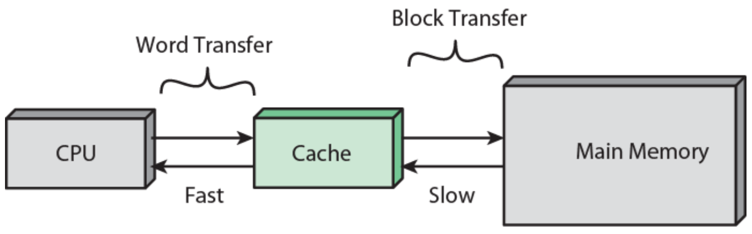
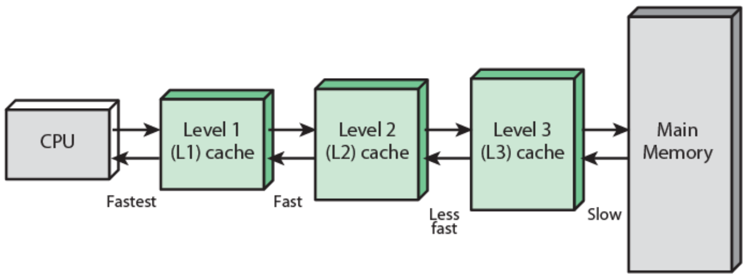
Cache Memory



(a) Single cache

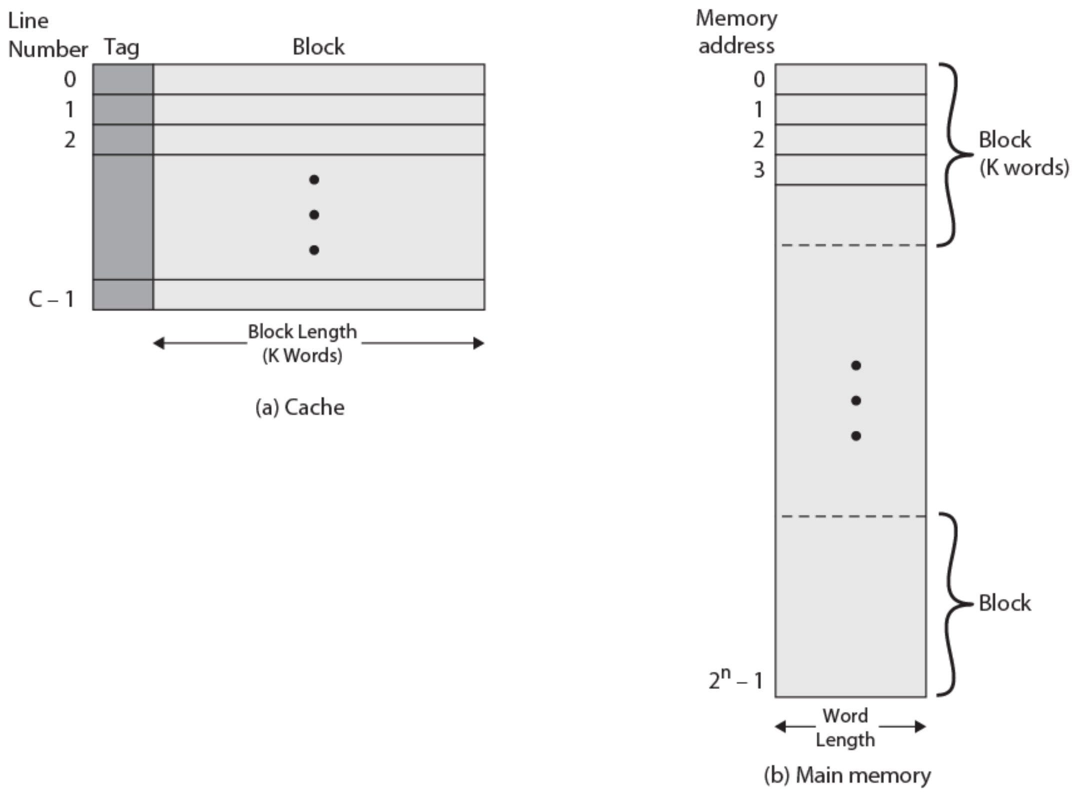


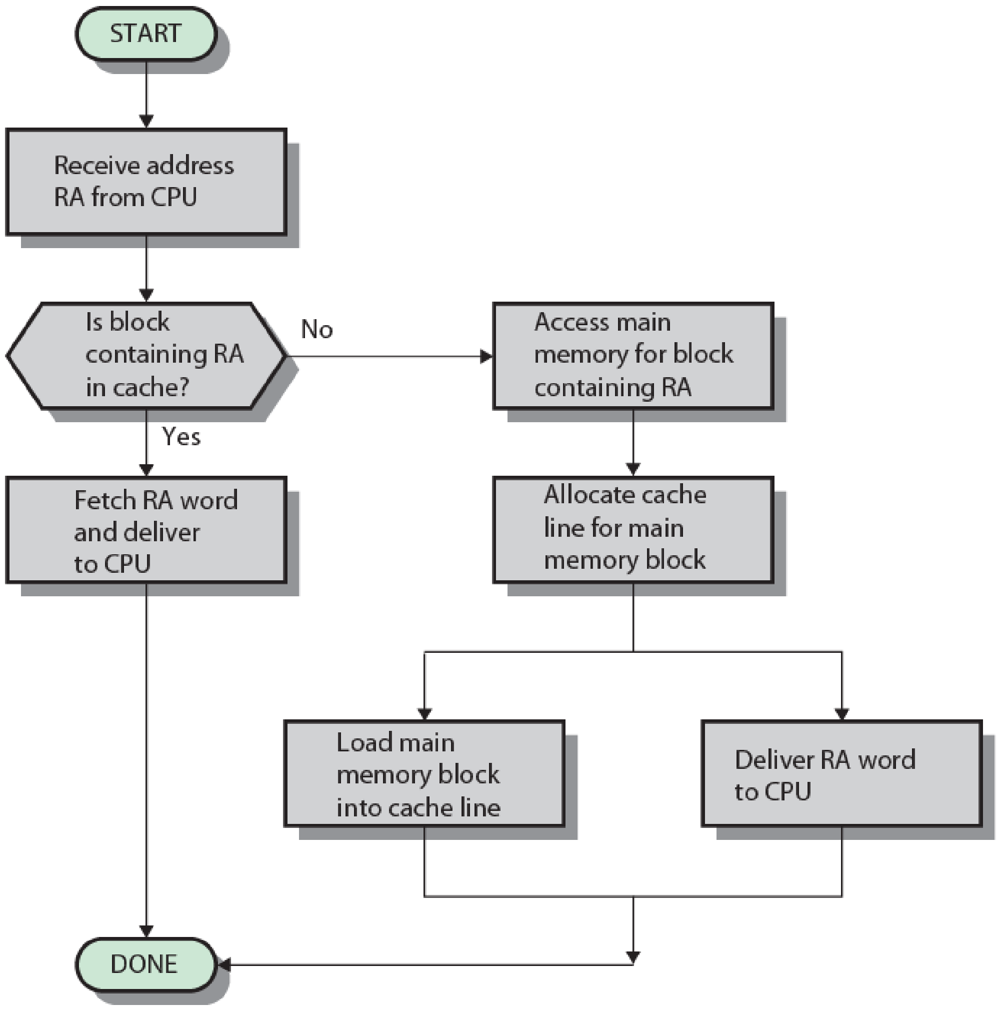
(b) Three-level cache organization

- Study of large program reveal that most of the execution time is spend, in the execution of a few routines (sub-sections). When the execution is localized within these routines, a number of instructions are executed repeatedly, This property of programs is known as LOCALITY OF REFERENCE.
- Thus while some localized area of the program are executed repeatedly, the other areas are executed less frequently.
- To reduce the execution time these most repeated segments may be placed in a fast memory known as CACHE (or Buffer) Memory.
- The memory control circuitry is designed to take advantage of the property of LOCALITY OF REFERENCE.
- If a word in a block of memory is read, that block is transferred to one of the slots of the cache.

Cache Operation:

- CPU requests contents of memory location
- 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


Line Number
Tag
Block
0
1
2
C – 1
Block Length (K Words)
(a) Cache
Memory address
0
1
2
3
Block (K words)
Block
$2^n - 1$
Word Length
(b) Main memory


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

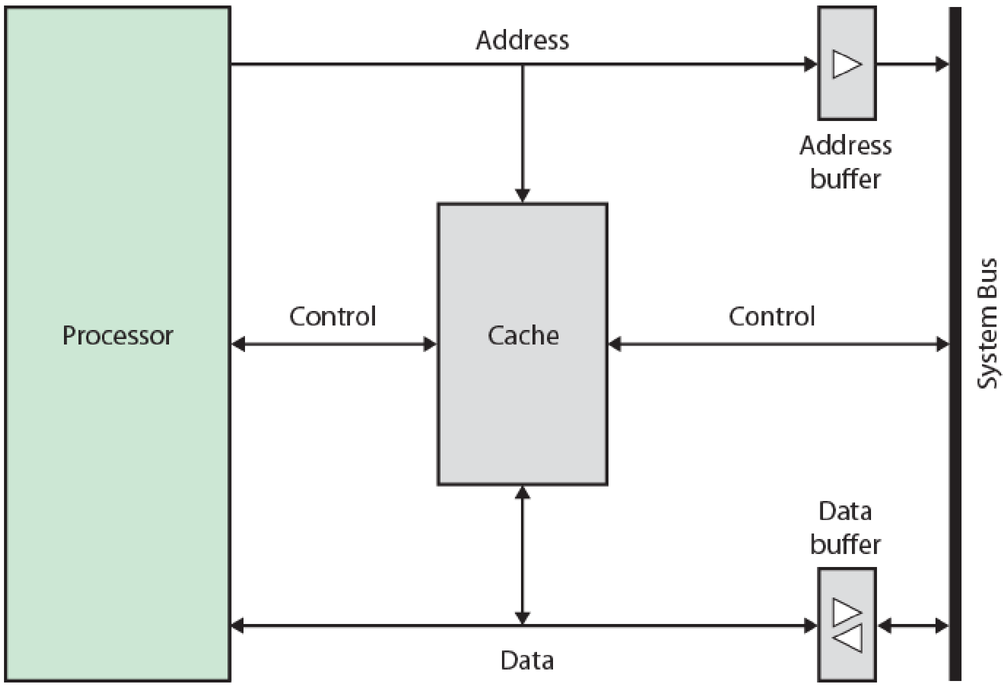


Fig: Cache Organization

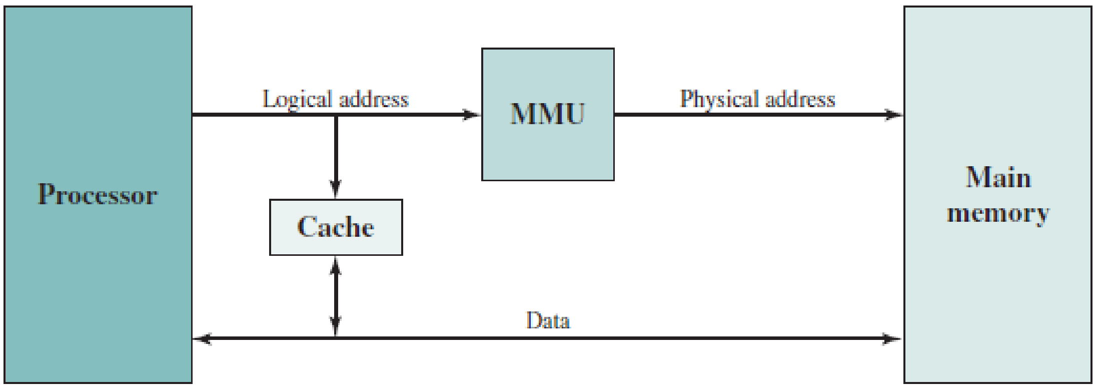
Elements of Cache Design:
Cache Size
Write Policy
Replacement Algorithm
Mapping Function
Block Size / Line Size

<u>CACHE SIZE:</u>

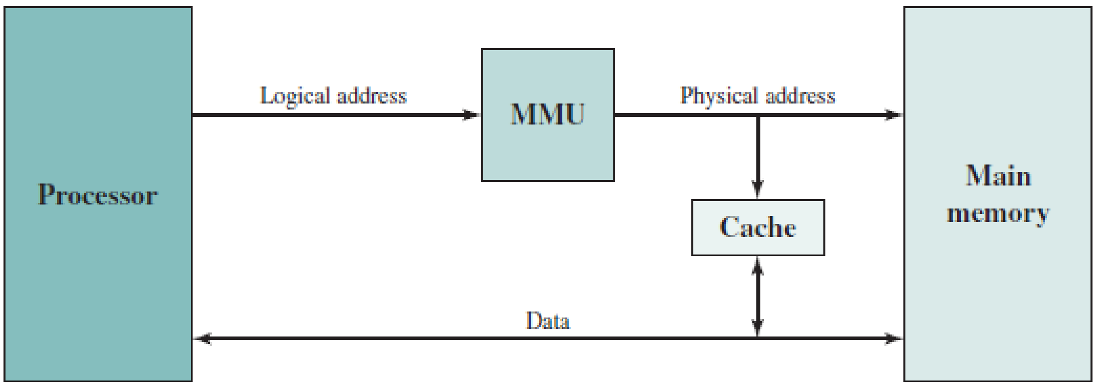
- Size of cache should be <u>small enough</u> so that average cost per bit is close to that of main memory and <u>large enough</u> so that average access time is close to that of a cache alone.
- The larger the cache, the larger the number of gates involved in addressing the cache, so large cache tend to be slightly slower than smaller ones.
- In a system with **n** address lines, Main memory consists of **2^n** addressable words.
- Cache is divided in to **m lines** of **k words** size
- Main memory will consists of **M** = $2^n/k$ blocks
- Since number of lines is considerably less than the number of main memory blocks
  m << M
- An individual line cannot uniquely and permanently dedicated to a particular block.
- Tag of few bits and some control bits are also associated with each line of cache.
- Each line include a tag that identifies which particular block is currently being stored in a particular cache line.
- Tag is usually a portion of main memory address.

<u>CACHE ADDRESS (LOGICAL OR PHYSICAL) :</u>

- Virtual Memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of Main Memory physically available.
- When virtual memory is used address field of Machine Instruction contain Virtual Addresses.
- A hardware Memory Management Unit (MMU) translates each Virtual Address into a Physical Address.
- With Virtual Address, system designer may choose to place the Cache between the Processor and the MMU or between the MMU and Main Memory.
- A <u>Logical Cache</u> (Virtual Cache) stores data using virtual addresses.
- The processor accesses the cache directly without going through the MMU.
- A <u>Physical Cache</u> stores data using main memory physical addresses.
- In logical cache, cache access speed is faster than for a physical cache but usually same virtual address in two different applications refer to two different physical addresses.
- Cache must completely flushed with each application context switch, or extra bits must be added to each line to identify which virtual address space this address refers to.



(a) Logical cache



(b) Physical cache

WRITE POLICY

For a Write Operation to a word in Cache:

Write Back Policy:

- Update only the cache & mark the location with a flag (dirty bit or use bit)
- When the block is removed then if dirty bit is set, the corresponding location in Main Memory is updated.

Write Through Policy:

- Main Memory and Cache are updated simultaneously.
- Main Memory is always updated (Valid).

For a Write Operation where Word is not in Cache:

- Perform write operations in Main Memory

- In a bus organization with more than one device having their separate Caches but shared memory the task of keeping memory valid become complex.
- A system that keep all Caches and Main Memory valid is said to maintain Cache Coherency.
- Techniques for keeping Cache Coherency are :
- Bus Watching With Write Through in which each cache controller monitor the address lines for other controller for write operation.
- Hardware transparency, where additional hardware is used to keep all cache and memory valid.
- Non Cacheable Memory where only a portion of Main Memory is shared by more than one processor and this is shared memory is designed to be non cacheable.

REPLACEMENT ALGORITHM

- When Cache is full, a decision is to be made about which block should be removed to make room for the other block from the main memory, Such a decision can potentially affect the system performance.
- The Locality of Reference provide a clue to reasonable strategy.
- As sub-program stay for reasonable periods of time, it can be assume that a block which has recently been referenced will also be referenced in near future.
- Thus the block that has stayed for long without being referenced should be removed. Such a block is known as LEAST RECENTLY USED Block and the algorithm to determine such a block is known as LRU Algorithm.
- A counter is used to keep track of LRU Block in Cache. For a 4 slots cache 2 bit counter is used, for a 8 slots cache 3 bits are required.

For a Read Operation:

In case of HIT (Content in Cache):

- Counter for the block referenced is set to '0'
- All other counters with value originally lower than referenced one are incremented by 1, while all others remain unchanged.

In case of MISS (Content not in Cache)

If Cache is Not FULL:

- The counter associated with the new block loaded form Main Memory is reset to '0'.
- Values of all other counters are incremented by 1.

If Cache is FULL:

- The block with the largest count is removed.
- New block is put in its place.
- Its counter is reset to ‘0’
- The remaining counters are incremented by 1

Other Algorithms are Least Frequently Used (LFU), First In First Out (FIFO) and Random (selecting line to be replaced at random).

Based on LOR chances are there that the block referenced most frequently is referenced next and the block which is less frequently used may not be referenced again so it can be removed (**Least Frequently Used**). It is also implemented by use of a counter. This algorithm is also relatively good. While tests have shown that Random technique is best while FIFO has shown poor performance.

MAPPING FUNCTION:

- Because there are fewer cache lines than main memory blocks an algorithm is needed for mapping main memory blocks into cache lines.
- Also a mean is needed for determining which main memory block currently occupies a cache line.
- Choice of mapping function dictates how the cache is organized.
- Three techniques can be used:
  - Direct
  - Associative
  - Set-Associative

Direct Mapping Function:

In Direct Mapping, each block of main memory can be mapped into only one possible cache line.
Mapping is expressed as

i = j modulo m

Where,

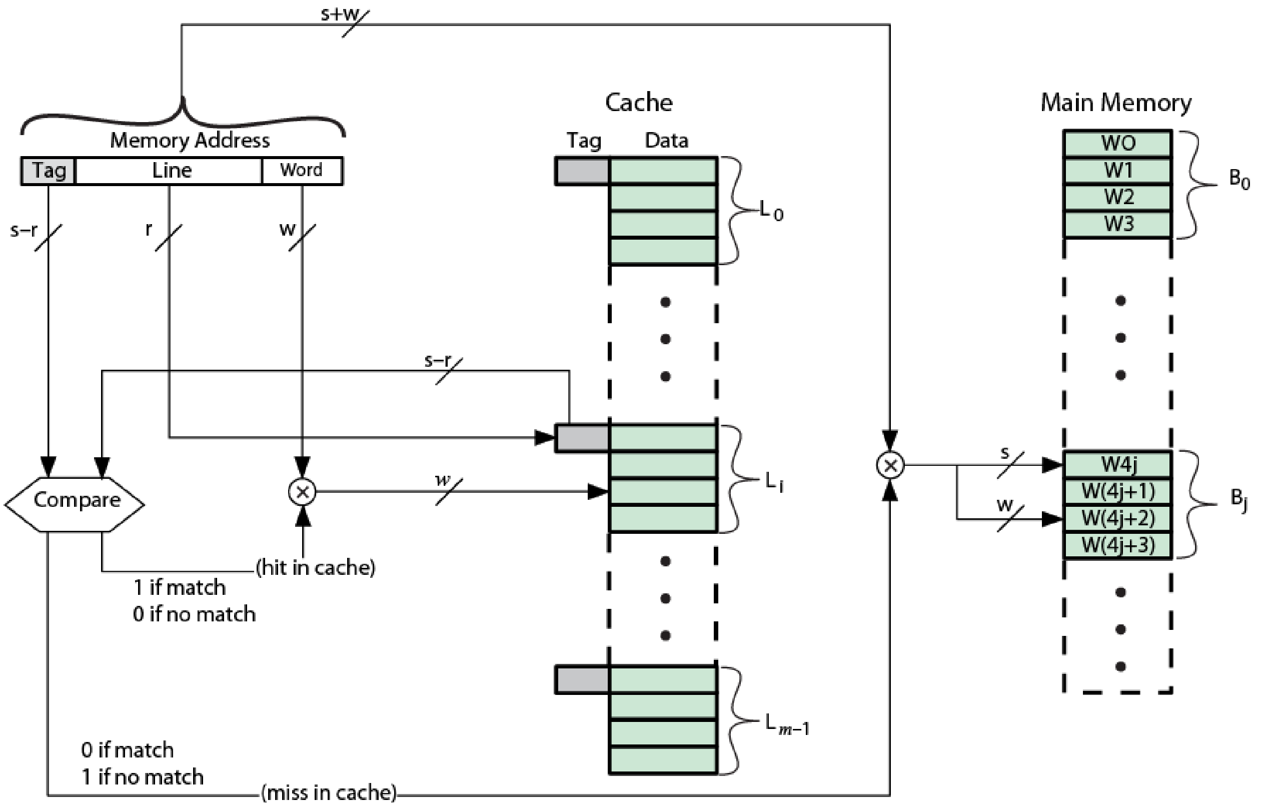
i = Cache Line number
j = Main Memory block number
m= number of lines in the cache

(line i will be allocated to block j of main memory, or block j can be mapped in to line i of the cache)
Mapping is implemented using main memory address:
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 Cache = 2^{r+w}
- Size of tag = (s-r) bits

The effect of direct mapping is that blocks of Main Memory are assigned to lines of the cache as follows

Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2^s-m
1	1,m+1, 2m+1…2^s-m+1
…	
m-1	m-1, 2m-1,3m-1…2^s-1



– Use of a portion of the address as a line number provide 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 bits **s-r** serve this purpose.
– Direct Mapping is simple and inexpensive to implement.
– Disadvantage: If a program reference words repeatedly from two different blocks that's maps into same line, then the blocks will be continuously swapped and the hit ratio will be low, this continuous swapping is known as **thrashing.**

Victims Cache: A small cache of 4 to 16 cache lines between directly mapped L1 cache and the next level of memory is proposed to hold discarded (thrashed) data.

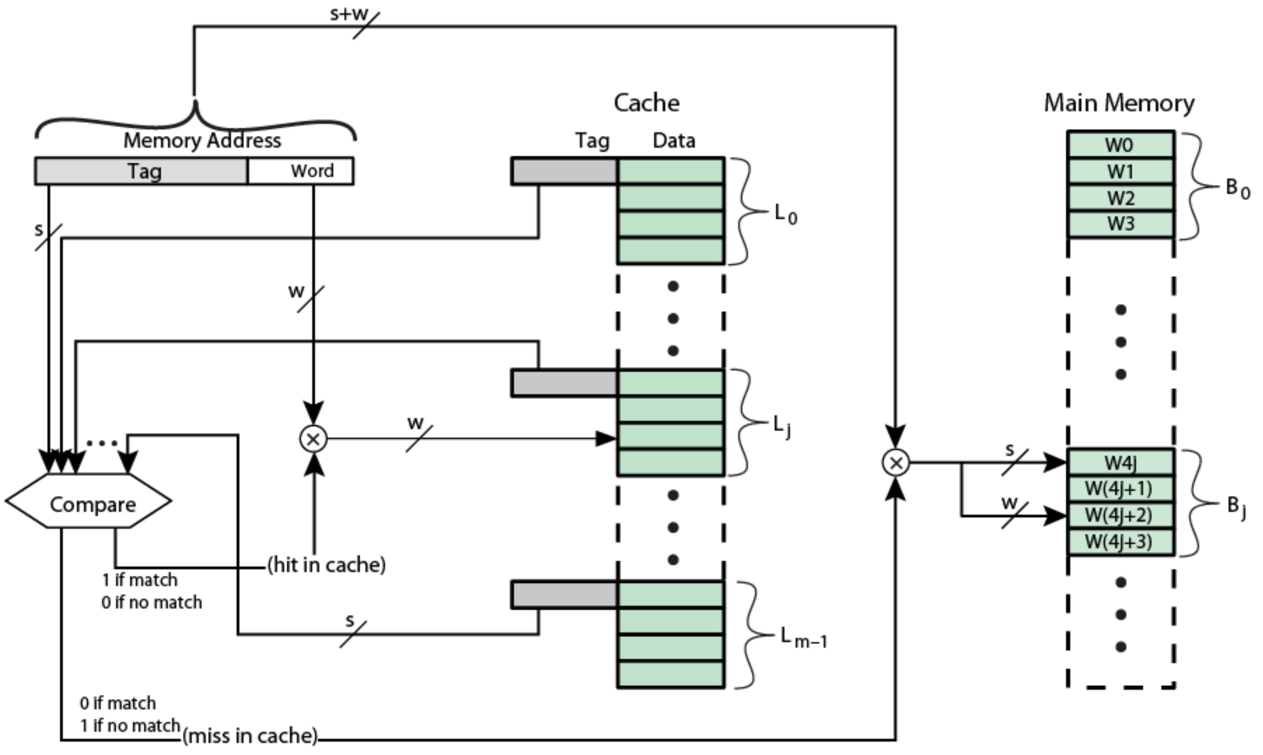
Associative Mapping Function:

- Each Main Memory block can be loaded into any line of the cache.
- Cache Control logic interprets a memory address simply as a Tag and a Word field.
- Tag field uniquely identifies a block of main memory.
- Cache control logic simultaneously examine every line’s tag for a match.

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 = undetermined from the address
- Size of tag = s bits

Disadvantage of Associative Mapping is the complex circuitry required to examine the tags of all cache lines in parallel.



Set-Associative Mapping :

The Cache consists of a number of sets, each consisting a number of lines

m = v x k

i = j 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

This is referred to as **k-way set associative mapping**.

Block B_j can be mapped into any of the lines of set j.

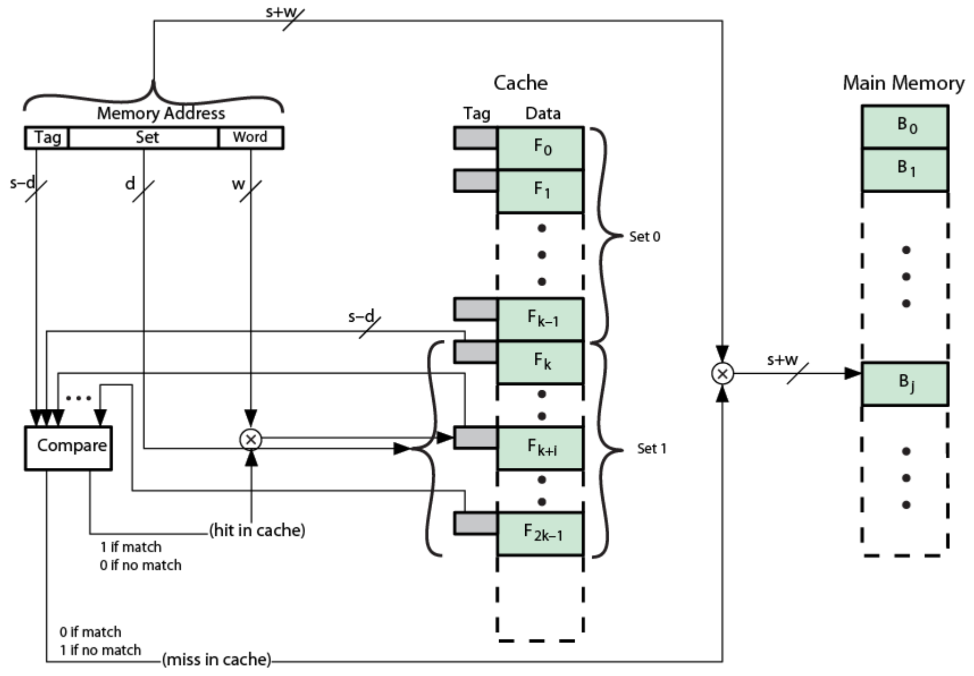
Cache Control logic interprets a memory address as three fields: Tag, Set and Word.

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 = k v = k x 2^d
- Size of Cache = k x 2^{d+w} words or bytes
- Size of tag = (s – d) bits

– s bits (of the tag and set field) specify one of the 2^s blocks of main memory.

– With k – way set associative mapping, the tag in a memory address is smaller (than of associative mapping) and is only compared to the k tags within a single set.



LINE SIZE

- As block size increases from small to large, hit ratio increases because of Principle Of Locality.
- Principle Of Locality: Data in the vicinity of a referenced word are likely to be referenced in the near future.

  Problems:
- Larger blocks reduces the number of blocks that fits into a cache. Small number of blocks result in data being overwritten shortly after they are fetched.
- As a block become larger, each additional word is farther from the requested word and therefore less likely to be needed in near future.
- Relationship between line size and hit ratio is complex and no optimum value has been found.
- 8 to 64 bytes of line size seems optimum in normal computers.
- However in HPC (High Processing Computers) 64 and 128 bytes cache line sizes are frequently used.

Number of Caches:

Level of Cache
Unified versus Split Cache (For Data and Instruction)

Multilevel Cache:

- If there is no L2 (Off chip) Cache and the processor makes an access request for a memory location not in the L1 cache, then processor must access the DRAM or ROM across the bus, resulting in poor performance.
- If an L2 SRAM (Static RAM) Cache is used with speed matching bus speed, than data can be accessed using a zero-wait state.
- Use of multi-level cache complicate all of the design issues related to cache, including size, replacement algorithm and write policy.
- Results show that L2 cache with at least double the size of L1 cache improve hit ratio.

Unified versus Split Cache

Benefits of Unified Cache:
A unified Cache has a higher hit rate than split cache.
Only one cache needs to be designed and implemented.
Benefits of Split Cache:
Split cache is used for parallel instruction execution for prefetching of predicted future instructions.
It eliminate contention for the cache between the instruction fetch / decode unit and the execution unit.

Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 kB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6×32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24×64 kB/ 128 kB	24×1.5 MB	24 MB L3 192 MB L4

Notes:

[a] Two values separated by a slash refer to instruction and data caches.

[b] Both caches are instruction only; no data caches.