344 1FFE
02C 12345678 1FFF 1FF 24682468
9 bits
32 bits
9 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation; other values are in hexadecimal

Two-way set-associative



CONTINUE ELEMENTS OF CACHE DESIGN

Cache Replacement Algorithms

- Replacement algorithms are only needed for associative and set associative techniques. To achieve high speed, these algorithms must be implemented in hardware.
  - Least Recently Used (LRU) – replace the cache line that has been in the cache the longest with no references to it (most effective)
  - First-in First-out (FIFO) – replace the cache line that has been in the cache the longest
  - Least Frequently Used (LFU) – replace the cache line that has experienced the fewest references
  - Random – pick a line at random from the candidate lines (simulations have shown this to be slightly inferior to the other algorithms)

Cache Write Policies

- If a cache line has not been modified, then it can be overwritten immediately; however, if one or more words have been written to a cache line, then main memory must be updated before replacing the cache line.

- There are two main potential write problems:

  - If an I/O module is able to read/write to memory directly, then if the cache has been modified a memory read cannot happen right away. If memory is written to, then the cache line becomes invalid.

  - If multiple processors each have their own cache, if one processor modifies its cache, then the cache lines of the other processors could be invalid.

Cache Write Policies

- **Write through** – this is the simplest technique where all write operations are made to memory as well as cache ensuring main memory is always valid. This generates a lot of main memory traffic and creates a potential bottleneck
- **Write back** – updates are made only to the cache and not to main memory until the line is replaced
- Note: Certain studies have shown that about 15% of memory references are writes except for HPC may approach 33% (vector-vector multiplication) and 50% (matrix transposition)
- **Cache coherency** – keeps the same word in other caches up to date using some technique. This is an active field of research.

Cache Coherency

- Cache coherency - keeps the same word in other caches up to date. This is an active field of research.
  - **Bus watching with write through** - each cache controller monitors bus lines to detect write operations by other bus masters. If so, the cache entry is marked invalid
  - **Hardware transparency** - additional hardware is used to write through changes to memory AND update all caches
  - **Noncacheable memory** - a portion of main memory is shared by more than one processor and all accesses are cache misses (.i.e. shared memory is never copied into the cache)

Unified vs Split Caches

- Recent cache designs have gone from a unified cache to a split cache design (one for instructions and one for data).

- Unified caches have the following advantages:
  - Unified caches typically have a higher hit rate
  - Only one cache is designed and implemented

- Split caches have the following advantages:
  - Parallel instruction execution and prefetching is better handled because of the elimination of contention between the instruction fetch/decode unit and execution unit.