locality of reference: Analysis of a large no. of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory

{ 90% of the execution time spent in executing 10% of the code }

Ex:- when a program loop is executed, the CPU repeatedly refers to the set of instructions that constitute the loop. Every time a given subroutine is called, its set of instructions are fetched from Memory. Thus loops & subroutines tend to localize the references to memory for fetching instructions.

→ Sometimes memory references to data also tend to be localized.

→ Over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while remainder of memory is accessed relatively infrequently.

→ If the active portions of the program & data are placed in a fast small memory, the avg memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a <u>cache Memory</u>

It is placed between CPU & Main Memory.

The cache Memory access time is less than the access time of main memory by a factor of 5 to 10.

→ The cache is the fastest component in the memory

→ The Fundamental idea of cache organization is that by keeping the most frequently accessed instructions & data in the fast cache Memory, the avg memory access time will approach the access time of the cache.

→ The basic operation of the cache :

when the CPU needs to access memory, the cache is examined. If the word is found in cache, It is read from fast memory. ~~If~~

If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the addressed one (just accessed) is then transferred from main memory to cache Memory.

Hit ratio :

The performance of cache Memory is measured in terms of a quantity called hit ratio.

Hit : when cpu refers to memory & finds the word in cache, It is said to produce a hit.

Miss : If the word is not found in cache, It is in main memory & It counts as a Miss.

$$\text{hit ratio} = \frac{\text{no. of hits}}{\text{total cpu references to Memory (hits + misses)}}$$

$$= \frac{\text{no. of hits}}{\text{no. of hits + no. of misses}}$$

→ The avg memory access time of a computer system can be improved considerably by use of a cache.

Mapping :

The transformation of data from main memory to cache Memory is referred to as a mapping process.

There are 3 types of mapping procedures :

1. Associative Mapping
2. Direct Mapping
3. set-associative Mapping

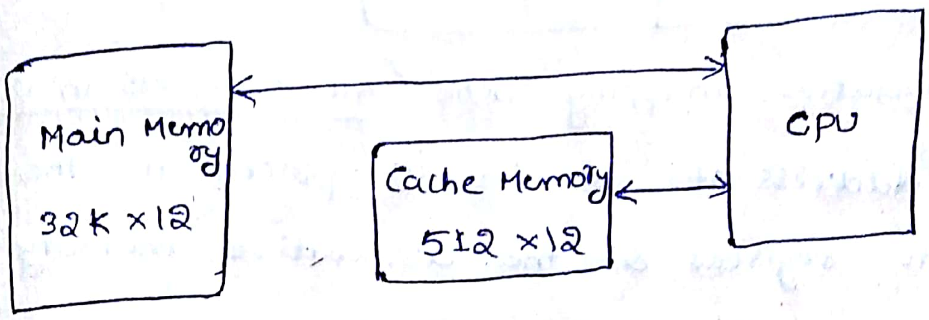


fig: Example of cache Memory

The above example is considered in having detailed explanation of mapping techniques.

Main memory is of size 32K words, ~~of~~ each word is of length 12 bits.

Cache Memory is of size 512 words of 12 bits each

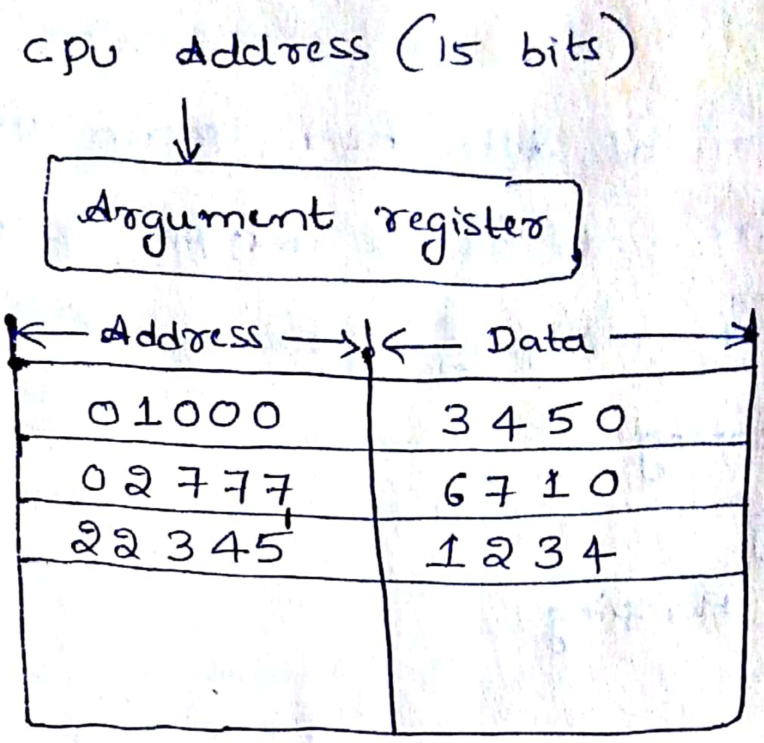
Main memory size = 32K words = 2^{15} words

$\Rightarrow$ 15 bits addressing for main memory

Associative Mapping:

The fastest & most flexible cache organization uses ~~cache~~ ~~Main~~ Associative memory. The associative memory stores both the address & content(data) of the memory word.

→ Any word in main memory can be stored in any location of the cache, as total main memory Address is stored along with word.



Address	Data
01000	3450
02777	6710
22345	1234

Address is also called as TAG

Data ⟹ word

Fig: Associative mapping cache (All numbers in octal)

→ A CPU Address of 15 bits is placed in the argument register & the associative memory is searched for a matching address.

→ If Address is found, the corresponding 12-bit data is read & sent to the CPU. If no match occurs, the main memory is accessed for the word

→ If miss occurs, that address-data pair is then transferred to the associative cache memory.

→ If cache is full, an existing address-data pair must be replaced with missed address-data pair

The decision as to what pair is replaced is determined from replacement Algorithm

{Ex:- First In First Out (FIFO) replacement policy

Direct Mapping:

Associative memories are expensive compared to random- acress memories because of added logic associated with each cell. ~~The~~

The CPU Address is divided into two fields

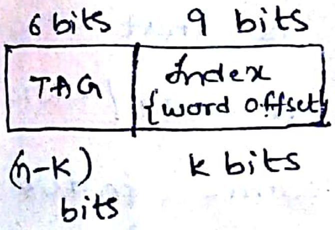
least significant bits = Index field (word offset) equal to no. of bits used for cache Addressing

remaining bits = TAG field

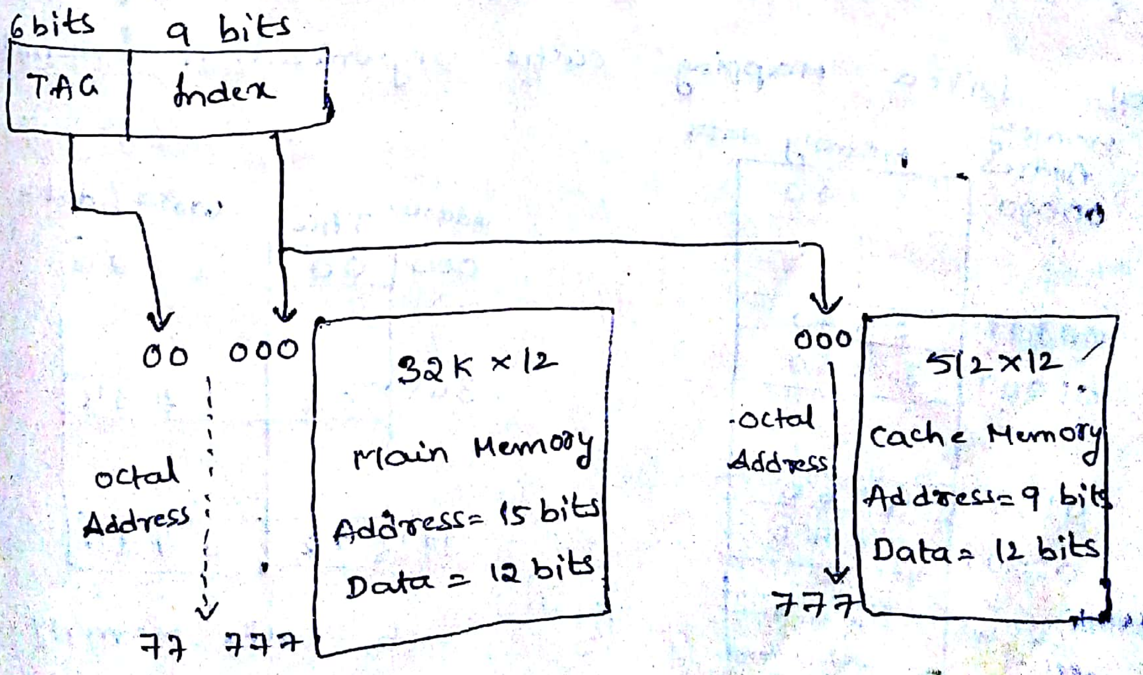
In the example considered,

9 bits are considered for word offset {As cache size 512 words = 2^9 words}

remaining are 6 bits ⇒ considered as TAG bits



where n = no. of bits for Main memory Addersing
k = no. of bits used for cache Addressing



Addressing relationships b/w Main memory & Cache Memory

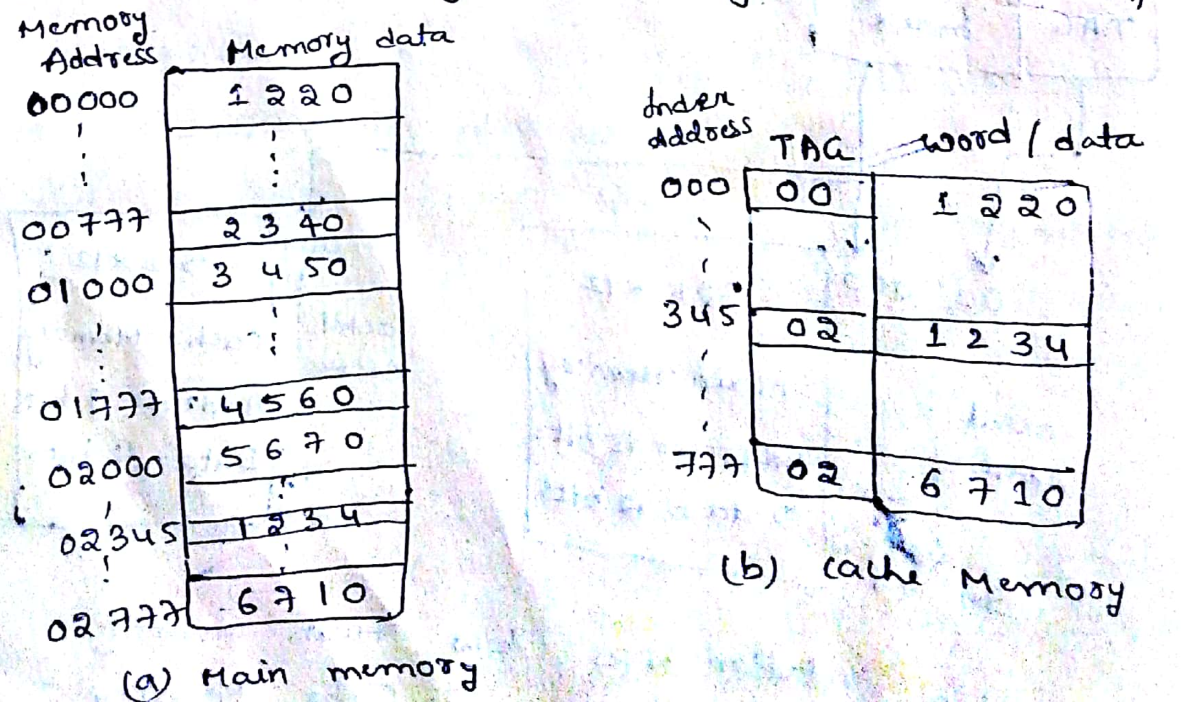
→ Each word in cache consists of the data word & its associated tag. When a new wor is first brought into the cache, the tag bits are stored alongside the data bits.

→ when CPU generates a memory request, th index field (word offset) is used for the address to access the cache. The TAG field is compared with TAG bits stored in the cache If the 2 tags match, there is a hit. otherwise m

→ If there is a miss, required word is read from main memory & then stored in the cache together with the new tag replacing the previous value.

<u>Disadvantage</u> of Direct Mapping is that the hit ratio can drop considerably if two / more words whose addresses have same index & but different tags are accessed repeatedly.

Ex!- Direct Mapping cache organization example



(b) cache Memory

(a) Main memory

In the example,

At an Index address (000, 345 ...)

only one word can be stored.

with the Index 000, different Tags are possible

00, 01, 02 77

Suppose that CPU wants to access the word

at address $\underset{\text{TAG}}{02}\underset{\text{Index}}{\underline{000}}$.

At Index 000, the cache Memory has the word

with TAG 00 but not 02. So It is a Miss.

∴ main memory is accessed & data word 5670

{which is at 02000} is transferred to the cache

by replacing the word 1220.

→ The direct mapping example described uses a

block size of one word. A block may contain

more than one word.

→ The Index field is now divided into 2 parts.

the block field & word field.

The address from CPU is divided as

$$\boxed{\text{TAG}}\underbrace{\boxed{\text{block offset}}\boxed{\text{word offset}}}_{\text{Index bits equal to no. of bits in cache address}}$$

Consider the block size of 8 words.

∴ no. of blocks in cache $= \frac{512}{8} = 64$ blocks

$= 2^6$ blocks

block offset = 6 bits

In each block, 8 words (2^3 words)

word offset = 3 bits

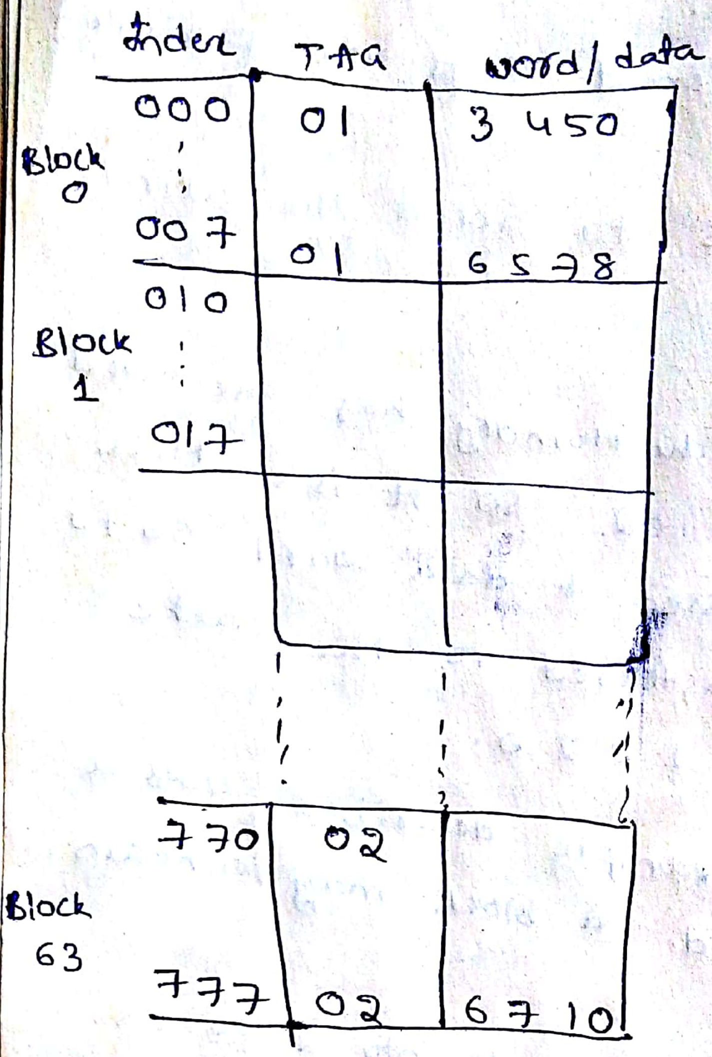
Therefore 9 bit Index field is divided into

6 bits block offset

3 bits word offset.

TAG	block	word
6	6	3

Q:- Direct Mapping cache with block size of 8 words



Everytime a miss occurs, an entire block of 8 words must be transferred from main memory to cache memory. Although this takes extra time, the hit ratio will most likely improve with a larger block size because of sequential nature of computer programs.