William Stallings
Computer Organization and Architecture
8th Edition

Chapter 4

Cache Memory

Chapter Outline

- Overview
- Access Methods
- Performance
- Characteristics
- Hierarchy
- Locality of Reference
- Semi-conductor Memory
- DRAM, SRAM, ROM
- Memory Hierarchy

Characteristics of Memory

- Location
- Capacity
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation

Location & Capacity

- Location
  - closeness to CPU
  - Internal (main memory)
  - External (Peripheral IO Storage Devices, i.e. Hard Discs
- Capacity Number of bytes or Word size
- Unit of transfer
  - Internal memory: usually governed by data bus width
  - External Memory transfer in large units usually a block
- Addressable unit
  - Smallest location which can be uniquely addressed
  - Many systems allow addressing at the byte level
- Word
  - Word: Natural unit of memory organisation 8, 16, 32 bits
  - Word size : # of bits to represent int., or an instruction length.

Access Methods (1)

Access time: (AS)Time from moment an address is selected till moment location is reached

- **Sequential**
  - -Start at the beginning and read through in order
  - -AS depends on location of data and previous location
  - -Specific linear sequence. Units of data=records,e.g. tape
- **Direct**
  - Shared R/W mechanism
  - Individual blocks have unique address based on physical location, e.g. disk
  - Access is by jumping to vicinity plus sequential search
  - AS depends on location and previous location

Access Methods (2)

- **Random**
  - Each addressable location has a unique wired-in addressing mechanism
  - Individual addresses identify locations exactly
  - AS to a given location is independent of the sequence of prior accesses and is equal
  - e.g. RAM

- **Associative**
  - Random access like
  - Data is located by comparison of desired bit locations within a word for a specified match, and do this for all words simultaneously
  - AS is independent of location or previous access
  - e.g. cache

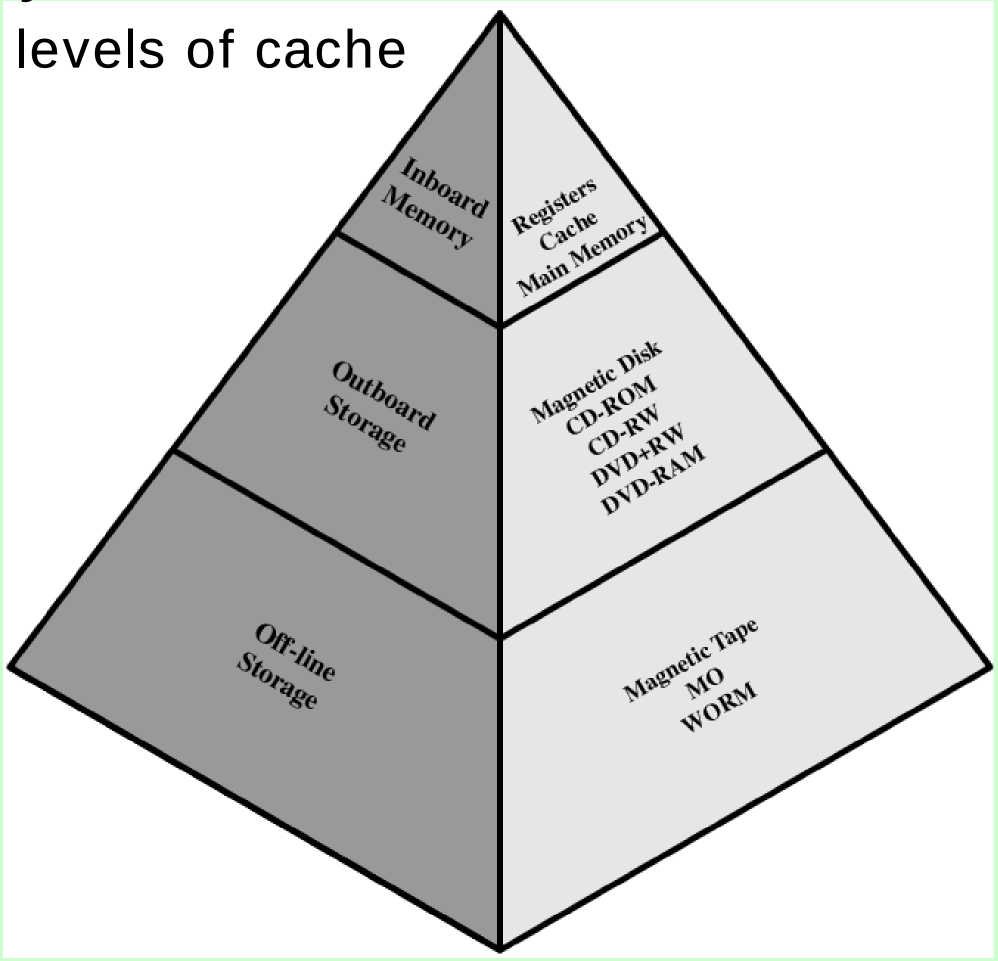
Memory Hierarchy

Memory Hierarchy - Diagram

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - “RAM”
- External memory
  - Backing store

As one goes down the following occur:

a. Decreasing cost per bit
b. Increasing capacity
c. Increasing access time
d. Decreasing frequency of access of the memory by the processor



Performance

- **Access time**
  - Time between presenting the address and getting the valid data
  - For RAM, it is time to perform a R/W operation
  - For non-RAM, it is time it takes to position R/W mechanism at desired location
- **Memory Cycle time**
  - Time may be required for the memory to “recover” before next access
  - Cycle time is access + recovery or (time required before a 2nd access (a.k.a. recovery time)
- **Transfer Rate**

  - Rate at which data can be moved

  $T_n = T_a + n/R$

  Where: T_n Average time to read or write n bits,

  T_a Average access time,

  n# of bits, R Transfer rate in bits per second (bps)

Physical Types	Physical Characteristics
• Semiconductor — RAM • Magnetic — Disk & Tape • Optical — CD & DVD • Others — Bubble — Hologram	• Decay • Volatility • Erasable • Power consumption

Organisation

- Physical arrangement of bits into words
- Not always obvious
- e.g. interleaved

The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?

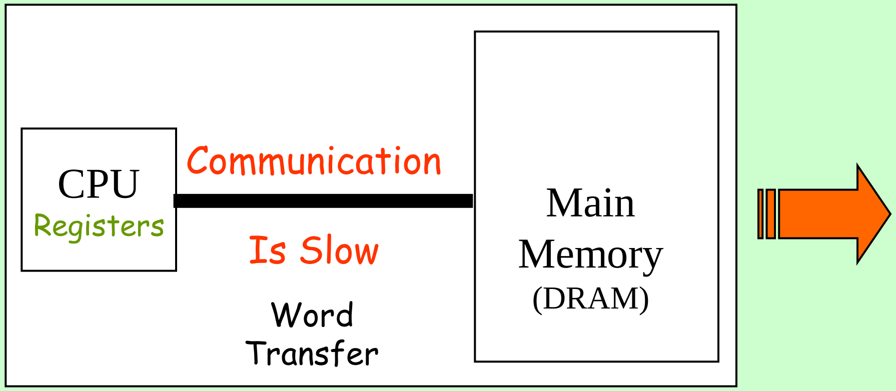
Latency & Bandwidth

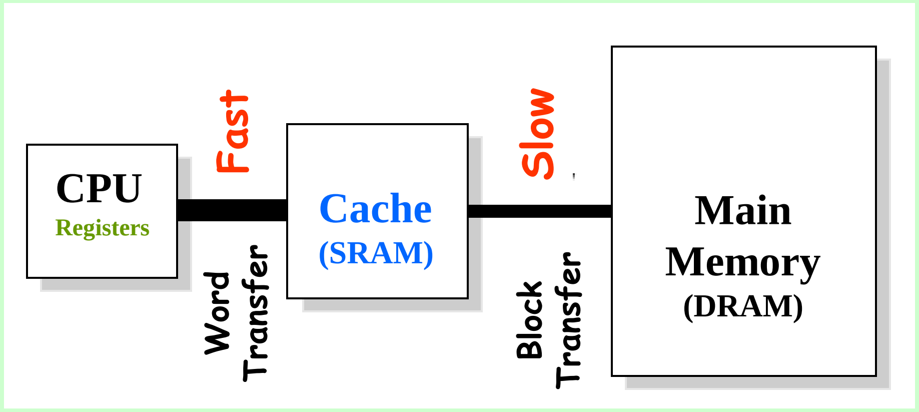
- Latency: Delay in supplying an operand
- Bandwidth: Amount of data supplied per unit time

Hierarchy List

- Registers
- L1 Cache, L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

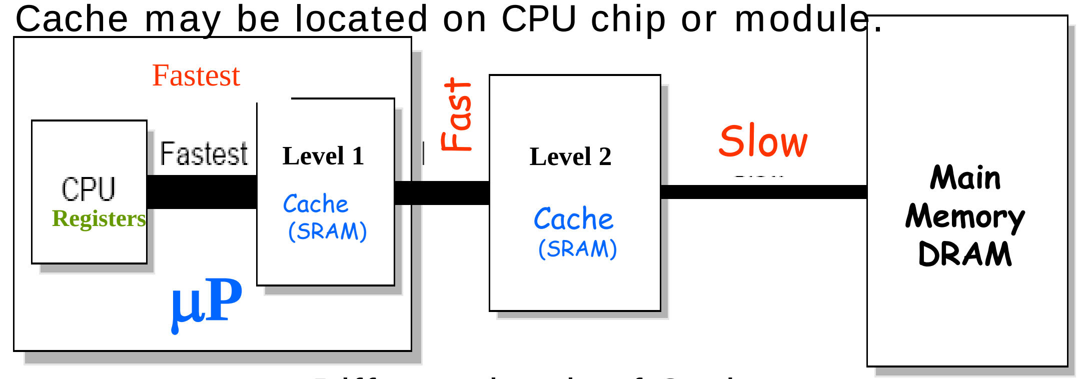
Cache Principles





- **CACHE**
- Masks the slow DRAM storing frequently accessed data, or may soon be accessed in a small fast SRAM memory (Locality).
- Small amount of fast memory may be located on CPU chip or module
- Sits between normal main memory and CPU
- Its job is to reduce memory access by CPU

Cache may be located on CPU chip or module.



Different levels of Cache.

Temporal Locality

Data that have been used recently, have high likelihood of being used again

- Code: loops, routines,…
- Local variables and data structures

Spatial Locality

Data which follow in the memory other data which are currently being used are likely to be used in the future

Code: usually read sequentially
Arrays

Cache operation – overview

CPU requests contents of memory location

- CPU Check cache for this data
- If present, get from cache (fast)

  -If not present:

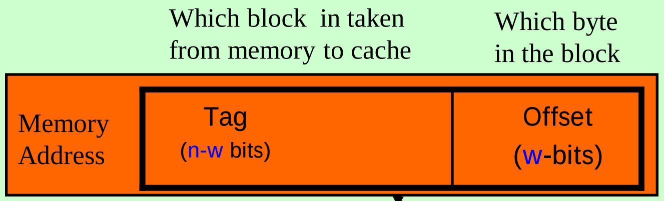
  - read required block from main memory to cache
  - Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

- The challenge in *cache design* is to ensure that the desired data and instructions are in the cache. The cache should achieve a *high hit ratio*.
- The cache system must be quickly searchable as it is checked every memory reference.

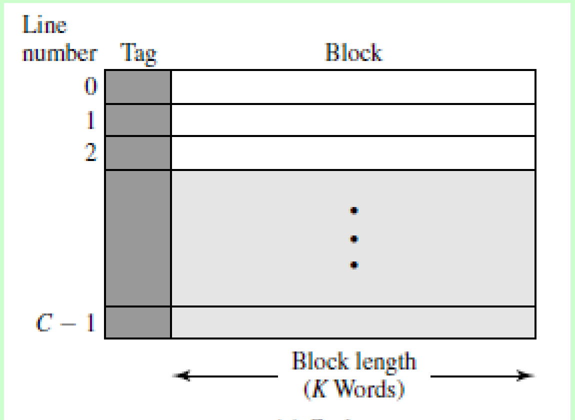
Cache/Main Memory Structure

A cache line contains two fields

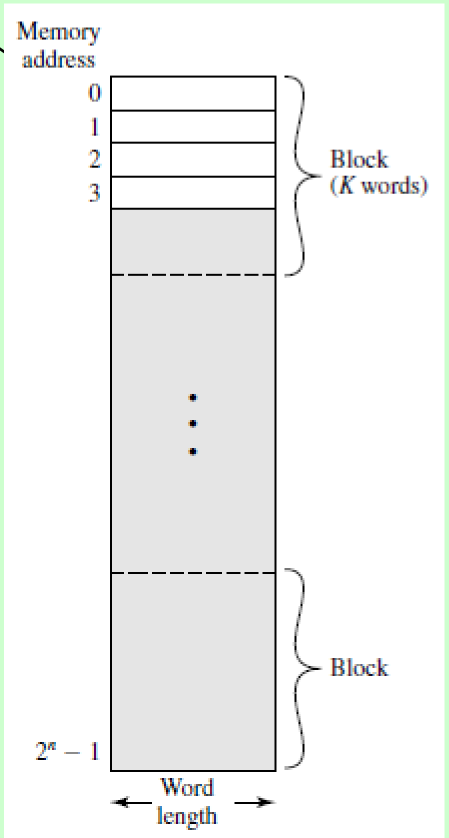
- Data from RAM (BLOCK)
- The address of the block currently in the cache (TAG).
- TAG field specifies the address currently in the cache line.



- MM consists of up to 2^n addressable words, each having a unique n-bit address.
- Cache consist of a # of fixed length blocks of K words each.



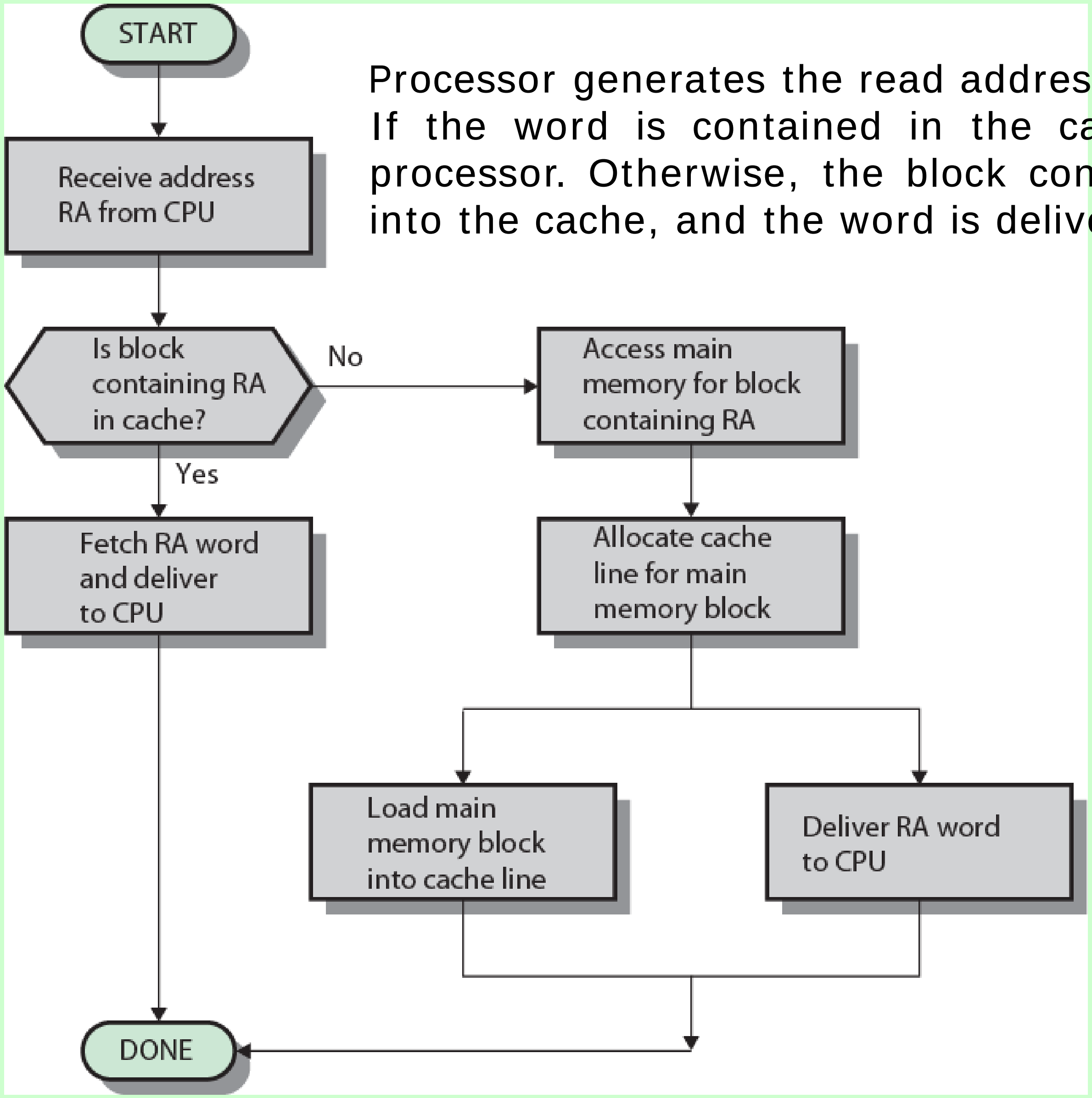
(a) Cache



(b) Main memory

- There are $M = 2^n/K$ blocks in main memory
- The cache consists of m blocks, called **lines**. Each line contains K words, plus a tag.
- The length of a line, without tag and control bits, is the **line size**.

Cache Read Operation - Flowchart


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Allocate cache line for main memory block
Fetch RA word and deliver to CPU
Load main memory block into cache line
Deliver RA word to CPU
DONE

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. Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor

Cache Design

- Addressing
- Size
  - more cache is expensive
  - More cache is faster
- Mapping Function
  - which part of memory goes where in cache
- Replacement Algorithm
  - when replacing data in cache what to throw out
- Write Policy
  - when storing results, store them in cache only?
- Block Size
- Number of Caches: L1, L2, L3

Cache Design

- Cache hit/miss
- Hit ratio: *h*, ratio of having hits to all cache references
- Effectively reduces memory latency

- Miss ratio: 1-*h*
- Miss penalty: time the processor is stalled because the required data or instruction is not available for execution

- Average access time: $t_{ave} = hC + (1-h)M$

  Where:
  C = access time to info in cache
  M = Miss Penalty

Cache Design

- Using multiple caches improves bandwidth & latency: L1, L2, L3
- L2 may be slower than L1, but its speed is less critical, only affecting the miss penalty of L1

- L2 must be a lot larger to:
  - Ensure a large hit ratio
  - reduce the impact of main memory speed on performance

- Average access time:

$$t_{ave} = h_1 C_1 + (1 - h_1)\, h_2 C_2 + (1 - h_1)(1 - h_2) M$$

Where $(1 - h_1)(1 - h_2)$ is the number of misses in the L2

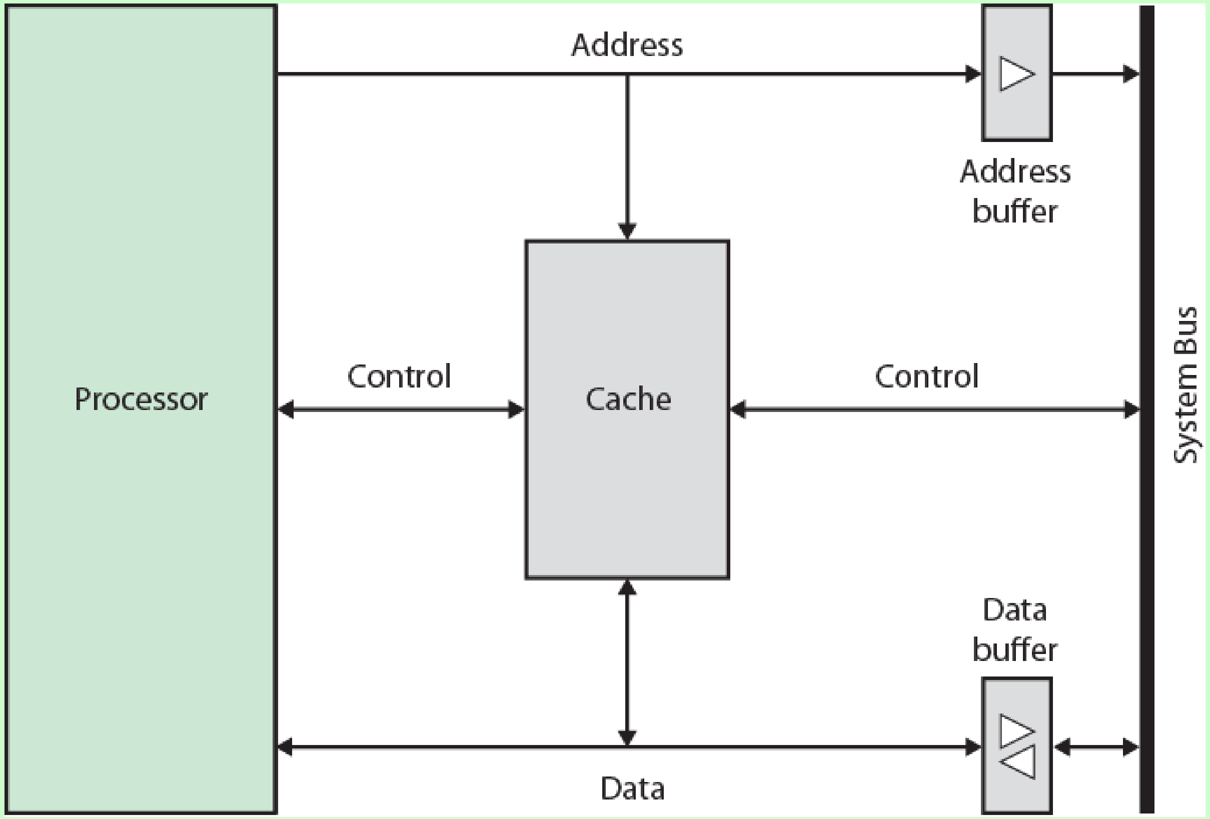
- If h1 & h2 are 90%, then the number of misses is less than 1% of the processor memory access

Cache Addressing

- Where does cache sit?
  - Between processor and virtual memory management unit
  - Between MMU and main memory
- Logical cache (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly, not thorough physical cache
  - Cache access faster, before MMU address translation
  - Virtual addresses use same address space for different applications
    - Must flush cache on each context switch
- Physical cache stores data using main memory physical addresses

Typical Cache Organization

- The cache connects to the processor via data, control, and address lines.
- The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.



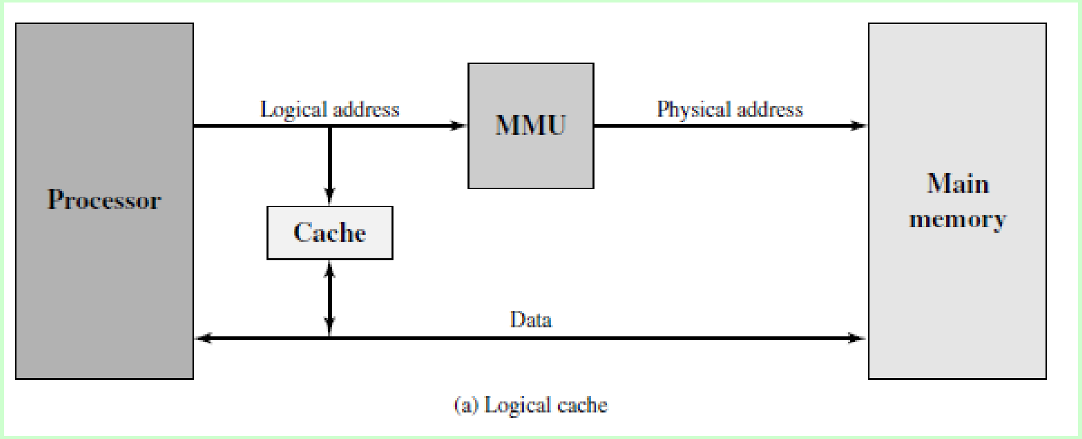
When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache with no system bus traffic. When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both cache and processor.

Logical and Physical Cache

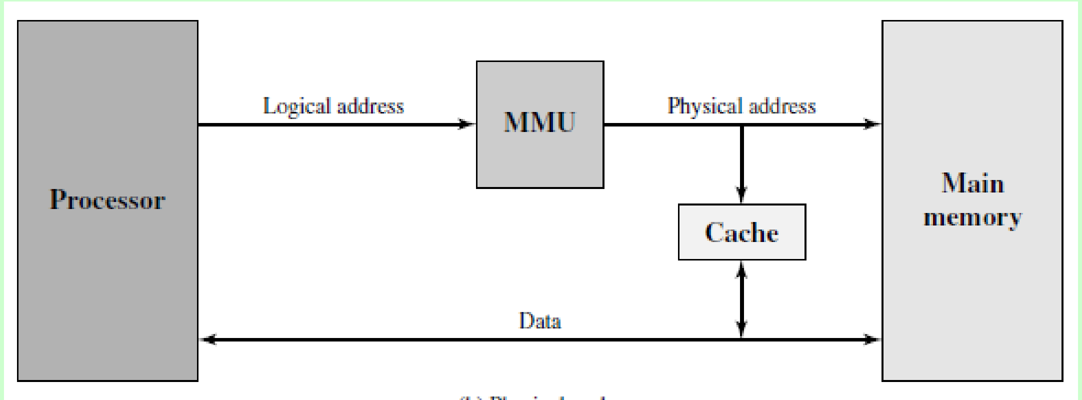
The cache can be places between the processor and the MMU or between the MMU and main memory.

A logical or virtual cache, stores data using **virtual addresses**.
Processor accesses the cache without going through the MMU.
Cache access speed is faster than physical cache, because it can respond before MMU performs any address translation

A physical cache stores data using main memory **physical addresses**.



(a) Logical cache



(b) Physical cache

Mapping Function

- A algorithm is needed for mapping main memory blocks (MMB)into cache lines
- A way is needed for determining which MMB currently occupies a cache line
- Choice of mapping function dictates how cache is organized:
  - Direct: each address has a specific place in the cache
  - Associative: search the entire cache for an address
  - Set associative: each address can be in any of a small set of the cache locations

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

Cache line	Main Memory blocks held
0	0, m, 2m, 3m..., 2^n-m (m = 2^k)
1	1, m+1, 2m+1..., 2^n-m+1
...	
2^k-1	m-1, 2m-1, 3m-1..., 2^n-1

- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r & a tag of s-r

Direct Mapping pros & cons

- Simple & Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

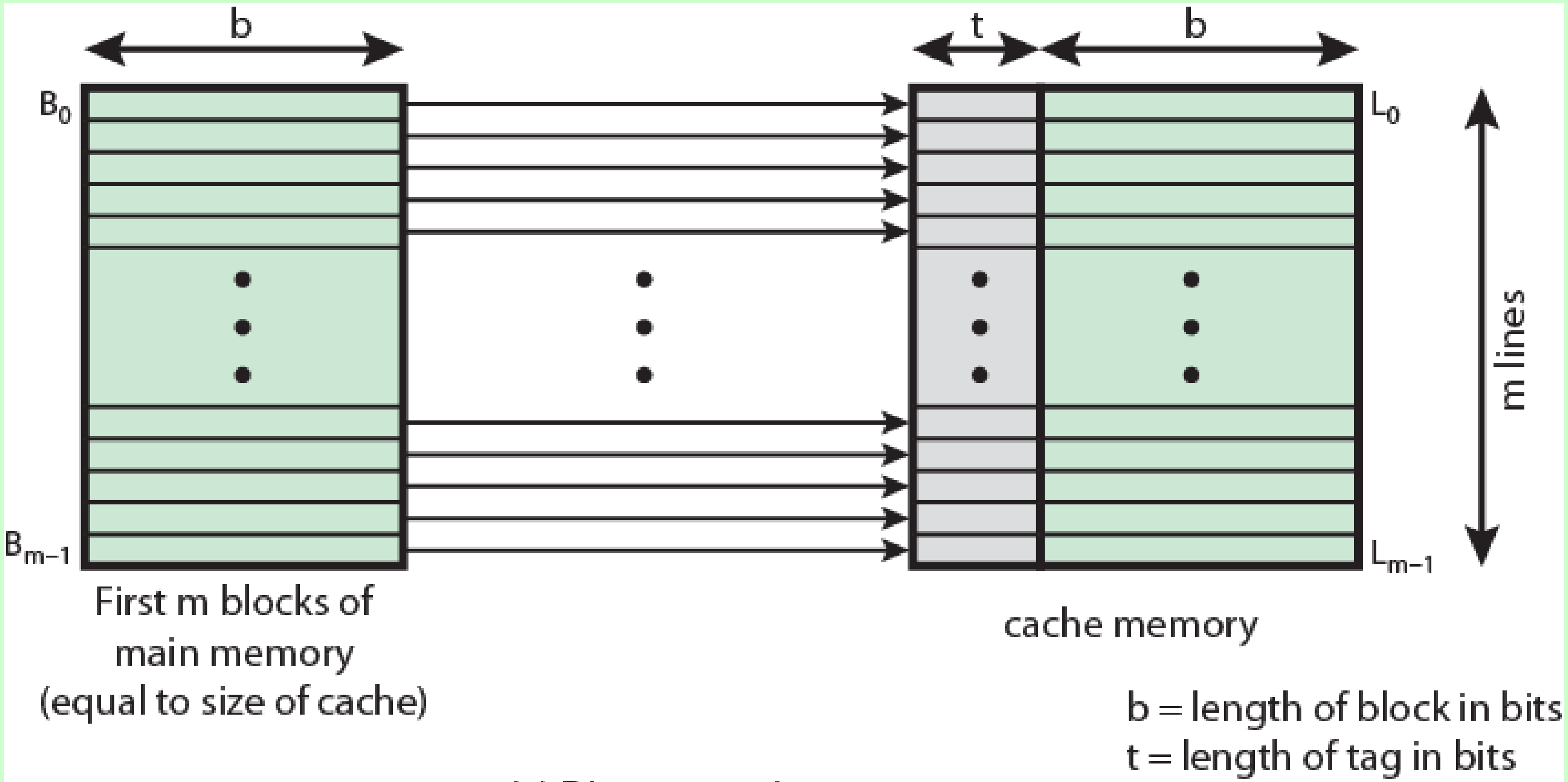
Direct Mapping Address Structure

- 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 the cache = 2^{r+w} words or bits
- Size of tag=(s–r) bits, cache size/block size =#of lines

TAG	LINE or Slot	Word
(s-r)	(r)	(w)

- The lower W bits=log2(block size) of memory address define which byte in the block
- The next r-bits =log2(number of lines) defines which line of the cache (Index)
- The remaining upper bits are the TAG field

Direct Mapping from Cache to Main Memory



b
t
b
B_0
B_{m-1}
L_0
L_{m-1}
m lines
First m blocks of main memory (equal to size of cache)
cache memory
b = length of block in bits
t = length of tag in bits

(a) Direct mapping

Direct Cache Mapping Example

Assuming

- 4 GB DRAM addressed then by n=32 bits. (2^{32} = 4 GB)
- 32 KB of cache
- Blocks of 64 bytes. Then the byte offset is w=6 bits. (2^6 = 64)
- Number of cache lines is then 32 KB / 64 = 512.
- The line index is then k=9 bits (= $\log_2(512)$ = 9).
- The TAG field is then n-k-w = 17 bits.

TAG	INDEX	OFFSET
(17)	(9)	(6)

With a memory address of 01001101011101110101101101101001

- Compare the TAG field of line 101101101 in cache
- for the value 01001101011101110.
- If it matches, return byte 101001 of the line.

Memory address

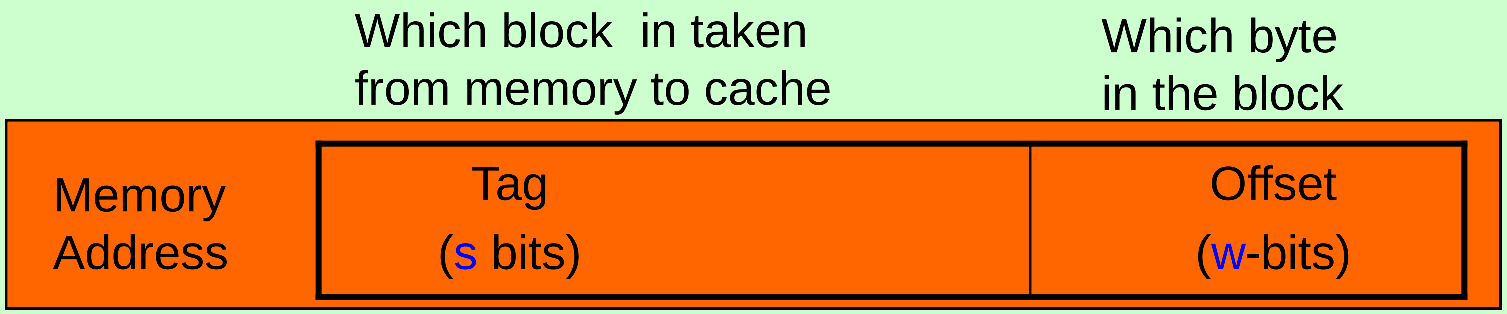
01001101011101110	101101101	101001
TAG	INDEX	OFFSET
(17)	(9)	(6)

Associative Mapping

- The data from any location in RAM can be stored in any location in cache
- A main memory block can load into any line of cache
- When processor wants an address, all tag fields in the cache are checked to determine if the data is already in the cache
- Each tag line requires circuitry to compare the desired address with the tag field of memory address
- All tags fields are checked in parallel
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory

Associative Mapping Address Structure

- 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



Fully Associative Cache Mapping Example

Assuming the same:

- 4 GB DRAM addressed then by n=32 bits.
- 32 KB of cache
- Blocks of 64 bytes. Then the offset is w=6 bits.
- **Number of cache lines is then 32 KB / 64 = 512.**
- **The line index is then k=9 bits (= log2(512)).**
- The TAG field is then n-w = 26 bits.

Memory Address	Tag (26-bits)	Offset (6-bits)

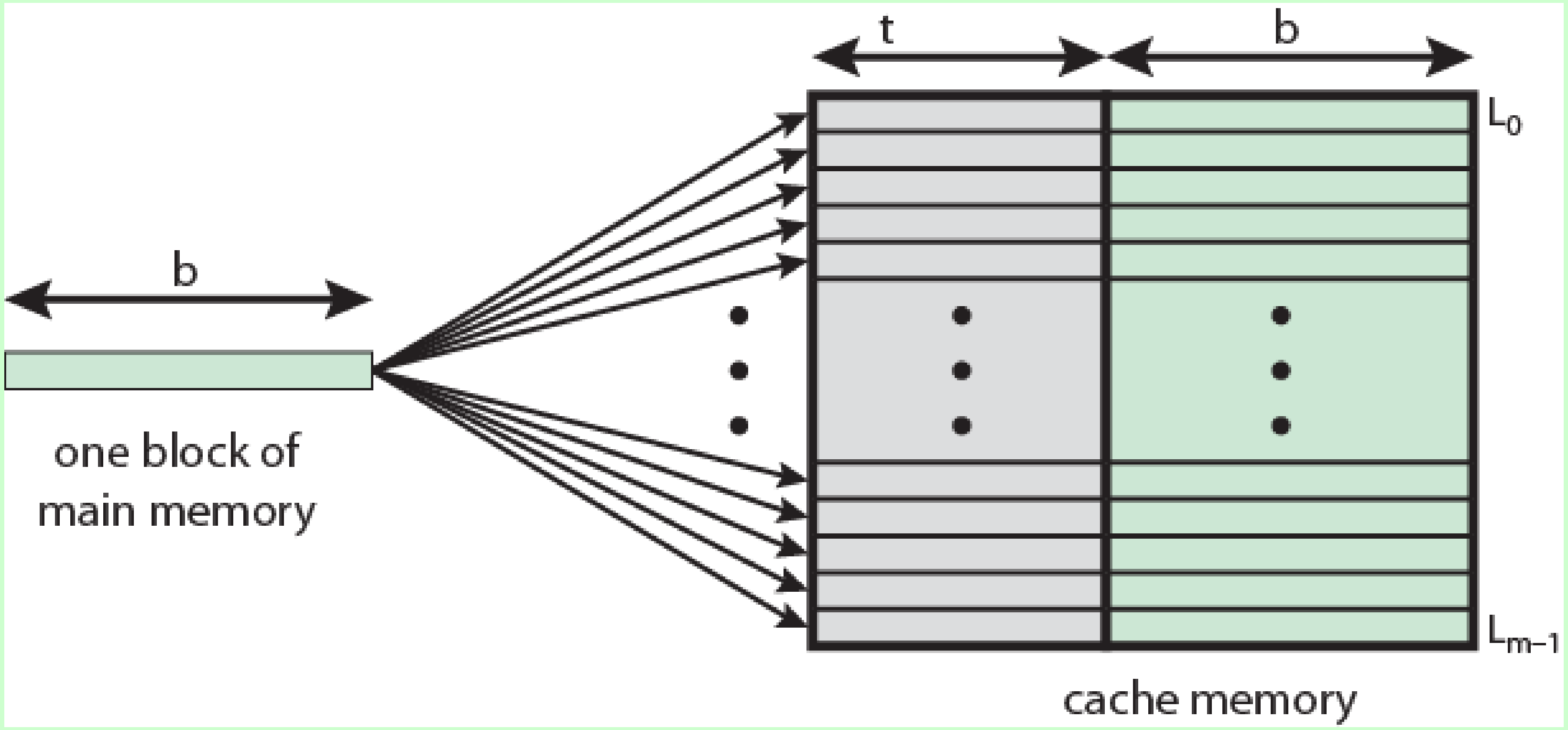
With a memory address of 01001101011101110101101101 101001

- Compare **all tag fields** in cache for the value 01001101011101110101101101.
- If a match is found, return byte 101001 of the line

Memory address

01001101011101110101101101	101001
TAG (26)	OFFSET (6)

Associative Mapping from Cache to Main Memory



The disadvantage of direct mapping is by permitting each main memory block to be loaded into any line of the cache.

Set Associative Mapping

- Set associative cache mapping is a mixture between direct and fully associative mapping
- Cache lines are divided into a number of sets
- Each set contains a number of lines that can vary from Z = 2 to 16 (Z way associative mapping)
- A portion of the address is used to specify which SET will hold an address
- The data can be stored in any of the lines in the SET.
- When the processor wants an address, it indexes to the SET and then searches the tag fields of all lines in the SET for the desired address.

Set Associative Mapping Address Structure

- 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^{d}
- 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
- m= $\log_2$(S=number of sets) bits to address the SETs
- and w = $\log_2$(block size) bits for byte offset

TAG	SET	Word
		(w)

Set Associative Mapping Example

- Assuming the same:
  - 4 GB DRAM addressed then by n=32 bits.
  - 32 KB of cache
  - Blocks of 64 bytes. Then the offset is w=6 bits.
  - Number of cache lines is then 32 KB / 64 = 512.
  - For (Z=**4)-way set associative**, the # of SETs is S=512/4=128
  - SET bits in memory address = m=log2(128) = 7.
  - The TAG field is then n-m-w = 19 bits.

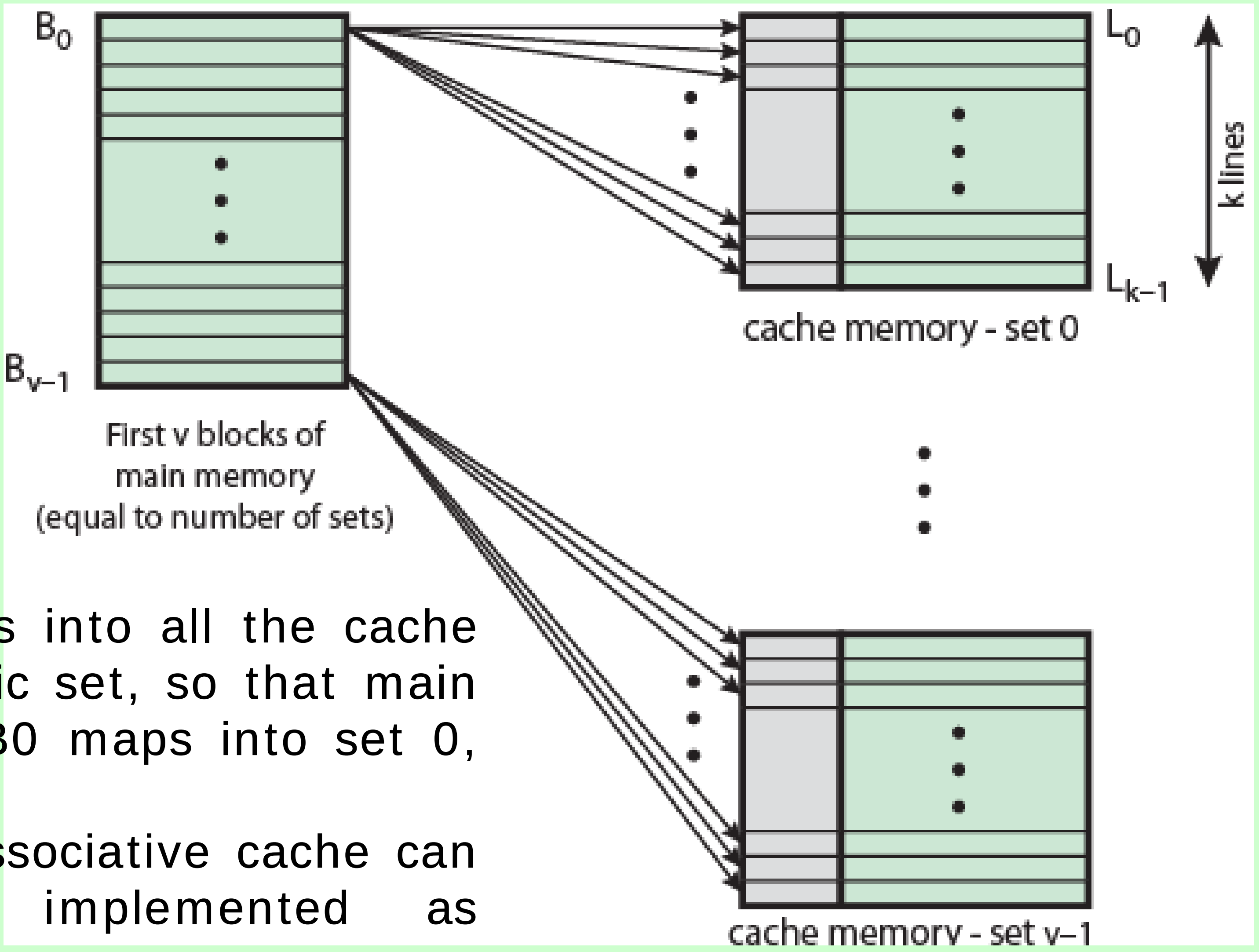
TAG	SET	OFFSET
(n-m-w=19)	(m=7)	(w=6)

- With a memory address of 0100110101110111010 1101101 101001
  - Compare **all tag fields** of lines 110110100 to 110110111 (as we have Z=4-way) with the value 0100110101110111010.
  - If a match is found, return byte 101001 of the corresponding line.

Memory address

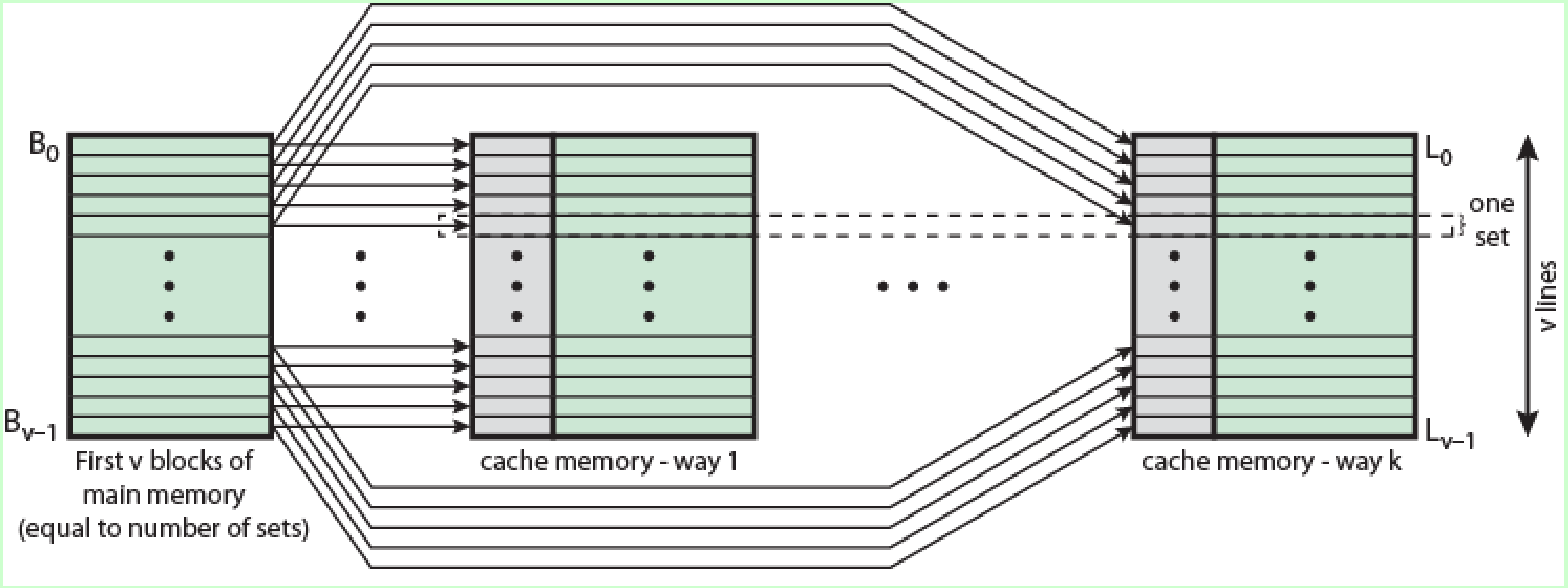
0100110101110111010	1101101	101001
TAG (19)	SET (m=7)	OFFSET (w=6)

Mapping From Main Memory to Cache: **v** Associative



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.

Mapping From Main Memory to Cache: k-way Associative



Each direct-mapped cache is referred to as a *way*, consisting of v lines. The first v lines of main memory are direct mapped into the v lines of each way; the next group of v lines of main memory are similarly mapped, and so on.

Replacement Algorithms

- When a cache miss occurs, data is copied into some location in cache.
- With Set Associative or Fully Associative mapping, the system must decide where to put the data and what values will be replaced.
- Cache performance depends greatly on the way of replacing data.

Replacement Algorithms

- **Direct mapping**
  - Each block only maps to one line
  - Replace that line
- **Associative & Set Associative mapping:**
  - Hardware implemented algorithm (speed)
  - Least Recently used (LRU)
  - e.g. in 2 way set associative
    - Which of the 2 block is LRU?
  - First in first out (FIFO)
    - replace block that has been in cache longest
  - Least frequently used
    - replace block which has had fewest hits
  - Random

LRU and Pseudo LRU Replacement

LRU Replacement

- LRU is easy to implement for 2-way set associative.
- You only need one bit per set to indicate which line in the set was most recently used.
- LRU is difficult to implement for larger ways.
    - For an N-way mapping, there are N! different permutations of use orders.
    - It would require $\log_2(N!)$ bits to keep track, so the **Solution** is Pseudo LRU Algorithm.

Pseudo LRU Replacement

- Pseudo LRU is frequently used in set associative mapping with more than 2-ways.
- In pseudo LRU there is a bit for each half of a group indicating which have was most recently used.
- For 4 way set associative, one bit indicates that the upper two or lower two was most recently used. For each half another bit specifies which of the two lines was most recently used.

Write Policy

When a write operation is performed on a cache, two mechanisms can be adopted:

- —Write through
- —Write back

Write through:

The information is written to both the block in the cache and to the block in the main memory.

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic => Slows down writes
- Remember bogus write through caches!

Write Policy

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.

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced,
  - write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

Line Size

- Retrieve not only desired word but a number of adjacent words as well
- Increased block size will increase hit ratio at first
  - the principle of locality
- Hit ratio will decreases as block becomes even bigger
  - Probability of using newly fetched information becomes less than probability of reusing replaced
- Larger blocks
  - Reduce number of blocks that fit in cache
  - Data overwritten shortly after being fetched
  - Each additional word is less local so less likely to be needed
- No definitive optimum value has been found
- 8 to 64 bytes seems reasonable
- For HPC systems, 64- and 128-byte most common

Multilevel Caches

- High logic density enables caches on chip
  - Faster than bus access
  - Frees bus for other transfers

- Common to use both on and off chip cache
  - L1 on chip, L2 off chip in static RAM
  - L2 access much faster than DRAM or ROM
  - L2 often uses separate data path
  - L2 may now be on chip
  - Resulting in L3 cache
    - Bus access or now on chip…

Cache Coherency

If more than one CPU is present w/a cache to each, & one shared memory

- If data in one cache is altered => data in other caches & in shared memory is invalid
- Maintaining cache coherency is done through:
- ▪ Bus watching w/ write through:
  - Every master on bus monitors address lines
  - If a master writes to a shared memory location that is also in cache
  - Cache controller invalidates cache entry
- ▪ Hardware transparency: all updates to main memory are reflected in all caches
- ▪ Non cacheable memory:
  - Only a portion of main memory is shared by more than 1CPU
  - Shared memory is designated as noncacheable
  - Shared memory is never copied into cache
  - All accesses to shared memory are cache misses

Unified v Split Caches

- One cache for data and instructions or two, one for data and one for instructions

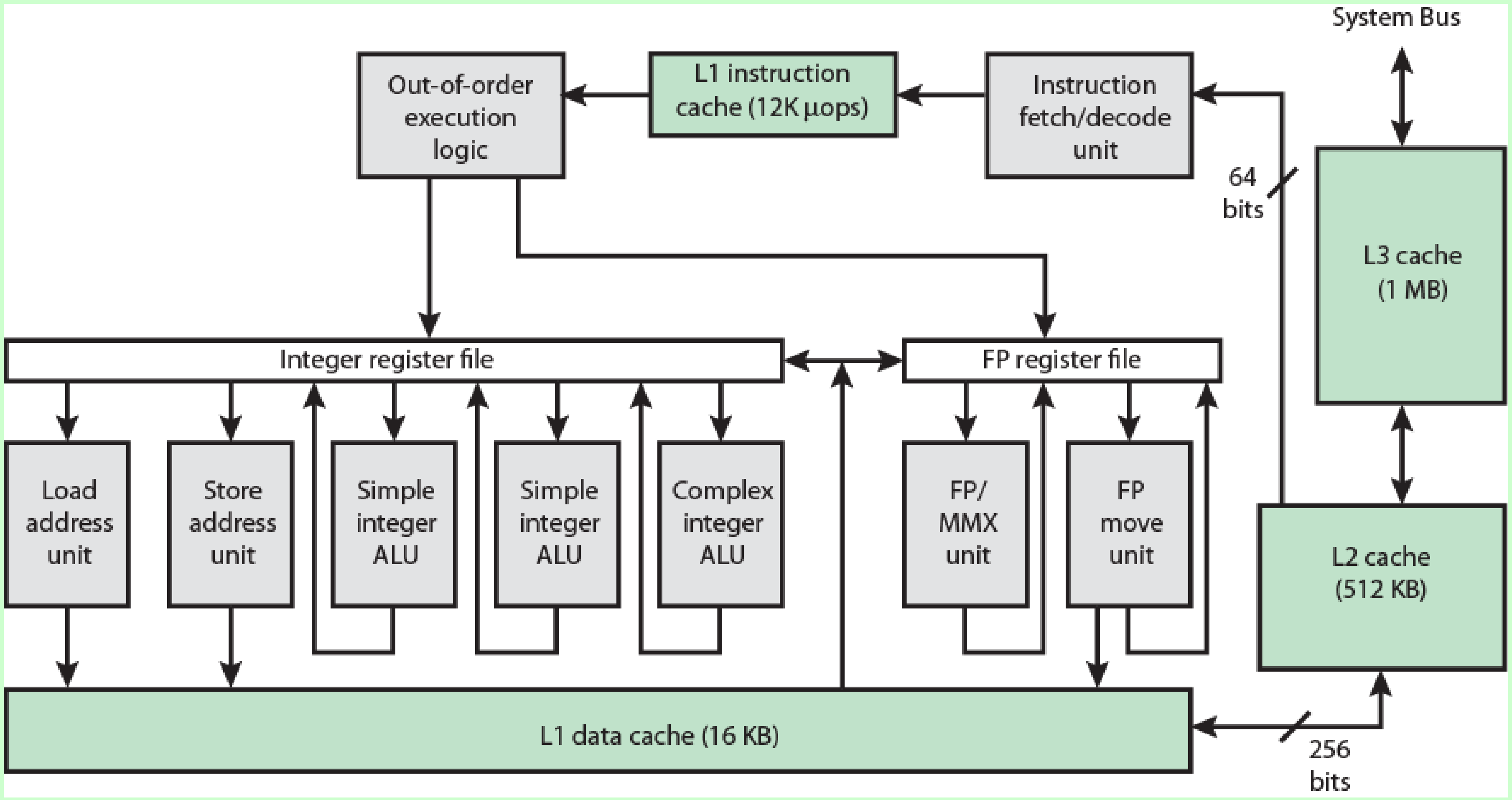
- Advantages of unified cache
  - Higher hit rate
    - Balances load of instruction and data fetch
    - Only one cache to design & implement

- Advantages of split cache
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

Pentium 4 Cache

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four way set associative organization
- Pentium (all versions) – two on chip L1 caches
  - Data & instructions
- Pentium III – L3 cache added off chip
- Pentium 4
  - L1 caches
    - 8k bytes
    - 64 byte lines
    - four way set associative
  - L2 cache
    - Feeding both L1 caches
    - 256k
    - 128 byte lines
    - 8 way set associative
  - L3 cache on chip

Pentium 4 Block Diagram



Pentium 4 Core Processor

- Fetch/Decode Unit
  - —Fetches instructions from L2 cache
  - —Decode into micro-ops
  - —Store micro-ops in L1 cache
- Out of order execution logic
  - —Schedules micro-ops
  - —Based on data dependence and resources
  - —May speculatively execute
- Execution units
  - —Execute micro-ops
  - —Data from L1 cache
  - —Results in registers
- Memory subsystem
  - —L2 cache and systems bus

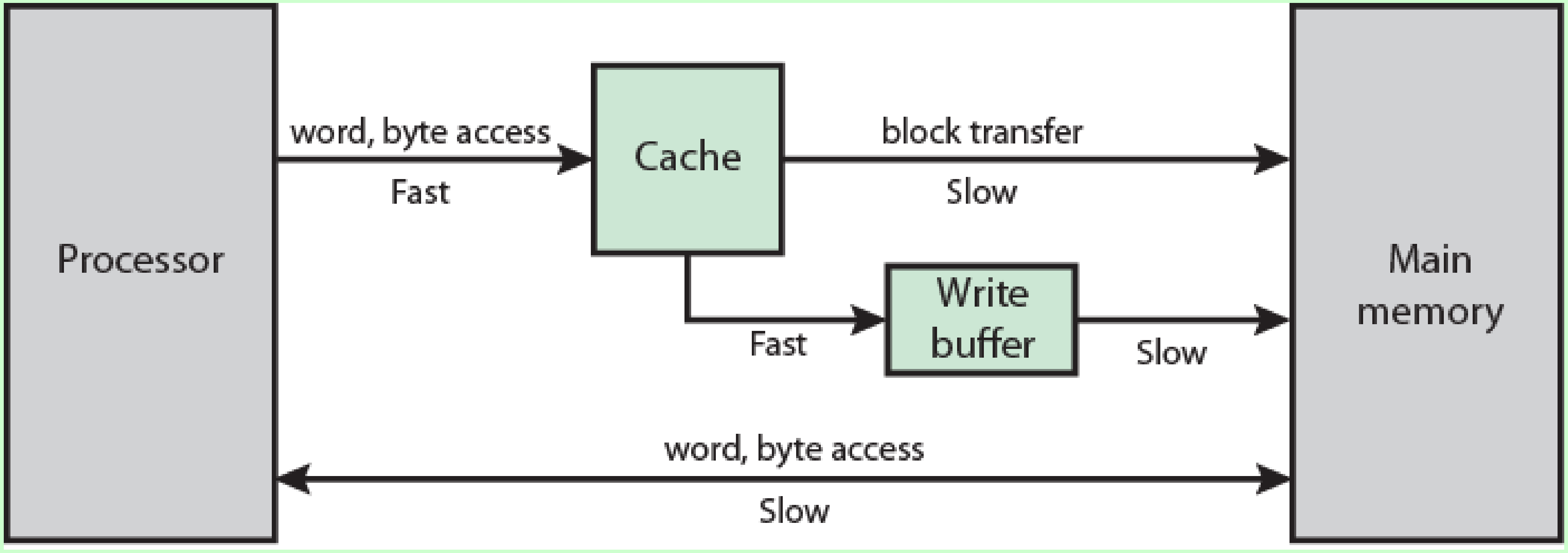
Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length
  - Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining
  - (More later – ch14)
- Data cache is write back
  - Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate
- L2 and L3 8-way set-associative
  - Line size 128 bytes

ARM Cache Organization

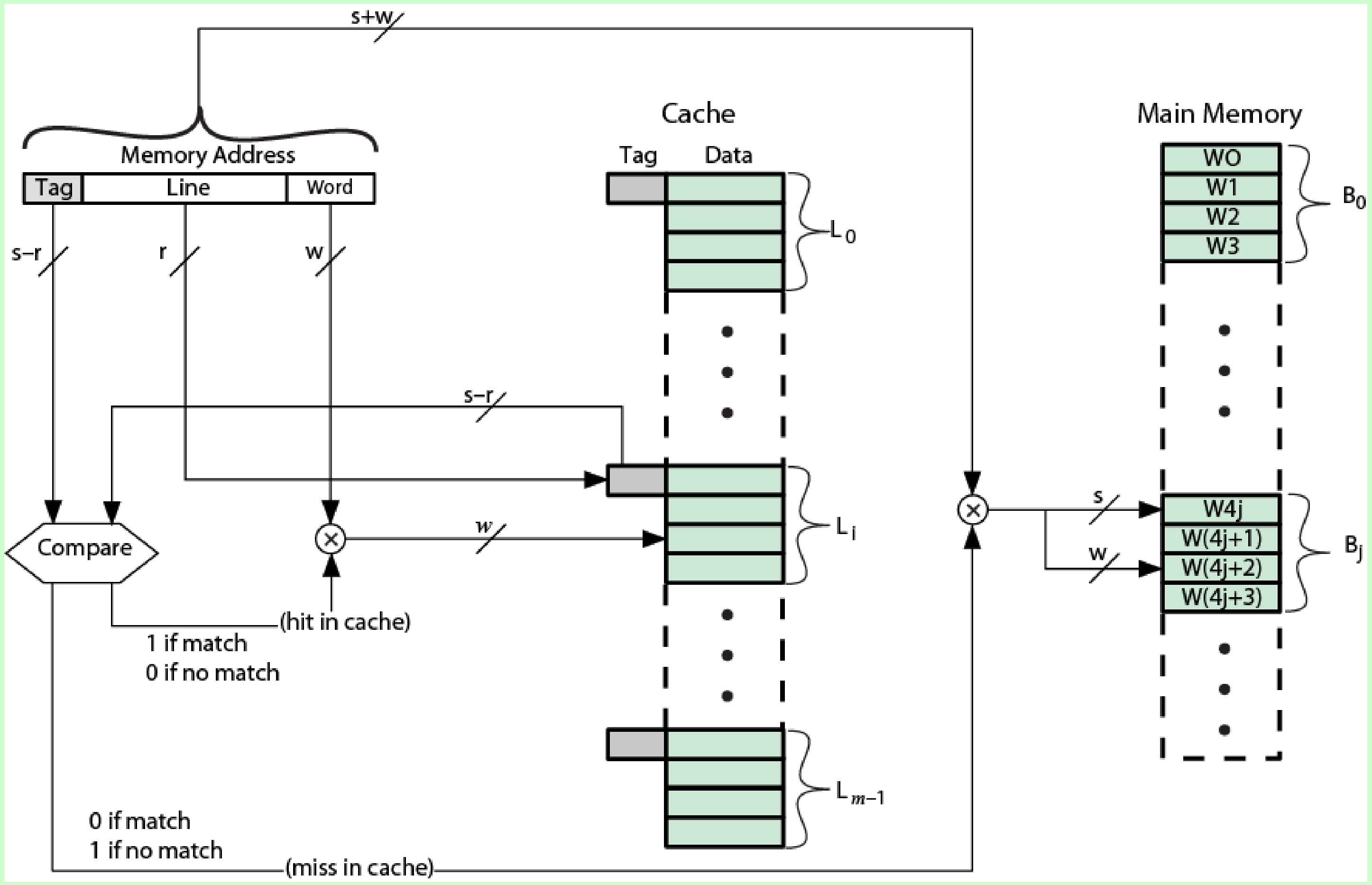
- Small FIFO write buffer
  - Enhances memory write performance
  - Between cache and main memory
  - Small c.f. cache
  - Data put in write buffer at processor clock speed
  - Processor continues execution
  - External write in parallel until empty
  - If buffer full, processor stalls
  - Data in write buffer not available until written
    - So keep buffer small

ARM Cache and Write Buffer Organization

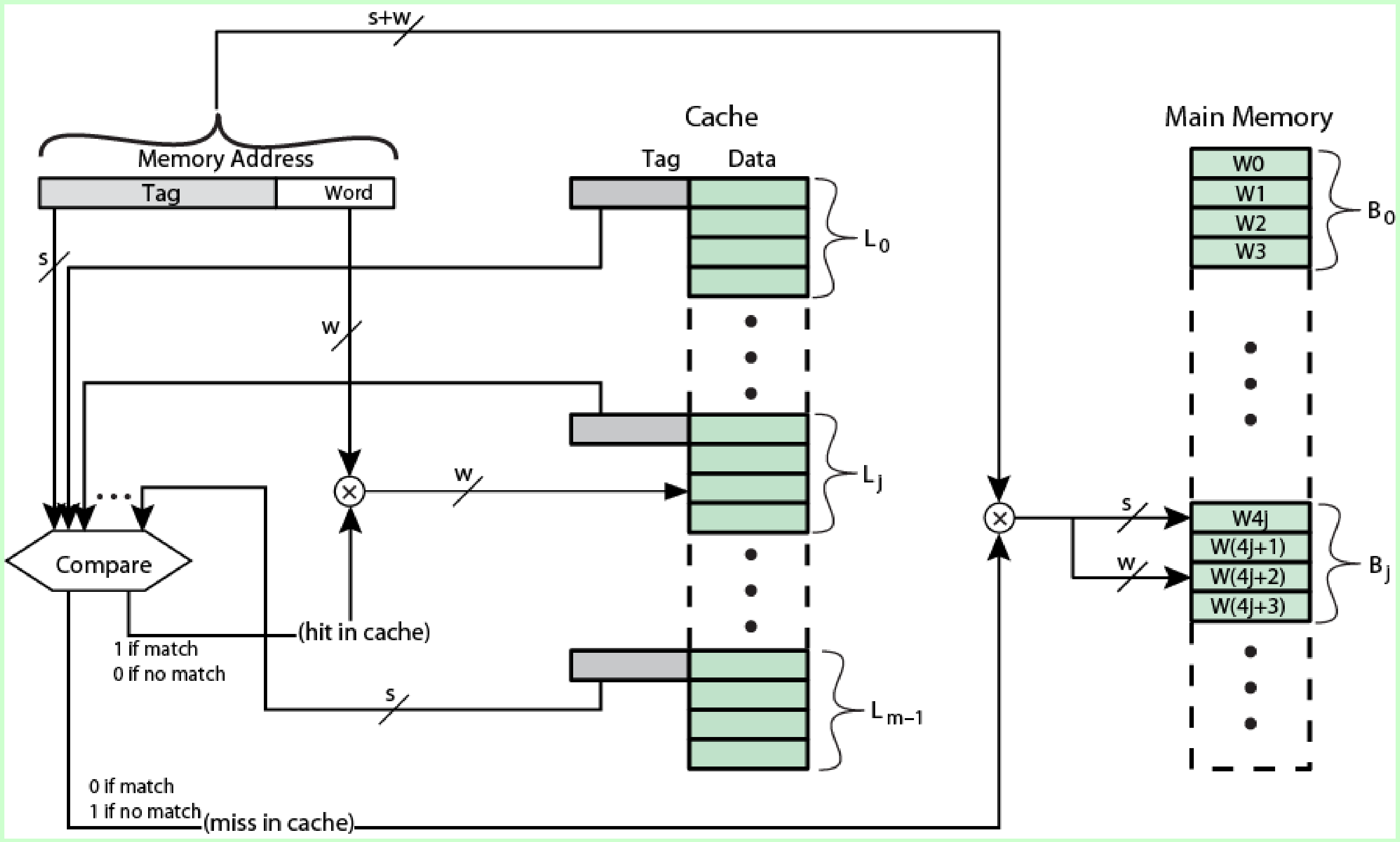


Direct Mapping Cache Organization

- Each main memory address can be viewed as consisting of three fields.
- The w bits identify a unique word or byte within a block of main memory
- The remaining s bits specify one of the 2^s blocks of main memory
- The cache logic interprets these s bits as a tag of s-r bits and a line field of r bits. This field identifies one of the $m=2^r$ lines of the cache

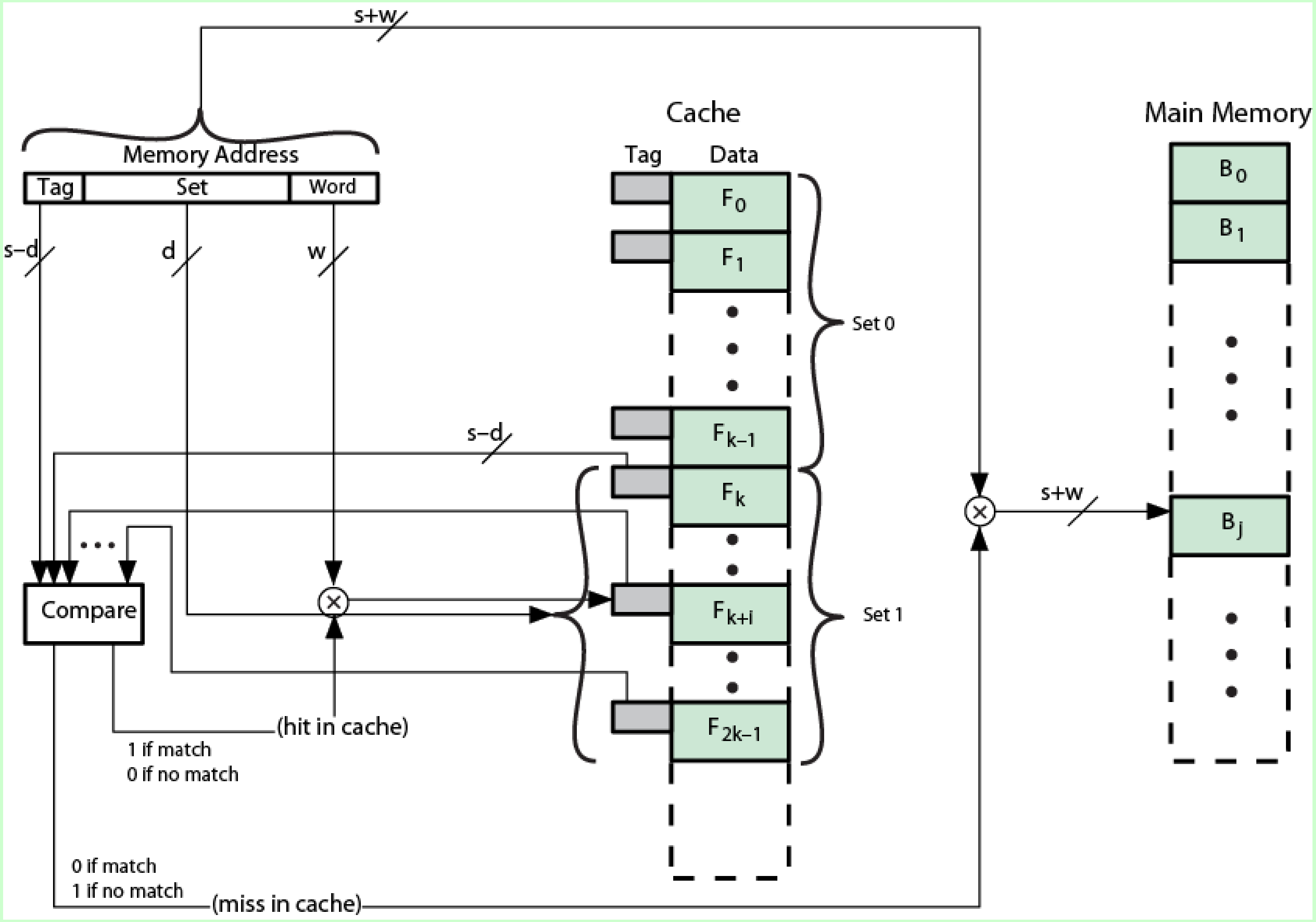


Fully Associative Cache Organization



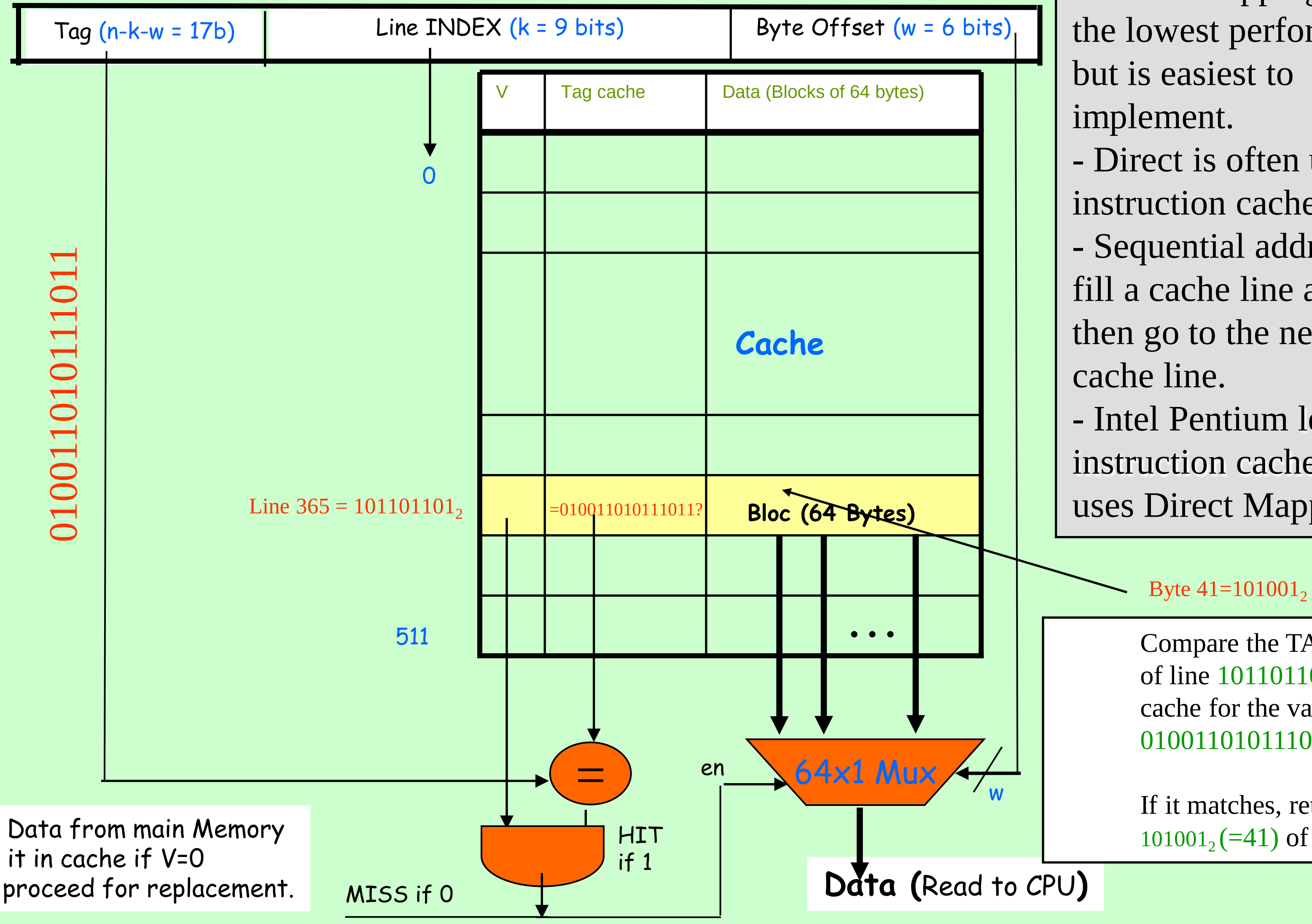
the cache control logic interprets a memory address simply as a Tag and a Word field. The Tag field uniquely identifies a block of main memory. The cache control logic must simultaneously examine every line's tag for a match. Note that no field in the address corresponds to the line #, so that the # of lines in the cache is not determined by the address format.

K-Way Set Associative Cache Organization



Direct Cache Mapping Example

Memory Address (n=32 bits) = 01001101011101 1 10101101101 101001



- Direct Mapping has the lowest performance, but is easiest to implement.
- Direct is often used for instruction cache.
- Sequential addresses fill a cache line and then go to the next cache line.
- Intel Pentium level 1 instruction cache uses Direct Mapping.

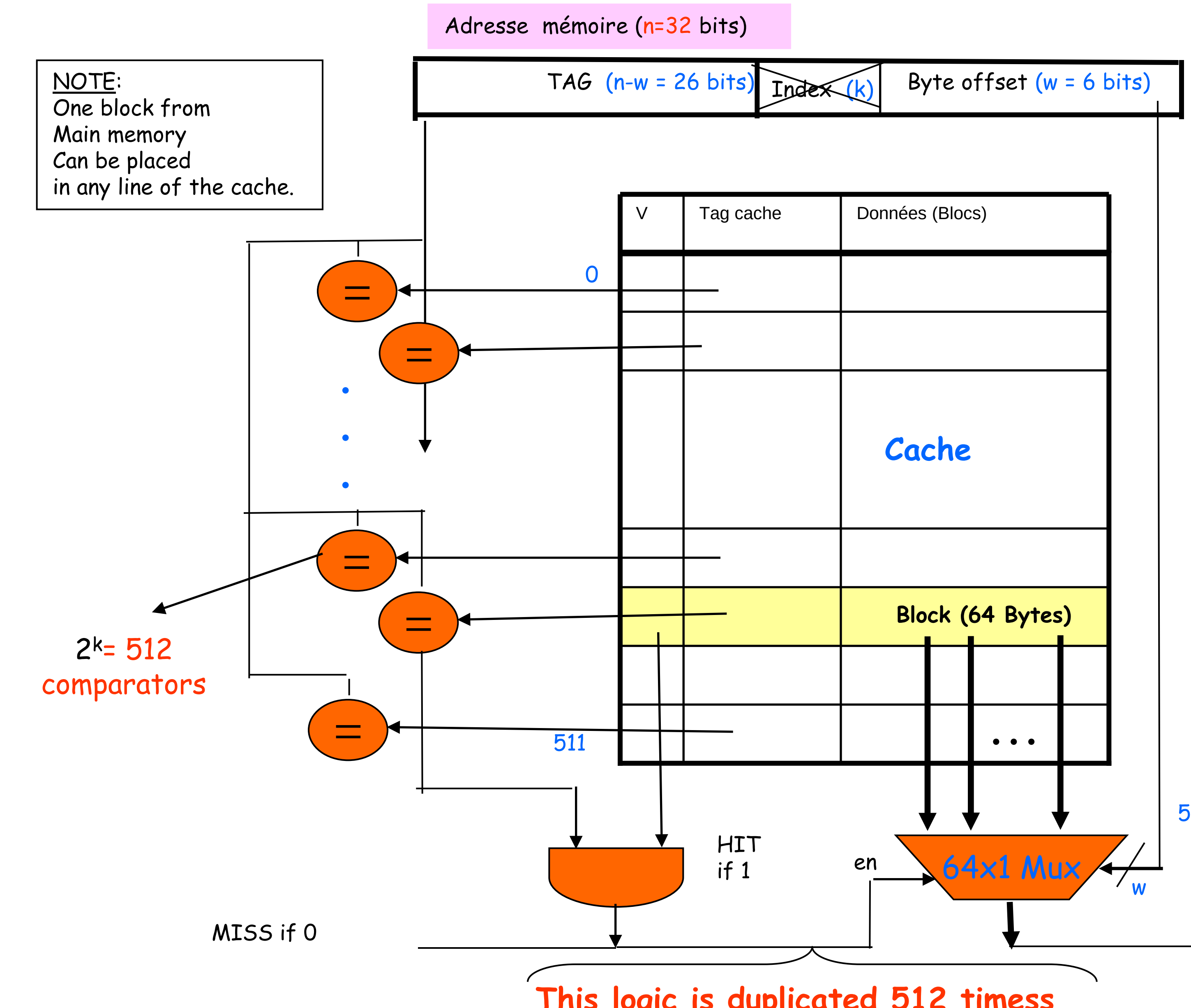
Compare the TAG field of line 101101101 in cache for the value 01001101011101110.

If it matches, return byte 101001_2 (=41) of the line.

CPU reads Data from main Memory and places it in cache if V=0 otherwise proceed for replacement.

Fully Associative Cache Mapping Example

NOTE:
One block from
Main memory
Can be placed
in any line of the cache.



- Fully Associative mapping works the best, but is complex to implement.

-Each tag line requires circuitry to compare the desired address with the tag field.