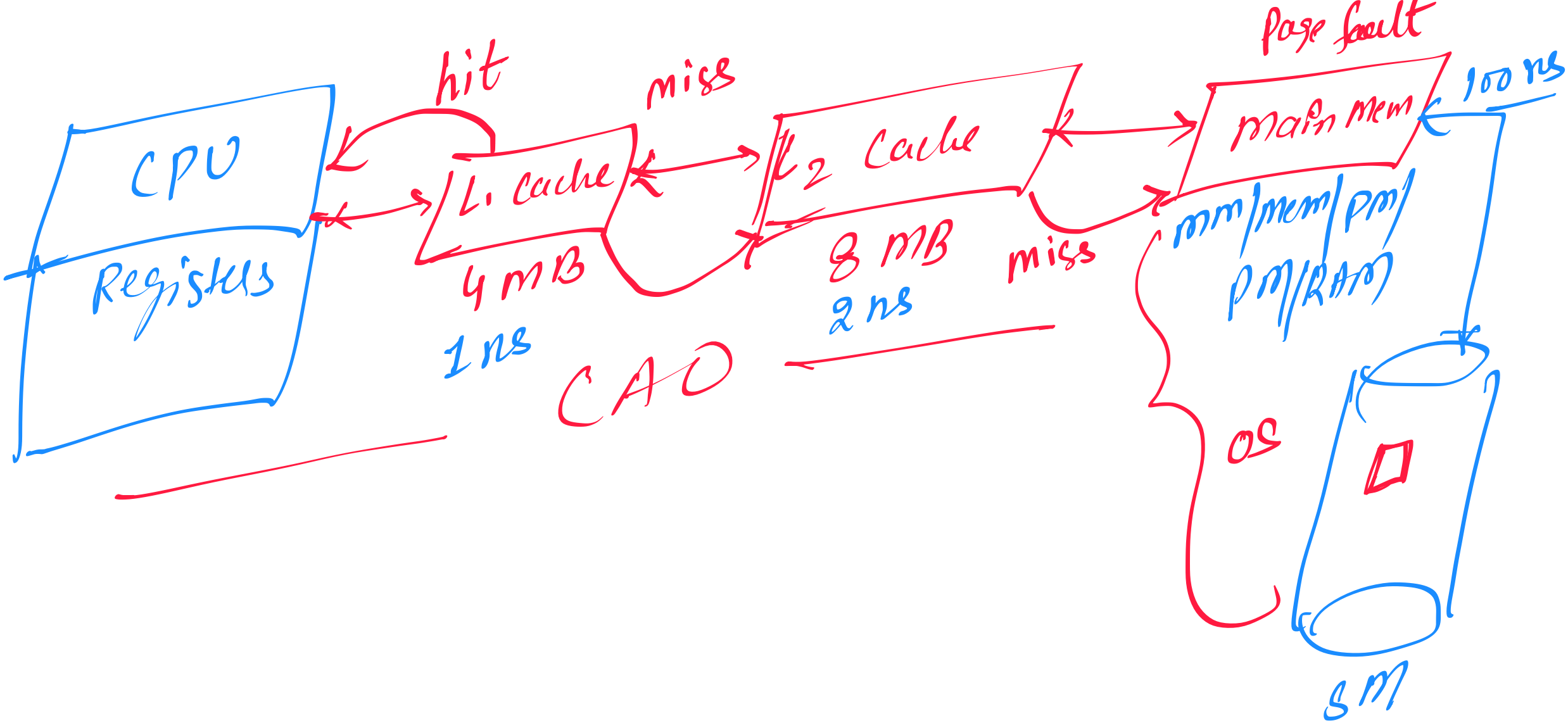
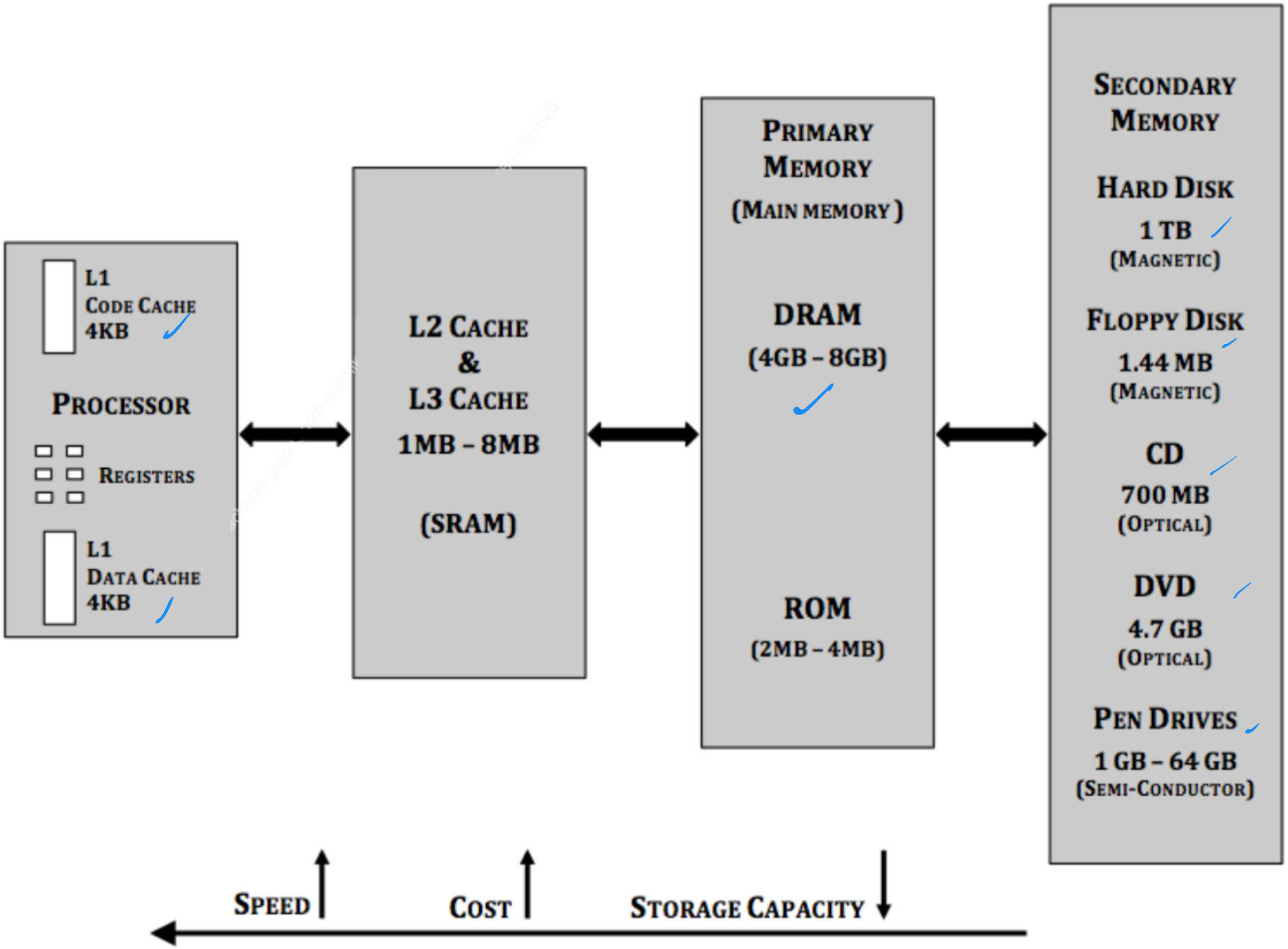
MEMORY ORGANIZATION

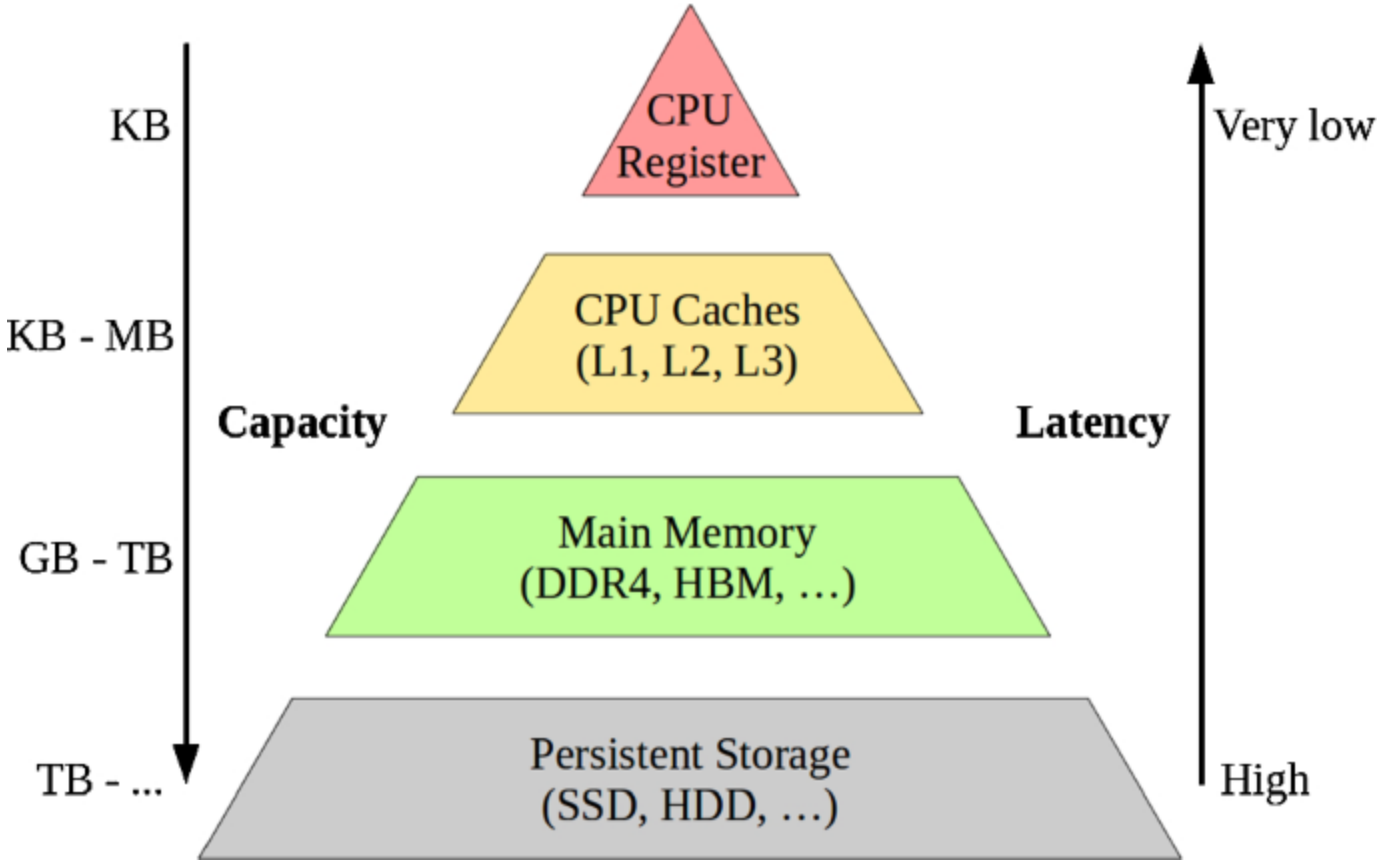
- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Virtual Memory
- Memory Management Hardware


CPU
Registers
hit
L1 cache
4 MB
1 ns
miss
L2 cache
8 MB
2 ns
miss
CAO
Page fault
main mem
100 ns
mm/mem/pm/
pm/RAM
OS
SM

MEMORY HIERARCHY

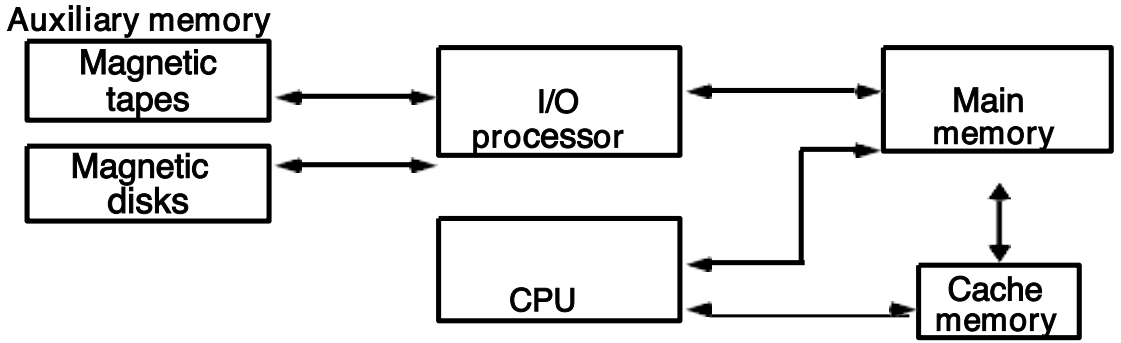


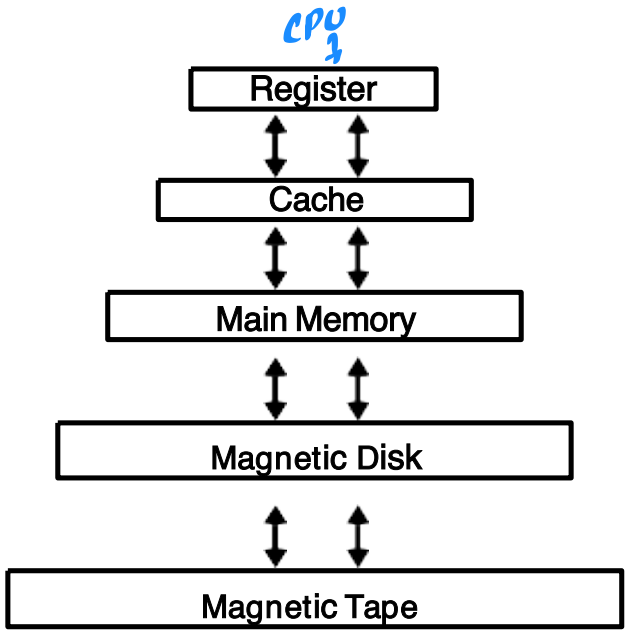
MEMORY HIERARCHY

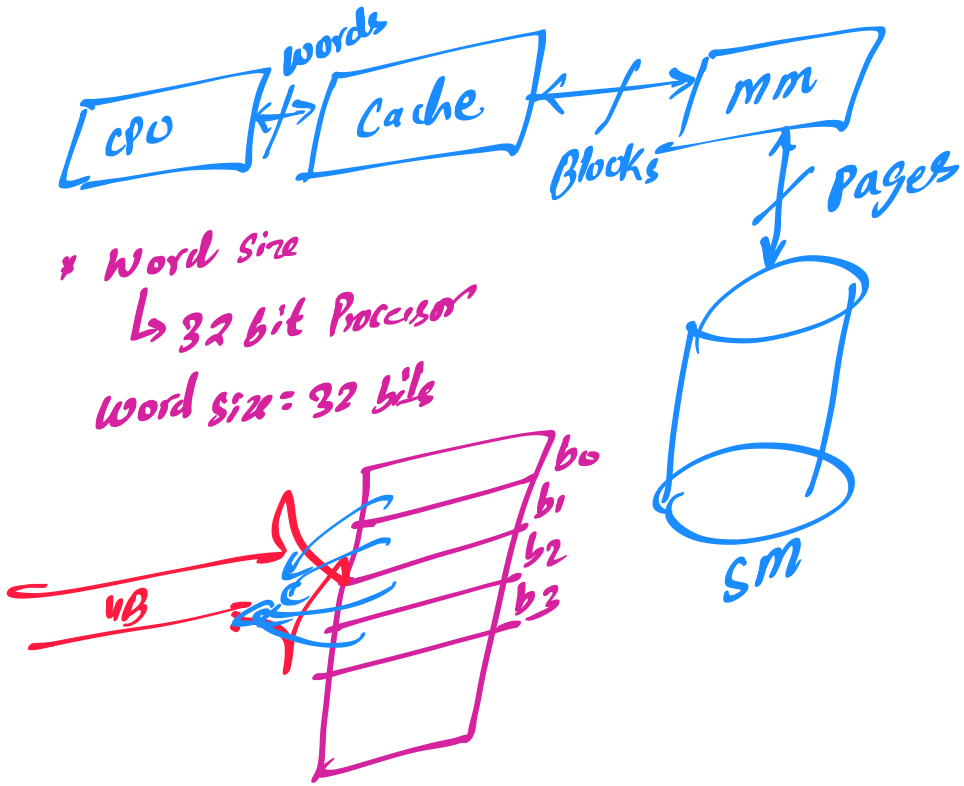


MEMORY HIERARCHY

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



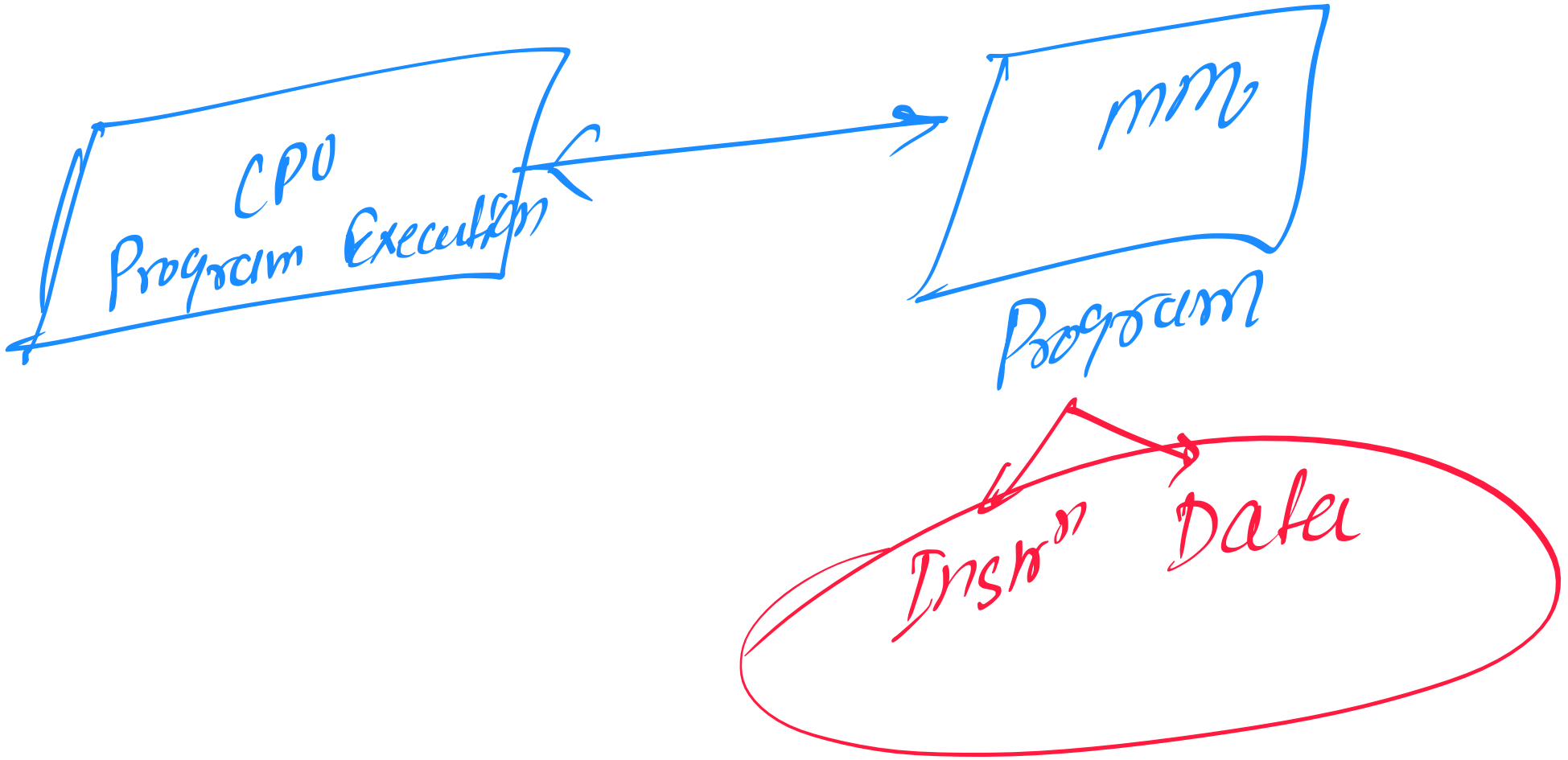



CPU
words
Cache
MM
Blocks
Pages
SM
* Word size
↳ 32 bit Processor
word size = 32 bits
b0
b1
b2
b3
WB

Memory Unit is an essential component in any digital computer since it is needed for storing programs and data. Memory is just like a brain which is useful for storing information into the computer.

Several types of memory are there:

1. **Main memory/Primary memory/Volatile memory** is used to communicate directly with the CPU. CPU is mainly used to execute any task on the computer. CPU can execute only that program which will be present into the main memory. If we are going to save any program by default, it gets saved into the secondary memory, but during execution, DMA/Operating system transfers the program to primary memory.
2. **Auxiliary memory/secondary memory/Non-volatile memory** is used to store the data permanently into the computer. Examples are Magnetic disk and Magnetic tapes. Previously everyone is using magnetic tapes but now a days magnetic disks are used. CPU cannot access the data of auxiliary memory.
3. **Cache memory** is small and faster memory which has higher accessibility. Cache memory is very fast memory and size of the memory is very small. It is mainly used to store the data which is frequently used. Main memory access time is slower but cache memory access time is fast. So if there are any frequently used instructions then operating system/IO transfers the instructions of main memory to cache memory so that access will be fast.
4. **I/O processor** is mainly used to transfer the data from secondary memory to main memory.


CPU
Program Execution
MM
Program
Instn
Data

Program Execution

100ms

90% time
Mem. Access

10% time
Execution (CPU)

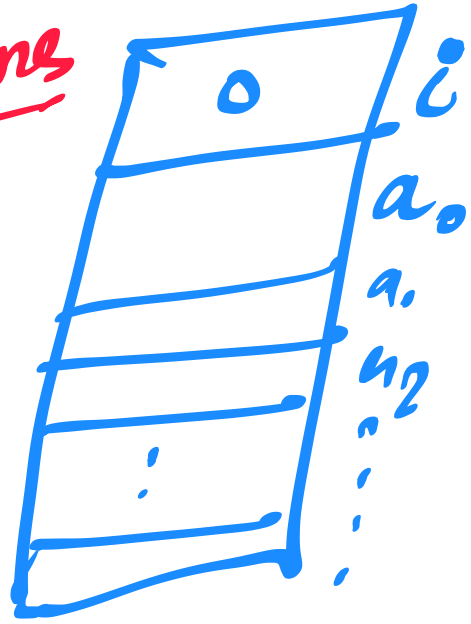
int a[10] = {... } 90 ms

10 ms

Tm = 100ms

Ex:

```
for (i=0; i<10; i++)
{
    a[i]++;
}
```



for (i=0; i<10; i++)

{

a[i]++;

}

each iteration

✓ LOAD i, R1

JNZ R1

INC R1

✓ STORE R1, i

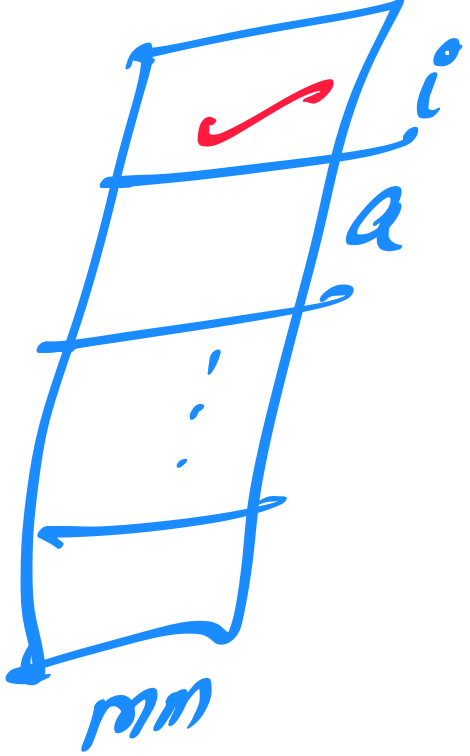
LOAD a, R2

INC a

✓ STORE a

a[0]++;

a[1]++;



4 mA

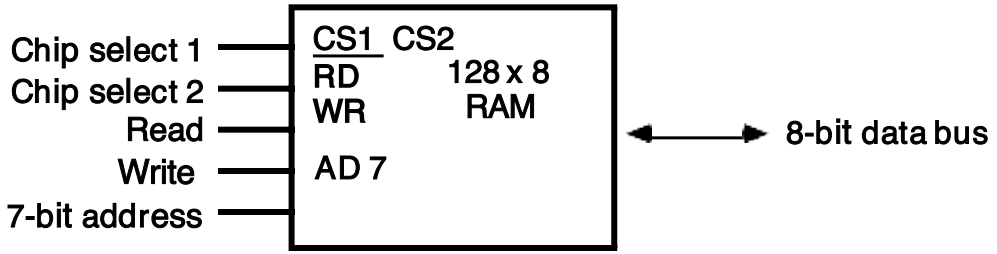
* 4 × 10 ns

= 40 ns

MAIN MEMORY

RAM and ROM Chips

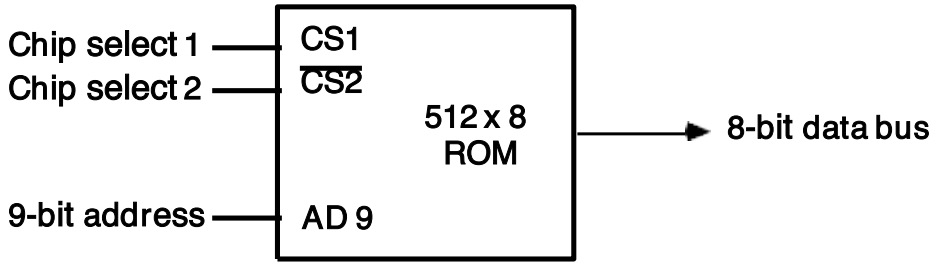
Typical RAM chip



Types of Memory
- Static
- Dynamic

CS1	$\overline{CS2}$	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 Write	High-impedence Input
1	0	0	1		data to RAM
1	0	1	x	Read Inhibit	Output data from RAM
1	1	x	x		High-impedence

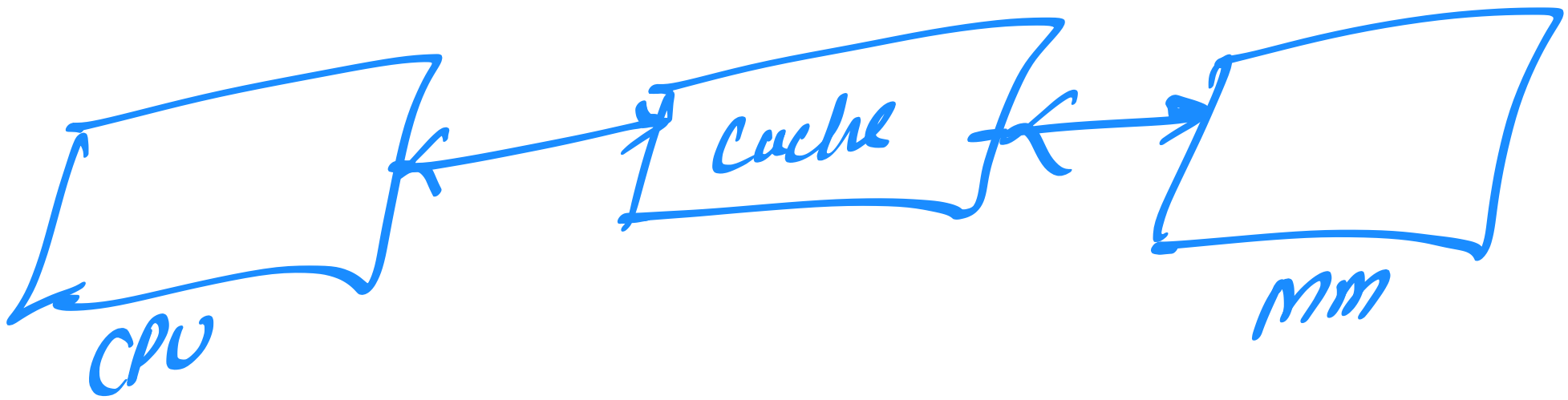
Typical ROM chip



Uses of ROM
- Bootstrap Loader
- Computer Startup

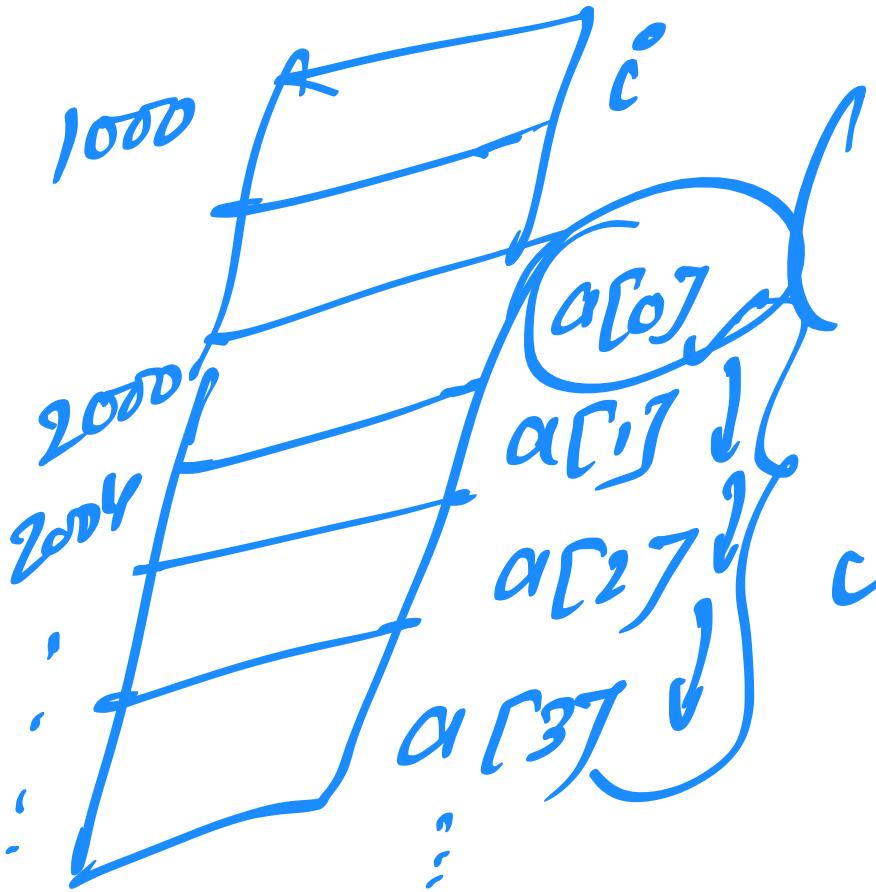
10 Iterations

* $10 * 40 ns = 400 ns$



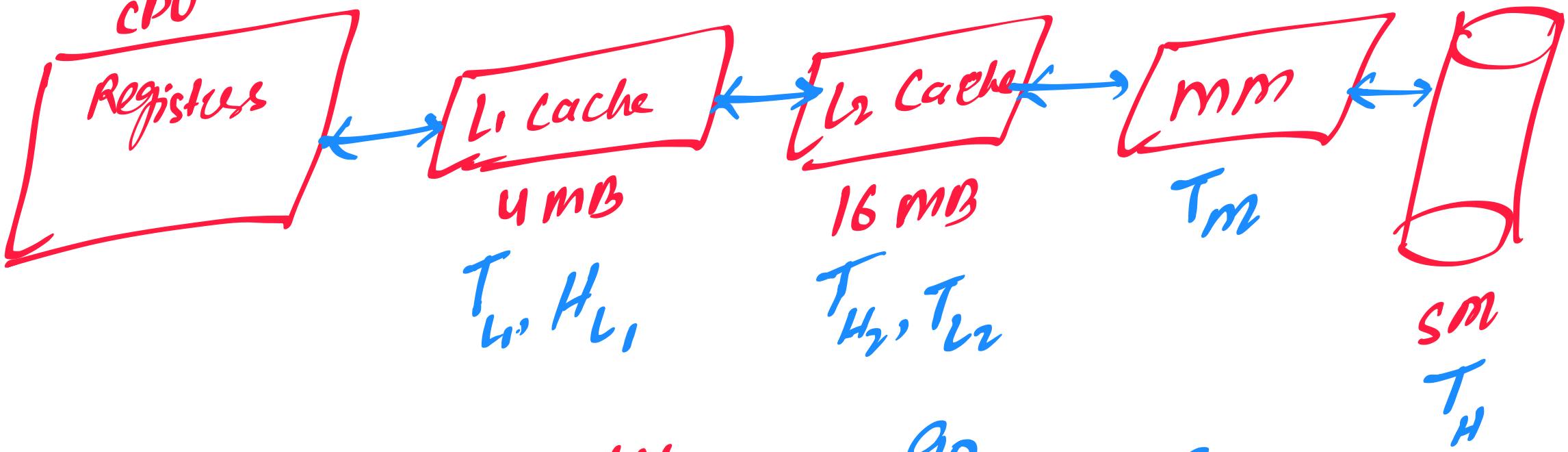
Locality of Reference

- Spatial Locality
- Temporal Locality



Space

(Time)



* Hit Ratio = $\frac{\#\ hit}{\#\ Access} = \frac{90}{100} = 0.9$

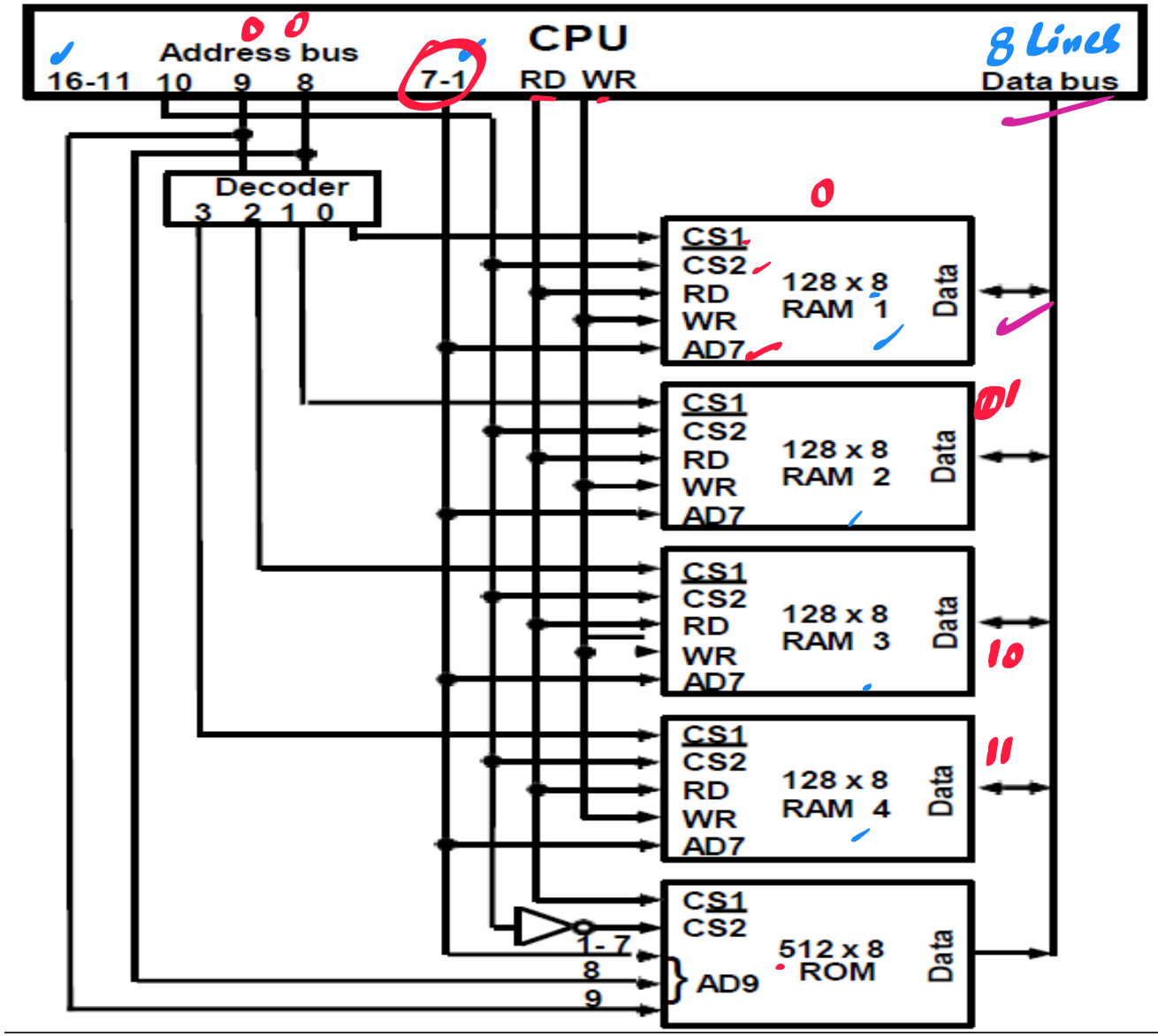
* Miss Ratio = (1 − Hit Ratio) = $\frac{\#\ miss}{Total\ Access} = \frac{10}{100}$

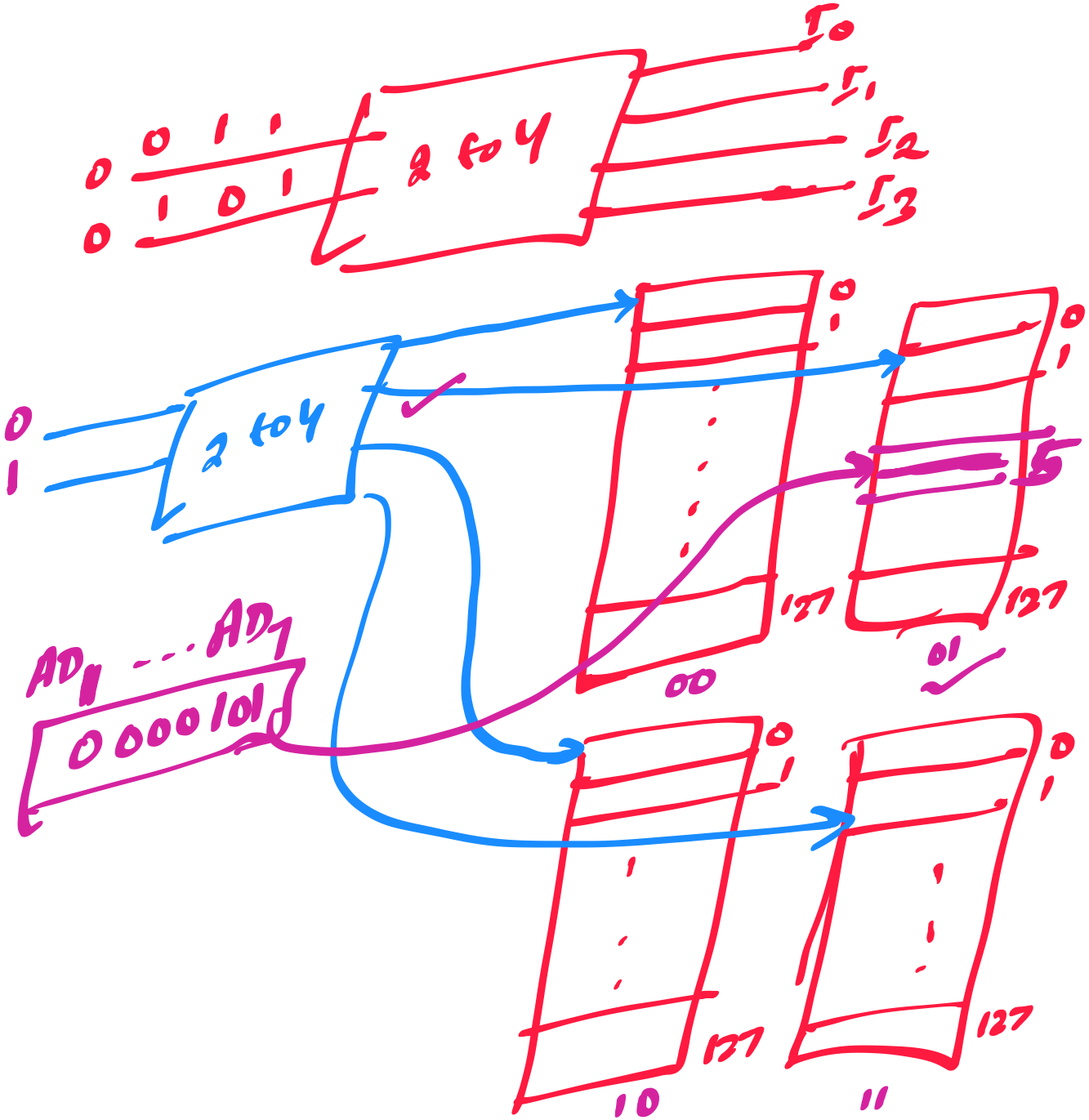
(1 − 0.9) = 0.10 = 0.10

The **static RAM** consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. It is very fast but the cost is high. Cache memory is designed with the help of SRAM. The **dynamic RAM** stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. Main memory is designed with the help of DRAM.

RAM is a volatile memory when we switch off the computer the entire contents of memory will be lost but ROM is a permanent memory that means it stores contents permanently. Most portion of the memory is RAM only but less is ROM.
Bootstrap loader: Among other things, the ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again. The startup of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use. RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size. To demonstrate the chip interconnection, we will show an example of a 1024 x 8 memory constructed with 128 x 8 RAM chips and 512 x 8 ROM chips.

CONNECTION OF MEMORY TO CPU




0
0
2 to 4
I_0
I_1
I_2
I_3
0
1
2 to 4
0
0
127
00
0
0
127
01
AD_{11} ... AD_1
0 0 0 0 1 0 1
0
0
127
10
0
0
127
11

* Chip → 128*8 ✓

* 1 M*8 ✓
  - 1 M → # cells
  - 8 → Size of each cell (bits)

Chips Required = $\frac{1\,M*8}{128*8} = \frac{2^{20}*8}{2^7*8} = 2^{13}$

MEMORY ADDRESS MAP

Address space assignment to each memory chip

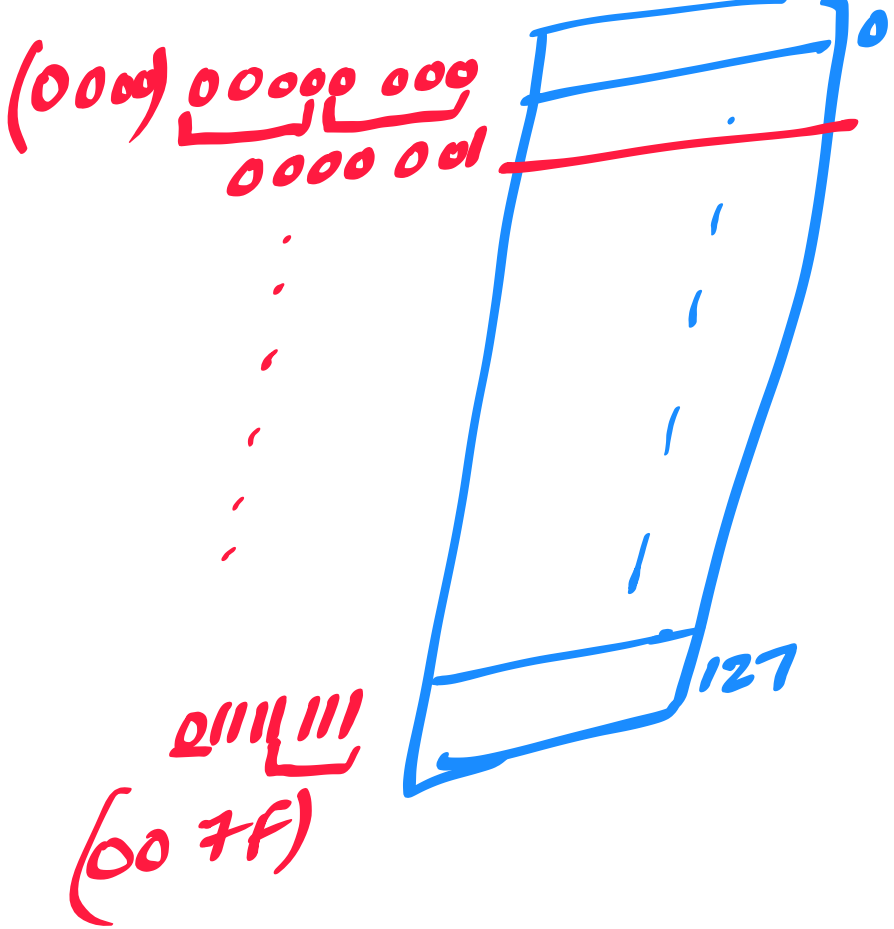
Example: 512 bytes RAM and 512 bytes ROM

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	x
RAM 2	0080 - 00FF	0	0	1	x	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		
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x		

Memory Connection to CPU

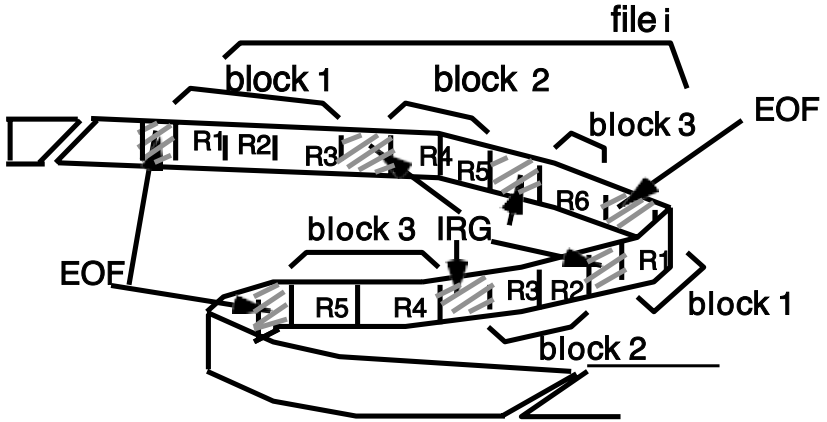
- RAM and ROM chips are connected to a CPU through the data and address buses

- 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


0
(0000 0000 000
0000 001
127
0111 111
(00 7F)

AUXILIARY MEMORY

Information Organization on Magnetic Tapes



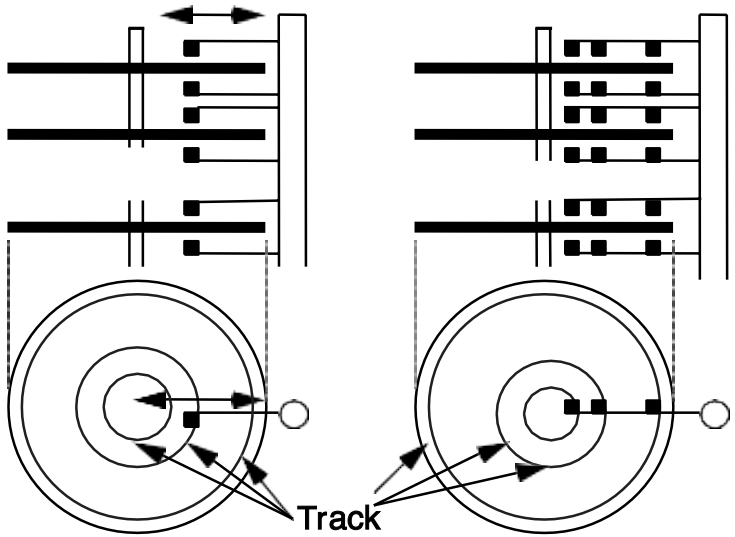
Characteristics of Auxiliary Memory

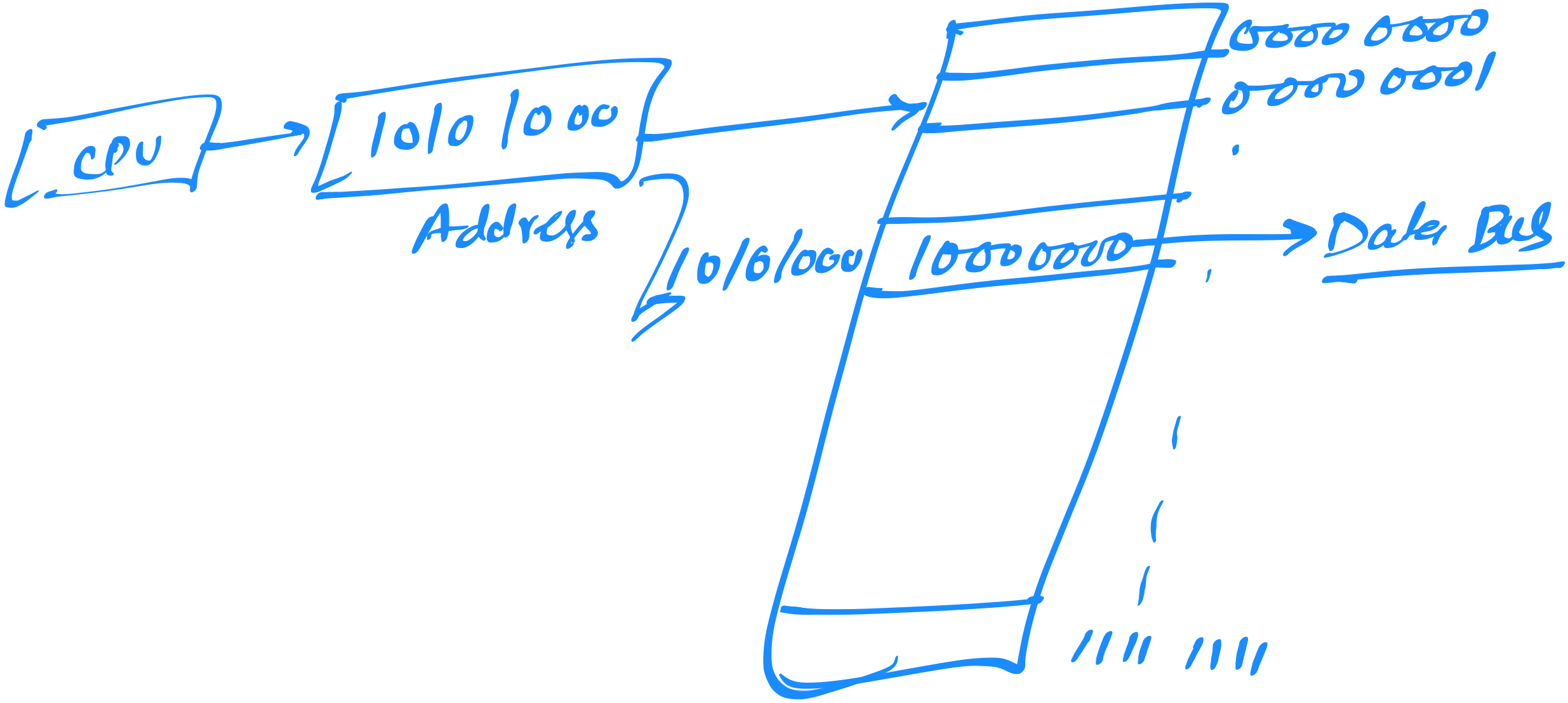
1. Access Mode
2. Access Time
3. Transfer rate
4. Capacity
5. Cost

- Access Time
  - Seek Time
  - Transfer Time

Organization of Disk Hardware

Moving Head Disk Fixed Head Disk

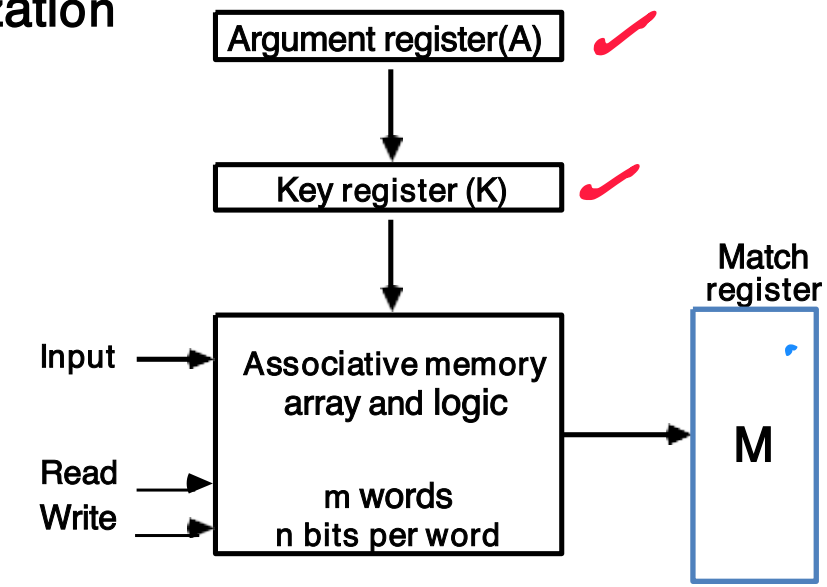



CPU
1010 1000
Address
1010 1000
0000 0000
0000 0001
10000000
Data Bus
1111 1111

ASSOCIATIVE MEMORY

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

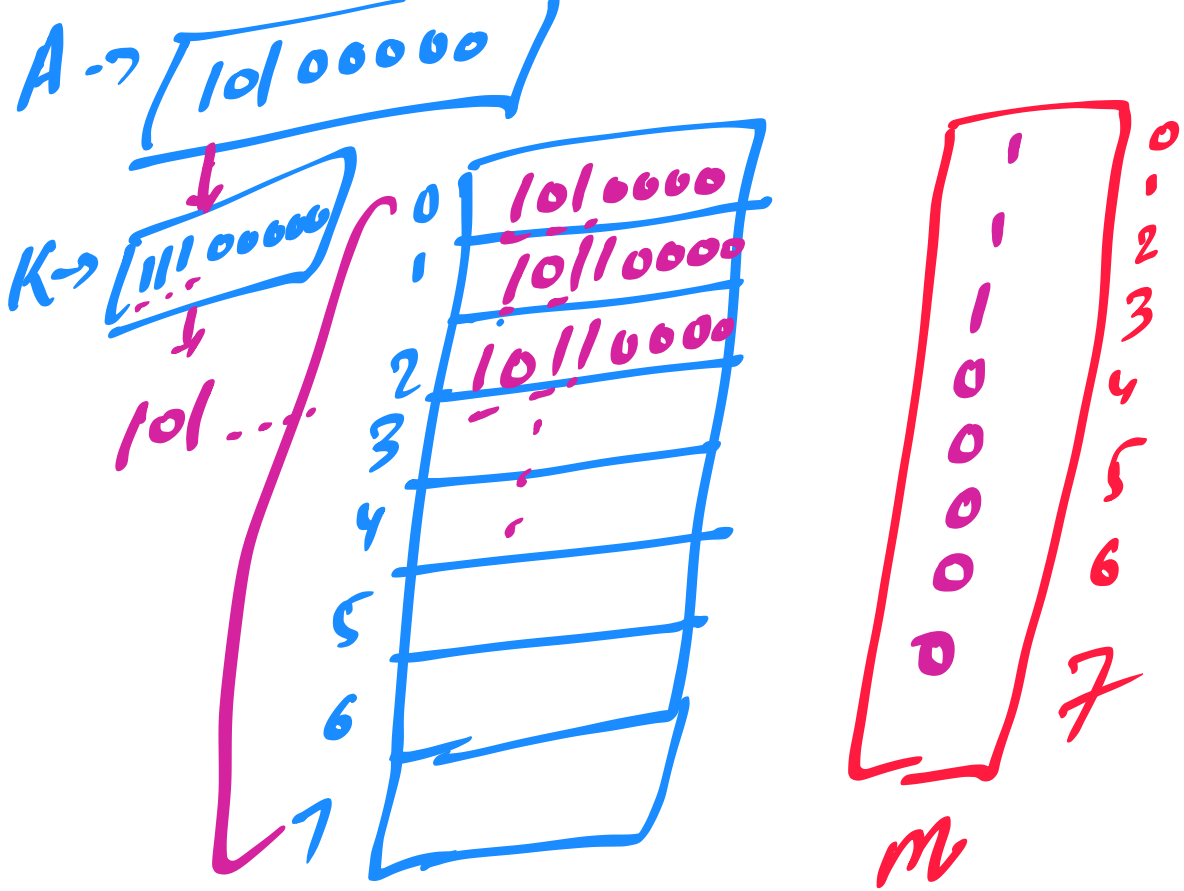
Hardware Organization



A: 101 111100
K: 111 000000
Word 1: 100 111100 (No Match)
Word 2: 101 000001 (Match)

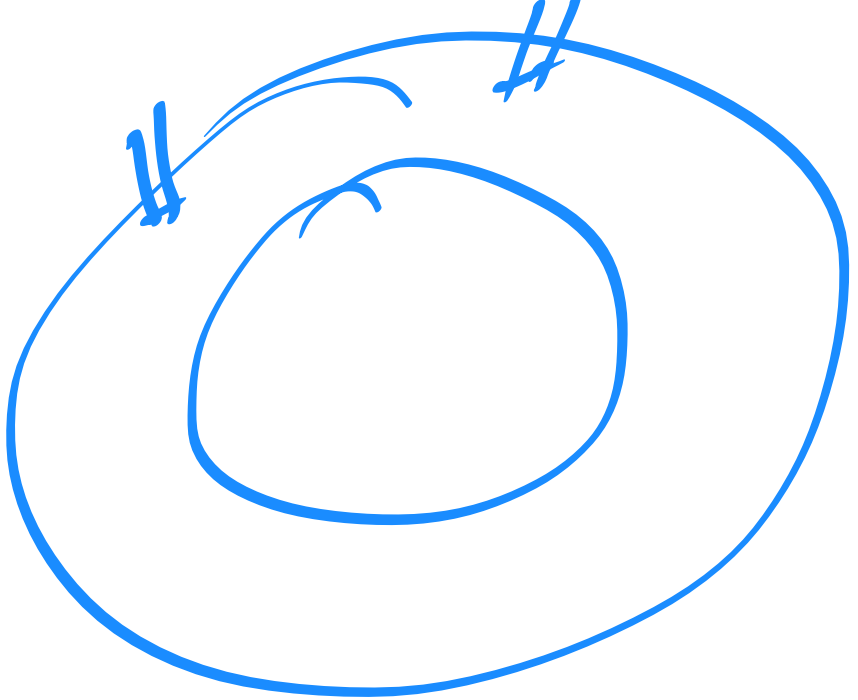
- Compare each word in CAM in parallel with the content of A(Argument Register)
- If CAM Word[i] = A, M(i) = 1
- Read sequentially accessing CAM for CAM Word(i) for M(i) = 1

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)

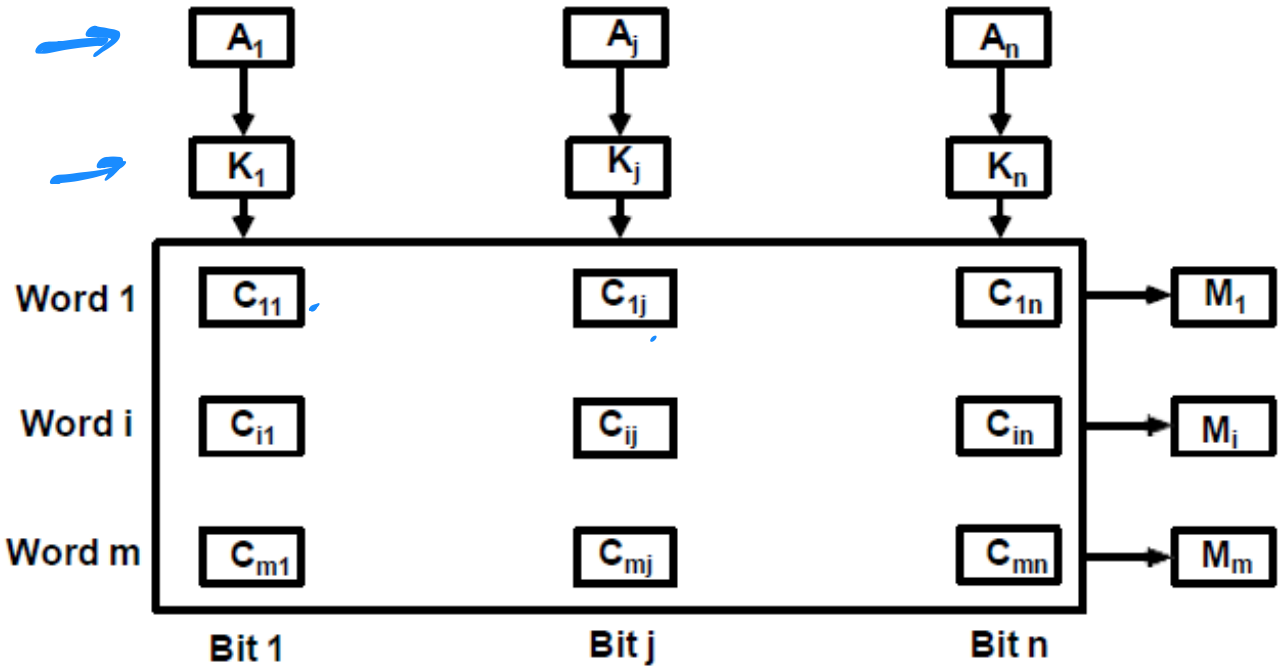

A->
101 00000
K->
111 00000
101...
0
1
2
3
4
5
6
7
101 0000
1011 0000
1011 0000
M
1
1
1
0
0
0
0
0
0
1
2
3
4
5
6
7

A = 1010 0000

K = 0010 1010



ORGANIZATION OF CAM



X-NOR $\Rightarrow$ $x'y' + xy$

MATCH LOGIC

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

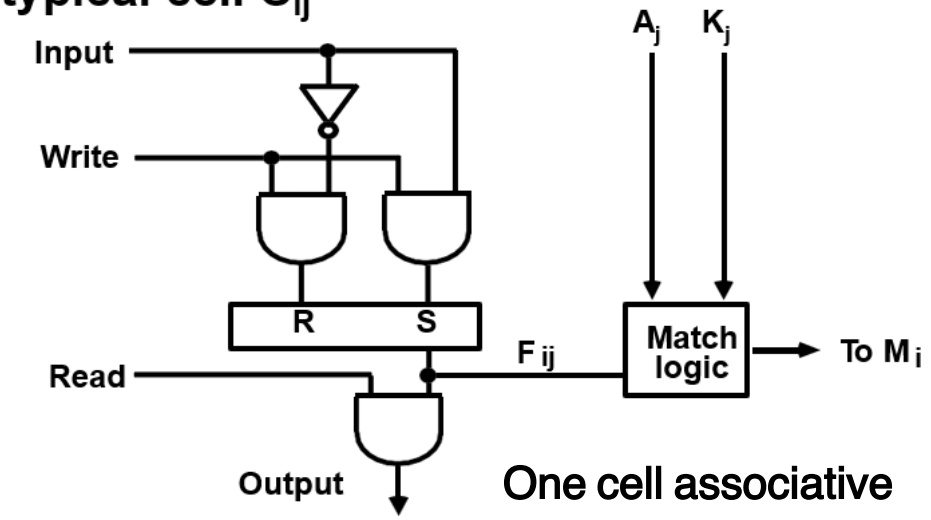
$$M_i = x_1 x_2 x_3 \ldots x_n$$

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

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

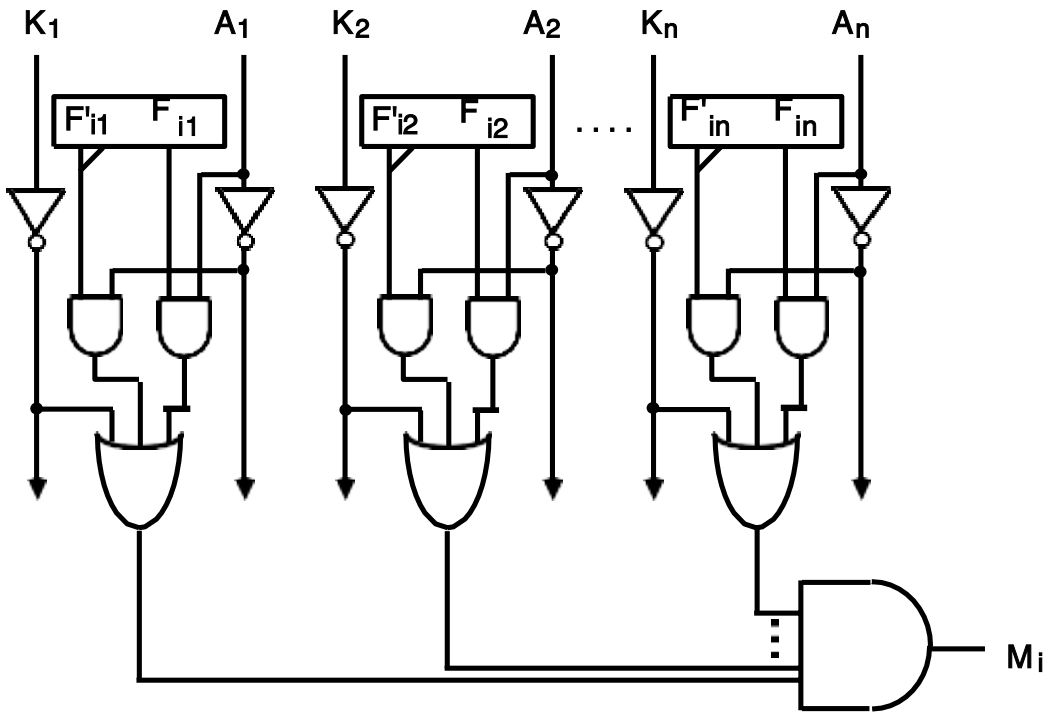
$$M_i = \prod_{j=1}^{n} (A_j F_{ij} + A_j' F_{ij}' + K_j')$$

Internal organization of a typical cell C_{ij}



One cell associative memory

MATCH LOGIC



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)

```
for( i=0; i<10; i++)
{
    a[i]++;
}
```

Ex: int : 4 Bytes

Cache Memory = 128 Bytes

Block Size = 16 Byte

1st iteration

i=0, a[0]++;

2nd iteration

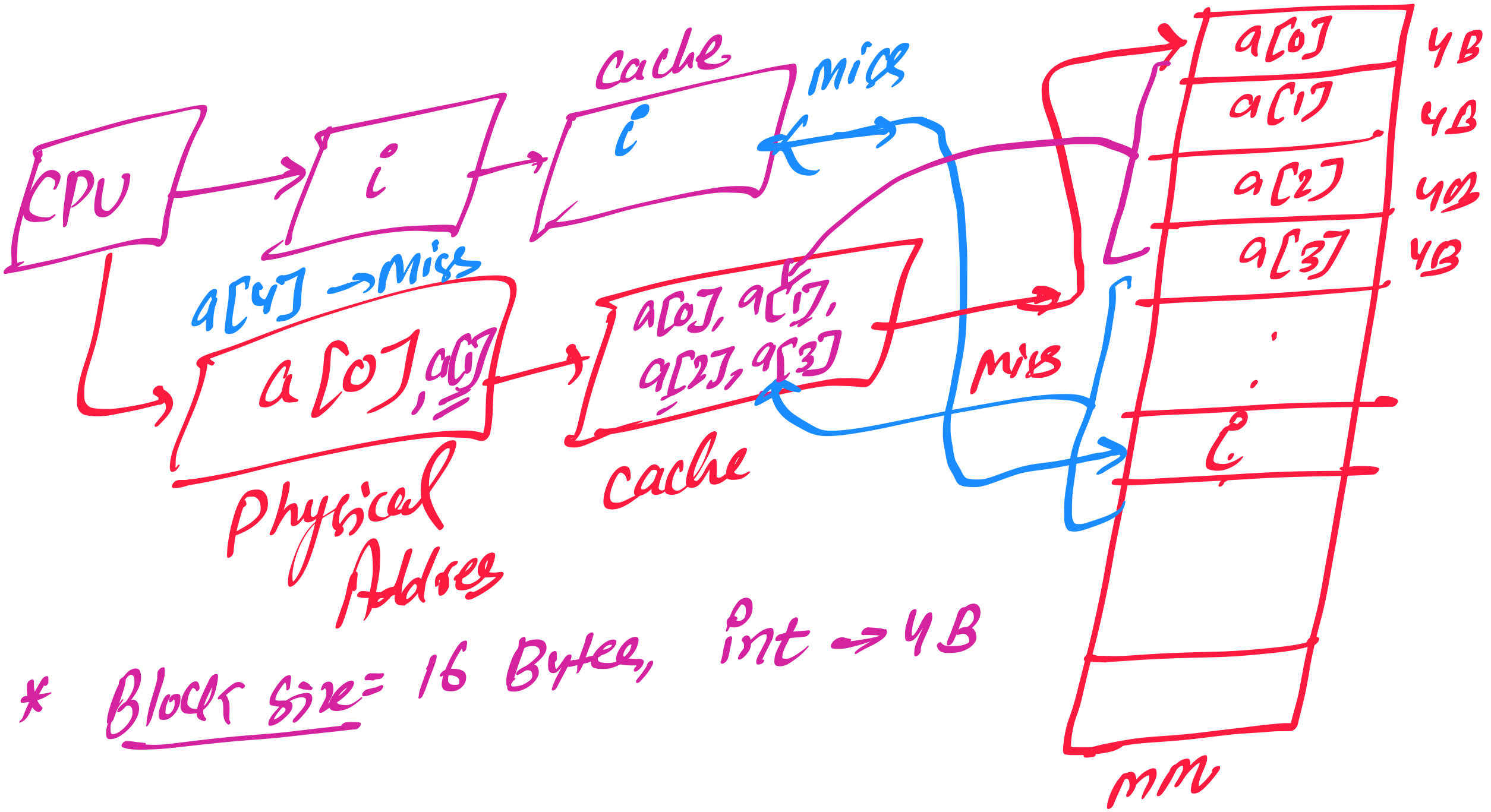
i=1, a[1]++

3rd iteration

i=2, a[2]++;

4th iteration

i=3, a[3]++.



* Block size = 16 Bytes, int → 4B

Ex:

```
for(i=0; i<128; i++)
{
    for(j=0; j<64; j++)
    {
        a[i][j]++;
    }
}
```

Cm Block Size = 128 Bytes

Integer → 4B

Cm = 4mB

miss ?

$128 * 64 \Rightarrow 2^7 * 2^6 = 2^{13}$ elements

1 Block Size = 128 Bytes

elements / Block = $\frac{128\,B}{4\,B} = 2^5 = 32$

a[0][0] → miss

a[0][1]
⋮
a[0][31] → Hit

Blocks = $\frac{2^{13}}{2^5} = 2^8$

$= 256$ miss

Ex: Temporal Locality :-

if any variable i is accessed at time t_0.

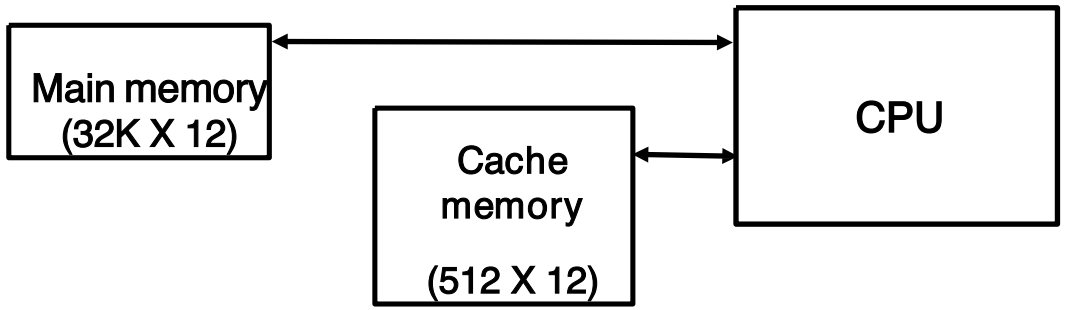
then there is high probability that this same variable will be accessed in near future at time t_1, t_2, t_3 - - -

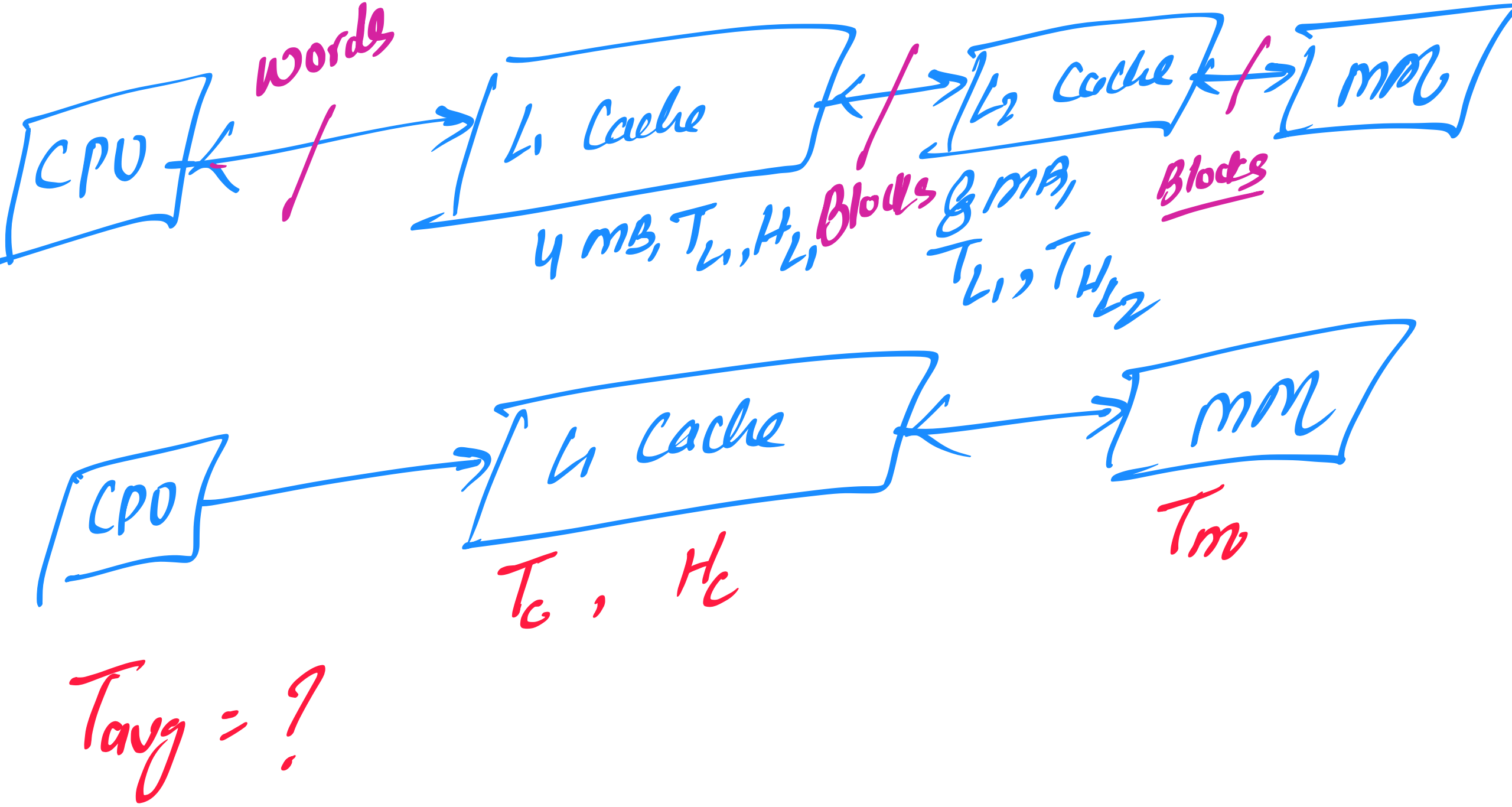
Spatial Locality :-

* if at time to a variable a[0] is accessed, then there is high probability that the variables a[1], a[2], a[3] will also be accessed in near future at t_1, t_2, t_3 ...

Cache

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




CPU
words
L1 Cache
L2 Cache
MM
4 MB, TL1, HL1
Blocks
8 MB,
Blocks
TL1, THL2
CPU
L1 Cache
MM
Tc , Hc
Tm
Tavg = ?

T_{avg} ?

Case-(i) Cache Hit | $T_c * H_c$

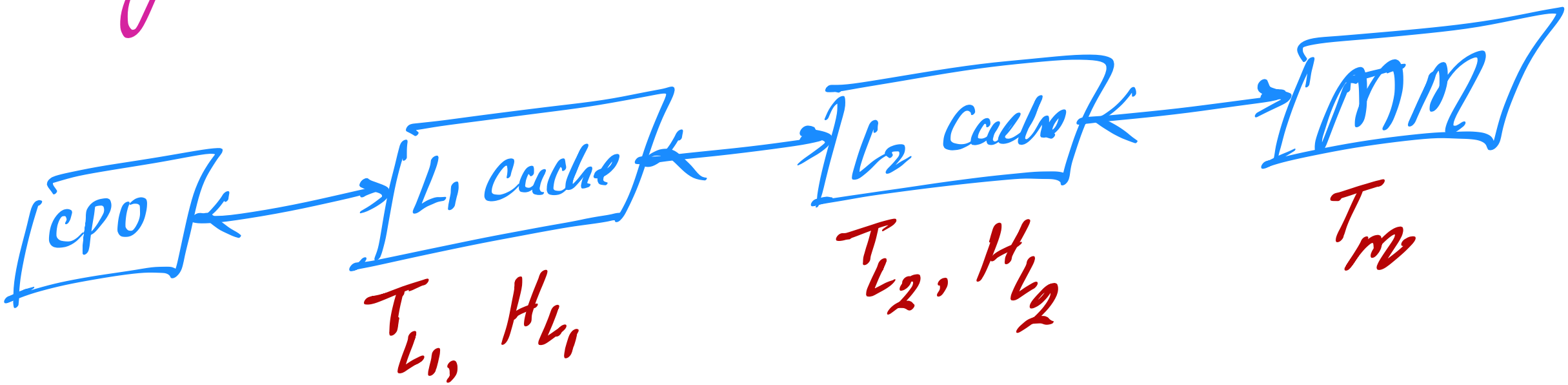
Case-ii Cache miss $\Rightarrow (1-H_c)(\underbrace{T_m + T_c}_{\text{miss Penalty}})$

$$T_{avg} = H_c * T_c + (1-H_c)(T_m + T_c)$$

H_c : Hit Ratio of Cache, T_m = Memory Access Time

T_c : Cache Access Time

$$T_{avg} = H_c * T_c + (1 - H_c)(T_m + T_c)$$



$T_{avg} = ?$

* Case-(i) L_1 Hit

* Case-ii L_1 miss

  a. L_2 Hit

  b. L_2 miss

$$T_{avg} = T_{L_1} * H_{L_1} + (1 - H_{L_1}) \left[H_{L_2} (T_{L_2} + T_{L_1}) + (1 - H_{L_2}) (T_m + T_{L_2} + T_{L_1}) \right]$$

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

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

$$h = \text{no. of hits}/(\text{no. of hits} + \text{no. of misses})$$

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

$$Te = 0.4*0.85 + (1 - 0.85) * (1.2+0.4) = 0.58 \text{ s}$$

MEMORY AND CACHE MAPPING

Mapping Function:

Specification of correspondence between main memory blocks and cache blocks

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

1. Associative Mapping

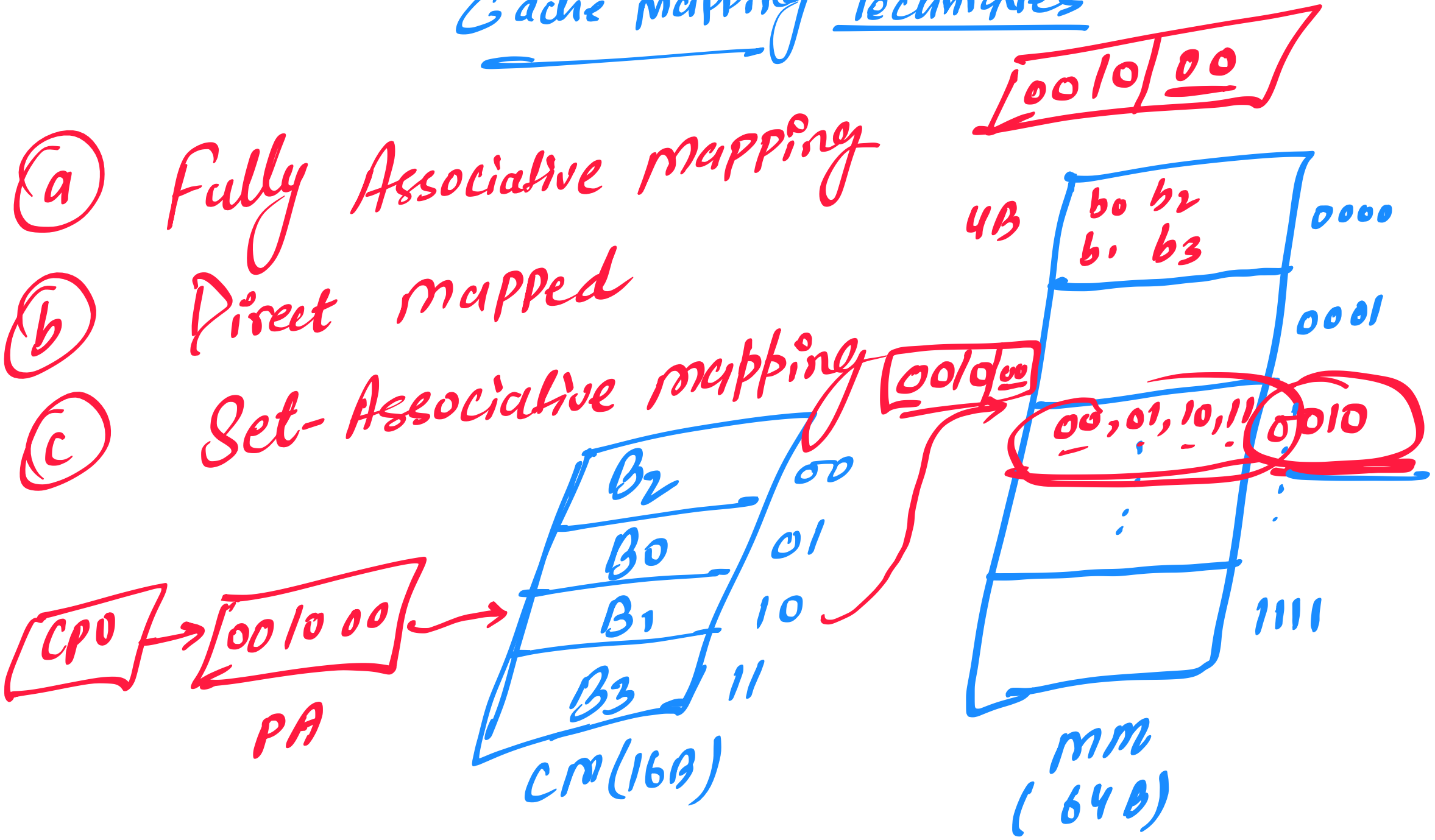
- Any block location in the Cache can store any block in memory
- Mapping Table: Stores both address and the content of the memory word
- Mapping Table is implemented in an associative memory
- Fast, very Expensive
- Most flexible

Cache mapping Techniques

(a) Fully Associative mapping

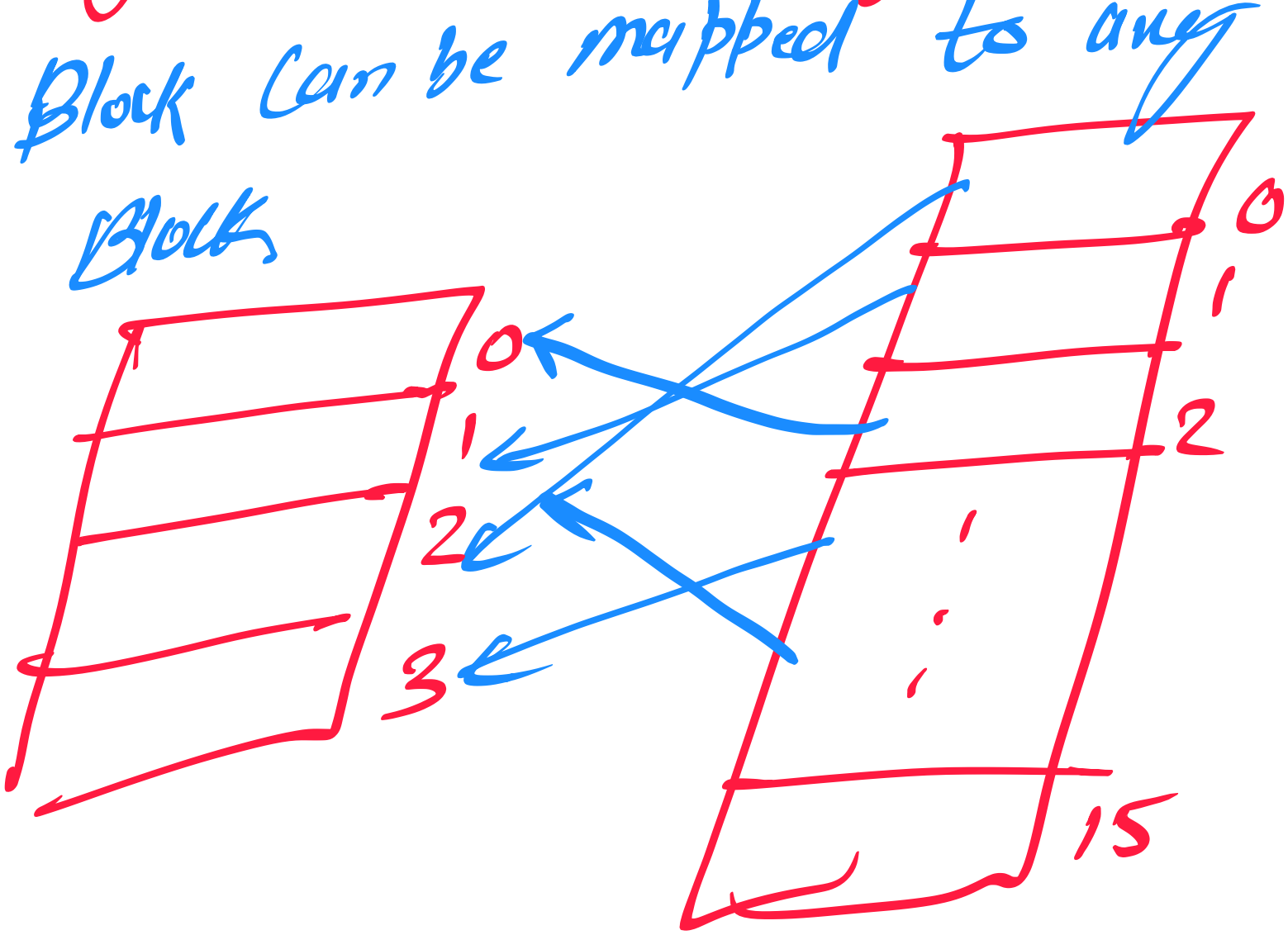
(b) Direct mapped

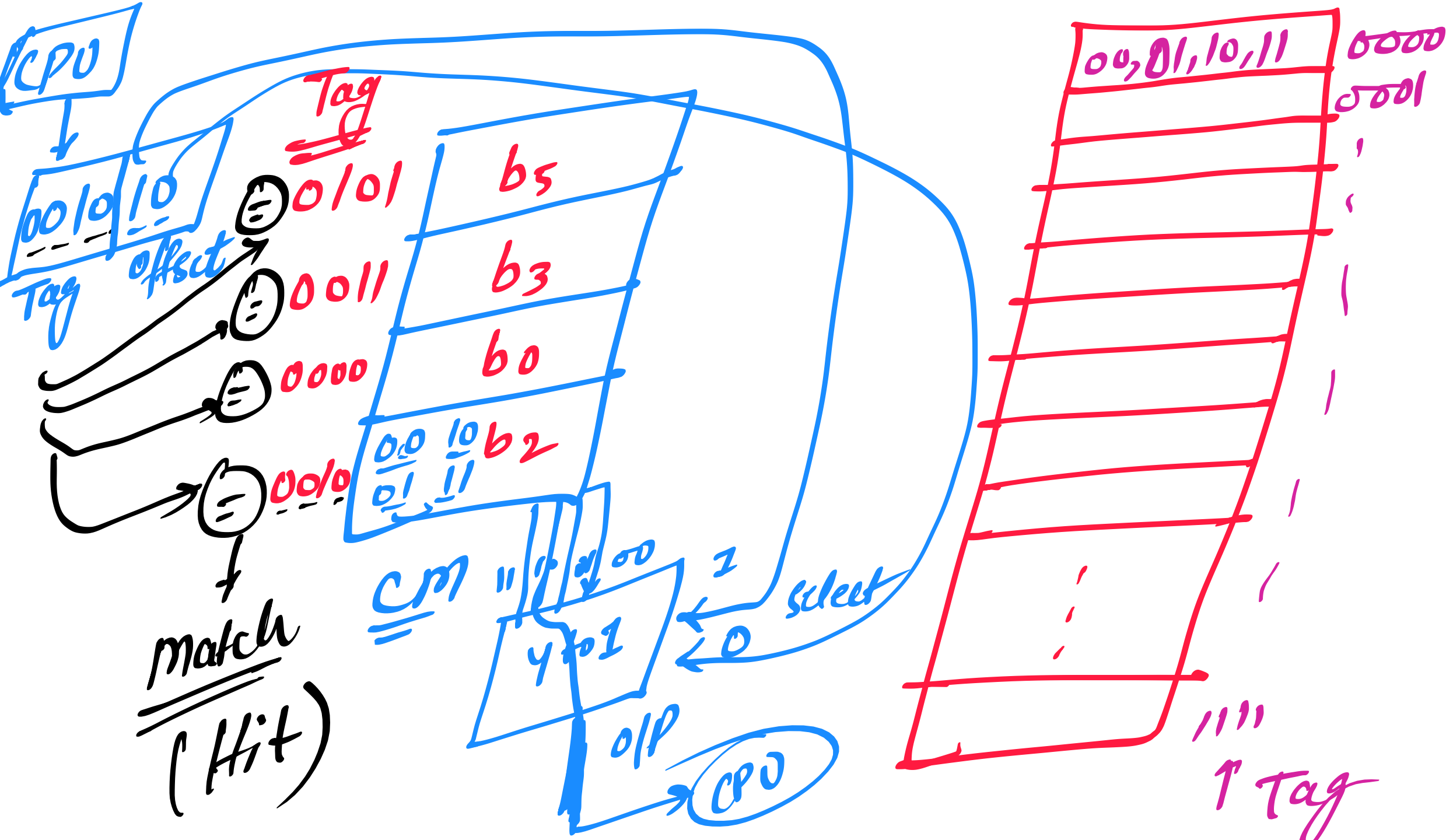
(c) Set-Associative mapping



ⓐ Fully Associative mapping

* Any memory Block can be mapped to any Cache memory Block




CPU
Tag
0010
10
Tag
offset
0101
0011
0000
0010
b5
b3
b0
00 10
01 11
b2
match
(Hit)
CPU
11 00
y to 1
I
select
O
O/P
CPU
00,01,10,11
0000
0001
1111
Tag

* Fully Associative Cache :-

$\Rightarrow$ offset = $\log_2$(Block size)

Ex: MM = 4 GB, Cache Memory = 4 MB

#C.Blocks = 4096 (2^{12})

Tag field = ?

PA = $\log_2$ (MM size)

= $\log_2$ (2^{32} B)

= 32 bit

Block size = ?

offset = $\log_2(2^{10}_B) = 10$ bits

Blocks = 2^{12}

CM Size = 4 MB = 2^{22} B

Block size = $\frac{\text{CM size}}{\text{\# Blocks.}}$ = $\frac{2^{22}\,B}{2^{12}}$ = 2^{10} B

\# Blocks = $\frac{\text{CM size}}{\text{Block size}}$

1. Associative Mapping

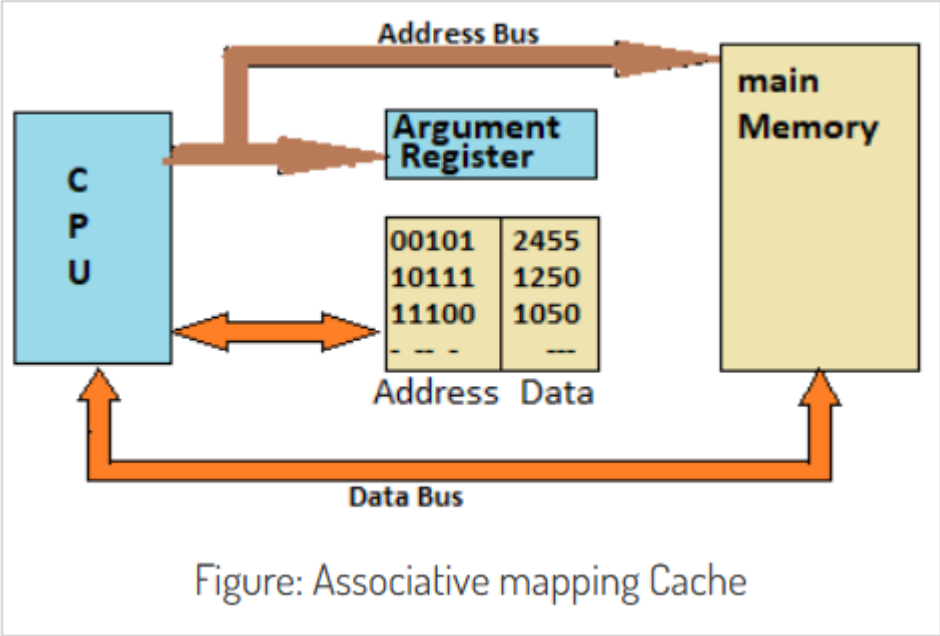
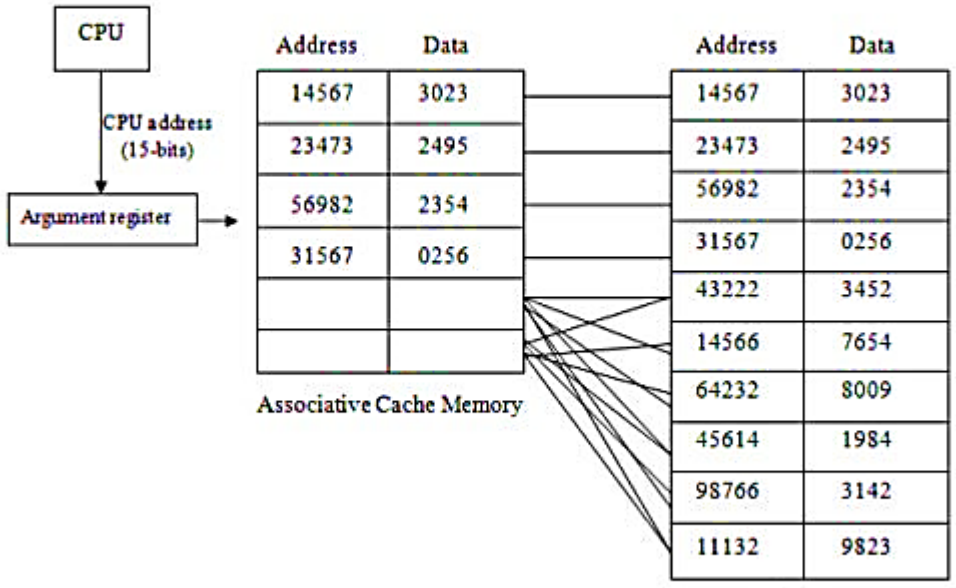


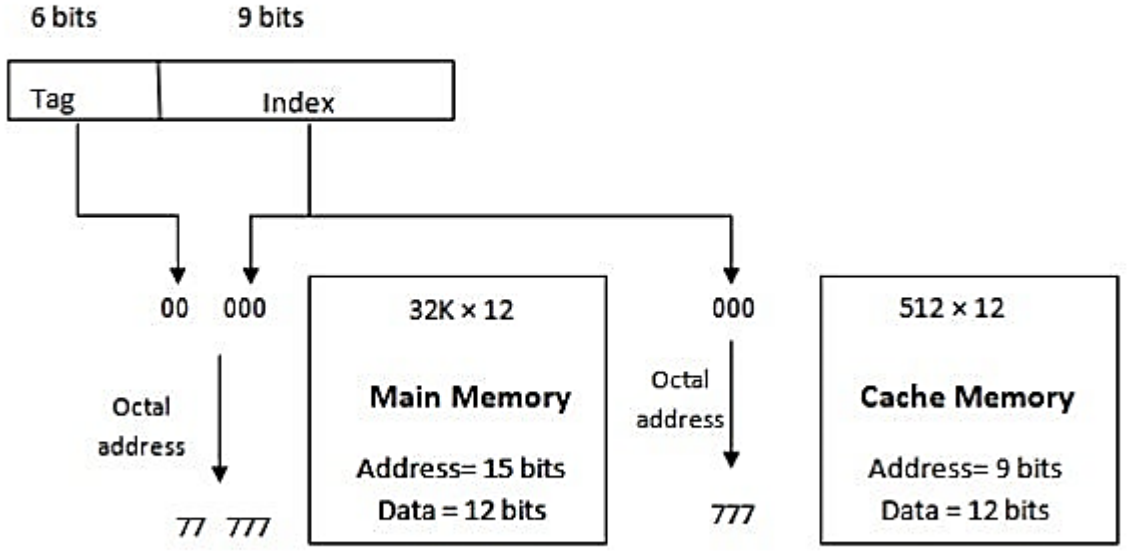
Figure: Associative mapping Cache

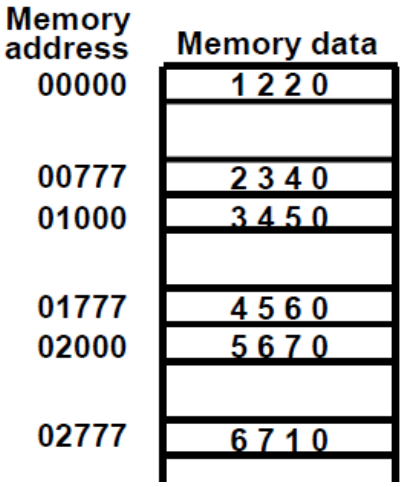


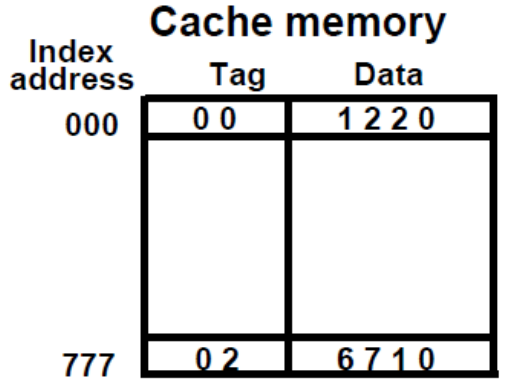
MEMORY AND CACHE MAPPING

2. Direct Mapping

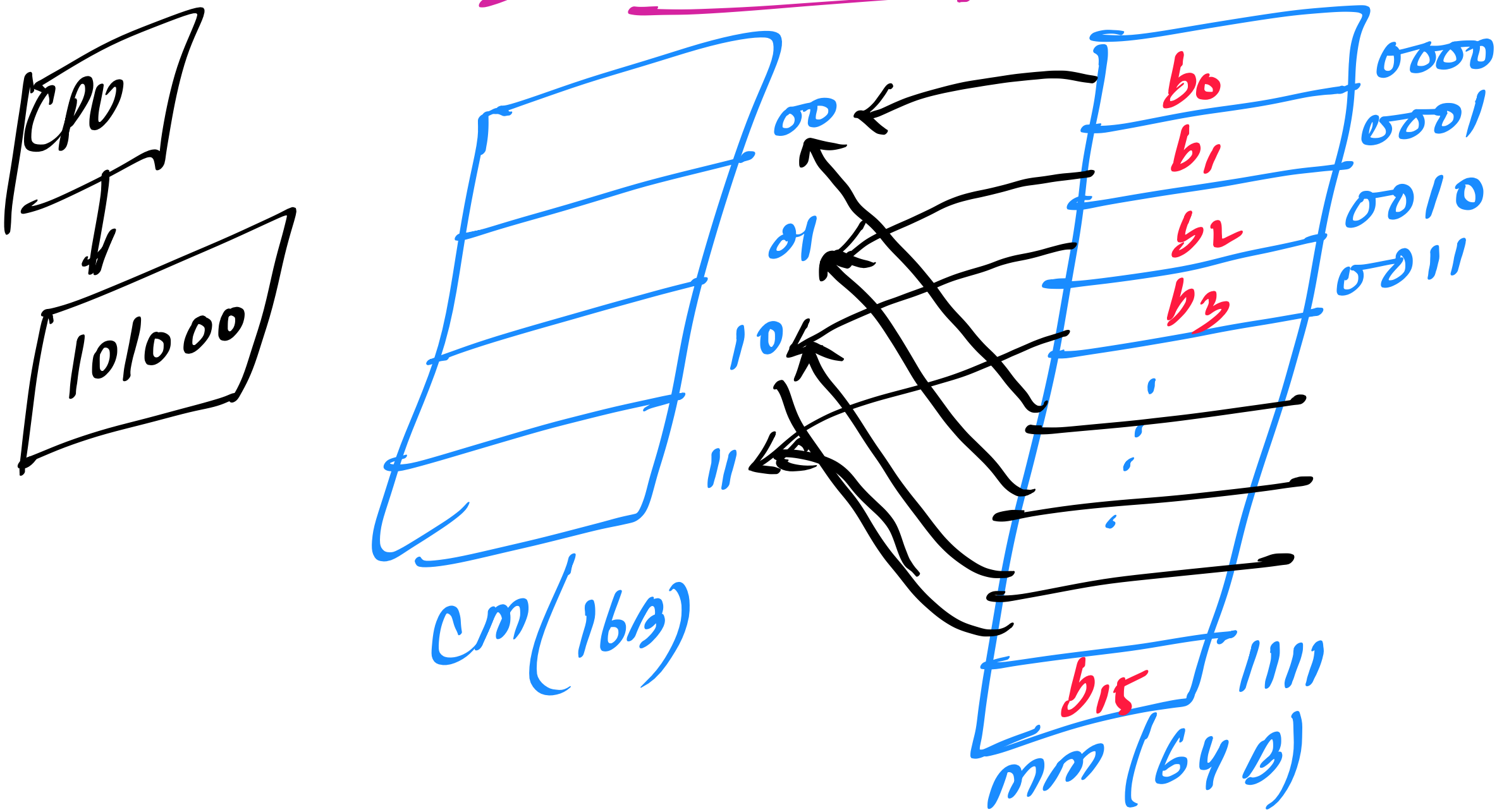
- It maps each **block** of main memory into only one possible cache.
- Mapping Table is made of RAM instead of CAM
- n-bit memory address consists of 2 parts': k bits of **Index field** and n-k bits of **Tag field**
- -n-bit addresses are used to access main memory and k-bit Index is used to access the Cache







b. Direct mapped Cache



* Direct mapped cache :-

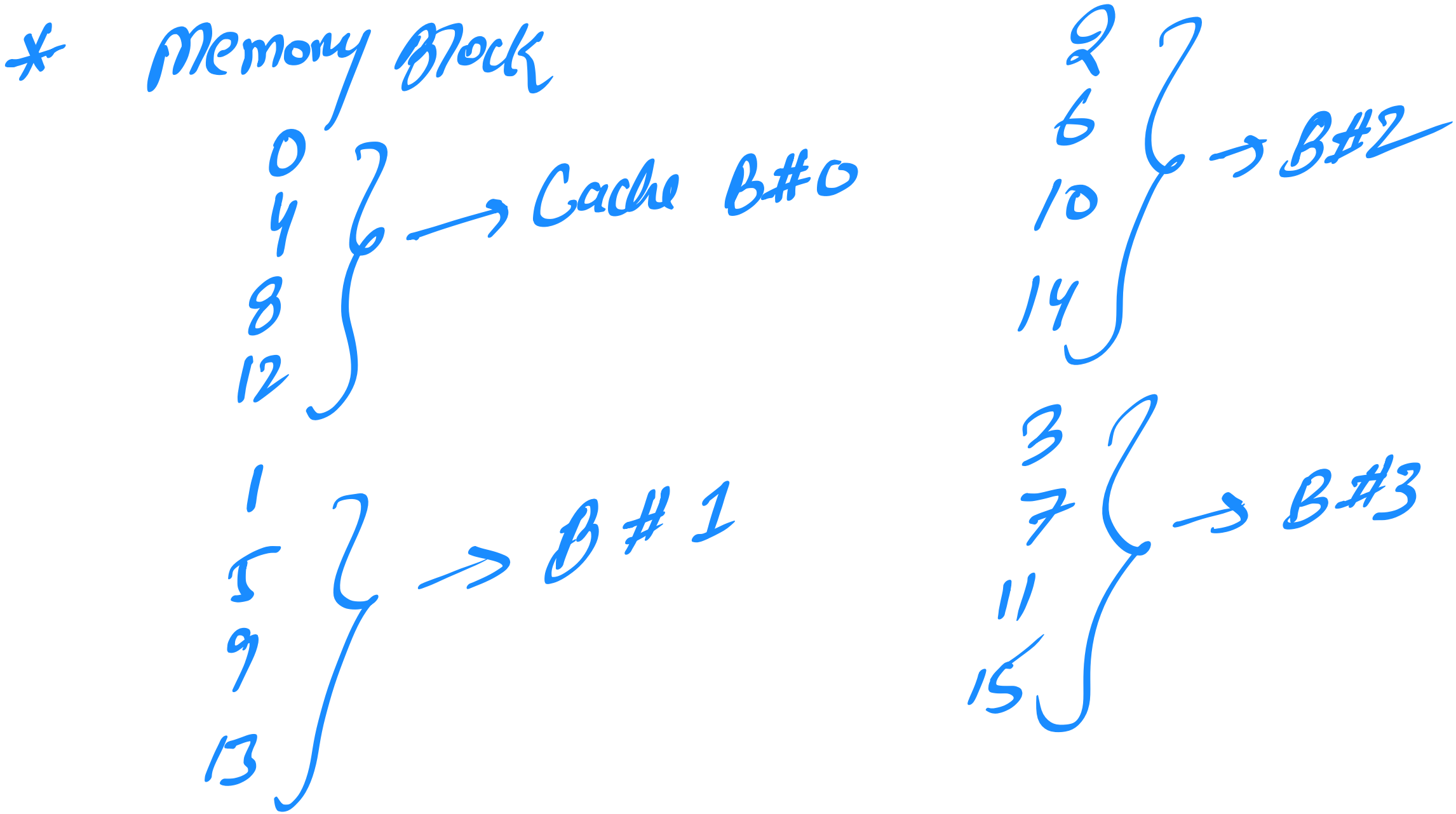
main memory Block Number M will be mapped to Cache Memory Block # = M mod # cm Block

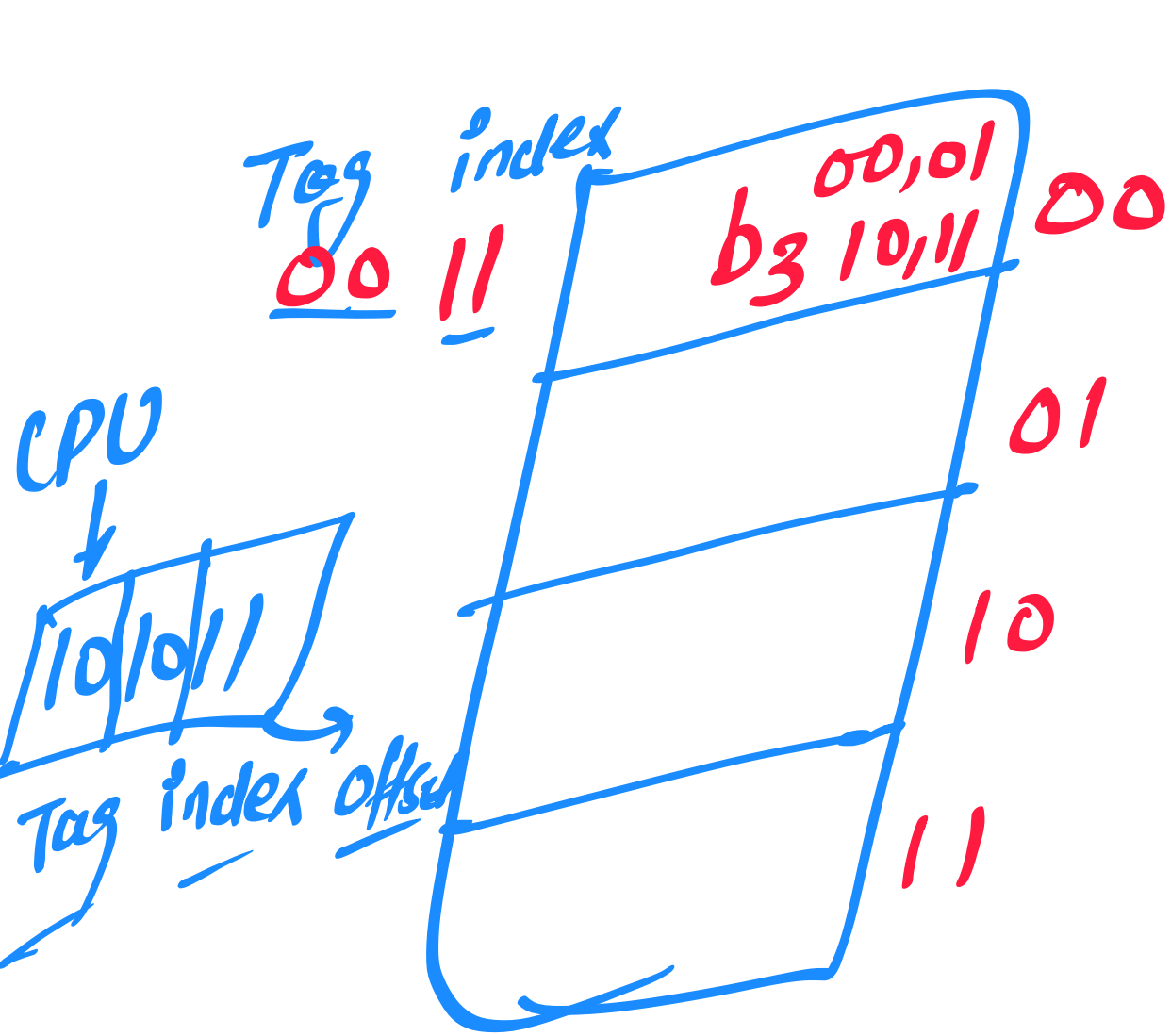
Cache memory = Memory Block Number % # Block in cm

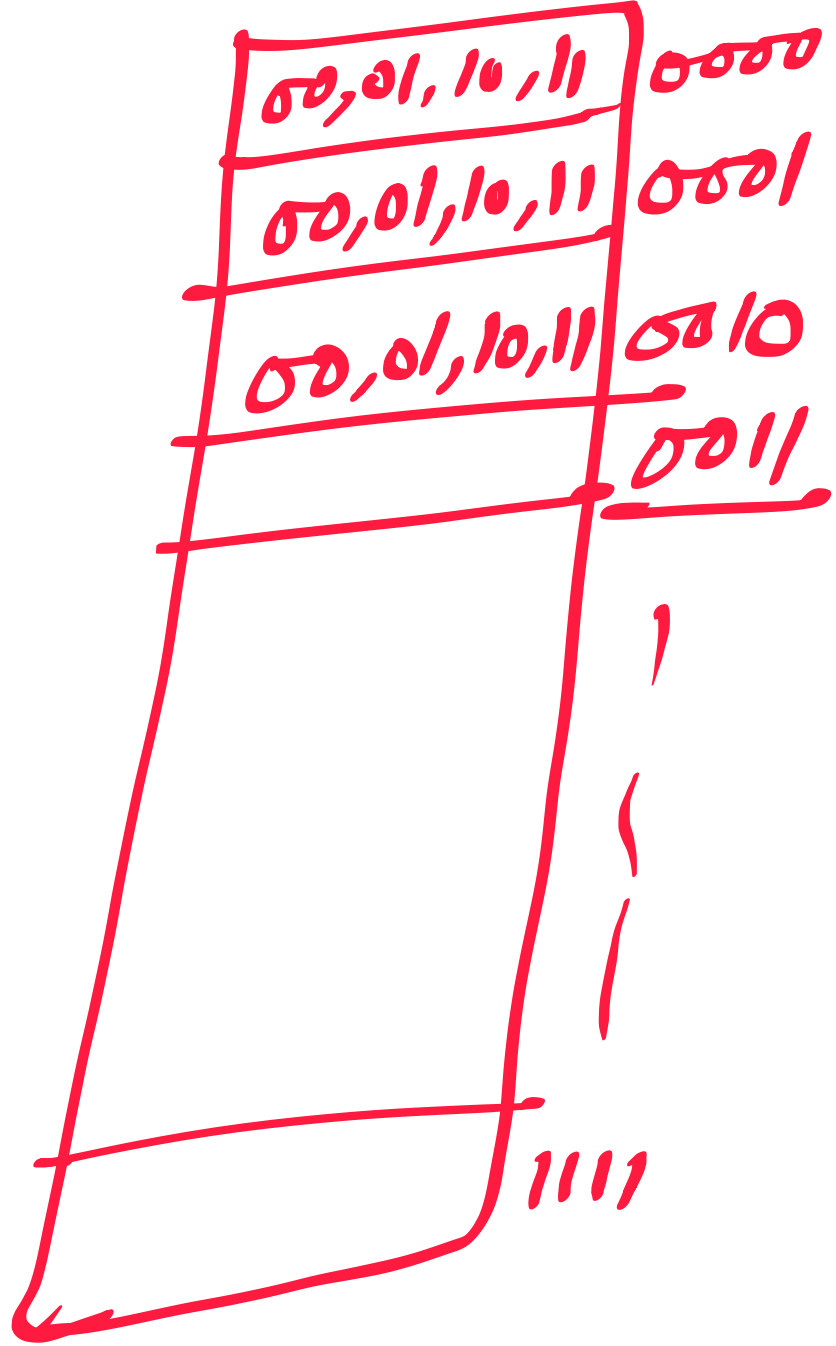
Block Number

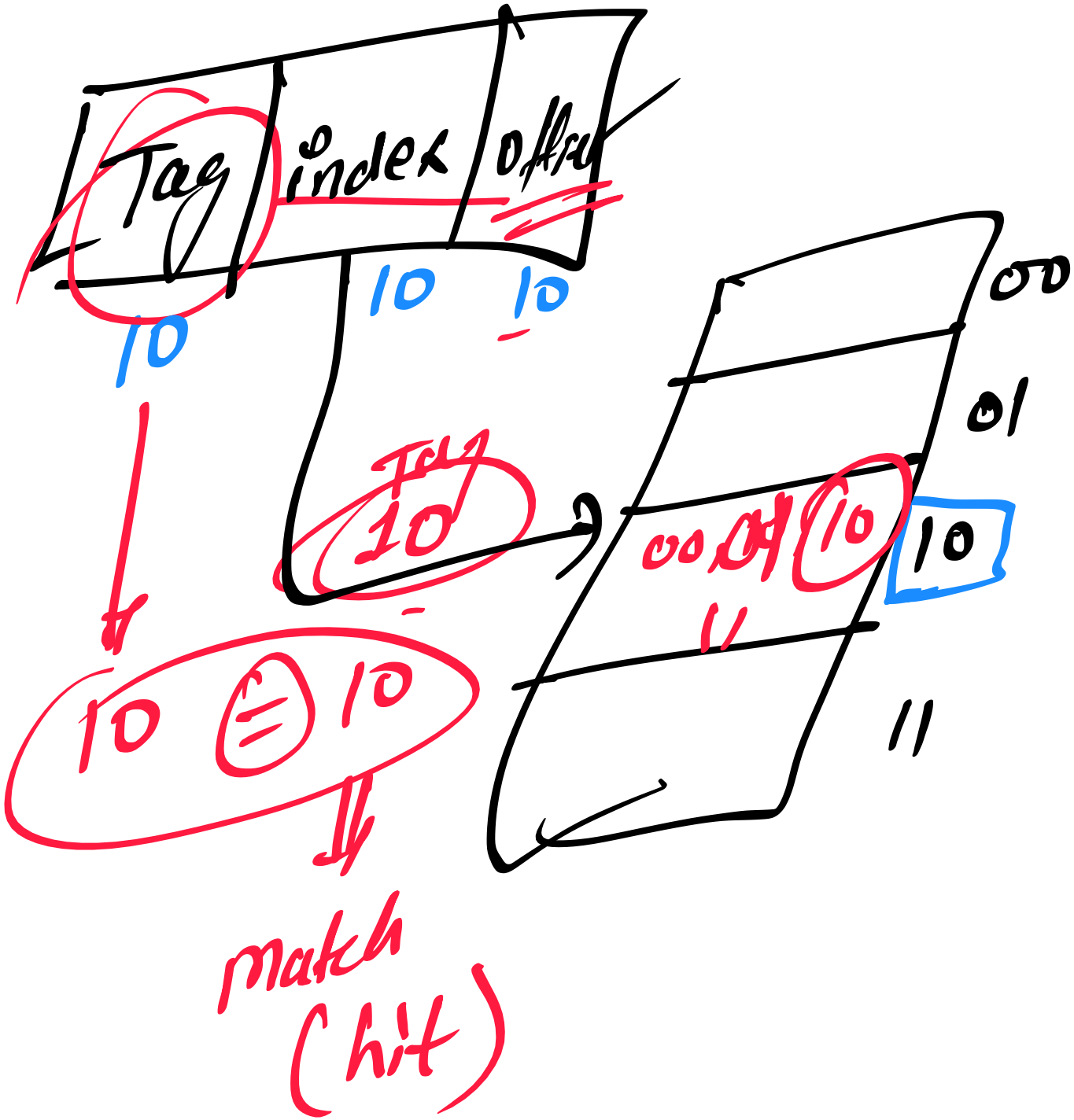
CM B# = MB# % N

Memory Block #	Cache Block #
0	0 % 4 = 0
1	1 % 4 = 1
2	2 % 4 = 2
3	3 % 4 = 3
4	4 % 4 = 0
5	5 % 4 = 1
6	6 % 4 = 2
7	7 % 4 = 3


* Memory Block
0
4
8
12
Cache B#0
2
6
10
14
B#2
1
5
9
13
B#1
3
7
11
15
B#3


Tag
index
00
11
00,01
b3 10,11
00
01
10
11
CPU
10 10 11
Tag index offset


00,01,10,11
0000
00,01,10,11
0001
00,01,10,11
0010
0011
1
1111

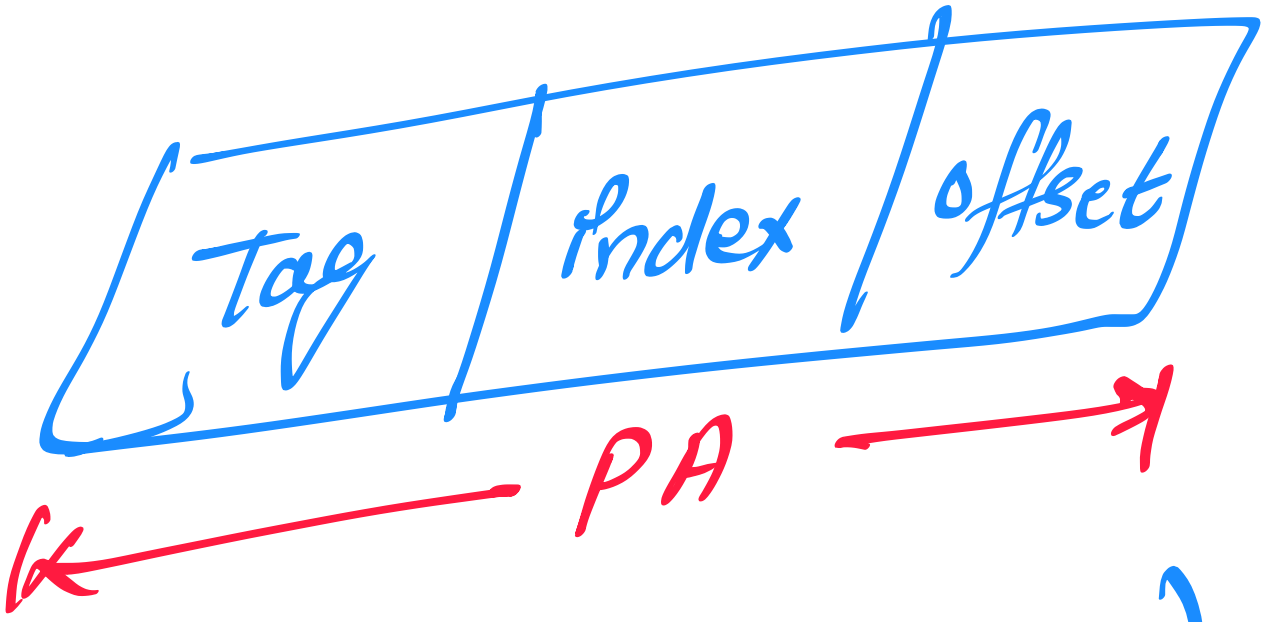


CM B#9

2 6 10 14

0010 0110 1010 1110

* Direct mapped Cache:



index = $\log_2$ (# CM Blocks)

offset = $\log_2$ (Block Size)

2. Direct Mapping: Operations

- 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)

Drawback:

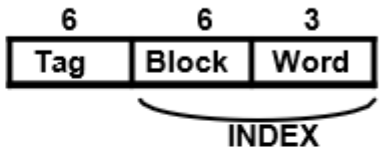
Two words with the same index cannot be stored inside the cache memory.

MEMORY AND CACHE MAPPING

2. Direct Mapping: (Cache memory size is 512 X 12)

Direct Mapping with block size of 8 words

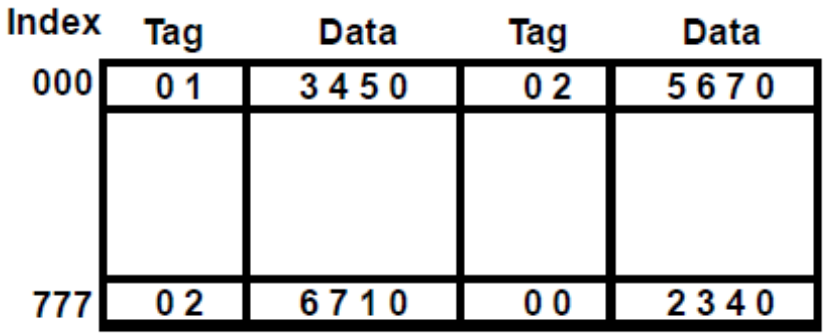
	Index	tag	data
Block 0	000	0 1	3 4 5 0
	007	0 1	6 5 7 8
Block 1	010		
	017		
Block 63	770	0 2	
	777	0 2	6 7 1 0



MEMORY AND CACHE MAPPING

3. Set Associative Cache Mapping

- Each memory block has a set of locations in the Cache to load
- Each word of a cache can store two or more words of memory under the same index address, creating a set.
- Each data word is stored together with its tag.
- The number of tag-data items in one word is said to form a **set**.
- Each index address refers to two data words and their associated tags.
- A set associative cache of set size k can accommodate k words of main memory.



Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Set Size = 2

Word Length = Tag + Data 2*(6+12) = 36

Size of cache = Number of bits in index = $9 = 2^9 = 512$ X 36.

This cache memory can accommodate 1024 words.

3. Set Associative Cache Mapping: Operations

- 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 ≠ 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)

Replacement Algorithms of Cache Memory

Replacement algorithms are used when there are no available space in a cache in which to place a data. Four of the most common cache replacement algorithms are described below:

- **Least Recently Used (LRU):**

  The LRU algorithm selects for replacement the item that has been least recently used by the CPU.

- **First-In-First-Out (FIFO):**

  The FIFO algorithm selects for replacement the item that has been in the cache from the longest time.

- **Least Frequently Used (LRU):**

  The LRU algorithm selects for replacement the item that has been least frequently used by the CPU.

Random:

The random algorithm selects for replacement the item randomly.

Writing into Cache

When memory write operations are performed, CPU first writes into the cache memory. These modifications made by CPU during a write operations, on the data saved in cache, need to be written back to main memory or to auxiliary memory.

These two popular cache write policies (schemes) are:

1. Write-Through
2. Write-Back

Write-Through

- In a write-through cache, the main memory is updated each time the CPU writes into the cache.
- The advantage of the write-through cache is that the main memory always contains the same data as the cache contains.
- This characteristic is desirable in a system that uses a direct memory access scheme of data transfer. The I/O devices communicating through DMA receive the most recent data.
- Slow, due to the memory access time

Write-Back

In a write-back scheme, only the cache memory is updated during a write operation.

The updated locations in the cache memory are marked by a flag so that later on when the word is removed from the cache, it is copied into the main memory.

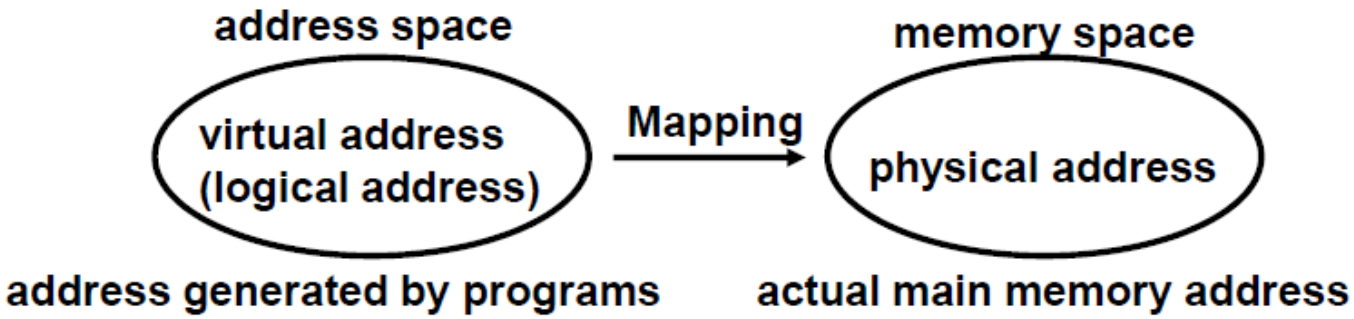
The words are removed from the cache from time to time to make room for a new block of words.

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

VIRTUAL MEMORY

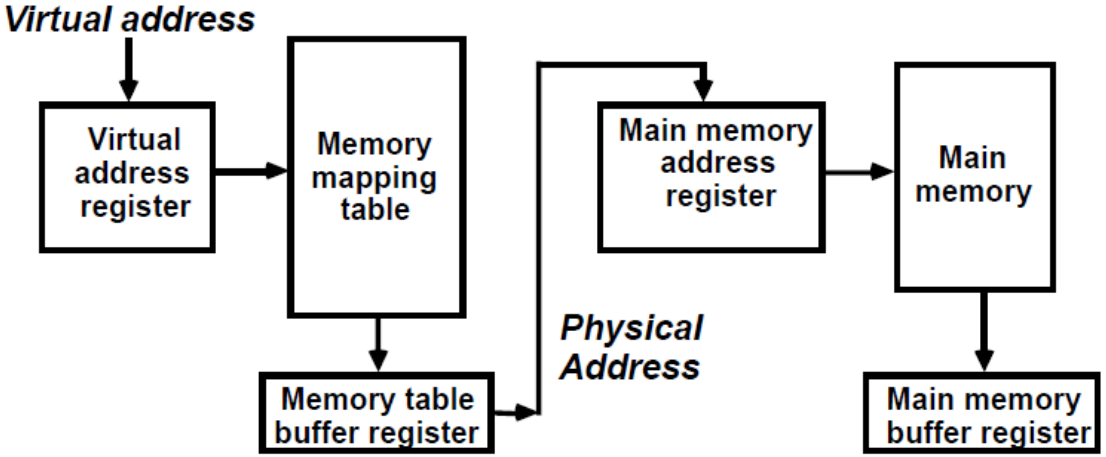
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)



Address Mapping

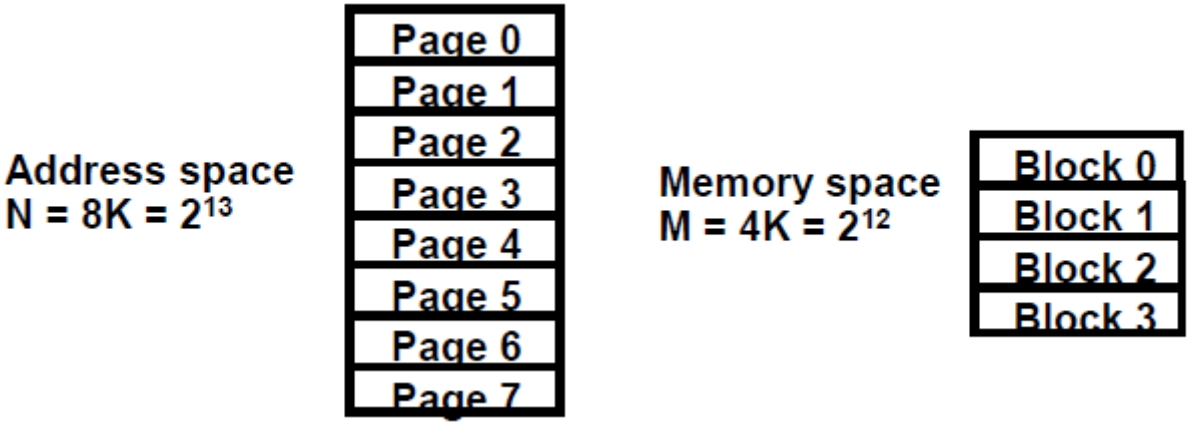
Memory *Mapping Table* for *Virtual Address -> Physical Address*



ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called *blocks* or *pages*

1K words group

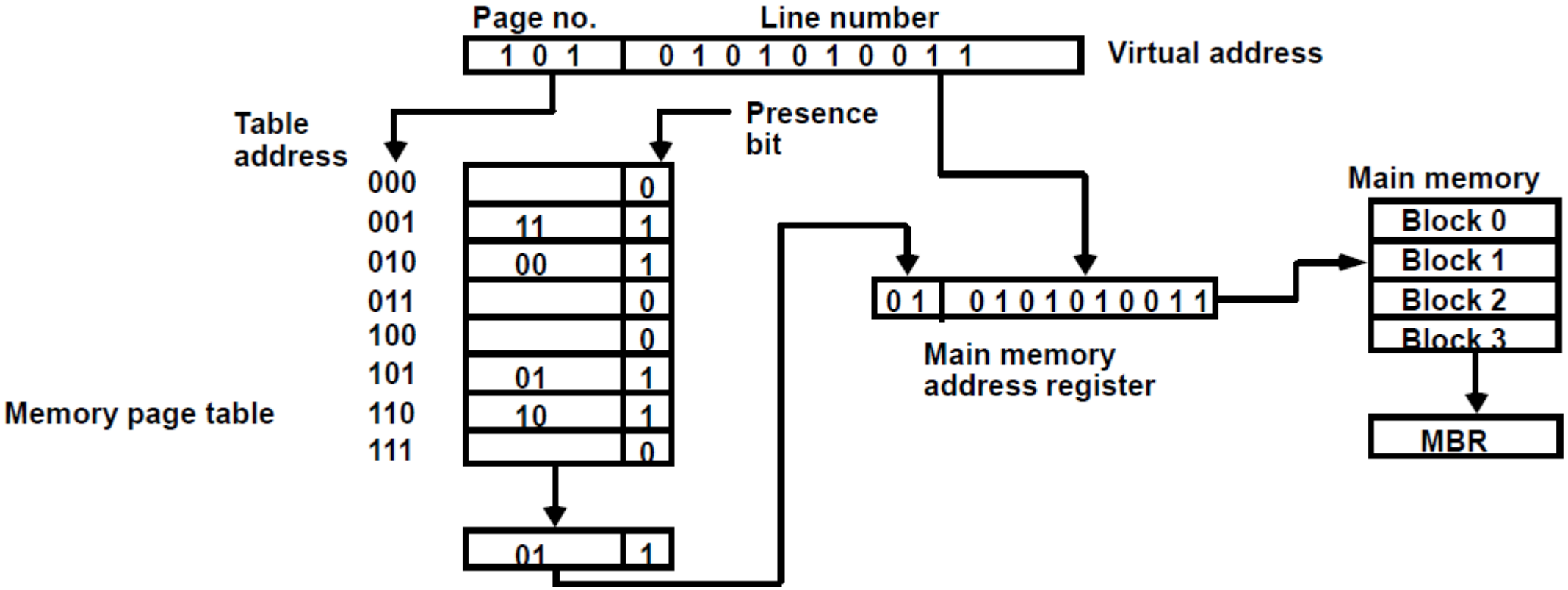


ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called *blocks* or *pages*

1K words group

Organization of memory Mapping Table in a paged system



ASSOCIATIVE MEMORY PAGE TABLE

Assume that

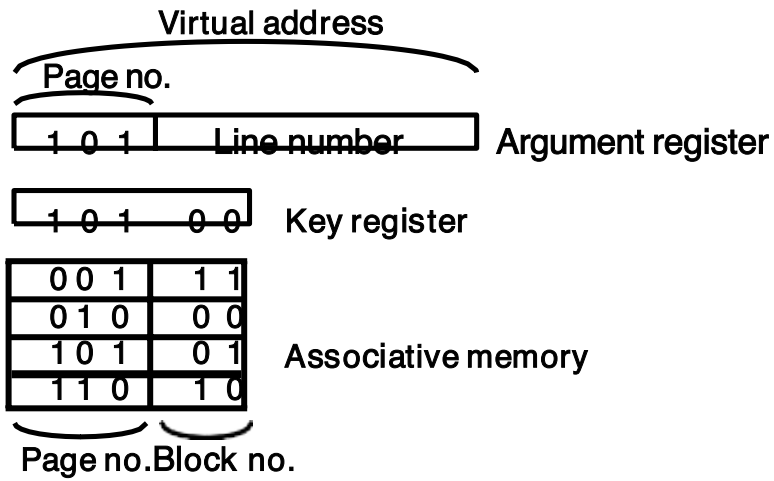
Number of Blocks in memory = m

Number of Pages in Virtual Address Space = n

Page Table

- Straight forward design -> n entry table in memory
  Inefficient storage space utilization
  <- n-m entries of the table is empty

- More efficient method is m-entry Page Table Page Table made of an Associative Memory m words;
  (Page Number:Block Number)

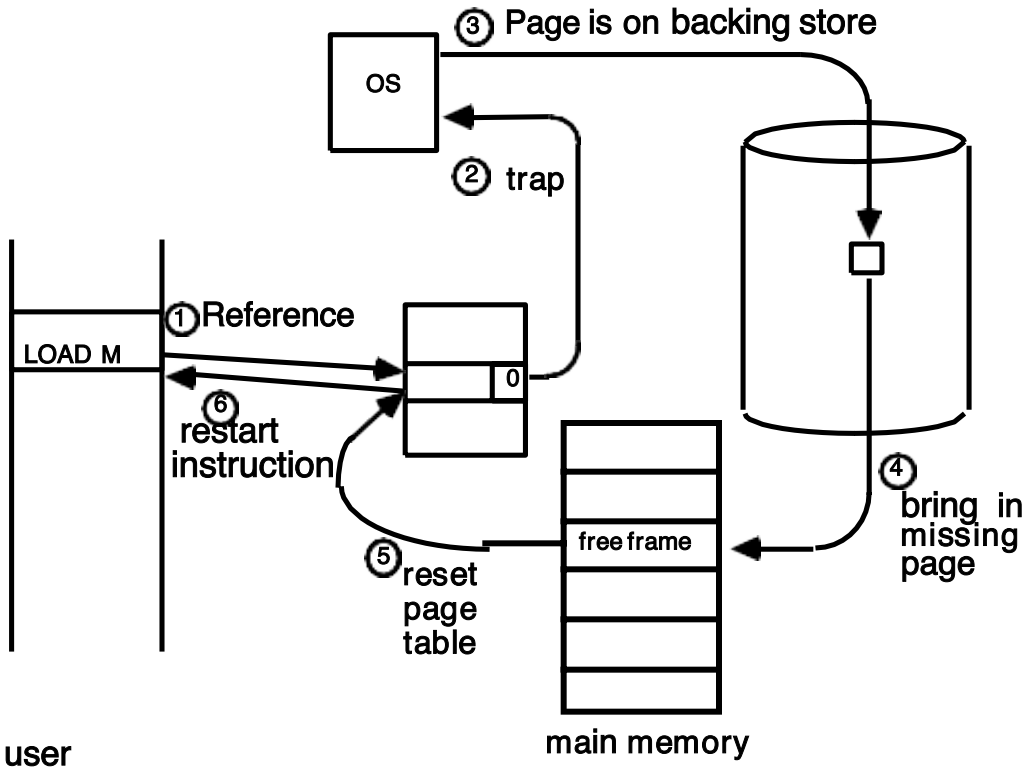


Page Fault

Page number cannot be found in the Page Table

PAGE FAULT

1. Trap to the OS
2. Save the user registers and program state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the backing store(disk)
5. Issue a read from the backing store to a free frame
   a. Wait in a queue for this device until serviced
   b. Wait for the device seek and/or latency time
   c. Begin the transfer of the page to a free frame
6. While waiting, the CPU may be allocated to some other process
7. Interrupt from the backing store (I/O completed)
8. Save the registers and program state for the other user
9. Determine that the interrupt was from the backing store
10. Correct the page tables (the desired page is now in memory)
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, program state, and new page table, then resume the interrupted instruction.



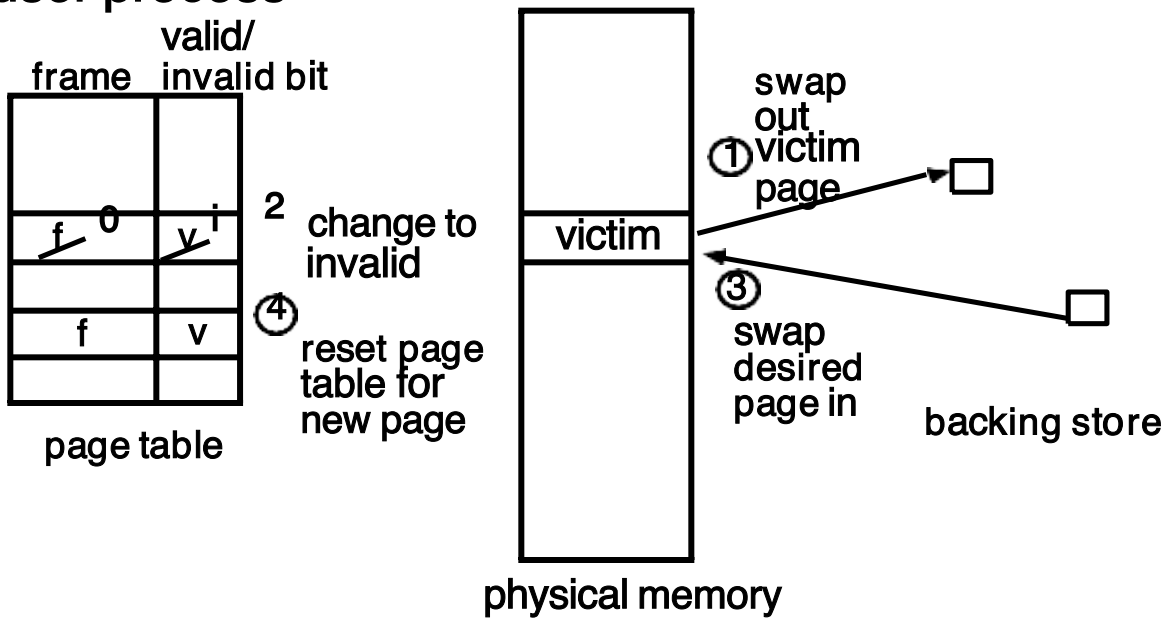
Processor architecture should provide the ability to restart any instruction after a page fault.

PAGE REPLACEMENT

Decision on which page to displace to make room for an incoming page when no free frame is available

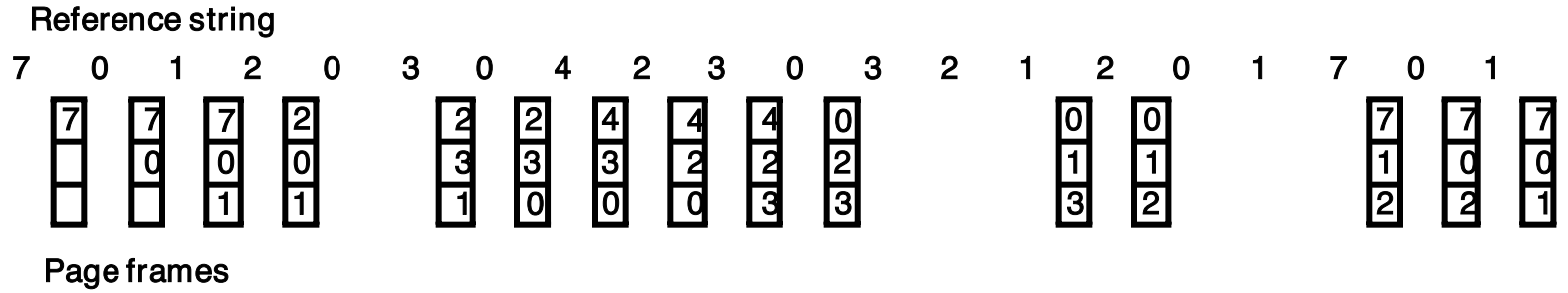
Modified page fault service routine

1. Find the location of the desired page on the backing store
2. Find a free frame
   - If there is a free frame, use it
   - Otherwise, use a page-replacement algorithm to select a *victim* frame
   - Write the victim page to the backing store
3. Read the desired page into the (newly) free frame
4. Restart the user process



PAGE REPLACEMENT ALGORITHMS

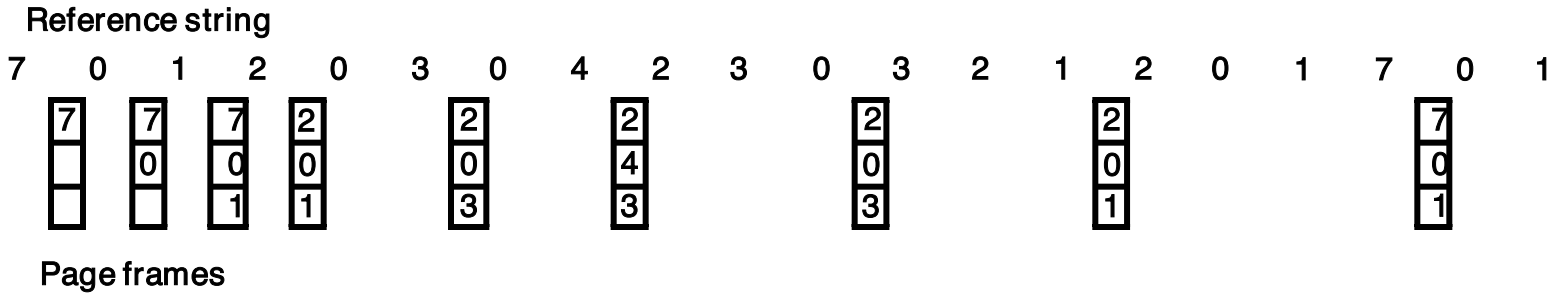
1. FIFO



FIFO algorithm selects the page that has been in memory the longest time
Using a queue - every time a page is loaded, its identification is inserted in the queue
Easy to implement
May result in a frequent page fault

Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

Replace that page which will not be used for the longest period of time

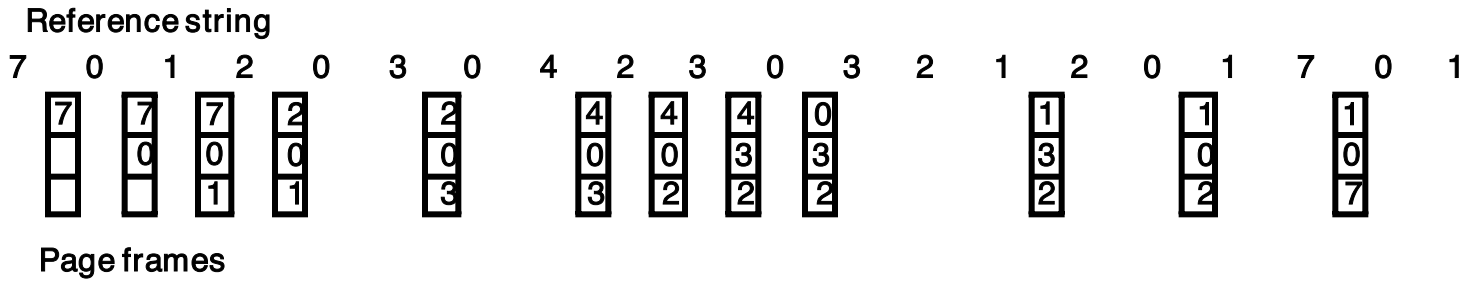


PAGE REPLACEMENT ALGORITHMS

LRU

- OPT is difficult to implement since it requires future knowledge
- LRU uses the recent past as an approximation of near future.

> Replace that page which has not been used for the longest period of time

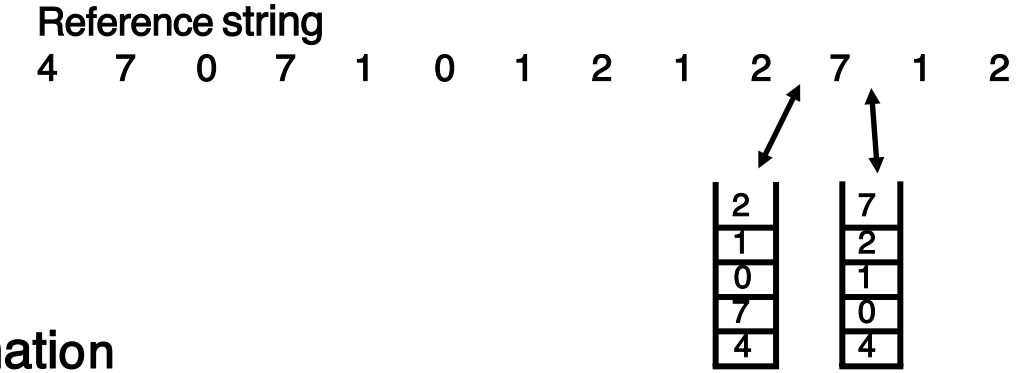


- LRU may require substantial hardware assistance
- The problem is to determine an order for the frames defined by the time of last use

PAGE REPLACEMENT ALGORITHMS

LRU Implementation Methods

- Counters
  - For each page table entry - time-of-use register
  - Incremented for every memory reference
  - Page with the smallest value in time-of-use register is replaced
- Stack
  - Stack of page numbers
  - Whenever a page is referenced its page number is removed from the stack and pushed on top
  - Least recently used page number is at the bottom



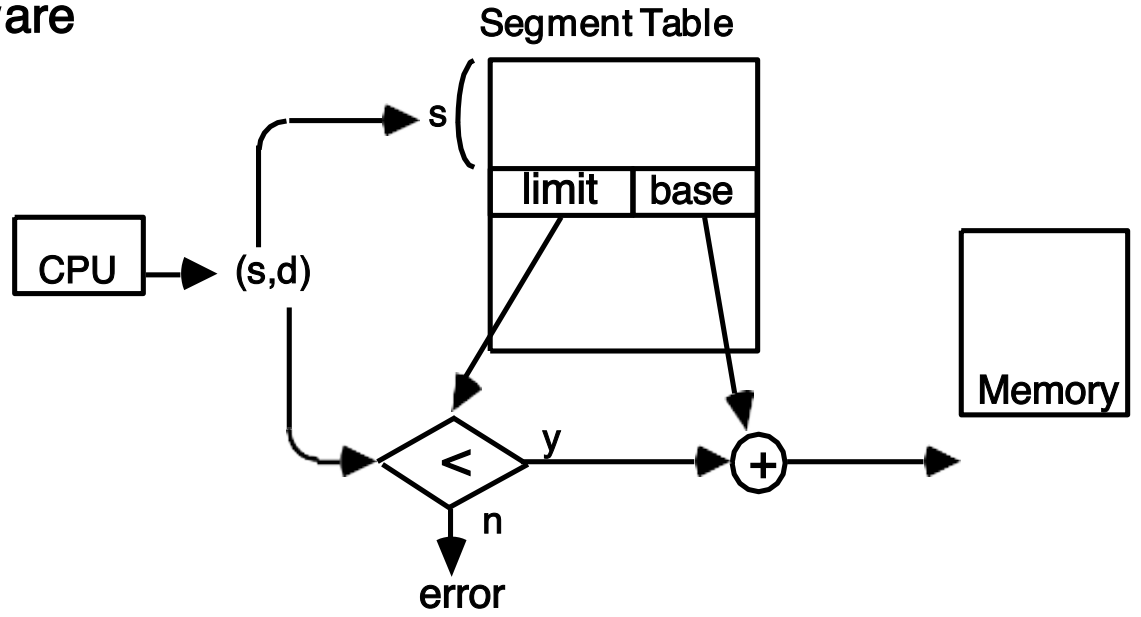
LRU Approximation

- Reference (or use) bit is used to approximate the LRU
- Turned on when the corresponding page is referenced after its initial loading
- Additional reference bits may be used

SEGMENTATION

- A memory management scheme which supports user's view of memory
- A logical address space is a collection of segments
- Each segment has a name and a length
- Address specify both the segment name and the offset within the segment.
- For simplicity of implementations, segments are numbered.

Segmentation Hardware



Numericals

* Required = $2048 * 8$

* Available = $128 * 8$

128*8 chips Required = $\frac{2^{11} * 8}{2^7 * 8} = 16$

Q1. i.) How many 128 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes?
ii.) How many lines of the address bus must be used to access 2048 byte of memory? How many of these lines will be common to all chips?
iii.) How many lines must be decoded for chip select? Specify the size of the decoders?

i) According to question, 128 x 8 RAM = 1024 bits and 1 bit = 1/8Byte Thus, 1024 bits = 1024 /8 = 128 Byte
1 RAM chip = 128 Byte
Lets consider there are n number of Chips to provide 2048 Bytes
Now, 1RAM Chip = 128 Byte
128n = 2048
n = 16 Therefore, 16 Chips will be needed to provide memory capacity of 2048 bytes.

ii) 2048 (2^{11}) locations needs same address lines as one need bits for representing 2048 11 lines of address bus required to access 2048 bytes of memory.
Here chips are of 128x8 in size,
128 (2^{7}) locations means we need 7 address lines. Therefore 11 lines of address bus required to access 2048 bytes of memory. And 7 address lines will be common to all chips.

iii) 4 address lines must be decoded to select the right chip. 4 address lines will need 16 chips, Size of decoder = 4x16.

Q2. A computer uses RAM chips of 1024 x 1 capacity.

a) **How many chips are needed and how should their address lines be connected to provide a memory capacity of 1024 bytes?**
b) **How many chips are needed to provide a memory capacity of 16K bytes?**
c) **Explain in words how the chips are to be connected to the address bus.**

(i) As given available chips = 1024 x 1 capacity and Required capacity = 1024 x 8 capacity Number of Chips=(1024X8)/(1024X1) = 8

(ii) Number Of Chips Required = (16 X 1024 X 8)/(1024 X 1) = 128

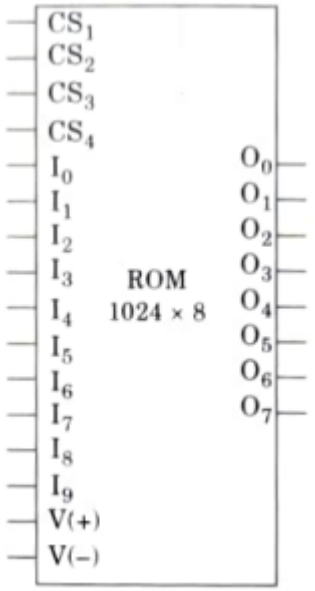
(iii) Use 14 address lines (16 k = 2^{14})

10 lines specify the chip address

4 lines are decoded into 16 chip-select inputs.

Q3. A ROM chip of 1024*8 has four select inputs and operates from a 5 volt power supply. How many pins are needed for the IC package? Draw a block diagram and label all input and output terminals in the ROM.

Size of ROM Chip = 1024 x 8
Number of input = 10 pin [$2^{10} = 1024$]
Number of output = 8 pin
Number of chip select = 4 pin
Power = 2 pin
Total 24 pins are required.



Q5. A computer employs RAM chips of 256 x 8 and ROM chips of 1024 x 8. the computer system needs 2k bytes of RAM, 4k bytes of ROM, and four interface units, each with four registers. a memory-mapped I/O configuration is used. the two highest-order bits of the address bus are assigned 00 for RAM, 01 for ROM, and 10 for interface registers.

a. How many ram and ROM chips are needed?
b. Draw a memory-address map for the system.
c. Give the address range in hexadecimal for RAM, ROM, and interface.

RAM	2048 /256 = 8 chips;	$2048 = 2^{11}$; $256 = 2^8$
ROM	4096 /1024 = 4 chips;	$4096 = 2^{12}$; $1024 = 2^{10}$
Interface	$4 \times 4 = 16$ registers; $16 = 2^4$	

Component	Address	16 15 14 13 12 11 10 19	8765	4321
RAM	0000-O7FF	0 0 0 0 0 ⟷ (3×8 decoder)	××××	××××
ROM	4000-4FFF	0 1 0 0 ⟷ (2×4 decoder)	×× ××××	××××
Interface	8000-800F	1 0 0 0 0 0 0 0	0000	××××

Q20. A virtual memory has a page size of 1K words. There are eight pages and four blocks. The associative memory page table contains the following entries:

Page	Block
0	3
1	1
4	2
6	0

Make a list of all virtual addresses (In decimal) that will cause a page fault if used by the CPU.

The pages that are not in main memory are:

Page	Address	address that will cause fault
2	2K	2048 – 3071
3	3K	3072 – 4095
5	5K	5120 – 6143
7	7K	7168 – 8191

Q23. The logical address space in a computer system consists of 128 segments. Each segment can have up to 32 pages of 4K words in each physical memory consists of 4K blocks of 4K words in each. Formulate the logical and physical address formats.

Logical address space has 128 segments x 32 pages x 4 K words

i.e., $128 \times 32 \times 4 \times 2^{10}$

$= 2^7 \times 2^5 \times 2^2 \times 2^{10} = 2^{24} = 24$ bit

Physical memory has address space of 4 K blocks x 4 K words

$= 4 \times 2^{10} \times 4 \times 2^{10}$

$= 2^{24} = 24$ bits

Logical address:

7 bits	5 bits	12 bits
Segment	Page	Word

= 24 bits

Physical address:

12 bits	12 bits
Block	Word

NUMERICALS ON MEMORY ORGANIZATION

1. A computer with cache access time of 30 ns, main memory access time of 150ns and hit ratio of 0.8 produces an average access time of CPU is _____.

Soln: Hit ratio of cache = Hcache = 0.8

Tcache = 30 ns

Tmemory = 150 ns

CPU access time = Hcache * Tcache + [(1 – Hcache)*(Tcache + Tmemory)]

= 0.8 * 30 + 0.2 * (30 + 150) = 60 ns

2. Consider a high speed 40ns memory cache with a successful hit ratio of 80%. The regular memory has an access time of 100ns. What is the effective access time for CPU to access memory?

Soln.: Effective CPU access time = cache hit ratio* cache access time + cache miss ratio*(cache access time + main memory access time)

cache miss ratio= (1-hit ratio)

=> 0.8 * 40 + [(1-0.80)*(40+100)] = 32+28 = 60ns

3. A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte. Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address.

Soln.: Block size = 64 bytes = 2^6 bytes = 2^6 words (since 1 word = 1 byte)

Therefore, **Number of bits in the Word field = 6**

Cache size = 2K-byte = 2^{11} bytes

Number of cache blocks = Cache size / Block size = $2^{11}/2^6 = 2^5$

Therefore, **Number of bits in the Block field = 5**

Total number of address bits = 16

Therefore, **Number of bits in the Tag field = 16 - 6 - 5 = 5**

For a given 16-bit address, the 5 most significant bits represent the Tag, the next 5 bits represent the Block, and the 6 least significant bits represent the Word.

4. Consider the performance of a main memory organization, when cache miss has occurred as

i) 4 clock cycles to send the address

ii) 24 clock cycles for the access time per word.

iii) 4 clock cycles to send a word of data.

Estimate:

a) The miss penalty for a cache block of 4 words.

b) The miss penalty for a 4 – ways interleaved main memory with a cache block of 4 words.

Soln.: To find a single word from main memory, the processor wastes 4+24 = 28 cycles.

Since, size of the main memory block = size of the cache memory block

To find 4 words in main memory processor access it one time and it takes the whole block to cache. So, miss penalty: 4*28+4 = 116

For a 4 – way interleaved memory, the 4 words to be present in 4 different banks.

So, the first 4 cycles, all the address are activated and time required to read = 24 cycles.

Let, it requires 4 clock cycle to send a word of data.

So, the total miss penalty = 4+(24+4*4) = 44.