CAO: Lecture 24
Memory Hierarchy

Topics Covered

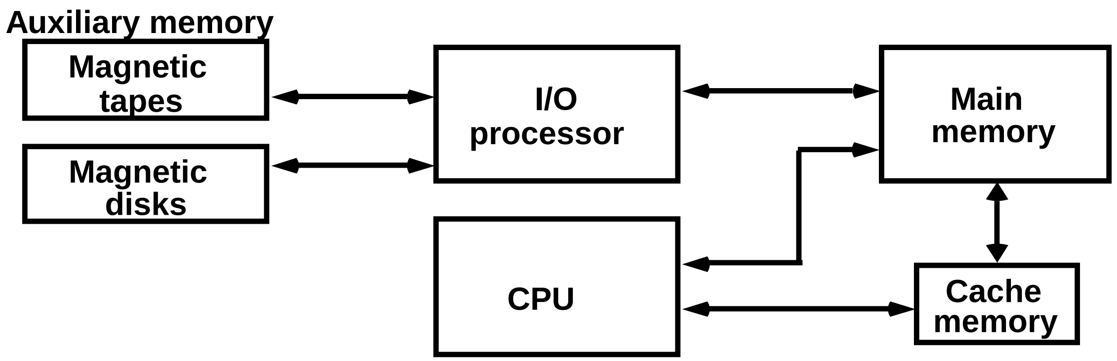
- Memory organization
- Memory hierarchy
- Main memory
- Memory address map
- Connection of memory to cpu
- Auxiliary memory
- Associative memory
- Cache memory

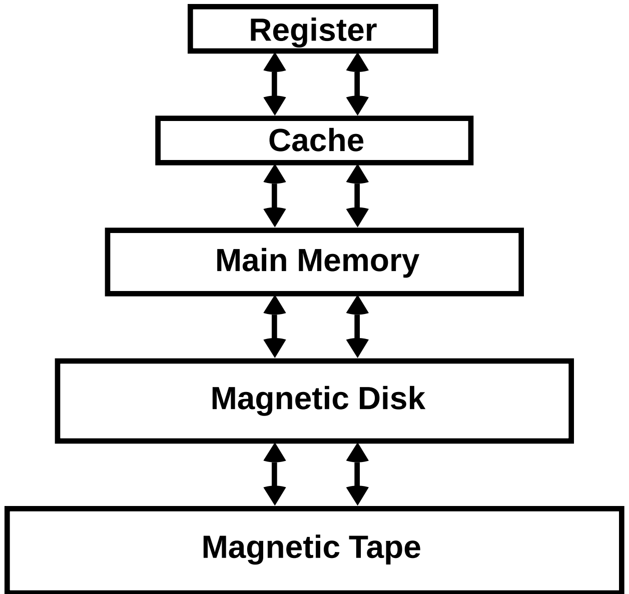
MEMORY ORGANIZATION

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Cache mapping

MEMORY HIERARCHY

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system

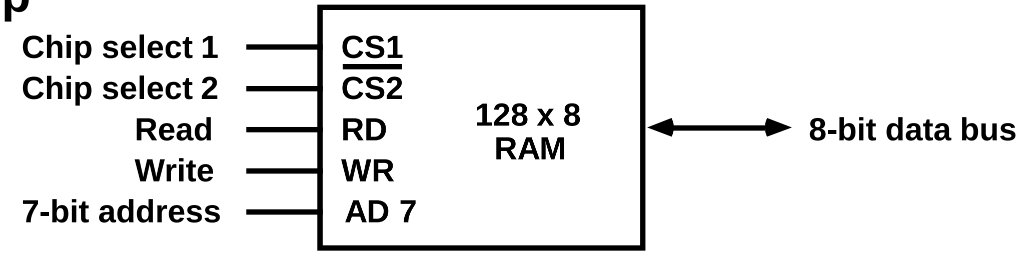




MAIN MEMORY

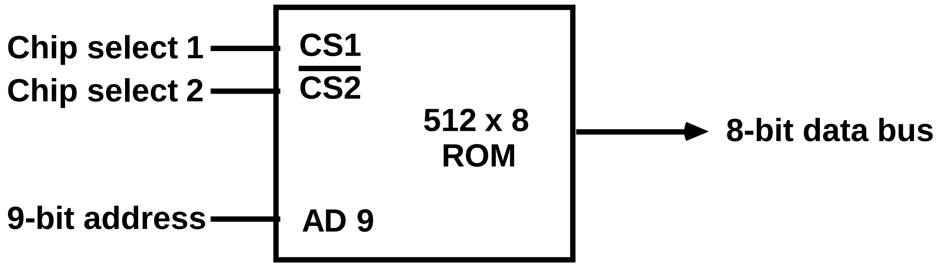
RAM and ROM Chips

Typical RAM chip



CS1	$\overline{\text{CS2}}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedence
0	1	x	x	Inhibit	High-impedence
1	0	0	0	Inhibit	High-impedence
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedence

Typical ROM chip



MEMORY ADDRESS MAP

Address space assignment to each memory chip

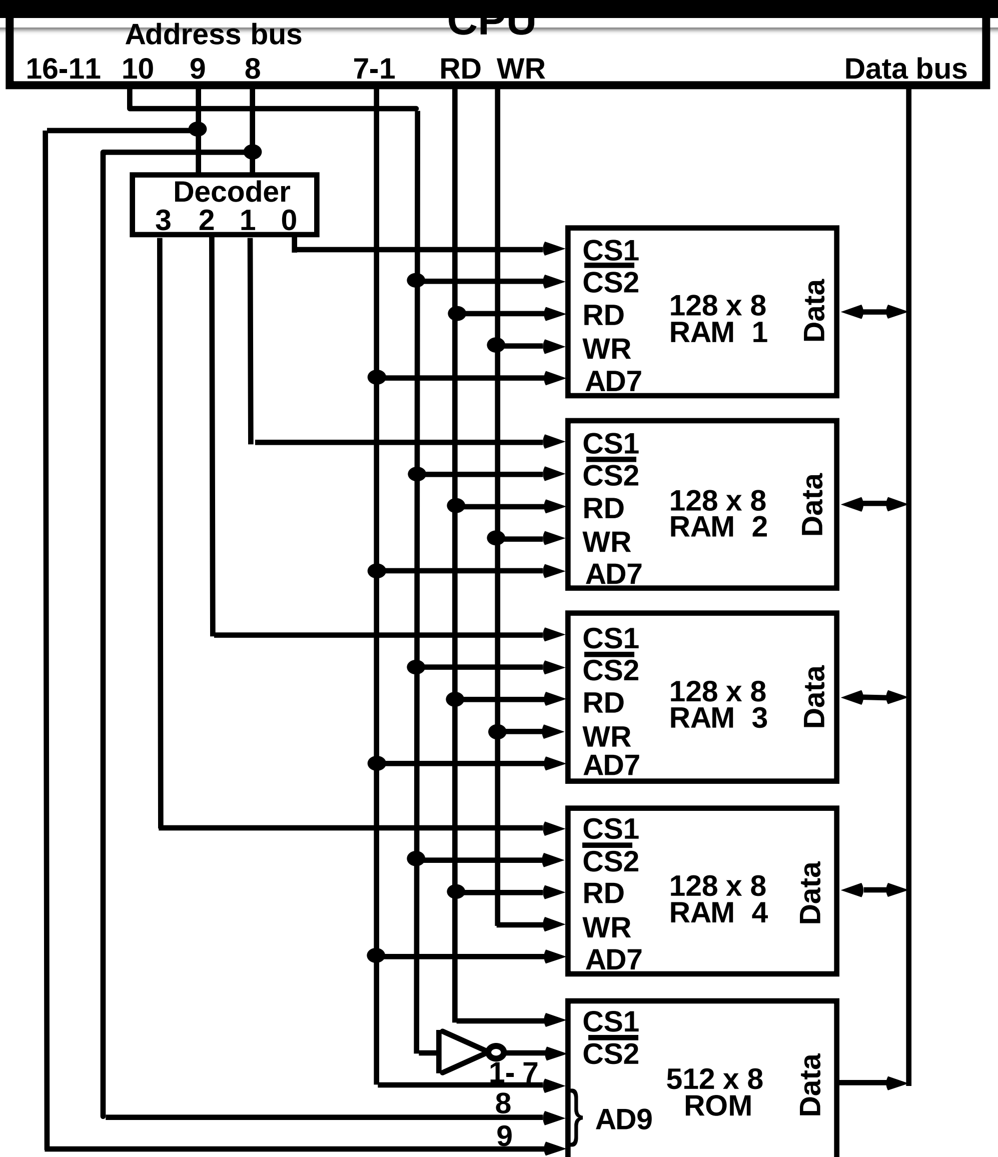
Example: 512 bytes RAM and 512 bytes ROM

Component	Hexa address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Memory Connection to CPU

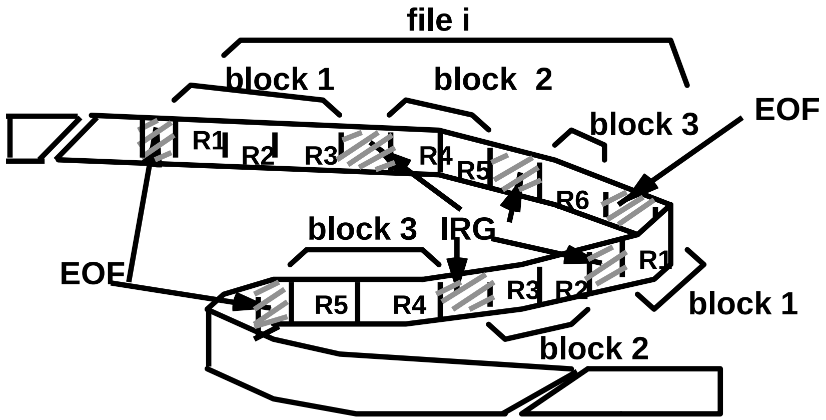
- **RAM and ROM chips are connected to a CPU through the data and address buses**
- **The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs**

CONNECTION OF MEMORY TO CPU

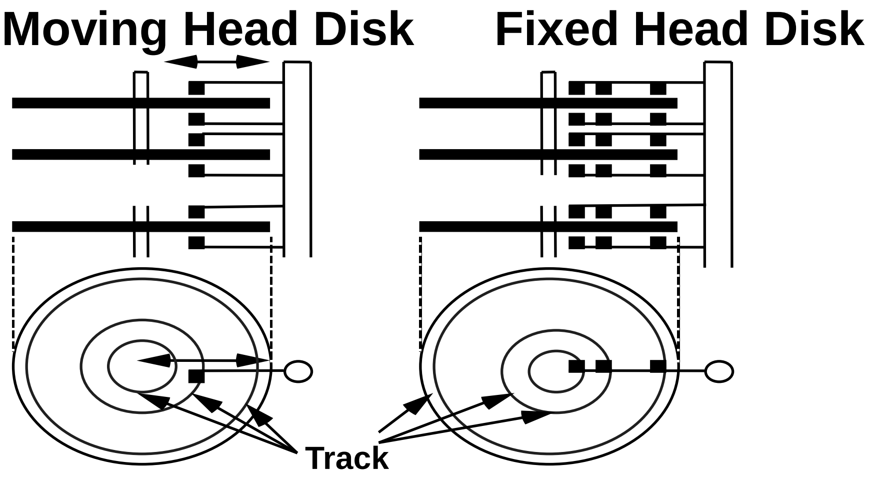


AUXILIARY MEMORY

Information Organization on Magnetic Tapes



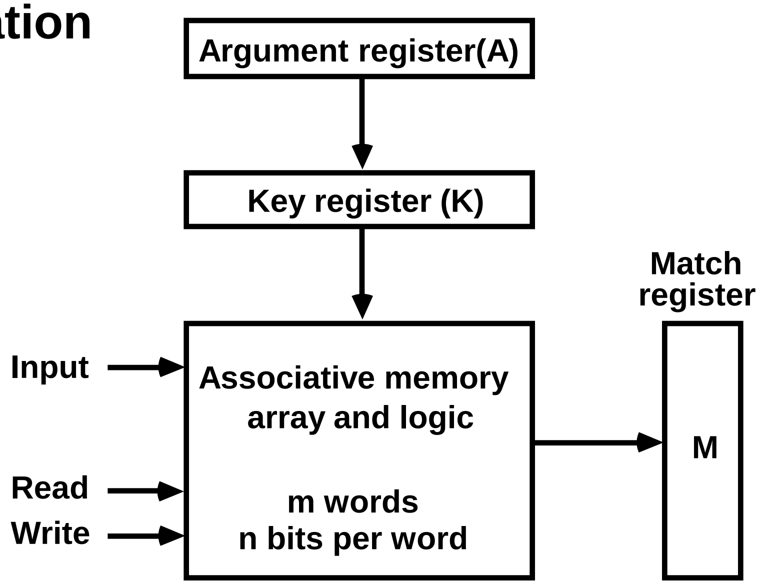
Organization of Disk Hardware



ASSOCIATIVE MEMORY

- Accessed by the content of the data rather than by an address
- Also called Content Addressable Memory (CAM)

Hardware Organization

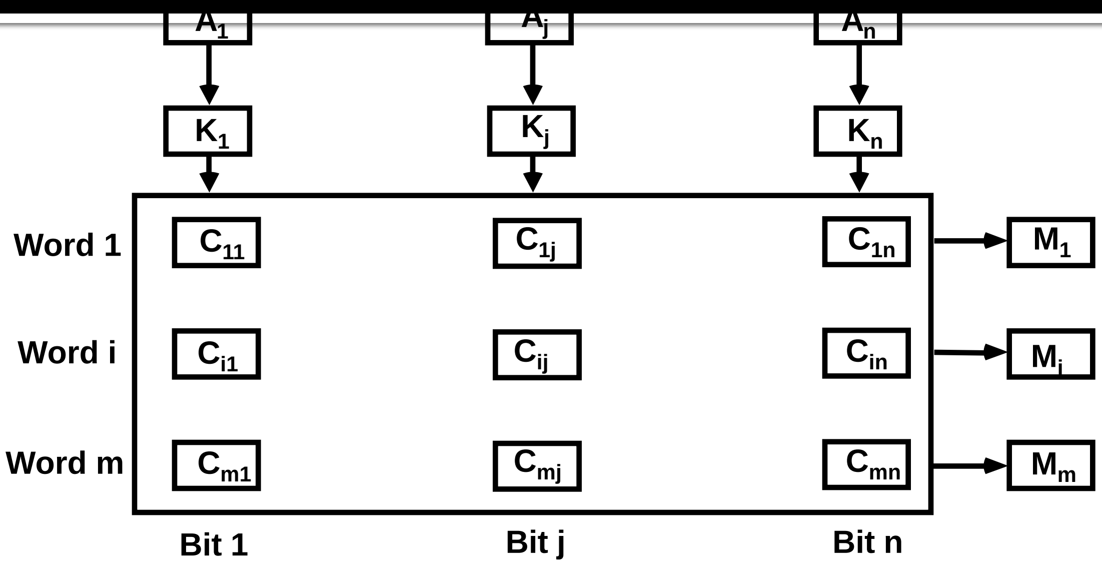


EXAMPLE:-

A	101 111100	
K	111 000000	
WORD1	100 111100	NO MATCH
WORD2	101 000001	MATCH

- Compare each word in CAM in parallel with the content of A(Argument Register)
- If CAM Word[i] = A, M(i) = 1
- Read sequentially accessing CAM for CAM Word(i) for M(i) = 1
- K(Key Register) provides a mask for choosing a particular field or key in the argument in A (only those bits in the argument that have 1's in their corresponding position of K are compared)

ORGANIZATION OF CAM

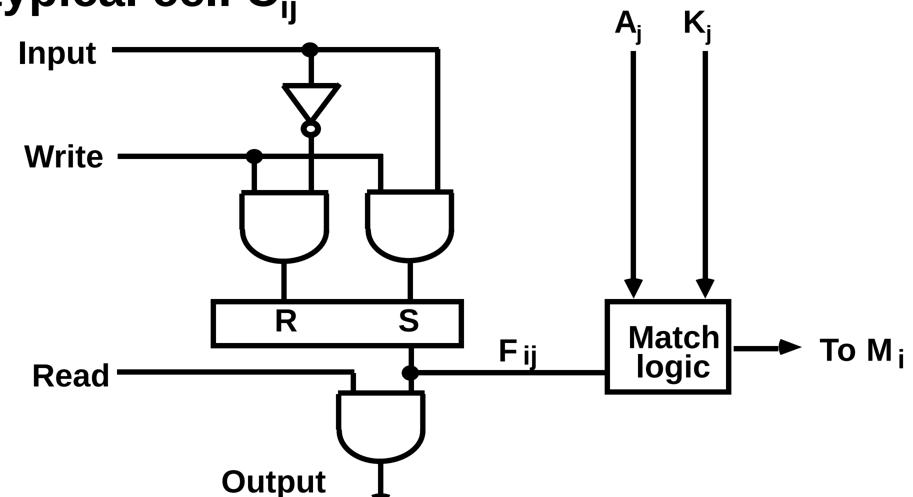


C_{ij} cell for bit j in word i

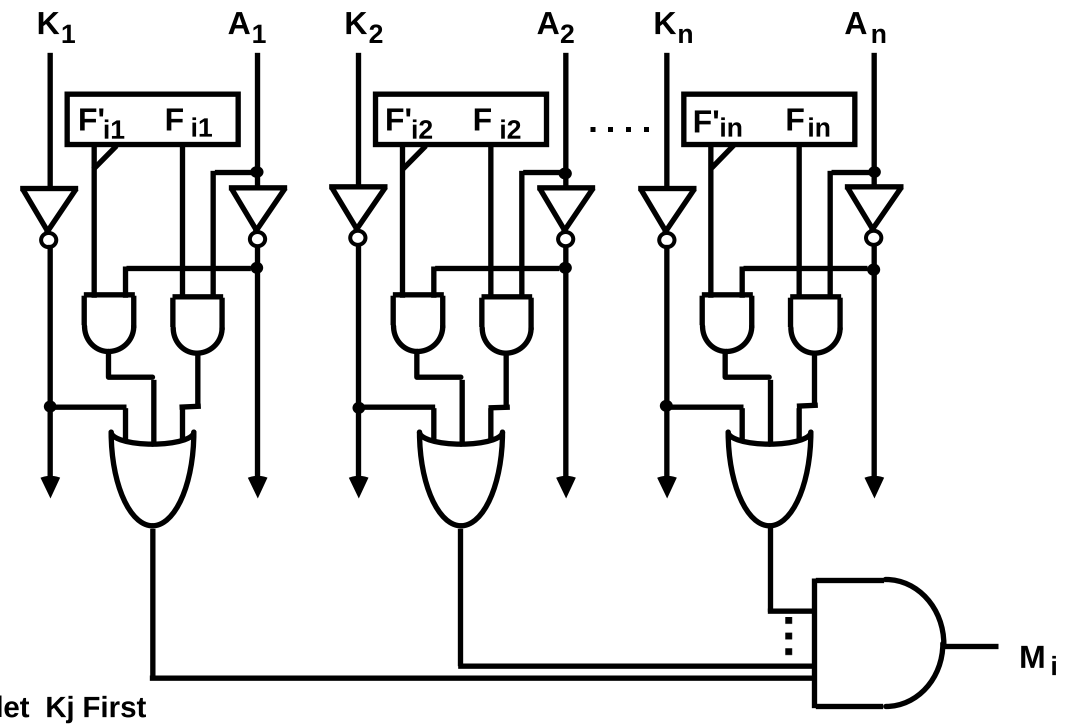
i->word no
J->bit position in word

Internal organization of a typical cell C_{ij}

Fij-flip flop storage element



MATCH LOGIC(one word of associative memory)



1) Neglet Kj First

$X_j = A_j F_{ij} + A_j' F_{ij}'$

$M_i = X_1 X_2 X_3 \ldots\ldots\ldots\ldots X_n$

2) Now include Kj

$$X_j + k_j' = \begin{cases} x_j & \text{if } k_j = 1 \\ 1 & \text{if } k_j = 0 \end{cases}$$

$M_i = (X_1 + K_1')(X_2 + K_2')(X_3 + K_3')\ldots\ldots\ldots(X_n + K_n')$

Or

$M_i = \Pi(A_j F_{ij} + A_j' F_{ij}' + k_j')$ $1 < j < n$

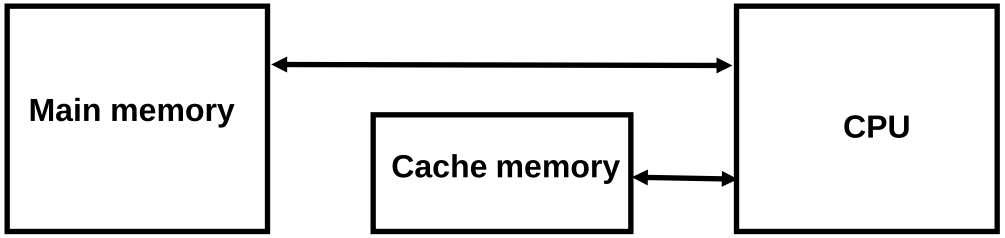
CACHE MEMORY

Locality of Reference

- **The references to memory at any given time interval tend to be confined within a localized areas**
- **This area contains a set of information and the membership changes gradually as time goes by**
- ***Temporal Locality***
  The information which will be used in near future is likely to be in use already(e.g. Reuse of information in loops)
- ***Spatial Locality***
  If a word is accessed, adjacent(near) words are likely accessed soon (e.g. Related data items (arrays) are usually stored together; instructions are executed sequentially)

Cache

- **The property of Locality of Reference makes the Cache memory systems work**
- **Cache is a fast small capacity memory that should hold those information which are most likely to be accessed**



PERFORMANCE OF CACHE

Memory Access

All the memory accesses are directed first to Cache
If the word is in Cache; Access cache to provide it to CPU
If the word is not in Cache; Bring a block (or a line) including that word to replace a block now in Cache

- **How can we know if the word that is required is there ?**
- **If a new block is to replace one of the old blocks, which one should we choose ?**

Performance of Cache Memory System

Hit Ratio - % of memory accesses satisfied by Cache memory system
Te: Effective memory access time in Cache memory system
Tc: Cache access time
Tm: Main memory access time

$$Te = Tc + (1 - h)\ Tm$$

Example: $Tc = 0.4\ \mu s$, $Tm = 1.2 \mu s$, $h = 0.85\%$

$$Te = 0.4 + (1 - 0.85) * 1.2 = 0.58 \mu s$$

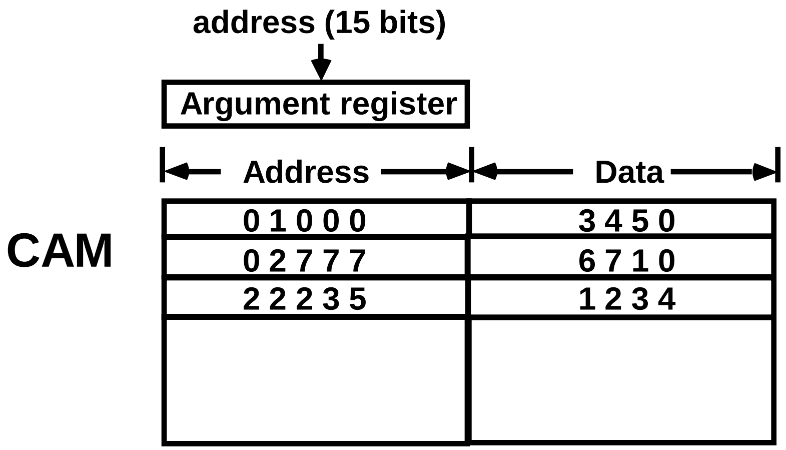
MEMORY AND CACHE MAPPING - ASSOCIATIVE MAPPLING -

Mapping Function :Specification of correspondence between main memory blocks and cache blocks

Associative mapping
Direct mapping
Set-associative mapping

Associative Mapping

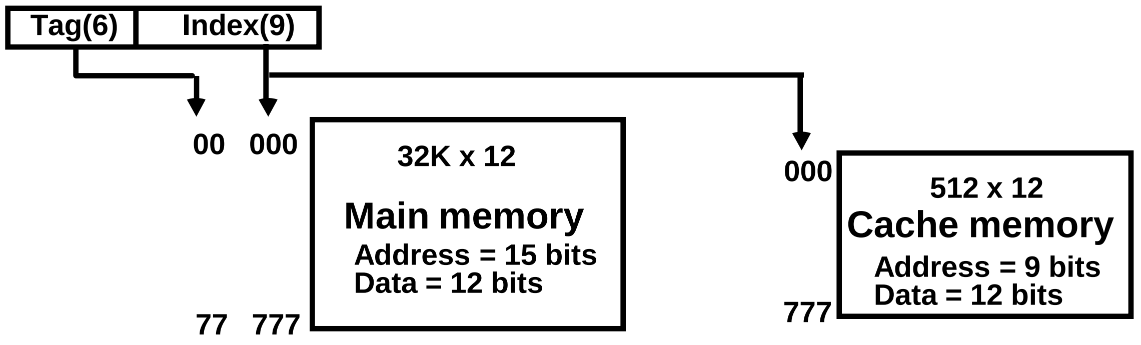
- **Any block location in Cache can store any block in memory**
  -> Most flexible
- **Mapping Table is implemented in an associative memory**
  -> Fast, very Expensive
- **Mapping Table**
  Stores both address and the content of the memory word



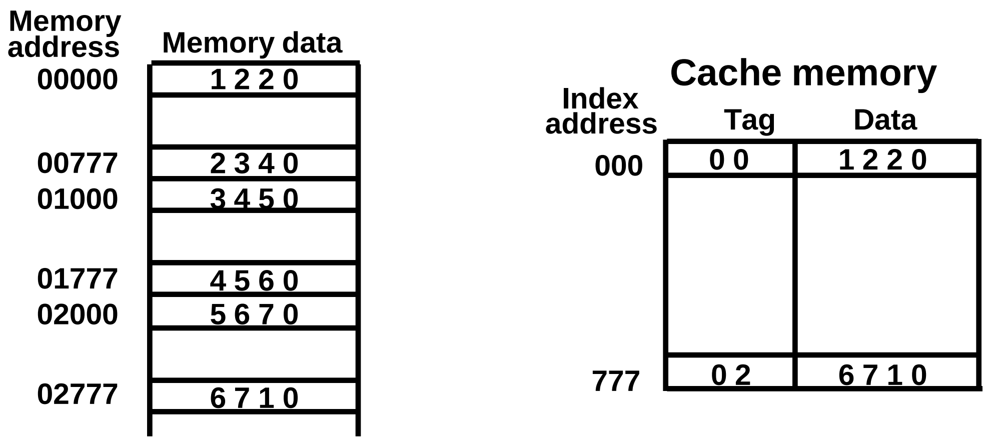
MEMORY AND CACHE MAPPING - DIRECT MAPPING -

- Each memory block has only one place to load in Cache
- Mapping Table is made of RAM instead of CAM
- n-bit memory address consists of 2 parts; k bits of Index field and n-k bits of Tag field
- n-bit addresses are used to access main memory and k-bit Index is used to access the Cache

Addressing Relationships



Direct Mapping Cache Organization

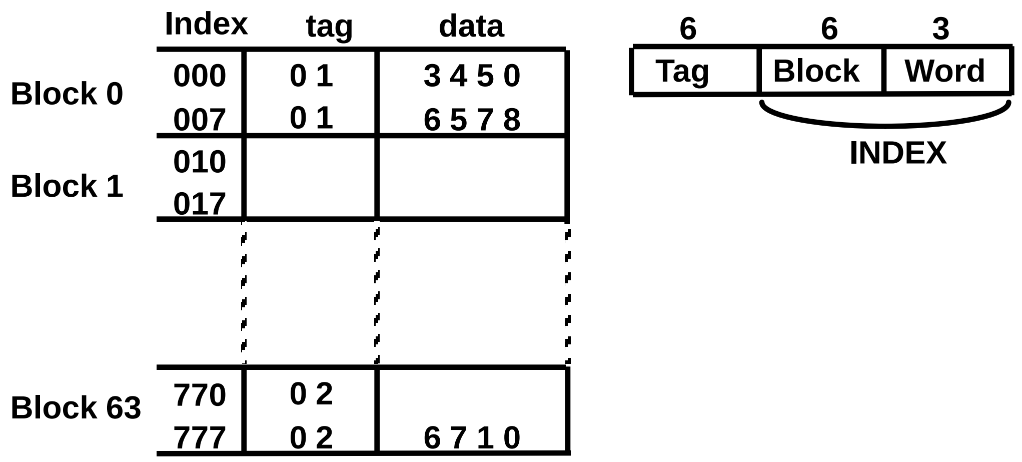


DIRECT MAPPING

Operation

- **CPU generates a memory request with (TAG;INDEX)**
- **Access Cache using INDEX ; (tag; data)**
  Compare TAG and tag
- **If matches -> Hit**
  Provide Cache[INDEX](data) to CPU
- **If not match -> Miss**
  M[tag;INDEX] <- Cache[INDEX](data)
  Cache[INDEX] <- (TAG;M[TAG; INDEX])
  CPU <- Cache[INDEX](data)

Direct Mapping with block size of 8 words



- Each memory block has a set of locations in the Cache to load

Set Associative Mapping Cache with set size of two

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Operation

- CPU generates a memory address(TAG; INDEX)
- Access Cache with INDEX, (Cache word = (tag 0, data 0); (tag 1, data 1))
- Compare TAG and tag 0 and then tag 1
- If tag i = TAG -> Hit, CPU <- data i
- If tag i $\neq$ TAG -> Miss,
  Replace either (tag 0, data 0) or (tag 1, data 1),
  Assume (tag 0, data 0) is selected for replacement,
  (Why (tag 0, data 0) instead of (tag 1, data 1) ?)
  M[tag 0, INDEX] <- Cache[INDEX](data 0)
  Cache[INDEX](tag 0, data 0) <- (TAG, M[TAG,INDEX]),
  CPU <- Cache[INDEX](data 0)

Many different block replacement policies are available
LRU(Least Recently Used) is most easy to implement

Cache word = (tag 0, data 0, *U0*);(tag 1, data 1, *U1*), Ui = 0 or 1(binary)

Implementation of LRU in the Set Associative Mapping with set size = 2

Modifications

```
Initially all U0 = U1 = 1
When Hit to (tag 0, data 0, U0), U1 <- 1(least recently used)
(When Hit to (tag 1, data 1, U1), U0 <- 1(least recently used))
When Miss, find the least recently used one(Ui=1)
    If U0 = 1, and U1 = 0, then replace (tag 0, data 0)
        M[tag 0, INDEX] <- Cache[INDEX](data 0)
        Cache[INDEX](tag 0, data 0, U0) <- (TAG,M[TAG,INDEX], 0); U1 <- 1
    If U0 = 0, and U1 = 1, then replace (tag 1, data 1)
        Similar to above; U0 <- 1
    If U0 = U1 = 0, this condition does not exist
    If U0 = U1 = 1, Both of them are candidates,
        Take arbitrary selection
```

CACHE WRITE

Write Through

When writing into memory

If Hit, both Cache and memory is written in parallel
If Miss, Memory is written
For a read miss, missing block may be overloaded onto a cache block

Memory is always updated
-> Important when CPU and DMA I/O are both executing

Slow, due to the memory access time

Write-Back (Copy-Back)

When writing into memory

If Hit, only Cache is written
If Miss, missing block is brought to Cache and write into Cache
For a read miss, candidate block must be written back to the memory

Memory is not up-to-date, i.e., the same item in Cache and memory may have different value