memory organisation

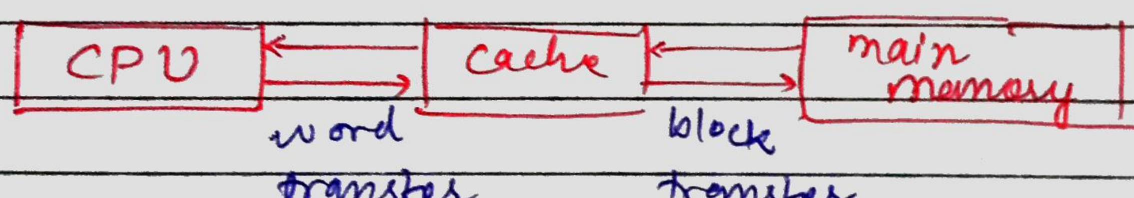
cache memory :- it is an extremely fast memory type that act as a buffer between main memory & CPU. It holds frequently requested data and instruction so that they are immediately available to the cpu when needed.

It is used to reduce the average time to access data from main memory. The cache is smaller and faster memory which store copies of data from frequently used main memory location

The basic operation of cache -

when the CPU needs to access memory. the cache is examined. if the word if found in cache, it is read from the fast memory (cache)

If the word addressed by CPU is not found in not found in cache, the main memory is accessed to read the words.



DATE
PAGE

cache hits :- when the CPU refers to memory and find the word in cache, it is said to produce a hit

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

cache miss :- if word is not found in cache, it is in main memory and it count as miss.

Associative Memory :- / content addressable memory.

The search procedure is the strategy for choosing a sequence of address, reading the content of memory at each address and comparing the information read with the item being searched until match occurs.

The time required to search an item stored in memory can be reduced considerably if stored data can be identified for access ~~of~~ by the content of data itself rather than by an address.

A memory unit accessed by its content is called an associative memory or content addressable memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by location

The word is written in memory address

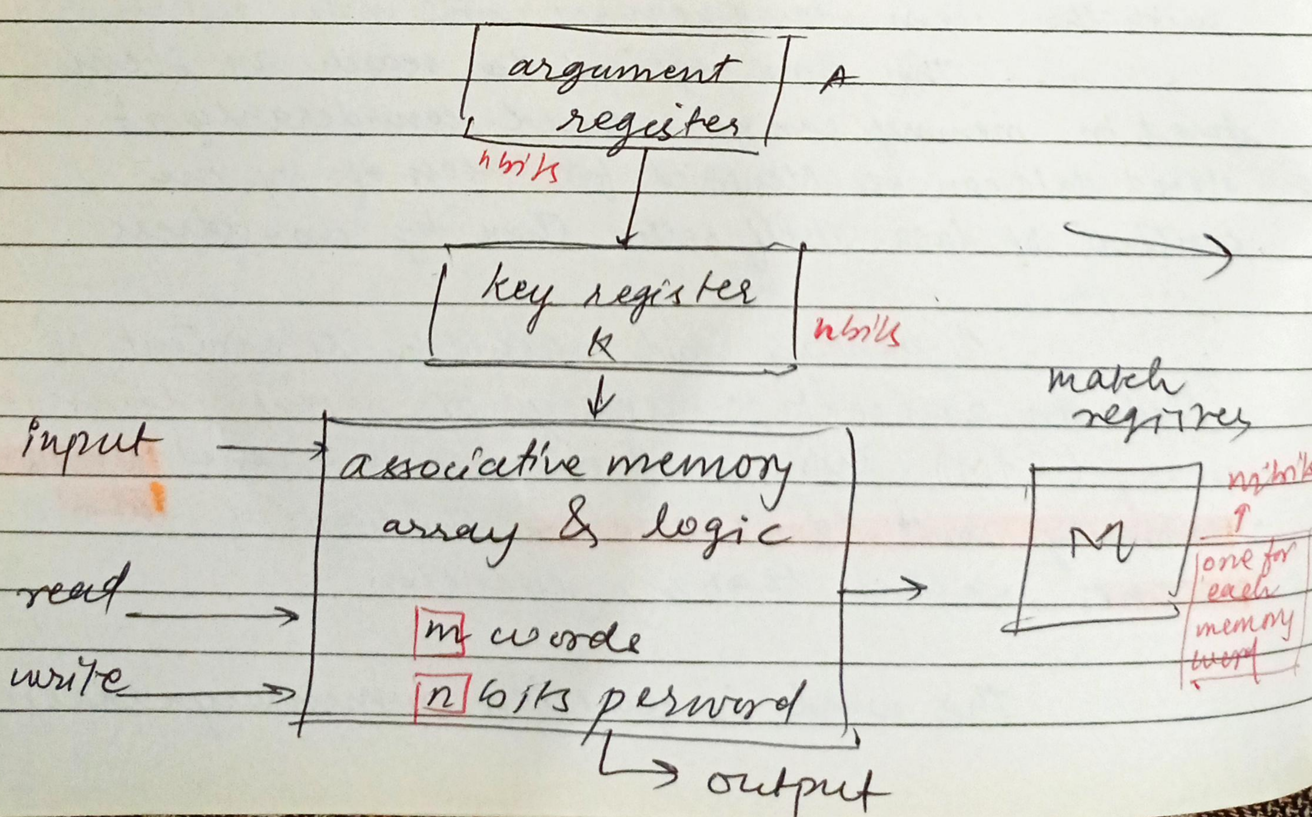
matched part of words or complete content.

no address is given. The memory is capable of finding an empty unused location to store the word.
The memory locates all the words which matches the specific content and marks them for reading.
because of its organisation the associative memory is uniquely suited to do parallel searches by data association. Searches can be done on entire word or specific field within a word.

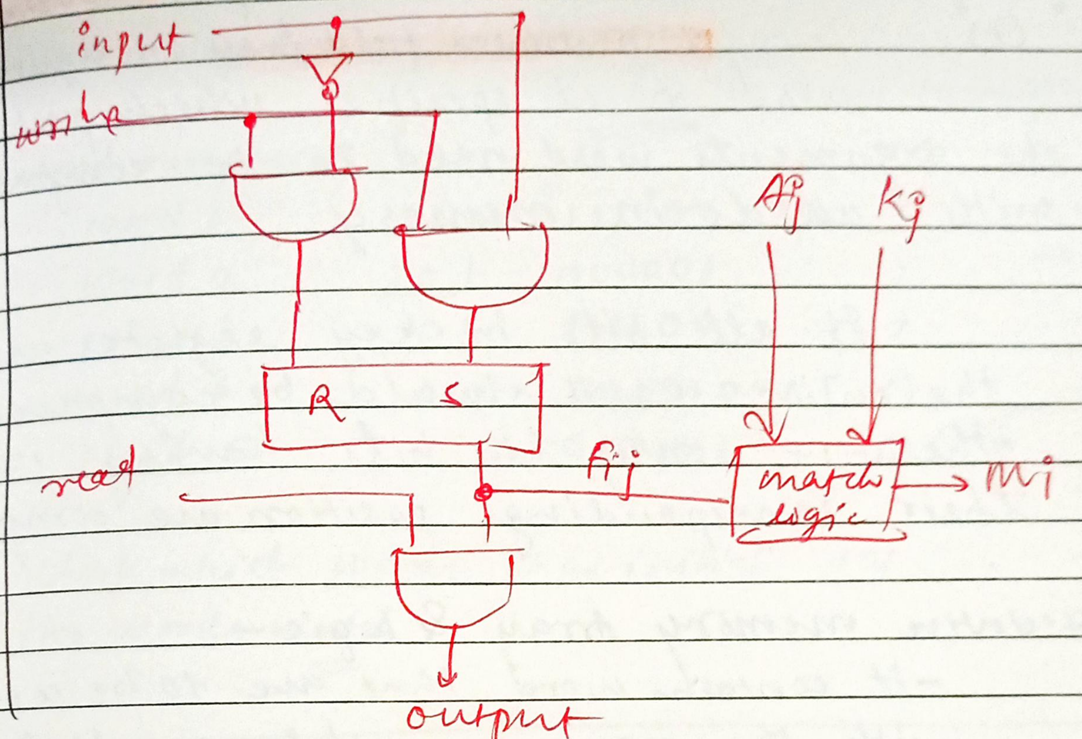
Associative memory is more expensive than RAM because each cell must have storage capacity ∴ associative memory are used where the search time is very critical.

↖ also contain matching circuit to match (logic).

Block diagram



one cell of association memory –



example :- 11100011 ← we can specify this
111... ← or even this can specify.

Argument register :- it contains words to be searched.
(n bit) 1 bit for each bit of word.

Match logic (m) : it has m bits. one bit corresponding to each word in memory.
After matching process, the bits corresponding to matching words in match register are set to 1

DATE
PAGE

key register :- (K) it provide mask for choosing a particular field/key in argument word or it specifies which part of the arguments word need to be compared with words in memory.

- If all bits in key register are 1's the entire word should be compared otherwise, only the bits having 1's in their corresponding position are compared.

Associative memory Array & logic -

- it contains word that are to be compared with the argument word in parallel

- it consist of m word with n bits per word.

Reading is accomplished by sequential access in memory for those word whose bits are set to 1

(*) searching → parallel reading - sequential example :- A 101 111100

K 111 000000

mask (no need to compare)

word samples -

			M	
word 1	100	111100	0	→ no match
word 2	101	000001	1	→ for match
word 3	101	111100	1	
word 4	110	001010	0	
word 5	111	000111	0	

Check which words has initial 101
two words match

internal organization of cell C_{ij}

→ it consist of flip-flop storage element F_{ij} & circuit for reading writing & matching

→ The input bit is transferred into storage cell during a write operation

→ The bit stored is read out during read op.

→ The match logic compares the content of storage all with the corresponding unmasked bit of argument & set the bit in M_i

DATE
PAGE

mapping –

The transformation of data from main memory to cache memory is reffered as mapping process

three types of mapping –

1. associative mapping
2. direct mapping
3. set associative mapping

Associative mapping :–

→ fastest & most flexible cache organisation uses associative memory

→ in associative ~~memory~~ mapping. caches are made up of associative memory. Associative memory is used to store both the address and content of memory word. (data)

→ ~~If~~ It permits any location in cache to store any words from main memory i.e it enables any word from main memory at any place in the cache memory (which doesn't happen in other mapping).

The main memory can store 32k words of 12 bit each. The cache is capable of storing 512 of these word at any time. for every word stored in cache their is duplicate copy in main memory. The CPU communicates both memory

Associative mapping cache-

(A) all numbers are in Octal

CPU address (15 bit)

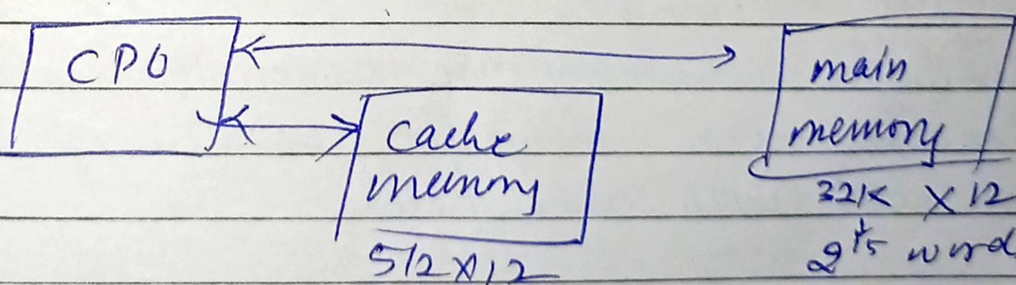
↓ stores ↑

Argument Register

15 bit address	12 bit data
01000	3450
02777	6710
22345	1234

it first sends 15 bit address to cache. If their is a hit the CPU accept 12 bit data from cache.

If their is a miss the CPU reads the word from main memory and transfer to Cache.



2^{15} words

15 bits = CPU add.

data = 12 bit

A cpu address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

If the address is found, the corresponding 12 bit data is read and sent to the cpu.

If no match found then main memory is accessed for word.

→ The address data pair then transferred to associative cache memory.

→ If cache memory is full then an address-data pair must be displaced to make space for pair that is needed.

→ The decision for replacement is done by an algorithm. a simple procedure is replace is done on basis of round-robin order. which constitute FIFO (first in first out) replacement policy

Direct Mapping

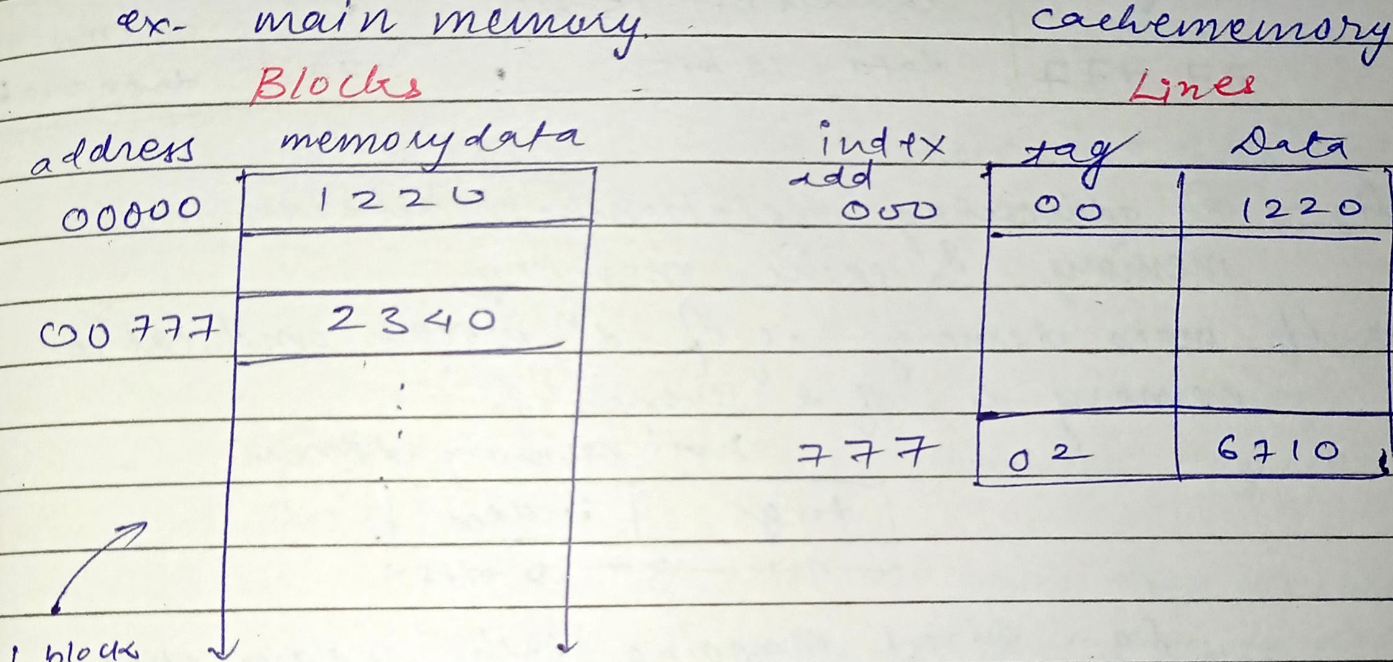
→ associative memory are expensive compared to, Random access memory (because of added logic associated with each cell).

→ for random access memory direct mapping

→ simplest technique-

direct mapping - it maps each block of main memory into only one possible cache line or it assign each memory block to a specific line in the cache.

ex-



may contain 1 word to 16 words

⇒ The CPU address of 15 bits is divided into two field the nine least significant bit constitutes index field and remaining 6 bit forms the tag field

tag	index

cpu address

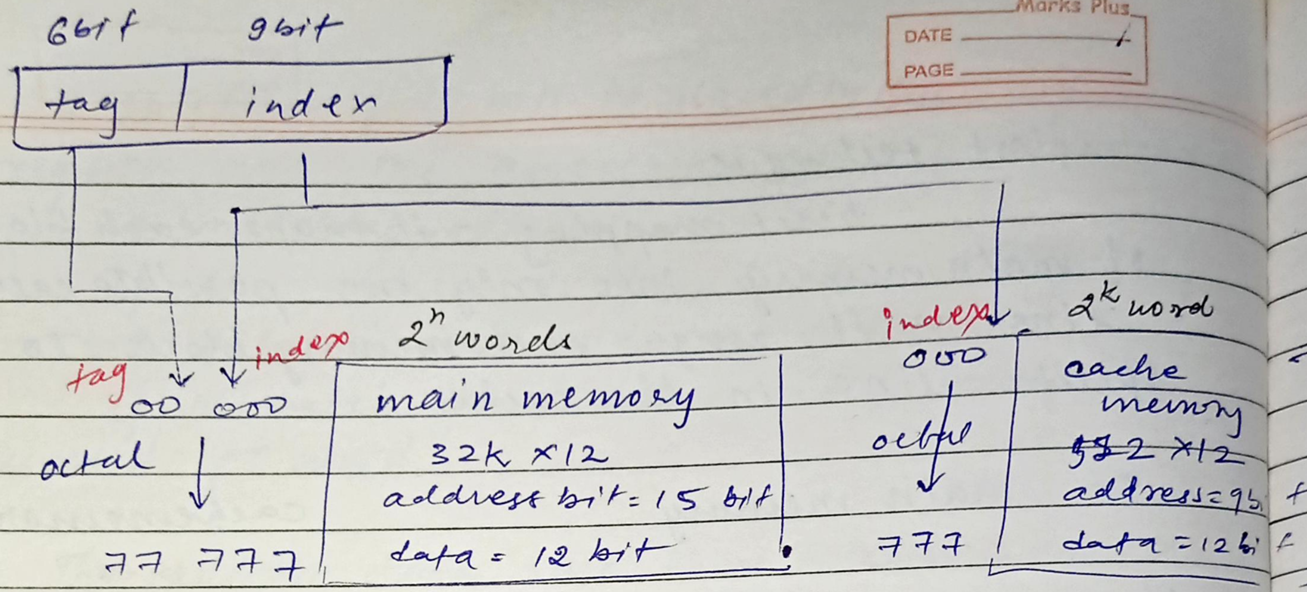


fig — ⊕ addressing relationship between main memory & cache memory.

→ if main memory is of 2^n words and cache memory is of 2^k words

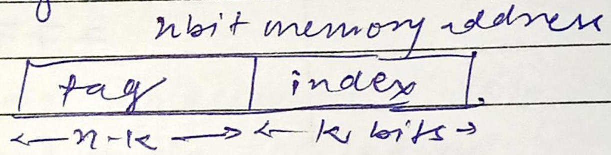
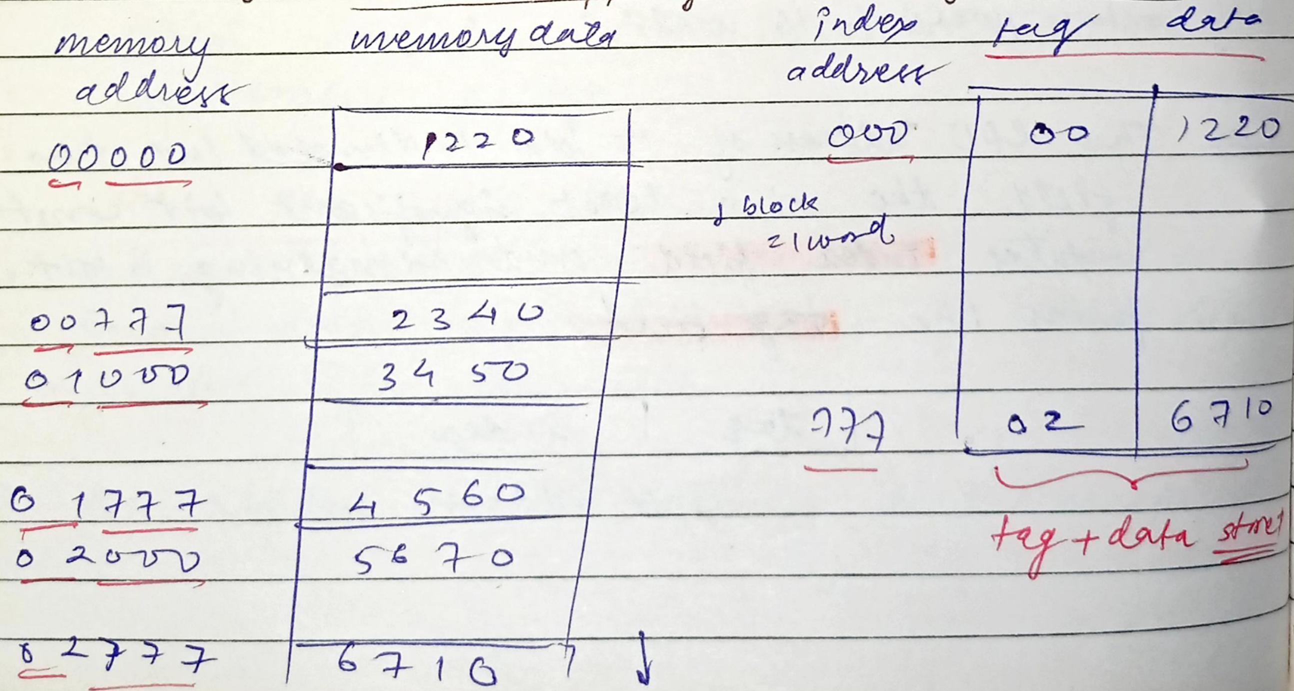


fig - Direct Mapping cache organisation



→ when the CPU generates memory request the index field is used for the address to access the cache.

→ The tag field of cpu address is compared with the tag ~~field of~~ in word read from cache

→ If the tag are match, there is a hit and desired word is in the cache memory.

→ If their is no match, there is a miss. then required word is read from main memory. It is then stored in cache memory together with new tag

~~Disa~~ Disadvantage → hit ratio drop considerably, if two or more data have same ~~tag~~ index but different tags.

example — the word at address zero is currently stored in cache (index = 000, tag = 00, data = 1220). suppose that cpu want to access the word at 02000, here the index address is 000, so it is used to match at cache memory now tag will be compared 02 ≠ 00 : which doesn't produce match, due to this main memory is accessed and data 5670 is transferred to cache memory at index 000 with replaced tag and data 02, 5670

DATE
PAGE

Block size of 8 words —

	index	tag	data
block 0	000	01	3450
	⋮		
	007	01	6578
block 1	010		
	⋮		
	017		
	⋮		
block 63	770	02	
	⋮		
	777	02	6710

tag (6)	block (6)	word (3)

index 9 bits (block + word)

→ every time miss occurs, an entire block of eight words must be transferred to MM.

→ this takes extra time but hit ratio definately get improved

cache memory size = 512
if 1 block = 8 word
total no. of block = $\frac{512}{8}$ = 64 block

= 2^{6} block
↑
6 bit size of 1 block

now 1 block = 8 word
= 2^3 word
↑
3 bit

Set associative Mapping

→ improved form of Direct Mapping, where drawbacks of direct mapping is removed.

→ Drawback of direct mapping –

Two words with same index and their address but with different tag values cannot reside in cache memory at same time.

eg –
01 000 ✓ →
02 000 ✓ →

index	tag	data
000	~~01~~	~~1280~~
~~000~~	02	3450

→ in set-associative mapping –

– each word of cache can store two or more words of memory under same index address, creating a set

– each data word is stored together with its tag.

– the number of tag-data item in one word is said to form a set

→ Set associative mapping combines direct mapping and associative mapping.

eg — two way set associative mapping

two words in each set

index	tag	data	tag	data
000	01	3450	02	5670
777	02	6710	03	2340
9 bits	6 bits	12 bits	6 bit	12 bit

1 word = 2(6+12) = 36 bit word length

→ each index address refers to two data words & their associative tags

✓ word length = 2 (6 + 12) = 36 bit
(6 = tag, 12 = data)

→ index 9 bits ⇒ $2^9 = 512$ words

two way set associative
cache size = 512 × 36

→ it can accomodate 1024 words of main memory. since each word of cache contain two data words (512 × 2) = 1024

→ in general, a set associative cache of set size k will accomodate k words of main memory in each word of cache

→ when cpu generates a memory request, the index value of address is used to access cache. The tag field of CPU address is then compared with both tags in cache to determine if match occur.

→ comparison logic is done by an associative search of tags in the set similar to associative memory.

→ The hit ratio will improve as set size increases However increase in set size increase the numbers of bit in word of cache and require complex comparison logic

→ when miss occur, it is necessary to replace one of tag-data item with new value

common replacement algorithm are —

1. FIFO (first in first out)
2. least recently used (LRU)
3. Random replacement policy