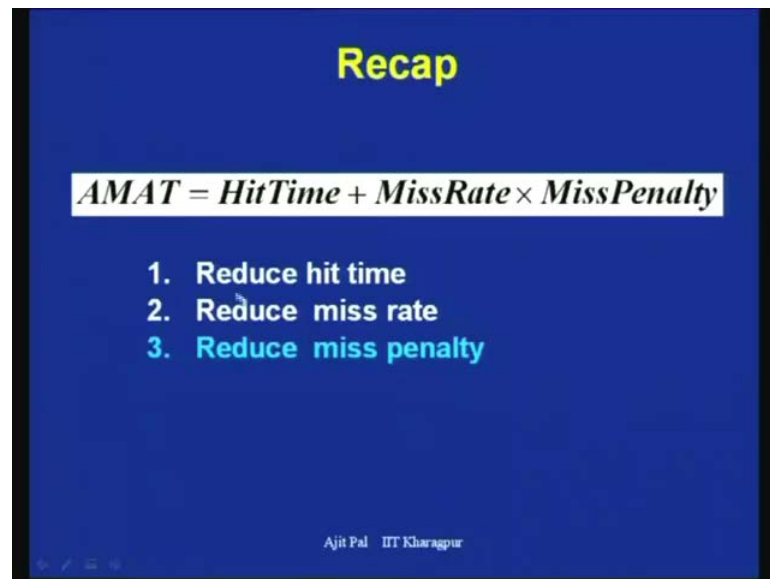


**High Performance Computer Architecture**  
**Prof. Ajit Pal**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 26**  
**Cache Optimization Techniques (Contd.)**

(Refer Slide Time: 00:59)



**Recap**

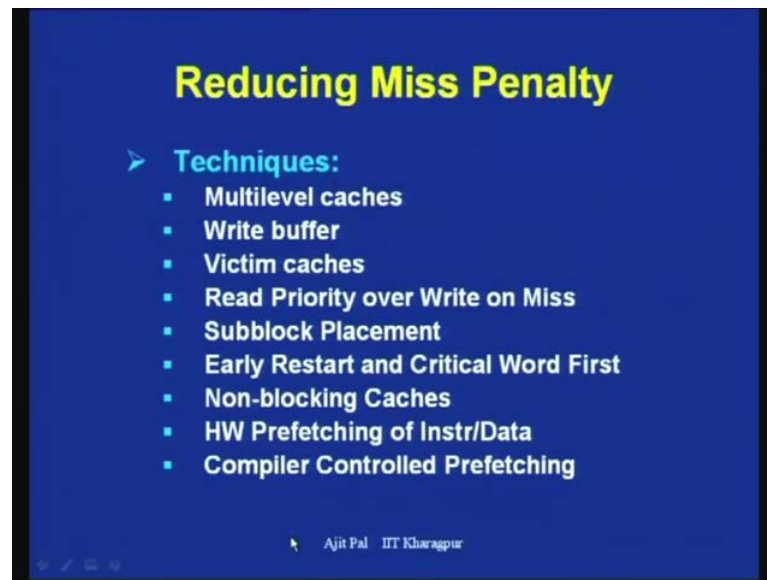
$$AMAT = HitTime + MissRate \times MissPenalty$$

1. Reduce hit time
2. Reduce miss rate
3. Reduce miss penalty

Ajit Pal IIT Kharagpur

Hello viewers welcome to today's lecture on cache optimization techniques, in the last lecture. In the last two lectures, we have discussed about how you can reduce hit time and how you can reduce miss rate. And today we shall focus on another very important parameter that is miss penalty. So, we shall discuss about various techniques for reducing miss penalty there exist a large number of miss penalty actually.

(Refer Slide Time: 01:24)



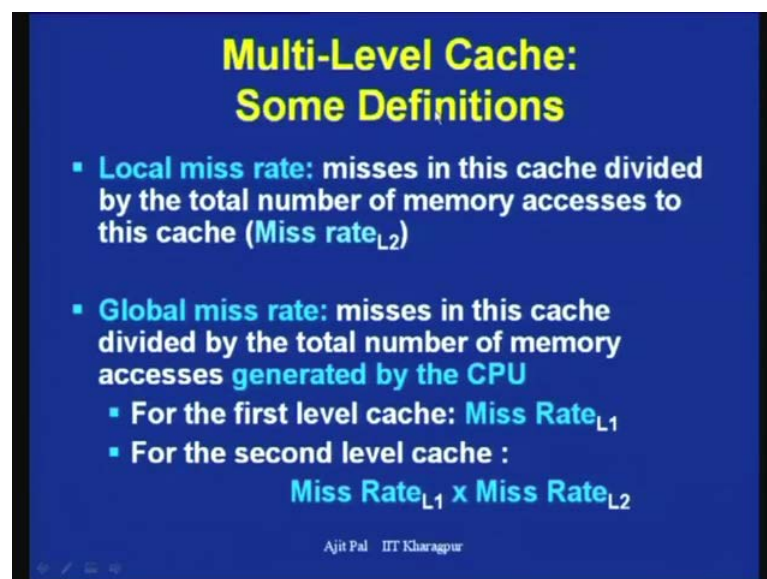
## Reducing Miss Penalty

- **Techniques:**
  - Multilevel caches
  - Write buffer
  - Victim caches
  - Read Priority over Write on Miss
  - Subblock Placement
  - Early Restart and Critical Word First
  - Non-blocking Caches
  - HW Prefetching of Instr/Data
  - Compiler Controlled Prefetching

Ajit Pal IIT Kharagpur

This is one of the thoroughly researched area and there are many techniques I mean proposed by researchers. So, I shall focus on a sub set of them and try to give an over view of the various techniques. So, these are the various techniques multi level cache write buffer victim cache read propriety over write on miss sub block placement early. Start and critical word first non blocking cache hardware perfecting of instruction and data complier controlled perfecting. So, these are the miss I shall briefly consider in today's lecture.

(Refer Slide Time: 02:11)



## Multi-Level Cache: Some Definitions

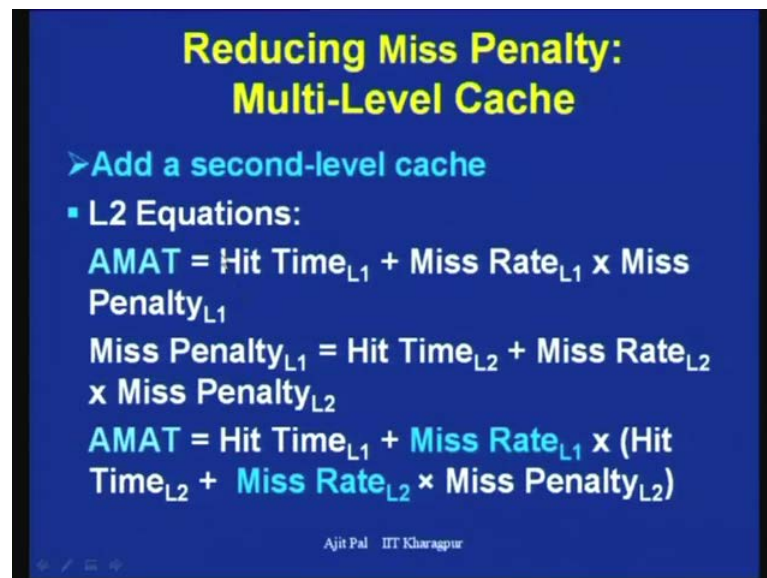
- **Local miss rate:** misses in this cache divided by the total number of memory accesses to this cache ( $\text{Miss rate}_{L_2}$ )
- **Global miss rate:** misses in this cache divided by the total number of memory accesses generated by the CPU
  - For the first level cache:  $\text{Miss Rate}_{L_1}$
  - For the second level cache :  
 $\text{Miss Rate}_{L_1} \times \text{Miss Rate}_{L_2}$

Ajit Pal IIT Kharagpur

First let us focus on multi level cache, so whenever we go for multi level cache we have to use modifier or definition about miss rate. Actually we have to introduce two terms one is known as local miss rate another is known as global miss rate. So, local miss rate actually represent the misses in this cache divided by the total number of memory accesses to this cache. So, let me briefly explain it you see your are first the processor to access the one cache and if it is not present in the L1 cache then it tries to get instruction or data from the L2 cache obviously the number of access from the L 2cache is smaller.

And I mean and miss rate will correspond to the total number of memory access to this cache. That means the number of access that has been tailed on L2 cache that will be used for the calculation of miss rate. And total number of misses by the total number of accesses to this cache on the other hand global miss rate is misses, in this cache divided by the total number of memory accesses generated by the CPU. So, global miss rate obviously will be smaller for L2 cases and for the first level cache it is miss rate L 1. And for the second level cache is a miss rate L1 into miss rate L2 that is global miss rate for L2 cache will be equal to miss rate L1 into miss rate L 2.

(Refer Slide Time: 04:03)



**Reducing Miss Penalty:  
Multi-Level Cache**

- Add a second-level cache
- L2 Equations:
 
$$AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

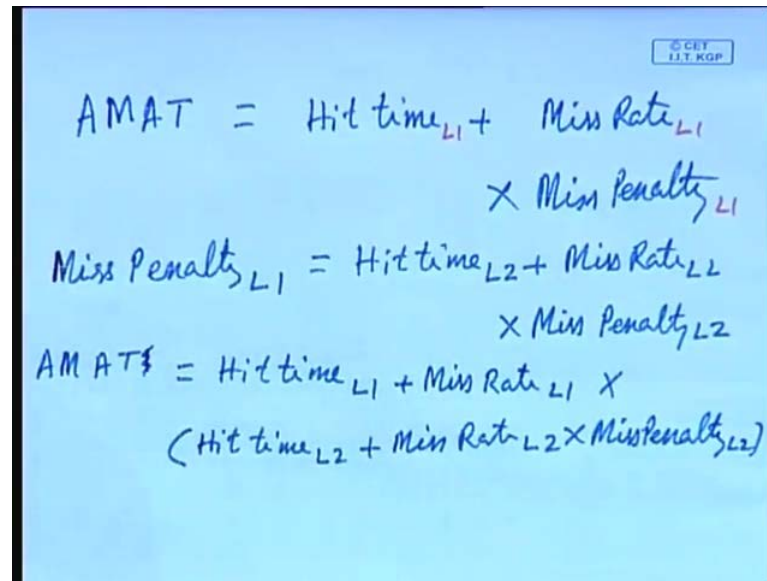
$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

Ajit Pal IIT Kharagpur

And whenever we add a second level cache, let us see how the equation for average memory access time changes.

(Refer Slide Time: 04:14)



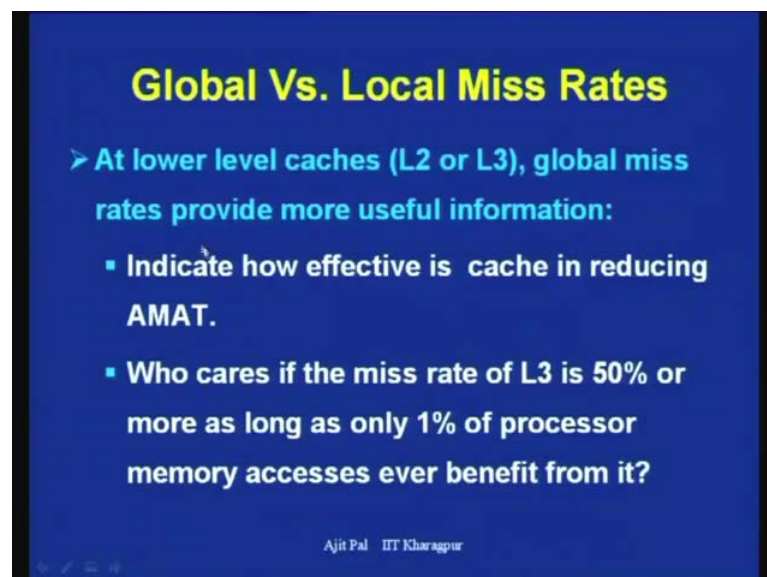
Handwritten equations on a blue background:

$$AMAT = Hit\ time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$$
$$Miss\ Penalty_{L1} = Hit\ time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$$
$$AMAT = Hit\ time_{L1} + Miss\ Rate_{L1} \times (Hit\ time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$$

Small logo in the top right corner: CCEP I.I.T. KGP

So, average memory access time AMAT has a no is equal to hit time plus miss rate into miss penalty, now if this correspond to L cache we give a suffix L1 L 1 and L 1. Now, in this case miss penalty miss penalty of L1 this miss penalty of L1 is, now equal to hit time L2 plus miss rate miss rate L2 into miss penalty L 2, so we find here miss penalty of L 1. If we can substitute here we shall get AMAT for a two level cache will be equal hit time L1 plus miss rate L1 into you have to multiply with hit time L2 plus miss rate L2 into miss penalty L 2. So, this is how the AMAT is gets modified whenever you are having a two level cache second level cache.

(Refer Slide Time: 06:10)



**Global Vs. Local Miss Rates**

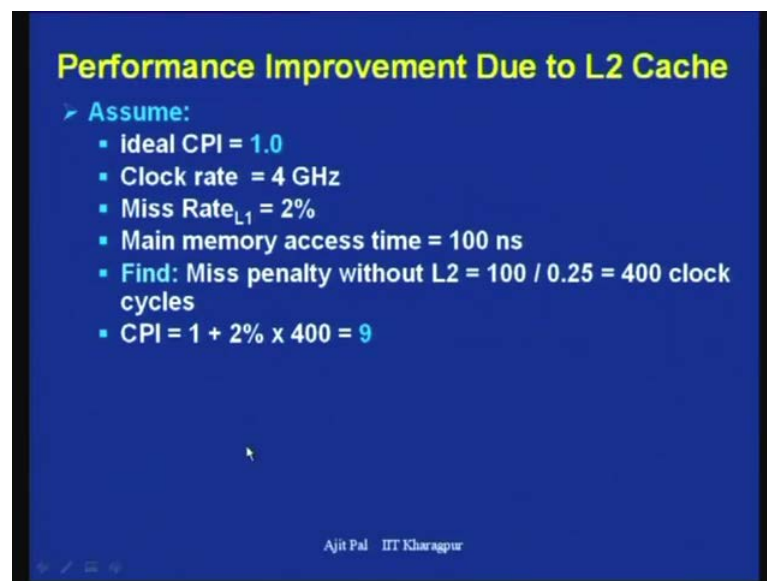
- At lower level caches (L2 or L3), global miss rates provide more useful information:
  - Indicate how effective is cache in reducing AMAT.
  - Who cares if the miss rate of L3 is 50% or more as long as only 1% of processor memory accesses ever benefit from it?

Ajit Pal IIT Kharagpur

And at lower level caches L 2 or L 3 global miss rates provide more useful information because that on the source. How many accesses are taking place to that as compare to total number of access. So, indicate how effective is cache in reducing AMAT that means how by adding to an L3 case cache memory access to those L2 and L3 are getting reduced, so this will get back.

So, question naturally arises what is the miss rate for L3 actually there is a possibility that miss rate for L2 or L3 can be higher than the miss rate of L1. So, who cares if the miss rate of L3 is 50 percent it may be as high as 50 percent or more as long as it is 1 percent of processor memory accesses ever benefit from it. That means if we get an overall improvement of performance a with large miss rate of L2, L3 access it does not really matter or overall performance or AMAT is important parameter.

(Refer Slide Time: 07:22)



**Performance Improvement Due to L2 Cache**

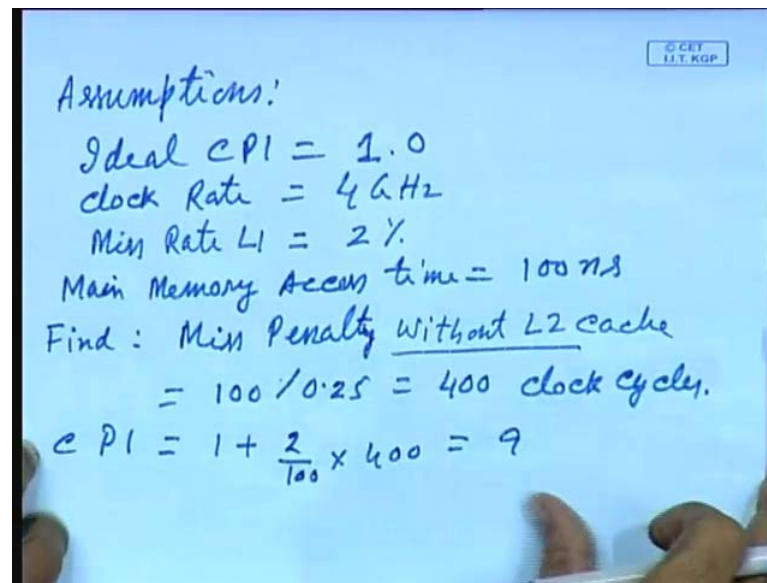
➤ Assume:

- Ideal CPI = 1.0
- Clock rate = 4 GHz
- Miss Rate<sub>L1</sub> = 2%
- Main memory access time = 100 ns
- Find: Miss penalty without L2 =  $100 / 0.25 = 400$  clock cycles
- CPI =  $1 + 2\% \times 400 = 9$

Ajit Pal IIT Kharagpur

So, let us consider let me illustrate with the help of an example. Let us assume that ideal CPI is equal to 1, so these are the assumptions.

(Refer Slide Time: 07:39)



Assumptions:

- Ideal CPI = 1.0
- clock Rate = 4 GHz
- Miss Rate L1 = 2%
- Main Memory Access time = 100 ns

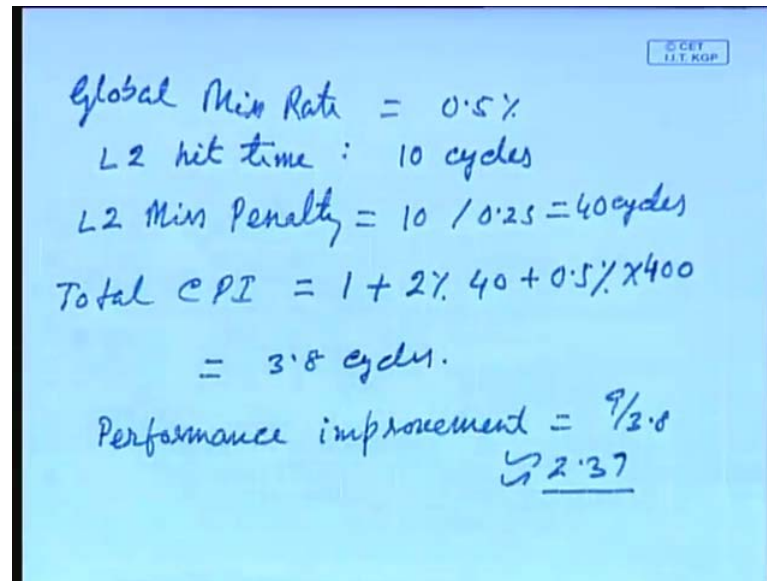
Find: Miss Penalty without L2 cache

$$= 100 / 0.25 = 400 \text{ clock cycles.}$$
$$CPI = 1 + \frac{2}{100} \times 400 = 9$$

So, assumptions are number one ideal CPI is equal to 1. That means one cycle 1.0 cycles per 1, that is that happens whenever at a get a cache hit in one cache and clock rate has been assumed to be is equal to 4 Giga hertz. And miss rate L1 is equal to let us assume its 2 percent and then main memory access time. Let assume to be is equal to 100 nano second, so whenever we have 100 nano second what is the find out miss penalty without L2.

So, miss penalty without L2 will be equal to 100 that is 100 nano second divided by 25 neon second that is this 4 Giga hertz is a clock rate. So, this is 0.25 second this will give you 400 clock cycles and obviously your CPI will be equal to 1 plus since the miss rate is 2 percent 2 by 100 into 400. So, this is the overall CPI without L2 cache, so this is equal to 9 cycles per instruction that is 9. Now, let us consider the situation when we add when we go for a second level cache.

(Refer Slide Time: 09:47)



Global Miss Rate = 0.5%  
L2 hit time : 10 cycles  
L2 Miss Penalty =  $10 / 0.25 = 40 \text{ cycles}$   
Total CPI =  $1 + 2\% \cdot 40 + 0.5\% \cdot 400$   
 $= 3.8 \text{ cycles.}$   
Performance improvement =  $9 / 3.8$   
 $\rightarrow \underline{2.37}$

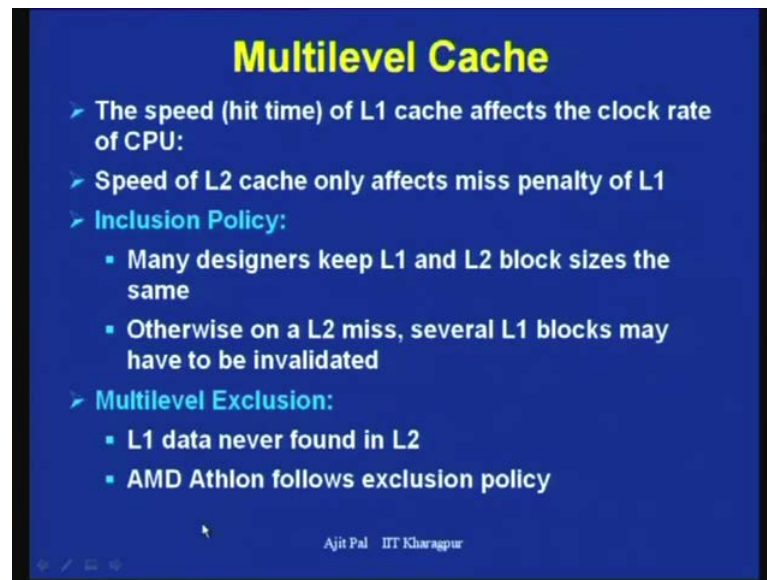
Let us assume global miss rate for L2 cache is equal to 0.5 percent it is quite small. So, L2 hit time obviously will be much smaller than main memory access time. So, that is considered to be 10 cycles compared to 100 cycles of main memory, so in this case getting from L2 cache and you are taking lesser time.

So, L2 miss penalty is equal to 10 divided by 0.25 that is 40 clock cycles. That is whenever the data is available in L2 cache you will require 40 clock cycles. So, total CPI cycles per instruction at such a situation is equal to 1 plus 2 percent into 40. So, this is the 40 and plus 0.5 percent into 100 400 that is the 400 was the clock cycle that you got. So, 400 now what happens your CPI this becomes equal to 3.8, so earlier we have seen the CPI was 9, now we have got 3.8.

So, the performance improvement is equal to 9 by 3.8 this is roughly equal to 2.37. So, this particular simple problem illustrates, how the performance is improving by using L2 cache. So, if you are having only L1 cache your CPI is 9 and when you have got L1 and L2 cache. Then your performance is improving by 2.37 times. So, CPI is reducing to 3.8 cycles, so with this benefit we may say that the speed of L1 cache affects the clock rate of CPU.



(Refer Slide Time: 12:17)



### Multilevel Cache

- The speed (hit time) of L1 cache affects the clock rate of CPU:
- Speed of L2 cache only affects miss penalty of L1
- Inclusion Policy:
  - Many designers keep L1 and L2 block sizes the same
  - Otherwise on a L2 miss, several L1 blocks may have to be invalidated
- Multilevel Exclusion:
  - L1 data never found in L2
  - AMD Athlon follows exclusion policy

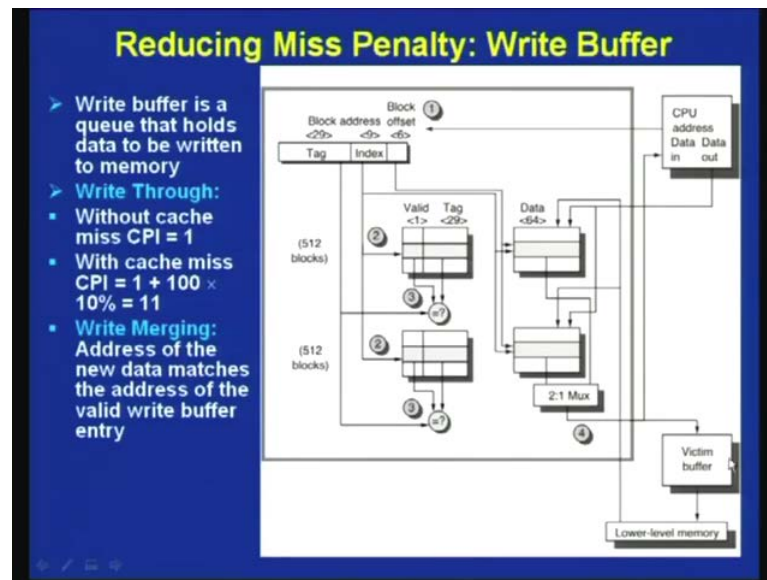
Ajit Pal IIT Kharagpur

As we know the CPU is directly communicating or interfacing with L1 cache. So, most of the communication between processor and memory that takes place with L1 cache and that is the reason why hit time of L1 cache affects the clock rate of CPU. So, clock rate is what primarily decided by the hit time of L1 on the other hand speed of L2 cache is only affects miss penalty of L2 as you have seen by this calculation. So, L2 cache will reduce the miss penalty and you can have two different policies one is known as inclusion policy which I have already mentioned in the beginning. So, many designers keep L1 and L2 block sizes the same, so that means the L1 and L2 you can have the same block sizes.

The number of words number of words that is present in the block is identical in L1 and L2 and if that is not true that means otherwise on a keep L1 and L2 block sizes not same. Then there will be several L1 blocks may have to be invalidated whenever there is L2 miss. So, if there is L2 miss be several L1 blocks may have to be invalidated if inclusion property is follow another one alternative is multi level exclusion in this case L1 data is never found in L2. So, normally L1 includes a L2 includes L1, but that is not deserved. So, that exclusion property is violated in some satiations, so that means case L1 data is never found in L2. That means commercial problems use this for example, AMD athlon follows exclusion policy this exclusion policy.



(Refer Slide Time: 14:25)

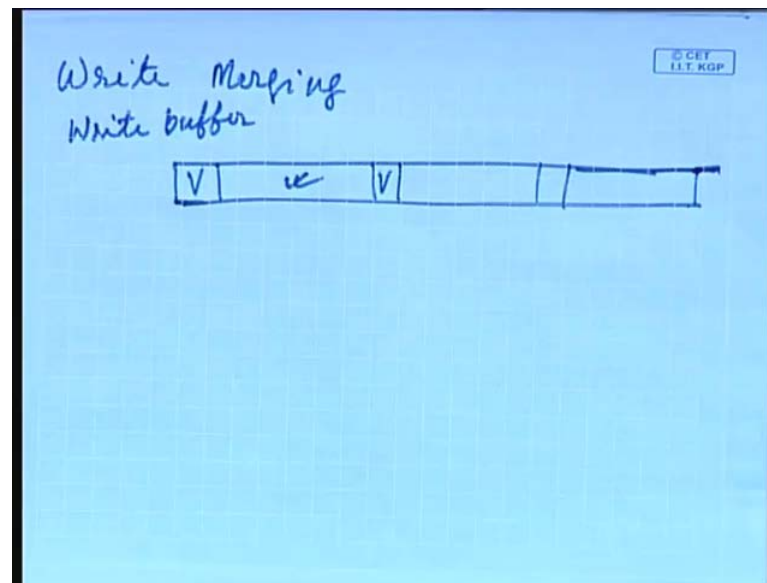


So, that inclusion property is not used in memory hierarchy. Now, let us consider another technique that is known as use of right buffer. So, right buffer is a queue that holds data to be written to memory as we know whenever you are writing data into the memory. Then if it is not in the cache or even if it is the cache you have to write it into the cache you have to also write it in the main memory it is quite inclusion property. So, and to afford that can be done we can add a victim buffer what is the role of the victim buffer.

Let us consider a situation through write through policy, so in case of write through the data is written into the cache memory I mean without cache miss. Since, it is available in the cache it is written into it and obviously we will take 1 clock cycle and if we use write through policy. And without using this victim buffer the number of I mean CPI will be equal to 1 plus 100 into 10 percent assuming that 10 percent is the miss rate of L1 cache.

So, in such a case you will require CPI of 11 cycles, so this will not be required if we use this victim buffer. So, we can see here the data is written in cache memory which are present here and then it is also written in victim buffer. And in the offline data writing takes place in the lower level memory. So, CPU is not concerned with that we know p q had already got data I mean CPU has got it from the, I mean that from the L1 cache. So, it is written into it data transferred and writing takes place in the cache. But, it is also written in victim buffer, but it is done in the offline. Another approach can be used along with this write buffer which is known as write merging.

(Refer Slide Time: 16:50)



So, address of the new data machines the address of the valid write buffer entry here what is being done, this write merging suppose a particular entry of a block entry of a block of a block is modified. So, this is being modified and this is being modified, so this is this is available in right buffer you may have multiple, multiple words in a single block, so this data has been modified. Now, there is a possibility that another data another data is also you also writing in another data. So, the address of new data matches to a address of new valid write buffer entry. So, in the right buffer its checks that it is present in right buffer then what is being done it is you are not writing into the buffer you are modifying here. So, for that you will require additional valid bit present in the present in the right valid merging technique.

(Refer Slide Time: 18:22)

[illegible]

And as a consequence you will have a valid bit here valid bit for here, but it definitely save the number of tags you will have a common tag. That is the reason why same address you are getting the, you are getting in the right buffer. So, right merging is the extension of this right buffer approach where you check the data in the victim buffer first before you try read the main memory. And write back policy is much more complex to implement whenever we use right buffer, so we shall not discuss in this lecture.

(Refer Slide Time: 18:54)

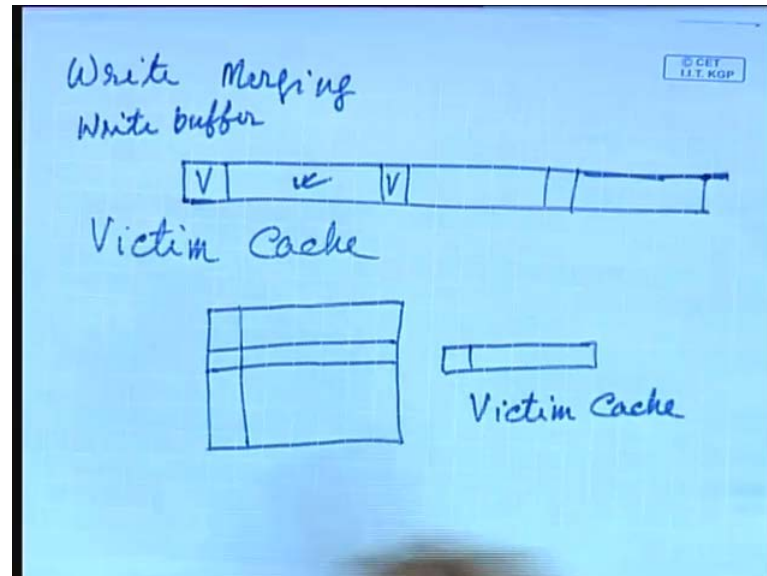
## Reducing Miss Penalty: Victim Cache

- How to combine fast hit time of direct mapped cache:
  - Yet still avoid conflict misses?
- Add a fully associative buffer (victim cache) to keep recently thrown out discarded data from cache.

Ajit Pal IIT Kharagpur

Now, we shall focus on another technique which is known as victim cache. So, reducing miss penalty by victim cache what is victim cache.

(Refer Slide Time: 19:07)



So, you have got your cache memory here now in addition to this cache memory you will have a small another small cache memory which is known as victim cache why we are calling it victim cache why we are calling it victim cache. The reason is that a full some data has fully associative buffer to keep recently thrown out discarded data from cache as we know. Whenever you have a replace a particular line you have to throw it out, now instead of throwing it out you are letting it into the victim cache. That means the which is the victim block to be replaced that particular block is written into the victim cache. So, victim cache will contain the RISC thrown out data.

So, next time you try to access it you may get it here instead of reading it from main memory. That means here we are trying to combine the first hit time of direct mapped cache as you know in case of direct mapped cache the advantage is it is very simple you directly read it from the cache. And there is only one word or block and in the case in 15 cache what you are doing you are reading it you are storing it and subsequently you can read it from here. So, you add a fully associated buffer to keep recently thrown out discarded data from cache.

(Refer Slide Time: 21:10)

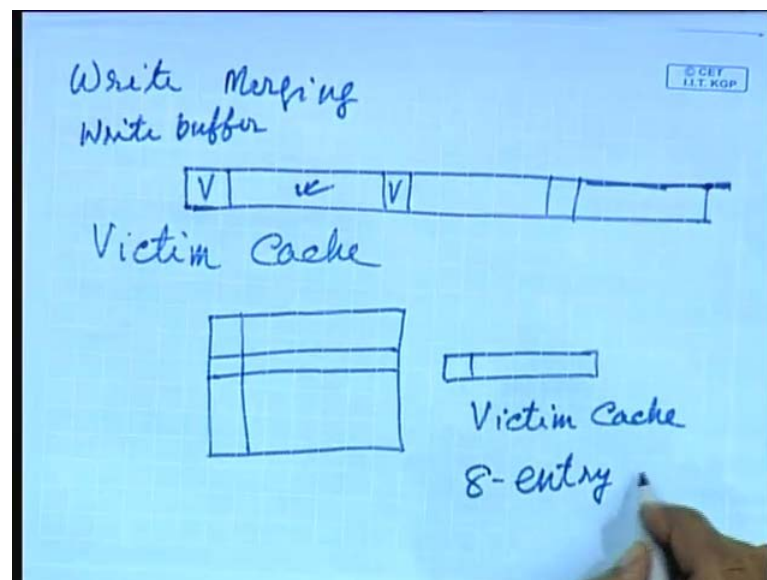
### Reducing Miss Penalty: Victim Cache

- How to combine fast hit time of direct mapped cache:
  - Yet still avoid conflict misses?
- Add a fully associative buffer (victim cache) to keep recently thrown out discarded data from cache.
- Jouppi [1990]:
  - A 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache.
- Used in Alpha, HP machines.
- AMD uses 8-entry victim buffer.

Ajit Pal IIT Kharagpur

And it has been found that a four entry victim cache removes 20 percent to 95 percent of conflicts for a 4 kilobyte direct mapped data cache and for example, it is used.

(Refer Slide Time: 21:40)



In alpha HP machines AMDs uses 8 entry victim buffer. That means you have got not only one can have 8 entry victim buffer and this will help you reducing the miss penalty. So, this is every useful and used in many situations.

(Refer Slide Time: 22:00)

**Reducing Miss Penalty:  
Read Priority over Write on Miss**

- In a write-back scheme:
  - Leads to RAW conflicts with main memory reads on cache misses.
  - Normally a dirty block is stored in a write buffer temporarily.
  - Usual: Write all blocks from the write buffer to memory, and then do the read.
  - Instead:
    - Copy the dirty block to a write buffer, then do the read, and then do the write.
    - CPU stall less since restarts as soon as do read.

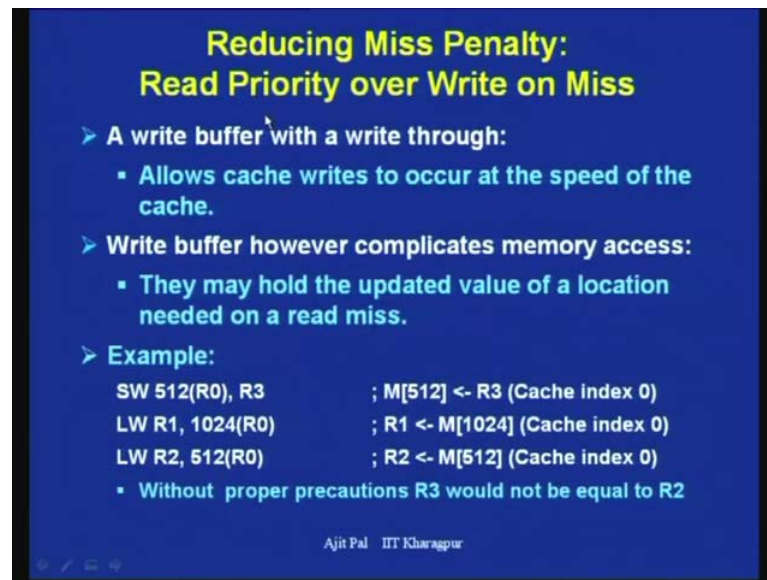
Ajit Pal IIT Kharagpur

Let us consider another technique which is called read priority of a write on miss. So, we are aware of the read after write conflict and whenever you are performing a read operation after write this particular approach. Read priority over write on miss leads to read after write conflict with main memory read on cache misses. Normally, a dirty block is stored in a write buffer as you have already discussed and usually write all blocks from the write buffer to memory and then do the read. So, you are writing into the write buffer then all the words of the buffer is transferred to the main memory..

Then you do the read, but instead of that what can be done copy the dirty block to a write buffer then do the read. And then do the write that means what you are doing here only the dirty block you are writing into to a write buffer. Then you are performing the write then you are doing the write not performing the write of all the block from the write buffer to memory you are not doing that. So, in this case CPU stall is less since restarts as soon as do read to perform a read operation. So, this is how you can avoid I mean you can avoid read priority with main memory reads on cache misses. So, this approach is also used in some processors.



(Refer Slide Time: 23:48)



**Reducing Miss Penalty:  
Read Priority over Write on Miss**

- A write buffer with a write through:
  - Allows cache writes to occur at the speed of the cache.
- Write buffer however complicates memory access:
  - They may hold the updated value of a location needed on a read miss.
- Example:  
SW 512(R0), R3                   ; M[512] <- R3 (Cache index 0)  
LW R1, 1024(R0)               ; R1 <- M[1024] (Cache index 0)  
LW R2, 512(R0)                ; R2 <- M[512] (Cache index 0)
  - Without proper precautions R3 would not be equal to R2

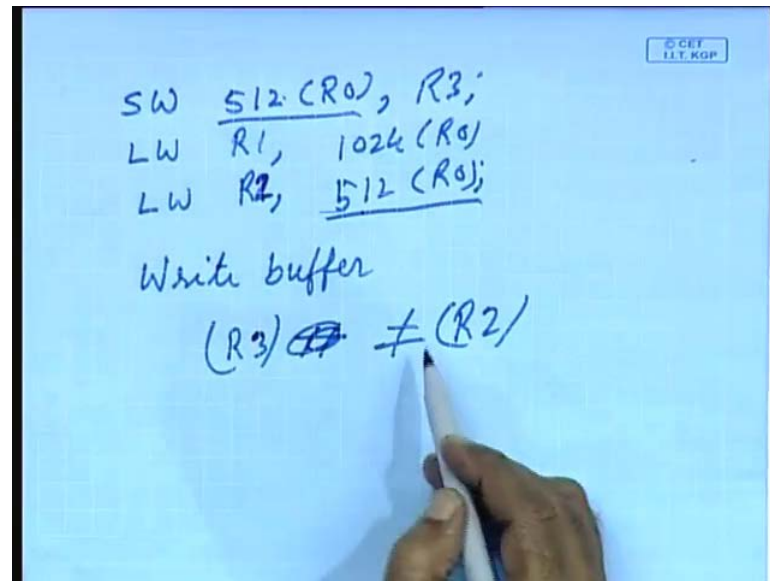
Ajit Pal IIT Kharagpur

Now, let us consider another approach which is read priority over write on miss. So, a write buffer with a write through that we have already discussed it allows cache writes to occur at the speed of the cache. That means as we have seen it writes into the cache memory and that is also written into buffer and in the offline writing takes place from write buffer to main memory. So, however so far the processor is concerned it does not take more than the time need to write to the cache memory.

So, this is the benefit of write buffer whenever you are using which I have already discussed in detail. And write buffer however complicates memory access what can happen they may hold the updated value of a location needed on a read miss. That means immediately after write you are trying you are there is a. This may be hold in updated value of a location needed on a read miss that means after write read miss is occurred, so let me illustrate this with the help of example.



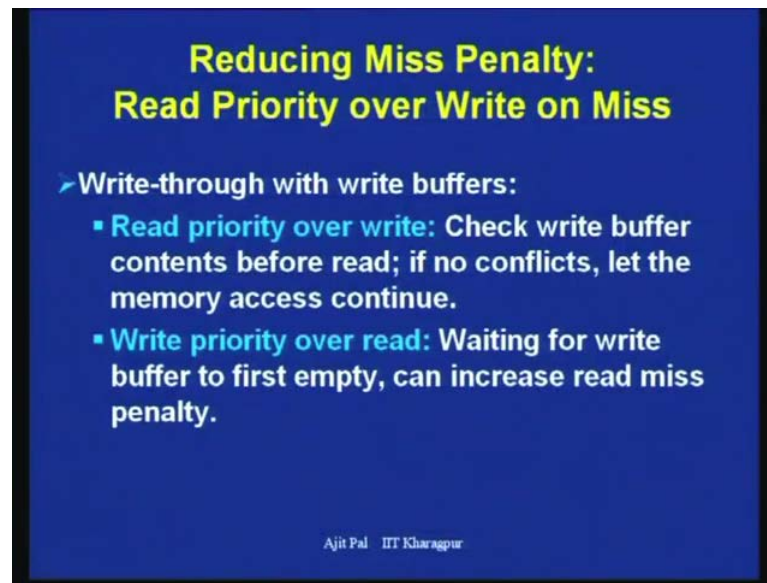
(Refer Slide Time: 25:09)



Say let us consider this program store word 512 R0 R3 and lowered word R1 1024 R0 lowered word R2 512 R0. Now, you can see here from R3 you are writing in the memory. And then again from the same memory location you are writing into the resistor. So, you can say this is write and after that you are reading transferring reading from memory. So, in this case you are reading data resistance contain is written into memory and then you are reading it here. Now, what can happen by using write buffer this was stored into write buffer data was transferred to write buffer. If this read takes place there is miss here and if you read it from a memory you will get the previous data. So, in such a case you will not get a updated data that means they contain of R3 and R2 may not be same.

So, R2 and R3 may not be sorry same as R2 contain of R2 and R3 may not be same if the from the right buffer writing has not taken place in the same memory location. Because these two are referring the same memory location, so this can happen. So, this may hold the updated value of location needed on a read miss. So, this you have to take care whenever you perform, so you have to read priority over write on miss. So, in such a situation you have to take proper precaution. So, that R3 the contain of R3 and R2 are identical, so whenever this type of situation occur necessary precaution must be taken.

(Refer Slide Time: 27:36)



**Reducing Miss Penalty:  
Read Priority over Write on Miss**

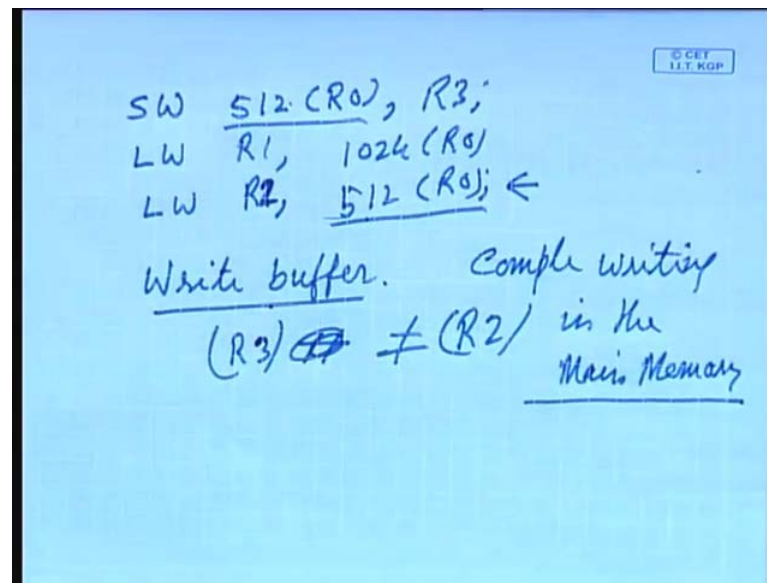
- Write-through with write buffers:
  - **Read priority over write:** Check write buffer contents before read; if no conflicts, let the memory access continue.
  - **Write priority over read:** Waiting for write buffer to first empty, can increase read miss penalty.

Ajit Pal IIT Kharagpur

So, this is reducing miss penalty by read priority over write on miss. So, this is write through with write buffer read priority over write check write buffer content before you read. If no conflict let the memory access continue that means what you are doing you are checking the write buffer contain before you perform the read operation. So, in this case before you perform the read operation from the memory. You will check the write buffer what is this value is present in write buffer.

So, if it is present in write buffer if there is no conflict only the you continue the memory access, means first write buffer contain check. And if it is present in the read buffer you read there and write priority over read in this case waiting for write buffer to the first empty and can increase read miss penalty in such a case what you are doing you are that.

(Refer Slide Time: 238:47)



Whatever have been written into that write buffer it must complete writing in the main memory. So, after the writing is complete that mean your write buffer is empty only then you perform the read, so this is write priority over read. So, you have to you have two different approaches in the first case read priority over write you are giving priority to read over the write into the write of the data. That is present in the write buffer in the main memory, but in such a case you have to check the right buffer first before you read it from the main memory. Second case you are waiting till the write buffer is empty and in such a case obviously read miss penalty is because the processor has to wait till the write buffer is becomes empty data is getting into the main memory.

(Refer Slide Time: 29:54)

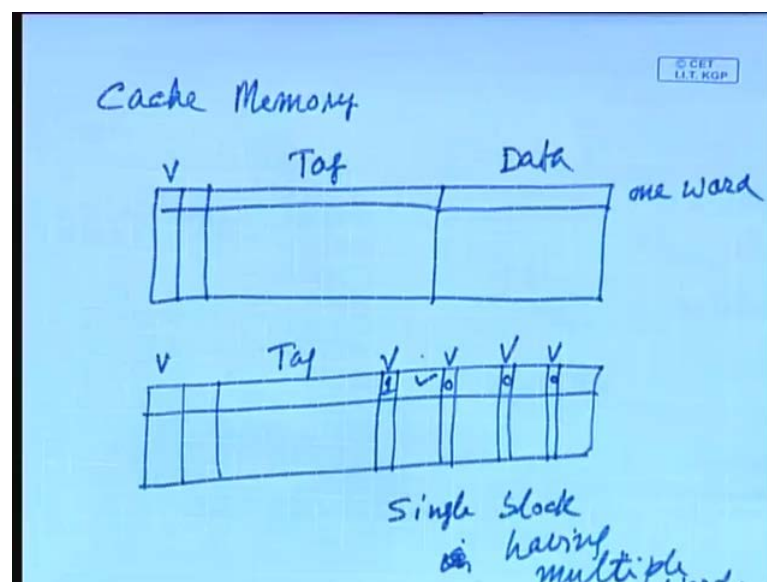
### Reduce Miss Penalty: Subblock Placement

- Single word per block requires large number of tag bits.
- The size of tag bits can be reduced by sharing a single tag and valid bits for multiple words in a block. This requires load of **full block** on a miss.
- This problem can be overcome by having valid bits per subblock (Originally invented to reduce **tag storage**).

Ajit Pal IIT Kharagpur

So, this is the situation of read priority write over miss the let us consider sub block placement. So, in this case single word per block requires large number of tag bits as you know if you are having a if you recall basic organization memory.

(Refer Slide Time: 30:19)



In the cache memory we have got the different value bit. Then you have got those check various other management bits read only and so on. Then you have got the tag field and here is your data or instruction whatever it may be. Now, there is a possibility of having only one word per block, so as you know you can have only one word. Now, to reduce

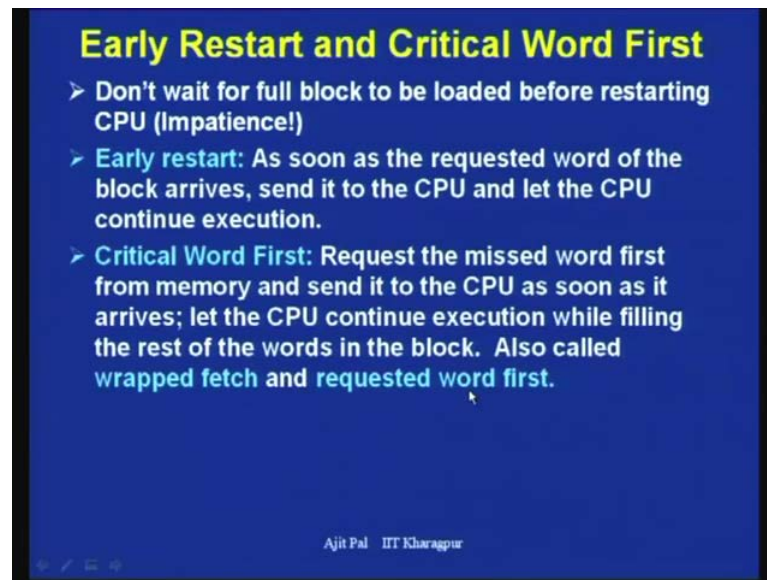
the size of the tag as we have already discussed you can have multiple word such a case multiple words on a single block. So, this part will be same, but your number of data word of a single block will be more. So, this is your tag filed and you have multiple words on a single block. So, we find that you are able to reduce the number of tags required in the memory by having multiple blocks.

In a multiple blocks in a multiple words in a single block in a single block having multiple words, so and also we have seen that we have a single tag field and valid bit field, so there is single tag field. And valid bit field that is being shared by all the word in the single block that is what is a situation. Now, in this case this requires load of full block on a miss. So, whenever there is a cache miss as we know the entire block we transferred all the word are to be transferred. So, in such a case what will happen the miss penalty will be more, so what can be done this problem can be overcome by a valid bit per sub block.

So, in this case in this case what you can do you can add separate valid bit valid bit her valid bit here valid bit here valid bit here in each of these for each of these words. So, in such a case what can happen instead of transferring the reading bit I mean instead of loading the full block you can load only a particular word. So, while having valid bit for sub block and that means you will transfer if there is a miss you will transfer this.

So, this will be one, but the other valid bits may be 0. So, this will reduce the miss penalty when I will be having sub block placement, so instead of reading the all the words from the main memory you can read only one word. And then modify the valid bit, but this will definitely increase the number of valid bits. But the benefit is that your miss penalty will be significantly reduced.

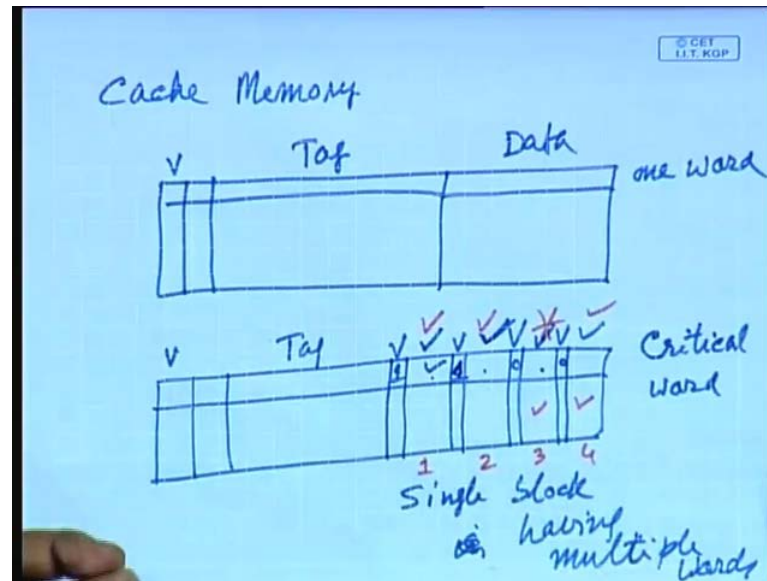
(Refer Slide Time: 34:04)



Another technique that can be used that is known as early restart and critical word first, so in this case do not wait for full block to be loaded before restarting CPU. So, in this case you are reading these words one after the other, because as we know the bandwidth the width of the processor memory bus is usually one word. So, you can transfer read one word at a time, so you will read one word then this word. Then this word then this word one after the other, now in this case what you are doing as soon as the requested word of the block arrives send it to the CPU and let the CPU continue execution. Suppose, the requested word is this one, so you modify you modify make it one and immediately you transfer this particular word to the CPU do not and do not wait for the other words to be read from the CPU.

So, this is known as early restart and critical word first, so this is the early restart technique and we shall discuss the critical first and then this. So, this is how you can reduce the miss penalty, but obviously it will complicate the situation little bit and you have to I mean the processor has to take care of this situation. So, it has to read the other words from the memory after sending it to CPU. So, then comes the critical word first, so request the missed word first from the memory and send it to the CPU as soon as it arrives. Let the CPU continue execution while filling the rest of the word in the block and also it is known as wrapped fetch and requested word first.

(Refer Slide Time: 36:05)



So, in this case you are not reading sequentially you are reading the critical word first, so critical word first means if this is the requested word read this particular word first you can get it if you get it accordingly. And it will read this word and transfer it to CPU in subsequently the other word of the block this one this one. And this one will be read one of after the other and in the mean time the CPU will continue to execution and the miss penalty is reduced.

(Refer Slide Time: 36:46)

### Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU (Impatience!)
- **Early restart:** As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution.
- **Critical Word First:** Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called **wrapped fetch** and **requested word first**.
- Generally useful only in **large blocks**.
- **Spatial locality** a problem; tend to want next sequential word, so it is not clear whether there is benefit by early restart.

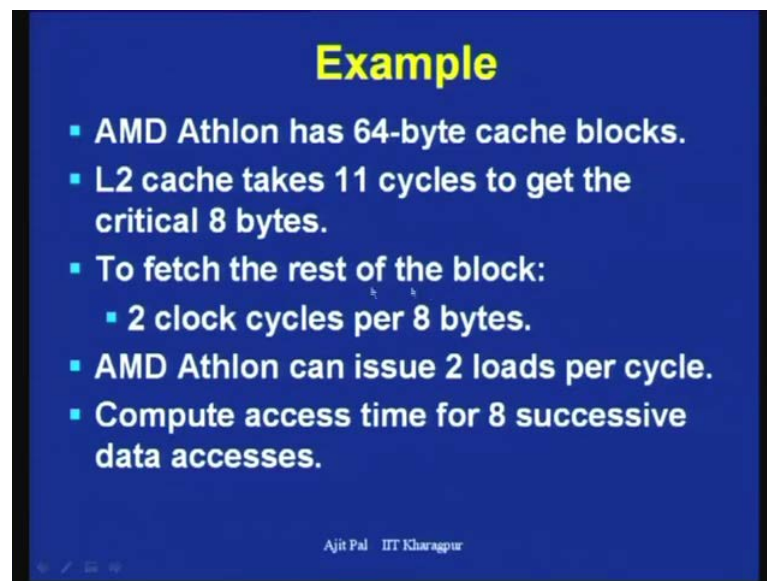
Ajit Pal IIT Kharagpur



So, this particular technique is also used in some cases and this is particularly useful whenever you have got a large block size. So, if you have got one or two word per block then this is not very effective. But, whenever you have got large number of words in a block this particular technique is useful and special locality is a problem it tends to wait next sequential word. So, it is not clear whether there is benefit by early restart you see what we are doing in this particular case we are reading we are reading this particular word. This is the critical word and then you may be reading this then this then this, but what the processor can ask for after this word processor may ask for this. So, again there will be cache miss, so the processor will read word three.

Then word one then word two then word four, but the CPU may ask word four after word three. So, again this will lead to a miss cache miss, so this is actually special locality is being violated because we used reading from sequential location to improve this locality word. Since, it is disturbing the locality it is not known whether there will be much benefit from the early restart technique.

(Refer Slide Time: 38:30)



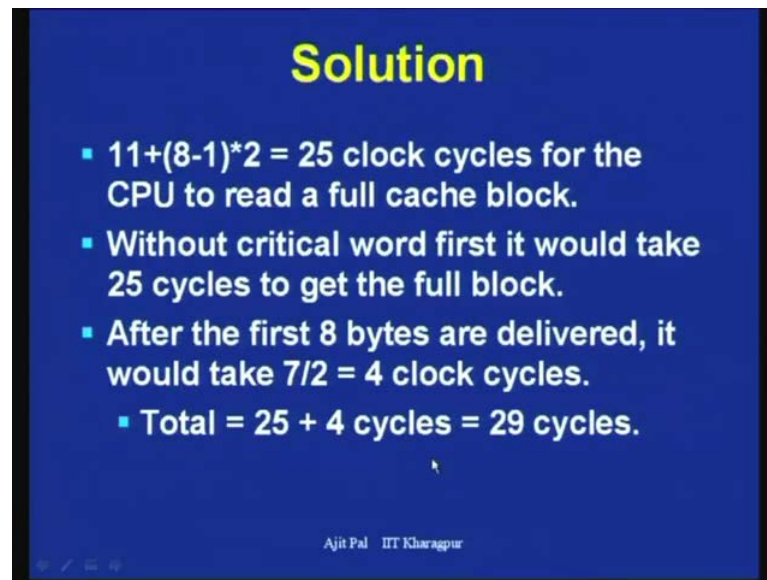
**Example**

- AMD Athlon has 64-byte cache blocks.
- L2 cache takes 11 cycles to get the critical 8 bytes.
- To fetch the rest of the block:
  - 2 clock cycles per 8 bytes.
- AMD Athlon can issue 2 loads per cycle.
- Compute access time for 8 successive data accesses.

Ajit Pal IIT Kharagpur

So, this is an example AMD Athlon has 64 byte cache blocks L2 cache takes 11 cycles to get the critical 8 bytes to fetch the rest if the block 2 clock cycles per 8 bytes. So, AMD Athlon can issue two loads per two loads per cycles, so compute access time for 8 successive data success. So, you can compute the access time for 8 successive data access.

(Refer Slide Time: 39:00)



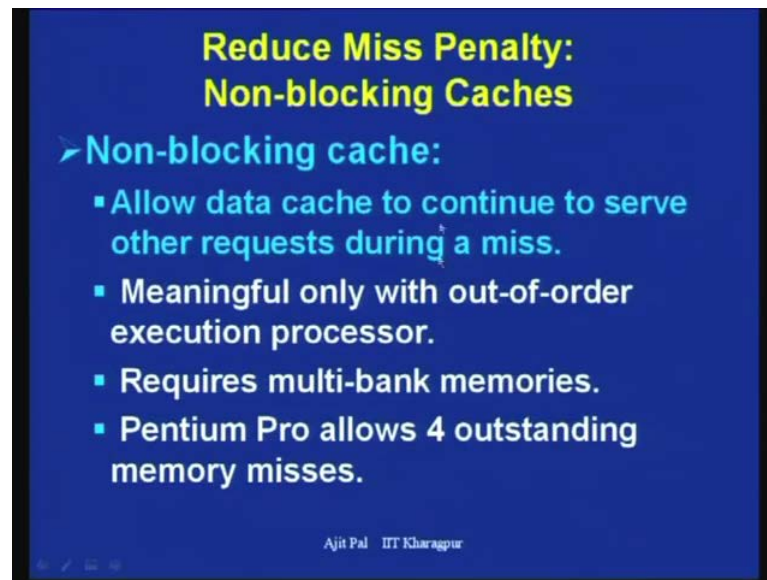
### Solution

- $11+(8-1)*2 = 25$  clock cycles for the CPU to read a full cache block.
- Without critical word first it would take 25 cycles to get the full block.
- After the first 8 bytes are delivered, it would take  $7/2 = 4$  clock cycles.
  - Total =  $25 + 4$  cycles = 29 cycles.

Ajit Pal IIT Kharagpur

The solution is  $11 + 8 - 1 \times 2$ . One you have already read into 2, so is 25 clock cycle CPU a full block cache for this particular problem. Now, without critical word it would take 25 clock cycles to get the full block. So, after the first 8 bytes are delivered it would take  $7 \div 2 = 4$  clock cycle. So, total of 20 plus 4 29 clock cycles are needed in this particular case. So, this shows access time for 8 successive data accesses if you read the critical word first. Then this is what will happen if you read it in this manner that is the time and if you read it in this way 25 plus 4 29 clock cycles. So, this problem this particular problem example high lights the critical word first reading in word I mean problems that arises whenever you do critical word first.

(Refer Slide Time: 40:12)



**Reduce Miss Penalty:  
Non-blocking Caches**

➤ **Non-blocking cache:**

- Allow data cache to continue to serve other requests during a miss.
- Meaningful only with out-of-order execution processor.
- Requires multi-bank memories.
- Pentium Pro allows 4 outstanding memory misses.

Ajit Pal IIT Kharagpur

Now, let us consider the case of non blocking cache memories. So, this non blocking cache allows data cache to continue to serve other. So, normally as you know whenever there is a cache miss, so processor is blocked in the scene there will be. There will be some co cycles or there will be some prox cycles that will be when the laps before the processor assume execute, so that is the reason that is called blocking. Now, if you are having a multi programming environment, so in such a case if a particular execution of particular process is blocked the processor can start execution of another process. So, that is what is being mentioned here, so this is meaningful only with out of order execution processor.

So, whenever you are doing out of order execution processor and in such a case this is very useful and obviously this will require multi bank memories. And pentium pro allows 4 outstanding memory misses. So, even there are memory misses it will continue the execution of out of order that means instruction are not executed in order did not be from multiple processor. But, here out of order execution is taking place different instruction. So, where the instructions are present in program memory it will not be executed in the same order and as I mentioned this will require multi bank memories. Pentium pro allows 4 outstanding memory misses and non blocking caches to reduce stalls on misses.

(Refer Slide Time: 42:09)



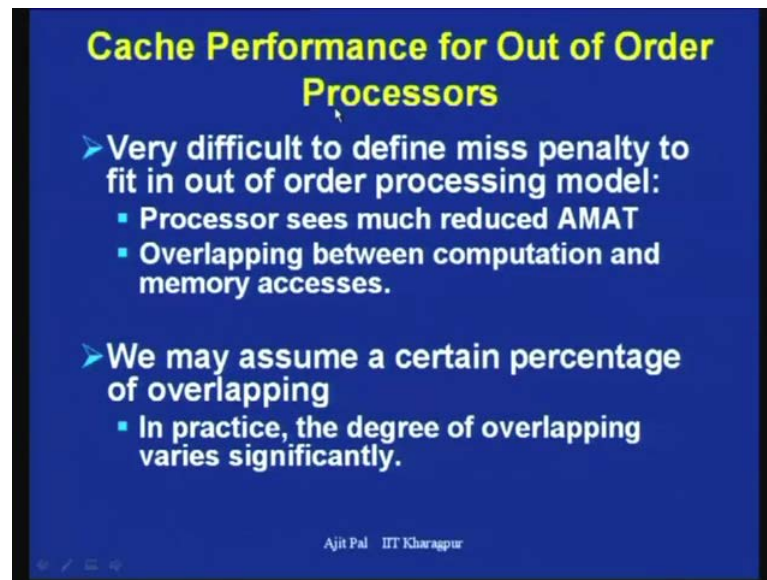
### Non-blocking Caches to Reduce Stalls on Misses

- **Hit under miss** reduces the effective miss penalty by working during miss.
- **“Hit under multiple miss”** or **“miss under miss”** may further lower the effective miss penalty:
  - By overlapping multiple misses.
  - Significantly increases the complexity of the cache controller as **there can be multiple outstanding memory accesses**.
  - Requires multiple memory banks.

Ajit Pal IIT Kharagpur

So, in this case hit under miss reduces the effective miss penalty by working during miss. So, hit under multiple miss or miss under miss may further lower the effective miss penalty this can happen because of multiple overlapping of multiple misses by overlapping multiple misses. And significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses. So, cache controller will become a very complex whenever you allow this multiple misses it will definitely require multiple memory banks which I have already mentioned means. That means without multiple memory banks this will not be effective. So, the edges will be generated and that is available from different banks and then you can read it at a faster rate.

(Refer Slide Time: 43:15)



**Cache Performance for Out of Order Processors**

- Very difficult to define miss penalty to fit in out of order processing model:
  - Processor sees much reduced AMAT
  - Overlapping between computation and memory accesses.
- We may assume a certain percentage of overlapping
  - In practice, the degree of overlapping varies significantly.

Ajit Pal IIT Kharagpur

So, the cache performance for out of order processors is difficult to define miss penalty we have been so far discussing miss penalty in the context of I mean the way the instruction are executed. But, whenever you do out of order execution out of order process it is very difficult to define miss penalty to fit in out of order processing model processor sees much reduced average memory access time. And over lapping between computation and memory access will takes place in the particular situation and we may assume a certain percentage of overlapping. So, we may assume some percentage, but this percentage will vary from application to application. So, in fact the degree of over lapping will vary significantly and that is the reason why it is difficult to define miss penalty whenever you are having out of order processors.

(Refer Slide Time: 44:25)

### Reducing Miss Rate and Miss Penalty: Hardware Prefetching

- Prefetch both data and instructions:
  - Instruction prefetching done in almost every processor.
  - Processors usually fetch two blocks on a miss: requested and the next block.

Ajit Pal IIT Kharagpur

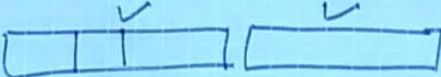
Now, we shall discuss another technique which is known as prefetching. And in fact this not only reduces penalty it reduces miss rate and the prefetching can be done in different ways. First one is known as hardware prefetching, so in this case we prefetch both data and instructions instruction prefetching is done in almost every processor. This is very common technique and processors usually fetch two blocks on a miss requested and next block. So, what you are doing because of special to exploit a special locality.

(Refer Slide Time: 45:03)

Spatial Locality

Cache Miss  $\Rightarrow$  Entire block comprising multiple words are fetched

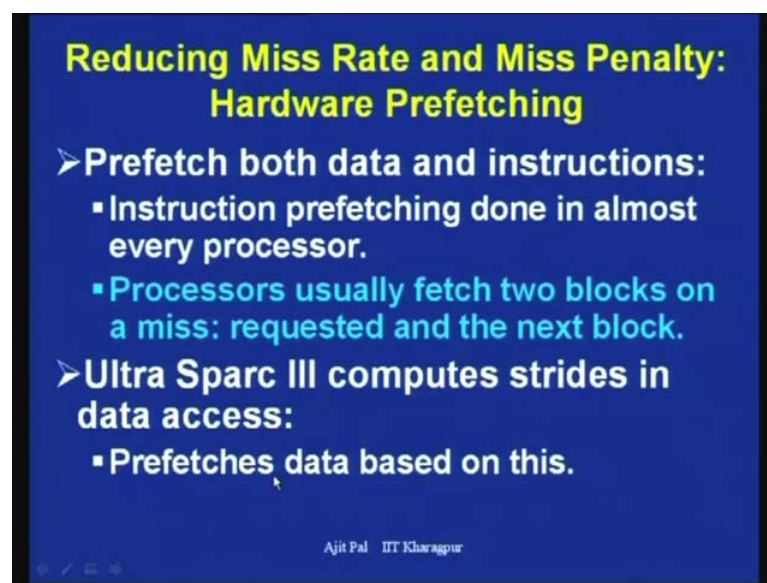
Fetching more than one block





Normally, as you know whenever there is a cache miss the entire block is fetched, now essentially to special locality. Now, we are going further what you are doing you are fetching more than one block all though you can block say for the first block. And also the second block although the requested word in the first block, this is the requested word you are fetching this block the next adjacent block. So, this is known as prefetching the processor usually fetch two blocks on a miss requested and the next block and there are several commercial.

(Refer Slide Time: 46:28)



**Reducing Miss Rate and Miss Penalty:  
Hardware Prefetching**

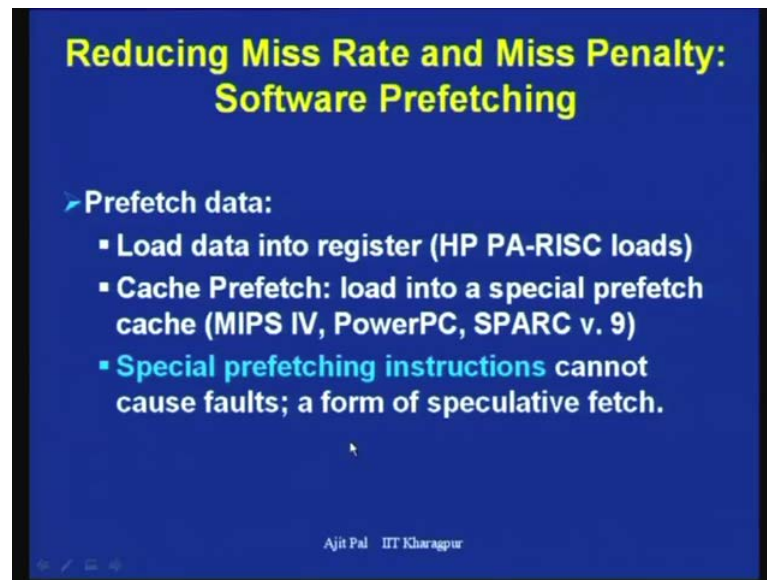
- **Prefetch both data and instructions:**
  - Instruction prefetching done in almost every processor.
  - Processors usually fetch two blocks on a miss: requested and the next block.
- **Ultra Sparc III computes strides in data access:**
  - Prefetches data based on this.

Ajit Pal IIT Kharagpur

Processors like ultra SPARC three computes strides in data access. So, prefetches data based on this based on this approach, so this technique may help in reducing the miss penalty because of the locality. So, next time whenever you try to read word from these you can block you can already pre fetch. So, it will not lead to any cache miss, so this is how the miss penalty reduced and also it reduces the miss rate because next time miss will not occur. So, may miss rate as well as miss penalty.



(Refer Slide Time: 47:14).



**Reducing Miss Rate and Miss Penalty:  
Software Prefetching**

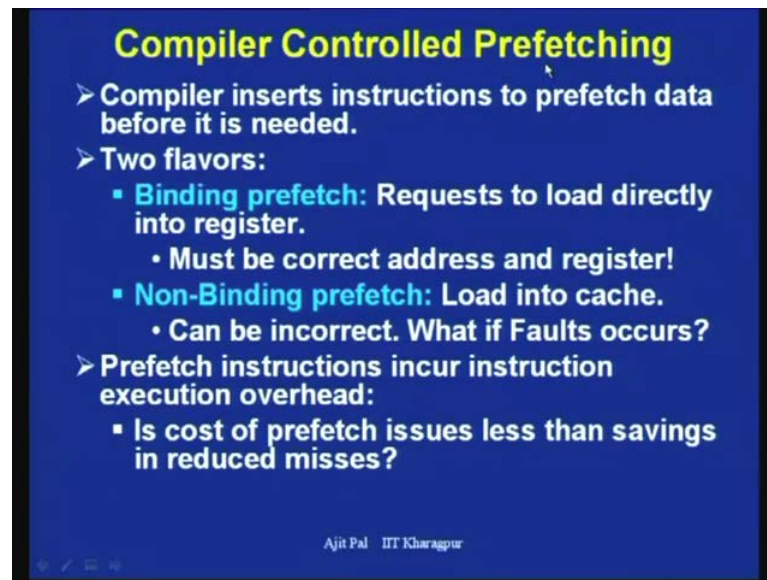
- Prefetch data:
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into a special prefetch cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative fetch.

Ajit Pal IIT Kharagpur

Now, another technique is software prefetching previous one was hardware prefetching hardware automatically does it. In this case you can prefetch data load the data into the register as it is done HP PA-RISC loads or you can perform cache prefetch into a special prefetch cache instead of loading into the resistor. You are loading into special prefetch cache as it is done in MIPS IV, PowerPC, SPARC version 9. And special prefetching instruction obviously whenever you are doing it by software you will require special prefetching instructions.

So, these special prefetching instructions cannot cause faults a form of speculative fetch. So, you are doing a kind of speculative fetch that means assuming that word or instruction from the next will be required you will be doing prefetching by using special instructions. So, you will require special instruction provided for the purpose of prefetch.

(Refer Slide Time: 48:25)



**Compiler Controlled Prefetching**

- Compiler inserts instructions to prefetch data before it is needed.
- Two flavors:
  - **Binding prefetch:** Requests to load directly into register.
    - Must be correct address and register!
  - **Non-Binding prefetch:** Load into cache.
    - Can be incorrect. What if Faults occurs?
- Prefetch instructions incur instruction execution overhead:
  - Is cost of prefetch issues less than savings in reduced misses?

Ajit Pal IIT Kharagpur

Then third approach based on compiler controlled prefetching, so in this case compiler inserts instruction to prefetch data before it is needed. There are two flavors one is known as binding prefetch request to load directly into register. So, must be correct address and register that means whenever we are writing into register I mean it is a strong binding. And you cannot really change it because you are already in the register and non binding prefetch you are doing it into loading it into cache. So, this can be incorrect whenever fault occurs you may have to discard the cache. And you have to reload it replacement has to take place can take place in particular case.

But, this cannot happen in the binding prefetch, so whenever you do prefetching done by software or done by compiler. With the help of compiler prefetching instruction will incur these are additional instruction obviously there is additional overhead. So, what you have to do whenever you are using prefetching. So, this additional prefetching instruction will make the execution time longer. So, you have to see whether benefit that means loss because of the additional instruction is being matched by the by the reduction of miss penalty. So, is the cost prefetch issues less than savings in reduced misses, so if the miss reduction miss penalty is reduced because of prefetch instruction prefetch data then it is fine.

So, this is a kind of trade off that you have to check that means you may get benefit you may not get benefit by prefetching, so that has to be checked. So, we have discussed

various techniques for cache memory. Optimization, now we shall summaries them very quickly to give an over view the techniques we have discussed in three lectures.

(Refer Slide Time: 51:04)

Cache Optimization Summary				
Technique	MR	MP	HT	Complexity
Small & Simple Caches	-	+	+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1
Simultaneous Tag Comparison and Data Reading			+	2
Write Strategy			+	2
Way Prediction and Pseudo-Associative Cache			+	2
Avoiding address translation (Virtual Cache)			+	2
Larger Block Size	+	-		0
Higher Associativity	+	-		1
Pseudo-Associative Caches	+			2
Compiler Reduce Misses	+			0
Second Level Caches		+		2
Write buffer		+		1
Victim Cache		+		1
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
HW Prefetching of Instr/Data		+	+	2
Compiler Controlled Prefetching		+	+	3

So, first one is simple small and simple cache that is used to reduce the hit time however it may increase the miss penalty and complexity is not much. So, complexity is 0 and avoiding address translation whenever you do address translation then again hit time is reduced. And it has got little bit complexity, so complexity level is two then third is pipelining writes. So, writing operation can be pipelined to reduce the hit time and complexity level is one. Then simultaneous tag comparison and data reading which I have discussed which reduces hit time having complexity of two then using suitable write strategy write back.

And we have discussed about the advantages and disadvantages of write through. And write with policies and by using write strategy you can have of two that means normally write through is used and with suitable precaution. Then way prediction and pseudo associative cache which also reduces hit time and complexity level is two and avoiding address translation. That is your virtual cache that also have a complexity of two then you have got various techniques are reducing. The miss rate in this larger block size whenever you go larger block size miss rate is reduced. However miss penalty increase because of larger block size you have to read them all.

The blocks and complexity level is 0 and then higher associative you can go for that also reduces the miss rate and miss penalty is increased. Whenever, we go for higher associative because of the complexity of the memory and it is little more complex pseudo reduces can be used with complexity level 2 to reduce the miss rate compiler. Various compiler techniques that you have discussed usual used to reduce the miss rate however in such a case there is no change in hardware.

So, complexity level is 0 then we have discussed various techniques for reducing the miss penalty second level cache level is two that reduces the miss penalty write buffer. It reduces the miss penalty level is one victim cache that also reduces the miss penalty complexity level is one priority to read misses. We discussed has complexity level 1 sub block placement we have seen if you say that miss penalty is hit time which has the level 1. And early restart and critical word first is quite complex level is two and it reduces the miss penalty non blocking cache memory that we have discussed which is really very complex.

So, complexity level is three and hardware prefetching of instruction and data that reduces both hit time and miss penalty with complexity level 2. And compiler controller prefetching which reduces both miss penalty and miss time has complexity I level 3. So, these tables give you the various widely used techniques in commercial processors, which are used to reduced the either the hit time or the miss rate or miss penalty. So, these are the various cache optimization techniques that we have discussed in today's lecture in my next lecture we shall focus on main memory.

Thank you.