Memory Organization – Computer Organization and Architecture.

Shriram K Vasudevan

Must Know Topic – Pay attn. please

Memory Heirarchy

- What would be desire of the programmers and the PC users from day – 1 of PCs invention?
  - Memory.
  - Fast memory. (☺, tricky)
- One cannot have more memory (unlimited memory) as it would be expensive (Though not much from HDD perspective, from primary memory perspective, it is expensive)
- Hence, what the solution is?
  - Create illusion.
  - Create intelligent illusion of having fast and large memory.
  - We are gonna use this concept.

Contd.,

- Let us take the library as an example.
- Select a topic to study Example MIPS architecture.
- What will you do?
  - You will go to the rack of computer architecture.
  - Pick the best of the books (say 5 or 6) that talks MIPS.
  - Bring them to your desk.
  - Now, most of what is required shall be available in the desk.
  - Reader need not walk to and fro to the rack.
  - Here is the point to appreciate:
  - **HAVING SEVERAL BOOKS ON THE DESK IN FRONT OF YOU SAVES TIME COMPARED TO HAVING ONLY ONE BOOK THERE AND CONSTANTLY HAVING TO GO BACK TO THE SHELVES TO RETURN IT AND TAKE OUT ANOTHER.**

Contd.,

- Understand this:
  - **One would not require access to all the books in the rack at a time. (at least not of equal priority)**
  - **Similarly, a program will not need access to all of the code or the data at the same point in time.**
  - **If all the data/code are getting accessed at once with equal priority, it would be impossible to get the speed we anticipate. ☺**
- Can we not have a huge memory but still it being fast?
  - Good question.
- **Otherwise, it would be impossible to make most memory accesses fast and still have large memory in computers, just as it would be impossible for you to fit all the library books on your desk and still find what you wanted quickly**

Let’s learn this – Locality

- Principle of Locality – Shall fit for the library and programs.
- As we know,
  - We access only very limited amount of books at a particular time (you can’t access all the books, makes logical)
  - This can be related to memory.

<u>Temporal Locality:</u>
If a particular data location is referenced, then there is a chance that it will be referred again soon.

<u>In library context:</u>
If you refer a book which is at desk , you will refer it soon again.

<u>Spatial Locality:</u>
If a particular data location is referenced, then, the locations nearby shall be referred in near future.

<u>In library context:</u>
If you refer a book, you may also refer to books in the same shelf/desk nearby in future soon again.

Let’s get this better…

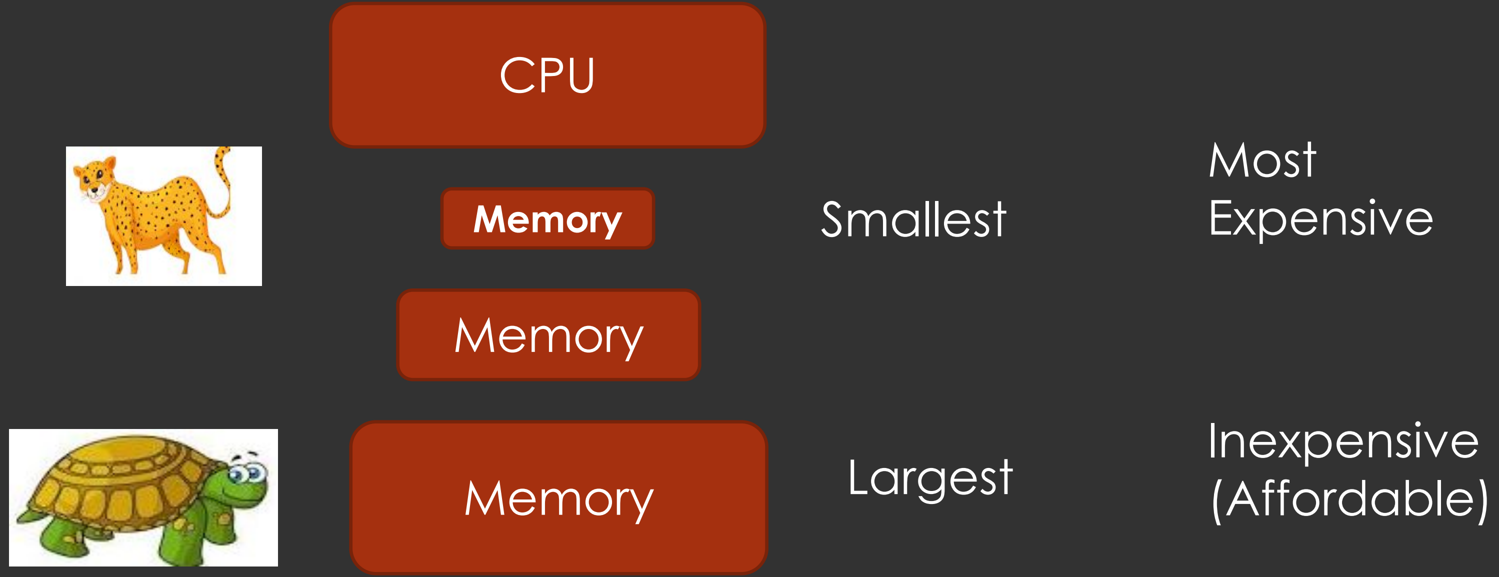
- **Let us connect the books to program.**
- Locality shall come in picture easily when we cite loops.
  - **The instructions and data (inside loop) are most likely to be accessed repeatedly, again and again and it is called temporal locality.**
  - Normal access of sequence of instructions in the program can be referred through **spatial locality.**
- Coming to the data, **a bit more detailed, if you take an array as reference, it would connect to spatial locality.**

Memory Hierarchy – A quick view.

- Shall we see memory as a hierarchy? It would be easier.
- Ground Rule:
  - Ferrari is expensive than Maruti.
  - Faster is expensive than slower memories.
- Main memory == RAM (Primary Memory, implemented as DRAM)
- Cache == Nearer to processor (Implemented as SRAM)
- **The largest and slowest is hard drive (Implemented as Magnetic Disk)**

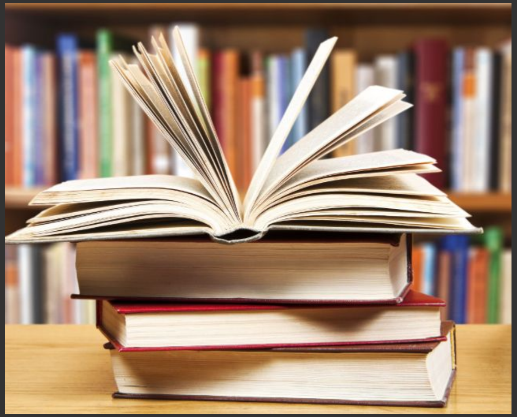
Contd.,

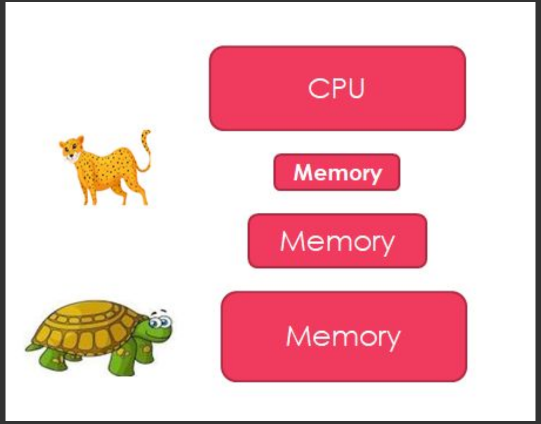
- Ground rule:
- **Faster memory is closer to the processor and the most expensive one.**
- **Slower memory is far from the processor (below in the hierarchy) and less expensive.**
- What is the goal? **Provide the user with as much memory, but should be inexpensive as well, while also providing the speed as fastest memory. (Sounds tough, huh??)**



Contd.,

- Understand this:
  - Each level in the hierarchy is a subset of the subsequent level.
  - Data shall be housed in the lowest level as it has the highest space available.
- Take the library example again. The books at the desk is a subset of the library. Your library is a subset of all the libraries in the university.
- If you are closer to the processors, it will be easier (less farther) to reach and in case far away, it will take longer time. (easy with library table example)

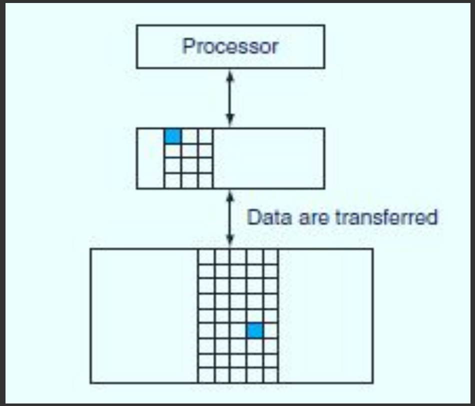




Contd.,

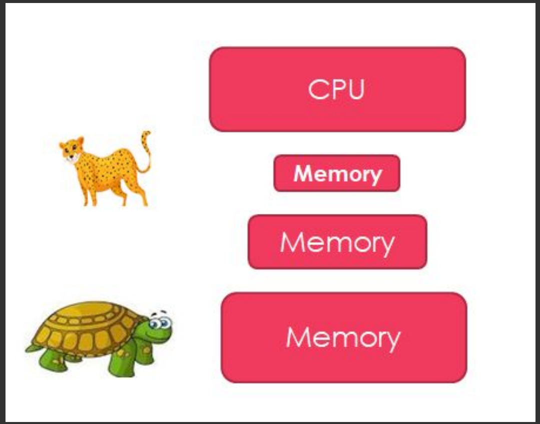
- Well, let us understand this point too.
- Though there are multiple levels of hierarchy, the data is copied (moved) between only any two adjacent levels at a time.
  - Hence, if you pay attention for any two levels where data is being transferred between, it is sufficient.
- Let us understand couple of quick definitions:

- **Within each level, the unit of information that is present or not is called a *block.***
- **Usually we transfer an entire block when we copy something between levels.**



Some more definitions..

- *If you search for a content and in case is available in the desk itself, we call it a hit!! Same here for computers.* ***If the data requested is found in any of the blocks in upper level, then we can refer it a hit.***
- *Well, if you can't find it in the upper level (i.e. the desk), it is referred a miss! (Means, you have to take some corrective action)*
- *So, what if the book is not found on the desk? Go to the next level, i.e. the shelf to find it.*
  - *Same is the case here. If the data is not found in the current level, then, search it in the next level. I.e. lower level in the hierarchy.*



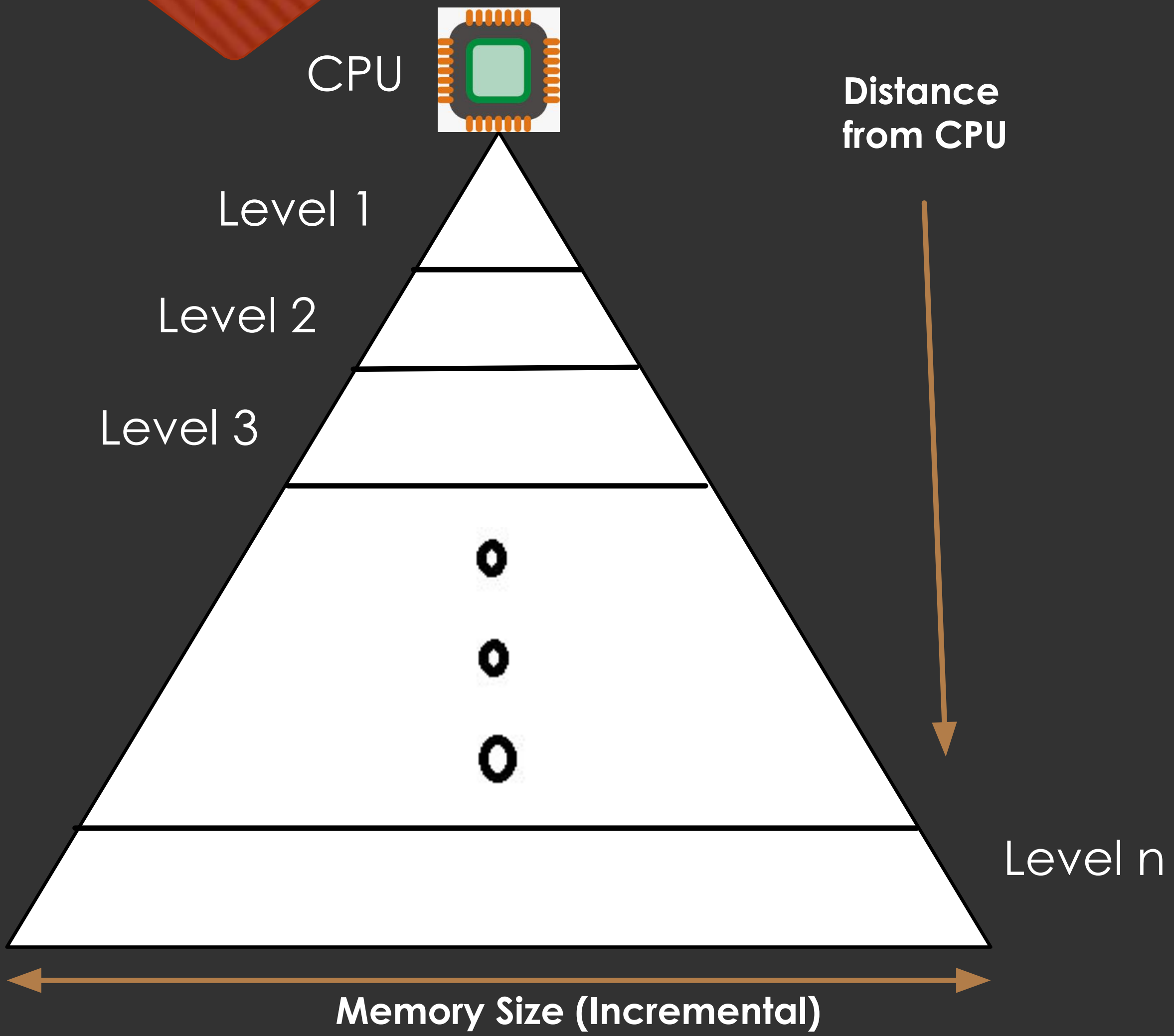
Definitions..

- The **hit rate, or *hit ratio***, is the fraction of memory accesses found in the upper level;
  - This literally defines the performance!
- The ***miss rate*** is the fraction of memory accesses not found in the upper level.
  - Again, can be used for performance measurement.
- **Hit time** is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss (that is, the time needed to look through the books on the desk).
- The **miss penalty** is the time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor (or, the time to get another book from the shelves and place it on the desk).
- Upper level = Faster == Smaller hit time ☺
  - You can check the books on the table faster than going to the shelf and to check others.

Why do we care about memory?

- All the programs spend almost all the time in accessing memory! (True, huh?) Hence, the memory plays a major role in determining system’s performance.

So,



- Let us come back to the concept of localities.
- A code will have both the temporal and spatial locality.
- A code may access the data which is accessed recently.
  - Also, it can refer the locations nearby (spatial).
- Refer the LHS figure:
  - Access @ the highest level shall be faster. (Hit even, is faster)
  - Access miss at higher level shall move the lower level. It will need more time is the point.

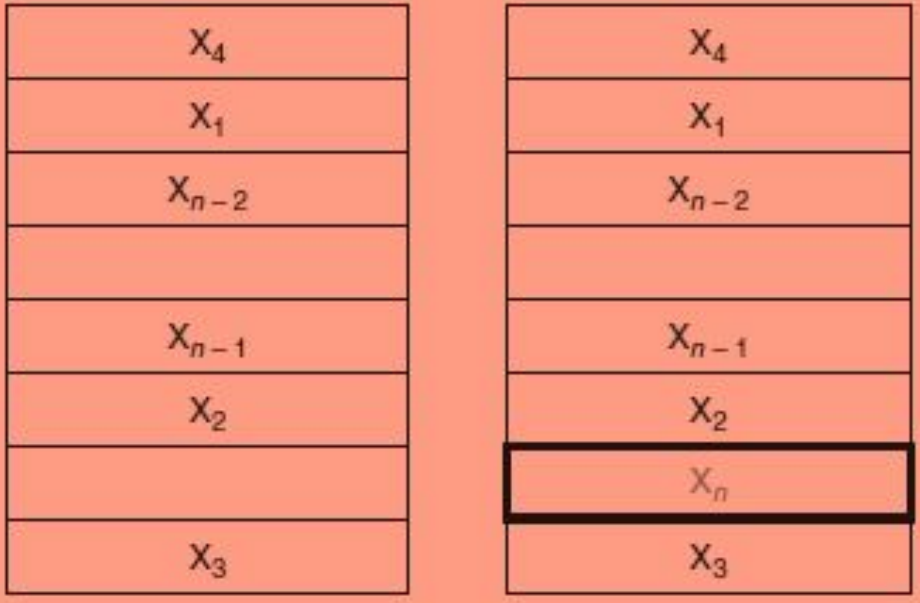
Let us learn about Cache

- What is Cache?
  - Going by Google Dictionary – Cache is a French word. It means hidden.
  - Coming to the COA analogy, we have a safe place to store things and it is called the cache.

- In the example Library scenario – Desk is the cache.
- We had books from the shelf to the desk, which is a safe place and it accommodates books.
- Let us learn about this a bit detailed.

Contd.,

- Let us have a simple cache – Before and after the data request process.
- Assume, the data is not there initially in the cache. (First time, it would be so)
- *Before the request, the cache contains a collection of recent references X1, X2, . . . , Xn – 1,* ***and the processor requests a word Xn that is not in the cache.***
- This request results in a **miss**, and the word X*n* is brought from memory into cache.

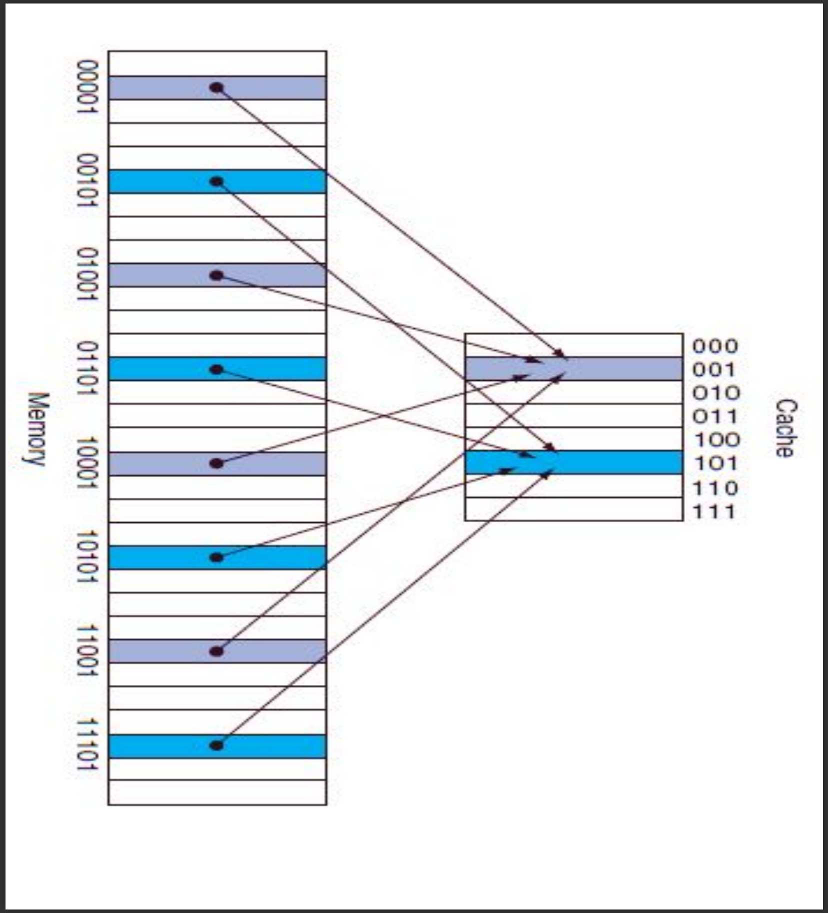


a. Before the reference to X_n b. After the reference to X_n

Contd.,

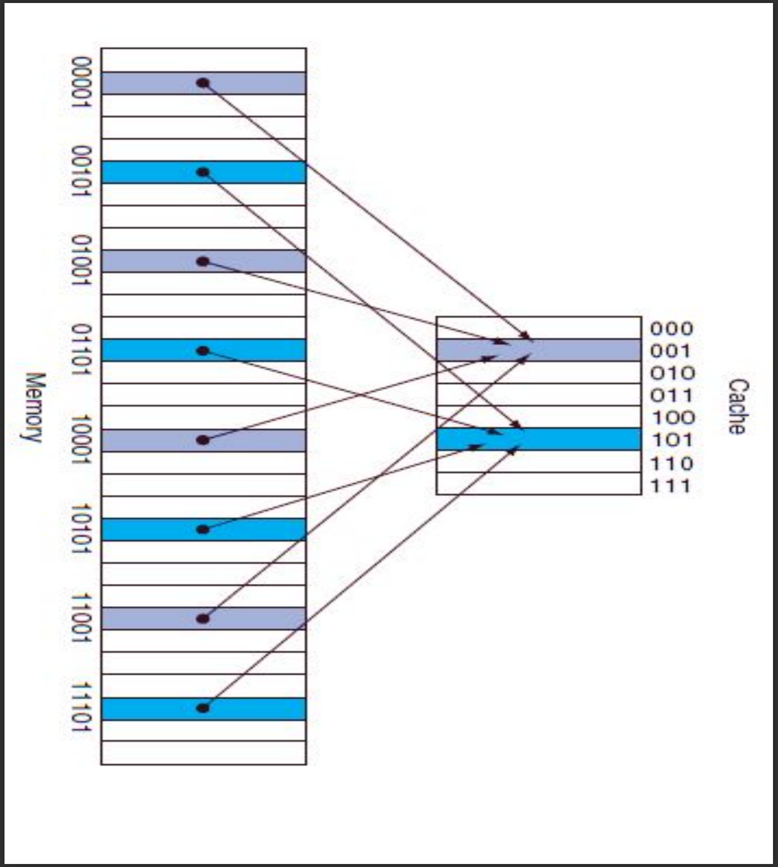
- **Lets answer couple of questions:**
- **Do we have a method to find if the data item is in the cache?**
- **If it is there, How do we zero in it?**
- The questions are definitely related and hence the answers would also be.
- Well, let us understand this:
  - If each word (data) can go in one place **(I mean exactly one place, one location) then, it is straight forward to the find the presence in the cache.**
- Next, how do you assign a location in the cache for the word in memory? (I mean, how do you give a slot to the word in the memory in cache?)
  - Can we use the address of the word in memory to do this?
  - If this done, then we can name it as **"Direct Mapped"** approach. Each memory location is mapped directly to one location in the cache.
  - This is very simple and straight.

Contd.,



- The picture shown in the left is taken as a reference for direct mapping.
- Addresses in the memory (from 00001 to 11101) are mapped to (000 to 101) in the cache. This is called direct mapping.
- Yes, this is easier to understand.
- **See this point. Though 000 to 111 are there in cache, we map to 000 to 101 only. (Again, this is not so technical)**

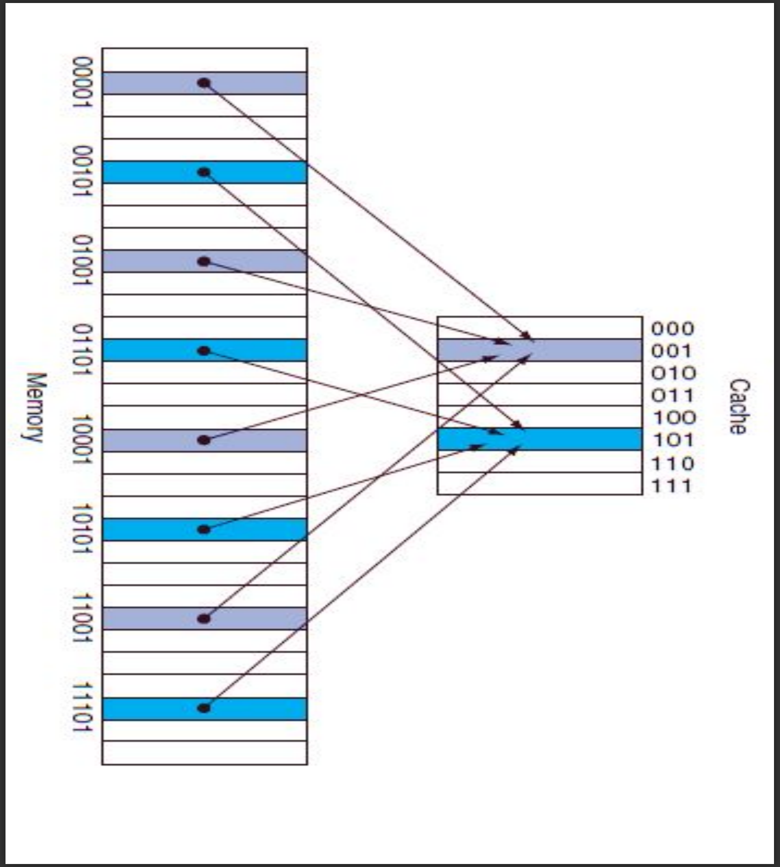
Contd.,



- Now, there is a challenge.
- Each cache location shall contain contents of many memory locations **(See the left, 001 has many arrows coming in)** how do we know if the data in the cache is the one that we are looking forward to ???
  - Challenging!
  - Means, how do we know if the word we search for is inside the cache or not?
- Well, we will make it a bit complex now. We will add something called tag. The will carry the address information which can help in identifying the words in the cache appropriately. Means, it helps in understanding if the word inside cache is the one we look for.

Contd.,

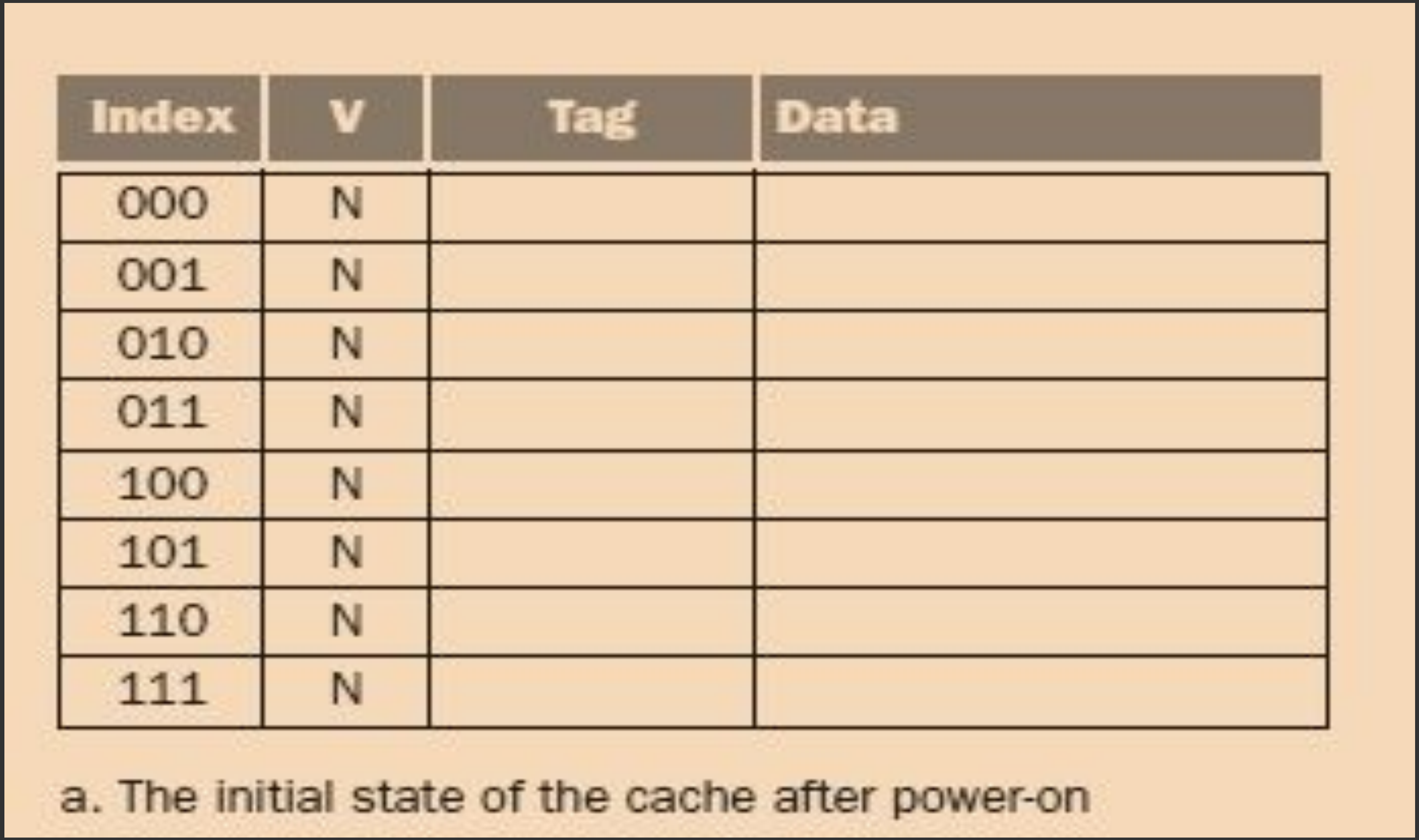
- Now comes the next question.
- How would the tag look like?
- For example, (See RHS) we need only to have the upper 2 of the 5 address bits in the tag, since the lower 3-bit index field of the address selects the block.



Contd.,

- There is a problem/Challenge – We need to identify if a cache block has no valid information.
- Add the valid bit!
- Simple. This valid bit will indicate if the entry has a valid address.
- If the bit is not set, there cannot be a match for this block.

An example is handy! Isn't it ?

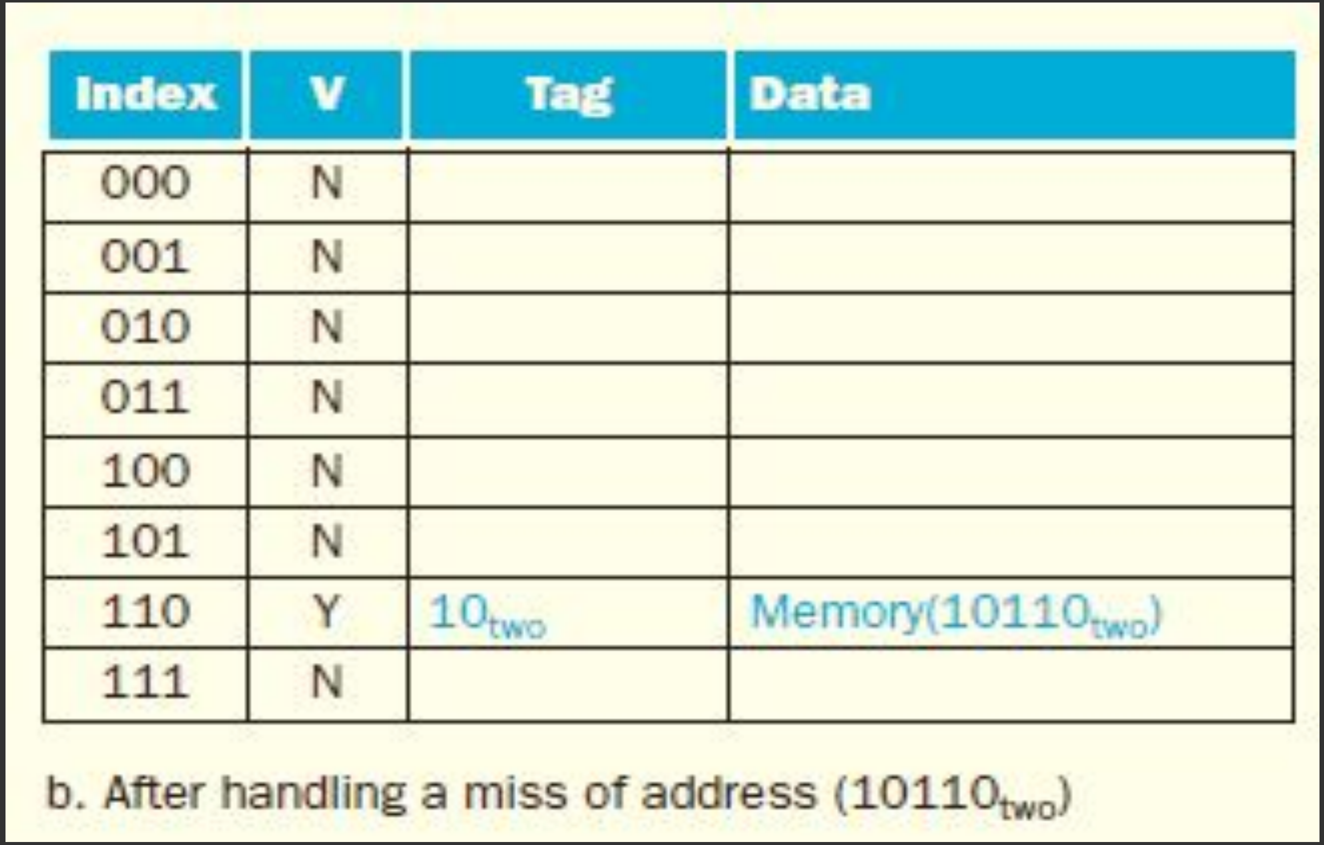


Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

The cache is initially empty, with all valid bits (V entry in cache) turned off (N).

Attempt -1: The processor requests the following address: 10110_{two}

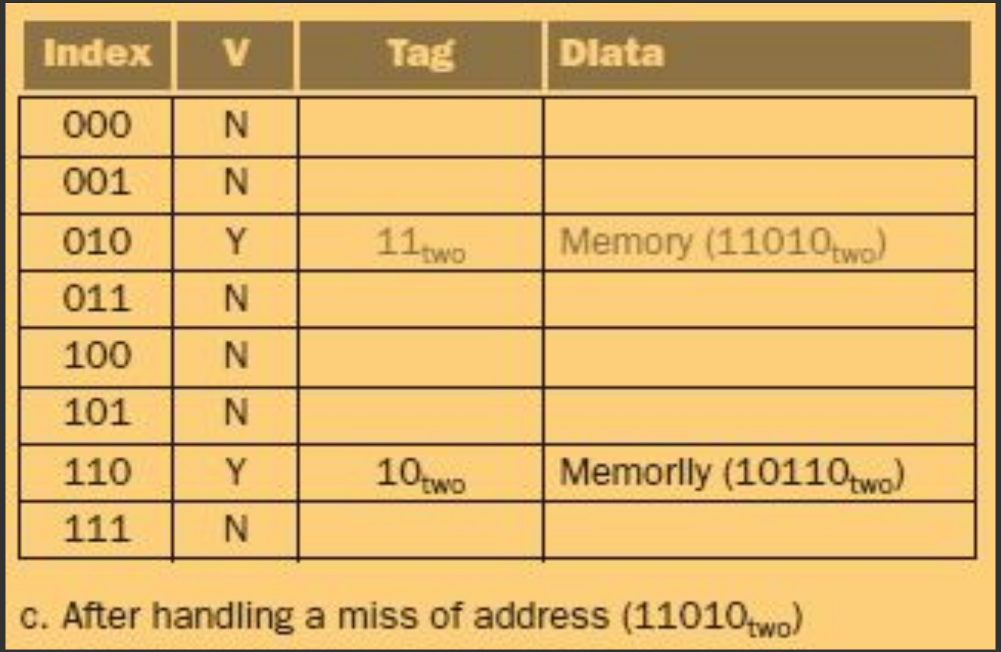


Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

The cache is initially empty, with all valid bits (V entry in cache) turned off (N). The processor requests the following addresses: 10110_{two}(miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).

Attempt -2: The processor requests the following address: 11010_{two} (miss)

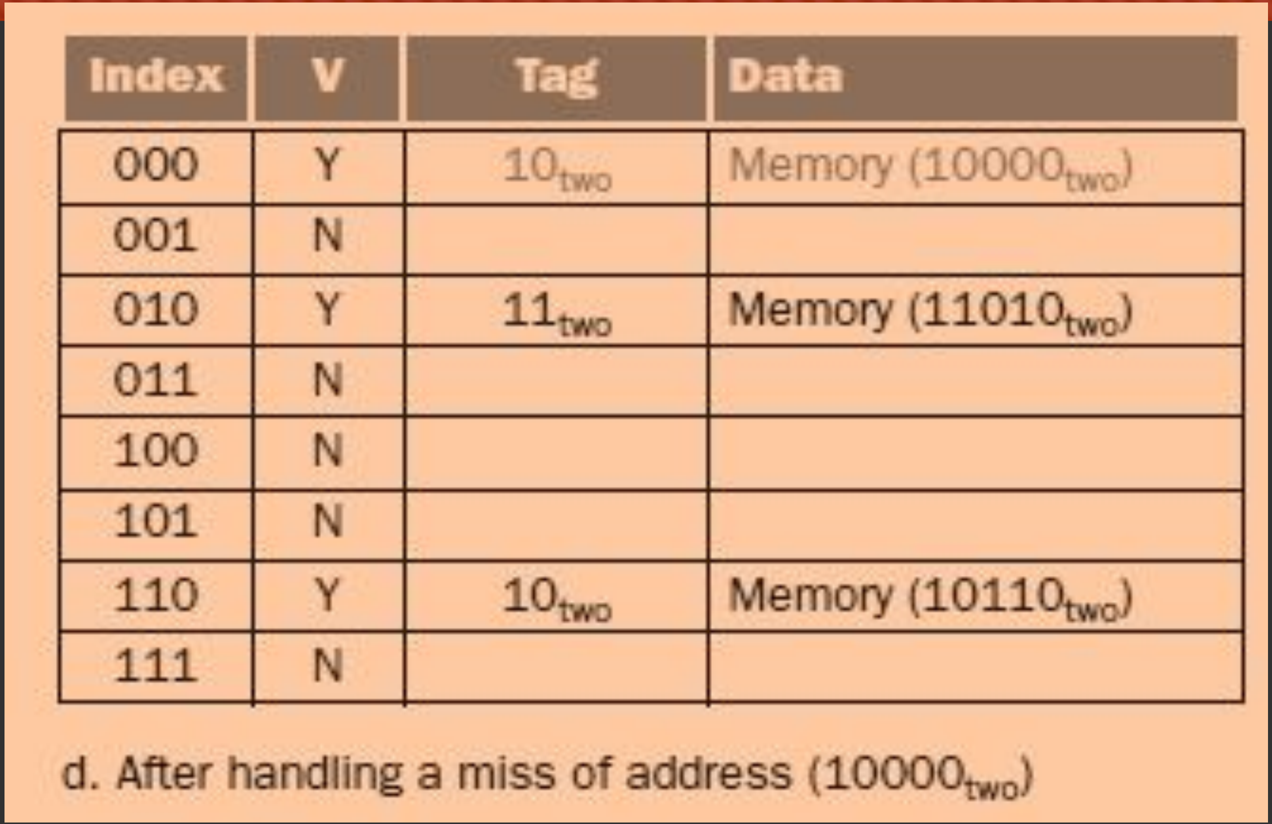


Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

The cache is initially empty, with all valid bits (V entry in cache) turned off (N). The processor requests the following addresses: 10110_{two}(miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).

Attempt -3: The processor requests the following address: 10110_{two} (hit)

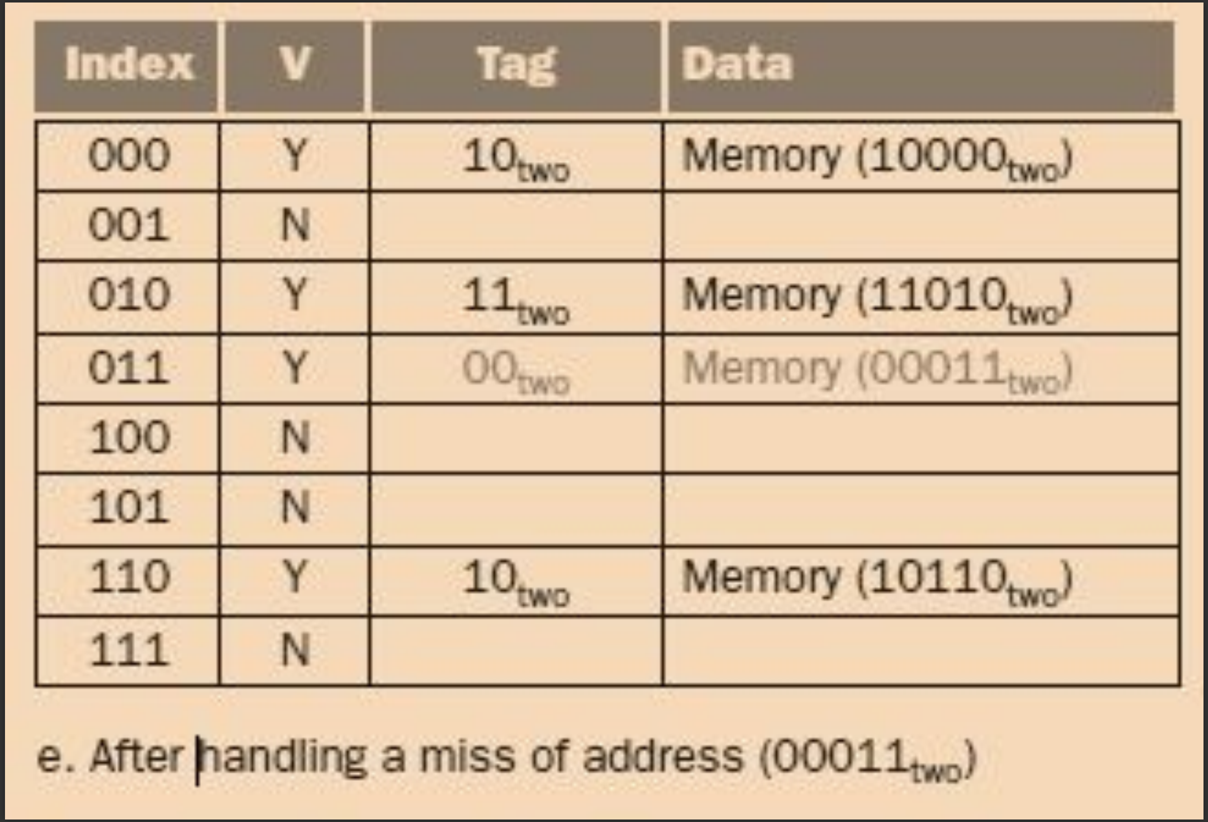


Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

The processor requests the following addresses: 10110_{two} (miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).

Attempt - 4

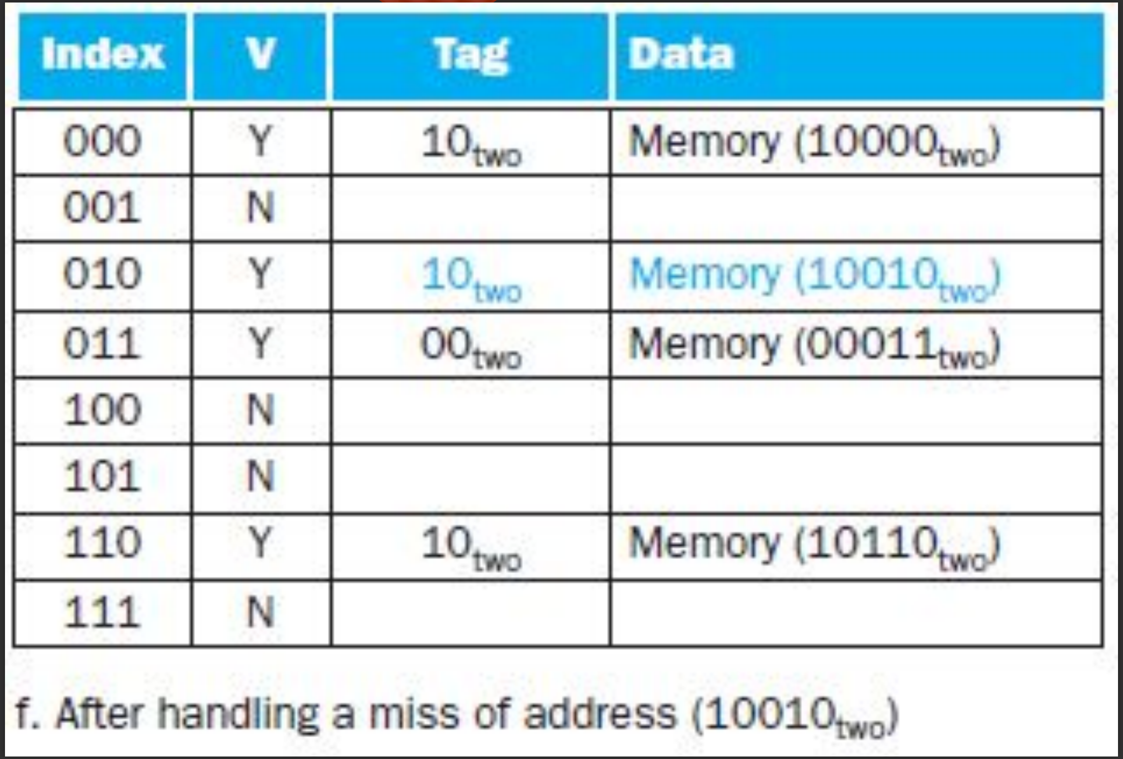


Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

The processor requests the following addresses: 10110_{two} **(miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).**

Attempt - 5



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

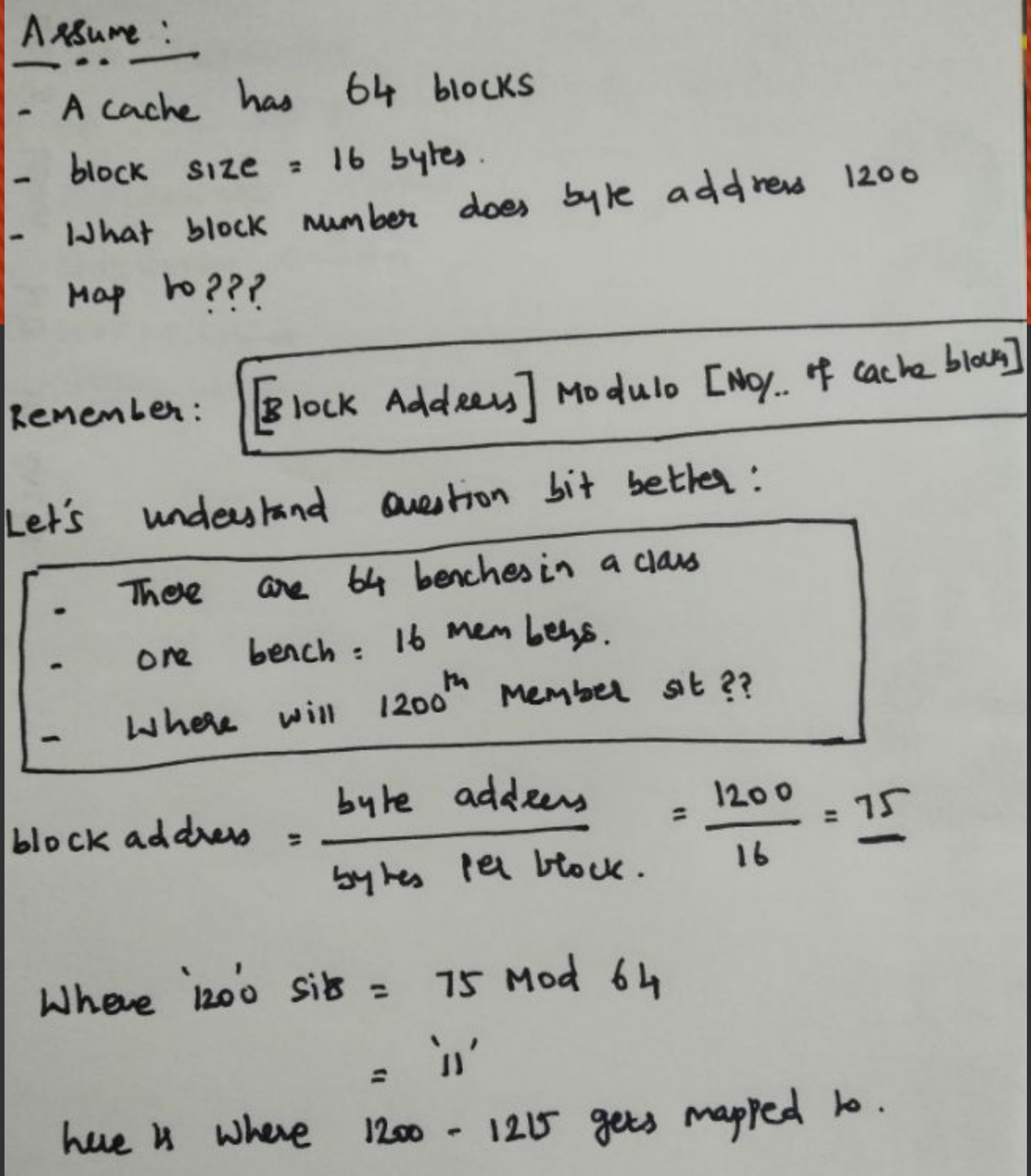
f. After handling a miss of address (10010_{two})

When address 10010_{two} (18) is referenced, the entry for address 11010_{two} (26) must be replaced, and a reference to 11010_{two} will cause a subsequent miss

Understand the way tag is used.

The processor requests the following addresses: 10110_{two} (miss), 11010_{two} (miss), 10110_{two} (hit), 11010_{two} (hit), 10000_{two} (miss), 00011_{two} (miss), 10000_{two} (hit), and 10010_{two} (miss).

Some math…



Assume:

- A cache has 64 blocks
- block size = 16 bytes.
- What block number does byte address 1200 map to???

Remember: [Block Address] Modulo [No. of cache blocks]

Let's understand question bit better:

- There are 64 benches in a class
- one bench = 16 members.
- Where will 1200th member sit??

$$\text{block address} = \frac{\text{byte address}}{\text{bytes per block.}} = \frac{1200}{16} = \underline{75}$$

Where '1200' sits = 75 Mod 64

= '11'

here is where 1200 - 1215 gets mapped to.

Handling cache miss … (Its complicated)

- You know by now, what is a cache miss!
- When the requested content is not in the cache, the request cannot be handled as the data is not available.
- Now, when there is a hit, the action is all trivial. Means, the things will go as planned.
- *If there is a miss, the process is going to be slightly different as the flow is to be changed to handle the miss.*
  - The cache miss handling is done with the processor control unit and with a separate controller that initiates the memory access and refills the cache.
- This miss needs processing to be done and this creates some delay. It is similar to the pipeline stall (remember??) which forced us to save the content in pipelining registers.
- For a cache miss, we can stall the entire processor, essentially freezing the contents of the temporary and programmer-visible registers, while we wait for memory.
  - In contrast, pipeline stalls, are more complex because we must continue executing some instructions while we stall others.

Contd.,

- A request for data from the cache that cannot be filled because the data is not present in the cache.
- The steps to be taken on an instruction cache miss:

1. **Send the original PC value (current PC – 4) to the memory.**
2. **Instruct main memory to perform a read and wait for the memory to complete its access.**
3. **Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.**
4. **Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.**

Cache and main memory Inconsistency

Cache and main memory are said to be inconsistent when a data is written only in cache without changing memory. (You agree right??) Techniques to counter it are listed below:

- **(a) write-through**: A scheme in which writes always update both the cache and the memory, ensuring that data is always consistent between the two writes
- (b) **write buffer** stores the data while it is waiting to be written to memory. After writing the data into the cache and into the write buffer, the processor can continue execution. When a write to main memory completes, the entry in the write buffer is freed. If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.
- (c) In a **write-back** scheme, when a write occurs, the new value is written only to the block in the cache. The modified block is written to the lower level of the hierarchy when it is replaced. Write-back scheme is, however, more complex to implement than write-through.

Reducing Cache Misses by More Flexible Placement of Blocks

- We have learnt the direct mapping techniques in the past!
- Let’s learn a bit more than that now.
- **Fully Associative and Set Associative – Let’s learn them in parallel.**

Fully Associative Approach.

- In this scheme,
  - a block can be placed in *any* location in the cache. (Flexible)
  - But, here comes a complexity.
  - In this approach, **all the entries in the cache must be searched because a block can be placed in any one.**
  - **Comparators will come into the picture here. A comparator is needed for each cache entry and is an expensive affair.**
  - **So, this is complex for a relatively larger cache.**

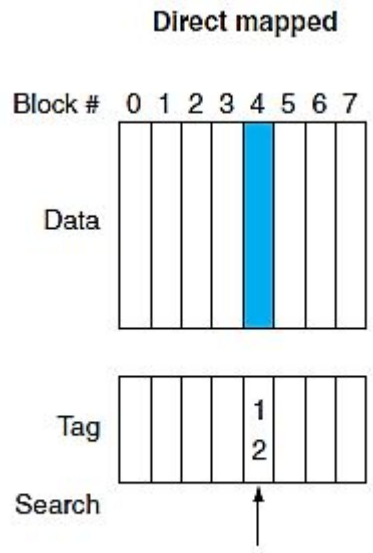
Set associative approach

- This is an intermediary approach.
- Not as complicated as the fully associative approach.
- **There are a fixed number of locations (at least two) where each block can be placed;**
- **It has the features and both the previous techniques.**
- **Here, a block is directly mapped into a set, and then all the blocks in the set are searched for a match.**

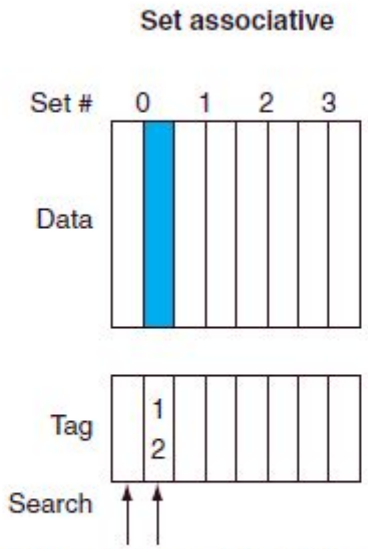
Formula.

- direct-mapped cache, the position of a memory block is given by
  - **(Block number) modulo (Number of cache blocks)**
- set-associative cache, the set containing a memory block is given by
  - **(Block number) modulo (Number of sets in the cache)**

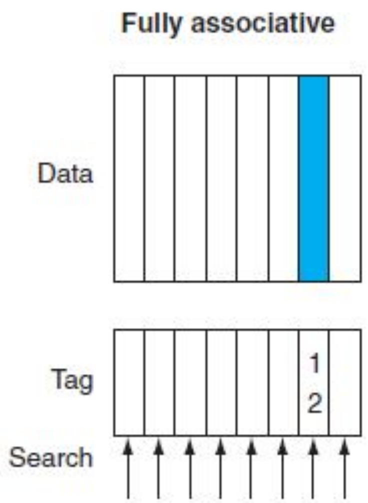
A simple example…



There can be one block. Memory block 12 shall be available in 12 mod 8 = 4.



8 Blocks (Given in the problem). Hence, for a 2 way set associative cache, it is 4 sets. 12 mod 4 = 0. The memory block shall be in either of the two elements of set 0.



No tough deal. Memory block 12 can be in any of the 8 cache blocks.

Measuring and Improving Cache Performance

- We explore two different techniques for improving cache performance.
  - **One focuses on reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location.**
  - **The second technique reduces the miss penalty by adding an additional level to the hierarchy.**
- This technique, called *multilevel caching*, first appeared in high-end computers and now available even in “Dabba” computers!

Contd.,

- CPU time can be divided **into the clock cycles that the CPU spends executing the program** and the **clock cycles that the CPU spends waiting for the memory system.**
- Normally, we assume that the costs of cache accesses that are hits are part of the normal CPU execution cycles.
- Thus,
  - **CPU time = (CPU execution clock cycles + Memory-stall clock cycles) * Clock cycle time**
- The memory-stall clock cycles come primarily from cache misses, and we make that assumption here.

Contd.,

- Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from writes:
- The read-stall cycles can be defined in terms of the number of read accesses per program, the miss penalty in clock cycles for a read, and the read miss rate:
  - **Memory-stall clock cycles = Read-stall cycles + Write-stall cycles**
- The read-stall cycles can be defined in terms of the number of read accesses per program, the miss penalty in clock cycles for a read, and the read miss rate:

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

Contd.,

- Writes are more complicated. For a write-through scheme, we have two sources of stalls:
  - **write misses**, which usually require that we fetch the block before continuing the write.
  - **write buffer stalls**, which occur when the write buffer is full when a write occurs.
- Thus, the cycles stalled for writes equals the sum of these two:

$$\text{Write-stall cycles} = \frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} + \text{Write buffer stalls}$$

Calculating Cache Performance

The number of memory miss cycles for instructions in terms of the Instruction count (I) is

$$\text{Instruction miss cycles} = I \times 2\% \times 100 = 2.00 \times I$$

The frequency of all loads and stores in SPECint2000 is 36%. Therefore, we can find the number of memory miss cycles for data references:

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I. This is more than 3 cycles of memory stall per instruction. Accordingly, the CPI with memory stalls is 2 + 3.44 = 5.44. Since there is no change in instruction count or clock rate, the ratio of the CPU execution times is

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

The performance with the perfect cache is better by $\frac{5.44}{2} = 2.72$.

thanks

Shriram K Vasudevan