Computer Architecture

V semester

Chapter 6: Memory Organization

6 hours

Compiled by :

Ankit Bhattarai, Assistant Professor
Email: ankitbhattarai@cosmoscollege.edu.np

RAID (Redundant Array of Independent Disks)

- RAID is a technology that is used to increase the performance and/or reliability of data storage.
- A RAID system consists of two or more drives working in parallel. These can be hard discs, but there is a trend to also use the technology for SSD (Solid State Drives).
- There are different RAID levels, each optimized for a specific situation.
- These are not standardized by an industry group or standardization committee.

Some of the RAID levels are:

RAID 0 – striping

RAID 1 – mirroring

RAID 3 – striping with parity

RAID 5 – Striped disks with distributed parity

RAID 10 – combining mirroring and striping

RAID (Redundant Array of Independent Disks)

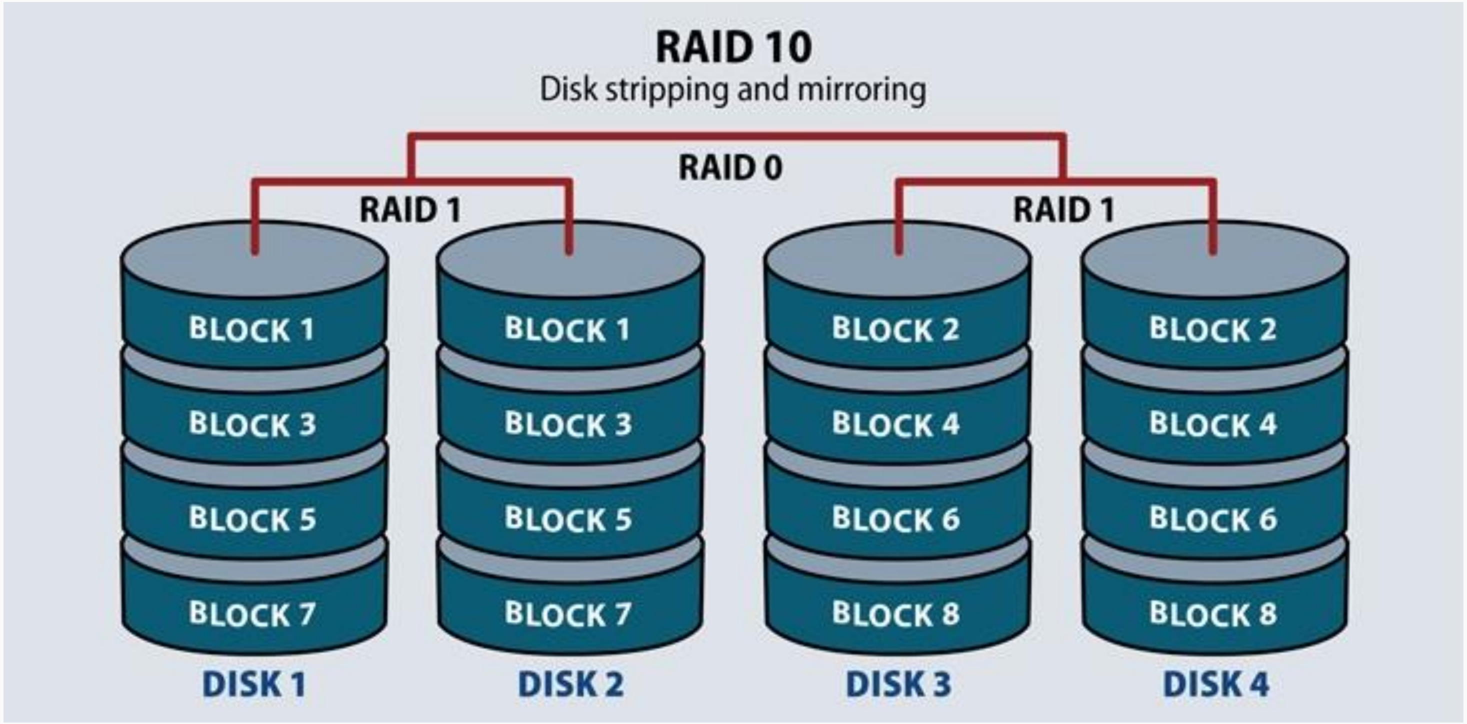
Why RAID ?: RAID is a storage technology that creates a data loss fail-safe by merging two or more hard disk drives (HDDs) or solid-state drives (SSDs) into one cohesive storage unit, or array.

RAID mode	Description	Operation	Advantages	Disadvantages	Recovery
RAID 0	Striped disks	Data is split evenly between two or more disks.	Large size and the fastest speed.	No redundancy.	If one or more drives fails, this results in array failure.
RAID 1	Mirrored disks	Two or more drives have identical data on them.	A single drive failure will not result in data loss.	Speed and size is limited by the slowest and smallest disk.	Only one drive is needed for recovery.

RAID (Redundant Array of Independent Disks)

RAID mode	Description	Operation	Advantages	Disadvantages	Recovery
RAID 3	Striped set with dedicated parity	Data is split evenly between two or more disks, plus a dedicated drive for parity storage.	High speeds for sequential read/write operations.	Poor performance for multiple simultaneous instructions.	A single drive failure will rebuild.
RAID 5	Striped disks with distributed parity	Data is split evenly between three or more disks. Parity is split between disks.	Large size, fast speed, and redundancy.	The total array size is reduced by parity.	A single drive failure will rebuild.
RAID 10	1+0; Striped set of Mirrored Subset	Four or more drives are made into two mirrors that are striped.	Larger size and higher speed than RAID-1, and more redundancy than RAID-0.	No parity.	Only one drive in a mirrored set can fail.

RAID


RAID 10
Disk stripping and mirroring
RAID 0
RAID 1
RAID 1
BLOCK 1
BLOCK 3
BLOCK 5
BLOCK 7
BLOCK 1
BLOCK 3
BLOCK 5
BLOCK 7
BLOCK 2
BLOCK 4
BLOCK 6
BLOCK 8
BLOCK 2
BLOCK 4
BLOCK 6
BLOCK 8
DISK 1
DISK 2
DISK 3
DISK 4

Associative Memory

- An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.
- Associative memory is often referred to as **Content Addressable Memory (CAM)**.
- It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on the address.

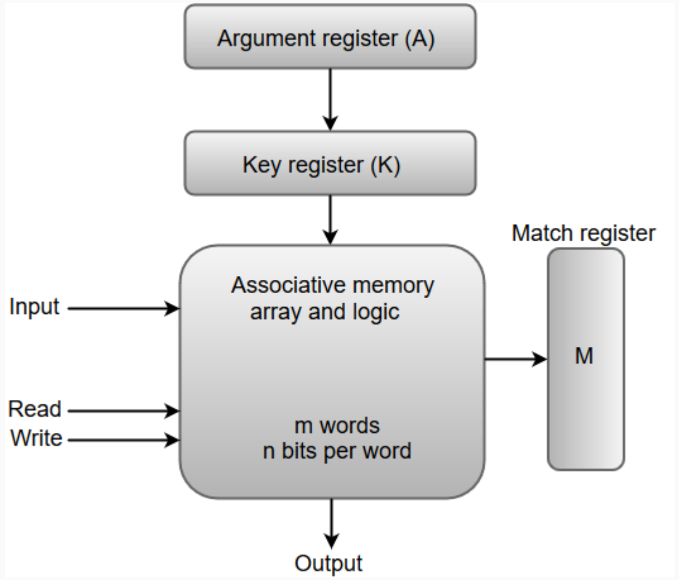


Fig. Hardware Organization

Associative Memory

- From the block diagram, we can say that an associative memory consists of a memory array and logic for 'm' words with 'n' bits per word.
- The functional registers like the argument register A and key register K each have n bits, one for each bit of a word. The match register M consists of m bits, one for each memory word.
- The words which are kept in the memory are compared in parallel with the content of the argument register.

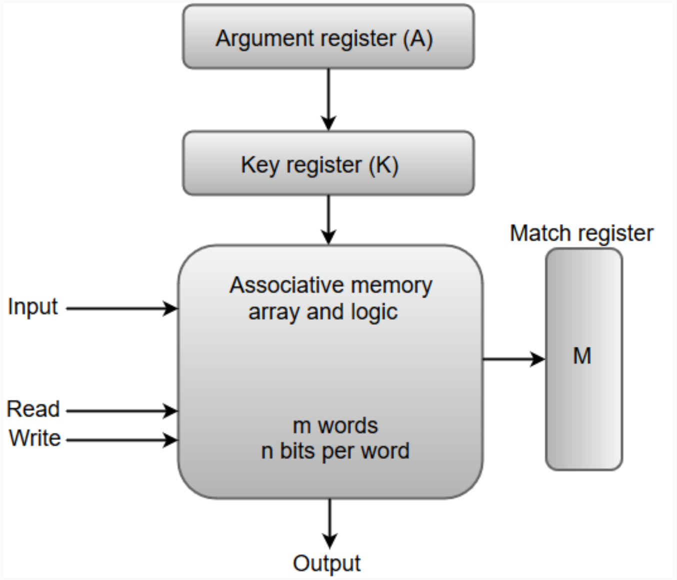


Fig. Hardware Organization

Associative Memory

- The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word.
- Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared.
- Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.

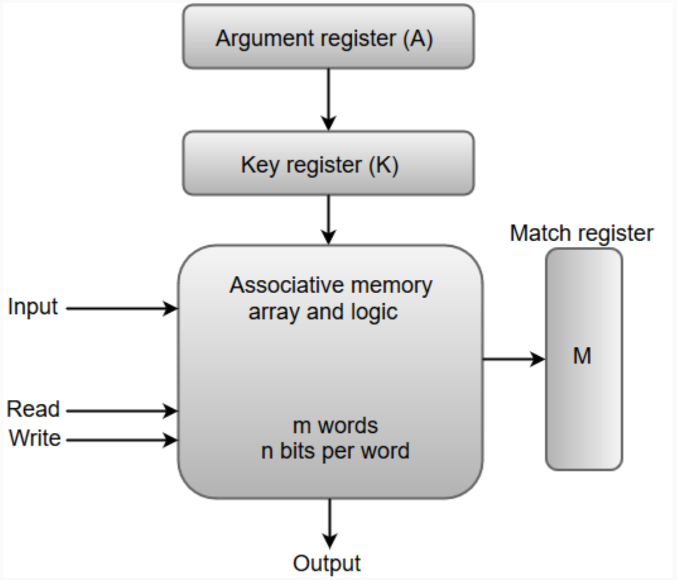
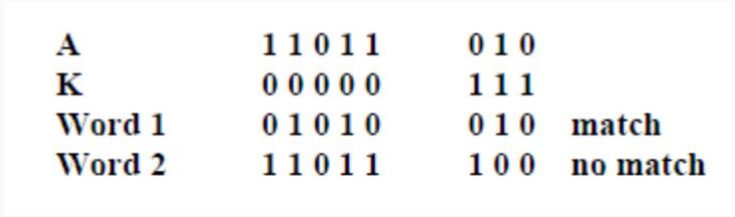


Fig. Hardware Organization

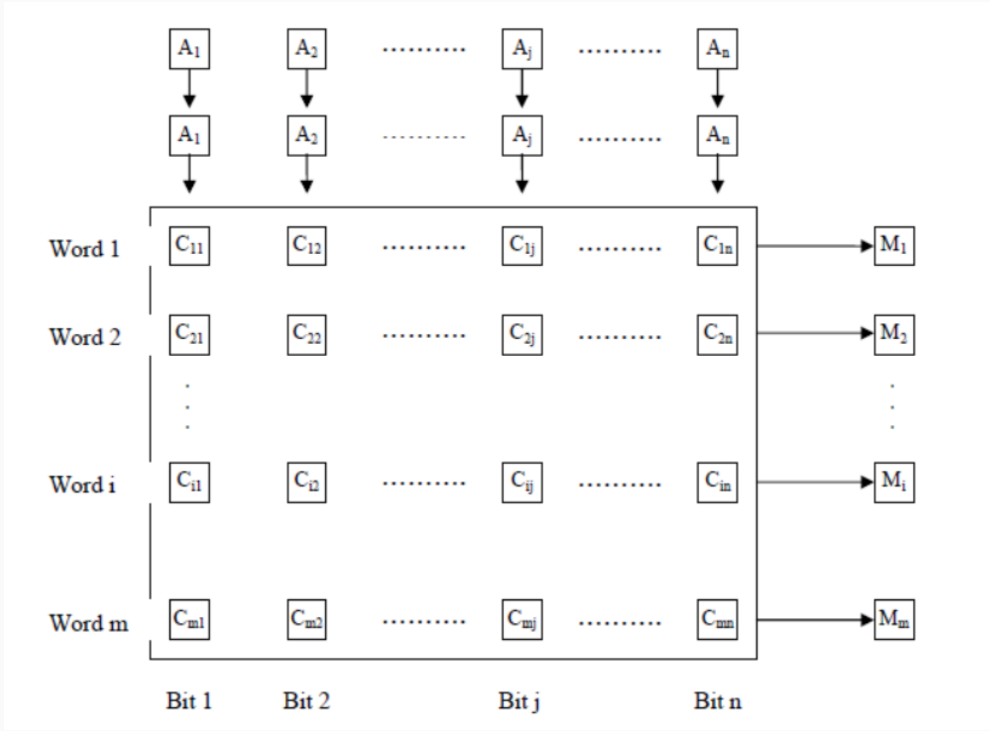
Associative memory : Address Matching Logic

- For example, if an argument register A and the key register K have the bit configuration shown below.



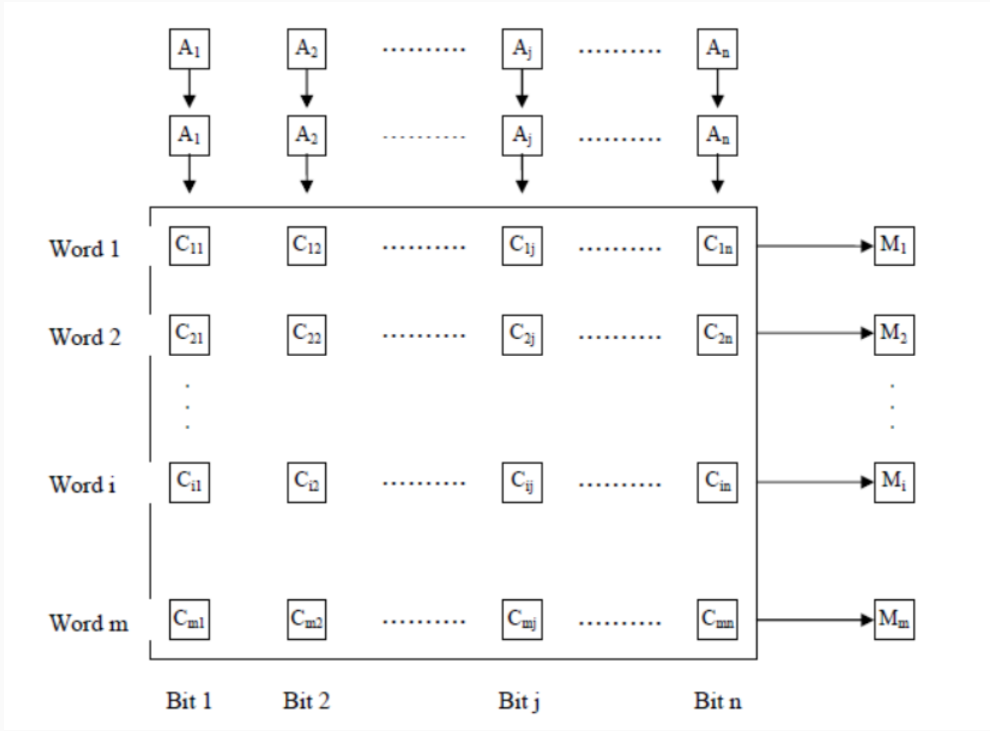
A	1 1 0 1 1	0 1 0	
K	0 0 0 0 0	1 1 1	
Word 1	0 1 0 1 0	0 1 0	match
Word 2	1 1 0 1 1	1 0 0	no match

- Only the three rightmost bits of A are compared with memory words because K has 1's in these positions.



Associative memory : Address Matching Logic

- The cells in the array are considered by the letter C with two subscripts.
- The first subscript provides the word number and the second determines the bit position in the word. Therefore cell C_{mj} is the cell for bit j in word m.
- If a match appears between all the unmasked bits of the argument and the bits in word i, the equivalent bit M_m in the match register is set to 1.


A_1 A_2 A_j A_n
A_1 A_2 A_j A_n
Word 1 C_{11} C_{12} C_{1j} C_{1n} M_1
Word 2 C_{21} C_{22} C_{2j} C_{2n} M_2
Word i C_{i1} C_{i2} C_{ij} C_{in} M_i
Word m C_{m1} C_{m2} C_{mj} C_{mn} M_m
Bit 1
Bit 2
Bit j
Bit n

Associative memory : Read/ Write Operations

- When a **write operation** is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.

- On the other hand, when the **word is to be read** from an associative memory, the content of the word, or part of the word, is specified.
- The words which match the specified content are located by the memory and are marked for reading.

Associative memory : Advantages /Disadvantages

Advantages of Associative memory :-

1. It is used where search time needs to be less or short.
2. It is suitable for parallel searches.
3. It is often used to speedup databases.
4. It is used in page tables used by the virtual memory and used in neural networks.

Disadvantages of Associative memory :-

1. It is more expensive than RAM.
2. Each cell must have storage capability and logical circuits for matching its content with external argument.

Associative memory Vs Cache Memory

	Associative Memory	Cache Memory
1	A memory unit access by content is called associative memory.	A fast and small memory is called cache memory.
2	It reduces the time required to find the item stored in memory.	It reduces the average memory access time.
3	Here data accessed by its content.	Here, data are accessed by its address.
4	It is used where search time is very short.	It is used when particular group of data is accessed repeatedly.
5	Its basic characteristic is its logic circuit for matching its content.	Its basic characteristic is its fast access.

Cache Memory

- Cache memory, also called cache, supplementary memory system that temporarily stores frequently used instructions and data for quicker processing by the central processing unit (CPU) of a computer.
- Cache holds a copy of only the most frequently used information or program codes stored in the main memory.
- It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers.

Cache Memory

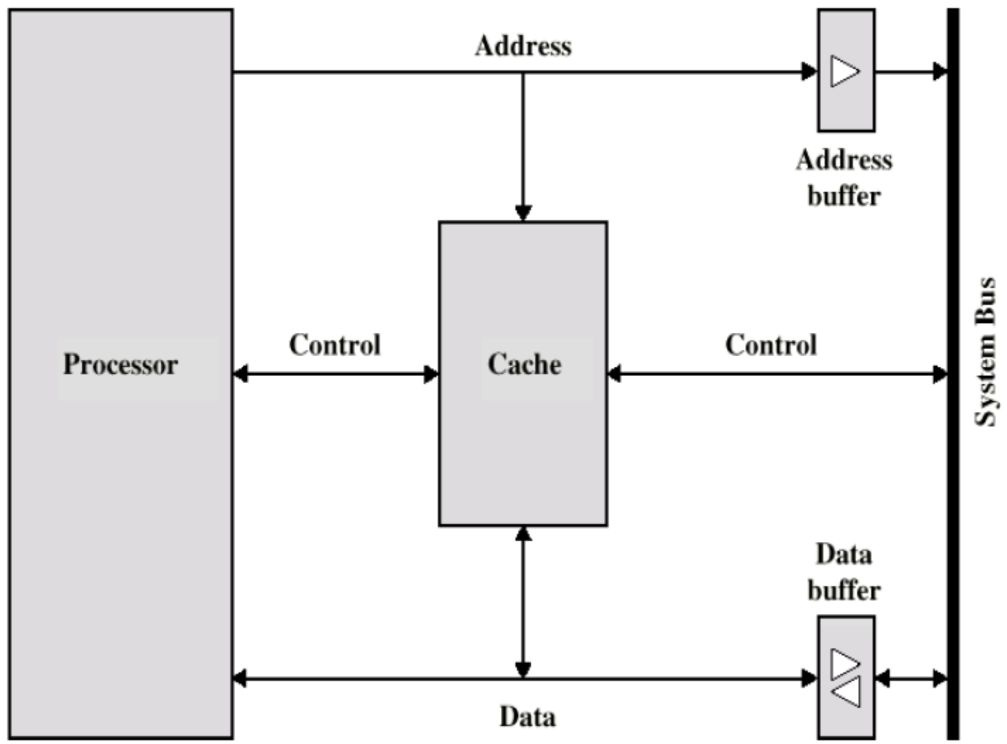


Fig: Typical Cache organization

Cache 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 fixed number of words is read into the cache and then the word is delivered to the processor.
- Cache connects to the processor via data control and address line. The data and address lines also attached to data and address buffer which attached to a system bus from which main memory is reached.

Cache Memory

- When a cache hit occurs, the data and address buffers are disabled and the communication is only between processor and cache with no system bus traffic.
- When a cache miss occurs, the desired word is first read into the cache and then transferred from cache to processor.
- For later case, the cache is physically interposed between the processor and main memory for all data, address and control lines.

Cache Memory

- When a cache hit occurs, the data and address buffers are disabled and the communication is only between processor and cache with no system bus traffic.
- When a cache miss occurs, the desired word is first read into the cache and then transferred from cache to processor.
- For later case, the cache is physically interposed between the processor and main memory for all data, address and control lines.

Cache Memory

Cache Operation Overview

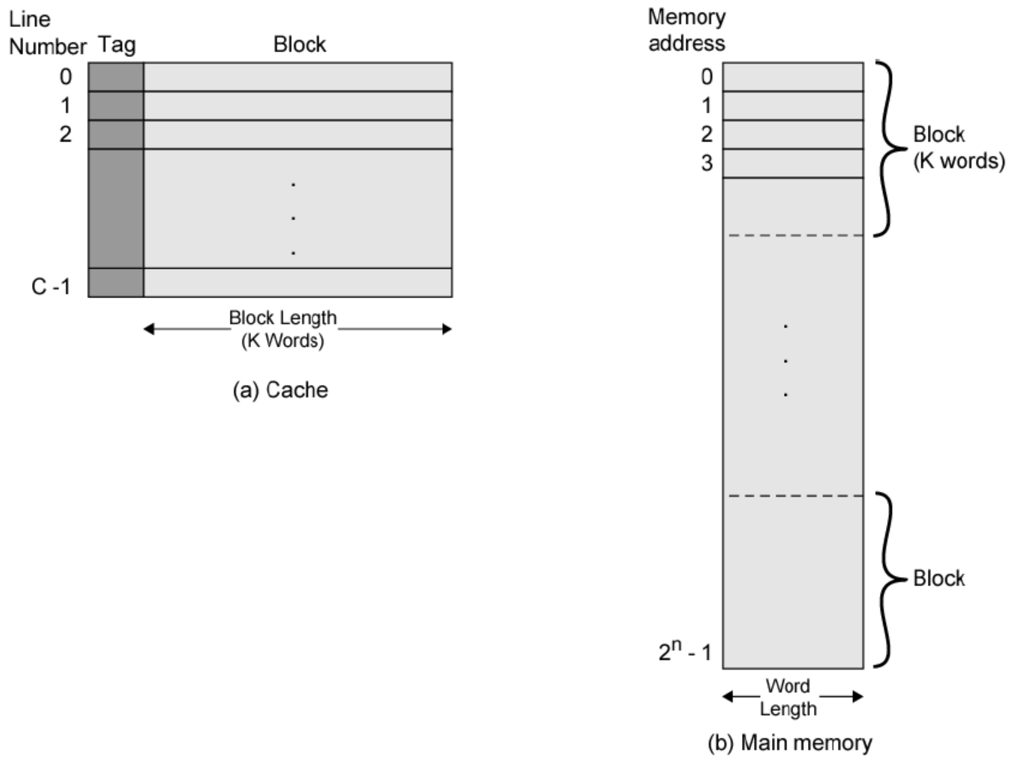


Fig: Cache memory / Main memory structure

Cache Memory

- CPU generates the receive address (RA) of a word to be moved (read).
- Check a block containing RA is in cache.
- If present, get from cache (fast) and return.
- If not present, access and read required block from main memory to cache.
- Allocate cache line for this new found block.
- Load block for cache and deliver word to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Memory

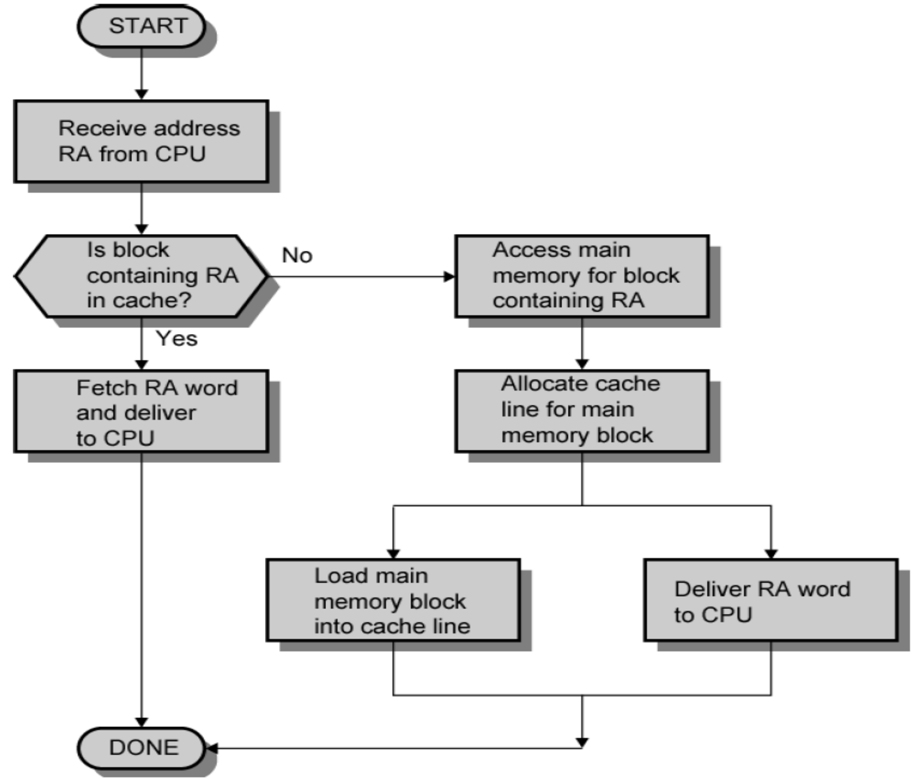

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Fig: Flowchart for cache read operation

Cache Mapping

- Cache mapping defines how a block from the main memory is mapped to the cache memory in case of a cache miss.
- In other words, Cache mapping is a technique by which the contents of main memory are brought into the cache memory.
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.

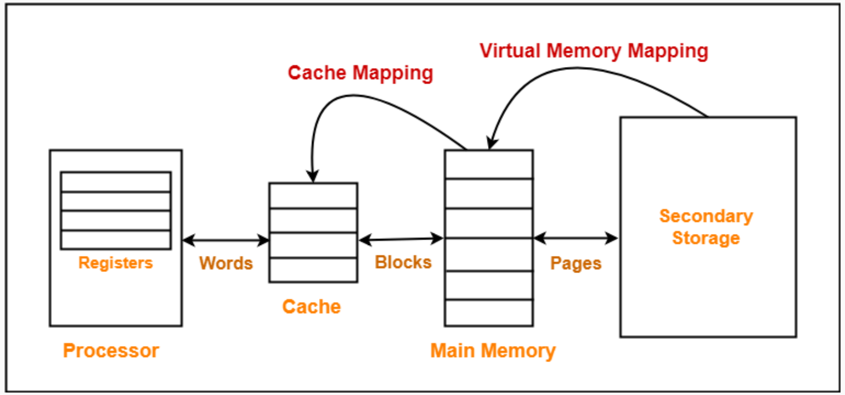


Figure. Illustrates the mapping process

Cache Mapping

When **cache hit** occurs,

- The required word is present in the cache memory.
- The required word is delivered to the CPU from the cache memory.

When **cache miss** occurs,

- The required word is not present in the cache memory.
- The page containing the required word has to be mapped from the main memory.

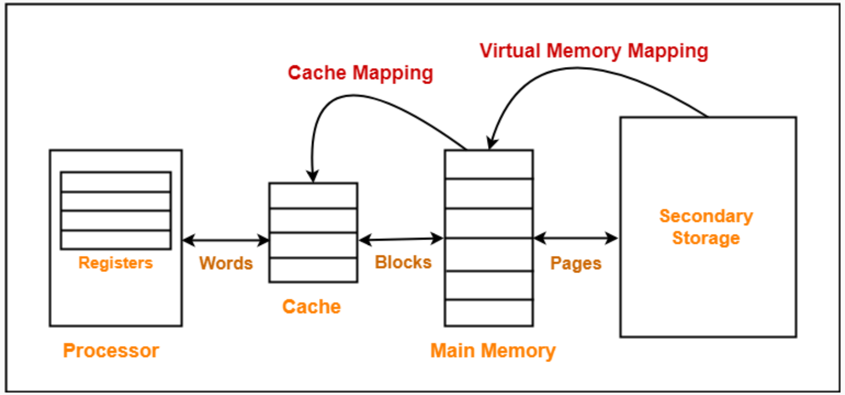


Figure. Illustrates the mapping process

Cache Mapping

Consideration,

The cache can hold 64 Kbytes

- Data is transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as 16Kbytes = 2^{14} lines of 4 bytes each.
- The main memory consists of 16 Mbytes with each byte directly addressable by a 24 bit address (2^{24} = 16Mbytes).
- Thus, for mapping purposes, we can consider main memory to consist of 4Mbytes blocks of 4 bytes each.

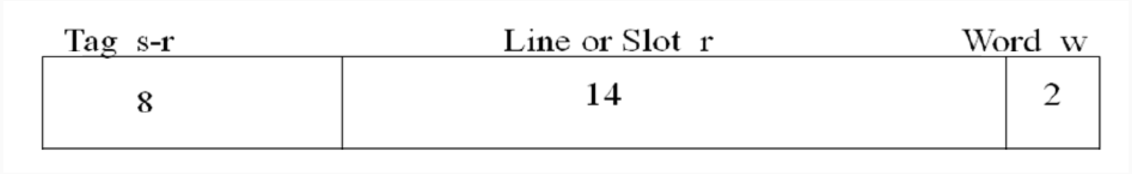
Cache Mapping: Direct mapping

- It is the simplex technique, maps each block of main memory into only one possible cache line i.e. a given main memory block can be placed in one and only one place on cache.

$$i = j \text{ modulo } m$$

Where, i = cache line number; j = main memory block number; m = number of lines in the cache

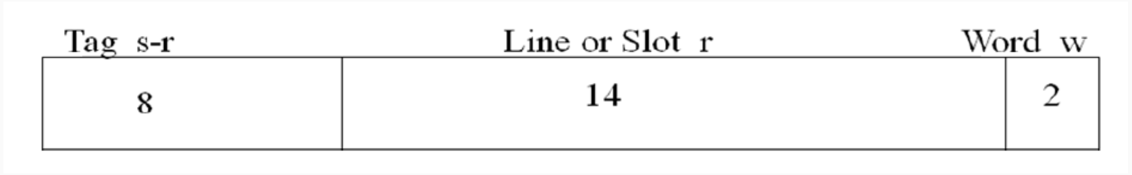
- In direct mapping, the physical address is divided as-



Tag s-r	Line or Slot r	Word w
8	14	2

Cache Mapping: Direct mapping

- The least significant 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 most significant position and a line field of r bits.
- The latter field identifies one of the $m = 2^r$ lines of the cache.



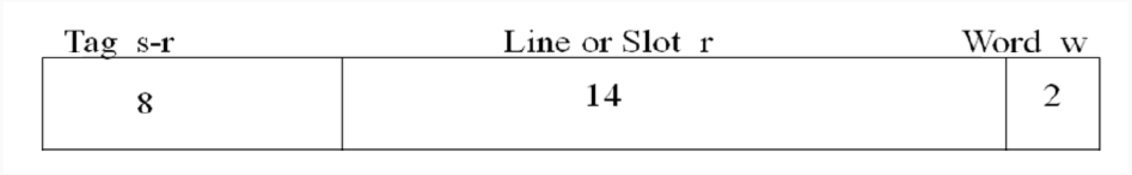
Tag s-r	Line or Slot r	Word w
8	14	2

Cache Mapping: Direct mapping

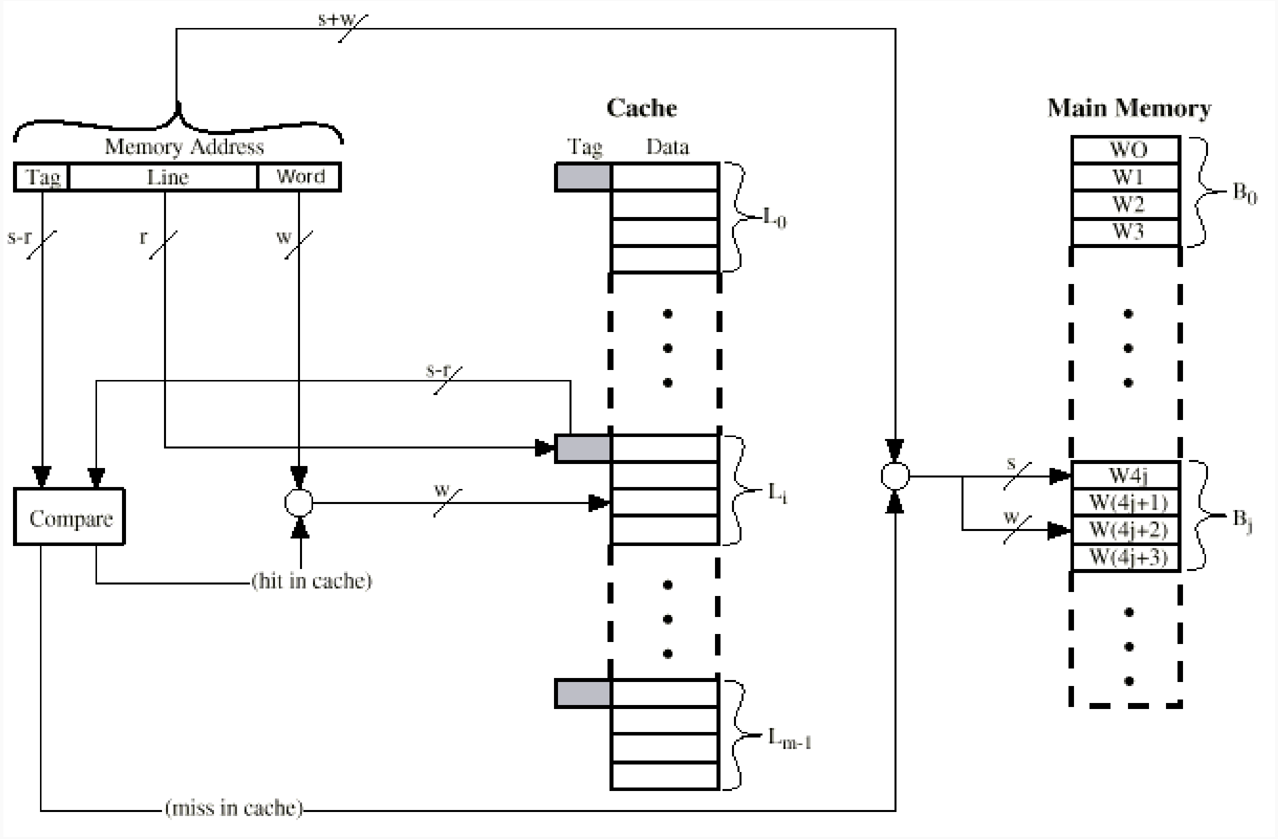
Address length = (s + w) bits

- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2w 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

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag (=22-14), 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag



Cache Mapping: Direct mapping



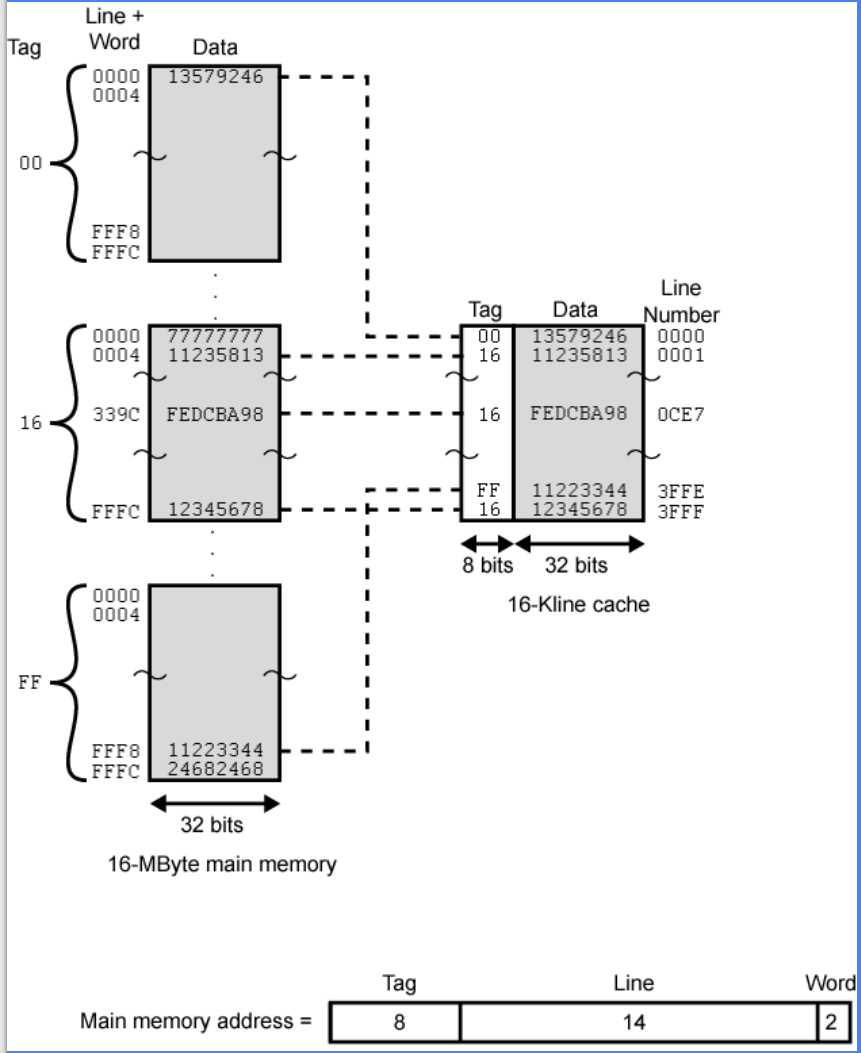
Cache Mapping: Direct mapping

After CPU generates a memory request,

- The line number field of the address is used to access the particular line of the cache.
- The tag field of the CPU address is then compared with the tag of the line.
- If the two tags match, a cache hit occurs and the desired word is found in the cache.
- If the two tags do not match, a cache miss occurs.
- In case of a cache miss, the required word has to be brought from the main memory.
- It is then stored in the cache together with the new tag replacing the previous one

Cache Mapping: Direct mapping

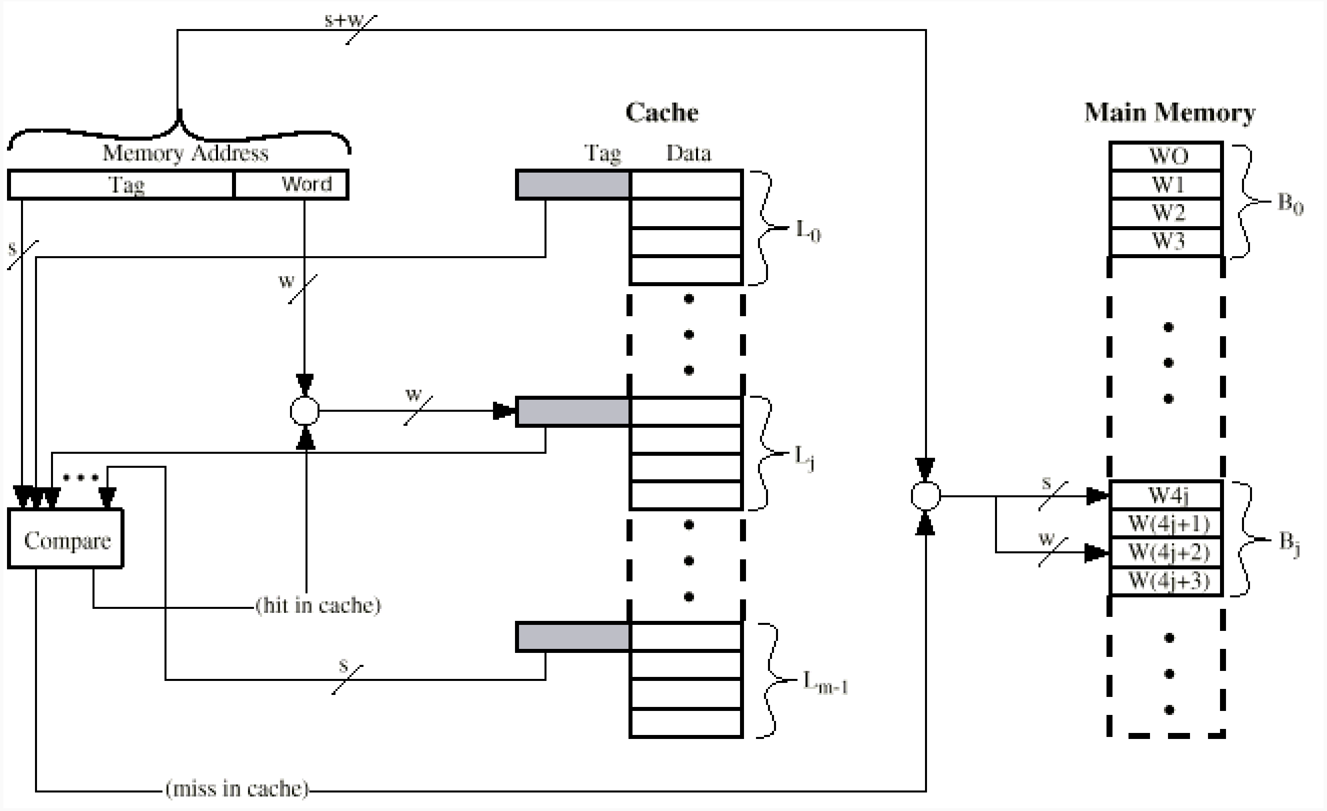
Example



Cache Mapping: Associative mapping

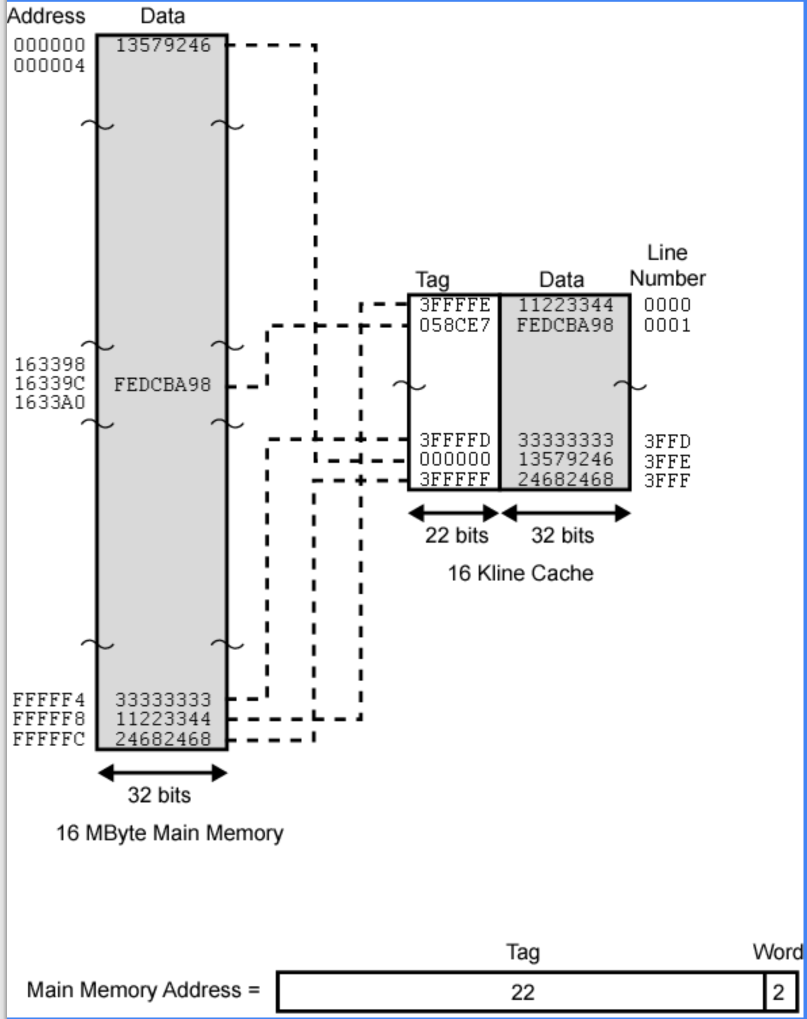
- It overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of cache.
- Cache control logic interprets a memory address simply as a tag and a word field
- Tag uniquely identifies block of memory
- Cache control logic must simultaneously examine every line’s tag for a match which requires fully associative memory
- Very complex circuitry, complexity increases exponentially with size
- Cache searching gets expensive

Cache Mapping: Associative mapping



Cache Mapping: Associate mapping

Example



Cache Mapping: Set-Associative mapping

- It is a compromise between direct and associative mappings that exhibits the strength and reduces the disadvantages
- Cache is divided into v sets, each of which has k lines; number of cache lines = vk

M = v X k

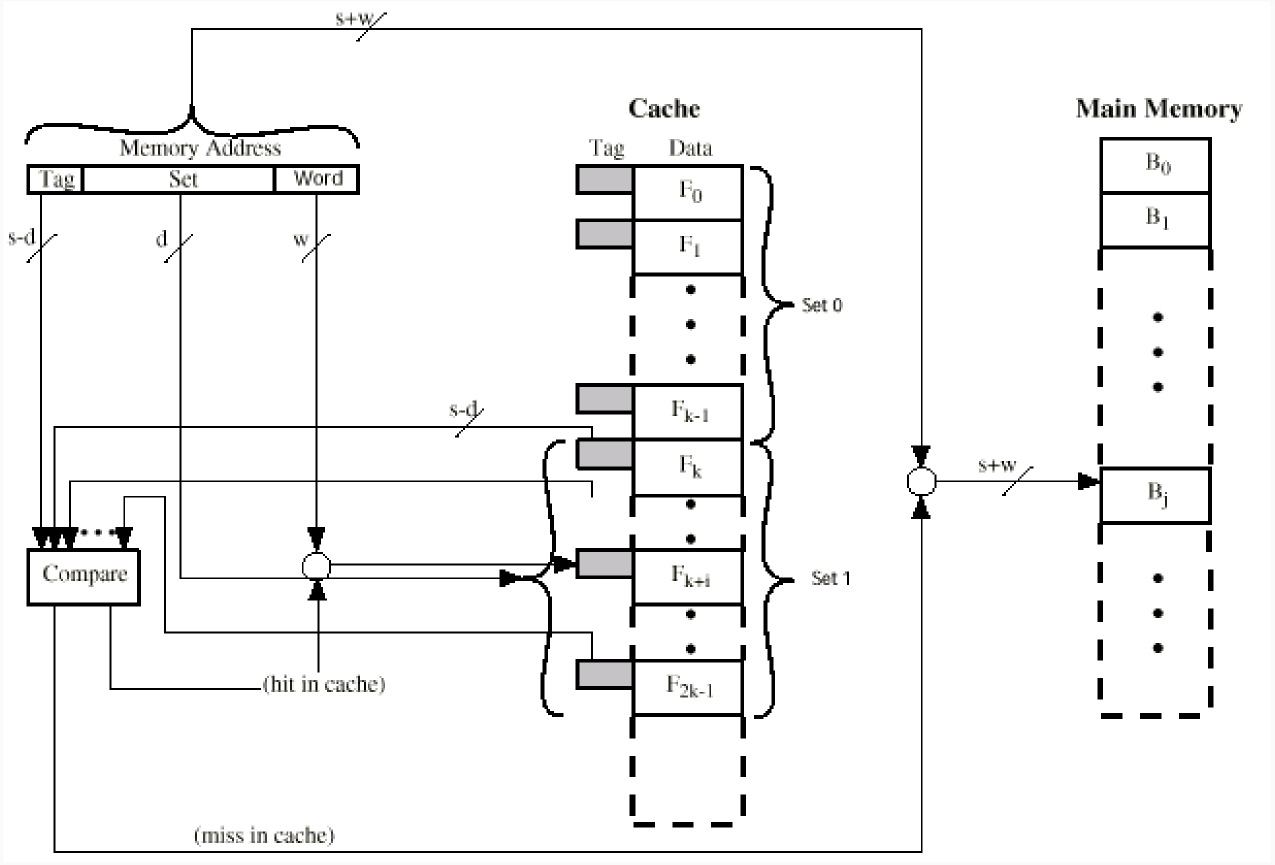
i = j modulo v

Where, i = cache set number; j = main memory block number; m = number of lines in the cache

In other words,

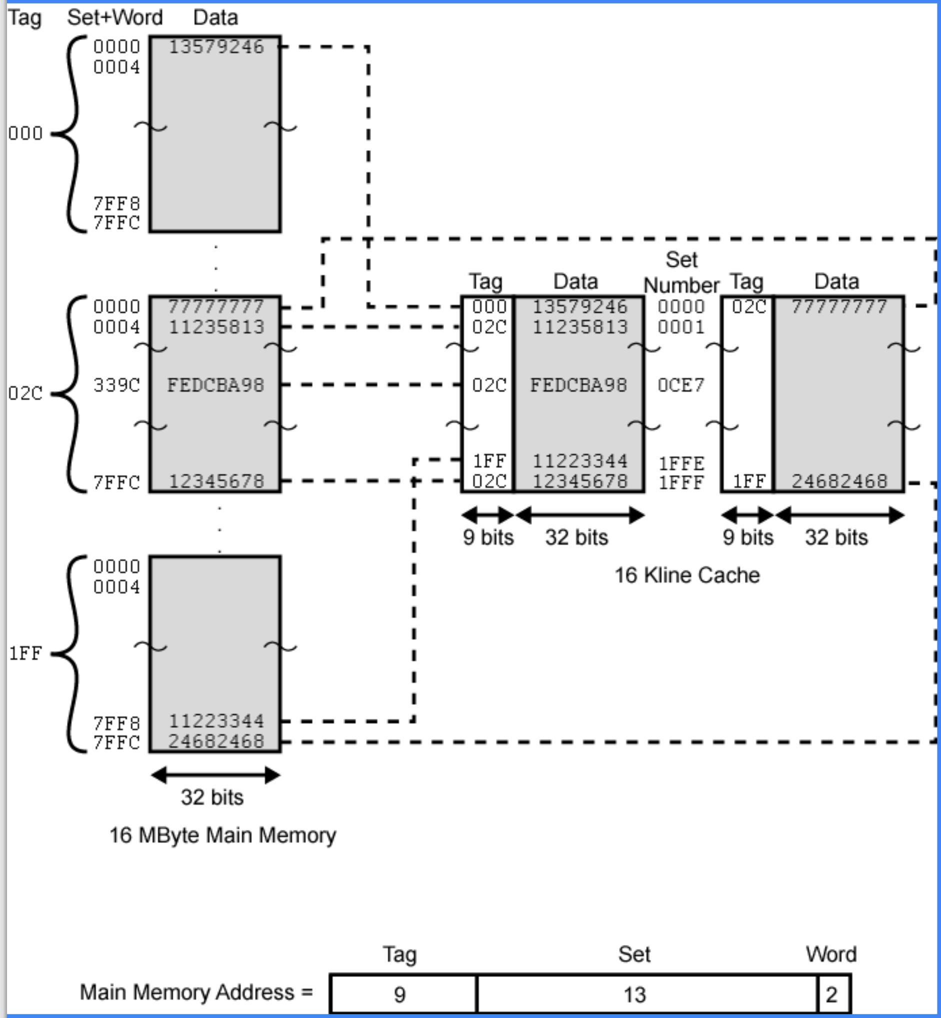
- Cache lines are grouped into sets where each set contains k number of lines.
- A particular block of main memory can map to only one particular set of the cache.
- However, within that set, the memory block can map to any freely available cache line.

Cache Mapping: Set-Associative mapping



Cache Mapping: Set-Associate mapping

Example



Replacement Algorithm

- When all lines are occupied, bringing in a new block requires that an existing line be overwritten.
- So, the replacement scheme is taken into consideration.
- **In direct mapping:**

✓ No choice possible with direct mapping

✓ Each block only maps to one line

✓ Replace that line

Replacement Algorithm

- **In associative and set-associative mapping:**

✓ Algorithms must be implemented in hardware for efficient speed.

1. *Least Recently used (LRU)*

✓ replace that block in the set which has been in cache longest with no reference to it

✓ Implementation:

With 2-way set associative, have a USE bit for each line in a set. When a block is read into cache, use the line whose USE bit is set to 0, then set its USE bit to one and the other line's USE bit to 0.

✓ Probably the most effective method

Replacement Algorithm

2. *First in first out (FIFO)*

✓ replace that block in the set which has been in the cache longest

✓ Implementation: use a round-robin or circular buffer technique.

3. *Least-frequently-used (LFU)*

✓ replace that block in the set which has experienced the fewest references or hits

✓ Implementation: associate a counter with each slot and increment when used

4. *Random*

✓ replace a random block in the set

✓ Interesting because it is only slightly inferior to algorithms based on usage

Write Policy

- When a line is to be replaced, must update the original copy of the line in main memory if any addressable unit in the line has been changed.
- If a block has been altered in cache, it is necessary to write it back out to main memory before replacing it with another block.
- It is forbidden to overwrite the cache block unless main memory is up to date
- Two types:

1. Write through
2. Write back

Write Policy

- When a line is to be replaced, must update the original copy of the line in main memory if any addressable unit in the line has been changed.
- If a block has been altered in cache, it is necessary to write it back out to main memory before replacing it with another block.
- It is forbidden to overwrite the cache block unless main memory is up to date
- Two types:

1. Write through
2. Write back

Write Policy

Write Through

- All write operations are made to main memory as well as to cache, so main memory is always valid
- Other CPU's monitor traffic to main memory to update their caches when needed
- This generates substantial memory traffic and may create a bottleneck
- Anytime a word in cache is changed, it is also changed in main memory
- Both copies always agree
- Generates lots of memory writes to main memory
- Lots of traffic
- Slows down writes

Write Policy

Write back

- When an update occurs, an UPDATE bit associated with that slot is set, so when the block is replaced it is written back first
- During a write, only change the contents of the cache
- Update main memory only when the cache line is to be replaced
- Causes “cache coherency” problems -- different values for the contents of an address are in the cache and the main memory

THANK YOU

Any Queries ?

ankitbhattarai@cosmoscollege.edu.np