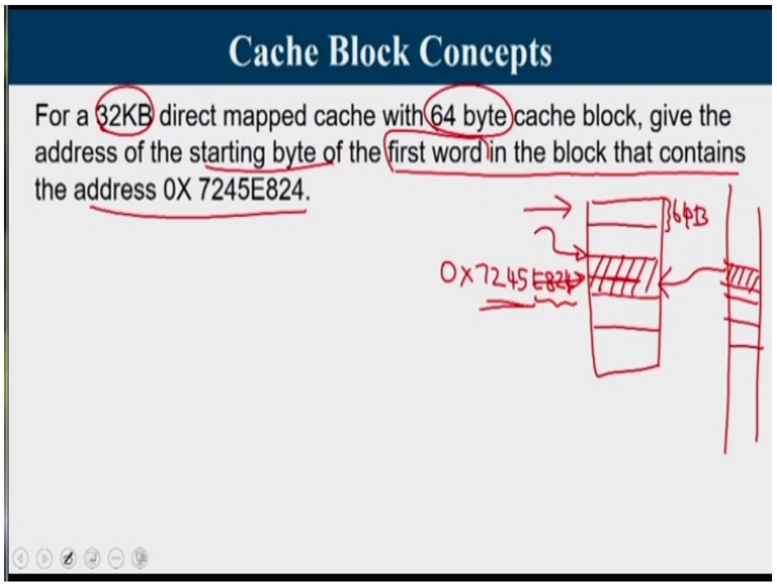
Advanced Computer Architecture

Prof. Dr. John Jose
Assistant Professor
Department of Computer Science & Engineering
Indian Institute of Technology-Guwahati

Lecture-20
Tutorial 6
Design Concepts in Cache Memory

Welcome all of you, tutorial number 6, in this tutorial we will be taking a couple of numerical questions associated with the basic design principles in cache memory. Over the lecture videos that was released in this week you might have seen the basics of cache memory and what do you mean by cache memory mapping and different cache block replacement techniques. This tutorial is designed keeping those topics that we have discussed in this week to get more clarity on the concepts that you have already view in the lecture videos.

(Refer Slide Time: 01:12)



Cache Block Concepts

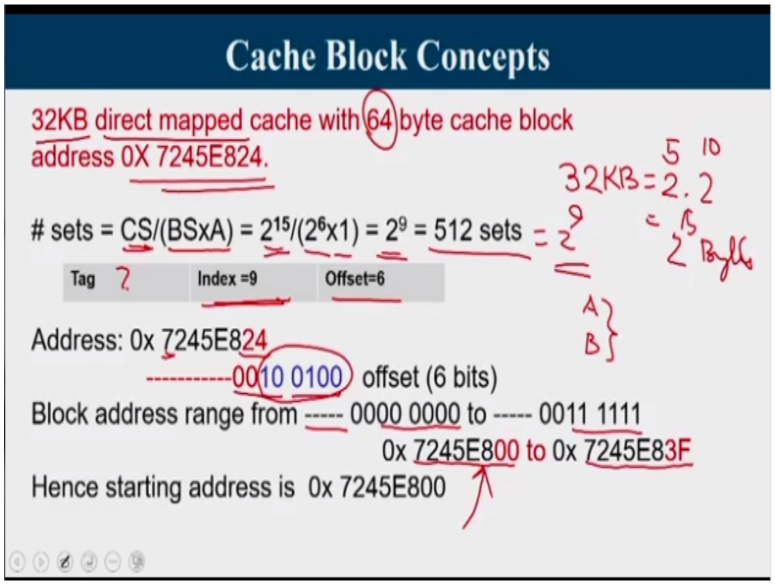
For a 32KB direct mapped cache with 64 byte cache block, give the address of the starting byte of the first word in the block that contains the address 0X 7245E824.

Our first question is on for a 32KB direct mapped cache with 64 byte cache clock give the address of the starting byte of the first word in the block that contains the address 0X7245E824. So here we have a direct mapped cache where we have multiple blocks inside the cache, now when you bring from main memory you are actually transferring a block of data. Let us say this is the block of data that get transferred into this particular block.

Now if this cache is say 32KB direct mapped cache and the block size this block size is 64 bytes give the address of the starting byte of the first word in the block that contains the address 0X7245E824. So what the question is let us say the address 0X7245E824 that is located let us say midway inside a cache block. The question that is asked is what will be the address of the starting byte of the first word in the block that contains this particular address.

We have to understand that when you bring contents from the main memory you are bringing a block of data and these are continuous blocks inside the main memory. So if you look at the address of those words that are brought into 1 cache block they will differ only in the few least significant bits whereas the most significant bits of the address will be same. So when you move to the very adjacent byte inside main memory only there is a change in the LSB's . So keeping this is in mind let us try to understand what is the context given in this question.

(Refer Slide Time: 03:26)



Cache Block Concepts

32KB direct mapped cache with 64 byte cache block address 0X 7245E824.

sets = CS/(BSxA) = $2^{15}/(2^6 \times 1) = 2^9$ = 512 sets $= 2^9$

Tag ?	Index =9	Offset=6

Address: 0x 7245E824

----------0010 0100 offset (6 bits)

Block address range from ----- 0000 0000 to ----- 0011 1111

0x 7245E800 to 0x 7245E83F

Hence starting address is 0x 7245E800

$32KB = 2^5 . 2^{10} = 2^{15}$ Bytes

A }
B }

So it is a 32KB direct mapped cache with 64 byte cache block, so let us first find out how many sets are there in the cache. So this is a direct mapped cache so the number of sets will be equal to the number of cache lines or cache block, number of sets is defined as cache size divided by block size into associativity. So here the cache is 32KB, so when you have 32KB we will try to represent them in power of 2

$$32 = 2^5$$

$$\text{and kilo} = 2^{10}$$

So that means we have total of

$$2^{15} \text{ bytes}$$

that is there in this particular cache. So 2 power 15 which represents 32KB divided by in the denominator we have block size,we know that the block size is 64 bytes,

$$64 = 2^6$$

and associativity is 1. Because it is a direct mapped cache, so it is

$$2^{16} / 2^6 = 2^9$$

so this particular cache has 512 sets inside it. Now we have to divide the physical address into tag index and offset.

But we do not know what is the total physical addressability the number of bits in the physical address. Anyway let us imagine that we have an n bit physical address out of it is the middle few bits is for set index. Since we have 512 sets which is equal to 2 power 9 what is been showed 9 bits in the physical address is used for indexing. And the block size is 64 bytes, so the last 6 bits is used for offset and the remaining bit is for the tag anyway here the tag bit is not relevant.

So what is a meaning of this, the last 6 bits of the address will represent the offset that means if I have some address A another address B if they belong to the same block then they differ only utmost in the last 6 bits whereas all other more significant bits will be same. So let us write the address that is been asked 0X7245E824 this is written in hexadecimal this is a 32 bit address where each of this represents a 4 bit value.

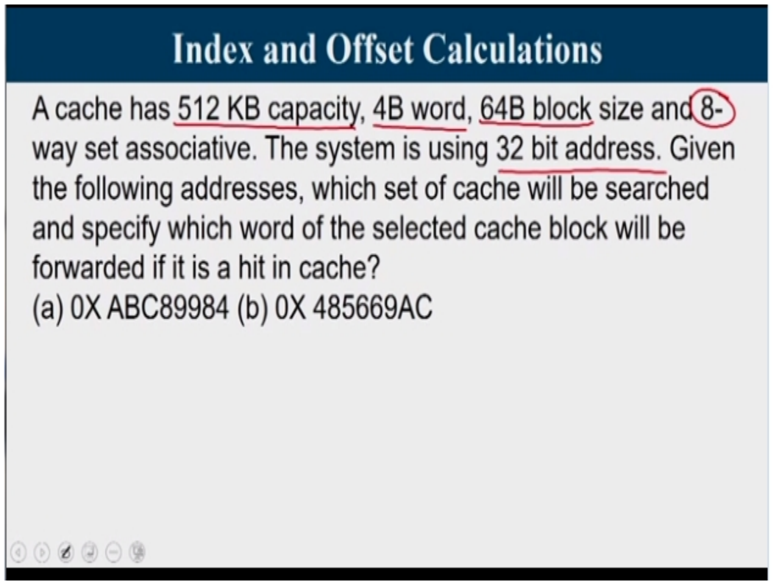
Let us take the last 2 digits in the hexadecimal notation 24 if I expand it to hexadecimal, 2 can be written as 0010 and 4 is written as 0100. Now out of is last 8 bits which comes in the address the last 6 bit will represent offset. So what you see in this blue colour that is the offset portion, so this is the offset that is been given. Now the block address typically range from whatever is a most significant bits it is not going to change, the last 6 bits will range from 6 zeros all the way up to 6 ones.

Whereas the remaining mass significant bits will remain same, so if you put all zeros for this particular address that is been given. If the last 6 bit of the address that is been given in the question if I make it 0 then it becomes 7245E800 that is the address of the first byte in the block. And 7245E83F that is the address of the last byte in the block, so the starting address is then this one 7245E800.

So what we have done in this question, in this question a cache memory detail is been given the block size is been mentioned all what we have to understand is when we copy something from main memory into the cache continuous bytes are been copied. So if you look at if you examine the address of these bytes that belong to 1 cache block they differ only in the least significant bits, how many least significant bit.

It is been defined by the offset bits what is the block size, in this particular question the block size is 64 bytes that means they these the bytes that are part of 1 block will differ only in the last 6 bits of it is address. So if the last 6 bit if you completely put 0 that is very first byte and then all combinations of those 6 bit will give you different byte in the same block ending with last 6 bit values given to all ones.

(Refer Slide Time: 07:57)



Index and Offset Calculations

A cache has 512 KB capacity, 4B word, 64B block size and 8-way set associative. The system is using 32 bit address. Given the following addresses, which set of cache will be searched and specify which word of the selected cache block will be forwarded if it is a hit in cache?

(a) 0X ABC89984 (b) 0X 485669AC

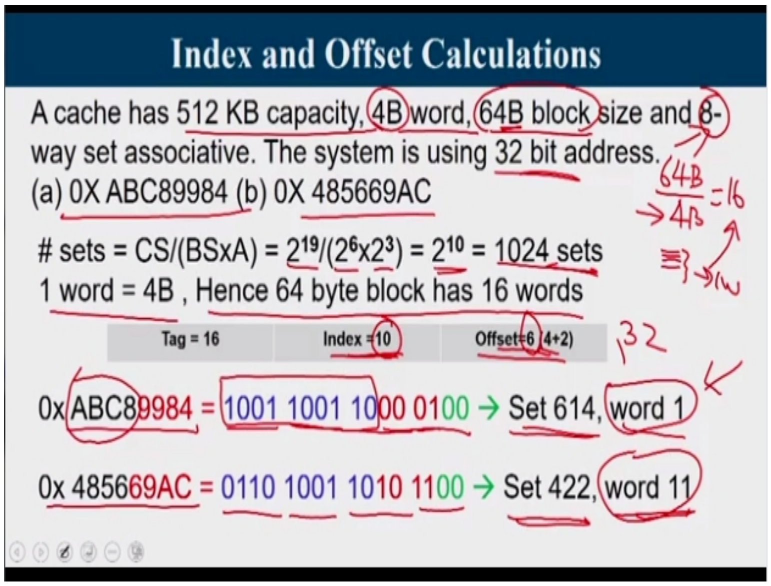
Let us now move onto the second question it is basically on index and offset calculation, a cache has 512KB capacity, 4 byte word, 64 byte block and it is 8 way associative. The system is using 32 bit address given the following addresses which set of the cache will be searched and specify which word of the selected cache block will be forwarded if it is a hit in the cache. Let me try to re-straight this question, given an address and given a specification of cache.

Now in this context we want to know to understand wherein cache memory the address matching happens. We are not searching in the entire cache, we know that cache is a very fast memory, it is going to respond back to the processor request quickly because it localizes it search to limited location. Now this particular question where I am giving an address and the question is where in the question in which set of the cache tag matching happens.

And if it is a hit then there is a block wherein we get a hit inside the block which is the word that gets transferred. So first we have to understand the split up of the physical address. Any given cache memory question it is specification like this, you have to find out the split up of the address from tag index and offset. And it is a index bit that will tell you bit set the search is and once you know the index number and then in that particular set index the tag matching has to happen.

And if at all there is a match then the offset will tell you which word is to be transferred, it will be more clear to you once we understand this example in it is detailed working.

(Refer Slide Time: 09:43)



Index and Offset Calculations

A cache has 512 KB capacity, 4B word, 64B block size and 8-way set associative. The system is using 32 bit address.
(a) 0X ABC89984 (b) 0X 485669AC

sets = CS/(BSxA) = $2^{19}/(2^6 x 2^3) = 2^{10}$ = 1024 sets
1 word = 4B , Hence 64 byte block has 16 words

Tag = 16	Index = 10	Offset = 6 (4+2)

0x ABC89984 = 1001 1001 1000 0100 → Set 614, word 1

0x 485669AC = 0110 1001 1010 1100 → Set 422, word 11

$\frac{64B}{4B} = 16$
32

So this is the details that is been given, it is a 512KB cache with 4 byte word and 64 byte block, it is a 4 byte word. So if it have 64 byte blocks and if you divide it into 4 bytes then there are 16 words that get stored in 1 block, it is a 8 way associative. So the first one is number of sets is equal to same like the previous question cache size divided by block size into associativity. Here cache size is 512KB, 512 is 2 power 9 and K is 2 power 10, so 2 power 19 that is your 512KB, block size is 64 bytes, so it is 2 power 6.

And associativity is 8, it is an 8 way associative cache so it is 2 power 3, so we will get 2 power 10 sets.

$$2^{19}/(2^{6} * 2^{3}) = 2^{10}$$

This particular cache has 1024 sets in it, it is mention that 1 word = 4 bytes, hence 64 byte block can accommodate

64 byte / 4 byte =16 words

that is been shown in our previous calculation. So if you have 1024 sets then there are 10 bits that is gone for index, we have total of 32 bit address that is been given in the question.

So this 32 bit address inside that 10 bit is going for index and these 10 bits will tell you which is a set out of 1024 set which set has to be searched and then we have an offset that is equal to 6 bits for the offset. Because it is 64 byte cache block, now out of the 6 bit the first 4 bit will tell you the word number, the next 2 bit tell you which is a byte within this word. You have to understand here we have 4 bytes and it is 4 continuous bytes that form 1 word.

So the address of the bytes within a word they will differ only in the last 2 bits that is by in the offset that is been given. Out of the 6 bit in the offset the first 4 bit is 4, the word number and the next 2 bit for the offset. So let us now consider the address 0xABC89984, so we have seen that out of the 32 bit address, so physical address is 32 bit more significant 16 bit is tag, next 10 bit is index and the last 6 bit is the offset.

So in this address whatever is given in the red portion they constitute the last 16 bits I am expanding the last 16 bits 9 stands for 1001 again 9 1001, 8 is 1000 and 4 is 0100. And here the blue portion that is 10 bit blue portion indicates a set number, the 4 bit red portion indicate the word number and the 2 bit green is telling the byte within the word. So if you compute the decimal value of this blue portion, this 10 bit blue you will get a number 614.

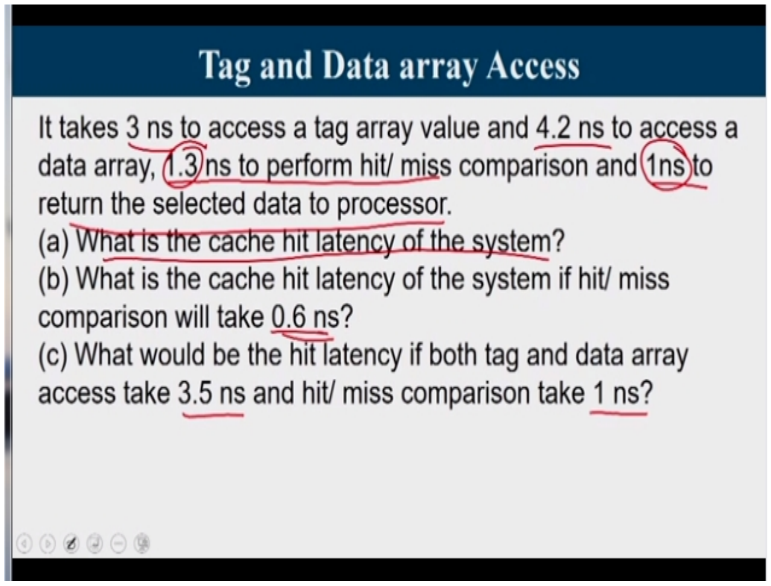
If all the values of blue bits are equal to 1 then that correspond to 1023 set number 1023, so in this particular address this 10 bits constitute a decimal value 614 and if you look at the red portion it is word number 1. That means when you get this particular address ABC89984 into a cache memory, the set 614 is been searched. In set 614 since it is 8 way associative there are 8 cache lines in all the 8 cache lines we see whether the tag value stored is same as this 16 bit ABC8, the hexadecimal ABC8 is the tag that is saved.

If the tag is matching then in that particular block where there a match was there you transfer the very first, you transfer word number 1 not word number 0. The very first one is word number 0, this is word number 1, so transfer word number 1 that is the required one. Similar to that if you explore the second address 485669AC the red portion indicates the last 16 bit expanding that 6 is 0110, 9 is 1001, a is 1010 and c is 1100.

So the blue portion indicate set number 422 and the red portion indicates word number it is word 11. So the search is restricted only to set number 422 within that perform a tag comparison and whichever block get a cache hit on the tag match. Then in that particular block transfer word number 11, so to summarize what we do in this question whenever you get the cache details and the address find out the set index portion that will tell you which is a set is to be searched and then you find out the corresponding offset.

So from the binary value find out the decimal and that is a set wherein the search for tag match happens.

(Refer Slide Time: 14:38)


Tag and Data array Access

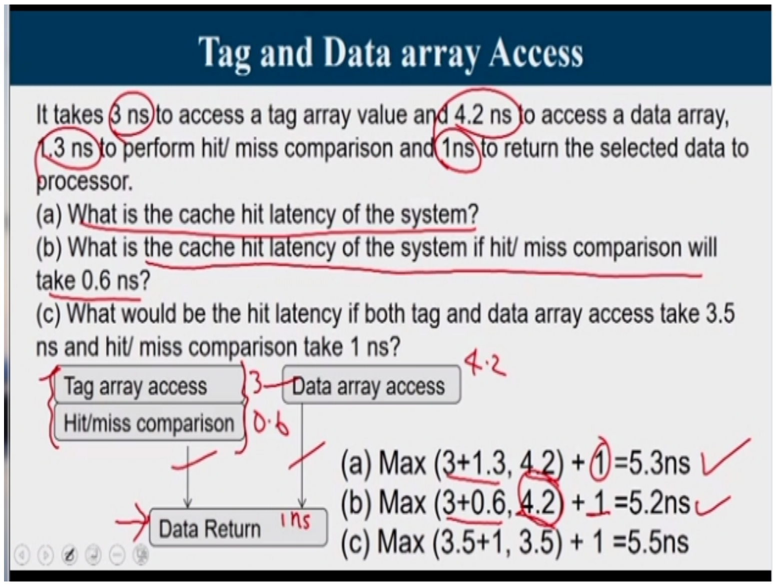
It takes 3 ns to access a tag array value and 4.2 ns to access a data array, 1.3 ns to perform hit/ miss comparison and 1ns to return the selected data to processor.
(a) What is the cache hit latency of the system?
(b) What is the cache hit latency of the system if hit/ miss comparison will take 0.6 ns?
(c) What would be the hit latency if both tag and data array access take 3.5 ns and hit/ miss comparison take 1 ns?

Now the next problem is on tag and data array access, it takes 3 nanosecond to access the tag array value and 4.2 nanosecond to access the data array value, 1.3 nanosecond is taken to perform hit or miss comparison and 1 nanosecond to return the selected data to the processor. So we have to understand that there is a tag array portion and there is a data array portion and time to access tag and data array portions are different.

And after the tag is been accessed you perform a hit or miss comparison, so that happens after the tag array operation and once the hit happens then it take sometime. Here in this case it is 1 nanosecond time to return the selected data, what is cache a hit latency of the system. And then the second portion tells that what is the cache hit latency of the system if the hit or miss comparison which in this question is initially 1.3 if that become 0.6 what is the change, what would be the hit latency.

If both tag and data access takes 3.5 nanosecond each and hit or miss comparison take only 1 nanosecond. So there 3 subdivisions, let us take it one by one.

(Refer Slide Time: 15:46)



Tag and Data array Access

It takes 3 ns to access a tag array value and 4.2 ns to access a data array, 1.3 ns to perform hit/ miss comparison and 1ns to return the selected data to processor.

(a) What is the cache hit latency of the system?

(b) What is the cache hit latency of the system if hit/ miss comparison will take 0.6 ns?

(c) What would be the hit latency if both tag and data array access take 3.5 ns and hit/ miss comparison take 1 ns?

(a) Max (3+1.3, 4.2) + 1 =5.3ns ✓

(b) Max (3+0.6, 4.2) + 1 =5.2ns ✓

(c) Max (3.5+1, 3.5) + 1 =5.5ns

So we have to understand that there is 2 portion there is tag array portion and data array portion, the tag array and data array are accessed parallely but access time may differ. Once the tag is accessed after that only the hit or miss comparison time is applied. So there is 2 wing the left wing for tag array access followed by hit or miss comparison and the right wing for data array access which will take the data.

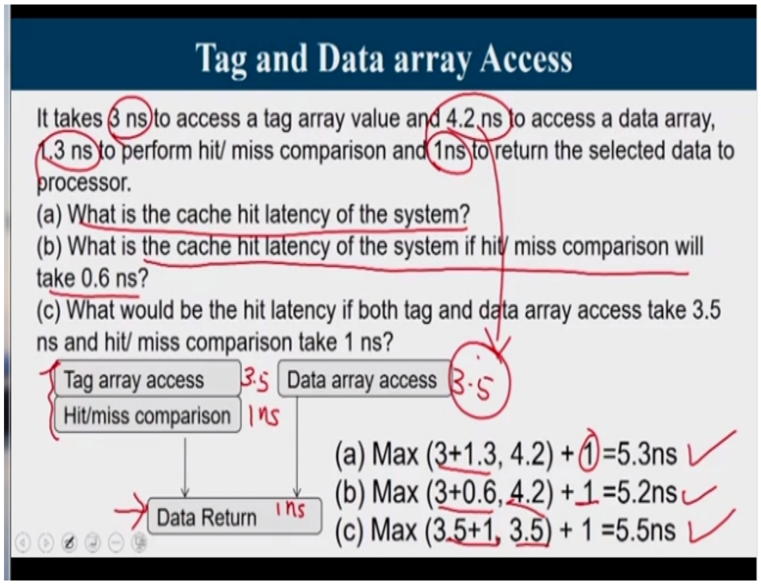
Now out of these 2 branches whichever is the more dominant one which is having more latency that determines when data return can start. So data return will start only if both the left side and the right side have completed the operation. Now in first case what is the cache hit latency, we know that the tag array access takes 3 nanosecond that is been given in the question and data array access take 4.2 nanosecond.

And this tag array section has hit or miss comparison logic which has 1.3 nanosecond. So this is 1.3 and this takes 1 nanosecond, so when you look at the left side it will take 3 + 1.3 that much is a time required to complete the tag array access and hit or miss comparison and the right side will take you 4.2. So left side is actually 4.3, 3 + 1.3 is 4.3 and the right side is 4.2, so the left side is dominant.

So it is 4.3 + 1 this 1 is this data return time, so the hit latency is 5.3 nanosecond, now the second question is what is the cache hit latency if hit or miss comparison take only 0.6 nanosecond. So this will become 0.6 then the left side is 3 + 0.6 and the right side there is no change it is 4.2 that is data array access is 4.2. So the dominant 1 is right side that is 4.2 is the dominant one, so 4.2 + 1, so your answer is 5.2 nanosecond.

So even though you reduce the left side from 1.3 nanosecond all the way up to 0.6 nanosecond you are not going to get much rewards because the right sided data array access is still the dominant fraction.

(Refer Slide Time: 17:58)



Tag and Data array Access

It takes 3 ns to access a tag array value and 4.2 ns to access a data array, 1.3 ns to perform hit/ miss comparison and 1ns to return the selected data to processor.

(a) What is the cache hit latency of the system?

(b) What is the cache hit latency of the system if hit/ miss comparison will take 0.6 ns?

(c) What would be the hit latency if both tag and data array access take 3.5 ns and hit/ miss comparison take 1 ns?

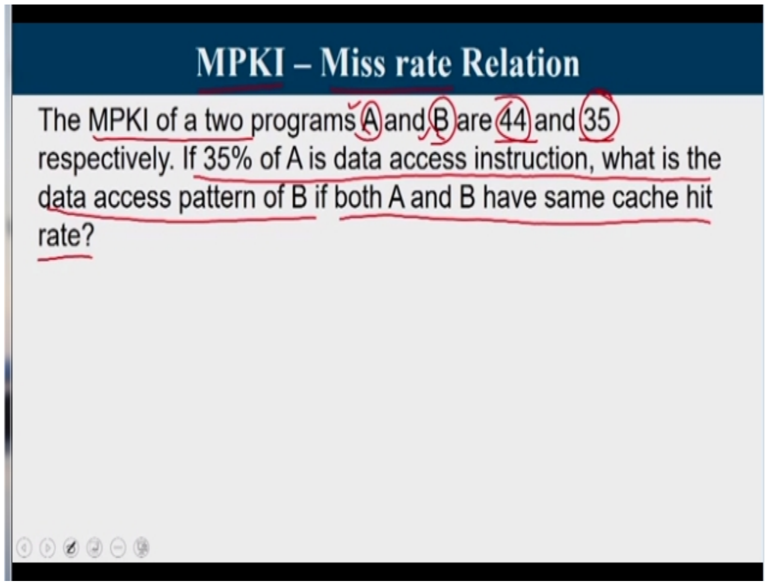
(a) Max (3+1.3, 4.2) + 1 =5.3ns ✓

(b) Max (3+0.6, 4.2) + 1 =5.2ns ✓

(c) Max (3.5+1, 3.5) + 1 =5.5ns ✓

Now we move onto the third one what would be the hit latency if both tag and data array access take 3.5 nanosecond. So this has 3.5 nanosecond, this also will be 3.5 nanosecond and the hit or miss comparison this takes 1 nanosecond. So the left side would be 3.5 + 1 that is total of 4.5 and the right side 3.5 only. So the dominant fraction is left side that is 4.5 + 1 that is 5.5 nanosecond. So even though the data array access got changed from 4.2 nanosecond all the way to 3.5 we are not getting rewards because the left side is now taking more amount of time.

(Refer Slide Time: 18:36)



MPKI – Miss rate Relation

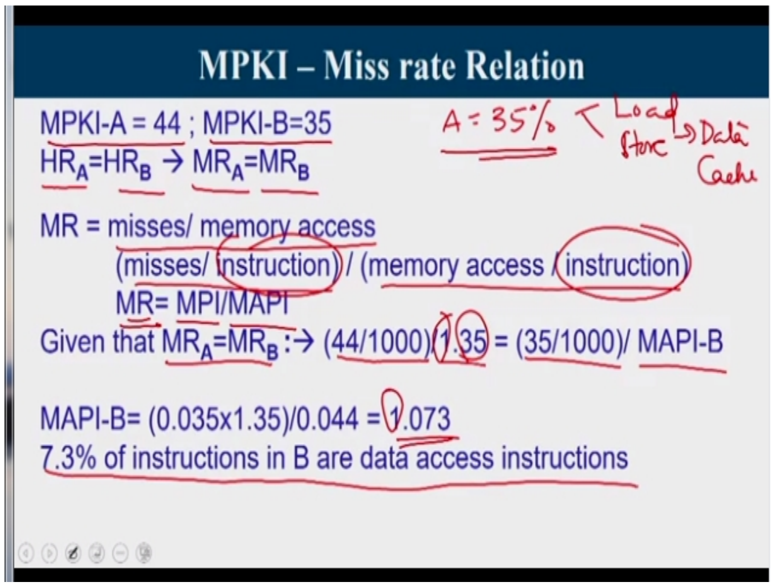
The MPKI of a two programs A and B are 44 and 35 respectively. If 35% of A is data access instruction, what is the data access pattern of B if both A and B have same cache hit rate?

So the question is all about the relation between MPKI, MPKI stands for misses per kilo instruction. So that is one of the important characteristic of a program how many times I miss when I execute 1000 instruction, 1kilo instructions. So when I complete 1kilo instruction how many misses are encountered that is called misses per kilo instruction. And you know that miss rate is number of misses divided by number of memory access.

So in this question the MPKI of 2 programs A and B are 44 and 35 respectively, that means when we run 1kilo instructions of A there are total of 44 misses, it can be misses in I cache as well as in d cache. Similarly when we execute 1000 instructions of B then there are 35 misses, so MPKI of B is less than that of MPKI of A. Now if 35% of A is data access instructions what is the data access pattern of B if both A and B have the same cache hit rate.

Now A and B have different misses per kilo instruction but as far as the hit in the cache is concerned they are same. The number of access A make to cache and the number of time there is a hit in cache, so number of miss divided by number of cache access is same in the case of A and B. That is why it is called cache hit rate is same.

(Refer Slide Time: 20:08)



MPKI – Miss rate Relation

MPKI-A = 44 ; MPKI-B=35

$HR_A=HR_B \rightarrow MR_A=MR_B$

MR = misses/ memory access

(misses/ instruction)/ (memory access / instruction)

MR= MPI/MAPI

Given that $MR_A=MR_B$:→ (44/1000)/1.35 = (35/1000)/ MAPI-B

MAPI-B= (0.035x1.35)/0.044 = 1.073

7.3% of instructions in B are data access instructions

A = 35% < Load Store → Data Cache

So now in this case we have to find out, so what we are telling is MPKI of A is 44, MPKI B is 35 that is given in the question, what is mentioned here hit rate A is same as hit rate of B. That means miss rate of A and B is same, so hit rate and miss rate will be same. Now miss rate is defined as

Miss rate = number of misses / number of memory access

or

Miss Rate= (misses / instruction) / (memory access / instruction)

Because the instruction component is common, so numerator we bring in a parameter called misses divided by instruction misses divided by instruction number of misses divided by instruction divided by number of memory access divided by instruction. So miss rate is defined as misses instruction divided by memory access divided by instruction MPI misses divided by instruction divided by MPI memory access divided by instruction.

MR=MPI/MAPI

Given that miss rate of A and B is same, so 44 that is the MPKI misses per kilo instruction.

So if you wanted to get MPI misses per instruction then 44 is divided with 1000 because 44 is the number of misses for executing 1000 instruction. So the number of misses for executing 1 instruction is

(44 /1000) /memory access per instruction

Here it is been mentioned that A has 35% of the data access instruction, data access means as per instruction pipeline they are the load and store instructions which are going to touch the data cache.

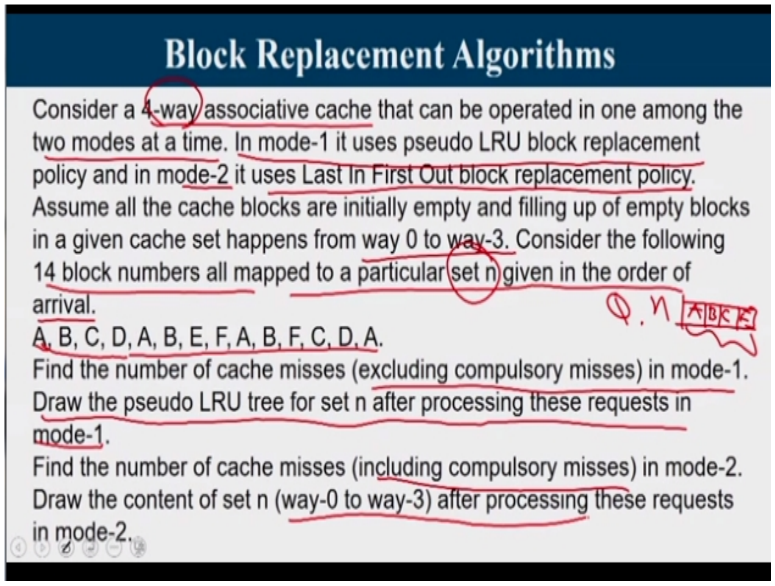
So 35% of the instructions are either load or store they are basically data transfer instruction. So we have to understand that for every instruction surely there is 1 we go to I cache, 1 I cache access is there every instruction has to be brought. But out of 100 instruction 35 of them will go to data cache, that means memory access per instruction is 1.35, this 1 is due to I cache and this 0.35 is due to d cache.

Or if you just imagine out of all the instructions that is being fetched none of them is load or store instruction then I go to memory only for fetching the instruction. So memory access per instruction is 1 just only one say in the instruction fetch stage and the mem stage would not be used. Let us assume that if all instructions are load or store instruction then every instruction will go to I cache once to bring the instruction and to d cache to bring the data or to access the data.

So in this case since it is 35% is data access instructions, we have 1.35 as memory access per instruction. That is same as

35 / 1000

that is misses per instruction for B divided by memory access per instruction of B which we do not know. So if you solve the equation memory access per instruction for B is 0.035 into 1.35 divided by 0.044 so you get it as memory access for per instruction for B is 1.073.
We know that 1 is I cache access there exist always 1 instruction is been accessed for every cycle. So once we have to go for every instruction there is 1 memory access from the I cache so remaining is 0.073, so 7.3% of instructions of B are data access instructions.
(Refer Slide Time: 23:33)



Block Replacement Algorithms

Consider a 4-way associative cache that can be operated in one among the two modes at a time. In mode-1 it uses pseudo LRU block replacement policy and in mode-2 it uses Last In First Out block replacement policy. Assume all the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3. Consider the following 14 block numbers all mapped to a particular set n given in the order of arrival.

A, B, C, D, A, B, E, F, A, B, F, C, D, A.

Find the number of cache misses (excluding compulsory misses) in mode-1. Draw the pseudo LRU tree for set n after processing these requests in mode-1.

Find the number of cache misses (including compulsory misses) in mode-2. Draw the content of set n (way-0 to way-3) after processing these requests in mode-2.

Q . n | A | B | C | F |

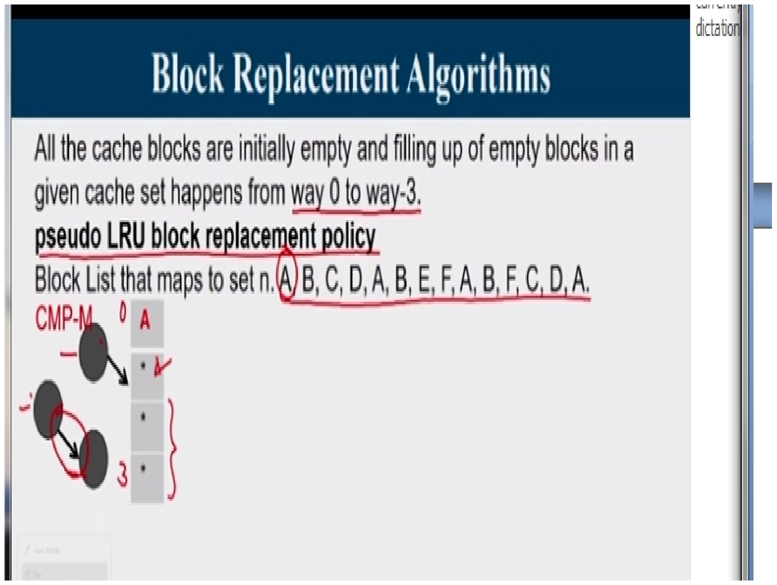
Now we move onto block replacement algorithms, consider a 4 way associative cache that is operated in 2 modes. In mode 1 it uses pseudo LRU block replacement policy and in mode 2 it uses last in first out block replacement policy. Assume that all the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way 3. So whenever the cache block is empty you always fill way 0 first and then followed by way 1, way 2, way 3 like that.

Once if all the blocks inside a cache set filled and all the ways are full then only replacement happens. So replacement is using pseudo LRU if it is in mode 1 and LRU and then last in first out in the case of mode 2. Consider the following 14 block numbers there are mapped to a particular set given in the order of arrival. So these are the block numbers all part of the same set, so they all these block numbers are mapping into the same set.

So they have to be kept in the 4 different ways, it is a 4 way associative cache in the 4 different way. So if you look at a snapshot let us say this is set number n and the set number n has 4 ways at some point there maybe A there, B, C let us say F. let us say this is the content, so in set number n at a given snapshot like there be block A, B, C, D and F residing. Now when let us say there is a new request Q that is going to come which one I am going to replace will I replace A or will I replace B, C or F, that is basically your block replacement question.

So find the number of cache misses that is happening excluding the compulsory miss in mode 1 draw the pseudo LRU tree for set n after processing this request in mode 1, that is part 1 of the question. Find the number of cache misses including the compulsory miss in mode 2 draw the content of set n way 0 to way 3 after processing this request in mode 2, so this is basically the question.

(Refer Slide Time: 25:44)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

pseudo LRU block replacement policy

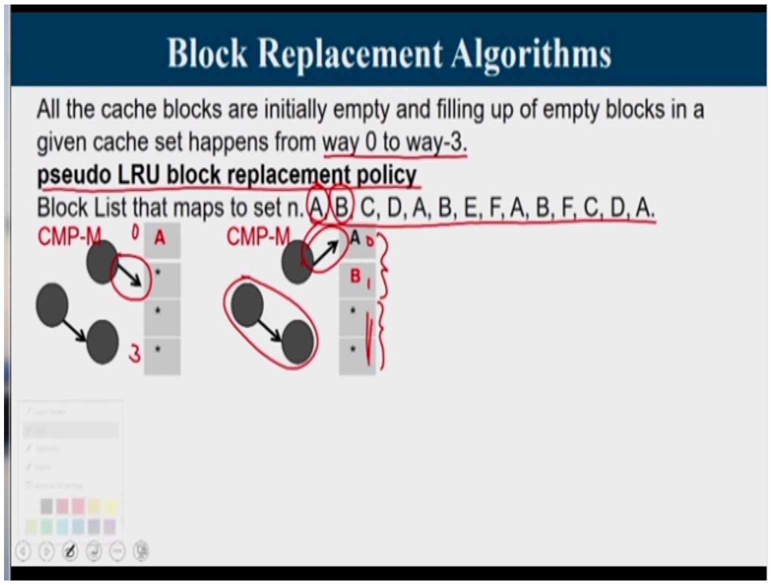
Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So all the cache blocks are initially empty and filling up of the empty blocks will happen from way 0 to way 3, we have pseudo LRU block replacement policy and these are all the block numbers that we are going to see. So since it is a 4 way associative cache we have a 3 node binary tree that tracks the pseudo LRU block. So this is the root and these are the 2 children after accessing a block the arrows point to the least recently used direction.

So when you have the very first block A then the cache block is empty or the cache set is empty, so you keep A in the way 0, this is your way 0 0th way and this is way 3. So to reach A I am going to access A, so from the root I want to go to the upper side since I am going to the upper side is the way how I access the recently used block A, so the arrow should point down so this indicates the lower side of these 2 are the least recently used.

And this indicate out of these 2 this is the least recently used, so after accessing A the LRU pseudo LRU tree looks like this.

(Refer Slide Time: 27:06)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

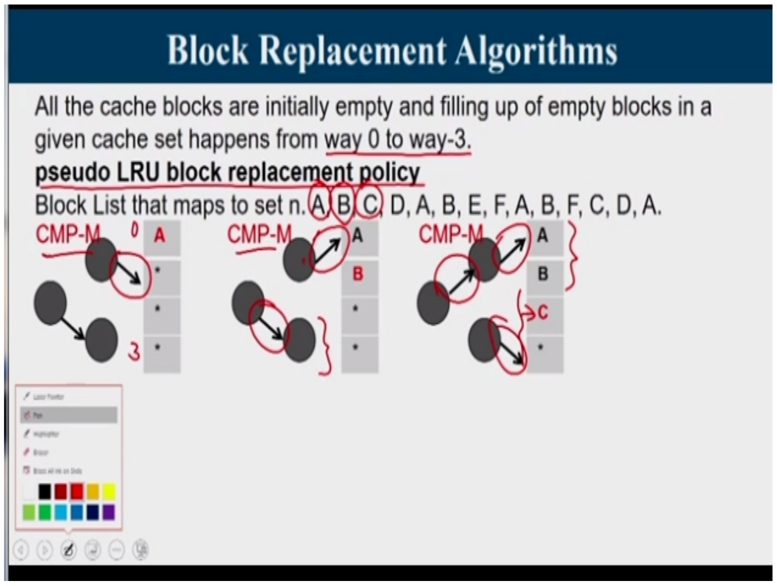
pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

And it is a compulsory miss because A is access for the very first time. Now let us see I am going to have the second request that is B. When I am going to access B since the cache set is empty I am going to fill up way number 1 this is way number 0. So when I go there from the root still this side is the least recently used, that means this arrow point to the least recently used portion of the set.

So these 2 lines are the least recently used when I come to this then initially the arrow was down. In this case the arrow is going up because when I access B then out of these 2 the upper 1 way 0 will be now the least recently used block.

(Refer Slide Time: 27:56)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

pseudo LRU block replacement policy

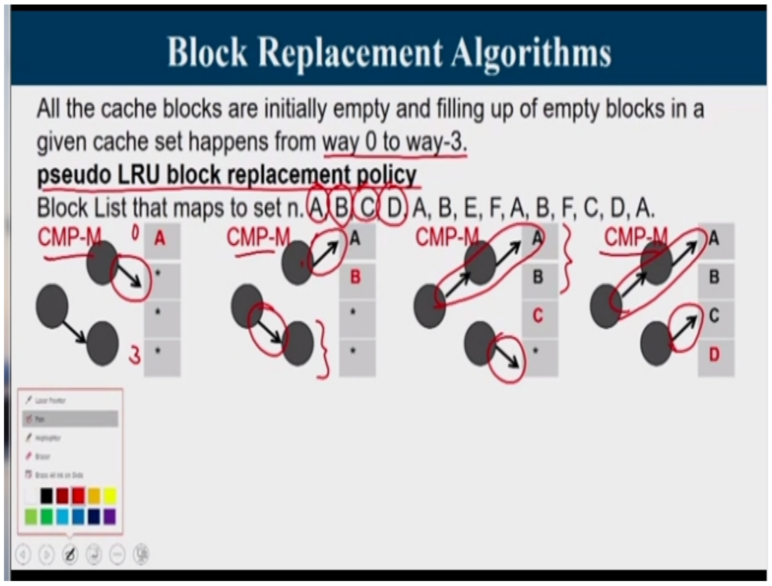
Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So after accessing B then the pseudo LRU tree looks like this, where the root is pointing down the upper child is pointing to A. Because after accessing B, A is the least recently used one, now going into C, so this is also a compulsory miss, so compulsory miss CMP stands for compulsory miss. When you go to C, C will be saved in way 2, so to access way 2 I have to go to the lower side this side.

That means the arrow which was initially pointing down now it has be flipped up because when you access C it is upper half of the tree that is the least recently used. So always this arrow is pointing to the least recently used one, since I am not touching A or B whatever was the arrow position for the upper sub tree it will remain same. Here there were no arrows because the lower portion was not access.

Now for accessing C you go to this particular block since I am going to that block this arrow should point to the other one, so this positions indicate which is the pseudo LRU position.

(Refer Slide Time: 29:08)


Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

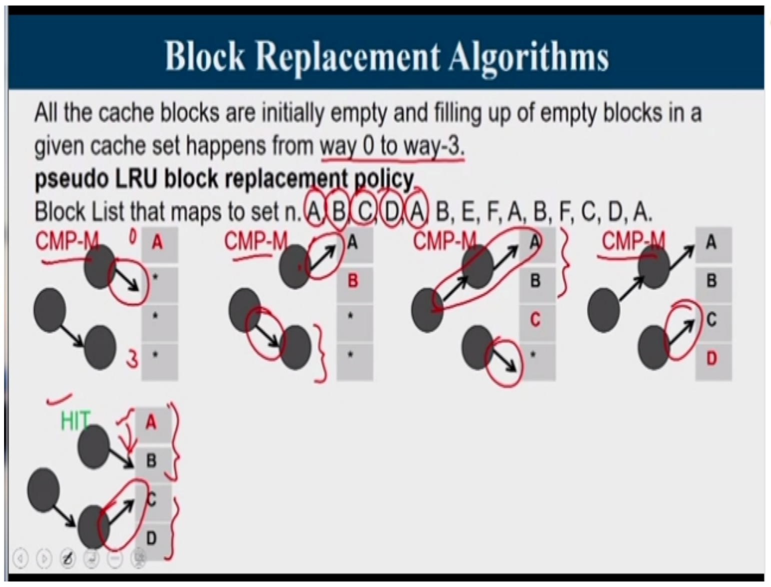
pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So to have a quick summary of what we have done till now the root tells that the upper half is the least recently used. The upper child tells that again the upper half is the least recently used, so if you look into this at this point this is the pseudo LRU block. Because root tell upper, upper child also tell upper, so A is the least recently used block as far as now is concerned. Let us continue further the next block what we have it is D.

So to bring D that is again one more compulsory miss because we have space and these access for the very first time. So A, B, C, D so way 3 is been filled once you filled up 3 this portion is untouched so it is same as what was there previously. And this was initially pointing to way 3 now it points to way 2, so after accessing D we have this as the position of the pseudo LRU tree.

(Refer Slide Time: 29:59)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

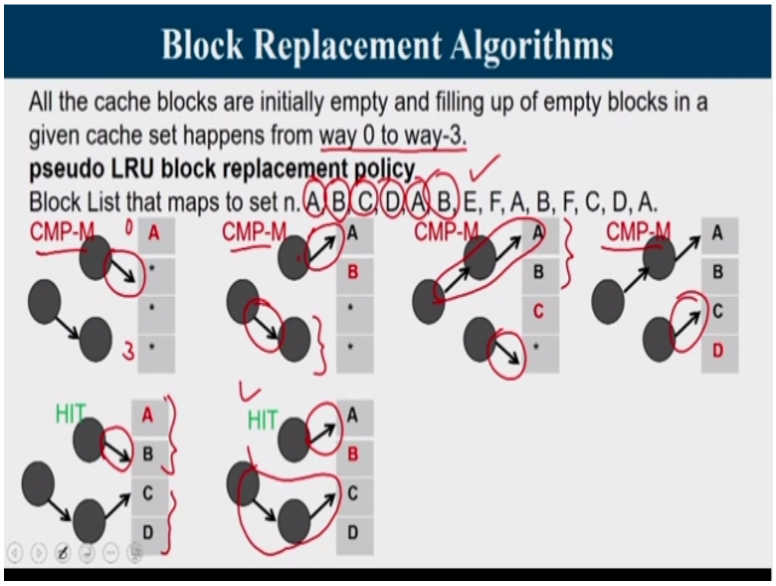
pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So compulsory miss stands for whenever a block is demanded for the very first time anyway it would not be available in the cache and that is known as a compulsory miss. Now continuing further. Now we have A that is coming, when you go with A you know that A is already present, so that is known as a hit. So when you access A still now the tree was like this A is in way 0, so the root will be now point into lower side.

Because current access was made to here the upper side, so the arrow will flip there, the arrow which was there for C and D there is no change in the arrow. Because we are not going to touch this portion since access was to A the arrow which was initially pointing to A now it will be pointing to B, so it is a hit.

(Refer Slide Time: 30:51)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

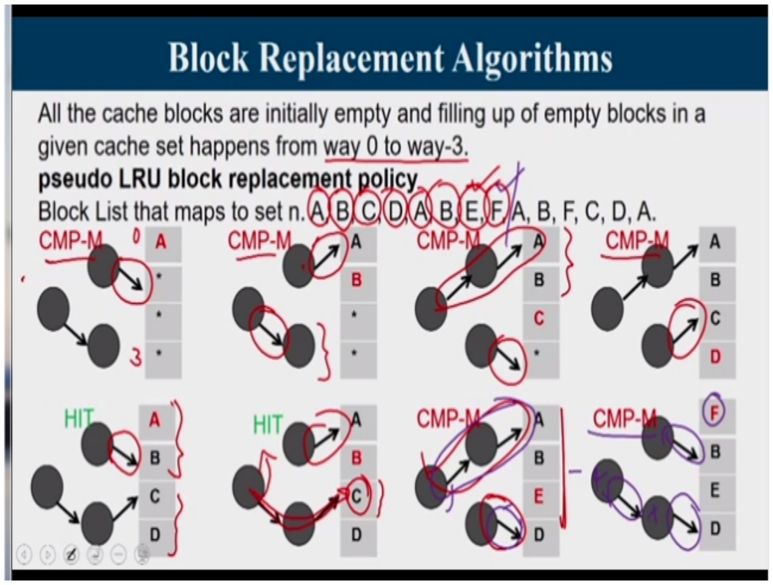
pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So after accessing A then the pseudo LRU tree wherein the root is having the LRU arrow towards down upper child arrow is also to down and the lower child arrow is to up. Now it is B, B is also a hit when you access B the only change that happen is an arrow that was pointing to B which indicates it was least recently used. Since B is used now, this arrow will get flipped and this portion there is no change at all, this is also another hit.

Now we have a new element E that is going to come, here is the first replacement, so far there was no replacement there are only compulsory misses. And now going to E it is a miss, now we have to find out which is the one that has to be replaced.

(Refer Slide Time: 31:40)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

So look into the status of it prior to bringing of the E the pseudo LRU tree shows like this if you travel through the arrows you will see this is the pseudo LRU block. Because it travel through the arrow and I will reach this particular line, so out of A, B, C, D at this particular junction C is the pseudo LRU it is not a least recently used it is a pseudo least recently used block. So the concept of pseudo LRU we have seen there during our lecture discussion.

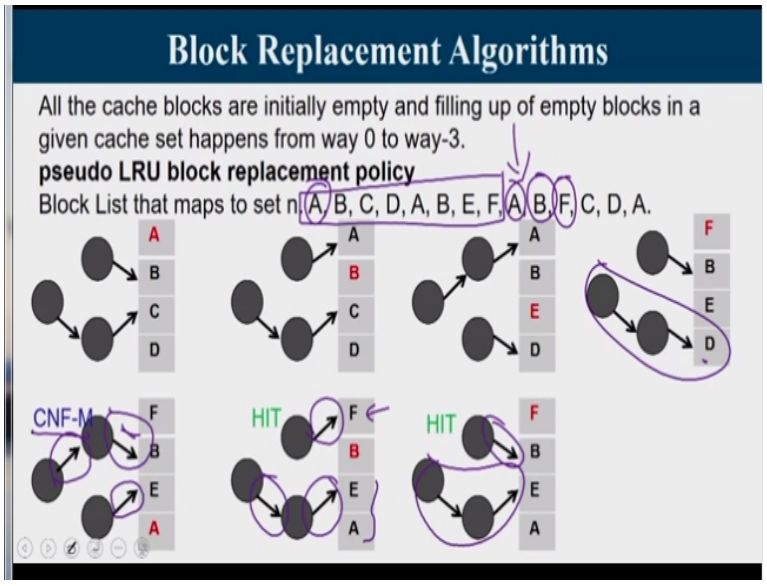
Continuing further when you are going to have E that is been brought, E is brought for the very first time and E is to be kept in a place C. So C is been taken out because the arrow point to C so whenever there is a new block that is coming the pseudo LRU block will be replaced. Since I access C this arrow should be flipped down and my access was to lower side of the tree then this will be the position of the arrows.

This remained unchanged and this arrow will get flipped upwards, that is what is happening here. Since E is also brought for the very first time that is a compulsory miss, then we have F you have to see that next element is F. And we see that a this particular juncture, after accessing the E this is the way how which the tree looks like. So this is the point of arrows, so A is a pseudo LRU block, so naturally when we are going to bring F since the cache is full it is a A that get replaced.

So again it is a compulsory miss this F is access for the very first time, A get replaced with F. So naturally this arrow get flipped to B and the root which was initially pointing upwards, now it is getting flipped down. So the root arrow will change and there is no change as for as this is concern wherever this was there it is unaffected. So when you bring F the arrows in the root get changed and the arrow in the bottom child also will get changed.

So whatever we have seen, so first 8 block request A, B, C, D, A, B, E, F that is been covered since I am moving into the very next slide whatever we completed till now this particular row will come up.

(Refer Slide Time: 33:55)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

pseudo LRU block replacement policy

Block List that maps to set n: A, B, C, D, A, B, E, F, A, B, F, C, D, A.

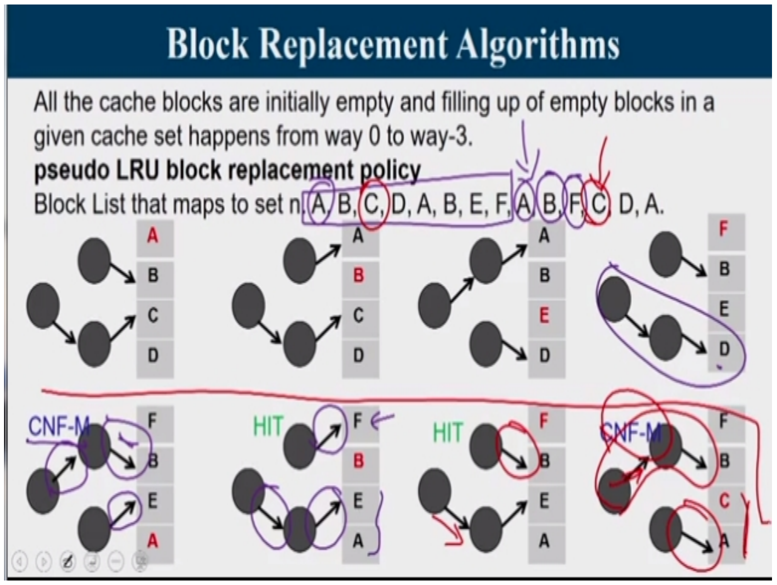
So this is what we have completed till now that is a position of the status at the rate of F, so this much is covered now. Now the next request is to A, we know that A is no longer present and A is a miss and pseudo LRU block at this point is D, so replacement of D happens. But A here has already demanded in the past and now I am demanding for A once again but it is not present A is been evicted out in this process.

So this is no longer a compulsory miss for A this is called a conflict miss, so that is called CNF conflict miss where we know that D get replaced with A. And since I am going to access the lower half the LRU is pointing upwards, this side no change, this arrow get flipped. Previously it was pointing to the last block, now it get pointing into way 2. So this is the position at which this is there, now we are going for B but is already there, so it will remain in a hit.

It is not only counting it as a hit but also the pseudo LRU arrow changes, so initially the arrow was pointing to B, now it get flipped upwards. Previously the arrow was pointing the root arrow was pointing upwards now that get also flipped. Because our access to B the resulted in making these portion least recently used, so B is a hit. Now we have F, F is also a hit, so once you have F it is a hit this portion remain unaffected, the arrow which was initially pointing to F, now that go point to B.

And then we have C, C is again a miss but we have to understand that at this point the pseudo LRU tree looks like this.

(Refer Slide Time: 35:33)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

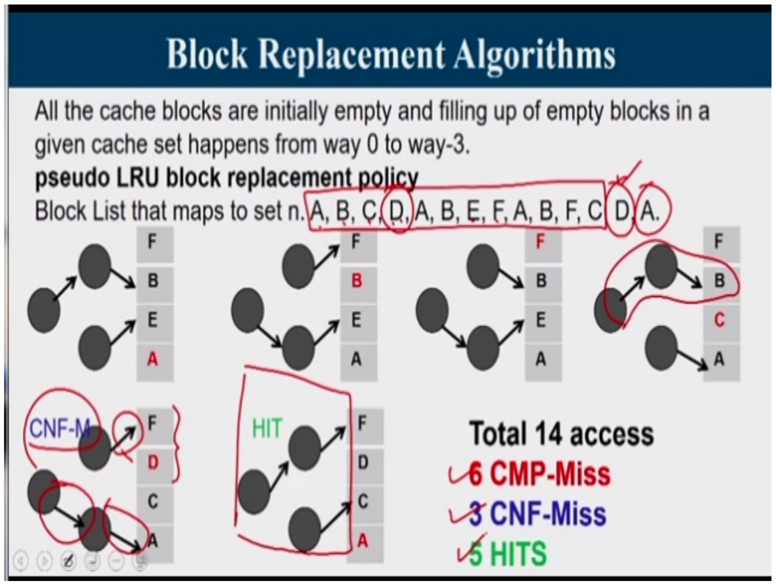
pseudo LRU block replacement policy

Block List that maps to set n: A, B, C, D, A, B, E, F, A, B, F, C, D, A.

Whenever there is a miss for C and C was accessed before but now C is not present, so that is a conflict miss, it is not a compulsory miss. Compulsory miss is whenever we encounter a miss for the very first time as for as the block is concerned. So in this case E will get replaced with C this is also a conflict miss and arrows are change accordingly because I am going to access the lower half this will not get changed the root which has initially pointing down, now the root is pointing upwards that is a change in the root.

And since I access C then this will pointing downwards, so always remember that the arrow is pointing to the least recently used portion. So we have completed up to C now there is only D, so whatever you see in this slide in the lower most portion this region that I again redrawn in the next slide.

(Refer Slide Time: 36:26)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

pseudo LRU block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

Total 14 access

6 CMP-Miss

3 CNF-Miss

5 HITS

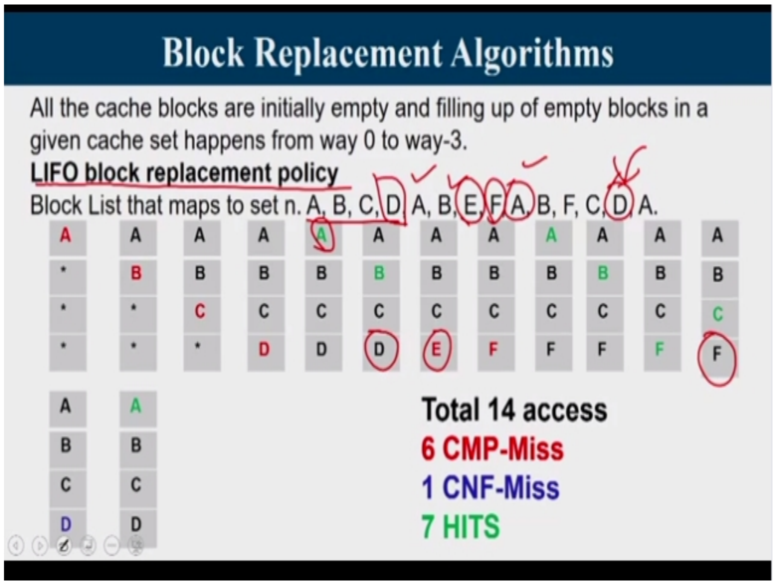
So this is the position up to C, now we have D that is coming in we know that D is also not present there but D was accessed before. So that time it was compulsory miss this time D it is a conflict miss and D is going to evict B because the pseudo LRU tree points to B as the pseudo LRU block. So B get replaced with D again 1 for conflict miss, so initial arrow that was pointing downward now it points upward.

Since the access was to upper side of the set, the root arrow is pointing downwards and there is no change in the arrow and the last one is A, A is already present there. So it is a hit but after accessing A the pseudo LRU tree looks like this. Now if you look at the statistics there are out 14 access 6 of them where compulsory misses, the very first access of A, B, C, D, E and F 3 of them are conflict miss and then we got 5 hits.

So now in this case what we have done is, we have given with a set of blocks for a given set and then we are processing them one by one. And every time when access is made appropriate change happen in the pseudo LRU tree, remember the arrow get flipped. Whenever there is an access to the upper side the arrow goes downwards whenever there is an access to downward side the arrow got flipped to upper side, that is a way how it is been manipulated.

So whenever there is a block replacement request that is coming that means a cache is full there is no place for the incoming one we have to evict out somebody that is a place where you find out the victim block. Victim block is find out by travelling through the arrows, so right from the root travel through the arrows that will be the pseudo LRU block.

(Refer Slide Time: 38:16)



Block Replacement Algorithms

All the cache blocks are initially empty and filling up of empty blocks in a given cache set happens from way 0 to way-3.

LIFO block replacement policy

Block List that maps to set n. A, B, C, D, A, B, E, F, A, B, F, C, D, A.

A	A	A	A	A	A	A	A	A	A	A	A
*	B	B	B	B	B	B	B	B	B	B	B
*	*	C	C	C	C	C	C	C	C	C	C
*	*	*	D	D	D	E	F	F	F	F	F

A	A
B	B
C	C
D	D

Total 14 access

6 CMP-Miss

1 CNF-Miss

7 HITS

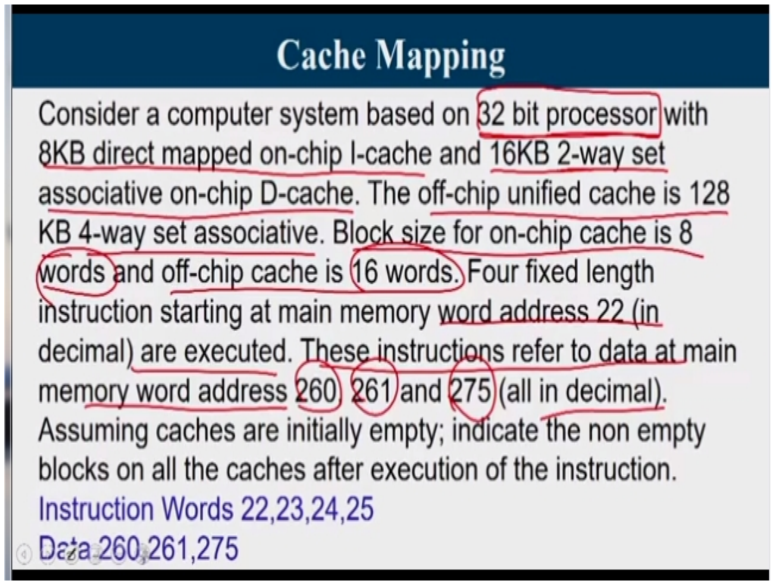
Now for the same question we had a second mode wherein these last in first out was the block replacement policy. So we will now try to see, so last in first out can be easily implemented using a stack. So consider this as stack let us say way 0 A is been brought then B is been brought, so it is a miss. Again compulsory miss then C is been brought and then D is been brought, so by this time now the particular set is full.

It has blocks A, B, C and D sitting in way 0, way1, way 2 and way 3, now A is again coming A is a hit, it is already present there. So A is a hit that is been shown by the green color no replacement needed then it is B, B is also present there. That is again a hit then it is E this is going to create a miss because E is not present there but it is a compulsory miss. But who will get evicted out it is a last in first out policy.

So D is the one was entering into the stack last, so D get replaced with E, so A, B, C, E that is a pattern in which you are going to get. And then we have F that is coming again F is not present but this E will get evicted out last in first out, so E will be replaced with F. again compulsory miss and then we have A that is a hit then we have B again it is a hit, then we have F again it is a hit and then we have C again it is a hit, and then we have D.

But D is no longer present here but D was initially brought at this point, so this is now a conflict miss it is not a compulsory miss. So to bring in D, F is going out last in first out, so D is going to come here this blue color indicate conflict miss and then the last one is A, A is a hit. So in this case we can see that out of total 14 access there were 6 compulsory misses 1 conflict miss and 7 hits.

(Refer Slide Time: 40:20)



Cache Mapping

Consider a computer system based on 32 bit processor with 8KB direct mapped on-chip I-cache and 16KB 2-way set associative on-chip D-cache. The off-chip unified cache is 128 KB 4-way set associative. Block size for on-chip cache is 8 words and off-chip cache is 16 words. Four fixed length instruction starting at main memory word address 22 (in decimal) are executed. These instructions refer to data at main memory word address 260, 261 and 275 (all in decimal). Assuming caches are initially empty; indicate the non empty blocks on all the caches after execution of the instruction.

Instruction Words 22,23,24,25

Data 260,261,275

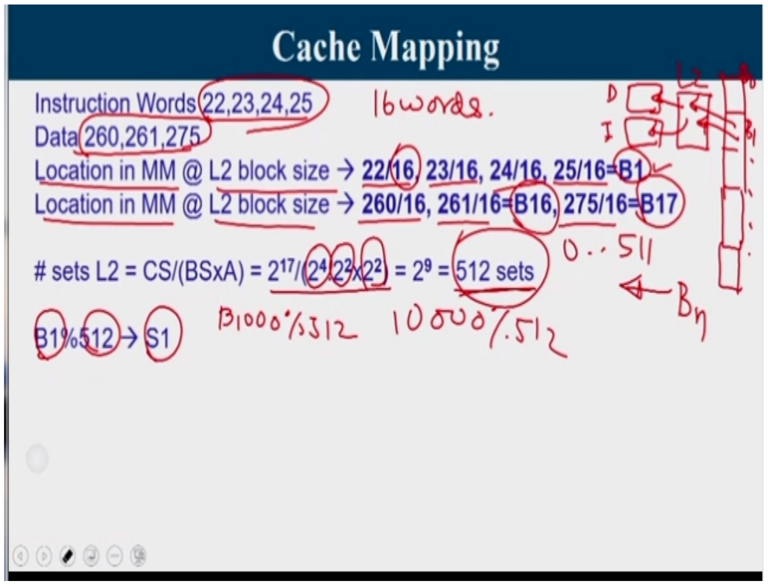
We move onto the next question, consider a computer system based 32 bit processor, so this 32 bit processor is working on an 8KB direct mapped on-chip I cache 16KB 2 way set associative on-chip D cache. The off-chip unified cache is 128KB 4-way set associative, block size for on-chip cache is 8 words and for off-chip cache is 16 words. So we have been given the configuration of the I cache L1 I cache and L1 D cache.

And the L2 cache the unified cache been given and L1 cache has a block size of 8 words and L2 cache has a block size of 16 words, I cache is direct mapped whereas D cache is 2 way associative map. This is the 32 bit processor, so the meaning is the word length is 4 byte or 32 bit, 4 fixed length instructions starting at main memory word address 22 in decimal are executed. So we have 4 fixed length instruction with word address 22, 23, 24 and 25 they are executed and these 4 instructions refer to data at main memory word address 260, 261 and 275 all in decimal.

Assuming caches are initially empty, indicate the non empty blocks on all the caches after execution of the instruction. So here in the given question we have been given with a processor with 2 levels of cache the L1 cache and the L2 cache. And a main memory is been given, 4 instructions are been executed so this instructions will be coming from main memory to the L2 the unified cache and then to the L1 I cache 22, 23, 24 and 25 are the word address in the main memory, they will come to some particular set in the L1 cache.

Similarly upon executing these 4 instructions words 261 and 275 are accessed. Now the question is initially the cache is empty upon execution there will be certain blocks that will be coming in certain sets get filled up. We are suppose to find out the non empty sets after execution of this, so this question gives you a clear understanding of whether you are understanding of mapping is correct, so instruction words 22, 23, 24 and 25 and data words 260, 261 and 275.

(Refer Slide Time: 42:54)



Cache Mapping

Instruction Words 22,23,24,25   16 words.

Data 260,261,275

Location in MM @ L2 block size → **22/16, 23/16, 24/16, 25/16=B1**

Location in MM @ L2 block size → **260/16, 261/16=B16, 275/16=B17**

sets L2 = CS/(BSxA) = $2^{17}/(2^4 \cdot 2^2 \times 2^2) = 2^9 =$ 512 sets   0 .. 511

B1%512 → S1   B1000%512   10000%512

So these are the ones that is been given 22, 23, 24 and 25 and data 260, 261, 275, so location inside your memory at L2 block size. So from the main memory, so this is the main memory from where you are brining to L2 and then we have the D cache as well as the I cache. So these 4 words it has to start from main memory it has to reach L2 and then it has to come to I cache whereas these 3 will come from main memory to L2 and then it goes to D cache.

So this main memory when you transfer something from main memory to L2 it is a block of data that get transferred and it is equal to the block size of L2. So L2 is divided with block size of 16 words, so if the word number

22 /16, 23 /16, 24 /16, 25 /16

these are all words which are part of block 1 in main memory. So your main memory has block 0, block 1 like that, so to find out word number n it is part of which block we have to divide it with the number of blocks in the word.

So when I divide with 16 I get it as B1, similarly location in main memory for L2 block size 260, 261 and 275, 260 and 261 will be part of B16 block 16 whereas 275 is part of block 17. So when you find out a number of sets in L2 it is a cache size divided by block size into associativity, L2 is 128 bytes or 2 power 17 divided by 16 words, so that is 2 power 4. And each word is 4 byte, so block size is 2 power 4 into 2 power 2 and it is a 4 way associative that is associativity, so I have 512 sets.

$$\text{No of sets (L2)} = 2^{17} / (2^4 * 2^2 * 2^2) = 512 \text{ sets}$$

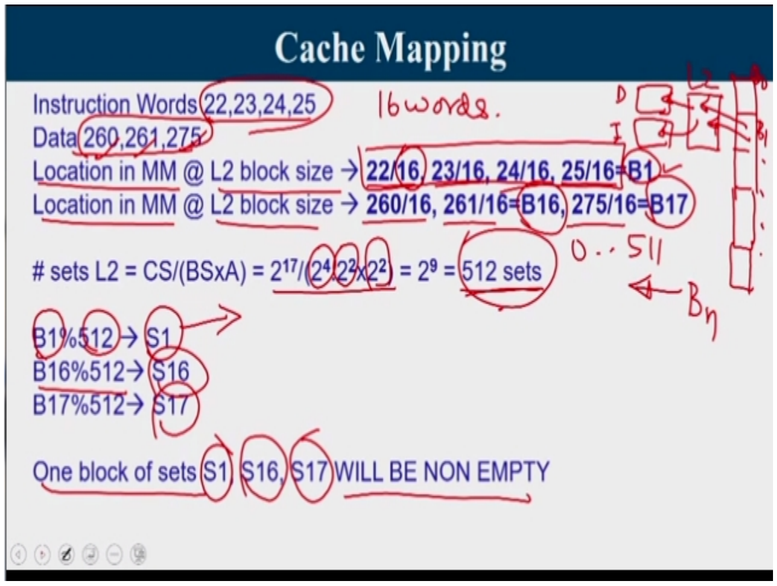
So main memory has block B0, B1, B2 divided based upon the block size of L2 it can go up to let us 1000s or lakhs of main memory blocks. But in my L2 I have only 512 sets, so whatever be the block number B0 up to Bn whatever be the block number in main memory that has to map to up to 0 to 511 because I have only 512 sets in my L2 cache. So my B1, so how will I know it is a modulus operation, B1 when you perform the mode operation on 512 it get mapped to set 1 in L2. If it the value was B1000 then we have to perform

1000 mod 512,

if it is the 10000th block then

10000 mod 512

(Refer Slide Time: 45:45)



Cache Mapping

Instruction Words 22,23,24,25

Data 260,261,275

Location in MM @ L2 block size → 22/16, 23/16, 24/16, 25/16=B1

Location in MM @ L2 block size → 260/16, 261/16=B16, 275/16=B17

sets L2 = CS/(BSxA) = $2^{17}/(2^4 2^2 \times 2^2) = 2^9 =$ 512 sets

B1%512→ S1

B16%512→ S16

B17%512→ S17

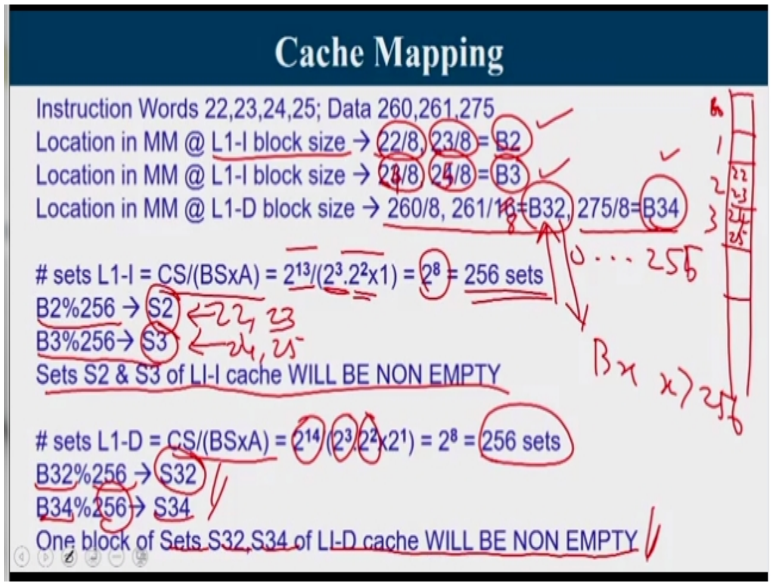
One block of sets S1, S16, S17 WILL BE NON EMPTY

16 words.
0 .. 511
L2
D
I
Bn

That is a process by which you find out given block number inside main memory, you wanted to find out where it gets mapped in cache. So B1 mod 512 is going to S1, similarly so when I bring B1 they all are part of the same blocks, so all these 4 words will come together and they get reside in set number 1 one of the block of set number 1. Because L2 cache is 4 way associative, so one of the block of set number 1 is going to be accessed by this block.

And B16 mod 512 you get set 16 and B17 mod 512 we get set 17. So 1 block of set number S1 we carry the instruction, 1 block of set number 16 will carry the words 260 and 261 and 1 block of set number 17 will carry block number 275. So these sets will be set 1, set 16 and set 17 will be non empty in L2.

(Refer Slide Time: 46:45)



Cache Mapping

Instruction Words 22,23,24,25; Data 260,261,275
Location in MM @ L1-I block size → 22/8, 23/8 = B2
Location in MM @ L1-I block size → 24/8, 25/8 = B3
Location in MM @ L1-D block size → 260/8, 261/8 = B32, 275/8 = B34

sets L1-I = CS/(BSxA) = $2^{13}/(2^3.2^2x1) = 2^8$ = 256 sets
B2%256 → S2
B3%256 → S3
Sets S2 & S3 of LI-I cache WILL BE NON EMPTY

sets L1-D = CS/(BSxA) = $2^{14}/(2^3.2^2x2^1) = 2^8$ = 256 sets
B32%256 → S32
B34%256 → S34
One block of Sets S32,S34 of LI-D cache WILL BE NON EMPTY

Now we wanted to bring from L2 all the way to L1 but we have to know that the L1 block size is only 8 words. So we wanted to know where this 22 divided by 8 and 23 divided by 8 will give you B2, they belong to block 2. So this is your main memory, so word number 22 and 23 belong to this is block 0 then 1, 2. So 22 and 23 they were part of block 2 of main memory it is a logically block 2 of main memory if you divide main memory as for L1 block size.

You may wonder that your transferring data from L2 to L1 but this is an abstraction of trying to understand how this mapping happens. Whereas word 23 and 24 is mapped to so this 22 and 23 this is the 24 and 25, this is 24 and this is 25. So 24 and 25 will be mapped to block 3, similarly 260 and 261 will be blocked to block 32 because you 260 divided by 8

$$260/8$$

and 261 also I am dividing with 8 you get 32

$$261/8= 32$$

and 275 I divide with 8 you get 34.

$$275/8=34$$

So number of sets in L1 I cache, cache size divided by block size into associativity, so L1 cache is 8KB, so 8KB 2 power 13 by block size is 8 words 2 power 3 into each word is 4 byte. So 2 power 3 into 2 power 2 that much is the number of byte because the both the denominator and a numerator should be in same unit. So my numerator was in bytes 2 power 13 bytes, so denominator also should be in bytes it should not be in words, that gives you 2 power 8, so my L1 cache has 256 sets.

$$2^{13}/ (2^3 * 2^2 * 1)= 2^8 = 256 \text{ bytes}$$

So whatever be the block number at the end when it been mapping it should mapped get to 0 all the way up to 255. So B2 when I perform a mod operation B2 get mapped to set 2 of L1 I cache and B3 get mapped to set 3 of L1 I cache. So the word number 22 and 23 will reside in set 2 of L1 I cache and word number 24 and 25 will reside inside set 3 of L1 I cache. So set S2 and S3 of L1 I cache will be non empty.

Similarly we find out the number of sets in L1 D cache, cache size divided by block size into associativity here cache size is 16KB, so it is 2 power 14, block size is 8 words. So 8 words into number of bytes in a word 4 byte in it is 2 way associative, so here also you have 256 sets,

$$2^{14}/ (2^3 * 2^2 * 1)= 256 \text{ sets}$$

so block 32 which we have already identified it has block 32, block 32 mod 256 it get mapped to set 32, block 34 mod 256 it can maps to set 34.

You may have find that this block number is same as a set number because all the block numbers is lower than 36. If the block number value let us say Bx where x is larger than 256 let us say block 300, so it should be

$$300 \bmod 256$$

that is a number that you get. So 1 block, so it is a 2 way associative cache, so in set number 32 and set number 34 out of the 2 block 1 block will be non empty, L1 D cache will be non empty in that scenario as well.

So this particular question what we have done is we have considered the specifications of I and D cache and we have mentioning what are the words where processor is trying to access. And based upon the mapping from the main memory all the way to cache we are able to find out which are the sets that are getting filled up which are the non empty sets, with this we come to the end of this tutorial. With this tutorial you might have got a fair understanding of the mapping and the replacement techniques that is been used in cache.

In the Henessy and Patterson computer architecture textbook there are plenty of problems that are given at the end of a memory chapter. So solving these exercises will give you more grip on understanding this topic, I hope the tutorial session was useful, kindly try to solve out as much of problems as possible and get back to us if at all you have any query, thank you.